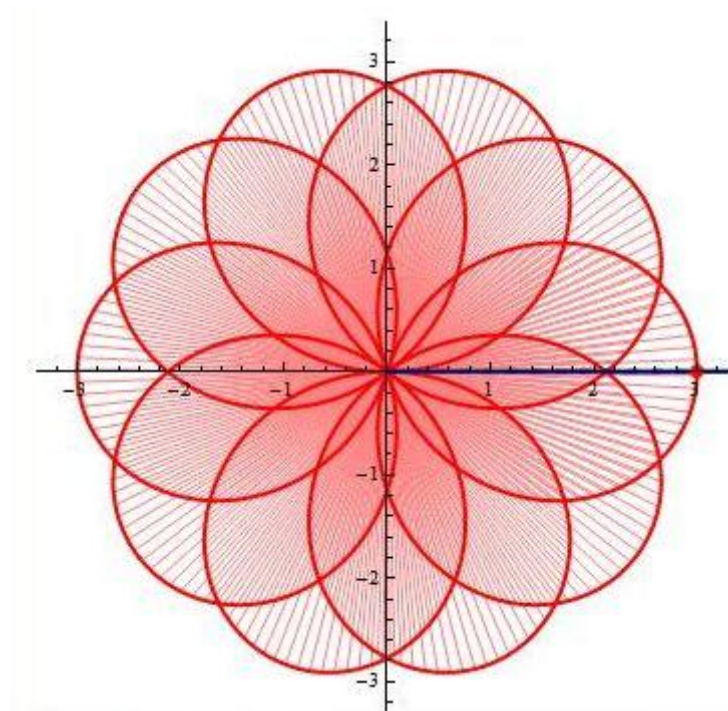


MÉTODOS NUMÉRICOS EN MATHEMATICA



HERNAN PEÑARANDA V.

materias-hpv.comli.com

Sucre, 2010

1. INTRODUCCIÓN A MATHEMATICA

1.1. Introducción

En esta materia estudiaremos métodos que nos permitirán resolver problemas matemáticos complejos mediante la realización de una gran número de cálculos aritméticos simples, sin embargo, debido al elevado número de cálculos que es necesario efectuar, estos métodos sólo son de utilidad práctica si se aplican con la ayuda de una computadora (o una calculadora programable).

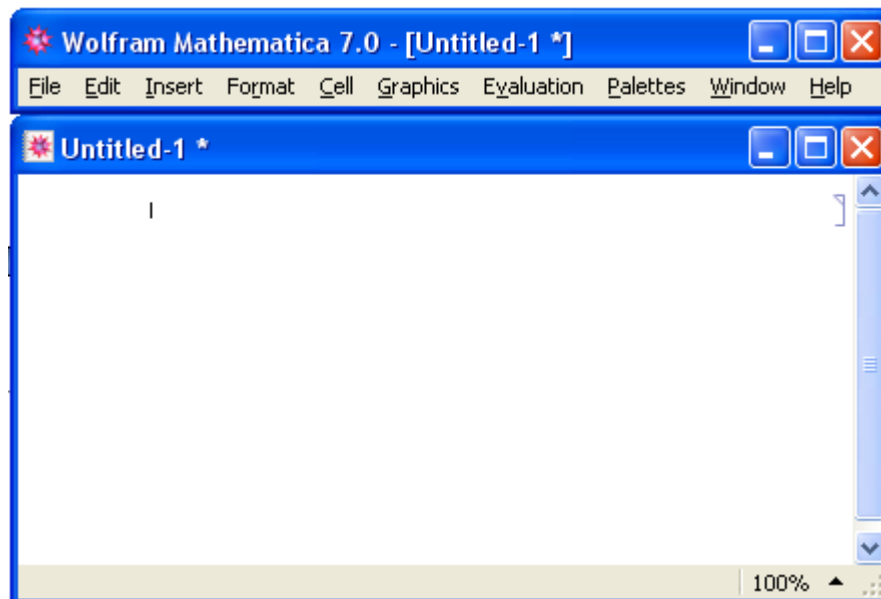
En el ámbito de las computadoras podemos elegir de entre una gran variedad de lenguajes y paquetes que nos permiten automatizar los procesos de cálculo, de dichos paquetes emplearemos en esta materia Mathematica, pues es una las herramientas más completas que existe cuando se trata de resolver problemas matemáticos, cuenta con una amplia librería de funciones que nos permiten resolver prácticamente cualquier problema matemático y cuenta también con un amplio conjunto de instrucciones que facilitan la elaboración de programas tanto simples como complejos.

El propósito de este capítulo es que el estudiante se familiarice con el manejo de esta herramienta y que esté en condiciones de resolver problemas mediante la elaboración de programas.

Puesto que en Informática I, que es la materia prerrequisito de Informática II, se estudian los principios de la programación estructurada y se elaboran programas en el lenguaje Pascal, en esta materia se seguirán aplicando dichos principios y para facilitar el aprendizaje de las nuevas instrucciones, se mostrará su equivalencia con las instrucciones respectivas de Pascal.

1.2. Los primeros pasos

Mathematica puede ser vista y usada como una calculadora con grandes capacidades de cálculo y una enorme librería de funciones. Veremos primero cómo se emplea esta herramienta para llevar a cabo cálculos simples. Las explicaciones y descripciones que demos en este curso se ajustan a la versión 7.0 de Mathematica, cuya interfaz tiene la siguiente apariencia:



Como se puede observar, la interfaz de Mathematica consta básicamente de dos partes: El menú (que se encuentra en la parte superior) y el "cuaderno" (notebook) que es la ventana en blanco (Untitled-1*) ubicada debajo del menú. Es en el cuaderno donde se escriben todas las instrucciones y donde Mathematica nos presenta los resultados. En el menú se pueden acceder a varias opciones del programa, siendo las más útiles por el momento las opciones "File->Save" (Ctrl+S), para guardar el trabajo hecho en el cuaderno y "Help->Documentation Center" para acceder a la ayuda de Mathematica.

El cuaderno de Mathematica se compone de "celdas", las cuales aparecen automáticamente a medida que se van escribiendo las instrucciones y obteniendo los resultados. Cada celda se identifica por un corchete (azul) que aparece al lado derecho del cuaderno. Para utilizar Mathematica como una calculadora, simplemente se escriben las operaciones en el cuaderno y se pulsan las teclas "Shift+Enter" para obtener el resultado:



Como se puede observar en la figura, para llevar a cabo esta operación Mathematica crea 3 celdas: La primera, donde se han escrito las operaciones, es una celda de entrada y a dichas celdas Mathematica las identifica con la palabra "In" seguida del número de celda entre corchetes. La segunda, donde muestra el resultado, es una celda de salida y a dichas celdas Mathematica la identifica con la palabra "Out" seguida del número de celda entre corchetes. La tercera simplemente agrupa las celdas de entrada y de salida.

Los números asociados a las celdas de entrada y principalmente a las de salida, permiten la reutilización de los resultados obtenidos en cálculos previos, para ellos simplemente se escribe "Out" y el número de celda respectivo, por ejemplo, podemos dividir el número 128 entre el resultado obtenido en el primer cálculo:



Por conveniencia, en lo sucesivo no mostraremos las celdas, sino directamente las instrucciones y los resultados obtenidos, sin embargo, en Mathematica siempre trabajará con celdas y sus identificadores respectivos.

Para corregir algún error, simplemente debe hacer click en la instrucción y corregir el error como lo haría en un cualquier procesador de textos (no debe olvidar pulsar "Shift+Enter" para procesar la instrucción).

Mathematica cuenta con todos los operadores y funciones de una calculadora científica (y muchas otras más). Mathematica diferencia entre mayúsculas y minúsculas y todas las funciones en Mathematica comienzan con mayúsculas pasándose los datos (los argumentos) entre corchetes.

Algunas de las funciones de uso más frecuente son:

Sqrt[x]:	Raíz cuadrada de "x", $\sqrt{x}$
Exp[x]:	Exponente de "x", e^x
Log[x]:	Logaritmo natural de "x", $\ln(x)$
Log[b,x]:	Logaritmo en base "b" de "x", $\log_b(x)$
Sin[x], Cos[x], Tan[x], Csc[x], Sec[x], Cot[x]:	Seno, coseno, tangente, cosecante, secante y cotangente de "x" (en radianes).

ArcSin[x], ArcCos[x], ArcTan[x], ArcCsc[x], ArcSec[x], ArcCot[x]: Funciones trigonométricas inversas ($\sin^{-1}(x)$, $\cos^{-1}(x)$, $\tan^{-1}(x)$, $\csc^{-1}(x)$, $\sec^{-1}(x)$, $\cot^{-1}(x)$).
 Sinh[x], Cosh[x], Tanh[x], Csch[x], Sech[x], Coth[x]: Funciones hiperbólicas.
 ArcSinh[x], ArcCosh[x], ArcTanh[x], ArcCsch[x], ArcSech[x], ArcCoth[x]: Funciones hiperbólicas inversas.
 n!: Factorial de "n"
 Abs[x]: Valor absoluto de "x"
 Round[x]: Valor entero más próximo al número real "x"
 Floor[x]: Entero más grande, menor o igual al número real "x"
 Ceiling[x]: Entero más pequeño, mayor o igual al número real "x"
 Chop[x]: Devuelve cero si "x" es menor a 10^{-10} .
 Chop[x, 10^{-n}]: Devuelve cero si "x" es menor a 10^{-n} .
 IntegerPart[x]: Parte entera del número real "x"
 FractionalPart[x]: Parte fraccionaria del número real "x"
 Re[x]: Parte real del número complejo "x"
 Im[x]: Parte imaginaria del número complejo "x"
 Mod[m,n]: Residuo de la división de "m" entre "n"
 Quotient[m,n]: Cociente de la división de "m" entre "n"
 RandomReal[]: Número pseudoaleatorio comprendido entre 0 y 1.
 Max[x,y,...], Min[x,y,...]: El máximo y mínimo respectivamente de los valores "x", "y", ...

Mathematica cuenta también con algunas constantes matemáticas, siendo las de uso más frecuente las siguientes:

Pi: Número pi, $\pi = 3.14159...$
 E: Número de Nepper, $e = 2.71828...$
 Degree: Factor para la conversión de grados a radianes, $^{\circ}$
 I: Número imaginario: $\sqrt{-1}$
 Infinity: Número infinito, ∞

Así por ejemplo para calcular la raíz cuadrada de 6.5 escribimos:

Sqrt[6.5]

Con lo que obtenemos:

2.54951

Sin embargo, si escribimos:

sqrt[6.5]

Obtenemos la misma expresión sin evaluar, porque Mathematica conoce la función "Sqrt", pero no "sqrt".

En Mathematica los resultados reales se muestran por defecto con 6 dígitos (tal como se puede observar en el primer ejemplo), sin embargo, es posible cambiar este número de dígitos en "Edit->Preferences->Appearance->Numbers->Number of digits displayed in output", o emplear funciones como "InputForm", para ver el resultado con todos sus dígitos:

InputForm[Sqrt[6.5]]
 2.5495097567963922

O "NumberForm" para ver el resultado con un número determinado de dígitos:

NumberForm[Sqrt[6.5],9]
 2.54950976

En Mathematica casi siempre existen al menos dos formas de hacer lo mismo, así por ejemplo se puede emplear el operador de posfijo `"/"` para aplicar una función al resultado de una expresión, con este operador la primera instrucción sería:

```
Sqrt[6.5]//InputForm
2.5495097567963922
```

Cuando se trabaja con números enteros Mathematica muestra todos los dígitos, por ejemplo cuando calculamos 56^{23} obtenemos:

```
56^23
16155656889615734329398214425629966729216
```

Que es el resultado exacto. Igualmente si calculamos el factorial de 100 obtenemos:

```
100!
9332621544394415268169923885626670049071596826438162146859296389521759999322
9915608941463976156518286253697920827223758251185210916864000000000000000000
000000
```

Sin embargo, si calculamos el factorial de `"100."` (un número real) obtenemos:

```
100.!
9.33262*10^157
```

Que es un resultado aproximado. Igualmente obtenemos un resultado exacto para la raíz cuadrada de 63 y uno aproximado para `"63."`:

```
Sqrt[63]
3  $\sqrt{7}$ 
Sqrt[63.]
7.93725
```

Siempre que le es posible Mathematica devuelve un resultado exacto, y para ello emplea un número ilimitado de dígitos o expresiones simbólicas. Se puede forzar la obtención de un resultado numérico aproximado empleando la función `"N"` (`N[expresión]` o `expresión//N`), por ejemplo para obtener el resultado aproximado de la raíz cuadrada de 63 escribimos:

```
Sqrt[63]//N
7.93725
```

Con `"N"` podemos fijar también el número de dígitos de precisión `"n"` con los que se quiere calcular un resultado (`N[expresión,n]`), sin embargo, para que el resultado sea calculado con la precisión establecida es necesario que la expresión sea exacta o tenga una precisión lo suficientemente alta (mayor a `"n"`). Por ejemplo, podemos calcular el factorial de 100 (que es un número entero y por lo tanto exacto) con 30 dígitos de precisión:

```
N[100!,30]
9.33262154439441526816992388563*10^157
```

Por defecto cuando Mathematica trabaja con números reales trabaja con 16 dígitos de precisión (precisión de máquina).

Como ya vimos anteriormente, se puede emplear cualquier resultado calculado previamente escribiendo `"Out"` y el número de resultado entre corchetes, alternativamente, se puede acceder al último resultado empleando el símbolo de porcentaje `"%"`, al penúltimo con `"%"`, al antepenúltimo con `"%%"` y así sucesivamente. Por ejemplo en las siguientes instrucciones calculamos la raíz cuadrada de 2. y luego calculamos el seno, coseno y tangente de la raíz cuadrada de 2.:

```
Sqrt[2.]
1.41421
Sin[%]
0.987766
Cos[%]
0.155944
Tan[%]
6.33412
```

1.2.1. Ejemplos

Calcule el valor de las siguientes expresiones:

1.
$$\sqrt[3]{\frac{\sin(9.2) - \cos(8.32)}{\tan(6.89)}}$$

```
((Sin[9.2]-Cos[8.32])/Tan[6.89])^(1/3)
0.989339
```

2.
$$\frac{|\sinh^{-1}(2.2) - \cosh^{-1}(3.4)|}{\tanh^{-1}(0.9)}$$

```
Abs[ArcSinh[2.2]-ArcCosh[3.4]]/ArcTanh[0.9]
0.247856
```

3.
$$\left(\ln(\csc(7) + \sec(9.2) + \cot(3.4))\right)^{9.3}$$

```
(Log[Csc[7]+Sec[9.2]+Cot[3.4]])^9.3
32.4698
```

4.
$$\sqrt[5]{\frac{\sin(45^\circ) - \cos(70^\circ)}{3.4 \tan(80^\circ)}}$$

```
((Sin[45 °]-Cos[70 °])/(3.4*Tan[80 °]))^(1/5)
0.452321
```

5. Parte entera de: $4.2^{7.6} - 9.2^{8.2}$

```
IntegerPart[4.2^7.6-9.2^8.2]
-79939972
```

6. Entero más pequeño mayor o igual al resultado de: $(6.8 \cdot \sin(125^\circ))^3$

```
Ceiling[(6.8*Sin[125 °])^3]
173
```

7. Resíduo de la división: $\frac{(6+5)! 5^3}{(3+4)^7}$

```
Mod[(6+5)!*5^3, (3+4)^7]
576506
```

8. 25 dígitos de precisión de: $(4(6+3))!$

```
N[(4*(6+3))!, 25]
3.719933267899012174679994*10^41
```

9. 40 dígitos de precisión de: $e^{6!+5^6}$

```
N[Exp[6!+5^6], 40]
3.493869734060630629923969473941009069382*10^7098
```

1.2.2. Ejercicios

Calcule el valor de las siguientes expresiones:

1. $\ln(6.573)e^{7.8234}$ 4704.385807047914
2. $\sqrt{\frac{\log(6.75)}{6.2+3^{6.7}}} \cos(30.2^\circ)$ 0.01980606637583929
3. $\sin^{-1}\left(\frac{\sqrt{7.2}+9.67}{8.43}\right)$ 1.570796327+0.930806824i
4. $e^{\frac{(\sinh(4.3))^{3.2}+(\cosh(2.3))^{5.4}}{\tan(9.8)}}$ 8.744452522473245e120264
5. $\sqrt[3]{(50!)(4^7)}$ 7.928023654338997e22
6. $\sqrt{\frac{5^3-2^2+3!}{(7-3)^2}}$ 2.817356917396161
7. Valor redondeado de: $\ln(6.7)+\log(4.3)$ 3
8. Parte fraccionaria de: $\left(\sin(9.2)-\frac{\sec(5.6)}{\sinh(9.2)}\right)^{0.34}$ 0.6000369946881564
9. Entero más pequeño menor o igual al resultado: $\ln\left(\frac{4.5^{9.2}-9.8^{7.6}+e^{9.87}}{\sqrt{6.32+2^{0.98}}}\right)$ 16+3i
10. Cociente de la división: $\frac{5!+3^4}{4^3}$ 3

1.3. Otras formas de escribir expresiones en Mathematica

En este subtítulo veremos brevemente algunas de las varias formas en las que se pueden escribir operaciones y funciones en Mathematica.

Por una parte podemos escribir las expresiones en su forma completa, que es la forma a la que Mathematica traduce las expresiones antes de evaluarlas. Todas las instrucciones y operadores en Mathematica son en realidad funciones, así por ejemplo cuando escribimos $3+4$, Mathematica traduce esta expresión a `Plus[3,4]`, donde "Plus" es la función para la suma (+). La función para la multiplicación es "Times", así $3*4$ se traduce a `Times[3,4]`, la función para la potenciación es "Power", así 3^5 se traduce a `Power[3,5]`, la función para la resta es "Subtract", así $10-5$ se traduce a `Subtract[10,5]` (y esta a `Plus[10,Times[-1,5]]`), la función para la división es "Divide", así $8/3$ se traduce a `Divide[8,3]` (y esta a `Times[8,Power[3,-1]]`).

Empleando los nombres de las funciones en lugar de los símbolos de los operadores, la siguiente expresión se escribe de la siguiente forma:

$$\sqrt{\frac{(9+10)^4-(6*4)^3}{(30+7)^2}}$$


```
Sqrt[Divide[Subtract[Power[Plus[9,10],4],Power[Times[6,4],3]],Power[Plus[30,7],2]]]
```

$$\frac{\sqrt{116497}}{37}$$

Empleando paréntesis y los símbolos de los operadores, la expresión es:

```
Sqrt[((9+10)^4-(6*4)^3)/(30+7)^2]
```

$$\frac{\sqrt{116497}}{37}$$

Que como se puede observar es mucho más fácil de entender y elaborar que la primera. Mientras más nos acercamos al nivel de la máquina más difícil resulta para los seres humanos entender y elaborar las instrucciones (aunque en contrapartida se ejecutan más eficientemente). Por esta razón, emplearemos esta forma sólo en aquellos raros casos donde faciliten la elaboración de las instrucciones y/o las hagan más claras. Para transformar una expresión a su forma completa, Mathematica cuenta con la función "FullForm".

En contrapartida podemos alejarnos más aún del nivel al que operan las máquinas y acercarnos más a la forma simbólica con la que estamos acostumbrados a trabajar los seres humanos. Mathematica nos permite trabajar directamente con la mayoría de los símbolos matemáticos conocidos. Para incorporar estos símbolos en una expresión se puede emplear la paleta matemática de entradas básicas (Palettes->BasicMathInput), emplear combinaciones de teclas (atajos), escribir los códigos de escape, o empleando el nombre completo del símbolo u operador ([Nombre]). Por ejemplo para utilizar el símbolo de la raíz cuadrada se puede seguir uno de los siguientes caminos: a) hacer clic en el símbolo respectivo en la paleta matemática de entradas básicas, b) pulsar las teclas "Ctrl+2", c) pulsar la tecla escape, escribir sqrt y volver a pulsar la tecla escape, d) escribir \[Sqrt].

Para escribir en forma simbólica la expresión escrita anteriormente pulsamos las siguientes teclas: Ctrl+2 Ctrl+/ (9 + 10) Ctrl+^ 4 Ctrl+Espacio - (6 * 4) Ctrl+^ 3 Ctrl+Espacio Tab (30 + 7) Ctrl+^ 2 Ctrl+Espacio Ctrl+Espacio Ctrl+Espacio:

$$\sqrt{\frac{(9+10)^4-(6*4)^3}{(30+7)^2}}$$

$$\frac{\sqrt{116497}}{37}$$

Que por supuesto es mucho más fácil de entender, aunque, como se puede observar, su escritura requiere más tiempo (y se evalúa también en más tiempo), esta es la razón por la cual tampoco emplearemos la forma simbólica con mucha frecuencia. Para transformar una expresión matemática a su forma simbólica Mathematica cuenta con la función "TraditionalForm".

Debido a que la forma estándar, es un intermedio entre la forma completa y la forma simbólica (tradicional), esta será la forma de notación que emplearemos para escribir la mayoría de las instrucciones en Mathematica. La forma estándar es lo suficientemente sencilla y clara como para escribir y comprender la mayoría de las instrucciones y al mismo tiempo es lo suficientemente rápida como para no retardar significativamente las operaciones.

1.3.1. Ejemplos

Calcule el valor de las siguientes expresiones escribiendo las instrucciones en su forma completa, simbólica y estándar.

$$10. \sqrt[3]{\frac{(3+4!)-6^{3.2}}{e^{6.5}+\cos(6.7^\circ)}}$$

```
Power[Divide[Subtract[Plus[3,Factorial[4]],Power[6,3.2]],Plus[Exp[6.5],
Cos[6.7*Degree]]],1/3]
0.375473+0.650338 i
```

$$\sqrt[3]{\frac{(3+4!)-6^{3.2}}{e^{6.5}+\cos(6.7^\circ)}}$$

```
0.375473+0.650338 i
```

```
(( (3+4!)-6^3.2)/(Exp[6.5]+Cos[6.7°]))^(1/3)
```

```
0.375473+0.650338 i
```

$$11. \frac{\sinh(6.5)\cosh(9.2)\tanh(9.3)}{45.2+6.3^{2.2}+6!}$$

```
Divide[Times[Sinh[6.5],Cosh[9.2],Tanh[9.3]],Plus[45.2,Power[6.3,2.2],
Factorial[6]]]
2000.78
```

$$\frac{\sinh(6.5)\cosh(9.2)\tanh(9.3)}{45.2+6.3^{2.2}+6!}$$

```
Sinh[6.5]*Cosh[9.2]*Tanh[9.3]/(45.2+6.3^2.2+6!)
2000.78
```

1.3.2. Ejercicios

Calcule el valor de las siguientes expresiones escribiendo las instrucciones en su forma completa, simbólica y estándar.

$$11. \frac{\sin^{-1}(0.35)+\cos^{-1}(0.89)+\tan^{-1}(5.45)}{3!e^{4/5}\pi} \quad 0.05292792928279825$$

$$12. \operatorname{Log}\left(\frac{4.5^{3.2}+9.8^{1.6}+e^{4.5}}{\sqrt[5]{6.32+2^{0.98}}}\right) \quad 2.217090924309038$$

2. VARIABLES Y REGLAS

2.1. Variables en Mathematica

Una variable es un sector reservado de memoria identificado por un nombre único. Como ya vimos en Mathematica podemos emplear los resultados de cálculos previos con "%..." y con "Out[#]", sin embargo, resulta más claro, sencillo, eficiente y versátil trabajar con variables.

En Mathematica existen dos tipos de variables: a) Las variables simbólicas (o símbolos), son aquellas que no tienen ningún dato asignado, estas variables se crean automáticamente cuando son empleadas por primera vez en una instrucción o expresión y normalmente se emplean en operaciones simbólicas (no existen este tipo de variables en Pascal); b) Las variables asignadas (o variables propiamente), son aquellas a las que se les asigna algún valor o expresión cuando son creadas y se comportan de manera similar a las variables en Pascal, sin embargo en Mathematica las variables pueden contener cualquier tipo de dato, desde datos simples como números, constantes, letras, etc., hasta matrices, listas, expresiones algebraicas, secuencias de instrucciones, programas, etc.

Como dijimos, las variables simbólicas se crean automáticamente cuando son empleadas por primera vez, por ejemplo al escribir la siguiente expresión (asumiendo que es la primera instrucción en el cuaderno), Mathematica crea automáticamente la variable simbólica "x" y realiza operaciones algebraicas con la misma:

```
6*x-2*x+x-9
-9+5 x
```

Para obtener información con relación a alguna variable podemos emplear la función "Information" o en forma abreviada "??", así si pedimos información con relación a "x" obtenemos:

```
Information[x]
Global`x
0
??x
Global`x
```

Entonces "x" es reconocida como una variable global, pero no tiene ningún valor asignado. Para obtener información resumida se debe emplear "?" en lugar de "??" o *Information[variable, LongForm->False]*. Si pedimos información con relación a una variable aún no creada, como "y" por ejemplo, obtenemos:

```
??y
Information::notfound : Symbol y not found. >>
```

Que como vemos no es reconocida como una variable. Information (o "??" o "?") puede ser empleada también para pedir información con relación a cualquier función de Mathematica, así por ejemplo podemos pedir información con relación a "Sqrt":

```
??Sqrt
Sqrt[z] or  $\sqrt{z}$  gives the square root of z. >>
Attributes[Sqrt]={Listable,NumericFunction,Protected}
```

O en su formato corto:

```
?Sqrt
```

Sqrt[z] or $\sqrt{z}$ gives the square root of z. >>

Es posible también emplear el comodín "*", en lugar de una o más letras, así por ejemplo para pedir información de todas las instrucciones (o variables) que comienzan con "Sin", escribimos:

```
?Sin*
```

Sin	SingularValueList
Sinc	SingularValues
SingleEvaluation	Sinh
SingleLetterItalics	SinhIntegral
SingleLetterStyle	SinIntegral

Por otra parte, las variables asignadas pueden ser creadas de dos formas: Mediante asignación inmediata (=) y mediante asignación retardada (:=).

Para la asignación inmediata se emplea el signo igual (=) de acuerdo al siguiente formato:

variable = expresión

En la asignación inmediata, la expresión es evaluada y el resultado es asignado a la variable, por ejemplo en la siguiente instrucción asignamos el resultado de " $\ln(3.4)+\cos(5.6)$ " a la variable "v1":

```
v1=Log[3.4]+Cos[5.6]
1.99934
```

Si ahora pedimos información con relación a "v1" obtenemos:

```
?v1
Global`v1
v1=1.99934
```

Vemos entonces que "v1" no sólo es una variable global, sino que además tiene asignado el valor 1.99934.

Para la asignación retardada se emplean dos puntos y el signo igual (:=), de acuerdo al siguiente formato:

variable := expresión

En la asignación retardada, la expresión permanece sin evaluar y es evaluada recién cuando se solicita el valor de la variable. Para comprender mejor la diferencia entre la asignación inmediata y la retardada asignemos la siguiente expresión: " $3\ln(5.4+3)-e^{1.4+1.6}+\cos(6.7-3)$ " a dos variables:

```
v2=3*Log[5.4+3]-Exp[1.4+1.6]+Cos[6.7-3]

v3:=3*Log[5.4+3]-Exp[1.4+1.6]+Cos[6.7-3]
```

Si pedimos el valor de estas variables obtenemos:

```
v2
-14.5489

v3
-14.5489
```

Aparentemente ambas variables tienen asignado el mismo valor, sin embargo, si pedimos información con relación a las mismas obtenemos:

```
?v2
```

```
Global`v2
v2=-14.5489
```

```
?v3
Global`v3
v3:=3 Log[5.4+3]-Exp[1.4+1.6]+Cos[6.7-3]
```

Vemos entonces que "v2" efectivamente tiene asignado el valor numérico "-14.5489", pero "v3" tiene asignada la expresión sin evaluar, por lo tanto, cada vez que se pide el valor de "v3" la expresión se evalúa en ese instante y se devuelve el resultado de dicha evaluación.

Una vez asignado un valor o expresión a una variable, esta asignación se mantiene hasta que es reemplazada por otra asignación o hasta que se quita explícitamente la misma. Para quitar una asignación se emplea el signo igual seguido de un punto (=.). Por ejemplo, la siguiente instrucción remueve la asignación de la variable "v1":

```
v1=.
```

Si ahora pedimos información con relación a "v1" obtenemos:

```
?v1
Global`v1
```

De modo que "v1" sigue siendo una variable global, pero ya no tiene ningún valor asignado. Alternativamente, se puede quitar el valor asignado a una o más variables con la función "Clear", por ejemplo la siguiente instrucción remueve las asignaciones de las variables "v2" y "v3":

```
Clear[v2,v3]
```

Si pedimos información con relación a "v2" y "v3" obtenemos:

```
?v2
Global`v2
```

```
?v3
Global`v3
```

Entonces tanto "v2" como "v3" siguen siendo variables globales, pero ya no tienen valores asignados (son entonces variable simbólicas). El olvidar una asignación hecha, es uno de los errores que más frecuentemente se comete al trabajar con Mathematica, razón por la cual es conveniente acostumbrarse a quitar las asignaciones tan pronto como ya no sean necesarias.

Para eliminar definitivamente una variable se emplea la función "Remove", así la siguiente instrucción elimina definitivamente las variables "v1", "v2" y "v3":

```
Remove[v1,v2,v3]
```

Si pedimos información con relación a "v1", "v2" y "v3" obtenemos:

```
?v1
Information::notfound: Symbol v1 not found. >>
```

```
?v2
Information::notfound: Symbol v2 not found. >>
```

```
?v3
Information::notfound: Symbol v3 not found. >>
```

Por lo tanto "v1", "v2" y "v3" ya no son más variables de Mathematica.

En Mathematica es posible asignar un mismo valor a más de una variable a la vez, así por ejemplo para asignar el valor 6.5 a las variables "a", "b", "c", "d", y "e" escribimos:

```
a=b=c=d=e=6.5
6.5
```

Si pedimos información con relación a cualquiera de estas variables, por ejemplo "c" obtenemos:

```
?c
Global`c
c=6.5
```

Que como se ve es una variable global y tiene asignado el valor 6.5.

Para ejecutar una instrucción sin que se muestre ninguna salida, se escribe un punto y coma (;) después de la misma. Por ejemplo en la siguiente instrucción asignamos el resultado de $\sinh^{-1}(6.5)/\cosh^{-1}(3.2+5.3)$ a la variable "x1" pero no se muestra el valor de la expresión:

```
x1=ArcSinh[6.5]/ArcCosh[3.2+5.3];
```

Por supuesto el valor se encuentra en la variable "x1":

```
x1
0.9085
```

El punto y coma (;) se emplea también para ejecutar dos o más instrucciones consecutivas, así por ejemplo podemos asignar los valores 1, 2, 3 y 4 a las variables x1, x2, x3 y x4, en una sola orden:

```
x1=1;x2=2;x3=3;x4=4;
```

Para obtener una lista de los nombres de las variables existentes se puede emplear la instrucción "Names", por ejemplo para obtener la lista de las variables globales (creadas por el usuario) escribimos:

```
Names["Global`*"]
{a,b,c,d,e,x,x1,x2,x3,x4}
```

Donde "Global`" identifica a las variables globales y "*" es un comodín que representa a cualquier letra o secuencia de letras. Se puede pedir también información más específica, por ejemplo con la siguiente instrucción pedimos la lista de todas las variables globales cuyos nombres comienzan con "x":

```
Names["Global`x*"]
{x,x1,x2,x3,x4}
```

Para pedir información con relación a todas las variables, tanto globales como del sistema, simplemente omitimos "Global`", por ejemplo con la siguiente instrucción se pide la lista de todas las variables que comienzan con "Sin":

```
Names["Sin*"]
{Sin,Sinc,SingleEvaluation,SingleLetterItalics,SingleLetterStyle,SingularValueDecomposition,SingularValueList,SingularValues,Sinh,SinhIntegral,SinIntegral}
```

Si queremos pedir información específicamente con relación a las variables del sistema, entonces debemos anteponer "System`" al criterio de búsqueda, así por ejemplo, con la siguiente instrucción se pide información de todas las variables del sistema que comienzan con "X":

```
Names["System`X*"]
{XMLElement,XMLElement,XMLObject,Xor}
```

2.1.1. Ejemplos

1. Pida información resumida de la función "Cos".

```
?Cos
Cos[z] gives the cosine of z.
O alternativamente:
Information[Cos,LongForm->False]
Cos[z] gives the cosine of z.
```

2. Pida información completa de la función "Cos".

```
??Cos
Cos[z] gives the cosine of z.
Attributes[Cos]={Listable,NumericFunction,Protected}
O alternativamente:
Information[Cos]
Cos[z] gives the cosine of z.
Attributes[Cos]={Listable,NumericFunction,Protected}
```

3. Empleando asignación inmediata asigne la siguiente expresión a la variable "f1", pida información resumida de la variable y calcule el valor de la variable para "x=3.2".

$$f1 = \sqrt[5]{\frac{(3x+5x^2-2x) - \ln(x+3+x) - \sinh^{-1}(2x+4x)}{e^{\frac{2x+3-x}{6x^2+8x-2-3x^2}}}}$$

```
f1=(( (3*x+5*x^2-2*x)-Log[x+3+x]-ArcSinh[2*x+4*x])/Exp[(2*x+3-x)/(6*x^2+8*x-2-3*x^2)])^(1/5);
```

```
?f1
```

```
Global`f1
```

```
f1: ( ( 3 x
| 2 8 x 3 x^2 | x+ 5 x^2 · ArcSinh[ 6 x ] · Log[ 3+ 2 x ] ) )^1/5
```

```
x=3.2;
```

```
f1
```

```
2.12448
```

4. Empleando asignación retardada asigne la expresión del ejercicio anterior a la variable "f2", pida información resumida de la variable y calcule el valor de la expresión para "x=3.2".

```
f2:=(( (3*x+5*x^2-2*x)-Log[x+3+x]-ArcSinh[2*x+4*x])/Exp[(2*x+3-x)/(6*x^2+8*x-2-3*x^2)])^(1/5);
```

```
?f2
```

```
Global`f2
```

```
f2: ( ( 3 x+ 5 x^2 · 2 x | · Log[ x+ 3+ x ] · ArcSinh[ 2 x+ 4 x ] ) )^1/5
| 6 x^2+ 8 x 2 3 x^2
```

```
x=3.2;
```

```
f2
```

```
2.12448
```

5. Asigne (asignación inmediata) el seno de 6.873 a las variables "x1", "x2" y "x3", luego asigne el valor de la expresión: $\sin(x_1) + \cos(x_2 + x_3) - \ln(x_3)$ a la variable x4, elimine las variables "x1", "x2" y "x3" y pida información resumida con relación a la variable "x4".

```
x1=x2=x3=Sin[6.873];
x4=Sin[x1]+Cos[x2+x3]-Log[x3];
Remove[x1,x2,x3]
```

```
?x4
```

```
Global`x4
```

```
x4=1.55708
```

6. Pida información de todas las variables globales que comienzan con "f".

```
Names["Global`f*"]
```

```
{f1,f2}
```

7. Pida información de todas las variables del sistema que comienzan con "Facto".

```
Names["System`Facto*"]
```

```
{Factor,FactorComplete,Factorial,Factorial2,FactorInteger,FactorList,FactorSquareFree,FactorSquareFreeList,FactorTerms,FactorTermsList}
```

8. Pida información de todas las variables (tanto globales como del sistema) que comienzan con "Cos".

```
Names["Cos*"]
```

```
{Cos,Cosh,CoshIntegral,CosineDistance,CosIntegral}
```

2.1.2. Ejercicios

1. Pida información resumida de la función "FindRoot".

2. Pida información completa de la función "FindRoot".

3. Asigne (asignación inmediata) a las variables "x", "y", "z" el resultado de: $\sqrt[3]{\ln(4.3)-e^{0.5}}$.

4. Asigne (asignación inmediata) los resultados de: $\ln(6.539)$, 0.932^2 y $12!$ a las variables "r1", "r2" y "r3".

5. Borre el valor asignado a las variables "x", "y" y "z".

6. Elimine las variables "r1", "r2" y "r3".

7. Asigne el resultado de la siguiente expresión a la variable "r", pida información resumida de dicha variable y quite el valor asignado a la misma.

$$\frac{\sin^{-1}(0.35)+\cos^{-1}(0.89)+\tan^{-1}(5.45)}{3!e^{4/5}\pi}$$

8. Asigne el resultado de la expresión del ejercicio anterior a la variable "z", pida información completa de la variable y elimine dicha variable.

9. Asigne (asignación inmediata) la siguiente expresión a la variable "f3", pida información resumida de la variable "f3", calcule el valor de la expresión para "x=1.2" y quite el valor asignado a la variable "x".

$$f3 = \frac{\ln(3x+2x+5x^2-4)+\tan(5x+7x)-\tan^{-1}(2x+3)}{\sqrt{\sinh(6x+2-4x)+\cosh(5x+3x-8)}}$$

10. Asigne (asignación retardada) la expresión del ejercicio anterior a la variable "f4", pida información completa de la variable "f3", calcule el valor de la expresión para "x=1.8" y elimine la variable "x".

11. Pida la lista de las variables globales.

12. Pida la lista de las variables del sistema que comienzan con "Tan".

13. Pida la lista de las variables globales y del sistema que comienzan con "Ru".

2.2. Reglas

Como ya se dijo en el anterior subtítulo, cuando se asigna un valor o expresión a una variable, la misma conserva dicha asignación hasta que es reemplazada por otra, o es explícitamente quitada (con `=.` o `Clear`) o la variable es eliminada (con `Remove`). Sin embargo en muchas ocasiones sólo se quiere asignar temporalmente un valor (o expresión) a una variable simbólica (o expresión), en tales casos es más eficiente, flexible y seguro emplear las denominadas "reglas".

En Mathematica una "regla" tiene la siguiente forma:

expresión_1 -> expresión_2

Y establece que la "expresión_1" deberá ser transformada en (reemplazada por) la "expresión_2". "Expresión_1" y "expresión_2" pueden ser variables simbólicas, valores numéricos o expresiones complejas. La "expresión_2" se evalúa inmediatamente y el resultado de dicha evaluación reemplaza a la "expresión_1".

Las reglas en Mathematica son muy potentes y flexibles, en realidad la asignación inmediata (`=`), es tan solo la regla "*li->ld*", la cual establece que el lado izquierdo (*li*) será reemplazado por el lado derecho (*ld*) en todo lugar donde aparezca el lado izquierdo.

Para aplicar una o más reglas a una expresión se emplea la instrucción `/.` (`ReplaceAll`), de acuerdo al siguiente formato:

expresión /. reglas

Por ejemplo, si asignamos la ecuación x^3+2x^2+3x+4 a la variable `fx`:

```
fx=x^3+2*x^2+3*x+4;
```

Podemos calcular el valor de la función para $x=10$, con la regla `x->10`:

```
fx/.x->10
1234
```

Pero también podemos calcular el valor de la función para $x=3.5$:

```
fx/.x->3.5
81.875
```

O, si empleamos listas (las cuales estudiaremos un poco más adelante), podemos calcular el valor de la función para varios valores de "x", por ejemplo $x=-2,-1.5,-1,-0.5,0,1,1.5,2,3$:

```
fx/.x->{-2,-1.5,-1,-0.5,0,1,1.5,2,3}
{-2,0.625,2,2.875,4,10,16.375,26,58}
```

Pero también podemos reemplazar el valor de "x" por otra variable simbólica, por ejemplo "z":

```
fx/.x->z
4+ 3 z+ 2 z^2+ z^3
```

O una expresión, por ejemplo z^2+3z-5 :

```
fx/.x->z^2+3*z-5
4+ 3 | · 5+ 3 z+ z^2 | + 2 | · 5+ 3 z+ z^2 |^2 | + 5+ 3 z+ z^2 |^3
```

Sin embargo, en ningún momento se cambia realmente el valor de la variable simbólica "x", por lo que si pedimos información de "x" obtenemos:

```
?x
Global`x
```

Y si pedimos el valor de la variable "fx", obtenemos:

```
fx
4 3 x 2 x^2 x^3
```

Por el contrario si empleamos una asignación inmediata, por ejemplo x=10:

```
x=10
10
```

Y pedimos el valor de la variable "fx", obtenemos:

```
fx
1234
```

Ahora si queremos calcular el valor de la función para x=3.5, debemos hacer una nueva asignación inmediata y pedir el valor de "fx":

```
x=3.5;fx
81.875
```

Mientras no cambiemos el valor asignado a "x", siempre que pidamos el valor de "fx" nos devolverá "81.875".

Con las reglas no solo es posible reemplazar el valor de una variable, sino de expresiones más complejas, por ejemplo podemos cambiar la función seno de la siguiente expresión por coseno:

```
Sin[x]+3*Sin[x]^2/.Sin[x]->Cos[z]
Cos[z] + 3 Cos[z]^2
```

Y podemos evaluar este resultado para "z=3.4":

```
./.z->3.4
1.8373
```

Por supuesto es posible llevar a cabo todo el proceso en una sola orden:

```
Sin[x]+3*Sin[x]^2/.Sin[x]->Cos[z]/.z->3.4
1.8373
```

Podemos reemplazar también una expresión por otra, como en el siguiente ejemplo, donde se reemplaza "x+y" por "w²":

```
Log[x+y]-Cos[x+y]/.x+y-> w^2
Cos[w^2] + Log[w^2]
```

Y en general es posible llevar a cabo casi cualquier tipo de reemplazo.

Al igual que existe la asignación inmediata y la asignación retardada, existen las reglas inmediatas, que son las estudiadas hasta ahora, y las reglas retardadas, las cuales tienen la siguiente forma:

expresión_1 :> *expresión_2*

Una regla retardada establece que la "expresión_2" será evaluada y reemplazará a la "expresión_1" sólo cuando la regla sea usada, a diferencia de la regla inmediata, en la cual la "expresión_2" es evaluada inmediatamente (antes de que la regla sea usada). Para comprender mejor la diferencia entre una regla retardada y una inmediata, veamos el siguiente ejemplo:

```
{x,x,x}/.x:>RandomReal[]
```

```
{0.674076,0.554287,0.737815}
```

Como se puede observar, cada "x" en la lista es reemplazada por un número aleatorio diferente, esto ocurre así porque la función "RandomReal" es evaluada cada vez que la regla es usada, y como es usada 3 veces, se generan 3 números aleatorios diferentes. Si en lugar de una regla retardada empleamos una regla inmediata obtenemos:

```
{x,x,x}/.x->RandomReal[]
{0.973061,0.973061,0.973061}
```

En este caso todas las "x" en la lista son reemplazadas por un mismo valor, esto ocurre así porque la función "RandomReal" es evaluada antes que la regla sea usada, generando un número aleatorio, luego ese número (ya generado) reemplaza a "x" cada vez que la regla es usada.

Al igual que la asignación inmediata es un caso especial de una regla inmediata, una asignación retardada (:=) es tan sólo la regla "*li:>ld*", la cual establece que el lado derecho será evaluado y reemplazará al lado izquierdo cada vez que aparezca el lado izquierdo en una expresión.

2.2.1. Ejemplos

9. Asigne la siguiente expresión a la variable "g1", calcule el valor de la expresión para "x=3.2", reemplace "x" por "2y+3z" y calcule los valores de la expresión resultante para "y" igual a 0.2, 0.3, 0.5, 0.8 y valores de "z" iguales a 1.2, 2.1, 1.3, 1.4. Finalmente quite el valor asignado a la variable "g1".

$$\text{Log}\left(\frac{x^{3.2} + x^{1.6} + e^x}{\sqrt[5]{6x + x^{0.98}}}\right)$$

```
g1=Log[10, (x^3.2+x^1.6+Exp[x])/(6*x+x^0.98)^(1/5)];
g1/.x->3.2
1.58945
g1/.x->2*y+3*z
Log[ $\frac{(2y)^{3.2} (3z)^{1.6} (2y+3z)^{3.2}}{(2y+3z)^{0.98} (6(2y+3z))^{1/5}}$ ]
Log[10]
%/.{y->{0.2,0.3,0.5,0.8},z->{1.2,2.1,1.3,1.4}}
{3,5,7}
g1=.
```

10. Asigne la siguiente expresión a la variable "g2", calcule los valores de la expresión para "x" igual a 2.1, 3.2, 4.3, 5.2 y "y" igual a 1.3, 1.7, 1.9, 2.2. Reemplace "x+y" por "z³" y asigne la expresión resultante a la variable "g3", calcule los valores de "g3" para valores de "z" iguales a 0.4, 0.8, 1.2, 4.6 y 7.9. Finalmente elimine las variables "g2" y "g3".

$$\left(\sin(x+y) - \frac{\sec(2(x+y))}{\sinh(3(x+y))}\right)^{0.34}$$

```
g2=(Sin[x+y]-Sec[2*(x+y)]/Sinh[3*(x+y)])^0.34;
g2/.{x->{2.1,3.2,4.3,5.2},y->{1.3,1.7,1.9,2.2}}
{0.302978 +0.551115 i,0.478863 +0.871048 i,0.206762 +0.376099 i,0.96434}
g3=g2/.(x+y)->z^3
(-Csch[3 z^3] Sec[2 z^3]+Sin[z^3])^0.34
g3/.z->{0.4,0.8,1.2,4.6,7.9}
```

```
{0.841374 +1.53045 i,0.346145 +0.629635 i,0.999816,0.369162,0.568444}
Remove[g2,g3]
```

11. Asigne la siguiente expresión a la variable "g4", calcule el valor de la expresión para "x=0.2". Reemplace "x" por "y²+z³" y asigne la expresión resultante a la variable "g5". Calcule el valor de "g5" para "y=0.3", "z=0.6". Reemplace en "g5", "y" por "r", "z" por "s" y asigne la expresión resultante a la variable "g6", calcule los valores de "g6" para "r" igual a 0.2, 0.4, 0.5, 0.6 y "s=0.5". Finalmente quite los valores asignados a las variables "g4", "g5" y "g6".

$$\frac{(\sinh(x))^{3.2} + (\cosh(2x))^{5.4}}{e^{\tan(4x)}}$$

```
g4=Exp[(Sinh[x]^3.2+Cosh[2*x]^5.4)/Tan[4*x]];
g4/.x->0.2
4.41632
g5=g4/.x->y^2+z^3
| Cot[4|y^2 z^3|] | Cosh[2|y^2 z^3|]^5.4 Sinh[y^2 z^3]^3.2 |
g5/.{y->0.3,z->0.6}
2.57702
g6=g5/.{y->r,z->s}
| Cot[4|r^2 s^3|] | Cosh[2|r^2 s^3|]^5.4 Sinh[r^2 s^3]^3.2 |
g6/.{r->{0.2,0.4,0.5,0.6},s->0.5}
{5.60732,2.90458,1.33556,0.0271527}
Clear[g4,g5,g6]
```

2.2.2. Ejercicios

14. Asigne la siguiente expresión a la variable "f1", calcule el valor de la expresión para "x=10", reemplace en la expresión la "x" por "z²+3", calcule el valor de la expresión resultante para "z=3.2" y quite el valor asignado a "f1".

$$\ln(x) + x^{5.6} + \sqrt{x-2}$$

15. Asigne la siguiente expresión a "f2", calcule los valores de la expresión para "y" igual a 1.1, 2.3, 3.5 y 6.7. Reemplace en la expresión "y" por "3x+2", calcule los valores de la expresión resultante para "x" igual a 3.1, 4.2 y 5.1 y elimine la variable "f2".

$$\sqrt[4]{\frac{e^y + 5y^2}{(y+3)^{2.3}}}$$

16. Asigne el valor de la siguiente expresión a la variable "f3", calcule el valor de la expresión para "x=3.1", "y=6.2", calcule el valor de la expresión para "x+y=2.32", reemplace "x+y" por "w²" y asigne la expresión resultante a la variable "f4", calcule los valores de la expresión para "w" igual a 1.2, 2.3, 3.5, 4.1 y 6.2. Finalmente elimine las variables "f3" y "f4".

$$\frac{\sin(x+y) + \cos(x+y)}{(x+y)^{0.23}}$$

17. Asigne la siguiente expresión a la variable "f5", calcule los valores de la misma para valores de "x" iguales a 1.2, 2.2, 3.3, 4.4 y valores de "y" iguales a 0.2, 0.5, 0.8, 1.2. Reemplace "x" por "z²" y asigne la expresión resultante a la variable "f6", calcule los valores de "f6"

para valores de "y" iguales a 2.2, 2.5, 3.2 y valores de "z" iguales a 1.1, 1.3, 1.5. Finalmente remueva los valores asignados a "f5" y "f6".

$$\ln(x+y)+3x^2-\frac{9}{x+2}$$

18. Asigne la siguiente expresión a la variable "f7", calcule el valor de la función para valores de "x" iguales a 3.2, 5.2, 7.1 y "y=6.2". Reemplace la variable "y" por " x^2+3x-2 " y asigne la expresión resultante a la variable "f8", calcule los valores de "f8" para "x" igual a 1.2, 1.5, 1.9, 2.1. Reemplace "x" por " $y+z^2$ " y asigne la expresión resultante a "f9", calcule el valor de "f9" para valores de "y" iguales a 2.3, 4.3, 6.3 y valores de "z" iguales a 1.1, 1.4, 1.6. Finalmente elimine las variables "f7", "f8" y "f9".

$$3y+2x^2+9x-8$$

3. LISTAS Y PATRONES

3.1. Listas

Cuando se trabaja con grupos de elementos en lugar de un elemento a la vez, las listas simplifican y facilitan el trabajo en Mathematica.

Una lista (List) es un conjunto de elementos encerrados entre llaves y separados por comas:

$$\{e_1, e_2, e_3, \dots, e_n\}$$

Las listas en Mathematica son muy flexibles y pueden ser tratadas como vectores, matrices, conjuntos y colecciones.

Supongamos por ejemplo que necesitamos calcular la raíz cuadrada de los números 3.1, 6.7, 9.2 y 10.4, sin listas sería necesario escribir las siguientes instrucciones:

```
Sqrt[3.1]
1.76068
Sqrt[6.7]
2.58844
Sqrt[9.2]
3.03315
Sqrt[10.4]
3.2249
```

Con listas es suficiente una instrucción:

```
Sqrt[{3.1,6.7,9.2,10.4}]
{1.76068,2.58844,3.03315,3.2249}
```

O:

```
Sqrt[x]/.x->{3.1,6.7,9.2,10.4}
{1.76068,2.58844,3.03315,3.2249}
```

O también:

```
{3.1,6.7,9.2,10.4} // Sqrt
{1.76068,2.58844,3.03315,3.2249}
```

Aunque esta es la forma en que mayormente se emplean las listas, Mathematica cuenta con varias funciones especializadas en el trabajo con listas, algunas de las cuales las estudiaremos en este capítulo y otras en capítulos posteriores cuando trabajemos con vectores y matrices.

Para ejemplificar el uso de las funciones especializadas en listas crearemos una lista y asignaremos la misma a la variable "L":

```
L={2,4,1,6,3,8,7,2,5,4,6,3,1,5,4,6};
```

Para determinar el número de elementos en la lista se emplea la función "Length[lista]", así el número de elementos en la lista "L" es:

```
Length[L]
16
```

El primer elemento de la lista puede ser extraído con "First[lista]":

```
First[L]
2
```

El último elemento de la lista puede ser extraído con "Last[lista]":

```
Last[L]
6
```

La lista sin el último elemento se obtiene con "Most[lista]":

```
Most[L]
{2,4,1,6,3,8,7,2,5,4,6,3,1,5,4}
```

La lista sin el primer elemento se obtiene con "Rest[lista]":

```
Rest[L]
{4,1,6,3,8,7,2,5,4,6,3,1,5,4,6}
```

Los primeros "n" elementos de la lista se obtienen con "Take[lista,n]", así los primeros 6 elementos de la lista son:

```
Take[L,6]
{2,4,1,6,3,8}
```

Si el número "n" es negativo, entonces se obtienen los últimos "n" elementos, así los últimos 4 elementos de la lista son:

```
Take[L,-4]
{1,5,4,6}
```

Si en lugar de un número se emplea una lista "{n₁,n₂}", entonces se obtiene la lista con los elementos que se encuentran desde la posición "n₁" hasta "n₂", así los elementos 6 al 10 de la lista son:

```
Take[L,{6,10}]
{8,7,2,5,4}
```

La lista sin los primeros "n" elementos se obtiene con "Drop[lista,n]", así la lista sin los primeros 5 elementos es:

```
Drop[L,5]
{8,7,2,5,4,6,3,1,5,4,6}
```

Si el número "n" es negativo, entonces se obtiene la lista sin los últimos "n" elementos, así la lista sin los últimos 7 elementos es:

```
Drop[L,-7]
{2,4,1,6,3,8,7,2,5}
```

Si en lugar de un número se emplea una lista "{n₁,n₂}", entonces se obtienen la lista sin los elementos que se encuentran desde la posición "n₁" hasta "n₂", así la lista sin los elementos 4 al 12 son:

```
Drop[L,{4,12}]
{2,4,1,1,5,4,6}
```

El elemento "n" de la lista se obtiene con "Part[lista,n]" o "lista[[n]]", así el séptimo elemento de la lista es:

```
Part[L,8]
2
```

o

```
L[[8]]
2
```

Si el número "n" es negativo, entonces se obtiene el enésimo elemento contando desde el último, así el quinto elemento contando desde el último es:

```
L[[-5]]
3
```


Si en lugar de un número, se emplean dos números separados por dos puntos y comas: "m;;n", entonces se obtienen los elementos que van desde la posición "m" hasta la posición "n" (inclusive), así los elementos 2 al 8 son:

```
L[[2;;8]]
{4,1,6,3,8,7,2}
```

Si en lugar de un número se emplea una lista con 2 o más números: "{n₁,n₂,...}", entonces se obtienen los elementos que se encuentran en las posiciones "n₁", "n₂", etc., así los elementos 3, 9, 12 y 15 son:

```
L[[{3,9,12,15}]]
{1,5,3,4}
```

La posición en la que se encuentra un determinado elemento se obtiene con "Position[lista,elemento]", así por ejemplo la posición en la que se encuentra el número 7 es:

```
Position[L,7]
{{7}}
```

Por lo tanto el número 7 se encuentra en la posición 7. Si existen 2 o más elementos iguales "Position" devuelve las posiciones de todos ellos, así por ejemplo las posiciones en las que se encuentra el número 4 son:

```
Position[L,4]
{{2},{10},{15}}
```

Si el elemento no se encuentra en la lista "Position" devuelve una lista vacía, por ejemplo si pedimos la posición del elemento 9 obtenemos:

```
Position[L,9]
{}
```

Para determinar si un elemento pertenece o no a una lista se emplea "MemberQ[lista,elemento]", el cual devuelve "True" (verdadero) si el elemento pertenece a la lista y "False" (falso) en caso contrario, así podemos determinar si el número 5 pertenece a la lista:

```
MemberQ[L,5]
True
```

Para el número 9 obtenemos:

```
MemberQ[L,9]
False
```

La prueba contraria, es decir determinar si un elemento no pertenece a una lista se la realiza con "FreeQ[lista,elemento]", así si preguntamos si el número 5 no pertenece a la lista obtenemos:

```
FreeQ[L,5]
False
```

Y para el número 9 obtenemos:

```
FreeQ[L,9]
True
```

El número de veces que un elemento se encuentra en una lista puede ser determinado con "Count[lista,elemento]", así el número de veces que el número 6 se encuentra en la lista es:

```
Count[L,6]
3
```

Se puede añadir un elemento al principio de una lista con "Prepend[lista,elemento]", así por ejemplo para añadir el número 9 al principio de la lista escribimos:

```
Prepend[L,9]
{9,2,4,1,6,3,8,7,2,5,4,6,3,1,5,4,6}
```

Para añadir un elemento al final de la lista se emplea "Append[lista,elemento]", así por ejemplo para añadir el número 12 al final de la lista escribimos:

```
Append[L,12]
{2,4,1,6,3,8,7,2,5,4,6,3,1,5,4,6,12}
```

Para insertar un elemento en una posición específica se emplea "Insert[lista,elemento,posición]", así por ejemplo para insertar el número 10 en la posición 5 de la lista escribimos:

```
Insert[L,10,5]
{2,4,1,6,10,3,8,7,2,5,4,6,3,1,5,4,6}
```

Si la posición es un número negativo, el elemento se inserta en esa posición pero contando desde el último elemento, así por ejemplo para insertar el número 10 en la posición 5, contando desde el final, escribimos:

```
Insert[L,10,-5]
{2,4,1,6,3,8,7,2,5,4,6,3,10,1,5,4,6}
```

Para intercalar un elemento entre los elementos de la lista se emplea "Riffle[Lista,elemento]", así por ejemplo para intercalar el número 0 entre los elementos de la lista escribimos:

```
Riffle[L,0]
{2,0,4,0,1,0,6,0,3,0,8,0,7,0,2,0,5,0,4,0,6,0,3,0,1,0,5,0,4,0,6}
```

Para quitar un elemento de la lista en una posición específica se emplea "Delete[lista,posición]", así por ejemplo para eliminar el elemento que se encuentra en la posición 7 (el número 7) escribimos:

```
Delete[L,7]
{2,4,1,6,3,8,2,5,4,6,3,1,5,4,6}
```

Al igual que "Insert", si la posición es negativa, se elimina el elemento que se encuentra en esa posición, pero contando desde el final, así para eliminar el elemento que se encuentra en la posición -3, contando desde el final (el número 5) escribimos:

```
Delete[L,-3]
{2,4,1,6,3,8,7,2,5,4,6,3,1,4,6}
```

Para reemplazar un elemento por otro se emplea ReplacePart[lista,posición->elemento]. Por ejemplo para reemplazar el quinto elemento por el número 10 escribimos:

```
ReplacePart[L,5->10]
{2,4,1,6,10,8,7,2,5,4,6,3,1,5,4,6}
```

ReplacePart genera una nueva lista con el resultado, sin embargo, si se quiere modificar un elemento de la lista original se puede emplear el formato: "Lista[[posición]]=elemento", así por ejemplo para reemplazar el quinto elemento de la lista original por el número 10 escribimos:

```
L[[5]]=10
10
```

Si ahora mostramos la lista "L" obtenemos:

```
L
{2,4,1,6,10,8,7,2,5,4,6,3,1,5,4,6}
```

Donde como se puede ver el quinto elemento ha sido reemplazado por el número 10. Esta forma de operar es válida también para listas donde los elementos son a su vez listas (matrices).

Los elementos de una lista pueden ser ordenados ascendentemente con "Sort[lista]", así para ordenar la lista "L" escribimos:

```
Sort[L]
{1,1,2,2,3,4,4,4,5,5,6,6,6,7,8,10}
```

Para ordenar una lista quitando los elementos duplicados se emplea "Union[lista]", así para ordenar la lista "L" quitando los elementos duplicados escribimos:

```
Union[L]
{1,2,3,4,5,6,7,8,10}
```

Para invertir el orden de los elementos de una lista se emplea "Reverse[lista]", así para invertir los elementos de la lista "L" escribimos:

```
Reverse[L]
{6,4,5,1,3,6,4,5,2,7,8,10,6,1,4,2}
```

Y para ordenar la lista "L" en orden inverso:

```
Reverse[Sort[L]]
{10,8,7,6,6,6,5,5,4,4,4,3,2,2,1,1}
```

Para rotar los elementos de una lista "n" lugares a la izquierda se emplea "RotateLeft[lista,n]", por ejemplo para rotar los elementos de la lista "L" 3 lugares hacia la izquierda escribimos:

```
RotateLeft[L,3]
{6,10,8,7,2,5,4,6,3,1,5,4,6,2,4,1}
```

Y para rotar una lista "n" lugares en el sentido contrario (a la derecha) se emplea "RotateRight[lista,n]", por ejemplo para rotar los elementos de la lista "L" 5 lugares hacia la derecha escribimos:

```
RotateRight[L,5]
{3,1,5,4,6,2,4,1,6,10,8,7,2,5,4,6}
```

Para rellenar una lista a la izquierda con un elemento "x" de manera que su número de elementos sea "n" se emplea: "PadLeft[lista,n,x]", así para rellenar la lista "L" con el número 0 de manera que tenga 30 elementos escribimos:

```
PadLeft[L,30,0]
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,4,1,6,10,8,7,2,5,4,6,3,1,5,4,6}
```

Para rellenar la lista a la derecha se emplea "PadRight[lista,n,x]", así para rellenar la lista "L" con el número 9 de manera que tenga 25 elementos escribimos:

```
PadRight[L,25,9]
{2,4,1,6,10,8,7,2,5,4,6,3,1,5,4,6,9,9,9,9,9,9,9,9,9,9}
```

El menor elemento de una lista se obtiene con "Min[lista]", por ejemplo el menor elemento de la lista "L" es:

```
Min[L]
1
```

El mayor elemento de una lista se obtiene con "Max[lista]", así el mayor elemento de la lista "L" es:

```
Max[L]
10
```

Una lista con elementos consecutivos, o separados por un incremento, se crea con "Range[valor_inicial,valor_final,incremento]". Si sólo se escribe el valor final "Range" genera una lista que va desde uno hasta dicho valor, así para generar una lista con elementos del 1 al 10 escribimos:

```
Range[10]
{1,2,3,4,5,6,7,8,9,10}
```

Para generar una lista con elementos que van del 5 al 20 escribimos:

```
Range[5,20]
{5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20}
```

Y para generar una lista con los números impares del 7 al 31:

```
Range[7,31,2]
{7,9,11,13,15,17,19,21,23,25,27,29,31}
```

Si en lugar de una lista de números se quiere generar una lista con letras (o caracteres) se emplea "CharacterRange[car₁,car₂]", que genera una lista con los caracteres que van desde "car₁" hasta "car₂", por ejemplo para generar una lista con las letras desde la "A" hasta la "Z" escribimos:

```
CharacterRange["A","Z"]
{A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z}
```

Una lista (o matriz) que tiene "n" veces el mismo elemento "x" se genera con "ConstantArray[x,n]", así para generar una lista con 20 elementos "a" escribimos:

```
ConstantArray[a,20]
{a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a,a}
```

Una lista (o matriz) con "n" elementos correspondientes a los valores consecutivos de una función se genera con "Array[función,n]", así por ejemplo un vector con los factoriales del 1 al 6 se genera con:

```
Array[Factorial,6]
{1,2,6,24,120,720}
```

Una lista (o matriz) generada con los resultados de una "expresión", cuando la misma es evaluada mientras la variable "i" va desde un valor inicial "i_{min}" hasta uno final "i_{max}", se genera con "Table[expresion,{i,i_{min},i_{max},incr}]". Por ejemplo un vector con los valores de la función seno cuando el ángulo varía desde π hasta 2π con incrementos de 0.1 se genera con:

```
Table[Sin[x],{x,Pi,2*Pi,0.1}]
{1.22465*10^-16,-0.0998334,-0.198669,-0.29552,-0.389418,-0.479426,-
0.564642,-0.644218,-0.717356,-0.783327,-0.841471,-0.891207,-0.932039,-
0.963558,-0.98545,-0.997495,-0.999574,-0.991665,-0.973848,-0.9463,-
0.909297,-0.863209,-0.808496,-0.745705,-0.675463,-0.598472,-0.515501,-
0.42738,-0.334988,-0.239249,-0.14112,-0.0415807}
```

Si sólo se escribe el valor final, la "expresión" es evaluada desde 1 hasta dicho valor, así para generar un vector con 7 veces el coseno de 0.3 escribimos:

```
Table[Cos[0.3],{7}]
{0.955336,0.955336,0.955336,0.955336,0.955336,0.955336,0.955336}
```

Si sólo se escriben la variable y el valor final, la "expresión" es evaluada mientras la variable incrementa su valor desde 1 hasta el valor final.

Por ejemplo podemos volver a generar una lista con los factoriales del 1 al 6 empleando Table:

```
Table[x!,{x,6}]
{1,2,6,24,120,720}
```

Si se escribe la variable y los límites, la expresión es evaluada mientras la variable incrementa su valor de uno en uno desde el límite inicial hasta el final. Por ejemplo para generar un vector con los valores del coseno desde 45 hasta 60° escribimos:

```
Table[Cos[a °],{a,45.,60.}]
{0.707107,0.694658,0.681998,0.669131,0.656059,0.642788,0.62932,0.615661,0.601815,0.587785,0.573576,0.559193,0.544639,0.529919,0.515038,0.5}
```

3.1.1. Ejemplos

1. Calcule la tangente de los siguientes ángulos: 10.°, 20.°, 45.°, 60.°, 80.°, 120.° y 200.°, empleando una lista como: parámetro directo, una regla y parámetro posfijo.

```
L={10.,20.,45.,60.,80.,120.,200.}°
{0.174533,0.349066,0.785398,1.0472,1.39626,2.0944,3.49066}
Sin[L]
{0.173648,0.34202,0.707107,0.866025,0.984808,0.866025,-0.34202}
Sin[x]/.x->L
{0.173648,0.34202,0.707107,0.866025,0.984808,0.866025,-0.34202}
L//Sin
{0.173648,0.34202,0.707107,0.866025,0.984808,0.866025,-0.34202}
```

2. Empleando RandomInteger con una tabla, una regla y por si solo genere una lista con 20 elementos aleatorios comprendidos entre 1 y 30.

```
Table[RandomInteger[{1,30}],{20}]
{7,25,17,30,1,29,2,12,1,10,3,25,4,11,14,18,22,2,1,16}
ConstantArray[x,20]/.x->RandomInteger[{1,30}]
{30,16,9,2,26,10,25,1,12,2,15,2,13,13,29,6,9,20,23,18}
RandomInteger[{1,30},20]
{28,29,14,20,6,19,1,1,3,10,15,16,24,5,28,30,11,9,22,28}
```

3. Genere una lista con 25 números enteros aleatorios comprendidos entre 1 y 50, luego extraiga el primer elemento de la lista, el último elemento de la lista, los tres primeros elementos de la lista, los cinco últimos elementos de la lista, los elementos que se encuentran entre las posiciones 10 a la 20 y ordene la lista en orden ascendente y descendente.

```
L=RandomInteger[{1,50},25]
O también:
L=Table[RandomInteger[{1,50}],{25}]
O también:
L=ConstantArray[x,25]/.x->RandomInteger[{1,50}]
{6,36,34,5,45,11,25,19,34,45,46,22,4,13,21,50,11,29,30,42,41,38,8,42,2}
First[L]
6
Last[L]
2
Take[L,3]
{6,36,34}
Take[L,-5]
{41,38,8,42,2}
Take[L,{10,20}]
{45,46,22,4,13,21,50,11,29,30,42}
```

```
Sort[L]
{2,4,5,6,8,11,11,13,19,21,22,25,29,30,34,34,36,38,41,42,42,45,45,46,50}
Reverse[Sort[L]]
{50,46,45,45,42,42,41,38,36,34,34,30,29,25,22,21,19,13,11,11,8,6,5,4,2}
```

4. Genere una lista con 15 números reales aleatorios comprendidos entre 20 y 25, luego deje la lista sin su último elemento, sin su primer elemento, sin los dos primeros elementos, sin los tres últimos elementos, sin los elementos que se encuentran entre el cuarto y octavo elemento, sin el quinto elemento y sin el antepenúltimo elemento.

```
L=RandomReal[{20,25},15]
O también:
L=Table[RandomReal[{20,25}],{15}]
O también:
L=ConstantArray[x,15]/.x->RandomReal[{20,25}]
{20.6362,20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
Most[L]
{20.6362,20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291}
Rest[L]
{20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
Drop[L,2]
{23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
Drop[L,-3]
{20.6362,20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844}
Drop[L,{4,8}]
{20.6362,20.3023,23.3966,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
Delete[L,5]
{20.6362,20.3023,23.3966,20.6696,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
O también:
Drop[L,{5,5}]
{20.6362,20.3023,23.3966,20.6696,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,24.1167,23.3291,20.0997}
Delete[L,{-3}]
{20.6362,20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,23.3291,20.0997}
O también:
Drop[L,{-3,-3}]
{20.6362,20.3023,23.3966,20.6696,23.1352,20.0805,20.8465,22.2488,20.1016,21.6347,23.2624,24.2844,23.3291,20.0997}
```

5. Genere una lista con las raíces cuadradas de 10 números reales aleatorios comprendidos entre 2 y 3, luego extraiga el menor elemento de la lista y ubique la posición en que se encuentra dicho elemento, extraiga el mayor elemento de la lista y ubique la posición en que se encuentra dicho elemento, determine el número de veces que el menor elemento se repite en la lista, determine el número de veces que el mayor elemento se repite en la lista, ordene ascendentemente los elementos de la lista eliminando los elementos repetidos, finalmente ordene descendentemente los elementos de la lista eliminando los elementos repetidos.

```
L=Table[Sqrt[RandomReal[{2,3}]],{10}]
O también:
```

```

L=Sqrt[RandomReal[{2,3},20]]
O también:
L=ConstantArray[x,20]/.x->Sqrt[RandomReal[{2,3}]]
{1.59239,1.63795,1.54223,1.48236,1.42748,1.51482,1.59226,1.52935,1.54327,1.6
8969}
Min[L]
1.42748
Position[L,Min[L]]
{{5}}
Max[L]
1.68969
Position[L,Max[L]]
{{10}}
Count[L,Min[L]]
1
Count[L,Max[L]]
1
Union[L]
{1.42748,1.48236,1.51482,1.52935,1.54223,1.54327,1.59226,1.59239,1.63795,1.6
8969}
Reverse[Union[L]]
{1.68969,1.63795,1.59239,1.59226,1.54327,1.54223,1.52935,1.51482,1.48236,1.4
2748}

```

6. Genere una lista con 20 números enteros aleatorios comprendidos entre 1 y 5, luego determine el número de veces que se repite el número 3, determine el menor elemento se repite en la lista, determine el número de veces el mayor elemento se repite en la lista, intercale entre los elementos de la lista el número 7, añada el número 0 al principio de la lista, añada el número 9 al final de la lista, añada el número 9 en la quinta posición de la lista, ordene la lista ascendentemente eliminando los números repetidos y ordene la lista descendientemente sin eliminar los elementos repetidos. Nota: En este ejemplo se utiliza "SeedRandom" para que en las tres formas en que se genera la lista, se obtengan siempre los mismos números.

```

SeedRandom[0];L=RandomInteger[{1,5},20]
{2,1,5,1,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5}
O también:
SeedRandom[0];L=Table[RandomInteger[{1,5}],{20}]
{2,1,5,1,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5}
O también:
SeedRandom[0];L=ConstantArray[x,20]/.x->RandomInteger[{1,5}]
{2,1,5,1,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5}
Count[L,3]
2
Count[L,Min[L]]
5
Count[L,Max[L]]
5
Riffle[L,7]
{2,7,1,7,5,7,1,7,5,7,2,7,3,7,2,7,1,7,4,7,4,7,5,7,3,7,1,7,5,7,2,7,4,7,1,7,4,7,
5}
Prepend[L,0]
{0,2,1,5,1,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5}
Append[L,9]
{2,1,5,1,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5,9}
Insert[L,9,5]
{2,1,5,1,9,5,2,3,2,1,4,4,5,3,1,5,2,4,1,4,5}

```

```

Union[L]
{1,2,3,4,5}
Reverse[Sort[L]]
{5,5,5,5,5,4,4,4,4,3,3,2,2,2,2,1,1,1,1,1}

```

7. Genere una lista con 20 números enteros aleatorios comprendidos entre 1 y 50, luego determine si el número 20 pertenece a la lista, si el número 30 no pertenece a la lista, rote la lista 6 posiciones a la izquierda, rote la lista 4 posiciones a la derecha, añada al principio de la lista el número -5, añada al final de la lista el número -7, rellene el principio de la lista con el número 1, de manera que la lista tenga un total de 30 elementos, rellene el final de la lista con el número 3, de manera que la lista tenga un total de 25 elementos, determine la posición del mayor elemento de la lista, determine el número de veces que se repite el menor elemento de la lista, finalmente ordene ascendente la lista.

```

SeedRandom[10];L=RandomInteger[{1,50},20]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
SeedRandom[10];L=Table[RandomInteger[{1,50}],{20}]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
SeedRandom[10];L=ConstantArray[y,20]/.y->RandomInteger[{1,50}]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
MemberQ[L,20]
True
FreeQ[L,30]
True
RotateLeft[L,6]
{33,44,22,31,12,41,49,8,25,25,20,5,27,9,43,48,24,36,39,23}
RotateRight[L,4]
{20,5,27,9,43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25}
Prepend[L,-5]
{-5,43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
O también:
Insert[L,-5,1]
{-5,43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
Append[L,-7]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9,-7}
O también:
Insert[L,-7,Length[L]+1]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9,-7}
PadLeft[L,30,1]
{1,1,1,1,1,1,1,1,1,1,43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9}
PadRight[L,25,3]
{43,48,24,36,39,23,33,44,22,31,12,41,49,8,25,25,20,5,27,9,3,3,3,3,3}
Position[L,Max[L]]
{{13}}
Count[L,Min[L]]
1
Sort[L]
{5,8,9,12,20,22,23,24,25,25,27,31,33,36,39,41,43,44,48,49}

```

8. Genere una lista con los números 1, 1.5, 2, 2.5, ...,10, calcule la suma de los elementos de la lista, calcule la productoria de los elementos de la lista, invierta los elementos de la lista, extraiga el décimo elemento de la lista, extraiga los elementos comprendidos entre el segundo y el décimo, extraiga los elementos 1, 4 y 12, reemplace el quinto elemento por 5.5, reemplace en la lista original el décimo elemento por 10.10 y reemplace en la lista original los elementos cuarto al sép-

timo por "1.1, 1.2, 1.3 y 1.4".

```
L=Range[1,10,0.5]
{1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
O también:
L=Table[i,{i,1,10,0.5}]
{1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
Total[L]
104.5
O también:
s=0;Table[s=s+L[[i]],{i,Length[L]};s
104.5
p=1;Table[p=p*L[[i]],{i,Length[L]};p
4.64039×1012
Reverse[L]
{10.,9.5,9.,8.5,8.,7.5,7.,6.5,6.,5.5,5.,4.5,4.,3.5,3.,2.5,2.,1.5,1.}
L[[10]]
5.5
O también:
Part[L,10]
5.5
L[[2;;10]]
{1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5}
O también:
Part[L,2;;10]
{1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5}
O también:
Take[L,{2,10}]
{1.5,2.,2.5,3.,3.5,4.,4.5,5.,5.5}
L[{{1,4,12}}]
{1.,2.5,6.5}
ReplacePart[L,5→5.5]
{1.,1.5,2.,2.5,5.5,3.5,4.,4.5,5.,5.5,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
L[[10]]=10.10;L
{1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,10.1,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
O también:
Part[L,10]=10.10;L
{1.,1.5,2.,2.5,3.,3.5,4.,4.5,5.,10.1,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
L[[4;;7]]={1.1,1.2,1.3,1.4};L
{1.,1.5,2.,1.1,1.2,1.3,1.4,4.5,5.,10.1,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
O también:
Part[L,4;;7]={1.1,1.2,1.3,1.4};L
{1.,1.5,2.,1.1,1.2,1.3,1.4,4.5,5.,10.1,6.,6.5,7.,7.5,8.,8.5,9.,9.5,10.}
```

9. Genere una lista con las letras "a" a "z", rote la lista 10 posiciones a la izquierda, rote la lista 15 posiciones a la derecha, invierta los elementos de la lista, extraiga el décimo elemento de la lista, extraiga los elementos que se encuentran en las posiciones 2, 4 y 9 de la lista, extraiga los elementos comprendidos entre el décimo y vigésimo elementos, reemplace la letra "c" por "c1", reemplace las letras "e, f, g" por "e1, f1, g1", reemplace en la lista original el elemento "x" por "x1" y reemplace en la lista original los elementos "p, q, r" por "p1, q1, r1".

```
L=CharacterRange["a","z"]
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
O también:
L=Table[FromCharacterCode[i],{i,ToCharacterCode["a"][[1]],ToCharacterCode["z"][[1]]}]
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
```

"FromCharCode" convierte el número de código en la letra respectiva, así por ejemplo convierte el número 65 en la letra "A". "ToCharCode" hace el proceso inverso, así convierte la letra "A" en el número 65.

O también:

```
Array[FromCharCode,26,ToCharCode["a"]]
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
RotateLeft[L,10]
{k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,a,b,c,d,e,f,g,h,i,j}
RotateRight[L,15]
{l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,a,b,c,d,e,f,g,h,i,j,k}
Reverse[L]
{z,y,x,w,v,u,t,s,r,q,p,o,n,m,l,k,j,i,h,g,f,e,d,c,b,a}
L[[10]]
```

j

O también:

```
Part[L,10]
```

j

```
L[{{2,4,9}}]
```

```
{b,d,i}
```

O también:

```
Part[L,{2,4,9}]
```

```
{b,d,i}
```

```
L[[10;;20]]
```

```
{j,k,l,m,n,o,p,q,r,s,t}
```

O también:

```
Part[L,10;;20]
```

```
{j,k,l,m,n,o,p,q,r,s,t}
```

O también:

```
Take[L,{10,20}]
```

```
{j,k,l,m,n,o,p,q,r,s,t}
```

```
ReplacePart[L,Position[L,"c"]->"c1"]
```

```
{a,b,c1,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
```

```
ReplacePart[L,{Position[L,"e"][[1]]->"e1",Position[L,"f"][[1]]->"f1",Position[L,"g"][[1]]->"g1"}]
```

```
{a,b,c,d,e1,f1,g1,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}
```

```
L[[Flatten[Position[L,"x"]]]]="x1";L
```

```
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x1,y,z}
```

Nota: "Flatten" elimina los subniveles de una lista (las llaves dentro de una lista), así si la lista es {1,2,{3,4},{5,6},{7,8},{9,10}}, "Flatten" devuelve: {1,2,3,4,5,6,7,8,9,10}

```
L[[Flatten[{Position[L,"p"],Position[L,"q"],Position[L,"r"]}]]]="p1","q1","r1";L
```

```
{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p1,q1,r1,s,t,u,v,w,x1,y,z}
```

10. Genere una lista con los senos de los números 1,2,3,...,15, determine la posición del mayor elemento, la posición del menor elemento, intercale entre los elementos de la lista el número 0.5, rellene la lista con ceros a la derecha para que tenga un total de 20 elementos, rellene la lista con unos a la izquierda para que tenga un total de 25 elementos, reemplace en la lista original el décimo elemento por 10.10, reemplace en la lista original el sexto elemento por 6.6, reemplace los elementos octavo a décimo segundo por "1, 2, 3, 4, 5", reemplace en la lista original el primer, quinto y décimo elemento por "1, 5 10", extraiga el sexto elemento y extraiga los elementos que se encuentran en las posiciones 3, 8, 10 y 12.

```
L=Array[Sin,15]/N
```

```

{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,-0.544021,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
O también:
L=Table[Sin[x],{x,15}]/N
{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,-0.544021,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
O también:
Sin[Range[15]]/N
{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,-0.544021,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
Position[L,Max[L]]
{{14}}
Position[L,Min[L]]
{{11}}
Riffle[L,0.5]
{0.841471,0.5,0.909297,0.5,0.14112,0.5,-0.756802,0.5,-0.958924,0.5,-
0.279415,0.5,0.656987,0.5,0.989358,0.5,0.412118,0.5,-0.544021,0.5,-
0.99999,0.5,-0.536573,0.5,0.420167,0.5,0.990607,0.5,0.650288}
PadRight[L,20,0]
{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,-0.544021,-0.99999,-
0.536573,0.420167,0.990607,0.650288,0,0,0,0,0}
PadLeft[L,25,1]
{1,1,1,1,1,1,1,1,1,1,0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,-0.544021,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
L[[10]]=10.10;L
{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,10.1,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
O también:
Part[L,10]=10.10;L
{0.841471,0.909297,0.14112,-0.756802,-0.958924,-
0.279415,0.656987,0.989358,0.412118,10.1,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
L[[6]]=6.6;L
{0.841471,0.909297,0.14112,-0.756802,-
0.958924,6.6,0.656987,0.989358,0.412118,10.1,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
O también:
Part[L,6]=6.6;L
{0.841471,0.909297,0.14112,-0.756802,-
0.958924,6.6,0.656987,0.989358,0.412118,10.1,-0.99999,-
0.536573,0.420167,0.990607,0.650288}
L[[8;;12]]={1,2,3,4,5};L
{0.841471,0.909297,0.14112,-0.756802,-
0.958924,6.6,0.656987,1,2,3,4,5,0.420167,0.990607,0.650288}
O también:
Part[L,8;;12]={1,2,3,4,5};L
{0.841471,0.909297,0.14112,-0.756802,-
0.958924,6.6,0.656987,1,2,3,4,5,0.420167,0.990607,0.650288}
L[{{1,5,10}}]={1,5,10};L
{1,0.909297,0.14112,-
0.756802,5,6.6,0.656987,1,2,10,4,5,0.420167,0.990607,0.650288}
O también:

```

```

Part[L,{1,5,10}]= {1,5,10};L
{1,0.909297,0.14112,-
0.756802,5,6.6,0.656987,1,2,10,4,5,0.420167,0.990607,0.650288}
L[[6]]
6.6
O también:
Part[L,6]
6.6
L[{{3,8,10,12}}]
{0.14112,1,10,5}
O también:
Part[L,{3,8,10,12}]
{0.14112,1,10,5}

```

3.1.2. Ejercicios

1. Calcule el coseno de los siguientes ángulos: 5.°, 20.°, 65.°, 70.°, 90.°, 110.° y 180.°, empleando una lista como: parámetro directo, una regla y parámetro posfijo.
2. Empleando RandomInteger con una tabla, una regla y por si solo, genere una lista con 23 elementos aleatorios comprendidos entre 10 y 40.
3. Genere una lista con 25 números enteros aleatorios comprendidos entre 10 y 30, luego extraiga el primer elemento de la lista, el último elemento de la lista, los tres primeros elementos de la lista, los cinco últimos elementos de la lista, los elementos que se encuentran entre las posiciones 10 a la 20 y ordene la lista en orden ascendente y descendente.
4. Genere una lista con 15 números reales aleatorios comprendidos entre 5 y 10, luego deje la lista sin su último elemento, sin su primer elemento, sin los dos primeros elementos, sin los tres últimos elementos, sin los elementos que se encuentran entre el cuarto y octavo elemento, sin el quinto elemento y sin el antepenúltimo elemento.
5. Genere una lista con las raíces cúbicas de 10 números reales aleatorios comprendidos entre 4 y 6, luego extraiga el menor elemento de la lista y ubique la posición en que se encuentra dicho elemento, extraiga el mayor elemento de la lista y ubique la posición en que se encuentra dicho elemento, determine el número de veces que el menor elemento se repite en la lista, determine el número de veces que el mayor elemento se repite en la lista, ordene ascendentemente los elementos de la lista eliminando los elementos repetidos, finalmente ordene descendentemente los elementos de la lista eliminando los elementos repetidos.
6. Genere una lista con 25 números enteros aleatorios comprendidos entre 10 y 15, luego determine el número de veces que se repite el número 13, determine el menor elemento se repite en la lista, determine el número de veces el mayor elemento se repite en la lista, intercale entre los elementos de la lista el número 7, añada el número 0 al principio de la lista, añada el número 9 al final de la lista, añada el número 9 en la quinta posición de la lista, ordene la lista ascendentemente eliminando los números repetidos y ordene la lista descendentemente sin eliminar los elementos repetidos.
7. Genere una lista con 25 números enteros aleatorios comprendidos entre 1 y 40, luego determine si el número 25 pertenece a la lista, si el número 33 no pertenece a la lista, rote la lista 6 posiciones a la izquierda, rote la lista 4 posiciones a la derecha, añada al principio de la lista el número -5, añada al final de la lista el número -7, rellene el

principio de la lista con el número 1, de manera que la lista tenga un total de 32 elementos, rellene el final de la lista con el número 3, de manera que la lista tenga un total de 30 elementos, determine la posición del mayor elemento de la lista, determine el número de veces que se repite el menor elemento de la lista, finalmente ordene ascendente-mente la lista.

8. Genere una lista con los números 2, 4, 6, 8, ...,30, calcule la suma de los elementos de la lista, calcule la productoria de los elementos de la lista, invierta los elementos de la lista, extraiga el décimo elemento de la lista, extraiga los elementos comprendidos entre el segundo y el décimo, extraiga los elementos que se encuentran en las posiciones 1, 4 y 12, reemplace el quinto elemento por 5.5, reemplace en la lista original el décimo elemento por 10 y reemplace en la lista original los elementos cuarto al séptimo por "2.1, 2.2, 2.3 y 2.4".
9. Genere una lista con las letras "A" a "Z", rote la lista 10 posiciones a la izquierda, rote la lista 15 posiciones a la derecha, invierta los elementos de la lista, extraiga el décimo elemento de la lista, extraiga los elementos que se encuentran en las posiciones 2, 4 y 9 de la lista, extraiga los elementos comprendidos entre el décimo y vigésimo elementos, reemplace la letra "C" por "C2", reemplace las letras "E, F, G" por "E2, F2, G2", reemplace en la lista original el elemento "X" por "X2" y reemplace en la lista original los elementos "P, R, T" por "P1, R1, T1".
10. Genere una lista con los cosenos de los números 1,2,3,...,15, determine la posición del mayor elemento, la posición del menor elemento, intercale entre los elementos de la lista el número 0.5, rellene la lista con ceros a la derecha para que tenga un total de 20 elementos, rellene la lista con unos a la izquierda para que tenga un total de 25 elementos, reemplace en la lista original el décimo elemento por 10, reemplaze en la lista original el sexto elemento por 6, reemplace los elementos octavo a décimo segundo por "11, 12, 13, 14, 15", reemplace en la lista original el primer, quinto y décimo elemento por "10, 55, 100", extraiga el sexto elemento y extraiga los elementos que se encuentran en las posiciones 2, 6, 10 y 14.

3.2. Patrones

Los patrones (patterns), se utilizan en Mathematica para representan clases de expresiones, es decir grupos de expresiones que se ajustan a alguna forma en particular. Al permitirnos trabajar con grupos de expresiones en lugar de expresiones específicas, los patrones viabilizan la realización de operaciones más generales.

Los patrones suelen emplearse en la generalización de asignaciones (=
:=), reglas (->, :>), así como en funciones tales como Position, MemberQ, FreeQ, Count, MatchQ, Cases y DeleteCases (las cuatro primeras estudiadas ya anteriormente).

El patrón más frecuentemente utilizado en Mathematica es "Blank", el cual se identifica con el símbolo de subrayado "_", y que representa a cualquier expresión. Así "_^_", es un patrón que representa a cualquier expresión elevada a cualquier otra expresión, empleando este patrón podemos reemplazar, por ejemplo, todas las potencias de la expresión " $x^a + y + y^b + z + w^c$ ", por " z^3 ":

```
x^a+y+y^b+z+w^c/._^_>z^3
y+z+3z^3
```

A los patrones se les puede asignar un nombre, en el caso del patrón "Blank" es suficiente que se escriba el nombre delante del símbolo de subrayado, por ejemplo, "w_" y "pat_", son dos patrones que siguen representando a cualquier expresión, pero que ahora se llaman "w" y "pat" respectivamente. El nombrar los patrones permite emplear dichos nombres al lado derecho de una regla o una asignación, lo que es muy útil cuando se crean funciones (como veremos más adelante). Así por ejemplo, nombrando los patrones, podemos reemplazar todas las potencias de la expresión " $x^a + y^b + z + w^c$ ", por la base multiplicado por el logaritmo natural de la potencia:

```
x^a+y+y^b+z+w^c/.r_s_>r*Log[s]
y+z+x Log[a]+y Log[b]+w Log[c]
```

En general, sin embargo, se puede asignar un nombre a cualquier patrón, sea simple o compuesto, empleando los dos puntos ":" (Pattern), de acuerdo al siguiente formato: "*nombre:patrón*". Por ejemplo, asignando un nombre al patrón compuesto " $x^a + y^b + z + w^c$ ", podemos reemplazar todas las potencias de la expresión " $x^a + y^b + z + w^c$ ", por el coseno de las mismas:

```
x^a+y+y^b+z+w^c/.r:_->Cos[r]
y+z+Cos[w^c]+Cos[x^a]+Cos[y^b]
```

Un patrón como "Blank" puede referirse a un tipo de dato en particular, para ello se añade el nombre del tipo (denominado en Mathematica "head" (cabeza) al final del patrón, por ejemplo "_Integer", es un patrón que representa a cualquier número entero, "_Real", es un patrón que representa a cualquier número real, "_Complex" un número complejo, "_List" una lista, "_Symbol" un símbolo, etc. Por supuesto a estos patrones también se les puede asignar un nombre, así "w_Cos" es un patrón que representa a cualquier coseno y cuyo nombre es "w". Por ejemplo podemos reemplazar la suma de la lista "{x+y+z, x*y*z, x/y/z}" por "w":

```
{x+y+z, x*y*z, x/y/z}/._Plus->w
{ w, x y z,  $\frac{x}{y z}$  }
```

Para representar uno, dos o más expresiones el patrón debe estar conformado por dos "Blank" consecutivos "__" (BlankSequence), empleando este patrón podemos reemplazar por ejemplo todas las funciones "f" con más de un argumento, de la siguiente expresión, por funciones "g"

```
f[x,y,z]+f[3x,2]+f[]+f[y]/.f[x__]->g[x]
f[]+g[y]+g[3 x,2]+g[x,y,z]
```

Como se puede ver en este ejemplo, la función sin argumentos "f[]" no ha sido reemplazada por "g[]", porque el patrón "__" representa a una o más expresiones, pero no a cero. Para representar a cero o más expresiones el patrón debe estar conformado por tres "Blanks" consecutivos "___" (BlankNullSequence). Así empleando este patrón en el anterior ejemplo obtenemos:

```
f[x,y,z]+f[3x,2]+f[]+f[y]/.f[x___]->g[x]
g[]+g[y]+g[3 x,2]+g[x,y,z]
```

Donde como vemos la función sin argumentos "f[]" ha sido reemplazada por "g[]". No obstante, los tres "Blanks" deben ser empleados sólo en aquellos casos donde se está seguro de manejar correctamente los casos con cero argumentos, pues de lo contrario pueden producirse múltiples errores, incluyendo ciclos infinitos.

Para representar a una expresión que se repite una o más veces se emplea al patrón "Repeated", que se representa por dos puntos seguidos: "..". Em-

pleando este patrón podemos reemplazar, en la siguiente lista, todas las funciones "f" que tienen una o más letras "a" por x^3 :

```
{f[a],f[a,a],f[a,b,c],f[],f[b,c,a,a],f[a,a,a,a]}/.f[a..]->x^3
{x^3,x^3,f[a,b,c],f[],f[b,c,a,a],x^3}
```

Si se quiere aplicar un patrón a las expresiones que se repiten 0 o más veces, entonces se debe emplear "RepeatedNull" que se representa por tres puntos seguidos: "...". Aplicando este patrón al anterior ejemplo resulta:

```
{f[a],f[a,a],f[a,b,c],f[],f[b,c,a,a],f[a,a,a,a]}/.f[a...]->x^3
{x^3,x^3,f[a,b,c],x^3,f[b,c,a,a],x^3}
```

Ahora "f[]" también ha sido reemplazada por x^3 . En el patrón "Repeated" (y RepeatedNull) se puede especificar el número de veces que debe repetirse la expresión, así para especificar que la letra "a" debe repetirse un máximo de 3 veces escribimos:

```
{f[a],f[a,a],f[a,b,c],f[],f[a,a,a],f[b,c,a,a],f[a,a,a,a]}/.f[Repeated[a,3]]->x^3
{x^3,x^3,f[a,b,c],f[],x^3,f[b,c,a,a],f[a,a,a,a]}
```

Como se puede observar, en este caso la función "f[a,a,a,a]" no ha sido reemplazada por x^3 , porque en la misma "a" se repite más de 3 veces. Se puede especificar también el intervalo de repetición de la expresión, por ejemplo para reemplazar por x^3 todas las funciones donde "a" se repite entre 2 y 3 veces escribimos:

```
{f[a],f[a,a],f[a,b,c],f[],f[a,a,a],f[b,c,a,a],f[a,a,a,a]}/.f[Repeated[a,{2,3}]]->x^3
{f[a],x^3,f[a,b,c],f[],x^3,f[b,c,a,a],f[a,a,a,a]}
```

Ahora no se ha reemplazado "f[a]" y "f[a,a,a,a]" porque están fuera de intervalo. Podemos especificar también el número exacto de veces que se debe repetir la expresión, así para reemplazar por x^3 todas las funciones donde "a" se repite 3 veces escribimos:

```
{f[a],f[a,a],f[a,b,c],f[],f[a,a,a],f[b,c,a,a],f[a,a,a,a]}/.f[Repeated[a,{3}]]->x^3
{f[a],f[a,a],f[a,b,c],f[],x^3,f[b,c,a,a],f[a,a,a,a]}
```

Y como se ve, la única función que ha sido reemplazada es "f[a,a,a]".

Para representar a dos o más expresiones alternativas se emplea el patrón "Alternatives" que se representa por una barra vertical: "|". Por ejemplo para reemplazar por "z" a las letras "a", "b" o "c" en la siguiente lista escribimos:

```
{a,c,d,b,a,e,b,c,a,d,e,f,g,c}/.a|b|c->z
{z,z,d,z,z,e,z,z,z,d,e,f,g,z}
```

En la siguiente lista se reemplazan las funciones que se llamen "f", "g" o "h" y tengan cualquier número de argumentos, por la función "p", siendo sus argumentos el nombre de la función y sus argumentos actuales:

```
{f[a,b],k[c,d],g[a],h[c,d],l[e],m[a,b,c],g[d,e],f[a,b,c]}/.(f:(f|g|h))[x__]->p[f,x]
{p[f,a,b],k[c,d],p[g,a],p[h,c,d],l[e],m[a,b,c],p[g,d,e],p[f,a,b,c]}
```

Se pueden crear patrones que representen a una secuencia específica de expresiones con "PatternSequence[p₁,p₂,...]", donde "p₁", "p₂", etc., son los patrones a los que debe ajustarse la secuencia de expresiones. Por ejemplo en la siguiente lista se reemplazan las listas que tienen una secuencia formada por una letra y el cuadrado de una letra por la letra "z":

```
{a,b^2},{c,d^2},{c,d},{a,c^2},{c^2,d}}/.PatternSequence[{x_,y_^2}]->z
{z,z,{c,d},z,{c^2,d}}
```

En ocasiones se quiere aplicar un patrón sólo si se cumple una determinada condición. En esos casos se emplea "Condition", que se representa por un quebrado y un punto y coma: "/" y "PatternTest", que se representa por un signo de interrogación "?".

"Condition" se emplean en la forma "patrón/prueba", donde "patrón" se aplica sólo si la "prueba" es verdadera, al construir la prueba se pueden emplear los nombres de los patrones. Por ejemplo en la siguiente lista se reemplazan los números negativos por el cuadrado de los mismos:

```
{1,3,-4,6,-12,25,-9}/.x_/;x<0->x^2
{1,3,16,6,144,25,81}
```

"PatternTest" se emplea en la forma "patrón?prueba", pero en este caso la "prueba" es una función que se aplica a las expresiones a las que debe ajustarse el patrón, así el anterior ejemplo empleando "PatternTest" es:

```
{1,3,-4,6,-12,25,-9}/.x_?Negative->x^2
{1,3,16,6,144,25,81}
```

Donde la función "Negative" se ha aplicado a cada uno de los elementos de la lista y el "patrón" (y su regla), se han aplicado sólo en aquellos casos donde la función es negativa (es decir Negative es True).

Para obtener una lista a partir de otra, con los elementos que se ajustan a un determinado patrón, se emplea "Cases[lista,patrón]". Así en el siguiente ejemplo se obtiene una lista con los números enteros:

```
Cases[{1,a,b,3,5,d,8,10,f,g},_Integer]
{1,3,5,8,10}
```

"Cases" se puede emplear también en conjunción con una regla, así en el siguiente ejemplo se obtiene una lista de los cuadrados de los números enteros:

```
Cases[{1,a,b,3,5,d,8,10,f,g},x_Integer->x^2]
{1,9,25,64,100}
```

Para invertir el alcance del patrón, es decir para que represente a todas las expresiones excepto a las que se ajustan al mismo, se emplea "Except[patrón]", así para obtener los cuadrados de todos los elementos de la anterior lista, con excepción de los números enteros escribimos:

```
Cases[{1,a,b,3,5,d,8,10,f,g},x:Except[_Integer]->x^2]
{a^2,b^2,d^2,f^2,g^2}
```

Alternativamente, para obtener una lista con todos los elementos que no concuerdan con un determinado patrón se puede emplear "DeleteCases[lista,patrón]", así en el siguiente ejemplo obtenemos una lista con los elementos que no son números:

```
DeleteCases[{1,a,b,3,5,d,8,10,f,g},_Integer]
{a,b,d,f,g}
```

Como ya se dijo, algunas de las funciones que se estudiaron en el anterior subtítulo pueden ser empleadas también con patrones, por ejemplo podemos emplear la función "Count" para contar todos los números enteros de la siguiente lista:

```
Count[{3,4,a,b,c,4,5,2,3,d,e,w,f,9,2,3},_Integer]
9
```


3.2.1. Ejemplos

11. Reemplace en la siguiente expresión todas las raíces cuadradas por las raíces cúbicas respectivas.

$$3x + \sqrt{2+3x} + \sqrt{\frac{2+3x}{y}} \cdot y$$

```
3*x-y+Sqrt[3*x+2]+Sqrt[(3*x+2)/y]/.x_^(1/2)->x^(1/3)//TraditionalForm
```

$$3x - y + \sqrt[3]{3x+2} + \sqrt[3]{\frac{3x+2}{y}}$$

12. Reemplace en la siguiente expresión todos los logaritmos naturales por logaritmos de base 10 y luego obtenga el valor numérico de la expresión resultante para "x=3.1", "y=2.2" y "z=1.2".

$$\frac{\ln(x) + x^2 - \ln(x - y + x^2)}{\ln(x + 2y - z) - z^3}$$

```
(Log[x]+x^2-Log[x-y+x^2])/(Log[x+2*y-z]-z^3)/.Log[x_]->Log[10,x]
```

$$\frac{x^2 \cdot \frac{\log x}{\log 10} - \frac{\log(x - y + x^2)}{\log 10}}{z^3 - \frac{\log(x + 2y - z)}{\log 10}}$$

```
%/.{x->3.1,y->2.2,z->1.2}
-9.77728
```

13. Reemplace en la siguiente expresión todas las funciones seno y coseno por tangente, luego evalúe el valor de la expresión resultante para "x=0.5", "y=0.3".

$$\sqrt[3]{\frac{\sin(x+y) - \cos(x-y)}{2\cos(x) + 3\sin(y) - \tan(x+y)}}$$

```
((Sin[x+y]-Cos[x-y])/(2*Cos[x]+3*Sin[y]-Tan[x+y]))^(1/3)/.Sin[r_]|Cos[r_]->Tan[r] //TraditionalForm
```

$$\sqrt[3]{\frac{\tan x + y \cdot \tan x \cdot y}{2 \tan x + 3 \tan y \cdot \tan x \cdot y}}$$

```
%/.{x->0.5,y->0.3}.
0.94146
```

14. Reemplace en la siguiente expresión todas las funciones seno, coseno o tangente por las raíces cuadradas de las mismas. Luego evalúe la expresión resultante para los siguientes valores: "x=2.1", "y=2.2", "z=2.3".

$$\frac{\cos y + \sin x \cdot \tan z}{\cos^2 x \cdot \tan^2 y + \sin^2 x + y^2 + z^2}$$

```
(Sin[x]+Cos[y]-Tan[z])/(Sin[x^2+y^2+z^2]-Cos[x]^2-Tan[y]^2)/.Sin[x_]|Cos[x_]|Tan[x_]->Sqrt[x]
```

$$\frac{\sqrt{x} + \sqrt{y} \cdot \sqrt{z}}{x \cdot y + \sqrt{x^2 + y^2 + z^2}}$$

```
%/.{x->2.1,y->2.2,z->2.3}
-2.908
```

15. Reemplace en las expresiones de la lista: {3x+y+z, x-2z+3, 2x+3}, todos los números enteros por los cuadrados de los mismos, luego evalúe las

expresiones para "x=1.1", "y=1.2", "z=1.3".

```
{3*x+y+z,x-2*z+3,2*x+3}/.x_Integer->x^2
{9 x+y+z,9+x+4 z,9+4 x}
%/.{x->1.1,y->1.2,z->1.3}
{12.4,15.3,13.4}
```

16. Reemplace en las expresiones de la lista: $\{2x+y^2+3z, 2x-z-3y, 2x^2-5y-z^{0.5}\}$, todas las expresiones que tienen sumas o restas por la raíz cuadrada de las mismas, luego reemplace todas las operaciones de suma o resta por multiplicaciones, finalmente evalúe la expresión resultante para "x=0.5", "y=0.6", "z=0.7".

```
{2*x+y^2+3*z,2*x-z-3*y,2*x^2-5 y-z^0.5}/.x_Plus->Sqrt[x]
|√ 2 x· y2+ 3 z , √ 2 x· 3 y· z , √ 2 x2· 5 y· z0.5 |
%/.Plus->Times
|√ 6 √ x y2 z , √ 6 √ x y z , √ 10 √ x2 y z0.5 |
%/.{x->0.5,y->0.6,z->0.7}
{0.869483,1.1225,1.12026}
```

17. Reemplace en la siguiente expresión: "f[x,x,x]+f[x,y,z,w]+f[y,z]+f[x,x,x,x]", todas las funciones "f" que tienen una o más veces la letra "x" por la función "g". En la expresión resultante, multiplique todos los argumentos de las funciones "g". En la expresión resultante sume todos los argumentos de las funciones "f", finalmente calcule el valor de la expresión resultante para "x=1", "y=2", "z=3" y "w=4".

```
e1=f[x,x,x]+f[x,y,z,w]+f[y,z]+f[x,x,x,x]/.f[r:x..]>g[r]
f[y,z]+f[x,y,z,w]+g[x,x,x]+g[x,x,x,x]
e1=e1/.g->Times
x3+x4+f[y,z]+f[x,y,z,w]
e1=e1/.f->Plus
w+x+x3+x4+2 y+2 z
e1/.{x->1,y->2,z->3,w->4}
17
```

18. Reemplace en la siguiente expresión: $f[x,x]+f[x]+f[y,z]+f[x,y,z]+f[x,2y,3z]+f[x,x,x,x]$, todas las funciones "f" donde "x" se repite una o más veces por la multiplicación de dichas "x". En la expresión resultante, reemplace todas las funciones "f" por la suma de sus argumentos. Finalmente calcule el valor de la expresión resultante para "x=1", "y=2" y "z=3".

```
e1=f[x,x]+f[x]+f[y,z]+f[x,y,z]+f[x,2*y,3*z]+f[x,x,x,x]/.f[r:x..]>g[r]/.g->Times
x+x2+x4+f[y,z]+f[x,y,z]+f[x,2 y,3 z]
e1=e1/.f->Plus
3 x+x2+x4+4 y+5 z
e1/.{x->1,y->1,z->3}
24
```

19. Reemplace en la siguiente expresión: $(f[x,y,z]+f[x,x,x]+f[x,x]+f[y,z])/(f[x,x,x,x]-f[y,z]-f[x,2 y,z])$, las funciones "f" que tienen dos o más "x" por la multiplicación de las mismas. En la expresión resultante reemplace los argumentos de las funciones "f" por la suma de los mismos. En la expresión resultante reemplace las funciones "f" por el logaritmo natural, finalmente evalúe la función para "x=1", "y=1", "z=1".

```
e1=(f[x,y,z]+f[x,x,x]+f[x,x]+f[y,z])/(f [x,x,x,x]-f[y,z]-f[x,2 y,z])/.f[r:
x..]>g[r]/.g->Times
(x2+x3+f[y,z]+f[x,y,z])/(x4-f[y,z]-f[x,2 y,z])
```

```

e1=e1/.f[x__]→f[g[x]]/.g→Plus
(x2+x3+f[y+z]+f[x+y+z])/(x4-f[y+z]-f[x+2 y+z])
e1=e1/.f→Log
(x2+x3+Log[y+z]+Log[x+y+z])/(x4-Log[y+z]-Log[x+2 y+z])
e1/.{x→1,y→2,z→3}/N
-2.00872

```

20. En la siguiente expresión: $(f[x+y+z]*f[x-2 y]+f[3 z+2])/(f[]+f[x+y]+f[2 x])$, reemplace todas las funciones "f" que tienen uno o más argumentos por la función "Sin". En la expresión resultante reemplace las funciones "f" con cero argumentos por "0". Finalmente calcule el valor de la expresión resultante para "x=3.1", "y=2.2", "z=1.3".

```

e1=(f[x+y+z]*f[x-2 y]+f[3 z+2]-2*f[])/(f[]+f[x+y]+f[2 x])/.f[x__]→g[x]/.g
→Sin
(-2 f[]+Sin[x-2 y] Sin[x+y+z]+Sin[2+3 z])/(f[]+Sin[2 x]+Sin[x+y])
e1=e1/.f[]→0
(Sin[x-2 y] Sin[x+y+z]+Sin[2+3 z])/(Sin[2 x]+Sin[x+y])
e1/.{x→3.1,y→2.2,z→1.3}
0.736396

```

21. En la siguiente expresión: $f[x,y,z]+f[b,b,c]-g[x,x]+f[a,a,b,c]-g[x,x,b]+f[a,b,c]$, reemplace las funciones cuyos dos primeros argumentos son iguales por el cuadrado de los mismos. En la expresión resultante reemplace los argumentos de las funciones "f" y "g" por la suma de los mismos. En la expresión resultante reemplace las funciones "f" por "Sin" y las funciones "g" por "Cos". Finalmente evalúe la función para "a=1", "b=2", "c=3", "x=1.5", "y=2.5", "z=3.5".

```

e1=f[x,y,z]+f[b,b,c]-g[x,x]+f[a,a,b,c]-
g[x,x,b]+f[a,b,c]/.h[x_,x_,z_]→h[x^2,z]
f[b2,c]+f[a,b,c]+f[a2,b,c]+f[x,y,z]-g[x2]-g[x2,b]
e1=e1/.{f[x__]→f[s[x]],g[x__]→g[s[x]]}/.s→Plus
f[a+b+c]+f[a2+b+c]+f[b2+c]+f[x+y+z]-g[x2]-g[b+x2]
e1=e1/.{f→Sin,g→Cos}
-Cos[x2]-Cos[b+x2]+Sin[a+b+c]+Sin[a2+b+c]+Sin[b2+c]+Sin[x+y+z]
e1/.{a→1,b→2,c→3,x→1.5,y→2.5,z→3.5}
2.11042

```

22. Para la siguiente lista: {1,4,2,-5,6,8,-3,9,4,10,-6,7,3}, calcule la suma de los números negativos, la suma de los números positivos, la suma de los números mayores a 5 y la suma de los números menores a 5.

```

L={1,4,2,-5,6,8,-3,9,4,10,-6,7,3};
Cases[L,x_/;x<0]
{-5,-3,-6}
%/.List→Plus
-14
O también:
Cases[L,_?Negative]
{-5,-3,-6}
%/.List→Plus
-14
Cases[L,x_/;x>0]
{1,4,2,6,8,9,4,10,7,3}
%/.List→Plus
54
O también:
Cases[L,_?Positive]
{1,4,2,6,8,9,4,10,7,3}
%/.List→Plus
54

```

54

Cases[L,x_/;x>5]

{6,8,9,10,7}

%/.List→Plus

40

Cases[L,x_/;x<5]

{1,4,2,-5,-3,4,-6,3}

%/.List→Plus

0

23. Para la siguiente lista: {1,3,2,1.3,2.5,6,8,10,12.1,4.6,7.2,6,9,10}, calcule la suma de los cuadrados de los números enteros, calcule la suma de las raíces cuadradas de los números reales, calcule la suma de los números mayores a 7, calcule la multiplicación de los números menores a 7 y calcule la suma de los números impares.

L={1,3,2,1.3,2.5,6,8,10,12.1,4.6,7.2,6,9,10};

Cases[L,_Integer]^2/.List→Plus

431

Sqrt[Cases[L,_Real]]/.List→Plus

11.0279

Cases[L,x_/;x>7]/.List→Plus

56.3

Cases[L,x_/;x<7]/.List→Times

3229.2

Cases[L,x_/;OddQ[x]]/.List→Plus

13

O también:

Cases[L,_?OddQ]/.List→Plus

13

24. Para la siguiente lista: {3,3.5,8,2.3,2,5,5.1,6,6.6,7,4,4.8,3,9,9.7,10,12,14}, calcule la suma de las raíces cúbicas de los números diferentes de 5, calcule la productoria de los números diferentes a 3 o 4, cuente el número de elementos diferentes de 4, cuente el número de elementos no reales y obtenga el número de elementos de la lista.

L={3,5,3.5,8,4,2.3,2,5,3,5.1,6,6.6,7,4,4.8,3,4,9,9.7,10,5,12,3,14};

DeleteCases[L,x_/;x==5]/.List→Plus

124.

O también:

Cases[L,x_/;x≠5]/.List→Plus

124.

DeleteCases[L,x_/;x==5]^(1/3)/.List→Plus

36.7102

O también:

Cases[L,x_/;x≠5→x^(1/3)]/.List→Plus

36.7102

DeleteCases[L,3|4]/.List→Times

 1.60234×10^{13}

O también:

Cases[L,Except[3|4]]/.List→Times

 1.60234×10^{13}

Count[L,Except[4]]

21

Count[L,Except[_Real]]

18

Count[L,_Real]

6

Length[L]

24
 O También:
 Count[L,_]
 24

3.2.2. Ejercicios

11. Reemplace en la siguiente expresión todas las raíces cúbicas por las raíces cuadradas respectivas. Luego evalúe la expresión resultante para valores de "x" iguales a: 3.2, 5.6, 7.8, 9.10 y 12.2

$$\frac{\sqrt[3]{x+2} - \sqrt[3]{3x^2+5x-2} + \ln(2x)}{e^{x+3} + \sqrt[3]{\frac{3x^3+2x-3}{x^2+5}} - x^{3/2}}$$

12. Reemplace en la siguiente expresión todos los logaritmos de base 10 por logaritmos naturales y luego obtenga el valor numérico de la expresión resultante para "x=3.1", "y=2.2" y "z=1.2".

$$\frac{\log(x) + 2x^2 - \log(x - y + x^2)}{\log(x + 2y - z) - 3z^3}$$

13. Reemplace en la siguiente expresión todas las funciones seno y coseno por seno hiperbólico y coseno hiperbólico, luego evalúe el valor de la expresión resultante para "x=0.5", "y=0.3".

$$\sqrt[3]{\frac{\text{sen}(2x+3y) - \text{cos}(3x-y)}{2\text{cos}(x^2) + 3\text{sen}(y^3) - \tan(5x+2y)}}$$

14. Reemplace en la siguiente expresión todas las funciones seno, coseno o tangente por las raíces cúbicas de las mismas. Luego evalúe la expresión resultante para los siguientes valores: "x=1.2", "y=1.3", "z=1.4".

$$\sqrt[5]{\frac{\text{sen}(2x) + \text{cos}(3y) - \tan(4z)}{2\text{cos}^2(3x^2) + 3\text{sen}^2(2y^3) - \tan^3(2x+3y)}}$$

15. Reemplace en las expresiones de la lista: {x+2y+z, x-2z+4, 3x+2}, todos los números enteros por los cuadrados de los mismos, luego evalúe las expresiones para "x=1.1", "y=1.2", "z=1.3".

16. Reemplace en las expresiones de la lista: {x+y²+3z³, 2x-z²-3y, 2x²-5y^{1.2}-z^{0.5}}, todas las expresiones que tienen sumas o restas por la raíz cuadrada de las mismas, luego reemplace todas las operaciones de suma o resta por multiplicaciones, finalmente evalúe la expresión resultante para "x= 0.5", "y=0.6", "z=0.7".

17. Reemplace en la siguiente expresión: "f[x,x,x,x]+f[x,y,z,x,w]+f[y,z]+f[x,x,x]", todas las funciones "f" que tienen una o más veces la letra "x" por la función "g". En la expresión resultante, multiplique todos los argumentos de las funciones "g". En la expresión resultante sume todos los argumentos de las funciones "f", finalmente calcule el valor de la expresión resultante para "x=1", "y=2", "z=3" y "w=4".

18. Reemplace en la siguiente expresión: f[x]+f[x,x,x]+f[x,y,z]+f[y,z]+f[x,2y,3z]+f[x,x,x,x], todas las funciones "f" donde "x" se repite una o más veces por la multiplicación de dichas "x". En la expresión resultante, reemplace todas las funciones "f" por la suma de sus argumentos. Finalmente calcule el valor de la expresión resultante para "x=1", "y=2" y "z=3".

19. Reemplace en la siguiente expresión: $(f[x,y,z]+f[x,x]+f[y,z]+f[x,x,x])/ (f[x,x,x]-f[y,z]-f[x,2y,z]+f[x])$, las funciones "f" que tienen dos o más "x" por la multiplicación de las mismas. En la expresión resultante reemplace los argumentos de las funciones "f" por la suma de los mismos. En la expresión resultante reemplace las funciones "f" por el logaritmo natural, finalmente evalúe la función para "x=1", "y=1", "z=1".
20. En la siguiente expresión: $(f[x+y+3z]*f[x-2y-z]+f[z+2x])/(f[]+f[3x+y]+f[2x])$, reemplace todas las funciones "f" que tienen uno o más argumentos por la función "Sin". En la expresión resultante reemplace las funciones "f" con cero argumentos por "0". Finalmente calcule el valor de la expresión resultante para "x=3.1", "y=2.2", "z=1.3".
21. En la siguiente expresión: $f[x,z]+f[b,b,a]-g[x,x]+f[a,a,c,b]-g[x,x,a]+f[a,b,c]$, reemplace las funciones cuyos dos primeros argumentos son iguales por el cuadrado de los mismos. En la expresión resultante reemplace los argumentos de las funciones "f" y "g" por la suma de los mismos. En la expresión resultante reemplace las funciones "f" por "Sin" y las funciones "g" por "Cos". Finalmente evalúe la función para "a=1", "b=2", "c=3", "x=1.5", "y=2.5", "z=3.5".
22. Para la siguiente lista: {1,3,2,-4,5,8,-2,9,10,4,-5,7,3,8}, calcule la suma de los números negativos, la suma de los números positivos, la suma de los números mayores a 5 y la suma de los números menores a 5.
23. Para la siguiente lista: {1,3,5,2.3,1.5,7,8,9,11.1,5.6,7.2,6,9,10,3}, calcule la suma de los cuadrados de los números enteros, calcule la suma de las raíces cuadradas de los números reales, calcule la suma de los números mayores a 6, calcule la multiplicación de los números menores a 6 y calcule la suma de los números impares.
24. Para la siguiente lista: {2,2.5,8,2.1,9,3,4,5.1,5,9,6.1,2,8,4.8,3,9,9.7, 10,4,12,14}, calcule la suma de las raíces cúbicas de los números diferentes de 4, calcule la productoria de los números diferentes a 2 o 9, cuente el número de elementos diferentes de 2, cuente el número de elementos no reales y obtenga el número de elementos de la lista.

4. OPERACIONES SIMBÓLICAS, FUNCIONES Y SECUENCIAS

4.1. Operaciones simbólicas

Aun cuando el objetivo de esta materia es la de encontrar soluciones numéricas a problemas matemáticos, estudiaremos brevemente algunas de las funciones para transformar expresiones simbólicas.

Ya vimos anteriormente que al escribir una expresión simbólica, Mathematica lleva a cabo automáticamente todas las operaciones algebraicas directas. Por ejemplo al escribir la siguiente expresión obtenemos:

```
27-10x+16*x*x-7x+10x*x-9*x*x*x+8*x*x*x*x-10*x*x*x+4*x*x*x*x-
x*x*x*x*x+x*x*x*x*x*x-2*x*x*x*x*x
27- 27 x+ 36 x2- 19 x3+ 12 x4- 3 x5+ x6
```

Donde como se puede observar Mathematica ha efectuado todas las multiplicaciones y sumas algebraicas obteniendo una expresión simplificada.

A más de las operaciones directas, Mathematica puede llevar a cabo muchas otras operaciones con expresiones simbólicas (incluyendo, como veremos posteriormente, la integración y diferenciación). Así por ejemplo podemos factorizar el anterior resultado con la función "Factor[expresión]":

```
Factor[%]
(3- x+ x2)3
```

Y podemos llevar a cabo la operación contraria, es decir expandir la expresión (realizando todas las operaciones de multiplicación y potenciación posibles) con "Expand[expresión]":

```
Expand[%]
27- 27 x+ 36 x2- 19 x3+ 12 x4- 3 x5+ x6
```

"Expand" no actúa sobre las expresiones que se encuentran dentro de otras, por ejemplo, aplicando "Expand" a la siguiente expresión, la misma no cambia:

```
Expand[Sqrt[(2-x)^3]]
√(2- x)3
```

En esos casos se debe emplear "Expand" directamente en la expresión interna:

```
Sqrt[Expand[(2-x)^3]]
√(8- 12 x+ 6 x2- x3)
```

O se puede emplear "ExpandAll[expresión]":

```
ExpandAll[Sqrt[(2-x)^3]]
√(8- 12 x+ 6 x2- x3)
```

Si sólo se quiere expandir un grupo específico de expresiones, tanto "Expand" como "ExpandAll" admiten un patrón, así por ejemplo en la siguiente expresión sólo se expanden las expresiones que tienen la expresión "1-x":

```
Expand[(1-x)^2+(3+x)^3-(1-x)^4+(5-x)^4,1-x]
(5- x)4+ 2 x- 5 x2+ 4 x3- x4+ (3+ x)3
```

Para llevar una expresión a una forma más simplificada se puede emplear "Simplify[expresión]", aplicando esta función a la siguiente expresión obtenemos:

```
Simplify[Sqrt[(3-4*x+2*x)/(2-5*x+6*x)]/Sqrt[3-3*x+x]]
```

$$\frac{\sqrt{\frac{3-2x}{2-x}}}{\sqrt{3-2x}}$$

En ocasiones es posible obtener una forma más simplificada empleando "FullSimplify[expresión]", la cual en contrapartida requiere mucho más tiempo que "Simplify", así aplicando "FullSimplify" a la anterior expresión obtenemos:

```
FullSimplify[Sqrt[(3-4*x+2*x)/(2-5*x+6*x)]/Sqrt[3-3*x+x]]
```

$$\sqrt{\frac{1}{2-x}}$$

Para agrupar dos o más términos en una expresión (con un denominador común) se puede emplear "Together[expresión]", aplicando esta función a la siguiente expresión se obtiene:

```
Together[(3+x)/(5x+2)+(6x+2)/(x+3)+(7x-3)/(x+5)]
```

$$\frac{47 + 144x + 287x^2 + 66x^3}{(3+x)(5+x)(2+5x)}$$

La operación contraria, es decir separar una expresión en dos o más expresiones con denominadores mínimos se consigue con "Apart[expresión]", así aplicando esta función al anterior resultado obtenemos:

```
Apart[%]
```

$$\frac{66}{5} + \frac{16}{3+x} + \frac{38}{5+x} + \frac{13}{5(2+5x)}$$

Que no reproduce la expresión original porque en la expresión original los denominadores no eran mínimos.

Para cancelar factores comunes entre el numerador y denominador de una o más expresiones se emplea "Cancel[expresión]". Si los términos son exactamente iguales no se requiere "Cancel", pues Mathematica los simplifica directamente, así al escribir la siguiente expresión obtenemos:

```
(3x+2)(5x^2+2)/((3x+2)(6x-3))
```

$$\frac{2+5x^2}{3+6x}$$

"Cancel" se aplica cuando se requieren operaciones adicionales para identificar factores comunes, por ejemplo al escribir la siguiente expresión no se lleva a cabo ninguna simplificación:

```
(x^3-x^2-10x-8)/(2+3x+x^2)
```

$$\frac{8+10x+x^2+x^3}{2+3x+x^2}$$

Sin embargo aplicando "Cancel" obtenemos:

```
Cancel[(x^3-x^2-10x-8)/(2+3x+x^2)]
```

$$-4+x$$

En ocasiones es conveniente extraer el numerador o denominador de una expresión, en tales casos se emplean las funciones "Numerator[expresión]" y "Denominator[expresión]". Por ejemplo aplicando estas funciones a la expresión del ejemplo anterior obtenemos:

```
Numerator[(x^3-x^2-10x-8)/(2+3x+x^2)]
```

$$8+10x+x^2+x^3$$

```
Denominator[(x^3-x^2-10x-8)/(2+3x+x^2)]
```


$$2 + 3x + x^2$$

Mathematica cuenta con muchas otras funciones para transformar y llevar a cabo operaciones simbólicas. Así cuando se trabaja con expresiones trigonométricas se pueden emplear "TrigExpand", "TrigFactor", "TrigReduce", "TrigToExp" y "ExpToTrig" para expandir, factorizar, reducir y transformar expresiones trigonométrica en exponencial y viceversa. Con un conjunto tan amplio de funciones, en conjunción con los patrones y las reglas de transformación estudiadas anteriormente, es posible resolver casi todos los problemas simbólicos que se puedan presentar, sin embargo, y como ya se dijo previamente, el objetivo de la materia es obtener soluciones numéricas, no simbólicas, razón por la cual no profundizaremos mayormente en estas funciones, aunque sí estudiaremos y aplicaremos algunas de ellas en la deducción de ecuaciones y en la obtención de valores de prueba.

4.1.1. Ejemplos

1. Agrupe los términos de la siguiente expresión, expanda la expresión resultante, factorice la expresión resultante y simplifique completamente la expresión resultante.

$$\frac{x^2 - 3x + 2}{x^2 + 2x - 3} + \frac{x^3 + 5x^2 + 10x + 8}{x^2 + 3x + 2}$$

```
(x^2-3*x+2)/(x^2+2*x-3)+(x^3+5*x^2+10*x+8)/(x^2+3*x+2)//Together
```

$$10 + 12x + 7x^2 + x^3$$

$$\frac{1 + x + 3x}{1 + x + 3x}$$

```
%//Expand
```

$$\frac{10}{1 + x + 3x} + \frac{12x}{1 + x + 3x} + \frac{7x^2}{1 + x + 3x} + \frac{x^3}{1 + x + 3x}$$

```
%//Factor
```

$$\frac{5 + x + 2 + 2x + x^2}{1 + x + 3x}$$

```
%//FullSimplify
```

$$3 + x + \frac{2}{1 + x} + \frac{5}{3 + x}$$

2. Separe los términos de la siguiente expresión, expanda la expresión resultante (incluidas las subexpresiones), una los términos de la expresión resultante y simplifique la expresión resultante.

$$\frac{(x+1)(x-3)(x+4)\sqrt{(1+x)(2+x)}}{(x-3)\ln((x-3)(x+2)(x-1))}$$

```
(x+1)*(x-3)*(x+4)*Sqrt[(1+x)*(2+x)]/((x-3)*Log[(x-3)*(x+2)*(x-1)])//Apart
```

$$\frac{4\sqrt{2 + 3x + x^2}}{\text{Log} \left[\frac{3 + x}{1 + x + 2x} \right]}$$

$$\frac{5x\sqrt{2 + 3x + x^2}}{\text{Log} \left[\frac{3 + x}{1 + x + 2x} \right]} + \frac{x^2\sqrt{2 + 3x + x^2}}{\text{Log} \left[\frac{3 + x}{1 + x + 2x} \right]}$$

```
%//ExpandAll
```

$$\frac{4\sqrt{2 + 3x + x^2}}{\text{Log} \left[6 + 5x + 2x^2 + x^3 \right]} + \frac{5x\sqrt{2 + 3x + x^2}}{\text{Log} \left[6 + 5x + 2x^2 + x^3 \right]} + \frac{x^2\sqrt{2 + 3x + x^2}}{\text{Log} \left[6 + 5x + 2x^2 + x^3 \right]}$$

```
%//Together
```

$$\frac{4\sqrt{2+3x+x^2} + 5x\sqrt{2+3x+x^2} + x^2\sqrt{2+3x+x^2}}{\text{Log}[6 \cdot 5x \cdot 2x^2 + x^3]}$$

%//Simplify

$$\frac{\sqrt{2+3x+x^2} \left[4 + 5x + x^2 \right]}{\text{Log}[6 \cdot 5x \cdot 2x^2 + x^3]}$$

3. Agrupe los términos de la siguiente expresión, cancele términos comunes y separe la expresión resultante.

$$\frac{x+1}{x+3} + \frac{x-5}{x+1} + \frac{x+7}{x+2}$$

(x+1)/(x+3)+(x-5)/(x+1)+(x+7)/(x+2)//Together

$$\frac{(x+1)(x+2)(x+1) + (x-5)(x+2)(x+3) + (x+7)(x+3)(x+1)}{(x+3)(x+1)(x+2)}$$

%//Cancel

$$\frac{(x+1)(x+2)(x+1) + (x-5)(x+2)(x+3) + (x+7)(x+3)(x+1)}{(x+3)(x+1)(x+2)}$$

%//Apart

3. $\frac{6}{1+x} + \frac{5}{2+x} + \frac{2}{3+x}$

4. Cancele términos comunes en la siguiente expresión, factorice la expresión resultante, expanda sólo el numerador de la expresión, luego expanda sólo el denominador de la expresión, luego factorice el numerador y simplifique completamente la expresión resultante.

$$\frac{x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100}{x^5 + x^4 - 27x^3 - x^2 + 146x - 120}$$

e=Cancel[(x^5+6*x^4-44*x^3-234*x^2+475*x+2100)/(x^5+x^4-27*x^3-x^2+146*x-120)]

$$\frac{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}$$

e=Factor[e]

$$\frac{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}$$

e=Expand[Numerator[e]]/Denominator[e]

$$\frac{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}$$

e=Numerator[e]/Expand[Denominator[e]]

$$\frac{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}$$

e=FullSimplify[e]

$$\frac{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}{(x^5 + 6x^4 - 44x^3 - 234x^2 + 475x + 2100)(x^5 + x^4 - 27x^3 - x^2 + 146x - 120)}$$

4.1.2. Ejercicios

1. Expanda el numerador y denominador de la siguiente expresión, simplifique términos comunes y separe la expresión resultante.

$$\frac{(x+y)^2(x+y) + x^3(x+y) + 3x^2(x+y)}{x(x+y) + 5xy(x+y)}$$

2. Una los términos de la siguiente expresión, cancele términos comunes,

factorice el numerador y denominador y simplifique la expresión resultante.

$$\frac{x^2}{x+y^2} + \frac{3x^2y}{x+y^2} + \frac{3xy^2}{x+y^2} + \frac{y^3}{x+y^2}.$$

3. Agrupe los términos de la siguiente expresión, expanda la expresión resultante, factorice la expresión resultante y simplifique completamente la expresión resultante.

$$\frac{x^2-2x+1}{x^2+x-1} + \frac{x^3+x^2+x+3}{x^2-3x+2}$$

4. Cancele términos comunes en la siguiente expresión, factorice la expresión resultante, expanda sólo el numerador de la expresión, luego expanda sólo el denominador de la expresión, luego factorice el numerador y simplifique completamente la expresión resultante.

$$\frac{x^6-9x^5-5x^4+165x^3+4x^2-876x-720}{x^6-14x^4+49x^2-36}$$

5. Separe los términos de la siguiente expresión, expanda la expresión resultante (incluidas las subexpresiones), una los términos de la expresión resultante y simplifique la expresión resultante.

$$\frac{(x-1)(x+3)(x+2)\sqrt{(x+3)(x+2)(x-1)}}{(x+1)(x-4)\ln((x+3)(x-2)(x+1))}$$

4.2. Funciones en Mathematica

Las funciones en Mathematica son simplemente variables a las cuales se les asigna una expresión, una serie de expresiones o un programa. Como todas las variables las funciones pueden ser creadas mediante asignación inmediata (=) o retardada (:=) y una función simple (con una sola expresión) se crea de acuerdo a uno de los siguientes formatos:

Nombre_de_la_función[parámetros]= expresión;

Nombre_de_la_función[parámetros]:= expresión

Donde el nombre de la función es cualquier combinación de letras o letras y números (sin espacios en blanco). Al respecto se debe tomar en cuenta que todas las funciones de Mathematica comienzan con una letra mayúscula, por lo que en lo posible es conveniente evitar este tipo de nombres, para no tener conflictos con los nombres ya existentes.

Los parámetros son "patrones" con nombres (separados con comas) que representan a los valores (argumentos) que recibe la función y que son empleados en la "expresión".

Por ejemplo para evaluar la siguiente expresión matemática:

$$\frac{3 \cdot 5x}{x^3} \left(\frac{3 + x^2}{\log x} \right)^{1/3}$$

Podemos crear la función "f1" con una asignación inmediata:

```
f1[x_]=((3+x^2)^2/Log[x])^(1/3)/Exp[(5*x+3)/x^3];
```

Ahora podemos emplear la función "f1" para obtener, por ejemplo, el valor de la expresión para x=5.1:

```
f1[5.1]
```

6.47235

O la expresión cuando la variable es "z":

```
f1[z]
      3 5z
      z3
      ( 3 + z2 )1/3
      Log[z]
```

Igualmente podemos crear otra función "f2" con una asignación retardada:

```
f2[x_]:=((3+x^2)^2/Log[x])^(1/3)/Exp[(5*x+3)/x^3]
```

Y evaluar la función para los dos casos anteriores:

```
f2[5.1]
6.47235
f2[z]
      3 5z
      z3
      ( 3 + z2 )1/3
      Log[z]
```

Pareciera entonces que da lo mismo crear una función con una asignación inmediata y una retardada, sin embargo, si "x" tiene un valor, por ejemplo "2.1", y creamos la función con una asignación inmediata:

```
x=2.1
2.1
f1[x_]:=((3+x^2)^2/Log[x])^(1/3)/Exp[(5*x+3)/x^3];
```

Al evaluar la función para "x=5.1" y para "x=z" obtenemos:

```
f1[5.1]
0.977254
f1[z]
0.977254
```

Que como vemos no corresponde a los resultados correctos, esto sucede porque (como ya vimos) en la asignación inmediata primero se evalúa la expresión y es el resultado de dicha evaluación el que se asigna a la variable. En consecuencia "f1" es en realidad una constante y sin importar que argumento se mande a la función siempre devolverá: "0.977254".

Ahora si volvemos a crear la función "f2", con una asignación retardada, teniendo la variable "x" el valor "2.1":

```
x=2.1;
f2[x_]:=((3+x^2)^2/Log[x])^(1/3)/Exp[(5*x+3)/x^3]
```

Y evaluamos la función para "x=5.1" y "x=z", obtenemos:

```
f2[5.1]
6.47235
f2[z]
      3 5z
      z3
      ( 3 + z2 )1/3
      Log[z]
```

Que son los resultados correctos, esto sucede así porque (como ya vimos) en la asignación retardada lo que se asigna a la variable es la expresión sin evaluar, siendo evaluada recién cuando se invoca a la variable (o función). Debido a que es muy fácil olvidar liberar variables, y en consecuencia cometer este tipo de errores, emplearemos casi exclusivamente la asignación retardada en la creación de funciones.

Como segundo ejemplo crearemos una función para evaluar una ecuación:

$$a + bx + cx^2 + dx^3$$

En este caso la función debe recibir cinco parámetros: "a, b, c, d y x":

```
cubica[a_,b_,c_,d_]:=a+b*x+c*x^2+d*x^3
```

Ahora podemos emplear esta función para calcular, por ejemplo, el valor de la ecuación: " $4+3x+2x^2+x^3$ " para " $x=5$ ":

```
cubica[4,3,2,1,5]
194
```

O para obtener una ecuación cúbica simbólica, donde los coeficientes son " a_1, a_2, a_3, a_4 " y la variable es "y":

```
cubica[a1,a2,a3,a4,y]
a1+y a2+y^2 a3+y^3 a4
```

Mathematica, al igual que Pascal, permite establecer valores por defecto para los parámetros de una función. Para asignar un valor por defecto el parámetro se escribe de acuerdo al siguiente formato: "patrón:valor". Los parámetros por defecto se escriben al final de la lista de parámetros y si su valor es omitido, se les asigna el valor por defecto, así en la siguiente función, que recibe cuatro parámetros, se asignan valores por defecto a los dos últimos:

```
f3[a_,b_,c_:1,d_:2]:=Sqrt[a^2+b^2+c^2+d^2]
```

En la siguiente llamada, "a", "b", "c" y "d" toman los valores "1", "2", "3" y "4" respectivamente:

```
f3[1,2,3,4]
√30
```

En la siguiente llamada, "a", "b" y "c" toman los valores "1", "2" y "3", mientras que "d" toma el valor por defecto "2":

```
f3[1,2,3]
3√2
```

En la siguiente llamada "a" y "b" toman los valores "1" y "2", mientras que "c" y "d" toman los valores por defecto "1" y "2" respectivamente:

```
f3[1,2]
√10
```

El problema de los valores por defecto es que Mathematica siempre asume que el parámetro que se omiten es el último (o los últimos). Cuando se quieren asignar valores por defecto a varios parámetros y el parámetro que se omite no siempre es el último (o últimos), resulta más eficiente el uso de parámetros opcionales. Los parámetros opcionales se asignan a una función en forma de reglas, empleando la instrucción "Options". Por ejemplo en la siguiente instrucción se asignan parámetros opcionales (" x_1 ", " x_2 ", " x_3 " y " x_4 ") a la función "f4":

```
Options[f4]:={x1->1,x2->2,x3->3,x4->4}
```

Con "Options" se pueden ver también los parámetros opcionales que tiene una función:

```
Options[f4]
{x1->1,x2->2,x3->3,x4->4}
```

Los parámetros opcionales de una función se representan con la instrucción "OptionsPattern[]", y los valores de los parámetros opcionales se obtienen con "OptionValue[parámetro_opcional]". Así en la siguiente instruc-

ción se crea la función "f4" con los parámetros opcionales asignados previamente:

```
f4[x0_,OptionsPattern[]]:=x0^2+OptionValue[x1]^2+OptionValue[x2]^2+OptionValue[x3]^2+OptionValue[x4]^2
```

Si evaluamos la función para "x0=5", los parámetros opcionales mantienen sus valores por defecto:

```
f4[5]
55
```

Los parámetros opcionales pueden ser cambiados escribiendo nuevos valores en forma de reglas. Así en el siguiente ejemplo se cambian los valores de los parámetros opcionales "x1" y "x4" a "0":

```
f4[5,x1->0,x4->0]
38
```

Como se puede observar, la ventaja de los parámetros opcionales, con relación a los parámetros por defecto, es que se puede omitir cualquiera de los parámetros y no necesariamente los últimos, en contrapartida, es necesario recordar el nombre del parámetro.

Por otra parte, Mathematica permite crear funciones puras (sin nombre), las que se crean y ejecutan directamente (sin asignarles un nombre), inclusive dentro de otras instrucciones. Para crear una función pura se emplea la instrucción "Function":

```
Function[{parámetros}, instrucciones]
```

O si no existen parámetros formales, una de las siguientes formas:

```
Function[instrucciones]
```

```
O instrucciones&
```

Cuando no se emplean parámetros formales, los nombres de los parámetros se denominan: "#", si la función recibe un solo parámetro y "#1", "#2", ..., "#n", si la función recibe "n" parámetros. Por ejemplo en la siguiente instrucción se crea una función para calcular la raíz cúbica y se aplica la misma a los elementos de una lista:

```
#^(1/3)&[{1.,2.,5.,8.,9.}]
{1.,1.25992,1.70998,2.,2.08008}
```

Por supuesto la misma función puede ser creada con "Function":

```
Function[#^(1/3)][{1.,2.,5.,8.,9.}]
{1.,1.25992,1.70998,2.,2.08008}
```

En la siguiente instrucción se crea una lista con las raíces cuadradas de la suma de dos números consecutivos que varían del 1 al 10:

```
Table[Sqrt[#1+#2]&[i,i+1],{i,1,10,2}]
|√3,√7,√11,√15,√19|
```

Cuando se emplean parámetros formales, los mismos sólo tienen validez al interior de la función, es decir que se comportan como variables locales. En la siguiente instrucción se emplean parámetros formales para crear una lista con 5 elementos correspondientes a la suma de los cuadrados de tres números conformados por el número, el doble del número y el triple del mismo:

```
Table[Function[{x,y,z},x^2+y^2+z^2][i,i*2,i*3],{i,1,5}]
{14,56,126,224,350}
```

Si ahora invocamos los valores de "x", "y", y "z" obtenemos:

```
{x,y,z}
{x,y,z}
```

Lo que demuestra que las variables "x", "y" y "z" de la función son sólo locales, es decir que sólo existen mientras se ejecuta la función.

4.2.1. Ejemplos

5. Cree una función para calcular el valor de la siguiente expresión y luego evalúe la función para "x = 3.,4.,5.,6.,7.,8.,9.,10.".

$$f(x) = \frac{e^{x^2} + 2x + 3 - \log(x)}{3x + 2}$$

```
f1[x_] := (Exp[x^2] + 2*x + 3 - Log[10, x]) / (3*x + 2)
f1[Range[3., 10.]]
{737.419, 634723., 4.23558*10^9, 2.15562*10^14, 8.29281*10^19, 2.39813*10^26, 5.19344*10^33, 8.40037*10^41}
O también:
Range[3., 10.] // f1
{737.419, 634723., 4.23558*10^9, 2.15562*10^14, 8.29281*10^19, 2.39813*10^26, 5.19344*10^33, 8.40037*10^41}
O también:
f1[x] /. x -> Range[3., 10.]
{737.419, 634723., 4.23558*10^9, 2.15562*10^14, 8.29281*10^19, 2.39813*10^26, 5.19344*10^33, 8.40037*10^41}
```

6. Cree una función para calcular el valor de la siguiente expresión y luego evalúe la función para "x=1., y=3."; "x=2.3, y=0.4"; "x=4.3, y=6.2", "x=0.4, y=0.9".

$$f(x, y) = \sqrt[3]{\frac{\log(x+y) + \ln(x+2y) + 3x + 4}{x^3 + 2xy^2 + 3y^2 + y}}$$

```
f2[x_, y_] := ((Log[10, x+y] + Log[x+2*y] + 3*x + 4) / (x^3 + 2*x*y^2 + 3*y^2 + y)) ^ (1/3)
f2[{1., 2.3, 4.3, 0.4}, {3., 0.4, 6.2, 0.9}]
{0.579747, 0.966994, 0.339142, 1.14719}
O también:
Sequence[{1., 2.3, 4.3, 0.4}, {3., 0.4, 6.2, 0.9}] // f2
{0.579747, 0.966994, 0.339142, 1.14719}
O también:
f2[x, y] /. {x -> {1., 2.3, 4.3, 0.4}, y -> {3., 0.4, 6.2, 0.9}}
{0.579747, 0.966994, 0.339142, 1.14719}
```

7. Cree una función para calcular el entero más pequeño menor o igual al resultado de la siguiente expresión, asignando a "y" un valor por defecto igual a "2.3". Luego evalúe la función para "x=1.1, y=2.1"; "x=1.4, y=3.2"; "x=1.6"; "x=1.7".

$$f(x, y) = \sin(x + y) + 2\cos(x - 2y) + 3xy$$

```
f3[x_, y_: 2.3] := Floor[Sin[x+y] + 2*Cos[x-2*y] + 3*x*y]
{f3[1.1, 2.1], f3[1.4, 3.2], f3[1.6], f3[1.7]}
{4, 13, 8, 9}
```

8. Cree una función para calcular la parte fraccionaria del resultado de la siguiente expresión, asignando a "y" y a "z" valores por defecto iguales a "1.2" y "2.3" respectivamente. Luego evalúe la función para "x=3.2"; "x=6.2, y=4.3", "x=4.3, y=3.4, z=5.1".

$$f(x,y,z) = \frac{\tan\left(\frac{x+y}{x+z}\right) + \tan^{-1}\left(\frac{2x+3}{3y+z}\right)}{\tanh\left(\frac{x+y+z}{2x+3y}\right) + \tanh^{-1}\left(\frac{y+2z}{x+3z}\right)}$$

```
f4[x_,y_: 1.2,z_: 2.3]:=FractionalPart[(Tan[(x+y)/(x+z)]+ArcTan[(2*x+3)/(3*y+z)])/(Tanh[(x+y+z)/(2*x+3*y)]+ArcTanh[(y+2*z)/(x+3*z)])]
{f4[3.2],f4[6.2,4.3],f4[4.3,3.4,5.1]}
{0.646652,0.826699,0.187312}
```

9. Cree una función para calcular el valor absoluto de la siguiente expresión, siendo "a", "b", "c" y "d" opcionales con valores por defecto iguales a "1", "2", "3" y "4" respectivamente. Luego evalúe la función para "x=3", "x=0.2", "x=2, a=3", "x=4,b=0.2,c=1.2", "x=3.2,a=3,d=1".

$$f(x) = a - b*x^2 + c*x^3 - d*x^4$$

```
Options[f5]={a->1,b->2,c->3,d->4}
{a->1,b->2,c->3,d->4}
f5[x_,OptionsPattern[]]:=Abs[OptionValue[a]-
OptionValue[b]*x^2+OptionValue[c]*x^3-OptionValue[d]*x^4]
{f5[3],f5[0.2],f5[2,a->3],f5[4,b->0.2,c->1.2],f5[3.2,a->3,d->1]}
{260,0.9376,45,949.4,24.0336}
```

10. Cree una función para calcular el valor redondeado de la siguiente expresión, siendo "c₁", "c₂" y "c₃" opcionales, con valores por defecto iguales a "3", "4" y "7" respectivamente. Luego evalúe la función para "x=0.2", "x=0.3, c₁=0.4", "x=0.9, c₂=0.8, c₃=0.4", "x=1.2, c₁=0.3, c₂=0.8, c₃=0.9".

$$f(x) = \frac{c_1 + c_2/x + c_3/\ln(x)}{\sqrt[3]{c_1 + c_2 + c_3}}$$

```
Options[f6]={c1->3,c2->4,c3->7}
{c1->3,c2->4,c3->7}
f6[x_,OptionsPattern[]]:=Round[(OptionValue[c1]+OptionValue[c2]/x+OptionValue[c3]/Log[x])/(OptionValue[c1]+OptionValue[c2]+OptionValue[c3])^(1/3)]
{f6[0.2],f6[0.3,c1->0.4],f6[0.9,c1->0.8,c3->0.4],f6[1.2,c1->0.3,c2->0.8,c3->0.9]}
{8,4,1,5}
```

11. Cree una lista con los resultados de "(x+5)²", variando "x" desde 1 hasta 3 con incrementos iguales a 0.2.

```
(#+5)^2&[Range[1,3,0.2]]
{36.,38.44,40.96,43.56,46.24,49.,51.84,54.76,57.76,60.84,64.}
O también:
Function[x,(x+5)^2][Range[1,3,0.2]]
{36.,38.44,40.96,43.56,46.24,49.,51.84,54.76,57.76,60.84,64.}
```

12. Cree una lista con los resultados de "sen(2x)+3*cos(x)", variando "x" desde 0 hasta 2π, con incrementos iguales a π/4.

```
Sin[2*#]+3*Cos[#]&[Range[0,2*Pi,Pi/4]]
{3, 1+ 3/sqrt(2), 0, -1+ 3/sqrt(2), -3, 1- 3/sqrt(2), 0, -1- 3/sqrt(2), 3}
O también:
Function[x,Sin[2*x]+3*Cos[x]][Range[0,2*Pi,Pi/4]]
{3, 1+ 3/sqrt(2), 0, -1+ 3/sqrt(2), -3, 1- 3/sqrt(2), 0, -1- 3/sqrt(2), 3}
```


13. Cree una lista con los cuadrados de los cocientes de " $x/3$ ", donde " x " varía desde 1 hasta 21 con incrementos iguales a 3.

```
Quotient[#,3]^2&[Range[1,21,3]]
{0,1,4,9,16,25,36}
```

O también

```
Function[x,Quotient[x,3]^2][Range[1,21,3]]
{0,1,4,9,16,25,36}
```

14. Cree una lista con los resultados de " $(x^3+2y+3)^{1/5}$ ", variando " x " desde 1 hasta 3, con incrementos de 0.2 y siendo " y " el cuadrado de " x ".

```
(#1^3+2*#2+3)^(1/5)&[Range[1,3,0.2],Range[1,3,0.2]^2]
{1.43097,1.50056,1.5741,1.64963,1.72586,1.80198,1.87748,1.95206,2.02554,2.09785,2.16894}
```

O también:

```
Function[{x,y},(x^3+2*y+3)^(1/5)][Range[1,3,0.2],Range[1,3,0.2]^2]
{1.43097,1.50056,1.5741,1.64963,1.72586,1.80198,1.87748,1.95206,2.02554,2.09785,2.16894}
```

4.2.2. Ejercicios

6. Cree una función para calcular el valor de la siguiente expresión y luego evalúe la función para " $x = -5., -4., -3., -2., -1., 0., 1., 2., 3., 4., 5.$ ".

$$f(x) = x^3 + 2x^2 + 3x + 4$$

7. Cree una función para calcular el valor de la siguiente expresión y luego evalúe la función para " $x=1., y=2.$ "; " $x=2.3, y=1.4$ "; " $x=4.1, y=5.2$ ", " $x=0.7, y=0.9$ ".

$$f(x,y) = \sqrt[4]{\frac{\ln(x+y) - x^2 + y^2}{e^{x+y} + x - y}}$$

8. Cree una función para calcular el entero más pequeño mayor o igual al resultado de la siguiente expresión, asignando a " y " un valor por defecto igual a "0.7". Luego evalúe la función para " $x=2.1, y=1.1$ "; " $x=2.4, y=1.2$ "; " $x=1.3$ "; " $x=2.7$ "; " $x=3.5$ ".

$$f(x,y) = x + 5.25xy^2 - \sec(\sqrt{0.68xy})$$

9. Cree una función para calcular la parte entera del resultado de la siguiente expresión, asignando a " y " y a " z " valores por defecto iguales a "3.2" y "1.3" respectivamente. Luego evalúe la función para " $x=1.2$ "; " $x=2.4$ ", " $x=3.2, y=1.3$ ", " $x=1.1, y=3.1$ ", " $x=2.3, y=2.4, z=2.5$ ".

$$f(x,y,z) = x^3yz - 3xy^2z + 4xyz^2$$

10. Cree una función para calcular el valor de la siguiente expresión, siendo " k ", " l ", " m " y " n " opcionales con valores por defecto iguales a "0.1", "0.2", "0.3" y "0.4" respectivamente. Luego evalúe la función para " $x=3$ ", " $x=0.2$ ", " $x=2, k=3$ ", " $x=4, l=0.2, m=1.2$ ", " $x=3.2, l=3, n=1$ ", " $x=0.2, k=1.2, l=3.1, n=0.3$ ".

$$f(x) = \sqrt[3]{\frac{(2k+3l)x - (6m-4n)x^2}{\tan^{-1}(k^x + mx^2 + nx^3)}}$$

11. Cree una función para calcular el entero más pequeño, mayor o igual al resultado de la siguiente expresión, siendo " a_1 ", " a_2 " y " a_3 " opciona-

les, con valores por defecto iguales a "1.0", "1.1" y "1.2" respectivamente. Luego evalúe la función para "x=1.1", "x=1.3", $a_1=0.8$, "x=0.9", $a_2=2.2$, $a_3=2.4$ ", "x=1.2", $a_1=3.3$, $a_2=3.8$, $a_3=3.9$ ".

$$f(x) = \frac{a_1 + a_2/x^2 + a_3/\ln(3+x^2)}{\sqrt[3]{\sin(a_1 + a_2)x^{1.2} - \cos(a_2 + a_3)x^{1.9}}}$$

12. Cree una lista con los resultados de $(3x^{2.2} + 5x^{1.3})^{1.3}$, variando "x" desde 1 hasta 5 con incrementos iguales a 0.4.
13. Cree una lista con los resultados de $\sin^{-1}(3x+2) + 3\cos^{-1}(5x+3)$, variando "x" desde 0 hasta 3π , con incrementos iguales a $\pi/5$.
14. Cree una lista con los cuadrados de los residuos de "x/7", donde "x" varía desde 12 hasta 35 con incrementos iguales a 2.
15. Cree una lista con los resultados de $(yx^3 + 2yx^{1.3} + 3xy)^{1/3}$, variando "x" desde 2 hasta 5, con incrementos de 0.2 y siendo "y" la mitad de "x".
16. Cree una lista con los resultados de $e^{3x+2y-xy}$, variando "x" desde 0.1 hasta 1.3, con incrementos de 0.1 y siendo "y" el cubo de "x".

4.3. Secuencias de instrucciones

Cuando en una función se requieren evaluar dos o más expresiones consecutivas, se debe escribir una "secuencia". Como recordarán de su estudio en Informática I, las estructuras de control que permiten escribir programas en un lenguaje estructurado (como Pascal) son tres: a) secuencia, b) selección e c) iteración. Con estas tres estructuras, conjuntamente la "recursividad", se pueden resolver desde los problemas más simples a los más complejos. Mathematica soporta los tres tipos de estructuras de control, así como la recursividad, posibilitando la resolución de problemas mediante la elaboración de programas.

La estructura de control más simple, pero al mismo tiempo la base para las otras estructuras, es la "secuencia". Recordemos que en Pascal una secuencia se crea encerrando las instrucciones consecutivas (separadas con puntos y comas ";") entre las palabras "begin" y "end". En Mathematica, al igual que en Pascal, una secuencia se crea escribiendo las instrucciones consecutivas separadas con puntos y comas ";", pero a diferencia de Pascal no existe una única forma de agruparlas. En Mathematica es posible incluso escribir una secuencia sin agruparla, así las siguientes instrucciones, donde se calcula el valor de la expresión:

$$\left(\frac{x^2 + y^2 + z^2}{xyz} \right)^{1/3}$$

Para "x=3.1", "y=2.2" y "z=1.3", constituye una secuencia:

```
x=3.1;y=2.2;z=1.3;((x^2+y^2+z^2)/(x*y*z))^(1/3)
1.22103
```

En ocasiones, sin embargo, sobre todo cuando se crean funciones, es necesario agrupar las secuencias y para ello Mathematica dispone de varias opciones. La forma más simple de ellas consiste en el uso de paréntesis. Así, empleando paréntesis, podemos crear una función para evaluar el valor de "f(x)":

$$f(x) = x + y^2 + z^2 - 14 = 0$$

$$y = \frac{3x^{2.1} + 7.0}{5}$$

$$z = \frac{14.3 - y^{1.2}}{4}$$

```
f[x_] := (y = (4*x^3.1 + 7)/5; z = (14.3 - y^1.2)/4; x + y^2 + z^2 - 14)
```

Y con la misma podemos calcular el valor "f" para "x=1.5":

```
f[1.5]
9.95292
```

O el valor de "f", cuando "x=w":

```
f[w]
. 14 + w + 1/25 (7 + 4 w^3.1)^2 + 1/16 (14.3 - 0.14495593273553914 (7 + 4 w^3.1)^1.2)^2
```

No obstante, la función "f" emplea dos variables globales: "y" y "z" cuyos valores son modificados cada vez que se llama a la función (pues no son parámetros), así los valores de estas variables después de la última llamada son:

```
y
1/5 (7 + 4 w^3.1)
z
1/4 (14.3 - 0.144956 (7 + 4 w^3.1)^1.2)
```

Ahora bien, si estas variables están siendo empleadas, por ejemplo, para guardar resultados de otros cálculos, dichos resultados se pierden definitivamente al llamar a la función. Por eso es siempre recomendable evitar el uso de variables globales cuando se trabaja con funciones, ello se puede lograr encerrando la secuencia de instrucciones en módulos (Module) de acuerdo al siguiente formato:

```
Module[{variables_locales}, instrucciones]
```

Donde "variables_locales" son los nombres de las variables locales (separadas con comas). Al igual que en Pascal, en Mathematica, las variables locales son aquellas que existen sólo mientras la función, o programa, está en ejecución, una vez que el programa termina las variables locales son destruidas, de esa manera no interfieren con ninguna variable global, evitando así los problemas antes mencionados.

A las "variables_locales" en Mathematica se les pueden asignar valores iniciales, para ello simplemente se escribe el valor a continuación del nombre de la variable, separados por el símbolo igual: "variable_local=valor_inicial". Si a una variable local no se le asigna valor, ya sea en la declaración o en la secuencia de instrucciones, se comporta como una variable simbólica.

Empleando "Module" podemos volver a crear la función "f", pero empleando ahora dos variables locales "y" y "z", para comprobar que el parámetro y variables locales de la función no interfieren con las variables globales, asignaremos también valores a las variables "x", "y" y "z":

```
x=1;y=2;z=3;
Remove[f]
f[x_] := Module[{y, z}, y = (4*x^3.1 + 7)/5; z = (14.3 - y^1.2)/4; x + y^2 + z^2 - 14]
```

Ahora calculemos el valor de la función para "x=1.5":

```
f[1.5]
9.95292
```

Como era de esperar obtenemos el mismo resultado que con la función creada previamente, sin embargo, si ahora invocamos los valores de "x", "y" y "z":

```
{x,y,z}
{1,2,3}
```

Podemos comprobar que sus valores no han sido modificados por la ejecución del programa.

Otra forma de agrupar secuencias y trabajar con valores locales es mediante la instrucción "Block", esta instrucción tiene una estructura muy similar a la de "Module" y se emplea esencialmente de la misma forma:

Block[{variables}, instrucciones]

En este caso sin embargo, las "variables" no son propiamente variables locales, sino variables globales, que tienen valores locales (temporales) dentro del bloque. La forma en que opera "Block" es ligeramente diferente a la de "Module". "Block" emplea variables globales a las cuales asigna valores temporales, dichos valores tienen efecto mientras se ejecutan las "instrucciones" y una vez que las mismas terminan, "Block" restaura los valores originales de las variables globales. En consecuencia "Block" se emplea sobre todo cuando se quiere trabajar con variables globales asignándoles valores temporales. No obstante en la mayoría de los casos se puede considerar que tanto "Block" como "Module" hacen esencialmente lo mismo, pues ambos permiten trabajar con variables sin que las mismas interfieran con las variables globales.

Así por ejemplo podemos crear una vez más la función "f", pero en este caso empleando "Block":

```
Remove[f,x,y,z]
x=1;y=2;z=3;
f[x_]:=Block[{y,z},y=(4*x^3.1+7)/5;z=(14.3-y^1.2)/4;x+y^2+z^2-14]
```

Ejecutando la función para "x=1.5":

```
f[1.5]
9.95292
```

Obtenemos el resultado correcto y si invocamos los valores de las variables "x", "y" y "z",

```
{x,y,z}
{1,2,3}
```

Vemos que sus valores no han sido modificados, tal como ocurre con "Module". Esto es lo que normalmente sucede en la mayoría de los casos, tanto "Module" como "Block" devuelven los mismos resultados. Sin embargo, si en la secuencia de instrucciones se invoca una expresión que emplea variables globales, "Module" y "Block" se comportan de manera diferente. Por ejemplo, si asignamos la siguiente expresión a la variable "g":

```
Remove[x,y,z]
g=(y+z)^2/(y*z)^3;
```

Y creamos la función "f1", donde invocamos a esta variable:

```
f1[x_]:=Module[{y,z},y=2*x;z=4*x;g]
```

Al evaluar la función para "x=1.1" obtenemos:

```
f1[1.1]
(y+z)^2
-----
y^3 z^3
```

Que es la expresión original asignada a "g", por lo tanto los valores asignados a las variables locales "y" y "z" no son reemplazados en la expresión (porque las variables de la expresión son globales). Pero, si en lugar de "Module" creamos la función con "Block":

```
f2[x_]:=Block[{y,z},y=2*x;z=4*x;g]
```

Y evaluamos la función para "x=1.1" obtenemos:

```
f2[1.1]
0.0480244
```

Donde, como se puede observar, los valores de "y" y "z" han sido efectivamente reemplazados en la expresión asignada a "g". Si en una función se evita invocar explícitamente a variables globales, una práctica que es altamente recomendada, se pueden emplear indistintamente "Block" y "Module".

Cuando en lugar de variables se tienen valores constantes se puede emplear la instrucción "With":

```
With[{constantes}, instrucciones]
```

Esta instrucción es similar a "Module" y "Block", pero en lugar de variables se tienen constantes. Las que se declaran en la forma: "*nombre de la constante = valor*". "With" opera en la misma forma que "Module", salvo que con "With" se crean constantes en lugar de variables (es decir identificadores cuyos valores no pueden ser modificados).

En el siguiente ejemplo se crea una función para calcular el valor de $(x+y)^2/(yz)^3$ (igual que la función "f2"):

```
Remove[f]
f[x_]:=With[{y=2*x,z=4*x},(y+z)^2/(y*z)^3]
```

Ahora podemos emplear la función para evaluar la expresión para x=1.1:

```
f[1.1]
0.0480244
```

"Module", "Block" y "With" pueden anidarse a cualquier nivel, es decir una instrucción "Module" o "Block" o "With", puede contener otra instrucción "Module" o "Block" o "With" y esta a su vez puede contener otra instrucción "Module" o "Block" o "With" y así sucesivamente hasta el nivel que sea necesario. Sin embargo, se recomienda no anidar estructuras más allá de un tercer nivel, porque el código se hace menos comprensible y resulta difícil de mantener. Por otra parte cuando en dos o más estructuras anidadas existen variables (o constantes) con el mismo nombre, la variable (o constante) que es empleada para los cálculos (u operaciones) es aquella que ha sido declarada en la estructura más interna.

4.3.1. Ejemplos

15. Cree tres funciones: "f1", "f2" y "f3" que empleando "Module", "Block" y "With", devuelvan el valor de la siguiente función. Luego calcule los valores de las funciones para: "x=0.1, 0.3, 1.5, 2.9, 3.2, 5.1, 8.3".

$$f(x) = y^2 - z^3 - 5x - 166.81$$

$$y = \frac{1500 - 20x^2 + x^3}{15}; \quad z = \frac{300 + 1.5x^2 - e^{\frac{x}{2}}}{9}$$

```
f1[x_]:=Module[{y,z},
  y=(1500-20*x^2+x^3)/15;
  z=(300+1.5*x^2-Exp[x/3])/9;
  y^2-z^3-5*x-166.81
]
f2[x_]:=Block[{y,z},
  y=(1500-20*x^2+x^3)/15;
  z=(300+1.5*x^2-Exp[x/3])/9;
  y^2-z^3-5*x-166.81
]
f3[x_]:=With[{
  y=(1500-20*x^2+x^3)/15,
  z=(300+1.5*x^2-Exp[x/3])/9},
  y^2-z^3-5*x-166.81
]
L={0.1,0.3,1.5,2.9,3.2,5.1,8.3};
f1[L]
{-26830.9,-26870.8,-28401.7,-32866.8,-34188.9,-45592.4,-77839.8}
f2[L]
{-26830.9,-26870.8,-28401.7,-32866.8,-34188.9,-45592.4,-77839.8}
f3[L]
{-26830.9,-26870.8,-28401.7,-32866.8,-34188.9,-45592.4,-77839.8}
```

16. Cree dos funciones: "f1" y "f2", que empleando "Module" y "Block" devuelvan el valor de la siguiente función. Luego calcule valores de las funciones para: "x=2.5,2.8,3.1,...,5.7,6.1"

$$f(x) = x + 2y^{2.2} - z^{2.1} - 16.3$$

$$y = \frac{3x^{2.1} - 7x}{5x}; \quad z = \frac{1.3x - y^{1.2}}{4x + 5y}$$

```
Remove[f1,f2]
f1[x_]:=Module[{y,z},
  y=(3*x^2.1-7*x)/(5*x);
  z=(1.3*x-y^1.2)/(4*x+5*y);
  x+2*y^2.2-z^2.1-16.3
]
f2[x_]:=Block[{y,z},
  y=(3*x^2.1-7*x)/(5*x);
  z=(1.3*x-y^1.2)/(4*x+5*y);
  x+2*y^2.2-z^2.1-16.3
]
f1[Range[2.5,6,0.3]]
{-13.7758,-13.1838,-12.3756,-11.3243,-10.0083,-8.40874,-6.50904,-4.29399,-
1.74951,1.13757,4.37966,7.98852}
f2[Range[2.5,6,0.3]]
{-13.7758,-13.1838,-12.3756,-11.3243,-10.0083,-8.40874,-6.50904,-4.29399,-
1.74951,1.13757,4.37966,7.98852}
```

17. Cree tres funciones "f1", "f2" y "f3" que empleando "Module", "Block" y "With" devuelvan el valor de la siguiente función, donde "P", "T" y "v" son parámetros opcionales con valores por defecto iguales a 87.3, 486.9 y 12.005 respectivamente. Luego calcule valores de las funciones para "b=0,1,2,...10", "b=1, P=100", "b=1, T=500", "b=1, v=10.2", "b=1, P=90, T=490, v=11.1".

$$f(b) = P - \frac{RT}{v-b} + \frac{A}{v(v+b)}$$

$$A=0.0837; \quad R=1.98$$

```

Clear[f1,f2,f3]
Options[f1]={P→87.3,T→486.9,v→12.005};
f1[b_,OptionsPattern[]]:=Module[{A=0.0837, R=1.98,
  P=OptionValue[P],
  T=OptionValue[T],
  v=OptionValue[v]},
  P-R*T/(v-b)+A/(v*(v+b))
]
Options[f2]={P→87.3,T→486.9,v→12.005};
f2[b_,OptionsPattern[]]:=Block[{A=0.0837, R=1.98,
  P=OptionValue[P],
  T=OptionValue[T],
  v=OptionValue[v]},
  P-R*T/(v-b)+A/(v*(v+b))
]
Options[f3]={P→87.3,T→486.9,v→12.005};
f3[b_,OptionsPattern[]]:=With[{A=0.0837, R=1.98,
  P=OptionValue[P],
  T=OptionValue[T],
  v=OptionValue[v]},
  P-R*T/(v-b)+A/(v*(v+b))
]
f1[Range[0,10]]
{6.99554,-0.301645,-9.05752,-19.7581,-33.132,-50.3244,-73.2428,-105.319,-
153.414,-233.519,-393.529}
{f1[1,P→100],f1[1,T→500],f1[1,v→10.2],f1[1,P→90,T→490,v→11.1]}
{12.3984,-2.65857,-17.4886,-6.05878}
f2[Range[0,10]]
{6.99554,-0.301645,-9.05752,-19.7581,-33.132,-50.3244,-73.2428,-105.319,-
153.414,-233.519,-393.529}
{f2[1,P→100],f2[1,T→500],f2[1,v→10.2],f1[1,P→90,T→490,v→11.1]}
{12.3984,-2.65857,-17.4886,-6.05878}
f3[Range[0,10]]
{6.99554,-0.301645,-9.05752,-19.7581,-33.132,-50.3244,-73.2428,-105.319,-
153.414,-233.519,-393.529}
{f3[1,P→100],f3[1,T→500],f3[1,v→10.2],f3[1,P→90,T→490,v→11.1]}
{12.3984,-2.65857,-17.4886,-6.05878}

```

18. Cree dos funciones "f1" y "f2", con "Module" y "Block", que devuelvan el valor de la siguiente función, donde "l", "I", "E", "y", "A", son parámetros opcionales con valores por defecto iguales a: 120, 170.6, 3×10^6 , 0.066 y 32 respectivamente. La constante debe ser definida con "With". Luego calcule valores de las funciones para "p=10,20,30,...,90,100", "p=70, E=4x10⁶", "p=70, I=180", "p=10,20,30, l=124,127,130, y=0.056,0.06,0.07, A=30,35,40".

$$f(p) = \cos(kl) \cosh(kl) + 1$$

$$k^2 = p/a; a^2 = EIg/(Ay); g = 386$$

```

Remove[f1,f2]
Options[f1]={l→120,i→170.6,e→3*10^6,y→0.066,A→32};
f1[p_,OptionsPattern[]]:=Module[{k,a,
  l=OptionValue[l],
  i=OptionValue[i],
  e=OptionValue[e],
  y=OptionValue[y],
  A=OptionValue[A]},
  With[{g=386},
    a=Sqrt[e*i*g/(A*y)];

```

```

k=Sqrt[p/a];
Cos[k*1]*Cosh[k*1]+1]
]

```

Nota: Observe que se ha cambiado "I" por "i" y "E" por "e" debido a que "I" y "E" son empleados por Mathematica para el número imaginario el número de Nepper respectivamente.

```

Options[f2]={l→120,i→170.6,e→3*10^6,y→0.066,A→32};
f2[p_,OptionsPattern[]]:=Block[{k,a,
  l=OptionValue[l],
  i=OptionValue[i],
  e=OptionValue[e],
  y=OptionValue[y],
  A=OptionValue[A]},
With[{g=386},
  a=Sqrt[e*i*g/(A*y)];
  k=Sqrt[p/a];
  Cos[k*1]*Cosh[k*1]+1]
]
f1[Range[10,100,10]]
{1.96307,1.85252,1.66905,1.41383,1.08849,0.695111,0.236245,-0.285117,-
0.865538,-1.50115}
{f1[70,e→4*10^6],f1[70,i→180],f1[{10,20,30},{1→124,1→127,1→130},{y→0.056,y
→0.06,y→0.07},{A→30,A→35,A→40}]}
{0.66846,0.326051,{1.96651,1.86622,1.69972}}
f2[Range[10,100,10]]
{1.96307,1.85252,1.66905,1.41383,1.08849,0.695111,0.236245,-0.285117,-
0.865538,-1.50115}
{f1[70,e→4*10^6],f1[70,i→180],f1[{10,20,30},{1→124,1→127,1→130},{y→0.056,y
→0.06,y→0.07},{A→30,A→35,A→40}]}
{0.66846,0.326051,{1.96651,1.86622,1.69972}}

```

19. Cree tres funciones "f1", "f2" y "f3" con "Module", "Block" y "With", que devuelvan el valor de la siguiente función, donde "a", "b", "c", "d", "θ" son parámetros opcionales con valores por defecto iguales a: 1, 2, 2, 2, y 20° respectivamente. Luego calcule los valores de las funciones para "φ=90°" y "θ= 0.°, 10.°, 20.°, 30.°, ..., 150.°, 160.°".

$$f(\phi) = R1 \cos(\theta) - R2 \cos(\phi) + R3 - \cos(\theta - \phi)$$

$$R1 = d/c; \quad R2 = d/a; \quad R3 = \frac{d^2 + a^2 - b^2 + c^2}{2ca}$$

```

Clear[f1,f2,f3]
Options[f1]={a→1,b→2,c→2,d→2,θ→20 °};
f1[φ_,OptionsPattern[]]:=Module[{R1,R2,R3,
  a=OptionValue[a],
  b=OptionValue[b],
  c=OptionValue[c],
  d=OptionValue[d],
  θ=OptionValue[θ]},
R1=d/c;
R2=d/a;
R3=(d^2+a^2-b^2+c^2)/(2*c*a);
R1*Cos[θ]-R2*Cos[φ]+R3-Cos[θ-φ]
]
Options[f2]={a→1,b→2,c→2,d→2,θ→20 °};
f2[φ_,OptionsPattern[]]:=Block[{R1,R2,R3,
  a=OptionValue[a],
  b=OptionValue[b],

```



```

    c=OptionValue[c],
    d=OptionValue[d],
    θ=OptionValue[θ]},
    R1=d/c;
    R2=d/a;
    R3=(d^2+a^2-b^2+c^2)/(2*c*a);
    R1*Cos[θ]-R2*Cos[φ]+R3-Cos[θ-φ]
  ]
Options[f3]={a→1,b→2,c→2,d→2,θ→20 °};
f3[φ_,OptionsPattern[]]:=With[{
  R1=OptionValue[d]/OptionValue[c],
  R2=OptionValue[d]/OptionValue[a],
  R3=(OptionValue[d]^2+OptionValue[a]^2-OptionValue[b]^2+OptionValue[c]^2)
    /(2*OptionValue[c]*OptionValue[a]),
  θ=OptionValue[θ]},
  R1*Cos[θ]-R2*Cos[φ]+R3-Cos[θ-φ]
]
f1[90 °,θ→Range[0. °,160. °,10. °]]
{2.25,2.06116,1.84767,1.61603,1.37326,1.12674,0.883975,0.652328,0.43884,0.25
,0.0915441,-0.0317128,-0.116025,-0.158832,-0.158832,-0.116025,-0.0317128}
f2[90 °,θ→Range[0. °,160. °,10. °]]
{2.25,2.06116,1.84767,1.61603,1.37326,1.12674,0.883975,0.652328,0.43884,0.25
,0.0915441,-0.0317128,-0.116025,-0.158832,-0.158832,-0.116025,-0.0317128}
f3[90 °,θ→Range[0. °,160. °,10. °]]
{2.25,2.06116,1.84767,1.61603,1.37326,1.12674,0.883975,0.652328,0.43884,0.25
,0.0915441,-0.0317128,-0.116025,-0.158832,-0.158832,-0.116025,-0.0317128}
20. Cree dos funciones "f1" y "f2" con "Module" y "Block", que devuelvan el
valor de la siguiente función, donde "x", "y", son parámetros con valo-
res por defecto iguales a 0.6 y 0.4 respectivamente. Las constantes
"kx" y "ky" deben ser definidas con "With". Luego calcule los valores
de las funciones para "xi=0.5", "xi=0.5, x=0.7", "xi=0.5, x=0.55,
y=0.25".

```

$$f(x_i, x, y) = \frac{k_x}{k_y} + \frac{y - y_i}{x - x_i}$$

$$y_i = -0.0405 + 1.03785714286x_i$$

$$k_x = \frac{k'_x}{(1-x)_{im}}; \quad k_y = \frac{k'_y}{(1-y)_{im}}$$

$$(1-x)_{im} = \frac{(1-x) - (1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)}$$

$$(1-y)_{im} = \frac{(1-y_i) - (1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

$$k'_x = 0.001967; \quad k'_y = 0.001465$$

```

f1[xi_, x_ : 0.6, y_ : 0.3] := Module[{yi, kx, ky, xim, yim},
  With[{kpx=0.001967, kpy=0.001465},
    yi=-0.0405+1.03785714286*xi;
    xim=((1-x)-(1-xi))/Log[(1-x)/(1-xi)];
    yim=((1-yi)-(1-y))/Log[(1-yi)/(1-y)];
    kx=kpx/xim;
    ky=kpy/yim;
    kx/ky-(y-yi)/(x-xi)
  ]

```

```

]
]
f2[xi_,x_: 0.6,y_: 0.3]:=Block[{yi,kx,ky,xim,yim},
  With[{kpx=0.001967,kpy=0.001465},
    yi=-0.0405+1.03785714286*xi;
    xim=((1-x)-(1-xi))/Log[(1-x)/(1-xi)];
    yim=((1-yi)-(1-y))/Log[(1-yi)/(1-y)];
    kx=kpx/xim;
    ky=kpy/yim;
    kx/ky-(y-yi)/(x-xi)
  ]
]
{f1[0.5],f1[0.5,0.7],f1[0.5,0.55,0.25]}
{3.60115,2.97175,6.34786}
{f2[0.5],f2[0.5,0.7],f2[0.5,0.55,0.25]}
{3.60115,2.97175,6.34786}

```

21. Cree dos funciones "f1" y "f2" con "Module" y "Block", que devuelvan el valor de la siguiente función, donde "T₁", "H₁", "P", "HP₂" son parámetros opcionales con valores por defecto iguales a 20, 0.02, 540 y 80 respectivamente. Luego calcule los valores de las funciones para "T₂=550,560,...,590,600", "T₂=600, T₁=520", "T₂=600, HP₂=90", "T₂=600, P=600", "T₂=600, T₁=550, H₁=0.015, P=620, HP₂=90".

$$f(T_2) = \frac{H_2 - H_1}{T_2 - T_1} + \frac{cs_1}{\lambda_2}$$

$$H_2 = \frac{HP_2 * HS_2}{100}$$

$$HS_2 = \frac{18.02}{28.97} * \frac{PV}{P - PV}$$

$$PV = e^{\left(18.3036 - \frac{3816.44}{T_2 + 227.02}\right)}$$

$$cs_1 = 1.005 + 1.88 * H_1$$

$$\lambda_2 = 2501.4 * \left(\frac{1 - \frac{T_2}{647.3}}{1 - 0.422} \right)^{0.375}$$

```

Clear[f1,f2]
Options[f1]={T1->20,H1->0.02,P->540,HP2->80};
f1[T2_,OptionsPattern[]]:=Module[{H2,HS2,PV,cs1,λ2,
  T1=OptionValue[T1],
  H1=OptionValue[H1],
  P=OptionValue[P],
  HP2=OptionValue[HP2]
},
  cs1=1.005+1.88*H1;
  λ2=2501.4*((1-T2/647.3)/(1-0.422))^0.375;
  PV=Exp[18.3036-3816.44/(T2+227.02)];
  HS2=18.02/28.97*PV/(P-PV);
  H2=HP2*HS2/100;
  (H2-H1)/(T2-T1)+cs1/λ2
]
Options[f2]={T1->20,H1->0.02,P->540,HP2->80};
f2[T2_,OptionsPattern[]]:=Block[{H2,HS2,PV,cs1,λ2,
  T1=OptionValue[T1],

```

```

H1=OptionValue[H1],
P=OptionValue[P],
HP2=OptionValue[HP2]
},
cs1=1.005+1.88*H1;
λ2=2501.4*((1-T2/647.3)/(1-0.422))^0.375;
PV=Exp[18.3036-3816.44/(T2+227.02)];
HS2=18.02/28.97*PV/(P-PV);
H2=HP2*HS2/100;
(H2-H1)/(T2-T1)+cs1/λ2
]
{f1[Range[550,600,10]],f1[600,T1→520],f1[600,HP2→90],f1[600,P→600],f1[600,T
1→550,H1→0.015,P→620,HP2→90]}
{{-0.000286729,-0.000239914,-0.000188854,-0.00013185,-
0.0000662769,0.0000122383},-0.00556883,-0.0000950728,0.0000121798,-
0.0106072}
{f2[Range[550,600,10]],f2[600,T1→520],f2[600,HP2→90],f2[600,P→600],f2[600,T
1→550,H1→0.015,P→620,HP2→90]}
{{-0.000286729,-0.000239914,-0.000188854,-0.00013185,-
0.0000662769,0.0000122383},-0.00556883,-0.0000950728,0.0000121798,-
0.0106072}

```

4.3.2. Ejercicios

17. Cree tres funciones: "f1", "f2" y "f3" que empleando "Module", "Block" y "With", devuelvan el valor de la siguiente función. Luego calcule los valores de las funciones para: "x=-5,-3,-1,...,7,10".

$$f(x) = e^{\frac{x+y}{5}} + x^2 y^2 - 70y - 15$$

$$y = (115 + 4x - 2x^2)/5x$$

18. Cree dos funciones: "f1" y "f2", que empleando "Module" y "Block", devuelvan el valor de la siguiente función. Luego calcule los valores de las funciones para: "x=0.1, 0.3, 0.5,...,4.8, 5.1".

$$f(x) = x + w + y - 39$$

$$z = \sqrt{35 - x^{1/2}}$$

$$y = 30 + \frac{8}{z} - e^{z/2}$$

$$w = 31 - z^2 + 2z$$

19. Cree tres funciones "f1", "f2" y "f3" que empleando "Module", "Block" y "With" devuelvan el valor de la siguiente función, donde " θ_c ", " θ_k ", " T_c ", " t " e " I " son parámetros opcionales con valores por defecto iguales a 2.97, 3.27, 0.625, 0.0191 y 0.0006 respectivamente. Luego calcule valores de las funciones para "k=0,1,2,..10", "k=3.2, $\theta_c=2.8$, $\theta_k=3.1$ ", "k=3.2, $T_c=0.721$, $t=0.0093$, $I_0=0.00067$ ", "k=3.3, $\theta_c=2.9$, $\theta_k=3.05$, $T_c=0.621$, $t=0.01$, $I_0=0.00059$ "

$$f(k) = \theta_c - \frac{T_0}{k} \left(1 - \cos \left(\sqrt{\frac{k}{I_0}} t \right) \right)$$

$$T_0 = T_c + \theta_k k$$

20. Cree tres funciones "f1", "f2" y "f3" que empleando "Module", "Block" y "With" devuelvan el valor de la siguiente función, donde " L_0 ", " x_0 ", " V_0 " y " y_0 " son parámetros opcionales con valores por defecto iguales a

300, 0, 100 y 0.2 respectivamente. Luego calcule valores de las funciones para " $x_1=0.1, 0.15, 0.2, \dots, 0.9, 0.95$ ", " $x_1=0.5$, $L_0=320$ ", " $x_1=0.5$, $V_0=80$ ", " $x_1=0.5$, $x_0=0.1$, $y_0=0.3$ ", " $x_1=0.5$, $L_0=310$, $x_0=0.05$, $V_0=120$, $y_0=0.35$ ".

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0$$

$$L_1 = \frac{L_c}{1 - x_1}$$

$$V_1 = M - L_1$$

$$y_1 = 1420 * x_1$$

$$L_c = L_0 * (1 - x_0)$$

$$M = L_0 + V_0$$

$$C_0 = L_0 * x_0 + V_0 * y_0$$

21. Cree dos funciones "f1" y "f2" que empleando "Module" y "Block", devuelvan el valor de la siguiente función, donde " X_{A_n} ", " M ", " CA_0 " y " CB_0 " son parámetros opcionales con valores por defecto iguales a 0.12, 32.1, 20.2 y 3.42 respectivamente. Luego calcule valores de las funciones para " $YA_1=0.1, 0.2, \dots, 0.9$ ", " $YA_1=0.3$, $M=40.2$ ", " $YA_1=0.35$, $CA_0=24.3$ ", " $YA_1=0.32$, $CA_0=25.3$, $CB_0=4.32$ ", " $YA_1=0.33$, $M=38.2$, $CA_0=25.3$, $CB_0=3.56$ ".

$$f(Y_{A1}) = L_N + V_1 - M$$

$$Y_{B1} = 2Y_{A1} + Y_{A1}^2 + 3.44Y_{A1}^3$$

$$X_{BN} = 2.2 + 5.1X_{AN} - 7.2X_{AN}^{2.2}$$

$$L_N = \frac{C_{A0}Y_{B1} - C_{B0}Y_{A1}}{X_{AN}Y_{B1} - X_{BN}Y_{A1}}$$

$$V_1 = \frac{C_{A0} - L_N X_{AN}}{Y_{A1}}$$

5. ESTRUCTURAS SELECTIVAS

5.1. Selección

Para implementar las estructuras selectivas, Mathematica cuenta también con varias alternativas. Como en todos los lenguajes las condiciones se forman con operadores relacionales y operadores lógicos, siendo el resultado de las operaciones falso "False" o verdadero "True".

Los operadores relacionales de uso más frecuente son:

Igual:	== (Equal)
Diferente:	!= (Unequal)
Idéntico:	=== (SameQ)
Menor:	< (Less)
Mayor:	> (Greater)
Menor o igual:	<= (LessEqual)
Mayor o igual:	>= (GreaterEqual)

Los operadores lógicos más usuales son:

Negación:	! (Not)
Y:	&& (And)
O:	(Or)
O excluyente:	Xor

La estructura selectiva por excelencia es "If", que como recordarán en Pascal tiene la forma: *"if condición then instrucciones_1 else instrucciones_2"*, donde si la "condición" es verdadera se evalúan las "instrucciones_1" y si es falsa las "instrucciones_2". En Mathematica la estructura "If" tiene la siguiente forma:

```
If[condición, instrucciones_1, instrucciones_2]
```

Y al igual que en Pascal, si la "condición" es verdadera se evalúan las "instrucciones_1" y si es falsa las "instrucciones_2". Como sucede en Pascal, las "instrucciones_2" son opcionales, es decir una estructura "If" puede ser escrita de la siguiente forma:

```
If[condición, instrucciones_1]
```

En este caso si la "condición" es falsa, "If" no hace nada y el programa continua con la siguiente instrucción (si es que existe).

Como sucede con las estructuras secuenciales ("Module", "Block" y "With") la estructura "If" puede anidarse a cualquier nivel, es decir una estructura "If" puede contener otra estructura "If", esta a su vez otra estructura "If" y así sucesivamente hasta el nivel que se requiera. Sin embargo, y al igual que en las estructuras secuenciales, se recomienda no anidar estructuras "If" más allá de un tercer nivel.

En la siguiente instrucción se emplea "If" para crear una función que devuelve "positivo" si el número es positivo y "negativo" en caso contrario:

```
posneg[x_] := If[x > 0, Print["Positivo"], Print["Negativo"]]
```

Ejecutando "posneg" con un número positivo, por ejemplo "x=2", resulta:

```
posneg[2]
Positivo
```

Y con un número negativo, por ejemplo x=-45, obtenemos:

```
posneg[-45]
```

Negativo

Sin embargo, si ejecutamos "posneg" con "x=0" obtenemos:

```
posneg[0]
Negativo
```

Lo que es erróneo, pues "0" no es ni positivo ni negativo. Podemos solucionar este problema empleando una segunda estructura "If" anidada:

```
posneg[x_] := If[x > 0, Print["Positivo"], If[x == 0, Print["Cero"], Print["Negativo"]]]
```

Que sigue devolviendo resultados correctos para "x=2" y "x=-45" y que devuelve el siguiente resultado para "x=0":

```
posneg[0]
Cero
```

Que ahora si es correcto, sin embargo, aún en este sencillo ejemplo se puede observar que cuando se emplean estructuras anidadas el código se hace menos entendible. Por ello Mathematica cuenta con otras estructuras selectivas que, en ocasiones, permiten resolver problemas sin recurrir al uso de estructuras anidadas, manteniendo así el código claro. Una de dichas estructuras es "Which", que tiene la siguiente estructura:

```
Which[condición_1, instrucciones_1, condición_2, instrucciones_2, ...]
```

En esta estructura se prueba la "condición_1" y si la misma es verdadera se evalúan las "instrucciones_1" y la ejecución de "Which" termina, caso contrario se prueba la "condición_2" y si la misma es verdadera se evalúan las "instrucciones_2" y la ejecución de "Which" termina, continuando de esta forma hasta que alguna de las condiciones es verdadera o hasta que no existen más condiciones, en cuyo caso "Which" devuelve "Null".

Para evitar que "Which" devuelva el valor nulo "Null", es decir para que "Which" ejecute unas instrucciones por defecto, la última condición debe ser simplemente "True", pues al ser verdadera asegura la ejecución de las instrucciones que se escriben a continuación de la misma:

```
Which[condición_1, instrucciones_1, ..., True, instrucciones_por_defecto]
```

Empleando "Which" podemos volver a crear la función "posneg":

```
posneg[x_] := Which[x > 0, Print["Positivo"], x < 0, Print["Negativo"], True, Print["Cero"]]
```

Como se puede observar, este código es más claro que el escrito con "If" anidados. Probando la función para los tres casos, obtenemos los resultados correctos:

```
posneg[50]
Positivo
posneg[-67]
Negativo
posneg[0]
Cero
```

En consecuencia resulta conveniente emplear la estructura "Which" cuando existen dos o más condiciones consecutivas, no obstante es importante verificar que dichas condiciones no se superpongan, es decir dos o más condiciones no deben ser verdaderas al mismo tiempo.

Otra alternativa a "Which", que resulta más adecuada cuando se compara el resultado de una expresión con valores determinados, es "Switch" la cual tiene una lógica similar a la instrucción "Case" de Pascal:

```
Switch[expresión, valor_1, instrucciones_1, valor_2, instrucciones_2,... ]
```

"Switch" opera de modo similar a "Case" de Pascal: evalúa la "expresión" y luego compara el resultado de esta expresión con el "valor_1", si son iguales evalúa las "instrucciones_1" y la estructura "Switch" termina, caso contrario compara el resultado de la expresión con el "valor_2", si son iguales evalúa las "instrucciones_2" y la estructura "Switch" termina, continuando de esta manera hasta que uno de los valores es igual al resultado o hasta que ya no existen más valores para comparar, en cuyo caso "Switch" queda tal como fue escrita (como una expresión sin evaluar). Para evitar que "Switch" quede como una expresión sin evaluar y ejecute algunas instrucciones por defecto, se puede emplear como último valor de comparación el patrón "_", que como sabemos representa a cualquier valor o expresión, por lo tanto siempre será igual a cualquier resultado:

```
Switch[expresión, valor_1,...,_,Instrucciones_por_defecto]
```

En el siguiente ejemplo se crea una función que devuelve el día de la semana equivalente al número introducido:

```
dia[x_Integer]:=Switch[x,1,Print["Domingo"],2,Print["Lunes"],3,Print["Martes"],4,Print["Miércoles"],5,Print["Jueves"],6,Print["Viernes"],7,Print["Sábado"],_,Print["No es un día de la semana"]]
```

Llamando a esta función con 3 y 7 obtenemos los días respectivos:

```
dia[3]
Martes
dia[7]
Sábado
```

Y si llamamos a la función con un número superior a 7 (o inferior a 1) obtenemos:

```
dia[10]
No es un día de la semana
dia[-3]
No es un día de la semana
```

Puesto que en el parámetro de la función se ha especificado que sea un número entero, si llamamos a la función con un número real (o cualquier otro tipo de dato), la función queda como una expresión sin evaluar:

```
dia[4.5]
dia[4.5]
```

Para comparar un resultado con uno de dos o más valores se pueden emplear patrones con alternativas (|), por ejemplo, en la siguiente función se compara el valor del parámetro "x" con los números "1", "3", "5", y "7" y si es uno de dichos números imprime "está en el conjunto impar", si los números son "2", "4", "6" u "8" imprime "está en el conjunto par" y si no es ninguno de esos números imprime "no está en ninguno de los conjuntos":

```
fp[x_]:=Switch[x,1|3|5|7,Print["Está en el conjunto impar"],2|4|6|8, Print["Está en el conjunto par"],_,Print["No está en ninguno de los conjuntos"]]
```

Llamando a la función con "5", "4" y "10" se obtiene:

```
fp[5]
Está en el conjunto impar
fp[4]
Está en el conjunto par
fp[10]
No está en ninguno de los conjuntos
```

Cuando para representar todo el rango de posibles valores de una función se requieren dos o más expresiones, es más eficiente emplear la estructura "Piecewise". El formato de "Piecewise" es el siguiente:

```
Piecewise[{{instrucciones_1,condición_1}, {instrucciones_2,condición_2},
          ...},instrucciones por defecto]
```

La forma en que funciona "Piecewise" es similar a la de "Which", pero en "Piecewise" por lo general las condiciones son desigualdades y como se puede ver las condiciones se encuentran después de las instrucciones asociadas a las mismas. "Piecewise" evalúa la "condición_1" y si es verdadera evalúa las "instrucciones_1" y la estructura "Piecewise" termina, caso contrario evalúa la "condición_2" y si es verdadera evalúa las "instrucciones_2" y la estructura "Piecewise" termina, continuando de esa manera hasta que una de las condiciones es verdadera. Si ninguna de las condiciones es verdadera "Piecewise" evalúa las instrucciones por defecto y si dichas instrucciones no existen devuelve 0.

Así en el siguiente ejemplo se crea una función que evalúa la expresión " x^2+2x-3 ", si el valor de " x " está comprendido entre -10 y 9.9999..., evalúa la expresión " x^3-3x^2 " si el valor de " x " está comprendido entre 10 y 19.999..., evalúa la expresión " x^3+x-5 ", si el valor de " x " está comprendido entre 20 y 49.999... y evalúa la expresión " x^2+2 " si el valor de " x " no está comprendido en ninguno de los intervalos anteriores:

```
Remove[f]
f[x_]:=Piecewise[{{x^2+2*x-3,-10<=x<10},{x^3-3*x^2,10<=x<20},{x^3+x-5,20<=
x<50}},x^2+2]
```

Podemos ver la función en su forma tradicional pidiendo información con relación a la misma:

```
?f
Global`f
f[x_] := {
  x^2 + 2 x - 3  -10 <= x < 10
  x^3 - 3 x^2    10 <= x < 20
  x^3 + x - 5    20 <= x < 50
  x^2 + 2        True
}
```

Por supuesto es posible crear la función escribiéndola directamente en su forma tradicional (empleando atajos, nombres y/o secuencias de escape como vimos anteriormente).

Ahora probamos la función para los cuatro posibles casos:

```
f[5]
32
f[15]
2700
f[30]
27025
f[70]
4902
```

Si una misma expresión debe ser evaluada cuando se cumplen dos o más condiciones, entonces en lugar de escribir la misma expresión dos o más veces se puede emplear el operador lógico O : "||". Por ejemplo en la siguiente función la expresión " $2+3x+x^2$ " se evalúa cuando el valor de " x " está comprendido entre "1 y 5", cuando está comprendido entre "20" y "25" o cuando está comprendido entre "30" y "35":

```
Remove[g]
```



```
g[x_]:=Piecewise[{{2+3*x+x^2,1<=x<=5 || 20<=x<=25 || 30<=x<=35},{1+2*x+3*x^3,
x>50},{2+4*x+5*x^2,x<-50}},x+2*x^2+3*x^3]
```

Llamando a la función con "x=3", "x=22" y "x=34" se evalúa la expresión "2+3x+x²":

```
g[3]
20
g[22]
552
g[34]
1260
```

Por supuesto los otros casos se evalúan también correctamente:

```
g[60]
648121
g[-55]
14907
g[7]
1134
```

Finalmente veremos la instrucción "Condition" (/;), que ya vimos al estudiar "patrones" y que puede ser aplicada también a otro tipo de expresiones:

expresion /; condición

Donde "expresión" puede ser un patrón (como ya vimos previamente), una regla retardada (:>) o una asignación retardada (:=), las cuales se ejecutan sólo si la "condición" es verdadera. Por ejemplo en la siguiente instrucción se crea una función para calcular la raíz cúbica de un número, pero dicha función se evalúa sólo si el número es positivo:

```
rcubica[x_]:=x^(1/3)/;x>0
```

Si ejecutamos la función con "x=27", obtenemos la raíz cúbica del número:

```
rcubica[27]
3
```

Pero si ejecutamos la función con "x=-27", la función queda sin evaluar:

```
rcubica[-27]
rcubica[-27]
```

5.1.1. Ejemplos

1. Elabore cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y "/;", que reciban dos parámetros y, si los mismos son números, devuelva 1 si el primer número es mayor al segundo, -1 si el primer número es menor al segundo y 0 si son iguales. Si los parámetros no son números las funciones debe devolver "?".

```
Remove[f1,f2,f3,f4,f5]
f1[x_,y_]:=If[NumberQ[x] && NumberQ[y],
  If[x>y,1,If[x<y,-1,0]],
  "?"]
f1[3,1]
1
f1[2,5]
-1
f1[3,3]
0
f1[A,B]
```

?

```
f2[x_,y_]:=Which[NumberQ[x] && NumberQ[y],
  Which[x>y,1,x<y,-1,True,0],
  True,"?"]
```

f2[3,1]

1

f2[2,5]

-1

f2[3,3]

0

f2[A,B]

?

```
f3[x_,y_]:=Switch[NumberQ[x] && NumberQ[y],
  True,Switch[x>y,True,1,
    False,Switch[x<y,True,-1,False,0]],
  False,"?"]
```

f3[3,1]

1

f3[2,5]

-1

f3[3,3]

0

f3[A,B]

?

```
f4[x_,y_]:=Piecewise[
  {{Piecewise[{{1,x>y},{-1,x<y}},0],
    NumberQ[x] && NumberQ[y]}},
  "?"]
```

f4[3,1]

1

f4[2,5]

-1

f4[3,3]

0

f4[A,B]

?

Para resolver el problema con Condition: `"/;"`, se crean en realidad dos funciones con el mismo nombre, la primera se ejecuta cuando los parámetros son numéricos y la segunda en cualquier otro caso:

```
f5[x_,y_]:=(x/.{_/;x>y→1,_/;x<y→-1,_/;x==y→0})/;NumberQ[x] && NumberQ[y]
```

f5[x_,y_]:= "?"

f5[3,1]

1

f5[2,5]

-1

f5[3,3]

0

f5[A,B]

?

2. Elabore cinco funciones con las instrucciones `"If"`, `"Which"`, `"Switch"`, `"Piecewise"` y `"/;"`, que reciban tres parámetros y si son números enteros, calcule la suma de los mismos y en función al resultado devuelva "menor a 100", "mayor o igual a 100 y menor a 500", "mayor o igual a

500 y menor a 10000" y "Mayor a 10000". Si los parámetros no son números enteros debe devolver "No son números enteros".

```

Remove[f1,f2,f3,f4,f5]
f1[x_,y_,z_]:=Module[{s},
  If[IntegerQ[x] && IntegerQ[y] && IntegerQ[z],
    s=x+y+z;
    If[s<100,"Menor a 100",
      If[100<=s<500,"Mayor o igual a 100 y menor a 500",
        If[500<=s<10000,"Mayor o igual a 500 y menor a 10000",
          "Mayor a 10000"]]], "No son números enteros"]]
f1[1,2,3]
Menor a 100
f1[50,60,70]
Mayor o igual a 100 y menor a 500
f1[100,200,300]
Mayor o igual a 500 y menor a 10000
f1[5000,6000,7000]
Mayor a 10000
f1[1.,2.,3.]
No son números enteros
f1["a","b","c"]
No son números enteros

f2[x_,y_,z_]:=Block[{s},
  Which[IntegerQ[x] && IntegerQ[y] && IntegerQ[z],
    s=x+y+z;
    Which[s<100,"Menor a 100",
      100<=s<500,"Mayor o igual a 100 y menor a 500",
      500<=s<10000,"Mayor o igual a 500 y menor a 10000",
      True,"Mayor a 10000"],
    True,"No son números enteros"]]
f2[1,2,3]
Menor a 100
f2[50,60,70]
Mayor o igual a 100 y menor a 500
f2[100,200,300]
Mayor o igual a 500 y menor a 10000
f2[5000,6000,7000]
Mayor a 10000
f2[1.,2.,3.]
No son números enteros
f2["a","b","c"]
No son números enteros

f3[x_,y_,z_]:=Module[{s},
  Switch[IntegerQ[x] && IntegerQ[y] && IntegerQ[z],
    True,s=x+y+z;
    Switch[s<100,True,"Menor a 100",
      False,Switch[100<=s<500,True,"Mayor o igual a 100 y menor a 500",
        False,Switch[500<=s<10000,True,"Mayor o igual a 500 y menor a 10000",
          False,"Mayor a 10000"]]],
    False,"No son números enteros"]]
f3[1,2,3]
Menor a 100
f1[50,60,70]
Mayor o igual a 100 y menor a 500
f1[100,200,300]
Mayor o igual a 500 y menor a 10000

```

```

f1[5000,6000,7000]
Mayor a 10000
f1[1.,2.,3.]
No son números enteros
f1["a","b","c"]
No son números enteros

f4[x_,y_,z_]:=Block[{s},
  Piecewise[{{
    s=x+y+z;
    Piecewise[{{"Menor a 100",s<100},
      {"Mayor o igual a 100 y menor a 500",100<=s<500},
      {"Mayor o igual a 500 y menor a 10000",500<=s<10000}},
    "Mayor a 10000"},
    NumberQ[x] && NumberQ[y] && NumberQ[z]}},
  "No son números enteros"]
]
f4[1,2,3]
Menor a 100
f1[50,60,70]
Mayor o igual a 100 y menor a 500
f1[100,200,300]
Mayor o igual a 500 y menor a 10000
f1[5000,6000,7000]
Mayor a 10000
f1[1.,2.,3.]
No son números enteros
f1["a","b","c"]
No son números enteros

f5[x_,y_,z_]:=Module[{s,r},
  s=x+y+z;
  r/.{_/;s<100->"Menor a 100",
    _/;100<=s<500->"Mayor o igual a 100 y menor a 500",
    _/;500<=s<10000->"Mayor o igual a 500 y menor a 10000",
    _/;s>10000->"Mayor a 10000"}
  ]/;IntegerQ[x] && IntegerQ[y]&&IntegerQ[z]
f5[x_,y_,z_]:="No son números enteros"
f5[1,2,3]
Menor a 100
f5[50,60,70]
Mayor o igual a 100 y menor a 500
f5[100,200,300]
Mayor o igual a 500 y menor a 10000
f5[5000,6000,7000]
Mayor a 10000
f5[1.,2.,3.]
No son números enteros
f5["a","b","c"]
No son números enteros

3.  Elabore cinco funciones, con las instrucciones "If", "Which", "Switch",
    "Piecewise" y "/", para encontrar las soluciones de la ecuación cua-
    drática:  $ax^2+bx+c=0$ .

Remove[f1,f2,f3,f4,f5]
f1[a_,b_,c_]:=Module[{d,r1,r2,im=0},
  If[NumberQ[a] && NumberQ[b]&&NumberQ[c],
    d=b^2-4*a*c;

```

```

    If[d>0,r1=(-b+Sqrt[d])/2/a;r2=(-b-Sqrt[d])/2/a,
    If[d<0,r1=r2=-b/2/a;im=Sqrt[Abs[d]]/2/a,
    r1=r2=-b/2/a]];
    {{r1+im I},{r2-im I}}//N,
    "Los coeficientes deben ser numéricos"]]
f1[1,-5,6]
{{3.},{2.}}
f1[1,-12,36]
{{6.},{6.}}
f1[1,-6,25]
{{3. +4. I},{3. -4. I}}
f1["a","b","c"]
Los coeficientes deben ser numéricos

```

```

f2[a_,b_,c_]:=Block[{d,r1,r2,im=0},
  Which[NumberQ[a] && NumberQ[b] && NumberQ[c],
    d=b^2-4*a*c;
    Which[d>0,r1=(-b+Sqrt[d])/(2*a);r2=(-b-Sqrt[d])/(2*a),
    d<0,r1=r2=-b/(2*a);im=Sqrt[Abs[d]]/(2*a),
    True,r1=r2=-b/(2*a)];
    {{r1+im I},{r2-im I}}//N,
    True,"Los coeficientes deben ser numéricos"]]
f2[1,-5,6]
{{3.},{2.}}
f2[1,-12,36]
{{6.},{6.}}
f2[1,-6,25]
{{3. +4. I},{3. -4. I}}
f2["a","b","c"]
Los coeficientes deben ser numéricos

```

Para resolver el problema con "Switch" empleamos en este caso la función "Boole" que convierte en 1 el valor logico verdadero (True) y en 0 el valor lógico falso (False), así "Boole[3>0]+Boole[3<0]*2" devuelve "1", mientras que "Boole[-2>0]+Boole[-2<0]*2" devuelve "2".

```

f3[a_,b_,c_]:=Block[{d,r1,r2,im=0},
  Switch[NumberQ[a] && NumberQ[b] && NumberQ[c],
    True,d=b^2-4*a*c;
    Switch[Boole[d>0]+Boole[d<0]*2,
      1,r1=(-b+Sqrt[d])/(2*a);r2=(-b-Sqrt[d])/(2*a),
      2,r1=r2=-b/(2*a);im=Sqrt[Abs[d]]/(2*a),
      _,r1=r2=-b/(2*a)];
    {{r1+im I},{r1-im I}}//N,
    False,"Los coeficientes deben ser numéricos"]]
f3[1,-5,6]
{{3.},{3.}}
f3[1,-12,36]
{{6.},{6.}}
f3[1,-6,25]
{{3. +4. I},{3. -4. I}}
f3["a","b","c"]
Los coeficientes deben ser numéricos

```

```

f4[a_,b_,c_]:=Module[{d,r1,r2,im=0},
  Piecewise[{{d=b^2-4*a*c;
    Piecewise[{{r1=(-b+Sqrt[d])/2/a;r2=(-b-Sqrt[d])/2/a,d>0},
    {r1=r2=-b/2/a;im=Sqrt[Abs[d]]/2/a,d<0}},

```

```

      r1=r2=-b/2/a];
      {{r1+im I},{r1-im I}}/N,
      NumberQ[a] && NumberQ[b] && NumberQ[c]}}},
      "Los coeficientes deben ser numéricos"]
f4[1,-5,6]
{{3.},{3.}}
f4[1,-12,36]
{{6.},{6.}}
f4[1,-6,25]
{{3. +4. I},{3. -4. I}}
f4["a","b","c"]
Los coeficientes deben ser numéricos

f5[a_,b_,c_]:=Block[{d,r1,r2,im=0},
  d=b^2-4*a*c;
  {r1,r2,im}={r1,r2,im}/.{
    r1/d>0->(-b+Sqrt[d])/(2*a),r2/d>0->(-b-Sqrt[d])/(2*a),
    r1|r2/d<=0->-b/(2*a),im/d<0->Sqrt[Abs[d]]/(2*a)};
  {{r1+im I},{r1-im I}}
  ]/;NumberQ[a] && NumberQ[b] && NumberQ[c]
f5[a_,b_,c_]:= "Los coeficientes deben ser numéricos"
f5[1,-5,6]
{{3},{3}}
f5[1,-12,36]
{{6},{6}}
f5[1,-6,25]
{{3+4 I},{3-4 I}}
f5["a","b","c"]
Los coeficientes deben ser numéricos

```

4. Elabore cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y "/", para determinar si un año es o no bisiesto. Un año es bisiesto si es divisible entre 4, sin embargo, si termina en 2 ceros (como 1900) deben eliminarse los dos ceros antes de llevar a cabo la división. La función debe devolver "True" si el año es bisiesto y "False" en caso contrario.

Como se puede observar en los siguientes ejemplos, en todas las funciones se declara una variable local "a", que inicialmente es igual al parámetro "anio", porque si "anio" es divisible entre 100, debe ser reemplazado por su división entre 100 y en "Mathematica" no se puede cambiar el valor de los parámetros.

```

Remove[f1,f2,f3,f4,f5]
f1[anio_]:=Block[{a=anio},
  If[IntegerQ[a],
    If[Divisible[a,100],a=a/100];
    Divisible[a,4],
    "El parámetro debe ser un número entero"]]
f1[2008]
True
f1[2007]
False
f1[2000]
True
f1[1900]
False

f2[anio_]:=Module[{a=anio},

```

```

Which[IntegerQ[a],
  Which[Divisible[a,100],a=a/100];
  Divisible[a,4],
  True,"El parámetro debe ser un número entero"]]
f2[2008]
True
f2[2007]
False
f2[2000]
True
f2[1900]
False

f3[anio_]:=Module[{a=anio},
  Switch[IntegerQ[a],
    True,a=Switch[Divisible[a,100],True,a/100,False,a];
    Divisible[a,4],
    False,"El parámetro debe ser un número entero"]]
f3[2008]
True
f3[2007]
False
f3[2000]
True
f3[1900]
False

f4[anio_]:=Block[{a=anio},
  Piecewise[{{a=Piecewise[{{a/100,Divisible[a,100]}}],a};
    Divisible[a,4],
    IntegerQ[a]}],
  "El parámetro debe ser un número entero"]]
f4[2008]
True
f4[2007]
False
f4[2000]
True
f4[1900]
False

f5[anio_]:=Block[{a=anio},
  a/.a/;Divisible[a,100]->a/100;
  Divisible[a,4]]/;IntegerQ[anio]
f5[anio_]:= "El parámetro debe ser un número entero"
f5[2008]
True
f4[2007]
False
f4[2000]
True
f4[1900]
False

```

5. Elabore cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y "/", que reciba un número y devuelva su raíz cuadrada si el número es impar y positivo, la raíz cúbica del número si el número es par y positivo, el cuadrado del número si es impar y negativo y el número sin modificar en cualquier otro caso.

```

Remove[f1,f2,f3,f4,f5]
f1[n_]:=If[IntegerQ[n],
  If[OddQ[n] && n>0,Sqrt[n],
    If[EvenQ[n] && n>0,n^(1/3),
      If[OddQ[n] && n<0,n^2,n]]],
  "El parámetro debe ser un número entero"]
f1[9]
3
f1[64]
4
f1[-3]
9
f1[-4]
-4
f1[a]
El parámetro debe ser un número entero

f2[n_]:=Which[IntegerQ[n],
  Which[OddQ[n] && n>0,Sqrt[n ],
    EvenQ[n] && n>0,n^(1/3),
    OddQ[n] && n<0,n^2,
    True,n],
  True,"El parámetro debe ser un número entero"]
f2[9]
3
f2[64]
4
f2[-3]
9
f2[-4]
-4
f2[a]
El parámetro debe ser un número entero

f3[n_]:=Switch[IntegerQ[n],
  True,Switch[Boole[OddQ[n] && n>0]+Boole[EvenQ[n] && n>0]*2
    +Boole[OddQ[n] && n<0]*3,
    1,Sqrt[n],
    2,n^(1/3),
    3,n^2,
    _,n],
  False,"El parámetro debe ser un número entero"]
f3[9]
3
f3[64]
4
f3[-3]
9
f3[-4]
-4
f3[a]
El parámetro debe ser un número entero

f4[n_]:=Piecewise[{{
  Piecewise[{{Sqrt[n],OddQ[n] && n>0},
    {n^(1/3),EvenQ[n] && n>0},
    {n^2,OddQ[n] && n<0}},
  n],

```



```

IntegerQ[n]}}},
"El parámetro debe ser un número entero"]
f4[9]
3
f4[64]
4
f4[-3]
9
f4[-4]
-4
f4[a]
El parámetro debe ser un número entero

f5[n_]:= (r/.{
  r;/;OddQ[n] && n>0->Sqrt[n],
  r;/;EvenQ[n] && n>0->n^(1/3),
  r;/;OddQ[n] && n<0->n^2,
  r->n})/;IntegerQ[n]
f5[n_]:= "El parámetro debe ser un número entero"
f5[9]
3
f5[64]
4
f5[-3]
9
f5[-4]
-4
f5[a]
El parámetro debe ser un número entero

```

5.1.2. Ejercicios

1. Escriba cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y ";/;", que reciban un número entero comprendido entre 1 y 10 y devuelva su equivalente en números romanos.
2. Escriba cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y ";/;", para determinar si un número entero es par, impar o cero.
3. Escriba cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y ";/;", que reciban un número y devuelva el cubo del mismo si está comprendido entre 0 y 10, el cuadrado si es mayor a 10 y menor o igual a 50, la raíz cuadrada si es mayor a 50 y menor o igual a 100 y el número sin modificar en cualquier otro caso.
4. Elabore cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y ";/;", que reciban tres lados y determine si los mismos conforman o no un triángulo. Tres lados conforman un triángulo si la suma de cualesquier dos lados es siempre mayor al tercero. El módulo debe devolver verdadero "True" si los lados conforma un triángulo y Falso "False" en caso contrario.
5. Elabore cinco funciones, con las instrucciones "If", "Which", "Switch", "Piecewise" y ";/;", que devuelvan el resultado de la expresión: $x^{1/3}+3x-\ln(x)$, si el valor de "x" es positivo, devuelva el resultado de la expresión x^2+3x , si es negativo y 4.75 si es cero.

6. ESTRUCTURAS ITERATIVAS

6.1. Iteración

Para implementar las estructuras iterativas (repetitivas) Mathematica nos proporciona igualmente varias alternativas. Comenzaremos el estudio de las mismas con la estructura "Do", que tiene el siguiente formato:

```
Do[instrucciones, {vc, vi, vf, inc}]
```

Esta estructura es similar a la estructura "for" de Pascal, que como recordarán tiene la forma: "for vc:=vi to vf do instrucciones", donde las "instrucciones" se ejecutan repetidamente desde que la variable de control "vc" toma un valor inicial "vi" hasta que alcanza un valor final "vf" incrementando en cada repetición su valor en uno. Igualmente "Do" ejecuta las "instrucciones" desde que la variable "vc" toma un valor inicial "vi" hasta que alcanza un valor final "vf", pero en este caso en cada repetición el valor de la variable de control "vc" se incrementa en "inc" (y no necesariamente en 1 como sucede en Pascal).

Si no se asigna un valor a "inc", entonces los incrementos son de uno en uno (igual que en Pascal), si no se asigna un valor a "vi" entonces el valor inicial es "1", si no se escribe la variable de control "vc", entonces las "instrucciones" se repiten "vf" veces.

Así por ejemplo en la siguiente instrucción se crea una función para calcular el factorial de un número entero (que como recordarán es: $n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot \dots \cdot n$):

```
factorial[n_Integer]:=Module[{f=1},Do[f=f*i,{i,n}];f]
```

Ejecutando la función para "n=5" obtenemos:

```
factorial[5]
120
```

Que es el resultado correcto, para "n=30" obtenemos:

```
factorial[30]
265252859812191058636308480000000
```

Que también es el resultado correcto y que puede ser corroborado con la función "Factorial" de Mathematica:

```
Factorial[30]
265252859812191058636308480000000
```

La instrucción "Do", al igual que la instrucción "For" de Pascal, se emplea cuando se conoce el número de veces que debe repetirse una secuencia de instrucciones. Como todas las estructuras de Mathematica, la estructura "Do" puede ser anidada, pero además es posible emplear dos o más contadores en una misma instrucción, de acuerdo al siguiente formato:

```
Do[instrucciones, {vc1, vi1, vf1, inc1},{vc2, vi2, vf2, inc2},...]
```

En este caso, para cada uno de los valores que toma la variable de control 1 "vc₁", la variable de control 2 "vc₂" va desde el valor inicial 2 "vci₂" hasta el valor final 2 "vfi₂", incrementando su valor en "inc₂". Por ejemplo en la siguiente instrucción se crea una función que calcula el resultado de la siguiente sumatoria:

$$z(n) = \sum_{i=3}^n \sum_{j=1,3,5,\dots}^i j$$

```
z[n_Integer]:=Module[{s=0},Do[s=s+j,{i,3,n},{j,1,i,2}];s]
```

Ejecutando la función para "n=7" se obtiene:

```
z[7]
42
```

Y para números menores a 3 se obtiene:

```
z[2]
0
```

Sin embargo, en el caso de las sumatorias y las productorias, generalmente resulta más sencillo emplear las instrucciones "Sum" (Σ) y "Product" (Π), que tienen la misma estructura y opciones que "Do":

```
Sum[instrucciones, {vc1, vi1, vf1, inc1},{vc2, vi2, vf2, inc2},...]
```

```
Product[instrucciones, {vc1, vi1, vf1, inc1},{vc2, vi2, vf2, inc2},...]
```

Pero en las cuales las sumatorias y productorias, se llevan a cabo automáticamente. Así la anterior función empleando "Sum" es:

```
z[n_Integer]:=Sum[j,{i,3,n},{j,1,i,2}]
```

Que como se ve es más sencilla que la función anterior y que devuelve por supuesto los mismos resultados:

```
z[7]
42
z[2]
0
```

La función "factorial" elaborada con "Product" es la siguiente:

```
factorial[n_Integer]:=Product[i,{i,n}]
```

Que, por supuesto, devuelve los mismos resultados:

```
factorial[5]
120
factorial[30]
265252859812191058636308480000000
```

Cuando no se sabe el número de veces que deben repetirse las instrucciones, se puede emplear la estructura "While", la cual es equivalente a la estructura "while" de Pascal y que como recordarán tiene la siguiente forma: "while condición do instrucciones", donde las "instrucciones" se evalúan repetidamente mientras la "condición" es verdadera. La estructura "While" de Mathematica tiene la forma:

```
While[condición, instrucciones]
```

Y al igual que en la estructura "while" de Pascal, las "instrucciones" se evalúan repetidamente mientras la condición es verdadera. Si la primera vez que se prueba la "condición" es falsa, entonces las "instrucciones" no se ejecutan ni una sola vez.

Por ejemplo para calcular la raíz cuadrada de un número, empleando la ecuación de Newton: $x_2 = (x_1 + n/x_1)/2$, se debe asumir un valor inicial para "x₁", normalmente "1.1", con ese valor se calcula un valor para "x₂" y se compara el valor asumido "x₁" con el valor calculado "x₂". Si son iguales, en un determinado número de dígitos, el proceso concluye siendo la raíz cuadrada "x₂" (o "x₁" puesto que los dos valores son casi iguales), de no ser así "x₁" toma el valor de "x₂" y el proceso se repite mientras los dos valores no sean iguales en el número de dígitos especificado.

Dos números " s_1 " y " s_2 " tienen al menos " n " dígitos iguales cuando se cumple (es verdadera) la siguiente relación:

$$\left| \frac{s_1}{s_2} - 1 \right| < 10^{-n}$$

Emplearemos frecuentemente esta relación en la solución de problemas y en la programación de los diferentes métodos numéricos, pues, por su naturaleza, los mismos son sólo aproximados, es decir permiten calcular resultados que son confiables hasta un determinado número de dígitos (una determinada precisión), pero casi nunca se calculan resultados que son exactos en todos sus dígitos, por una parte porque en la mayoría de los cálculos prácticos no se requiere más allá de un determinado número de dígitos y por otra porque normalmente se trabaja con números reales, los cuales pueden tener un número infinito de dígitos.

Empleando la anterior relación y la estructura "While", la función para calcular la raíz cuadrada de un número, con 12 dígitos de precisión, es:

```
rcuadrada[n_?Positive]:=Module[{x2=1.1,x1=2.},
  While[Abs[x1/x2-1]>= 10^-12,x1=x2;x2=(x1+n/x1)/2];
  x2]
```

Observe que el valor inicial "1.1" se asigna a "x2" y no a "x1", como se dijo al describir el proceso, se procede así porque la primera instrucción dentro de "While" es "x1=x2", por lo tanto el primer valor que toma "x1", dentro del ciclo, es el de "x2" (es decir 1.1). Por otra parte a "x1" se le ha asignado el número "2.", esto es simplemente para que "x1" y "x2" tengan valores diferentes al entrar al ciclo "While" y evitar así que la "condición" sea falsa de entrada, pues de ser así las "instrucciones" no se evaluarían ni una sola vez. En consecuencia a "x1" se le puede asignar cualquier valor siempre que sea diferente al de "x2".

Ejecutando "rcuadrada" para "n=2" obtenemos:

```
rcuadrada[2]//InputForm
1.414213562373095
```

Donde se ha empleado "InputForm" para ver el resultado con todos sus dígitos. Podemos corroborar este resultado empleando la función "Sqrt" de Mathematica:

```
Sqrt[2.]//InputForm
1.4142135623730951
```

Comprobamos así que el resultado es exacto no sólo en los 12 dígitos requeridos sino en 16, lo que puede suceder pues, como se dijo, la relación que empleamos como "condición" garantiza que el resultado sea exacto en "al menos" " n " dígitos, pero no impide que sea exacto en un mayor número de dígitos.

Si ejecutamos la función con un número negativo, obtenemos la expresión sin evaluar (debido a que el parámetro especifica un número positivo):

```
rcuadrada[-5.]
rcuadrada[-5.]
```

Si creamos la función siguiendo directamente el procedimiento descrito, es decir asignando el valor inicial a "x1", calculando "x2" e igualando luego "x1" a "x2":

```
rcuadrada2[n_?Positive]:=Module[{x1=1.1,x2=2.},
  While[Abs[x1/x2-1]>= 10^-12,x2=(x1+n/x1)/2;x1=x2];
  x2]
```

Y ejecutamos la misma para "n=2" obtenemos:

```
rcuadrada2[2]//InputForm
1.459090909090909
```

Que como se ve es un resultado erróneo y ello sucede porque con esta lógica las instrucciones se evalúan sólo una vez. Al ser la última instrucción "x1=x2", "x1" y "x2" son iguales, por lo tanto al probar la "condición" la misma resulta ser siempre falsa y en consecuencia el ciclo concluye.

Con frecuencia cuando se emplea la estructura "While" sin modificaciones, es necesario modificar la lógica para que se ajuste a la estructura, tal como se ha podido evidenciar en el anterior ejemplo. En general, sin embargo, es más eficiente y claro modificar la estructura para que se ajuste a la lógica en lugar de modificar la lógica para que se ajuste a la estructura.

La estructura "While", así como las estructuras iterativas "Do" y "For", pueden ser modificadas mediante las instrucciones "Break[]", "Continue[]" y "Return[expresión]". Estas instrucciones son equivalentes a los comandos "Break", "Continue" y "Exit" de Pascal.

La instrucción "Break[]" termina la ejecución de la estructura iterativa, la instrucción "Continue[]" provoca que se ejecute el siguiente ciclo iterativo (sin esperar a que concluya el actual) y la instrucción "Return[expresión]" termina la ejecución de una función devolviendo el resultado de la "expresión" (si la "expresión" no existe, "Return" devuelve "Null").

En la mayoría de los problemas numéricos, la lógica implica que el ciclo concluya cuando una condición es verdadera no al principio del mismo (como ocurre con "While"), sino en algún lugar intermedio o al final. En esos casos, que son los más frecuentes, la condición de la estructura "While" es simplemente "True" (lo que crea un ciclo infinito) y la condición o condiciones, que provocan la conclusión del ciclo, están en algún lugar dentro de la secuencia de instrucciones, normalmente al interior de una estructura "If" y conjuntamente una instrucción "Break" o "Return":

```
While[True, instrucción_1;...;If[condición,Break[]];...;instrucción_n]
```

Empleando estas modificaciones la función "rcuadrada" puede ser rescrita de la siguiente forma:

```
rcuadrada[n_?Positive]:=Module[{x1=1.1,x2},
  While[True,x2=(x1+n/x1)/2;If[Abs[x1/x2-1]<10^-12,Break[]];x1=x2;];
  x2]
```

Como se puede observar, en este caso se ha seguido la lógica natural: "x1" toma el valor inicial "1.1", se calcula "x2", se comparan "x1" con "x2" y si dichos valores son iguales en 12 dígitos el proceso concluye (con Break), caso contrario "x1" toma el valor de "x2" y el proceso se repite.

Por supuesto la función sigue devolviendo resultados correctos:

```
rcuadrada[2]//InputForm
1.414213562373095
```

En lugar de "Break" podemos emplear igualmente "Return":

```
rcuadrada2[n_?Positive]:=Module[{x1=1.1,x2},
  While[True,x2=(x1+n/x1)/2;If[Abs[x1/x2-1]<10^-12,Return[x2]];x1=x2;];
  ]
```

Que sigue la misma lógica que con "Break" y que por supuesto sigue devolviendo resultados correctos:

```
rcuadrada2[2]//InputForm
1.414213562373095
```

Aunque en este caso no resulta adecuado, pues da lugar a una lógica más complicada, podemos emplear (sólo como ejemplo) la modificación "Continue", conjuntamente "Return":

```
rcuadrada3[n_?Positive]:=Module[{x1=1.1,x2},
  While[True,
    x2=(x1+n/x1)/2;If[Abs[x1/x2-1]>= 10^-12,x1=x2;Continue[]];Return[x2]];
]
```

Que igualmente devuelve resultados correctos:

```
rcuadrada3[2]//InputForm
1.414213562373095
```

Otra estructura iterativa que podemos emplear en Mathematica es "For", la misma que, a pesar del nombre, no tiene nada en común con la estructura "for" de Pascal (la cual como vimos anteriormente es equivalente a la estructura "Do"). La estructura "For" de Mathematica tiene la siguiente forma:

For[instrucciones_1, condición, instrucciones_3, instrucciones_2]

En esta estructura se evalúan primero las "instrucciones_1", las cuales normalmente inicializan variables, luego se prueba la "condición" y si la misma es falsa la estructura "For" concluye, caso contrario (si es verdadera), se evalúan las "instrucciones_2", que normalmente constituyen el cuerpo de la estructura y luego las "instrucciones_3", que usualmente se emplean para incrementar variables. Posteriormente se vuelve a probar la "condición", prosiguiendo de esta manera hasta que la "condición" es falsa o hasta que el ciclo concluye por la ejecución de una instrucción "Break" o "Return".

La función "rcuadrada" programada con la instrucción "For" y siguiendo la lógica natural es la siguiente:

```
rcuadrada[n_?Positive]:=Module[{x1,x2},
  For[x1=1.1,True,x1=x2,x2=(x1+n/x1)/2;If[Abs[x1/x2-1]<=10^-12,Return[x2]]];
]
```

Como era de esperar, llamando a la función con "n=2" se obtiene la respuesta correcta:

```
rcuadrada[2]//InputForm
1.414213562373095
```

La instrucción "For" puede ser empleada de manera análoga a la función "for" de Pascal, por ejemplo se puede implementar la función "factorial" empleando "For":

```
factorial[n_Integer]:=Module[{f,i},For[f=1;i=1,i<=n,i++,f=f*i];f]
```

Que devuelve los resultados correctos:

```
factorial[5]
120
```

Pero que como se puede observar resulta más complicada que con la estructura "Do" (y más aún que con "Product").

6.1.1. Ejemplos

1. Elabore cuatro funciones que empleando "Sum", "Do", "While" y "For" calculen el valor de la sumatoria: $\sum (i^{0.5} + j^{0.5})$, donde "i=2,3,4,...,n" y "j=1,1.2,1.4,...,m" (para cada valor de "i").

```
Remove[f1,f2,f3,f4]
f1[n_,m_]:=If[NumberQ[n] && NumberQ[m],
```

```

Sum[Sqrt[i]+Sqrt[j],{i,2,n},{j,1,m,0.2}],
"Los parámetros deben ser numéricos"]
f1[5,3]
142.604
f1[a,b]
Los parámetros deben ser numéricos

f2[n_,m_]:=Which[NumberQ[n] && NumberQ[m],
Module[{s=0},
Do[s=s+Sqrt[i]+Sqrt[j],{i,2,n},{j,1,m,0.2}];s],
True,"Los parámetros deben ser numéricos"]
f2[5,3]
142.604
f2[a,b]
Los parámetros deben ser numéricos

f3[n_,m_]:=Switch[NumberQ[n] && NumberQ[m],True,
Block[{s=0,i=2,j},
While[i≤n,j=1;
While[j≤m,s+=Sqrt[i]+Sqrt[j];j+=0.2];i+=1];s],
False,"Los parámetros deben ser numéricos"]
f3[5,3]
142.604
f3[a,b]
Los parámetros deben ser numéricos

```

```

f4[n_,m_]:=Module[{s=0,i,j},
For[i=2,i≤n,i+=1,
For[j=1,j≤m,j+=0.2,s+=Sqrt[i]+Sqrt[j]]];
s]/;NumberQ[n] && NumberQ[m]
f4[n_,m_]:= "Los parámetros deben ser numéricos"
f4[5,3]
142.604
f4[a,b]
Los parámetros deben ser numéricos

```

2. Elabore cuatro módulos que empleando "Product", "Do", "While" y "For" calcule el valor de la productoria: $\prod (i+j)^{0.5}$, donde "i=3,5,7,...,n" y "j=2,5,8,...,m" (para cada valor de "i").

```

Remove[f1,f2,f3,f4]
f1[n_,m_]:=If[NumberQ[n] && NumberQ[m],
Product[Sqrt[i+j],{i,3,n,2},{j,2,m,3}],
"Los parámetros deben ser numéricos"]
f1[5,7]
20  $\sqrt{7}$ 
f1["6","9"]
Los parámetros deben ser numéricos

f2[n_,m_]:=Which[NumberQ[n] && NumberQ[m],
Module[{p=1},
Do[p*=Sqrt[i+j],{i,3,n,2},{j,2,m,3}];p],
True,"Los parámetros deben ser numéricos"]
f2[5,7]
20  $\sqrt{7}$ 
f2["6","9"]
Los parámetros deben ser numéricos

```



```
f3[n_,m_]:=Piecewise[
  {{Block[{p=1,i=3,j},
    While[i≤n,j=2;
      While[j≤n,p*=Sqrt[i+j];j+=3;i+=2];p],
    NumberQ[n] && NumberQ[m]}},
  "Los parámetros deben ser numéricos"]
f3[5,7]
20  $\sqrt{7}$ 
f3["6","9"]
Los parámetros deben ser numéricos
```

```
f4[n_,m_]:=Switch[NumberQ[n] && NumberQ[m],True,
  Block[{p=1,i,j},
    For[i=3,i≤n,i+=2,
      For[j=2,j≤m,j+=3,p*=Sqrt[i+j]]];p],
  False,"Los parámetros deben ser numéricos"]
f4[5,7]
20  $\sqrt{7}$ 
f4["6","9"]
Los parámetros deben ser numéricos
```

3. Elabore tres funciones que empleando "Do", "While" y "For" calculen el Fibonacci de un número entero positivo. El Fibonacci de un número se calcula con la expresión $F_n = F_{n-1} + F_{n-2}$, donde $F_1 = F_2 = 1$.

```
Fibonacci1[n_]:=If[IntegerQ[n] && Positive[n],
  Module[{f1=1,f2=1,f3},
    Do[f3=f2+f1;f1=f2;f2=f3,{i,3,n}];f2],
  "El parámetro debe ser entero y positivo"]
Fibonacci1[15]
610
```

Resultado que podemos corroborar con la función "Fibonacci" de "Mathematica"

```
Fibonacci[15]
610
Fibonacci[0]
El parámetro debe ser entero y positivo
```

```
Fibonacci2[n_]:=Switch[IntegerQ[n] && Positive[n],True,
  Block[{f1=1,f2=1,f3,i=3},
    While[i≤n,f3=f2+f1;f1=f2;f2=f3;i++];f2],
  False,"El parámetro debe ser entero y positivo"]
Fibonacci2[15]
610
Fibonacci2[0]
El parámetro debe ser entero y positivo
```

```
Fibonacci3[n_]:=Module[{f1=1,f2=1,f3,i},
  For[i=3,i≤n,i++,f3=f2+f1;f1=f2;f2=f3];
  f2];NumberQ[n] && Positive[n]
Fibonacci3[n_]:= "El parámetro debe ser entero y positivo"
Fibonacci3[15]
610
Fibonacci3[0]
El parámetro debe ser entero y positivo
```

4. Elabore tres funciones que empleando "While", "For" y "Do" calculen la raíz cuadrada de un número positivo o negativo "n" empleando la ecuación de Newton: $x_2 = (x_1 + n/x_1)/2$.

```

rcuadrada1[n0_] := Block[{x1=1.1, x2, n=Abs[n0]},
  While[True,
    x2=(x1+n/x1)/2;
    If[Abs[x1/x2-1]<10^-12, Break[]];
    x1=x2];
  If[Positive[n0], x2, 0.+x2 I]]
rcuadrada1[2]
1.41421
rcuadrada1[-2]
0. +1.41421 i

rcuadrada2[n0_] := Module[{x1, x2, n=Abs[n0]},
  For[x1=1.1, True, x1=x2,
    x2=(x1+n/x1)/2;
    If[Abs[x1/x2-1]<10^-12, Break[]]];
  If[Positive[n0], x2, 0.+x2 I]]
rcuadrada2[2]
1.41421
rcuadrada2[-2]
0. +1.41421 i

```

Para que "Do" se comporte de manera similar a un ciclo infinito, se fija un límite de iteraciones lo suficientemente alto como para que no pueda ser alcanzado en la práctica. Así como la raíz cuadrada de un número normalmente se calcula en menos de 50 iteraciones, fijando un límite de 300, se tiene un límite prácticamente inalcanzable y por lo tanto un comportamiento equivalente al de un ciclo infinito:

```

rcuadrada3[n0_] := Block[{x1=1.1, x2, n=Abs[n0]},
  Do[x2=(x1+n/x1)/2;
    If[Abs[x1/x2-1]<10^-12, Break[]];
    x1=x2, {300}];
  If[Positive[n0], x2, 0.+x2 I]]
rcuadrada3[2]
1.41421
rcuadrada3[-2]
0. +1.41421 i

```

5. Elabore tres funciones que empleando "While", "For" y "Do" determinen si un número es o no primo.

```

Primol[n_] := If[IntegerQ[n] && Positive[n],
  Module[{i=2},
    While[ i<=n/2,
      If[Divisible[n,i], Return[False]];
      i++];
    True],
  "El parámetro debe ser entero positivo"]
Primol[211]
True
Primol[220]
False

```

podemos corroborar estos resultados con "PrimeQ":

```

PrimeQ[211]
True
PrimeQ[220]
False

```

```

Primo2[n_]:=Module[{i},
  For[i=2,i<=n/2,i++,
    If[Divisible[n,i],Return[False]]];
  True];IntegerQ[n] && Positive[n]
Primo2[n_]:= "El parámetro debe ser entero positivo"
Primo2[211]
True
Primo2[220]
False

```

Debido a un error en el programa "Mathematica", el comando "Return" no funciona con "Do" y por ello la siguiente función siempre devuelve "True":

```

Primo3[n_]:=Which[IntegerQ[n] && Positive[n],
  Do[If[Divisible[n,i],Return[False]],
    {i,2,Quotient[n,2]};True,
  True,"El parámetro debe ser entero positivo"]

```

Entonces, cuando se trabaja con "Do", no queda otra alternativa que emplear "Catch" y "Throw": "Throw[valor]" termina la ejecución y su "valor" se convierte en el resultado del "Catch" que lo encierra:

```

Primo3[n_]:=Which[IntegerQ[n] && Positive[n],
  Catch[Do[If[Divisible[n,i],Throw[False]],
    {i,2,Quotient[n,2]};True],
  True,"El parámetro debe ser entero positivo"]
Primo3[211]
True
Primo3[220]
False

```

6. Elabore tres funciones que empleando "While", "For" y "Do" inviertan los dígitos de un número entero. Por ejemplo si el número es "12345" el resultado deberá ser "54321".

```

Invertir1[n0_]:=Module[{n=Abs[n0],ni=0},
  While[n<0,
    ni=ni*10+Mod[n,10];
    n=Quotient[n,10]];
  If[Positive[n0],ni,-ni];IntegerQ[n0]
Invertir1[n0_]:= "El parámetro debe ser entero"
Invertir1[12345]
54321
Invertir1[-12345]
-54321

```

```

Invertir2[n0_]:=If[IntegerQ[n0],
  Block[{n=Abs[n0],ni},
    For[ni=0,n<0,n=Quotient[n,10],
      ni=ni*10+Mod[n,10];
      If[Positive[n0],ni,-ni]],
  "El parámetro debe ser entero"]
Invertir2[12345]
54321
Invertir2[-12345]
-54321

```

```

Invertir3[n0_Integer]:= Module[{n=Abs[n0],ni=0},
  Do[If[n==0,Break[]];

```

```

ni=ni*10+Mod[n,10];
n=Quotient[n,10],{50}];
If[Negative[n0],-ni,ni]],
Invertir3[12345]
54321
Invertir3[-12345]
-54321

```

7. Elabore dos funciones que empleando "While" y "For" calculen el exponente de un número real con 12 dígitos de precisión, empleando la serie de Taylor: $e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \dots + \infty$.

Al programar una serie se siguen tres pasos: a) inicializar variables; b) calcular el nuevo término de la serie empleando el penúltimo término calculado; c) calcular el nuevo valor de la serie (empleando el nuevo término); d) comprobar si los dos últimos valores de la serie son iguales en un determinado número de dígitos y si es así el resultado es el último valor calculado, caso contrario, se cambian variables, incrementan contadores y se repite el proceso desde el paso (b).

Siempre que sea posible se deben evitar las restas, pues cuando se restan números muy grandes o muy pequeños se pierden dígitos, dando lugar a resultados erróneos. Si no se pueden evitar las restas se debe procurar trabajar con valores pequeños. En este caso por ejemplo se pueden evitar las restas trabajando con el valor absoluto del número, entonces el resultado negativo se obtiene tomando en cuenta que $e^{-x} = 1/e^x$:

```

Expol[x_]:=Which[NumberQ[x],
Module[{xx=Abs[x],ter=1,i=1,s1,s2},
s1=ter;
While[True,
ter=ter*xx/i;
s2=s1+ter;
If[Abs[s1/s2-1]<10^-12,Break[]];
s1=s2;i++];
If[Negative[x],1/s2,s2]],
True,"El parámetro debe ser numérico"]

```

Los resultados mostrados con los 12 dígitos de precisión (empleando "NumberForm") son:

```

NumberForm[Expol[5.78],12]
323.759190417

```

```

NumberForm[Expol[-76.34],12]
7.01389486149×10-34

```

Los cuales pueden ser comprobados con la función "Exp":

```

NumberForm[Exp[5.78],12]
323.759190417

```

```

NumberForm[Exp[-76.34],12]
7.01389486149×10-34

```

Por supuesto cuando el parámetro no es numérico se obtiene:

```

Expol["a"]

```

El parámetro debe ser numérico

```

Expo2[x_]:=If[NumberQ[x],
Block[{xx=Abs[x],i=1,ter=1,s1,s2},
For[s1=ter,True,s1=s2;i++,
ter=ter*xx/i;
s2=s1+ter;
If[Abs[s1/s2-1]<10^-12,Break[]]];
If[Positive[x],s2,1/s2]],
"El parámetro debe ser numérico"]

```

```

NumberForm[Expo2[5.78],12]
323.759190417
NumberForm[Expo2[-76.34],12]
7.01389486149×10(-34)
Expo2["a"]
El parámetro debe ser numérico

```

```

Expo3[x_]:=Module[{xx=Abs[x],i=1,ter=1,s1,s2},
  s1=ter;
  Do[ter=ter*xx/i;
    s2=s1+ter;
    If[Abs[s1/s2-1]<10^-12,Break[]];
    s1=s2;i++,{300}];
  If[Positive[x],s2,1/s2]]/;NumberQ[x]
Expo3[x_]:= "El parámetro debe ser numérico"
NumberForm[Expo3[5.78],12]
323.759190417
NumberForm[Expo3[-76.34],12]
7.01389486149×10(-34)
Expo3["a"]
El parámetro debe ser numérico

```

8. Elabore tres funciones que empleando "While", "For" y "While", calculen el seno de un ángulo en radianes con 14 dígitos de precisión, empleando la serie de Taylor: $\text{sen}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots + \infty$.

En este caso las restas no pueden evitarse, por lo que se evita trabajar con números muy grandes reduciendo el ángulo a su equivalente entre 0 y 2π :

```

Senol[x0_]:=If[NumberQ[x0],
  Block[{x,ter,i=2,j=3,xx,s1,s2},
    x=FractionalPart[x0/(2*Pi)]*2*Pi;
    If[x==0,Return[0]];
    ter=x;xx=-x^2; s1=ter;
    While[True,
      ter=ter*xx/(i*j);
      s2=s1+ter;
      If[Abs[s1/s2-1]<10^-14,Return[s2]];
      s1=s2;i+=2;j+=2]],
  "El parámetro debe ser numérico"]
NumberForm[Senol[41.32],14]
-0.46115417603206
Resultado que puede ser corroborado con la función "Sin":
NumberForm[Sin[41.32],14]
-0.46115417603206
Senol[a]
El parámetro debe ser numérico

```

```

Seno2[x0_]:=Piecewise[{{
  Block[{x,ter,i=2,j=3,xx,s1,s2},
    x=FractionalPart[x0/2/Pi]*2*Pi;
    If[x==0,Return[0]];
    ter=x;xx=-x^2;
    For[s1=ter,True,s1=s2;i+=2;j+=2,
      ter=ter*xx/i/j;
      s2=s1+ter;
      If[Abs[s1/s2-1]<10^-14,Return[s2]]]],
  NumberQ[x0]}],
  "El parámetro debe ser numérico"]

```

```
NumberForm[Seno2[41.32],14]
-0.46115417603206
```

```
Seno3[x0_]:=Switch[NumberQ[x0],True,
Module[{x,ter,i=2,j=3,xx,s1,s2},
x=FractionalPart[x0/(2*Pi)]*2*Pi;
If[x==0,Return[0]];
ter=x;s1=ter;xx=-x^2;
Catch[Do[ter=ter*xx/i/j;
s2=s1+ter;
If[Abs[s1/s2-1]<10^-14,Throw[s2]];
s1=s2;i+=2;j+=2,{300}]]],
False,"El parámetro debe ser numérico"]
NumberForm[Seno3[41.32],14]
-0.46115417603206
```

6.1.2. Ejercicios

1. Elabore cuatro funciones que empleando "Sum", "Do", "While" y "For", calculen el valor de la siguiente sumatoria $\sum (i^{0.2}-j^{0.3})$, donde "i=1,2,...,5" y "j=3,3.5,4,...,7".
2. Elabore cuatro funciones que empleando Product, "Do", "While" y "For", calculen el valor de la siguiente productoria $\prod (i+j)^{0.25}$, donde "i=1,1.4,1.8,...,3" "j=2,2.3,2.6,...,5".
3. Elabore tres funciones que empleando "While", "For" y "Do" calculen la raíz cúbica de un número positivo o negativo. La ecuación de Newton para el cálculo de la raíz cúbica es: $x_2=(2x_1+n/x_1^2)/3$.
4. Elabore tres funciones que empleando "Do", "While" y "For" calculen el Chebyshev "n" de un número real "x". El Chebyshev se calcula con la siguiente expresión: $C_{n+1}(x)=2xC_n(x)-C_{n-1}(x)$, donde $C_0(x)=1$, $C_1(x)=x$.
5. Elabore tres funciones que empleando "While", "For" y "Do" calculen el arco tangente hiperbólico de un número real empleando la serie de Taylor: $\tanh^{-1}(x)=x+x^3/3+x^5/5+x^7/7+\dots+\infty$.
6. Elabore tres funciones que empleando "While", "For" y "Do" calculen el coseno de un ángulo en radianes empleando la serie de Taylor: $\cos(x)=1-x^2/2+x^4/4-x^6/6+\dots+\infty$.
7. Elabore tres funciones que empleando "While", "For" y "Do" determinen si un número es o no perfecto. Un número es perfecto si la suma de todos sus divisores (excepto él mismo) es igual al número, así el número 6 es perfecto, porque la suma de sus divisores (1+2+3) es igual a 6.

6.2. Otras estructuras iterativas

En Mathematica se puede emplear también la instrucción "Goto[etiqueta]" y su complemento "Label[etiqueta]", que son equivalentes a los comandos "goto" y la palabra reservada "Label" de Pascal. Al igual que en Pascal, "Goto" hace que el programa salte hasta el lugar donde se encuentra la instrucción "Label" respectiva y que la ejecución continúe con la instrucción que se encuentra a continuación de la misma.

Empleando "Goto" la función para calcular la raíz cuadrada es:

```
Remove[rcuadrada]
rcuadrada[n_?Positive]:=Module[{x1=1.1,x2},
Label[inicio]; x2=(x1+n/x1)/2;
```



```
69.95226944138409, 69.95226944138409}
```

Vemos que se requieren 11 repeticiones para calcular el resultado correcto. El hecho de que el número de iteraciones necesarias varíe con el número cuya raíz se quiera calcular, hace que "Nest" no sea la estructura más adecuada para este caso. Una mejor alternativa para este caso es "FixedPoint", cuyo formato es:

FixedPoint[función, expresión, n]

"FixedPoint" opera de manera similar a "Nest", es decir emplea el resultado de la "expresión" como argumento inicial de la "función" y en las posteriores iteraciones emplea los resultados como los nuevos argumentos de la función, en este caso, sin embargo, el proceso concluye cuando los dos últimos valores calculados son iguales o cuando se ha llegado al número "n" de repeticiones. Si no se especifica el número de iteraciones "n", "FixedPoint" asume que dicho número es infinito (Infinity) y repite el proceso hasta que los dos últimos resultados sean iguales, lo que en ocasiones puede dar lugar a un ciclo infinito.

La función "rcuadrada" escrita con "FixedPoint" es la siguiente:

```
rcuadrada[n_?Positive]:=FixedPoint[(#+n/#)/2&,1.1,20]
```

Que por supuesto devuelve resultados correctos:

```
rcuadrada[2]//InputForm
1.414213562373095
```

Con "FixedPoint", al igual que con "Nest", podemos ver los resultados de cada iteración si empleamos "FixedPointList":

```
rcuadrada[n_?Positive]:=FixedPointList[(#+n/#)/2&,1.1,20]
```

Si ahora evaluamos la función obtenemos:

```
rcuadrada[2]//InputForm
{1.1, 1.459090909090909, 1.414903709997168, 1.4142137306897584,
 1.4142135623731051, 1.414213562373095, 1.414213562373095}
```

Y como se puede observar, con "FixedPoint", el proceso no se repite las 20 veces (como con "Nest"), sino sólo hasta que los dos últimos valores son iguales (el proceso se repite 20 veces, sólo en aquellos casos donde los dos últimos valores no llegan a ser iguales).

Otra estructura que en ocasiones ayuda a resolver problemas iterativos de forma más eficiente es "NestWhile", cuyo formato es el siguiente:

NestWhile[función, expresión, condición, m, n]

Esta estructura opera de manera similar a las dos anteriores, es decir emplea la "expresión" como argumento inicial para la "función" y luego, en las posteriores iteraciones, emplea los resultados como argumento para volver a evaluar la "función", en este caso, sin embargo, el proceso se repite mientras la "condición" es verdadera (True). Adicionalmente se puede establecer el número de resultados más recientes, "m", que serán empleados en la condición y el máximo número de veces, "n", que se repetirá el proceso. Si no se fija el número de resultados "m" a ser empleados en la condición, "NestWhile" asume que dicho número es 1, igualmente si no se fija el máximo número de repeticiones "n", "NestWhile", asume que dicho número es infinito (Infinity) y repite el proceso hasta que la condición sea falsa lo que (al igual que con "FixedPoint") puede dar lugar a un ciclo infinito.

En "NestWhile" la condición debe ser escrita en forma de función en línea o debe ser una de las funciones con las que cuenta Mathematica y que devuelven un resultado de tipo lógico verdadero (True) o falso (False), tales como

"OddQ", "EvenQ", "IntegerQ", "NumberQ", etc. Se puede emplear también cualquiera de los operadores relacionales estudiados previamente, pero escritos en forma de funciones tales como "Equal", "Unequal", "Greater", "Less", etc.

Por ejemplo, podemos rescribir la función "rcuadrada", siguiendo la misma lógica que con "FixedPoint", pero empleando "NestWhile":

```
rcuadrada[n_?Positive]:=NestWhile[(#+n/#)/2&,1.1,Unequal,2,20]
```

En este caso la condición es "Unequal", que toma los "2" últimos resultados y los compara, devolviendo verdadero si son diferentes y falso en caso contrario. Ejecutando la función con "n=2" obtenemos el resultado correcto:

```
rcuadrada[2]//InputForm
1.414213562373095
```

Al igual que en las dos anteriores instrucciones, podemos emplear "NestWhileList" si queremos ver los resultados de cada iteración:

```
rcuadrada[n_?Positive]:=NestWhileList[(#+n/#)/2&,1.1,Unequal,2,20]
```

Ejecutando la función con "n=2" vemos que se lleva a cabo una iteración menos que con "FixedPoint":

```
rcuadrada[2]//InputForm
{1.1, 1.459090909090909, 1.414903709997168, 1.4142137306897584,
1.4142135623731051, 1.414213562373095}
```

Con "NewtWhile" podemos escribir una condición más elaborada, por ejemplo podemos elaborar la función "rcuadrada" repitiendo el proceso mientras los dos últimos resultados no sean iguales en 12 dígitos:

```
rcuadrada[n_?Positive]:=NestWhile[(#+n/#)/2&,1.1,Abs[#1/#2-1]>10^-12&,2,20]
```

Como se puede observar la "condición" está en forma de una función pura y si la "condición" no es falsa en 20 iteraciones el proceso concluye. Ejecutando la función con "n=2" obtenemos el resultado correcto:

```
rcuadrada[2]//InputForm
1.414213562373095
```

6.2.1. Ejemplos

9. Elabore 4 funciones que empleando "Goto", "Nest", "NestWhile" y "FixedPoint", calcule el factorial de un número entero positivo.

```
factorial1[n_]:=If[IntegerQ[n] && n>=0,
Module[{f=1,i=2},
Label[uno];If[i>n,Goto[dos]];
f=f*i;i++;Goto[uno];
Label[dos];f],
"El parámetro debe ser entero positivo"]
```

```
factorial1[7]
```

```
5040
```

```
factorial1[3.4]
```

```
El parámetro debe ser entero positivo
```

```
factorial2[n_]:=Block[{i=1},
Nest[#*i++&,1,n]]/IntegerQ[n] && n>=0
factorial2[n_]:= "El parámetro debe ser entero positivo"
factorial2[7]
5040
```

```
factorial3[n_]:=Which[IntegerQ[n] && n>=0,
```

```
Module[{i=1},
  NestWhile[#*i++&,1,i≤n&]],
  True,"El parámetro debe ser entero positivo"]
factorial3[7]
5040
```

Con fixed point no se aprovecha la condición de igualdad, por lo que en realidad sólo se emplea su condición de límite de iteraciones "n":

```
factorial4[n_]:=Switch[IntegerQ[n] && n≥0,True,
  Module[{i=1,f=1},
    FixedPoint[(f=f*i;i++)&,3,n];f],
  False,"El parámetro debe ser entero positivo"]
factorial4[7]
5040
```

10. Elabore cuatro funciones que empleando "Goto", "Nest", "NestWhile" y "FixedPoint" calculen el Fibonacci de un número entero positivo. El Fibonacci de un número se calcula con la expresión $F_n = F_{n-1} + F_{n-2}$, donde $F_1 = F_2 = 1$.

```
Fibol[n_]:=If[IntegerQ[n] && Positive[n],
  Module[{f1=1,f2=1,i=3,f3},
    Label[1];If[i>n,Goto[2]];
    f3=f2+f1;f1=f2;f2=f3;i++ Goto[1];
    Label[2];f2]]
Fibol[50]
12586269025
```

```
Fibo2[n_]:=Which[IntegerQ[n] && n>0,
  If[n>2,Block[{f1,f2=1},
    Nest[(f1=f2;f2=#;f1+f2)&,1,n-2]],
  1]]
Fibo2[50]
12586269025
```

```
Fibo3[n_]:=If[n>2,
  Module[{f1=1,f2=1,f3,i=3},
    NestWhile[(f3=f1+f2;f1=f2;f2=f3;
      i++)&,1,(#<n)&];f3],
  1]/IntegerQ[n]&&Positive[n]
Fibo3[50]
12586269025
```

```
Fibo4[n_]:=If[NumberQ[n] && n>0,
  Switch[n>2,
    True,Block[{f1,f2=1},FixedPoint[(f1=f2;f2=#1;f1+f2)&,1,n-2]],
    False,1]]
Fibo4[50]
12586269025
```

11. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile" calculen la raíz cuadrada de un número positivo o negativo "n" empleando la ecuación de Newton: $x_2 = (x_1 + n/x_1)/2$. (Donde corresponda debe emplear 14 dígitos de precisión)

```
rcuad1[x_]:=If[NumberQ[x],
  Module[{x1=1.1,x2,n=Abs[x]},
    Label[uno];x2=(x1+n/x1)/2;
    If[Abs[x1/x2-1]<10^-14,Goto[dos]];
    x1=x2;Goto[uno];
```

```

Label[dos];
If[Negative[x],0.+x2 I,x2]]]
NumberForm[rcuad1[4.56],14]
2.1354156504063
NumberForm[Sqrt[4.56],14]
2.1354156504063

rcuad2[x_]:=Which[NumberQ[x],
Block[{n=Abs[x],r},
r=FixedPoint[(#+n/#)/2&,1.1,50];
If[Positive[x],r,0.+ r I]]]
NumberForm[rcuad2[4.56],14]
2.1354156504063

```

```

rcuad3[x_]:=Block[{n=Abs[x],r},
r=NestWhile[(#+n/#)/2&,1.1,
Abs[#1/#2-1]>10^-14&,2,50];
If[Negative[x],0.+r I,r]]/;NumberQ[x]
NumberForm[rcuad3[4.56],14]
2.1354156504063

```

12. Elabore tres funciones que empleando "Goto", "NestWhile" y "FixedPoint" determinen si un número es o no primo.

```

primo1[n_]:=Module[{i=1,r=False},
Label[1];If[++i>n/2,r=True;Goto[2]];
If[Divisible[n,i],r=False;Goto[2]];
i++;Goto[1];
Label[2];r]/;IntegerQ[n] &&Positive[n]
primo1[151]
True
primo1[120]
False

```

```

primo2[n_]:=If[IntegerQ[n] &&Positive[n],
Module[{i=1},
NestWhile[!Divisible[n,++i] &,True,
#&,1,Quotient[n,2]];
If[i>Quotient[n,2],True,False]]]
primo2[151]
True
primo2[120]
False

```

```

primo3[n_]:=Which[IntegerQ[n] && n>0
Module[{i=1},
FixedPoint[!Xor[Divisible[n,++i], #]&,
True,Quotient[n,2]];
If[i>Quotient[n,2],True,False]]]
primo3[151]
True
primo3[120]
False

```

13. Elabore tres funciones que empleando "Goto", "NestWhile" y "FixedPoint" inviertan los dígitos de un número entero. Por ejemplo si el número es "12345" el resultado deberá ser "54321".

```

inv1[n0_Integer]:=Module[
{n=Abs[n0],ni=0},

```

```

Label[ciclo];If[n==0,Goto[fin]];
ni=ni*10+Mod[n,10];
n=Quotient[n,10];Goto[ciclo];
Label[fin];If[Negative[n0],-ni,ni]]
inv1[12345]
54321
inv1[-549283]
-382945

```

```

inv2[n0_Integer]:=Block[{ni=0},
  NestWhile[(ni=ni*10+Mod[#,10];
    Quotient[#,10])&,Abs[n0],##0&];
  If[Positive[n0],ni,-ni]]
inv2[12345]
54321
inv2[-549283]
-382945

```

```

inv3[n0_Integer]:=Module[{ni=0},
  FixedPoint[(ni=ni*10+Mod[#,10];
    Quotient[#,10])&,Abs[n0]];
  If[n0<0,-ni/10,ni/10]]
inv3[12345]
54321
inv3[-549283]
-382945

```

14. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile" calculen el exponente de un número real con 15 dígitos de precisión, empleando la serie de Taylor: $e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \dots + \infty$.

```

expol[x_?NumberQ]:=Module[
  {xx=Abs[x],ter=1,i=1,s1,s2},s1=ter;
  Label[1];ter=ter*xx/i;s2=s1+ter;
  If[Abs[s1/s2-1]<10^-15,Goto[2]];
  s1=s2;i++;Goto[1];
  Label[2];If[Negative[x],1/s2,s2]]
NumberForm[expol[5.78],15]
323.759190417243
NumberForm[expol[-76.34],15]
7.01389486148609×10(-34)

```

```

expo2[x_?NumberQ]:=Block[
  {xx=Abs[x],ter=1,i=1,r},
  r=FixedPoint[(#+(ter*=xx/i++))&,
    1,300];If[Negative[x],1/r,r]]
NumberForm[expo2[5.78],15]
323.759190417243
NumberForm[expo2[-76.34],15]
7.01389486148608×10(-34)

```

```

expo3[x_?NumberQ]:=Module[
  {xx=Abs[x],ter=1,i=1,r},
  r=NestWhile[(#+(ter*=xx/i++))&,1,
    Abs[#1/#2-1]>10^-15&,2,300];
  If[Positive[x],r,1/r]]
NumberForm[expo3[5.78],15]
323.759190417243
NumberForm[expo3[-76.34],15]

```

$7.01389486148609 \times 10^{-34}$

15. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile", calculen el seno de un ángulo en radianes con 14 dígitos de precisión, empleando la serie de Taylor: $\text{sen}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots + \infty$.

```
seno1[x0_?NumberQ]:=Block[
  {x=FractionalPart[x0/(2*Pi)]*2*Pi,
   ter,i=2,j=3,xx,s1,s2},
  If[x==0,s2=0;Goto[fin]];
  ter=x;xx=-x^2;s1=ter;
  Label[ciclo];
  ter=ter*xx/(i*j);s2=s1+ter;
  If[Abs[s1/s2-1]<10^-15,Goto[fin]];
  s1=s2;i+=2;j+=2;Goto[ciclo];
  Label[fin];s2]
NumberForm[seno1[41.32],15]
-0.461154176032062
```

```
seno2[x0_?NumberQ]:=Module[
  {x=FractionalPart[x0/2/Pi]*2*Pi,
   ter,i=0,j=1,xx,r},
  If[x==0,Return[0]];xx=-x^2;ter=x;
  FixedPoint[#+(ter*xx/((i+=2)*
    (j+=2)))&,x,300]]
NumberForm[seno2[41.32],15]
-0.461154176032062
```

```
seno3[x0_?NumberQ]:=Module[
  {x=FractionalPart[x0/2/Pi]*2*Pi,
   ter,i=0,j=1,xx,r},
  If[x==0,Return[0]];xx=-x^2;ter=x;
  NestWhile[#+(ter*xx/((i+=2)*
    (j+=2)))&,x,
  Abs[#1/#2-1]>= 10^-15&,2,300]]
NumberForm[seno3[41.32],15]
-0.461154176032062
```

6.2.2. Ejercicios

8. Elabore cuatro funciones que empleando "Goto", "Nest", "NestWhile" y "FixedPoint", calculen el valor de la siguiente sumatoria $\sum (i^{0.2} - j^{0.3})$, donde "i=1,2,...,5" y "j=3,3.5,4,...,7".
9. Elabore cuatro funciones que empleando "Goto", "Nest", "NestWhile" y "FixedPoint" calculen el valor de la siguiente productoria $\prod (i+j)^{0.25}$, donde "i=1,1.4,1.8,...,3" "j=2,2.3,2.6,...,5".
10. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile" calculen la raíz cúbica de un número positivo o negativo. La ecuación de Newton para el cálculo de la raíz cúbica es: $x_2 = (2x_1 + n/x_1^2)/3$.
11. Elabore cuatro funciones que empleando "Goto", "Nest" y "NestWhile" y "FixedPoint", calculen el Chebyshev "n" de un número real "x". El Chebyshev se calcula con la siguiente expresión: $C_n(x) = 2xC_{n-1}(x) - C_{n-2}(x)$, donde $C_0(x) = 1$, $C_1(x) = x$.
12. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile" calculen el arco tangente hiperbólico de un número real empleando la serie de Taylor: $\tanh^{-1}(x) = x + x^3/3 + x^5/5 + x^7/7 + \dots + \infty$, $|x| < 1$.

13. Elabore tres funciones que empleando "Goto", "FixedPoint" y "NestWhile" calculen el coseno de un ángulo en radianes empleando la serie de Taylor: $\cos(x) = 1 - x^2/2 + x^4/4 - x^6/6 + \dots + \infty$.
14. Elabore tres funciones que empleando "Goto", "Nest" y "NestWhile" determinen si un número es o no perfecto. Un número es perfecto si la suma de todos sus divisores (excepto él mismo) es igual al número, así el número 6 es perfecto, porque la suma de sus divisores (1+2+3) es igual a 6.

7. RECURSIVIDAD

7.1. Recursividad en Mathematica

Como dijimos, Mathematica, al igual que Pascal, soporta la recursividad. Como recordarán la recursividad es la propiedad por la cual una función (o procedimiento) se llama a sí mismo. Al llamarse a sí mismo da inicio a un proceso repetitivo, en el cual la función se ejecuta una y otra vez hasta que se cumple una determinada condición. En consecuencia la recursividad puede ser empleada para resolver problemas iterativos.

Aunque en el fondo su naturaleza es iterativa, la recursividad implica sobre todo otra forma de pensar. En la recursividad un problema iterativo se resuelve pensando en la forma de encontrar un resultado a partir de otro resultado conocido o calculable.

El ejemplo clásico es el del factorial, que desde el punto de vista iterativo es una productoria (como ya vimos previamente):

$$n! = \prod_{i=1}^n i = 1 * 2 * 3 * \dots * n$$

Y que desde el punto de vista recursivo es la siguiente relación:

$$n! = (n-1)! * n; \quad 0! = 1$$

Es decir, desde el punto de vista recursivo el factorial de un número "n" se calcula multiplicando el factorial de "n-1" por "n". Además, en la solución recursiva es imprescindible contar con un valor conocido, o condición límite. Así en esta relación se establece que el factorial de "0" es "1", es decir que si "n" es "0" el resultado es "1".

Al comprobar la lógica recursiva sólo se debe verificar que la relación sea correcta, así por ejemplo podemos verificar si la relación recursiva es correcta para el factorial de 5:

$$5! = (5-1)! * 5 = 4! * 5 = 24 * 5 = 120$$

Entonces como la relación devuelve el valor correcto, concluimos que la misma es correcta. Se debe verificar también que la condición límite (o valor conocido) sea correcto, en este caso el valor conocido corresponde a una definición matemática (el factorial de "0" es "1"), por lo que no es necesario verificarla. Es importante, para una eficiente aplicación de la recursividad, no preocuparse por los detalles del proceso recursivo, así en la anterior comprobación, no interesa de donde sale el factorial de 4, sólo interesa que al multiplicar ese factorial por 5 se obtiene el factorial de 5.

Los valores que se requieren en una relación recursiva se calculan en las diferentes llamadas recursivas, así el factorial de 4 se calcula al llamarse a sí mismo con $4! = 3! * 4$, el de 3 en otra llamada con: $3! = 2! * 3$, el de 2 con: $2! = 1! * 2$, el de 1, con: $1! = 0! * 1$ y para 0 se sabe que el factorial es 1. Sin embargo, todo el proceso es realizado automáticamente por Mathematica (o el lenguaje que se esté empleando) por lo que en realidad no forma parte del problema y esa es la razón por la cual se señala que en la solución de un problema recursivo es importante no preocuparse de los detalles del proceso. En realidad la única razón por la que se emplea la recursividad es precisamente porque simplifica la solución de los problemas iterativos. En general las soluciones recursivas son menos eficientes que las no recursivas porque consumen más tiempo y memoria: en cada llamada recursiva se crea una copia de la función (lo que consume memoria y tiempo) y una vez que una copia devuelve un resultado la misma es destruida (lo que también consume tiempo).

La función recursiva para el cálculo del factorial es:

```
Remove[factorial]
factorial[n_Integer]:=If[n==0,1,factorial[n-1]*n]
```

Que como vemos es muy simple y que por supuesto devuelve los resultados correctos:

```
factorial[5]
120
factorial[30]
265252859812191058636308480000000
```

Mathematica permite la sobrecarga automática de funciones (algo que en Pascal se logra con la palabra reservada "overload"), es decir Mathematica permite definir dos o más funciones con el mismo nombre pero con diferentes tipos y/o diferente número de parámetros, en realidad cuando con "Condition" (/;) definimos dos o más funciones que cumplen condiciones diferentes, estamos haciendo uso de la sobrecarga de funciones.

Debido a que en Mathematica la sobrecarga de funciones es automática, es fácil cometer errores al declarar inadvertidamente dos o más funciones con el mismo nombre pero con alguna variación en sus parámetros, por ello, cuando se crea una nueva función y se quiere asegurar que sea la única vigente, se debe eliminar previamente (con "Remove") cualquier definición previa.

La sobrecarga de funciones permite simplificar aún más la creación de funciones recursivas, por ejemplo podemos volver a crear la función factorial con las siguientes declaraciones:

```
Remove[fatorial]
factorial[0]=1;
factorial[n_Integer]:=factorial[n-1]*n
```

Ahora existen dos definiciones para "factorial", la primera se aplica sólo cuando el número es 0, en cuyo caso es reemplazada directamente por el número 1, la segunda se aplica a cualquier número entero diferente de 0. Si pedimos información con relación a "factorial" obtenemos:

```
?factorial
Global`factorial
  factorial[0]=1
  factorial[n_Integer]:=factorial[n-1] n
```

En general Mathematica trata de organizar las funciones con el mismo nombre colocando primero las más específicas y al final las más generales. De esa manera evita que una función más general eclipse a una más específica. Si Mathematica no encuentra criterios para organizar las funciones las deja en el orden en que fueron definidas, por lo que en esos casos es importante organizar cuidadosamente las funciones de manera que ninguna de ellas eclipse a otras. En este caso por ejemplo cuando se llama a "factorial" Mathematica trata de aplicar la primera definición (la más específica) y sólo si no le es posible aplica la segunda, ello asegura que la función sea reemplazada con "1" cuando "n" es cero.

Como era de esperar, ejecutando la función con "n=5" y "n=30" obtenemos los resultados correctos:

```
factorial[5]
120
factorial[30]
265252859812191058636308480000000
```


7.1.1. Ejemplos

1. Elabore una función recursiva que calcule la sumatoria $\sum i$ $\{i=1,2,3, \dots, n\}$.

Este es un caso muy parecido al del factorial, sólo que en lugar de multiplicar ahora se suma:

```
suma[n_]:=suma[n-1]+n
suma[0]=0;
```

Probamos el módulo con 25 y corroboramos el resultado con "Total" y "Sum":

```
suma[25]
325
Total[Range[25]]
325
Sum[i,{i,1,25}]
325
```

2. Elabore una función recursiva que calcule la sumatoria de los primeros "n" números impares ($\sum i$ $\{i=1,3,5, \dots, 2*n-1\}$).

Este es un caso similar al del anterior ejemplo, sólo que ahora se suman los número impares (los que se consiguen con $2*n-1$):

```
sumai[n_]:=sumai[n-1]+2*n-1
sumai[0]=0;
```

Probamos el módulo con 25 y corroboramos el resultado con "Sum" y "Total":

```
sumai[25]
625
Sum[i,{i,1,25*2-1,2}]
625
Total[Range[1,25*2-1,2]]
625
```

3. Elabore una función recursiva para invertir los dígitos de un número entero. Por ejemplo si el número es "12345" el resultado deberá ser "54321".

En este caso el razonamiento recursivo se basa en el siguiente ejemplo: podemos calcular la inversa de "12345" si conocemos la inversa de "1234" (4321) sumándole $5*10000$ ($1234+50000=54321$). Ahora bien, "1234" es el cocientes de "12345" entre "10", "5" es el residuo de "12345" entre "10" y "10000" es "10" elevado al número de dígitos de "1234" (en este ejemplo obtenemos el número de dígitos con "Length" e "IntegerDigits", esta última función nos devuelve una lista con los números del entero y como sabemos "Length" nos devuelve el número de elementos en una lista). La condición límites es " $n < 10$ ", porque si el número es menor a 10, entonces sólo existe un dígito y la inversa de un dígito es el mismo número:

```
invertir[n_]:=If[n<10,n,
  invertir[Quotient[n,10]]+
  Mod[n,10]*10^Length[IntegerDigits[
    Quotient[n,10]]]]/IntegerQ[n]&&n>0
invertir[n_]:=If[Negative[n],-invertir[Abs[n]]]
```

Como se puede observa, la segunda función llama a la primera cuando el número es negativo. Probando la función obtenemos los resultados:

```
invertir[12345]
```

```
54321
invertir[-123]
-321
```

4. Elabore una función recursiva que calcule el Fibonacci de un número entero positivo. ($F_n = F_{n-1} + F_{n-2}$; $F_1 = F_2 = 1$).

Aun cuando en este caso se puede programar la función directamente, ello no resulta en un algoritmo eficiente, porque existe dos llamadas recursivas (con $n-1$ y con $n-2$) lo que implica el cálculo repetitivo de los mismos valores. Para evitar ello en Mathematica se pueden almacenar los resultados a medida que se calculan, de manera que si se requiere uno de ellos, simplemente se lo utiliza, evitando así su recálculo. Para ello se lleva a cabo una asignación inmediata del resultado a una variable que tiene el mismo nombre de la función (y cuyos parámetros son los valores recibidos en la función), tal como se muestra a continuación:

```
fibo[n_]:=fibo[n]=fibo[n-1]+fibo[n-2]
fibo[1]=fibo[2]=1;
```

Probamos entonces el algoritmo para "n=50" y corroboramos el resultado con la función "Fibonacci":

```
fibo[50]
12586269025
Fibonacci[50]
12586269025
```

5. Elabore una función recursiva que calcule el Legendre enésimo ("n") de un número real ("x"): $L_n(x) = (2-1/n)xL_{n-1}(x) - (1-1/n)L_{n-2}(x)$; $L_0(x)=1$; $L_1(x)=x$.

Este es esencialmente el mismo caso que del Fibonacci, sólo que ahora se tienen dos parámetros en lugar de uno:

```
leg[n_,x_]:=leg[n,x]=
(2-1/n)*x*leg[n-1,x]-(1-1/n)*leg[n-2,x]
leg[0,x_]=1;leg[1,x_]:=x;
```

Probamos la función para "n=5" y "x=0.75" corroborando los resultados con la función "LegendreP":

```
leg[5,0.75]
-0.416382
LegendreP[5,0.75]
-0.416382
```

6. Elabore una función recursiva que calcule la potencia entera de un número real.

El razonamiento recursivo se basa en lo siguiente: se puede calcular la potencia entera de por ejemplo 6.7^{456} si se conoce el valor de 6.7^{228} (siendo 228 la mitad de 456) pues elevando este valor al cuadrado se obtiene el resultado buscado. Si la potencia es impar, por ejemplo 6.7^{457} , entonces se calcula el resultado con $6.7^{228} \cdot 6.7$. Se sabe también que cualquier valor elevado a 0 es 1, por lo tanto la condición de finalización es que si la potencia es 0, el resultado es 1:

```
potent[x_,n_]:=If[OddQ[n],
  potent[x,Quotient[n,2]]^2*x,
  potent[x,Quotient[n,2]]^2
]/;IntegerQ[n]&&Positive[n]
potent[x_,0]=1;
```

Probamos la función para "x=3.2" y "n=25", corroborando el resultado con "^":

```
potent[3.2,25]
```

```
4.25353×1012
```

```
3.2^25
```

```
4.25353×1012
```

7. Elabore una función que empleando la recursividad, calcule el común divisor entre dos números (A y B), mediante el algoritmo de Euclides:
 $MCD(A,A)=A$; $MCD(A,B)=MCD(B,A)$ si $A>B$; $MCD(A,B)=MCD(A,B-A)$ si $A<B$.

Al ser esta una definición recursiva, simplemente se la programa:

```
mcd[a_Integer,b_Integer]:=
```

```
Which[a==b,a,a>b,mcd[b,a],a<b,mcd[a,b-a]]
```

Probamos la función con "A=30, B=40" y corroboramos el resultado con "GCD":

```
mcd[30,40]
```

```
10
```

```
GCD[30,40]
```

```
10
```

7.1.2. Ejercicios

1. Elabore una función recursiva que calcule la sumatoria de los primeros "n" números pares ($\sum_{i=2,4,6,\dots,2n}$).
2. Elabore una función recursiva que devuelva el número de dígitos que tiene un número entero positivo.
3. Elabore una función recursiva que calcule el Chebyshev enésimo ("n") de un número real ("x"). ($C_n(x)=2xC_{n-1}(x)-C_{n-2}(x)$; $C_0(x)=1$; $C_1(x)=x$).
4. Elabore una función que calcule el valor Ackerman para cualquier par de números enteros no negativos: $a(p,q)=\{q+1$ si $p=0$; $a(p-1,1)$ si $p<>0$ y $q=0$; $a(p-1,a(p,q-1))$ si $p<>0$ y $q<>0$.
5. Elabore una función recursiva que calcule el polinomio de Hermite: $H_n(x)=2xH_{n-1}(x)-2nH_{n-1}(x)$; $H_0(x)=1$; $H_1(x)=2x$.
6. Elabore una función recursiva que devuelva la parte entera de un número real.

8. GRÁFICOS

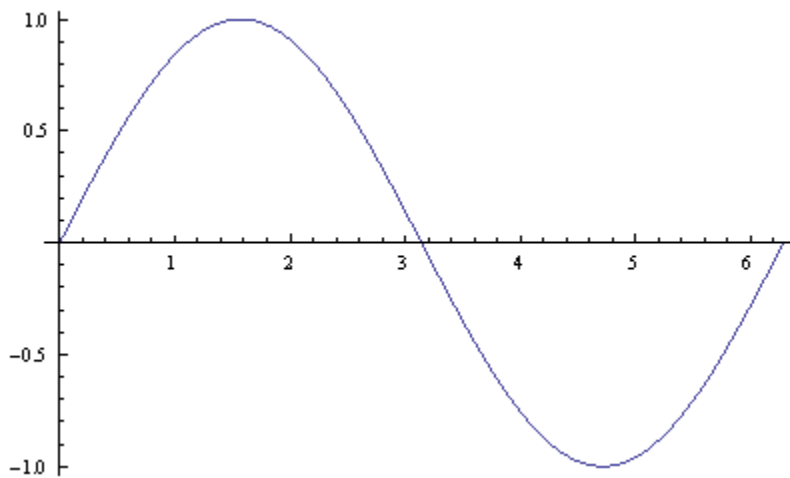
8.1. Gráficos en Mathematica

Uno de los aspectos más interesantes de Mathematica es la facilidad con la que se pueden elaborar gráficos en 2 y 3 dimensiones. Para elaborar gráficos en dos dimensiones a partir de una expresión matemática se emplea el comando Plot:

$$\text{Plot}[\text{expresión}, \{x, x_{\min}, x_{\max}\}]$$

"Plot" grafica la "expresión", que deberá ser dependiente de la variable "x", desde " $x_{\min}$ " hasta " $x_{\max}$ ", por ejemplo para graficar el seno desde 0 hasta 2π escribimos:

```
Plot[Sin[x], {x, 0, 2*Pi}]
```



Existen varias opciones que se pueden cambiar en un gráfico, aunque en general y como se puede ver, las opciones por defecto producen un resultado bastante aceptable.

Las opciones de la función "Plot", al igual que las de cualquier otra función, son modificadas escribiéndolas en forma de reglas (->) y se puede obtener un listado de las mismas con "Options":

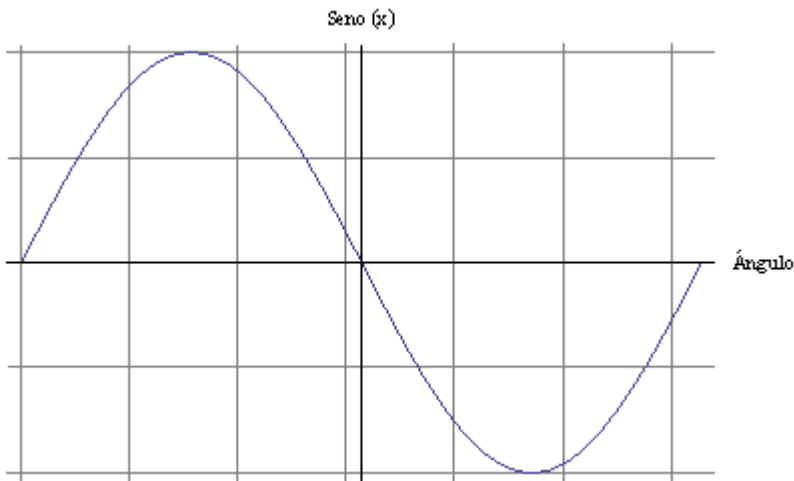
```
Options[Plot]
```

```
{AlignmentPoint->Center, AspectRatio->1/GoldenRatio, Axes->True, AxesLabel->None, AxesOrigin->Automatic, AxesStyle->{}, Background->None, BaselinePosition->Automatic, BaseStyle->{}, ClippingStyle->None, ColorFunction->Automatic, ColorFunctionScaling->True, ColorOutput->Automatic, ContentSelectable->Automatic, DisplayFunction->DisplayFunction, Epilog->{}, Evaluated->Automatic, EvaluationMonitor->None, Exclusions->Automatic, ExclusionsStyle->None, Filling->None, FillingStyle->Automatic, FormatType->TraditionalForm, Frame->False, FrameLabel->None, FrameStyle->{}, FrameTicks->Automatic, FrameTicksStyle->{}, GridLines->None, GridLinesStyle->{}, ImageMargins->0., ImagePadding->All, ImageSize->Automatic, LabelStyle->{}, MaxRecursion->Automatic, Mesh->None, MeshFunctions->{#1&}, MeshShading->None, MeshStyle->Automatic, Method->Automatic, PerformanceGoal->PerformanceGoal, PlotLabel->None, PlotPoints->Automatic, PlotRange->{Full, Automatic}, PlotRangeClipping->True, PlotRangePadding->Automatic, PlotRegion->Automatic, PlotStyle->Automatic, PreserveImageOptions->Automatic, Prolog->{}, RegionFunction->(True&), RotateLabel->True, Ticks->Automatic, TicksStyle->{}, WorkingPrecision->MachinePrecision}
```

Algunas de las opciones de uso más frecuente son: "Axes", para dibujar (True) o no (False) los ejes; "AxesLabel", para escribir etiquetas (*etiqueta x*, *etiqueta y*) o no (None) en los ejes"; "AxesOrigin" para fijar el origen de los ejes ($\{x, y\}$) o dejar que Mathematica fije el origen (Automatic); "GridLines" para dibujar (Automatic o $\{\{x_1, x_2, \dots\}, \{y_1, y_2, \dots\}\}$) o no (None) una cuadrícula en el gráfico; "PlotLabel" para asignar una etiqueta (texto) o no (None) al gráfico; "Frame" para dibujar (True) o no (False) un marco alrededor del gráfico; "Ticks" para dibujar (Automatic o $\{\{x_1, x_2, \dots\}, \{y_1, y_2, \dots\}\}$) o no (None) marcas.

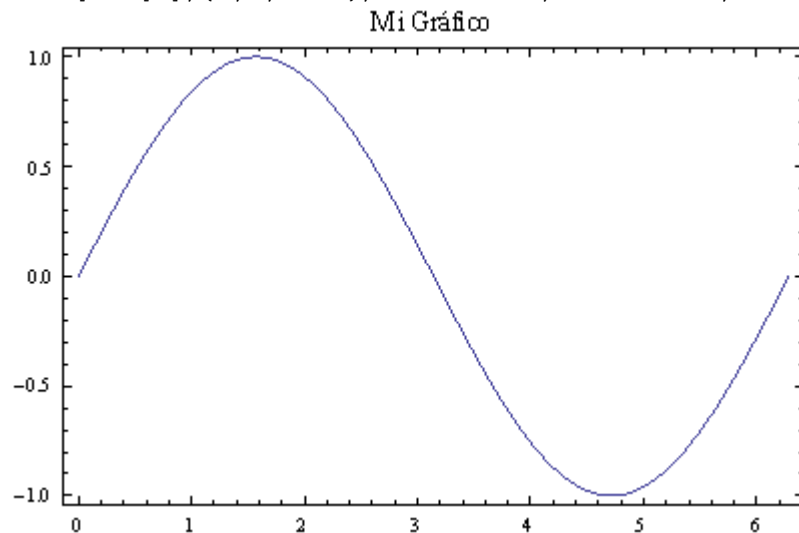
Por ejemplo para dibujar la función seno, siendo el origen del eje "x" π y del eje "y" "0", con la etiqueta "Ángulo" para el eje "x" y "Seno(x)" para el eje "y", con cuadrícula automática y sin marcas de división escribimos:

```
Plot[Sin[x], {x, 0, 2*Pi}, AxesOrigin -> {Pi, 0}, AxesLabel -> {"Ángulo", "Seno(x)"}, GridLines -> Automatic, Ticks -> None]
```



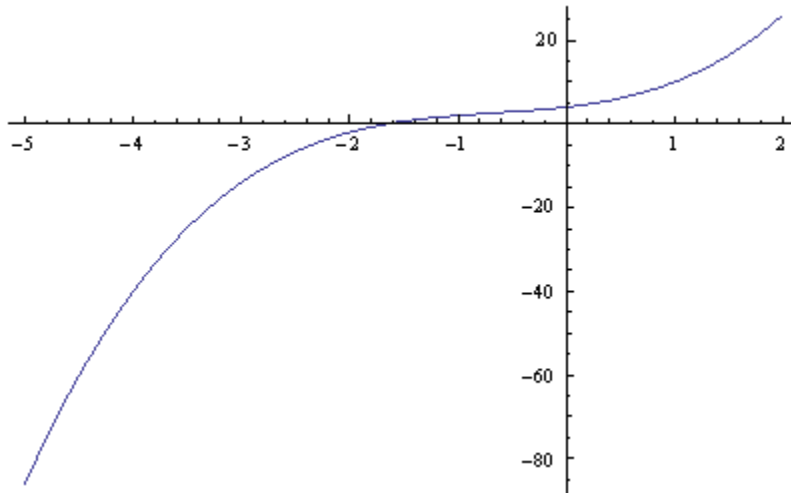
Y el mismo gráfico pero sin ejes, con un marco y con la etiqueta "Mi gráfico" es:

```
Plot[Sin[x], {x, 0, 2*Pi}, Axes -> False, Frame -> True, PlotLabel -> "Mi Gráfico"]
```



Para graficar una expresión matemática, simplemente se la escribe especificando el nombre de la variable independiente, así por ejemplo para graficar $x^3 + 2x^2 + 3x + 4$, entre -5 y 2 escribimos:

```
Plot[x^3+2*x^2+3*x+4,{x,-5,2}]
```



Para graficar puntos en lugar de funciones empleamos el comando "ListPlot":

```
ListPlot[{y1,y2,y3,...,yn}]
```

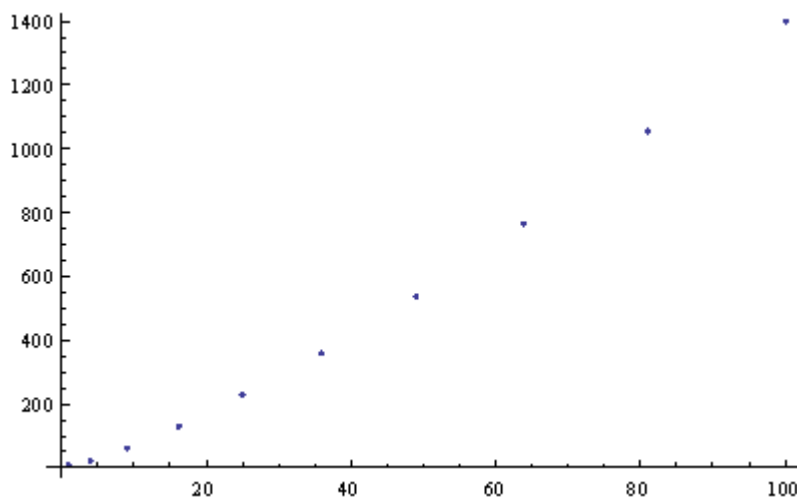
o

```
ListPlot[{ {x1,y1}, {x2,y2}, {x3,y3}, ..., {xn,yn} ]
```

En la primera forma se asumen valores consecutivos para "x": 1, 2, 3, ..., n. Por ejemplo para graficar los puntos de la tabla escribimos:

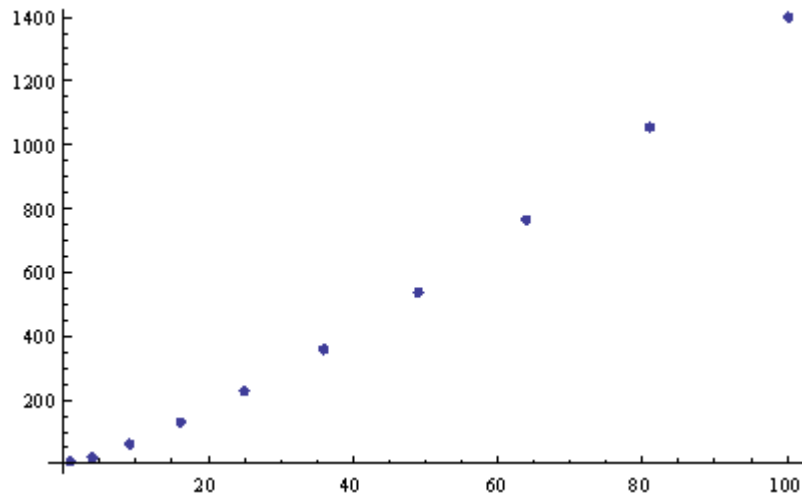
x	1	4	9	16	25	36	49	64	81	100
y	5	24	63	128	225	360	539	768	1053	1400

```
ListPlot[{ {1,5}, {4,24}, {9,63}, {16,128}, {25,225}, {36,360}, {49,539}, {64,768},  
          {81,1053}, {100,1400} ]
```



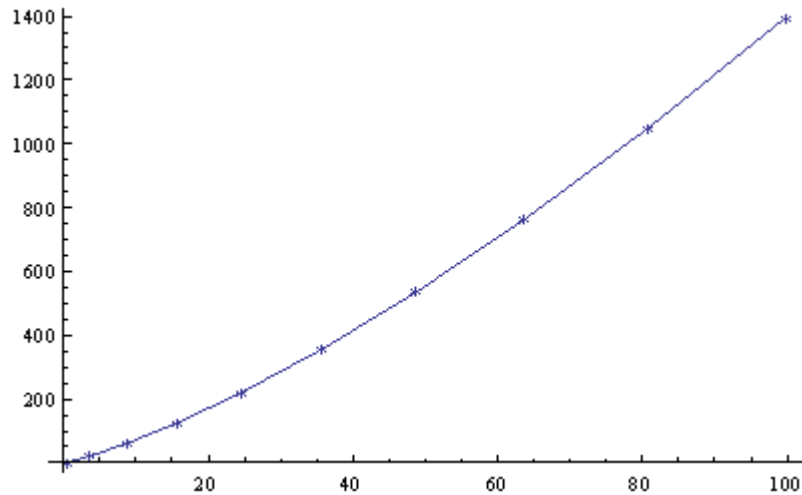
Al igual que "Plot", "ListPlot" tiene varias opciones (las cuales se pueden ver con "Options"). Así para cambiar el diámetro de los puntos se emplea la opción "PlotStyle" con la directiva "AbsolutePointSize", por ejemplo el anterior gráfico, pero con puntos de 5 pixeles de diámetro es:

```
ListPlot[{{1,5},{4,24},{9,63},{16,128},{25,225},{36,360},{49,539},{64,768},
          {81,1053},{100,1400}},PlotStyle->{AbsolutePointSize[5]}]
```



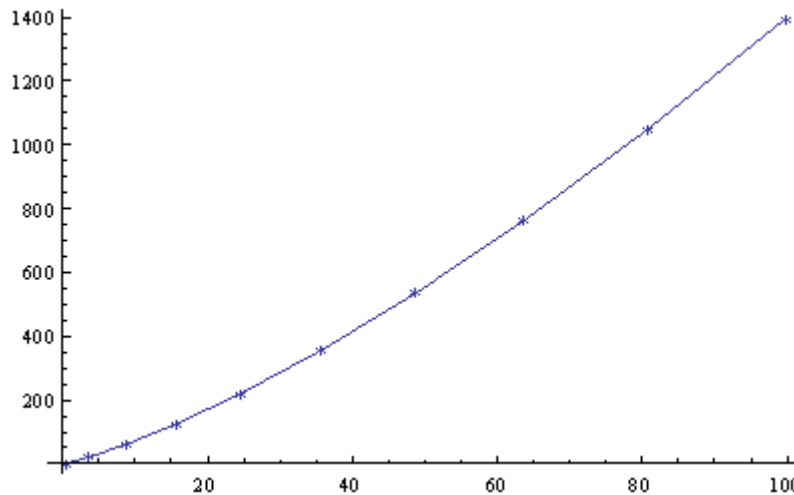
Para que los puntos se muestren unidos con líneas rectas se puede emplear la opción "Joined", la cual debe ser fijada en "True" y para que los puntos sean dibujados con un carácter diferente al punto (.), se emplea la opción "PlotMarkers", la cual se fija en el carácter o caracteres que se quieren emplear para los puntos, tal como se muestra en la siguiente instrucción:

```
ListPlot[{{1,5},{4,24},{9,63},{16,128},{25,225},{36,360},{49,539},{64,768},
          {81,1053},{100,1400}},Joined->True,PlotMarkers->"*"]
```



Alternativamente, en lugar de la opción "Joined" se puede emplear la instrucción "ListLinePlot", que por defecto dibuja los puntos unidos con líneas rectas:

```
ListLinePlot[{{1,5},{4,24},{9,63},{16,128},{25,225},{36,360},{49,539},
              {64,768},{81,1053},{100,1400}},PlotMarkers->"*"]
```

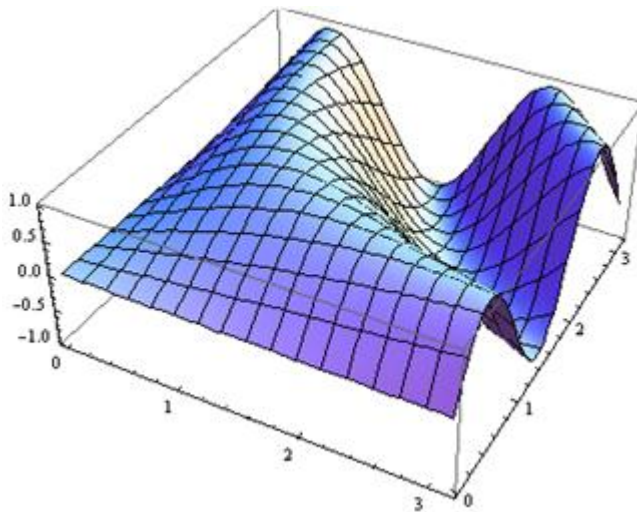



Con Mathematica también podemos hacer gráficos tridimensionales, así para graficar funciones de dos variables empleamos "Plot3D":

```
Plot3D[f, {x, x_min, x_max}, {y, y_min, y_max}]
```

Por ejemplo para graficar el " $\sin(x*y)$ " entre 0 y π , escribimos:

```
Plot3D[Sin[x*y], {x, 0, Pi}, {y, 0, Pi}]
```

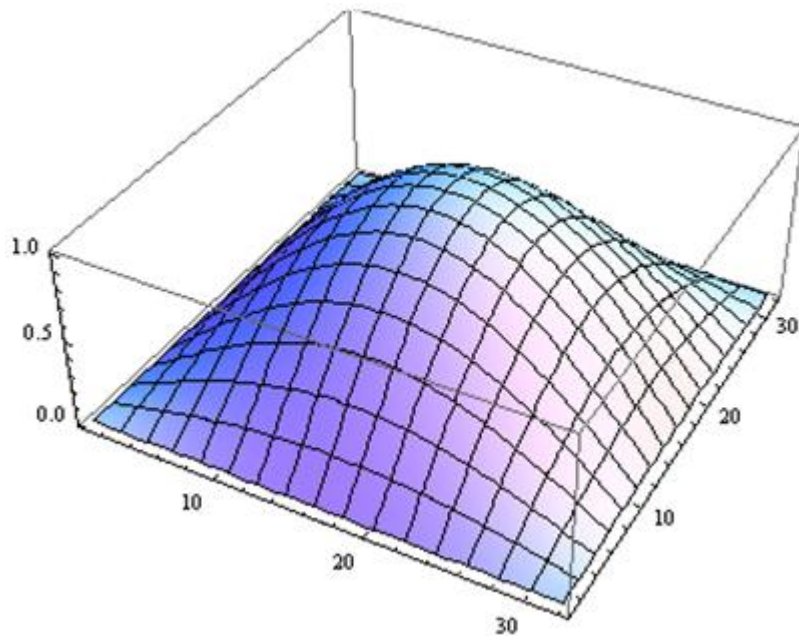


Para graficar datos tabulados tridimensionales empleamos "ListPlot3D":

```
ListPlot3D[datos]
```

Por ejemplo la siguiente orden grafica la lista que se genera con "Table" para la función " $\sin(x)*\cos(y)$ " para ángulos comprendidos entre 0 y π , con incrementos de 0.1:

```
ListPlot3D[Table[Sin[x]*Sin[y], {x, 0, Pi, 0.1}, {y, 0, Pi, 0.1}]]
```

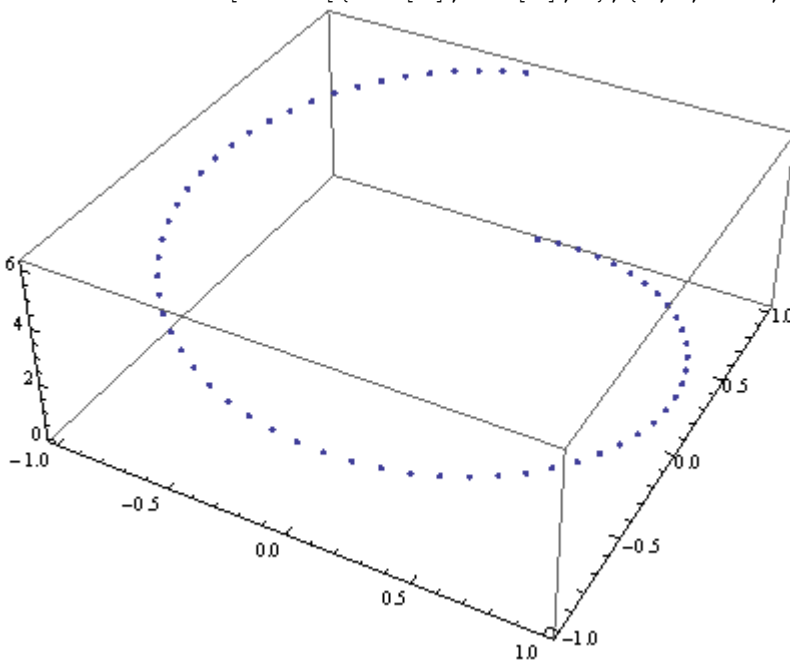


Podemos graficar también puntos tridimensionales específicos con "ListPointPlot3D":

```
ListPointPlot3D[{{x1,y1,z1},{x2,y2,z2},...}]
```

Por ejemplo la siguiente instrucción grafica los puntos generados con "Table" siendo los valores de "x", "y" y "z" el seno de "u", el coseno de "u" y "u" respectivamente, variando "u" desde 0 hasta 2π con incrementos de 0.1 en 0.1:

```
ListPointPlot3D[Table[{Sin[u],Cos[u],u},{u,0,2*Pi,0.1}]]
```

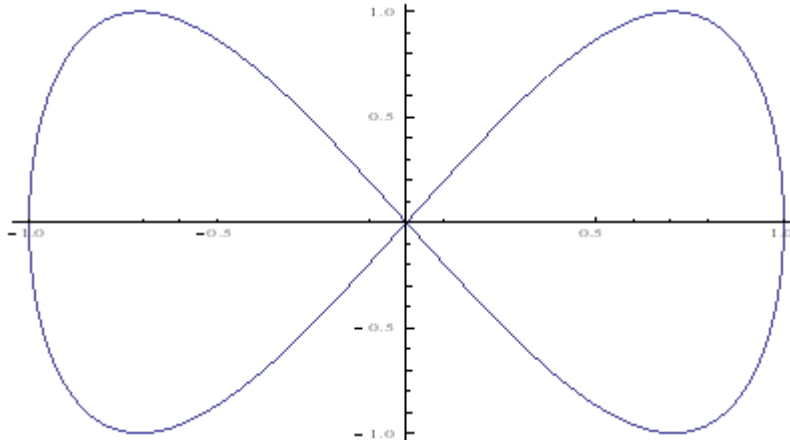


Adicionalmente en Mathematica podemos realizar gráficos paramétricos, que son aquellos en los cuales los valores de las variables independientes ("x", "y", "z") son función de un tercer parámetro (digamos "u"). Para realizar gráficos paramétricos bidimensionales empleamos "ParametricPlot":

```
ParametricPlot[{fx, fy}, {u, umin, umax}]
```

Por ejemplo con la siguiente instrucción se construye una gráfica paramétrica donde los valores de "x" corresponden al seno de "u" y los valores de "y" al seno de "2u", variando "u" desde 0 hasta 2π .

```
ParametricPlot[{Sin[u], Sin[2*u]}, {u, 0, 2*Pi}]
```



Y para gráficas paramétricas tridimensionales empleamos "ParametricPlot3D":

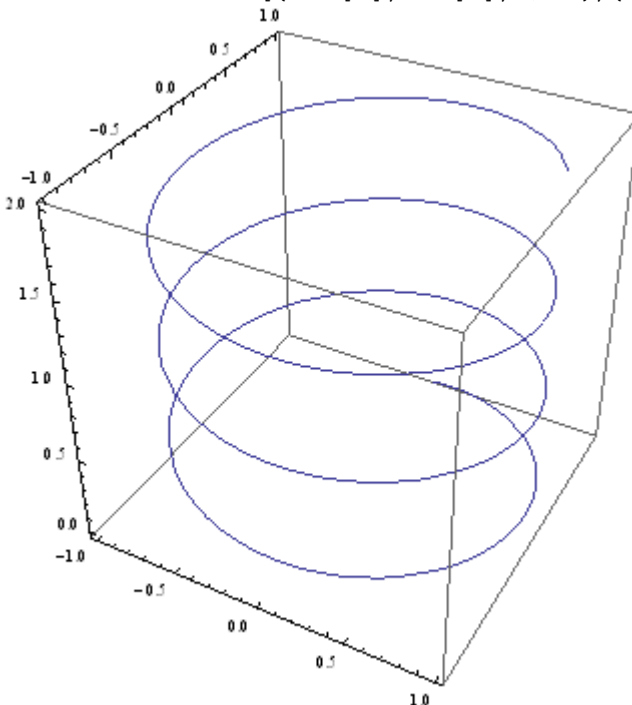
```
ParametricPlot3D[{fx, fy, fz}, {u, umin, umax}]
```

O en caso de depender de dos variables ("u" y "v"):

```
ParametricPlot3D[{fx, fy, fz}, {u, umin, umax}, {v, vmin, vmax}]
```

Así por ejemplo la siguiente instrucción dibuja una curva paramétrica tridimensional, donde la variable "x" es el seno de "u", la variable "y" es el coseno de "u" y la variable "z" es "u/10", variando "u" de 0 a 20:

```
ParametricPlot3D[{Sin[u], Cos[u], u/10}, {u, 0, 20}]
```

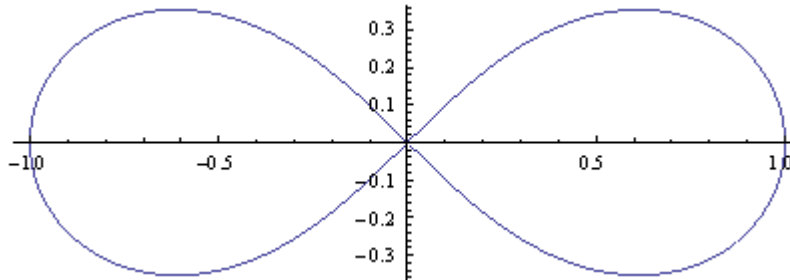


Otras funciones que resultan más prácticas para graficar curvas planas y notables son las siguientes:

`PolarPlot`[ecuación que devuelve el radio "r" en función del ángulo, {ángulo, valor inicial, valor final}]

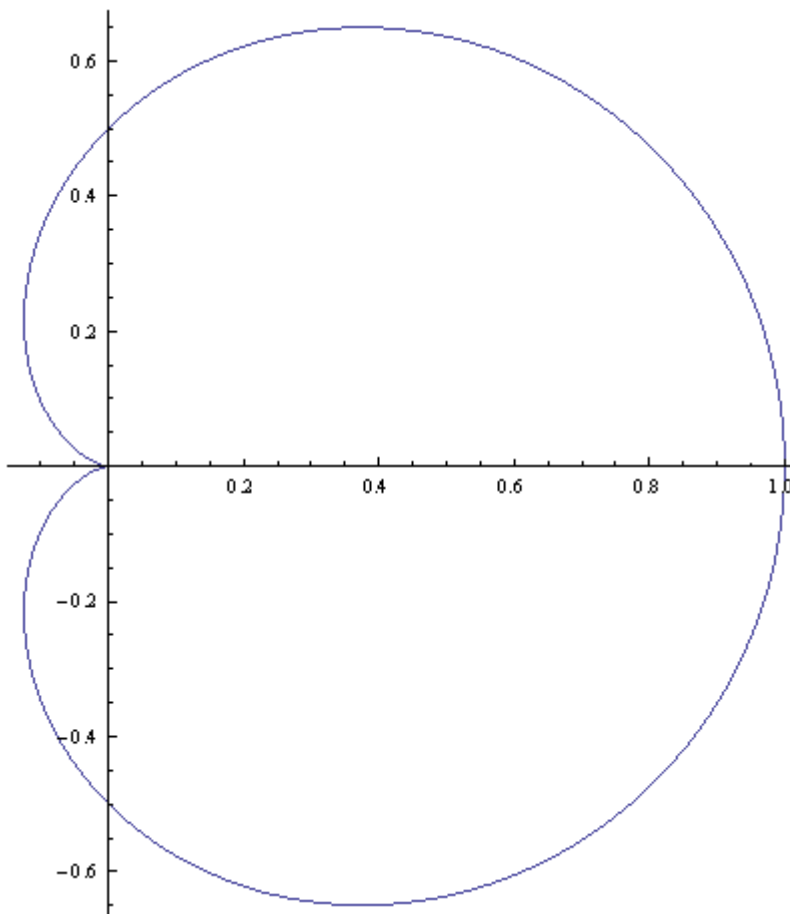
Por ejemplo, la gráfica de la curva "Lemniscata: $r^2 = a^2 \cos(\theta)$ ", puede ser realizada fácilmente con esta función:

```
PolarPlot[Sqrt[a^2*Cos[2*Θ]]/.a→1,{Θ,0,2*Pi}]
```



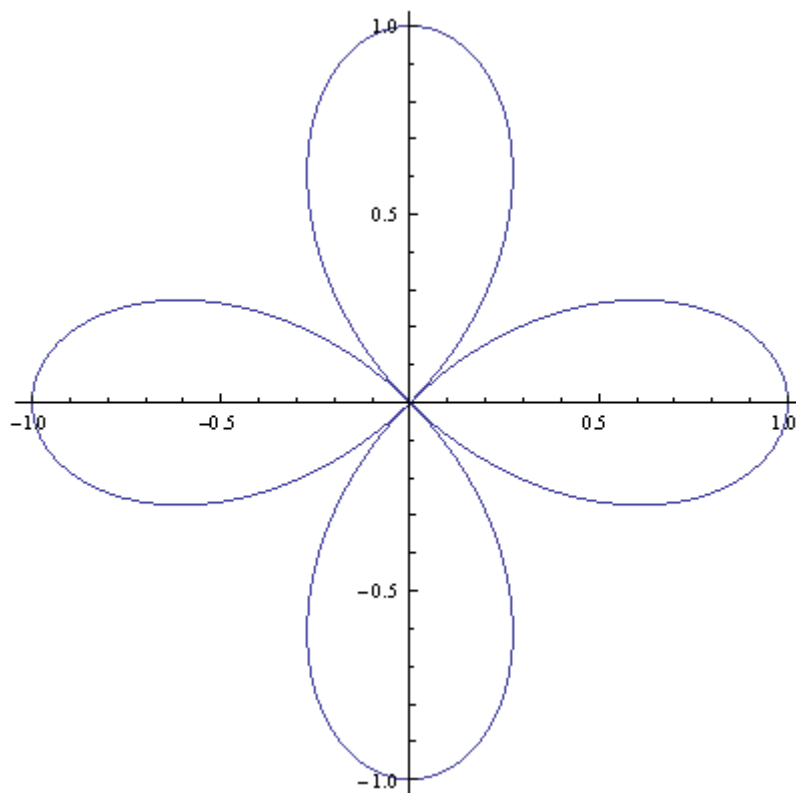
La "Cardioid: $r = a(1 + \cos(\theta))$ ":

```
PolarPlot[a*(1+Cos[Θ])/.a→0.5,{Θ,0,2*Pi}]
```



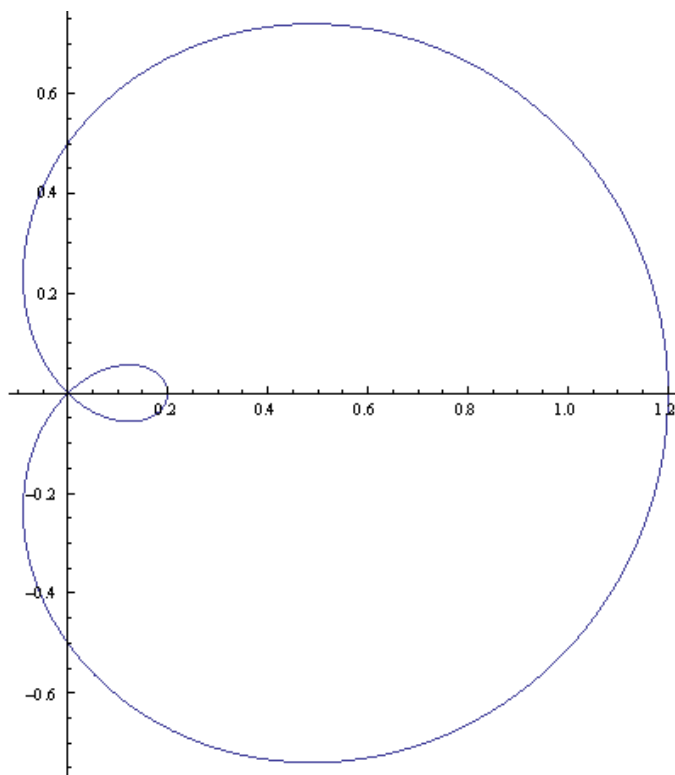
La "Rosa de cuatro pétalos: $r = a \cos(2\theta)$ ":

```
PolarPlot[a*Cos[2*Θ]/.a→1,{Θ,0,2*Pi}]
```



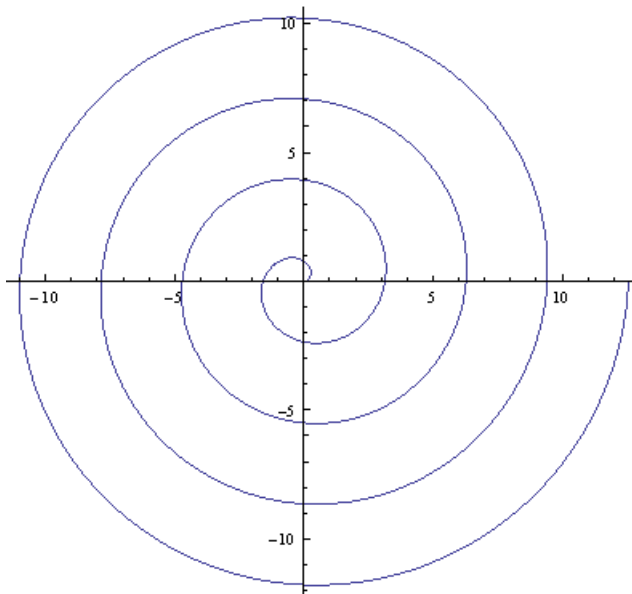
El "Caracol de Pascal: $r = b + a \cos(\theta)$ ":

`PolarPlot[b+a*Cos[θ]/.{a->0.7,b->0.5},{θ,0,2*Pi}]`



Y "La Espiral de Arquímedes: $r = a\theta$ ":

`PolarPlot[a*θ/.a->0.5,{θ,0,8*Pi}]`

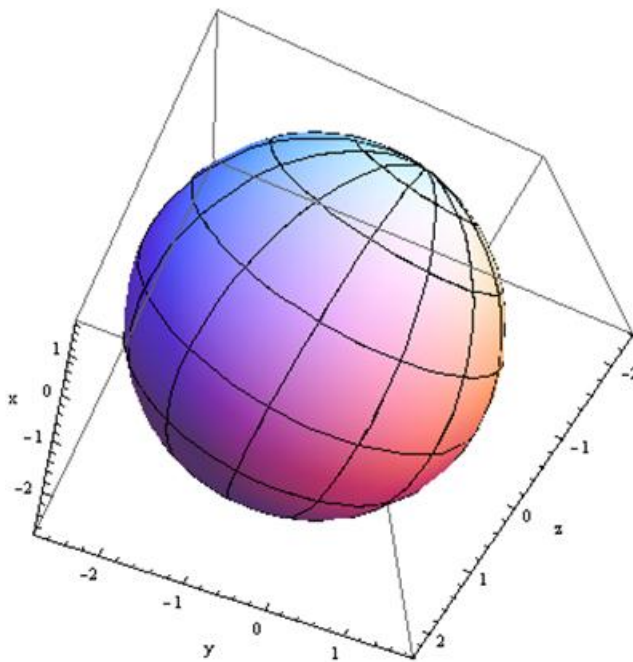


Igualmente, para figuras esféricas resulta más práctico emplear "SphericalPlot3D":

SphericalPlot3D[Ecuación que devuelve el radio "R" en función de el ángulo θ y ϕ , {ángulo θ , valor mínimo, valor máximo}, {ángulo ϕ , valor mínimo, valor máximo}]

Así la gráfica de la esfera: $R^2 = r^2 + r_0^2 - 2r_0r \sin(\theta) \sin(\theta_0) \cos(\phi - \phi_0)$ es:

SphericalPlot3D[Sqrt[r^2+r0^2-2*r0*r*Sin[θ]*Sin[θ0]*Cos[φ-φ0]]/.
{r0→2,θ0→Pi/4,φ0→Pi/4,r→0.5},{θ,0,2*Pi},{φ,0,2*Pi},AxesLabel→{"x","y","z"}]

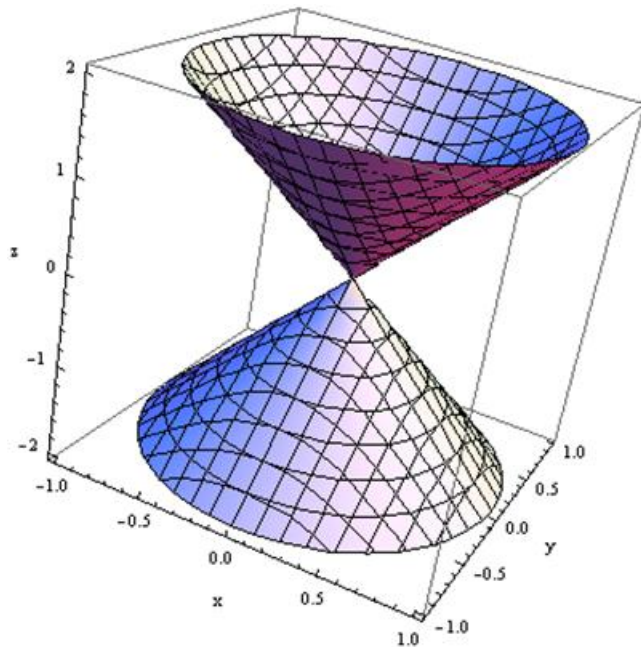


Y para graficar superficies notables, resulta más práctico emplear "ContourPlot3D":

ContourPlot3D[ecuación que depende de "x", "y" y "z",{ "x", valor mínimo, valor máximo},{ "y", valor mínimo, valor máximo},{ "z", valor mínimo, valor máximo}]

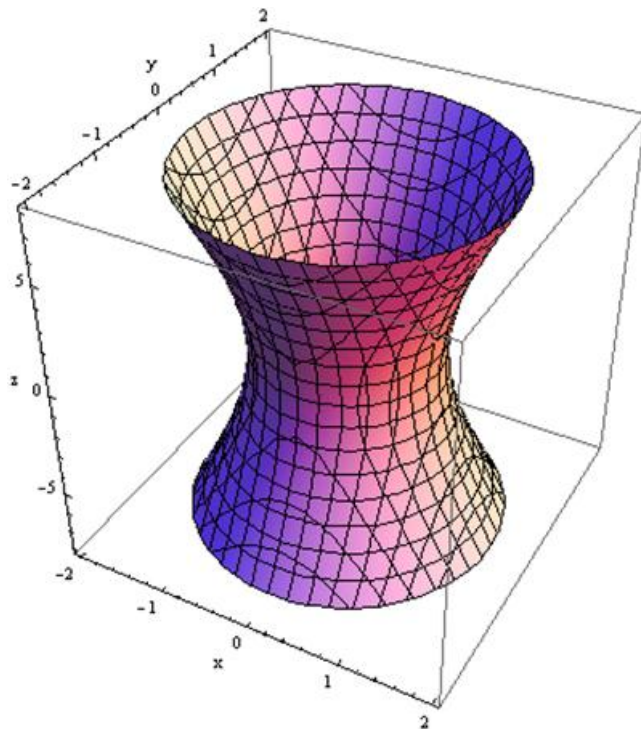
Así por ejemplo para graficar un cono elíptico con el eje en "z":
 $x^2/a^2 + y^2/b^2 = z^2/c^2$ escribimos:

```
ContourPlot3D[(x^2/a^2+y^2/b^2-z^2/c^2/.{a->1,b->1,c->2})==0,{x,-1,1},{y,-1,1},{z,-2,2},AxesLabel->{"x","y","z"}]
```



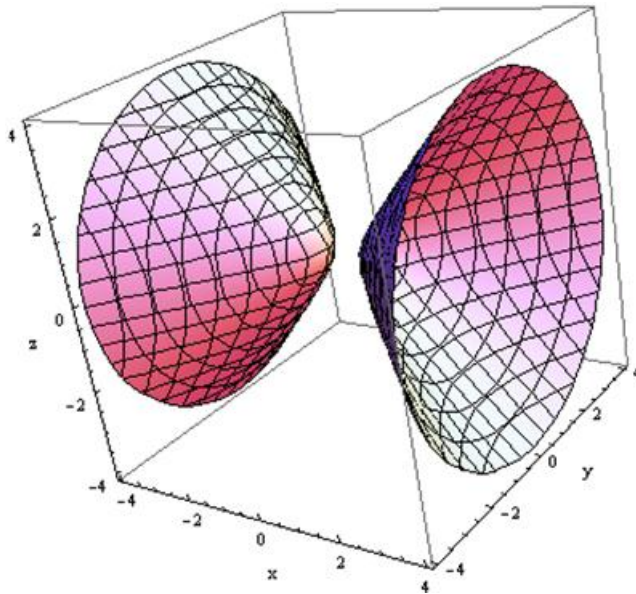
Para graficar el "hiperboloide de una sola hoja: $x^2/a^2 + y^2/b^2 - z^2/c^2 = 1$ ", escribimos:

```
ContourPlot3D[(x^2/a^2+y^2/b^2-z^2/c^2/.{a->1,b->1,c->6})==1,{x,-2,2},{y,-2,2},{z,-8,8},AxesLabel->{"x","y","z"}]
```



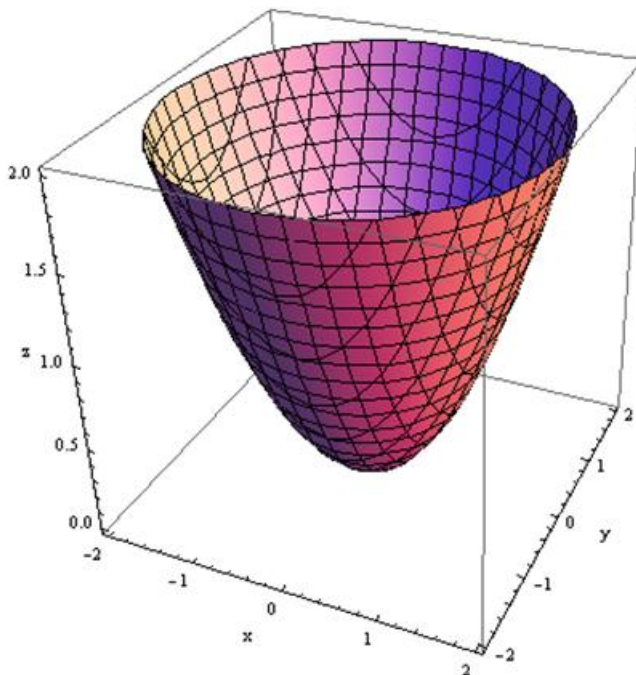
Para graficar el "hiperboloide de dos hojas: $x^2/a^2 - y^2/b^2 - z^2/c^2 = 1$ ", escribimos:

```
(x^2/a^2-y^2/b^2-z^2/c^2/.{a->0.5,b->0.5,c->0.5})==1,{x,-4,4},{y,-4,4},{z,-4,4},AxesLabel->{"x","y","z"}]
```



Y para graficar el "hiperboloide elíptico: $x^2/a^2 + y^2/b^2 = z/c$ " escribimos:

```
ContourPlot3D[(x^2/a^2+y^2/b^2/.{a->2,b->2})==(z/c/.c->2),{x,-2,2},{y,-2,2},{z,0,2},AxesLabel->{"x","y","z"}]
```



3.1.1. Ejemplos

1. Grafique la siguiente función entre 0 y 2, siendo el título de la gráfica la función, en el tipo de letra "Courier New", 16 puntos, con los estilos: negrita, cursiva, subrayado y en color café.

$$f(x) = \frac{e^{x^2} + 2x + 3 - \log(x)}{3x + 2}$$

Para dar formato a un texto (o expresión) se emplea la intrucción "Style" (Style[expresión,directivas]) siendo las directivas las opciones que permiten cambiar el estilo de la expresión o texto, tales como "FontFamily" (para elegir un tipo de letra), "FontWeight" para el estilo negrita o plano, "FontSlant" para el estilo cursiva o plano, "Underlined" para que el texto esté subrayado, "FontColor" para el color de la letra "Background" para el color del fondo del texto, existiendo además muchos estilos predefinidos tales como "Title", "Section", "Subsection", "SmallText", "Menu", "PopupMenu", "Button", etc., que muestran el texto o expresión en dichos formatos.

Las directivas pueden ser escritas en forma explícita tal como se muestra en la siguiente solución:

```
f[x_] := (Exp[x^2] + 2*x + 3 - Log[10, x]) / (3*x + 2)
Plot[f[x], {x, 0, 2}, PlotLabel -> Style[f[x],
  FontFamily -> "Courier New", FontSize -> 16,
  FontWeight -> Bold, FontSlant -> Italic,
  FontColor -> Brown]]
```

Y cuyo resultado se muestra en la siguiente página, o puede ser escrita también en forma implícita:

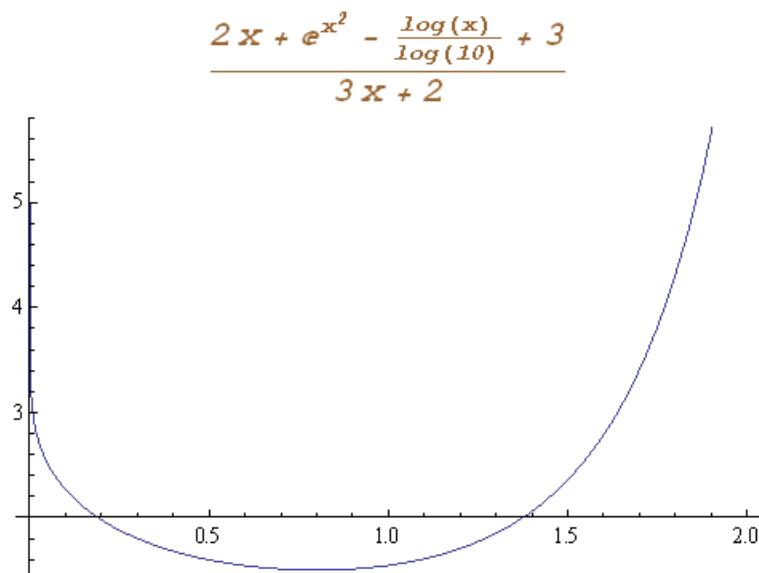
```
Plot[f[x], {x, 0, 2}, PlotLabel -> Style[f[x],
  FontFamily -> "CourierNew", 16, Bold,
  Italic, Brown]]
```

Donde como se puede ver se omite el nombre de la opción modificada (excepto en FontFamily), siendo el resultado por supuesto el mismo.

También se pueden agrupar dos o más directivas en una con la instrucción "Directive", aun cuando en este caso en particular ello no es necesario:

```
Plot[f[x], {x, 0, 2}, PlotLabel -> Style[f[x],
  Directive[FontFamily -> "Courier New",
  16, Bold, Italic, Brown]]]
```

Por supuesto en todos los casos se obtiene el mismo resultado:

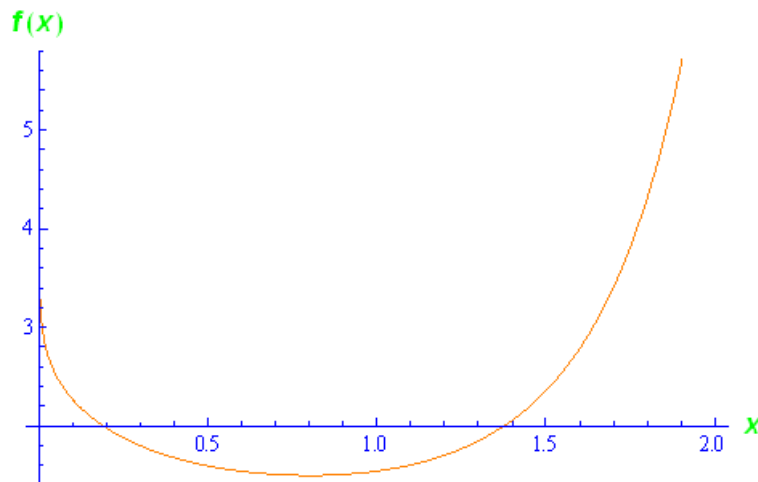


2. Grafique la función del ejercicio anterior, con el mismo título, pero añadiendo las etiquetas "x" y "f(x)" a los ejes con el tipo de letra "Arial" en color verde, 14 puntos, con los estilos negrita y cursiva, dibujando los ejes en color azul y la curva en color anaranjado.

La solución a este problema es similar a la del anterior ejercicio, sólo que en este caso se añaden además etiquetas a los ejes (formateando cada uno de ellos) y se cambia el color de los ejes (AxesStyle) y de la curva (PlotStyle):

```
Plot[f[x], {x, 0, 2}, PlotLabel -> Style[f[x],
  FontFamily -> "Courier New", FontSize -> 16,
  FontWeight -> Bold, FontSlant -> Italic,
  FontColor -> Brown],
  AxesLabel -> {Style["x", FontFamily -> "Arial",
    FontColor -> Green, FontSize -> 14,
    FontWeight -> Bold, FontSlant -> Italic],
    Style["f(x)", FontFamily -> "Arial",
      FontColor -> Green, FontSize -> 14,
      FontWeight -> Bold, FontSlant -> Italic]}},
  AxesStyle -> Blue, PlotStyle -> Orange]
```

$$\frac{2x + e^{x^2} - \frac{\log(x)}{\log(10)} + 3}{3x + 2}$$



3. Grafique la siguiente función entre 0.9 y 1.5, siendo el título del gráfico la función en color negro, 12 puntos itálica. La gráfica debe contar con un marco, siendo las etiquetas del marco "x" (superior e inferior) y "f(x)" (izquierda y derecha), en color rojo, con los estilos negrita, cursiva y subrayado, teniendo la gráfica líneas de división punteadas y rellenando el área que existe entre la curva y los ejes con un relleno de color verde y una opacidad del 40%.

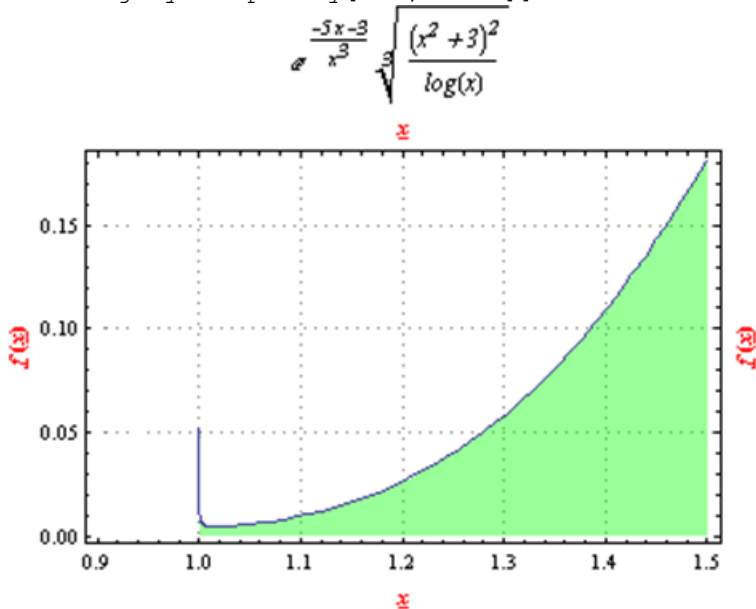
$$e^{\frac{3+5x}{x^3}} \sqrt[3]{\frac{(3+x^2)^2}{\ln(x)}}$$

Para que la gráfica tenga marco la propiedad "Frame" debe ser fijada en "True". Para establecer las etiquetas del marco se emplea la propiedad "FrameLabel" (de manera similar a las propiedades "PlotLabel" y "AxesLabel"). Para que la gráfica tenga líneas de división se fija la

propiedad "Gridlines" en "Automatic" (o en una lista con las posiciones en las cuales se quiere aparezcan las líneas). Para cambiar el estilo de las líneas se emplea la propiedad "GridLineStyle", siendo algunos valores usuales "Dotted" (puntos), "Dashed" (guiones), "DotDashed" (puntos y guiones). Para rellenar el área que existen entre la curva y los ejes se fija la propiedad "Filling" en "Axis" (otros posibles valores son "Bottom" (rellena bajo la curva), "Top" (rellena sobre la curva), "None" (sin relleno). Para cambiar el estilo del relleno se emplea la propiedad "FillingStyle", pudiendo cambiarse el color directamente o con instrucciones como "Opacity" (Opacity[opacidad,color]) "Hue" (Hue[tonalidad,saturación,brillo,opacidad], "RGBColor" [rojo,verde,azul,opacidad]) etc.

```
f[x_] := Exp[-(3+5*x)/x^3]*((3+x^2)^2/Log[x])^(1/3)
```

```
Plot[f[x], {x, 0.9, 1.5}, PlotLabel -> Style[
  f[x], Black, 12, Italic], Frame -> True,
  FrameLabel -> ({"f(x)", "f(x)"}, {"x", "x"}),
  /. x_?StringQ -> Style[x, Red, Bold,
    Italic, Underlined]],
  GridLines -> Automatic,
  GridLineStyle -> Dotted, Filling -> Axis,
  FillingStyle -> Opacity[0.4, Green]]
```



4. Vuelva a graficar la función del ejercicio anterior sin marco ni título, ubicando el origen de los ejes en $x=1.2$, $y=0.05$, trazando líneas de división discontinuas (guiones) en color verde con una opacidad del 30%, rellenando la parte superior de la curva con color azul y una opacidad del 50%, dibujando la curva con líneas discontinuas gruesas de color rojo y siendo las etiquetas de los ejes "x" y "f(x)" respectivamente.

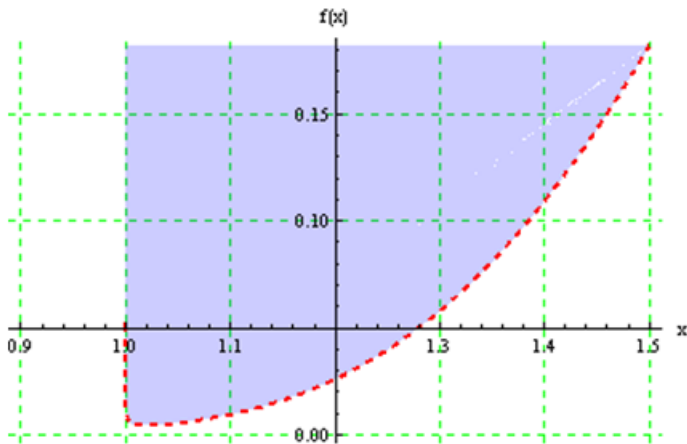
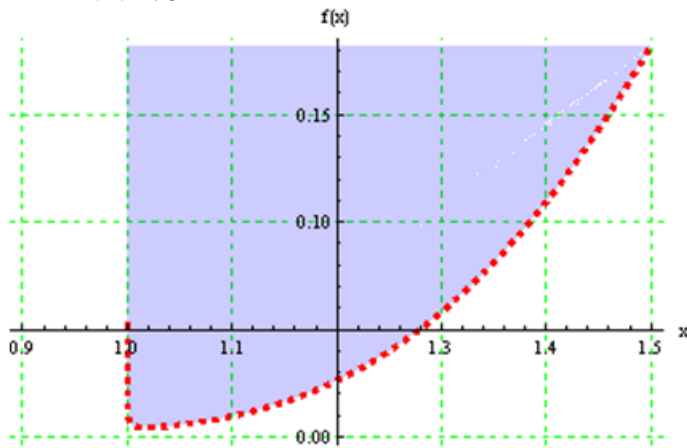
Para establecer el origen de los ejes se emplea la propiedad "AxesOrigin". Como ya vimos en el anterior ejemplo para cambiar el estilo de la curva se emplea "PlotStyle", siendo el valor para grueso "Thick" ("Thin" para delgado):

```
Plot[f[x], {x, 0.9, 1.5}, AxesOrigin ->
  {1.2, 0.05}, GridLines -> Automatic,
  GridLineStyle -> Directive[Dashed,
    Green], Filling -> Top, FillingStyle ->
  Opacity[0.2, Blue], PlotStyle -> Directive[
```

```
Red,Thick,Dashed],AxesLabel→{"x",
"f(x)"}]
```

El resultado se presenta en la siguiente página. En lugar de "Thick" se puede emplear también "Thickness" (Thickness[grosor], donde "grosor" es un valor comprendido entre 0 y 1, siendo 1 el ancho de la gráfica). De manera similar en lugar de "Dashed" se puede emplear "Dashing" (Dashing[largo], donde "largo" es un número comprendido entre 0 y 1, siendo 1 el ancho de la gráfica):

```
Plot[f[x],{x,0.9,1.5},AxesOrigin→
{1.2,0.05},GridLines→Automatic,
GridLineStyle→Directive[Dashing[0.01],
Green],Filling→Top,FillingStyle→
Opacity[0.2,Blue],PlotStyle→
Directive[Red,Thickness[0.01],
Dashing[0.01]],AxesLabel→{"x",
"f(x)"}]
```

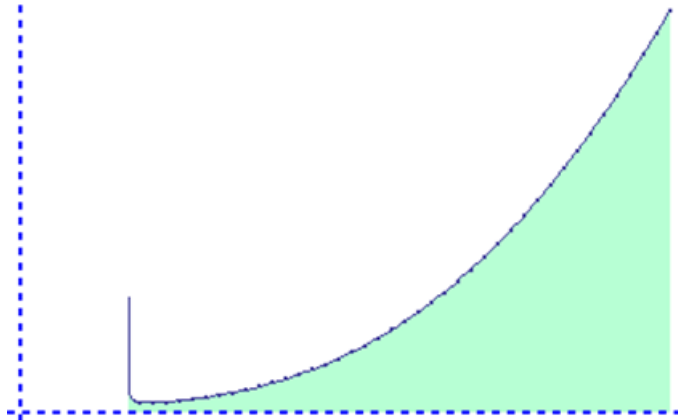


5. Grafique la función del ejercicio anterior rellenando el área bajo la curva con 0.5 de tonalidad, 0.4 de saturación, 1 de brillo y 0.7 de opacidad. Los ejes de la gráfica deben ser de color azul, discontinuos y gruesos, no deben tener marcas de división y los números deben ser de color rojo con los estilos negrita y cursiva. La curva debe ser dibujada de manera que se vean los puntos empleados para construirla.

Para establecer el color del relleno con las características dadas se emplea "Hue". Para que la gráfica no tenga marcas de división la opción "Ticks"

debe ser fijada en "None" (por defecto es "Automatic"). Para que los ejes sean dibujados con el estilo especificado se modifica la opción "AxesStyle". Para que se vean los puntos con los que se ha creado la curva se coloca la opción "Mesh" en "Full" (también se puede emplear "All" para mostrar todos los puntos, "Automatic" para mostrar automáticamente los punto más significativos o un número específico para mostrar dicho número de puntos):

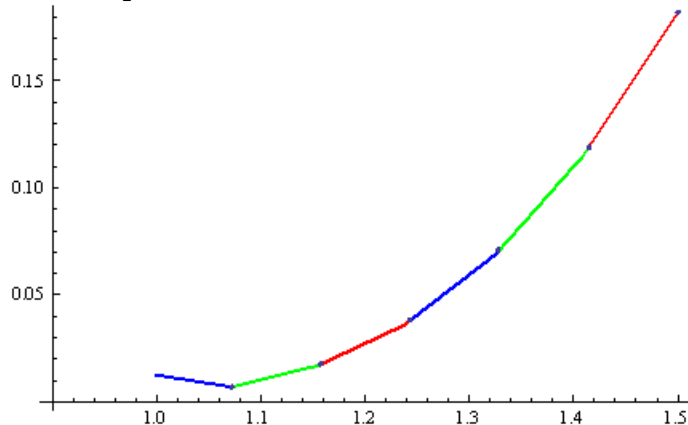
```
Plot[f[x], {x, 0.9, 1.5}, Filling -> Bottom,
  FillingStyle -> Hue[0.4, 0.4, 1, 0.7],
  Ticks -> None, AxesStyle -> Directive[Blue,
    Dashed, Thick], Mesh -> Full]
```



6. Grafique la función del ejercicio anterior empleando sólo 8 puntos para construir la curva. La gráfica debe mostrar los puntos empleados y las líneas que unen los mismos deben alternarse entre los colores verde, rojo y azul, siendo las líneas gruesas.

Para dibujar la curva empleando un determinado número de dígitos se fija la opción "PlotPoints" en dicho número y se fija la opción "MaxRecursion" en cero (porque de lo contrario, si la gráfica no resulta continua, Mathematica aplica recursivamente el número de puntos especificado hasta lograr una gráfica continua). Para que las líneas que unen los puntos se dibuje rotando un determinado número de colores se emplea la opción "MeshShading":

```
Plot[f[x], {x, 0.9, 1.5}, Mesh -> Full,
  PlotPoints -> 8, MaxRecursion -> 0,
  MeshShading -> {Green, Red, Blue},
  PlotStyle -> Thick]
```

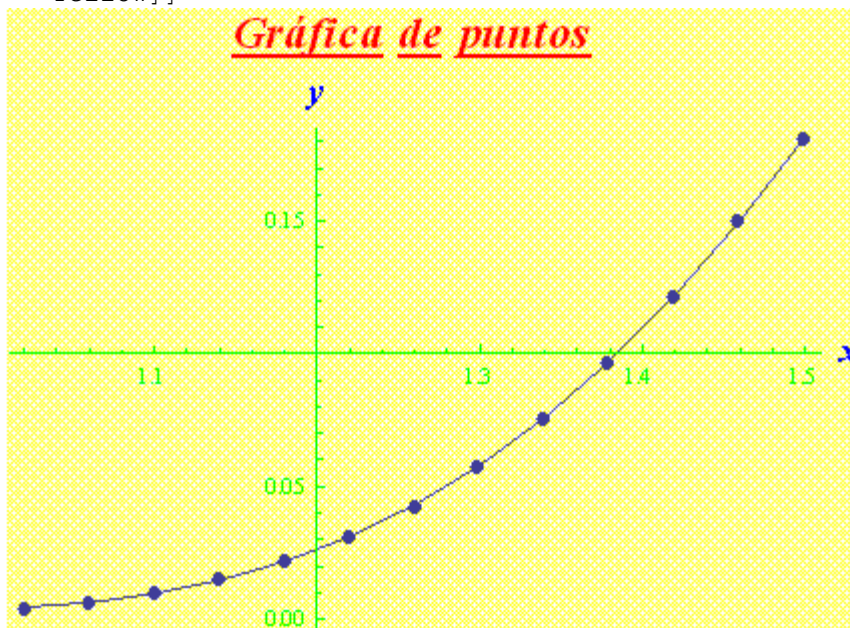


7. Grafique la función del ejercicio anterior generando 15 puntos igualmente espaciados con "Table". El título de la función debe ser "Gráfica de puntos" en 20 puntos, color rojo, negrita, cursiva y subrayado. Las

etiquetas de los ejes deben ser "x" y "y" en 14 puntos, color azul, negrita y cursiva. El origen de los ejes debe estar en "x=1.2", "y=0.1" y los ejes debe ser de color verde. El fondo de la gráfica debe ser de color amarillo, con una opacidad del 40%.

Para graficar puntos (en lugar de funciones) empleamos "ListPlot". Para que los puntos estén unidos por líneas, fijamos la opción "Joined" en "True". Para que se vean los puntos fijamos la opción "PlotMarkers" en "Automatic" (o en el carácter que querramos para los puntos). Para cambiar el fondo de la gráfica fijamos la opción "Background" en el color deseado.

```
ListPlot[Table[{x,f[x]},{x,0.9,1.5,(1.5-0.9)
/15}],Joined->True,PlotMarkers->
Automatic,PlotLabel->Style[
"Gráfica de puntos",20,Red,Bold,
Italic,Underlined],AxesLabel->({ "x",
"y"}/.x_?StringQ->Style[x,14,Blue,
Bold,Italic]),AxesOrigin->{1.2,0.1},
AxesStyle->Green,Background->Opacity[0.4,
Yellow]]
```



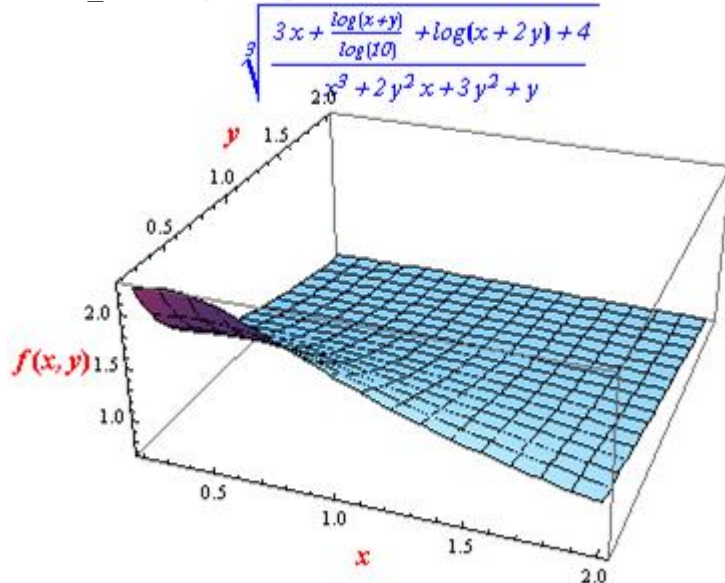
8. Grafique la siguiente función para "x=0.2->2", "y=0.2->2". El punto de vista de la gráfica debe ser: "x=1,y=-2.7,z=1.5". El título de la gráfica debe ser la función, en 12 puntos, azul, negrita y cursiva. Las etiqueta de los ejes deben ser "x", "y" y "f(x,y)" en 14 puntos, rojo, negrita y cursiva.

$$f(x,y) = \sqrt[3]{\frac{\log(x+y) + \ln(x+2y) + 3x + 4}{x^3 + 2xy^2 + 3y^2 + y}}$$

Para graficar funciones con tres dimensiones ("x","y","z") empleamos "Plot3D". El punto de vista de la gráfica se fija con "ViewPoint". Las otras opciones son esencialmente las mismas que con dos dimensiones.

```
f[x_,y_] := ((Log[10,x+y]+Log[x+2*y]+3*x+4)
/(x^3+2*x*y^2+3*y^2+y))^(1/3)
Plot3D[f[x,y],{x,0.2,2},{y,0.2,2},
ViewPoint->{1,-2.7,1.5},PlotLabel->
```

```
Style[f[x,y],12,Blue,Bold,Italic],
AxesLabel->({"x","y","f(x,y)"} /.
  x_?StringQ->Style[x,14,Red,Bold,Italic])]
```



9. Con la función del anterior ejercicio genere una tabla con 6 puntos igualmente espaciados para el eje "x" (entre 0.2 y 2), 6 puntos igualmente espaciados para el eje "y" (entre 0.2 y 2) y grafique la superficie formada por dichos puntos. La gráfica debe tener el título "puntos tridimensionales" en color verde, 18 puntos, negrita, cursiva y subrayado. Las etiquetas de los ejes deben ser "x", "y", "z" en 14 puntos, azul, negrita y cursiva.

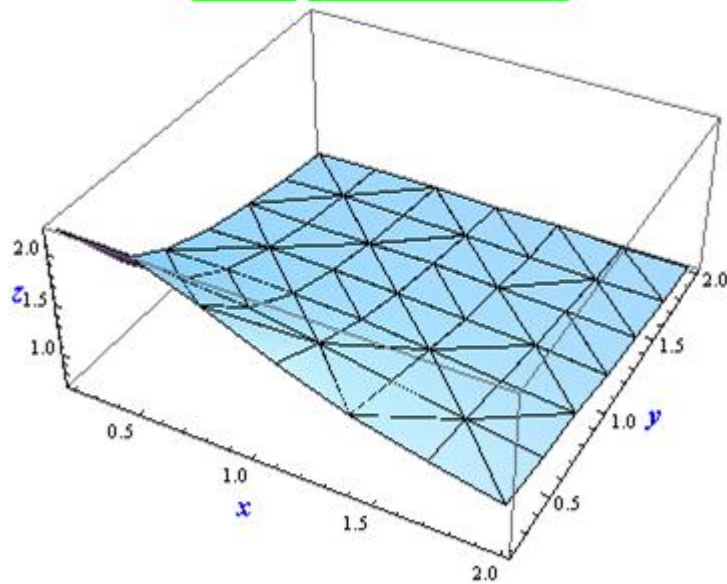
En este caso, además de "Table" se debe emplear "Flatten" para eliminar las filas (nivel 1) que por defecto genera "Table" cuando trabaja con dos variables, pues "ListPlot3D" requiere en realidad una lista con puntos. Para que dibuje sólo los puntos de la tabla se debe colocar la opción "Mesh" en "All" (La gráfica resultante se presenta en la siguiente página):

```
f[x_,y_] := ((Log[10,x+y]+Log[x+2*y]+3*x+4)
  /(x^3+2*x*y^2+3*y^2+y))^(1/3)
ListPlot3D[Flatten[Table[{x,y,f[x,y]},
  {x,0.2,2,0.3},{y,0.2,2,0.3}],1],
  Mesh->All,PlotLabel->Style[
  "Puntos Tridimensionales",Green,18,
  Bold,Italic,Underlined],AxesLabel->
  ({"x","y","z"} /. x_?StringQ->
    Style[x,14,Blue,Bold,Italic])]
```

10. Con la función del anterior ejercicio genere una tabla con 10 puntos igualmente espaciados para el eje "x" (entre 0.2 y 2), 10 puntos igualmente espaciados para el eje "y" (entre 0.2 y 2) y grafique dichos puntos. La gráfica debe tener el título "Puntos en el espacio" en color azul, 18 puntos, negrita, cursiva y subrayado. Las etiquetas de los ejes deben ser "x", "y", "z" en 16 puntos, rojo, negrita y cursiva y se debe rellenar el espacio debajo de los puntos.

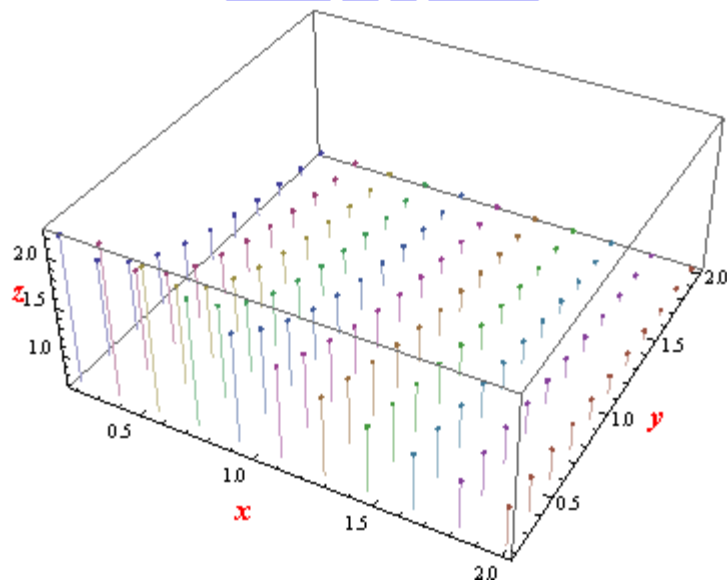
La única diferencia con el anterior ejercicio es que con "ListPointPlot3D" no se requiere la función "Flatten", pues "ListPointPlot3D" dibuja cada línea de puntos por separado.

Puntos Tridimensionales



```
f[x_,y_] := ((Log[10,x+y]+Log[x+2*y]+3*x+4)
/ (x^3+2*x*y^2+3*y^2+y))^(1/3)
ListPointPlot3D[Table[{x,y,f[x,y]},
{x,0.2,2,0.18},{y,0.2,2,0.18}],
PlotLabel->Style[
"Puntos en el espacio",Blue,18,
Bold,Italic,Underlined],AxesLabel->
({{"x","y","z"}/.x_?StringQ->
Style[x,16,Red,Bold,Italic]),
Filling->Bottom]
```

Puntos en el espacio



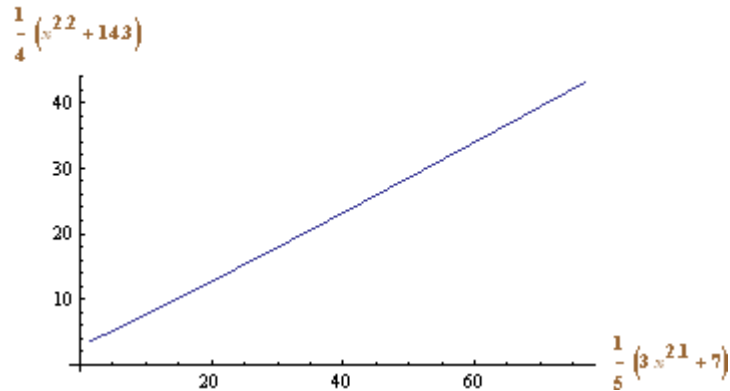
11. Elabore un gráfico paramétrico de las siguientes funciones:
 $f_x = (3x^{2.1} + 7)/5$; $f_y = (14.3 - x^{2.2})/4$, variando "x" entre 0 y 10. La gráfica debe tener el título "Gráfico paramétrico" en color anaranjado, 16 puntos, negrita y los ejes deben ser las funciones en color café, negrita.

```
fx[x_] := (3*x^2.1+7)/5;fy[x_] := (14.3+x^2.2)/4;
```



```
ParametricPlot[{fx[x],fy[x]},{x,0,10},
  PlotLabel->Style["Gráfico paramétrico",
    Orange,16,Bold],AxesLabel->
  ({fx[x],fy[x]}/.r:fx[x]|fy[x]->
    Style[r,Brown,Bold)])]
```

Gráfico paramétrico



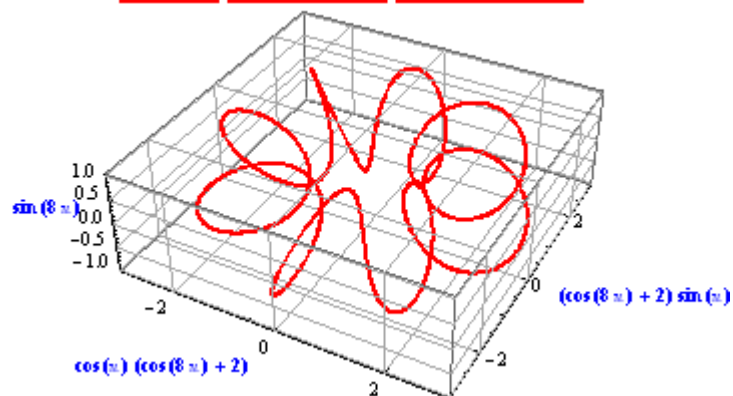
12. Elabore un gráfico paramétrico de las funciones: $f_x=(2+\cos(8u))\cos(u)$; $f_y=(2+\cos(8u))\sin(u)$; $f_z=\sin(8u)$, variando "u" entre 0 y 2π . La curva debe ser dibujada en rojo y con línea gruesa. Debe tener el título "Gráfico paramétrico tridimensional" en color rojo, 16 puntos, negrita, subrayado. La etiqueta de los ejes deben ser las funciones respectivas en azul y negrita. La gráfica debe tener líneas de división.

```
fx[u_]:= (2+Cos[8*u])*Cos[u];
fy[u_]:= (2+Cos[8*u])*Sin[u];
fz[u_]:= Sin[8*u]
```

En los gráficos tridimensionales las líneas de división se fijan con la opción "FaceGrids":

```
ParametricPlot3D[{fx[u],fy[u],fz[u]},
  {u,0,2*Pi},PlotStyle->Directive[Red,
    Thick],PlotLabel->Style[
  "Gráfico paramétrico tridimensional",
    Red,16,Bold,Underlined],
  AxesLabel->({fx[u],fy[u],fz[u]}/.
    r:fx[u]|fy[u]|fz[u]->Style[r,Blue,
      Bold]),FaceGrids->All]
```

Gráfico paramétrico tridimensional



8.1.2. Ejercicios

1. Grafique la siguiente función entre 0 y 2, siendo el título de la gráfica la función, en el tipo de letra "Courier New", con un tamaño de 16 puntos, con los estilos: negrita, cursiva, subrayado y en color azul.

$$f(x) = (52 + 3\sqrt{x} - 8x^{0.8})^{0.36} - x$$

2. Grafique la función del ejercicio anterior, con el mismo título, pero añadiendo las etiquetas "x" y "f(x)" a los ejes con el tipo de letra "Alba" en color azul, 14 puntos, con los estilos negrita y cursiva, dibujando los ejes en color verde y la curva en color anaranjado.
3. Grafique la siguiente función entre -2 y 0, siendo el título del gráfico la función en color azul, 12 puntos, itálica. La gráfica debe contar con un marco, siendo las etiquetas del marco "x" (superior e inferior) y "f(x)" (izquierda y derecha), en color naranja, con los estilos negrita, cursiva y subrayado, teniendo la gráfica líneas de división punteadas y rellenando el área que existe entre la curva y los ejes con un relleno de color cian y una opacidad del 40%.

$$f(x) = x^3 + 2x^2 + 3x + 4$$

4. Vuelva a graficar la función del ejercicio anterior sin marco ni título, ubicando el origen de los ejes en $x=-1$, $y=0$, trazando líneas de división discontinuas (guiones) en color azul con una opacidad del 30%, rellenando la parte superior de la curva con color verde y una opacidad del 50%, dibujando la curva con líneas discontinuas gruesas de color café y siendo las etiquetas de los ejes "x" y "f(x)" respectivamente.
5. Grafique la función del ejercicio anterior rellenando el área bajo la curva con 0.6 de tonalidad, 0.5 de saturación, 0.8 de brillo y 0.8 de opacidad. Los ejes de la gráfica deben ser de color verde, discontinuos y gruesos, no deben tener marcas de división y los números deben ser de color azul con los estilos negrita y cursiva. La curva debe ser dibujada de manera que se vean los puntos empleados para construirla.
6. Grafique la función del ejercicio anterior empleando sólo 6 puntos para construir la curva. La gráfica debe mostrar los puntos empleados y las líneas que unen los mismos deben alternarse entre los colores rojo, amarillo y azul, siendo las líneas gruesas.
7. Grafique la función del ejercicio anterior generando 10 puntos igualmente espaciados con "Table". El título de la función debe ser "Gráfica de puntos" en 20 puntos, color verde, negrita, cursiva y subrayado. Las etiquetas de los ejes deben ser "x" y "y" en 14 puntos, color rojo, negrita y cursiva. El origen de los ejes debe estar en " $x=-1.5$ ", " $y=0$ " y los ejes debe ser de color azul. El fondo de la gráfica debe ser de color cian, con una opacidad del 40%.
8. Grafique la siguiente función para " $x=0 \rightarrow 5$ ", " $y=0 \rightarrow 5$ ". El punto de vista de la gráfica debe ser: " $x=1.5, y=-3.0, z=2.5$ ". El título de la gráfica debe ser la función, en 12 puntos, verde, negrita y cursiva. Las etiqueta de los ejes deben ser "x", "y", "f(x,y)" en 14 puntos, azul, negrita y cursiva.

$$f(x,y) = \sqrt[4]{\frac{\ln(x+y) - x^2 + y^2}{e^{x+y} + x - y}}$$

9. Con la función del anterior ejercicio genere una tabla con 10 puntos igualmente espaciados para el eje "x" (entre 0 y 5), 10 puntos igual-

- mente espaciados para el eje "y" (entre 0 y 5) y grafique la superficie formada por dichos puntos. La gráfica debe tener el título "Superficie de puntos 3D" en color cian, 18 puntos, negrita, cursiva y subrayado. Las etiquetas de los ejes deben ser "x", "y", "z" en 14 puntos, naranja, negrita y cursiva.
10. Con la función del anterior ejercicio genere una tabla con 10 puntos igualmente espaciados para el eje "x" (entre 0 y 5), 10 puntos igualmente espaciados para el eje "y" (entre 0 y 5) y grafique dichos puntos. La gráfica debe tener el título "Puntos 3D" en color verde, 18 puntos, negrita, cursiva y subrayado. Las etiquetas de los ejes deben ser "x", "y", "z" en 16 puntos, azul, negrita y cursiva y se debe rellenar el espacio debajo de los puntos.
 11. Elabore un gráfico paramétrico de las funciones: $f_x = \sin(12u)\cos(u)$; $f_y = \sin(12u)\sin(u)$, variando "u" entre 0 y 2π . La gráfica debe tener el título "Gráfico paramétrico 2D" en color café, 16 puntos, negrita, subrayado y los ejes deben ser las funciones en color azul y negrita.
 12. Elabore un gráfico paramétrico de las funciones: $f_x = (2 + \cos(v))\cos(u)$; $f_y = (2 + \cos(v))\sin(u)$; $f_z = \sin(v)$, variando "u" y "v" entre 0 y 2π . La gráfica debe tener el título "Gráfico Paramétrico 3D" en color magenta, 16 puntos, negrita, subrayado. La etiqueta de los ejes deben ser las funciones respectivas en rojo y negrita. La gráfica debe tener líneas de división.

9. ERRORES EN LOS CÁLCULOS NUMÉRICOS

9.1. Introducción

Con frecuencia, en el área de la ingeniería, es necesario encontrar las soluciones numéricas de las relaciones matemáticas que se forman cuando se resuelve un problema o se realiza un diseño.

Frecuentemente también, estas relaciones toman formas cuyas soluciones analíticas son demasiado complicadas, o que de hecho no pueden ser resueltas analíticamente. En estos casos sólo es posible encontrar una solución recurriendo a los métodos numéricos, los cuales en contrapartida, tienen la desventaja de requerir una gran cantidad de cálculos, por lo que sólo son de utilidad práctica si se aplican con la ayuda de una computadora, y no únicamente por la velocidad con la que realizan los cálculos numéricos, sino por la precisión con la que trabajan, algo que es muy importante en el área de la ingeniería donde los resultados obtenidos se emplean en la práctica, traducándose en dimensiones, propiedades y objetos físicos concretos, tales como el espesor de una viga, la altura de una torre, la temperatura de un proceso, el espesor de un equipo, el número de equipos en serie, etc.

En esta materia estudiamos algunos de los métodos numéricos y el propósito de la misma es que, a su conclusión, estén capacitados para "encontrar las soluciones numéricas, de las relaciones matemáticas que se forman en la solución de problemas en el área de la ingeniería y de la ciencia en general, dando así soluciones concretas a problemas prácticos, con la seguridad, precisión y rapidez que son requeridas en ingeniería".

Usualmente para resolver una relación matemática, empleando métodos numéricos, sólo se requieren conocer los fundamentos básicos y no así las relaciones y reglas que derivan de ellas. Así por ejemplo para calcular el valor numérico de una derivada sólo se requiere conocer el concepto de derivada y no las reglas de derivación ni los cientos de relaciones que resultan de las mismas. Por esta razón, los métodos numéricos tienen un carácter más general, así por ejemplo una vez programado un método numérico para la derivación numérica de funciones, se puede emplear para derivar cualquier función, sin importar lo compleja que esta sea, algo que no ocurre con los métodos analíticos, donde existen diferentes procedimientos y reglas que se deben aplicar dependiendo de la forma que tenga la función.

Como ya se dijo, la principal desventaja de los métodos numéricos, y la razón por la cual no fueron empleados en la práctica hasta la invención de las computadoras, es la de requerir un elevado número de cálculos numéricos, generalmente repetitivos, pues tanto los métodos numéricos, como los fundamentos en los que se basan, existen ya hace siglos, varios de ellos datan por ejemplo del siglo XVI y algunos incluso de siglos antes, sin embargo, no fueron empleados intensivamente en la solución de problemas prácticos hasta mediados del siglo XX, donde las computadoras hicieron posible la ejecución y repetición de cálculos numéricos a grandes velocidades y sin los errores y cansancio que son propios del ser humano.

Actualmente las computadoras se han convertido en instrumentos esenciales para el progreso, educación, comunicación e incluso esparcimiento de la humanidad y están disponibles en prácticamente todas las instituciones, empresas y un número cada vez mayor de hogares, por lo que

el saber emplearlas, es hoy en día, casi tan importante como saber leer y escribir.

Debido a la masificación y la constante evolución de estos instrumentos, las diferentes profesiones las utilizan cada vez con mayor frecuencia y esto es algo más cierto aún en el área de la ingeniería, donde fueron empleadas desde un principio. Actualmente las diferentes ramas de la ingeniería cuentan con varias materias donde los cálculos, diseños, pruebas y simulaciones no podrían ser realizadas sin la ayuda de una computadora.

Lo anterior es válido en la presente materia, donde sería irracional resolver los problemas numéricos con lápiz y papel (aún con la ayuda de una calculadora no programable) cuando podemos emplear computadoras capaces de realizar las operaciones en una fracción del tiempo y sin cometer los errores que se cometen al pulsar las teclas de una calculadora y al anotar resultados intermedios en papel.

Sin embargo, y a pesar de la creencia errónea de que las computadoras no cometen errores, lo cierto es que sí los cometen (como veremos en este capítulo). Las causas de estos errores se encuentran en las limitaciones de tiempo y memoria que toda computadora tiene. Es necesario entonces conocer y tomar en cuenta estos errores a fin de minimizarlos e interpretar correctamente los resultados.

En este capítulo se estudiarán algunos de estos errores de manera que al finalizar el mismo estén capacitados para "identificar los errores que se presentan en los cálculos numéricos con el fin de minimizarlos o evitar sus efectos garantizando así la precisión de los resultados".

9.2. Cuantificación de los errores

Antes de pasar al estudio de los errores que se cometen en los cálculos numéricos se procederá previamente al estudio de las formas en que las que se cuantifican. En general existen dos formas en las que se cuantifica el error que son: el error absoluto y el error relativo.

El error absoluto se calcula con la siguiente expresión:

$$\text{Error absoluto} = |\text{valor verdadero} - \text{valor aproximado}|$$

Donde valor verdadero es el valor más exacto disponible. El problema con el error absoluto es que devuelve valores grandes cuando las cantidades que se compara son grandes. Por ejemplo si el valor verdadero es 45897.25 y el valor aproximado 45845.26, el error absoluto es:

$$|45897.25 - 45845.26| = 51.99$$

Que por su magnitud parece ser un error importante, aunque en realidad los dos valores son aproximadamente iguales y por lo tanto el error no es realmente muy grande. Por el contrario si el valor verdadero es 0.00002478 y el valor aproximado 0.00000578, el error absoluto es:

$$|0.00002478 - 0.00000578| = 0.000019$$

Que por su magnitud parece ser un error insignificante, no obstante los dos valores son totalmente diferentes por lo que en este caso el error si es muy grande. Podemos concluir entonces que el error absoluto no informa realmente sobre la magnitud del error cometido.

Debido a lo anterior se prefiere el error relativo que se calcula con la expresión:

$$\text{Error relativo} = \left| \frac{\text{valor verdadero} - \text{valor aproximado}}{\text{valor verdadero}} \right|$$

Este error es prácticamente independiente de la magnitud de los valores, así para los mismos datos considerados anteriormente en el primer ejemplo el error relativo es:

$$\left| \frac{45897.25 - 45845.26}{45897.25} \right| = 0.00113 \text{ (1.1\%)}$$

Para el primer caso y:

$$\left| \frac{0.00002478 - 0.00000578}{0.00002478} \right| = 0.76675 \text{ (76.7\%)}$$

Para el segundo. Como se puede observar estos valores nos informan realmente sobre la magnitud del error cometido, mostrando que en el primer caso el error es pequeño (1.1%) y en el segundo grande (76.7%).

Sin embargo, en varios de los métodos numéricos es más importante determinar la precisión con la que se calcula un resultado que el error cometido. Muchos de dichos métodos son iterativos y en los mismos los cálculos se repiten hasta que los dos últimos valores calculados son iguales en un determinado número de dígitos. Para determinar cuántos dígitos son iguales entre dos números dados se emplea la siguiente relación (la misma que ha sido empleada ya en el anterior capítulo):

$$\left| \frac{s_1}{s_2} - 1 \right| < 1 \times 10^{-n} \quad (1)$$

Donde " s_1 " y " s_2 " son los valores a comparar y " n " es el número de dígitos que deben ser iguales en estos dos valores. Cuando la relación se cumple (es decir cuando el resultado de la comparación es verdadero), los dos números son iguales en por lo menos " n " dígitos, aunque la condición no impide que sean iguales en un mayor número de dígitos. Para la mayoría de los cálculos que se realizan en el campo de la ingeniería suele ser suficiente que los dos últimos valores sean iguales en 6 dígitos.

Por ejemplo si los dos últimos resultados son: $s_1 = 12.345678943$ y $s_2 = 12.345678832$, aplicando la ecuación (1) se obtiene

$$\left| \frac{12.345678943}{12.345678832} - 1 \right| = 0.00000000899 = 8.99 \times 10^{-9}$$

Que es menor a 1×10^{-8} , por lo tanto estos valores son iguales en los 8 primeros dígitos (algo que se puede comprobar visualmente). Es importante hacer notar que esta expresión compara los números sin importar la potencia a la que estén elevadas, así con: $s_1 = 12.345678943 \times 10^{-20}$ y $s_2 = 12.345678832 \times 10^{-20}$, se obtiene el mismo resultado que en el caso anterior pues siguen siendo iguales en los primeros 8 dígitos. Esta es la relación que emplearemos con mayor frecuencia en la práctica.

9.2.1. Ejemplos

1. Elabore una función que calcule la raíz cuadrada de un número real positivo empleando la ecuación de Newton: $x_2 = (x_1 + n/x_1)/2$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 2 dígitos. Luego empleando la función creada calcule los errores absoluto y relativo cometidos al calcular las raíces cuadradas de 3.4, 6.7,

9.10, 12.43 y 50.342, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sqrt".

Primero creamos la función:

```
rcuad[n_]:=Switch[n,0,0,1,1,_,
Module[{x1=1.1,x2},While[True,
x2=(x1+n/x1)/2;
If[Abs[x1/x2-1]<10^-2,Return[x2]];
x1=x2]]];NumberQ[n]&&n≥0
```

Para que la función pueda trabajar con listas le asignamos el atributo "Listable":

```
Attributes[rcuad]=Listable;
```

Asignamos los valores cuyas raíces se quieren calcular a una lista:

```
l={3.4,6.7,9.10,12.43,50.342};
```

Calculamos las raíces cuadradas con "rcuad" y "Sqrt" mostrando los resultados con todos sus dígitos:

```
r1=rcuad[l];r1//InputForm
{1.8439702026490632, 2.588438381307977,
3.01665003463313, 3.525620520297118,
7.095209680671321}
r2=Sqrt[l];r2//InputForm
{1.8439088914585775, 2.588435821108957,
3.0166206257996713, 3.5256205127608387,
7.09520965158888}
```

Calculamos los errores absolutos de las raíces cuadradas de estos valores:

```
Abs[r2-r1]
{0.0000613112,2.5602×10-6,0.0000294088,7.53628×10-9,2.90824×10-8}
```

Calculamos los porcentajes de los errores relativos:

```
Abs[(r2-r1)/r2*100]
{0.00332507,0.0000989091,0.000974893,2.13758×10-7,4.09888×10-7}
```

Como se puede ver los errores son en todos los casos menores al número de dígitos especificado (2), pues como se dijo la condición garantiza que el resultado sea exacto por lo menos en el número de dígitos especificado, pero el resultado puede ser (como ha ocurrido en todos estos casos) más exacto que dicho número.

2. Elabore una función que calcule el seno de un ángulo en radianes empleando la serie de Taylor: $\text{sen}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 3 dígitos. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2°, 56.8°, 70.4°, 120.3° y 165.4°, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sin".

```
sen[x0_]:=If[x0==0,0,
Block[{x=FractionalPart[x0/(2*Pi)]*2*Pi,
```



```

    ter,xx,i=2,s1,s2},ter=x;xx=-x^2;
    For[s1=ter,True,s1=s2;i+=2,
      ter*=xx/(i*(i+1));s2=s1+ter;
      If[Abs[s1/s2-1]<10^-3,Return[s2]]]]]
Attributes[sen]=Listable;
l={43.2 °,56.8 °,70.4 °,120.3 °,165.4 °};
r1=sen[l];r1//InputForm
{0.6845468900325345, 0.8367617876801978,
 0.9420401010338478, 0.8633931267131064,
 0.25207533557541534}
r2=Sin[l];r2//InputForm
{0.6845471059286887, 0.8367643134589616,
 0.9420574527872968, 0.8633955506067718,
 0.25206935824311344}
Abs[r2-r1]
{2.15896×10^-7,2.52578×10^-6,0.0000173518,2.42389×10^-6,5.97733×10^-6}
Abs[(r2-r1)/r2*100]
{0.0000315385,0.000301851,0.0018419,0.00028074,0.0023713}

```

3. Elabore una función que calcule el exponente de un número real empleando la serie de Taylor: $e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \dots \infty$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 1 dígito. Luego calcule los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Exp".

```

exp[x0_]:=If[x0==0,1,Module[{x=Abs[x0],
  ter,xx,s1,s2,r,i=1},ter:=1;s1=ter;
  r=Catch[Do[ter*=x/i;s2=s1+ter;
    If[Abs[s1/s2-1]<10^-1,Throw[s2]];
    s1=s2;i++,{k,1000}]]];
  If[Negative[x0],1/r,r]]]
Attributes[exp]=Listable;
l={4.32,7.890,-10.23,40.3,-39.2};
r1=exp[l];r1//InputForm
{69.73711725074374, 2392.324350673517,
 0.00004258147573194809,
 2.0466887056074406*^17,
 1.450496350993588*^-17}
r2=Exp[l];r2//InputForm
{75.18862829202311, 2670.443920676805,
 0.00003607176974758541,
 3.177368756135782*^17,
 9.454886273886542*^-18}
Abs[r2-r1]
{5.45151,278.12,6.50971×10^-6,1.13068×10^17,5.05008×10^-18}
Abs[(r2-r1)/r2*100]
{7.25045,10.4147,18.0465,35.5854,53.4124}

```

4. Elabore una función que calcule la función "J" de Bessel de orden cero empleando la serie: $J_0(x) = 1 - x^2/2^2 + x^4/(2^2 4^2) - x^6/(2^2 4^2 6^2) + x^8/(2^2 4^2 6^2 8^2) - \dots \infty$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 3 dígitos. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

```

J0[x_]:=If[x==0,1,Block[{ter=1,xx=-x^2,
  i=2,s1,s2},s1=ter;

```

```

Label[1];ter*=xx/i^2;s2=s1+ter;
If[Abs[s1/s2-1]<10^-3,Goto[2]];
s1=s2;i+=2;Goto[1];Label[2];s2]]
Attributes[J0]=Listable;
l={3.45,6.445,9.123,20.43};
r1=J0[l];r1//InputForm
{-0.37273850884322385,
 0.25120075188512014,
 -0.11955036413608296,
 0.1243661321765517}
r2=BesselJ[0,l];r2//InputForm
{-0.37273486289473656,
 0.2512072062343679,
 -0.11954773510214133,
 0.12436111498875699}
Abs[r2-r1]
{3.64595×10-6,6.45435×10-6,2.62903×10-6,5.01719×10-6}
Abs[(r2-r1)/r2*100]
{0.000978161,0.00256933,0.00219915,0.00403437}

```

9.2.2. Ejercicios

1. Elabore una función que calcule la raíz cúbica de un número real positivo empleando la ecuación de Newton: $x_2 = (2x_1 + n/x_1^2)/3$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 1 dígito. Luego calcule los errores absoluto y relativo cometidos al calcular las raíces cuadradas de 3.4, 6.7, 9.10, 12.43 y 50.342, asumiendo que los valores verdaderos (o más exactos) son los calculados con $^{(1/3)}$.
2. Elabore una función que calcule el coseno de un ángulo en radianes empleando la serie de Taylor: $\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + \dots \infty$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 2 dígitos. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2°, 56.8°, 70.4°, 120.3° y 165.4°, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Cos".
3. Elabore una función que calcule el seno hiperbólico de un número real empleando la serie de Taylor: $\sinh(x) = x + x^3/3! + x^5/5! + x^7/7! + \dots \infty$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 3 dígitos. Luego calcule los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sinh".
4. Elabore una función que calcule la función "J" de Bessel de primer orden empleando la serie: $J_1(x) = x/2 - x^3/(2^2 \cdot 4) + x^5/(2^2 \cdot 4^2 \cdot 6) - x^7/(2^2 \cdot 4^2 \cdot 6^2 \cdot 8) + x^9/(2^4 \cdot 6^2 \cdot 8^2 \cdot 10) - \dots \infty$. Se debe garantizar que los dos últimos valores calculados sean iguales en por lo menos 2 dígitos. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

9.3. Errores de redondeo

El error de redondeo es el tipo de error que más frecuentemente se comete cuando se trabaja con computadoras. Este error se produce porque las computadoras trabajan con un número finito de dígitos (lo que se

denomina precisión), por lo tanto los resultados son redondeados a dicho número y se comete un error porque se desprecian los dígitos restantes. Por ejemplo si el número es 4.564232345235 y se trabaja con 4 dígitos, debe ser redondeado a 4.564, despreciándose la cantidad 0.000232345235, generándose en consecuencia un error de dicha magnitud. Como en métodos numéricos los cálculos se repiten cientos y hasta miles de veces, estos errores se van acumulando pudiendo llegar a ser de tal magnitud que los resultados sean del todo erróneos debido a ellos.

La mayoría de los lenguajes de programación tiene un conjunto limitado de tipos numéricos, cada con un determinado número de dígitos de precisión. Obviamente, mientras mayor es el número de dígitos de precisión, menor será el error de redondeo, aunque por otra parte, mayor será el tiempo consumido en los cálculos. Puesto que en la práctica es imposible trabajar con un número infinito de dígitos, pues se requerirían tiempo y memoria infinitos, es imposible evitar los errores de redondeo. Lo único que se puede hacer en la práctica es tratar de minimizarlos, trabajando con un número alto de dígitos, pero sin incrementar notoriamente la velocidad de los cálculos. Se debe cuidar además que el número de dígitos con los que se trabaje no sea menor al número de dígitos de precisión requeridos en el resultado.

Por defecto en Mathematica las operaciones con números reales se realizan con la precisión de máquina (\$MachinePrecision) que corresponde aproximadamente a 16 dígitos (15.9546). Sin embargo Mathematica permite fijar la precisión tanto de números simples como de números en expresiones. Para crear un número con una precisión arbitraria se escribe el número seguido de la tilde invertida ("`") y el número de dígitos de precisión, así por ejemplo:

```
3234.32343`40
```

Crea un número con 40 dígitos de precisión.

Se puede averiguar la precisión de un número con "Precisión", así por ejemplo la instrucción:

```
Precision[3234.32343`40]
```

Devuelve "40" (que es la precisión del número).

Para fijar la precisión con la que se llevan a cabo las operaciones se puede emplear "SetPrecision", así por ejemplo la siguiente instrucción:

```
SetPrecision[(3.45+6.534)/7.99,20]
```

Fija la precisión de todos los números de la expresión en 20 dígitos y luego realiza las operaciones con dichos números, sin embargo, y al igual que ocurre con la condición que se emplea para garantizar que dos números sean iguales en un determinado número de dígitos ($|s_1/s_2 - 1| < 10^{-n}$), cuando se fija la precisión, Mathematica garantiza que los resultados tengan por lo menos dicha precisión, pero no impide que tengan una precisión mayor, por lo que frecuentemente los resultados tienen una precisión mayor a la fijada. Para garantizar que los resultados tengan la precisión requerida, Mathematica debe guardar dígitos extras en las operaciones intermedias, el máximo número de dígitos extra que se puede guardar en dichas operaciones se fija en la variable "\$MaxExtraPrecision" que por defecto es igual a 50.

Se puede fijar también el máximo y mínimo número de dígitos de precisión permitidos cambiando el valor de las variables "\$MaxPrecision" y

"\$MinPrecision", que por defecto son $+\infty$ y $-\infty$. Así si se fijan estas variables en:

```
$MaxPrecision = $MinPrecision = 20
```

Todas las operaciones se llevan a cabo con 20 dígitos de precisión, pero cuando dicho número es menor a 16 dígitos (la precisión de máquina), Mathematica trabaja con la precisión de máquina aunque sólo muestra el resultado con la precisión fijada.

Algunas funciones como "N", permiten fijar la precisión (el número de dígitos) con la que se quiere obtener el resultado de una expresión y siempre que sea posible devuelven el resultado con dicha precisión, así por ejemplo la siguiente instrucción calcula la raíz cuadrada de 2 con 40 dígitos de precisión:

```
N[Sqrt[2],40]
1.414213562373095048801688724209698078570
```

Aplicando a este resultado la instrucción "Precisión" se obtiene 40 (que es la precisión requerida):

```
%//Precision
40.
```

Además de la precisión se puede fijar la exactitud, que en Mathematica se entiende como el número de dígitos después del punto. Puesto que dicho número varía constantemente con los cálculos, la exactitud (tal como la define Mathematica) es menos importante que la precisión. Para crear un número con una exactitud determinada se escribe el número seguido de dos tildes invertidas ``" y el número de dígitos de exactitud, por ejemplo 5.689``20 crea un número con 20 dígitos de exactitud (20 dígitos después del punto). Para fijar la exactitud de los números en una expresión se emplea "SetAccuracy" de manera similar a "SetPrecision" y para averiguar la exactitud de un número se emplea "Accuracy".

Debido a que con las funciones estándar de Mathematica no se puede trabajar realmente con un número de dígitos menor al de la precisión de máquina (\$MachinePrecision), elaboraremos una función para redondear números a un determinado número de dígitos, sin importar que el mismo sea menor a la precisión de máquina. Para ello haremos uso de las instrucciones "Round", "Sign", "Take" y "Part" (o "[[]]") ya estudiadas previamente, así como de las instrucciones "RealDigits", "IntegerDigits" y "FromDigits".

"RealDigits" devuelve una lista cuyos elementos son los dígitos del número real. Así por ejemplo al aplicar esta instrucción a Exp[25.2] se obtiene:

```
RealDigits[Exp[25.2]]
{{8,7,9,4,6,9,8,2,6,5,1,7,2,8,4,3},11}
```

Donde el primer elemento es una lista con los dígitos del resultado (16 pues como dijimos por defecto Mathematica trabaja con 16 dígitos) y el segundo elemento (11) es el número de dígitos en la parte entera. Podemos llevar a cabo la operación inversa, es decir convertir la lista en un número, con la instrucción "FromDigits":

```
FromDigits[%]
8794698265172843/100000
```

Que nos devuelve el resultado en forma de un número racional y que puede ser transformado en un número real con "N":

```
%//N//InputForm
```

```
8.794698265172842*^10
```

Donde se ha empleado "InputForm" para ver el resultado con todos sus dígitos.

De manera similar, "IntegerDigits" devuelve una lista con los dígitos de un número entero, por ejemplo aplicando esta instrucción al factorial de 15 obtenemos:

```
IntegerDigits[Factorial[15]]
{1,3,0,7,6,7,4,3,6,8,0,0,0}
```

Que puede volver a ser transformado en un número con "FromDigits":

```
%//FromDigits
1307674368000
```

La función que redondea un número a un determinado número de dígitos es la siguiente:

```
Redondear[x_Real,n_Integer]:=Module[
  {di=RealDigits[x],de,v1,v2,v3},
  de=di[[2]];v1={Take[di[[1]],n+1],n};
  v2=Round[FromDigits[v1]];
  v3=IntegerDigits[v2];
  N[Sign[x]*FromDigits[{v3,de}]]];
```

Así por ejemplo el exponente de 6.5 redondeado a 6 dígitos es:

```
Redondear[Exp[6.5],6]//InputForm
665.142
```

Mientras que el exponente sin redondear es:

```
Exp[6.5]//InputForm
665.1416330443618
```

9.3.1. Ejemplos

5. Elabore una función que calcule, con 4 dígitos, la raíz cuadrada de un número real positivo empleando la ecuación de Newton: $x_2 = (x_1 + n/x_1)/2$. Luego calcule los errores absoluto y relativo cometidos al calcular las raíces cuadradas de 3.4, 6.7, 9.10, 12.43 y 50.342, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sqrt".

Primero programamos la función redondeando el valore inicial y los resultados intermedios a 4 dígitos (con la función "Redondear" elaborada previamente):

```
Remove[rcuad]
rcuad[n0_]:=Which[n0==0,0,n0==1,1,True,
  Block[{n=Redondear[n0,4]},
    FixedPoint[Redondear[(#+n/#)/2.,4]&,
      1.1,50]]]
```

Hacemos que la función pueda aplicarse a listas y guardamos los valores a calcular (en forma de lista) en la variable "l":

```
Attributes[rcuad]=Listable;
l={3.4,6.7,9.12,12.43,50.342};
```

Calculamos la raíz cuadrada con los resultados con "rcuad" y "Sqrt":

```
r1=rcuad[l];r1//InputForm
```

```
{1.844, 2.588, 3.02, 3.526, 7.095}
r2=Sqrt[l];r2//InputForm
{1.8439088914585775, 2.588435821108957,
 3.0199337741082997, 3.5256205127608387,
 7.09520965158888}
```

Finalmente calculamos los errores absoluto y relativo (en forma de porcentaje):

```
Abs[r2-r1]
{0.0000911085,0.000435821,0.0000662259,0.000379487,0.000209652}
Abs[(r2-r1)/r2*100]
{0.00494105,0.0168372,0.00219296,0.0107637,0.00295483}
```

6. Elabore una función que calcule, trabajando con 3 dígitos de precisión, el seno de un ángulo en radianes empleando la serie de Taylor: $\text{sen}(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots^\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2° , 56.8° , 70.4° , 120.3° y 165.4° , asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sin".

```
Remove[sen]
sen[x0_]:=If[x0==0,0,
  Block[{x,ter,i=2.,xx},x=Redondear[
    FractionalPart[x0/(2*Pi)]*2*Pi,3];
    ter=x;xx=-Redondear[x^2,3];
    FixedPoint[Redondear [#+(ter*=
      xx/(i++*i++)),3]&,x,50]]]
Attributes[sen]=Listable;
l={43.2°,56.8°,70.4°,120.3°,165.4°};
r1=sen[l];r1//InputForm
{0.685, 0.837, 0.942, 0.862, 0.251}
r2=Sin[l];r2//InputForm
{0.6845471059286887, 0.8367643134589616,
 0.9420574527872968, 0.8633955506067718,
 0.25206935824311344}
Abs[r2-r1]
{0.000452894,0.000235687,0.0000574528,0.00139555,0.00106936}
Abs[(r2-r1)/r1*100]
{0.0661159,0.0281585,0.00609902,0.161897,0.426039}
```

7. Elabore una función que calcule, trabajando con 4 dígitos de precisión, el exponente de un número real empleando la serie de Taylor: $e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + \dots^\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Exp".

```
Remove[exp]
exp[x0_]:=If[x0==0,1,
  Module[{x=Redondear[Abs[x0],4],
    ter=1.,i=1.,r},r=FixedPoint[
    Redondear[#+(ter*=x/i++),4]&,
    1.,50];If[Positive[x0],r,
    Redondear[1/r,4]]]
Attributes[exp]=Listable;
l={4.32,7.890,-10.23,40.3,-39.2};
r1=exp[l];r1//InputForm
{75.18, 2671., 0.00003608, 2.992*^17,
 9.852*^-18}
```

```

r2=Exp[l];r2//InputForm
{75.18862829202311, 2670.443920676805,
 0.00003607176974758541,
 3.177368756135782*^17,
 9.454886273886542*^-18}
Abs[r2-r1]
{0.00862829,0.556079,8.23025*10^-9,1.85369*10^16,3.97114*10^-19}
Abs[(r2-r1)/r2*100]
{0.0114755,0.0208235,0.0228163,5.83403,4.20009}

```

8. Elabore una función que calcule, trabajando con 5 dígitos de precisión, la función "J" de Bessel de orden cero empleando la serie: $J_0(x)=1-x^2/2^2+x^4/(2^24^2)-x^6/(2^24^26^2)+x^8/(2^24^26^28^2)-\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

```

Remove[J0]
J0[x0_]:=Which[x0==0,1,True,Module[
  {x=Redondear[x0,5],ter=1.,xx,
  i=0.},xx=Redondear[-x^2,5];
  FixedPoint[Redondear[#+(ter*=
    xx/(i+=2)^2),5]&,1.,100]]]
Attributes[J0]=Listable;
l={3.45,6.445,9.123,11.43};
r1=J0[l];r1//InputForm
{-0.37272, 0.25094, -0.12131, -0.1049}
r2=BesselJ[0,l];r2//InputForm
{-0.37273486289473656,
 0.2512072062343679,
 -0.11954773510214133,
 -0.08351018994956715}
Abs[r2-r1]
{0.0000148629,0.000267206,0.00176226,0.0213898}
Abs[(r2-r1)/r2*100]
{0.00398752,0.106369,1.47411,25.6134}

```

Como se puede observar en estos ejemplos cuanto más grande (o más pequeño) es el resultado mayor es el error de redondeo.

9.3.2. Ejercicios

5. Elabore una función que calcule, trabajando con 3 dígitos de precisión, la raíz cúbica de un número entero real empleando la ecuación de Newton: $x_2=(2x_1+n/x_1^2)/3$. Luego calcule los errores absoluto y relativo cometidos al calcular las raíces cuadradas de 3.4, 6.7, 9.10, 12.43 y 50.342, asumiendo que los valores verdaderos (o más exactos) son los calculados con " $^(1/3)$ ".
6. Elabore una función que calcule, trabajando con 4 dígitos de precisión, el coseno de un ángulo en radianes empleando la serie de Taylor: $\cos(x)=1-x^2/2!+x^4/4!-x^6/6!+\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2° , 56.8° , 70.4° , 120.3° y 165.4° , asumiendo que los valores verdaderos (o más exactos) son los calculados con "Cos".
7. Elabore una función que calcule, trabajando con 5 dígitos de precisión, el seno hiperbólico de un número real con 7 dígitos de precisión empleando la serie de Taylor: $\sinh(x)=x+x^3/3!+x^5/5+x^7/7!+\dots\infty$. Luego calcu-

le los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sinh".

8. Elabore una función que calcule, trabajando con 3 dígitos de precisión, la función "J" de Bessel de primer orden empleando la serie: $J_1(x) = x/2 - x^3/(2^2 \cdot 4) + x^5/(2^2 \cdot 4^2 \cdot 6) - x^7/(2^2 \cdot 4^2 \cdot 6^2 \cdot 8) + x^9/(2^2 \cdot 4^2 \cdot 6^2 \cdot 8^2 \cdot 10) - \dots \infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

9.4. Errores de truncamiento

Este nombre se da a los errores que son causados por el método en si (el término se origina porque algunos de los métodos numéricos pueden ser comparados a una serie de Taylor truncada) y es el tipo de error que requiere más atención. Por ejemplo, podemos calcular aproximadamente el valor de e^x con la serie de Taylor truncada en su cuarto término:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!}$$

Sin embargo, sabemos que para calcular e^x realmente se requiere una serie infinita:

$$e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \sum_{n=4}^{\infty} \frac{x^n}{n!}$$

Por lo tanto al aproximar la serie sólo hasta el cuarto término se comete el error correspondiente a la sumatoria. El error se produce porque se ha truncado la serie, y nada tiene que ver con la precisión o exactitud de la computadora que se esté empleando. Para los métodos iterativos, que son la mayoría en los métodos numéricos, este error puede ser reducido repitiendo los cálculos un mayor número de veces, sin embargo, puesto que la vida es finita y el tiempo valioso, en la práctica debemos conformarnos sólo con aproximaciones a la solución analítica exacta.

Los procesos iterativos suelen repetirse hasta que los dos últimos valores son aproximadamente iguales, es decir hasta que los dos últimos valores son iguales en un determinado número de dígitos (empleando para ese fin la expresión estudiada en el acápite 2.2). Alternativamente, cuando se conoce el valor final, se puede repetir el proceso hasta que dicho valor es calculado con cierta tolerancia (error absoluto o relativo permitido).

9.4.1. Ejemplos

9. Elabore una función que calcule el seno de un ángulo en radianes empleando los primeros 4 términos de la serie de Taylor: $\sin(x) = x - x^3/3! + x^5/5! - x^7/7! + \dots \infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2°, 56.8°, 70.4°, 120.3° y 165.4°, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sin".

```
Remove[sen]
sen[x0_] := If[x0 == 0, 0, Module[
  {x = FractionalPart[x0/(2*Pi)]*2*Pi,
   ter, xx, i = 2}, ter = x; xx = -x^2;
```



```

Nest[#+(ter*=xx/(i++*i++))&,x,3]]]
Attributes[sen]=Listable;
l={43.2°,56.8°,70.4°,120.3°,165.4°};
r1=sen[l];r1//InputForm
{0.6845468900325345, 0.8367617876801978,
 0.9420401010338478, 0.8612953445015702,
 0.21646011332315318}
r2=Sin[l];r2//InputForm
{0.6845471059286887, 0.8367643134589616,
 0.9420574527872968, 0.8633955506067718,
 0.25206935824311344}
Abs[r2-r1]
{2.15896×10-7,2.52578×10-6,0.0000173518,0.00210021,0.0356092}
Abs[(r2-r1)/r2*100]
{0.0000315385,0.000301851,0.0018419,0.24325,14.1268}

```

10. Elabore una función que calcule el exponente de un número real empleando los primeros 5 términos de la serie de Taylor: $e^x=1+x+x^2/2!+x^3/3!+x^4/4!+\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Exp".

```

Remove[exp]
exp[x0_]:=If[x0==0,1,Block[{
  x=Abs[x0],ter=1,i=1,r},
  r=Nest[#+(ter*=x/i++)&,1,4];
  If[x0<0,1/r,r]]]
Attributes[exp]=Listable;
l={4.32,7.890,-10.23,40.3,-39.2};
r1=exp[l];r1//InputForm
{42.60001024, 283.34939293375,
 0.0014319825348692186,
 121664.66317083327,
 9.154672349223242*^-6}
r2=Exp[l];r2//InputForm
{75.18862829202311, 2670.443920676805,
 0.00003607176974758541,
 3.177368756135782*^17,
 9.454886273886542*^-18}
Abs[r2-r1]
{32.5886,2387.09,0.00139591,3.17737×1017,9.15467×10-6}
Abs[(r2-r1)/r2*100]
{43.3425,89.3894,3869.82,100.,9.68248×1013}

```

11. Elabore una función que calcule la función "J" de Bessel de orden cero empleando los primeros 4 dígitos de la serie: $J_0(x)=1-x^2/2^2+x^4/(2^24^2)-x^6/(2^24^26^2)+x^8/(2^24^26^28^2)-\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

```

Remove[J0]
J0[x_]:=If[x==0,1,
  Module[{xx=-x^2,ter=1,i=0,s=1},
    Do[s+=(ter*=xx/(i+=2)^2),{4}];s]]
Attributes[J0]=Listable;
l={3.45,6.445,9.123,93.43};
r1=J0[l];r1//InputForm
{-0.35779576063287494,

```

```

6.657490780168711, 163.61208307617107,
3.908820078340668*^10}
r2=BesselJ[0,1];r2//InputForm
{-0.37273486289473656,
0.2512072062343679,
-0.11954773510214133,
-0.0027828485431025574}
Abs[r2-r1]
{0.0149391,6.40628,163.732,3.90882*10^10}
Abs[(r1-r2)/r2*100]
{4.00797,2550.2,136959.,1.40461*10^15}

```

9.4.2. Ejercicios

9. Elabore una función que calcule el coseno de un ángulo en radianes empleando los primeros 5 términos de la serie de Taylor: $\cos(x)=1-x^2/2!+x^4/4!-x^6/6!+\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los senos de 43.2° , 56.8° , 70.4° , 120.3° y 165.4° , asumiendo que los valores verdaderos (o más exactos) son los calculados con "Cos".
10. Elabore una función que calcule el seno hiperbólico de un número real empleando los primeros 4 términos de la serie de Taylor: $\sinh(x)=x+x^3/3!+x^5/5!+x^7/7!+\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los exponentes de 4.32, 7.890, -10.23, 40.3 y -39.2, asumiendo que los valores verdaderos (o más exactos) son los calculados con "Sinh".
11. Elabore una función que calcule la función "J" de Bessel de primer orden empleando los primeros 5 términos la serie: $J_1(x)=x/2-x^3/(2^2*4)+x^5/(2^2*4^2*6)-x^7/(2^2*4^2*6^2*8)+x^9/(2^2*4^2*6^2*8^2*10)-\dots\infty$. Luego calcule los errores absoluto y relativo cometidos al calcular los valores de la función para 3.45, 6.445, 9.123 y 93.43, asumiendo que los valores verdaderos (o más exactos) son los calculados con "BesselJ".

9.5. Errores en los datos originales

En los problemas del mundo real, donde se modelan situaciones físicas mediante ecuaciones matemáticas, frecuentemente estas ecuaciones tienen coeficientes y/o parámetros que han sido determinados de manera empírica y cuya precisión se desconoce, inclusive el modelo matemático en si puede no reflejar adecuadamente la situación física que modela. En estos casos, los métodos numéricos no pueden hacer nada para minimizar o eliminar este tipo de errores, pero nosotros podemos llevar a cabo algunas pruebas para tratar de minimizar posibles errores y proporcionar resultados confiables.

En particular se recomienda llevar a cabo dos tipos de pruebas: a) Una prueba de validez y b) un análisis de sensibilidad.

La prueba de validez permite determinar cuán bien refleja el modelo matemático el mundo real, para ello se deben comparar resultados reales con los resultados obtenidos con el modelo matemático. Entonces se calculan los errores de los resultados obtenidos con el modelo y en base a los mismos se determina si el modelo es o no confiable. Por supuesto para ello es imprescindible contar con datos reales confiables.

El análisis de sensibilidad permite determinar cuán sensible es el modelo matemático con relación a determinados coeficientes o parámetros. Una vez determinados los parámetros o coeficientes que son más

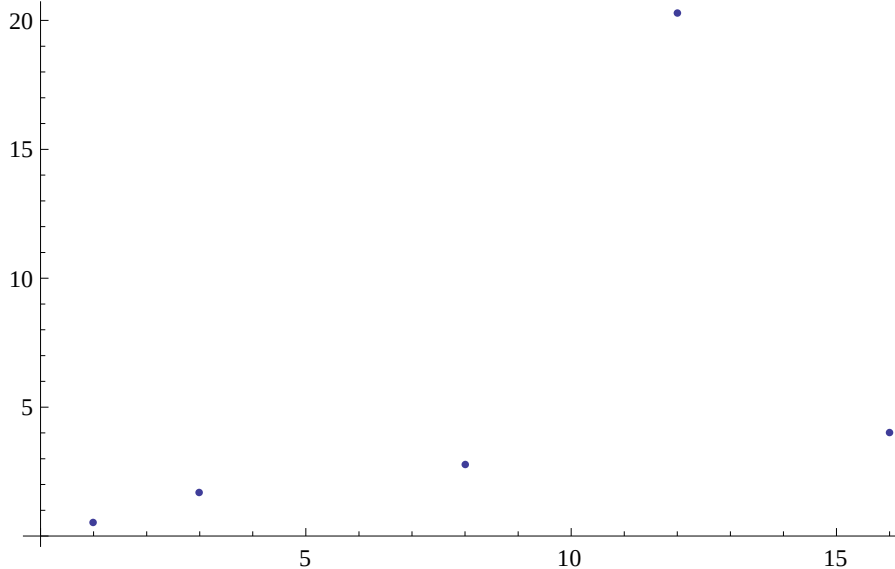
sensibles, se debe procurar obtener un valor más preciso de los mismos a fin de minimizar los errores. Para llevar a cabo una análisis de sensibilidad se calculan valores de la ecuación (o modelo matemático) para diferentes valores de la variable o variables independientes, luego, con los mismo valores de la variable independiente, se vuelve a evaluar la función pero cambiando cada uno de los parámetros en $\pm$ un determinado porcentaje (generalmente un 5%) y se calculan los errores relativos (o absolutos) de estos resultados con relación a los resultados obtenidos con la ecuación sin modificar, entonces, analizando estos errores se determina si la ecuación es o no sensible a uno u otro parámetro (mientras más grandes sean los errores producidos al variar uno de los parámetros, más sensible será la ecuación al mismo).

En otras ocasiones, los datos que se emplean en los métodos numéricos provienen de tablas experimentales y no es raro que alguno o algunos de dichos datos hayan sido mal medidos o registrados, por lo que es necesario hacer un análisis visual o gráfico de los mismos para determinar posibles inconsistencias y no emplear en el cálculo los datos incoherentes. Así por ejemplo analizando los siguientes datos podemos concluir que el valor de " $y=20.32$ " correspondiente al valor de " $x=12$ " es inconsistente, porque no sigue la secuencia lógica y presenta un valor muy elevado con relación a los otros valores de la tabla, razón por la cual no debería ser descartado.

x	1	3	8	<u>12</u>	16
y	0.56	1.68	2.76	<u>20.32</u>	4.02

Graficando estos datos podemos ver con mayor claridad dicha inconsistencia:

```
ListPlot[{{1,0.56},{3,1.68},{8,2.76},
{12,20.32},{16,4.02}}]
```



Como se puede observar, el punto correspondiente a (12,20.32) está muy alejado de los otros puntos.

Ejemplos

9.5.1. Ejemplos

12. Haga un análisis de sensibilidad y determine a cuál de los parámetros "a" o "b" es más sensible la ecuación $y=a+bx$, donde $a=3.246$ y $b=2.251$. Calcule valores de la función para $x=3, 10, 20, 50$ y 70 .

Primero creamos la función:

```
f[x_,a_,b_]:=a+ b*x
```

Entonces calculamos los valores para la ecuación normal (r1) y los valores cuando "a" incrementa en un 5% (r2):

```
{r1,r2}=Table[f[x,a,2.251],
  {a,3.246*{1,1.05}},
  {x,{3,10,20,50,70}}}
{{9.999,25.756,48.266,115.796,160.816},
 {10.1613,25.9183,48.4283,115.958,160.978}}
```

Con estos valores calculamos los porcentajes de error relativo:

```
Abs[(r1-r2)/r1*100]
{1.62316,0.630144,0.336262,0.14016,0.100923}
```

Ahora calculamos los valores cuando se hace variar "b" en +5% (r3):

```
r3=Table[f[x,3.246,2.251*1.05],
  {x,{3,10,20,50,70}}}
{10.3367,26.8815,50.517,121.424,168.695}
```

Y calculamos los porcentajes de error relativo:

```
Abs[(r1-r3)/r1*100]
{3.37684,4.36986,4.66374,4.85984,4.89908}
```

Como se puede observar, los errores son más grandes cuando se hace variar "b" que "a", por lo que podemos concluir que la ecuación es más sensible al parámetro "b".

13. Haga un análisis de sensibilidad y determine a cuál de los parámetros "a", "b" o "c" es más sensible la ecuación: $a+bx+c/x^2$, donde $a=2.65$, $b=7.65$ y $d=14.5$. Calcule valores de la función para $x=1, 5, 20, 40$ y 100 .

Programamos la función:

```
Remove[f]
f[a_,b_,c_,x_]:=a+b*x+c/x^2
```

Calculamos los valores con los parámetros normales (r1) y haciendo variar "a" en un 5% (r2):

```
{r1,r2}=Table[f[a,7.65,14.5,x],
  {a,2.65*{1,1.05}},
  {x,{1,5,20,40,100}}}
{{24.8,41.48,155.686,308.659,767.651},
 {24.9325,41.6125,155.819,308.792,767.784}}
```

Calculamos los valores haciendo variar "b" (r3):

```
r3=Table[f[2.65,7.65*0.95,14.5,x],
  {x,{1,5,20,40,100}}]
{24.4175,39.5675,148.036,293.359,729.401}
```

Calculamos los valores haciendo variar "c" (r4):

```
r4=Table[f[2.65,7.65,14.5*1.05,x],
  {x,{1,5,20,40,100}}]
{25.525,41.509,155.688,308.66,767.652}
```

Ahora calculamos los porcentajes de error relativo para "a":

```
Abs[(r1-r2)/r1*100]
{0.534274,0.319431,0.0851071,0.0429276,0.0172604}
```

Los porcentajes de error relativo para "b":

```
Abs[(r1-r3)/r1*100]
{1.54234,4.61066,4.91373,4.95693,4.98273}
```

Y los porcentajes de error relativo para "c":

```
Abs[(r1-r4)/r1*100]
{2.92339,0.0699132,0.0011642,0.000146804,9.44439×10-6}
```

Analizando estos resultados concluimos que la ecuación (o modelo) es más sensible al parámetro "b".

9.5.2. Ejercicios

12. Haga un análisis de sensibilidad y determine a cuál de los parámetros "a", "b" o "c" es más sensible la ecuación: $ax+bx^2+cx^3$, donde $a=0.45$, $b=0.86$ y $c=0.056$. Calcule valores de la función para $x = -10, -5, 0, 5, 10, 30, 60$ y 100 .
13. Haga un análisis de sensibilidad y determine a cuál de los parámetros "a", "b", "c" o "d" es más sensible la ecuación: $a+b/x+c/x^2+d/x^3$, donde $a=1.98$, $b=3.21$, $c=9.37$ y $d=16.83$. Calcule valores de la función para $x = 1, 3, 11, 19, 27, 51$ y 71 .

9.6. Errores humanos

Es frecuente que en la elaboración de programas se cometan errores, es también frecuente que se cometan errores al emplear un programa elaborado (principalmente en la introducción de datos).

Puesto que por naturaleza los seres humanos cometen errores, es imposible eliminarlos, pero si es posible reducirlos con la práctica. Mientras más se utilice una determinada herramienta, menores serán los errores que se cometan. Es por ello primordial que en el presente curso no sólo se lea el documento, sino se prueben todos los ejemplos y se resuelvan todos los ejercicios en computadora, y ello no sólo para reducir los errores humanos, sino porque la única forma de aprender a hacer algo es haciéndolo.

Por otra parte es importante también probar los programas elaborados, comparando los resultados con valores conocidos, no olvidando probar el programa para valores límite.

9.7. Propagación del error

Error propagado es aquel que aparece en las etapas sucesivas del proceso debido a la existencia de un error previo. Este error es adicional al error producido en esa etapa, es algo similar a cuando se inicia el cálculo con datos erróneos. Se presenta en aquellos métodos que emplean los resultados calculados en una etapa para calcular los resultados de la siguiente y los de estos para los de la siguiente y así sucesivamente. De esta manera los errores cometidos en la primera etapa se propagan a la siguiente y los de estos a la subsiguiente, incrementando el error en cada etapa.

Para reducir el error de propagación, se debe tratar de minimizar el error de cada una de las etapas antes de pasar a la siguiente.

9.8. Convergencia, divergencia y oscilación

La mayoría de los métodos numéricos son métodos iterativos (repetitivos), es decir son procesos que se repiten hasta que se cumple una determinada condición o se llega a una determinada precisión.

Cuando se trabaja con métodos iterativos se pueden presentar uno de los siguientes casos:

- a) Divergencia: Se dice que el método está divergiendo cuando los valores calculados se alejan cada vez más de la solución. Por ejemplo si la solución es 5.6473 y en cinco iteraciones sucesivas se obtienen los valores: 7.893, 24.324, 78.489, 145.544, 275.895, entonces se puede afirmar que el método está divergiendo porque en cada iteración se aleja cada vez más de la solución.
- b) Convergencia: Se dice que está convergiendo cuando los valores calculados en iteraciones sucesivas se acercan cada vez más a la solución. Por ejemplo, si la solución exacta es 3.643 y en cinco iteraciones sucesivas se obtienen los valores: 12.435, 8.948, 5.878, 4.123, 2.043, entonces podemos afirmar que el método está divergiendo porque en cada iteración se acerca cada vez más a la solución.
- c) Oscilación: Se dice que el método está oscilando cuando los valores calculados en iteraciones sucesivas se alternan entre dos valores. Así por ejemplo si la solución es 43.231 y en 6 iteraciones sucesivas se obtienen los valores: 15.432, 19.434, 15.431, 19.435, 15.430, 19.436, entonces se puede afirmar que el método está oscilando porque prácticamente se obtienen los mismos valores cada 2 iteraciones.

Cuando el método oscila, dichas oscilaciones pueden ser a su vez (a) puras, cuando la oscilación ocurre entre dos valores que no cambian, por ejemplo si las iteraciones son: 6.8, 9.2, 6.8, 9.2, etc. entonces se tienen una oscilación pura; b) divergente cuando oscila entre dos cantidades pero las mismo se alejan de la solución, así por ejemplo si la solución es 2.4 y las iteraciones son: 6.8, 9.2, 12.4, 16.3, 17.7, 30.3, etc. entonces se tiene una oscilación divergente y c) Convergente cuando oscila entre dos cantidades, pero las mismos se acercan hacia la solución, así por ejemplo si la solución es 1.43 y las iteraciones son: 9.8, 5.6, 6.2, 3.3, 2.2, 3.1, etc. entonces se tiene una oscilación convergente.

Cuando el método converge se encuentra la solución, cuando diverge no se encuentra la solución y generalmente se produce un error porque los valores se hacen excesivamente grandes. Cuando se tiene una oscila-

ción pura no se encuentra el resultado y el proceso se repite indefinidamente (ciclo infinito).

Si un método numérico generalmente converge, se dice que es estable, por el contrario si por lo general *diverge* u *oscila* entonces se dice que es inestable. Por supuesto los métodos más utilizados en la práctica son los métodos estables. Veremos en la práctica estos comportamientos en el siguiente capítulo (y en los posteriores), cuando comencemos a estudiar los métodos numéricos.

10. ECUACIONES ALGEBRAICAS CON UNA INCÓGNITA

En este capítulo estudiaremos algunos de los métodos que permiten resolver ecuaciones algebraicas con una incógnita, es decir cualquier función que depende de una sola variable.

Como veremos a medida que resolvamos ejemplos, una función no necesariamente es una expresión algebraica, sino que puede ser todo un sistema de ecuaciones (o expresiones algebraicas) pero que en última instancia dependen de una sola variable.

Al concluir el estudio de los diferentes métodos de este capítulo, deberán estar capacitados para encontrar las soluciones numéricas de ecuaciones algebraicas con una incógnita, explícitas e implícitas.

10.1. Diagramas de actividades

Antes de pasar a estudiar los métodos de este capítulo, explicaremos brevemente la notación gráfica que emplearemos para expresar los algoritmos de los diferentes métodos. Recordemos que los algoritmos describen la secuencia lógica de las acciones que deben llevarse a cabo para resolver un determinado problema. Existen muchas formas en las que se puede expresar un algoritmo, de ellas hemos escogido los *diagramas de actividades*, por ser la forma gráfica más empleada en la actualidad en el área de la informática.

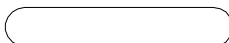
El inicio de un diagrama de actividades se representa con un círculo relleno:



El final del diagrama de actividades se representa con un círculo que contiene un círculo relleno en su interior:



Una acción (o sentencia) se representa con un rectángulo con sus bordes redondeados:



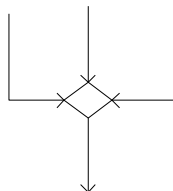
Un flujo, que es el símbolo empleado para unir los elementos del diagrama y señalar la dirección en que deben seguir las acciones, se representa por una flecha continua abierta:



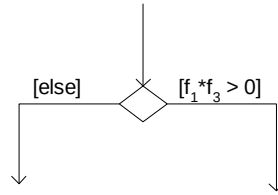
Una unión o bifurcación se representa por un pequeño rombo:



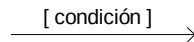
Cuando se emplea como unión llegan dos o más flujos y sale uno.



Cuando se emplea como bifurcación ingresan uno o más flujos y salen dos o más flujos acompañados de alguna condición.



Una condición, tal como se puede observar en la anterior figura, se representa como texto encerrado entre corchetes y siempre está asociado a algún flujo:



El texto de la condición puede ser una expresión matemática, lógica o literal que siempre debe devolver un valor lógico falso o verdadero. Para la condición por defecto se puede emplear *[else]* (caso contrario).

Una actividad representa un conjunto de acciones (o subprograma) que se detalla luego por separado empleando otro diagrama de actividades. Se representa como una acción con un icono de una acción llamando a otra en la parte inferior derecha:



Las notas se emplean para comentar y documentar los diagramas de actividades. Se representan como una hoja con una esquina doblada y asociada a cualquiera de los elementos de un diagrama de actividades mediante una línea discontinua:



Al elaborar un diagrama de actividades se debe tener presente que se trata de un lenguaje y como tal es necesario respetar su sintaxis (es decir los símbolos) pues cada uno de ellos tiene su propio significado y si se cambia, cambia también la lógica del algoritmo. Por ejemplo, para representar un flujo siempre se debe emplear la flecha continua abierta, no una flecha cerrada y rellena: $\longrightarrow$, una flecha cerrada no rellena: $\longrightarrow\triangleright$, o una flecha discontinua abierta: $\cdots\longrightarrow$ pues cada una de ellas tienen un significado muy diferente (mensaje, herencia y dependencia).

10.2. Resolución de funciones con una incógnita en Mathematica

Encontrar la solución numérica de una ecuación algebraica consiste en encontrar el valor de la variable independiente que, al ser reemplazado en la ecuación, cumple con la igualdad.

Para este fin lo práctico es igualar la ecuación a cero, de manera que la solución se encuentra cuando al reemplazar un valor de la variable independiente en la ecuación, el resultado es cero. Gráficamente la solución se encuentra cuando la función interseca el eje "x" (variable independiente). Por ejemplo para encontrar la solución gráfica de la siguiente ecuación:

$$x = \left(52 + 3\sqrt{x} - 8x^{0.8} \right)^{0.36} \quad (3.1)$$

La igualamos a cero:

$$f(x) = \left(52 + 3\sqrt{x} - 8x^{0.8} \right)^{0.36} - x = 0$$

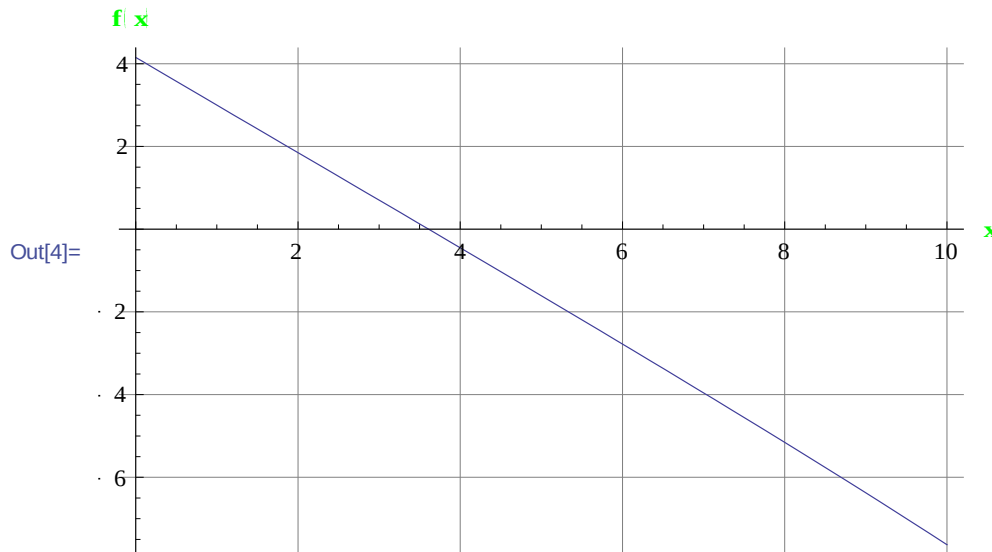
Programamos la función:

```
f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x
```

Y la graficamos probando algunos límites (en nuestro caso 0 y 10):

```
Plot[f[x], {x, 0, 10}, PlotLabel->Style[f[x],
  Blue, Bold], AxesLabel->({ "x", "f(x)" } /.
  r_String->Style[r, Bold, Green]),
  GridLines->Automatic]
```

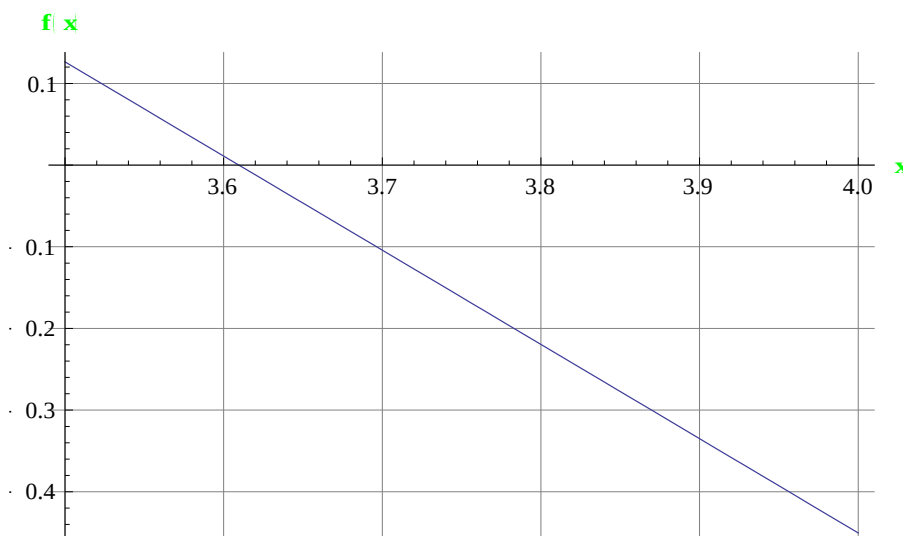
$$(-8x^{0.8} + 3\sqrt{x} + 52)^{0.36} - x$$



Como la curva interseca con el eje "x" aproximadamente en $x=3.6$, esa es la solución aproximada de la ecuación. Podemos ver con mayor precisión el resultado si volvemos a graficar la función entre 3.5 y 4:

```
Plot[f[x], {x, 3.5, 4}, PlotLabel->Style[f[x],
  Blue, Italic], AxesLabel->({ "x", "f(x)" } /.
  r_String->Style[r, Bold, Green]),
  GridLines->Automatic]
```

$$(-8x^{0.8} + 3\sqrt{x} + 52)^{0.36} - x$$



Ahora vemos que la solución es aproximadamente 3.61. Como se puede observar, gráficamente sólo se pueden encontrar soluciones aproximadas, para encontrar una solución más exacta, Mathematica cuenta con algunos comandos, siendo la más afín a los métodos numéricos que emplearemos en este capítulo "FindRoot", que en su forma básica tiene la siguiente sintaxis:

$$\text{FindRoot}[f(x), \{x, x_0\}]$$

Donde "x" es el nombre de la variable independiente, "x0" es el valor inicial asumido, "f(x)" es la función que depende de la variable independiente y puede estar también en forma de una expresión algebraica, aunque emplearemos casi exclusivamente funciones para este fin. "FindRoot" devuelve la solución en forma de una regla.

Así por ejemplo para encontrar la solución de la ecuación (3.1) con la precisión de máquina (16 dígitos), algo que es prácticamente imposible gráficamente, podemos escribir la expresión igualada a cero (empleando como valor inicial 1.1):

```
FindRoot[(52+3*Sqrt[x]-8*x^0.8)^0.36-x,
{x,1.1}]/InputForm
{x -> 3.60967088614005}
```

Pero también podemos escribir la ecuación original sin despejarla:

```
FindRoot[(52+3*Sqrt[x]-8*x^0.8)^0.36==x,
{x,1.1}]/InputForm
{x -> 3.60967088614005}
```

O emplear la función programada previamente (que es la forma que emplearemos con mayor frecuencia):

```
FindRoot[f[x],{x,1.1}]/InputForm
{x -> 3.60967088614005}
```

Cuando las ecuaciones son simples (como la del ejemplo), la primera y segunda forma son las más sencillas, pero no cuando la función tiene constantes y está compuesta por dos o más ecuaciones, además una vez que se programa la función puede ser empleada no sólo para encontrar la solución, sino también para verificar la exactitud del resultado, probar otras alternativas y graficarla (tal como se ha hecho en los ejemplos previos).

A más de la forma básica "FindRoot" tiene otras formas y opciones adicionales, así por ejemplo se puede buscar la solución a partir de dos valores iniciales: "FindRoot[f(x),{x,x₀,x₁}]", cuando se emplean dos valores iniciales "FindRoot" emplea otro método para encontrar la solución (el método de la secante que estudiaremos luego), así para resolver la ecuación 3.1 empleando dos valores iniciales (2.1 y 2.2) escribimos:

```
FindRoot[f[x],{x,2.1,2.2}]/InputForm
{x -> 3.6096708861400497}
```

Como se puede observar, este resultado difiere ligeramente del obtenido anteriormente (debido a que "FindRoot" emplea diferentes métodos en ambos casos). "FindRoot" también puede encontrar una solución si se le da un valor inicial y los límites mínimo y máximo entre los cuales debe buscar la solución: "FindRoot[f(x),{x₀,x_{min},x_{max}}]", por ejemplo con la siguiente instrucción volvemos a resolver la ecuación 3.1, comenzando con un valor inicial igual a 3.1 y buscando la solución entre 2 y 5.

```
FindRoot[f[x],{x,3.1,2,5}]/InputForm
{x -> 3.60967088614005}
```

Además "FindRoot" tiene varias opciones, de las cuales las más usuales son: "PrecisionGoal", para establecer la precisión del resultado (que por

defecto es la mitad de `WorkingPrecision`), `"AccuracyGoal"` para fijar la exactitud del resultado, lo que vendría a ser el error absoluto del resultado (que por defecto también es igual a la mitad de `WorkingPrecision`), `"WorkingPrecision"` para fijar el número de dígitos con los cuales se realizarán las operaciones (por defecto es igual a `$MachinePrecision`, aproximadamente 16 dígitos), `"Method"` para elegir el método que empleará "Mathematica" para resolver el problema (por defecto es `"Automatic"` y se puede elegir entre `"Newton"`, `"Secant"` y `"Brent"`), `"MaxIterations"` para fijar el máximo número de iteraciones permitidas (por defecto el máximo de iteraciones es 100), `"StepMonitor"` para evaluar una expresión cada vez que `"FindRoot"` lleva a cabo una iteración (por defecto es `"None"`), `"EvaluationMonitor"` para evaluar una expresión que cada vez que `"FindRoot"` calcula el valor de la función o ecuación (por defecto también es `"None"`).

Cuando se fija `"PrecisionGoal"` o `"AccuracyGoal"` se debe verificar que el valor de `"WorkingPrecision"` sea por lo menos igual a estos valores pues de lo contrario Mathematica no puede devolver el resultado con la precisión o exactitud requerida.

Por ejemplo con la siguiente instrucción se encuentra la solución de la ecuación 3.2 con 25 dígitos de precisión, sin tomar en cuenta la exactitud:

```
FindRoot[SetPrecision[f[x],25],{x,1.1},
  PrecisionGoal->25,AccuracyGoal->Infinity,
  WorkingPrecision->25]//InputForm
{x -> 3.60967088614004992408302314366772063629`25.}
```

Como se puede ver para que no se tome en cuenta la exactitud, se fija `"AccuracyGoal"` en `"Infinity"` (infinito), de manera que la exactitud nunca sea alcanzada. Observe que en este caso a más de fijar `"WorkingPrecision"` en 25 es necesario fijar también la precisión de la expresión en 25 (con `"SetPrecision"`), pues por defecto tiene la precisión de máquina (16 dígitos) y de no incrementarse su precisión a 25, se generaría un error como se muestra a continuación (aun cuando el resultado es calculado con los dígitos requeridos):

```
FindRoot[f[x],{x,1.1},
  PrecisionGoal->25,AccuracyGoal->Infinity,
  WorkingPrecision->25]//InputForm
FindRoot::precw: The precision of the argument function ((52+3 $\sqrt{x}$ -8 x0.8)0.36-x) is less than WorkingPrecision (25.`). >>
{x -> 3.60967088614004992408302314366772063629`25.}
```

En la siguiente instrucción, al trabajar con 40 dígitos, aseguramos que el resultado tenga 20 dígitos de precisión o, si se alcanza primero, 20 dígitos de exactitud (pues por defecto, y como se dijo, ambas opciones son iguales a la mitad de `"WorkingPrecision"`):

```
FindRoot[SetPrecision[f[x],40],{x,1.1},WorkingPrecision->40]//InputForm
{x -> 3.6096708861400499240830231436677168986023370481421412393927`40.}
```

Calculando el error absoluto de este resultado (tomando en cuenta que el resultado exacto es cero) obtenemos:

```
Abs[(SetPrecision[f[x],40]/.%) - 0]
0.×10-39
```

Vemos entonces que se ha alcanzado una exactitud mayor a los 20 dígitos. No podemos calcular directamente el error relativo, pues al ser el resultado exacto 0, se produciría un error por división entre cero, pero sí podemos despejar una de las constantes, por ejemplo 52, y trabajar con la ecuación despejada para calcular el error relativo:

```
Abs[((SetPrecision[x^(1/0.36)+8*x^0.8-3*Sqrt[x],40]/.%%)-52)/52*100]
1.75871576586577421593791×10-14
```

Que por supuesto es también un error muy pequeño.

Con la siguiente instrucción encontramos el resultado con 20 dígitos de exactitud, sin tomar en cuenta la precisión:

```
FindRoot[SetPrecision[f[x],20],{x,1.1},
  AccuracyGoal→20,PrecisionGoal→Infinity,
  WorkingPrecision→20]//InputForm
{x -> 3.60967088614004992408302314365703834692`20.}
```

Cuyo error absoluto es:

```
Abs[(SetPrecision[f[x],20]/.%) - 0]
0.×10-19
```

Por lo tanto en este caso la exactitud requerida ha sido alcanzada.

Podemos también especificar el método que Mathematica empleará para resolver la ecuación, así para resolver la ecuación 3.1 empleando el método de la Secante escribimos:

```
FindRoot[f[x],{x,1.1,1.2},Method→"Secant"]//InputForm
{x -> 3.60967088614005}
```

Para realizar alguna acción o ver resultados intermedios en cada una de las iteraciones que lleva a cabo "FindRoot" podemos emplear "StepMonitor", por ejemplo, para ver los valores de "x" en cada una de las iteraciones de la anterior instrucción escribimos:

```
Reap[FindRoot[f[x],{x,1.1,1.2},Method→"Secant",
  StepMonitor→Sow[x]]]//InputForm
{{x -> 3.60967088614005},
 {{3.6094473457279346,
  3.6096716168396714, 3.6096708861396634,
  3.60967088614005}}}
```

Obtenemos así los valores de las cuatro iteraciones que se han llevado a cabo para encontrar el resultado. Observe que los dos últimos valores calculados son iguales en 11 dígitos, por lo que esa es la precisión del resultado. En este ejemplo, para recolectar los resultados intermedios en una lista se ha empleado el par de instrucciones "Reap" y "Sow", que operan de manera similar a "Catch" y "Throw", sólo que en este caso "Sow" no termina la ejecución del bloque, sino que manda valores (o expresiones) que son recolectados por "Reap" en una lista, de manera que "Reap" devuelve el resultado de la expresión principal y una lista con los valores o expresiones mandados por "Sow" (como se ve en el ejemplo).

Veamos ahora las iteraciones que se llevan a cabo cuando se resuelve el problema con el método de "Brent":

```
Reap[FindRoot[f[x],{x,1.,5.},Method→"Brent",StepMonitor→Sow[x]]]//InputForm
{{x -> 3.60967088614005},
 {{5., 3.6031067904576983,
  3.609644772995118, 3.609670886056749,
  3.6096708861400497}}}
```

Vemos entonces que "Brent" requiere 5 iteraciones (una más que "Secante") y que los dos últimos valores son iguales en 10 dígitos, por lo que el resultado es menos preciso que el obtenido con el método de la "Secante".

Con el método de "Newton" obtenemos lo siguiente:

```
Reap[FindRoot[f[x],{x,1.1},Method→"Newton",StepMonitor→ Sow[x]]]//InputForm
```

```
{{x -> 3.60967088614005},
 {{3.608916103437996, 3.609670887487825,
 3.60967088614005}}}
```

En consecuencia "Newton" requiere solo tres iteraciones y los dos últimos valores son iguales en 9 dígitos, por lo que esa es la precisión del resultado.

Sin embargo, el número de iteraciones no refleja necesariamente la eficiencia del método, porque dependiendo del método cada iteración puede requerir dos o más evaluaciones funcionales (dos o más veces que se evalúa la función o ecuación que se está resolviendo). Para ver resultados intermedios, o realizar alguna acción, cada vez que la función es evaluada empleamos "EvaluationMonitor", así los valores de "x" para cada una de las evaluaciones que se requieren en el método de la "Secante" son:

```
Reap[FindRoot[f[x],{x,1.1,1.2},Method->"Secant",
 EvaluationMonitor-> Sow[x]]]//InputForm
{{x -> 3.60967088614005},
 {{1.1, 1.2, 1.200000137230348,
 3.6094473457279346, 3.6094473476443567,
 3.6096716168396714, 3.60967161673028,
 3.6096708861396634, 3.609670886139743,
 3.60967088614005}}}
```

Vemos entonces que si bien el método de la "Secante" requiere sólo 4 iteraciones, en las mismas la función es evaluada 10 veces. Con el método de "Brent" obtenemos lo siguiente:

```
Reap[FindRoot[f[x],{x,1,5},Method->"Brent",
 EvaluationMonitor-> Sow[x]]]//InputForm
{{x -> 3.60967088614005},
 {{1., 5., 3.6031067904576983,
 3.609644772995118, 3.609670886056749,
 3.6096708861400497, 3.60967088614005}}}
```

Entonces "Brent" realiza dos evaluaciones más que el número de iteraciones, requiriendo en consecuencia menos evaluaciones que el método de la "Secante". Sin embargo, cuando el método calcula derivadas numéricas, como en el caso de "Newton", "EvaluationMonitor" no toma en cuenta las evaluaciones que se llevan a cabo en dicho proceso, por lo que devuelve un menor número de evaluaciones funcionales que las realmente ocurren.

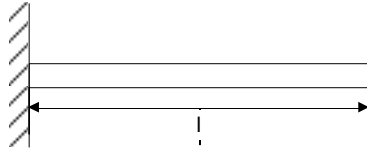
Finalmente, podemos cambiar el máximo número de iteraciones permitido con "MaxIterations". Normalmente se cambia este límite cuando con el valor establecido por defecto (100) no se logra encontrar el resultado. Para probar esta opción, en la siguiente instrucción intentamos calcular el resultado de la ecuación 3.1 con el método de Newton, recolectando los valores intermedios de la variable independiente en una lista, pero limitando el número de iteraciones a 2:

```
Reap[FindRoot[f[x],{x,1.1},Method->"Newton",
 StepMonitor-> Sow[x],MaxIterations->2]]//InputForm
FindRoot::cvmit: Failed to converge to the requested accuracy or precision
within 2 iterations. >>
{{x -> 3.609670887487825},
 {{3.608916103437996, 3.609670887487825}}}
```

Como se ve los valores calculados en las dos iteraciones son iguales en 3 dígitos, por lo que no se alcanza la precisión por defecto (8), razón por la cual aparece el mensaje de error y "FindRoot" devuelve el último valor calculado. Este valor puede ser empleado para comenzar otro proceso iterativo.

10.2.1. Ejemplos

1. Calcule la frecuencia natural de una viga "p", con 20 dígitos de precisión, empleando el método de la "Secante", trabajando con 30 dígitos de precisión y recolectando los valores de "p", de cada iteración, en una lista. Para estimar los valores iniciales grafique la función entre "p=0" y "p=100", mostrando la función en el título de la gráfica, rotulando los ejes y mostrando líneas de división punteadas. La ecuación que permite calcular la frecuencia natural "p" de una viga es:



$$\cos(kl)\cosh(kl)=-1$$

$$k^2 = p/a; \quad a^2 = EIg/(Ay)$$

$$l=120 \text{ plg (longitud); } I=170.6 \text{ plg}^4 \text{ (momento de inercia)} \quad (3.2)$$

$$E=3 \times 10^6 \text{ lb/plg}^2 \text{ (módulo elástico); } y=0.066 \text{ lb/plg}^3 \text{ (densidad)}$$

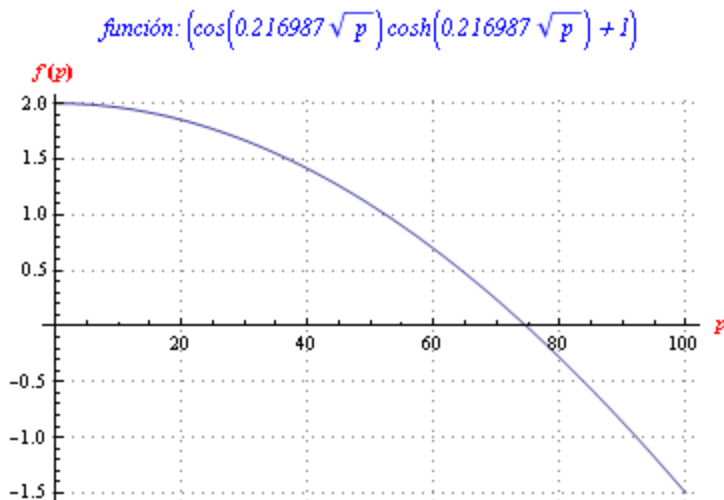
$$A=32 \text{ plg}^2 \text{ (sección transversal); } g=386 \text{ plg/s}^2 \text{ (aceleración de la gravedad)}$$

Primero creamos la función y dado que en este caso existen constantes, empleamos "With" para declararlas (aunque también podríamos declararlas como variables inicializadas con "Module" o "Block").

```
fp[p_]:=With[{l=120,I=170.6,E=3*10^6,
  y=0.066,A=32,g=386},Module[{a,k},
  a=Sqrt[E*I*g/(A*y)];k=Sqrt[p/a];
  Cos[k*l]*Cosh[k*l]+1]]
```

Ahora graficamos la función para ver el lugar de la solución:

```
Plot[fp[p ],{p,0,100},PlotLabel->Style[
  ("función:" fp[p]),Italic,Blue],
  AxesLabel->({"p","f(p)"/.x_String->Style[x,
    Red,Bold,Italic]),GridLines->Automatic,
  GridLinesStyle->Dotted]
```



Vemos entonces que la solución se encuentra entre 60 y 80 (cerca a 75). Por el título de la gráfica podemos ver que cuando se llama a la función Mathematica reemplaza las constantes y variables temporales y deja la expresión

sión en función de la variable independiente "p". Para evitar que ese trabajo sea hecho cada vez que se llama a la función, se puede emplear "Evaluate". Como sabemos, sin "Evaluate" la función queda tal como se la escribió:

```
?fp
Global`fp
fp[p_] := With[
  {l: 120, I: 170.6, E: 3*10^6, y: 0.066, A: 32, g: 386},
  Module[{a, k}, a: Sqrt[I*g/(A*y)]; k: Sqrt[p/a]; Cos[k*l] Cosh[k*l] + 1]]
```

Aplicando "Evaluate":

```
fp[p_] := Evaluate[With[{l=120, I=170.6, E=3*10^6,
  y=0.066, A=32, g=386}, Module[{a, k},
  a=Sqrt[E*I*g/(A*y)]; k=Sqrt[p/a];
  Cos[k*l]*Cosh[k*l]+1]]]
```

La función es almacenada como:

```
?fp
Global`fp
fp[p_] := 1+ Cos[0.216987 Sqrt[p]] Cosh[0.216987 Sqrt[p]]
```

Ahora encontramos la solución, empleando como valores iniciales 70 y 71:

```
Reap[FindRoot[SetPrecision[fp[p], 30],
  {p, 70, 71}, Method->"Secant",
  PrecisionGoal->20, AccuracyGoal->Infinity,
  WorkingPrecision->30, StepMonitor->Sow[p]
]]//InputForm
{ {p -> 74.67669629022705013622712231262410401003`30.},
  {{74.78449788435475467726304000859525285122`30.,
    74.67430443673385928331353342059908110479`30.,
    74.67669477825036433420044524596705039796`30.,
    74.67669629024827413733890086359044150462`30.,
    74.67669629022705013603879560880733750724`30.,
    74.67669629022705013622712231262410401003`30.}}}
```

2. Elabore una función que reciba como parámetros la longitud, sección transversal y momento de inercia de una viga y calcule con las mismas su frecuencia natural, empleando la ecuación del ejercicio anterior y declarando como parámetros opcionales el módulo elástico (E), la densidad (y), la aceleración de la gravedad (g) y el valor inicial asumido (p0), con valores por defecto iguales a: 3×10^6 , 0.0066, 386 y 50.1 respectivamente.

En la práctica, la mayoría de las constantes de una ecuación, como las de la ecuación 3.2, son en realidad parámetros que varían en función de las condiciones del problema que se quiere resolver, por lo tanto una forma más útil de programarlas es recibiendo como parámetros los valores que varían más frecuentemente y como parámetros opcionales aquellos que no varían con mucha frecuencia, tal como se propone en este ejemplo.

```
fp[l_, A_, I_, OptionsPattern[]] := Module[{a, k,
  f}, a=Sqrt[OptionValue[E0]*I*
  OptionValue[g]/(A*OptionValue[y])];
  f[p_] := (k=Sqrt[p/a]; Cos[k*l]*Cosh[k*l]+1);
  FindRoot[f[p], {p, OptionValue[p0]}]]
```

Con esta función podemos volver a resolver la ecuación para los valores del ejercicio anterior:

```
fp[120,32,170.6]//InputForm
{p -> 74.67669629022706}
```

Pero podemos emplearla también con otros valores:

```
fp[140,32,140]//InputForm
{p -> 49.701102383095375}
fp[90,32,180]//InputForm
{p -> 136.36700704296445}
fp[120,32,170.6,y->0.08]//InputForm
{p -> 67.82847778978105}
```

3. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales. Para ubicar el segmento de solución trace una gráfica entre 0 y 5.

$$3x^{2.1} - 5y = 7.0$$

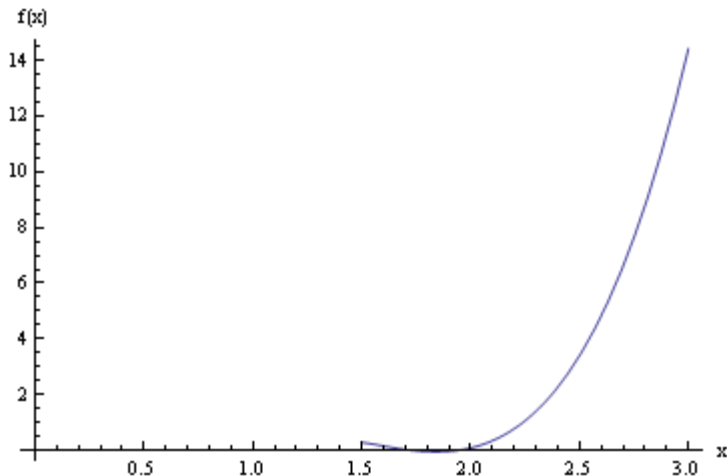
$$\begin{aligned} y^{1.2} + 4z &= 14.3 \\ x + y^2 + z^2 &= 14.0 \end{aligned} \quad (3.3)$$

Aunque este sistema tiene 3 variables, en realidad depende sólo de una: "x", pues con "x" se puede calcular "y", y con "y" se puede calcular "z":

$$\begin{aligned} f(x) &= x + y^2 + z^2 - 14.0 = 0 \\ y &= \frac{3x^{2.1} - 7.0}{5} \\ z &= \frac{14.3 - y^{1.2}}{4} \end{aligned}$$

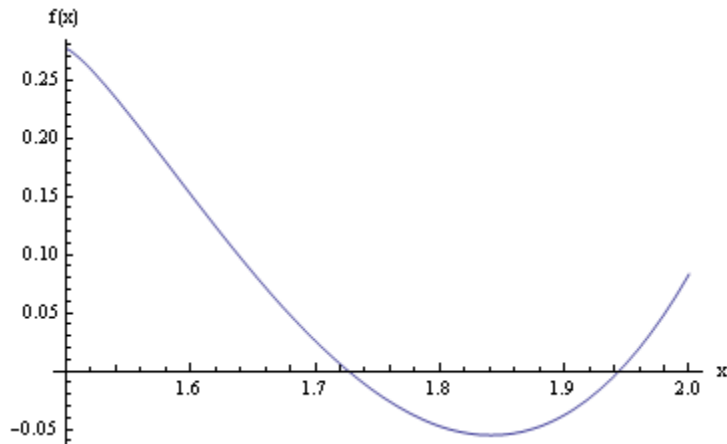
Para encontrar el segmento de solución primero creamos la función y la graficamos:

```
fx[x_] := Module[{y, z},
  y = (3*x^2.1 - 7.) / 5; z = (14.3 - y^1.2) / 4;
  x + y^2 + z^2 - 14]
Plot[fx[x], {x, 0, 3}, AxesLabel -> {"x", "f(x)"}]
```



Como se puede ver en realidad existen dos soluciones comprendidas entre 1.5 y 2.0. Para ver con más detalle el lugar de las soluciones volvemos a graficar la función entre esos dos límites:

```
Plot[fx[x], {x, 1.5, 2}, AxesLabel -> {"x", "f(x)"}]
```



Vemos que las dos soluciones se encuentran entre 1.7 y 1.8 y entre 1.9 y 2.0. Ahora podemos encontrar soluciones más exactas con "FindRoot", pero para devolver las tres soluciones programamos la función dentro de un módulo empleando variables locales:

```
Module[{f,x,y,z},f:=(y=(3*x^2.1-7.)/5;
  z=(14.3-y^1.2)/4;x+y^2+z^2-14);
x=x/.FindRoot[f,{x,1.75,1.7,1.8}];
Print["Primera solución: "];
Print["x = ",x," y = ",y," z = ",z];
x=x/.FindRoot[f,{x,1.95,1.9,2.0}];
Print["Segunda solución: "];
Print["x = ",x," y = ",y," z = ",z];]
Primera solución:
x = 1.72653 y = 0.488926 z = 3.46907
Segunda solución:
x = 1.9438 y = 1.0228 z = 3.31814
```

4. Elabore una función que reciba como parámetros opcionales los valores de "L₀", "x₀", "V₀", "y₀" y resuelva la siguiente función devolviendo los valores de "L₁", "x₁", "V₁" y "y₁". Los valores por defecto de los parámetros opcionales son: "L₀=300", "x₀=0", "V₀=100", "y₀=0.2". Para tener una idea del lugar de la solución grafique la función entre 0 y 1 empleando los valores por defecto.

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0 = 0$$

$$L_1 = \frac{L_c}{1 - x_1}$$

$$V_1 = M - L_1$$

$$y_1 = 1.420x_1 \quad (3.4)$$

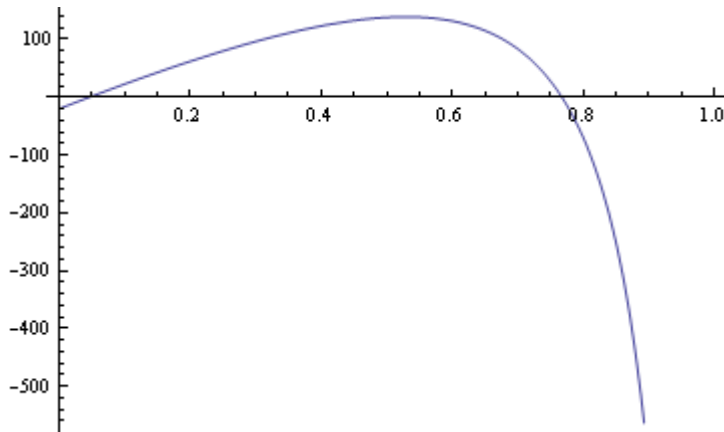
$$L_c = L_0(1 - x_0)$$

$$M = L_0 + V_0$$

$$C_0 = L_0x_0 + V_0y_0$$

Para graficar la función primero la programamos empleando los valores por defecto:

```
fx1[x1_]:=With[{L0=300,x0=0,V0=100,y0=0.2},
Module[{L1,V1,y1,C0,M,Lc},
  C0=L0*x0+V0*y0;M=L0+V0;Lc=L0*(1-x0);
  L1=Lc/(1-x1);V1=M-L1;y1=1.420*x1;
  L1*x1+V1*y1-C0]]
Plot[fx1[x1],{x1,0,1}]
```



Como se puede ver existen dos soluciones que se encuentran cerca de 0.05 y 0.75 aproximadamente. En general, sin embargo, se sabe que para esta función la solución se encuentra entre 0 y 1, por lo que un valor inicial por defecto igual a 0.5 es generalmente adecuado y este será el valor que emplearemos (x_{li}) para programar la función requerida:

```
Remove[fx1]
Options[fx1]={L0→300,x0→0,V0→100,y0→0.2,xli→0.5};
fx1[OptionsPattern[]]:=Module[{L1,x1,V1,y1,
  C0,M,Lc,L0=OptionValue[L0],
  x0=OptionValue[x0],V0=OptionValue[V0],
  y0=OptionValue[y0]},C0=L0*x0+V0*y0;
M=L0+V0;Lc=L0*(1-x0);f:=(L1=Lc/(1-x1);
  V1=M-L1;y1=1.420*x1;L1*x1+V1*y1-C0);
x1=x1/.FindRoot[f,{x1,OptionValue[xli],0,1}];
{Symbol["L1"]→L1,Symbol["x1"]→x1,
  Symbol["V1"]→V1,Symbol["y1"]→y1}]
```

Ahora con esta función podemos encontrar una solución más exacta:

```
fx1[]
{L1→314.425,x1→0.0458777,V1→85.5749,y1→0.0651464}
```

Que como vemos converge hacia la primera solución. Para encontrar la segunda solución podemos cambiar el valor inicial por defecto a 0.7:

```
fx1[xli→0.7]
{L1→1290.34,x1→0.767503,V1→-890.337,y1→1.08985}
```

Y podemos encontrar también la solución empleando valores diferentes a los valores por defecto:

```
fx1[L0→350,x0→0.05,V0→200,y0→0.3]
{L1→379.873,x1→0.124708,V1→170.127,y1→0.177085}
```

10.2.2. Ejercicios

1. Programe una función que resuelva la ecuación: $f(x)=2x^2+1-e^x=0$, empleando el método "Secant", con 18 dígitos de precisión (sin tomar en cuenta la exactitud), trabajando con 25 dígitos de precisión y mostrando los valores de "x" de cada iteración. Para estimar los valores iniciales del método grafique la función entre $x=-1$ y $x=2$, siendo el título de la gráfica la función (negrita, cursiva, 16, naranja), las etiquetas de los ejes "x" y "f(x)" (azul, cursiva, negrita) y teniendo la gráfica líneas de división (verde, guiones finos).

2. Programe una función que resuelva la ecuación: $f(x)=x^3+2x^2+3x+4=0$, empleando el método "Newton", con 16 dígitos de exactitud (sin tomar en cuenta la precisión), trabajando con 20 dígitos de precisión y mostrando los valores de "x" cada vez que la función es evaluada. Para estimar el valor inicial del método grafique la función entre $x=-3$ y $x=1$, siendo el título de la gráfica la ecuación (verde y cursiva), las etiquetas de los ejes "x" y "f(x)" (azul y negrita) y teniendo la gráfica líneas de división (guiones, rojo).
3. Programe una función que resuelva la ecuación: $f(z)=e^{z/2}+z^2-60=0$, empleando el método "Brent", con 20 dígitos de precisión o exactitud (el que se logre primero), trabajando con 30 dígitos de precisión y mostrando los valores de la función "f(z)" de cada iteración. Para estimar el valor inicial del método grafique la función entre $z=0$ y $z=10$, siendo el título de la gráfica la ecuación (en rojo y cursiva, negrita, 10 puntos) y las etiquetas de los ejes "z" y "f(z)" (en verde, cursiva, negrita, 10 puntos).
4. Programe una función que resuelva la ecuación: $f(z)=\ln(z)+e^{z+3}-z^{3.1}-42=0$, dando el valor inicial y los límites de búsqueda, con 12 dígitos de precisión o 16 de exactitud (el que se logre primero), trabajando con 20 dígitos de precisión y mostrando los valores de la función "f(z)" cada vez que la función es evaluada. Para estimar el valor inicial del método grafique la función entre $z=0.1$ y $z=2$, siendo el origen de los ejes (0,0), el título de la gráfica la ecuación (azul, cursiva, negrita, 14 puntos, "Arial") y las etiquetas de los ejes "z" y "f(z)" (naranja, cursiva, negrita, 12 puntos, "Times New Roman").
5. Programe una función que resuelva el siguiente sistema de ecuaciones no lineales, devolviendo los valores de "x" y "y" en forma de reglas, con 10 dígitos de precisión (sin importar la exactitud) y trabajando con la una precisión de 20 dígitos. Para estimar el valor inicial programe el sistema de ecuaciones en función de "x" y grafique la función entre 2 y 10, siendo los rótulos de los ejes "x" y "f(x)" (rojo, negrita, cursiva, 10 puntos) y teniendo la gráfica líneas de división (guiones y puntos, verde).

$$\begin{aligned} 2x^2 + 5xy - 4x &= 115 \\ e^{\frac{x+y}{5}} + x^2y^2 - 70y &= 15 \end{aligned}$$

6. Programe una función que resuelva el siguiente sistema de ecuaciones no lineales, devolviendo los valores de "x", "y" y "z" con la precisión, exactitud y dígitos por defecto y proporcionando dos valores iniciales. Para estimar los valores iniciales programe el sistema de ecuaciones en función de "x" y grafique la función entre 0 y 14, siendo los rótulos de los ejes "x" y "f(x)" (púrpura, negrita, cursiva, 14 puntos) y teniendo la gráfica líneas de división (guiones cortos, azul).

$$\begin{aligned} 15y + 20x^2 - x^3 &= 1500 \\ 9z - 1.5x^2 + e^{\frac{x}{2}} &= 300 \\ y^2 - z^3 - 5x &= 166.81 \end{aligned}$$

7. Programe una función que resuelva la ecuación de Frank y Kamenetsky: $f(t)=\exp((-t/2)\cosh^{-1}(e^{t/2}))-(L_{cr}/2)^{1/2}=0$, siendo "L_{cr}" un parámetro opcional con un valor por defecto igual a 0.088, el valor inicial asumido (t0) debe ser también un parámetro opcional y para estimar el valor inicial programe la función (empleando el valor por defecto) y gráfiquela entre 0 y 10 siendo el título de la gráfica "Ecuación de Frank y

Kamenetsky" (azul, cursiva, negrita, 16 puntos), las etiquetas de los ejes "t" y "f(t)" (azul, cursiva, negrita, 14 puntos). Emplee los valores por defecto de "FindRoot".

8. Programe una función para calcular el factor de fricción "f" con la ecuación de Lee - Duffy: $F(f)=1/f^{1/2}-(1/k)\ln(R_e f^{1/2})-(14-5.6/k)=0$, siendo "R_e" y "k" parámetros opcionales con valores por defecto iguales a 3750 y 0.28 respectivamente, el valor inicial asumido (f₀) debe ser también un parámetro opcional y para estimar el valor inicial programe la función (empleando los valores por defecto) y grafíquela entre 0 y 0.05 siendo el título de la gráfica "Ecuación de Lee-Duffy" (azul, cursiva, 18 puntos, "Helvetica"), las etiquetas de los ejes "f" y "F(f)" (verde, cursiva, negrita, 14 puntos), líneas de división (azul, punteada). El resultado debe ser calculado con 16 dígitos de precisión (sin importar la exactitud) trabajando con 26 dígitos de precisión.
9. Programe una función para calcular el volumen "V" con la ecuación de Soave-Redlich-Kwong, donde "T", "P", "T_c", "P_c" y "w", son parámetros opcionales con valores por defecto iguales a: "T=400", "P=20", "T_c=190.6", "P_c=45.4" y "w=0.008", el valor inicial asumido (V₀) debe ser también un parámetro opcional y para estimar el mismo debe programar la función (empleando los valores por defecto) y grafíquela entre 0.3 y 5 siendo el título de la gráfica "Ecuación de Soave-Redlich-Kwong" (rojo, 18 puntos, negrita, "Courier New"), las etiquetas de los ejes "V" y "f(V)" (azul, cursiva, negrita, 16 puntos, "Arial"), líneas de división (verde, guiones cortos). El resultado debe ser calculado con 20 dígitos de exactitud (sin importar la precisión) trabajando con 30 dígitos de precisión. La ecuación de Soave-Redlich-Kwong es la siguiente:

$$\begin{aligned} \frac{PV}{RT} &= \frac{V}{V-b} - \frac{\Omega_a}{\Omega_b} \frac{b}{V+b} F \\ a &= \frac{\Omega_b R^2 T_c^{2.5}}{P_c} \\ b &= \frac{\Omega_a R T_c}{P_c} \\ F &= \frac{1}{T_r} \left[1 + (0.480 + 1.574\omega - 0.176\omega^2)(1 - T_r^{0.5}) \right]^2 \\ T_r &= \frac{T}{T_c} \\ \Omega_a &= \left[9 \left(2^{1/3} - 1 \right) \right]^{-1} \\ \Omega_b &= \frac{2^{1/3} - 1}{3} \\ R &= 0.08206 \frac{\text{l.atm}}{\text{K.gmol}} \end{aligned}$$

10. Programe una función para calcular el volumen "V" con la ecuación de Beattie-Bridgeman, donde "P", "T", "A₀", "B₀", "a", "b" y "c", son parámetros opcionales con valores por defecto iguales a: "P=36", "T=408", "A₀=16.6037"; "B₀=0.2354"; "a=0.11171"; "b=0.07697" y "c= 300x10⁴". El valor inicial asumido (V₀) debe ser calculado con la ecuación V₀=R*T/P, pero para tener una idea del lugar de la solución debe programar la función (empleando los valores por defecto) y graficarla entre 0 y 2 siendo el título de la gráfica "Ecuación de Beattie-Bridgeman" (azul,

16 puntos, cursiva, "Century Gothic"), las etiquetas de los ejes "V" y "f(V)" (rojo, cursiva, 10 puntos, "Arial Black"), líneas de división (cian, guiones). El resultado debe ser calculado con la precisión y exactitud por defecto de "FindRoot". La ecuación de estado de Beattie-Bridgeman es la siguiente:

$$V = \left(RT + \frac{\beta}{V} + \frac{\gamma}{V^2} + \frac{\delta}{V^3} \right) \frac{1}{P}$$

$$\beta = RTB_0 - A_0 - \frac{Rc}{T^2}$$

$$\gamma = -RTB_0b + aA_0 - \frac{RB_0c}{T^2}$$

$$\delta = \frac{RB_0bc}{T^2}$$

$$R = 0.08206 \frac{\text{atm.l}}{\text{K.gmol}}$$

11. Programe una función que calcule los valores de "T₂" y "H₂" con la ecuación de saturación que se presenta al pie de la pregunta, donde "T₁", "H₁", "P" y "H_{P2}", son parámetros opcionales con valores por defecto iguales a: "T₁=20", "H₁=0.002", "P=540" y "H_{P2}=80". El valor inicial asumido (T₀) también debe ser un parámetro opcional y para tener una idea de su valor inicial debe programar la función (empleando los valores por defecto) y graficarla entre 0 y 15 siendo el título de la gráfica "Ecuación de Saturación" (púrpura, 18 puntos, cursiva, "Alba"), las etiquetas de los ejes "T₂" y "f(T₂)" (azul, cursiva, 12 puntos, "Alba"), líneas de división (verde, punteado). El resultado debe ser calculado con 12 dígitos de precisión (sin importar la exactitud) y trabajando con 24 dígitos de precisión. La ecuación de saturación es la siguiente:

$$f(T_2) = \frac{H_2 - H_1}{T_2 - T_1} + \frac{c_{s1}}{\lambda_2} = 0$$

$$H_2 = \frac{H_{P2} * H_{S2}}{100}$$

$$H_{S2} = \frac{18.02}{28.97} \frac{P_{V2}}{P - P_{V2}}$$

$$P_{V2} = e^{\left(18.3036 - \frac{3816.44}{T_2 + 227.02} \right)}$$

$$c_{s1} = 1.005 + 1.88 * H_1$$

$$\lambda_2 = 2501.4 * \left(\frac{1 - \frac{T_2}{647.3}}{1 - 0.422} \right)^{0.375}$$

12. Programe una función que calcule los valores de "x_i" y "y_i" con la ecuación de equilibrio que se presenta al pie de la pregunta, donde "x", "y", "k'_x" y "k'_y", son parámetros opcionales con valores por defecto iguales a: "x=0.6", "y=0.4", "k'_x=0.002967" y "k'_y=0.001465" respectivamente. El valor inicial asumido (x_{i0}) también debe ser un parámetro opcional y para tener una idea de su valor inicial debe programar la función (empleando los valores por defecto) y graficarla entre 0 y 1 siendo el título de la gráfica "Ecuación de Equilibrio" (rojo, 16 puntos, cursiva, "Arial"), las etiquetas de los ejes "x_i" y "f(x_i)" (na-

ranja, cursiva, 12 puntos, "Alba"), líneas de división (azul, punteado y guiones). El resultado debe ser calculado con la precisión y exactitud por defecto. La ecuación de equilibrio es la siguiente:

$$\frac{k_x}{k_y} = -\frac{y - y_i}{x - x_i}$$

$$y_i = -0.0405 + 1.03785714286x_i$$

$$k_x = \frac{k'_x}{(1-x)_{im}}$$

$$k_y = \frac{k'_y}{(1-y)_{im}}$$

$$(1-x)_{im} = \frac{(1-x) - (1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)}$$

$$(1-y)_{im} = \frac{(1-y_i) - (1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

11. SUSTITUCIÓN DIRECTA Y WEGSTEIN

11.1. Método de sustitución directa

Ahora que podemos resolver funciones con una incógnita en Mathematica, estudiaremos algunos de los métodos numéricos que existen para este fin. Comenzaremos el estudio con el más sencillo de todos: el método de *Sustitución Directa*, el cual requiere que la función a resolver se encuentre en la forma:

$$x = g(x) \quad (3.5)$$

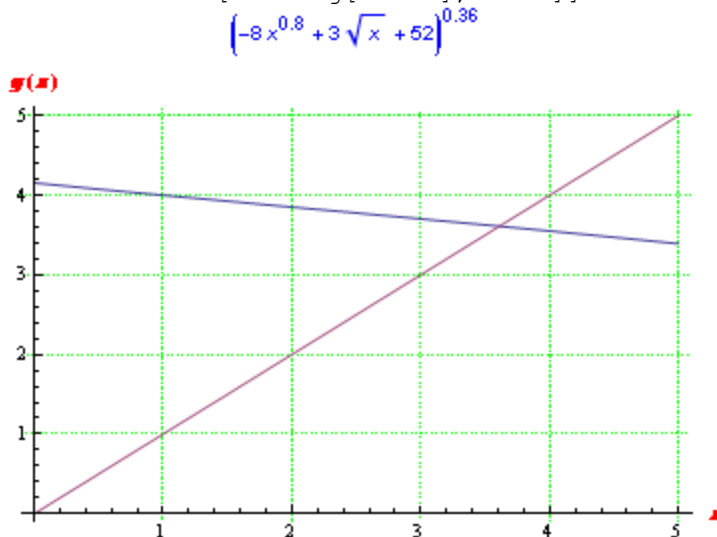
Siendo esta su principal desventaja, pues en la mayoría de los casos es necesario despejar previamente la variable independiente, algo que no siempre es sencillo y en algunas ocasiones puede resultar incluso imposible.

El algoritmo básico del método es el siguiente:

- Se asume un valor inicial de "x" (x_1).
- Con el valor asumido se calcula el valor de la función $x_2 = g(x_1)$.
- Se compara el valor calculado con el asumido (x_1 y x_2). Si son aproximadamente iguales el proceso concluye, siendo la solución " x_2 ", caso contrario x_1 toma el valor de x_2 ($x_1 = x_2$) y se repite el proceso desde el paso b.

Cuando la ecuación está en la forma " $x = g(x)$ ", la solución gráfica se encuentra en la intersección de la curva " $g(x)$ " con la recta " $g = x$ ", por lo tanto para ver gráficamente el lugar de la solución se debe graficar la función y la recta, así para ver el lugar de la solución de la ecuación 3.1 escribimos las siguientes instrucciones:

```
g[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36
Plot[{g[x],x},{x,0,5},PlotLabel->Style[
  g[x],Blue,Bold,10,FontFamily->
  "Arial"],AxesLabel->({ "x", "g(x)" } /.
  x_String->Style[x,Red,Bold,Italic,
  FontFamily->"Arial Black"]),
  GridLines->Automatic,GridLinesStyle->
  Directive[Dashing[0.001],Green]]
```



Como era de esperar la solución (la intersección) se encuentra, como antes, entre 3 y 4. La lógica básica del método corresponde al procedimiento general que sigue "FixedPoint", por lo que podemos aplicar dicha instrucción para resolver ecuaciones por el método de sustitución directa, así para resolver la ecuación 3.1 escribimos lo siguiente:

```
FixedPoint[g,1.1,50]//InputForm
3.60967088614005
```

Podemos ver los valores calculados en cada una de las iteraciones con la siguiente instrucción (o también con FixedPointList):

```
Reap[FixedPoint[(Sow[#];g[#])&,1.1,50]]//InputForm
{3.60967088614005, {{1.1, 3.9840553877884246, 3.552012132021903,
  3.618479599247315, 3.608323565491999, 3.609876925738287,
  3.609639376571533, 3.6096757048676587, 3.609670149216255,
  3.6096709988371494, 3.6096708689053827, 3.6096708887757325,
  3.609670885736977, 3.6096708862016915, 3.609670886130623,
  3.6096708861414912, 3.6096708861398294, 3.6096708861400835,
  3.609670886140045, 3.6096708861400506, 3.6096708861400497}}}
```

Como se puede observar, este método requiere muchas más iteraciones (21) que "FindRoot" (que requieren como máximo 5. Gráficamente el método de sustitución directa sigue el camino que se muestra en la siguiente figura:

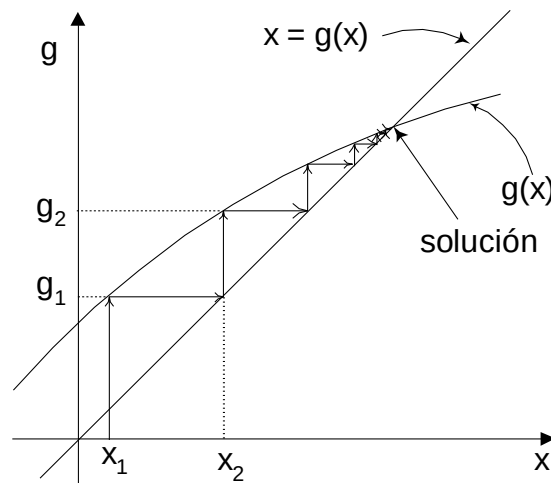


Figura 3.1. Método de Sustitución Directa

En la práctica sólo son visibles las primeras iteraciones, pues las últimas tienen variaciones tan pequeñas que no son apreciables a simple vista.

11.1.1. Programa

El algoritmo básico ha sido descrito en el anterior acápite, sin embargo para programar el método es necesario hacer algunas consideraciones adicionales. En primer lugar debemos tomar en cuenta que no siempre se logra la convergencia, y este método en particular es uno de los menos estables, por lo tanto es necesario establecer un límite de iteraciones, de manera que al ser alcanzado se detenga el proceso y se genere un error. En segundo lugar debemos permitir fijar la precisión y/o exactitud del resultado, repitiendo el proceso hasta que los dos últimos valores sean iguales en un determinado número de dígitos o hasta que la igualdad se cumpla con cierto error absoluto permitido: $|g[x]-x| < 1 \times 10^{-n}$ (donde "n" es la exactitud requerida).

El algoritmo elaborado tomando en cuenta las anteriores consideraciones es el siguiente:

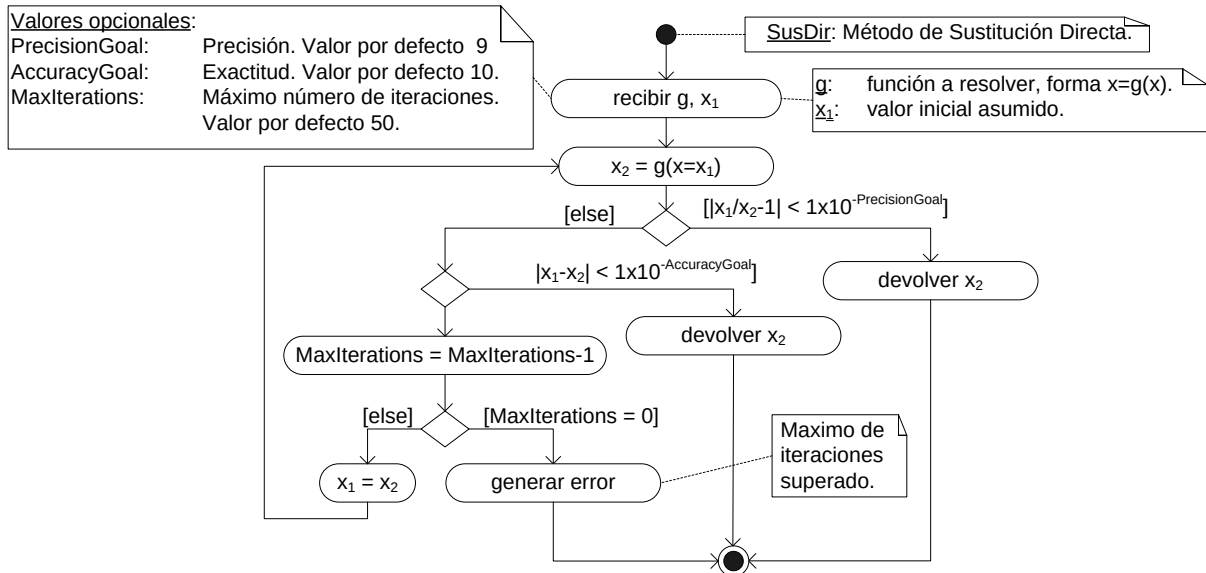


Figura 3.2. Diagrama de Actividades del método de Sustitución Directa

Donde los "valores opcionales" nos permiten especificar la precisión, exactitud y límite de iteraciones. Para ser coherentes con las funciones definidas por Mathematica (y evitar así el tener que aprender otros términos) se han nombrado los parámetros opcionales con los mismos nombres definidos para la instrucción "FindRoot". En los programas que elaboraremos en este y en los siguientes temas, trabajaremos con la precisión por defecto de Mathematica, es decir con 16 dígitos (\$MachinePrecision), podemos incrementar dicha precisión con las instrucciones estudiadas en el anterior capítulo, empleando por ejemplo la función SetPrecisión y/o fijando la precisión o exactitud con los operadores ' y '', así como estableciendo el valor de las constantes \$MaxPrecision, \$MinPrecision, y \$MaxExtraPrecision, sin embargo, dado que en la mayoría de las aplicaciones ingenieriles es suficiente calcular el resultado con unos 6 dígitos de precisión, en general es suficiente trabajar con la precisión por defecto.

Para que los programas que elaboremos puedan ser empleados desde cualquier cuaderno, sin tener que copiar (o reprogramar) el método cada vez que se utilice, haremos uso de los paquetes (Package). Un paquete es el equivalente aproximado de una unidad (Unit) de Pascal.

Al igual que una unidad, un paquete tiene dos sectores: uno público, donde se dan a conocer los nombres de las funciones, variables y constantes que podrán ser empleadas desde otros cuadernos (o paquetes) y un sector privado, donde se escriben los programas propiamente y se definen variables y constantes que sólo son empleadas internamente. La estructura general de un paquete es la siguiente:

```

BeginPackage["Nombre_del_paquete`", {"paquetel`", "paquete 2`", ...}]
  Nombres de las funciones, variables y constantes públicas
Begin["`Private`"]
  Programas y nombres de las funciones variables y constantes privadas
End[]
  Instrucciones adicionales
EndPackage[]
  
```

La lista {"paquete1`", "paquete2`", ...} corresponde a los nombres de los paquetes que serán utilizados dentro del paquete y es opcional (es decir puede no existir).

Los nombres de las funciones, variables y constantes públicas se declaran empleando el rótulo "usage" de acuerdo a la siguiente sintaxis:

Nombre_variable::usage= "texto de ayuda con relación a la variable"

Las *instrucciones adicionales* normalmente corresponden a la instrucción "Protect", empleada para evitar que las funciones, variables y constantes públicas sean borradas o sobreescritas en los cuadernos (o paquetes) donde son importados:

Protect[variable1,variable2,...]

Para familiarizarnos con los paquetes crearemos primero un paquete con una función muy simple (para calcular el cubo de un número). Para ello elija la opción "File->New->Package" y en la ventana que aparece escriba lo siguiente (estando cada instrucción en una celda):

```
BeginPackage["Prueba`"]

cubo::usage="cubo[x]: Devuelve el cubo de \"x\""
```

Begin["`Private`"]

```
cubo[x_]:=x^3

End[]

Protect[cubo]

EndPackage[]
```

Y guarde el paquete (File->Save as) con el nombre "Prueba" (es importante que guarde el paquete con el mismo nombre empleado en la instrucción "BeginPackage") en el directorio donde normalmente guarda sus cuadernos (para este ejemplo asumiremos que dicho directorio es: "C:\MAT205\").

Una vez creado el paquete, el mismo debe ser importado para que sus funciones, variables y constantes puedan ser utilizadas, ello se puede lograr con las instrucciones "Get" y "Needs". La diferencia más importante entre estas dos instrucciones es que "Get" siempre importa el paquete, aun cuando ya hubiese sido importado previamente, mientras que "Needs" sólo importa el paquete si el mismo no ha sido todavía importado. Como al crear paquetes se cometen errores con frecuencia y el paquete debe ser corregido y vuelto a importar una y otra vez, resulta más útil el comando "Get", siendo esa la razón por la cual emplearemos esta instrucción, cuya sintaxis es:

Get[Nombre_del_paquete,Path->Directorio]

Donde "Directorio" es el directorio en el cual el paquete ha sido guardado, en nuestro ejemplo "C:\MAT205" (cuando no es necesario escribir el directorio o se da el nombre con el directorio incluido, se puede emplear el poerador "<<" en lugar de "Get").

Ahora abra un cuaderno nuevo (File->New->Notebook) e importe el paquete "Prueba":

```
Get["Prueba`",Path->"C:\MAT205"]
```

Entonces al pedir ayuda con relación a "cubo" obtendrá:

```
?cubo
```

```
cubo[x]: Devuelve el cubo de "x"
```

Si intenta eliminar la variable "cubo" obtendrá el siguiente mensaje:

```
Remove[cubo]
```

```
Remove::rmptc : Symbol cubo is Protected and cannot be removed. >>
```

Y si emplea la función "cubo" obtendrá la respuesta correcta:

```
cubo[3.]
```

```
27.
```

Como ya se dijo, cuando se crean paquetes es frecuente cometer muchos errores, de manera que el paquete debe ser corregido una y otra vez, lo que puede resultar moroso cuando las variables están protegidas, pues antes de volver a importar el paquete es necesario desproteger las variables (con "UnProtect"), eliminarlas (con "Remove") y volver a importar el paquete (con Get), por ello, es recomendable no proteger las variables hasta que todos los errores hayan sido corregidos.

Ahora que ya hemos creado un paquete, crearemos otro (File->New->Package) donde primero programaremos el método de Sustitución Directa y luego iremos añadiendo los otros métodos que estudiaremos en este capítulo. Dicho paquete tendrá el nombre "Algel" (File->Save as->Algel) y tendrá el siguiente código (recordando que cada instrucción se encuentra en una celda):

```
BeginPackage["Algel`"]
```

```
SusDir::usage="SusDir[g,x1]: resuelve  
la función \"g\", que se encuentra en la  
forma \"x=g(x)\" empleando el método de  
Sustitución Directa con un valor inicial  
igual a \"x1\"";
```

```
Begin["`Private`"]
```

```
Options[SusDir]={PrecisionGoal->9,  
AccuracyGoal->10,MaxIterations->50};
```

```
SusDir::maxit="Se ha alcanzado el límite  
de iteraciones: `1`, sin haber encontrado la  
solución";
```

```
SusDir[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x0,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
li=mi;
While[True,
x2=g[x1];
If[Abs[x1/x2-1]<10^-pg,Return[x2]];
If[Abs[x1-x2]<10^-ag,Return[x2]];
If[--mi==0,Message[SusDir::maxit,li];Return[x2]];
x1=x2]]
```

```
End[]
```

```
Protect[SusDir];
```

```
EndPackage[]
```

Observe que en el sector público sólo está el nombre de la función con el rótulo "usage", pues lo único que debe estar disponible para otros paquetes (o cuadernos) es el nombre de la función: "SusDir".

En el sector privado se asigna a "SusDir" las opciones con los valores por defecto y se declara otro rótulo "maxit", para mostrar un mensaje cuando no se encuentra el resultado en el máximo de iteraciones permitido. El `1` de este rótulo es un parámetro que es reemplazado por el valor que se designa cuando se emplea la instrucción "Message" (tal como se puede observar en la función SusDir). Se pueden emplear tantos parámetros como se quiera enumerándolos consecutivamente: `1`, `2`, `3`, ..., `n`.

Recuerde no proteger la función (Protect[SusDir]) hasta que todos los errores hayan sido corregidos o inclusive puede dejar la función sin proteger.

Ahora podemos probar la función resolviendo una vez más la ecuación 3.1. Para ello creamos un nuevo cuaderno (File->New->Notebook), importamos el paquete (asumiendo que ha sido guardado en el directorio "C:\MAT205\"):

```
Get["Alge1", Path->"C:\MAT205"]
```

Programamos la ecuación 3.1:

```
g[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36
```

Y resolvemos la ecuación empleando los valores por defecto y un valor inicial igual a 1.1:

```
SusDir[g,1.1]//InputForm
3.609670885736977
```

Podemos encontrar el resultado con digamos 12 dígitos de precisión (sin importar la exactitud):

```
SusDir[g,1.1,PrecisionGoal->12,AccuracyGoal->Infinity]//InputForm
3.6096708861398294
```

O encontrar la solución con digamos 14 dígitos de exactitud (sin importar la precisión):

```
SusDir[g,1.1,PrecisionGoal->Infinity,AccuracyGoal->14]//InputForm
3.6096708861400506
```

Puesto que estamos trabajando con la precisión por defecto de Mathematica, 16 dígitos, la precisión y exactitud, no deben ser nunca superiores a 15.

Si reducimos el máximo de iteraciones a 10, el método muestra el mensaje de error, correspondiente al rótulo "maxit" y muestra el último valor calculado.

```
SusDir[g,1.1,MaxIterations->10]//InputForm
SusDir::maxit: Se ha alcanzado el límite
de iteraciones: 10, sin haber encontrado la
solución
3.6096708689053827
```

Si pedimos información de "SusDir" obtenemos el texto escrito en el rótulo "usage":

```
?SusDir
SusDir[g,x_]: resuelve
la función "g", que se encuentra en la
forma "x=g(x)" empleando el método de
```

Sustitución Directa con un valor inicial igual a " x_1 "

Como es de suponer la función "SusDir" creada no es la única forma de programar el método, podemos emplear las otras instrucciones de Mathematica: Do, For, Goto, Nest, NestWhile, así como la recursividad, conjuntamente las estructuras selectivas Which, Switch, Condition y Piecewise, para programar el método en una variedad de formas. Por otra parte, y como ya hemos visto, "FixedPoint" nos permite aplicar el método prácticamente sin necesidad de programarlo.

Así el método programado empleando las instrucciones "Do" y "Which" es:

```
SusDir[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,x1=x0,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
Catch[Do[x2=g[x1];
Which[Abs[x1/x2-1]<10^-pg,Throw[x2],
Abs[x1-x2]<10^-ag,Throw[x2],
True,x1=x2],{mi}];
Message[SusDir::maxit,mi];Throw[x2]]]
```

Donde como se puede observar se han empleado las instrucciones "Catch" y "Throw", porque como ya vimos la instrucción "Return" no funciona correctamente con "Do".

Con "Nest" el programa es:

```
SusDir[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
Catch[Nest[(x2=g[#];
If[Abs[#/x2-1]<10^-pg,Throw[x2]];
If[Abs[#-x2]<10^-ag,Throw[x2]];
x2)&,x0,mi];
Message[SusDir::maxit,mi];Throw[x2]]]
```

Con "Goto" el programa es:

```
SusDir[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x0,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
li=mi;
Label[1];x2=g[x1];
If[Abs[x1/x2-1]<10^-pg,Goto[2]];
If[Abs[x1-x2]<10^-ag,Goto[2]];
If[--mi==0,Message[SusDir::maxit,li];Goto[2]];
x1=x2;Goto[1];
Label[2];x2]
```

Empleando la recursividad el programa es:

```
SusDir[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x2,f},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
li=mi;
f[x1_]:= (x2=g[x1];
Which[Abs[x1/x2-1]<10^-pg,x2,
Abs[x1-x2]<10^-ag,x2,
```

```
--mi==0,Message[SusDir5::maxit,li];x2,
True,f[x2]]);
f[x0]]
```

Con For el programa es:

```
SusDir6[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x0,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}]];
For[li=mi,True,x1=x2,
x2=g[x1];
If[Abs[x1/x2-1]<10^-pg,Break[]];
If[Abs[x1-x2]<10^-ag,Break[]];
If[--mi==0,Message[SusDir::maxit,li];Break[]]];
x2]
```

Con NestWhile el programa es:

```
SusDir7[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,i=0,x1=x0,x2},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}]];
x2=NestWhile[(++i;g[#])&,x0,
(Abs[#1/#2-1]>=10^-pg && Abs[#1-#2]>=10^-ag)&,
2,mi];
If[i==mi,Message[SusDir::maxit,mi]];x2]
```

En todos los casos los programas siguen la lógica mostrada en el diagrama de actividades de la figura 3.2, variando solamente las herramientas empleadas, por lo tanto todos los programas devuelven los mismos resultados. Como ya se mencionó, es posible programar este método (así como los otros métodos que estudiaremos posteriormente) en una multitud de formas más, empleando las estructuras iterativas y selectivas estudiadas previamente.

11.1.2. Ejemplos

1. Programe una función que resuelva la ecuación: $f(x)=x^3+2x^2+3x+4=0$, empleando el método Sustitución Directa, con 12 dígitos de exactitud (sin tomar en cuenta la precisión), permitiendo un máximo de 100 iteraciones. Para estimar el valor inicial del método grafique la función despejada y la recta $x=g$, entre $x=-3$ y $x=1$. Luego obtenga los resultados intermedios de las primeras 7 iteraciones y construya una gráfica de las mismas.

Primero despejamos uno de los términos de la variable independiente para que la función se encuentre en la forma $x=g(x)$:

$$x = -(x^3+2x^2+4)/3$$

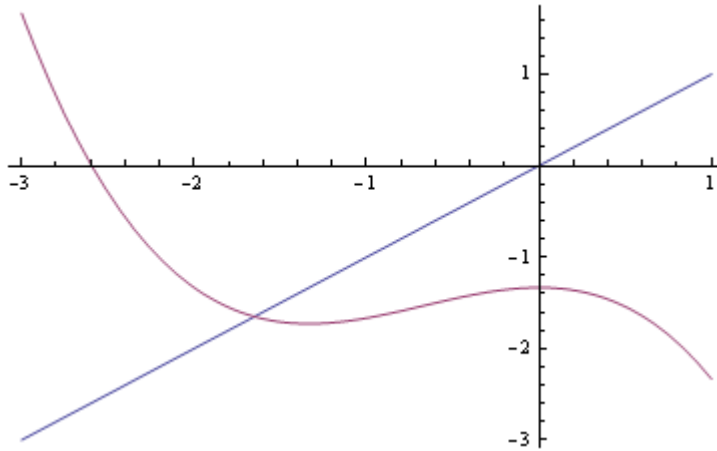
Luego importamos el paquete "Algel" y programamos la función despejada:

```
Get["Algel`",Path->"C:\MAT205 "]
g[x_]:=- (x^3+2*x^2+4)/3
```

Ahora graficamos la función "g[x]" y la recta $g=x$:

```
Plot[{x,g[x]},{x,-3,1}]
```

Con lo que obtenemos la gráfica:



Donde vemos que la solución se encuentra entre -2 y -1, por lo que podemos emplear como valor inicial -1.5:

```
SusDir[g,-1.5,PrecisionGoal->Infinity,AccuracyGoal->12,MaxIterations-
>100]//InputForm
-1.6506291914392013
```

Reemplazando este resultado en la función obtenemos:

```
x-g[x]/.x->%
2.84883*10^-13
```

Que como se puede ver tiene 12 dígitos después del cero, es decir cumple con la exactitud requerida.

Para obtener los resultados de las primeras 7 iteraciones podemos emplear la instrucción "Table", haciendo variar el límite de iteraciones (MaxIterations) entre 1 y 7:

```
Off[SusDir::maxit]
L=Table[SusDir[g,-1.5,MaxIterations->i],{i,7}]
{-1.70833,-1.61707,-1.66711,-1.64173,-1.65521,-1.64821,-1.65189}
```

La instrucción "Off" evita que se muestre el rótulo que se especifica, en este caso el rótulo correspondiente al máximo de iteraciones, se ha procedido así para evitar que el mensaje aparezca 7 veces (cada vez que se llama a SusDir). Para que el mensaje vuelva a ser mostrado se emplea la instrucción "On". Como se puede observar en este caso se trata de un proceso oscilatorio convergente.

Para graficar estas iteraciones y observar visualmente el proceso oscilatorio convergente, debemos recordar que el método sigue el camino que se muestra en la figura 3.1, es decir intercepta la curva (lo que corresponde a graficar el punto $(x, g(x))$), luego x toma el valor de la función (lo que corresponde al punto (x, x)) y repite este procedimiento hasta que x es igual a $g(x)$, por lo tanto podemos generar estos puntos a partir de la lista "L", haciendo variar el índice desde 1 hasta 6 y extrayendo en cada repetición los valores ubicados en las posiciones $\{L_i, L_i\}$ y $\{L_i, L_{i+1}\}$:

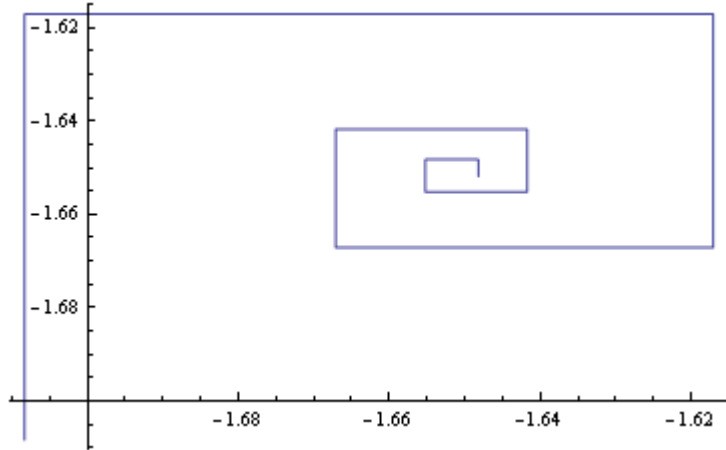
```
L=Table[{{L[[i]],L[[i]]},{L[[i]],L[[i+1]]}},{i,6}]
{{{ -1.70833,-1.70833},{ -1.70833,-1.61707}},
 {{ -1.61707,-1.61707},{ -1.61707,-1.66711}},
 {{ -1.66711,-1.66711},{ -1.66711,-1.64173}},
 {{ -1.64173,-1.64173},{ -1.64173,-1.65521}},
 {{ -1.65521,-1.65521},{ -1.65521,-1.64821}},
 {{ -1.64821,-1.64821},{ -1.64821,-1.65189}}}
```

Que tiene los puntos buscados, pero que como se puede ver están agrupados en pares de puntos y para graficarlos se requiere una lista con los puntos consecutivos, no pares de puntos, por lo tanto es necesario eliminar las listas que de primer nivel (nivel 1), lo que se puede hacer con la instrucción Flatten:

```
L=Flatten[L,1]
{{-1.70833,-1.70833},{-1.70833,-1.61707},{-1.61707,-1.61707},{-1.61707,-1.66711},
{-1.66711,-1.66711},{-1.66711,-1.64173},{-1.64173,-1.64173},{-1.64173,-1.65521},
{-1.65521,-1.65521},{-1.65521,-1.64821},{-1.64821,-1.64821},{-1.64821,-1.65189}}
```

Ahora si tenemos la lista con los puntos consecutivos y en consecuencia pueden ser graficados:

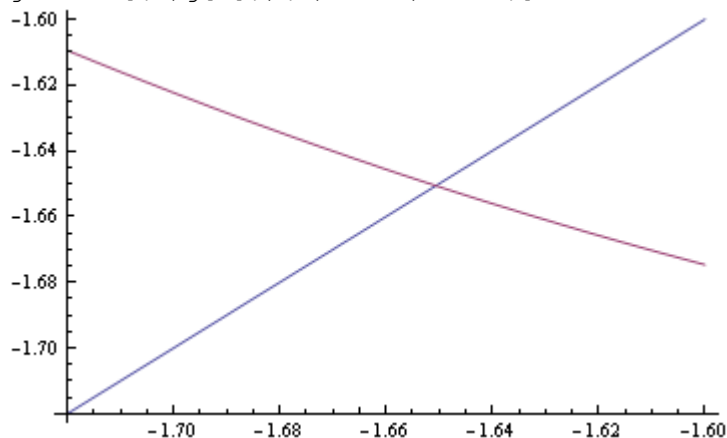
```
g1=ListLinePlot[L]
```



Con lo que podemos ver claramente el proceso oscilatorio convergente, observe que la gráfica resultante ha sido guardada en la variable "g1".

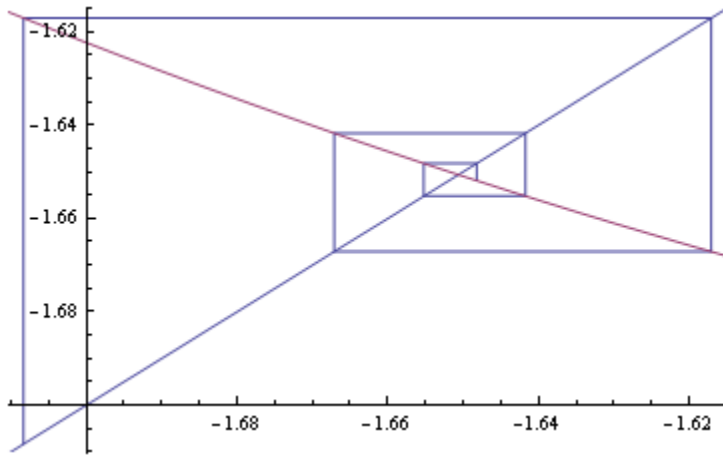
Para apreciar mejor el proceso graficamos la función (y la recta $x=g$) aproximadamente en el mismo intervalo (es decir entre $x=-1.72$ y $x=-1.60$) y guardamos la gráfica resultante en la variable "g2":

```
g2=Plot[{x,g[x]},{x,-1.72,-1.60}]
```



Finalmente mostramos las dos gráficas "g1" y "g2" para poder observar como el proceso oscilatorio se aproxima con cada iteración a la solución (es decir a la intersección):

```
Show[g1,g2]
```



Por claridad se ha mostrado el proceso para crear la gráfica paso a paso, sin embargo, es posible realizar la gráfica con una sola instrucción:

```
Show[ListLinePlot[L],Plot[{x,g[x]},{x,-1.72,-1.60}]]
```

Obteniéndose por su puesto el mismo resultado.

2. Programe una función que resuelva el siguiente sistema de ecuaciones no lineales, devolviendo el valor de "x" con el método de Sustitución Directa, calculando el resultado con 9 dígitos de precisión, sin tomar en cuenta la exactitud. Para estimar el valor inicial programe el sistema de ecuaciones en función de "x" y grafique la función entre 0 y 14. Luego grafique las 10 primeras iteraciones cuando el resultado es calculado empleando un valor inicial igual a 1.75, luego haga lo mismo con los primeros 6 puntos cuando se emplea un valor inicial igual a 1.72.

$$3x^{2.1} - 5y = 7.0$$

$$y^{1.2} + 4z = 14.3$$

$$x + y^2 + z^2 = 14.0$$

Primero colocamos una de las ecuaciones del sistema en la forma $x=g(x)$, y las otras en función de la variable independiente:

$$\begin{aligned} x &= g(x) = 14.0 - y^2 - z^2 \\ y &= \frac{3x^{2.1} - 7.0}{5} \\ z &= \frac{14.3 - y^{1.2}}{4} \end{aligned}$$

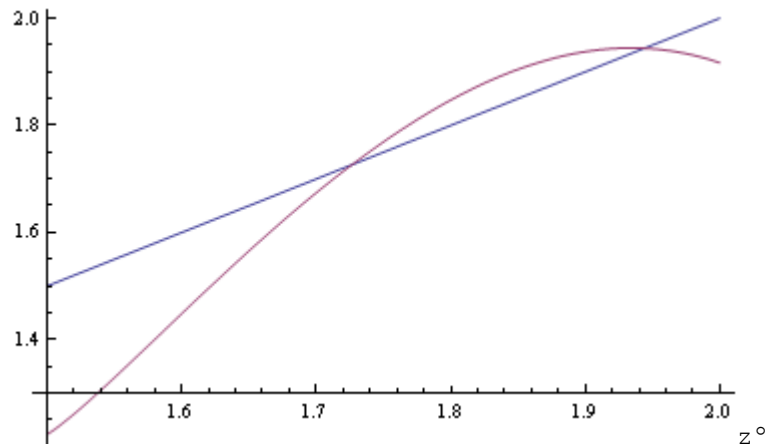
Ahora importamos el paquete "Algel" (si todavía no lo hemos hecho) y programamos la función:

```
Get["Algel`",Path->"C:\MAT205 "]
```

```
g[x_]:=Module[{y,z},
  y=(3*x^2.1-7.)/5;
  z=(14.3-y^1.2)/4;
  14-y^2-z^2]
```

Graficamos la función para ver el lugar de la solución:

```
Plot[{x,g[x]},{x,1.5,2.0}]
```



Como se puede ver, existen dos soluciones, una cerca a 1.7 y la otra cerca a 1.9. Empleando un valor inicial igual a 1.9 obtenemos:

```
NumberForm[SusDir[g,1.9,AccuracyGoal->Infinity],9]
```

```
1.94379708
```

Pero con un valor inicial igual a 1.7, Mathematica se queda iterando y después de un prolongado tiempo muestra algunos mensajes de error:

```
NumberForm[SusDir[g,1.7,AccuracyGoal->Infinity],9]
```

```
General::ovfl: Overflow occurred in computation. >>
```

```
...
```

```
General::stop: Further output of General::ovfl will be suppressed during this calculation. >>
```

Analizando la gráfica y tomando en cuenta la forma en que opera el método de Sustitución Directa, podemos ver que si se da un valor inicial por debajo de la primera raíz, en cada iteración se alejará más y más de la solución (proceso divergente) y si se da un valor por encima de la primera raíz igualmente se alejará de esta raíz y llegará a la segunda, de manera que con este método no es posible encontrar la primera raíz.

Para ver con más claridad lo que ocurre veamos los valores las 10 primeras iteración cuando se emplea un valor inicial igual a 1.75:

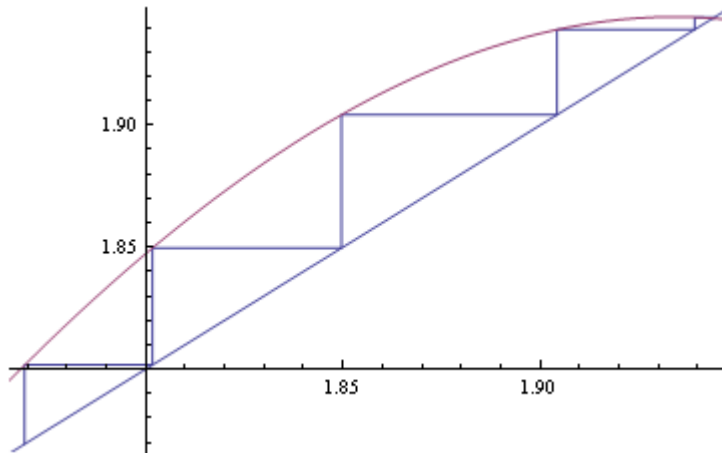
```
Off[SusDir::maxit]
L=Table[SusDir[g,1.75,MaxIterations->i],{i,10}]
{1.76931,1.80169,1.84965,1.9043,1.93919,
 1.94422,1.94375,1.9438,1.9438,1.9438}
```

Como vemos, en cada iteración los valores se alejan de la primera raíz y convergen hacia la segunda. Para ver con mayor claridad el proceso graficamos estos valores, para ello generamos primero los puntos:

```
L=Flatten[Table[{{L[[i]],L[[i]]},{L[[i]],L[[i+1]]}},{i,9}],1]
{{1.76931,1.76931},{1.76931,1.80169},{1.80169,1.80169},{1.80169,1.84965},
 {1.84965,1.84965},{1.84965,1.9043},{1.9043,1.9043},{1.9043,1.93919},
 {1.93919,1.93919},{1.93919,1.94422},{1.94422,1.94422},{1.94422,1.94375},
 {1.94375,1.94375},{1.94375,1.9438},{1.9438,1.9438},{1.9438,1.9438},
 {1.9438,1.9438},{1.9438,1.9438}}
```

Entonces graficamos los puntos y las curvas:

```
Show[ListLinePlot[L],Plot[{x,g[x]},{x,1.76,1.95}]]
```



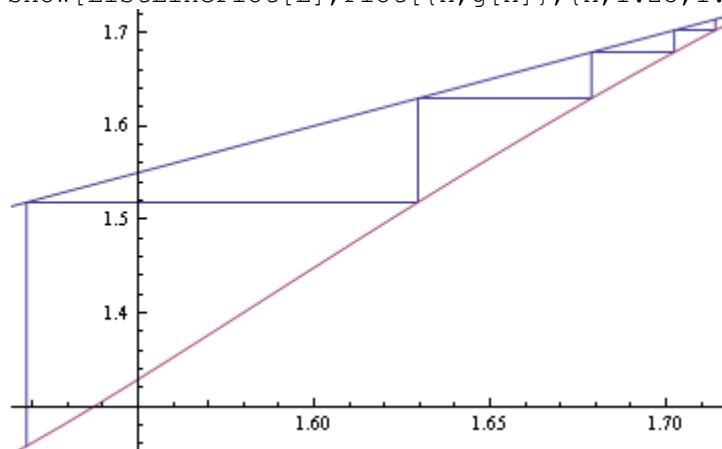
Como vemos se trata de un proceso convergente (hacia la segunda raíz).

Ahora veamos los valores intermedios que se obtienen en las primeras 7 iteraciones cuando se emplea un valor inicial igual a 1.72 (por debajo de la primera raíz):

```
L=Table[SusDir[g,1.72,MaxIterations->i],{i,7}]
{1.71401,1.7023,1.67881,1.62952,1.51839,1.25773,0.509344 -0.388194 i}
```

Como se puede ver, en este caso, en cada iteración los valores se alejan más y más de la raíz, por lo que el proceso es divergente, pero además a partir de la séptima iteración los valores son imaginarios (siendo esta la razón por la cual Mathematica demora tanto tiempo en mostrar los mensajes de error). Para ver con más claridad el proceso graficamos los valores obtenidos en las primeras 6 iteraciones (donde los valores son todavía reales):

```
L=Table[SusDir[g,1.72,MaxIterations->i],{i,6}]
{1.71401,1.7023,1.67881,1.62952,1.51839,1.25773}
L=Flatten[Table[{{L[[i]],L[[i]]},{L[[i]],L[[i+1]]}},{i,5}],1]
{{1.71401,1.71401},{1.71401,1.7023},{1.7023,1.7023},{1.7023,1.67881},
{1.67881,1.67881},{1.67881,1.62952},{1.62952,1.62952},{1.62952,1.51839},
{1.51839,1.51839},{1.51839,1.25773}}
Show[ListLinePlot[L],Plot[{x,g[x]},{x,1.25,1.72}]]
```



Vemos entonces un proceso divergente, donde en cada iteración (cada escalón) el proceso se aleja más y más de la raíz (solución).

Finalmente, para que el mensaje de límite de iteraciones superado se muestre (y no se interprete como solución correcta una incorrecta), volvemos a encender el rótulo "maxit" (con "On"):

On[SusDir::maxit]

3. Programe una función que resuelva la ecuación que se presenta al final de la pregunta devolviendo los valores de " L_1 ", " x_1 ", " V_1 " y " y_1 ", donde " L_0 ", " x_0 ", " V_0 " y " y_0 " son parámetros opcionales con valores por defecto iguales a: " $L_0=300$ ", " $x_0=0$ ", " $V_0=100$ ", " $y_0=0.2$ ". El valor inicial asumido (x_{1i}) también debe ser un parámetro opcional y para tener una idea de su valor inicial debe programar la función (empleando los valores por defecto) y graficarla entre 0 y 1. El resultado debe ser calculado con 14 dígitos de precisión o exactitud (lo que ocurra primero).

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0 = 0$$

$$L_1 = \frac{L_c}{1 - x_1}$$

$$V_1 = M - L_1$$

$$y_1 = 1.420x_1$$

$$L_c = L_0(1 - x_0)$$

$$M = L_0 + V_0$$

$$C_0 = L_0x_0 + V_0y_0$$

Primero colocamos la ecuación en la forma $x=g(x)$:

$$x_1 = \frac{C_0 - V_1 * y_1}{L_1} = g(x_1)$$

Importamos el paquete "Algel`" (si aún no lo hemos hecho):

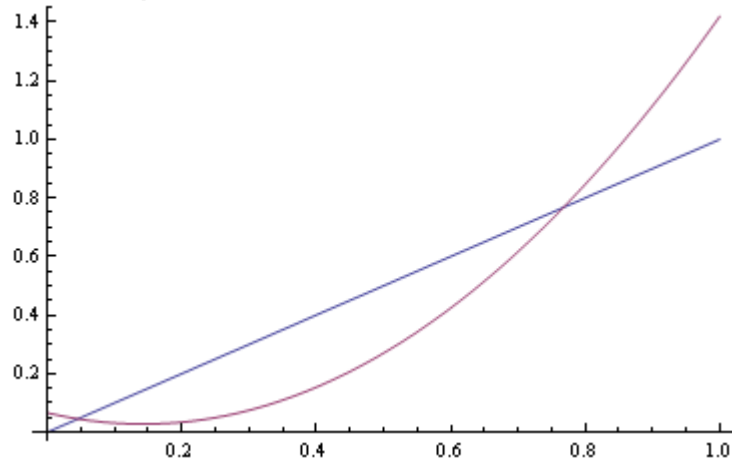
```
Get["Algel`", Path->"C:\MAT205 "]
```

Programamos la función con sus valores por defecto:

```
g[x1_]:=With[{L0=300,x0=0,V0=100,
y0=0.2},Module[{L1,V1,y1,C0,M,Lc},
C0=L0*x0+V0*y0;M=L0+V0;Lc=L0*(1-x0);
L1=Lc/(1-x1);V1=M-L1;y1=1.420*x1;
(C0-V1*y1)/L1]]
```

Y la graficamos entre 0 y 1:

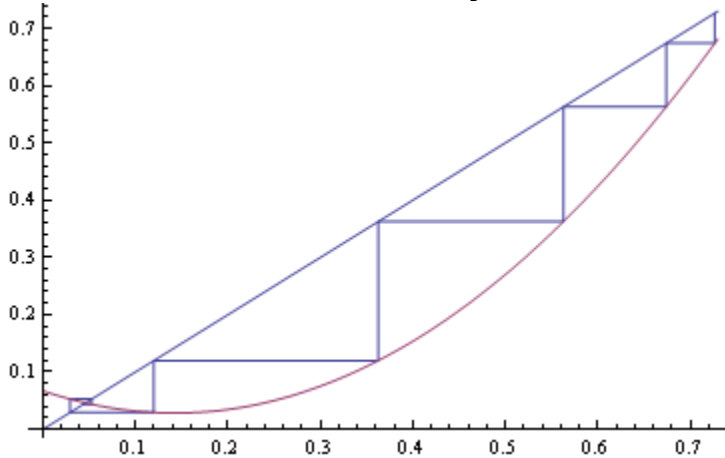
```
Plot[{x,g[x]},{x,0,1}]
```



Como se puede ver, en este caso también existen dos soluciones, una cerca a 0.05 y otra cerca a 0.75. Por la forma de la curva podemos deducir que en este caso el método sólo convergerá hacia la primera raíz pero nunca a la segunda. Lo que podemos ver con mayor claridad si graficamos las primeras 10

iteraciones comenzando con un valor inicial igual a 0.75 (que debería converger hacia la segunda raíz por estar cerca a la misma):

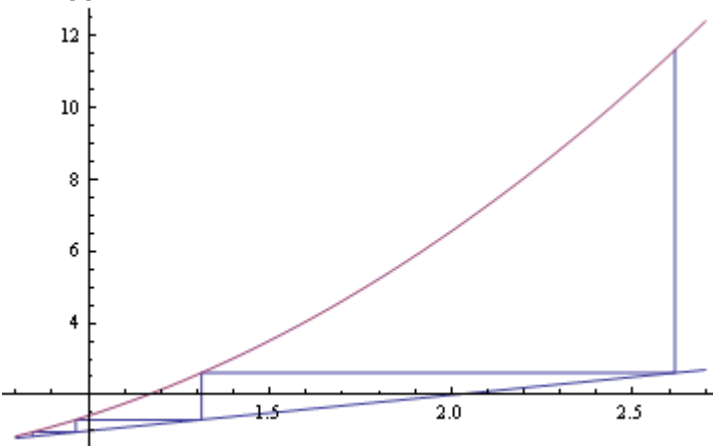
```
Off[SusDir::maxit]
L=Table[SusDir[g,0.75,MaxIterations->i],{i,10}]
{0.726667,0.674031,0.562865,0.362559,0.119762,0.0291512,
 0.052534,0.0435236,0.0467505,0.0455595}
L=Flatten[Table[{{L[[i]],L[[i]]},{L[[i]],L[[i+1]]}},{i,9}],1]
{{0.726667,0.726667},{0.726667,0.674031},{0.674031,0.674031},
 {0.674031,0.562865},{0.562865,0.562865},{0.562865,0.362559},
 {0.362559,0.362559},{0.362559,0.119762},{0.119762,0.119762},
 {0.119762,0.0291512},{0.0291512,0.0291512},{0.0291512,0.052534},
 {0.052534,0.052534},{0.052534,0.0435236},{0.0435236,0.0435236},
 {0.0435236,0.0467505},{0.0467505,0.0467505},{0.0467505,0.0455595}}
Show[ListLinePlot[L],Plot[{x,g[x]},{x,0,0.73}]]
```



Vemos que el método diverge de la segunda raíz, convergiendo hacia la primera (siendo oscilatorio cerca a la solución).

Ahora veamos lo que sucede con las primeras 5 iteraciones, cuando se busca la solución con un valor inicial igual a 0.8 (que también debería converger hacia la segunda raíz, por estar cerca a la misma):

```
L=Table[SusDir[g,0.8,MaxIterations->i],{i,5}]
{0.8464,0.965981,1.31174,2.61613,11.6122}
L=Flatten[Table[{{L[[i]],L[[i]]},{L[[i]],L[[i+1]]}},{i,4}],1]
{{0.8464,0.8464},{0.8464,0.965981},{0.965981,0.965981},{0.965981,1.31174},
 {1.31174,1.31174},{1.31174,2.61613},{2.61613,2.61613},{2.61613,11.6122}}
Show[ListLinePlot[L,PlotRange->All],Plot[{x,g[x]},{x,0.8,2.7},PlotRange->All]]
```



Que como se ve es un proceso divergente que se aleja rápidamente de la solución.

Ahora que sabemos que el método sólo converge hacia una de las raíces y que el valor inicial no debe ser muy alto, programamos la función empleando para x_{i1} un valor inicial por defecto igual a 0.5:

```
Remove[f]
Options[f]={L0->300,x0->0,V0->100,y0->0.2,xli->0.5}
{L0->300,x0->0,V0->100,y0->0.2,xli->0.5}
f[OptionsPattern[]]:=Block[{L0,x0,V0,y0,xli},
  {L0,x0,V0,y0,xli}=OptionValue[{L0,x0,V0,y0,xli}];
  Module[{L1,x1,V1,y1,C0,M,Lc,g},
    C0=L0*x0+V0*y0;
    M=L0+V0;
    Lc=L0*(1-x0);
    g:=(L1=Lc/(1-x1);
      V1=M-L1;
      y1=1.420*x1;
      (C0-V1*y1)/L1);
    x1=SusDir[g,xli];
    {Symbol["L1"]->L1,Symbol["x1"]->x1,
      Symbol["V1"]->V1,Symbol["y1"]->y1}]]
```

Ahora encontramos las soluciones con los valores por defecto:

```
f[]
{L1->314.425,x1->0.0458777,V1->85.5749,y1->0.0651464}
```

Si empleamos otro valor inicial (menor o igual a 0.75) converge también a la misma solución (como ya vimos previamente):

```
f[xli->0.7]
{L1->314.425,x1->0.0458777,V1->85.5749,y1->0.0651464}
```

11.1.3. Ejercicios

1. Resuelva el ejercicio 1, del tema 10, empleando el método de Sustitución Directa y grafique las 6 primeras iteraciones.
2. Resuelva el ejercicio 4, del tema 10, empleando el método de Sustitución Directa y grafique las 7 primeras iteraciones.
3. Resuelva el ejercicio 5, del tema 10, empleando el método de Sustitución Directa.
4. Resuelva el ejercicio 9, del tema 10, empleando el método de Sustitución Directa.
5. Resuelva el ejercicio 11, del tema 10, empleando el método de Sustitución Directa.

11.2. Método de Wegstein

El método de Wegstein, al igual que el método de sustitución directa, requiere que la ecuación algebraica sea colocada en la forma $x=g(x)$.

El algoritmo básico del método es el siguiente:

- a) Se asume un valor inicial: x_1 .
- b) Con x_1 se evalúa la función: $g_1=g(x_1)$.
- c) x_2 toma el valor de g_1 : $x_2=g_1$.
- d) Con x_2 se evalúa la función: $g_2=g(x_2)$.

- e) Con la ecuación de Wegstein se calcula x_3 y se evalúa la función: $g_3 = g(x_3)$.
- f) Si g_3 es aproximadamente igual a x_3 el proceso concluye siendo la respuesta g_3 .
- g) Se hace un cambio de variables: $x_1 = x_2$, $x_2 = x_3$, $g_1 = g_2$, $g_2 = g_3$ y se repite el proceso desde el paso "e".

Gráficamente el método sigue el proceso que se muestra en la figura 3.3, y como se puede ver en la misma, el nuevo valor se encuentra en la intersección entre la recta que pasa por los puntos (x_1, g_1) , (x_2, g_2) y la recta $x = g$.

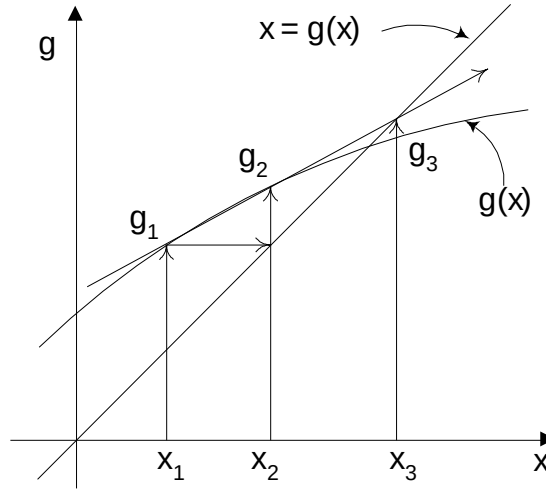


Figura 3.3. Método de Wegstein

La ecuación de la línea recta es:

$$g = a + bx \quad (3.3)$$

Por lo tanto la pendiente de la recta que pasa a través de los dos puntos (b) es:

$$\left. \begin{aligned} g_1 &= a + bx_1 \\ g_2 &= a + bx_2 \end{aligned} \right\} b = \frac{g_1 - g_2}{x_1 - x_2} \quad (3.4)$$

Entonces el valor de la ordenada en el origen (a) es:

$$g_1 = a + \frac{g_1 - g_2}{x_1 - x_2} x_1 \Rightarrow a = g_1 - \frac{g_1 - g_2}{x_1 - x_2} x_1 = \frac{g_1 x_1 - g_1 x_2 - g_1 x_1 + g_2 x_1}{x_1 - x_2} \Rightarrow a = \frac{g_2 x_1 - g_1 x_2}{x_1 - x_2} \quad (3.5)$$

Y la ecuación de la línea recta que pasa a través de los dos puntos es:

$$g = \frac{g_2 x_1 - g_1 x_2}{x_1 - x_2} + \frac{g_1 - g_2}{x_1 - x_2} x \quad (3.6)$$

En la intersección $x = x_3$ y $g = x_3$, por lo tanto:

$$\begin{aligned} x_3 &= \frac{g_2 x_1 - g_1 x_2}{x_1 - x_2} + \frac{g_1 - g_2}{x_1 - x_2} x_3 \\ x_3 \left(1 - \frac{g_1 - g_2}{x_1 - x_2} \right) &= \frac{g_2 x_1 - g_1 x_2}{x_1 - x_2} \end{aligned}$$

$$x_3 \left(\frac{x_1 - x_2 - g_1 + g_2}{x_1 - x_2} \right) = \frac{g_2 x_1 - g_1 x_2}{x_1 - x_2}$$

$$x_3 = \frac{g_2 x_1 - g_1 x_2}{g_2 - g_1 + x_1 - x_2} \quad (3.7)$$

Que es la ecuación del método de *Wegstein*.

Con excepción del cálculo de los dos primeros puntos y el empleo de la ecuación (3.7), la lógica del método es esencialmente la misma que el de Sustitución Directa. Por ejemplo para resolver la siguiente ecuación por este método:

$$f(x) = 2x^2 + 1 - e^x = 0 \quad (3.8)$$

La colocamos primero en la forma $x=g(x)$:

$$x = g(x) = \sqrt{\frac{e^x - 1}{2}}$$

Y creamos un módulo donde programamos la función y seguimos el algoritmo básico del método, siendo el valor inicial asumido 1.1 y terminando el proceso cuando x_3 y g_3 son iguales en 12 dígitos:

```
Module[{g,x1,x2,x3,g1,g2,g3},
  g[x_]:=Sqrt[(Exp[x]-1)/2];
  x1=1.1;g1=g[x1];x2=g1;g2=g[x2];
  Label[1];
  x3=(g2*x1-g1*x2)/(g2-g1+x1-x2);
  g3=g[x3];
  If[Abs[x3/g3-1]<10^-12,Goto[2]];
  x1=x2;x2=x3;g1=g2;g2=g3;Goto[1];
  Label[2];g3]//InputForm
```

0.7408504297811536

Por supuesto podemos emplear también otra instrucción (aparte de "Goto") para el ciclo iterativo. Por ejemplo con "For" la solución es:

```
Module[{g,x1,x2,x3,g1,g2,g3},
  g[x_]:=Sqrt[(Exp[x]-1)/2];
  x1=1.1;g1=g[x1];x2=g1;g2=g[x2];
  For[x3=x2;g3=g2,
    Abs[x3/g3-1]>=10^-12,
    x1=x2;x2=x3;g1=g2;g2=g3,
    x3=(g2*x1-g1*x2)/(g2-g1+x1-x2);
    g3=g[x3]];
  g3]//InputForm
```

0.7408504297811536

Que sigue la misma lógica y devuelve el mismo resultado.

11.2.1. Programa

Al igual que ocurre con el método de Sustitución Directa, al programar el método no es suficiente el algoritmo básico, sino que se deben tomar en cuenta otros parámetros como el límite de iteraciones, la precisión y la exactitud.

El algoritmo que toma en cuenta dichos aspectos es el siguiente:

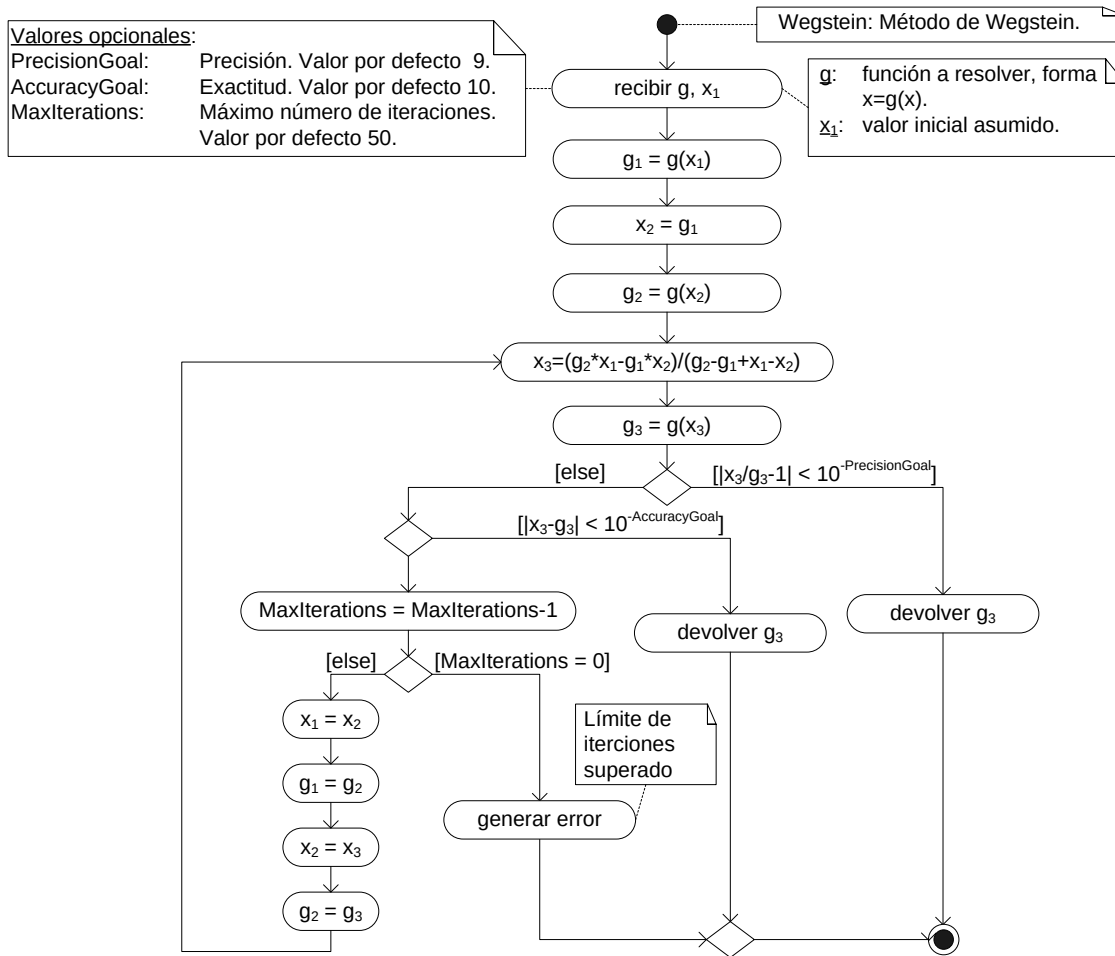


Figura 3.4. Diagrama de actividades para el método de Wegstein

Ahora abrimos el paquete "Alge1" y en el sector público del mismo (entre "BeginPackage" y "Begin") añadimos (en una celda) el rótulo "usage" del método y en el sector privado (entre "Begin" y "End") añadimos, en celdas independientes, las opciones, el rótulo "maxit" y el programa del método (siguiendo el algoritmo de la figura 3.4):

```

BeginPackage["Alge1`"]
...
Wegstein::usage="Wegstein[g,x1] resuelve
  la función \"g\", que se encuentra en la
  forma \"x=g(x)\" empleando el método de
  Wegstein con un valor inicial igual a \"x1\"";
...
Begin["`Private`"]
...
Options[Wegstein]={PrecisionGoal->9,
  AccuracyGoal->10,MaxIterations->50};

Wegstein::maxit="Se ha alcanzado el límite
  de iteraciones: `1`, sin haber encontrado la
  solución";

Wegstein[g_,x0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x0,x2,x3,g1,g2,g3},
  {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,

```

```

    MaxIterations}}];
li=mi;
g1=g[x1];
x2=g1;
g2=g[x2];
While[True,
  x3=(g2*x1-g1*x2)/(g2-g1+x1-x2);
  g3=g[x3];
  Which[Abs[x3/g3-1]<10^-pg,Return[g3],
    Abs[x3-g3]<10^-ag,Return[g3],
    --mi==0,Message[Wegstein::maxit,li];Return[g3]];
  x1=x2;g1=g2;x2=x3;g2=g3]]
...
End[]
EndPackage[]

```

Una vez añadidas estas instrucciones al paquete "Algel", guardamos los cambios("File->Save").

Ahora podemos probar el método abriendo un nuevo cuaderno (File->New->Notebook) e importando en el mismo el paquete "Algel":

```
Get["Algel",Path->"C:\ MAT205"]
```

Entonces podemos emplear el método para resolver la ecuación 3.8 (empleando los valores por defecto):

```

g[x_]:=Sqrt[(Exp[x]-1)/2]
Wegstein[g,1.1]//InputForm
0.7408504297532585

```

Podemos encontrar la solución con 10 dígitos de precisión (sin importar la exactitud):

```

Wegstein[g,1.1,PrecisionGoal->10,AccuracyGoal->Infinity]//InputForm
0.7408504297532585

```

Si reducimos a 3 el límite de iteraciones obtenemos el mensaje de error respectivo:

```

Wegstein[g,1.1,MaxIterations->3]
Wegstein::maxit: Se ha alcanzado el límite
  de iteraciones: 3, sin haber encontrado la
  solución
0.740851

```

En este caso no podemos obtener directamente los puntos intermedios del proceso iterativo a menos que modifiquemos el programa, pues el nuevo valor de prueba: " x_3 ", no es " g_3 ", sino que es calculado con la ecuación de Wegstein. Para ver estos puntos intermedios crearemos otra versión del método de Wegstein, la cual también añadiremos al paquete "Algel" y que denominaremos "Wegsteinb":

```
BeginPackage["Algel"]
```

```

...
Wegsteinb::usage="Wegstein[g,{x,x1}] resuelve
  la función \"g\", que se encuentra en la
  forma \"x=g(x)\" empleando el método de
  Wegstein con un valor inicial igual a \"x1\",
  devolviendo la solución y una lista de los
  valores de cada iteración.";
...
Begin["`Private`"]

```

```

...
Options[Wegsteinb]={PrecisionGoal->9,
  AccuracyGoal->10,MaxIterations->50};

Wegsteinb[g_,x0_,OptionsPattern[]]:=
Reap[Catch[Block[{pg,ag,mi,li,x1=x0,x2,x3,g1,g2,g3},
  {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
    MaxIterations}]];
  li=mi;
  g1=g[x1];Sow[{x1,g1}];
  x2=g1;Sow[{x2,g1}];
  g2=g[x2];Sow[{x2,g2}];
  While[True,
    x3=(g2*x1-g1*x2)/(g2-g1+x1-x2);
    g3=g[x3];Sow[{x3,g3}];
    Which[Abs[x3/g3-1]<10^-pg,Throw[g3],
      Abs[x3-g3]<10^-ag,Throw[g3],
      --mi==0,Message[Wegstein::maxit,li];Throw[g3]];
    x1=x2;g1=g2;x2=x3;g2=g3]]];
...
End[]
EndPackage[]

```

Donde como se puede observar se han añadido las instrucciones "Reap", la cual captura y coloca en una lista los valores devueltos por "Sow" y "Catch" que, como ya hemos visto, captura el valor devuelto por "Throw". Resolviendo la ecuación con esta función obtenemos:

```

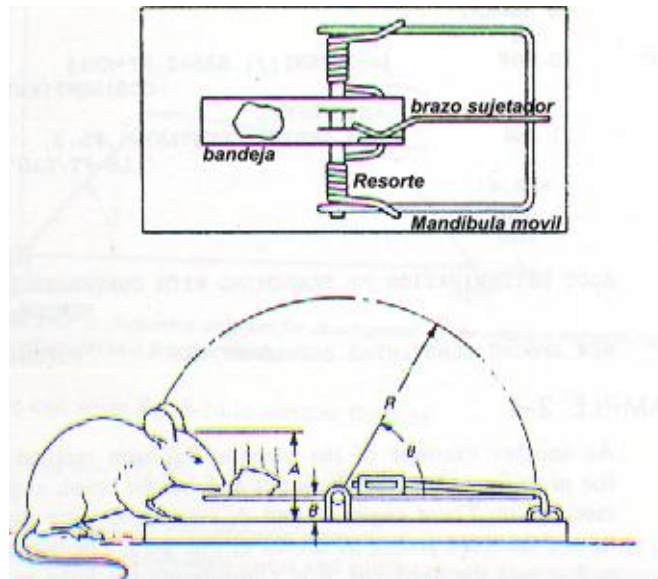
Wegsteinb[g,1.1]
{0.74085,{{{1.1,1.00104},{1.00104,1.00104},{1.00104,0.927662},
  {0.717165,0.724091},{0.74165,0.741416},{0.740851,0.740851},
  {0.74085,0.74085}}}}

```

Donde como se puede observar la convergencia se logra en cuatro iteraciones (pues los tres primeros puntos son los puntos iniciales del método).

11.2.2. Ejemplos

4. En la trampa para ratas de la figura:



La relación que existe entre el desplazamiento angular de la mandíbula y el tiempo necesario para lograr dicho desplazamiento está dada por la siguiente ecuación:

$$\theta = \frac{T_0}{k} \left(1 - \cos \left(\sqrt{\frac{k}{I_0}} t \right) \right)$$

Donde:

θ = Desplazamiento angular del dispositivo estrangulador.
 T_0 = Torque ejercido por el resorte sobre la mandíbula $\theta = 0$, (lb.pie).
 k = Constante de torsión del resorte, (lb.pie/radianes).
 I_0 = Momento de inercia de la mandíbula alrededor del eje de rotación, (lb.pie.seg²).
 t = Tiempo, seg.

Para una trampa con las dimensiones $A=1.125$ pulgadas, $B=0.5$ pulgadas y $R=3.75$ pulgadas se puede determinar que θ_c (desplazamiento angular de la mandíbula hasta tocar a la rata) es igual a 2.97 radianes y θ_k (desplazamiento angular de la mandíbula en su posición cerrada) es igual a 3.97 radianes.

El torque T_0 está relacionado con la constante del resorte por la siguiente ecuación:

$$T_0 = T_c + \theta_k k$$

Donde T_c es el torque cuando la mandíbula está cerrada.

El fabricante ha recibido quejas porque la trampa cierra muy lentamente (0.0382 segundos para golpear a la rata) permitiendo que muchas ratas escapen después de accionar la trampa. Por consiguiente es necesario instalar un nuevo resorte que golpee a la rata dos veces más rápido que la trampa actual, pero que todavía mantenga su torque en la posición cerrada ($T_c=0.625$ lb.pie) debido a que las trampas actuales se arman fácilmente con ese torque. Es también deseable mantener el mismo momento de inercia (0.0006 lb.pie.seg²) para la mandíbula, de manera que se disponga de energía adicional para matar a la rata.

Elabore una función cuyos parámetros opcionales sean " θ_c ", " θ_k ", " T_c ", " t " e " I_0 ", siendo sus valores por defecto los dados en el enunciado del problema. La función debe devolver la constante del resorte calculada con el método de Wegstein. El valor inicial asumido " k_0 " también debe ser un parámetro opcional y para estimar su valor por defecto debe graficar la función (empleando los valores por defecto) entre $k=1$ y $k=12$.

En la nueva trampa se requiere que el dispositivo recorra los 2.97 radianes ($\theta=\theta_c$) en la mitad del tiempo que emplea actualmente, es decir en: $t = 0.0382/2 = 0.0191$ s. Todas las otras constantes son proporcionadas en el planteamiento del problema.

Para el método de Wegstein, colocamos la ecuación en la forma $x=g(x)$, despejando k del denominador del segundo término:

$$k = g(k) = \frac{T_0}{\theta_c} \left(1 - \cos \left(\sqrt{\frac{k}{I_0}} t \right) \right)$$

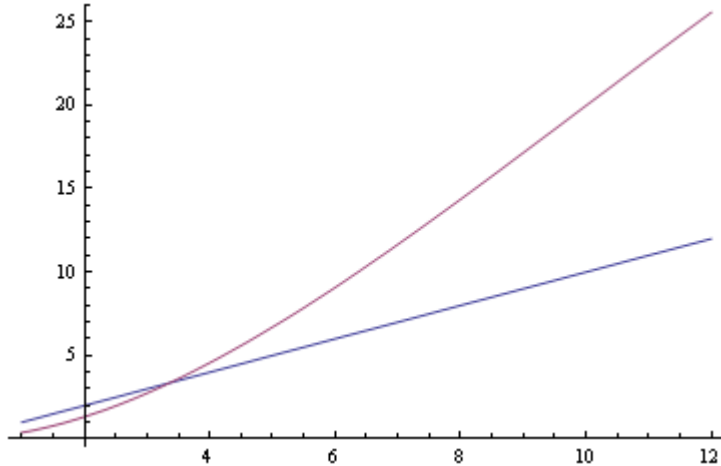
Ahora importamos el paquete "Algel" (si aún no lo hemos hecho) y programamos la función (empleando los valores por defecto):

```
Get["Algel`", Path->"C:\MAT205 "]
```

```
Remove[g]
g[k_]:=With[{θc=2.97,θk=3.27,Tc=0.625,
  t=0.0191,I0=0.0006},
  Module[{T0=Tc+θk*k},
    T0/θc*(1-Cos[Sqrt[k/I0]*t])]]
```

Graficamos la función entre $k=0$ y $k=12$ para tener una idea del valor inicial:

```
Plot[{k,g[k]},{k,1,12}]
```



Como la solución es cercana a 3, podemos emplear un valor inicial igual a 3.1. Sin embargo, por la forma de la curva se puede inferir que si se emplea un valor superior a 3, por ejemplo 10.1, se logrará igualmente encontrar la solución:

```
NumberForm[Wegstein[g,10.1],9]
3.36194979
```

Para ver con mayor claridad el proceso iterativo podemos graficar los primeros puntos del proceso iterativo, empleando para ello "Wegsteinb" en lugar de "Wegstein":

```
Wegsteinb[g,10.1]
{{3.36195,{{10.1,20.2574},{20.2574,20.2574},{20.2574,43.5212},
  {2.22802,1.60941},{2.69502,2.26686},{3.74497,4.06593},
  {3.29512,3.24449},{3.3564,3.35214},{3.36204,3.36211},
  {3.36195,3.36195},{3.36195,3.36195}}}}
```

Los tres últimos puntos están muy cercanos como para poder ser visualizados gráficamente, por lo que no se incluirán en la gráfica (eliminandolos con Drop):

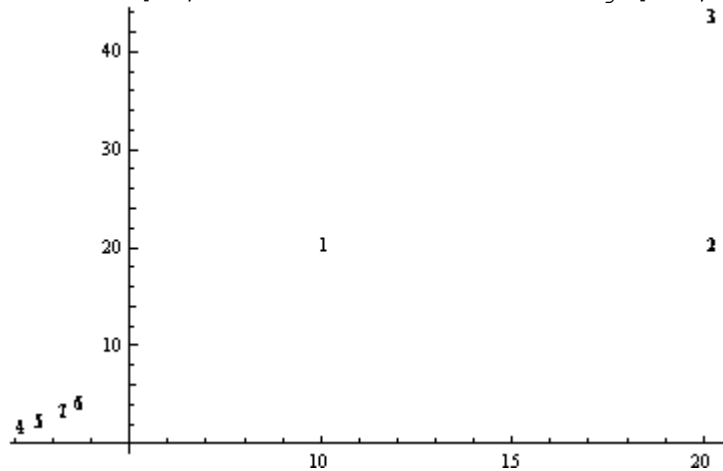
```
L=%[[2,1]];
L=Drop[L,-4]
{{10.1,20.2574},{20.2574,20.2574},{20.2574,43.5212},{2.22802,1.60941},
  {2.69502,2.26686},{3.74497,4.06593},{3.29512,3.24449}}
```

Para ver el orden en que el método va generando los diferentes puntos, graficamos cada punto por separado, colocando cada punto en una lista con la instrucción "Table":

```
L2=Table[{i},{i,L]}
{{{10.1,20.2574}},{20.2574,20.2574}},{20.2574,43.5212}},
  {{2.22802,1.60941}},{2.69502,2.26686}},{3.74497,4.06593}},
  {{3.29512,3.24449}}}
```

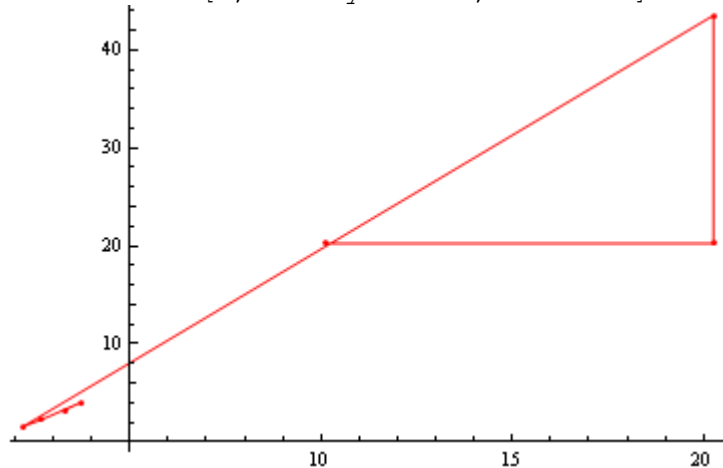
Ahora graficamos estos puntos con ListPlot, marcando cada uno con un número consecutivo, generado en este caso con CharacterRange, en color negro:

```
ListPlot[L2, PlotMarkers->CharacterRange["1", "7"], PlotStyle->Black]
```



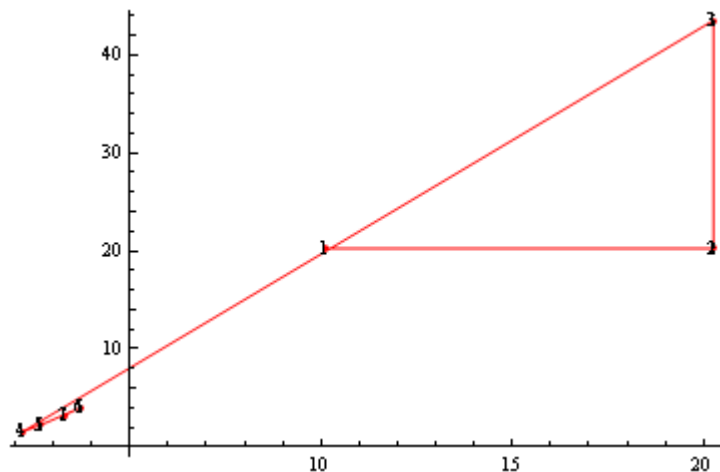
Graficamos los puntos unidos con una línea recta de color rojo:

```
ListLinePlot[L, PlotStyle->Red, Mesh->All]
```



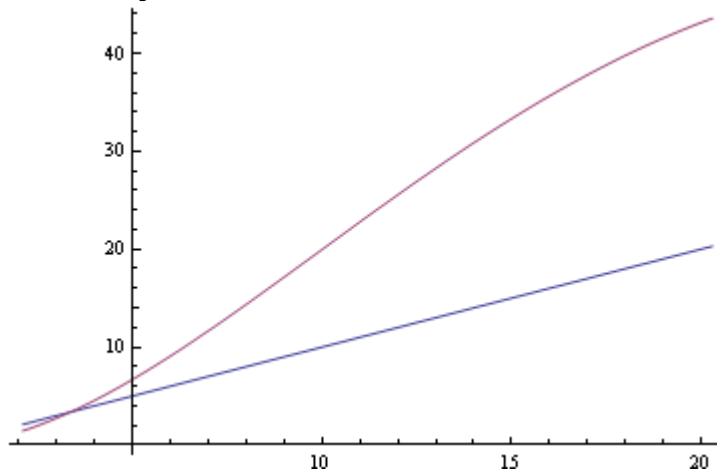
Ahora mostramos las dos gráficas juntas:

```
Show[%, %%]
```



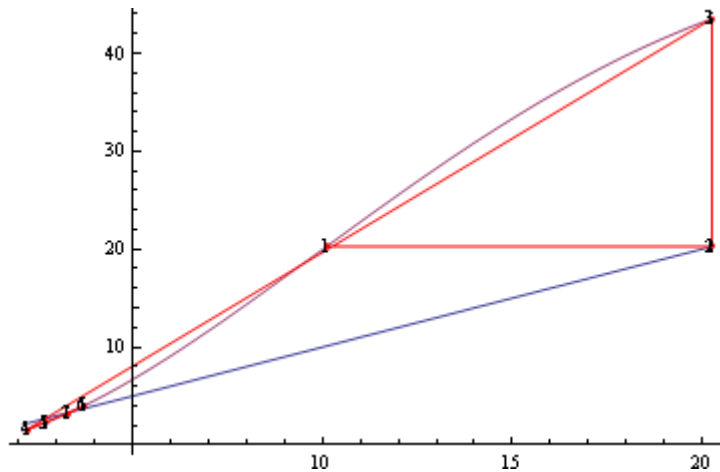
Graficamos también la función (y la recta $x=g(x)$):


```
Plot[{k,g[k]},{k,2.1,20.3}]
```



Finalmente mostramos estas dos gráficas juntas para ver todo el proceso:

```
Show[%,%%]
```



Ahora podemos ver con mayor claridad el camino que sigue el método y corroborar así que se logra una convergencia mucho más rápida que con el método de Sustitución Directa, de hecho, como se puede observar, el método de Sustitución Directa (recuerde que los tres primeros puntos corresponden en a dicho método) diverge de la solución. Por supuesto todo el proceso para crear la gráfica puede ser llevado a cabo con una sola instrucción, obteniendo el mismo resultado:

```
Show[ListPlot[L2,PlotMarkers->CharacterRange["1","7"],
PlotStyle->Black],ListLinePlot[L,PlotStyle->Red,
Mesh->All],Plot[{k,g[k]},{k,2.1,20.3}]]
```

Empleando un valor inicial asumido igual a 3.1 (cercano a la solución), programamos la solución de la función:

```
Options[fk]={θc→2.97,θk→3.27,Tc→0.625,
t→0.0191,I0→0.0006,k0→3.1};
fk[OptionsPattern[]]:=Block[{ec,ek,Tc,t,I0,k0},
{ec,ek,Tc,t,I0,k0}=OptionValue[{ec,ek,Tc,t,I0,k0}];
Module[{g,T0},
g[k_]:= (T0=Tc+ek*k; T0/ec*(1-Cos[Sqrt[k/I0]*t]));
Wegstein[g,k0]]
```

Ahora podemos encontrar la solución con los valores por defecto (con 9 dígitos de precisión):

```
NumberForm[fk[],9]
3.36194979
```

Podemos encontrar también otras soluciones para otros valores diferentes a los valores por defecto:

```
NumberForm[fk[{c→2.9,k→3.3,Tc→0.64,t→0.0195,l0→0.0007}],9]
3.63007444
```

5. Programe un módulo que resuelva el siguiente sistema de ecuaciones no lineales, devolviendo los valores de "x", "y", "z" y "w" con el método de Wegstein. Para estimar el valor inicial de "x" programe el sistema de ecuaciones en función de "x" y grafique la función entre 0 y 20. Luego muestre los valores de la variable independiente y de la función obtenidos en cada iteración.

$$\begin{aligned}x^{1/2} + z^2 &= 35 \\ y + e^{z/2} - \frac{8}{z} &= 30 \\ w + z^2 - 2z &= 31 \\ x + w + y &= 39\end{aligned}$$

Primero colocamos una de las ecuaciones del sistema en la forma $x=g(x)$ y las otras en función de "x":

$$\begin{aligned}z &= \sqrt{35 - x^{1/2}} \\ y &= 30 + \frac{8}{z} - e^{z/2} \\ w &= 31 - z^2 + 2z \\ x &= g(x) = 39 - w - y\end{aligned}$$

Ahora importamos el paquete "Algel`" (si aún no lo hemos hecho) y programamos la función:

```
Get["Algel`",Path->"C:\MAT205 "]
Remove[g]
g[x_]:=Module[{y,z,w},
  z=Sqrt[35-Sqrt[x]];
  y=30+8/z-Exp[z/2];
  w=31-z^2+2*z;
  39-w-y]
```

Graficamos la función entre 0 y 20 (la gráfica se muestra en la siguiente página):

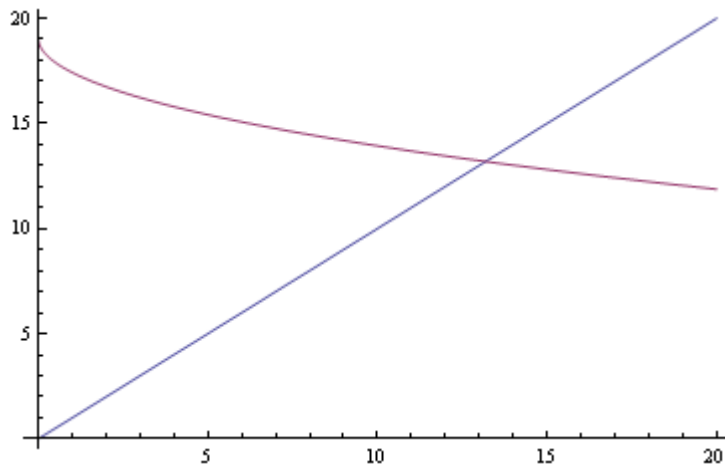
```
Plot[{x,g[x]},{x,0,20}]
```

Como se puede observar, la solución se encuentra entre 10 y 15, por lo que se puede emplear como valor inicial por ejemplo 12.5, sin embargo, por la forma de la curva es posible predecir que también se logrará convergencia con valores mayores o menores a estos, así por ejemplo con un valor inicial igual a 1.1 obtenemos:

```
NumberForm[Wegstein[g,1.1],9]
13.1891886
```

Y con un valor inicial igual a 30.1, obtenemos el mismo resultado:

```
NumberForm[Wegstein[g,30.1],9]
13.1891886
```

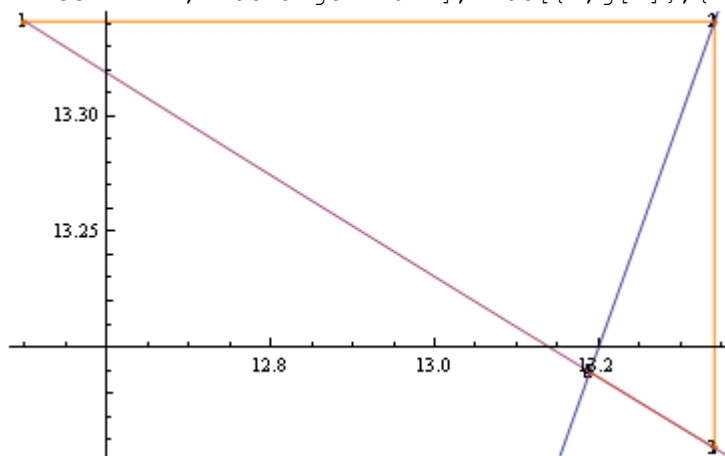


Los puntos de las diferentes iteraciones las obtenemos con "Wegsteinb":

```
NumberForm[Wegsteinb[g,12.5],9]
{13.1891886,{{{12.5,13.3410832},{13.3410832,13.3410832},
{13.3410832,13.1562772},{13.1895686,13.189106},
{13.1891884,13.1891886},{13.1891886,13.1891886}}}}
```

Y como se puede observar el método requiere sólo 3 iteraciones para alcanzar la solución con 9 dígitos de precisión. Gráficamente no es posible distinguir las iteraciones del método pues se encuentran muy cercanas:

```
L=%[[2,1]]
{{12.5,13.3411},{13.3411,13.3411},{13.3411,13.1563},{13.1896,13.1891},
{13.1892,13.1892},{13.1892,13.1892}}
L2=Table[{L[[i]]},{i,Length[L]}]
{{{12.5,13.3411}},{13.3411,13.3411}},{13.3411,13.1563}},
{{13.1896,13.1891}},{13.1892,13.1892}},{13.1892,13.1892}}}
Show[ListPlot[L2,PlotMarkers->CharacterRange["1","7"],
PlotStyle->Black,PlotRange->Full],ListLinePlot[L,PlotStyle->Orange,
Mesh->All,PlotRange->Full],Plot[{x,g[x]},{x,12.5,13.5},PlotRange->Full]]
```



Ahora resolvemos el problema (donde mostramos los valores de todas las variables) empleando un valor inicial igual 12.5:

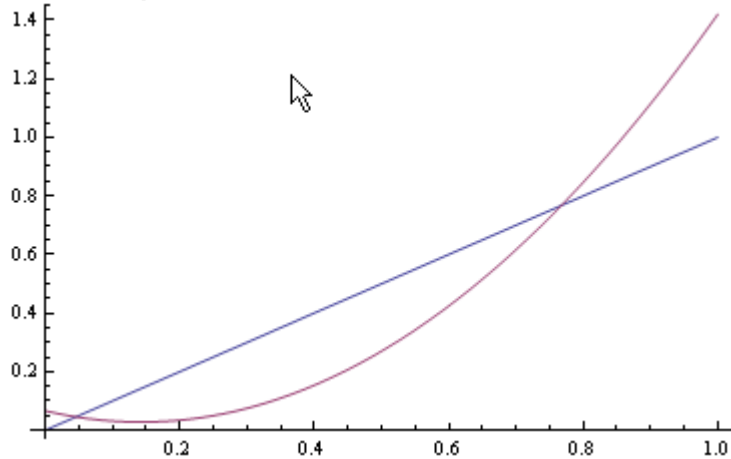
```
Module[{g,x,y,z,w},
g[x_]:= (z=Sqrt[35-Sqrt[x]]);
y=30+8/z-Exp[z/2];
w=31-z^2+2*z;
39-w-y);
x=Wegstein[g,12.5];
```

```
{Symbol["x"]->x,Symbol["y"]->y,Symbol["z"]->z,Symbol["w"]->w}]
{x->13.1892,y->14.9776,z->5.60074,w->10.8332}
```

6. Resuelva el ejemplo 3 empleando el método de Wegstein.

La función y la gráfica son las mismas que en el ejemplo 3:

```
g[x1_]:=With[{L0=300,x0=0,V0=100,y0=0.2},
Module[{L1,V1,y1,C0,M,Lc},
C0=L0*x0+V0*y0;M=L0+V0;Lc=L0*(1-x0);
L1=Lc/(1-x1);V1=M-L1;y1=1.420*x1;
(C0-V1*y1)/L1]]
Plot[{x,g[x]},{x,0,1}]
```



Empleando un valor inicial cercano a la primer solución (0.2) obtenemos:

```
NumberForm[Wegstein[g,0.2],9]
0.04587772
```

Y empleando un valor inicial cercano a la segunda solución (0.7) obtenemos:

```
NumberForm[Wegstein[g,0.7],9]
0.767502561
```

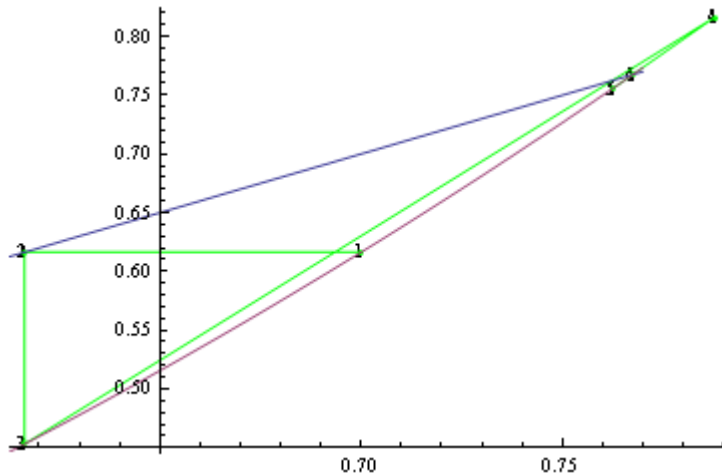
Vemos entonces que el método de Wegstein converge a la solución que se encuentra más cercana, a diferencia del método de Sustitución Directa que siempre converge hacia la primera.

Las iteraciones del método, empleando un valor inicial igual a 0.7, son:

```
NumberForm[Wegsteinb[g,0.7],9]
{0.767502561,{{0.7,0.6164},{0.6164,0.6164},{0.6164,0.453180698},
{0.787779719,0.816262389},{0.762316486,0.755281796},
{0.767359822,0.767164838},{0.767503595,0.767505007},
{0.767502562,0.767502561}}}
```

Ahora graficamos estos puntos, pero puesto que los dos últimos están muy cercanos los excluimos de la gráfica:

```
L=%[[2,1]];L=Drop[L,-2]
{{0.7,0.6164},{0.6164,0.6164},{0.6164,0.453181},{0.78778,0.816262},{0.762316,
0.755282},{0.76736,0.767165}}
L2=Table[{L[[i]]},{i,Length[L]}]
{{{0.7,0.6164},{0.6164,0.6164},{0.6164,0.453181},{0.78778,0.816262},{
0.762316,0.755282},{0.76736,0.767165}}}
Show[ListPlot[L2,PlotMarkers->CharacterRange["1","6"],PlotStyle->
Black],ListLinePlot[L,PlotStyle->Green,Mesh->All],Plot[{x1,g[x1]},{x1,0.45,0.77}]]
```



Una vez más comprobamos que el método converge rápidamente hacia la solución, después de que los tres primeros puntos (correspondientes al método de Sustitución Directa), se alejan de la misma.

Ahora resolvemos el problema, donde mostramos todas las soluciones, empleando un valor inicial igual a 0.7:

```
Remove[fx1]
Options[fx1]={L0->300,x0->0,V0->100,y0->0.2,x1i->0.7};
fx1[OptionsPattern[]]:=Block[{L0,x0,V0,y0,x1i},
  {L0,x0,V0,y0,x1i}=OptionValue[{L0,x0,V0,y0,x1i}];
  Module[{L1,x1,V1,y1,C0,M,Lc,g},
    C0=L0*x0+V0*y0;M=L0+V0;Lc=L0*(1-x0);
    g[x1_]:= (L1=Lc/(1-x1);V1=M-L1;y1=1.420*x1;
      (C0-V1*y1)/L1);
    x1=Wegstein[g,x1i];
    {Symbol["L1"]->L1,Symbol["x1"]->x1,
      Symbol["V1"]->V1,Symbol["y1"]->y1}]]
```

Haciendo correr la función con el valor inicial por defecto (0.7) obtenemos:

```
fx1[]
{L1->1290.34,x1->0.767503,V1->-890.337,y1->1.08985}
```

Que son los resultados obtenidos con la segunda raíz.

11.2.3. Ejercicios

6. Cree un paquete con el nombre "Algelc" y en el mismo programe el método de Wegstein empleando la instrucción "Do".
7. Cree un paquete con el nombre "Algeld" y en el mismo programe le método de Wegstein empleando la recursividad.
8. Cree un paquete con el nombre "Algele" y en el mismo programe le método de Wegstein empleando la instrucción "NestWhile".
9. Resuelva el ejercicio 2, del tema 10, empleando el método de Wegstein y muestre los valores de la variable independiente y de la función obtenidos en cada iteración.
10. Resuelva el ejercicio 3, del tema 10, empleando el método de Wegstein y muestre los valores de la variable independiente y de la función obtenidos en cada iteración.
11. Resuelva el ejercicio 6, del tema 10, empleando el método de Wegstein.

12. Resuelva el ejercicio 10, del tema 10, empleando el método de Wegstein.
13. Resuelva el ejercicio 12, del tema 10, empleando el método de Wegstein.

12. INCREMENTAL Y BISECCIÓN

12.1. Método Incremental

Estudiaremos ahora el primero de los métodos que permiten resolver ecuaciones algebraicas que se encuentran en la forma $f(x) = 0$, es decir cuando la función está igualada a cero (tal como se requiere para la instrucción "FindRoot"). Como ya vimos, en estos casos la solución se encuentra en la intersección entre la función " $f(x)$ " y la recta " $x=0$ ", tal como se muestra en la siguiente figura.

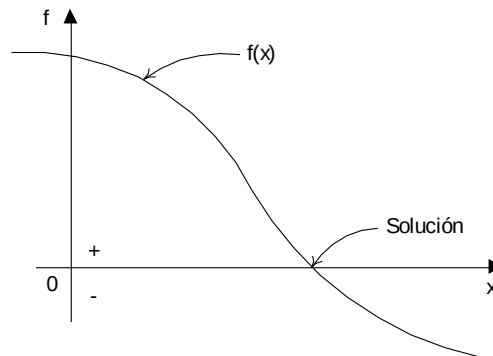


Figura 3.5. Resolución de ecuaciones algebraicas de la forma $f(x)=0$

El método incremental es, conceptualmente, el método más sencillo que existe para resolver ecuaciones de este tipo. Se basa en el hecho de que a ambos lados de la solución la función cambia de signo (como se puede ver en la figura 3.5) y básicamente este método busca el lugar donde ocurre dicho cambio.

Para ello comenzando con un valor inicial asumido (x_1) y un incremento (Δx) se van calculando valores sucesivos de x ($x_{i+1}=x_i+\Delta x$) hasta que el valor de la función cambia de signo (como se muestra en la figura 3.6).

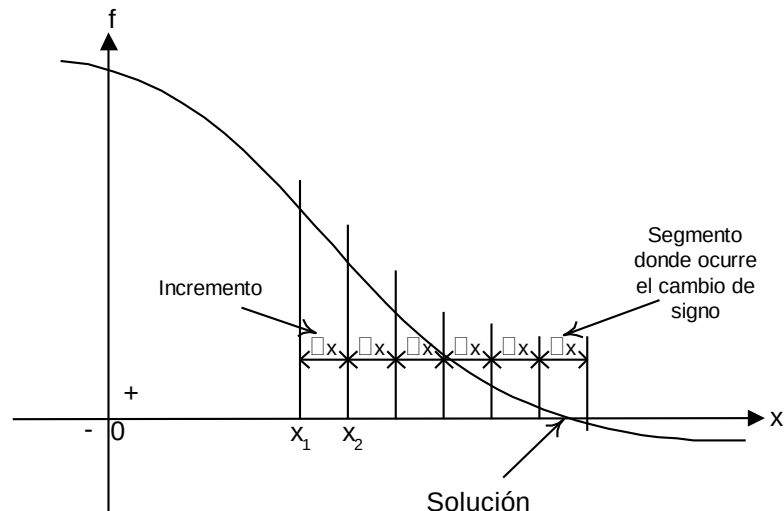


Figura 3.6. Representación gráfica del método Incremental

Una vez encontrado el segmento donde ocurre el cambio, se puede volver a repetir el método pero con un incremento más pequeño, usualmente la décima parte del incremento original, lo que permite determinar el lugar donde se

encuentra la solución con mayor exactitud. Este proceso se puede repetir hasta que el segmento donde ocurre el cambio es tan pequeño que prácticamente es la solución buscada. No obstante ello requiere demasiadas repeticiones, razón por la cual el método generalmente se emplea sólo para encontrar el segmento de solución y se recurre a otros métodos, que son más rápidos, para encontrar la solución con mayor precisión.

Como el determinar el segmento de solución constituye la aplicación más importante del método Incremental, es conveniente contar con un programa que lleve a cabo dicha tarea, siendo el algoritmo básico:

- a) Asumir un valor inicial: x_1 y un incremento: Δx .
- b) Evaluar la función para el valor asumido: $f_1=f(x_1)$.
- c) Incrementar el valor de x : $x_2=x_1+\Delta x$.
- d) Evaluar la función para el nuevo valor de x : $f_2=f(x_2)$.
- e) Si $f_1*f_2<0$, el proceso concluye, encontrándose el segmento de solución entre x_1 y x_2 .
- f) x_1 toma el valor de x_2 ($x_1=x_2$) y se repite el proceso desde el paso (c).

Por ejemplo, podemos encontrar el segmento de solución de la siguiente función (asumiendo un valor inicial igual a 1 y un incremento igual a 0.1):

$$f(x)=(52+3\sqrt{x}-8x^{0.8})^{0.36}-x=0 \quad (3.9)$$

Para ello programamos en un módulo la función y el algoritmo básico descrito previamente.

```
Module[{f,x1=1,Δx=0.1,x2,f1,f2},
  f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x;
  f1=f[x1];
  Label[c];
  x2=x1+Δx;f2=f[x2];
  If[f1*f2<0,Goto[g]];
  x1=x2;Goto[c];
  Label[g];
  {Symbol["x1"]→x1,Symbol["x2"]→x2}]
```

Ejecutando el módulo se obtiene:

```
{x1→3.6,x2→3.7}
```

Por lo tanto la solución se encuentra entre 3.6 y 3.7. Emplendo "Reap" y "Sow" podemos obtener además la lista con los valores de " x_2 " y " f_2 " de cada iteración.

```
Reap[Module[{f,x1=1,Δx=0.1,x2,f1,f2},
  f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x;
  f1=f[x1];
  Label[c];x2=x1+Δx;f2=f[x2];Sow[{x2,f2}];
  If[f1*f2<0,Goto[g]];
  x1=x2;Goto[c];
  Label[g];{Symbol["x1"]→x1,Symbol["x2"]→x2}]]
{{x1→3.6,x2→3.7},
{{ {1.1,2.88406},{1.2,2.76913},{1.3,2.65424},{1.4,2.53939},
{1.5,2.42457},{1.6,2.30977},{1.7,2.19498},{1.8,2.0802},
{1.9,1.96542},{2.,1.85065},{2.1,1.73587},{2.2,1.62107},
{2.3,1.50627},{2.4,1.39145},{2.5,1.27661},{2.6,1.16175},
{2.7,1.04685},{2.8,0.931934},{2.9,0.81698},{3.,0.70199},
{3.1,0.586962},{3.2,0.471893},{3.3,0.35678},{3.4,0.24162},
{3.5,0.126411},{3.6,0.0111496},{3.7,-0.104166}}}}}
```


12.1.1. Programa para ubicar el segmento de solución

La principal diferencia de este método, con relación a los otros que hemos estudiado y estudiaremos en este capítulo, es que, como ya hemos visto, no devuelve la solución, sino los valores entre los cuales se encuentra la solución (es decir el segmento de solución).

El algoritmo elaborado, es el siguiente:

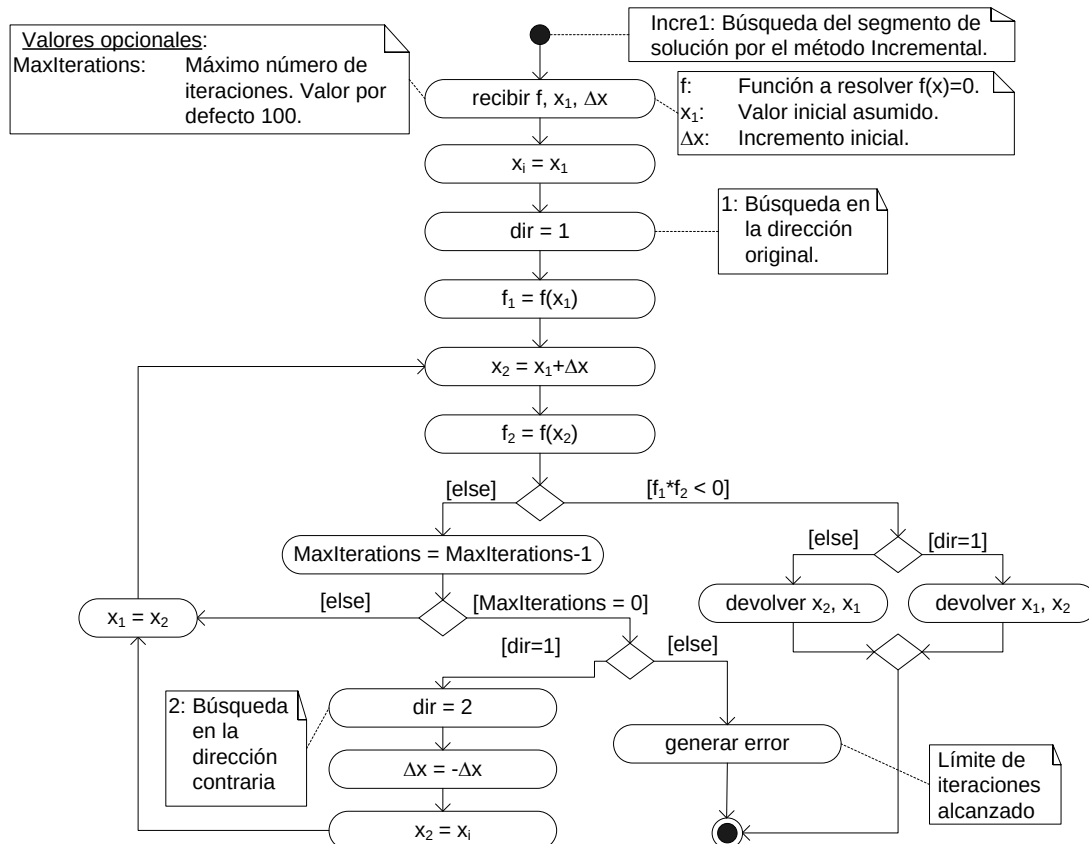


Figura 3.7. Método incremental: Determinación de segmento de solución.

Como se puede observar, en el mismo se realiza la búsqueda en dos direcciones: primero se realiza la búsqueda en la dirección correspondiente al incremento "Δx", pero si en 100 iteraciones no se encuentra el segmento de solución, entonces se realiza una búsqueda en la dirección contraria. Sólo si después de realizar ambas búsquedas no se encuentra el segmento de solución se genera un error.

En este método se requiere un límite de iteraciones por las siguientes razones: a) Con el incremento correspondiente a los valores iniciales el método puede alejarse de la solución (divergencia), b) El incremento puede ser muy pequeño y requerir demasiadas iteraciones para alcanzar el segmento de solución y c) El incremento puede ser demasiado grande y saltar a través de dos o más soluciones sin encontrarlas nunca. No son valores opcionales ni la precisión de la exactitud, porque el objetivo del método es encontrar el segmento donde se encuentra la solución y no propiamente la solución.

Como en todos los métodos el algoritmo puede ser programado empleando cualquiera de las estructuras iterativas existentes en Mathematica. Al igual que hicimos con los otros métodos, añadimos la función para este método en la unidad "Algel`":

```

BeginPackage["Algel`"]
...
Incret1::usage="Incret1[f,x1,Δx] encuentra
  el segmento de solución de la función \"f\", que
  se encuentra en la forma \"f(x)=0\" empleando el
  método Incremental con un valor inicial igual a
  \"x1\" y un incremento igual a \"Δx\"";
...
Begin["`Private`"]
...
Options[Incret1]={MaxIterations->100};

Incret1::maxit="Se ha alcanzado el límite
  de iteraciones: `1`, sin haber encontrado el
  segmento de solución";

Incret1[f_,x0_,dx0_,OptionsPattern[]]:=
Block[{mi,li,x1=x0,x2,x3,f1,f2,dir,xi,dx=dx0},
  mi=OptionValue[MaxIterations];
  li=mi;xi=x1;dir=1;
  f1=f[x1];
  While[True,
    x2=x1+dx;f2=f[x2];
    If[f1*f2<=0,Break[]];
    If[--mi==0,If[dir==1,dir=2;dx=-dx;x2=xi;mi=li,
      Message[Incret1::maxit,li];Break[]]];
    x1=x2];
  If[dir==1,{x1,x2},{x2,x1}]]
...
End[]

```

EndPackage[]

Una vez escrito el código, guardamos los cambios (File -> Save) y probamos el programa en un cuaderno nuevo (File -> New -> Notebook), donde primero importamos el paquete:

```
Get["Algel`",Path->"C:\MAT205 "]
```

Programamos la ecuación:

```
f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x
```

Y la resolvemos con el programa elaborado:

```
Incret1[f,1,0.1]
{3.6,3.7}
```

Ahora probamos el método con un valor inicial igual a 4 (que como sabemos se encuentra por encima de la raíz) y un incremento igual a 0.1, fijando el máximo de iteraciones en 50:

```
Incret1[f,4,0.1,MaxIterations->50]
{x1->3.6,x2->3.7}
```

Como vemos el método ha cambiado de dirección y ha encontrado el segmento de solución. En este ejemplo se ha fijado el límite de iteraciones en 50 porque para valores mayores de "x" la función devuelve resultados imaginarios, lo que provoca la aparición de una serie de mensajes de error, aunque finalmente se obtiene el resultado correcto (haga la prueba).

Para ver los valores calculados en cada una de las iteraciones del método programamos otra versión del mismo, al cual denominaremos "Increlb" (y que también será añadido al paquete "Alg1`":

```
BeginPackage["Alg1`"]
...

Increb::usage="Increl[f,x1,Δx] encuentra
el segmento de solución de la función \"f\", que
se encuentra en la forma \"f(x)=0\" empleando el
método Incremental con un valor inicial igual a
\"x1\" y un incremento igual a \"Δx\",
mostrando los puntos intermedios calculados.";

...

Begin["`Private`"]

...

Options[Increlb]={MaxIterations->100};

Increlb[f_,x0_,dx0_,OptionsPattern[]]:=
Reap[Block[{mi,li,x1=x0,x2,x3,f1,f2,dir,xi,dx=dx0},
mi=OptionValue[MaxIterations];
li=mi;xi=x1;dir=1;
f1=f[x1];Sow[{x1,f1}];
While[True,
x2=x1+dx;f2=f[x2];Sow[{x2,f2}];
If[f1*f2<=0,Break[]];
If[--mi==0,If[dir==1,dir=2;dx=-dx;x2=xi;mi=li,
Message[Increl::maxit,li];Break[]]];
x1=x2];
If[dir==1,{x1,x2},{x2,x1}]]]

...

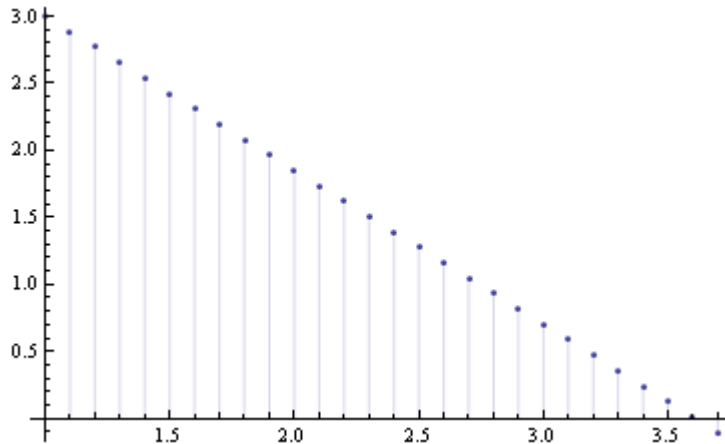
End[]
```

Resolviendo la ecuación con esta versión del programa obtenemos:

```
Increlb[f,1,0.1]
{{3.6,3.7},{1,2.99904},{1.1,2.88406},{1.2,2.76913},{1.3,2.65424},
{1.4,2.53939},{1.5,2.42457},{1.6,2.30977},{1.7,2.19498},{1.8,2.0802},
{1.9,1.96542},{2.,1.85065},{2.1,1.73587},{2.2,1.62107},{2.3,1.50627},
{2.4,1.39145},{2.5,1.27661},{2.6,1.16175},{2.7,1.04685},{2.8,0.931934},
{2.9,0.81698},{3.,0.70199},{3.1,0.586962},{3.2,0.471893},{3.3,0.35678},
{3.4,0.24162},{3.5,0.126411},{3.6,0.0111496},{3.7,-0.104166}}}
```

Podemos visualizar gráficamente estos valores con "ListPlot" y empleando "Filling" para ver más claramente cada uno de los puntos:

```
L=%[[2,1]];
ListPlot[L,Filling->Axis]
```



Puntos que pueden ser visualizados conjuntamente la gráfica de la función:

```
Show[ListPlot[L,Filling->Axis],Plot[f[x],{x,1,3.8}]]
```



En esta gráfica se hace evidente el elevado número de iteraciones que requiere el método para encontrar el segmento de solución.

Como el programa "Incretl" sólo devuelve el segmento de solución, no resolveremos ejemplos con el mismo, sino que será empleado posteriormente para obtener los valores iniciales que se requieren en otros métodos.

12.1.2. Programa para encontrar la solución

Como ya se dijo anteriormente, una vez determinado el segmento donde se encuentra la solución, se puede volver a aplicar el método pero empleando un incremento igual a la décima parte del incremento original. Entonces, como se puede observar en la figura de la siguiente página, se vuelve a determinar el segmento de solución pero con una precisión 10 veces superior a la del segmento original, o lo que es lo mismo: con un incremento igual a la décima parte del incremento original.

El proceso se puede repetir en el nuevo segmento (empleando un incremento igual a la décima parte del incremento anterior) y continuar así hasta que el resultado tenga la exactitud o precisión requerida.

En consecuencia, el algoritmo consiste básicamente en llamar repetidas veces a "Incretl", empleando en cada llamada incrementos iguales a la décima parte del incremento anterior, hasta que el segmento sea lo suficientemente pequeño como para asumir que sus límites son esencialmente iguales.

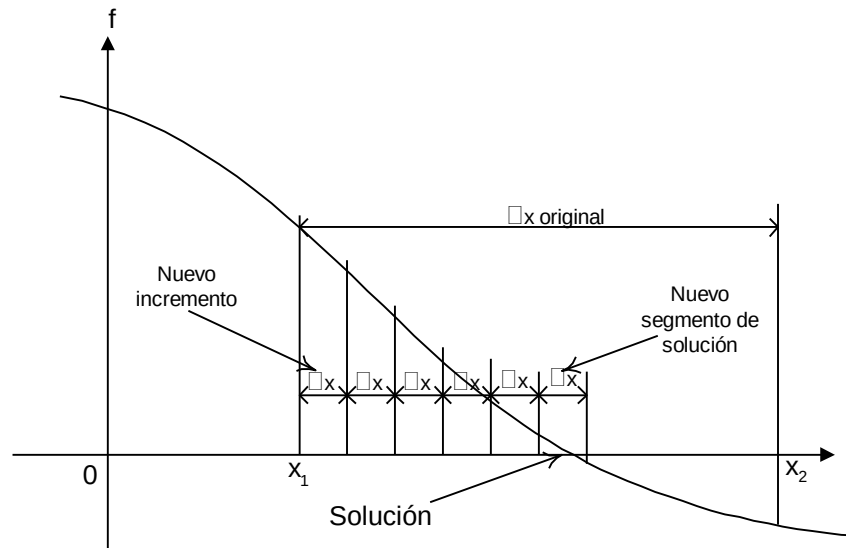


Figura 3.8. Cálculo de la solución con el método Incremental.

El algoritmo básico es el siguiente:

- Asumir un valor inicial: x_1 y un incremento: Δx .
- Con "Incret1" encontrar el segmento de solución: x_1, x_2 .
- Si Δx es menor a una tolerancia especificada, el proceso concluye siendo la respuesta el valor de "x" que acerca más la función a 0.
- Disminuir el incremento a la décima parte: $\Delta x = \Delta x / 10$ y repetir el proceso desde el paso (b).

Por ejemplo, podemos elaborar un módulo para resolver la ecuación (3.9), aplicando este algoritmo básico, con un valor inicial igual a 1, un incremento igual a 0.1 y una exactitud (tolerancia) de 12 dígitos:

```
Get["Algel`", Path->"C:\MAT205 "]
```

```
Module[{f,x1=1,dx=0.1,x2},
  f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x;
  Label[1];
  {x1,x2}=Incret1[f,x1,dx];
  If[dx<10^-12,Goto[2]];
  dx=dx/10;Goto[1];
  Label[2];
  If[Abs[f[x1]]<Abs[f[x2]],x1,x2]]//InputForm
```

```
3.6096708861400018
```

El diagrama de actividades del método, donde además se toma en cuenta la precisión, el límite de iteraciones y los dígitos de precisión, se presenta en la siguiente página. En el mismo se ha fijado el límite de iteraciones en 16 porque en cada repetición del método se mejora la precisión en un dígito y dado que por defecto sólo se trabaja con 16 dígitos (la precisión de máquina), en la práctica no es posible superar dicho límite (a menos que se cambie también los dígitos de precisión).

Al igual que los anteriores métodos escribimos el programa (siguiendo el algoritmo) en el paquete "Algel":

```
BeginPackage["Algel`"]
```

```
...
```

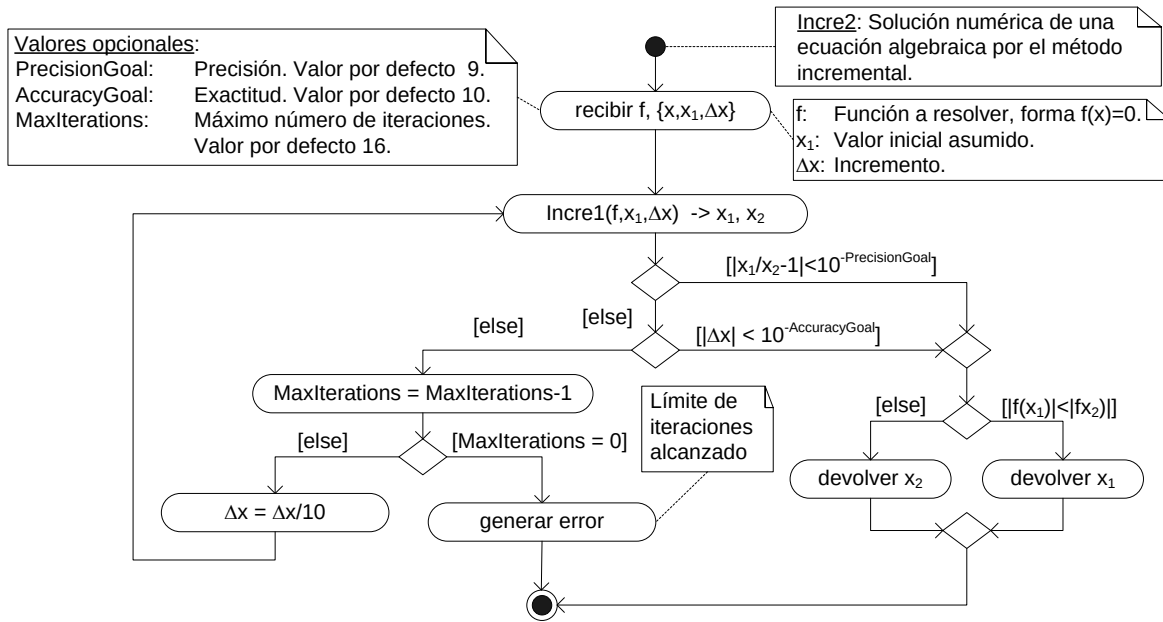


Figura 3.9. Método Incremental: Cálculo de la solución

Ince2::usage = "Ince2[f,x1,dx] encuentra la solución de la función \"f\", que se encuentra en la forma \"f(x)=0\" empleando el método Incremental con un valor inicial igual a \"x1\" y un incremento igual a \"dx\"";

...

Begin["`Private`"]

...

Options[Ince2]={PrecisionGoal->9,AccuracyGoal->10,MaxIterations->16};

Ince2::maxit="Se ha alcanzado el límite de iteraciones: `1`, sin haber encontrado la solución";

```

Ince2[f_,x0_,dx0_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x0,x2,dx=dx0},
  {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
    MaxIterations}];
  li=mi;
  While[True,
    {x1,x2}=Ince1[f,x1,dx];
    If[Abs[x1/x2-1]<10^-pg,Break[]];
    If[Abs[dx]<10^-ag,Break[]];
    If[--mi==0,Message[Ince2::maxit,li];Break[]];
    dx=dx/10;
    If[Abs[f[x1]]<Abs[f[x2]],x1,x2]]
  
```

...

```
End[]
```

```
EndPackage[]
```

Ahora importamos el paquete y probamos "Incre2" volviendo a resolver la ecuación (3.9):

```
Get["Algel`",Path->"C:\MAT205 "]
```

```
f[x_]:= (52+3*Sqrt[x]-8*x^0.8)^0.36-x
```

```
NumberForm[Incre2[f,1,0.1],9]
3.60967089
```

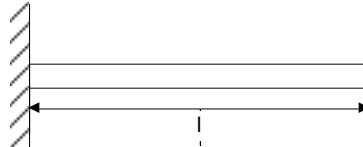
Que es el resultado con la exactitud por defecto. Con 12 dígitos de exactitud la solución es:

```
Incre2[f,1,0.1,PrecisionGoal->Infinity,AccuracyGoal->12]//InputForm
3.6096708861400018
```

Como gráficamente el método, después de ubicar el segmento de solución con "Incel", sólo reduce el tamaño del incremento, no resulta muy instructivo visualizar estos decrementos, razón por la cual no se elaborará un módulo que muestre dichos puntos.

12.1.3. Ejemplos

1. Como primer ejemplo elaboraremos una función para calcular la frecuencia natural de una viga, siendo parámetros opcionales: "l", "I", "E", "y", "A" y "g", con valores por defecto iguales a: "l=120", "I=170.6", "E=3x10⁶", "y=0.066", "A=32" y "g=386". La solución debe ser calculada con 12 dígitos de exactitud (sin importar la precisión), siendo parámetros opcionales también el valor inicial asumido (x_0) y el incremento inicial (Δx_0), con valores por defecto iguales a: $x_0=1.1$ y $\Delta x_0=1$.



Recordemos que para el método incremental la ecuación tiene que estar en la forma $f(x)=0$:

$$f(k) = \cos(kl)\cosh(kl) + 1 = 0$$

Importamos el paquete "Algel" (si todavía no lo hemos hecho):

```
Get["Algel`",Path->"C:\ MAT205"]
```

Asignamos los valores por defecto a los parámetros opcionales:

```
Options[fp]={l->120,i->170.6,e->3*10^6,y->0.066,A->32,
g->386,x0->1.1,dx0->1};
```

Y programamos la función

```
fp[OptionsPattern[]]:=
Block[{l,i,e,y,A,g,x0,dx0,a,f,k},
{ l,i,e,y,A,g,x0,dx0}=OptionValue[{l,i,e,y,A,g,x0,dx0}];
a=Sqrt[e*i*g/(A*y)];
f[p_]:= (k=Sqrt[p/a];
Cos[k*l]*Cosh[k*l]+1);
Incre2[f,x0,dx0,AccuracyGoal->12, PrecisionGoal->Infinity]]
```

Evaluando la función se obtiene la solución:

```
fp[]//InputForm
74.67669629022707
```

Por supuesto podemos emplear valores diferentes a los por defecto, así para un momento de inercia igual a 190, obtenemos:

```
fp[i->190]//InputForm
78.80837697737057
```

2. Como segundo volveremos a resolver el sistema de ecuaciones no lineales resuelto en el ejemplo 6, en este caso con 10 dígitos de precisión (sin importar la exactitud). Se deben encontrar las dos soluciones empleando valores iniciales iguales a 1.6 y 1.8, empleando en ambos casos un incremento inicial igual a 0.1.

$$\begin{aligned} 3x^{2.1} - 5y &= 7.0 \\ y^{1.2} + 4z &= 14.3 \\ x + y^2 + z^2 &= 14.0 \end{aligned}$$

Primero colocamos una de las ecuaciones del sistema en la forma $f(x)=0$:

$$\begin{aligned} f(x) &= x + y^2 + z^2 - 14 = 0 \\ y &= \frac{3x^{2.1} - 7.0}{5} \\ z &= \frac{14.3 - y^{1.2}}{4} \end{aligned}$$

Primero importamos el paquete "Algel" (si aún no lo hemos hecho):

```
Get["Algel`", Path->"C:\MAT205"]
```

Ahora programamos la función y la resolvemos con "Incr2":

```
Module[{f,x,y,z},
  f[x_]:= (y=(3*x^2.1-7)/5;
    z=(14.3-y^1.2)/4;
    x+y^2+z^2-14);
  Print["Soluciones empleando un valor inicial igual a 1.5:"];
  x=Incr2[f,1.6,0.1,PrecisionGoal->10,AccuracyGoal->Infinity];
  Print[{Symbol["x"]->x,Symbol["y"]->y,Symbol["z"]->z}];
  Print["Soluciones empleando un valor inicial igual a 1.8:"];
  x=Incr2[f,1.8,0.1,PrecisionGoal->10,AccuracyGoal->Infinity];
  Print[{Symbol["x"]->x,Symbol["y"]->y,Symbol["z"]->z}]]
```

Ejecutando el módulo se obtiene:

```
Soluciones empleando un valor inicial igual a 1.5:
{x->1.72653,y->0.488926,z->3.46907}
Soluciones empleando un valor inicial igual a 1.8:
{x->1.9438,y->1.0228,z->3.31814}
```

12.1.4. Ejercicios

1. Cree un paquete con el nombre "Algeld" y en el mismo elabore los programas para "Incr1" e "Incr2" empleando la instrucción "Nest".
2. Cree un paquete con el nombre "Algele" y en el mismo elabore los programas para "Incr1" e "Incr2" empleando la instrucción "For".
3. Cree un paquete con el nombre "Algelf" y en el mismo elabore los programas para "Incr1" e "Incr2" empleando la recursividad.

4. Resuelva el ejercicio 3, del tema 10, empleando "Incre2".
5. Resuelva el ejercicio 6, del tema 10, empleando "Incre2".
6. Resuelva el ejercicio 10, del tema 10, empleando "Incre2".
7. Resuelva el ejercicio 12, del tema 10, empleando "Incre2".

12.2. Método de la Bisección (Boltzano)

Según vimos en el anterior acápite, una vez determinado el segmento de solución con el método incremental (Incre1), se puede volver a aplicar el mismo método para encontrar la solución (Incre2), sin embargo, ello requiere demasiadas iteraciones, por lo que en la práctica normalmente solo se encuentra el segmento de solución con el método incremental (Incre1) y luego se recurre a otros métodos para encontrar la solución final. Uno de dichos métodos es el de la Bisección.

El método de la bisección requiere que se conozca el segmento de solución y, como su nombre sugiere, divide dicho segmento en 2 calculando el valor de la función en la mitad del segmento, entonces si el valor de la función es cero (o casi cero) el proceso concluye, caso contrario se vuelve a repetir el procedimiento pero trabajando ahora con la mitad del segmento inicial, aquella mitad donde se encuentra la solución, tal como se puede ver en la figura 3.10.

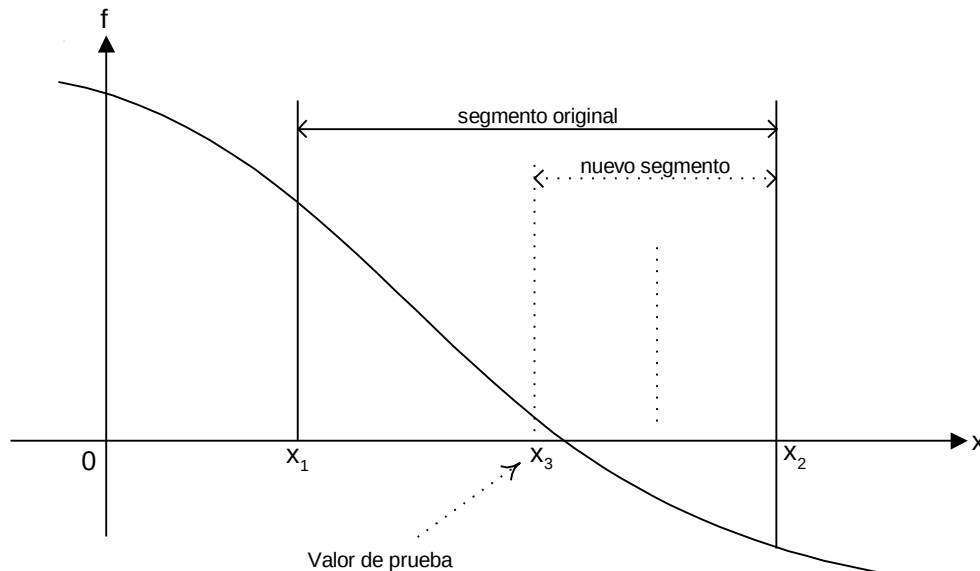


Figura 3.10. Método de la Bisección.

La parte donde se encuentra la solución se determina comparando el valor de la función al inicio del segmento con el valor de la función en el centro del segmento. Si existe cambio de signo (es decir si la multiplicación de estos valores es negativa), entonces la solución se encuentra en el segmento izquierdo, caso contrario se encuentra en el segmento derecho.

La ecuación que permite calcular el valor de la variable independiente ("x") a la mitad del segmento es:

$$x_3 = \frac{x_2 - x_1}{2} + x_1 = \frac{x_2 - x_1 + 2x_1}{2} = \frac{x_1 + x_2}{2} \quad (3.10)$$

El algoritmo básico del método de la Bisección es el siguiente:

- a) Proporcionar los valores de la variable independiente al inicio y final del segmento de solución: x_1, x_2 .
- b) Calcular el valor de la función en " x_1 ": $f_1=f(x_1)$.
- c) Calcular un nuevo valor de prueba " x_3 " con la ecuación (3.10)
- d) Calcular el valor de la función para el nuevo valor de prueba: $f_3=f(x_3)$.
- e) Si f_3 es casi cero, dentro de un margen de tolerancia predeterminado, el proceso concluye, siendo la solución x_3 .
- f) Si $f_1 \cdot f_3 < 0$, hacer que $x_2=x_3$, caso contrario hacer que $x_1=x_3$.
- g) Repetir el proceso desde el paso (c).

Siguiendo el algoritmo básico del método de la Bisección encontraremos una de las soluciones de la ecuación:

$$f(x) = x^3 + 2x^2 + 3x + 4 = 0 \quad (3.11)$$

Para encontrar el segmento de solución (los valores de x_1 y x_2) emplearemos el método "Incretl" por lo que deberemos importar primeramente el paquete "Algel".

```
Get["Algel`", Path->"C:\MAT205"]
```

```
Module[{f,x1,x2,x3,f1,f3},
f[x_]:=x^3+2*x^2+3*x+4;
{x1,x2}=Incretl[f,1.1,1];
f1=f[x1];
Label[c];
x3=(x1+x2)/2;
f3=f[x3];
If[Abs[f3]<10^-14,Goto[h]];
If[f1*f3<0,x2=x3,x1=x3];
Goto[c];
Label[h];x3]//InputForm
```

Ejecutando el programa se obtiene:

```
-1.650629191439387
```

12.2.1. Programa

El algoritmo del método, donde se toman en cuenta la precisión, exactitud y el límite de iteraciones se presenta en la figura de la siguiente página.

Al igual que en todos los métodos anteriores añadiremos el método al paquete "Algel":

```
BeginPackage["Algel`"]
```

```
...
```

```
Biseccion::usage="Biseccion[f,x1,x2] encuentra
la solución de la función \"f\", que se encuentra
en la forma \"f(x)=0\" empleando el método Incremental
en el segmento de solución comprendido entre \"x1\" y
\"x2\"";
```

```
...
```

```
Begin["`Private`"]
```

```
...
```

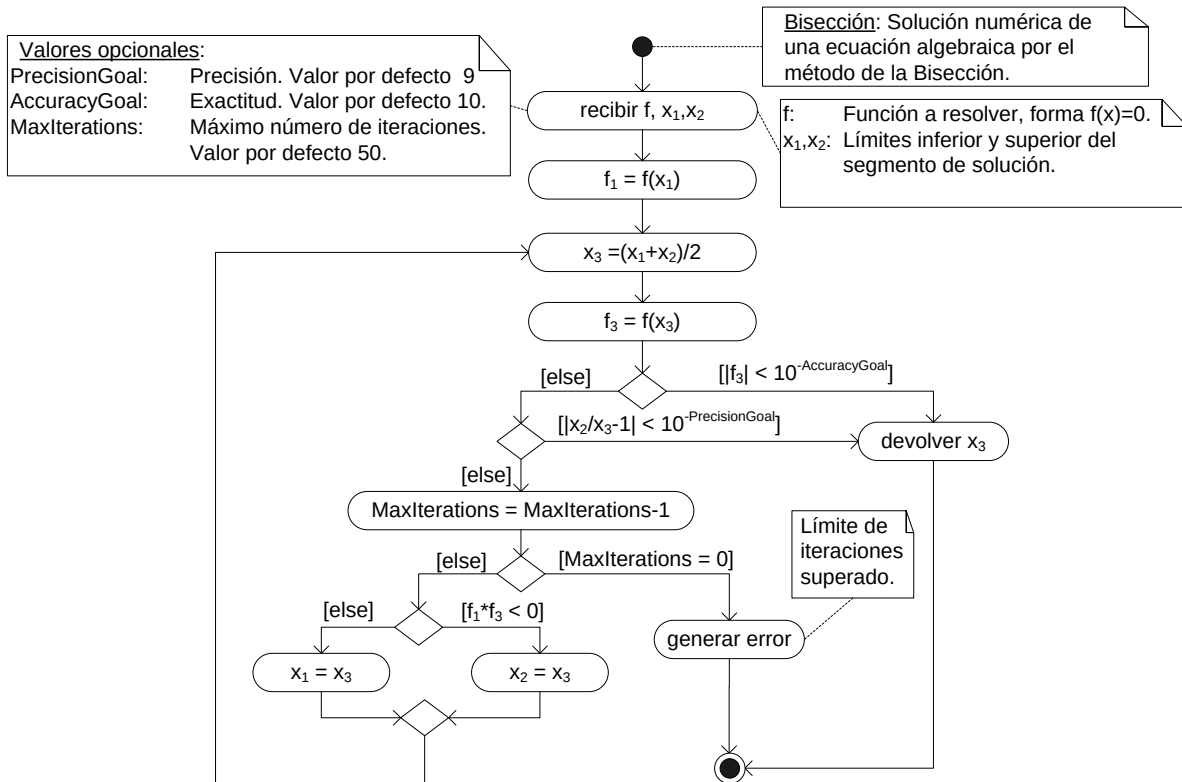


Figura 3.11. Método de la Bisección.

```
Options[Biseccion]={PrecisionGoal->9,AccuracyGoal->10,
  MaxIterations->50};
```

```
Biseccion::maxit="Se ha alcanzado el límite
  de iteraciones: `1`, sin haber encontrado la
  solución";
```

```
Biseccion[f_,x01_,x02_,OptionsPattern[]]:=
Module[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f3},
  {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
    MaxIterations}];
  f1=f[x1]; li=mi;
  While[True,
    x3=(x1+x2)/2;
    f3=f[x3];
    If[Abs[f3]<10^-ag,Break[]];
    If[Abs[x2/x3-1]<10^-pg,Break[]];
    If[--mi==0,Message[Biseccion::maxit,li];Break[]];
    If[f1*f3<0,x2=x3,x1=x3];
    x3]\[AliasDelimiter]
```

...

```
End[]
```

```
EndPackage[]
```

Guardamos los cambios y probamos el programa resolviendo una vez más la ecuación 3.11, con los valores por defecto y sabiendo que la solución está entre -2 y -1:

```
Get["Alge1`",Path->"C:\MAT205 "]
```

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
Incre1[f,1.1,1]
```

```
{-1.9,-0.9}
```

```
Biseccion[f,-1.9,-0.9]//InputForm
```

```
-1.6506291912868618
```

Con 14 dígitos de exactitud obtenemos el resultado:

```
Biseccion[f,-1.9,-0.9,PrecisionGoal->14,AccuracyGoal->Infinity]//InputForm
```

```
-1.650629191439387
```

Para ver los puntos generados por el método en las diferentes iteraciones creamos una nueva versión del programa que denominaremos "Biseccionb":

```
BeginPackage["Alge1`"]
```

```
...
```

```
Biseccionb::usage="Biseccionb[f,x1,x2] encuentra  
la solución de la función \"f\", que se encuentra  
en la forma \"f(x)=0\" empleando el método Incremental  
en el segmento de solución comprendido entre \"x1\" y  
\"x2\", mostrando los valores de cada iteración.";
```

```
...
```

```
Begin["`Private`"]
```

```
...
```

```
Options[Biseccionb]={PrecisionGoal->9,AccuracyGoal->10,  
MaxIterations->50};
```

```
Biseccionb[f_,x01_,x02_,OptionsPattern[]]:=
```

```
Reap[Module[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f3},  
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,  
MaxIterations}]];
```

```
f1=f[x1]; li=mi;
```

```
While[True,
```

```
  x3=(x1+x2)/2;
```

```
  f3=f[x3]; Sow[{x3,f3}];
```

```
  If[Abs[f3]<10^-ag,Break[]];
```

```
  If[Abs[x2/x3-1]<10^-pg,Break[]];
```

```
  If[--mi==0,Message[Biseccion::maxit,li];Break[]];
```

```
  If[f1*f3<0,x2=x3,x1=x3];
```

```
  x3]]
```

```
...
```

```
End[]
```

```
EndPackage[]
```

Ahora, empleando esta versión del programa podemos obtener y posteriormente graficar las 7 primeras iteraciones:

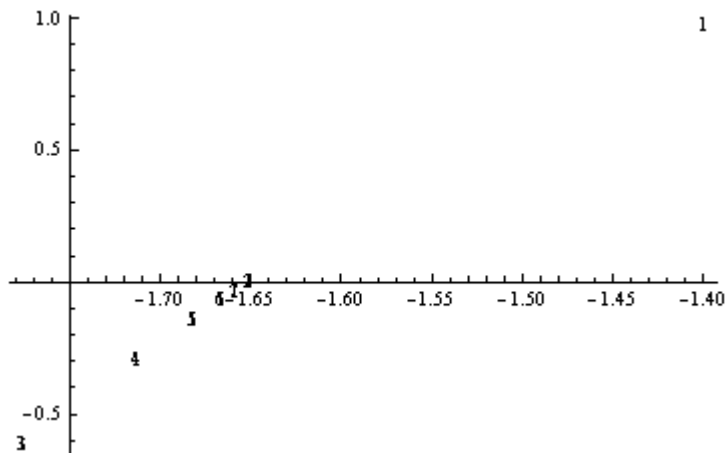
```
Off[Biseccion::maxit]
Biseccionb[f,-1.9,-0.9,MaxIterations->7]
{-1.65781,{{{ -1.4,0.976},{-1.65,0.002875},{-1.775,-0.616109},
  {-1.7125,-0.294361},{-1.68125,-0.142771},{-1.66563,-0.0692162},
  {-1.65781,-0.0329891}}}}
```

```
L=%[[2,1]]
{{{ -1.4,0.976},{-1.65,0.002875},{-1.775,-0.616109},
  {-1.7125,-0.294361},{-1.68125,-0.142771},{-1.66563,-0.0692162},
  {-1.65781,-0.0329891}}}
```

Para visualizar gráficamente el orden en que se han generado estos puntos, colocamos cada punto en una lista y la graficamos con un número consecutivo:

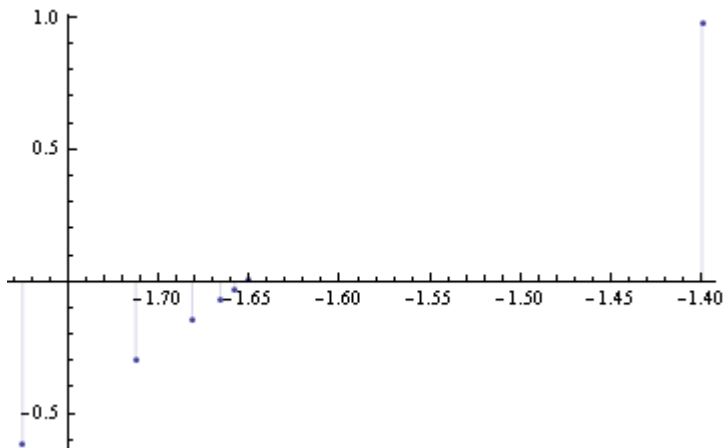
```
L2=Table[{L[[i]]},{i,Length[L]}]
{{{ -1.4,0.976}},{-1.65,0.002875}},{-1.775,-0.616109}},
  {{{ -1.7125,-0.294361}},{-1.68125,-0.142771}},{-1.66563,-0.0692162}},
  {{{ -1.65781,-0.0329891}}}
```

```
ListPlot[L2,PlotMarkers->CharacterRange["1","7"],
  PlotRange->Full,PlotStyle->Black]
```



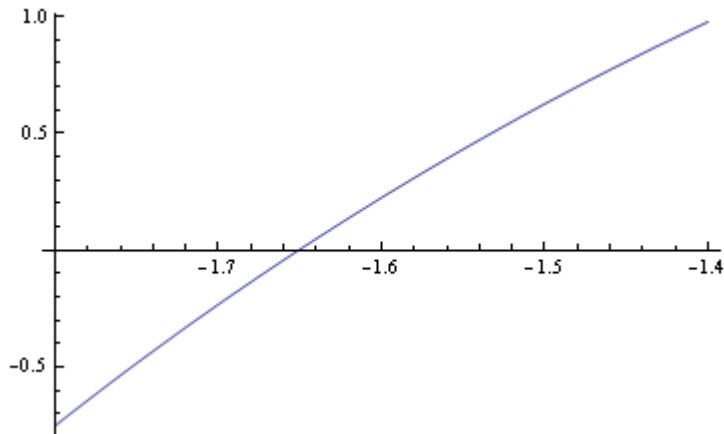
Graficamos también los puntos, proyectando una sombra hacia los ejes para visualizarlos con mayor claridad:

```
ListPlot[L,Filling->Axis,PlotRange->Full]
```



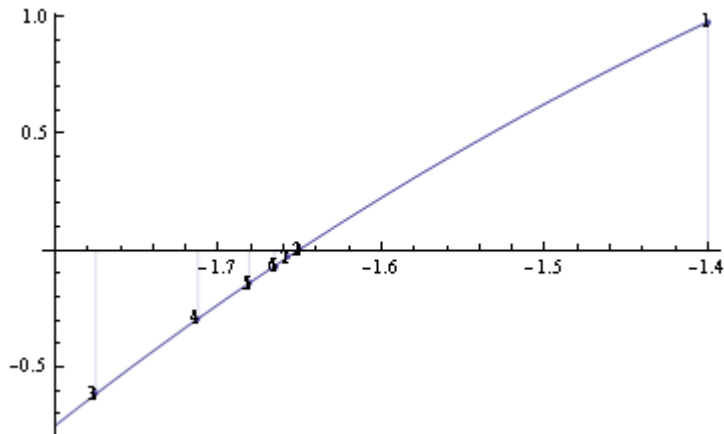
Graficamos también la función entre -1.8 y -1.4:

```
Plot[f[x], {x, -1.8, -1.4}]
```



Finalmente mostramos todas las gráficas en uno:

```
Show[%,%%,%%]
```



Como se puede observar, en cada iteración del método no siempre se consigue mejorar la exactitud del resultado, así la segunda iteración está más cercana a la solución que la tercera, cuarta, quinta y sexta.

Si bien el método de la Bisección reduce el número de iteraciones con relación al método Incremental, todavía requiere un elevado número de iteraciones, así para encontrar la solución con los valores por defecto se requieren 30 iteraciones, como se puede evidenciar resolviendo la función con "Biseccionb":

```
Biseccionb[f, -1.9, -0.9]
```

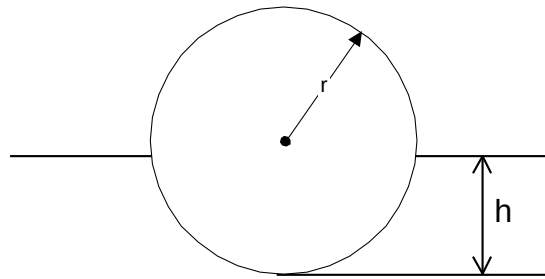
```
{-1.65063, {{{-1.4, 0.976}, {-1.65, 0.002875}, {-1.775, -0.616109},
  {-1.7125, -0.294361}, {-1.68125, -0.142771},
  {-1.66563, -0.0692162}, {-1.65781, -0.0329891},
  {-1.65391, -0.0150119}, {-1.65195, -0.00605716},
  {-1.65098, -0.00158826}, {-1.65049, 0.000644072},
  {-1.65073, -0.00047192}, {-1.65061, 0.0000861201},
  {-1.65067, -0.000192889}, {-1.65064, -0.0000533817},
  {-1.65063, 0.0000163699}, {-1.65063, -0.0000185057},
  {-1.65063, -1.06788*10^-6}, {-1.65063, 7.65101*10^-6},
  {-1.65063, 3.29157*10^-6}, {-1.65063, 1.11185*10^-6},
  {-1.65063, 2.19836*10^-8}, {-1.65063, -5.22948*10^-7},
  {-1.65063, -2.50482*10^-7}, {-1.65063, -1.14249*10^-7},
```

```
{-1.65063,-4.61328*10^-8},{-1.65063,-1.20746*10^-8},
{-1.65063,4.95451*10^-9},{-1.65063,-3.56004*10^-9},
{-1.65063,6.97231*10^-10}}}
```

12.2.2. Ejemplos

3. Una esfera de densidad d se encuentra parcialmente sumergida en agua, como se muestra en la figura de la siguiente página. El peso de la esfera de radio r es $\frac{4}{3}(\pi r^3 d)$ y el volumen del segmento sumergido es $(\pi/3)(3rh^2 - h^3)$. Elabore una función que reciba como parámetro la densidad de la esfera " d ", siendo su valor por defecto 0.6, y devuelva la altura " h " a la cual se encuentra sumergida como una fracción de su radio.

Como se sabe, por el principio del empuje, el peso de la esfera debe ser igual al volumen de agua desplazado por la esfera:



$$\frac{4}{3}\pi r^3 d = \frac{\pi}{3}(3rh^2 - h^3)$$

Donde se asume que la densidad del agua es 1. Ahora colocamos " h " como una función del radio (r): " $h=rx$ ", donde x es la fracción de " r " a la cual se sumerge la esfera y reemplazamos esta relación en la anterior ecuación:

$$\frac{4}{3}\pi r^3 d = \frac{\pi}{3}(3rr^2x^2 - r^3x^3) = \frac{\pi}{3}r^3(3x^2 - x^3)$$

$$x^3 - 3x^2 + 4d = 0$$

Por lo tanto el problema puede ser resuelto si se encuentra la solución (o raíz) de esta ecuación cúbica. Se debe tomar en cuenta también que la densidad " d " no puede ser mayor a 1, pues de ser así la esfera se sumergiría completamente llegando hasta el fondo del recipiente, por lo tanto en esos casos la altura sumergida es directamente mayor a 2 veces el radio:

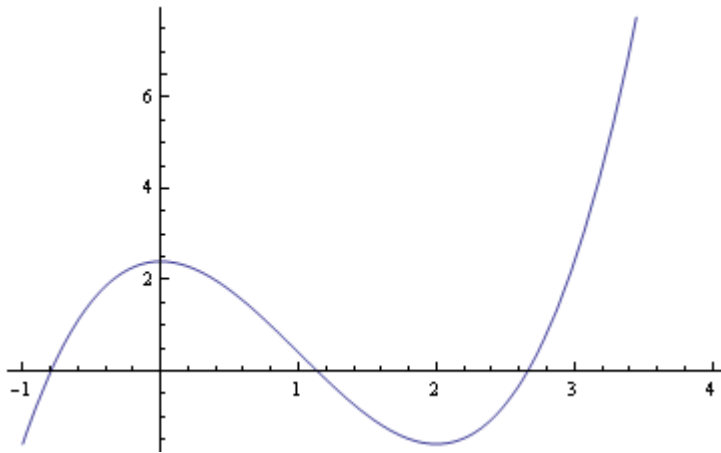
```
Get["Algel`",Path->"C:\MAT205"]
```

```
f[x_,d_:=0.6]:=Return["Mayor a 2 veces el radio"];/;d>1
f[x_,d_:=0.6]:=x^3-3*x^2+4*d
```

Para ver el comportamiento de esta función la graficamos entre -1 y 4 (el resultado se muestra en la siguiente página):

```
Plot[f[x],{x,-1,4}]
```

Como se observa en la gráfica, existen 3 soluciones reales, una comprendida entre -1 y 0, otra entre 0 y 2 y la tercera entre 2 y 3. Podemos encontrar estas tres soluciones con el método de la Bisección, sin embargo, y como se dijo previamente, la solución real debe estar comprendida entre 0 y 2, porque valores menores a cero son físicamente imposibles y superiores a 2 tampoco (la esfera estaría en el fondo del recipiente, a una profundidad desconocida).



Simplemente para comprobar que el método de la Bisección puede encontrar las tres soluciones encontramos las dos que no tienen significado físico:

```
Biseccion[f,-1.,0]
-0.79522
```

```
Biseccion[f,2.,3.]
2.66108
```

Ahora elaboramos la función para encontrar la solución que si tiene sentido físico:

```
fd[d_:=0.6]:=Module[{f,r,x},
  f[x_]:=x^3-3*x^2+4*d;
  If[d<=1,x=Biseccion[f,0,2.],x="Mayor a 2"];
  {Symbol["h"]->x*Symbol["r"]}]
```

Y calculamos con la misma la profundidad a la que se hunde la esfera con la densidad por defecto:

```
fd[]
{h->1.13414 r}
```

En consecuencia para una densidad igual a 0.6 la esfera se hunde 1.13 veces su radio. Por supuesto podemos emplear la función para otras densidades:

```
fd[0.3]
{h->0.726515 r}
```

```
fd[0.9]
{h->1.6084 r}
```

Y para una densidad mayor a 1, obtenemos:

```
fd[2.1]
{h->Mayor a 2 r}
```

4. Como segundo ejemplo volveremos a resolver el problema de la trampa para ratas (ejemplo 4, tema 11) empleando el método de la Bisección.

Recordemos que la función a resolver es:

$$f(k) = \theta_c - \frac{T_0}{k} \left(1 - \cos \left(\sqrt{\frac{k}{I_0}} t \right) \right) = 0$$

$$T_0 = T_c + \theta_k k$$

Donde los parámetros opcionales y sus valores por defecto son:

$$\theta_c = 2.97; \theta_k = 3.27; T_c = 0.625; t = 0.0191; I_0 = 0.0006$$

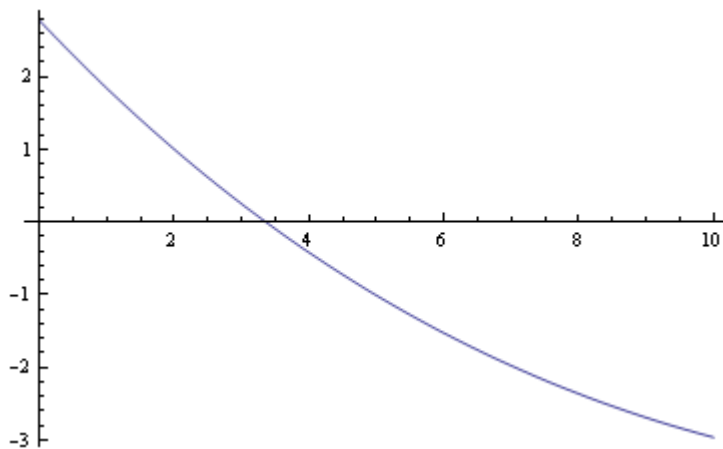
Aunque el lugar (la raíz) de la solución no cambia con relación a la solución encontrada en el tema 11, la forma de la función sí (porque para el método de la Bisección la ecuación tiene que estar igualada a cero), por ello volvemos a graficarla (con los valores por defecto) simplemente para ver la forma de la curva:

```
Get["Alge1`", Path->"C:\MAT205 "]
```

```
Remove[f]
```

```
f[k_]:=With[{θc=2.97,θk=3.27,Tc=0.625,t=0.0191,I0=0.0006},
Module[{T0=Tc+θk*k},
  θc-T0/k*(1-Cos[Sqrt[k/I0]*t])]]
```

```
Plot[f[k], {k, 0, 10}]
```



En este caso, para encontrar el segmento de solución emplearemos el método incremental (Incr1):

```
Remove[fk]
```

```
Options[fk]={θc->2.97,θk->3.27,Tc->0.625,t->0.0191,I0->0.0006};
```

```
fk[OptionsPattern[]]:=Block[{θc,θk,Tc,t,I0,k01,k02,f,T0},
{θc,θk,Tc,t,I0}=OptionValue[{θc,θk,Tc,t,I0}];
f[k_]:= (T0=Tc+θk*k;
  θc-T0/k*(1-Cos[Sqrt[k/I0]*t]));
{k01,k02}=Incr1[f,1.,0.1];
Biseccion[f,k01,k02]]
```

Ejecutando la función con los valores por defecto se obtiene:

```
fk[]
3.36195
```

Que es la constante del resorte buscada. Por supuesto podemos emplear la función con otros valores diferentes a los por defecto:

```
fk[θc->2.9,θk->3.3,Tc->0.64,t->0.0195,I0->0.0007]
3.63007
```

12.2.3. Ejercicios

8. Cree un paquete con el nombre "Algeli" y en el mismo elabore el programa para el método de la Bisección empleando la instrucción "For".
9. Cree un paquete con el nombre "Algelih" y en el mismo elabore el programa para el método de la Bisección empleando la instrucción "Goto".
10. Cree un paquete con el nombre "Algeli" y en el mismo elabore el programa para el método de la Bisección empleando la instrucción "Do".
11. Resuelva el ejercicio 4, del tema 10, empleando el método de la Bisección.
12. Resuelva el ejercicio 7, del tema 10, empleando el método de la Bisección.
13. Resuelva el ejercicio 10, del tema 10, empleando el método de la Bisección.
14. Resuelva el ejercicio 11, del tema 11, empleando el método de la Bisección.

13. REGULA FALSI - SECANTE

13.1. Método de Regula-Falsi

Al igual que el método de la Bisección, este método requiere que se conozca el segmento donde se encuentra la solución, lo que se puede conseguir con "Incr1" o graficando la función. En este caso, sin embargo, el nuevo valor de prueba (x_3) se calcula interpolando entre los dos valores que limitan el segmento, es decir trazando una línea recta entre estos dos puntos y encontrando la intersección de esta línea con la recta $f=0$, tal como se muestra en la siguiente figura:

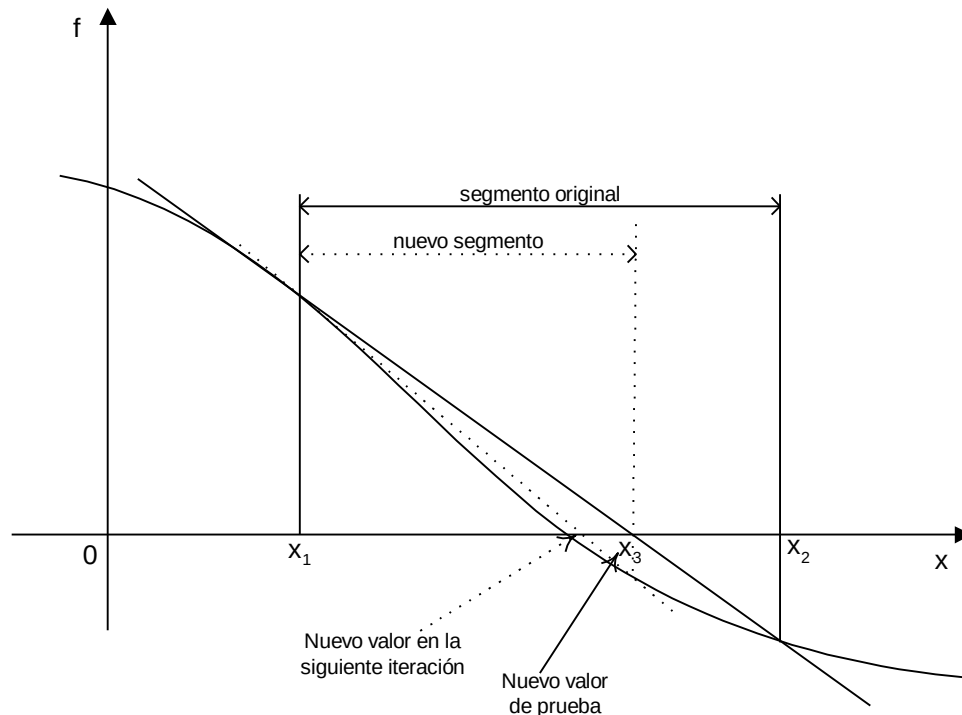


Figura 3.12. Método de Regula - Falsi.

Entonces se comprueba si el nuevo valor de prueba corresponde a la solución (es decir si $f(x_3)$ es aproximadamente cero), de ser así el proceso concluye, caso contrario se toma como nuevo segmento aquel que contiene la solución (el segmento izquierdo en la figura) y se repite el proceso en este segmento (líneas punteadas), procediendo de esta manera hasta que el nuevo valor de prueba corresponde a la solución o hasta que los valores de "x" de dos iteraciones sucesivas son iguales en un determinado número de dígitos.

Para determinar el segmento donde se encuentra la solución se procede igual que en el método de la bisección, es decir se multiplica $f_1 \cdot f_3$ y si el resultado es menor a cero (negativo) entonces la solución se encuentra en el segmento izquierdo, caso contrario se encuentra en el segmento derecho.

La ecuación que permite calcular el nuevo valor de prueba (resultante de la intersección de dos líneas rectas) puede ser deducida en base a la forma general de la ecuación de la línea recta ($f = a + b \cdot x$), sustituyendo en la misma los dos puntos que limitan el segmento:

$$f_1 = a + b x_1 \quad (3.12)$$

$$f_2 = a + bx_2 \quad (3.13)$$

Restando ambas ecuaciones y efectuando algunas operaciones obtenemos la expresión para la constante b :

$$f_1 - f_2 = b(x_1 - x_2) \Rightarrow b = \frac{f_1 - f_2}{x_1 - x_2} \quad (3.14)$$

Sustituyendo ahora esta expresión en la ecuación (3.12), y efectuando algunas operaciones obtenemos la expresión para la constante a :

$$f_1 = a + \frac{f_1 - f_2}{x_1 - x_2} x_1 \Rightarrow a = \frac{f_1 x_1 - f_1 x_2 - f_2 x_1 + f_2 x_1}{x_1 - x_2} = \frac{f_2 x_1 - f_1 x_2}{x_1 - x_2} \quad (3.15)$$

Reemplazando estas expresiones en la ecuación general de la línea recta y tomando en cuenta que en la intersección la función es cero mientras que el nuevo valor de " x " se denomina x_3 , se tiene:

$$f = \frac{f_2 x_1 - f_1 x_2}{x_1 - x_2} + \frac{f_1 - f_2}{x_1 - x_2} x_3 \quad (3.16)$$

Por consiguiente la ecuación que permite calcular el nuevo valor de prueba para el método de Regula Falsi es:

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} \quad (3.17)$$

Salvo por esta ecuación, donde se requieren no solo los valores de " x ", sino también los valores de la función, el algoritmo de este método es esencialmente el mismo que el de la Bisección, en consecuencia el algoritmo básico del método es el siguiente:

- a) Proporcionar los valores de la variable independiente al inicio y final del segmento de solución: x_1, x_2 .
- b) Calcular los valores de la función en los límites: $f_1 = f(x_1)$, $f_2 = f(x_2)$.
- c) Calcular un nuevo valor de prueba " x_3 " con la ecuación (3.17).
- d) Calcular el valor de la función para el nuevo valor de prueba: $f_3 = f(x_3)$.
- e) Si f_3 es casi cero, dentro de un margen de tolerancia predeterminado, el proceso concluye, siendo la solución x_3 .
- f) Si $f_1 \cdot f_3 < 0$, hacer que $x_2 = x_3$ y $f_2 = f_3$, caso contrario hacer que $x_1 = x_3$ y $f_1 = f_3$.
- g) Repetir el proceso desde el paso (c).

Siguiendo el algoritmo básico del método de Regula Falsi volveremos a encontrar una de las soluciones de la ecuación: $f(x) = x^3 + 2x^2 + 3x + 4 = 0$.

Para encontrar el segmento de solución (los valores de x_1 y x_2) emplearemos el método "Incretl" por lo que deberemos importar primeramente el paquete "Algel".

```
Get["Algel`", Path -> "C:\MAT205"]
```

```
Module[{f, f1, f2, f3, x1, x2, x3, i=0},
  f[x_] := x^3 + 2*x^2 + 3*x + 4;
  {x1, x2} = Incretl[f, -10., -1];
  f1 := f[x1]; f2 := f[x2];
  Label[c];
  x3 = (f2*x1 - f1*x2) / (f2 - f1);
  f3 = f[x3];
  If[Abs[f3] < 10^-12, Goto[h]];
```

```

    If [f1*f3<0,x2=x3;f2=f3,f1=f3;x1=x3];
    Goto[c];
    Label[h];x3]//InputForm
-1.650629191439341

```

El algoritmo considerando la precisión, exactitud y límite de iteraciones se presenta en la siguiente figura.

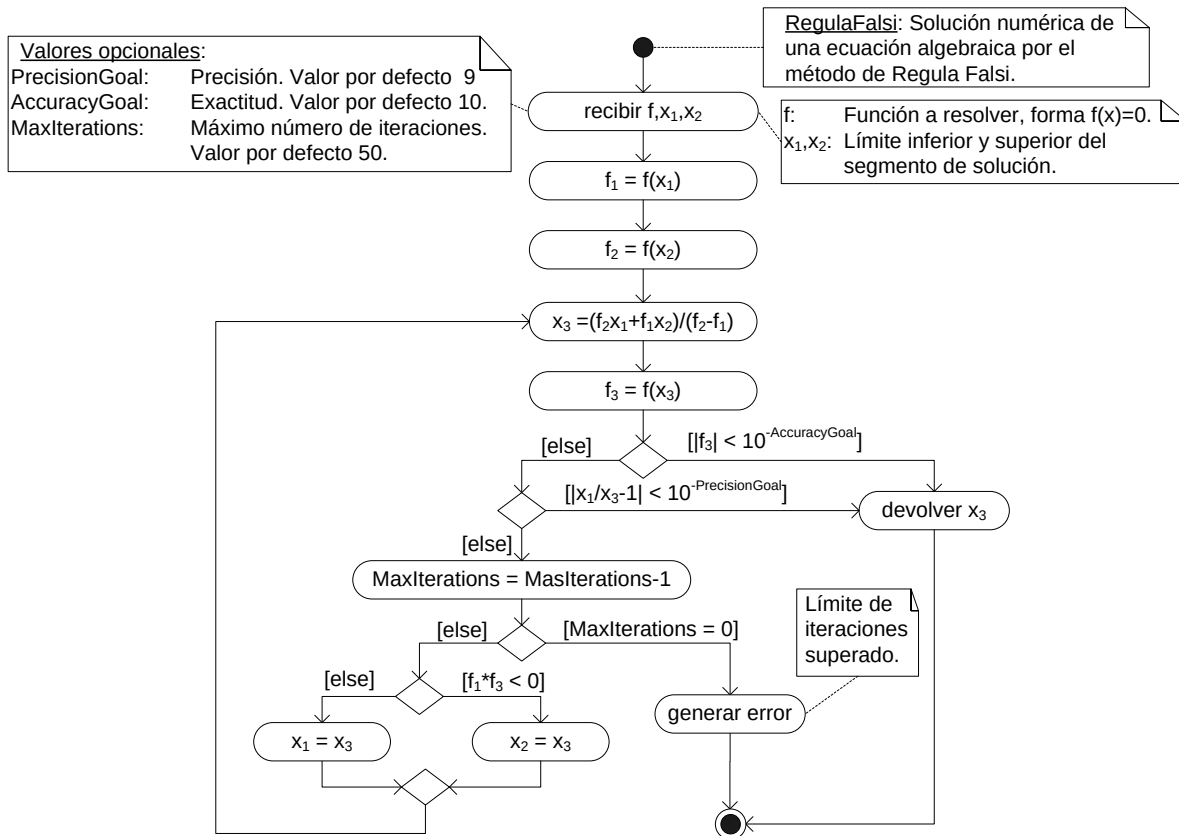


Figura 3.13. Método de Regula - Falsi

Al igual que en los otros métodos añadimos el programa para el método de Regula Falsi en la unidad Algel:

```

BeginPackage["Algel`"]

```

```

...

```

```

RegulaFalsi::usage="RegulaFalsi[f,x1,x2] encuentra
la solución de la función \"f\", que se encuentra
en la forma \"f(x)=0\" empleando el método de Regula Falsi
en el segmento de solución comprendido entre x1 y x2";

```

```

...

```

```

Begin["`Private`"]

```

```

...

```

```

Options[RegulaFalsi]={PrecisionGoal->9,AccuracyGoal->10,
MaxIterations->50};

```

```

RegulaFalsi::maxit="Se ha alcanzado el límite

```

de iteraciones: `1`, sin haber encontrado la solución";

```
RegulaFalsi[f_,x01_,x02_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f2,f3},
  {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
    MaxIterations}]];
f1=f[x1];
f2=f[x2];
li=mi;
While[True,
  x3=(f2*x1-f1*x2)/(f2-f1);
  f3=f[x3];
  If[Abs[f3]<10^-ag,Break[]];
  If[Abs[x1/x3-1]<10^-pg,Break[]];
  If[--mi==0,Message[RegulaFalsi::maxit,li];Break[]];
  If[f1*f3<0,x2=x3;f2=f3,x1=x3;f1=f3]];
x3]
```

...

End[]

EndPackage[]

Ahora probamos el método volviendo a resolver la ecuación 3.11 empleando como valores iniciales -1.6 y -1.7 y empleando los valores por defecto del método:

```
Get["Alge1`",Path->"C:\MAT205"]
```

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
RegulaFalsi[f,-1.6,-1.7]//InputForm
-1.650629191390124
```

Con 12 dígitos de precisión obtenemos el mismo resultado que con el algoritmo básico:

```
RegulaFalsi[f,-2.,-1.,PrecisionGoal->Infinity,AccuracyGoal->12]//InputForm
-1.650629191439341
```

Para ver los resultados intermedios de las diferentes iteraciones, creamos otra versión del programa a la que denominamos "RegulaFalsib":

```
BeginPackage["Alge1`"]
```

...

```
RegulaFalsib::usage="RegulaFalsib[f,x1,x2] encuentra
la solución de la función \"f\", que se encuentra
en la forma \"f(x)=0\" empleando el método de Regula Falsi
en el segmento de solución comprendido entre \"x1\" y \"x2\",
devolviendo los valores intermedios de las iteraciones\".;
```

...

```
Begin["`Private`"]
```

...

```
Options[RegulaFalsib]={PrecisionGoal->9,AccuracyGoal->10,
```

```

MaxIterations->50};

RegulaFalsib[f_,x01_,x02_,OptionsPattern[]]:=
  Reap[Block[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f2,f3},
    {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
      MaxIterations}]];
    f1=f[x1];
    f2=f[x2];
    li=mi;
    While[True,
      x3=(f2*x1-f1*x2)/(f2-f1);
      f3=f[x3]; Sow[{x3,f3}];
      If[Abs[f3]<10^-ag,Break[]];
      If[Abs[x1/x3-1]<10^-pg,Break[]];
      If[--mi==0,Message[RegulaFalsi::maxit,li];Break[]];
      If[f1*f3<0,x2=x3;f2=f3,x1=x3;f1=f3]];
    x3]]

...

End[]

EndPackage[]

```

Resolviendo la ecuación con esta versión del programa obtenemos:

```

RegulaFalsib[f,-1.6,-1.7]//InputForm
{-1.650629191390124,
  {{{{-1.6490153172866522, 0.0073696789278292485},
    {-1.6505784959018195, 0.00023173253519193082},
    {-1.650627599727929, 7.276045299597911*^-6},
    {-1.6506291414644232, 2.2844622371565038*^-7},
    {-1.6506291898703251, 7.172522131781989*^-9},
    {-1.650629191390124, 2.2519675013654705*^-10}}}}}

```

Como se puede ver el método requiere sólo 6 iteraciones para encontrar el resultado con la precisión por defecto, sin embargo es muy difícil observar los valores gráficamente pues se encuentran muy cercanos uno del otro.

13.1.1. Ejemplos

1. Como primer ejemplo volveremos a resolver el siguiente sistema de ecuaciones, no lineales empleando el método de Regula Falsi :

$$\begin{aligned}
 x^{1/2} + z^2 &= 35 \\
 y + e^{z/2} - \frac{8}{z} &= 30 \\
 w + z^2 - 2z &= 31 \\
 x + w + y &= 39
 \end{aligned}$$

Primero colocamos la ecuación en la forma $f(x)=0$:

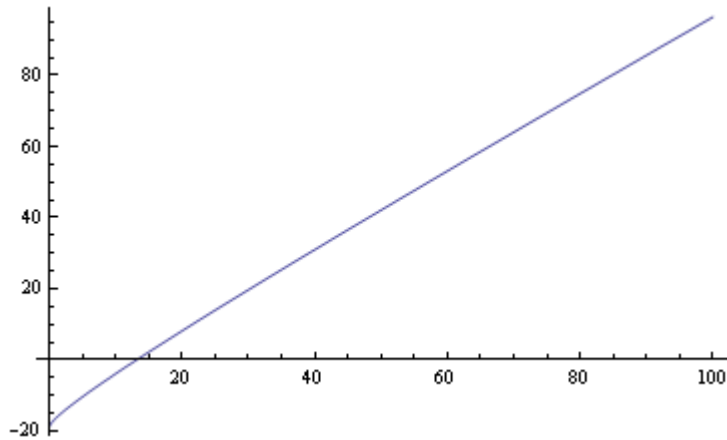
$$\begin{aligned}
 f(x) &= x + w + y - 39 = 0 \\
 z &= \sqrt{35 - x^{1/2}} \\
 y &= 30 + \frac{8}{z} - e^{z/2} \\
 w &= 31 - z^2 + 2z
 \end{aligned}$$

Y para tener una idea del lugar de la solución la graficamos entre 0 y 100:

```
Get["Algel`", Path->"C:\HPV\MAT205\1-2008\Cuadernos Mathematica"]
```

```
f[x_]:=Module[{y,z,w},
  z=Sqrt[35-Sqrt[x]];
  y=30+8/z-Exp[z/2];
  w=31-z^2+2*z;
  x+w+y-39]
```

```
Plot[f[x], {x, 0, 100}]
```



En consecuencia podemos buscar la solución entre 1. y 20.:

```
Module[{f,x,y,z,w,r},
  f[x_]:= (z=Sqrt[35-Sqrt[x]];
  y=30+8/z-Exp[z/2];
  w=31-z^2+2*z;
  x+w+y-39);
  x=RegulaFalsi[f,1.,20.];
  {Symbol["x"]->x, Symbol["y"]->y, Symbol["z"]->z, Symbol["w"]->w}]
```

```
{x->13.1892, y->14.9776, z->5.60074, w->10.8332}
```

2. Como segundo ejemplo volveremos a programar una función para resolver la ecuación que se presenta al final de la pregunta devolviendo los valores de " L_1 ", " x_1 ", " V_1 " y " y_1 ", donde " L_0 ", " x_0 ", " V_0 " y " y_0 " son parámetros opcionales con valores por defecto iguales a: " $L_0=300$ ", " $x_0=0$ ", " $V_0=100$ ", " $y_0=0.2$ ". Los valores iniciales asumidos (x_{i1} y x_{i2}) también deben ser parámetros opcionales y para tener una idea de su valor inicial debe programar la función (empleando los valores por defecto) y graficarla entre 0 y 1 siendo el título de la gráfica "Etapa de Equilibrio" (rojo, 16 puntos, negrita, cursiva, "Alba"), las etiquetas de los ejes " x_1 " y " $f(x_1)$ " (azul, cursiva, 12 puntos, "Arial"), con líneas de división punteadas en color verde. El resultado debe ser calculado con 16 dígitos de precisión o exactitud (lo que ocurra primero), trabajando con 25 dígitos de precisión.

$$f(x_1) = L_1 * x_1 + V_1 * y_1 - C_0 = 0$$

$$L_1 = \frac{L_c}{1 - x_1}$$

$$V_1 = M - L_1$$

$$y_1 = 1.420x_1$$

$$L_c = L_0(1 - x_0)$$

$$M = L_0 + V_0$$

$$C_0 = L_0x_0 + V_0y_0$$

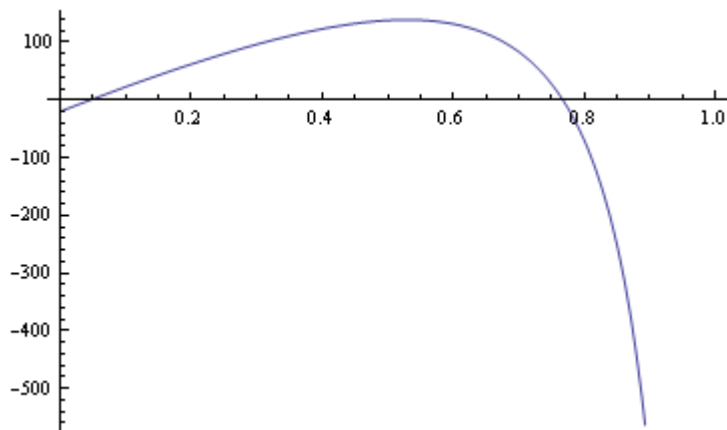
Importamos "Algel" y programamos la función con los valores por defecto:

```
Get["Algel`", Path->"C:\MAT205 "]
```

```
f[x1_]:=With[{L0=300,x0=0,V0=100,
  y0=0.2},Module[{L1,V1,y1,C0,M,Lc},
  C0=L0*x0+V0*y0;
  M=L0+V0;
  Lc=L0*(1-x0);
  L1=Lc/(1-x1);
  V1=M-L1;
  y1=1.420*x1;
  L1*x1+V1*y1-C0]]
```

Graficamos la función entre 0 y 1:

```
Plot[f[x1],{x1,0,1}]
```



Como se puede ver existen dos soluciones, una cerca a 0.1 y otra cerca a 0.8. En este caso programaremos la función de manera que encuentre la primera solución empleando valores iniciales por defecto iguales a: 0.001 y 0.5:

```
Remove[fx1]
```

```
Options[fx1]={L0->300,x0->0,V0->100,y0->0.2,xi1->0.001,xi2->0.5};
```

```
fx1[OptionsPattern[]]:=
Block[{L0,x0,V0,y0,xi1,xi2},
  {L0,x0,V0,y0,xi1,xi2}=OptionValue[{L0,x0,V0,y0,xi1,xi2}];
Module[{L1,x1,V1,y1,C0,M,Lc,f},
  C0=L0*x0+V0*y0;
  M=L0+V0;
  Lc=L0*(1-x0);
```

```

f[x1_] := (L1 = Lc / (1 - x1);
V1 = M - L1;
y1 = 1.420 * x1;
L1 * x1 + V1 * y1 - C0);
x1 = RegulaFalsi[f, xi1, xi2];
{Symbol["L1"] -> L1, Symbol["x1"] -> x1,
Symbol["V1"] -> V1, Symbol["y1"] -> y1}]

```

Evaluando la función con sus valores por defecto se obtienen las soluciones requeridas:

```

fx1[]
{L1->314.425, x1->0.0458777, V1->85.5749, y1->0.0651464}

```

13.1.2. Ejercicios

1. Cree un paquete con el nombre "Algelj" y en el mismo elabore el programa para el método de la Regula Falsi empleando la instrucción "For".
2. Cree un paquete con el nombre "Algelk" y en el mismo elabore el programa para el método de la Regula Falsi empleando la instrucción "Goto".
3. Cree un paquete con el nombre "Algell" y en el mismo elabore el programa para el método de la Regula Falsi empleando la instrucción "Do".
4. Resuelva el ejercicio 4, del tema 10, empleando el método de la Regula Falsi.
5. Resuelva el ejercicio 7, del tema 10, empleando el método de la Regula Falsi.
6. Resuelva el ejercicio 10, del tema 10, empleando el método de la Regula Falsi.
7. Resuelva el ejercicio 11, del tema 10, empleando el método de la Regula Falsi.

13.2. Método de la Secante

En este método, como se puede ver en la figura de la anterior página, se asumen dos valores iniciales, los cuales por lo general son dos valores consecutivos. Con estos valores se calcula el nuevo valor de prueba " x_3 " en la intersección entre la línea recta que pasa a través de los dos puntos y la recta " $f=0$ ". Entonces se comprueba si se ha encontrado la solución (es decir si el valor de " $f(x_3)$ " es casi cero), de ser así el proceso concluye, caso contrario el proceso se repite empleando siempre los dos últimos puntos calculados.

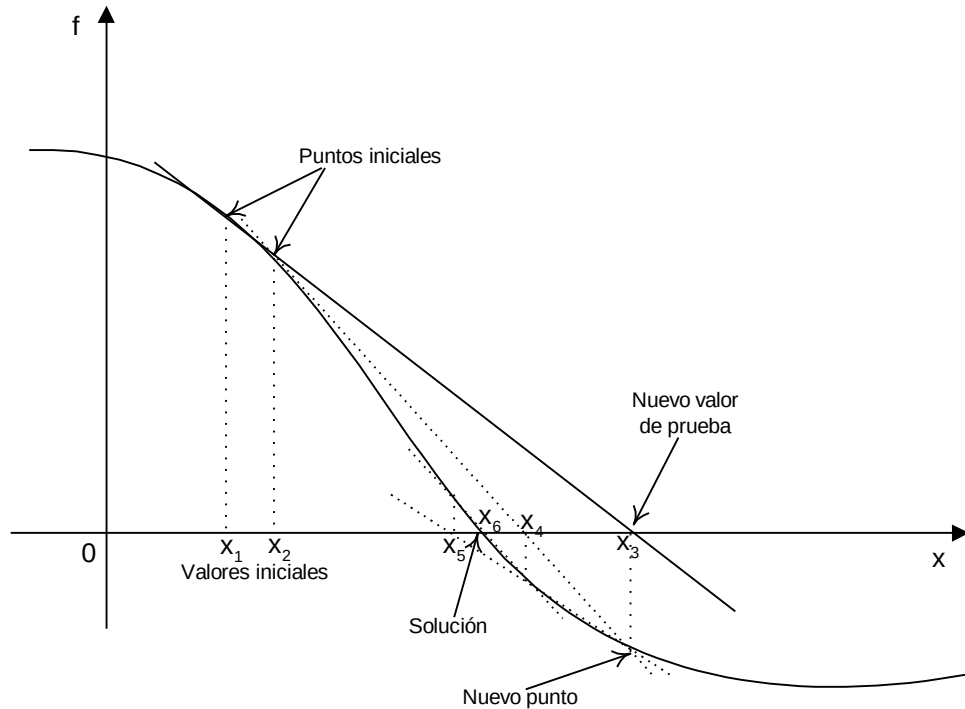


Figura 3.14. Método de la Secante

La ecuación que permite calcular el nuevo valor de prueba " x_3 " es la misma que la del método de Regula Falsi, pues en ambos casos resulta de la intersección de la línea recta que pasa por 2 puntos y la recta " $f=0$ ":

$$x_3 = \frac{f_2 x_1 - f_1 x_2}{f_2 - f_1} \quad (3.17)$$

La diferencia con el método de Regula Falsi, es que en el método de la Secante siempre se emplean los dos últimos puntos calculados, mientras que en el método de Regula Falsi se emplean los puntos que se encuentran a ambos lados de la solución.

El algoritmo básico del método es el siguiente:

- Asumir dos valores iniciales: x_1, x_2 .
- Calcular los valores de la función para los valores asumidos: $f_1=f(x_1)$, $f_2=f(x_2)$.
- Calcular un nuevo valor de prueba " x_3 " con la ecuación (3.17).

- d) Calcular el valor de la función para el nuevo valor de prueba: $f_3=f(x_3)$.
- e) Si f_3 es casi cero, dentro de un margen de tolerancia predeterminado, el proceso concluye, siendo la solución x_3 .
- f) Realizar el cambio de variables: $x_1=x_2$, $f_1=f_2$, $x_2=x_3$ y $f_2=f_3$.
- g) Repetir el proceso desde el paso (c).

Siguiendo el algoritmo básico del método de la Secante volveremos a encontrar una de las soluciones de la ecuación (3.11): $f(x)=x^3+2x^2+3x+4=0$, trabajando con una exactitud de 12 dígitos y encontrando los valores iniciales con el método "Incretl":

```
Get["Algel`",Path->"C:\MAT205"]
```

```
Module[{f,x1,x2,x3,f1,f2,f3},
f[x_]:=x^3+2*x^2+3*x+4;
{x1,x2}=Incretl[f,-10,0.1];
f1=f[x1];f2=f[x2];
Label[c];
x3=(f2*x1-f1*x2)/(f2-f1);
f3=f[x3];
If[Abs[f3]<10^-12,Goto[h]];
x1=x2;f1=f2;x2=x3;f2=f3;
Goto[c];
Label[h];
x3]//InputForm
```

```
-1.6506291914393882
```

13.2.1. Programa

El algoritmo que toma en cuenta la precisión, exactitud y el límite de iteraciones se presenta en la figura (3.15).

Como de costumbre, añadimos el código de este programa al paquete "Algel":

```
BeginPackage["Algel`"]
```

```
...
```

```
Secante::usage="Secante[f,x1,x2] encuentra
la solución de la función \"f\", que se encuentra
en la forma \"f(x)=0\" empleando el método de la Secante
empleando como valores iniciales \"x1\" y \"x2\"";
```

```
...
```

```
Begin[``Private`"]
```

```
...
```

```
Options[Secante]={PrecisionGoal->9,AccuracyGoal->10,
MaxIterations->50};
```

```
Secante::maxit="Se ha alcanzado el límite
de iteraciones: `1`, sin haber encontrado la
solución";
```

```
Secante[f_,x01_,x02_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f2,f3},
```

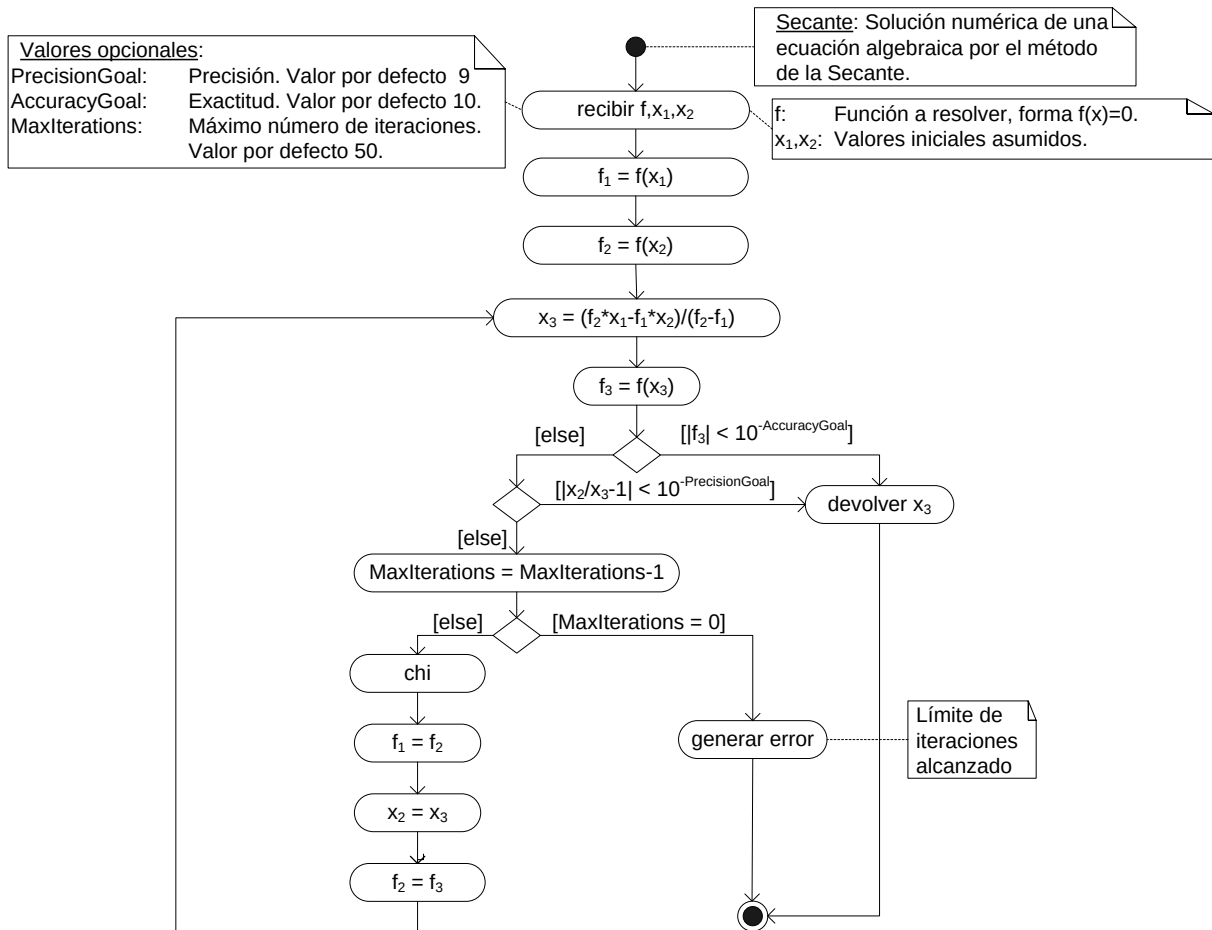


Figura 3.15. Algoritmo del método de la Secante.

```

{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
  MaxIterations}];
li=mi;
f1=f[x1];
f2=f[x2];
While[True,
  x3=(f2*x1-f1*x2)/(f2-f1);
  f3=f[x3];
  If[Abs[f3]<10^-ag,Break[]];
  If[Abs[x2/x3-1]<10^-pg,Break[]];
  If[--mi==0,Message[Secante::maxit,li];Break[]];
  x1=x2;f1=f2;x2=x3;f2=f3];
x3]

```

...

End[]

EndPackage[]

Para probar el método volvemos a resolver la ecuación 3.11, con la precisión por defecto y empleando como valores iniciales -1.7 y -1.6:

```
Get["Algel`",Path->"C:\MAT205"]
```

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
Secante[f,-1.7,-1.6]//InputForm
-1.6506291914374533
```

La solución con 14 dígitos de precisión (sin importar la exactitud) es:

```
NumberForm[Secante[f,-1.7,-1.6,PrecisionGoal->14,AccuracyGoal->Infinity],14]
-1.6506291914394
```

Para obtener los valores de cada iteración creamos una nueva versión del programa, a la cual denominamos "Secanteb" y que también añadimos al paquete "Algel":

```
BeginPackage["Algel`"]
```

```
...
```

```
Secanteb::usage="Secanteb[f,x1,x2] encuentra
la solución de la función \"f\", que se encuentra
en la forma \"f(x)=0\" empleando el método de la Secante
empleando como valores iniciales \"x1\" y \"x2\"";
```

```
...
```

```
Begin["`Private`"]
```

```
...
```

```
Options[Secanteb]={PrecisionGoal->9,AccuracyGoal->10,
MaxIterations->50};
```

```
Secanteb[f_,x01_,x02_,OptionsPattern[]]:=
Reap[Block[{pg,ag,mi,li,x1=x01,x2=x02,x3,f1,f2,f3},

{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
MaxIterations}];
li=mi;
f1=f[x1];
f2=f[x2];
While[True,
x3=(f2*x1-f1*x2)/(f2-f1);
f3=f[x3]; Sow[{x3,f3}];
If[Abs[f3]<10^-ag,Break[]];
If[Abs[x2/x3-1]<10^-pg,Break[]];
If[--mi==0,Message[Secante::maxit,li];Break[]];
x1=x2;f1=f2;x2=x3;f2=f3];
x3]]
```

```
...
```

```
End[]
```

```
EndPackage[]
```

Ahora resolvemos la función empleando esta versión del programa, pero empleando dos valores iniciales fuera del segmento de solución:

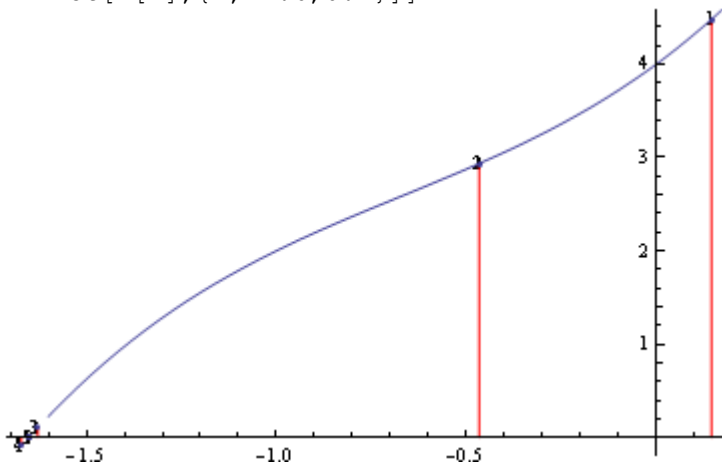
```
Secanteb[f,1.1,1.2]//InputForm
{-1.6506291914393865,
{{0.14485738980120758, 4.479579134869358},
```

```
{-0.46672871600777055, 2.9338150667463205},
{-1.6275012872151065, 0.10415599268807263},
{-1.6702277840385529, -0.09073071164086421},
{-1.65033619894525, 0.0013390778418642668},
{-1.6506255053614767, 0.000016849808698538027},
{-1.6506291921368637, -3.188310060409094*^-9},
{-1.6506291914393865, 7.993605777301127*^-15}}}
```

Como se puede ver, a partir de la sexta iteración los valores varían muy poco, por lo que es difícil apreciarlos gráficamente, por ello mostraremos sólo los primeros 5 puntos:

```
L=%[[2,1]];L2=Table[{L[[i]]},{i,5}]
{{{0.144857,4.47958}},{{-0.466729,2.93382}},{{-1.6275,0.104156}},
{{-1.67023,-0.0907307}},{{-1.65034,0.00133908}}}
g2=Plot[f[x],{x,-1.7,0.2}];

Show[
ListPlot[L2,PlotMarkers->CharacterRange["1","5"],PlotStyle->Black],
ListPlot[L,Filling->Axis,FillingStyle->Red,PlotRange->Full],
Plot[f[x],{x,-1.6,0.2}]]
```



Como se puede observar el método converge rápidamente hacia la solución, a pesar de haber comenzado la búsqueda al lado derecho de la solución, y como se previó, las últimas iteraciones se encuentran muy cercanas una de otra.

13.2.2. Ejemplos

3. Programe una función que resuelva la ecuación: $f(z) = \ln(z) + e^{z+3} - z^{3.1} - 42 = 0$, empleando el método de la secante, con 10 dígitos de precisión o 12 de exactitud (el que se logre primero), empleando como valores iniciales 2.1 y 2.2. Grafique luego las cuatro primeras iteraciones del método.

Importamos el paquete "Alge1" y programamos la función:

```
Get["Alge1`",Path->"C:\MAT205"]
```

```
f[z_]:=Log[z]+Exp[z+3]-z^3.1-42
```

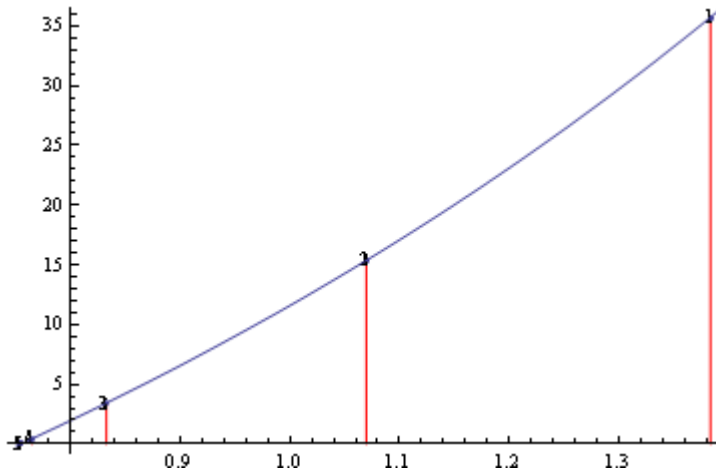
Ahora resolvemos la función empleando como valores iniciales 2.1 y 2.2:

```
Secante[f,2.1,2.2,PrecisionGoal->10,AccuracyGoal->12]
```

0.754179

Y para graficar las primeras 5 iteraciones volvemos a resolver el problema con la versión "Secanteb":

```
Secanteb[f,2.1,2.2,PrecisionGoal->10,AccuracyGoal->12]
{0.754179,
 {{1.38386,35.7336},{1.06961,15.3694},
 {0.832439,3.42528},{0.764425,0.435408},
 {0.75452,0.0144242},{0.754181,0.0000631176},
 {0.754179,9.19514*10^-9},{0.754179,0.}}}}
L=%[2,1];L2=Table[{L[[i]]},{i,5}];
Show[
ListPlot[L,Filling->Axis,FillingStyle->Red,PlotRange->Full],
ListPlot[L2,PlotMarkers->CharacterRange["1","5"],PlotStyle->Black],
Plot[f[z],{z,0.75,1.4}]]
```



Una vez más la convergencia es rápida y los últimos valores se encuentran muy cercanos unos de otros.

4. Elabore una función que calcule los valores de Y_{A1} , Y_{B1} , L_N , X_{BN} y V_1 , resolviendo la ecuación que se presenta al pie de la pregunta, donde X_{AN} , M , C_{A0} y C_{B0} son parámetros opcionales con los siguientes valores por defecto: $X_{AN}=0.12$, $M=32.1$, $C_{A0}=20.2$ y $C_{B0}=3.42$. Se sabe que el valor de Y_{A1} está comprendido entre 0.0001 y 0.999 por lo que se pueden emplear los mismos como valores iniciales.

$$\begin{aligned}
 f(Y_{A1}) &= L_N + V_1 - M = 0 \\
 Y_{B1} &= 2Y_{A1} + Y_{A1}^2 + 3.44Y_{A1}^3 \\
 X_{BN} &= 2.2 + 5.1X_{AN} - 7.2X_{AN}^{2.2} \\
 L_N &= \frac{C_{A0}Y_{B1} - C_{B0}Y_{A1}}{X_{AN}Y_{B1} - X_{BN}Y_{A1}} \\
 V_1 &= \frac{C_{A0} - L_N X_{AN}}{Y_{A1}}
 \end{aligned}$$

Creemos la función:

```
Get["Algel`",Path->"C:\MAT205"]
```

```
Options[fya1]={XAN->0.12,M->32.1,CA0->20.2,CB0->3.42};
```

```
fya1[OptionsPattern[]]:=
```

```
Block[{f,YA1,YB1,LN,XAN,XBN,V1,M,CA0,CB0},
```



```
{XAN,M,CA0,CB0}=OptionValue[{XAN,M,CA0,CB0}];
XBN=2.2+5.1*XAN-7.2*XAN^2.2;
f[YA1_]:= (YB1=2*YA1+YA1^2+3.44*YA1^3;
LN=(CA0*YB1-CB0*YA1)/(XAN*YB1-XBN*YA1);
V1=(CA0-LN*XAN)/YA1;
LN+V1-M);
YA1=Secante[f,0.0001,0.999];
{SymbolName[Unevaluated[YA1]]->YA1,
SymbolName[Unevaluated[YB1]]->YB1,
SymbolName[Unevaluated[LN]]->LN,
SymbolName[Unevaluated[XBN]]->XBN,
SymbolName[Unevaluated[V1]]->V1}}
```

Evaluando la función con sus valores por defecto se obtiene:

```
fya1[]
{YA1->0.411749,YB1->1.23317, LN->-23.9345,XBN->2.74415,V1->56.0345}
```

5. Elabore una función que calcule el valor de "V" resolviendo la ecuación de Benedict-Webb-Rubin que se presenta al pie de la pregunta. Los valores de a , A_0 , b , B_0 , c , C_0 , α y γ son valores opcionales cuyos valores por defecto son: $a=7.11671$, $A_0=1.44373 \times 10^1$, $b=1.09131 \times 10^{-1}$, $B_0=1.77813 \times 10^{-1}$, $c=1.51276 \times 10^6$, $C_0=3.31935 \times 10^6$, $\alpha=2.81086 \times 10^{-3}$, $\gamma=6.66849 \times 10^{-2}$. "P" y "T" son parámetros con valores por defecto iguales a 30 y 600 respectivamente y "R" es una constante igual a 0.08206. Los valores iniciales de "V" deben ser 1.1 y 1.2 veces el resultado de: $V=R*T/P$.

$$P = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{bRT - a}{V^3} + \frac{a\alpha}{V^6} + \frac{c}{V^3 T^2} \left(1 + \frac{\gamma}{V^2}\right) e^{\frac{-\gamma}{V^2}}$$

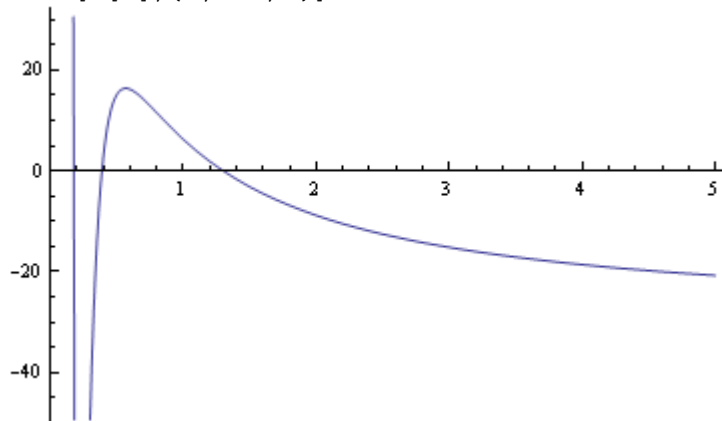
Primero colocamos la ecuación en la forma $f(x)=0$:

$$f(V) = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{bRT - a}{V^3} + \frac{a\alpha}{V^6} + \frac{c}{V^3 T^2} \left(1 + \frac{\gamma}{V^2}\right) e^{\frac{-\gamma}{V^2}} - P = 0$$

Simplemente para ver la forma de la función la graficamos empleando para ello los valores por defecto:

```
Get["Algel`",Path->"C:\MAT205"]
```

```
f[V_]:=With[{a=7.11671,A0=1.44373*10,b=1.09131*10^-1,
B0=1.77813*10^-1,c=1.51276*10^6,C0=3.31935*10^6,
α=2.81086*10^-3,γ=6.66849*10^-2,P=30,T=600,R=0.08206},
R*T/V+(B0*R*T-A0-C0*T^-2)/V^2+(b*R*T-a)/V^3+a*α/V^6+
c/(V^3*T^2)*(1-γ/V^2)*Exp[-γ/V^2]-P]
Plot[f[V],{V,0.1,5}]
```



Como se puede observar, la función tiene una discontinuidad entre 0 y 0.2, por lo que si los valores iniciales están en ese intervalo lo más probable es que no se encuentre ninguna solución. Como también se puede ver, existen dos soluciones después de la discontinuidad, algo que es frecuente en este tipo de ecuaciones (donde una solución corresponde a un estado líquido y otra al gaseoso). Al generar los valores iniciales con la ecuación del gas ideal ($V=RT/P$), prácticamente se garantiza que se encuentre la segunda solución (la de la fase gaseosa).

Ahora establecemos los valores por defecto y creamos la función:

```
Options[BWR]={a->7.11671,A0->1.44373*10,b->1.09131*10^-1,
  B0->1.77813*10^-1,c->1.51276*10^6,C0->3.31935*10^6,
  α->2.81086*10^-3,γ->6.66849*10^-2};
```

```
BWR[P_: 30,T_: 600,OptionsPattern[]]:=
  With[{R=0.08206},
    Block[{f,a,A0,b,B0,c,C0,α,γ,V1,V2},
      {a,A0,b,B0,c,C0,α,γ}=OptionValue[{a,A0,b,B0,c,C0,α,γ}];
      f[V_]:=R*T/V+(B0*R*T-A0-C0*T^2)/V^2+(b*R*T-a)/V^3+
        a*α/V^6+c/(V^3*T^2)*(1-γ/V^2)*Exp[-γ/V^2]-P;
      V1=R*T/P*1.1;V2=R*T/P*1.2;
      Secante[f,V1,V2]]]
```

Evalutando la función con los valores por defecto obtenemos:

```
BWR[]
1.30183
```

Para $P=35$, obtenemos:

```
BWR[35]
1.05404
```

Para $P=37$ y $T=650$ obtenemos:

```
BWR[37,650]
1.18788
```

Y Para $P=40$, $T=700$ y $a=7.12345$, se obtiene:

```
BWR[40,700,a->7.12345]
1.24758
```

13.2.3. Ejercicios

8. Cree un paquete con el nombre "Algelm" y en el mismo elabore el programa para el método de la Secante empleando la instrucción "Nest".
9. Cree un paquete con el nombre "Algeln" y en el mismo elabore el programa para el método de la Secante empleando la instrucción "For".
10. Cree un paquete con el nombre "Algelo" y en el mismo elabore el programa para el método de la Secante empleando la recursividad.
11. Resuelva el ejercicio 3 (tema 10) empleando el método de la Secante.
12. Resuelva el ejercicio 6 (tema 10) empleando el método de la Secante.
13. Resuelva el ejercicio 10 (tema 10) empleando el método de la Secante.
14. Resuelva el ejercicio 12 (tema 10) empleando el método de la Secante.

14. NEWTON RAPHSON

14.1. Método de Newton - Raphson. Derivada analítica

El método de Newton es probablemente el método más empleado en la práctica para resolver ecuaciones algebraicas con una incógnita. En este método se asume un valor inicial (x_1) y con el mismo se calcula la derivada de la función, que como se ve en la figura, es la tangente a la curva. Entonces en la intersección de la tangente con la recta $f=0$ se determina el nuevo valor de prueba (x_2) y con el mismo el valor de la función (f_2). Si el nuevo valor de la función es casi cero el proceso concluye, caso contrario se repite (empleando el nuevo valor de prueba) hasta que la función es casi cero (dentro de un margen de tolerancia especificado), o hasta que los dos últimos valores de prueba (x_1 y x_2) son iguales en un determinado número de dígitos.

La ecuación que permite calcular el nuevo valor de prueba (x_2) puede ser deducida del triángulo que forman la tangente con los segmentos ($x_1 \rightarrow x_2$) y ($0 \rightarrow f_1$):

$$\tan(\alpha) = f'(x_1) = \frac{f_1 - 0}{x_1 - x_2}$$

$$x_1 - x_2 = \frac{f_1}{f'(x_1)}$$

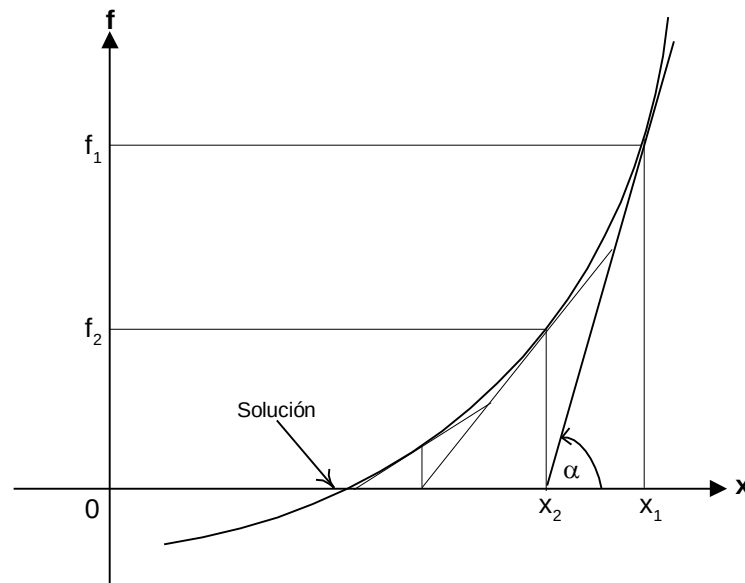


Figura 3.16. Método de Newton - Raphson

$$x_2 = x_1 - \frac{f_1}{f'(x_1)} \quad (3.18)$$

Como se puede observar, la ecuación es sencilla, pero involucra el cálculo de la derivada de la función.

Mathematica permite calcular la derivada analítica de funciones explícitas (aunque no de funciones implícitas). Sin embargo, el cálculo de la deri-

vada analítica es un proceso que suele consumir mucho tiempo (generalmente más tiempo del requerido para resolver la ecuación en sí), por ello en la práctica sólo se emplea la derivada analítica en aquellos casos donde la función es explícita y debe ser resuelta frecuentemente. En esos casos se calcula la derivada analítica por separado (una sola vez) y se la reemplaza en la ecuación de Newton - Raphson, conjuntamente la función, obteniendo así la expresión analítica para la función en particular.

Por ejemplo, para calcular la raíz cuadrada de un número "n" por el método de Newton - Raphson, debemos resolver la siguiente función:

$$x = \sqrt{n}$$

Donde "x" es la raíz cuadrada del número "n". Elevamos ambos términos al cuadrado y llevamos esta función a la forma "f(x)=0":

$$f(x) = x^2 - n = 0$$

Aplicando la ecuación del método Newton (ecuación 3.18) a esta ecuación resulta la expresión:

$$x_2 = x_1 - \frac{x_1^2 - n}{\frac{\partial}{\partial x_1}(x_1^2 - n)} \quad (3.19)$$

Donde, como se ve, es necesario calcular la derivada de la función para obtener la ecuación que nos permita calcular la raíz cuadrada de un número. En este caso en particular el cálculo de la derivada es muy sencillo, sin embargo, para acostumbrarnos a utilizar las instrucciones de Mathematica, encontraremos la derivada empleando la instrucción "D", que tiene la siguiente sintaxis:

$$D[f, \{x, n\}]$$

Donde "f" es la función a derivar, "x" es el nombre de la variable independiente y "n" es el orden de la derivada, cuando la derivada es de primer orden (como en el método de Newton Raphson), el orden puede ser omitido, en cuyo caso la derivada se obtiene con:

$$D[f, x]$$

Así por ejemplo, para obtener la derivada primera de la función: $x^3 + 2x^2 + 3x + 4$, escribimos:

```
D[x^3+2*x^2+3*x+4,x]
3+4 x+3 x^2
```

Aunque es recomendable crear primero la función y luego derivarla:

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
D[f[x],x]
3+4 x+3 x^2
```

De esa manera la función puede ser empleada luego para graficarla y realizar otras operaciones, como por ejemplo calcular la derivada segunda:

```
D[f[x],{x,2}]
4+6 x
```

Alternativamente la instrucción "D" puede ser escrita en la forma: " $\partial_x f$ ", donde el símbolo " ∂ " se consigue con la secuencia de escape: "Esc pd Esc", el nombre de la instrucción: "`\[PartialD]`" o la paleta "BasicMathInput". La variable independiente "x" se introduce como un subíndice (Ctrl+_), así con

esta notación, la derivada primera de la anterior función se obtiene en la forma:

```
{ x f x
3+4 x+3 x^2
```

Las derivadas también pueden ser obtenidas con la instrucción "Derivative" (') que tiene la siguiente sintaxis:

$$\text{Derivative}[n][f]$$

O

$$f', f'', f''', \dots$$

Donde "n" (o ', '', ''', etc.) es el orden de la derivada y "f" es la función a derivar. Esta instrucción devuelve la derivada de la función asumiendo que el primer argumento de la misma es la variable independiente. Así para calcular la derivada primera de la anterior función escribimos:

```
Derivative[1][f][x]
3+4 x+3 x^2
```

O alternativamente:

```
f'[x]
3+4 x+3 x^2
```

Igualmente para la derivada segunda escribimos:

```
Derivative[2][f][x]
4+6 x
```

O su equivalente:

```
f''[x]
4+6 x
```

Aunque con el método de Newton - Raphson trabajamos principalmente con las derivadas parciales, Mathematica nos permite calcular también las derivadas totales. Para ello se emplea la instrucción "Dt":

$$\text{Dt}[f, \{x, n\}]$$

Siendo los parámetros los mismos que "D", pero mientras "D" asume que todas las cantidades que no dependen explícitamente de la variable independiente son constantes, "Dt" asume lo contrario, es decir asume que todas las cantidades no explícitas son variables (depende de la variable independiente). Con la instrucción "D", para identificar explícitamente una cantidad como no constante se emplea el parámetro opcional "nonConstants", por el contrario con la instrucción "Dt", para identificar explícitamente una cantidad como constante se emplea el parámetro opcional "Constants".

Con la función de ejemplo tanto "D" como "Dt" devuelven el mismo resultado porque en la misma las únicas cantidades no explícitas son constantes (números):

```
D[f[x],x]
3+4 x+3 x^2
```

Pero si la función es: $a+bx^2+cx^3+dx^4$:

```
Clear[f]
f[x_]:=a+b*x^2+c*x^3+d*x^4
```

Las instrucciones "Derivative" y "D" devuelven:

```
f'[x]
```

$2 b x + 3 c x^2 + 4 d x^3$

`D[f[x],x]`
 $2 b x + 3 c x^2 + 4 d x^3$

Porque asumen que "a", "b", "c" y "d" son constantes. Mientras que la instrucción "Dt" devuelve:

`Dt[f[x],x]//TraditionalForm`

$$\frac{da}{dx}x^4 + 4dx^3 + \frac{dc}{dx}x^3 + 3cx^2 + \frac{db}{dx}x^2 + 2bx + \frac{da}{dx}$$

Porque asume que "a", "b", "c" y "d" son dependientes de "x". Para que "Dt" devuelva el mismo resultado que "D" debemos indicar explícitamente que "a", "b", "c" y "d" son constantes (empleando la opción "Constants"):

`Dt[f[x],x,Constants->{a,b,c,d}]`
 $2 b x + 3 c x^2 + 4 d x^3$

Por el contrario para que "D" trate a los coeficientes "a", "b", "c" y "d" como variables, debemos identificarlos explícitamente como no constantes (empleando la opción "NonConstants"):

`D[f[x],x,NonConstants->{a,b,c,d}]`
 $2 b x + 3 c x^2 + 4 d x^3 +$
 $x^2 D[a,x,NonConstants->{a,b,c,d}] +$
 $x^3 D[b,x,NonConstants->{a,b,c,d}] +$
 $x^4 D[c,x,NonConstants->{a,b,c,d}] +$
 $x^4 D[d,x,NonConstants->{a,b,c,d}]$

Ahora que ya sabemos cómo derivar en Mathematica volvamos al problema de la raíz cuadrada creando la función:

`Clear[f]`

`f[x_]:=x^2-n`

Evaluando el lado derecho de la ecuación (3.19):

`x1-f[x1]/f'[x1]`
 $x1 - (-n + x1^2) / (2 x1)$

Factorizando términos comunes:

`%//FactorTerms`
 $1/2 (n/x1 + x1)$

Y reemplazando este resultado en el lado derecho de la ecuación (3.19):

$$x_2 = \frac{1}{2} \left(x_1 + \frac{n}{x_1} \right)$$

Que es la ecuación para el cálculo de la raíz cuadrada, que ya hemos empleado en el capítulo 1.

Siguiendo el mismo procedimiento se pueden obtener las ecuaciones de Newton - Raphson, para encontrar la solución de funciones de uso frecuente.

El algoritmo básico del método de Newton Raphson, cuando se dispone de la derivada analítica de la función, es el siguiente:

- a) Asumir un valor inicial: x_1 .
- b) Calcular un nuevo valor de prueba: " x_2 ", con la ecuación de Newton - Raphson (ecuación 3.18).

- c) Si " x_1 " y " x_2 " son iguales en determinado número de dígitos (precisión), el proceso concluye siendo la solución " x_2 ".
- d) Realizar el cambio de variables: $x_1 = x_2$.
- e) Repetir el proceso desde el paso (b).

Siguiendo el algoritmo básico del método de Newton - Raphson volveremos a encontrar una de las soluciones de la ecuación (3.11): $f(x) = x^3 + 2x^2 + 3x + 4 = 0$, trabajando con una precisión de 12 dígitos y empleando un valor inicial igual a 1.1:

```
Module[{f,df,x1=1.1,x2},
  f[x_]:=x^3+2*x^2+3*x+4;
  df[x_]:=Evaluate[f'[x]];
  Label[b];
  x2=x1-f[x1]/df[x1];
  If[Abs[x1/x2-1]<10^-12,Goto[f]];
  x1=x2;
  Goto[b];
  Label[f];
  x2]//InputForm
-1.6506291914393882
```

Observe al asignar el resultado de la derivada primera a la función "df" se ha empleado la instrucción "Evaluate", ello para que la función sea derivada y su resultado asignado a la función, pues de lo contrario (si no se emplea "Evaluate") la derivada recién sería calculada cuando se llama a la función "df" y en consecuencia la derivada sería calculada (una y otra vez) en cada iteración. Otra forma de conseguir el mismo resultado es empleando una asignación inmediata (=) en lugar de una retardada (:=), pero como sabemos ello puede provocar otros problemas.

14.1.1. Programa

El algoritmo del método, que toma en cuenta la exactitud, precisión y límite de iteraciones se presenta en la siguiente figura:

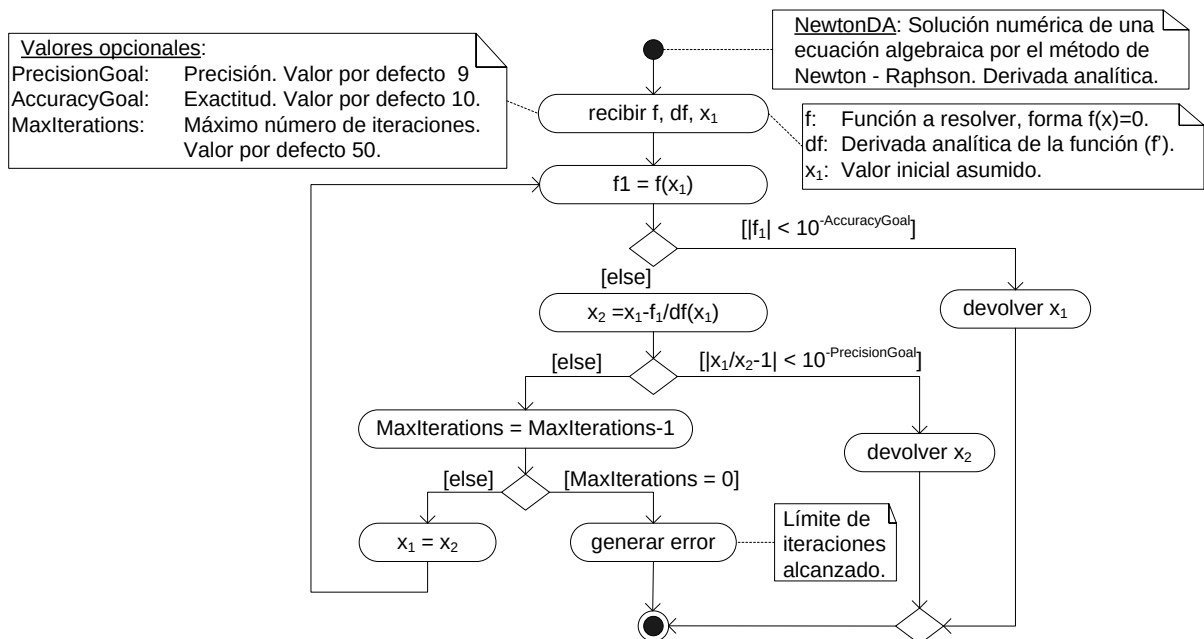


Figura 3.17. Método de Newton - Raphson: Derivada analítica

Al igual que los otros métodos añadimos el método de Newton Raphson a la unidad "Alge1":

```
BeginPackage["Alge1`"]
```

```
...
```

```
NewtonDA::usage="NewtonDA[f,df,x1] encuentra la solución  
de la función \"f\" que se encuentra en la forma \"f(x)=0\"  
y cuya derivada es \"df\", empleando el método de Newton Raphson  
con un valor inicial igual a x1";
```

```
...
```

```
Begin["`Private`"]
```

```
...
```

```
Options[NewtonDA]={PrecisionGoal->9,AccuracyGoal->10,  
MaxIterations->50};
```

```
NewtonDA::maxit="Se ha alcanzado el límite  
de iteraciones: `1`, sin haber encontrado la  
solución";
```

```
NewtonDA[f_,df_,x01_,OptionsPattern[]]:=
Block[{pg,ag,mi,li,x1=x01,x2,f1},
{pg,ag,mi}=OptionValue[{PrecisionGoal,
AccuracyGoal,MaxIterations}];
li=mi;
While[True,
f1=f[x1];
If[Abs[f1]<10^-ag,x2=x1;Break[]];
x2=x1-f1/df[x1];
If[Abs[x1/x2-1]<10^-pg,Break[]];
If[--mi==0,Message[NewtonDA::maxit,li];Break[]];
x1=x2];
x2]
```

```
...
```

```
End[]
```

```
...
```

```
EndPackage[]
```

Probamos el programa volviendo a resolver la ecuación 3.11 con 12 dígitos de precisión:

```
Get["Alge1`",Path->"C:\MAT205 "]
```

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
df[x_]:=Evaluate[f'[x]]
```

```
NewtonDA[f,df,1.1]//InputForm  
-1.6506291914393905
```


Con 12 dígitos de precisión se obtiene el mismo resultado que con el algoritmo básico:

```
NewtonDA[f,df,1.1,PrecisionGoal->12,AccuracyGoal->Infinity]//InputForm
-1.6506291914393882
```

Para obtener los valores intermedios de las iteraciones elaboramos otra versión del método, a la cual denominaremos "NewtonDAb", y que como de costumbre añadiremos a la unidad "Algel":

```
BeginPackage["Algel`"]
```

```
...
```

```
NewtonDAb::usage="NewtonDAb[f,df,x1] encuentra la solución
de la función \"f\" que se encuentra en la forma \"f(x)=0\"
y cuya derivada es \"df\", empleando el método de Newton Raphson
con un valor inicial igual a x1, devolviendo los valores de las
iteraciones intermedias.\";
```

```
...
```

```
Begin["`Private`"]
```

```
...
```

```
Options[NewtonDAb]={PrecisionGoal->9,AccuracyGoal->10,
MaxIterations->50};
```

```
NewtonDAb[f_,df_,x01_,OptionsPattern[]]:=
Reap[Block[{pg,ag,mi,li,x1=x01,x2,f1},
{pg,ag,mi}=OptionValue[{PrecisionGoal,
AccuracyGoal,MaxIterations}];
li=mi;
While[True,
f1=f[x1]; Sow[{x1,f1}];
If[Abs[f1]<10^-ag,x2=x1;Break[]];
x2=x1-f1/df[x1];
If[Abs[x1/x2-1]<10^-pg,Break[]];
If[--mi==0,Message[NewtonDA::maxit,li];Break[]];
x1=x2];
x2]]
```

```
...
```

```
End[]
```

```
...
```

```
EndPackage[]
```

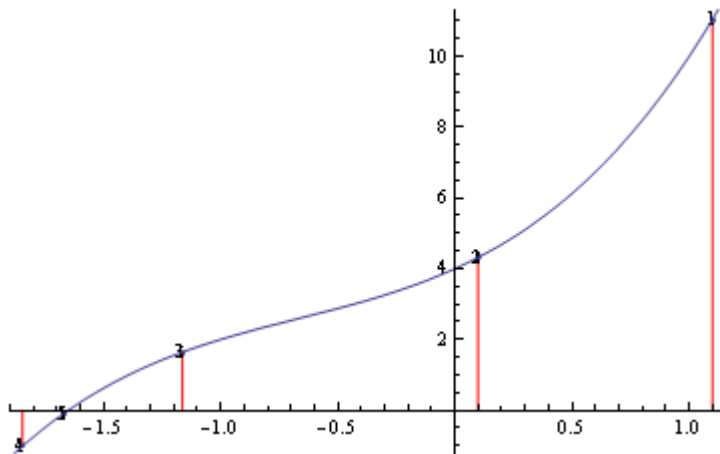
Volviendo a resolver la función con esta versión del programa obtenemos:

```
NewtonDAb[f,df,1.1]//InputForm
{-1.6506291914393905, {
{{1.1, 11.051000000000002},
{0.09809610154125115, 4.314477958492669},
{-1.16298515499483, 1.6431359654047188},
{-1.846013537321704, -1.0132907830733258},
```

```
{-1.6724823714029426, -0.10131569347598468},
{-1.6509334625984056, -0.001391161721238099},
{-1.6506292512128982, -2.732374815295202*^-7},
{-1.6506291914393905, -9.769962616701378*^-15}}}
```

Como se puede observar la convergencia es rápida. La gráfica de las 5 primeras iteraciones (pues las tres últimas están muy cercanas como para ser vistas gráficamente) es:

```
L=%[[2,1]];L=Take[L,5];L2=Table[{L[[i]]},{i,Length[L]}];
Show[ListPlot[L,Filling->Axis,FillingStyle->Red,PlotRange->Full],
ListPlot[L2,PlotMarkers->CharacterRange["1","6"],PlotStyle->Black],
Plot[f[x],{x,-1.9,1.2}]]
```



14.1.2. Ejemplos

1. Resuelva la ecuación (3.1): $f(y) = y - (52 + 3\sqrt{y} - 8y^{0.8})^{0.36} = 0$, empleando el método de Newton Raphson, con 11 dígitos de precisión (sin importar la exactitud), con un valor inicial igual a 1.1. Luego obtenga los valores de las iteraciones y grafique las primeras 2.

```
Get["Alge1`",Path->"C:\MAT205"]
```

```
f[y_]:=y-(52+3*Sqrt[y]-8*y^0.8)^0.36
```

```
df[y_]:=Evaluate[f'[y]]
```

```
NumberForm[NewtonDA[f,df,1.1,PrecisionGoal->11,AccuracyGoal->Infinity],11]
3.6096708861
```

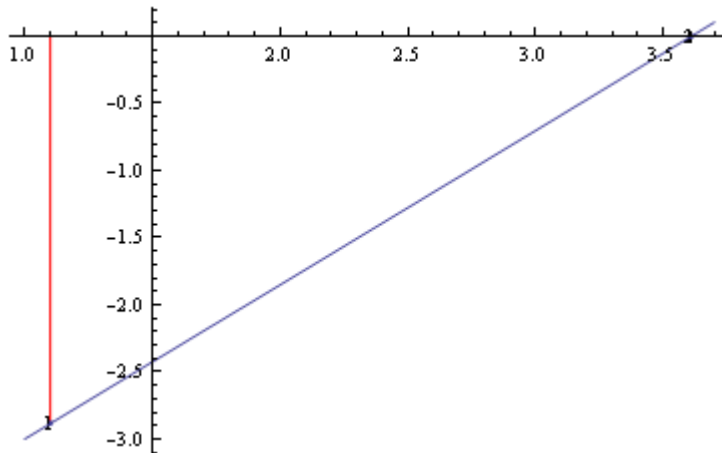
```
NewtonDAb[f,df,1.1,PrecisionGoal->11,AccuracyGoal->Infinity]//InputForm
{3.60967088614005, {
  {{1.1, -2.8840553877884245},
   {3.608916103437996, -0.0008702093959995771},
   {3.609670887487825, 1.553889017458232*^-9},
   {3.60967088614005, 0.}}}}
```

Como se puede observar, la solución se encuentra en tan sólo 4 iteraciones y las dos últimas están muy cercanas como para poder ser apreciadas gráficamente.

```
L=Take[%[[2,1]],2];L2=Table[{L[[i]]},{i,Length[L]}];
```

```
Show[ListPlot[L,Filling->Axis,FillingStyle->Red,PlotRange->All],
```

```
ListPlot[L2,PlotMarkers->{"1","2"},PlotStyle->Black],
Plot[f[y],{y,1.,3.7}]]
```



2. La frecuencia natural de una viga "p" puede ser calculada resolviendo la ecuación compuesta: $f(p) = \cos(kl)\cosh(kl) + 1 = 0$; $k^2 = p/a$; $a^2 = E I_g / (A y)$; donde "l", "I", "E", "y", "A" y "g" son parámetros opcionales con valores por defecto iguales a: "l=120", "I=170.6", "E=3x10⁶", "y=0.066", "A=32" y "g=386", el valor inicial asumido "p₀" debe ser también un parámetro opcional. Se sabe que esta función tiene varias soluciones y para estimar valores iniciales, para dos de ellas se graficará la misma entre "p=0" y "p=500", empleando los valores por defecto. Elabore una función que con un valor inicial estimado en base a la gráfica, para converger a la primera solución, calcule el valor de "p" empleando el método de Newton Raphson. Luego grafique las primeras iteraciones gráficamente distinguibles.

Puesto que en este caso se trata de una función implícita, se debe evaluar la expresión para que Mathematica lleve a cabo las operaciones algebraicas necesarias y reemplace la variable independiente "p" en la función de manera que la misma quede como una función explícita de "p".

```
Get["Algel`",Path->"C:\MAT205"]
```

```
Remove[f,fp]
```

```
f[p_]:=Evaluate[
  With[{l=120,i=170.6,e=3*10^6,y=0.066,A=32,g=386},
    Module[{a=Sqrt[e*i*g/(A*y)],k},
      k=Sqrt[p/a];
      Cos[k*l]*Cosh[k*l]+1]]]
```

Si pedimos información de la función podemos comprobar que la misma está en forma explícita:

```
?f
```

```
Global`f
```

```
f[p_] := 1 + Cos[0.216987 Sqrt[p]] Cosh[0.216987 Sqrt[p]]
```

En consecuencia, ahora puede ser derivada:

```
df[p_]:=Evaluate[f'[p]]
```

?df

Global`df

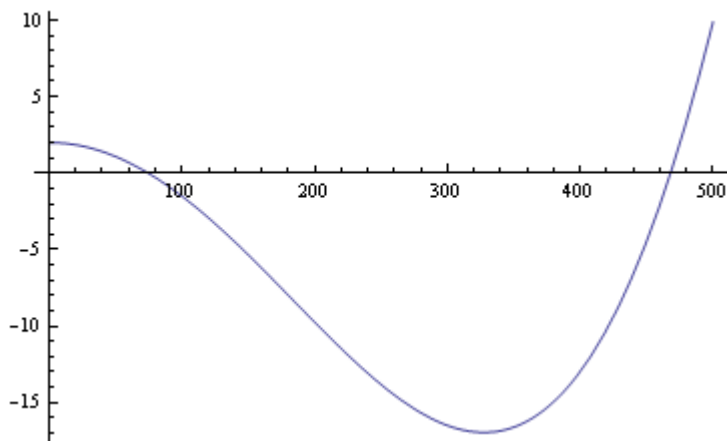
df[p_] :=

$$-\frac{0.108493 \cosh[0.216987 \sqrt{p}] \sin[0.216987 \sqrt{p}]}{\sqrt{p}} + \frac{0.108493 \cos[0.216987 \sqrt{p}] \sinh[0.216987 \sqrt{p}]}{\sqrt{p}}$$

Si no se procede de esta forma (con "Evaluate") de todas formas se obtienen los resultados correctos, sólo que en un mayor tiempo, porque, en ese caso, cada vez que se llama a "f" y "df" se reemplaza (una y otra vez) el valor de la variable independiente "p" y se llevan a cabo las operaciones respectivas.

Graficando la función entre 0 y 500 se obtiene:

Plot[f[p], {p, 0, 500}]



Como se puede observar, la primera solución se encuentra en las proximidades de 100 y la segunda en las de 500 y por la forma de la curva, podemos deducir que con valores iniciales comprendidos entre 60 y 260, aproximadamente, el método de Newton Raphson convergirá a la primera solución y para valores mayores a 380 a la segunda, lo que podemos comprobar llamando al método:

```
NewtonDA[f,df,60.]
74.6767
```

```
NewtonDA[f,df,260.1]
74.6767
```

```
NewtonDA[f,df,380.1]
467.991
```

Ahora elaboramos la función requerida, asignando primero los valores opcionales:

```
Options[fp]={l->120,i->170.6,e->3*10^6,y->0.066,A->32,g->386,p0->60.1};
```

Y programamos la función recordando evaluarla para reducir el número de cálculos:

```
fp[OptionsPattern[]]:=
  Block[{f,df,a,k,l,i,e,y,A,g,p0},
    {l,i,e,y,A,g,p0}=OptionValue[{l,i,e,y,A,g,p0}];
```

```

a=Sqrt[e*i*g/(A*y)];
f[p_]:=Evaluate[k=Sqrt[p/a];Cos[k*1]*Cosh[k*1]+1];
df[p_]:=Evaluate[f'[p]];
NewtonDA[f,df,p0]

```

Y encontramos la solución con los valores por defecto:

```

fp[]
74.6767

```

Si empleamos un valor inicial cercano a 500, obtenemos, como era de esperar la segunda solución:

```

fp[p0->400]
467.991

```

Para obtener los resultados de las primeras iteraciones modificamos la función llamando a "NewtonDAb" en lugar de "NewtonDA":

```

fp[OptionsPattern[]]:=
Block[{f,df,a,k,l,i,e,y,A,g,p0},
  {l,i,e,y,A,g,p0}=OptionValue[{l,i,e,y,A,g,p0}];
  a=Sqrt[e*i*g/(A*y)];
  f[p_]:=Evaluate[k=Sqrt[p/a];Cos[k*1]*Cosh[k*1]+1];
  df[p_]:=Evaluate[f'[p]];
  NewtonDAb[f,df,p0]

```

Con la cual obtenemos:

```

fp[]//InputForm
{74.67669629022707,
 {{60.1, 0.6908419261754808},
 {76.2699247200938,-0.0835454243401712},
 {74.69123307540596,-0.0007553013354604232},
 {74.67669753010397,-6.441594835315811*^-8},
 {74.67669629022707,-6.661338147750939*^-16}}}}

```

Como vemos, el método sólo requiere 5 iteraciones para encontrar la solución, de las cuales sólo las tres primera están lo suficientemente separadas como para ser vistas gráficamente:

```

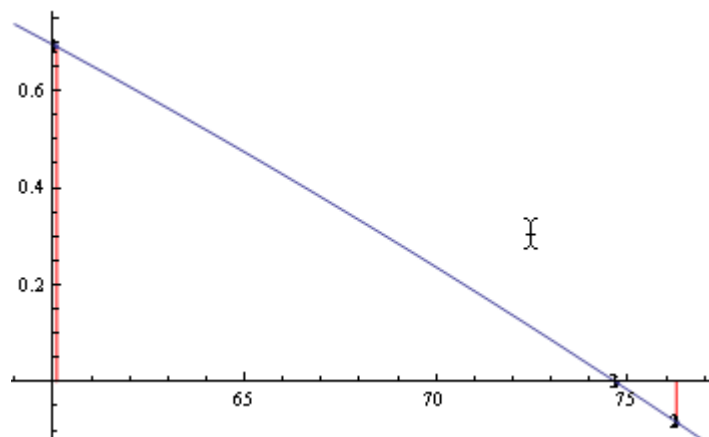
L=Take[%,{2,1}],3];L2=Table[{L[[i]]},{i,Length[L]}];

```

```

Show[ListPlot[L,Filling->Axis,FillingStyle->Red,PlotRange->All],
 ListPlot[L2,PlotMarkers->{"1","2","3"},PlotStyle->Black],
 Plot[f[p],{p,59,77}]]

```



3. Como tercer ejemplo volveremos a resolver problema de la esfera parcialmente sumergida (ejemplo 3, tema 12), es decir elaboraremos una función que reciba como parámetro la densidad de la esfera "d", siendo su valor por defecto 0.6, y devuelva la altura "h" a la cual se encuentra sumergida como una fracción de su radio.

$$f(x) = x^3 - 3x^2 + 4d = 0$$

La función es la misma que en dicho ejemplo, sólo que ahora adicionalmente se debe calcular la derivada:

```
Get["Algel`", Path->"C:\MAT205"]
```

```
fd[d_:=0.6]:=Module[{f,r,x},
  f[x_]:=x^3-3*x^2+4*d;
  df[x_]:=Evaluate[f'[x]];
  If[d<=1,x=NewtonDA[f,df,0.5],x="Mayor a 2"];
  {Symbol["h"]->x*Symbol["r"]}]
```

Con esta función (como era de esperar), se obtienen los mismos resultados que con el método de la Bisección:

```
fd[]
{h->1.13414 r}
fd[0.3]
{h->0.726515 r}
fd[0.9]
{h->1.6084 r}
fd[2.1]
{h->Mayor a 2 r}
```

14.1.3. Ejercicios

1. Derive la ecuación para calcular la raíz cúbica de un número empleando el método de Newton - Raphson.
2. Derive la ecuación para encontrar una de las soluciones de la ecuación cúbica " $f(x)=a+bx+cx^2+dx^3=0$ " empleando el método de Newton-Raphson.
3. Cree un paquete con el nombre "Algelo" y en el mismo elabore el programa para el método de Newton Raphson con derivada analítica empleando la instrucción "Nest".
4. Cree un paquete con el nombre "Algeloq" y en el mismo elabore el programa para el método de Newton - Raphson con derivada analítica empleando la instrucción "For".
5. Cree un paquete con el nombre "Algelo" y en el mismo elabore el programa para el método de Newton - Raphson con derivada analítica empleando la recursividad.
6. Resuelva el ejercicio 3 (tema 10) empleando el método de Newton - Raphson con derivada analítica.
7. Resuelva el ejercicio 6 (tema 10) empleando el método de Newton - Raphson con derivada analítica.
8. Resuelva el ejercicio 10 (tema 10) empleando el método de Newton - Raphson con derivada analítica.
9. Resuelva el ejercicio 12 (tema 10) empleando el método de Newton - Raphson con derivada analítica.

14.2. Método de Newton - Raphson. Derivada numérica

Como ya se señaló en el acápite anterior el cálculo de la derivada analítica suele requerir mucho tiempo y en muchos casos es imposible su cálculo pues la función es implícita y no puede ser colocada en una forma explícita. Por estas razones el método de Newton - Raphson es de mayor utilidad cuando la derivada se calcula numéricamente.

14.2.1. Cálculo numérico de la derivada

El cálculo numérico de las derivadas se efectúa empleando fórmulas que son deducidas en base al concepto de la derivada:

$$\frac{dy}{dx} = \lim_{\Delta x \rightarrow 0} \left(\frac{\Delta y}{\Delta x} \right) \quad (3.19)$$

Donde numéricamente se consigue una aproximación al límite empleando valores de " Δx " pequeños. Básicamente se toman valores consecutivos de " x ", los que constituyen el incremento " Δx " y para estos valores se calculan los correspondientes valores de la función, los que constituyen los valores de " Δy ".

Así por ejemplo para calcular la derivada primera para un determinado valor de " x ", podemos tomar dos valores cercanos a " x ": uno ubicado al lado izquierdo: " $x-h$ " y otro al lado derecho: " $x+h$ ", donde " h " es un valor pequeño (una fracción de x). Entonces al sustituir en la ecuación (3.19) (asumiendo que con este valor pequeño de " h " se alcanza el límite), obtenemos:

$$\frac{dy}{dx} \approx \left(\frac{f(x+h) - f(x-h)}{(x+h) - (x-h)} \right) = \frac{f(x+h) - f(x-h)}{2h}$$

Que es la fórmula de "diferencia central" para calcular la derivada primera. Si en lugar de tomar dos valores a ambos lados de " x " se toma uno al lado izquierdo: " $x-h$ " se tiene:

$$\frac{dy}{dx} \approx \left(\frac{f(x) - f(x-h)}{(x) - (x-h)} \right) = \frac{f(x) - f(x-h)}{h}$$

Que es la fórmula de "diferencia hacia atrás" para calcular la derivada primera. Si en lugar de tomar un valor al lado izquierdo de " x " se toma uno al lado derecho: " $x+h$ ", se tiene:

$$\frac{dy}{dx} \approx \left(\frac{f(x+h) - f(x)}{(x+h) - (x)} \right) = \frac{f(x+h) - f(x)}{h}$$

Que es la fórmula de "diferencia hacia adelante" para calcular la derivada primera.

En general mientras más valores se tomen a ambos lados de " x " más exacto es el resultado calculado, es por ello que de estas tres fórmulas, la más exacta es la de diferencia central, porque toma dos valores.

La deducción de las fórmulas para las derivadas de segundo, tercer y otros órdenes superiores sigue el mismo principio, pero en lugar de la función original se emplean las fórmulas para las derivadas de orden inferior deducidas previamente. Así para obtener la derivada segunda se emplea las fórmulas de la derivada primera, para la tercer las fórmulas de la derivada segunda y así sucesivamente.

En función al número de valores que se toman alrededor de "x" se tienen diferentes tipos de errores: de primer orden si sólo se toma un valor a la derecha o a la izquierda (diferencia hacia adelante o hacia atrás); de segundo orden si se toman dos valores a ambos lados de "x" (diferencia central) y de cuarto orden si se toman 4 valores: 2 a la izquierda y 2 a la derecha de "x".

Aun cuando para el método de Newton - Raphson sólo necesitamos las fórmulas para las derivadas de primer orden, presentamos a continuación las fórmulas para calcular las derivadas hasta de cuarto orden, las mismas que podrán ser empleadas luego, cuando así se requieran.

14.2.1.1. Fórmulas de diferencia central. Error de orden h^2

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x-h)}{2h} \\ f''(x) &= \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \\ f'''(x) &= \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} \\ f^{(4)}(x) &= \frac{f(x+2h) - 4f(x+h) + 6f(x) - 4f(x-h) + f(x-2h)}{h^4} \end{aligned} \quad (3.20)$$

14.2.1.2. Fórmulas de diferencia central. Error de orden h^4

$$\begin{aligned} f'(x) &= \frac{-f(x+2h) + 8f(x+h) + 8f(x-h) + f(x-2h)}{12h} \\ f''(x) &= \frac{-f(x+2h) + 16f(x+h) + 30f(x) + 16f(x-h) - f(x-2h)}{12h^2} \\ f'''(x) &= \frac{f(x+3h) + 8f(x+2h) - 13f(x+h) + 13f(x-h) - 8f(x-2h) + f(x-3h)}{8h^3} \\ f^{(4)}(x) &= \frac{-f(x+3h) + 12f(x+2h) - 39f(x+h) + 56f(x) - 39f(x-h) - 12f(x-2h) - f(x-3h)}{6h^4} \end{aligned} \quad (3-21)$$

14.2.1.3. Fórmulas de diferencia hacia adelante. Error de orden h

$$\begin{aligned} f'(x) &= \frac{f(x+h) - f(x)}{h} \\ f''(x) &= \frac{f(x+2h) - 2f(x+h) + f(x)}{h^2} \\ f'''(x) &= \frac{f(x+h) - 3f(x+2h) + 3f(x+h) - f(x)}{h^3} \\ f^{(4)}(x) &= \frac{f(x+4h) - 4f(x+3h) + 6f(x+2h) - 4f(x+h) + f(x)}{h^4} \end{aligned} \quad (3.22)$$

14.2.1.4. Fórmulas de diferencia hacia adelante. Error de orden h^2

$$\begin{aligned} f'(x) &= \frac{-f(x+2h) + 4f(x+h) - 3f(x)}{2h} \\ f''(x) &= \frac{-f(x+3h) + 4f(x+2h) - 5f(x+h) + 2f(x)}{h^2} \end{aligned}$$

$$f'''(x) = \frac{-3f(x+4h) + 14f(x+3h) - 24f(x+2h) + 18f(x+h) - 5f(x)}{2h^3}$$

$$f''''(x) = \frac{-2y_{i+5} + 11y_{i+4} - 24y_{i+3} + 26y_{i+2} - 14y_{i+1} + 3y_i}{h^4} \quad (3.23)$$

14.2.1.5. Fórmulas de diferencia hacia atrás. Error de orden h

$$f'(x) = \frac{f(x) - f(x-h)}{h}$$

$$f''(x) = \frac{f(x) - 2f(x-h) + f(x-2h)}{h^2}$$

$$f'''(x) = \frac{f(x) - 3f(x-h) + 4f(x-2h) - f(x-3h)}{h^3}$$

$$f''''(x) = \frac{f(x) - 5f(x-h) + 6f(x-2h) - 4f(x-3h) + f(x-4h)}{h^4} \quad (3.24)$$

14.2.1.6. Fórmulas de diferencia hacia atrás. Error de orden h^2

$$f'(x) = \frac{3f(x) - 4f(x-h) + f(x-2h)}{2h}$$

$$f''(x) = \frac{2f(x) - 5f(x-h) + 4f(x-2h) - f(x-3h)}{h^2}$$

$$f'''(x) = \frac{5f(x) - 18f(x-h) + 24f(x-2h) - 14f(x-3h) + 3f(x-4h)}{2h^3}$$

$$f''''(x) = \frac{3f(x) - 14f(x-h) + 26f(x-2h) - 24f(x-3h) + 11f(x-4h) - 2f(x-5h)}{h^4} \quad (3.25)$$

En teoría, cuanto mayor es el orden de error más exacto es el resultado, no obstante, en la práctica las fórmulas de orden elevado implican un mayor número de operaciones, lo que implica un mayor error de redondeo. Por esta razón en la práctica suelen emplearse las fórmulas de primer o segundo orden. Igualmente en teoría, mientras menor sea el valor de " h " más exacto es el resultado (porque más cerca está del límite 0). Una vez más esto no es cierto en la práctica debido a los errores de redondeo.

Emplearemos la fórmula de diferencia central para el cálculo de la derivada numérica. Como incremento " h " emplearemos el valor de " x " multiplicado por 1×10^{-4} , es decir: $h = x \cdot 10^{-4}$, valor que se ha comprobado es lo suficientemente pequeño como para permitir un cálculo preciso de la derivada, sin ser demasiado pequeño como para generar errores apreciables de redondeo.

Así por ejemplo para calcular la derivada primera de la función: $f(x) = x^3 + 2x^2 + 3x + 4$, para $x = 3.5$, escribimos:

```
f[x_] := x^3 + 2*x^2 + 3*x + 4
```

```
Module[{x=3.5,h},h=x*10^-4;(f[x+h]-f[x-h])/(2*h)]
174.5
```

Y podemos corroborar este resultado con "Derivative":

```
f'[3.5]
174.5
```

Que como vemos da exactamente el mismo resultado, lo que nos da una idea de lo confiable de la fórmula de diferencia central.

El algoritmo para el método de Newton con diferenciación numérica sólo difiere del algoritmo con derivada analítica en el hecho de que la derivada se calcula con la fórmula de diferencia central, así el algoritmo básico del método es:

- a) Asumir un valor inicial: " x_1 ".
- b) Calcular el valor numérico de la derivada: " df ".
- c) Calcular un nuevo valor de prueba: " x_2 ", con la ecuación de Newton - Raphson (ecuación 3.18).
- d) Si " x_1 " y " x_2 " son iguales en determinado número de dígitos (precisión), el proceso concluye siendo la solución " x_2 ".
- e) Realizar el cambio de variables: $x_1 = x_2$.
- f) Repetir el proceso desde el paso (b).

Siguiendo el algoritmo básico podemos volver a resolver la ecuación 3.11, con 12 dígitos de precisión:

```
Module[{f,df,h,x1=1.1,x2},
  f[x_]:=x^3+2*x^2+3*x+4;
  Label[b];
  h=x1*10^-4;
  df=(f[x1+h]-f[x1-h])/(2*h);
  x2=x1-f[x1]/df;
  If[Abs[x1/x2-1]<10^-12,Goto[g]];
  x1=x2;
  Goto[b];
  Label[g];
  x2]//InputForm
-1.6506291914393882
```

Y como se puede comprobar, se obtiene el mismo resultado que con la derivada analítica.

14.2.2. Programa

El algoritmo que toma en cuenta la precisión, exactitud y límite de iteraciones es igualmente muy similar al de la derivada analítica, sólo que en este caso la derivada se calcula empleando la ecuación 3.20. Dicho algoritmo se presenta en el diagrama de actividades de la siguiente página.

El programa elaborado en base a dicho algoritmo es añadido, como de costumbre, a la unidad "Algel":

```
BeginPackage["Algel`"]

...

NewtonDN::usage="NewtonDN[f,x1] encuentra la solución
de la función \"f\" que se encuentra en la forma \"f(x)=0\",
empleando el método de Newton Raphson con un valor inicial
igual a x1 (la derivada es calculada numéricamente)";

...

Begin["`Private`"]

...

Options[NewtonDN]={PrecisionGoal->9,AccuracyGoal->10,
  MaxIterations->50};
```

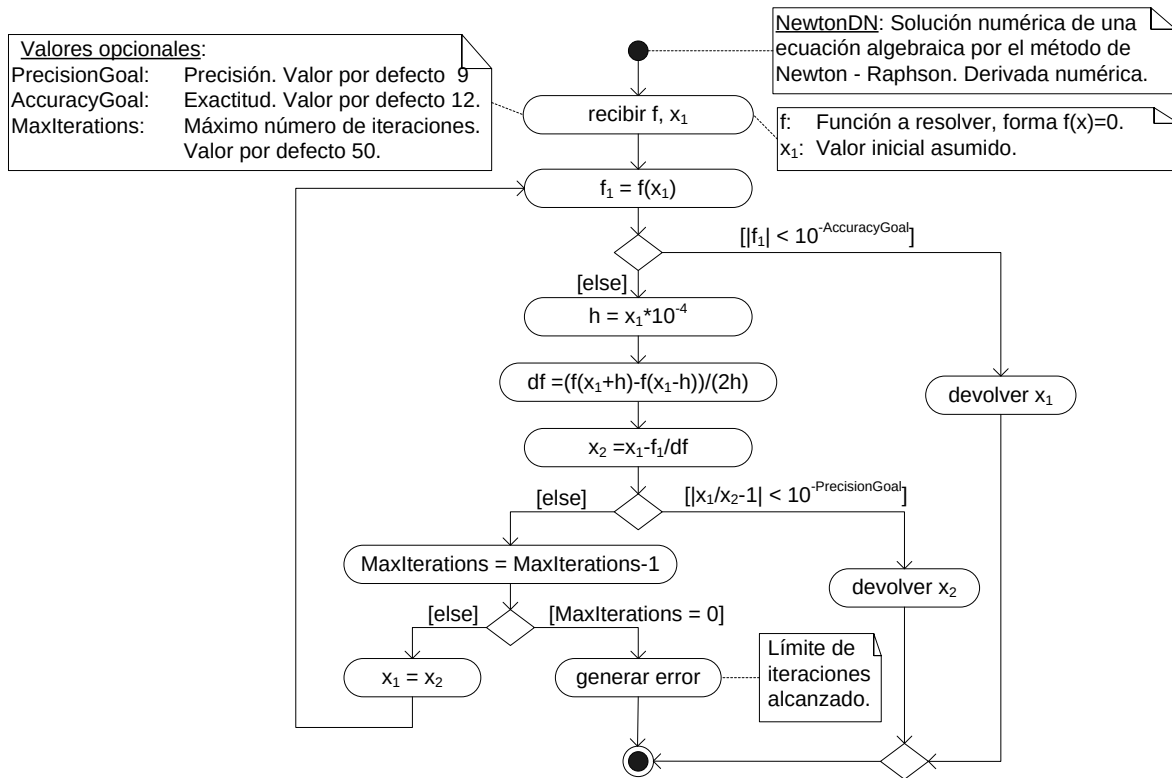


Figura 3.19. Método de Newton-Raphson. Derivada numérica

NewtonDN::maxit="Se ha alcanzado el límite de iteraciones: `1`, sin haber encontrado la solución";

```

NewtonDN[f_, x01_, OptionsPattern[]]:=
Block[{pg, ag, mi, li, x1=x01, x2, f1, h, df},
{pg, ag, mi}=OptionValue[{PrecisionGoal, AccuracyGoal,
MaxIterations}];
li=mi;
While[True,
f1=f[x1];
If[Abs[f1]<10^-ag, x2=x1; Break[]];
h=x1*10^-4;
df=(f[x1+h]-f[x1-h])/(2*h);
x2=x1-f1/df;
If[Abs[x1/x2-1]<10^-pg, Break[]];
If[--mi==0, Message[NewtonDN::maxit, n]; Break[]];
x1=x2];
x2]

```

...

End[]

...

EndPackage[]

Probamos el programa volviendo a resolver la ecuación 3.11:

```
Get["Alge1`", Path->"C:\MAT205"]
```

```
f[x_]:=x^3+2*x^2+3*x+4
```

```
NewtonDN[f,1.1]//InputForm
-1.650629191439391
```

Y con 12 dígitos de precisión se obtiene (como era de esperar) el mismo resultado que con el algoritmo básico:

```
NewtonDN[f,1.1,PrecisionGoal->12,AccuracyGoal->Infinity]//InputForm
-1.6506291914393882
```

14.2.3. Ejemplos

4. Resuelva la ecuación (3.1): $f(y)=y-(52+3*y^{0.5}-8*y^{0.8})^{0.36}=0$, empleando el método de Newton Raphson, con 14 dígitos de precisión (sin importar la exactitud), con un valor inicial igual a 1.1.

```
Get["Algel`",Path->"C:\MAT205"]
```

```
f[y_]:=y-(52+3*Sqrt[y]-8*y^0.8)^0.36
```

```
NewtonDN[f,1.1,PrecisionGoal->14,AccuracyGoal->Infinity]//InputForm
3.60967088614005
```

5. Elabore una función que calcule el valor de "V" resolviendo la ecuación de Benedict-Webb-Rubin que se presenta al pie de la pregunta. Los valores de a , A_0 , b , B_0 , c , C_0 , α y γ son valores opcionales cuyos valores por defecto son: $a=7.11671$, $A_0=1.44373 \times 10^1$, $b=1.09131 \times 10^{-1}$, $B_0=1.77813 \times 10^{-1}$, $c=1.51276 \times 10^6$, $C_0=3.31935 \times 10^6$, $\alpha=2.81086 \times 10^{-3}$, $\gamma=6.66849 \times 10^{-2}$. "P" y "T" son parámetros con valores por defecto iguales a 30 y 600 respectivamente y "R" es una constante igual a 0.08206. El valor inicial de "V" debe ser calculado con la ecuación: $V=RT/P$.

$$P = \frac{RT}{V} + \frac{B_0 RT - A_0 - C_0 T^{-2}}{V^2} + \frac{b RT - a}{V^3} + \frac{a \alpha}{V^6} + \frac{c}{V^3 T^2} \left(1 + \frac{\gamma}{V^2} \right) e^{\frac{-\gamma}{V^2}}$$

Importamos el paquete "Algel":

```
Get["Algel`",Path->"C:\MAT205"]
```

Asignamos los valores opcionales de la ecuación:

```
Options[BWR]={a->7.11671,A0->1.44373*10,b->1.09131*10^-1,
  B0->1.77813*10^-1,c->1.51276*10^6,C0->3.31935*10^6,
  alpha->2.81086*10^-3,gamma->6.66849*10^-2};
```

Y programamos la función resolviéndola con el método de Newton:

```
BWR[P_: 30,T_: 600,OptionsPattern[]]:=
  With[{R=0.08206},
    Block[{f,a,A0,b,B0,c,C0,alpha,gamma,V1},
      {a,A0,b,B0,c,C0,alpha,gamma}=OptionValue[{a,A0,b,B0,c,C0,alpha,gamma}];
      f[V_]:=R*T/V+(B0*R*T-A0-C0*T^-2)/V^2+
        (b*R*T-a)/V^3+alpha/V^6+
        c/(V^3*T^2)*(1-gamma/V^2)*Exp[-gamma/V^2]-P;
      V1=R*T/P;
      NewtonDN[f,V1]]]
```

El resultado con los valores por defecto es:

```
BWR[]
1.30183
```

Para P=35 se obtiene:

BWR[35]
1.05404

Para P=37 y T=600 se obtiene:

BWR[37,650]
1.18788

Y para P=40, T=700 y a=7.12345 se

BWR[40,700,a->7.12345]
1.24758

6. Elabore una función que calcule los valores de Y_{A1} , Y_{B1} , L_N , X_{BN} y V_1 , resolviendo la ecuación que se presenta al pie de la pregunta, donde X_{AN} , M , C_{A0} y C_{B0} son parámetros opcionales con los siguientes valores por defecto: $X_{AN}=0.12$, $M=32.1$, $C_{A0}=20.2$ y $C_{B0}=3.42$. Se sabe que el valor de Y_{A1} está comprendido entre 0.0001 y 0.999 por lo que un buen valor inicial por defecto es 0.5.

$$f(Y_{A1}) = L_N + V_1 - M = 0$$

$$Y_{B1} = 2Y_{A1} + Y_{A1}^2 + 3.44Y_{A1}^3$$

$$X_{BN} = 2.2 + 5.1X_{AN} - 7.2X_{AN}^{2.2}$$

$$L_N = \frac{C_{A0}Y_{B1} - C_{B0}Y_{A1}}{X_{AN}Y_{B1} - X_{BN}Y_{A1}}$$

$$V_1 = \frac{C_{A0} - L_N X_{AN}}{Y_{A1}}$$

Importamos el paquete "Algel":

```
Get["Algel`", Path->"C:\MAT205"]
```

Asignamos los valores por defecto a la función:

```
Options[fya1]={XAN->0.12,M->32.1,CA0->20.2,CB0->3.42,YA10->0.5};
```

Programamos la función resolviéndola con el método de Newton:

```
fya1[OptionsPattern[]]:=
Block[{f,YA1,YB1,LN,XAN,XBN,V1,M,CA0,CB0,YA10},
{XAN,M,CA0,CB0,YA10}=OptionValue[{XAN,M,CA0,CB0,YA10}];
XBN=2.2+5.1*XAN-7.2*XAN^2.2;
f[YA1_]:= (YB1=2*YA1+YA1^2+3.44*YA1^3;
LN=(CA0*YB1-CB0*YA1)/(XAN*YB1-XBN*YA1);
V1=(CA0-LN*XAN)/YA1;
LN+V1-M);
YA1=NewtonDN[f,YA10];
{SymbolName[Unevaluated[YA1]]->YA1,
SymbolName[Unevaluated[YB1]]->YB1,
SymbolName[Unevaluated[LN]]->LN,
SymbolName[Unevaluated[XBN]]->XBN,
SymbolName[Unevaluated[V1]]->V1}}
```

Con los valores por defecto se obtiene:

```
fya1[]
{YA1->0.411749,YB1->1.23317,LN->-23.9345,XBN->2.74415,V1->56.0345}
```

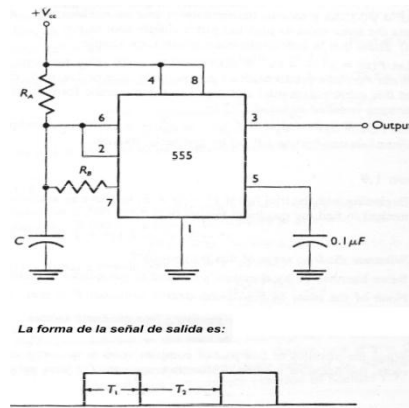
Para $X_{AN}=0.30$ se obtiene:

```
fya1[XAN->0.3]
```

{YA1->0.476448,YB1->1.55195, LN->-27.8044,XBN->3.22067,V1->59.9044}

14.2.4. Ejercicios

10. Cree un paquete con el nombre "Algelj" y en el mismo elabore el programa para el método de Newton Raphson con diferenciación numérica empleando la instrucción "For".
11. Cree un paquete con el nombre "Algelk" y en el mismo elabore el programa para el método de Newton Raphson con diferenciación numérica empleando la instrucción "Goto".
12. Cree un paquete con el nombre "Algell" y en el mismo elabore el programa para el método de Newton Raphson con diferenciación numérica empleando la instrucción "Do".
13. La ecuación de Redlich-Kwong: $P = \frac{RT}{v-b} - \frac{A}{v(v+b)}$. Puede ser empleada para calcular el volumen de gases a presiones superiores a la atmosférica. En una prueba experimental se han determinado los siguientes valores para el propano: $P=87.3$, $T=486.9$, $v=12.005$, $A=0.0837$ y $R=1.98$. Con estos datos calcule la constante "b" de la ecuación de Redlich-Kwong, empleando el método de Newton Raphson con diferenciación numérica con 16 dígitos de precisión o 16 de exactitud (lo que ocurra primero) y trabajando con 32 dígitos de precisión. Para estimar el valor inicial del método grafique la ecuación entre -2 y 4, como una función de "b", empleando los valores por defecto, siendo el título de la gráfica la función, las etiquetas de los ejes "b" y "f(b)", con líneas de división verde punteadas.
14. En el circuito que se muestra en la figura:



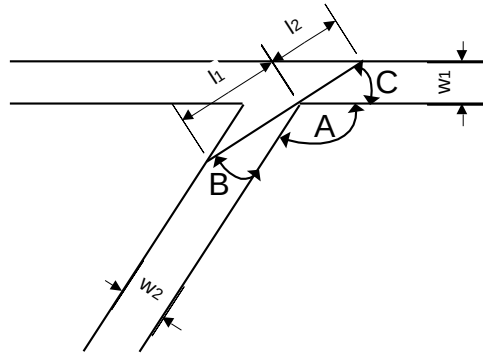
Se cumple la siguiente relación:

$$T_2 = \frac{R_A R_B C}{R_A + R_B} * \ln \left(\frac{R_A - 2R_B}{2R_A - R_B} \right)$$

Para un circuito específico se tienen los siguientes valores: $R_A = 8670$, $C = 0.01 \times 10^{-6}$, $T_2 = 1.4 \times 10^{-4}$. Calcule el valor de la resistencia R_B con el método de Newton Raphson con diferenciación numérica con 14 dígitos de precisión (sin importar la exactitud), mostrando en una lista los valores de R_B en cada iteración. Para estimar el valor inicial grafique la ecuación como una función de " R_B ".

15. En una mina existen dos túneles que se unen en un ángulo A, tal como se muestra en la figura. Para realizar trabajos en el interior de la mina se requiere introducir escaleras y mientras más largas sean estas me-

jor. Por consiguiente antes de adquirir las escaleras es necesario determinar la longitud de la escalera más larga que puede girar por la intersección. Para una mina en particular se tienen los siguientes valores: $A = 123^\circ$, $w_1 = 7$ pies y $w_2 = 9$. Calcule la longitud de la escalera más larga que puede ser introducida en esta mina con 20 dígitos de exactitud (sin importar la precisión) y trabajando con 40 dígitos. En el cálculo desprecie el espesor de la escalera y asuma que la misma no está inclinada, recuerde que el máximo se determina igualando la derivada de la función a cero.



15. ECUACIONES POLINOMIALES

Con los métodos estudiados en el anterior capítulo se puede encontrar una de las soluciones de la ecuación polinomial de la forma:

$$P_n(x) = a_1 x^n + a_2 x^{n-1} + a_3 x^{n-2} + a_4 x^{n-3} + \dots + a_n x + a_{n+1} = 0 \quad (1)$$

Para encontrar todas las soluciones: reales e imaginarias, existen otros métodos algunos de los cuales estudiaremos en este tema.

En consecuencia el propósito del presente tema es que al concluir el mismo estén capacitados para encontrar las soluciones reales e imaginarias de las ecuaciones polinomiales que aparecen en la solución de problemas reales en el campo de la ingeniería.

15.1. Solución de ecuaciones polinomiales en Mathematica

Como sucede en la mayoría de los casos Mathematica dispone de varias instrucciones para encontrar las soluciones de ecuaciones polinomiales, permitiendo inclusive encontrar soluciones analíticas.

El comando más general (que puede ser empleado también para resolver algunas de las ecuaciones algebraicas estudiadas en el anterior capítulo) y que permite trabajar con ecuaciones analíticas es "Solve", cuya sintaxis es la siguiente:

$$\text{Solve}[\{\text{ecuaciones}\}, \{\text{variables}\}]$$

Donde las *ecuaciones* se escriben en la forma *lado_izquierdo==lado_derecho*. En caso de existir una sola ecuación no se requiere una lista, igualmente, si sólo existe una variable no se requiere una lista. Si se omite la lista de variables Mathematica trata de resolver las ecuaciones para todas las variables simbólicas que aparecen en las mismas.

Así por ejemplo para encontrar las soluciones de la ecuación cuadrática: $ax^3+bx^2+cx+d=0$, escribimos:

```
Solve[{a*x^2+b*x+c==0},{x}]
```

$$\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\}$$

Y como vemos nos devuelve la conocida solución general de una ecuación cuadrática. Como se trata de una ecuación y una variable podemos omitir las listas:

```
Solve[a*x^2+b*x+c==0,x]
```

$$\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\}$$

Sin embargo no podemos omitir la variable porque Mathematica asumiría que los coeficientes (a, b, c y d) son también variables y trataría de encontrar soluciones para los coeficientes, generando en consecuencia un mensaje de error:

```
Solve[a*x^2+b*x+c==0]
```

```
Solve::svars:Equations may not give solutions for all "solve" variables. >>
{{c->-b x-a x^2}}
```

El resultado puede ser empleado para encontrar soluciones numéricas, por ejemplo para encontrar la solución de la ecuación: $3x^2+2x-10=0$, podemos encontrar primero la solución general:

```
r=Solve[a*x^2+b*x+c==0,x]
```

$$\left\{ \left\{ x \rightarrow \frac{-b - \sqrt{b^2 - 4ac}}{2a} \right\}, \left\{ x \rightarrow \frac{-b + \sqrt{b^2 - 4ac}}{2a} \right\} \right\}$$

Y en la solución reemplazar los valores de los coeficientes:

```
r/.{a->3,b->2,c->-10}
```

$$\left\{ \left\{ x \rightarrow \frac{1}{6} (-2 - 2\sqrt{31}) \right\}, \left\{ x \rightarrow \frac{1}{6} (-2 + 2\sqrt{31}) \right\} \right\}$$

O en forma aproximada:

```
r/.{a->3,b->2,c->-10} //N
```

```
{{x->-2.18925},{x->1.52259}}
```

Por supuesto, podemos escribir también directamente la ecuación a resolver (y podemos obviar la variable porque en este caso no existen más valores simbólicos):

```
Solve[3*x^2+2*x-10==0] //N
```

```
{{x->-2.18925},{x->1.52259}}
```

Puesto que en esta materia nos interesa encontrar las soluciones numéricas, es más práctico emplear "NSolve", el cual es muy similar a "Solve", pero en lugar de encontrar soluciones simbólicas, encuentra directamente las soluciones numéricas, su sintaxis es la siguiente:

NSolve[función o ecuación polinomial, variable]

Y cuando se resuelve un sistema de ecuaciones polinomiales:

NSolve[{Ec. 1, Ec. 2, ..., Ec. n}, {Var. 1, Var. 2, ..., Var. n}]

Así por ejemplo para encontrar las soluciones reales e imaginarias de la ecuación polinomial:

$$P_3(x) = x^3 + 2x^2 + 3x + 4 = 0$$

Escribimos la siguiente instrucción:

```
NSolve[x^3+2*x^2+3*x+4,x]
```

```
{{x->-1.65063},{x->-0.174685-1.54687 i},{x->-0.174685+1.54687 i}}
```

O alternativamente programamos primero la función:

```
f[x_]:=x^3+2*x^2+3*x+4
```

Y la resolvemos con "NSolve":

```
NSolve[f[x],x]
```

```
{{x->-1.65063},{x->-0.174685-1.54687 i},{x->-0.174685+1.54687 i}}
```

La ventaja de programar primeramente la función, es la de poder emplear la función con otras instrucciones: por ejemplo para hacer una gráfica. Así por ejemplo, para resolver la siguiente ecuación polinomial:

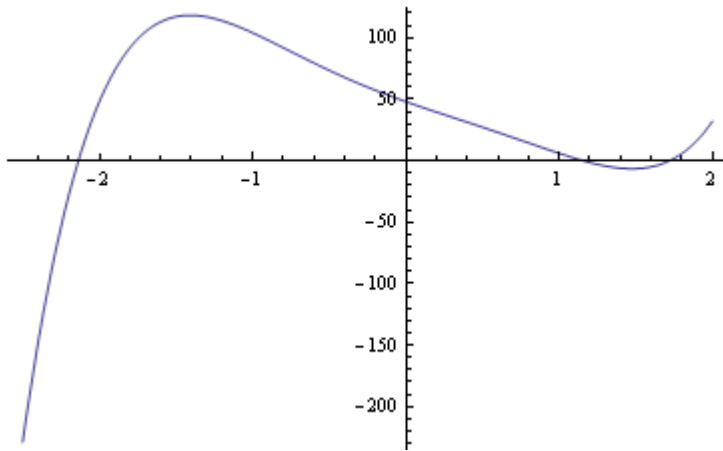
$$P_5(x) = 5x^5 - 3x^4 - 10x^3 + 10x^2 - 44x + 48 = 0$$

Creemos la función:

```
f[x_]:=5*x^5-3*x^4-10*x^3+10*x^2-44*x+48
```

Y la podemos emplear la misma para crear una grafica y ver así el lugar de las soluciones reales, (claro que con "NSolve" no necesitamos los valores iniciales):

```
Plot[f[y],{y,-2.5,2}]
```



Ahora, si encontramos las soluciones con "NSolve", podremos corroborar las soluciones gráficas:

```
NSolve[f[x],x]
{{x->-2.13671},{x->-0.0696045-1.50153 i},
{x->-0.0696045+1.50153 i},{x->1.15645},{x->1.71947}}
```

Como se puede observar las soluciones reales concuerdan con la gráfica y además se han calculado las dos soluciones imaginarias.

Para familiarizarnos con "NSolve" resolvamos dos ecuaciones más:

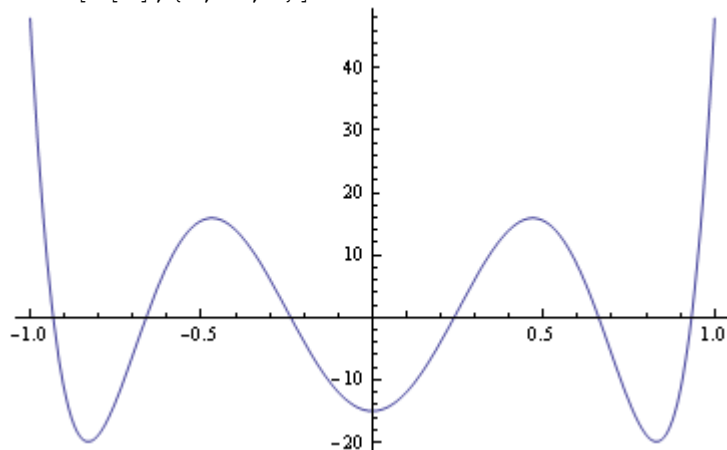
$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

Primero programamos la función:

```
f[x_]:=693*x^6-945*x^4+315*x^2-15
```

La graficamos para ver el lugar de las soluciones reales:

```
Plot[f[z],{z,-1,1}]
```



Como podemos observar en este caso todas las soluciones son reales, lo que podemos corroborar con "NSolve":

```
NSolve[f[x],x]
{{x->-0.93247},{x->-0.661209},{x->-0.238619},{x->0.238619},
{x->0.661209},{x->0.93247}}
```

Resolvamos ahora la siguiente ecuación de octavo grado:

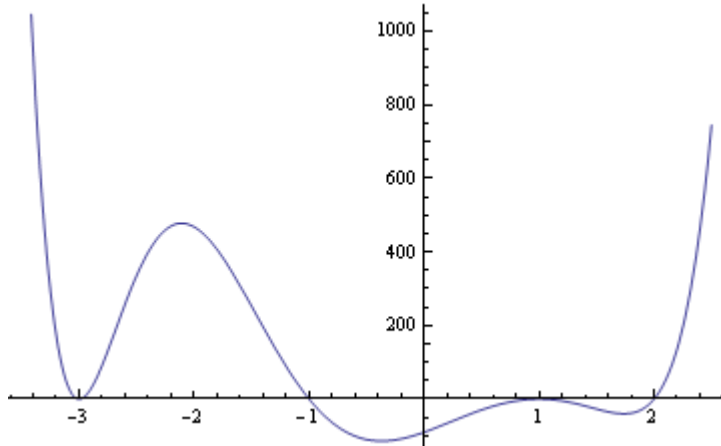
$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

Primero programamos la función:

```
f[x_] := x^8 + x^7 - 9*x^6 + 13*x^5 + 21*x^4 - 125*x^3 + 77*x^2 + 111*x - 90
```

La graficamos para ver el lugar de las soluciones reales:

```
Plot[f[x], {x, -3.5, 2.5}]
```



Como se puede observar en este caso existen 2 soluciones iguales (una en -3 y otra en 1), las cuales son encontradas también con "NSolve":

```
NSolve[f[x], x]
```

```
{{x -> -3.}, {x -> -3.}, {x -> -1.}, {x -> 1.}, {x -> 1. - 2. i}, {x -> 1. + 2. i}, {x -> 1.}, {x -> 2.}}
```

Vemos entonces que "NSolve" es un método bastante confiable para resolver ecuaciones polinomiales e inclusive nos permite resolver sistemas de ecuaciones polinomiales, como el siguiente:

$$2x^2 + 3y^2 - 5z = 45$$

$$x - 2y^2 + 8z^2 = 23$$

$$x + y^2 - z^2 = 42$$

```
NSolve[{2*x^2+3*y^2-5*z==45, x-2*y^2+8*z^2==23, x+y^2-z^2==42}, {x, y, z}]
```

```
{{x -> 1.24049 - 3.57247 i, y -> -4.28527 - 0.625247 i, z -> -2.70592 - 0.33006 i},
 {x -> 1.24049 + 3.57247 i, y -> -4.28527 + 0.625247 i, z -> -2.70592 + 0.33006 i},
 {x -> 0.13367, y -> 4.43089, z -> 2.78684},
 {x -> 1.24049 - 3.57247 i, y -> 4.28527 + 0.625247 i, z -> -2.70592 - 0.33006 i},
 {x -> 1.24049 + 3.57247 i, y -> 4.28527 - 0.625247 i, z -> -2.70592 + 0.33006 i},
 {x -> 1.88535, y -> -4.12375, z -> 2.62501},
 {x -> 1.88535, y -> 4.12375, z -> 2.62501},
 {x -> 0.13367, y -> -4.43089, z -> 2.78684}}
```

Y como se puede observar se obtienen todas las soluciones del sistema (reales e imaginarias). Por supuesto el grado de los polinomios del sistema puede ser superior a 2.

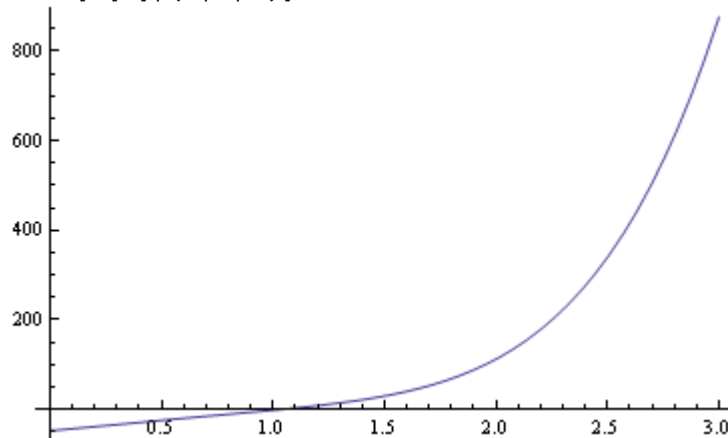
15.1.1. Ejemplos

1. Programe el polinomio: $-48 + 44x + 10x^2 - 10x^3 - 3x^4 + 5x^5 = 0$. Grafique el polinomio entre 0 y 3; encuentre las soluciones reales e imaginarias y sepárelas en dos listas. Guarde la solución real en "x1" y resuelva la siguiente ecuación no lineal empleando el método de Newton Raphson con diferenciación numérica: $x^{3.2} + 4x^{1.2} - 8x - x_1 = 0$.

Primero programamos y graficamos el polinomio:

```
f[x_] := -48 + 44*x + 10*x^2 - 10*x^3 - 3*x^4 + 5*x^5
```

```
Plot[f[x],{x,0,3}]
```



Como se puede observar sólo existe una solución real. Ahora resolvemos la ecuación con "NSolve":

```
r=NSolve[f[x],x]
{{x->-1.4951-0.98787 i},{x->-1.4951+0.98787 i},
 {x->1.04206},{x->1.27407 -1.11608 i},{x->1.27407 +1.11608 i}}
```

Para separar las soluciones reales de las imaginarias reemplazamos las reglas para trabajar solo con los valores:

```
r=x/.r
{-1.4951-0.98787 i,-1.4951+0.98787 i,1.04206,1.27407 -1.11608 i,
 1.27407 +1.11608 i}
```

Ahora seleccionamos las soluciones complejas con "Cases" (por supuesto podemos hacer lo mismo con "Select", "Table", "Reap-Sow y Do", etc.):

```
Cases[r,_Complex]
{-1.4951-0.98787 i,-1.4951+0.98787 i,1.27407 -1.11608 i,1.27407 +1.11608 i}
```

Hacemos lo mismo con la solución real:

```
Cases[r,_Real]
{1.04206}
```

Y guardamos esta solución en la variable "x1":

```
x1=%[[1]]
1.04206
```

Ahora importamos el paquete "Alge1", para emplear "NewtonDN":

```
Get["Alge1",Path->"C:\MAT205"]
```

Programamos la segunda function y la resolvemos con "NewtonDN":

```
f2[x_]:=x^3.2+4*x^1.2-8*x-x1
```

```
NewtonDN[f2,1.1]
1.88109
```

2. Los coeficientes de la siguiente ecuación algebraica (r_1, r_2, r_3, r_4) : $r_1 x^{2.1} + r_2 x^{1.3} - r_3 x - r_4 = 0$, son las soluciones reales de la ecuación polinomial: $r^4 - 11.1r^3 + 43.31r^2 - 69.801r + 38.502 = 0$. Programe la ecuación algebraica, gráfiquela entre 0 y 5 y encuentre la solución de la ecuación

con el método de Regula Falsi, buscando el segmento de solución a partir de 0 con incrementos de 0.1 en 0.1.

Primero programamos el polinomio:

```
Remove[r,f,fr]
```

```
f[r_]:=r^4-11.1*r^3+43.31*r^2-69.801*r+38.502
```

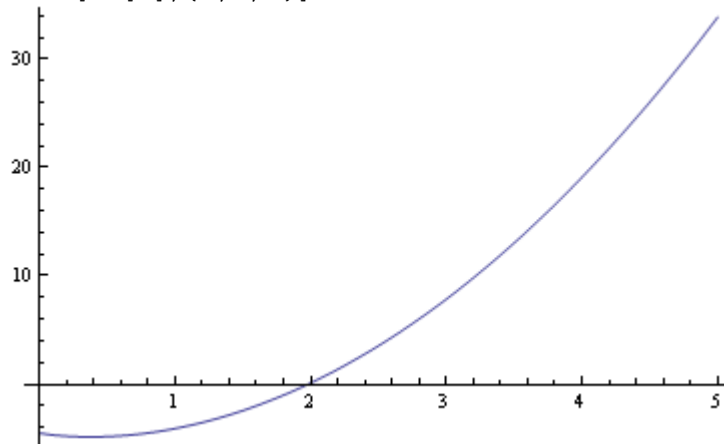
Encontramos las soluciones con "NSolve" las guardamos en las variables "r1", "r2", "r3" y "r4":

```
{r1,r2,r3,r4}=x/.NSolve[f[x],x]
{1.2,2.3,3.1,4.5}
```

Ahora programamos la ecuación algebraica y la graficamos entre 0 y 5.

```
fr[x_]:=r1*x^2.1+r2*x^1.3-r3*x-r4
```

```
Plot[fr[x],{x,0,5}]
```



Finalmente buscamos el segmento de solución con "Incret1" y encontramos la solución con "RegulaFalsi":

```
Get["Algel1`",Path->"C:\HPV"]
```

```
{x1,x2}=Incret1[fr,0,0.1]
{1.9,2.}
```

```
RegulaFalsi[fr,x1,x2]
1.98189
```

3. Resuelva el sistema de ecuaciones que se presenta al final de la pregunta donde los coeficientes " x_1 ", " x_2 ", " x_3 " y " x_4 " son las soluciones reales de la ecuación polinomial. El valor de " y " debe ser calculado con el método de la "Biseccion", buscando la solución a partir 1 con incrementos de 0.1, y el valor de " z " con el método de la secante, siendo los valores iniciales 3.1 y 3.2:

$$39.039-128.34x+162.09x^2-98.35x^3+30.6x^4-6x^5+x^6=0$$

$$x_1y^{0.7}+x_2y^{0.8}-x_3y^{0.9}-x_4y^{0.6}=16.37$$

$$z^{1.1}-4.2z^{1.4}+1.4z^{2.1}=1.93*y$$

Primero programamos el polinomio y encontramos las soluciones con "NSolve":

```
p[x_]:=39.039-128.34x+162.09x^2-98.35x^3+30.6x^4-6x^5+x^6
```

```
r=x/.NSolve[p[x],x]
{0.5-4. i,0.5+4. i,1.10011,1.19928,1.30117,1.39944}
```

Extraemos las soluciones reales y las guardamos en las variables "x1", "x2", "x3" y "x4":

```
{x1,x2,x3,x4}=Cases[r,_Real]
{1.10011,1.19928,1.30117,1.39944}
```

Ahora, con estos resultados programamos la segunda ecuación:

```
fy[y_]:=x1*y^0.7+x2*y^0.8+x3*y^0.9+x4*y ^0.6-16.37
```

Encontramos el segmento de solución con "Incr1" y la solución con "Bi-seccion":

```
{y1,y2}=Incr1[fy,1,0.1]
{4.7,4.8}
```

```
y=Biseccion[fy,y1,y2]
4.77703
```

Ahora con el valor de "y" calculado resolvemos la última ecuación con el método de la "Secante":

```
fz[z_]:=z^1.1-4.2*z^1.4+1.4*z^2.1-1.83-1.93*y
```

```
RegulaFalsi[fz,3.1,3.2]
5.49877
```

15.1.2. Ejercicios

1. Programe el polinomio: $x^6+x^5-8x^4+231x^3+21x^2-265x+150$ y gráfiquelo entre -7 y 0. Encuentre las soluciones reales e imaginarias y sepárelas en dos listas. Guarde las soluciones reales en las variables "x1" y "x2". Con los valores calculados resuelva la siguiente ecuación no lineal empleando el método de Wegstein: $y^{0.7}-9*y+3*y^{4.2}=x_1+x_2$, empleando un valor inicial igual a 3.1.
2. Los coeficientes de la siguiente ecuación algebraica (r_1, r_2, r_3, r_4): $r_1x^{2.1}+r_2x^{1.3}-r_3x-r_4=0$, son las soluciones reales de la ecuación polinomial: $12x^{11}+23x^{10}-8x^8-5x^6-3x^4+5x^2-3x-10$. Programe la ecuación algebraica, gráfiquela entre 1 y 1.5 y encuentre la solución de la ecuación con el método de Newton con diferenciación analítica, empleando un valor inicial igual a 0.5.
3. Resuelva el sistema de ecuaciones que se presenta al final de la pregunta donde los coeficientes "x1", "x2", "x3" y "x4" son la parte real de las soluciones de la ecuación polinomial. El valor de "y" debe ser calculado con el método Incremental, buscando la solución a partir 5.0 con incrementos de 0.1, y el valor de "z" con el método de la Regula Falsi, buscando los valores iniciales a partir de 1.2 con incrementos de 0.1:

$$x^4-3x^3+2x^2-8x+3=0$$

$$x_1y^{3.1}+x_2y^{2.8}+x_3y^{2.9}+x_4y^{1.6}=22.831$$

$$3.2*z-4/z^{1.9}+7/z^{1.2}=y+4.62$$

15.2. División sintética de un polinomio entre un monomio

La división sintética es la forma más eficiente de calcular el valor de un polinomio y el de su derivada, siendo esa la razón por la cual se la em-

plea como parte de algunos métodos que resuelven ecuaciones polinomiales, como es el método de Newton Raphson para ecuaciones polinomiales.

Cuando se divide un polinomio entre un monomio que está en la forma: " $x - x_1 = 0$ " (o lo que es lo mismo " $x = x_1$ "), donde " x_1 " es un número real, el residuo de la división es el valor del polinomio para " x " igual a " x_1 ", si el cociente de esta división se vuelve a dividir entre " $x - x_1$ ", el residuo de esta nueva división es la derivada primera de la función para " x " igual a " x_1 ".

Así por ejemplo para calcular el valor del siguiente polinomio, así como el de su derivada, para un valor de " x " igual a "2":

$$P_3(x) = x^3 + x^2 - 3x - 3 = 0$$

Llevamos a cabo dos divisiones sintéticas consecutivas, tal como se muestra a continuación:

$$\begin{array}{r|rrrr} x_1=2 & 1 & 1 & -3 & -3 \\ & & 2 & 6 & 6 \\ \hline & 1 & 3 & 3 & \boxed{3} \leftarrow \text{valor de la función: } f(2) \\ & & 2 & 10 & \\ \hline & 1 & 5 & \boxed{13} \leftarrow \text{derivada de la función: } f'(2) \end{array}$$

Como puede observar en este ejemplo, si " a " son los coeficientes del polinomio original, los coeficientes resultantes de la división sintética: " b " se calculan con:

$$b_i = a_i + b_{i-1}x_1 \quad \begin{cases} b_1 = a_1 \\ i = 2 \rightarrow n+1 \\ \text{residuo} = b_{n+1} \end{cases} \quad (2)$$

Donde " n " es el grado del polinomio.

Mathematica cuenta con dos funciones que nos permiten calcular el cociente y el residuo de la división de dos polinomios " p " y " q ", cuando ambos son función de una variable " x ".

PolynomialQuotient[p, q, x]

PolynomialRemainder[p, q, x]

Así para calcular en Mathematica el valor de la ecuación cúbica: $x^3 + x^2 - 3x - 3 = 0$ y su derivada para " $x=2$ ", escribimos las siguientes instrucciones:

```
f[x_]:=x^3+x^2-3*x-3
```

```
PolynomialRemainder[f[x],x-2,x]
3
```

```
PolynomialRemainder[PolynomialQuotient[f[x],x-2,x],x-2,x]
13
```

Es posible obtener el cociente y el residuo al mismo tiempo con la función "**PolynomialQuotientRemainder**", que tiene la siguiente sintaxis:

PolynomialQuotientRemainder[p, q, x]

Esta función devuelve una lista con el cociente y el residuo de la división, así aplicando esta función a la ecuación cúbica obtenemos:

```
PolynomialQuotientRemainder[f[x],x-2,x]
```



```
{3+3 x+x^2,3}
```

Entonces podemos emplear el cociente (el primer elemento de la lista) para calcular la derivada primera del polinomio:

```
PolynomialRemainder[%[[1]],x-2,x]
13
```

No obstante, es más eficiente trabajar directamente con los coeficientes del polinomio que con toda la ecuación, razón por la cual es conveniente elaborar una función que, empleando la ecuación (2), devuelva los coeficientes y el residuo de la división sintética.

Puesto que sólo trabajaremos con los coeficientes es conveniente poder extraerlos de la ecuación polinomial, para ello Mathematica cuenta con la función "CoefficientList":

CoefficientList[Polinomio, variable]

Así para extraer los coeficientes de la ecuación cúbica se escribe:

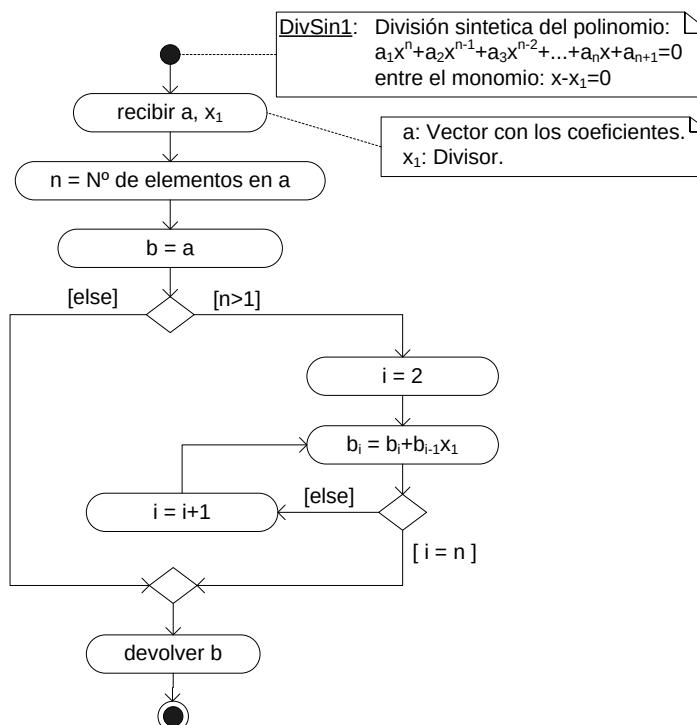
```
CoefficientList[f[x],x]
{-3,-3,1,1}
```

Que como se ve devuelve la lista de coeficientes comenzando con el término independiente y terminando en el coeficiente correspondiente a la mayor potencia. Para que los coeficientes correspondan al orden de la ecuación (1), simplemente se invierte la lista:

```
Reverse[%]
{1,1,-3,-3}
```

15.2.1. Programa

Elaboraremos una función que lleve a cabo la división sintética de un polinomio entre un monomio, empleando sólo los coeficientes del polinomio (es decir aplicando la ecuación 2). El algoritmo es el siguiente:



El programa elaborado en base a dicho algoritmo y añadido al paquete "Alge2" es el siguiente:

```
BeginPackage["Alge2`"]

DivSin1::usage="DivSin1[a,x1], devuelve los coeficientes de la
  división sintética del polinomio con coeficientes \"a\" entre
  el monomio: \"x-x1=0\"."

Begin["Private`"]

DivSin1[a_List,x1_?NumberQ]:=
  Block[{n=Length[a],b=a},
    Do[b[[i]]=b[[i]]+b[[i-1]]*x1,{i,2,n}];
    b]

End[]

EndPackage[]
```

Ahora importamos el paquete en un cuaderno y probamos la función dividiendo la ecuación cúbica: " $x^3+x^2-3x-3=0$ ", entre " $x-2=0$ ":

```
Get["Alge2`",Path->"E:\MAT205"]
```

```
DivSin1[{1,1,-3,-3},2]
{1,3,3,3}
```

Donde los tres primeros valores corresponden a los coeficientes del polinomio residual (x^2+3x+3) y el último es el residuo de la división sintética, o lo que es lo mismo, el valor del polinomio para " $x=2$ ". Como se puede observar los resultados son los mismos que en el ejemplo manual y con las instrucciones: "PolynomialRemainder", "PolynomialQuotient" y "PolynomialQuotientRemainder".

Como otro ejemplo realizamos la división sintética del siguiente polinomio entre " $x=3.1$ ":

$$P_7(x)=7x^7+3x^5-8x^4+3x-9=0$$

```
f[x_]:=7*x^7+3*x^5-8*x^4+3*x-9
```

```
a=Reverse[CoefficientList[f[x],x]]
{7,0,3,-8,0,0,3,-9}
```

```
b=DivSin1[a,3.1]
{7,21.7,70.27,209.837,650.495,2016.53,6254.25,19379.2}
```

Donde los siete primeros elementos corresponden al cociente de la división: $7x^6+21.7x^5+70.27x^4+209.837x^3+650.495x^2+2016.53x+6254.25$ y el último es el valor de la función para " $x=3.1$ " (19379.2). Observe que para las potencias que no aparecen en la ecuación se deben escribir ceros.

Podemos corroborar el resultado obtenido con "DivSin1", empleando "PolynomialQuotientRemainder":

```
c=PolynomialQuotientRemainder[f[x],x-3.1,x]
{6254.25+2016.53 x+650.495 x^2+209.837 x^3+70.27 x^4+21.7 x^5+7 x^6,19379.2}
```

Y como vemos ambos resultados concuerdan. Calculemos ahora la derivada primera del polinomio, volviendo a dividir el cociente de la primera división sintética (sin el último término, es decir sin el residuo):

```
b=Most[b]
{7,21.7,70.27,209.837,650.495,2016.53,6254.25}
```

```
DivSin1[b,3.1]
{7,43.4,204.81,844.748,3269.21,12151.1,43922.6}
```

Por lo tanto la derivada primera del polinomio para "x=3.1" es 43922.6. Podemos corroborar este resultado volviendo a aplicar "PolynomialDivision" al cociente de la primera división:

```
PolynomialRemainder[c[[1]],x-3.1,x]
43922.6
```

Y también derivando directamente la función para x=3.1:

```
f'[3.1]
43922.6
```

15.2.2. Ejemplos

4. Mediante división sintética calcule el valor del polinomio: $x^8 - x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$ y el de su derivada primera para valores de "x" iguales a "1.1, 3.9, 12.4 y 14.1". Compruebe los resultados con: PolynomialRemainder, PolynomialQuotient y Derivative.

Primero importamos el paquete "Alge2" (si aún no lo hemos hecho):

```
Get["Alge2",Path->"C:\MAT205"]
```

Programamos la ecuación polinomial y extraemos los coeficientes de la misma:

```
p[x_]:=x^8-x^7-9 x^6+13 x^5+21 x^4-125 x^3+77 x^2+111 x-90
```

```
a=Reverse[CoefficientList[p[x],x]]
{1,-1,-9,13,21,-125,77,111,-90}
```

Guardamos los valores de "x" en la variable "vx":

```
vx={1.1,3.9,12.4,14.1}
{1.1,3.9,12.4,14.1}
```

Y llevamos a cabo las dos divisiones sintéticas (para los cuatro valores) en un ciclo "Do", guardando los resultados en las variables "b" y "c":

```
Do[b=DivSin1[a,vx[[i]]];
  c=DivSin1[Most[b],vx[[i]]];
  Print["p(",vx[[i]],")=",b[[Length[b]]]];
  Print["p'(",vx[[i]],")=",c[[Length[c]]]],{i,4}]
p(1.1)=-5.17145
p'(1.1)=-50.1582
p(3.9)=18814.9
p'(3.9)=51460.5
p(12.4)=4.85239*10^8
p'(12.4)=3.20977*10^8
p(14.1)=1.38848*10^9
p'(14.1)=8.04019*10^8
```

Corroboramos estos resultados con "PolynomialRemainder" y "PolynomialQuotient":

```
Do[
  Print["p(",vx[[i]],")=",
    PolynomialRemainder[p[x],x-vx[[i]],x]];
  Print["p'(",vx[[i]],")=",PolynomialRemainder[
```

```

PolynomialQuotient[p[x],x-vx[[i]],x],
x-vx[[i]],x]],{i,4}]
p(1.1)=-5.17145
p'(1.1)=-50.1582
p(3.9)=18814.9
p'(3.9)=51460.5
p(12.4)=4.85239*10^8
p'(12.4)=3.20977*10^8
p(14.1)=1.38848*10^9
p'(14.1)=8.04019*10^8

```

Como se puede ver los resultados concuerdan en todos los casos. Igualmente corroboramos los resultados con la función "p" y Derivative "'":

```

Do[Print["p(",vx[[i]],")=",p[vx[[i]]]];
Print["p'(",vx[[i]],")=",p'[vx[[i]]]],{i,Length[vx]}]
p(1.1)=-5.17145
p'(1.1)=-50.1582
p(3.9)=18814.9
p'(3.9)=51460.5
p(12.4)=4.85239*10^8
p'(12.4)=3.20977*10^8
p(14.1)=1.38848*10^9
p'(14.1)=8.04019*10^8

```

Una vez más los resultados concuerdan en todos los casos.

5. Mediante división sintética calcule el valor del polinomio: $693x^6 - 945x^4 + 315x^2 - 15 = 0$ y el de su derivada primera para valores de "x" iguales a "3.2, 4.1, 5.3, 5.9, 6.2 y 6.8". Corrobore los resultados con "PolynomialQuotientRemainder".

El procedimiento es esencialmente el mismo que el del anterior ejemplo:

```

Get["Alge2`",Path->"C:\HPV"]

p[x_]:=693x^6-945x^4+315x^2-15

a=Reverse[CoefficientList[p[x],x]]
{693,0,-945,0,315,0,-15}

vx={3.2,4.1,5.3,5.9,6.2,6.8}

Do[b=DivSin1[a,vx[[i]]];
c=DivSin1[Most[b],vx[[i]]];
Print["p(",vx[[i]],")=",b[[Length[b]]]];
Print["p'(",vx[[i]],")=",c[[Length[c]]]],
{i,Length[vx]}];
p(3.2)=648223.
p'(3.2)=1.27335*10^6
p(4.1)=3.03007*10^6
p'(4.1)=4.55936*10^6
p(5.3)=1.46231*10^7
p'(5.3)=1.68292*10^7
p(5.9)=2.8097*10^7
p'(5.9)=2.89539*10^7
p(6.2)=3.79783*10^7
p'(6.2)=3.71958*10^7
p(6.8)=6.65092*10^7
p'(6.8)=5.92703*10^7

```

Para corroborar estos resultados empleamos ahora "PolynomialQuotientRemainder":

```
Do[{co, re1}=PolynomialQuotientRemainder[p[x], x-vx[[i]], x];
  re2=PolynomialQuotientRemainder[co, x-vx[[i]], x][[2]];
  Print["p(", vx[[i]], ")=", re1];
  Print["p'(", vx[[i]], ")=", re2],
  {i, Length[vx]}}
p(3.2)=648223.
p'(3.2)=1.27335*10^6
p(4.1)=3.03007*10^6
p'(4.1)=4.55936*10^6
p(5.3)=1.46231*10^7
p'(5.3)=1.68292*10^7
p(5.9)=2.8097*10^7
p'(5.9)=2.89539*10^7
p(6.2)=3.79783*10^7
p'(6.2)=3.71958*10^7
p(6.8)=6.65092*10^7
p'(6.8)=5.92703*10^7
```

Y como se puede observar se obtienen los mismos resultados.

15.2.3. Ejercicios

4. Elabore una función que, empleando la estructura While, lleve a cabo la división sintética de un polinomio entre un monomio, trabajando sólo con los coeficientes.
5. Elabore una función que, empleando la estructura For, lleve a cabo la división sintética de un polinomio entre un monomio, trabajando sólo con los coeficientes.
6. Elabore una función que, empleando la estructura Nest, lleve a cabo la división sintética de un polinomio entre un monomio, trabajando sólo con los coeficientes.
7. Elabore una función que, empleando la estructura Do, devuelva el valor del polinomio y el de su derivada para un valor dado de la variable independiente.
8. Mediante división sintética calcule el valor del polinomio: $20x^{20}+3x^{17}-4x^{15}+12x^{12}-9x^8+6x^6-2x^4-4x^3+3x^2-x+9$ y el de su derivada primera para valores de "x" iguales a "0.3, 0.9, 1.4, 2.8, 3.3, 3.9 y 4.3". Compruebe los resultados con: PolynomialRemainder, PolynomialQuotient y Derivative.
9. Mediante división sintética calcule el valor del polinomio: $7x^7+3x^5-8x^4+3x-9=0$ y el de su derivada primera para valores de "x" iguales a "3.3, 4.5, 5.3, 6.3, 6.8, 7.1, 7.6 y 8.2". Para corroborar los resultados programe el polinomio y compruebe las derivadas con: PolynomialQuotientRemainder.

15.3. Método de Newton - Raphson

Como ya vimos en el tema anterior, el método de *Newton - Raphson* (al igual que los otros métodos estudiados) permite encontrar una de las soluciones reales de una ecuación polinomial.

La raíz encontrada depende del valor inicial asumido. Si se dan valores iniciales cercanos a las soluciones, el método de *Newton - Raphson*, puede encontrar todas las soluciones reales del polinomio.

El método de *Newton - Raphson* no cambia cuando se trata de polinomios, sólo que ahora en lugar de trabajar con la función se trabaja con los coeficientes del polinomio, evaluando la función y su derivada mediante división sintética, que como vimos es el método más eficiente para evaluar polinomios y sus derivadas. Una vez encontrada una raíz, se puede emplear el polinomio residual (obtenido en la división sintética) para encontrar otra de las raíces y continuar con este procedimiento hasta que el polinomio residual es de segundo grado, en cuyo caso se resuelve con la fórmula general para una ecuación de segundo grado.

No obstante, el procedimiento descrito es una puerta abierta para la propagación del error, pues el error cometido al calcular la primera raíz (algo que casi siempre ocurre con los métodos numéricos), existe también en los coeficientes del polinomio residual y en consecuencia se propaga al cálculo de las otras raíces, teniendo las nuevas raíces un error cada vez mayor. Por esta razón, este procedimiento no se emplea con mucha frecuencia en la práctica, siendo preferible, aunque resulta moroso, calcular las soluciones con los coeficientes originales, empleando valores iniciales cercanos a las soluciones que se quieren calcular. Para encontrar dichos valores iniciales existen métodos (como "QD"), y por supuesto también es posible graficar la función para ver el lugar de las soluciones.

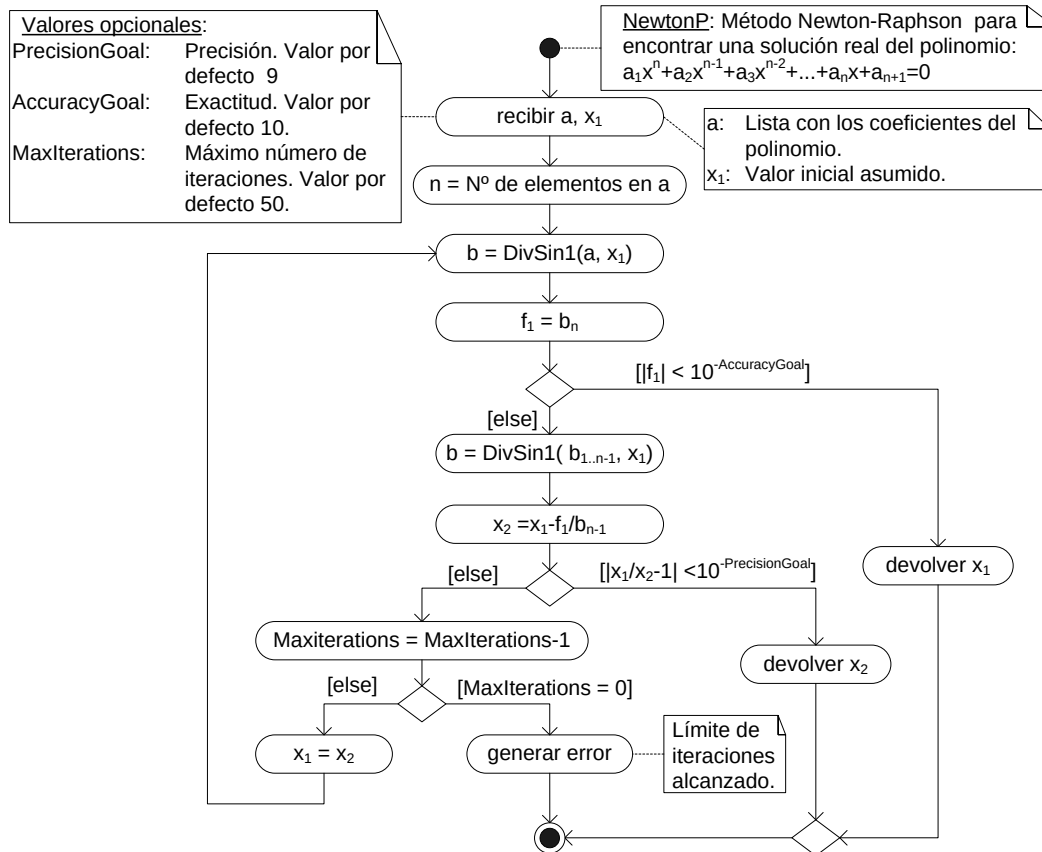
15.3.1. Programa

El algoritmo del método de *Newton - Raphson*, para el cálculo de una de las soluciones reales de un polinomio, se muestra en la siguiente página y el código añadido a la unidad "Alge2" es el siguiente:

```
BeginPackage["Alge2`"]
...
NewtonP::usage="NewtonP[a,x1], encuentra una de las soluciones del
  polinomio cuya lista de coeficientes se pasa en \"a\", comenzando
  la búsqueda con el valor inicial \"x1\"."
...
Begin["Private`"]
...
Options[NewtonP]={PrecisionGoal->9,AccuracyGoal->10,MaxIterations->50}

NewtonP::maxit="Newton ha alcanzado el límite de iteraciones: `1`,
  sin haber encontrado la solución."

NewtonP[a_List,x01_?NumberQ,OptionsPattern[]]:=
  Block[{n=Length[a],x1,x2=x01,ag,pg,mi},
    {ag,pg,mi}=OptionValue[{AccuracyGoal,
      PrecisionGoal,MaxIterations}];
    Catch[Do[x1=x2;
      b=DivSin1[a,x1];
      f1=b[[n]];
      If[Abs[f1]<10^-ag,Throw[x1]];
      b=DivSin1[Most[b],x1];
      x2=x1-f1/b[[n-1]];
      If[Abs[x1/x2-1]<10^-pg,Throw[x2]],
        {mi}];
    Message[NewtonP::maxit,mi];x2]]
```



```

...
End[]
...
EndPackage[]

```

Probamos ahora el método importando el paquete "Alge2" y resolviendo la ecuación cúbica $P_3(x) = x^3 + 2x^2 + 3x + 4 = 0$, empleando como valor inicial: $x_1 = -1.6$:

```
Get["Alge2`", Path->"E:\MAT205"]
```

```
NewtonP[{1,2,3,4},-1.6]/InputForm
-1.6506291914416713
```

Resultado que puede ser corroborado resolviendo la ecuación cúbica con "NSolve":

```
NSolve[x^3+2*x^2+3*x+4,x]/InputForm
{{x -> -1.6506291914393882},
 {x -> -0.1746854042803059 - 1.5468688872313963*I},
 {x -> -0.1746854042803059 + 1.5468688872313963*I}}
```

Como se puede observar el resultado concuerda en los primeros 11 dígitos. Por supuesto es posible mejorar la precisión del resultado:

```
NewtonP[{1,2,3,4},-1.6,PrecisionGoal->12,AccuracyGoal->12]/InputForm
-1.6506291914393882
```

El resultado puede ser corroborado también con "NewtonDA":

```
f[x_]:=x^3+2*x^2+3*x+4
df[x_]:=Evaluate[f'[x]]
```

```
NewtonDA[f,df,-1.6]//InputForm
-1.6506291914416713
```

Como se puede observar en este caso (al tratarse en realidad del mismo método) los resultados concuerdan en todos los dígitos.

15.4. Ejemplos

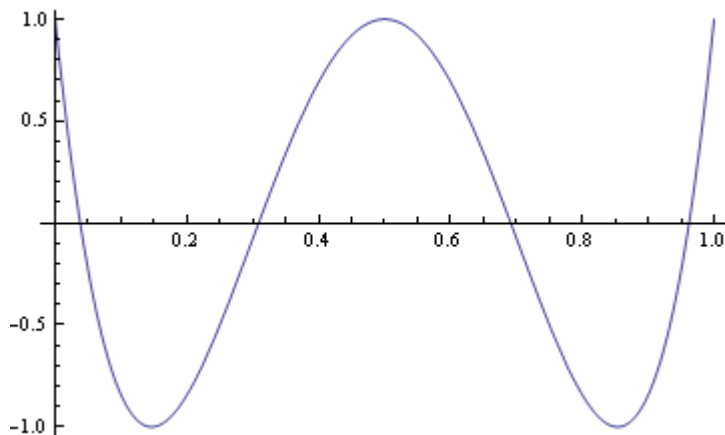
6. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para obtener los valores iniciales necesarios grafique primero el polinomio entre 0 y 1. Encuentre todas las soluciones con una sola instrucción.

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Primero programamos y graficamos el polinomio:

```
f[x_] := 128*x^4 - 256*x^3 + 160*x^2 - 32*x + 1
```

```
Plot[f[x], {x, 0, 1}]
```



Como se puede observar, las cuatro soluciones son reales y las mismas están alrededor de 0.1, 0.3, 0.7 y 0.9, por lo que podemos emplear dichos valores como valores iniciales para el método de Newton.

Para encontrar las cuatro soluciones con una sola instrucción guardamos los valores iniciales en una variable "x" y dentro de un ciclo "Do" llamamos cuatro veces a "NewtonP", empleando en cada llamada uno de los valores almacenados en "x":

```
Block[{x={0.1,0.3,0.7,0.9},a=Reverse[CoefficientList[f[x],x]]},
  Do[Print["x",i,"=",NewtonP[a,x[[i]]]],{i,Length[x]}]
x1=0.0380602
x2=0.308658
x3=0.691342
x4=0.96194
```

Podemos corroborar estas soluciones con "NSolve":

```
NSolve[f[x],x]
{{x->0.0380602},{x->0.308658},{x->0.691342},{x->0.96194}}
```

Y como se puede observar los resultados concuerdan en los primeros 6 dígitos.

7. Programe y encuentre todas las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales

grafique el polinomio entre 3.5 y 11.5 y encuentre todas las soluciones con una sola instrucción.

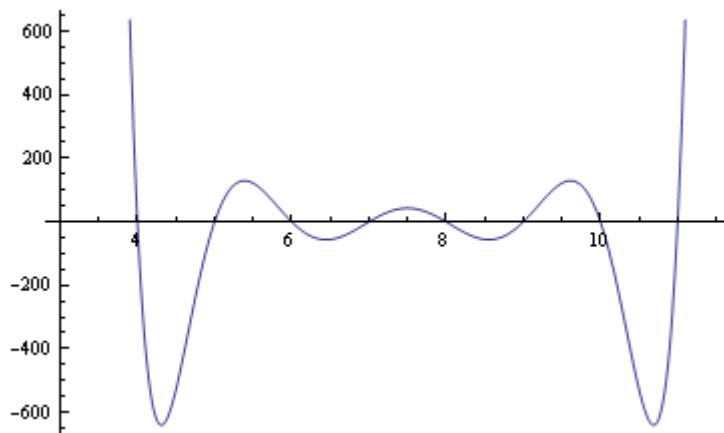
$$P_8(x) = x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x + 6652800 = 0$$

Para ello primero programamos y graficamos el polinomio:

```
f[x_]:=x^8-60*x^7+1554*x^6-22680*x^5+203889*x^4-1155420*x^3+
4028156*x^2-7893840*x+6652800
```

```
Plot[f[x],{x,3.5,11.5},AxesOrigin->{3,0}]
```

Como se puede observar en la gráfica resultante, que se muestra en la siguiente página, en este caso las soluciones parecen pasar exactamente por los puntos 4, 5, 6, 7, 8, 9, 10 y 11, por lo que podríamos emplear valores iniciales cercanos a estos, como ser: 3.8, 4.8, 5.8, 6.8, 7.8, 8.8, 9.8 y 10.8.



Al igual que el caso anterior resolveremos el problema en una sola instrucción:

```
Module[{x={3.8,4.8,5.8,6.8,7.8,8.8,9.8,10.8},
a=Reverse[CoefficientList[f[x],x]]},
Do[Print["x",i,"=",NewtonP[a,x[[i]]]],{i,Length[x]}]]
x1=4.
x2=5.
x3=6.
x4=7.
x5=8.
x6=9.
x7=10.
x8=11.
```

Una vez más podemos corroborar estos resultados con "NSolve":

```
NSolve[f[x],x]
{{x->4.},{x->5.},{x->6.},{x->7.},{x->8.},{x->9.},{x->10.},{x->11.}}
```

8. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales programe y grafique el polinomio entre 0 y 6. Encuentre todas las soluciones con una sola instrucción.

$$P_5(x) = x^5 - 11x^4 + 43x^3 - 79x^2 + 76x - 30 = 0$$

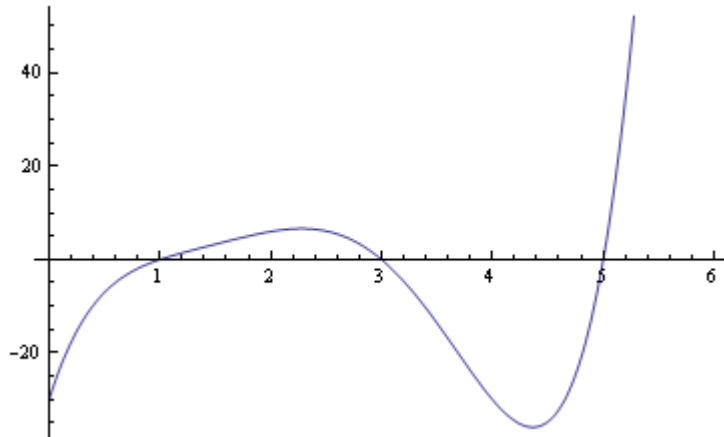
Al igual que en los dos casos anteriores primero programamos y graficamos el polinomio (la gráfica que muestra en la siguiente página):

```
f[x_]:=x^5-11*x^4+43*x^3-79*x^2+76*x-30
```

```
Plot[f[x],{x,0,6}]
```

Como se puede ver en las soluciones son prácticamente 1, 3 y 5, por lo que podemos emplear valores iniciales iguales a 0.9, 2.9 y 4.9.

```
With[{x={0.9,2.9,4.9},
      a=Reverse[CoefficientList[f[x],x]]},
      Do[Print["x",i,"=",NewtonP[a,x[[i]]]],{i,3}]]
x1=1.
x2=3.
x3=5.
```



Soluciones que una vez más podemos corroborar con "NSolve":

```
NSolve[f[x],x]
{{x->1.},{x->1.-1.i},{x->1.+1.i},{x->3.},{x->5.}}
```

Sin embargo, puesto que en este caso sólo nos interesan las soluciones reales, en lugar de "NSolve" podríamos emplear "Reduce", que es similar a "NSolve", pero que además permite fijar el dominio (en este caso reales) (Adicionalmente "Reduce" también permite trabajar con desigualdades).

```
Reduce[f[x]==0,x,Reals]
x==1||x==3||x==5
```

También podríamos emplear "FindInstance" que es similar a "Reduce", pero que además permite fijar el número de instancias (soluciones) a encontrar:

```
FindInstance[f[x]==0,x,Reals,3]
{{x->5},{x->3},{x->1}}
```

9. En el diseño de un equipo se requiere calcula la única solución real de la siguiente ecuación de séptimo grado. Elabore un programa que encuentre dicha solución por el método de Newton, empleando un valor inicial igual a 0.1 y que corrobore dicho resultado con NSolve, Reduce y FindInstance.

$$\begin{aligned}
 a_1 x^7 + a_2 x^6 + a_3 x^5 + a_4 x^4 + a_5 x^3 + a_6 x^2 + a_7 x + a_8 &= 0 \\
 a_1 &= 2.3 * w^{4.3} + 5v \\
 a_2 &= u^2 + v^2 + w^2 \\
 a_3 &= e^{\frac{u+v}{w}} \\
 a_4 &= \ln(u+v) - \ln(w) \\
 a_5 &= v^{2.1} + w^{4.2} \\
 a_6 &= \left[\frac{u+v+w}{4.65} \right]^{1.3233} \\
 a_7 &= u^v + w^v \\
 a_8 &= u+v+w \\
 u &= 1.32344 \\
 v &= 0.76833 \\
 w &= 1.78372
 \end{aligned}$$

El programa elaborado para resolver el problema es el siguiente:

```

Module[{w=1.78372,v=0.76833,u=1.32344,
  a1,a2,a3,a4,a5,a6,a7,a8,a,p},
  a1=2.3*w^4.3+5*v;
  a2=u^2+v^2+w^2;
  a3=Exp[(u+v)/w];
  a4=Log[u+v]-Log[w];
  a5=v^2.1+w^4.2;
  a6=((u+v+w)/4.65)^1.3233;
  a7=u^v+w^v;
  a8=u+v+w;
  p[x_]:=a1*x^7+a2*x^6+a3*x^5+a4*x^4+a5*x^3+a6*x^2+a7*x+a8;
  a=Reverse[CoefficientList[p[x],x]];
  Print["      Newton: x = ",NewtonP[a,0.1]];
  Print["      NSolve: x = ",Cases[Flatten[x/.NSolve[p[x],x]],_Real][[1]]];
  Print["      Reduce: x = ",Reduce[p[x]==0,x,Reals][[2]]];
  Print["FindInstance: x = ",x/.FindInstance[p[x]==0,x,Reals,1][[1,1]]]
]

```

Los resultados obtenidos con el programa son:

```

      Newton: x = -0.555893
      NSolve: x = -0.555893
      Reduce: x = -0.555893
FindInstance: x = -0.555893

```

Y como se puede observar todos los métodos devuelven el mismo resultado en los primeros 6 dígitos.

15.5. Ejercicios

10. Elabore una función para encontrar una de las soluciones de una ecuación polinomial por el método de Newton Raphson, empleando la estructura While.
11. Elabore una función para encontrar una de las soluciones de una ecuación polinomial por el método de Newton Raphson, empleando la estructura For.
12. Elabore una función para encontrar una de las soluciones de una ecuación polinomial por el método de Newton Raphson, empleando el comando Goto.

13. Elabore una función para encontrar una de las soluciones de una ecuación polinomial por el método de Newton Raphson, empleando la estructura Nest.
14. Elabore una función para encontrar una de las soluciones de una ecuación polinomial por el método de Newton Raphson, empleando la recursividad.
10. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales programe y grafique el polinomio. Encuentre todas las soluciones con una sola instrucción.

$$P_3(x) = 5x^3 - 250x^2 + 4000x - 20000 = 0$$

11. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales programe y grafique el polinomio. Encuentre todas las soluciones con una sola instrucción.

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

12. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales programe y grafique el polinomio. Encuentre todas las soluciones con una sola instrucción.

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

13. Encuentre las soluciones reales del polinomio que se presenta al final de la pregunta. Para estimar los valores iniciales programe y grafique el polinomio. Encuentre todas las soluciones con una sola instrucción.

$$P_8(x) = x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

16. BAIRSTOW

16.1. División sintética de un polinomio entre $x^2-rx-s=0$

Al igual que la división sintética entre un monomio es útil para encontrar una de las raíces o soluciones de una ecuación polinomial, la división entre un binomio (en nuestro caso x^2-rx-s) es útil para encontrar dos de las raíces o soluciones de una ecuación polinomial por el método de *Bairstow*. Una segunda división (del residuo de la división anterior) permite calcular las derivadas parciales de los coeficientes con respecto a los coeficientes del binomio, es decir con relación a "r" y "s".

Por ejemplo para dividir el polinomio:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

Entre x^2+x+1 , efectuamos las siguientes operaciones:

$$\begin{array}{r|rrrrr}
 & 1 & -1.1 & 2.3 & 0.5 & 3.3 \\
 r=-1 & & -1.0 & 2.1 & -3.4 & 0.8 \\
 s=-1 & & & -1.0 & 2.1 & -3.4 \\
 \hline
 & 1 & -2.1 & 3.4 & -0.8 & 0.7 \\
 & & -1.0 & 3.1 & -5.5 & \\
 & & & -1.0 & 3.1 & \\
 \hline
 & 1 & -3.1 & 5.5 & -3.2 &
 \end{array}$$

Si denominamos "b" a los coeficientes resultantes de la primera división, el residuo de la división es: $b_n(x-r)+b_{n+1}$, que con los resultados del ejemplo da: $-0.8*(x+1)+0.7 = -0.8*x-0.1$.

Si por otra parte denominamos "c" a los coeficientes resultantes de la segunda división, entonces las derivadas parciales de los coeficientes "b" con relación a los coeficientes "r" y "s" son: $\partial b_i / \partial r = c_{i-1}$ y $\partial b_i / \partial s = c_{i-2}$.

La ecuación que nos permite calcular los coeficientes de la división sintética, la cual puede ser deducida del procedimiento mostrado en el ejemplo, es la siguiente:

$$b_i = a_i + b_{i-1}r + b_{i-2}s \quad \begin{cases} i=3 \rightarrow n+1 \\ b_1 = a_1 \\ b_2 = a_2 + b_1r \end{cases} \quad \text{residuo} = b_n(x-r) + b_{n+1} \quad (1)$$

Donde "n" es el grado del polinomio.

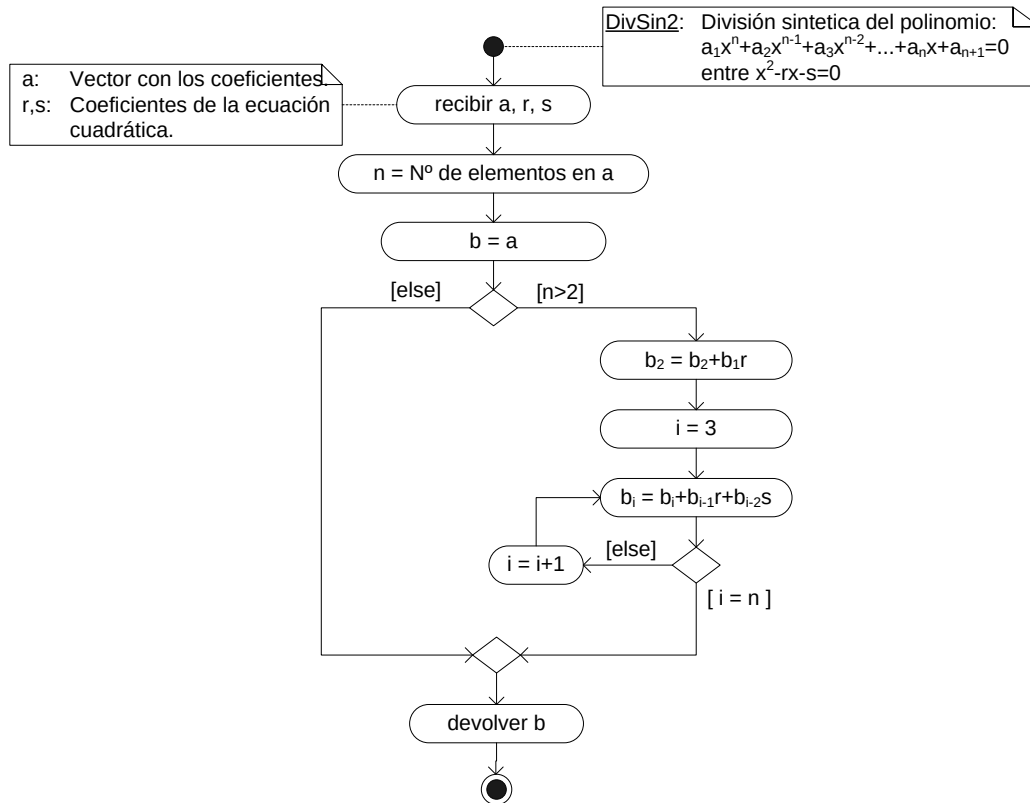
16.1.1. Programa

El algoritmo que resuelve la ecuación (3) se presenta en la siguiente página. El código elaborado en base a dicho algoritmo y añadido a la unidad "Alge2`" es el siguiente:

```
BeginPackage["Alge2`"]
```

```
...
```

```
DivSin2::usage="DivSin2[a,r,s], Devuelve los coeficientes
resultantes de la división sintética del polinomio:
a[[1]]*x^n+a[[2]]*x^(n-1)+a[[3]]*x^(n-2)+...+a[[n+1]]=0 entre
x^2-rx-s."
```



```

...
Begin["Private`"]
...
DivSin2[a_List,r_, s_] := Module[{n=Length[a],b=a},
  If[n>2,
    b[[2]] = b[[2]] + b[[1]]*r;
    Do[b[[i]] = b[[i]] + b[[i-1]]*r + b[[i-2]]*s, {i, 3, n}];
    b]
...
End[]

EndPackage[]

```

Podemos probar el programa con el polinomio resuelto en el ejemplo manual, para ello primero importamos el paquete y programamos el polinomio:

```
Get["Alge2`", Path->"E:\MAT205"]
```

```
p[x_] := x^4 - 1.1*x^3 + 2.3*x^2 + 0.5*x + 3.3
```

Extraemos los coeficientes tanto del polinomio como del binomio:

```
a = Reverse[CoefficientList[p[x], x]]
{1, -1.1, 2.3, 0.5, 3.3}
```

```
{r, s} = -Reverse[CoefficientList[x + 1, x]]
{-1, -1}
```

Y llevamos a cabo la división sintética:

```
b = DivSin2[a, r, s]
{1, -2.1, 3.4, -0.8, 0.7}
```

Como se puede ver los resultados concuerdan con los del ejemplo manual, por lo tanto el cociente de la división es:

$$x^2 - 2.1x + 3$$

Y el residuo:

```
Expand[b[[4]]*(x-r)+b[[5]]]
-0.1-0.8 x
```

Resultados que podemos corroborar también con la función "PolynomialQuotientRemainder" (o PolynomialQuotient y PolynomialRemainder):

```
o=PolynomialQuotientRemainder[p[x],x^2-r*x-s,x]
{3.4-2.1 x+x^2,-0.1-0.8 x}
```

Que como se puede observar devuelve los mismos resultados (aunque como de costumbre Mathematica ordena el polinomio de la menor a la mayor potencia). Dividiendo ahora el resultado de la primera división obtenemos:

```
c=DivSin2[Most[b],-1,-1]
{1,-3.1,5.5,-3.2}
```

Que una vez más concuerda con el resultado del ejemplo manual, siendo el cociente: "x-3.1" y el residuo: "Expand[c[[3]]*(x-r)+c[[4]]] => 2.3+5.5x". Resultado que puede ser corroborado con "PolynomialQuotientRemainder":

```
q[x_]:=Expand[o[[1]]*x+o[[2,2]]/x];q[x]
-0.8+3.4 x-2.1 x^2+x^3
```

```
PolynomialQuotientRemainder[q[x],x^2-r*x-s,x]
{-3.1+x,2.3+5.5 x}
```

16.1.2. Ejemplos

1. Divida el polinomio que se presenta al final de la pregunta entre x^2+x+1 , luego divida el polinomio resultante entre x^2+x+1 . Corrobore el resultado con "PolynomialQuotient" y "PolynomialRemainder":

$$P_5(x) = x^5 - 3x^4 - 10x^3 + 10x^2 + 44x + 48 = 0$$

Programamos el polinomio y extraemos los coeficientes:

```
p[x_]:=x^5-3*x^4-10*x^3+10*x^2+44*x+48
```

```
a=Reverse[CoefficientList[p[x],x]]
{1,-3,-10,10,44,48}
```

```
r=-1;s=-1;
```

Llevamos a cabo la primera división:

```
b=DivSin2[a,r,s]
{1,-4,-7,21,30,-3}
```

Por lo tanto el cociente es: $x^3 - 4x^2 - 7x + 21$ y el residuo:

```
Expand[b[[Length[b]-1]]*(x-r)+b[[Length[b]]]]
27+30 x
```

Ahora llevamos a cabo la segunda división:

```
c=DivSin2[Drop[b,-1],r,s]
{1,-5,-3,29,4}
```

Como vemos el cociente es: $x^2 - 5x - 3$ y el residuo:

```
Expand[c[[Length[c]-1]]*(x-r)+c[[Length[c]]]]
33+29 x
```

Ahora corroboramos estos resultados con "PolynomialQuotient" y "PolynomialRemainder". Para la primera división resulta:

```
q1=PolynomialQuotient[p[x],x^2+x+1,x]
21-7 x-4 x^2+x^3
```

```
q2=PolynomialRemainder[p[x],x^2+x+1,x]
27+30 x
```

Resultados que concuerdan con los obtenidos con "DivSin2". Para la segunda división resulta:

```
q[x_]:=Expand[q1*x+q2[[2]]/x];q[x]
30+21 x-7 x^2-4 x^3+x^4
```

```
PolynomialQuotient[q[x],x^2+x+1,x]
-3-5 x+x^2
```

```
PolynomialRemainder[q[x],x^2+x+1,x]
33+29 x
```

Resultados que una vez más concuerdan con los obtenidos con "DivSin2".

2. Divida el polinomio que se presenta al final de la pregunta entre x^2+3x-4 , luego divida el polinomio resultante entre x^2+3x-4 . Corrobo-re el resultado con "PolynomialQuotientRemainder":

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

Importamos el paquete "Alge2`" (si aún no lo hemos hecho), programamos el polinomio y extraemos los coeficientes:

```
Get["Alge2`",Path->"E:\MAT205"]
```

```
p[x_]:=x^6+x^5-8*x^4+213*x^3+21*x^2-265*x+150
```

```
a=Reverse[CoefficientList[p[x],x]]
{1,1,-8,213,21,-265,150}
```

```
r=-3;s=4;
```

Ahora llevamos a cabo la primera división sintética:

```
b=DivSin2[a,r,s]
{1,-2,2,199,-568,2235,-8827}
```

Siendo el residuo

```
Expand[b[[Length[b]-1]]*(x-r)+b[[Length[b]]]]
-2122+2235 x
```

Con los coeficientes calculados ("b") realizamos la segunda división:

```
c=DivSin2[Take[b,Length[b]-1],r,s]
{1,-5,21,116,-832,5195}
```

Siendo el residuo:

```
Expand[c[[Length[c]-1]]*(x-r)+c[[Length[c]]]]
2699-832 x
```

Corroboramos los resultados con "PolynomialQuotientRemainder":

```
{q1,q2}=PolynomialQuotientRemainder[p[x],x^2+3x-4,x]
```



```
{-568+199 x+2 x^2-2 x^3+x^4,-2122+2235 x}
```

Y como vemos los resultados de la primera división concuerdan. Ahora corroboramos los resultados de la segunda división:

```
q[x_]:=Expand[q1*x+q2[[2]]/x];q[x]
2235-568 x+199 x^2+2 x^3-2 x^4+x^5
```

```
PolynomialQuotientRemainder[q[x],x^2+3x-4,x]
{116+21 x-5 x^2+x^3,2699-832 x}
```

Y una vez más vemos que los resultados concuerdan.

16.1.3. Ejercicios

1. Elabore un módulo que, empleando la estructura For, lleve a cabo la división sintética de un polinomio entre x^2-rx-s .
2. Elabore un módulo que, empleando la estructura While, lleve a cabo la división sintética de un polinomio entre x^2-rx-x .
3. Elabore un módulo que, empleando la estructura Nest, lleve a cabo la división sintética de un polinomio entre x^2-rx-x .
4. Elabore un módulo que, empleando el comando Goto, lleve a cabo la división sintética de un polinomio entre x^2-rx-x .
5. Elabore un módulo que, empleando la recursividad, lleve a cabo la división sintética de un polinomio entre x^2-rx-x .
6. Divida el polinomio que se presenta al final de la pregunta entre x^2+x+1 , luego divida el polinomio resultante entre x^2-5x+2 . Corrobore el resultado con "PolynomialQuotient" y "PolynomialRemainder":

$$P_8(x) = x^8 + x^7 - 9x^6 + 13x^5 + 21x^4 - 125x^3 + 77x^2 + 111x - 90 = 0$$

7. Divida el polinomio que se presenta al final de la pregunta entre x^2+x+1 , luego divida el polinomio resultante entre x^2+2x-3 . Corrobore el resultado con "PolynomialQuotientRemainder":

$$P_{10}(x) = 12x^{10} - 8x^9 + 6x^7 - 3x^3 + 5x^2 - 3x - 10 = 0$$

16.2. Raíces de una ecuación cuadrática

Cuando se tiene una ecuación polinomial de segundo grado, la forma más eficiente de resolverla es aplicando la solución analítica, ampliamente conocida, así si la forma de la ecuación es:

$$ax^2 + bx + c = 0$$

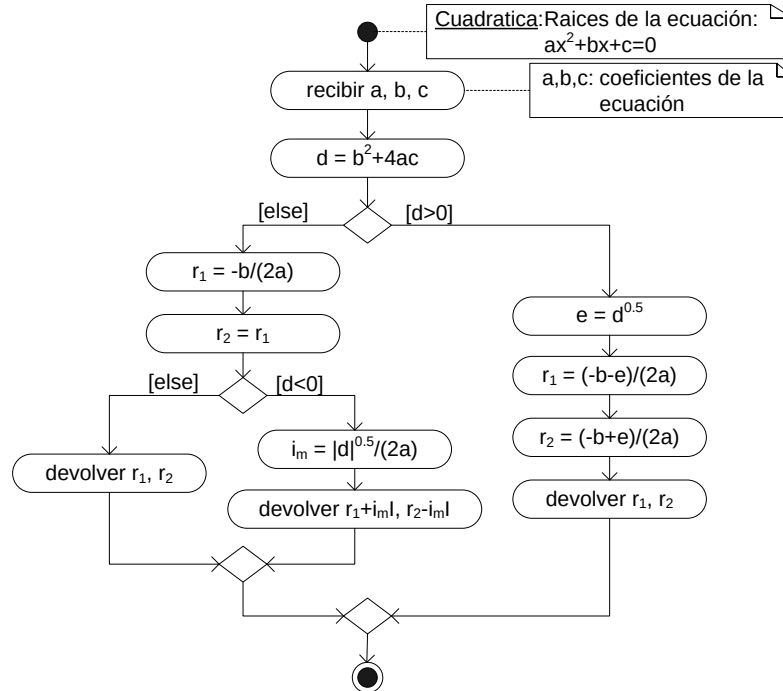
Las dos soluciones se calculan con:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \quad (2)$$

Donde, dependiendo del valor de " b^2-4ac ", denominado discriminante, se presentan tres casos: a) Si es positivo las dos soluciones son reales y distintas, b) Si es cero, las dos soluciones son reales e iguales y c) Si es negativo, las dos soluciones son complejas (un par conjugado). Aun cuando en Mathematica podemos resolver la ecuación de segundo grado con "NSolve" o "Solve" y a pesar de haber resuelto ya el problema en el tema 5, volveremos a elaborar un programa para este fin, el mismo que emplearemos luego conjuntamente el método de Bairstow.

16.2.1. Programa

El algoritmo para resolver la ecuación de segundo orden es el siguiente:



El código elaborado en base a este algoritmo y añadido al paquete "Alge2`" es:

```

BeginPackage["Alge2`"]
...
Cuadratica::usage="Cuadratica[a,b,c], encuentra las soluciones
de la ecuación cuadrática ax^2+bx+c."
...
Begin["Private`"]
...
Cuadratica[a_,b_,c_] := Module[{d=b^2-4*a*c,e,r1,r2,im},
  If[d>0,e=Sqrt[d];r1=(-b-e)/(2*a);
    r2=(-b+e)/(2*a);{r1,r2},
  r1=-b/(2*a); r2=r1;
  If[d<0,im=Sqrt[Abs[d]]/(2*a);{r1+im I,r2-im I},
    {r1,r2}]]]
...
End[]
...
EndPackage[]
  
```

Podemos probar el programa resolviendo la ecuación: $x^2+2x+3=0$:

```
Get["Alge2`",Path->"E:\MAT205"]
```

```

Cuadratica[1,2,3]
{-1 - 1.41421 i, -1 + 1.41421 i}
  
```

Resultado que podemos corroborar con "Solve":

```

Solve[x^2+2x+3==0,x]
{{x -> -1 - 1.41421 i}, {x -> -1 + 1.41421 i}}
  
```

O también con "NSolve":

```
NSolve[x^2+2x+3==0,x]
{{x -> -1. - 1.41421 i}, {x -> -1. + 1.41421 i}}
```

Observe que con "NSolve" se obtienen siempre soluciones aproximadas (soluciones reales). También podemos corroborar el resultado con "Reduce":

```
Reduce[x^2+2*x+3==0,x]
x == -1 - i sqrt(2) || x == -1 + i sqrt(2)
```

O con "FindInstance":

```
FindInstance[x^2+2*x+3==0,x,2]
{{x -> -1 - i sqrt(2)}, {x -> -1 + i sqrt(2)}}
```

Resolvamos ahora la ecuación (soluciones iguales): $x^2-14x+49$:

```
Cuadratica[1,-14,49]
{7,7}
```

Resultados que podemos corroborar con "Solve", "NSolve", "Reduce" y "FindInstance":

```
Solve[x^2-14*x+49==0,x]
{{x->7},{x->7}}
```

```
NSolve[x^2-14x+49,x]
{{x->7.},{x->7.}}
```

```
Reduce[x^2-14x+49==0,x]
x==7
```

```
FindInstance[x^2-14x+49==0,x,2]
{{x->7}}
```

Finalmente resolvamos la ecuación (soluciones diferentes): $x^2-8x+15$:

```
Cuadratica[1,-8,15]
{3,5}
```

Resultados que pueden ser corroborados con "Solve", "NSolve", "Reduce" y "FindInstance":

```
Solve[x^2-8*x+15==0,x]
{{x->3},{x->5}}
```

```
NSolve[x^2-8*x+15,x]
{{x->3.},{x->5.}}
```

```
Reduce[x^2-8*x+15==0,x]
x==3 || x==5
```

```
FindInstance[x^2-8*x+15==0,x,2]
{{x->5},{x->3}}
```

16.2.2. Ejercicios

8. Elabore una función que, empleando la estructura Which, resuelva la ecuación cuadrática.
9. Elabore una función que, empleando la estructura Switch, resuelva la ecuación cuadrática.

10. Elabore una función que, empleando la estructura Piecewise, resuelva la ecuación cuadrática.
11. Elabore una función que, empleando el comando Condition, resuelva la ecuación cuadrática.
12. Con la función "Cuadratica" resuelva la ecuación: $x^2-8x+41=0$. Corrobore los resultados con "Solve", "NSolve", "Reduce" y "FindInstance".
13. Con la función "Cuadratica" resuelva la ecuación: $x^2-19x+81=0$. Corrobore los resultados con "Solve", "NSolve", "Reduce" y "FindInstance".
14. Con la función "Cuadratica" resuelva la ecuación: $x^2-13x+22=0$. Corrobore los resultados con "Solve", "NSolve", "Reduce" y "FindInstance".

16.3. Método de Bairstow

El método de Bairstow permite calcular 2 de las raíces de una ecuación polinomial. Para ello se divide el polinomio entre una ecuación de segundo grado y se van cambiando los coeficientes de esta ecuación hasta que el residuo de la división se hace cero, entonces se calculan dos raíces del polinomio con la ecuación de segundo grado que hace cero el residuo.

El procedimiento puede ser repetido con el polinomio residual (cuyo grado es "n-2") encontrándose así otras dos soluciones y continuar de esa manera hasta encontrar todas las soluciones del polinomio. No obstante y al igual que ocurre con el método de Newton - Raphson, este procedimiento tiene el inconveniente de ir propagando el error de los cálculos previos, por esta razón, generalmente se emplea el polinomio original, y se van encontrando las raíces (por pares) empleando valores iniciales cercanos a las soluciones buscadas. Los valores iniciales normalmente se encuentran con otros métodos, como el método QD.

Otra de las desventajas de este método radica en que no siempre converge hacia las soluciones, ello depende sobre todo de los valores iniciales proporcionados.

El fundamento del método es el siguiente: al dividir el polinomio general, entre la ecuación de segundo grado: x^2-rx-s , resulta:

$$\frac{P_n(x)}{x^2-r-s} = Q_{n-2}(x)(x^2-rx-s) + \text{resíduo} = 0 \quad (3)$$

Entonces, el método de Bairstow busca los valores de "r" y "s" que hacen el residuo cero (o casi cero). Cuando esto sucede la ecuación (3) queda igualada a cero y se cumple que:

$$\begin{aligned} Q_{n-2}(x) &= 0 \\ x^2 - rx - s &= 0 \end{aligned} \quad (4)$$

Entonces dos de las raíces (soluciones) pueden ser calculadas resolviendo la ecuación de segundo grado. Posteriormente es posible encontrar otras dos raíces repitiendo el procedimiento con el polinomio Q_{n-2} y continuar de esa manera hasta determinar todas las raíces del polinomio original. Aunque como ya se dijo este procedimiento tiene el inconveniente de ir propagando el error cometido en los cálculos previos.

La división del polinomio $P_n(x)$ entre la ecuación de segundo grado es simplemente la división sintética, la misma que ya ha sido programada con el nombre "DivSin2". Como ya se dijo, el objetivo del método de Bairstow es hacer el residuo $(b_n(x-r)+b_{n+1})$ cero. Para ello, se expanden en series de Taylor los coeficientes b_n y b_{n+1} en función de "r" y "s":

$$b_n(r + \Delta r, s + \Delta s) = b_n(r, s) + \frac{\partial b_n(r, s)}{\partial r} \Delta r + \frac{\partial b_n(r, s)}{\partial s} \Delta s + \frac{1}{2!} \left(\frac{\partial b_n(r, s)}{\partial r} \Delta r + \frac{\partial b_n(r, s)}{\partial s} \Delta s \right)^2 + \dots \infty = 0$$

$$b_{n+1}(r + \Delta r, s + \Delta s) = b_{n+1}(r, s) + \frac{\partial b_{n+1}(r, s)}{\partial r} \Delta r + \frac{\partial b_{n+1}(r, s)}{\partial s} \Delta s + \frac{1}{2!} \left(\frac{\partial b_{n+1}(r, s)}{\partial r} \Delta r + \frac{\partial b_{n+1}(r, s)}{\partial s} \Delta s \right)^2 + \dots \infty = 0$$

Donde " Δr " y " Δs " son los valores que deben sumarse a " r " y " s " para que tanto " b_n " como " b_{n+1} " se igualen a cero. Por supuesto, no es posible en la práctica resolver este sistema (que es más complicado aún que el problema original), por eso sólo se toman los términos de primer orden con lo que el sistema se simplifica a:

$$\begin{aligned} b_n + \frac{\partial b_n}{\partial r} \Delta r + \frac{\partial b_n}{\partial s} \Delta s &= 0 \\ b_{n+1} + \frac{\partial b_{n+1}}{\partial r} \Delta r + \frac{\partial b_{n+1}}{\partial s} \Delta s &= 0 \end{aligned} \quad (5)$$

En estas ecuaciones las funciones siguen siendo evaluadas en (r, s) . Observe que si se pueden calcular las derivadas, entonces la ecuación (7) es simplemente un sistema de dos ecuaciones lineales con 2 incógnitas.

Las derivadas parciales pueden ser calculadas realizando una segunda división sintética del polinomio residual " Q_{n-2} " entre la ecuación cuadrática $x^2 - rx - s$. Si denominamos " c " a los coeficientes resultantes de esta segunda división, entonces las derivadas parciales son:

$$\begin{aligned} \frac{\partial b_i}{\partial r} &= c_{i-1} \\ \frac{\partial b_i}{\partial s} &= c_{i-2} \end{aligned} \quad (6)$$

Empleando estas relaciones la ecuación (16) queda en la forma:

$$\begin{aligned} c_{n-1} * \Delta r + c_{n-2} * \Delta s &= -b_n \\ c_n * \Delta r + c_{n-1} * \Delta s &= -b_{n+1} \end{aligned} \quad (7)$$

Que puede ser resuelta con la regla de Cramer:

$$\begin{aligned} \Delta r &= \frac{\begin{vmatrix} -b_n & c_{n-2} \\ -b_{n+1} & c_{n-1} \end{vmatrix}}{\begin{vmatrix} c_{n-1} & c_{n-2} \\ c_n & c_{n-1} \end{vmatrix}} = \frac{-b_n * c_{n-1} + b_{n+1} * c_{n-2}}{c_{n-1}^2 - c_n * c_{n-2}} \\ \Delta s &= \frac{-b_n - c_{n-1} * \Delta r}{c_{n-2}} \end{aligned} \quad (8)$$

Sin embargo, puesto que estos valores han sido calculados despreciando los términos de segundo orden y superiores, son solamente valores aproximados, que al ser sumados a " r " y " s " no igualan " b_n " y " b_{n+1} " a cero, aunque si los aproximan a este valor.

Por lo tanto, el cálculo de los valores de " r " y " s " que hacen " b_n " y " b_{n+1} " cero, es iterativo: comenzando con valores iniciales asumidos se calcula por división sintética, los coeficientes " b " y se verifica si " b_n " y " b_{n+1} " son cero (o casi cero), de ser así el proceso termina habiéndose encontrado los valores de " r " y " s " que hacen cero el residuo, caso contrario se calculan los coeficientes " c ", mediante otra división sintética, y con los mismos los valores de " Δr " y " Δs ". Si estos valores son cercanos a cero

el proceso concluye, caso contrario se calculan nuevos valores de "r" y "s": $r=r+\Delta r$, $s=s+\Delta s$ y se repite el proceso.

Para comprender mejor el método de Bairstow, resolvamos la siguiente ecuación de cuarto grado:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

Nuestros valores asumidos serán $r=-1$ y $s=-1$. Para no volver a escribir una y otra vez la misma expresión, programamos primero las ecuaciones (10):

```
Get["Alge2`",Path->"E:\MAT205 "]
Deltar[n_]:=(-b[[n]]*c[[n-1]]+b[[n+1]]*c[[n-2]])/
(c[[n-1]]^2-c[[n]]*c[[n-2]])
Deltas[n_]:=(-b[[n]]-c[[n-1]]*dr)/c[[n-2]]
```

Programamos el polinomio, extraemos los coeficientes y asignamos los valores asumidos a las variables "r" y "s":

```
p[x_]:=x^4-1.1*x^3+2.3*x^2+0.5*x+3.3
```

```
a=Reverse[CoefficientList[p[x],x]]
{1,-1.1,2.3,0.5,3.3}
```

```
r=-1;s=-1;
```

```
n=Length[a]-1
4
```

Llevamos a cabo la primera división sintética:

```
b=DivSin2[a,r,s]
{1,-2.1,3.4,-0.8,0.7}
```

Con el residuo llevamos a cabo la segunda división sintética:

```
c=DivSin2[Most[b],r,s]
{1,-3.1,5.5,-3.2}
```

Calculamos los valores de Δr , Δs :

```
dr=Deltar[n]
0.10969
```

```
ds=Deltas[n]
-0.063453
```

Y con ellos calculamos los nuevos valores de "r" y "s":

```
r=r+dr
-0.89031
```

```
s=s+ds
-1.06345
```

Puesto que estos valores son diferentes a los asumidos (-1 y -1), volvemos a repetir el proceso empleando los nuevos valores de "r" y "s":

```
b=DivSin2[a,r,s]
{1,-1.99031,3.00854,-0.0619314,0.155698}
In[44]:= c=DivSin2[Most[b],r,s]
Out[44]= {1,-2.88062,4.50973,-1.01359}
In[45]:= dr=Deltar[n]
Out[45]= -0.00971481
In[46]:= ds=Deltas[n]
```

```

Out[46]= -0.0367083
In[47]:= r=r+dr
Out[47]= -0.900025
In[48]:= s=s+ds
Out[48]= -1.10016

```

Estos nuevos valores de "r" y "s" siguen siendo todavía algo diferentes a los del anterior cálculo, por lo que repetimos el proceso una vez más:

```

b=DivSin2[a,r,s]
{1,-2.00002,2.99991,0.00035642,-0.000706041}

```

```

c=DivSin2[Most[b],r,s]
{1,-2.90005,4.50987,-0.868111}

```

```

dr=Deltar[n]
0.0000246977

```

```

ds=Deltas[n]
0.000161309

```

```

r=r+dr
-0.9

```

```

s=s+ds
-1.1

```

Estos nuevos valores de "r" y "s" son ya bastante parecidos a los del último cálculo, pero para corroborar estos resultados repetimos el proceso una vez más:

```

b=DivSin2[a,r,s]
{1,-2.,3.,5.64998*10^-9,-9.18789*10^-11}

```

```

c=DivSin2[Most[b],r,s]
{1,-2.9,4.51,-0.869}

```

```

dr=Deltar[n]
-1.41498*10^-9

```

```

ds=Deltas[n]
-2.5227*10^-10

```

```

r=r+dr
-0.9

```

```

s=s+ds
-1.1

```

Estos nuevos valores son los mismos que los del último cálculo (y los valores de Δr y Δs son prácticamente cero), por lo que el proceso concluye y podemos calcular dos de las raíces de la ecuación polinomial:

```

Cuadratica[1,-r,-s]
{-0.45-0.947365 i,-0.45+0.947365 i}

```

Las otras dos raíces pueden ser calculadas del polinomio residual (de segundo grado), obtenido en la división sintética:

```

Cuadratica[Take[b,3]/.List->Sequence]
{1. -1.41421 i,1. +1.41421 i}

```

Resultados que pueden ser corroborados con "NSolve", "Solve", "Reduce" y "FindInstance":

```
NSolve[p[x],x]
{{x→-0.45-0.947365 i},{x→-0.45+0.947365 i},
 {x→1. -1.41421 i},{x→1. +1.41421 i}}

Solve[p[x]==x,x]
{{x→-0.400349-1.07593 i},{x→-0.400349+1.07593 i},
 {x→0.950349 -1.26523 i},{x→0.950349 +1.26523 i}}

Reduce[p[x]==x,x]
x== -0.400349-1.07593 i || x== -0.400349+1.07593 i ||
x== 0.950349 -1.26523 i || x== 0.950349 +1.26523 i

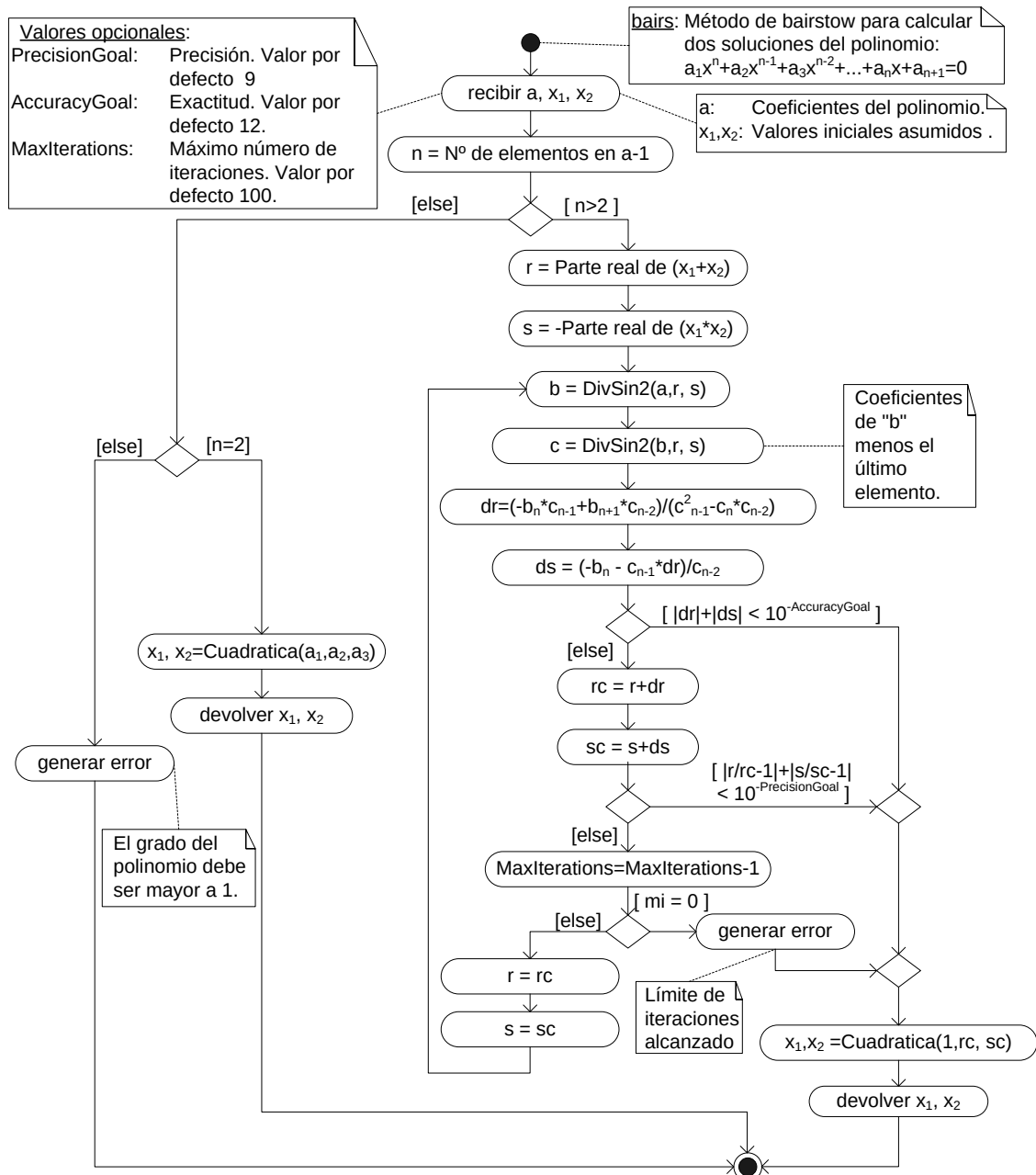
FindInstance[p[x]==x,x,4]
{{x→0.950349 -1.26523 i},{x→0.950349 +1.26523 i},
 {x→-0.400349-1.07593 i},{x→-0.400349+1.07593 i}}
```

16.3.1. Programa

Para programar el método, lo único que se debe hacer es repetir el procedimiento, del anterior acápite, dentro de un ciclo iterativo, hasta que los dos últimos valores de "r" y "s" sean prácticamente iguales o hasta que los incrementos " Δr " y " Δs " sean casi cero.

El algoritmo del método de *Bairstow* se presenta en la siguiente página, y la función elaborada en base al mismo y añadido al paquete "Alge2`" es el siguiente:

```
BeginPackage["Alge2`"]
...
Bairstow::usage = "Bairstow[a,r,s], encuentra dos de
  las soluciones de la ecuación polinomial:
  a[[1]]*x^n+a[[2]]*x^(n-1)+a[[3]]*x^(n-2)+...+a[[n+1]]=0,
  'r' y 's' son los valores iniciales asumidos para
  los coeficientes del divisor: x^2-rx-s=0."
...
Begin["Private`"]
...
Bairstow[a_List,x1_,x2_,OptionsPattern[]]:=
  Block[{r,s,rc,sc,b,c,n,dr,ds,pg,ag,mi,li},
    {pg,ag,mi}=OptionValue[{PrecisionGoal,
      AccuracyGoal,MaxIterations}];
    n=Length[a]-1; li=mi;
    If[n>2,r=Re[x1+x2];s=-Re[x1*x2];
      While[True,
        b=DivSin2[a,r,s];
        c=DivSin2[Most[b],r,s];
        dr=(-b[[n]]*c[[n-1]]+b[[n+1]]*c[[n-2]])/
          (c[[n-1]]^2-c[[n]]*c[[n-2]]);
        ds =(-b[[n]]-c[[n-1]]*dr)/c[[n-2]];
        If[(Abs[dr]+Abs[ds])<10^-ag,Break[]];
        rc=r+dr; sc=s+ds;
```

```

If [ (Abs[r/rc-1]+Abs[s/sc-1]) < 10^-pg, Break[] ];
If [ --mi==0, Message[Bairstow::maxit,li]; Break[] ];
r=rc; s=sc;];
Return[Cuadratica[1,-rc,-sc]]];
If[n==2, Return[Cuadratica[a[[1]],a[[2]],a[[3]]]]];
Message[Bairstow::grado]]
...
End[]
...
EndPackage[]

```

Como se puede observar en el algoritmo, se reciben los valores asumidos para las soluciones o raíces del polinomio, **no los valores de "r" y "s"**, ello debido a que es más fácil obtener valores iniciales para las soluciones que para los coeficientes del divisor. Los valores iniciales de "r" y "s" se

calculan dentro del programa con los valores asumidos para las soluciones, en base a la multiplicación:

```
Expand[(x-x1) (x-x2)]
x2-x x1-x x2+x1 x2
```

```
Collect[%,x]
x2+x (-x1-x2)+x1 x2
```

Comparando con el divisor " x^2-rx-s ", se puede observar que " r " equivale a: " x_1+x_2 " y " s " equivale a: " $-x_1*x_2$ "; que son las operaciones realizadas en el algoritmo para obtener los valores iniciales de " r " y " s ". Pero como además, las soluciones pueden ser imaginarias, en el algoritmo se extrae la parte real de estos resultados, porque, al ser pares conjugados, la parte imaginaria resultante de estas operaciones es siempre cero.

Ahora podemos probar el programa volviendo a resolver el polinomio:

$$P_4(x) = x^4 - 1.1x^3 + 2.3x^2 + 0.5x + 3.3 = 0$$

Empleando valores iniciales cercanos a dos de las soluciones conocidas: $x_1 = -0.4 + 0.9i$; $x_2 = -0.4 - 0.9i$:

```
Get["Alge2`", Path->"E:\MAT205"]
```

```
p[x_] := x^4 - 1.1*x^3 + 2.3*x^2 + 0.5*x + 3.3
```

```
a = Reverse[CoefficientList[p[x], x]]
{1, -1.1, 2.3, 0.5, 3.3}
```

```
Bairstow[a, -0.4 - 0.9I, -0.4 + 0.9I]
{-0.45 - 0.947365 i, -0.45 + 0.947365 i}
```

Podemos encontrar la otras dos soluciones con los valores iniciales: $x_1 = 1 - 1.4i$; $x_2 = 1 + 1.4i$:

```
Bairstow[a, 1 - 1.4I, 1 + 1.4I]
{1. - 1.41421 i, 1. + 1.41421 i}
```

Resultados que pueden ser corroborados con "NSolve":

```
NSolve[p[x], x]
{{x -> -0.45 - 0.947365 i}, {x -> -0.45 + 0.947365 i},
 {x -> 1. - 1.41421 i}, {x -> 1. + 1.41421 i}}
```

16.3.2. Ejemplos

3. Encuentre las cuatro soluciones de la ecuación polinomial, empleando valores iniciales: $x_1 = 0.04$, $x_2 = 0.3$ y $x_1 = 0.6$, $x_2 = 1$.

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

Programamos la función, extraemos los coeficientes y llamamos dos veces al método de Bairstow:

```
p[x_] := 128*x^4 - 256*x^3 + 160*x^2 - 32*x + 1
```

```
a = Reverse[CoefficientList[p[x], x]]
{128, -256, 160, -32, 1}
```

```
Bairstow[a, 0.04, 0.3]
{0.0380602, 0.308658}
```

```
Bairstow[a,0.6,1]
{0.691342,0.96194}
```

Soluciones que pueden ser corroboradas con "NSolve":

```
NSolve[p[x],x]
{{x->0.0380602},{x->0.308658},{x->0.691342},{x->0.96194}}
```

En lugar de llamar por separado dos veces al método de Bairstow, se puede programar la solución en un módulo:

```
Module[{p,a,v={0.04,0.3,0.6,1},x1,x2},
  p[x_]:=128*x^4-256*x^3+160*x^2-32*x+1;
  a=Reverse[CoefficientList[p[x],x]];
  Do[{x1,x2}=Bairstow[a,v[[i]],v[[i+1]]];
    Print["x",i,"=",x1,"\n","x",i+1,"=",x2],
    {i,1,Length[a]-1,2}]]
x1 = 0.0380602
x2 = 0.308658
x3 = 0.691342
x4 = 0.96194
```

Igualmente, podemos encontrar dos de las soluciones comenzando con dos valores iniciales cualesquiera, digamos 1.1 y 1.2:

```
{x1,x2}=Bairstow[a,1.1,1.2]
{0.0380602,0.96194}
```

Luego, con estos resultados, obtenemos el polinomio residual:

```
d=CoefficientList[Expand[(x-x1)*(x-x2)],x]
{0.0366117,-1.,1}
```

```
b=Take[DivSin2[a,-d[[2]],-d[[1]]],3]
{128,-128.,27.3137}
```

Y encontramos las otras dos soluciones a partir de este polinomio:

```
Bairstow[b,1.1,1.2]
{0.308658,0.691342}
```

Como se puede observar se encuentran las mismas soluciones que en los anteriores casos, sólo que ahora el orden es diferente.

- Encuentre las 8 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1 = 3.8$, $x_2 = 4.8$, $x_3 = 5.8$, $x_4 = 6.8$, $x_5 = 7.8$, $x_6 = 8.8$, $x_7 = 10.2$, $x_8 = 10.8$. Luego encuentre las 8 soluciones, calculando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual y así hasta encontrar las 8 soluciones. En todos los casos emplee como valores iniciales 1.1 y 1.2.

$$P_8(x) = x^8 - 60x^7 + 1554x^6 - 22680x^5 + 203889x^4 - 1155420x^3 + 4028156x^2 - 7893840x + 6652800 = 0$$

Primero encontramos las soluciones con los 8 valores asumidos:

```
Module[{p,a,v={3.8,4.8,5.8,6.8,7.8,8.8,10.2,10.8},x1,x2},
  p[x_]:=x^8-60*x^7+1554*x^6-22680*x^5+203889*x^4-1155420*x^3+4028156*x^2-
  7893840*x+6652800;
  a=Reverse[CoefficientList[p[x],x]];
  Do[{x1,x2}=Bairstow[a,v[[i]],v[[i+1]]];
    Print["x",i,"=",x1,"\n","x",i+1,"=",x2],
    {i,1,Length[a]-1,2}]]
x1 = 4.
x2 = 5.
```

```
x3 = 6.
x4 = 7.
x5 = 8.
x6 = 9.
x7 = 10.
x8 = 11.
```

Resultados que pueden ser corroborados con "NSolve":

```
NSolve[p[x],x]
{{x->4.},{x->5.},{x->6.},{x->7.},{x->8.},{x->9.},{x->10.},{x->11.}}
```

Puede comprobar en este caso que si para la penúltima raíz no se da un valor superior a 10, no se logra encontrar dicha solución. En general, a medida que incrementa el grado del polinomio, es más difícil lograr la convergencia hacia todas las raíces.

Ahora resolvamos el problema encontrando dos soluciones en cada caso y trabajando luego con los polinomios residuales:

```
Module[{p,a,b,d,n,x1,x2},
  p[x_]:=x^8-60*x^7+1554*x^6-22680*x^5+203889*x^4
    -1155420*x^3+4028156*x^2-7893840*x+6652800;
  a=Reverse[CoefficientList[p[x],x]];
  n=Length[a]-1;
  Do[{x1,x2}=Bairstow[a,1.1,1.2];
    Print["x",i,"=",x1,"\n","x",i+1,"=",x2];
    d=CoefficientList[Expand[(x-x1)*(x-x2)],x];
    a=Take[DivSin2[a,-d[[2]],-d[[1]]],n-i],
    {i,1,n,2}]]
x1 = 4.
x2 = 11.
x3 = 5.
x4 = 10.
x5 = 6.
x6 = 9.
x7 = 7.
x8 = 8.
```

Como se puede comprobar, se obtienen igualmente las 8 soluciones, aunque el orden de las mismas difiere con relación a "NSolve".

- Encuentre las soluciones de la ecuación polinomial que se presenta al pie de la pregunta empleando como valores iniciales: $x_1=0.9-0.9i$, $x_2=0.9+0.9i$, $x_3=2.9$, $x_4=5.1$, $x_5=1.1$, para la última raíz emplee el método de Newton. Luego encuentre la primera solución con el método de Newton y las otras cuatro con el método de Bairstow, calculando primero 2 soluciones con el polinomio original y luego otras dos con el polinomio residual. En todos los casos emplee como valores iniciales 1.1 y 1.2.

$$P_5(x) = x^5 - 11x^4 + 43x^3 - 79x^2 + 76x - 30 = 0$$

Encontramos las soluciones con los valores iniciales dados en el siguiente módulo:

```
Module[{p,a,v={0.9-0.9 I,0.9+0.9 I,2.9,5.1,1.1},x1,x2},
  p[x_]:=x^5-11*x^4+43*x^3-79*x^2+76*x-30;
  a=Reverse[CoefficientList[p[x],x]];
  Do[{x1,x2}=Bairstow[a,v[[i]],v[[i+1]]];
    Print["x",i,"=",x1,"\n","x",i+1,"=",x2],
    {i,1,Length[a]-2,2}];
  Print["x5=",NewtonP[a,v[[5]]]]]
```

```

x1 = 1. -1. i
x2 = 1. +1. i
x3 = 3.
x4 = 5.
x5 = 1.

```

Resultados que pueden ser corroborados con "NSolve":

```
p[x_]:=x^5-11*x^4+43*x^3-79*x^2+76*x-30;
```

```

NSolve[p[x],x]
{{x->1.},{x->1. -1. i},{x->1. +1. i},{x->3.},{x->5.}}

```

Encontramos las soluciones trabajando con los residuos en el siguiente módulo:

```

Module[{p,a,n,d,v={0.9-0.9 I,0.9+0.9 I,2.9,5.1,1.1},x1,x2},
  p[x_]:=x^5-11*x^4+43*x^3-79*x^2+76*x-30;
  a=Reverse[CoefficientList[p[x],x]];
  n=Length[a]-1;
  x1=NewtonP[a,1.1];
  Print["x5=",x1];
  a=Most[DivSin1[a,x1]];
  Do[{x1,x2}=Bairstow[a,1.1,1.2];
    Print["x",i,"=",x1,"\n","x",i+1,"=",x2];
    d=Re[CoefficientList[Expand[(x-x1)*(x-x2)],x]];
    a=Take[DivSin2[a,-d[[2]],-d[[1]]],n-i],
    {i,2,n,2}]]
x5 = 1.
x2 = 1. -1. i
x3 = 1. +1. i
x4 = 3.
x5 = 5.

```

Y como vemos en ambos casos se obtienen los mismos resultados.

16.4. Ejercicios

15. Elabore una función que, empleando la estructura Do, encuentre dos de las raíces de una ecuación polinomial por el método de Bairstow.
16. Elabore una función que, empleando la estructura For, encuentre dos de las raíces de una ecuación polinomial por el método de Bairstow.
17. Elabore una función que, empleando el comando Goto, encuentre dos de las raíces de una ecuación polinomial por el método de Bairstow.
18. Elabore una función que, empleando la recursividad, encuentre dos de las raíces de una ecuación polinomial por el método de Bairstow.
19. Elabore una función que, empleando el comando Nest, encuentre dos de las raíces de una ecuación polinomial por el método de Bairstow.
20. Encuentre las 4 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1=0.3$, $x_2=1.7$, $x_3=4.5$, $x_4=9$. Luego encuentre las 4 soluciones, encontrando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual. En todos los casos emplee como valores iniciales 1.1 y 1.2. Corrobore los resultados obtenidos con "Solve".

$$P_4(x) = x^4 - 16x^3 + 72x^2 - 96x + 24 = 0$$

21. Encuentre las 6 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1=-1$, $x_2=-0.7$, $x_3=-0.3$, $x_4=0.4$, $x_5=0.7$, $x_6=1$. Luego encuentre las 6 soluciones, calculando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual y así hasta encontrar las 6 soluciones. En todos los casos emplee como valores iniciales 1.1 y 1.2. Corrobore los resultados obtenidos con "Reduce".

$$P_6(x) = 32x^6 - 48x^4 + 18x^2 - 1 = 0$$

22. Encuentre las 6 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1=-0.9$, $x_2=-0.7$, $x_3=-0.2$, $x_4=0.2$, $x_5=0.7$, $x_6=0.9$. Luego encuentre las 6 soluciones, calculando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual y así hasta encontrar las 6 soluciones. En todos los casos emplee como valores iniciales 1.1 y 1.2. Corrobore los resultados obtenidos con "FindInstance".

$$P_6(x) = 693x^6 - 945x^4 + 315x^2 - 15 = 0$$

23. Encuentre las 6 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1=-7$, $x_2=-1$, $x_3=0.6-0.3i$, $x_4=0.6+0.3i$, $x_5=3-5i$, $x_6=3+5i$. Luego encuentre las 6 soluciones, calculando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual y así hasta encontrar las 6 soluciones. En todos los casos emplee como valores iniciales 1.1 y 1.2. Corrobore los resultados obtenidos con "NSolve".

$$P_6(x) = x^6 + x^5 - 8x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

24. Encuentre las 8 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, empleando como valores iniciales: $x_1=-3$, $x_2=2.1+1.1i$, $x_3=2.1-1.1i$, $x_4=1.9-0.9i$, $x_5=1.9+0.9i$, $x_6=2.1$, $x_7=-0.9$, $x_8=0.9$. Luego encuentre las 8 soluciones, encontrando primero 2 soluciones con el polinomio original, luego otras dos con el polinomio residual y así hasta encontrar las 8 soluciones. En todos los casos emplee como valores iniciales 1.1 y 1.2. Corrobore los resultados obtenidos con "Reduce".

$$P_8(x) = x^8 - 7x^7 + 11x^6 + 41x^5 - 183x^4 + 231x^3 + 21x^2 - 265x + 150 = 0$$

25. Encuentre las 9 soluciones de la ecuación polinomial que se presenta al pie de la pregunta, calculando la primera solución con el método de Newton (con un valor inicial igual a 1.1), luego encuentre las 8 soluciones restantes con el método de Bairstow, trabajando con los residuos de las divisiones anteriores y empleando en todos los casos valores iniciales iguales a 1.1 y 1.2. Corrobore los resultados obtenidos con "FindInstance".

$$P_9(x) = 5x^9 - 8x^7 + 4x^6 - 2x^5 + x^4 - 6x^3 + 3x^2 - 2x - 80 = 0$$

17. SISTEMAS DE ECUACIONES LINEALES

Con frecuencia en el campo de la ingeniería es necesario resolver sistemas de ecuaciones lineales que tienen entre 2 y cientos de miles de ecuaciones lineales.

Los métodos que permiten resolver ecuaciones lineales pueden clasificarse en dos: a) *No iterativos o directos*, que permiten resolver sistemas de ecuaciones que cuentan con hasta algunos cientos de ecuaciones lineales (no por las limitaciones de los métodos, sino por los errores de redondeo) y b) *Los métodos iterativos*, que permiten resolver sistemas con hasta cientos de miles de ecuaciones lineales, aunque como normalmente sucede con este tipo de métodos, no siempre logran la convergencia.

Primero estudiaremos algunas de las funciones con las que cuenta Mathematica para la resolución de sistemas de ecuaciones lineales.

17.1. Solución de ecuaciones lineales con Mathematica

Mathematica posee varias funciones para resolver sistemas de ecuaciones lineales en forma tanto simbólica como numérica. Dentro de estas funciones podemos mencionar a "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

"Solve", es una función genérica permitiendo resolver no sólo sistemas de ecuaciones lineales, sino y como ya vimos, ecuaciones polinomiales y sistemas de ecuaciones polinomiales. También permite resolver ecuaciones trigonométricas y exponenciales, sin embargo, al ser general y permitir encontrar soluciones simbólicas, requiere normalmente más tiempo que la mayoría de las otras funciones con las que cuenta Mathematica.

`Solve[{ecuaciones},variables]`

Donde las *ecuaciones* están en la forma: *lado_izquierdo == lado_derecho*, así por ejemplo para resolver el siguiente sistema de ecuaciones lineales:

$$\begin{array}{rrcrcl} 2x_1 & + & 8x_2 & + & 2x_3 & = & 14 \\ x_1 & + & 6x_2 & - & x_3 & = & 13 \\ 2x_1 & - & x_2 & + & 2x_3 & = & 5 \end{array} \quad (1)$$

Escribimos:

```
Solve[{2*x1+8*x2+2*x3==14,
      x1+6*x2-x3==13,
      2*x1-x2+2*x3==5},
      {x1,x2,x3}]
{{x1->5,x2->1,x3->-2}}
```

En lugar de escribir las ecuaciones en una lista pueden ser escritas también en forma de una expresión lógica, separando las ecuaciones con el operador "y" ("&&"):

```
Solve[2*x1+8*x2+2*x3==14&&
      x1+6*x2-x3==13&&
      2*x1-x2+2*x3==5,
      {x1,x2,x3}]
{{x1->5,x2->1,x3->-2}}
```

En lugar de "Solve" podemos emplear "NSolve", prácticamente con el mismo formato, sólo que "NSolve" busca soluciones aproximadas (en lugar de soluciones exactas) y por defecto espera la lista de variables:

```
NSolve[2*x1+8*x2+2*x3==14&&
  x1+6*x2-x3==13&&
  2*x1-x2+2*x3==5,
  {x1,x2,x3}]
{{x1→5.,x2→1.,x3→-2.}}
```

Además, a "NSolve" se le puede mandar directamente las ecuaciones igualadas a cero y no siempre igualdades, como se requiere para "Solve".

```
NSolve[{2*x1+8*x2+2*x3-14,
  x1+6*x2-x3-13,
  2*x1-x2+2*x3-5},
  {x1,x2,x3}]
{{x1→5.,x2→1.,x3→-2.}}
```

"Reduce" al igual que "Solve" es una función de carácter general, que además permite resolver desigualdades y elegir el dominio, por lo que normalmente requiere inclusive más tiempo que "Solve". Su sintaxis es similar a la de "Solve" (permitiendo elegir además el dominio) así para resolver el anterior sistema de ecuaciones con esta función escribimos:

```
Reduce[{2*x1+8*x2+2*x3==14,
  x1+6*x2-x3==13,
  2*x1-x2+2*x3==5},
  {x1,x2,x3}]
x1==5&& x2==1&& x3== -2
```

O también:

```
Reduce[2*x1+8*x2+2*x3==14&&
  x1+6*x2-x3==13&&
  2*x1-x2+2*x3==5,
  {x1,x2,x3}]
x1==5&& x2==1&& x3== -2
```

"FindInstance" es también una función de carácter general, que además permite fijar el dominio y el número de instancias. Su sintaxis es similar a la de "Reduce" y "Solve", pero con "FindInstance" la lista de variables es obligatoria:

```
FindInstance[{2*x1+8*x2+2*x3==14,
  x1+6*x2-x3==13,
  2*x1-x2+2*x3==5},
  {x1,x2,x3}]
{{x1→5,x2→1,x3→-2}}
```

O también:

```
FindInstance[2*x1+8*x2+2*x3==14&&
  x1+6*x2-x3==13&&
  2*x1-x2+2*x3==5,
  {x1,x2,x3}]
{{x1→5,x2→1,x3→-2}}
```

Las otras funciones son más especializadas en la solución de sistemas de ecuaciones lineales (tanto numéricas como simbólicas), por lo que en general encuentran las soluciones en menos tiempo que "Solve", "Reduce" o "FindInstance".

Veamos primero la función "LinearSolve", cuya sintaxis es la siguiente:

LinearSolve[matriz_de_coeficientes, vector_o_matriz_de_constantes]

Así para resolver el sistema de ecuaciones lineales (1) podemos escribir directamente la matriz de los coeficientes y el vector de las constantes:

```
LinearSolve[{{2,8,2},{1,6,-1},{2,-1,2}},{14,13,5}]
{5,1,-2}
```

O también podemos escribir la matriz de los coeficientes y la matriz de las constantes:

```
LinearSolve[{{2,8,2},{1,6,-1},{2,-1,2}},{{14},{13},{5}}}
{{5},{1},{-2}}
```

En ambos casos obtenemos los mismos resultados, pero en este último caso se obtiene una matriz (con una columna) en lugar de un vector, algo que se puede apreciar mejor si se emplea el comando "MatrixForm":

```
%//MatrixForm

$$\begin{pmatrix} 5 \\ 1 \\ -2 \end{pmatrix}$$

```

Si tenemos escritas las ecuaciones, podemos extraer los coeficientes con el comando "CoefficientArrays", que devuelve una lista con dos matrices dispersas: la primera con las constantes del sistema (con el signo cambiado) y la segunda con los coeficientes del sistema (una matriz dispersa es una matriz en la que sólo se guardan los elementos cuyos valores son diferentes a cero). Así por ejemplo, si guardamos el sistema de ecuaciones en la variable "f":

```
f={2*x1+8*x2+2*x3==14,
  x1+6*x2-x3==13,
  2*x1-x2+2*x3==5}
{2x1+8x2+2 x3==14,x1+6x2-x3==13,2 x1-x2+2x3==5}
```

Podemos extraer los coeficientes y la constante con:

```
CoefficientArrays[f]
{SparseArray[< 3 >,{3}],SparseArray[< 9 >,{3,3}]}
```

Que, como se puede observar, devuelve dos matrices dispersas, la primera con 3 valores (<3>), siendo un vector de 3 elementos ({3}) y la segunda con 9 valores (<9>), siendo una matriz con 3 filas y 3 columnas ({3,3}). Podemos convertir esta matriz dispersa a su forma normal con el comando "Normal":

```
%//Normal
{{-14,-13,-5},{2,8,2},{1,6,-1},{2,-1,2}}}
```

Comprobamos así, que la primera matriz son las constantes (pero con signo cambiado) y la segunda los coeficientes. Entonces podemos emplear estas matrices como datos en "LinearSolve":

```
{b,m}=%
{{-14,-13,-5},{2,8,2},{1,6,-1},{2,-1,2}}}
```

```
LinearSolve[m,-b]
{5,1,-2}
```

También podemos mandar a "LinearSolve" sólo la matriz de los coeficientes, en cuyo caso nos devuelve una función (LinearSolveFunction) que recibe como datos el vector (o matriz de los coeficientes) y devuelve como resultado las soluciones del sistema de ecuaciones:

```
g=LinearSolve[m]
LinearSolveFunction[{3,3},<>]
```

```
g[-b]
{5,1,-2}
```

Esta forma se emplea sobre todo cuando se requiere resolver dos o más sistemas de ecuaciones lineales que sólo difieren en las constantes, por ejemplo para resolver el siguiente sistema de ecuaciones lineales, que difiere del anterior sólo en los coeficientes:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 3 \\ x_1 + 6x_2 - x_3 &= 7 \\ 2x_1 - x_2 + 2x_3 &= 9 \end{aligned} \quad (2)$$

Simplemente mandamos a la función "g" las nuevas constantes:

```
g[{3,7,9}]
{91/12,-2/3,-41/12}
```

Obteniéndose así los resultados del sistema, mismos que pueden ser corroborados multiplicando el vector de las soluciones por la matriz de los coeficientes y verificando si se reproduce el vector de las constantes:

```
m.%
{3,7,9}
```

Comprobamos así que los resultados son correctos.

Otra función especializada en la resolución de sistemas de ecuaciones lineales es "RowReduce", cuya sintaxis es la siguiente:

RowReduce[matriz_aumentada]

Donde "*matriz_aumentada*", es la matriz de los coeficientes "aumentada" con la columna (o columnas) de las constantes. Esta función nos devuelve la matriz en la forma de filas reducidas, es decir la matriz unidad en las columnas correspondientes a los coeficientes y las soluciones en las columnas correspondientes a las constantes, así por ejemplo para resolver el sistema de ecuaciones lineales (1) escribimos:

```
RowReduce[{{2,8,2,14},{1,6,-1,13},{2,-1,2,5}}]
{{1,0,0,5},{0,1,0,1},{0,0,1,-2}}
```

Podemos ver con mayor claridad que las soluciones se encuentran en la última columna y que la matriz de los coeficientes ha quedado en forma de una matriz unidad, si colocamos el resultado en su forma matricial:

```
%//MatrixForm

$$\begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

```

Y para quedarnos sólo con los resultados extraemos todas las filas (All) de la columna 4:

```
%%[All,4]
{5,1,-2}
```

Por supuesto también podemos emplear la matriz de los coeficientes y el vector de las constantes, obtenidos con "CoefficientArrays", para armar la matriz aumentada. Para ello debemos añadir a la matriz de los coeficientes el vector de las constantes (como una columna), lo que se puede lograr transponiendo la matriz de los coeficientes (con "Transpose"), añadiéndole el vector de las constantes como una fila (con "Append") y transponiendo la matriz resultante:

```
b=-b
{14,13,5}
```

```

Transpose[m]
{{2,1,2},{8,6,-1},{2,-1,2}}

Append[%,b]
{{2,1,2},{8,6,-1},{2,-1,2},{14,13,5}}

Transpose[%]
{{2,8,2,14},{1,6,-1,13},{2,-1,2,5}}

%/MatrixForm

$$\begin{pmatrix} 2 & 8 & 2 & 14 \\ 1 & 6 & -1 & 13 \\ 2 & -1 & 2 & 5 \end{pmatrix}$$


```

O en una sola instrucción:

```

Transpose[Append[Transpose[m],b]]
{{2,8,2,14},{1,6,-1,13},{2,-1,2,5}}

```

Como casi siempre sucede en Mathematica no es la única alternativa, así, en lugar de "Append", podemos emplear "Join":

```

Transpose[Join[Transpose[m],{b}]]
{{2,8,2,14},{1,6,-1,13},{2,-1,2,5}}

```

O "ArrayFlatten", que forma una matriz con las matrices que recibe como argumentos, siempre y cuando las matrices que se manden en la misma fila tengan el mismo número de fila y las que se manden en la misma columna tengan el mismo número de columnas, así en este caso es necesario transponer el vector "b" para que tenga el mismo número de filas que la matriz "m":

```

ArrayFlatten[{{m,Transpose[{b}]}}]
{{2,8,2,14},{1,6,-1,13},{2,-1,2,5}}

```

"RowReduce" puede ser empleada también para resolver dos o más sistemas de ecuaciones que difieren si es que las mismas sólo difieren en las constantes. Para ello se añaden las columnas de las constantes a la matriz de los coeficientes, así para resolver los sistemas de ecuaciones (1) y (2) escribimos, o armamos, la matriz:

```

RowReduce[{{2,8,2,14,3},{1,6,-1,13,7},{2,-1,2,5,9}}]
{{1,0,0,5,91/12},{0,1,0,1,-2/3},{0,0,1,-2,-41/12}}

```

```

%/MatrixForm

$$\begin{pmatrix} 1 & 0 & 0 & 5 & \frac{91}{12} \\ 0 & 1 & 0 & 1 & -\frac{2}{3} \\ 0 & 0 & 1 & -2 & -\frac{41}{12} \end{pmatrix}$$


```

Como vemos los resultados del primer sistema se encuentran en la cuarta columna y los del segundo en la quinta.

"RowReduce" puede ser empleada también para calcular la matriz inversa de los coeficientes, para ello se aumenta a la matriz de los coeficientes la matriz unidad. Así por ejemplo para calcular la matriz inversa de los coeficientes del sistema de ecuaciones (1) escribimos la matriz aumentada:

```

a={{2,8,2,1,0,0},{1,6,-1,0,1,0},{2,-1,2,0,0,1}}
{{2,8,2,1,0,0},{1,6,-1,0,1,0},{2,-1,2,0,0,1}}

```

```

a//MatrixForm

```

$$\begin{pmatrix} 2 & 8 & 2 & 1 & 0 & 0 \\ 1 & 6 & -1 & 0 & 1 & 0 \\ 2 & -1 & 2 & 0 & 0 & 1 \end{pmatrix}$$

Y mandamos esta matriz a "RowReduce":

```
RowReduce[a]
{{1,0,0,-11/36,1/2,5/9},{0,1,0,1/9,0,-1/9},{0,0,1,13/36,-1/2,-1/9}}
```

```
%//MatrixForm

$$\begin{pmatrix} 1 & 0 & 0 & -\frac{11}{36} & \frac{1}{2} & \frac{5}{9} \\ 0 & 1 & 0 & \frac{1}{9} & 0 & -\frac{1}{9} \\ 0 & 0 & 1 & \frac{13}{36} & -\frac{1}{2} & -\frac{1}{9} \end{pmatrix}$$

```

Como se puede observar, la matriz invertida queda en las tres últimas columnas, en el lugar ocupado originalmente por la matriz unidad. En vez de escribir manualmente la matriz unidad (como se ha hecho en el ejemplo), podemos generarla con "IdentityMatrix":

IdentityMatrix[Número_de_filas]

Esta función genera una matriz unidad con "n" filas, así por ejemplo para genera una matriz unidad con 5 filas escribimos:

```
IdentityMatrix[5]
{{1,0,0,0,0},{0,1,0,0,0},{0,0,1,0,0},{0,0,0,1,0},{0,0,0,0,1}}
```

```
%//MatrixForm

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

```

Así para generar la matriz "a" podemos escribir:

```
ArrayFlatten[{{m,IdentityMatrix[3]}}]
{{2,8,2,1,0,0},{1,6,-1,0,1,0},{2,-1,2,0,0,1}}
```

O también:

```
Transpose[Join[Transpose[m],Transpose[IdentityMatrix[3]]]]
{{2,8,2,1,0,0},{1,6,-1,0,1,0},{2,-1,2,0,0,1}}
```

Para una matriz con tres filas, el proceso descrito parece más complicado que escribir directamente las columnas de la matriz unidad, sin embargo, cuando se trabaja con decenas o centenas de filas resulta mucho más sencillo y eficiente. Claro que "RowReduce" no es el comando más conveniente para encontrar la matriz inversa, para ese fin Mathematica cuenta con "Inverse", cuya sintaxis es la siguiente:

Inverse[matriz_cuadrada]

Así para encontrar la matriz inversa del sistema de ecuaciones (1) escribimos:

```
mi=Inverse[m]
{{-11/36,1/2,5/9},{1/9,0,-1/9},{13/36,-1/2,-1/9}}
```

La matriz inversa constituye otra forma de resolver sistemas de ecuaciones lineales, pues una vez calculada, se obtienen las soluciones del sistema multiplicándola por el vector (o matriz) de las constantes, así para calcular las soluciones del sistema de ecuaciones (1), multiplicamos la matriz inversa "mi" por el vector de las constantes "b":

```
mi.b
{5,1,-2}
```

Como se puede observar en Mathematica el operador para multiplicar dos matrices es punto "." (Dot).

Con la matriz inversa se pueden resolver también dos o más sistemas de ecuaciones lineales que difieren sólo en las constantes, multiplicando la matriz inversa por la matriz de las constantes. Así para resolver los sistemas de ecuaciones (1) y (2) escribimos:

```
mi.{{14,3},{13,7},{5,9}}
{{5,91/12},{1,-2/3},{-2,-41/12}}
```

17.2. Ejemplos

1. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$10x_1 + x_2 + 2x_3 = 44$$

$$2x_1 + 10x_2 + x_3 = 51$$

$$x_1 + 2x_2 + 10x_3 = 61$$

```
f={10*x1+x2+2*x3==44,
  2*x1+10*x2+x3==51,
  x1+2*x2+10*x3==61};
```

```
Solve[f]
{{x1->3,x2->4,x3->5}}
```

```
NSolve[f,{x1,x2,x3}]
{{x1->3.,x2->4.,x3->5.}}
```

```
Reduce[f]
x3==5&& x2==4&& x1==3
```

```
FindInstance[f,{x1,x2,x3}]
{{x1->3,x2->4,x3->5}}
```

```
{b,m}=Normal[CoefficientArrays[f]]
{{-44,-51,-61},{10,1,2},{2,10,1},{1,2,10}}}
```

```
b=-b
{44,51,61}
```

```
LinearSolve[m,b]
{3,4,5}
```

```
g=LinearSolve[m]
LinearSolveFunction[{3,3},<>]
```

```
g[b]
{3,4,5}
```

```
RowReduce[ArrayFlatten[{{m,Transpose[{b]}]}]]][[All,4]]
{3,4,5}
```

```
Inverse[m].b
{3,4,5}
```

2. Encuentre las soluciones de los siguientes sistemas de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\ x_1 + 4x_2 + 2x_4 &= 0 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

$$\begin{aligned} 3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\ y_1 + 4y_2 + 2y_4 &= 2 \\ 2y_1 + y_2 + 2y_3 - y_4 &= 3 \\ y_1 + y_2 - y_3 + 3y_4 &= 4 \end{aligned}$$

$$\begin{aligned} 3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\ z_1 + 4z_2 + 2z_4 &= 2 \\ 2z_1 + z_2 + 2z_3 - z_4 &= 1 \\ z_1 + z_2 - z_3 + 3z_4 &= 0 \end{aligned}$$

Con las cuatro primeras instrucciones no existe forma de resolver los tres sistemas de ecuaciones en uno, sin embargo, se puede escribir uno de los sistemas:

```
f1={3*x1+2*x2-x3+2*x4==0,
    x1+4*x2+2*x4==0,
    2*x1+x2+2*x3-x4==1,
    x1+x2-x3+3*x4==0};
```

Extraer la matriz de coeficientes y el vector de constantes:

```
CoefficientArrays[f1]
{SparseArray[< 1 >,{4}],SparseArray[< 15 >,{4,4}]}

{b,m}=%//Normal
{{0,0,-1,0},{3,2,-1,2},{1,4,0,2},{2,1,2,-1},{1,1,-1,3}}}
```

```
b=-b
{0,0,1,0}
```

Escribir los otros dos vectores de coeficientes:

```
c={-2,2,3,4}
{-2,2,3,4}
```

```
d={2,2,1,0}
{2,2,1,0}
```

Y armar los otros dos sistemas con estos datos:

```
f2=Thread[m.{y1,y2,y3,y4}==c]
{3 y1+2 y2-y3+2 y4==-2,y1+4 y2+2 y4==2,2 y1+y2+2 y3-y4==3,y1+y2-y3+3 y4==4}
```

```
f3=Thread[m.{z1,z2,z3,z4}==d]
{3 z1+2 z2-z3+2 z4==2,z1+4 z2+2 z4==2,2 z1+z2+2 z3-z4==1,z1+z2-z3+3 z4==0}
```

Entonces se mandan estos sistemas a las instrucciones respectivas:

```
Solve[f1]
{{x1->3/22,x2->-5/44,x3->1/2,x4->7/44}}
```

```
Solve[f2]
{{y1->-17/22,y2->-45/44,y3->9/2,y4->151/44}}
```

```

Solve[f3]
{{z1→9/22,z2→29/44,z3→-1/2,z4→-23/44}}

NSolve[f1,{x1,x2,x3,x4}]
{{x1→0.136364,x2→-0.113636,x3→0.5,x4→0.159091}}

NSolve[f2,{y1,y2,y3,y4}]
{{y1→-0.772727,y2→-1.02273,y3→4.5,y4→3.43182}}

NSolve[f3,{z1,z2,z3,z4}]
{{z1→0.409091,z2→0.659091,z3→-0.5,z4→-0.522727}}

Reduce[f1,{x1,x2,x3,x4}]
x1==3/22&& x2== -5/44&& x3==1/2&& x4==7/44

Reduce[f2,{y1,y2,y3,y4}]
y1== -17/22&& y2== -45/44&& y3==9/2&& y4==151/44

Reduce[f3,{z1,z2,z3,z4}]
z1==9/22&& z2==29/44&& z3== -1/2&& z4== -23/44

FindInstance[f1,{x1,x2,x3,x4}]
{{x1→3/22,x2→-5/44,x3→1/2,x4→7/44}}

FindInstance[f2,{y1,y2,y3,y4}]
{{y1→-17/22,y2→-45/44,y3→9/2,y4→151/44}}

FindInstance[f3,{z1,z2,z3,z4}]
{{z1→9/22,z2→29/44,z3→-1/2,z4→-23/44}}

```

Con "LinearSolve" podemos resolver los tres sistemas a la vez mandando la matriz de los coeficientes y la matriz de las constantes (armada a partir de los vectores de las constantes):

```

LinearSolve[m,Transpose[{b,c,d}]]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-23/44}}

```

También podemos mandar solo la matriz de los coeficientes y con la función resultante ("LinearSolveFunction") y la matriz de las constantes, resolver los tres sistemas:

```

g=LinearSolve[m]
LinearSolveFunction[{4,4},<>]

g[Transpose[{b,c,d}]]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-23/44}}

```

Para "RowReduce" armamos primero la matriz aumentada:

```

a=ArrayFlatten[{{m,Transpose[{b,c,d}]}}]
{{3,2,-1,2,0,-2,2},{1,4,0,2,0,2,2},{2,1,2,-1,1,3,1},{1,1,-1,3,0,4,0}}

```

Mandamos esta matriz a "RowReduce" y extraemos las tres últimas columnas de la matriz resultante:

```

RowReduce[a][[All,5;;7]]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-23/44}}

```

Con "Inverse" podemos resolver también los tres sistemas en uno:

```
Inverse[m].Transpose[{b,c,d}]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-
23/44}}
```

Y aunque no es lo más eficiente, podemos encontrar también la matriz inversa con "RowReduce" (o "LinearSolve") y emplear dicha matriz para encontrar las soluciones:

```
RowReduce[ArrayFlatten[{m,IdentityMatrix[4]}]][[All,5;;8]]
{{7/22,-2/11,3/22,-1/22},{3/44,7/22,-5/44,-13/44},{-1/2,0,1/2,1/2},{-13/44,-
1/22,7/44,27/44}}
```

```
%.Transpose[{b,c,d}]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-
23/44}}
```

O:

```
LinearSolve[m,IdentityMatrix[4]]
{{7/22,-2/11,3/22,-1/22},{3/44,7/22,-5/44,-13/44},{-1/2,0,1/2,1/2},{-13/44,-
1/22,7/44,27/44}}
```

```
%.Transpose[{b,c,d}]
{{3/22,-17/22,9/22},{-5/44,-45/44,29/44},{1/2,9/2,-1/2},{7/44,151/44,-
23/44}}
```

17.3. Ejercicios

1. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$3x_1 - x_2 + 2x_3 = 12$$

$$x_1 + 2x_2 + 3x_3 = 11$$

$$2x_1 - 2x_2 - x_3 = 2$$

2. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$2x_1 - 2x_2 + 5x_3 = 13$$

$$2x_1 + 3x_2 + 4x_3 = 20$$

$$3x_1 - x_2 + 3x_3 = 10$$

3. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$x_1 + x_2 + x_3 + x_4 = -1$$

$$4x_1 + 5x_2 + 6x_3 + 7x_4 = 0$$

$$6x_1 + 10x_2 + 15x_3 + 21x_4 = 0$$

$$12x_1 + 30x_2 + 60x_3 + 105x_4 = 0$$

4. Encuentre las soluciones del siguiente sistema de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 = 12$$

$$2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 = 24$$

$$x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 = 17$$

$$3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 = 37$$

$$4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 = 23$$

$$5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 = 24$$

5. Encuentre las soluciones de los siguientes sistemas de ecuaciones lineales empleando "Solve", "NSolve", "Reduce", "FindInstance", "LinearSolve", "LinearSolveFunction", "RowReduce" e "Inverse".

$$\begin{aligned} 30x_1 + 1x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 7 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 7 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 3 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 4 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 4 \end{aligned}$$

$$\begin{aligned} 30y_1 + 1y_2 + 2y_3 + 6y_4 + 5y_5 - 3y_6 &= 14 \\ y_1 + 2y_2 + 33y_3 + 5y_4 - y_5 + 2y_6 &= 1 \\ 3y_1 + 5y_2 + 7y_3 - 45y_4 + 4y_5 + y_6 &= 3 \\ 4y_1 + 3y_2 + 2y_3 + 3y_4 + 40y_5 - 4y_6 &= 2 \\ 5y_1 + 6y_2 - 3y_3 - 4y_4 + 3y_5 + 36y_6 &= 2 \\ 2y_1 + 25y_2 + y_3 - y_4 + 2y_5 - 5y_6 &= 7 \end{aligned}$$

$$\begin{aligned} 30z_1 + 1z_2 + 2z_3 + 6z_4 + 5z_5 - 3z_6 &= 4 \\ z_1 + 2z_2 + 33z_3 + 5z_4 - z_5 + 2z_6 &= 27 \\ 3z_1 + 5z_2 + 7z_3 - 45z_4 + 4z_5 + z_6 &= 27 \\ 4z_1 + 3z_2 + 2z_3 + 3z_4 + 40z_5 - 4z_6 &= 13 \\ 5z_1 + 6z_2 - 3z_3 - 4z_4 + 3z_5 + 36z_6 &= 14 \\ 2z_1 + 25z_2 + z_3 - z_4 + 2z_5 - 5z_6 &= 14 \end{aligned}$$

18. GAUSS, GAUSS JORDAN Y CHOLESKY

Aunque en la solución de la mayoría de los problemas prácticos emplearemos las funciones que vienen con Mathematica y que hemos estudiado en el anterior tema, es conveniente conocer algunos de los métodos existentes para comprender el proceso de solución y poder elegir así el método más adecuado para resolver el problema. Es necesario también conocer los métodos en aquellos casos donde debamos elaborar nuestros propios programas debido a que el sistema (o lenguaje) con el que estemos trabajando, no cuente con funciones para resolver este tipo de problemas. Por esta razón en este tema estudiaremos tres métodos directos (no iterativos).

18.1. Método de eliminación de Gauss

Comenzaremos el estudio de los métodos numéricos para resolver sistemas de ecuaciones lineales con el método de eliminación de Gauss, el cual transforma el sistema de ecuaciones lineales en un sistema triangular superior. Así el siguiente sistema de 4 ecuaciones lineales:

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4 \end{aligned} \quad (1)$$

Se transforma en el siguiente sistema triangular superior:

$$\begin{aligned} x_1 + a'_{1,2}x_2 + a'_{1,3}x_3 + a'_{1,4}x_4 &= c'_1 \\ 0 + x_2 + a'_{2,3}x_3 + a'_{2,4}x_4 &= c'_2 \\ 0 + 0 + x_3 + a'_{3,4}x_4 &= c'_3 \\ 0 + 0 + 0 + x_4 &= c'_4 \end{aligned} \quad (2)$$

Donde los apóstrofes se han empleado simplemente para diferenciar los coeficientes originales de los coeficientes del sistema resultante.

Una vez reducido el sistema a la forma triangular superior, las soluciones se calculan por sustitución inversa.

Como se puede observar, el valor de la última variable es directamente el último elemento del vector de constantes, en nuestro caso " c'_4 ". Entonces como se conoce " x_4 ", el valor de " x_3 " puede ser calculado con la penúltima ecuación:

$$x_3 = c'_3 - a'_{3,4}x_4$$

Ahora que se conocen los valores de " x_4 " y " x_3 " podemos calcular el valor de " x_2 " con la antepenúltima ecuación:

$$x_2 = c'_2 - a'_{2,3}x_3 - a'_{2,4}x_4$$

Finalmente como conocemos los valores de " x_4 ", " x_3 " y " x_2 " podemos calcular el valor de " x_1 " con la primera ecuación:

$$x_1 = c'_1 - a'_{1,2}x_2 - a'_{1,3}x_3 - a'_{1,4}x_4$$

Procediendo de esta manera (por sustitución inversa) se encuentran las soluciones del sistema, sin importar el número de ecuaciones existentes.

En general al implementar el método de eliminación de Gauss (y los otros métodos que estudiaremos posteriormente) sólo se trabaja con la matriz au-

mentada. Así, en forma matricial, la aplicación del método de Gauss al sistema de ecuaciones lineales (3), es:

$$\left[\begin{array}{ccccc} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{array} \right] \xrightarrow{\text{GAUSS}} \left[\begin{array}{ccccc} 1 & a'_{1,2} & a'_{1,3} & a'_{1,4} & a'_{1,5} \\ 0 & 1 & a'_{2,3} & a'_{2,4} & a'_{2,5} \\ 0 & 0 & 1 & a'_{3,4} & a'_{3,5} \\ 0 & 0 & 0 & 1 & a'_{4,5} \end{array} \right] \quad (3)$$

Donde en la matriz izquierda, los elementos de la última columna (la columna 5) son las constantes del sistema de ecuaciones lineales.

Para convertir la matriz aumentada en una matriz diagonal superior, se deben efectuar reducciones de filas (donde se convierten en unos los elementos de la diagonal principal) y reducciones de columnas (donde se convierten en ceros los elementos que se encuentran por debajo de la diagonal principal). Estas operaciones se deben efectuar primero para la fila y columna 1, luego para la fila y columna 2, luego para la fila y columna 3 y así sucesivamente hasta llegar a la última fila del sistema.

Puesto que los valores de la matriz sólo se emplean una vez en los cálculos, la matriz resultante puede ser almacenada en la matriz original, ahorrando así memoria.

En las siguientes ecuaciones llamaremos "p" (pivote) al contador que determina la fila y columna que se está reduciendo, "m" al número de ecuaciones del sistema y "n" al número de columnas de la matriz aumentada.

Para reducir el elemento a_{pp} a 1, dividimos los elementos de la fila p entre a_{pp} , es decir:

$$a_p = \frac{a_p}{a_{p,p}} \quad (4)$$

Donde " a_p " es la fila "p" de la matriz "a".

Para reducir a cero los elementos que se encuentra en la columna "p" por debajo del elemento " a_{pp} " (que con la anterior operación ya se ha convertido en 1), restamos a cada una de las filas que se encuentran por debajo de la fila "p", los elementos de la fila "p" multiplicada por el valor del elemento " $a_{i,p}$ ", siendo "i" el número de fila que se está reduciendo, es decir:

$$a_i = a_i - a_p * a_{i,p} \quad \{i = p+1 \rightarrow m\} \quad (5)$$

Donde " a_i " es la fila "i" de la matriz "a" y al igual que antes " a_p " es la fila "p" de la matriz "a" (fila pivote).

Una vez reducida la matriz a la forma triangular superior, aplicando las ecuaciones (6) y (7) desde " $p=1$ " hasta " $p=n$ ", las soluciones del sistema de ecuaciones se calculan por sustitución inversa.

La ecuación general para calcular los resultados por sustitución inversa y almacenarlos en las columnas de las constantes es la siguiente:

$$a_{ij} = a_{ij} - \sum_{k=i+1}^m a_{ik} * a_{kj} \quad \begin{cases} i = m-1 \rightarrow 1 \\ j = m+1 \rightarrow n, \text{ para cada valor de } i \end{cases} \quad (6)$$

18.1.1. Ejemplo manual

Para comprender mejor el método y tener valores de referencia que nos permitan corregir los errores del programa, resolveremos el siguiente sistema de ecuaciones lineales aplicando los pasos antes descritos.

$$\begin{array}{rrcr}
 2x_1 & + & 8x_2 & + & 2x_3 & = & 14 \\
 x_1 & + & 6x_2 & - & x_3 & = & 13 \\
 2x_1 & - & x_2 & + & 2x_3 & = & 5
 \end{array}
 \tag{7}$$

La matriz aumentada es:

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

El número de filas "m" y el número de columnas "n" de este sistema son:

```
m=3;n=4;
```

Inicialmente le pivote "p" es 1:

```
p=1;
```

Ahora reducimos la fila "p" (la fila 1) aplicando la ecuación (4):

```
a[[p]]=a[[p]]/a[[p,p]]
{1,4,1,7}
```

Reducimos la columna "p" (la columna 1), aplicando la ecuación (5):

```
Do[a[[i]]=a[[i]]-a[[p]]*a[[i,p]],{i,p+1,m}]
```

```
a//MatrixForm
```

```


$$\begin{pmatrix}
 1 & 4 & 1 & 7 \\
 0 & 2 & -2 & 6 \\
 0 & -9 & 0 & -9
 \end{pmatrix}$$


```

Como era de esperar, al aplicar estas dos ecuaciones, el primer elemento de la diagonal principal queda convertido en 1 y los elementos de la columna debajo del mismo en cero.

Ahora incrementamos el valor de "p" (para que valga 2) y volvemos a aplicar las ecuaciones (4) y (5):

```
p++;
```

```
a[[p]]=a[[p]]/a[[p,p]]
{0,1,-1,3}
```

```
Do[a[[i]]=a[[i]]-a[[p]]*a[[i,p]],{i,p+1,m}]
```

```
a//MatrixForm
```

```


$$\begin{pmatrix}
 1 & 4 & 1 & 7 \\
 0 & 1 & -1 & 3 \\
 0 & 0 & -9 & 18
 \end{pmatrix}$$


```

Con esta segunda aplicación el segundo elemento de la diagonal principal queda convertido en 1 y el elemento debajo del mismo en cero.

Ahora incrementamos una vez más el valor de "p" (p=3) y aplicamos la ecuación (4), ya no la (5) porque no existen más elementos debajo del tercer elemento de la diagonal principal:

```
p++;
```

```
a[[p]]=a[[p]]/a[[p,p]]
{0,0,1,-2}
```

```
a//MatrixForm
```

```


$$\begin{pmatrix}
 1 & 4 & 1 & 7 \\
 0 & 1 & -1 & 3 \\
 0 & 0 & 1 & -2
 \end{pmatrix}$$


```

Con ello la matriz queda en la forma triangular superior, por lo que se puede aplicar la ecuación (6) para calcular las soluciones:

```
Do[a[[i,j]]=a[[i,j]]-Sum[a[[i,k]]*a[[k,j]],{k,i+1,m}],
  {i,m-1,1,-1},{j,m+1,n}]
```

```
a//MatrixForm
```

```

$$\begin{pmatrix} 1 & 4 & 1 & 5 \\ 0 & 1 & -1 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

```

Vemos entonces que las soluciones del sistema quedan en la última columna ($x_1=5$, $x_2=1$, $x_3=-2$).

18.1.2. Pivotaje

Al aplicar el método de eliminación de Gauss en ocasiones puede ocurrir que el elemento de la diagonal principal (el elemento pivote) sea cero, en cuyo caso se produciría un error por división entre cero. En esos casos se debe realizar un intercambio de filas y/o columnas para evitar que ello ocurra.

Por otra parte es conveniente que el elemento pivote tenga el mayor valor absoluto posible, pues de esa manera se reducen considerablemente los errores de redondeo. Para ello con frecuencia se deben realizar intercambios de filas y columnas a fin de colocar el elemento con el mayor valor absoluto en la posición pivote.

Al proceso de búsqueda (en las filas y columnas no reducidas) del elemento con el mayor valor absoluto y el posterior intercambio de filas y columnas para que dicho valor quede en la posición pivote se conoce como *pivotaje total*. Si la búsqueda del mayor valor absoluto se realiza sólo en la columna pivote, el proceso se conoce como *pivotaje parcial*.

En la práctica, al implementar el método de eliminación de Gauss, es necesario llevar a cabo como mínimo un pivotaje parcial y de ser necesario inclusive un pivotaje total.

18.1.2.1. Pivotaje parcial

El algoritmo para realizar el pivotaje parcial es muy sencillo: en la columna pivote, se ubica la fila donde se encuentra el elemento con el mayor valor absoluto, buscando sólo desde la fila pivote hasta la última fila. Una vez ubicado el elemento, se intercambia la fila pivote con la fila donde se encuentra el mayor valor absoluto. Si el mayor valor absoluto es 0, entonces se trata de un sistema homogéneo y por lo tanto no puede ser resuelto por lo que en ese caso se genera un error. El diagrama de actividades para el pivotaje parcial se presenta en la siguiente página.

Los programas de este y el siguiente tema serán escritos en otro paquete: "SisEcu" y se comenzará programando en el mismo la función para el pivotaje parcial.

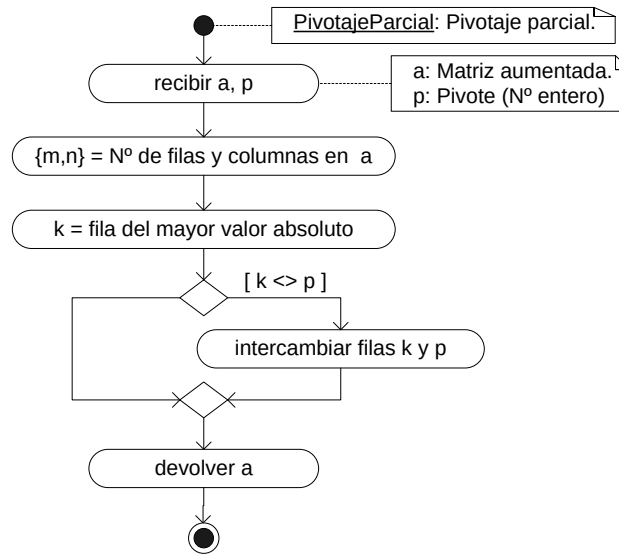
```
BeginPackage["SisEcu`"]
```

```
PivotajeParcial::usage="PivotajeParcial[a,p], realiza el pivotaje parcial
  de la matriz \"a\", comenzando en la fila y columna \"p\"."
```

```
Begin["Private`"]
```

```
SisLin::homogeneo=
```

```
"El sistema de ecuaciones es homogéneo. No puede ser resuelto."
```



```

PivotajeParcial[a0_List,p_Integer]:=
Module[{a=a0,m,n,k=p},
  {m,n}=Dimensions[a];
  Do[If[Abs[a[[i,p]]]>Abs[a[[k,p]]],k=i},{i,p+1,m}];
  If[k!=p,{a[[k]],a[[p]]}={a[[p]],a[[k]]};a]
End[]

EndPackage[]

```

Como se puede observar, en este programa la posición del mayor elemento se consigue con un ciclo "Do", comenzando la búsqueda en la fila "p+1" y guardando la posición del mayor elemento en la variable "k". Como casi siempre sucede cuando se trabaja con Mathematica esta no es la única alternativa, así es posible obtener dicha posición con las funciones "Position" (que devuelve la posición de un elemento en un vector o matriz) y "Max" (que devuelve el valor máximo de un vector o matriz), por ejemplo, para buscar el mayor valor en la primera columna se escribe:

```

Position[Abs[a[[All,1]]],Max[Abs[a[[All,1]]]]]
{{3}}

```

Que devuelve igualmente el valor correcto (3). Aunque se complica un poco cuando se busca el mayor valor en otra columna, por ejemplo en la segunda, pues no se tiene que buscar en los elementos que se encuentran por encima del pivote:

```

Position[Abs[a[[2;;4,2]]],Max[Abs[a[[2;;4,2]]]]]
{{3}}

```

En este caso devuelve "3", pero como no se ha tomado en cuenta el primer elemento de la columna, esta posición corresponde en realidad a la posición "4". Es por esta razón que en el programa se ha optado por la estructura "Do" en lugar de las instrucciones "Position" y "Max".

Una vez creado el paquete importamos el mismo en un cuaderno:

```

Get["SisEcuas",Path->"E:\MAT205"]

```

Y probamos la función con la siguiente matriz:

```

a={{3,2,4,6,3},{2,2,6,4,1},{8,10,0,2,8},{1,-13,0,4,3}};

```

```

a//MatrixForm

```

$$\begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 2 & 2 & 6 & 4 & 1 \\ 8 & 10 & 0 & 2 & 8 \\ 1 & -13 & 0 & 4 & 3 \end{pmatrix}$$

Donde como se puede observar, el mayor elemento de la columna 1 (el número 8) se encuentra en la tercera fila. Mandando esta matriz a "PivotajeParcial" se obtiene:

```
PivotajeParcial[a,1]
{{8,10,0,2,8},{2,2,6,4,1},{3,2,4,6,3},{1,-13,0,4,3}}
```

```
%//MatrixForm
```

$$\begin{pmatrix} 8 & 10 & 0 & 2 & 8 \\ 2 & 2 & 6 & 4 & 1 \\ 3 & 2 & 4 & 6 & 3 \\ 1 & -13 & 0 & 4 & 3 \end{pmatrix}$$

Y como se puede ver el resultado es correcto, pues se ha intercambiado la fila 3 con la fila 1, quedando el mayor elemento en la posición pivote (la fila y columna 1). Si nuestro pivote fuera 2 ($p=2$), entonces al efectuar el pivotaje parcial, el mayor elemento, por debajo de la fila 2 (en este caso el número -13), debería pasar a la fila 2:

```
PivotajeParcial[a,2]
{{3,2,4,6,3},{1,-13,0,4,3},{8,10,0,2,8},{2,2,6,4,1}}
```

```
%//MatrixForm
```

$$\begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 1 & -13 & 0 & 4 & 3 \\ 8 & 10 & 0 & 2 & 8 \\ 2 & 2 & 6 & 4 & 1 \end{pmatrix}$$

Como efectivamente sucede. Si probamos ahora el programa con " $p=3$ " (para la columna 3), no debería existir ningún intercambio de filas, pues el único elemento por debajo de la posición pivote (fila y columna 3) es el número cero:

```
PivotajeParcial[a,3]
{{3,2,4,6,3},{2,2,6,4,1},{8,10,0,2,8},{1,-13,0,4,3}}
```

```
%//MatrixForm
```

$$\begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 2 & 2 & 6 & 4 & 1 \\ 8 & 10 & 0 & 2 & 8 \\ 1 & -13 & 0 & 4 & 3 \end{pmatrix}$$

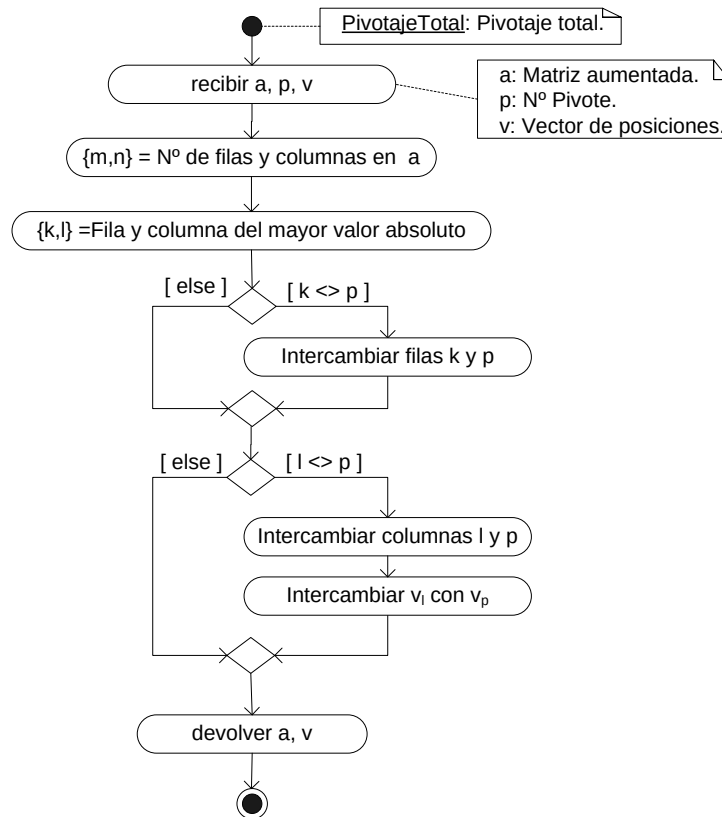
18.1.2.2. Pivotaje total

Para realizar el *pivotaje total* se debe ubicar la posición del elemento con el mayor valor absoluto buscando en las filas y columnas no reducidas. Una vez ubicada la posición del elemento con el mayor valor absoluto se efectúan intercambios de filas y columnas, según sea necesario, para llevar el mayor valor absoluto a la posición pivote.

En el pivotaje total, se debe hacer un seguimiento de las columnas intercambiadas, pues un intercambio de columna implica también un intercambio en la posición de los resultados, así por ejemplo si se intercambia la columna 2 con la columna 4, entonces el resultado de la variable x_2 se encontrará en la fila 4 (y no en la fila 2), mientras que el resultado de la variable x_4 se encontrará en la fila 2 (y no en la fila 4), esto se logra con un vector de posiciones, el cual inicialmente tiene números enteros ordenados ascendentemente desde 1 hasta el número de filas del sistema y que al final del

proceso tendrá el número de solución que se encuentra en cada una de las filas, así si el vector de posiciones tiene el número 2 en la posición 6 ($v[[6]]==2$), entonces sabemos que en la fila 6 se encuentra la solución correspondiente a la variable 2.

El diagrama de actividades del algoritmo para llevar a cabo el pivotaje total es el siguiente:



El programa elaborado en base al mismo, el cual, al igual que el pivotaje parcial, se añade al paquete "SisEcu", es el siguiente:

```

BeginPackage["SisEcu`"]
...
PivotajeTotal::usage="PivotajeTotal[a,p,v], realiza el pivotaje total
  de la matriz \"a\", comenzando en la fila y columna \"p\", registrando
  el intercambio de columnas en el vector de posiciones \"v\"."
...
Begin["Private`"]
...
PivotajeTotal[a0_List,p_Integer,v0_List]:=
  Module[{a=a0,v=v0,m,n,l=p,k=p},
    {m,n}=Dimensions[a];
    Do[If[Abs[a[[i,j]]]>Abs[a[[k,l]]],{k,l}={i,j},{i,p,m},{j,p,m}];
    If[k!=p,{a[[k]],a[[p]]}={a[[p]],a[[k]]}];
    If[l!=p,{a[[All,l]],a[[All,p]]}={a[[All,p]],a[[All,l]]};
    {v[[1]],v[[p]]}={v[[p]],v[[1]]};{a,v}
  ]
...
End[]
EndPackage[]
  
```

Ahora importamos el paquete:

```
Get["SisEcuac", Path->"E:\MAT205"]
```

Y probamos el programa con la misma matriz:

```
a={{3,2,4,6,3},{2,2,6,4,1},{8,10,0,0,8},{1,-13,0,0,3}};
```

```
%//MatrixForm
```

$$\begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 2 & 2 & 6 & 4 & 1 \\ 8 & 10 & 0 & 0 & 8 \\ 1 & -13 & 0 & 0 & 3 \end{pmatrix}$$

El vector de posiciones "v" contiene inicialmente números secuenciales del 1 al 4:

```
v={1,2,3,4};
```

Con "p=1", el programa debería ubicar el mayor elemento (-13) y como se encuentra en la cuarta fila y segunda columna, debería intercambiar las filas 1 con 4 y luego intercambiar las columnas 1 con 2, cambiando también los valores de los elementos 1 y 2 (columnas 1 y 2) del vector de posiciones:

```
{b,c}=PivotajeTotal[a,1,v]
```

```
{{{-13,1,0,0,3},{2,2,6,4,1},{10,8,0,0,8},{2,3,4,6,3}},{2,1,3,4}}
```

```
{b//MatrixForm, c//MatrixForm}
```

$$\left\{ \begin{pmatrix} -13 & 1 & 0 & 4 & 3 \\ 2 & 2 & 6 & 4 & 1 \\ 10 & 8 & 0 & 2 & 8 \\ 2 & 3 & 4 & 6 & 3 \end{pmatrix}, \begin{pmatrix} 2 \\ 1 \\ 3 \\ 4 \end{pmatrix} \right\}$$

Y como se puede ver eso es exactamente lo que ocurre. El vector de posiciones nos indica que en la primera fila se encuentra el segundo resultado y que en la segunda el primero.

Ahora probemos el programa con "p=2". Una vez más el mayor valor absoluto es -13, pero el mismo se encuentra ahora en la columna pivote (2), por lo que en este caso no se debería producir un intercambio de columnas sino solamente un intercambio de filas:

```
{b,c}=PivotajeTotal[a,2,v]
```

```
{{{3,2,4,6,3},{1,-13,0,0,3},{8,10,0,0,8},{2,2,6,4,1}},{1,2,3,4}}
```

```
{b//MatrixForm,c//MatrixForm}
```

$$\left\{ \begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 1 & -13 & 0 & 0 & 3 \\ 8 & 10 & 0 & 0 & 8 \\ 2 & 2 & 6 & 4 & 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \right\}$$

Y como se puede ver eso es lo que ocurre. Finalmente probemos el programa con "p=3". Puesto que los elementos a partir de la fila y columna 3 son todos cero (recuerde que no se toman en cuenta las columnas de las constantes) el programa no debería llevar a cabo ningún intercambio

```
{b,c}=PivotajeTotal[a,3,v]
```

```
{{{3,2,4,6,3},{2,2,6,4,1},{8,10,0,0,8},{1,-13,0,0,3}},{1,2,3,4}}
```

```
{b//MatrixForm,c//MatrixForm}
```

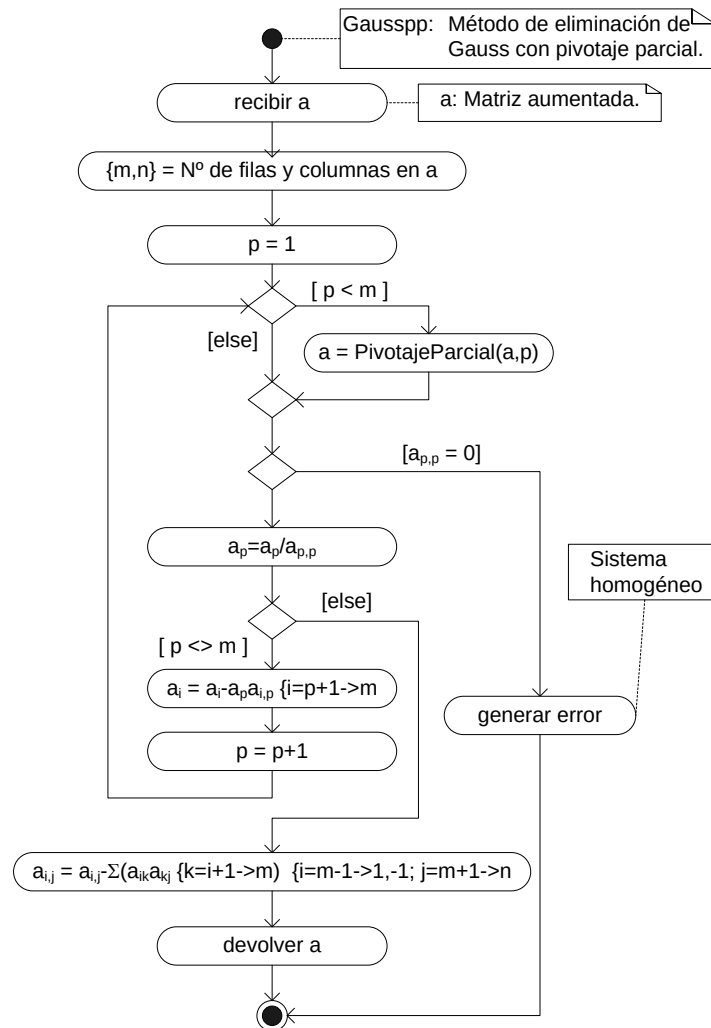
$$\left\{ \begin{pmatrix} 3 & 2 & 4 & 6 & 3 \\ 2 & 2 & 6 & 4 & 1 \\ 8 & 10 & 0 & 0 & 8 \\ 1 & -13 & 0 & 0 & 3 \end{pmatrix}, \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \end{pmatrix} \right\}$$

Una vez más vemos que eso es lo que ocurre, por lo que podemos asumir que el programa responde correctamente en todos los casos.

18.1.3. Eliminación de Gauss con pivotaje parcial

Ahora estamos en condiciones de elaborar el programa para el método de eliminación de Gauss con pivotaje parcial, que a más del pivotaje, básicamente se trata de aplicar repetitivamente las ecuaciones (4) y (5), hasta que el pivote "p" sea igual al número de filas "m". Posteriormente los resultados se calculan, por sustitución inversa, aplicando la ecuación (6).

El algoritmo para el método de Gauss con pivotaje parcial es el siguiente:



El código elaborado en base al mismo y añadido al paquete "SisEcua" es:

```

BeginPackage["SisEcua`"]
...
Gausspp::usage= "Gausspp[a], resuelve el sistema de ecuaciones lineales
  cuya matriz aumentada es \"a\", empleando el método de eliminación de
  Gauss con pivotaje parcial."
...
Begin["Private`"]
...

```

```
Gausspp[a0_List]:=
Module[{a=a0,m,n,k,l},
{m,n}=Dimensions[a];
Do[If[p<m,a=PivotajeParcial[a,p]];
If[a[[p,p]]==0,Message[SisLin::homogeneo]];
a[[p]]=a[[p]]/a[[p,p]];
If[p!=m,
Do[a[[i]]=a[[i]]-a[[p]]*a[[i,p]],{i,p+1,m}]],{p,m}];
Do[a[[i,j]]=a[[i,j]]-Sum[a[[i,k]]*a[[k,j]],{k,i+1,m}],
{i,m-1,1,-1},{j,m+1,n}];a]
End[]

EndPackage[]
```

Probamos este programa, importando el paquete en un cuaderno:

```
Get["SisEcua`",Path->"E:\MAT205"]
```

Y resolviendo el siguiente sistema de ecuaciones lineales:

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

```
Gausspp[a]
{{1,4,1,5},{0,1,0,1},{0,0,1,-2}}
```

```
%//MatrixForm
```

```

$$\begin{pmatrix} 1 & 4 & 1 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

```

Como se puede observar, se obtienen los mismos resultados que en el ejemplo manual. Las tres soluciones, que se encuentran en la columna 4, pueden ser extraídas con:

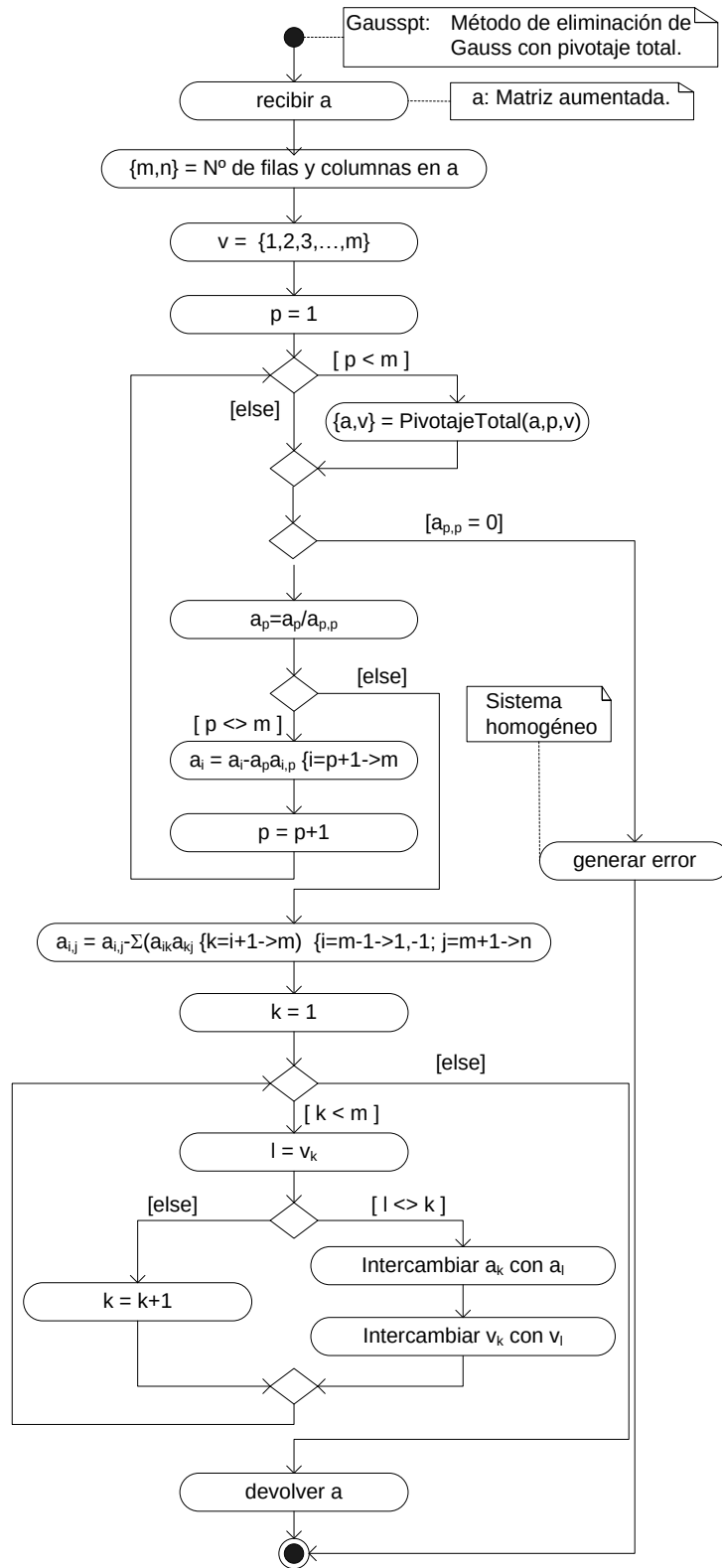
```
%[[All,4]]
{5,1,-2}
```

18.1.4. Eliminación de Gauss con pivotaje total

Podemos elaborar igualmente el módulo para el método de eliminación de Gauss con pivotaje total, donde además del tipo de pivotaje, la diferencia con el anterior programa está en la necesidad de ordenar los resultados en función al vector de posiciones (debido a los intercambios de columnas que se pueden dar con el pivotaje total).

El algoritmo del método de eliminación de Gauss con pivotaje total se presenta en la siguiente página. El programa elaborado en base al mismo y añadido al paquete "SisEcua" es:

```
BeginPackage["SisEcua`"]
...
Gausspt::usage= "Gausspt[a], resuelve el sistema de ecuaciones lineales
cuya matriz aumentada es \"a\", empleando el método de eliminación de
Gauss con pivotaje total."
...
Begin["Private`"]
...
Gausspt[a0_List]:=
Module[{a=a0,m,n,v,l,k=1},
{m,n}=Dimensions[a];
v=Range[m];
Do[If[p<m,{a,v}=PivotajeTotal[a,p,v]];
```



```

If[a[[p,p]]==0,Message[SisLin::homogeneo]];
a[[p]]=a[[p]]/a[[p,p]];
If[p!=m,
  Do[a[[i]]=a[[i]]-a[[p]]*a[[i,p]],{i,p+1,m}]],{p,m}];
Do[a[[i,j]]=a[[i,j]]-Sum[a[[i,k]]*a[[k,j]],{k,i+1,m}],

```

```

        {i,m-1,1,-1},{j,m+1,n}];
While[k<m,
  l=v[[k]];
  If[l!=k,
    {a[[k]],a[[l]]}={a[[l]],a[[k]]};
    {v[[k]],v[[l]]}={v[[l]],v[[k]]},
    k++];
  a]
...
End[]

EndPackage[]

```

Ahora importamos el paquete:

```
Get["SisEcuas",Path->"E:\MAT205"]
```

Y probamos ahora el programa volviendo a resolver el sistema de ecuaciones del ejemplo manual:

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

```
Gausspt[a]
{{0,0,1,5},{1,1/4,1/4,1},{0,1,1/5,-2}}
```

```
%//MatrixForm
```

$$\begin{pmatrix} 0 & 0 & 1 & 5 \\ 1 & \frac{1}{4} & \frac{1}{4} & 1 \\ 0 & 1 & \frac{1}{5} & -2 \end{pmatrix}$$

Como era de esperar obtenemos los mismos resultados que en ejemplo manual.

18.1.5. Ejemplos

1. Encuentre las soluciones del siguiente sistema de ecuaciones empleando el método de Gauss con pivotaje parcial y total.

$$\begin{aligned} 2x_2 &+ x_4 = 0 \\ 2x_1 + 2x_2 + 3x_3 + 2x_4 &= -2 \\ 4x_1 - 3x_2 + x_4 &= -7 \\ 6x_1 + x_2 - 6x_3 - 5x_4 &= 6 \end{aligned} \tag{8}$$

Primero importamos el paquete (si todavía no lo hemos hecho):

```
Get["SisEcuas",Path->"E:\MAT205"]
```

Escribimos la matriz aumentada:

```
a={{0,2,0,1,0},{2,2,3,2,-2},{4,-3,0,1,-7},{6,1,-6,-5,6}};
```

Y mandamos esta matriz a los programas "Gausspp" y "Gausspt", extrayendo del resultado la columna 5, que es donde se encuentran las soluciones:

```
Gausspp[a][[All,5]]
{-1/2,1,1/3,-2}
```

```
Gausspt[a][[All,5]]
{-1/2,1,1/3,-2}
```

Finalmente, podemos comprobar los resultados con "RowReduce":

```
RowReduce[a][[All,5]]
{-1/2,1,1/3,-2}
```

2. Resuelva, los siguientes sistemas de ecuaciones lineales con el método de eliminación de Gauss.

$$\begin{aligned}x+2y+3z &= 21 \\ 5x-9y+2z &= 12 \\ 3x+y-4z &= 15\end{aligned}$$

$$\begin{aligned}u+2v+3w &= 11 \\ 5u-9v+2w &= 2 \\ 3u+v-4w &= 4\end{aligned}\tag{9}$$

$$\begin{aligned}r+2s+3t &= 1 \\ 5r-9s+2t &= 2 \\ 3r+s-4t &= 3\end{aligned}$$

Para resolver dos o más sistemas de ecuaciones que sólo difieren en las constantes, se manda al método la matriz aumentada con las columnas de las constantes, tal como sucede con "RowReduce".

Primero escribimos la matriz aumentada:

```
a={{1,2,3,21,11,1},{5,-9,2,12,2,2},{3,1,-4,15,4,3}};
```

Y mandamos esta matriz a "Gausspp" y "Gausspt", extrayendo las tres últimas columnas que es donde se encuentran las soluciones de los tres sistemas de ecuaciones:

```
Gausspp[a][[All,4;;6]]
{{1311/182,20/7,149/182},{45/14,12/7,3/14},{447/182,11/7,-15/182}}
```

```
Gausspt[a][[All,4;;6]]
{{1311/182,20/7,149/182},{45/14,12/7,3/14},{447/182,11/7,-15/182}}
```

Finalmente podemos corroborar estos resultados con "RowReduce":

```
RowReduce[a][[All,4;;6]]
{{1311/182,20/7,149/182},{45/14,12/7,3/14},{447/182,11/7,-15/182}}
```

3. Encuentre la inversa de la matriz de los coeficientes del sistema de ecuaciones lineales del ejemplo 1 empleando el método de Gauss con pivotaje parcial y total.

Para encontrar la inversa de una matriz se procede igual que con "RowReduce" es decir se manda la matriz de los coeficientes aumentada con la matriz unidad.

Primero escribimos la matriz de los coeficientes:

```
a={{0,2,0,1},{2,2,3,2},{4,-3,0,1},{6,1,-6,-5}};
```

Aumentamos la matriz unidad:

```
b=ArrayFlatten[{{a,IdentityMatrix[4]}}]
{{0,2,0,1,1,0,0,0},{2,2,3,2,0,1,0,0},{4,-3,0,1,0,0,1,0},{6,1,-6,-5,0,0,0,1}}
```

Y mandamos la matriz aumentada a "Gausspp" y "Gausspt":

```
Gausspp[b][[All,5;;8]]
{{-1/39,5/39,7/78,5/78},{7/39,4/39,-5/39,2/39},
{-62/117,37/117,-17/117,-1/117},{25/39,-8/39,10/39,-4/39}}
```

```
Gausspt[b][[All,5;;8]]
{{-1/39,5/39,7/78,5/78},{7/39,4/39,-5/39,2/39},
{-62/117,37/117,-17/117,-1/117},{25/39,-8/39,10/39,-4/39}}
```

Finalmente corroboramos los resultados con "Inverse" y también con "RowReduce":

```
Inverse[a]
{{-1/39,5/39,7/78,5/78},{7/39,4/39,-5/39,2/39},
{-62/117,37/117,-17/117,-1/117},{25/39,-8/39,10/39,-4/39}}
```

```
RowReduce[b][[All,5;;8]]
{{-1/39,5/39,7/78,5/78},{7/39,4/39,-5/39,2/39},
{-62/117,37/117,-17/117,-1/117},{25/39,-8/39,10/39,-4/39}}
```

18.2. Método de eliminación de Gauss - Jordán

El método de eliminación de Gauss - Jordán, es muy parecido al método de eliminación de Gauss, sólo que en este caso la matriz de los coeficientes es transformada en una matriz identidad (o unidad). Así si aplicamos el método de eliminación de Gauss-Jordán al sistema de ecuaciones (3):

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4 \end{aligned} \quad (3)$$

Se transforma en:

$$\begin{aligned} x_1 + 0 + 0 + 0 &= c'_1 \\ 0 + x_2 + 0 + 0 &= c'_2 \\ 0 + 0 + x_3 + 0 &= c'_3 \\ 0 + 0 + 0 + x_4 &= c'_4 \end{aligned} \quad (10)$$

Donde como se puede observar las soluciones del sistema se encuentran en el vector de las constantes. En forma matricial, la aplicación del método de Gauss-Jordán a la matriz aumentada logra la siguiente transformación:

$$\left[\begin{array}{ccccc|c} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{array} \right] \xrightarrow{\text{GAUSS-JORDAN}} \left[\begin{array}{ccccc|c} 1 & 0 & 0 & 0 & a'_{1,5} \\ 0 & 1 & 0 & 0 & a'_{2,5} \\ 0 & 0 & 1 & 0 & a'_{3,5} \\ 0 & 0 & 0 & 1 & a'_{4,5} \end{array} \right] \quad (11)$$

Donde las soluciones se encuentran en la columna (o columnas) de las constantes (igual que sucede con la función "RowReduce").

Las operaciones que se efectúan para llevar la matriz aumentada a la matriz identidad son prácticamente las mismas que en el método de Gauss, excepto que ahora la eliminación de las columnas se la realiza no sólo en la parte inferior de la diagonal, sino también en la parte superior de la misma. En consecuencia para la reducción de las columnas aplicamos la ecuación (5) pero comenzando en 1 en lugar de p+1:

$$a_i = a_i - a_p * a_{i,p} \quad \{i=1 \rightarrow m, i \neq p\} \quad (12)$$

Para la eliminación de las filas se aplica la misma ecuación que en el método de Gauss (ecuación 4) y en este caso no se requiere la sustitución

inversa, pues los resultados se encuentran ya en la columna (o columnas) de las constantes.

Como en el método de Gauss, antes de efectuar la reducción de filas y columnas se debe realizar un pivotaje parcial o total para reducir los errores de redondeo y evitar divisiones entre cero.

18.2.1. Ejemplo manual

Para comprender mejor el método resolveremos el sistema de ecuaciones (7) con el método de Gauss-Jordán.

La matriz aumentada es:

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

Los números de filas y columnas son:

```
m=3;n=4;
```

El pivote inicialmente es 1:

```
p=1;
```

Primero reducimos la fila "p" (1), aplicando la ecuación (4):

```
a[[p]]=a[[p]]/a[[p,p]]
{1,4,1,7}
```

Luego reducimos la columna "p" (1), aplicando la ecuación (12):

```
Do[If[i#p,a[[i]]=a[[i]]-a[[p]]*a[[i,p]]],{i,1,m}]
```

```
a//MatrixForm
```

$$\begin{pmatrix} 1 & 4 & 1 & 7 \\ 0 & 2 & -2 & 6 \\ 0 & -9 & 0 & -9 \end{pmatrix}$$

Ahora incrementamos el pivote (2) y volvemos a reducir filas y columnas:

```
p++;
```

```
a[[p]]=a[[p]]/a[[p,p]]
{0,1,-1,3}
```

```
Do[If[i#p,a[[i]]=a[[i]]-a[[p]]*a[[i,p]]],{i,1,m}]
```

```
a//MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 5 & -5 \\ 0 & 1 & -1 & 3 \\ 0 & 0 & -9 & 18 \end{pmatrix}$$

Finalmente incrementamos el pivote (3) y volvemos a reducir filas y columnas:

```
p++;
```

```
a[[p]]=a[[p]]/a[[p,p]]
{0,0,1,-2}
```

```
Do[If[i#p,a[[i]]=a[[i]]-a[[p]]*a[[i,p]]],{i,1,m}]
```

```
a//MatrixForm
```

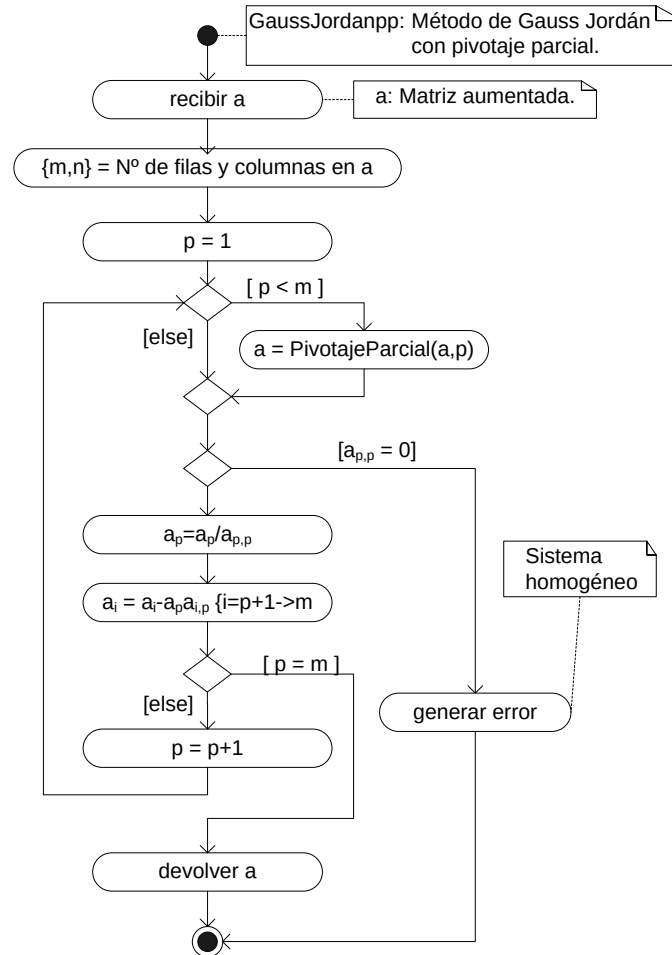
$$\begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

Como se puede observar, las soluciones quedan en la columna de las constantes, y la matriz de los coeficientes queda reducida a la matriz unidad, tal como sucede con "RowReduce".

18.2.2. Eliminación de Gauss- Jordán con pivotaje parcial

Al igual que en el método de Gauss, en el de Gauss-Jordán se debe emplear por lo menos pivotaje parcial para evitar divisiones entre cero y reducir así los errores de redondeo.

El algoritmo para resolver un sistema de ecuaciones lineales con pivotaje parcial se presenta en el siguiente diagrama de actividades:



El código elaborado en base a este algoritmo y añadido al paquete "SisE-cua" es el siguiente:

```
BeginPackage["SisEcua`"]
```

```
...
```

```
GaussJordanpp::usage= "GaussJordanpp[a], resuelve el sistema de
    ecuaciones lineales cuya matriz aumentada es \"a\", empleando el método
    de eliminación de Gauss Jordán con pivotaje parcial."
```

```
...
```

```
Begin["Private`"]
```

```
...
```

```
GaussJordanpp[a0_List] :=
    Module[{a=a0,m,n,l},
```

```

    {m,n}=Dimensions[a];
    Do[If[p<m,a=PivotajeParcial[a,p]];
      If[a[[p,p]]==0,Message[SisLin::homogeneo]];
      a[[p]]=a[[p]]/a[[p,p]];
      Do[If[i#p,a[[i]]=a[[i]]-a[[p]]*a[[i,p]]],{i,m}],{p,m}];
    a]
...
End[]

EndPackage[]

```

Podemos probar el programa resolviendo el sistema del ejemplo manual:

```

Get["SisEcu`",Path->"E:\MAT205"]

a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};

GaussJordanpp[a]
{{1,0,0,5},{0,1,0,1},{0,0,1,-2}}

```

```
%//MatrixForm
```

$$\begin{pmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -2 \end{pmatrix}$$

Y dado que se obtienen los resultados correctos, podemos asumir el programa ha sido correctamente elaborado.

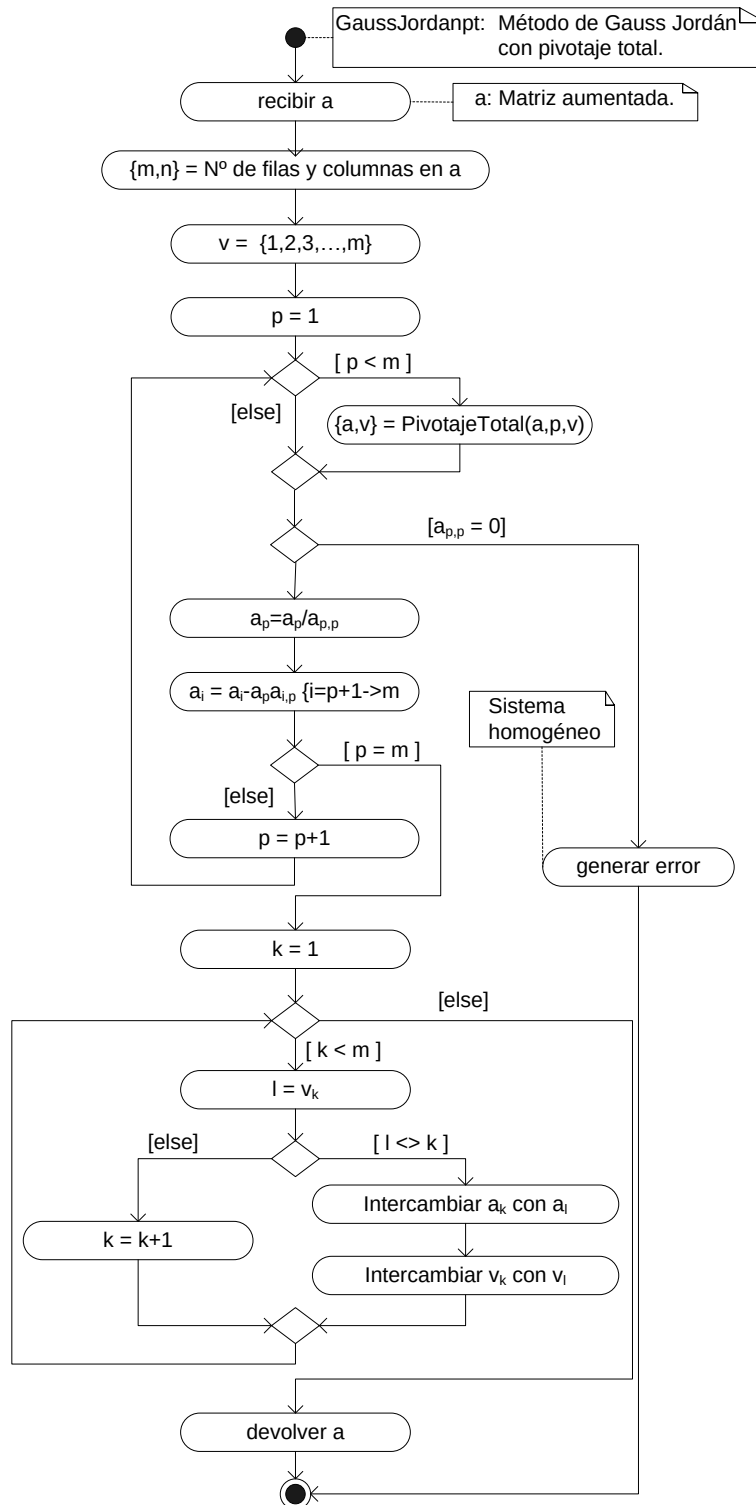
18.2.3. Eliminación de Gauss-Jordán con pivotaje total

El algoritmo del método de eliminación de Gauss Jordán con pivotaje total se presenta en la siguiente página. El código elaborado en base al mismo y añadido al paquete "SisEcu" es:

```

BeginPackage["SisEcu`"]
...
GaussJordanpt::usage= "GaussJordanpt[a], resuelve el sistema de
    ecuaciones lineales cuya matriz aumentada es \"a\", empleando el método
    de eliminación de Gauss Jordán con pivotaje total."
...
Begin["Private`"]
...
GaussJordanpt[a0_List]:=
    Module[{a=a0,m,n,v,l,k=1},
      {m,n}=Dimensions[a];
      v=Range[m];
      Do[If[p<m,{a,v}=PivotajeTotal[a,p,v]];
        If[a[[p,p]]==0,Message[SisLin::homogeneo]];
        a[[p]]=a[[p]]/a[[p,p]];
        Do[If[i#p,a[[i]]=a[[i]]-a[[p]]*a[[i,p]]],{i,m}],{p,m}];
      While[k<m,
        l=v[[k]];
        If[l!=k,
          {a[[k]],a[[l]]}={a[[l]],a[[k]]};
          {v[[k]],v[[l]]}={v[[l]],v[[k]]},
          k++];
      a]
...
End[]
EndPackage[]

```



Probamos el programa volviendo a resolver el sistema del ejemplo manual:

```
Get["SisEcuas", Path->"E:\MAT205"]
```

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

```
GaussJordanpt[a]
```

```
{{1,0,0,1},{0,1,0,-2},{0,0,1,5}}
```

```
%//MatrixForm
```

```
 $\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & -2 \\ 0 & 0 & 1 & 5 \end{pmatrix}$ 
```

Y como los resultados son correctos podemos asumir que el programa ha sido correctamente elaborado.

Como se puede observar, el uso del método de Gauss Jordán es el mismo que el de Gauss y RowReduce, por lo que se pueden resolver los mismos ejemplos que en el método de Gauss, sólo que en lugar de llamar a Gausspp o Gausspt, se llama a GaussJordanpp y GaussJordanpt respectivamente. Tan sólo para ilustrar este aspecto volveremos a resolver el ejemplo 1:

```
Get["SisEcu", Path->"E:\MAT205"]
```

```
a={{0,2,0,1,0},{2,2,3,2,-2},{4,-3,0,1,-7},{6,1,-6,-5,6}};
```

```
GaussJordanpp[a][[All,5]]
{-1/2,1,1/3,-2}
```

```
GaussJordanpt[a][[All,5]]
{-1/2,1,1/3,-2}
```

```
RowReduce[a][[All,5]]
{-1/2,1,1/3,-2}
```

18.3. Método de Cholesky

El método de Cholesky, al igual que en el método de eliminación de Gauss, transforma el sistema de ecuaciones lineales en un sistema triangular superior. Por consiguiente aplicando el método al sistema de ecuaciones (3):

$$\begin{aligned} a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + a_{1,4}x_4 &= c_1 \\ a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + a_{2,4}x_4 &= c_2 \\ a_{3,1}x_1 + a_{3,2}x_2 + a_{3,3}x_3 + a_{3,4}x_4 &= c_3 \\ a_{4,1}x_1 + a_{4,2}x_2 + a_{4,3}x_3 + a_{4,4}x_4 &= c_4 \end{aligned} \quad (3)$$

Se obtiene:

$$\begin{aligned} x_1 + a'_{1,2}x_2 + a'_{1,3}x_3 + a'_{1,4}x_4 &= c'_1 \\ 0 + x_2 + a'_{2,3}x_3 + a'_{2,4}x_4 &= c'_2 \\ 0 + 0 + x_3 + a'_{3,4}x_4 &= c'_3 \\ 0 + 0 + 0 + x_4 &= c'_4 \end{aligned} \quad (4)$$

En forma matricial, el método de Cholesky logra el siguiente cambio:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} & a_{1,5} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} & a_{2,5} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} & a_{3,5} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} & a_{4,5} \end{bmatrix} \xrightarrow{\text{CHOLESKY}} \begin{bmatrix} 1 & a'_{1,2} & a'_{1,3} & a'_{1,4} & a'_{1,5} \\ 0 & 1 & a'_{2,3} & a'_{2,4} & a'_{2,5} \\ 0 & 0 & 1 & a'_{3,4} & a'_{3,5} \\ 0 & 0 & 0 & 1 & a'_{4,5} \end{bmatrix} \quad (13)$$

Que es lo mismo que hace el método de Gauss y al igual que en dicho método, una vez que el sistema ha sido reducido a una matriz triangular superior, los resultados se calculan por sustitución inversa.

Sin embargo, la forma en que el sistema de ecuaciones es llevado a un sistema triangular superior es muy diferente en ambos métodos. En el método de Cholesky, el sistema triangular superior se calcula asumiendo que existen dos matrices, una triangular inferior (L) y una triangular superior (U) (que es la matriz que nos interesa calcular). Estas matrices tienen la propiedad de que al multiplicarlas reproducen la matriz aumentada original, es decir, para el anterior sistema se cumple que:

$$\begin{vmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{4,4} \end{vmatrix} * \begin{vmatrix} 1 & u_{12} & u_{13} & u_{14} & u_{15} \\ 0 & 1 & u_{23} & u_{24} & u_{25} \\ 0 & 0 & 1 & u_{34} & u_{35} \\ 0 & 0 & 0 & 1 & u_{4,5} \end{vmatrix} = \begin{vmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{4,4} & a_{4,5} \end{vmatrix} \quad (14)$$

Las relaciones entre las matrices U , L y la matriz original A , pueden deducirse fácilmente llevando a cabo la multiplicación e igualando los elementos. Las relaciones resultantes son las siguientes:

Para el cálculo de los elementos de la matriz L :

$$l_{ij} = a_{ip} - \sum_{k=1}^{p-1} l_{ik} * u_{kp} \quad \{i = p \rightarrow m\} \quad (15)$$

Donde al igual que en los anteriores métodos " p " es el número pivote y " m " es el número de ecuaciones (o filas).

Para el cálculo de los elementos de la matriz U se tiene la siguiente expresión:

$$u_{pj} = \frac{a_{pj} - \sum_{k=1}^{p-1} l_{pk} * u_{kj}}{l_{pp}} \quad \{j = p+1 \rightarrow n\} \quad (16)$$

Donde " n " es el número de columnas.

Como se puede observar, cuando el pivote es 1 ($p=1$), todos los valores de la columna 1 de la matriz L son iguales a los valores de la columna 1 de la matriz A . Igualmente cuando el pivote es 1, los elementos de la fila 1 de la matriz U se calculan directamente dividiendo los elementos de la primera fila de la matriz A entre el elemento l_{pp} .

Para las otras columnas y filas (comenzando con $p=2$) se calculan primero los elementos de la columna pivote de la matriz L y luego los elementos de la fila pivote de la matriz U . El proceso continúa hasta que el pivote es igual al número de ecuaciones ($p=m$).

Como no es necesario almacenar los ceros de la matriz L , ni los unos de la matriz U y dado que los elementos de la matriz A , sólo se emplean una vez, todos los valores pueden ser almacenados directamente en la matriz aumentada A , evitando así la necesidad de crear dos matrices adicionales. En consecuencia, las ecuaciones (18) y (19) pueden volver a ser escritas de la siguiente manera:

$$a_{ip} = a_{ip} - \sum_{k=1}^{p-1} a_{ik} * a_{kp} \quad \{i = p \rightarrow m\} \quad (17)$$

$$a_{pj} = \frac{a_{pj} - \sum_{k=1}^{p-1} a_{pk} * a_{kj}}{a_{pp}} \quad \{j = p+1 \rightarrow n\} \quad (18)$$

18.3.1. Ejemplo manual

Para comprender mejor el proceso de cálculo, y contar con valores de referencia que nos permitan elaborar y probar el programa, resolveremos paso a paso el sistema de ecuaciones (7).

La matriz aumentada es:

```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

El número de filas (m), columnas (n) y el número pivote son:

```
m=3;n=4;p=1;
```

Ahora, cuando "p" vale 1, aplicamos las ecuaciones (17) y (18):

```
Do[a[[i,p]]=a[[i,p]]-Sum[a[[i,k]]*a[[k,p]],{k,p-1}],{i,p,m}]
```

```
Do [a[[p,j]]=(a[[p,j]]-Sum[a[[p,k]]*a[[k,j]],{k,p-1}))/a[[p,p]],{j,p+1,n}]
```

```
a//MatrixForm
```

$$\begin{pmatrix} 2 & 4 & 1 & 7 \\ 1 & 6 & -1 & 13 \\ 2 & -1 & 2 & 5 \end{pmatrix}$$

Entonces incrementamos el valor de "p" (p=2) y volvemos a aplicar las ecuaciones (17) y (18):

```
p++;
```

```
Do[a[[i,p]]=a[[i,p]]-Sum[a[[i,k]]*a[[k,p]],{k,p-1}],{i,p,m}]
```

```
Do [a[[p,j]]=(a[[p,j]]-Sum[a[[p,k]]*a[[k,j]],{k,p-1}))/a[[p,p]],{j,p+1,n}]
```

```
a//MatrixForm
```

$$\begin{pmatrix} 2 & 4 & 1 & 7 \\ 1 & 2 & -1 & 3 \\ 2 & -9 & 2 & 5 \end{pmatrix}$$

Incrementamos una vez más el valor de "p" (p=3) y volvemos a aplicar las ecuaciones (17) y (18):

```
p++;
```

```
Do[a[[i,p]]=a[[i,p]]-Sum[a[[i,k]]*a[[k,p]],{k,p-1}],{i,p,m}]
```

```
Do [a[[p,j]]=(a[[p,j]]-Sum[a[[p,k]]*a[[k,j]],{k,p-1}))/a[[p,p]],{j,p+1,n}]
```

```
a//MatrixForm
```

$$\begin{pmatrix} 2 & 4 & 1 & 7 \\ 1 & 2 & -1 & 3 \\ 2 & -9 & -9 & -2 \end{pmatrix}$$

Finalmente calculamos las soluciones por sustitución inversa, aplicando la ecuación (6):

```
Do[a[[i,j]]=a[[i,j]]-Sum[a[[i,k]]*a[[k,j]],{k,i+1,m}],  
{i,m-1,1,-1},{j,m+1,n}]
```

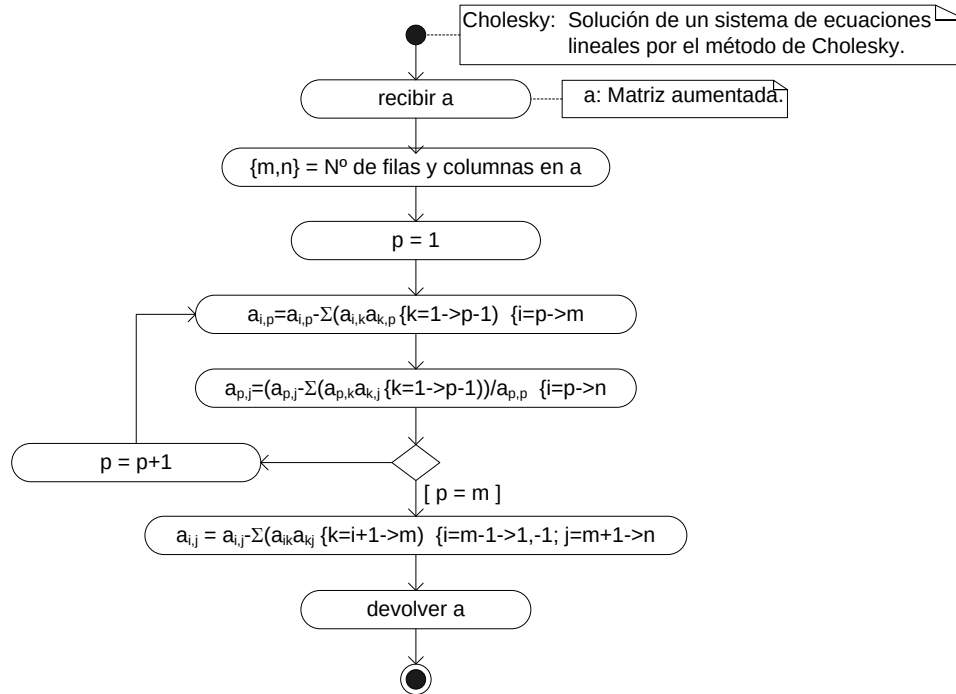
```
a//MatrixForm
```

$$\begin{pmatrix} 2 & 4 & 1 & 5 \\ 1 & 2 & -1 & 1 \\ 2 & -9 & -9 & -2 \end{pmatrix}$$

Al igual que en los métodos de Gauss y Gauss Jordán, las soluciones quedan en la columna de las constantes.

18.3.2. Programa para el método de Cholesky

El algoritmo del método de Cholesky, donde básicamente se aplican las ecuaciones (18) y (19) desde "p=1" hasta "m" y posteriormente se calculan las soluciones por sustitución inversa, aplicando la ecuación (6), es el siguiente:



El código del programa elaborado siguiendo el algoritmo y añadido al paquete "SisEcua" es:

```

BeginPackage["SisEcua`"]
...
Cholesky::usage= "Cholesky[a], resuelve el sistema de ecuaciones lineales
  cuya matriz aumentada es \"a\", empleando el método de Cholesky."
...
Begin["Private`"]
...
Cholesky[a0_List]:=
  Module[{a=a0,m,n},
    {m,n}=Dimensions[a];
    Do[
      Do[a[[i,p]]=a[[i,p]]-Sum[a[[i,k]]*a[[k,p]],
        {k,p-1}],{i,p,m}];
      Do[a[[p,j]]=(a[[p,j]]-Sum[a[[p,k]]*a[[k,j]],{k,p-1}))/
        a[[p,p]],{j,p+1,n}],
      {p,m}];
    Do[a[[i,j]]=a[[i,j]]-Sum[a[[i,k]]*a[[k,j]],{k,i+1,m}],
      {i,m-1,1,-1},{j,m+1,n}];
    a]
End[]

EndPackage[]
  
```

Podemos probar este módulo resolviendo el sistema de ecuaciones (1):

```
Get["SisEcua`",Path->"E:\HPV\MAT205\1-2009\CUADERNOS MATHEMATICA"]
```



```
a={{2,8,2,14},{1,6,-1,13},{2,-1,2,5}};
```

```
Cholesky[a]
{{2,4,1,5},{1,2,-1,1},{2,-9,-9,-2}}
```

```
%//MatrixForm
```

```

$$\begin{pmatrix} 2 & 4 & 1 & 5 \\ 1 & 2 & -1 & 1 \\ 2 & -9 & -9 & -2 \end{pmatrix}$$

```

Y como se puede observar se obtienen los mismos resultados que en el ejemplo manual, por lo que podemos asumir que el programa ha sido correctamente elaborado.

La resolución de ecuaciones con el método de *Cholesky* es igual que con los métodos de *Gauss* y *Gauss-Jordán*, con la única diferencia de que en lugar de llamar a *Gausspp*, *Gausspt*, *GaussJordanpp* o *GaussJordanpt* se llama a *Cholesky*, por lo tanto los ejemplos resueltos en el método de Gauss son válidos también para este método, sólo se debe cuidar que el primer elemento de la matriz no sea cero (en cuyo caso deberá llevarse a cabo un pivotaje parcial) para que no se produzca un error por división entre cero. Para ilustrar lo anterior volveremos a resolver el ejemplo 1 del método de Gauss:

```
Get["SisEcuas", Path->"E:\MAT205"]
```

```
a={{0,2,0,1,0},{2,2,3,2,-2},{4,-3,0,1,-7},{6,1,-6,-5,6}};
```

```
a=PivotajeParcial[a,1]
{{6,1,-6,-5,6},{2,2,3,2,-2},{4,-3,0,1,-7},{0,2,0,1,0}}
```

```
Cholesky[a][[All,5]]
{-1/2,1,1/3,-2}
```

```
RowReduce[a][[All,5]]
{-1/2,1,1/3,-2}
```

18.3.3. Preguntas y ejercicios

1. ¿Cómo queda la matriz aumentada cuando se aplica el método de eliminación de Gauss?
2. ¿Cómo queda la matriz aumentada cuando se aplica el método de eliminación de Gauss Jordán?
3. ¿En qué se diferencia la reducción de columnas entre los métodos de Gauss y Gauss Jordán?
4. ¿En qué consiste la sustitución inversa?
5. ¿En qué consiste el pivotaje parcial?
6. ¿En qué consiste el pivotaje total?
7. ¿Por qué es necesario llevar a cabo por lo menos un pivotaje parcial en los métodos de Gauss y Gauss Jordán?
8. ¿Cómo queda la matriz aumentada cuando se aplica el método de Cholesky?
9. ¿Qué se debe hacer en el método de Cholesky cuando el primer coeficiente del sistema es cero?
10. ¿Por qué no es necesaria la sustitución inversa en el método de Gauss Jordán?

11. Elabore un programa que lleve a cabo el pivoteaje parcial empleando la estructura For en lugar de Do.
12. Elabore un programa que lleve a cabo el pivoteaje parcial empleando la estructura While en lugar de Do.
13. Elabore un programa que lleve a cabo el pivoteaje total empleando la estructura Nest en lugar de Do.
14. Elabore un programa que lleve a cabo el pivoteaje total empleando la estructura Which en lugar de If.
15. Elabore un programa para el método de Gauss con pivoteaje parcial empleando la estructura For en lugar de Do.
16. Elabore un programa para el método de Gauss con pivoteaje total empleando la estructura Nest en lugar de Do.
17. Elabore un programa para el método de Gauss Jordán con pivoteaje parcial empleando la estructura While en lugar de Do.
18. Elabore un programa para el método de Gauss Jordán con pivoteaje total empleando la estructura For en lugar de While.
19. Elabore un programa para el método de Cholesky empleando la estructura For en lugar de Do.
20. Elabore un programa para el método de Cholesky empleando la estructura While en lugar de Do.
21. Resuelva el siguiente sistema de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolve, RowReduce e Inverse.

$$10x_1 + x_2 + 2x_3 = 44$$

$$2x_1 + 10x_2 + x_3 = 51$$

$$x_1 + 2x_2 + 10x_3 = 61$$

22. Resuelva el siguiente sistema de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolveFunction, RowReduce e Inverse.

$$3x_1 - x_2 + 2x_3 = 12$$

$$x_1 + 2x_2 + 3x_3 = 11$$

$$2x_1 - 2x_2 - x_3 = 2$$

23. Resuelva el siguiente sistema de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolve, RowReduce e Inverse.

$$2x_1 - 2x_2 + 5x_3 = 13$$

$$2x_1 + 3x_2 + 4x_3 = 20$$

$$3x_1 - x_2 + 3x_3 = 10$$

24. Resuelva el siguiente sistema de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolveFunction, RowReduce e Inverse.

$$x_1 + x_2 + x_3 + x_4 = -1$$

$$4x_1 + 5x_2 + 6x_3 + 7x_4 = 0$$

$$6x_1 + 10x_2 + 15x_3 + 21x_4 = 0$$

$$12x_1 + 30x_2 + 60x_3 + 105x_4 = 0$$

25. Resuelva el siguiente sistema de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolve, RowReduce e Inverse.

$$\begin{aligned}
 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\
 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\
 x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\
 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \\
 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \\
 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24
 \end{aligned}$$

26. Resuelva, al mismo tiempo, los siguientes sistemas de ecuaciones empleando los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con LinearSolveFunction, RowReduce e Inverse.

$$\begin{aligned}
 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\
 x_1 + 4x_2 + 2x_4 &= 0 \\
 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\
 x_1 + x_2 - x_3 + 3x_4 &= 0
 \end{aligned}$$

$$\begin{aligned}
 3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\
 y_1 + 4y_2 + 2y_4 &= 2 \\
 2y_1 + y_2 + 2y_3 - y_4 &= 3 \\
 y_1 + y_2 - y_3 + 3y_4 &= 4
 \end{aligned}$$

$$\begin{aligned}
 3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\
 z_1 + 4z_2 + 2z_4 &= 2 \\
 2z_1 + z_2 + 2z_3 - z_4 &= 1 \\
 z_1 + z_2 - z_3 + 3z_4 &= 0
 \end{aligned}$$

27. Resuelva, al mismo tiempo, los siguientes sistemas de ecuaciones, calculando la matriz inversa con los métodos de Gauss, Gauss Jordán y Cholesky, y corroborando los resultados con Inverse.

$$\begin{aligned}
 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\
 x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 7 \\
 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 7 \\
 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 3 \\
 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 4 \\
 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 4
 \end{aligned}$$

$$\begin{aligned}
 30y_1 + y_2 + 2y_3 + 6y_4 + 5y_5 - 3y_6 &= 14 \\
 y_1 + 2y_2 + 33y_3 + 5y_4 - y_5 + 2y_6 &= 1 \\
 3y_1 + 5y_2 + 7y_3 - 45y_4 + 4y_5 + y_6 &= 3 \\
 4y_1 + 3y_2 + 2y_3 + 3y_4 + 40y_5 - 4y_6 &= 2 \\
 5y_1 + 6y_2 - 3y_3 - 4y_4 + 3y_5 + 36y_6 &= 2 \\
 2y_1 + 25y_2 + y_3 - y_4 + 2y_5 - 5y_6 &= 7
 \end{aligned}$$

$$\begin{aligned}
 30z_1 + z_2 + 2z_3 + 6z_4 + 5z_5 - 3z_6 &= 4 \\
 z_1 + 2z_2 + 33z_3 + 5z_4 - z_5 + 2z_6 &= 27 \\
 3z_1 + 5z_2 + 7z_3 - 45z_4 + 4z_5 + z_6 &= 27 \\
 4z_1 + 3z_2 + 2z_3 + 3z_4 + 40z_5 - 4z_6 &= 13 \\
 5z_1 + 6z_2 - 3z_3 - 4z_4 + 3z_5 + 36z_6 &= 14 \\
 2z_1 + 25z_2 + z_3 - z_4 + 2z_5 - 5z_6 &= 14
 \end{aligned}$$

19. JACOBI Y GAUSS SEIDEL

19.1. Método de Jacobi

Cuando el número de ecuaciones en un sistema es elevado (con cientos o miles de ecuaciones) los métodos de eliminación estudiados hasta ahora no devuelven los resultados correctos y ello sucede no porque los métodos fallen, sino porque se produce un error por propagación del error: los resultados intermedios se emplean para calcular otros y estos a su vez para calcular otros y así sucesivamente hasta llegar a los resultados finales y dado que en todos estos cálculos se producen errores de redondeo, dichos errores se propagan y cuando el número de ecuaciones es muy elevado el error acumulado es tan grande que los resultados son erróneos.

En estos casos se debe recurrir a los métodos iterativos, en los cuales partiendo de valores iniciales asumidos se calculan en iteraciones sucesivas nuevos valores hasta que las igualdades del sistema se cumplen con cierto margen de error. De esta manera se evita la propagación del error y el error permitido se fija de antemano.

Uno de los métodos iterativos más sencillos es el de *Jacobi*, que conceptualmente es equivalente al método de sustitución directa, sólo que se emplea para resolver sistemas de ecuaciones lineales en lugar de una ecuación no lineal. En este método se asumen valores iniciales para todas las incógnitas, usualmente dichos valores son simplemente ceros. Empleando los valores asumidos se calculan nuevos valores para las incógnitas empleando la primera ecuación del sistema para calcular el valor de la primera incógnita, la segunda ecuación para la segunda incógnita y así sucesivamente. Entonces, con los valores calculados, se verifica si se cumplen las igualdades del sistema (es decir si las ecuaciones se igualan a cero), de ser así el proceso concluye y se tienen ya las soluciones del sistema (los valores calculados), caso contrario los valores calculados se convierten en valores asumidos y se repite el proceso.

Para terminar el proceso iterativo, en lugar de verificar si las igualdades se cumplen (la exactitud), se pueden comparar también los valores calculados en dos iteraciones sucesivas (la precisión): si dichos valores son casi iguales en un determinado número de dígitos el proceso concluye, caso contrario el proceso se repite.

19.1.1. Ejemplo manual

Para comprender mejor el método y contar con valores de referencia para el programa, resolvamos el siguiente sistema de 3 ecuaciones lineales:

$$\begin{array}{rrcr} 10x_1 & + & x_2 & + & 2x_3 & = & 44 \\ 2x_1 & + & 10x_2 & + & x_3 & = & 51 \\ x_1 & + & 2x_2 & + & 10x_3 & = & 61 \end{array} \quad (1)$$

Como no contamos con valores iniciales conocidos, nuestros valores iniciales serán todos iguales a cero (que como se dijo es lo más usual en este método):

```
{x1,x2,x3}=ConstantArray[0,3];
```

Con los valores asumidos calculamos nuevos valores, a los cuales llamaremos "y", empleando la primera ecuación para calcular el primer valor (y_1), la segunda para el segundo valor (y_2) y la tercera para el tercero (y_3):

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
```

```
{4.4,5.1,6.1}
```

Donde se ha empleado la función "N" simplemente para mostrar los resultados en forma de números reales.

Ahora verificamos si con los valores calculados se obtienen los resultados correctos (es decir si se cumplen las ecuaciones):

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{61.3,65.9,75.6}
```

Como era de esperar, para la primera iteración, los resultados están todavía lejos de los correctos: 44, 51 y 61. Y aunque a simple vista se ve que los valores iniciales asumidos (0, 0, 0) no son iguales a los calculados (4.4, 5.1, 6.1), comprobamos también ese hecho verificando si su división es igual a 1:

```
{x1/y1,x2/y2,x3/y3} //N
{0.,0.,0.}
```

Como no sucede así, se repite el proceso, pero empleando como valores de prueba los valores calculados:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{2.67,3.61,4.64}
```

Y verificamos si se cumplen las ecuaciones o si los dos últimos valores calculados son aproximadamente iguales:

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{39.59,46.08,56.29}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.64794,1.41274,1.31466}
```

Si bien ahora los resultados se aproximan más a los correctos y los valores calculados y asumidos no son tan diferentes, todavía existe mucho error, por lo que el proceso debe ser repetido:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.111,4.102,5.111}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{45.434,52.353,62.425}
```

```
{x1/y1,x2/y2,x3/y3} //N
{0.858245,0.880059,0.907846}
```

Como aún no se cumplen las ecuaciones y los valores prueba y calculados no son iguales, repetimos el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{2.9676,3.9667,4.9685}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.5797,50.5707,60.586}
```

```
{x1/y1,x2/y2,x3/y3} //N
```

```
{1.04832,1.03411,1.02868}
```

Si bien los resultados se aproximan más a los correctos y los valores calculados y asumidos son cada vez más cercanos, todavía difieren bastante (son sólo iguales en el primer dígito), por lo que es necesario repetir el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.00963,4.00963,5.0099}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.1257,51.1255,61.1279}
```

```
{x1/y1,x2/y2,x3/y3} //N
{0.986035,0.989293,0.991736}
```

Como aún los resultados difieren se repite el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{2.99706,3.99708,4.99711}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.9619,50.9621,60.9623}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.0042,1.00314,1.00256}
```

Como los resultados siguen difiriendo se repite el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.00087,4.00088,5.00088}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.0113,51.0114,61.0114}
```

```
{x1/y1,x2/y2,x3/y3} //N
{0.99873,0.999052,0.999247}
```

Como los resultados siguen difiriendo se repite el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{2.99974,3.99974,4.99974}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.9966,50.9966,60.9966}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.00038,1.00028,1.00023}
```

Como los resultados siguen difiriendo se repite el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.00008,4.00008,5.00008}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.001,51.001,61.001}
```

```
{x1/y1,x2/y2,x3/y3} //N
{0.999886,0.999915,0.999932}
```

Ahora ya se tiene una precisión de entre 3 y 4 dígitos, pero aún no se tienen los resultados exactos por lo que el proceso se repite:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{2.99998,3.99998,4.99998}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.9997,50.9997,60.9997}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.00003,1.00003,1.00002}
```

Ahora la precisión es de 4 dígitos. Repitiendo el proceso resulta:

```
{x1,x2,x3}={y1,y2,y3};
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.00001,4.00001,5.00001}
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.0001,51.0001,61.0001}
{x1/y1,x2/y2,x3/y3} //N
{0.99999,0.999992,0.999994}
```

Donde como se puede ver la precisión mejora a 5 dígitos. Repitiendo el proceso una vez más se obtiene:

```
{x1,x2,x3}={y1,y2,y3};
{y1=(44-x2-2*x3)/10,y2=(51-2*x1-x3)/10,y3=(61-x1-2*x2)/10} //N
{3.,4.,5.}
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.,51.,61.}
{x1/y1,x2/y2,x3/y3} //N
{1.,1.,1.}
```

Ahora los resultados son exactos y los valores calculados y asumidos son iguales, por lo que el proceso concluye. Aún en un sistema tan simple como el del ejemplo se puede apreciar la principal desventaja de los métodos numéricos: el elevado número de repeticiones. Es por ello que los métodos numéricos sólo son de utilidad práctica si el proceso se automatiza y ejecuta mediante un programa de computadora.

Al ser un proceso iterativo es probable que en ocasiones el método no converja y que en consecuencia no se puedan encontrar los resultados, por lo que es necesario incluir un contador en el programa para terminar el proceso si el número de iteraciones sobrepasa cierto límite.

La convergencia de este método está asegurada para sistemas con una *diagonal dominante*. Un sistema de ecuaciones lineales tiene una diagonal dominante cuando los valores absolutos de los elementos de la diagonal principal son mayores a la suma de los valores absolutos de los otros coeficientes de la misma fila. Por ejemplo el sistema resuelto es un sistema diagonal dominante porque 10 es mayor a $1+2=3$ en la primera fila, 10 es mayor a $2+1=3$ en la segunda fila y 10 es mayor a $1+2=3$ en la tercera fila.

19.1.2. Ecuación general

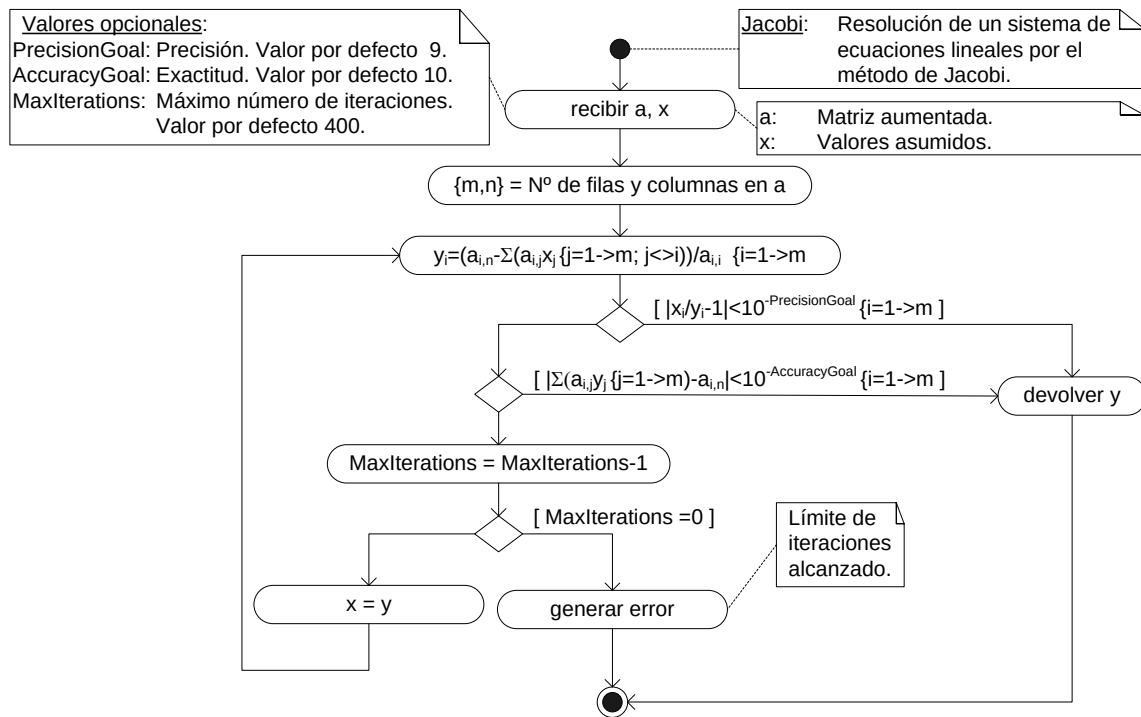
La ecuación general para el cálculo de los nuevos valores de "x", deducida en base al ejemplo manual, es la siguiente:

$$y_i = \frac{a_{i,n} - \sum_{j=1}^m a_{ij} * x_j}{a_{ii}} \quad \{j \neq i\} \quad \{i=1 \rightarrow m\} \quad (2)$$

Donde los elementos de la matriz "a" corresponden a la matriz aumentada del sistema de ecuaciones lineales.

19.1.3. Programa para el método de Jacobi

El algoritmo para resolver un sistema de ecuaciones lineales con el método de *Jacobi*, el cual básicamente consiste en la aplicación repetitiva de la ecuación (21), se presenta en el siguiente diagrama de actividades:



Como se puede observar en el algoritmo, el método concluye cuando se cumple una de tres condiciones: a) Que todos los valores calculados en las dos últimas iteraciones sean iguales en un determinado número de dígitos (precisión); b) Que todas las ecuaciones del sistema de un resultado aproximadamente igual a cero (exactitud) y c) Que se haya alcanzado el límite de iteraciones, en cuyo caso el proceso concluye generando un error pues no se ha logrado encontrar los resultados buscados.

El código elaborado en base al algoritmo y añadido al paquete "SisEcu" es el siguiente:

```
BeginPackage["SisEcu`"]
```

```
...
```

```
Jacobi::usage= "Jacobi[a,x], resuelve el sistema de ecuaciones lineales  

  cuya matriz aumentada es \"a\", con los valores asumidos \"x\",  

  empleando el método de Jacobi."
```

```

...
Begin["Private`"]
...
Options[Jacobi]={PrecisionGoal->9,AccuracyGoal->10,
  MaxIterations->400};

Jacobi::maxit = "Jacobi ha llegado al límite de iteraciones: `1`, sin
  encontrar los resultados";

Jacobi[a0_List,x0_List,OptionsPattern[]]:=
  Block[{a=a0,x=x0,y=x0,m,n,pg,ag,mi},
    {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
      MaxIterations}];
    {m,n}=Dimensions[a];
    Catch[Do[
      Do[y[[i]]=(a[[i,n]]-Sum[If[j!=i,a[[i,j]]*x[[j]],0},{j,m}))/
        a[[i,i]],{i,m}];
      Do[If[Abs[x[[i]]/y[[i]]-1]>10^-pg,Goto[1]],{i,m}]; Throw[y];
      Label[1];
      Do[If[Abs[Sum[a[[i,j]]*y[[j]],{j,m}]-a[[i,n]]]>10^-ag ,Goto[2]],
        {i,m}];Throw[y];
      Label[2];x=y,{mi}];
      Message[Jacobi::maxit,mi];Throw[y]]]
...
End[]

EndPackage[]

  Para probar el programa volvemos a resolveremos el sistema de ecuaciones
  (1), empleando ceros como valores asumidos:

Get["SisEcua`",Path->"E:\MAT205"]

a={{10,1,2,44},{2,10,1,51},{1,2,10,61}};

x=ConstantArray[0,3];

Jacobi[a,x]
{150000000002324503251/5000000000000000000,400000000004649065551/10000000000000
0000000,500000000004649065551/10000000000000000000}

  Como se puede observar los resultados son devueltos en forma de números
  exactos (porque todos los coeficientes del sistema son exactos). En forma de
  números reales (aproximados) los resultados son:

%/N
{3.,4.,5.}

  Que como se puede ver son los mismos resultados obtenidos en el ejemplo
  manual.

```

19.2. Método de Gauss - Seidel

El método de Gauss - Seidel es muy similar al método de Jacobi, con la diferencia de que los nuevos valores se calculan empleando siempre los últimos valores calculados y no los inicialmente asumidos (como ocurre en el método de *Jacobi*). Así la primera incógnita se calcula con los valores asumidos, pero la segunda se calcula con el valor calculado para la primera incógnita y los valores asumidos, la tercera se calcula con los valores cal-

culados para la primera y segunda incógnita y los valores asumidos, procediendo así hasta calcular todas las incógnitas.

19.2.1. Ejemplo manual

Para comprender mejor el método de Gauss - Seidel y contar con valores de referencia para el programa, volveremos a resolver el sistema de ecuaciones lineales (1):

$$\begin{array}{rrcr} 10x_1 & + & x_2 & + & 2x_3 & = & 44 \\ 2x_1 & + & 10x_2 & + & x_3 & = & 51 \\ x_1 & + & 2x_2 & + & 10x_3 & = & 61 \end{array} \quad (1)$$

Al igual que en Jacobi, emplearemos valores iniciales iguales a cero:

```
{x1,x2,x3}=ConstantArray[0,3];
```

Calculamos nuevos valores empleando para la primera incógnita: x_2 y x_3 , para la segunda: y_1 y x_3 , y para la tercera: y_1 y y_2 :

```
{y1=(44-x2-2*x3)/10,y2=(51-2*y1-x3)/10,y3=(61-y1-2*y2)/10} //N
{4.4,4.22,4.816}
```

Comprobamos si con los nuevos valores calculados (valores de "y") se cumplen las igualdades de la ecuación (1) o si los valores calculados son aproximadamente iguales a los supuestos:

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{57.852,55.816,61.}
```

```
{x1/y1,x2/y2,x3/y3} //N
{0.,0.,0.}
```

Como no es así se repite el proceso empleando como nuevos valores de prueba los valores calculados:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*y1-x3)/10,y3=(61-y1-2*y2)/10} //N
{3.0148,4.01544,4.99543}
```

Y volvemos a verificar si se cumplen las ecuaciones o si los valores calculados son aproximadamente iguales a los supuestos:

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.1543,51.1794,61.}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.45947,1.05094,0.964081}
```

Si bien ahora está más cerca, las igualdades todavía no se cumplen, por lo que es necesario repetir el proceso:

```
{x1,x2,x3}={y1,y2,y3};
```

```
{y1=(44-x2-2*x3)/10,y2=(51-2*y1-x3)/10,y3=(61-y1-2*y2)/10} //N
{2.99937,4.00058,4.99995}
```

```
{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.9942,51.0045,61.}
```

```
{x1/y1,x2/y2,x3/y3} //N
{1.00514,1.00371,0.999097}
```

Ahora las soluciones están más cerca a las correctas, pero aún no son exactas, por lo que repetimos el proceso:

```
{x1,x2,x3}={y1,y2,y3};

{y1=(44-x2-2*x3)/10,y2=(51-2*y1-x3)/10,y3=(61-y1-2*y2)/10} //N
{2.99995,4.00001,5.}

{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{43.9995,51.0001,61.}

{x1/y1,x2/y2,x3/y3} //N
{0.999806,1.00014,0.999989}
```

Repetimos el proceso una vez más:

```
{x1,x2,x3}={y1,y2,y3};

{y1=(44-x2-2*x3)/10,y2=(51-2*y1-x3)/10,y3=(61-y1-2*y2)/10} //N
{3.,4.,5.}

{10*y1+y2+2*y3,2*y1+10*y2+y3,y1+2*y2+10*y3} //N
{44.,51.,61.}

{x1/y1,x2/y2,x3/y3} //N
{0.999985,1.,1.}
```

Ahora los resultados son prácticamente exactos, por lo que podemos concluir el proceso, siendo las soluciones: $x_1=3$; $x_2=4$; $x_3=5$.

Como se puede observar, el método de Gauss - Seidel requiere menos iteraciones que el método de Jacobi. No obstante el método de Jacobi puede ser programado en computadoras con múltiples procesadores (como las computadoras actuales con 2, cuatro y hasta 8 núcleos).

19.2.2. Ecuación General

La ecuación general para calcular los nuevos valores en el método de Gauss - Seidel es:

$$y_i = \frac{a_{i,n} - \sum_{j=1}^{i-1} a_{ij} * y_j - \sum_{j=i+1}^m a_{ij} * x_j}{a_{ii}} \quad \{i=1 \rightarrow m\} \quad (3)$$

Donde "a" son los coeficientes de la matriz aumentada, "x" el vector con los valores asumidos, "y" el vector con los valores calculados, "m" es el número de ecuaciones en el sistema (filas de la matriz aumentada) y "n" el número de columnas (m+1).

Si inicialmente los valores asumidos se asignan al vector "y", entonces los nuevos valores pueden ser calculados empleando únicamente el vector "y", si se procede de esa manera la ecuación (22) toma la forma:

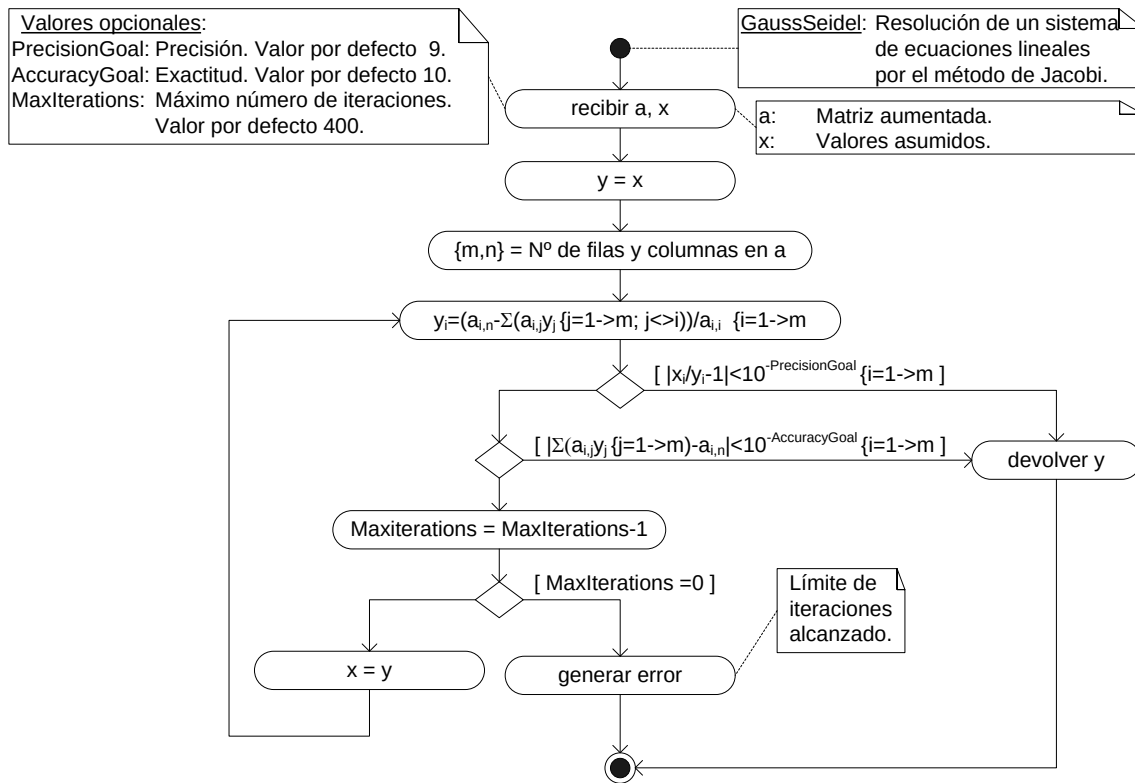
$$y_i = \frac{a_{i,n} - \sum_{j=1}^m a_{ij} * y_j}{a_{ii}} \quad \{j \neq i\} \quad \{i=1 \rightarrow m\} \quad (4)$$

19.2.3. Programa para el método de Gauss-Seidel

Como se puede deducir de la forma de las ecuaciones (22) y (23), el algoritmo para los métodos de Jacobi y Gauss-Seidel es esencialmente el mismo,

sólo que en el método de *Gauss-Seidel* los nuevos valores se calculan siempre empleando el vector "y" en lugar del vector "x".

El diagrama de actividades para este método es el siguiente:



El programa elaborado en base al mismo y añadido al paquete "SisEcu" es el siguiente:

```
BeginPackage["SisEcu`"]
```

```
...
```

```
GaussSeidel::usage= "GaussSeidel[a,x], resuelve el sistema de ecuaciones  
lineales cuya matriz aumentada es \"a\", con los valores asumidos \"x\",  
empleando el método de Gauss Seidel."
```

```
...
```

```
Begin["Private`"]
```

```
...
```

```
Options[GaussSeidel]={PrecisionGoal->9,AccuracyGoal->10,  
MaxIterations->400};
```

```
GaussSeidel::maxit = "GaussSeidel ha llegado al límite de iteraciones: `1`,  
sin encontrar los resultados";
```

```
GaussSeidel[a0_List,x0_List,OptionsPattern[]]:=
Block[{a=a0,x=x0,y=x0,m,n,pg,ag,mi},
{pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,  
MaxIterations}];
{m,n}=Dimensions[a];
Catch[Do[
Do[y[[i]]=(a[[i,n]]-Sum[If[j!=i,a[[i,j]]*y[[j]],0},{j,m}))/  
a[[i,i]],{i,m}];
Do[If[Abs[x[[i]]/y[[i]]-1]>10^-pg,Goto[1]],{i,m}]; Throw[y];
Label[1];
```

```

Do[If[Abs[Sum[a[[i,j]]*y[[j]],{j,m}]-a[[i,n]]]>10^-ag ,Goto[2]],
  {i,m}];Throw[y];
Label[2];x=y,{mi}];
Message[GaussSeidel::maxit,mi];Throw[y]]
...
End[]
...
EndPackage[]

Probamos el programa volviendo a resolver el sistema de ecuaciones (1):
Get["SisEcua`",Path->"E:\MAT205"]

a={{10,1,2,44},{2,10,1,51},{1,2,10,61}};

x=ConstantArray[0,3];

GaussSeidel[a,x]
{14648437500066307589/4882812500000000000,48828124999775758923/1220703125000
0000000,1220703125000565426363/24414062500000000000}

%/N
{3.,4.,5.}

```

Y como se puede ver, esencialmente se obtienen los mismos resultados que con el método de Jacobi.

19.2.4. Preguntas y ejercicios

1. ¿Por qué es necesario asumir valores iniciales para los métodos de Jacobi y Gauss Seidel?
2. ¿Cuándo termina el proceso iterativo en los métodos de Jacobi y Gauss Seidel?
3. ¿Qué tipo de matriz asegura la convergencia en los métodos de Jacobi y Gauss Seidel?
4. ¿Cuál es la diferencia entre los métodos de Jacobi y Gauss Seidel?
5. Elabore un programa para el método de Jacobi empleando la estructura While para el ciclo externo y la estructura For para los internos.
6. Elabore un programa para el método de Jacobi empleando la recursividad para el ciclo externo y la estructura While para los internos.
7. Elabore un programa para el método de Gauss Seidel empleando la estructura Nest tanto para el ciclo externo como para los internos.
8. Elabore un programa para el método de Gauss Seidel empleando la estructura NestWhile para el ciclo externo y la estructura For para el ciclo interno.
9. Resuelva el siguiente sistema de ecuaciones lineales por los métodos de *Jacobi* y *Gauss Seidel* con 12 dígitos de precisión, sin importar la exactitud. Corrobore los resultados con *LinearSolve*, *RowReduce*, *Inverse*, *Gaussppp*, *Gausspt*, *GaussJordanppp*, *GaussJordanpt* y *Cholesky*.

$$\begin{array}{rrcrrcrl}
 5x_1 & + & 22x_2 & + & x_3 & = & 66 \\
 15x_1 & + & 3x_2 & + & 2x_3 & = & 52 \\
 2x_1 & + & 4x_2 & + & 31x_3 & = & 82
 \end{array}$$

10. Resuelva el siguiente sistema de ecuaciones lineales por los métodos de *Jacobi* y *Gauss Seidel* con 12 dígitos de exactitud, sin importar la precisión. Corrobore los resultados con *LinearSolve*, *RowReduce*, *Inverse*, *Gausspp*, *Gausspt*, *GaussJordanpp*, *GaussJordanpt* y *Cholesky*.

$$\begin{array}{rcccccccccccl} 2x_1 & + & 25x_2 & + & x_3 & - & x_4 & + & 2x_5 & - & 5x_6 & = & 24 \\ 30x_1 & + & x_2 & + & 2x_3 & + & 6x_4 & + & 5x_5 & - & 3x_6 & = & 12 \\ 4x_1 & + & 3x_2 & + & 2x_3 & + & 3x_4 & + & 40x_5 & - & 4x_6 & = & 23 \\ x_1 & + & 2x_2 & + & 33x_3 & + & 5x_4 & - & x_5 & + & 2x_6 & = & 17 \\ 5x_1 & + & 6x_2 & - & 3x_3 & - & 4x_4 & + & 3x_5 & + & 36x_6 & = & 24 \\ 3x_1 & + & 5x_2 & + & 7x_3 & - & 45x_4 & + & 4x_5 & + & x_6 & = & 37 \end{array}$$

20. ECUACIONES LINEALES TRIDIAGONALES

20.1. Sistemas tridiagonales

Con frecuencia en la resolución de problemas en el campo de la ingeniería se forman sistemas de ecuaciones lineales que sólo tienen valores en la diagonal principal y a ambos lados de la misma, tales sistemas se conocen con el nombre de *sistemas tridiagonales* y tienen la siguiente forma general:

$$\begin{array}{cccccccccccccccc}
 a_{1,2}x_1 & + & a_{1,3}x_2 & + & 0x_3 & + & 0x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{1,n} \\
 a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & 0x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{2,n} \\
 0x_1 & + & a_{3,2}x_2 & + & a_{3,3}x_3 & + & a_{3,4}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{3,n} \\
 0x_1 & + & 0x_2 & + & a_{4,3}x_3 & + & a_{4,4}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{4,n} \\
 0x_1 & + & 0x_2 & + & 0x_3 & + & a_{5,4}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{5,n} \\
 \dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots \\
 0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & \dots & + & a_{m,m-1}x_{m-1} & + & a_{m,m}x_m & = & a_{m,n}
 \end{array} \quad (1)$$

Donde, como en los anteriores sistemas "m" es el número de filas (o incógnitas) y "n" es el número de columnas. Dado que en estos sistemas la mayoría de los elementos son cero es un desperdicio de memoria y tiempo almacenarlos y realizar operaciones con ellos. Es más eficiente y económico almacenar y trabajar únicamente con los elementos diferentes de cero: los elementos de la diagonal principal, los adyacentes a la misma y la columna de las constantes:

$$\begin{array}{cccccccccccccccc}
 a_{1,2}x_1 & + & a_{1,3}x_2 & + & 0x_3 & + & 0x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{1,4} \\
 a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & 0x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{2,4} \\
 0x_1 & + & a_{3,1}x_2 & + & a_{3,2}x_3 & + & a_{3,3}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{3,4} \\
 0x_1 & + & 0x_2 & + & a_{4,1}x_3 & + & a_{4,2}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{4,4} \\
 0x_1 & + & 0x_2 & + & 0x_3 & + & a_{5,1}x_4 & + & \dots & + & 0x_{m-1} & + & 0x_m & = & a_{5,4} \\
 \dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots & & \dots \\
 0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & + & \dots & + & a_{m,1}x_{m-1} & + & a_{m,2}x_m & = & a_{m,4}
 \end{array} \quad (2)$$

De esta manera se almacenan $m \cdot 4$ elementos en lugar de $m \cdot n$, lográndose así un gran ahorro de memoria y tiempo pues sin los ceros el sistema es:

$$\begin{array}{ccccccc}
 0x_0 & + & a_{1,2}x_1 & + & a_{1,3}x_2 & = & a_{1,4} \\
 a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & = & a_{2,4} \\
 a_{3,1}x_2 & + & a_{3,2}x_3 & + & a_{3,3}x_4 & = & a_{3,4} \\
 a_{4,1}x_3 & + & a_{4,2}x_4 & + & a_{4,3}x_5 & = & a_{4,4} \\
 a_{5,1}x_4 & + & a_{5,2}x_5 & + & a_{5,3}x_6 & = & a_{5,4} \\
 \dots & & \dots & & \dots & & \dots \\
 a_{m,1}x_{m-1} & + & a_{m,2}x_m & + & 0x_{m+1} & = & a_{m,4}
 \end{array} \quad (3)$$

Como se puede apreciar mejor en esta forma, el sistema involucra en realidad $m+2$ variables, pero dos de ellas: x_0 y x_{m+1} son conocidas, por lo que el sistema en realidad consta de "m" ecuaciones y "m" incógnitas.

En Mathematica los sistemas tridiagonales se resuelven empleando "LinearSolve", pero la matriz tridiagonal debe ser introducida como una matriz dispersa, empleando por ejemplo las funciones "SparseArray" y "Band". Así para resolver el siguiente sistema de ecuaciones lineales (que es un sistema tridiagonal):

$$\begin{aligned}
 2x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 &= 6 \\
 x_1 + 4x_2 + x_3 + 0x_4 + 0x_5 &= 36 \\
 0x_1 + x_2 + 4x_3 + x_4 + 0x_5 &= 72 \\
 0x_1 + 0x_2 + x_3 + 4x_4 + x_5 &= 108 \\
 0x_1 + 0x_2 + 0x_3 + x_4 + 2x_5 &= 66
 \end{aligned}
 \tag{4}$$

Primero guardamos las tres diagonales y la columna de la constante en cuatro variables: d1, d2, d3 y b:

```
{d1,d2,d3,b}={{1,1,1,1},{2,4,4,4,2},{1,1,1,1},{6,36,72,108,66}};
```

Construimos la matriz dispersa correspondiente a la matriz de los coeficientes:

```
a=SparseArray[{Band[{2,1}]→d1,Band[{1,1}]→d2,Band[{1,2}]→d3}]
SparseArray[< 13 >,{5,5}]
```

Para ver la matriz creada la mostramos en su forma normal:

```
Normal[a]//MatrixForm
```

$$\begin{pmatrix}
 2 & 1 & 0 & 0 & 0 \\
 1 & 4 & 1 & 0 & 0 \\
 0 & 1 & 4 & 1 & 0 \\
 0 & 0 & 1 & 4 & 1 \\
 0 & 0 & 0 & 1 & 2
 \end{pmatrix}$$

Finalmente mandamos la matriz dispersa de los coeficientes y el vector de las constantes a "LinearSolve":

```
LinearSolve[a,b]
{0,6,12,18,24}
```

Por lo tanto las soluciones son: $x_1=0$, $x_2=6$, $x_3=12$, $x_4=18$ y $x_5=24$.

Todos los métodos estudiados en los dos temas anteriores pueden ser adaptados para resolver sistemas tridiagonales, trabajando sólo con los coeficientes de la ecuación (1). De dichos métodos, en este tema modificaremos los métodos de *Gauss* y *Gauss-Seidel*, por ser los que mejor se adaptan a esta forma de ecuaciones.

20.1.1. Método de Gauss para sistemas tridiagonales

Para deducir las ecuaciones generales a emplear en el método de *Gauss* cuando se resuelven sistemas tridiagonales, aplicaremos el método a un sistema tridiagonal de 5 ecuaciones con 5 incógnitas.

Con el fin de facilitar la aplicación del método escribiremos también los ceros:

$$\begin{aligned}
 a_{1,2}x_1 + a_{1,3}x_2 + 0x_3 + 0x_4 + 0x_5 &= a_{1,4} \\
 a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + 0x_4 + 0x_5 &= a_{2,4} \\
 0x_1 + a_{3,1}x_2 + a_{3,2}x_3 + a_{3,3}x_4 + 0x_5 &= a_{3,4} \\
 0x_1 + 0x_2 + a_{4,1}x_3 + a_{4,2}x_4 + a_{4,3}x_5 &= a_{4,4} \\
 0x_1 + 0x_2 + 0x_3 + a_{5,1}x_4 + a_{5,2}x_5 &= a_{5,4}
 \end{aligned}
 \tag{5}$$

Recordemos que al aplicar el método de *Gauss* este sistema debe ser llevado a la forma triangular superior:

$$\begin{aligned}
a_{1,2}x_1 + a_{1,3}x_2 + 0x_3 + 0x_4 + 0x_5 &= a_{1,4} \\
0x_1 + a_{2,2}x_2 + a_{2,3}x_3 + 0x_4 + 0x_5 &= a_{2,4} \\
0x_1 + 0x_2 + a_{3,2}x_3 + a_{3,3}x_4 + 0x_5 &= a_{3,4} \\
0x_1 + 0x_2 + 0x_3 + a_{4,2}x_4 + a_{4,3}x_5 &= a_{4,4} \\
0x_1 + 0x_2 + 0x_3 + 0x_4 + a_{5,2}x_5 &= a_{5,4}
\end{aligned} \tag{6}$$

Por lo tanto sólo debemos convertir en cero los elementos que se encuentran debajo de la diagonal principal. Así para convertir en cero el elemento $a_{2,1}$, restamos a la segunda fila la primera multiplicada por $a_{2,1}/a_{1,2}$. Como no es necesario almacenar ni calcular los ceros, se puede aprovechar el elemento $a_{2,1}$ para almacenar el valor de esta división y con el mismo calcular los nuevos valores de los elementos $a_{2,2}$ y $a_{2,4}$ (el valor de $a_{2,3}$ no cambia):

$$\begin{aligned}
a_{2,1} &= \frac{a_{2,1}}{a_{1,2}} \\
a_{2,2} &= a_{2,2} - a_{2,1} * a_{1,3} \\
a_{2,4} &= a_{2,4} - a_{2,1} * a_{1,4}
\end{aligned}$$

De la misma forma, para convertir el elemento $a_{3,1}$ en cero restamos a la tercera fila la segunda fila multiplicada por $a_{3,1}/a_{2,2}$, al igual que antes almacenando este valor en el elemento $a_{3,1}$:

$$\begin{aligned}
a_{3,1} &= \frac{a_{3,1}}{a_{2,2}} \\
a_{3,2} &= a_{3,2} - a_{3,1} * a_{2,3} \\
a_{3,4} &= a_{3,4} - a_{3,1} * a_{2,4}
\end{aligned}$$

De manera similar, las operaciones para la cuarta y quinta fila son:

$$\begin{aligned}
a_{4,1} &= \frac{a_{4,1}}{a_{3,2}} \\
a_{4,2} &= a_{4,2} - a_{4,1} * a_{3,3} \\
a_{4,4} &= a_{4,4} - a_{4,1} * a_{3,4} \\
a_{5,1} &= \frac{a_{5,1}}{a_{4,2}} \\
a_{5,2} &= a_{5,2} - a_{5,1} * a_{4,3} \\
a_{5,4} &= a_{5,4} - a_{5,1} * a_{4,4}
\end{aligned}$$

Analizando estas ecuaciones se pueden deducir fácilmente la forma general de las ecuaciones para reducir a cero los elementos que se encuentran debajo de la diagonal principal:

$$\left. \begin{aligned}
a_{i,1} &= \frac{a_{i,1}}{a_{i-1,2}} \\
a_{i,2} &= a_{i,2} - a_{i,1} * a_{i-1,3} \\
a_{i,4} &= a_{i,4} - a_{i,1} * a_{i-1,4}
\end{aligned} \right\} i = 2 \rightarrow m \tag{7}$$

Una vez que el sistema ha quedado reducido a la forma (6) (aplicando la ecuación 7), los resultados se calculan por sustitución inversa y pueden ser almacenados en la última columna (la columna 4). Así el valor de la última variable (x_5) se calcula con:

$$x_5 = a_{5,4} = \frac{a_{5,4}}{a_{5,2}}$$

El valor de la penúltima variable (x_4) con:

$$x_4 = a_{4,4} = \frac{a_{4,4} - a_{4,3}x_5}{a_{4,2}} = \frac{a_{4,4} - a_{4,3}a_{5,4}}{a_{4,2}}$$

De manera similar las variables x_3 , x_2 y x_1 se calculan con:

$$x_3 = a_{3,4} = \frac{a_{3,4} - a_{3,3}x_4}{a_{3,2}} = \frac{a_{3,4} - a_{3,3}a_{4,4}}{a_{3,2}}$$

$$x_2 = a_{2,4} = \frac{a_{2,4} - a_{2,3}x_3}{a_{2,2}} = \frac{a_{2,4} - a_{2,3}a_{3,4}}{a_{2,2}}$$

$$x_1 = a_{1,4} = \frac{a_{1,4} - a_{1,3}x_2}{a_{1,2}} = \frac{a_{1,4} - a_{1,3}a_{2,4}}{a_{1,2}}$$

Analizando estas ecuaciones, podemos deducir la forma general de las ecuaciones que nos permiten calcular los resultados por sustitución inversa:

$$\left. \begin{aligned} x_m &= a_{m,4} = \frac{a_{m,4}}{a_{m,2}} \\ x_i &= a_{i,4} = \frac{a_{i,4} - a_{i,3}x_{i+1}}{a_{i,2}} = \frac{a_{i,4} - a_{i,3}a_{i+1,4}}{a_{i,2}} \end{aligned} \right\} \quad i = m-1 \rightarrow 1 \quad (8)$$

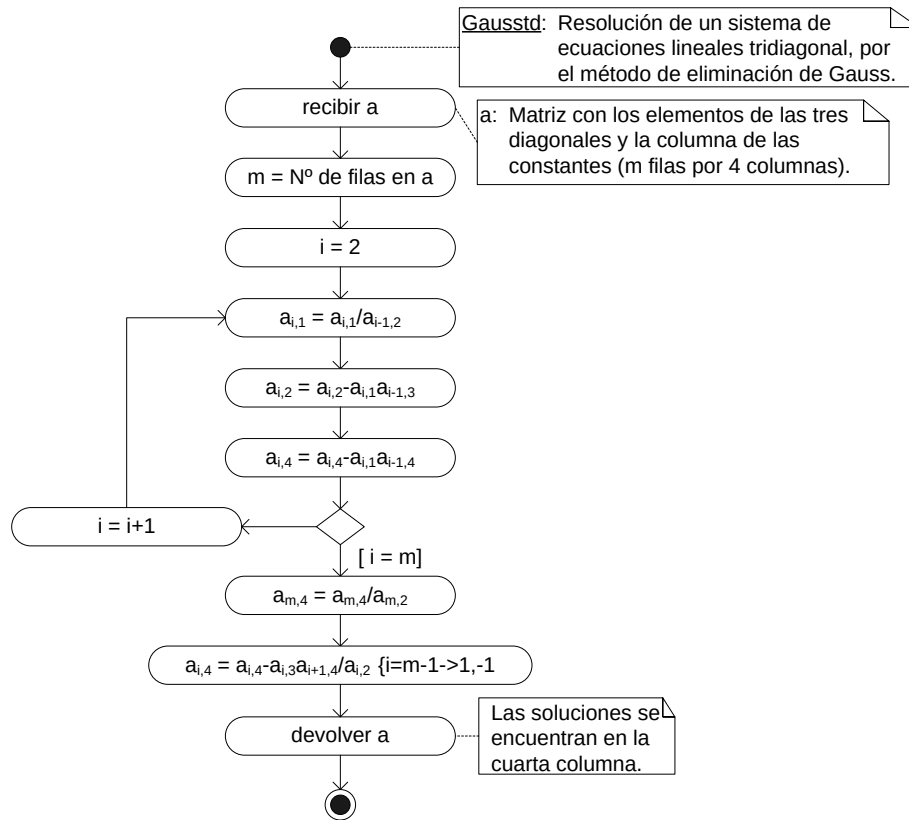
20.1.1.1. Programa para resolver sistemas tridiagonales por el método de eliminación de Gauss.

El algoritmo para resolver sistemas tridiagonales con el método de Gauss aplicando las ecuaciones (7) y (8), se presenta en la siguiente página. El código elaborado y añadido al paquete "SisEcu" es el siguiente:

```
BeginPackage["SisEcu`"]
...
Gausstd::usage="Gausstd[a], resuelve un sistema de ecuaciones lineales
tridiagonal por el método de Gauss, donde \"a\" es la matriz con los
elementos de las tres diagonales (en las tres primeras columnas) y
las constantes en la cuarta."

Begin["Private`"]
...
Gausstd[a0_List]:=
Module[{a=a0,m},
m=Length[a];
Do[a[[i,1]]=a[[i,1]]/a[[i-1,2]];
a[[i,2]]=a[[i,2]]-a[[i,1]]*a[[i-1,3]];
a[[i,4]]=a[[i,4]]-a[[i,1]]*a[[i-1,4]],
{i,2,m}];
a[[m,4]]=a[[m,4]]/a[[m,2]];
Do[a[[i,4]]=(a[[i,4]]-a[[i,3]]*a[[i+1,4]])/a[[i,2]],
{i,m-1,1,-1}];
a]

End[]
EndPackage[]
```



Probamos el programa resolviendo el siguiente sistema de ecuaciones (4), para ello primero importamos el paquete "SisEcua":

```
Get["SisEcua`", Path->"E:\MAT205"]
```

Ahora creamos la matriz con los elementos de las diagonales y las constantes. Dicha matriz puede ser escrita directamente:

```
a={{0,2,1,6},{1,4,1,36},{1,4,1,72},{1,4,1,108},{1,2,0,66}}
{{0,2,1,6},{1,4,1,36},{1,4,1,72},{1,4,1,108},{1,2,0,66}}
```

```
%//MatrixForm
```

```

$$\begin{pmatrix} 0 & 2 & 1 & 6 \\ 1 & 4 & 1 & 36 \\ 1 & 4 & 1 & 72 \\ 1 & 4 & 1 & 108 \\ 1 & 2 & 0 & 66 \end{pmatrix}$$

```

O armada a partir de los vectores de las diagonales y las constantes, las cuales recordemos que ya fueron guardadas con la siguiente instrucción.

```
{d1,d2,d3,b}={{1,1,1,1},{2,4,4,4,2},{1,1,1,1},{6,36,72,108,66}};
```

```
a=Transpose[ArrayFlatten[{Prepend[d1,0],d2,Append[d3,0],b}]]
{{0,2,1,6},{1,4,1,36},{1,4,1,72},{1,4,1,108},{1,2,0,66}}
```

```
%//MatrixForm
```

```

$$\begin{pmatrix} 0 & 2 & 1 & 6 \\ 1 & 4 & 1 & 36 \\ 1 & 4 & 1 & 72 \\ 1 & 4 & 1 & 108 \\ 1 & 2 & 0 & 66 \end{pmatrix}$$

```

Entonces mandamos esta matriz a "Gausstd":

```
Gausstd[a][[All,4]]
{0,6,12,18,24}
```

Y como se puede ver, se obtienen los mismos resultados que con "Linear-Solve".

20.1.2. Método de Gauss - Seidel para sistemas tridiagonales

Como generalmente los sistemas tridiagonales tienen además una diagonal dominante, el método de Gauss - Seidel resulta conveniente para resolver estos sistemas (recordemos que el método de Gauss Seidel converge cuando la matriz tiene una diagonal dominante).

Al igual que en el método de Gauss emplearemos el sistema tridiagonal de 5 ecuaciones (ecuación 5) para deducir las ecuaciones generales del método:

$$\begin{aligned}
 a_{1,2}x_1 + a_{1,3}x_2 + 0x_3 + 0x_4 + 0x_5 &= a_{1,4} \\
 a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + 0x_4 + 0x_5 &= a_{2,4} \\
 0x_1 + a_{3,1}x_2 + a_{3,2}x_3 + a_{3,3}x_4 + 0x_5 &= a_{3,4} \\
 0x_1 + 0x_2 + a_{4,1}x_3 + a_{4,2}x_4 + a_{4,3}x_5 &= a_{4,4} \\
 0x_1 + 0x_2 + 0x_3 + a_{5,1}x_4 + a_{5,2}x_5 &= a_{5,4}
 \end{aligned} \tag{5}$$

Al igual que se hizo con el método general de Gauss-Seidel se trabajará directamente con el vector de valores calculados "y", donde inicialmente "y" deberá contener los valores asumidos. Entonces los 5 nuevos valores se calculan con:

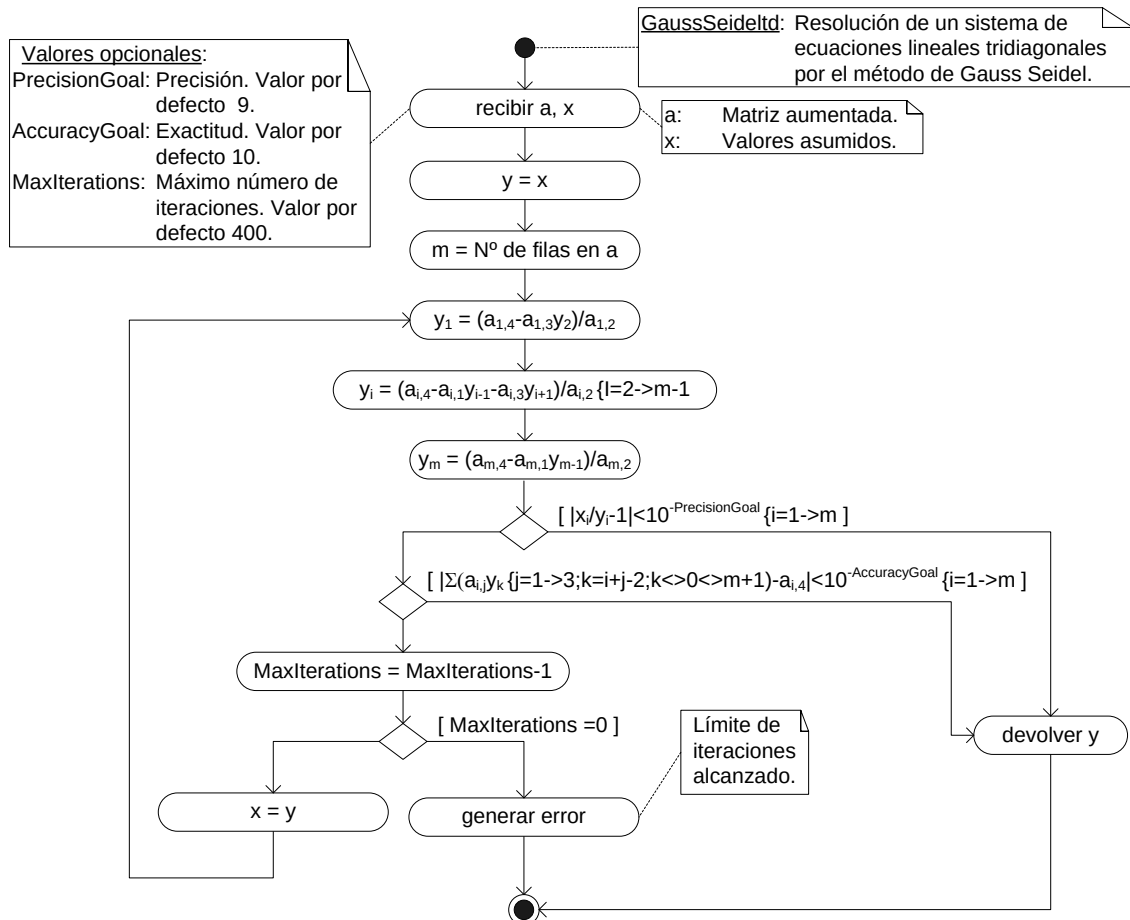
$$\begin{aligned}
 y_1 &= \frac{a_{1,4} - a_{1,3}y_2}{a_{1,2}} \\
 y_2 &= \frac{a_{2,4} - a_{2,1}y_1 - a_{2,3}y_3}{a_{2,2}} \\
 y_3 &= \frac{a_{3,4} - a_{3,1}y_2 - a_{3,3}y_4}{a_{3,2}} \\
 y_4 &= \frac{a_{4,4} - a_{4,1}y_3 - a_{4,3}y_5}{a_{4,2}} \\
 y_5 &= \frac{a_{5,4} - a_{5,1}y_4}{a_{5,2}}
 \end{aligned}$$

Analizando estas ecuaciones se pueden deducir las ecuaciones generales del método de Gauss-Seidel para sistemas tridiagonales:

$$\begin{aligned}
 y_1 &= \frac{a_{1,4} - a_{1,3}y_2}{a_{1,2}} \\
 y_i &= \frac{a_{i,4} - a_{i,1}y_{i-1} - a_{i,3}y_{i+1}}{a_{i,2}} \quad \left. \vphantom{y_i} \right\} \quad i = 2 \rightarrow m-1 \\
 y_m &= \frac{a_{m,4} - a_{m,1}y_{m-1}}{a_{m,2}}
 \end{aligned} \tag{9}$$

20.1.2.1. Programa para resolver sistemas tridiagonales con el método de Gauss - Seidel

El algoritmo para resolver sistemas tridiagonales por el método de Gauss Seidel, es el siguiente:



El programa elaborado en base a este algoritmo y añadido al paquete "SisEcua" es el siguiente:

```
BeginPackage["SisEcua`"]
```

```
...
```

```
GaussSeideltd::usage="GaussSeideltd[a], resuelve un sistema de
  ecuaciones lineales tridiagonal por el método de Gauss, donde \"a\"
  es la matriz con los elementos de las tres diagonales (en las tres
  primeras columnas) y las constantes en la cuarta."
```

```
Begin["Private`"]
```

```
...
```

```
Options[GaussSeideltd]={PrecisionGoal->9,AccuracyGoal->10,
  MaxIterations-> 400};
```

```
GaussSeideltd::maxit="Gauss Seidel para sistemas tridiagonales ha llegado
  al límite de iteraciones: `1` sin encontrar los resultados";
```

```
GaussSeideltd[a0_List,x0_List,OptionsPattern[]]:=
  Module[{a=a0,x=x0,y=x0,m,i,k,s,pg,ag,mi,li},
    {pg,ag,mi}=OptionValue[{PrecisionGoal,AccuracyGoal,
```

```

        MaxIterations}}];
m=Length[a]; li=mi;
While[True,
  y[[1]]=(a[[1,4]]-a[[1,3]]*y[[2]])/a[[1,2]];
  Do[y[[i]]=(a[[i,4]]-
    a[[i,1]]*y[[i-1]]-a[[i,3]]*y[[i+1]])/a[[i,2]],{i,2,m-1}];
  y[[m]]=(a[[m,4]]-a[[m,1]]*y[[m-1]])/a[[m,2]];
  For[i=1,i<=m,i++,If[Abs[x[[i]]/y[[i]]-1]>10^-pg,Break[]]];
  If[i==m+1,Return[y]];
  For[i=1,i<=m,i++,s=Sum[k=i+j-2;
    If[k!=0 && k!=m+1,a[[i,j]]*y[[k]],0],{j,3}];
    If[Abs[s-a[[i,4]]]>10^-ag ,Break[]]];
  If[i==m+1,Return[y]];
  If[--mi==0,Message[GaussSeideltd::maxit,li];Return[y],
    x=y]]]

End[]

EndPackage[]

```

Podemos probar este volviendo a resolver el sistema de ecuaciones (4).

$$\begin{aligned}
 2x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 &= 6 \\
 x_1 + 4x_2 + x_3 + 0x_4 + 0x_5 &= 36 \\
 0x_1 + x_2 + 4x_3 + 1x_4 + 0x_5 &= 72 \\
 0x_1 + 0x_2 + x_3 + 4x_4 + x_5 &= 108 \\
 0x_1 + 0x_2 + 0x_3 + x_4 + 2x_5 &= 66
 \end{aligned} \tag{4}$$

Primero importamos el paquete "SisEcu":

```
Get["SisEcu",Path->"E:\MAT205"]
```

Armamos la matriz a partir de los vectores de las diagonales y las constantes:

```
a=Transpose[ArrayFlatten[{Prepend[d1,0],d2,Append[d3,0],b}]]];
```

```
a//MatrixForm
```

$$\begin{pmatrix} 0 & 2 & 1 & 6 \\ 1 & 4 & 1 & 36 \\ 1 & 4 & 1 & 72 \\ 1 & 4 & 1 & 108 \\ 1 & 2 & 0 & 66 \end{pmatrix}$$

Empleamos como valores iniciales asumidos ceros:

```
x=ConstantArray[0,5];
```

Y mandamos al método de Gauss Seidel a resolver el sistema:

```
GaussSeideltd[a,x]/N
{-6.27545×10-11,6.,12.,18.,24.}
```

Como se puede observar, en este caso no se obtienen los resultados exactos: el primer resultado no es propiamente cero, sino "casi" cero. Ello es algo que ocurre con frecuencia en los métodos iterativos (como el presente), que como sabemos sólo devuelven resultados aproximados. Recordemos que con "Chop", podemos convertir los números cercanos a cero (menores a 10^{-10}) en ceros:

```
Chop[%]
{0,6.,12.,18.,24.}
```


20.1.2.2. Preguntas y ejercicios

1. ¿Por qué son más eficientes (requieren menos memoria y tiempo) los métodos que resuelven sistemas tridiagonales?
2. ¿Qué va en cada una de las cuatro columnas de la matriz que se manda a los métodos "Gausstd" y "GaussSeideltd"?
3. Escriba un programa para el método de Gauss empleando la estructura For en lugar de Do.
4. Escriba un programa para el método de Gauss empleando la estructura While en lugar de Do.
5. Escriba un programa para el método de Gauss Seidel empleando la recursividad para el ciclo externo y la estructura While, en lugar de las estructura Do y For, para los ciclos internos.
6. Escriba un programa para el método de Gauss Seidel empleando la estructura NewtWhile para el ciclo externo y la estructura Nest, en lugar de las estructuras Do y For, para los ciclos internos.
7. Escriba un programa para el método de Gauss Seidel empleando la estructura Nest para el ciclo externo y las estructuras Do y Which en lugar de For e If, para los ciclos internos.
8. Resuelva el siguiente sistema de ecuaciones lineales empleando "Gausstd", "GaussSeideltd" y corroborando los resultados con "Linear-Solve".

$$0.6x_1 + 0.1x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 = -1.23$$

$$0.1x_1 + 0.4x_2 + 0.1x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 = 3.96$$

$$0x_1 + 0.1x_2 + 0.4x_3 + 0.1x_4 + 0x_5 + 0x_6 + 0x_7 = 9.60$$

$$0x_1 + 0x_2 + 0.1x_3 + 0.4x_4 + 0.1x_5 + 0x_6 + 0x_7 = 11.52$$

$$0x_1 + 0x_2 + 0x_3 + 0.1x_4 + 0.4x_5 + 0.1x_6 + 0x_7 = -21.42$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0.1x_5 + 0.4x_6 + 0.1x_7 = -4.50$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0.1x_6 + 0.6x_7 = 0.60$$

9. Resuelva el siguiente sistema de ecuaciones lineales empleando "Gausstd", "GaussSeideltd" y corroborando los resultados con "Linear-Solve".

$$-2.0304x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 = -1.952$$

$$x_1 - 2.0288x_2 + x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 = 0.056$$

$$0x_1 + x_2 - 2.0272x_3 + x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 = 0.064$$

$$0x_1 + 0x_2 + x_3 - 2.0256x_4 + x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 = 0.072$$

$$0x_1 + 0x_2 + 0x_3 + x_4 - 2.0240x_5 + x_6 + 0x_7 + 0x_8 + 0x_9 = 0.080$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + x_5 - 2.0224x_6 + x_7 + 0x_8 + 0x_9 = 0.088$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + x_6 - 2.0208x_7 + x_8 + 0x_9 = 0.096$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + x_7 - 2.0192x_8 + x_9 = 0.104$$

$$0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + x_8 - 2.0176x_9 = 1.112$$

21. SISTEMAS DE ECUACIONES NO LINEALES

Para la resolución de sistemas de ecuaciones no lineales, no se cuentan con métodos analíticos, por lo tanto dichos sistemas sólo pueden ser resueltos mediante métodos numéricos.

Como sucede con prácticamente todos los métodos numéricos, en estos métodos se requieren valores iniciales asumidos (valores de prueba) para comenzar el proceso. En general, a medida que incrementa el número de ecuaciones del sistema, incrementa también el tiempo requerido para lograr convergencia, por lo que es importante contar con valores iniciales razonablemente cercanos a las soluciones.

Al concluir el presente tema, deberán estar capacitados para encontrar las soluciones de sistemas de ecuaciones no lineales empleando las funciones de Mathematica, los métodos estudiados en los temas 10 a 14 y el método de la Gradiente.

21.1. RESOLUCIÓN DE SISTEMAS DE ECUACIONES NO LINEALES CON LAS FUNCIONES DE MATHEMATICA

Mathematica en realidad cuenta con una función para encontrar soluciones numéricas de sistemas de ecuaciones no lineales: "FindRoot", sin embargo, cuenta con otras funciones de optimización, que pueden ser empleadas también para encontrar las soluciones de sistemas de ecuaciones no lineales.

Para resolver un sistema de ecuaciones no lineales con "FindRoot", se emplea la siguiente sintaxis:

$$\text{FindRoot}[\{\text{eqn1}, \text{eqn2}, \dots\}, \{\{x, x_0\}, \{y, y_0\}, \dots\}]$$

Donde "eqn1", "eqn2", etc., son las ecuaciones del sistema, "x", "y", etc., son las variables del sistema y "x₀", "y₀", etc. son los valores iniciales asumidos.

Así por ejemplo para resolver el siguiente sistema de ecuaciones no lineales:

$$\begin{aligned} x^2 + 2y^2 &= 22 \\ -2x^2 + xy - 3y &= -11 \end{aligned} \tag{1}$$

Empleando valores iniciales iguales a 1 y 2 (x=1, y=2), escribimos lo siguiente:

```
FindRoot[{x^2+2*y^2==22,-2*x^2+x*y-3*y==-11},{x,1},{y,2}]]
{x→2.,y→3.}
```

Y como vemos las soluciones son: x=2 y y=3. Por supuesto podemos guardar también las ecuaciones en una variable y emplear la variable en "FindRoot":

```
f={x^2+2*y^2==22,-2*x^2+x*y-3*y==-11};
```

```
FindRoot[f,{x,1},{y,2}]]
{x→2.,y→3.}
```

La ventaja de emplear una variable es que luego podemos emplear la misma en otras operaciones como por ejemplo comprobar la exactitud de los resultados:

```
f/.%
{True,True}
```

En lugar de mandar las ecuaciones como igualdades pueden ser mandadas también como funciones igualadas a cero (tal como ocurre cuando se resuelve una ecuación):

```
f={x^2+2*y^2-22,-2*x^2+x*y-3*y+11};
```

```
FindRoot[f,{{x,1},{y,2}}]  
{x→2.,y→3.}
```

Y podemos comprobar la exactitud de los resultados:

```
f/.%/InputForm  
{0., 0.}
```

Corroboramos así, una vez más, que los resultados obtenidos en este caso son exactos.

Como segundo ejemplo resolveremos el siguiente sistema de ecuaciones no lineales:

$$\begin{aligned} 5xy + 3x^2z - 2z &= 319 \\ 6yz - x^{2.1} &= 42.6345 \\ 3x^2 + 2y^2 - z^3 &= 80 \end{aligned} \quad (2)$$

Asumiendo como valores iniciales $x=1$, $y=2$, $z=3$.

```
f={5*x*y+3*x^2*z-2*z-319,6*y*z-x^2.1-42.6345,3*x^2+2*y^2-z^3-80};
```

```
FindRoot[f,{{x,1},{y,2},{z,3}}]  
{x→5.,y→4.,z→3.}
```

En consecuencia las soluciones son aproximadamente $x=5$, $y=4$ y $z=3$. Podemos comprobar la exactitud de los resultados reemplazando las soluciones en el sistema de ecuaciones:

```
NumberForm[f/.,16]  
{5.684341886080801×10-14, -1.4210854715202×10-14, 2.842170943040401×10-14}
```

Y como vemos en este caso las soluciones tienen una exactitud de unos 14 dígitos, que es una exactitud más que suficiente en la mayoría de los cálculos ingenieriles.

Como ya se dijo, si bien Mathematica no cuenta con otras funciones para resolver sistemas de ecuaciones no lineales, cuenta con varias funciones de optimización. Las funciones de optimización permiten encontrar el mínimo (o máximo) de una función igualada a cero (denominada función de error).

Un sistema de ecuaciones no lineales puede ser convertido en una función de error y en consecuencia en un problema de optimización, si se igualan las ecuaciones del sistema a cero y se suman sus cuadrados. El sumar los cuadrados de las ecuaciones igualadas a cero, en lugar de las ecuaciones solamente, tiene el propósito de eliminar el signo y evitar así las restas, las cuales pueden llevar a encontrar falsos mínimos (falsos ceros), como consecuencia de restar dos valores iguales.

Así por ejemplo, la función de error para el sistema de ecuaciones no lineales (1) es:

$$f(x,y) = (x^2 + 2y^2 - 22)^2 + (-2x^2 + xy - 3y + 11)^2 = 0 \quad (3)$$

Una de las funciones de optimización con las que cuenta Mathematica es "FindMinimum" que tiene el siguiente formato:

```
FindMinimum[f,{{x,x0},{y,y0},...}]
```

Donde "f" es la función a optimizar (función de error); "x", "y", etc. son los nombres de variables independientes y "x0", "y0", etc., son los valores iniciales asumidos para cada una de las variables.

Para resolver con "FindMinimum" el sistema de ecuaciones no lineales (1) escribimos lo siguiente:

```
f=(x^2+2*y^2-22)^2+(-2*x^2+x*y-3*y+11)^2;
```

```
FindMinimum[f,{{x,1},{y,2}}]
{0.,{x→2.,y→3.}}
```

Y como se puede observar se obtienen las mismas soluciones que con "FindRoot", además "FindMinimum" devuelve el valor de la función de error, lo que nos da una idea de la exactitud del resultado, en este ejemplo vemos que los valores encontrados son exactos, pues la función de error es cero.

De la misma manera podemos resolver el sistema de ecuaciones lineales (3) con "FindMinimum":

```
f=(5*x*y+3*x^2*z-2*z-319)^2+(6*y*z-x^2.1-42.6345)^2+(3*x^2+2*y^2-z^3-80)^2;
```

```
FindMinimum[f,{{x,1},{y,2},{z,3}}]
{5.9985×10-27,{x→5.,y→4.,z→3.}}
```

Otra función de optimización con la que cuenta Mathematica es "NMinimize" que tiene la siguiente sintaxis:

```
NMinimize[{f, restricciones},{x, y,...}]
```

Donde "f" es la función de error (igual que en FindMinimum), "restricciones" son las restricciones que deben cumplir las variables independientes (las restricciones son opcionales) y "x", "y", etc., son los nombres de las variables independientes del sistema.

Al igual que "FindMinimum", "NMinimize" devuelve el valor de la función de error y una lista de reglas con los valores de las variables independientes.

La ventaja de esta función con relación a "FindMinimum" es que no requiere valores iniciales, sin embargo, aun cuando la "restricciones" son opcionales, generalmente deben ser fijadas para asegurar la convergencia.

Como primer ejemplo encontraremos las soluciones del sistema de ecuaciones no lineales (1):

```
f=(x^2+2*y^2-22)^2+(-2*x^2+x*y-3*y+11)^2;
```

```
NMinimize[{f},{x,y}]
{0.,{x→2.,y→3.}}
```

Como se puede ver, en este caso en particular se encuentran los resultados sin incluir ninguna restricción, pero como se dijo, ello no es lo más frecuente. Incluyendo restricciones para que los resultados se busquen en el sector positivo:

```
NMinimize[{f,x>=0,y>=0},{x,y}]
{6.31089×10-30,{x→2.,y→3.}}
```

Se obtienen prácticamente los mismos resultados que en el caso anterior, aunque con una exactitud un tanto inferior.

Como segundo ejemplo encontraremos las soluciones del sistema de ecuaciones no lineales (2):

```
f=(5*x*y+3*x^2*z-2*z-319)^2+(6*y*z-x^2.1-42.6345)^2+(3*x^2+2*y^2-z^3-80)^2;
```

```
NMinimize[{f},{x,y,z}]
NMinimize::nrnum: The function value 109739. +7.00505 i is not a real number
at {x,y,z} = {-0.523097,0.275125,-0.797803}.
NMinimize[{{(-319+5 x y-2 z+3 x^2 z)^2+(-42.6345-x^2.1+6 y z)^2+(-80+3 x^2+2
y^2-z^3)^2},{x,y,z}]
```

En este caso no se encuentran las soluciones sino que se genera un mensaje de error porque en el proceso de búsqueda se ha encontrado un resultado imaginario (1009739+7.00505i) para valores de "x" y "z" negativos. Por ello debemos volver a resolver el problema pero restringiendo la búsqueda de soluciones a los números positivos:

```
NMinimize[{f,x>=0,y>=0,z>=0},{x,y,z}]
{5.9985*10^-27,{x->5.,y->4.,z->3.}}
```

En cuanto a la velocidad con la que se encuentran las soluciones normalmente el orden es: "FindRoot" (el más rápido), seguido por "FindMinimum" y finalmente "NMinimize".

Como ya vimos, cuando un sistema de ecuaciones no lineales puede ser reordenado de manera que dependa de una sola variable, es posible encontrar las soluciones con los métodos estudiados en los temas 10 a 14, así por ejemplo el sistema de ecuaciones no lineales (1) puede ser reorganizado de la siguiente forma:

$$f(x) = x^2 + 2y^2 - 22 = 0$$

$$y = \frac{2x^2 - 11}{x - 3}$$

Entonces la función a resolver es:

```
f[x_]:=Module[{y},y=(2*x^2-11)/(x-3);x^2+2*y^2-22]
```

Importamos el paquete "Alge1", que es donde se encuentran programados los métodos del capítulo 1:

```
Get["Alge1",Path->"E:\MAT205 "]
```

Y resolvemos la función empleando por ejemplo el método de la Secante:

```
Secante[f,1.1,1.2]
2.
```

Como se puede observar el resultado para "x" es 2. El valor de "y" se obtiene reemplazando esta solución en la segunda ecuación:

```
(2*x^2-11)/(x-3)/.x->%
3.
```

Y como era de esperar el valor de "y" es 3 (igual que con los otros métodos). Por supuesto podemos emplear otro método en lugar de la "Secante", por ejemplo el método de Newton:

```
NewtonDN[f,1.2]
2.
```

Obteniéndose igualmente el mismo resultado. Estos métodos sin embargo suelen ser mucho más sensibles al valor inicial asumido que los métodos específicos para sistemas de ecuaciones lineales, así por ejemplo si en lugar de 1.2, empleamos 1.1, obtenemos:

```
NewtonDN[f,1.1]
-3.54961
```

Que también es correcta, sólo que se encuentra en el sector negativo.

21.2. MÉTODO DE LA GRADIENTE

El método de la gradiente es esencialmente el método de Newton aplicado a sistemas de ecuaciones no lineales en lugar de una sola ecuación no lineal.

En este método se expanden las ecuaciones que conforman el sistema empleando series de Taylor, por ejemplo, si en el sistema tenemos 3 ecuaciones no lineales:

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0 \end{aligned} \quad (1)$$

Siendo x_1 , x_2 y x_3 , los valores asumidos para las tres variables del sistema. Expandiendo las tres ecuaciones en series de Taylor (todas las funciones se evalúan en x_1 , x_2 y x_3) obtenemos:

$$\begin{aligned} f_1(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_1 + \frac{\partial f_1}{\partial x_1} \Delta x_1 + \frac{\partial f_1}{\partial x_2} \Delta x_2 + \frac{\partial f_1}{\partial x_3} \Delta x_3 + \dots = 0 \\ f_2(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_2 + \frac{\partial f_2}{\partial x_1} \Delta x_1 + \frac{\partial f_2}{\partial x_2} \Delta x_2 + \frac{\partial f_2}{\partial x_3} \Delta x_3 + \dots = 0 \\ f_3(x_1 + \Delta x_1, x_2 + \Delta x_2, x_3 + \Delta x_3) &= f_3 + \frac{\partial f_3}{\partial x_1} \Delta x_1 + \frac{\partial f_3}{\partial x_2} \Delta x_2 + \frac{\partial f_3}{\partial x_3} \Delta x_3 + \dots = 0 \end{aligned} \quad (2)$$

Donde Δx_1 , Δx_2 y Δx_3 son los valores que deberían añadirse a los valores asumidos (x_1 , x_2 y x_3) para que las funciones (f_1 , f_2 y f_3) se hagan cero. En otras palabras si fuera posible calcular los valores de Δx_1 , Δx_2 y Δx_3 , las soluciones del sistema serían $y_1 = x_1 + \Delta x_1$, $y_2 = x_2 + \Delta x_2$ y $y_3 = x_3 + \Delta x_3$. Por supuesto al tratarse de series infinitas, no es posible en la práctica calcular dichos valores, sin embargo, si es posible obtener una aproximación de los mismos tomando en cuenta sólo los términos de primero orden (que es lo que hace el método de Newton), es decir:

$$\begin{aligned} \frac{\partial f_1}{\partial x_1} \Delta x_1 + \frac{\partial f_1}{\partial x_2} \Delta x_2 + \frac{\partial f_1}{\partial x_3} \Delta x_3 &= -f_1 \\ \frac{\partial f_2}{\partial x_1} \Delta x_1 + \frac{\partial f_2}{\partial x_2} \Delta x_2 + \frac{\partial f_2}{\partial x_3} \Delta x_3 &= -f_2 \\ \frac{\partial f_3}{\partial x_1} \Delta x_1 + \frac{\partial f_3}{\partial x_2} \Delta x_2 + \frac{\partial f_3}{\partial x_3} \Delta x_3 &= -f_3 \end{aligned} \quad (3)$$

Que como vemos conforman un sistema de 3 ecuaciones lineales con 3 incógnitas (asumiendo por supuesto que sea posible calcular las derivadas parciales). Debemos recordar no obstante que estos valores, al resultar de los términos de primer orden, son sólo aproximados y que por lo tanto no nos permiten calcular realmente las soluciones aunque si nos aproximan a las mismas, por lo tanto ahora $y_1 = x_1 + \Delta x_1$, $y_2 = x_2 + \Delta x_2$ y $y_3 = x_3 + \Delta x_3$, son valores más cercanos a la solución que los valores asumidos (x_1 , x_2 y x_3), pero no constituyen la solución del sistema.

En consecuencia y como es la característica del método de Newton, la solución del sistema se convierte en un problema iterativo: comenzando con los valores asumidos x_1 a x_n , se calculan nuevos valores y_1 a y_n resolviendo el sistema de ecuaciones lineales (previo cálculo de las derivadas parciales), entonces se comparan los valores calculados con los asumidos y si son aproximadamente iguales el proceso concluye siendo las soluciones los valores

desde y_1 a y_n , caso contrario los valores calculados (y_1 a y_n) se convierten en los nuevos valores asumidos (x_1 a x_n) y se repite el proceso.

En lugar de comparar los valores asumidos con los calculados, se pueden sustituir los valores calculados (y_1 a y_n) en las funciones y comprobar si las funciones son aproximadamente iguales a cero, de ser así el proceso concluye, siendo las soluciones los valores desde y_1 a y_n , caso contrario los valores calculados (y_1 a y_n) se convierten en los nuevos valores asumidos (x_1 a x_n) y se repite el proceso.

Si bien en algunos casos donde las funciones son sencillas, es posible calcular las derivadas analíticamente, en la mayoría de los casos esa alternativa no es práctica debiendo ser calculadas numéricamente. Para este fin se puede emplear, como se hizo en el método de Newton, la fórmula de diferencia central de segundo orden, adaptada para el cálculo de las derivadas parciales:

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(x_1, x_2, \dots, x_{j+h}, \dots, x_n) - f_i(x_1, x_2, \dots, x_{j-h}, \dots, x_n)}{2h}; \quad h = x_j * 10^{-4} \quad (4)$$

Se ha comprobado que el valor propuesto para h ($x_j * 10^{-4}$) es un valor adecuado en la mayoría de los casos, no obstante, la fórmula de diferencia central, sólo requiere que dicho valor sea un incremento pequeño de x_j , por lo tanto dicho valor ser reemplazado por otro que se considere más adecuado.

El sistema de ecuaciones lineales que se forma en el método, puede ser resuelto empleando alguno de los métodos estudiados en los temas 17 a 19. En este tema emplearemos el método de Cholesky.

21.2.1. Ejemplo manual

Para comprender mejor el método y contar con valores de prueba, resolveremos manualmente el siguiente sistema de ecuaciones:

$$\begin{aligned} x^2 + 2y^2 &= 22 \\ -2x^2 + xy - 3y &= -11 \end{aligned}$$

Para estar de acuerdo con la simbología empleada en la explicación del método, colocamos primero estas ecuaciones en función de las variables x_1 y x_2 e igualamos las ecuaciones a cero:

$$\begin{aligned} f_1 &= x_1^2 + 2x_2^2 - 22 = 0 \\ f_2 &= -2x_1^2 + x_1x_2 - 3x_2 + 11 = 0 \end{aligned}$$

Asumiendo los siguientes valores iniciales: $x_1=1.5$ y $x_2=2$, procedemos a calcular las funciones y las derivadas parciales que conforman el sistema de ecuaciones lineales (2 en el ejemplo):

$$\begin{aligned} h &= x_1 * 10^{-4} = 1.5 * 10^{-4} \\ \frac{\partial f_1}{\partial x_1} &= \frac{f_1(x_{1+h}, x_2) - f_1(x_{1-h}, x_2)}{2h} = \frac{f_1(1.50015, 2) - f_1(1.49985, 2)}{2 * 1.5 * 10^{-4}} = 3 \\ \frac{\partial f_2}{\partial x_1} &= \frac{f_2(x_{1+h}, x_2) - f_2(x_{1-h}, x_2)}{2h} = \frac{f_2(1.50015, 2) - f_2(1.49985, 2)}{2 * 1.5 * 10^{-4}} = -4 \\ h &= x_2 * 10^{-4} = 0.0002 \\ \frac{\partial f_1}{\partial x_2} &= \frac{f_1(x_1, x_{2+h}) - f_1(x_1, x_{2-h})}{2h} = \frac{f_1(1.5, 2.0002) - f_1(1.5, 1.9998)}{2 * 0.0002} = 8 \end{aligned}$$

$$\begin{aligned}\frac{\partial f_2}{\partial x_2} &= \frac{f_2(x_1, x_{2+h}) - f_2(x_1, x_{2-h})}{2h} = \frac{f_2(1.5, 2.002) - f_2(1.5, 1.9998)}{2*0.0002} = -1.5 \\ -f_1 &= -f_1(x_1, x_2) = -f_1(1.5, 2) = 11.75 \\ -f_2 &= -f_2(x_1, x_2) = -f_2(1.5, 2) = -3.5\end{aligned}$$

Por lo tanto el sistema de ecuaciones lineales a resolver es:

$$\begin{aligned}\frac{\partial f_1}{\partial x_1} \Delta x_1 + \frac{\partial f_1}{\partial x_2} \Delta x_2 &= 3\Delta x_1 + 8\Delta x_2 = 11.75 = -f_1 \\ \frac{\partial f_2}{\partial x_1} \Delta x_1 + \frac{\partial f_2}{\partial x_2} \Delta x_2 &= -4\Delta x_1 - 1.5\Delta x_2 = -3.5 = -f_2\end{aligned}$$

Resolviendo el sistema se obtiene:

$$\begin{aligned}\Delta x_1 &= 0.377272727272727 \\ \Delta x_2 &= 1.3272727272727274\end{aligned}$$

Por lo tanto los nuevos valores de las incógnitas son:

$$\begin{aligned}y_1 &= x_1 + \Delta x_1 = 1.5 + 0.377272727272727 = 1.877272727272727 \\ y_2 &= x_2 + \Delta x_2 = 2 + 1.3272727272727274 = 3.3272727272727274\end{aligned}$$

Que difieren de los valores originales (1.5 y 2), por la que el proceso debe repetirse empleando como valores asumidos los valores calculados.

Para llevar a cabo esta iteración en Mathematica importamos el paquete "SisEcua`" (que es donde se encuentra programado el método de Cholesky):

```
Get["SisEcua`", Path->"E:\HPV\MAT205\1-2009\CUADERNOS MATHEMATICA"]
```

Programamos las funciones:

```
f1[x1_, x2_] := x1^2 + 2*x2^2 - 22;
f2[x1_, x2_] := -2*x1^2 + x1*x2 - 3*x2 + 11;
```

Programamos también las derivadas parciales con respecto a "x1" y "x2":

```
dfx1[f_Symbol, x1_, x2_] := Module[{h = x1*10^-4},
  (f[x1+h, x2] - f[x1-h, x2]) / (2*h)]
dfx2[f_Symbol, x1_, x2_] := Module[{h = x1*10^-4},
  (f[x1, x2+h] - f[x1, x2-h]) / (2*h)]
```

Asignamos los valores iniciales a "x1" y "x2":

```
x1=1.5; x2=2;
```

Calculamos los valores de las derivadas y las funciones:

```
dfx1[f1, x1, x2]
3.
```

```
dfx2[f1, x1, x2]
8.
```

```
-f1[x1, x2]
11.75
```

```
dfx1[f2, x1, x2]
-4.
```

```
dfx2[f2, x1, x2]
-1.5
```

```
-f2[x1,x2]
-3.5
```

Componemos la matriz aumentada con los resultados obtenidos:

```
a={{% % % % %, % % % %, % % % % }, {% % %, % %, % }}
{{3., 8., 11.75}, {-4., -1.5, -3.5}}
```

Resolvemos el sistema de ecuaciones lineales y asignamos los resultados a dos variables ("dx1" y "dx2"):

```
{dx1,dx2}=Cholesky[a][[All,3]]
{0.377273,1.32727}
```

Finalmente calculamos los nuevos valores de prueba:

```
y1=x1+dx1
1.87727
```

```
y2=x2+dx2
3.32727
```

Al ser los valores asumidos y calculados diferentes (y los valores de "dx" y "dy" diferentes de cero), repetimos el proceso:

```
x1=y1;x2=y2;
```

```
dfx1[f1,x1,x2]
3.75455
```

```
dfx2[f1,x1,x2]
13.3091
```

```
-f1[x1,x2]
-3.66564
```

```
dfx1[f2,x1,x2]
-4.18182
```

```
dfx2[f2,x1,x2]
-1.12273
```

```
-f2[x1,x2]
-0.216074
```

```
a={{% % % % %, % % % %, % % % % }, {% % %, % %, % }}
{{3.75455,13.3091,-3.66564}, {-4.18182,-1.12273,-0.216074}}
```

```
{dx1,dx2}=Cholesky[a][[All,3]]
{0.135909,-0.313764}
```

```
y1=x1+dx1
2.01318
```

```
y2=x2+dx2
3.01351
```

Ahora los valores calculados y asumidos son iguales en los dos primeros dígitos. Repitiendo el proceso una vez más obtenemos:

```
x1=y1;x2=y2;
```

```
dfx1[f1,x1,x2]
4.02636
```

```
dfx2[f1,x1,x2]
12.054
```

```
-f1[x1,x2]
-0.215367
```

```
dfx1[f2,x1,x2]
-5.03922
```

```
dfx2[f2,x1,x2]
-0.986818
```

```
-f2[x1,x2]
0.0795857
```

```
a={{%%%%%,%%%%%,%%%%%},{%%%,%%,%}}
{{4.02636,12.054,-0.215367},{-5.03922,-0.986818,0.0795857}}
```

```
{dx1,dx2}=Cholesky[a][[All,3]]
{-0.0131549,-0.0134727}
```

```
y1=x1+dx1
2.00003
```

```
y2=x2+dx2
3.00004
```

Ahora los valores calculados y asumidos son iguales en los primeros 5 dígitos y los valores de "dx" cercanos a cero, repitiendo el proceso una vez más obtenemos:

```
x1=y1;x2=y2;
```

```
dfx1[f1,x1,x2]
4.00005
```

```
dfx2[f1,x1,x2]
12.0001
```

```
-f1[x1,x2]
-0.000536081
```

```
dfx1[f2,x1,x2]
-5.00007
```

```
dfx2[f2,x1,x2]
-0.999973
```

```
-f2[x1,x2]
0.000168872
```

```
a={{%%%%%,%%%%%,%%%%%},{%%%,%%,%}}
{{4.00005,12.0001,-0.000536081},{-5.00007,-0.999973,0.000168872}}
```

```
{dx1,dx2}=Cholesky[a][[All,3]]
{-0.000026614,-0.0000358015}
```

$y1=x1+dx1$
2.

$y2=x2+dx2$
3.

Que corresponden ya a los resultados exactos, aunque los valores de "dx" no son todavía cero.

21.2.2. Algoritmo y código

El algoritmo del método de la Gradiente, que básicamente es el proceso llevado a cabo en el acápite previo, se presenta en el diagrama de actividades de la siguiente página.

El código elaborado en base a dicho algoritmo y añadido al paquete "SisEcu" es el siguiente:

```
BeginPackage["SisEcu`"]
```

```
...
```

```
Gradiente::usage="Gradiente[f,x], resuelve un sistema de ecuaciones no  
lineales por el método de la gradiente, donde \"f\" es la lista con  
las ecuaciones del sistema igualadas a cero y \"x\" es la lista con  
los nombres de las variables y valores iniciales asumidos en la forma  
{variable1,valor1},{variable2,valor2},...\"."
```

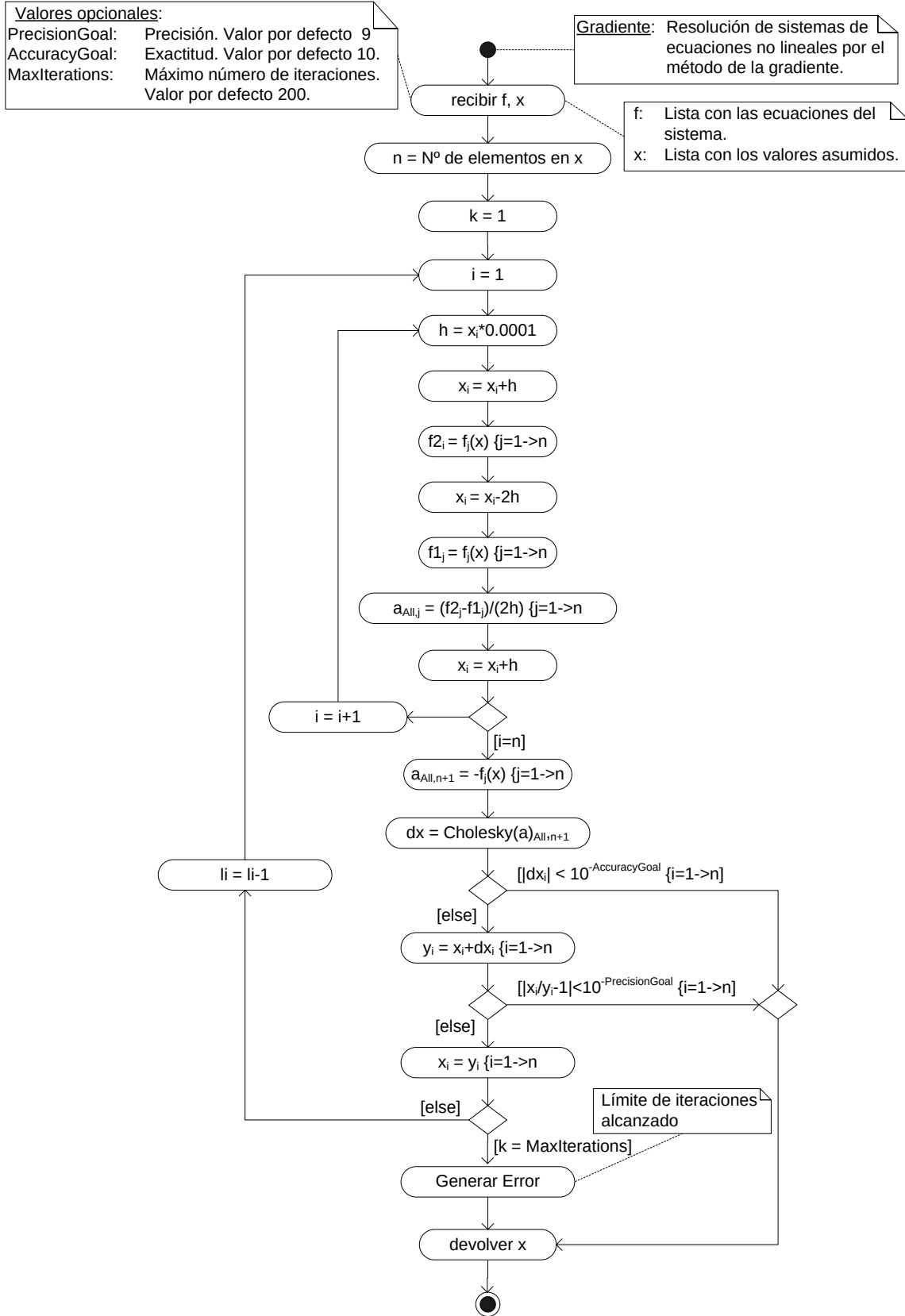
```
Begin["Private`"]
```

```
...
```

```
Options[Gradiente]={PrecisionGoal->9,AccuracyGoal->10,  
MaxIterations->200};
```

```
Gradiente::maxit="Gradiente ha llegado al límite de iteraciones `1`,  
sin haber encontrado los resultados";
```

```
Gradiente[f_List,x0_List,OptionsPattern[]]:=Module[  
{x,y,n=Length[x0],h,f1,f2,a,dx,pg,ag,li},  
x=Table[x0[[i]]/.List->Rule,{i,n}];  
y=x;a=ConstantArray[0,{n,n+1}];  
{pg,ag,li}=OptionValue[{PrecisionGoal,AccuracyGoal,  
MaxIterations}];  
Do[  
Do[h=x[[i]][[2]]*0.0001;  
x[[i]][[2]]=x[[i]][[2]]+h;  
f2=f/.x;  
x[[i]][[2]]=x[[i]][[2]]-2*h;  
f1=f/.x;  
a[[All,i]]=(f2-f1)/(2*h);  
x[[i]][[2]]=x[[i]][[2]]+h,{i,n}];  
a[[All,n+1]]=-f/.x;  
dx=Cholesky[a][[All,n+1]];  
Do[If[Abs[dxx[[i]]]>=10^-ag,Goto[a]],{i,n}];  
Goto[c];  
Label[a]; Do[y[[i]][[2]]=x[[i]][[2]]+dx[[i]],{i,n}];  
Do[If[Abs[x[[i]][[2]]/y[[i]][[2]]-1]>=10^-pg,
```



```

Goto[b]], {i, n}];
Goto[c]; Label[b]; x=y, {li}];

```

```

Message[Gradiente::maxit,li];
Label[c]; Return[x]
Goto[c];Label[b];x=y,{li}];

```

Para probar el programa resolvemos con el mismo el sistema de ecuaciones del ejemplo manual:

```

Get["SisEcua`",Path->"C:\MAT205 "]
f={x^2+2*y^2-22,-2*x^2+x*y-3*y+11};
vx={{x,1.5},{y,2}};
Gradiente[f,vx]
{x->2.,y->3.}

```

Y como se pudo observar se obtienen los mismos resultados que en el ejemplo manual y con los otros métodos estudiados en este tema.

Podemos probar también el programa resolviendo el sistema de ecuaciones no lineales (2) (resuelto también con los otros métodos):

```

f={5*x*y+3*x^2*z-2*z-319,6*y*z-x^2.1-42.6345,3*x^2+2*y^2-z^3-80};
vx={{x,1},{y,2},{z,3}};
Gradiente[f,vx]
{x->5.,y->4.,z->3.}

```

Una vez más se obtienen las soluciones correctas.

21.3. Ejercicios

1. Programe el método de la Gradiente empleando la estructura Nest y fijando la precisión en 7 dígitos, la exactitud en 7 dígitos y el máximo de iteraciones en 100.
2. Programe el método de la Gradiente empleando la estructura While.
3. Programe el método de la Gradiente empleando la estructura For.
4. Programe el método de la Gradiente empleando la Recursividad.
5. Programe el método de la Gradiente empleando la estructura NestWhile.
6. Programe el método de la Gradiente empleando para los el comando Goto.
7. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: $x=-1$, $y=-2$.

$$4 - x^2 + y^2 = 0$$

$$1 - e^x - y = 8$$

8. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 7 con "FindMinimum".
9. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 7 con "NMinimize", restringiendo "x" y "y" a valores negativos.
10. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 7 con el método de la "Gradiente".
11. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: $x=1.1$, $y=1.2$.

$$e^x - y = 0$$

$$xy - e^x = 0$$

12. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 11 con "FindMinimum".
13. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 11 con "NMinimize", restringiendo "x" y "y" a valores positivos.
14. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 11 con el método de la "Gradiente".
15. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: x=2, y=3, z=4.

$$3xy - y + z^2 = 29$$

$$xyz - yz^2 = 8$$

$$2xy - y^2 - z^2 = 15$$

16. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 15 con "FindMinimum".
17. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 15 con "NMinimize", restringiendo "x", "y" y "z" a valores positivos.
18. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 15 con el método de la "Gradiente".
19. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: x=1, y=1.2, z=1.6.

$$x^2 + y^2 + z^2 = 17$$

$$xyz = 12$$

$$x + y - z^2 = -4$$

20. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 19 con "FindMinimum".
21. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 19 con "NMinimize", sin restringir los valores de "x", "y" y "z".
22. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 19 con el método de la "Gradiente".
23. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: x=1, y=1.1, z=1.2.

$$xyz - x^2 + y^2 = 1.34$$

$$xy - z^2 = 0.09$$

$$e^x - e^y + z = 0.41$$

24. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 23 con "FindMinimum".

25. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 23 con "NMinimize", restringiendo "x", "y" y "z" a valores positivos.
26. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 23 con el método de la "Gradiente".
27. Encuentre las soluciones del siguiente sistema de ecuaciones no lineales con "FindRoot", empleando como valores iniciales: $x_1=1$, $x_2=1$, $x_3=1$, $x_4=x$, $x_5=1$, $x_6=1$, $x_7=1$, $x_8=1$, $x_9=1$, $x_{10}=1$ y $x_{11}=10$.

$$x_1 + x_4 = 3$$

$$2x_1 + x_2 + x_4 + x_7 + x_8 + x_9 + 2x_{10} = 10 + r$$

$$x_2 + 2x_5 + x_6 + x_7 = 8$$

$$2x_3 + x_5 = 4r$$

$$x_1 x_5 = a_1 x_2 x_4$$

$$x_6 x_2^{1/2} = a_2 (x_2 x_4 x_{11})^{1/2}$$

$$x_7 x_4^{1/2} = a_3 (x_1 x_4 x_{11})^{1/2}$$

$$x_8 x_4 = a_4 x_2 x_{11}$$

$$x_9 x_4 = a_5 x_1 (x_3 x_{11})^{1/2}$$

$$x_{10} x_4^2 = a_6 x_4^2 x_{11}$$

$$x_{11} = x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10}$$

$$a_1 = 0.193; \quad a_2 = 0.002597; \quad a_3 = 0.003448;$$

$$a_4 = 0.00001799; \quad a_5 = 0.002155; \quad a_6 = 0.00004836;$$

$$r = 4.056734$$

28. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 27 con "FindMinimum".
29. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 27 con "NMinimize", restringiendo "x", "y" y "z" a valores positivos.
30. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 27 con el método de la "Gradiente".
31. Encuentre los valores de "x", "y" y "z" del siguiente sistema de ecuaciones no lineales. Los valores de r_1 , r_2 y r_3 deben ser encontrados con "RowReduce" y el sistema de ecuaciones no lineales resuelto con "FindRoot", empleando como valores iniciales 1.1, 1.2 y 1.3.

$$x + 2y^2 - z = r_1$$

$$4x^3 - 2xy + z^2 = r_2$$

$$xyz - 4x^2 + z^3 = r_3$$

$$5r_1 + r_2 - r_3 = 10$$

$$4r_1 - 2r_2 + r_3 = 35$$

$$6r_1 + 3r_2 - 2r_3 = 5$$

32. Encuentre las soluciones del sistema de ecuaciones de la pregunta 31, empleando "LinearSolve" para resolver el sistema de ecuaciones lineales y "FindMinimum" para el sistema de ecuaciones no lineales (valores iniciales 1.1, 1.2 y 1.3).

33. Encuentre las soluciones del sistema de ecuaciones de la pregunta 31, empleando "Inverse" para resolver el sistema de ecuaciones lineales y "NMinimize" para el sistema de ecuaciones no lineales (restricción: valores positivos).
34. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 31 con el método de la "Gradiente".
35. Encuentre los valores de "x", "y" y "z" del siguiente sistema de ecuaciones no lineales. Los valores de r_1 , r_2 y r_3 deben ser encontrados con "NSolve" y el sistema de ecuaciones no lineales resuelto con "FindRoot", empleando como valores iniciales 0.5, 0.6 y 0.7

$$3x^{2.1} + 5y^{1.3} - z = r_3$$

$$6x^{1.2} - 4y^{1.4} + z^{2.2} = r_2$$

$$xyz - x^2 - y^2 + z^2 = r_1$$

$$r^3 - 14.7577r^2 + 48.8074r - 45.5124 = 0$$

36. Encuentre las soluciones del sistema de ecuaciones de la pregunta 35, empleando "NewtonP" (valor inicial 1.1) y DivSin1, para encontrar las dos primeras raíces del polinomio, siendo la tercera raíz despejada directamente del polinomio residual. El sistema de ecuaciones no lineales debe ser resuelto con "FindMinimum" (valores iniciales 0.5, 0.6 y 0.7).
37. Encuentre las soluciones del sistema de ecuaciones de la pregunta 35, empleando "Bairstow" (valores iniciales 1.1 y 1.2) y el método de la "Gradiente".
38. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 35 con el método de la "Gradiente".
39. Encuentre los valores de "x", "y" y "z" del siguiente sistema de ecuaciones no lineales. El valor de "u" debe ser calculado con "NewtonDN" (valor inicial 7.1), el valor de "v" con "Secante" (valores iniciales 16 y 17), el valor de "w" con "Incremental" (valor inicial 10, incremento 1). El sistema de ecuaciones no lineales debe ser resuelto con "FindRoot" (valores iniciales 2, 1.5 1).

$$5x + 2y - 4z^2 = u$$

$$6xy - 4yz^3 + y^2z^2 = v$$

$$x^3 - 5xy + z = w$$

$$e^{u/3} - 2u^{2.1} + 31u = 23.4544$$

$$7v^{1.2} - v^{2.1} + 30v = 50.0094$$

$$w^{2.7} - 15w^2 - 46w = 22.3214$$

40. Encuentre los valores de "x", "y" y "z" del siguiente sistema de ecuaciones no lineales de la pregunta 39. El valor de "u" debe ser calculado con "SegSol" (valor inicial 10, incremento 0.1) y "RegulaFalsi", el valor de "v" con "SegSol" (valor inicial 16, incremento 1) y "Biseccion", el valor de "w" con "SegSol" (valor inicial 10, incremento 1) y "Secante". El sistema de ecuaciones no lineales debe ser resuelto con "FindMinimum" (valores iniciales 2, 1.5 1).

41. Encuentre los valores de "x", "y" y "z" del siguiente sistema de ecuaciones no lineales de la pregunta 39. El valor de "u" debe ser calculado con "FindRoot" (valor inicial 10.1, valor mínimo 5, valor máximo 25), el valor de "v" con "NewtonDN" (valor inicial 18.2), el valor de "w" con "FindRoot" (valor inicial 35.1). El sistema de ecuaciones no lineales debe ser resuelto con "NMinimize" (restricción: valores positivos).
42. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 39 con el método de la "Gradiente".
43. Encuentre los valores de "s", "t" y "o" del siguiente sistema de ecuaciones. Los valores de "u", "v" y "w" deben ser calculados con "LinearSolve", los valores de "r₁", "r₂" y "r₃" con "NSolve", el valor de "s" con "FindRoot" (valor inicial 5.1), el valor de "t" con "NewtonDN" (valor inicial 4.1), el valor de "q" con "Secante" (valores iniciales 20 y 21) y los valores de "s", "t" y "o" con "FindRoot" (valores iniciales 1.5, 1.6 y 1.7).

$$s.m^{1.1} + \frac{t}{2}m.n - q.s.o^{1.2} = -259.327$$

$$(s+t+q).m.o - 2.t.q.n^{1.3} + 5.s.o^{1.4} = -244.151$$

$$s.t.n.m.o - (s+t-q).m^{1.1} - 3.s.q.n^{1.3} = 275.166$$

$$2r_1s^{2.2} - r_1(r_3 - r_2)s^{3.2} + r_2s = -89.0059$$

$$2r_1(t+5)^{0.3} - (r_1+r_3)t^{0.6} + (2r_3+r_2)t = 30.7328$$

$$r_1q^{3.3} - 2(r_1+r_2)q^{1.5} - 3r_3q^{2.3} = 4.89479$$

$$r_1q^{3.3} - 2(r_1+r_2)q^{1.5} - 3r_3q^{2.3} = 4.89479$$

$$r^3 - vr^2 + \frac{2w+v}{4}r + \frac{v-3u}{4} = 0$$

$$2u + 5v - 3w = 2y - x$$

$$4u - 2v + 7w = 2(x+z) + y$$

$$5u - 3v + 4w = 2x + z$$

$$x=5; y=6; z=7$$

44. Encuentre los valores de "s", "t" y "o" de sistema de ecuaciones de la pregunta 43. Los valores de "u", "v" y "w" deben ser calculados con "RowReduce", los valores de "r₁", "r₂" con "NewtonP" y "DivSin1", siendo el valor de "r₃" despejado del polinomio residual, el valor de "s" con "SegSol" y "RegulaFalsi" (valor inicial 1, incremento 1), el valor de "t" con "SegSol" y "Biseccion" (valor inicial 1, incremento 1), el valor de "q" con "SegSol" y "Secante" (valor inicial 1, incremento 1) y los valores de "s", "t" y "o" con "FindMinimum" (valores iniciales 1.5, 1.6 y 1.7).
45. Encuentre los valores de "s", "t" y "o" de sistema de ecuaciones de la pregunta 43. Los valores de "u", "v" y "w" deben ser calculados con

"Inverse", los valores de " r_1 ", " r_2 " con "Bairstow" siendo el valor de " r_3 " despejado del polinomio residual obtenido con "DivSin2", el valor de " s " con "Incremental" (valor inicial 1, incremento 1), el valor de " t " con "Secante" (valores iniciales 5 y 6), el valor de " q " con "FindRoot" (valor inicial 7.1, valor mínimo 6, valor máximo 20) y los valores de " s ", " t " y " o " con "NMinimize" (restricción: valores positivos).

46. Encuentre las soluciones del sistema de ecuaciones no lineales de la pregunta 43 con el método de la "Gradiente".

22. AJUSTE DE DATOS

Frecuentemente, al diseñar o resolver un problema de ingeniería, se debe trabajar con datos tabulados en lugar de funciones o ecuaciones simbólicas. Estos datos son empleados normalmente para realizar una serie de cálculos que pueden ir desde la estimación de valores intermedios hasta la integración y diferenciación numérica.

Una de las alternativas para trabajar con datos tabulados es el ajustar los mismos a una ecuación simbólica, es decir calcular los coeficientes y constantes de una ecuación simbólica de manera que la misma reproduzca con la mayor fidelidad posible los datos originales. Una vez realizado el ajuste, la ecuación resultante puede ser empleada igual que cualquier ecuación analítica, pudiendo por ejemplo ser graficada, derivada, integrada, empleada para calcular valores, etc.

Normalmente se elige el ajuste, en lugar de la interpolación (que estudiaremos en el subsiguiente tema) cuando los datos no son exactos y esta situación se presenta en la mayoría de los datos experimentales en el área de la ingeniería.

En general el ajuste de datos a expresiones simbólicas se basa en el método de los mínimos cuadrados. Y este método puede ser abordado desde el punto de vista estadístico así como desde el punto de vista del cálculo diferencial e integral. En este capítulo abordaremos el problema desde el punto de vista del cálculo diferencial e integral, específicamente estudiaremos el ajuste de datos a expresiones simbólicas de la siguiente forma:

$$y = c_1 f_1(x) + c_2 f_2(x) + \dots + c_m f_m(x) \quad (1)$$

Donde "y" es la variable dependiente, "x" es la variable independiente y "c₁" a "c_m" son los coeficientes cuyos valores deben ser calculados de manera que la ecuación refleje lo más fielmente posible el comportamiento de los datos.

Supongamos por ejemplo que contamos con los siguientes datos experimentales:

x	4.0	3.2	6.8	9.3	1.2	0.7	5.5	7.4	6.5	2.4
y	6.2	5.3	9.1	12.0	3.3	1.5	7.9	10.6	8.7	6.3

Y queremos ajustar los mismos a la ecuación simbólica:

$$y = a + bx^2 + cx^4 + d \ln(x)$$

Entonces el problema radica en calcular los coeficientes: "a", "b", "c" y "d" de manera que al reemplazar un valor de "x" en la ecuación, obtengamos un valor para "y" que sea lo más cercano posible al valor tabulado, así en este ejemplo si reemplazamos 9.3 en la ecuación ajustada deberíamos obtener como resultado 12.0 o un valor cercano a 12. Mientras más cercana sea la coincidencia entre los valores calculados y los valores tabulados, mejor será el ajuste logrado.

Como se puede observar, la ecuación de este ejemplo se ajusta a la forma de la ecuación (1), siendo el número de funciones 4 (m=4): $f_1=1$; $f_2=x^2$; $f_3=x^4$ y $f_4=\ln(x)$, siendo los coeficientes: $c_1=a$, $c_2=b$, $c_3=c$ y $c_4=d$.

Para calcular las constantes de la ecuación se reemplaza cada par de datos (x_i , y_i) en la ecuación 2 y dado que el ajuste puede no ser perfecto (y generalmente no lo es), esta igualdad no se cumple exactamente, razón por la cual existe una diferencia o residuo. Si llamamos "r_i" al residuo que se

obtiene al reemplazar cada par de datos " $x_i - y_i$ " en la ecuación 2, se obtiene el siguiente sistema de ecuaciones:

$$\begin{aligned} a + b(4.0)^{0.2} + c(4.0)^4 + d \ln(4.0) - 6.2 &= r_1 \\ a + b(3.2)^{0.2} + c(3.2)^4 + d \ln(3.2) - 5.3 &= r_2 \\ &\dots \\ a + b(2.4)^{0.2} + c(2.4)^4 + d \ln(2.4) - 6.3 &= r_{10} \end{aligned} \quad (2)$$

Si el ajuste fuera perfecto, los residuos serían cero, por consiguiente para lograr el mejor ajuste posible se deben minimizar estos residuos, haciendo que los mismos tiendan a cero y esto es justamente lo que se hace en el método de los mínimos cuadrados.

En general para cualquier ecuación de la forma (1) y para un número " n " de datos " $x_i - y_i$ ", se forma el siguiente sistema de ecuaciones:

$$\begin{aligned} c_1 f_1(x_1) + c_2 f_2(x_1) + \dots + c_m f_m(x_1) - y_1 &= r_1 \\ c_1 f_1(x_2) + c_2 f_2(x_2) + \dots + c_m f_m(x_2) - y_2 &= r_2 \\ &\dots \\ c_1 f_1(x_n) + c_2 f_2(x_n) + \dots + c_m f_m(x_n) - y_n &= r_n \end{aligned} \quad (3)$$

Donde las incógnitas son los coeficientes " c_1 " a " c_m " y el objetivo es minimizar la los residuos, es decir minimizar su sumatoria:

$$\sum_{i=1}^n r_i = \text{mínimo} \quad (4)$$

Puesto que en esta sumatoria los residuos (r_i) pueden tomar valores positivos o negativos y un valor positivo puede anular a otro negativo dando así un falso cero, se debe eliminar el signo, lo que se logra simplemente elevando los residuos al cuadrado. Por consiguiente el objetivo es minimizar el cuadrado de los residuos, es decir:

$$\sum_{i=1}^n r_i^2 = \text{mínimo} \quad (5)$$

Esta es la ecuación del método de los mínimos cuadrados y de la cual (como se puede ver) deriva su nombre. El mínimo puede ser encontrado empleando diferentes métodos de optimización. En nuestro caso y como ya dijimos emplearemos los principios del cálculo diferencial, donde sabemos que el mínimo puede ser calculado si se iguala la derivada primera a cero.

Antes de proceder a encontrar el mínimo de la ecuación (5), es conveniente escribir el sistema de ecuaciones (3) en forma matricial:

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \dots & f_m(x_1) \\ f_1(x_2) & f_2(x_2) & \dots & f_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ f_1(x_n) & f_2(x_n) & \dots & f_m(x_n) \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix} - \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_n \end{bmatrix} \quad (6)$$

O en forma simbólica:

$$F * C - Y = R \quad (7)$$

Entonces derivamos la ecuación (5) e igualamos la derivada a cero:

$$\frac{\partial}{\partial c_k} \left(\sum_{i=1}^n r_i^2 \right) = \sum_{i=1}^n \frac{\partial r_i^2}{\partial c_k} = \sum_{i=1}^n 2r_i \frac{\partial r_i}{\partial c_k} = 2 \sum_{i=1}^n r_i \frac{\partial r_i}{\partial c_k} = 0 \quad (8)$$

De donde obtenemos:

$$\sum_{i=1}^n r_i \frac{\partial r_i}{\partial c_k} = 0 \quad \{k=1 \rightarrow m\} \quad (9)$$

Que de manera desarrollada es:

$$\begin{aligned} r_1 \frac{\partial r_1}{\partial c_1} + r_2 \frac{\partial r_2}{\partial c_1} + \dots + r_n \frac{\partial r_n}{\partial c_1} &= 0 \\ r_1 \frac{\partial r_1}{\partial c_2} + r_2 \frac{\partial r_2}{\partial c_2} + \dots + r_n \frac{\partial r_n}{\partial c_2} &= 0 \\ &\vdots \\ r_1 \frac{\partial r_1}{\partial c_m} + r_2 \frac{\partial r_2}{\partial c_m} + \dots + r_n \frac{\partial r_n}{\partial c_m} &= 0 \end{aligned} \quad (10)$$

Este sistema de ecuaciones no depende de la forma que tenga la ecuación a la cual se quieren ajustar los datos, pudiendo ser empleada para ajustar datos a cualquier forma de expresión simbólica.

Aplicándola específicamente a la forma simbólica de la ecuación (1), es decir al sistema de ecuaciones (3), se puede constatar que las derivadas parciales con relación a los coeficientes son directamente iguales a sus funciones asociadas. Así por ejemplo la derivada parcial de r_2 con respecto a c_1 ($\partial r_2 / \partial c_1$) es igual a la función f_1 ($f_1(x_2)$); la derivada parcial de r_1 con respecto a c_2 , ($\partial r_1 / \partial c_2$) es igual a la función f_2 ($f_2(x_1)$) y en general para cualquier derivada se cumple la siguiente relación general:

$$\frac{\partial r_i}{\partial c_k} = f_k(x_i) \begin{cases} i=1 \rightarrow n \\ k=1 \rightarrow m \end{cases} \quad (11)$$

Reemplazando la ecuación (11) en la ecuación (10) obtenemos:

$$\begin{aligned} r_1 f_1(x_1) + r_2 f_1(x_2) + \dots + r_n f_1(x_n) &= 0 \\ r_1 f_2(x_1) + r_2 f_2(x_2) + \dots + r_n f_2(x_n) &= 0 \\ &\vdots \\ r_1 f_m(x_1) + r_2 f_m(x_2) + \dots + r_n f_m(x_n) &= 0 \end{aligned} \tag{12}$$

Que expresado en forma matricial es:

$$\begin{bmatrix} f_1(x_1) & f_1(x_2) & \dots & f_1(x_n) \\ f_2(x_1) & f_2(x_2) & \dots & f_2(x_n) \\ \vdots & \vdots & \ddots & \vdots \\ f_m(x_1) & f_m(x_2) & \dots & f_m(x_n) \end{bmatrix} * \begin{bmatrix} r_1 \\ r_2 \\ \vdots \\ r_m \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (13)$$

La matriz de funciones de la ecuación (12) es la transpuesta de la matriz de funciones de la ecuación (6). Por consiguiente la ecuación (13) puede ser escrita como:

$$F^T * R = 0 \quad (14)$$

Reemplazando ahora la ecuación (7) en la ecuación (14) resulta:

$$F^T*(F*C-Y)=0 \quad (15)$$

Realizando algunas operaciones y reordenando términos se tiene:

$$F^T * F * C = F^T * Y \quad (16)$$

La ecuación (16) es un sistema de "m" ecuaciones lineales con "m" incógnitas, donde las incógnitas son los valores de "C" ($c_1, c_2, \dots, c_m$) la matriz del sistema de ecuaciones se forma al multiplicar: $F^T * F$ y el vector de las constantes al multiplicación: $F^T * Y$. En consecuencia este sistema de ecuaciones lineales puede ser resuelto con los métodos estudiados en los temas 17 a 19.

22.1 Coeficientes de correlación

El coeficiente de correlación nos proporciona un valor numérico que nos permite determinar cuán bien se ajustan los datos a la expresión simbólica. Se calcula con la siguiente ecuación:

$$cor = \frac{\sqrt{\sum_{i=1}^n (y_{ci} - \bar{y})^2}}{\sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} \quad (17)$$

Donde "n" es el número de datos, " x_i " son los datos del vector "x" (variable independiente); " y_i " es el iésimo valor del vector "y" (variable dependiente), y_{ci} es el valor de la variable dependiente calculado al reemplazar " x_i " en la ecuación ajustada y $\bar{y}$ es el promedio de los datos del vector "y".

Si el coeficiente de correlación calculado con la ecuación (17) es 1, entonces el ajuste es perfecto, es decir la ecuación ajustada pasa por todos los puntos conocidos. Mientras más cercano sea a 1 el coeficiente de correlación, mejor es el ajuste, por el contrario mientras más alejado se encuentre de 1, peor es el ajuste.

22.2 Ejemplo

Para comprender mejor el método ajustaremos los siguientes datos:

x	1.0	2.5	3.5	4.0
y	3.8	15.0	26.0	33.0

A la ecuación:

$$y = a + bx^2$$

Que escrita en la forma general de la ecuación (1) es:

$$y = c_1 f_1(x) + c_2 f_2(x) = a * (1) + b * (x^2)$$

Donde como se puede observar: $c_1 = a$; $c_2 = b$; $f_1(x) = 1$ y $f_2(x) = x^2$, $m = 2$ (existen dos funciones) y $n = 4$ (existen cuatro datos). Entonces la matriz de funciones (F) es:

$$F = \begin{bmatrix} f_1(x_1) & f_2(x_1) \\ f_1(x_2) & f_2(x_2) \\ f_1(x_3) & f_2(x_3) \\ f_1(x_4) & f_2(x_4) \end{bmatrix}$$

Para calcular los valores de esta matriz primero guardamos los valores tabulados en dos vectores:

$vx = \{1, 2.5, 3.5, 4.0\}; vy = \{3.8, 15.0, 26.0, 33.0\};$

Con estos vectores creamos la tabla (matriz) de puntos y leemos el número de puntos:


```
p=Transpose[{vx,vy}]
{{1,3.8},{2.5,15.},{3.5,26.},{4.,33.}}
```

```
n=Length[p]
4
```

Creamos el vector de funciones y leemos el número de funciones:

```
f[x_]:=1,x^2}
```

```
m=Length[f[x]]
2
```

Creamos la matriz de funciones "F" y obtenemos su transpuesta:

```
F=Thread[f[p[[All,1]]]]
{{1,1},{1,6.25},{1,12.25},{1,16}}
```

```
FT=Transpose[F]
{{1,1,1,1},{1,6.25,12.25,16.}}
```

Calculamos la matriz de los coeficientes y el vector de las constantes:

```
A=FT.F
{{4.,35.5},{35.5,446.125}}
```

```
B=FT.vy
{77.8,944.05}
```

Entonces resolvemos el sistema de ecuaciones con "LinearSolve":

```
{a,b}=LinearSolve[A,B]
{2.27897,1.93476}
```

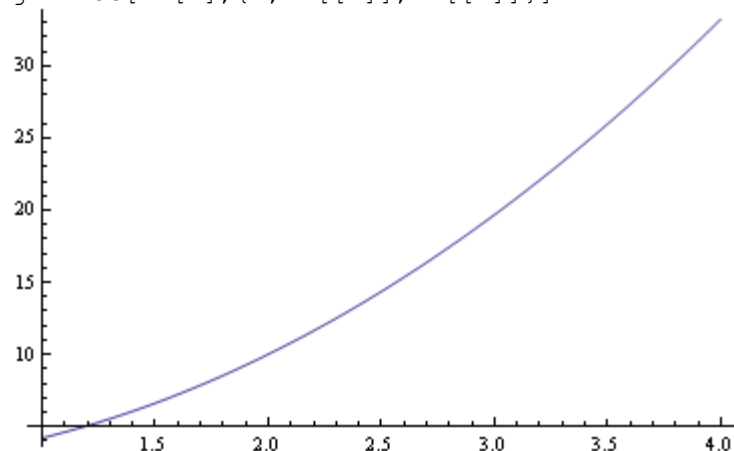
Siendo estos los coeficientes que ajustan los datos a la ecuación. En consecuencia la ecuación ajustada es:

```
fx[x_]:=Evaluate[{a,b}.f[x]]
```

```
fx[x]
2.27897 +1.93476 x2
```

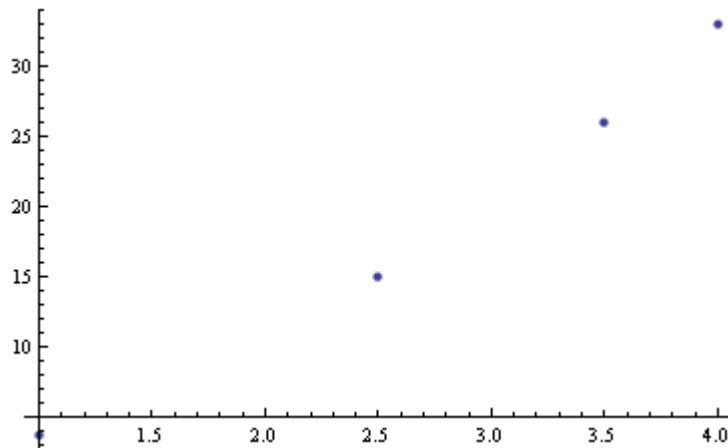
Para apreciar gráficamente cuán bien se ajustan los datos a la curva podemos graficarla:

```
g1=Plot[fx[x],{x,vx[[1]],vx[[n]]}]
```



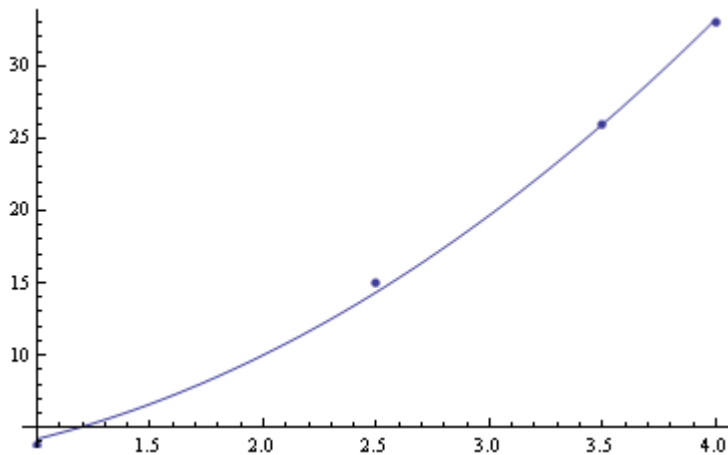
Graficamos los puntos conocidos:

```
g2=ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]
```



Y mostramos las dos gráficas en una (con Show):

```
Show[g1,g2]
```



Por supuesto es posible llevar a cabo el proceso con una sola instrucción:

```
Show[Plot[fx[x],{x,vx[[1]],vx[[n]]}],ListPlot[p,
PlotStyle->AbsolutePointSize[Medium]]]
```

Como se puede observar en este caso los datos se ajustan bastante bien a la ecuación (los puntos se encuentran bastante cercanos a la curva), por lo que podemos concluir que el ajuste es bueno. Para corroborar numéricamente esta apreciación, podemos calcular el coeficiente de correlación, para ello calculamos primero la media aritmética (con Mean):

```
ym=Mean[vy]
19.45
```

Y aplicamos la ecuación (17):

```
Sqrt[Sum[(fx[vx[[i]]]-ym)^2,{i,4}]/Sum[(vy[[i]]-ym)^2,{i,4}]]
0.999366
```

Al ser el valor del coeficiente de correlación cercano a uno, se corrobora nuestra apreciación (pues como se dijo mientras más cercano esté su valor de 1 mejor es el ajuste).

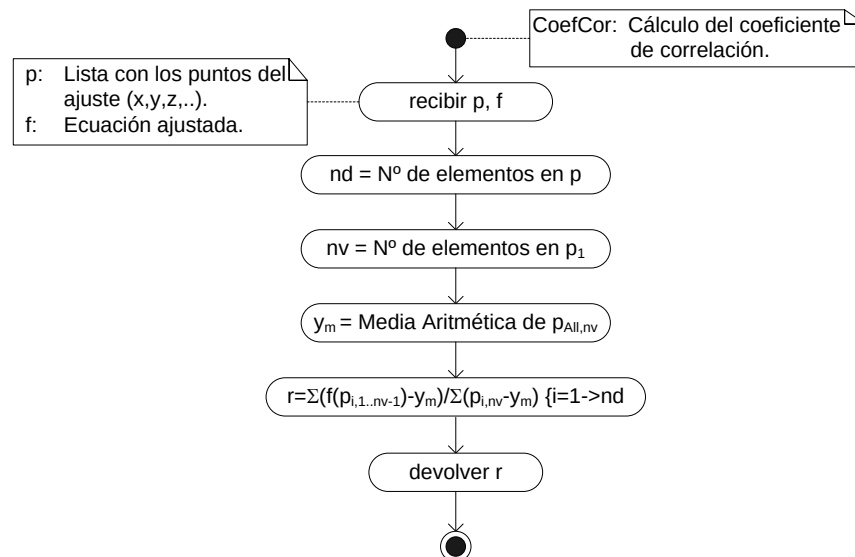
Ahora podemos emplear la ecuación ajustada como lo haríamos con cualquier ecuación analítica, empleándola por ejemplo para calcular los valores de "y" para los siguientes valores de "x": 2, 3, 3.5, 3.7, 4:

```
fx[{2,3,3.5,3.7,4}]
{10.018,19.6918,25.9798,28.7659,33.2352}
```

Como se puede observar, a pesar de ser el ajuste muy bueno, los valores calculados no corresponden exactamente a los valores conocidos, así para $x=3.5$ el valor calculado es 25.9798 y no 26 (que es el valor exacto) y para $x=3$ el valor calculado es 33.2352 y no 33 (que es el valor exacto). Esto es algo que siempre ocurre cuando se ajustan datos, no obstante, con frecuencia los valores ajustados son más confiables que los datos originales, debido a que el ajuste suele corregir o disminuir los errores experimentales de los datos originales.

22.2.1 Algoritmo y código

Puesto que el cálculo del coeficiente de correlación es de utilidad tanto para el método antes descrito como para cualquier otro método de ajuste, elaboraremos primero una función para el coeficiente de correlación, cuyo algoritmo es el siguiente:



Este algoritmo está pensado en el cálculo de los coeficientes de correlación no sólo con ajuste monovariable (como el estudiado), sino también multivariable como ocurre con los métodos que veremos más adelante. Por ello en el algoritmo se leen tanto el número de datos (nd) como el número de variables (nv). La ecuación ajustada "f" debe recibir como parámetros las variables independientes.

El código elaborado en base al algoritmo y codificado en el paquete "Ajuste" es el siguiente:

```
BeginPackage["Ajuste`"]
```

```
CoefCor::usage="CoefCor[p,f], calcula el coeficiente de correlación
de la ecuación ajustada \"f\" (forma f[x_,y_,...]:=...), siendo los datos
del ajuste \"p\" (forma: {{x1,y1,z1,...},{x2,y2,z2,...},...,
{xn,yn,zn,...}})."
```

```
Begin["Private`"]
```

```
CoefCor[p_List,f_] := Module[{ym,nd,nv},
  nd=Length[p];
```

```

nv=Length[p[[1]]];
ym:=Mean[p[[All,nv]]];
Sqrt[Sum[(f[p[[i,1];nv-1]]-ym)^2,{i,nd}]/
      Sum[(p[[i,nv]]-ym)^2,{i,nd}]][[1]]

End[]

EndPackage[]

Podemos probar este programa con los datos del ejemplo:
Get["Ajuste`",Path->"E:\MAT205\"]

```

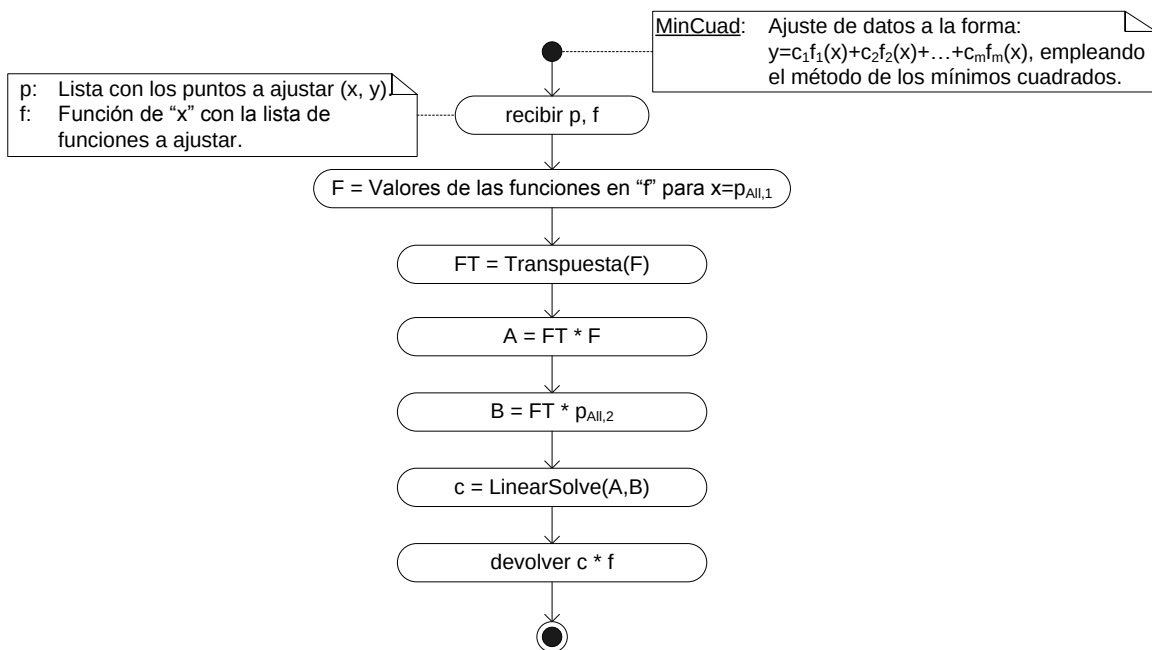
```

CoefCor[p,fx]
0.999366

```

Y como se ve se obtienen el mismo resultado, por lo que podemos asumir que el programa ha sido correctamente elaborado.

El algoritmo para el método de mínimos cuadrados (monovariable), es el siguiente:



El programa elaborado en base al mismo y añadido al paquete "Ajuste`" es:

```

BeginPackage["Ajuste`"]

```

...

```

MinCuad::usage="MinCuad[p,f]], ajusta los datos de la matriz \"p\"
a la ecuación  $y=c_1f_1(x)+c_2f_2(x)+\dots+c_mf_m(x)$ , siendo \"f\" la lista
con las funciones \"fi(x)\", de la ecuación (forma  $f[x_]:= \{f_1(x),f_2(x),\dots,f_m(x)\}$ ). MinCuad devuelve la ecuación ajustada."

```

```

Begin["Private`"]

```

...

```

MinCuad[p_List,f_]:=Module[

```

```

    {F, FT, A, B, c},
    F=Thread[f[p[[All,1]]]];
    FT=Transpose[F];
    A=FT.F;
    B=FT.p[[All,2]];
    c=LinearSolve[A,B];
    c.f[Global`x]]
End[]

EndPackage[]

```

Podemos probar este programa con los datos del ejemplo:

```

Get["Ajuste`",Path->"E:\MAT205\"];

p={{1,3.8},{2.5,15},{3.5,26},{4,33}};

f[x_]:= {1,x^2}

MinCuad[p,f]
2.27897 +1.93476 x^2

```

Y como se puede observar se obtiene los mismos resultados del ejemplo, por lo que podemos asumir que el programa ha sido correctamente elaborado.

22.3 Funciones de Mathematica para el ajuste de datos

Al ser el ajuste de datos un problema de optimización (minimización), se pueden emplear funciones como "FindMinimum" y "NMinimize" para ajustar datos a expresiones algebraicas, sin embargo, Mathematica cuenta con tres funciones especializadas para el ajuste de datos: "LeastSquares", "Fit" y "FindFit".

"LeastSquares", como su nombre sugiere, realiza el ajuste por el método de mínimos cuadrados estudiado en el anterior acápite, aunque permite elegir entre varios métodos internos disponibles. Su sintaxis es la siguiente:

$$\text{LeastSquares}[m,b]$$

Donde "m" es la matriz de las funciones para los valores conocidos de "x" (la variable independiente) y "b" es la matriz (o vector) con los valores de la variable independiente. "LeastSquares" permite realizar más de un ajuste a la vez si dichos ajustes sólo difieren en los valores de la variable independiente, en cuyo caso se mandan dichos valores (en forma de matriz) como el parámetro "b". "LeastSquares" devuelve los coeficientes (c_i) de la ecuación ajustada.

Así para resolver el problema del ejemplo con "LeastSquares" escribimos lo siguiente:

```

p={{1,3.8},{2.5,15},{3.5,26},{4,33}};

f[x_]:= {1,x^2}

m=Thread[f[p[[All,1]]]]
{{1,1},{1,6.25},{1,12.25},{1,16}}

b=p[[All,2]]
{3.8,15,26,33}

LeastSquares[m,b]
{2.27897,1.93476}

```

Por lo tanto la ecuación ajustada es:

```
%f[x]
2.27897 +1.93476 x2
```

Por otro lado "Fit" también sigue el proceso de cálculo explicado en el anterior acápite, aunque amplía su aplicación pues permite el ajuste a funciones con dos o más variables (ajuste multivariable), su sintaxis es la siguiente:

Fit[puntos, funciones, variables]

Donde "puntos" es la matriz con los datos de las variables: una columna por cada variable, siendo la última columna la de la variable dependiente (valor de la función). "Funciones" es la lista con las funciones existentes en la expresión simbólica a ajustar (ecuación 1). "Variables" es la lista de variables independientes (si es una sola, no es necesario escribirla dentro de una lista).

Así para resolver el ejemplo con "Fit" escribimos lo siguiente:

```
p={{1,3.8},{2.5,15},{3.5,26},{4,33}};

f={1,x^2};

Fit[p,f,x]
2.27897 +1.93476 x2
```

Y como se puede ver se obtiene el mismo resultado que con el anterior método y con el ejemplo.

La otra función de Mathematica: "FindFit", es más general que "Fit" pues no está limitada a la forma de la ecuación 1, permitiendo el ajuste de datos a cualquier forma de ecuación. Su sintaxis es:

FindFit[Puntos, Ecuación, Coeficientes, Variables]

Donde "Puntos" (al igual que en "Fit") son los datos a ajustar (una columna por cada variable, siendo la última columna la de la variable dependiente), "Ecuación" es la expresión simbólica a la cual se ajustan los datos, "Coeficientes" son los coeficientes o parámetros de la ecuación que deben ser calculados, "FindFit" permite asignar valores iniciales a los coeficientes, para que la búsqueda de la solución comience con los mismos. "Variables" es la lista de variables independientes (como de costumbre, si es una no es necesario escribir una lista).

Así para resolver el ejemplo del acápite anterior con "FindFit" escribimos el siguiente módulo:

```
Remove[a,b,x,f]

p={{1,3.8},{2.5,15},{3.5,26},{4,33}};

f[x_]:=a+b*x^2

FindFit[p,f[x],{a,b},x]
{a→2.27897,b→1.93476}
```

Por lo tanto la ecuación ajustada es:

```
f[x]/.%
2.27897 +1.93476 x2
```

22.4 Ejemplo

Ajuste los datos que se presentan en la tabla a la ecuación: $y = a + bx^2 + cx^4 + d \ln(x)$, calcule el coeficiente de correlación de la ecuación ajustada, grafique la función y los datos y calcule los valores de "y" para los siguientes valores de "x": 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. El problema debe ser resuelto empleando: a) MinCuad, b) LeastSquares, c) "Fit" y d) "FindFit".

x	4.0	3.2	6.8	9.3	1.2	0.7	5.5	7.4	6.5	2.4
Y	6.2	5.3	9.1	12.0	3.3	1.5	7.9	10.6	8.7	6.3

a) Con "MinCuad":

Primero importamos el paquete "Ajuste":

```
Get["Ajuste`", Path->"E:\HPV\MAT205\1-2009\CUADERNOS MATHEMATICA"]
```

Guardamos en las variables "p" y "vx" los puntos a ajustar y los valores de "x" para los cuales se calcularán los respectivos valores de "y":

```
p={{4,6.2},{3.2,5.3},{6.8,9.1},{9.3,12},{1.2,3.3},{0.7,1.5},{5.5,7.9},{7.4,10.6},{6.5,8.7},{2.4,6.3}};
```

```
vx=Range[10];
```

Creamos la función con la lista de funciones de la ecuación a ajustar:

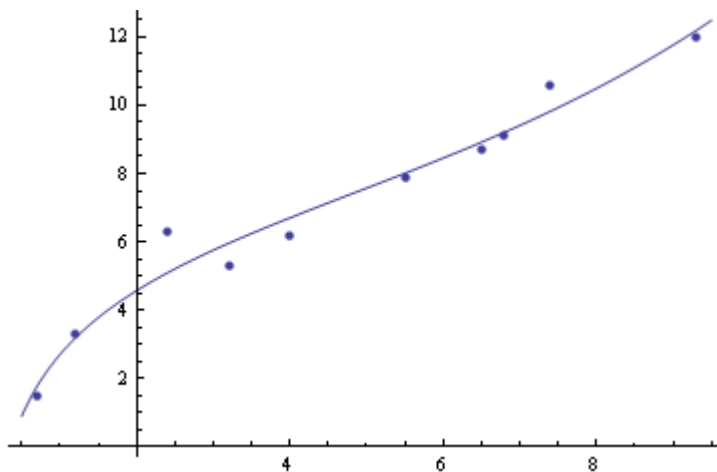
```
f[x_]:= {1,x^2,x^4,Log[x]}
```

Encontramos la ecuación ajustada y la guardamos en la función "fx":

```
fx[x_]:=Evaluate[MinCuad[p,f]];fx[x]
2.689 +0.0229008 x^2+0.000232866 x^4+2.59102 Log[x]
```

Graficamos la ecuación ajustada y los datos conocidos:

```
Show[Plot[fx[x],{x,Min[p[[All,1]]]-0.2,Max[p[[All,1]]]+0.2}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Observe que en este caso los límites de la gráfica se encuentran con las funciones "Min" y "Max", esto debido a que los datos no están ordenados, además para que el primer y último punto queden claramente visibles se le resta 0.2 al límite inferior y se le suma 0.2 al superior.

El coeficiente de correlación se calcula con el programa "CoefCor":

```
CoefCor[p,fx]
```

0.984245

En este caso, aún cuando algunos de los puntos no son muy cercanos a la curva ajustada, el coeficiente de correlación nos indica que se trata de un buen ajuste (cercano a uno), esto se debe a que dichos puntos se encuentran proporcionalmente distribuidos a ambos lados de la curva, compensando así sus errores.

Finalmente los valores de "y" se calculan mandando a la función "fx" el vector "vx":

```
fx[vx]
{2.71213,4.58028,5.76049,6.70694,7.57714,8.4577,9.41213,10.4963,11.7648,13.2738}
```

b) Con "LeastSquares":

Asumiendo que los datos introducidos para el inciso "a" se encuentran aún en memoria calculamos primero los coeficientes de la ecuación ajustada:

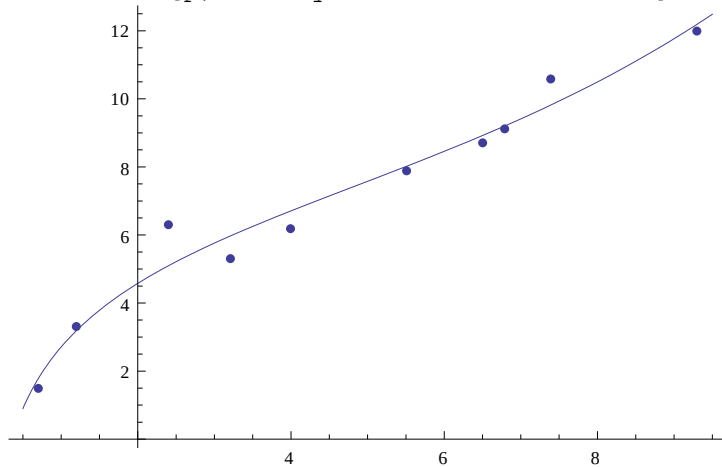
```
{a,b,c,d}=LeastSquares[Thread[f[p[[All,1]]]],p[[All,2]]]
{2.689,0.0229008,0.000232866,2.59102}
```

Con estos coeficientes y la función "f" determinamos la ecuación ajustada:

```
fx2[x_]:=Evaluate[{a,b,c,d}.f[x]];fx2[x]
2.689 +0.0229008 x^2+0.000232866 x^4+2.59102 Log[x]
```

Como era de esperar se obtienen los mismos resultados que con "MinCuad". Por supuesto lo mismo ocurre con la gráfica, el coeficiente de correlación y los valores de "y" calculados:

```
Show[Plot[fx2[x],{x,Min[p[[All,1]]]-0.2,Max[p[[All,1]]]+0.2}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



```
CoefCor[p,fx2]
0.984245
```

```
fx2[vx]
{2.71213,4.58028,5.76049,6.70694,7.57714,8.4577,9.41213,10.4963,11.7648,13.2738}
```

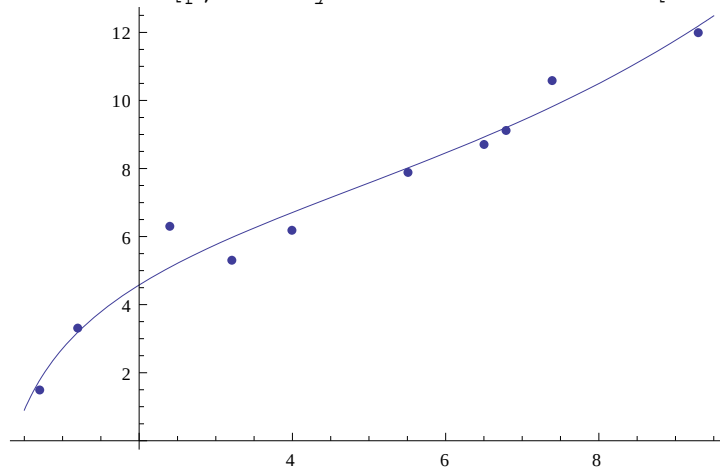
c) Con "Fit":

Como era de suponer con "Fit" se obtienen también los mismos resultados:

```
fx3[x_]:=Evaluate[Fit[p,f[x],x]];fx3[x]
2.689 +0.0229008 x^2+0.000232866 x^4+2.59102 Log[x]
```



```
Show[Plot[fx3[x], {x, Min[p[[All, 1]]] - 0.2, Max[p[[All, 1]]] + 0.2}],
  ListPlot[p, PlotStyle -> AbsolutePointSize[Medium]]]
```



```
CoefCor[p, fx3]
0.984245
```

```
fx3[vx]
{2.71213, 4.58028, 5.76049, 6.70694, 7.57714, 8.4577, 9.41213, 10.4963, 11.7648, 13.2738}
```

d) Con "FindFit":

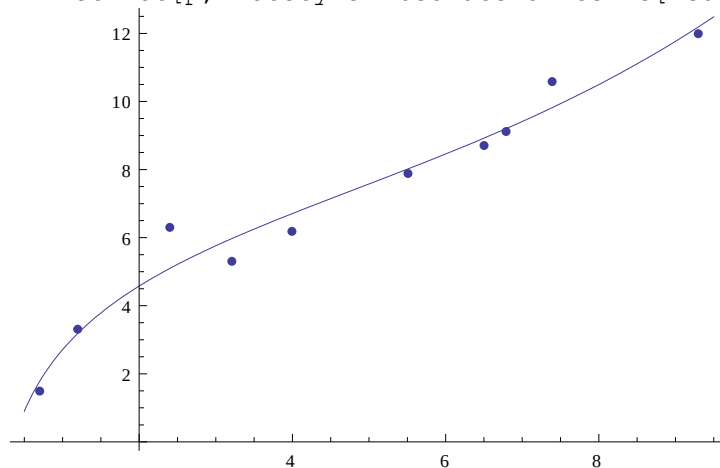
Como era de esperar con "FindFit" también se obtienen los mismos resultados:

```
Remove[a, b, c, d]
```

```
f2={a, b, c, d}.f[x]
a+b x^2+c x^4+d Log[x]
```

```
fx4[x_]:=Evaluate[f2/.Evaluate[FindFit[p, f2, {a, b, c, d}, x]]];
fx4[x]
2.689 +0.0229008 x^2+0.000232866 x^4+2.59102 Log[x]
```

```
Show[Plot[fx4[x], {x, Min[p[[All, 1]]] - 0.2, Max[p[[All, 1]]] + 0.2}],
  ListPlot[p, PlotStyle -> AbsolutePointSize[Medium]]]
```



```
CoefCor[p, fx4]
0.984245
```

```
fx4[vx]
```

{2.71213, 4.58028, 5.76049, 6.70694, 7.57714, 8.4577, 9.41213, 10.4963, 11.7648, 13.2738}

22.5 Ejercicios

1. Elabore un programa para el método de los mínimos cuadrados empleando "RowReduce" en lugar de "LinearSolve".
2. Elabore un programa para el método de los mínimos cuadrados empleando "Cholesky" en lugar de "LinearSolve".
3. Elabore el programa para el método de los mínimos cuadrados empleando "GaussJordanpt" en lugar de "LinearSolve".
4. Elabore el programa para el método de los mínimos cuadrados empleando "Reduce" en lugar de "LinearSolve".
5. Elabore el programa para el método de los mínimos cuadrados empleando "FindInstance" en lugar de "LinearSolve".
6. Elabore el programa para el método de los mínimos cuadrados empleando "NSolve" en lugar de "LinearSolve".
7. Empleando "MinCuad" ajuste los datos de la siguiente tabla a la ecuación: $y = a*x^2 + b/x + c*e^{x/8}$, luego grafique la ecuación ajustada y los puntos conocidos, calcule su coeficiente de correlación y calcule los valores de "y" para los siguientes valores de "x": 2, 4, 7, 9, 11, 12, 13, 14, 16, 17, 18.

X	2.3	4.5	6.7	8.4	9.2	10.1	11.2	12.6	14.5	16.3	17.2	18.1
Y	11	41	90	141	169	205	251	318	421	545	592	655

8. Repita el ejercicio 7, empleando "LeastSquares".
9. Repita el ejercicio 7, empleando "Fit".
10. Repita el ejercicio 7, empleando "FindFit".
11. Empleando "MinCuad" ajuste los datos de la siguiente tabla a la ecuación: $y = a*x^3 + b*x^{0.5} + c/x^{1/3}$, luego grafique la ecuación ajustada y los puntos conocidos, calcule el coeficiente de correlación y resuelva la ecuación ajustada empleando el método "Incremental" (valor inicial 2, incremento 1).

x	3.8	1.3	9.2	3.4	1.0	5.1	5.7	9.0	8.0	6.4	6.6	8.1
y	15	8	124	13	8	27	35	116	84	46	50	87

12. Repita el ejercicio 11, empleando "LeastSquares" y "Secante".
13. Repita el ejercicio 11, empleando "Fit" y "FindRoot".
14. Repita el ejercicio 11, empleando "FindFit" y "NewtonDN".
15. Empleando "MinCuad" ajuste los datos de la siguiente tabla a la ecuación: $y = a*\ln(x) + b*\sin(x) - c*\cos(x)$, grafique la ecuación ajustada y los puntos conocidos, calcule el coeficiente de correlación y resuelva la ecuación ajustada empleando "RegulaFalsi" encontrando el segmento de solución con "Incret1" (valor inicial 5, incremento -1).

x	1	3	4	5	8	11	14	15	20

y	1	3	2	2	5	4	6	6	7

16. Repita el ejercicio 15, empleando "LeastSquares" y "Biseccion".
17. Repita el ejercicio 15, empleando "Fit" y "Wegstein".
18. Repita el ejercicio 15, empleando "FindFit" y "FindRoot".
19. Elabore un módulo que empleando "FindFit" ajuste los datos de la siguiente tabla a la ecuación: $z = (a+b*\ln(x)+c*\ln(y))/(d/x+e/y)$, luego grafique la ecuación ajustada y los puntos conocidos, calcule el coeficiente de correlación y calcule los valores de "z" para los siguientes valores de "x" = {1, 2.2, 3.3, 4.4, 5.6, 7.7, 8.8} y "y" = {1.5, 2.3, 3.5, 4.8, 5.9, 7.9, 9.0}. nota: "e" en la ecuación no es el número de Nepper, sino uno de los coeficientes (o parámetros) de la ecuación.

X	2.0	1.4	4.0	3.5	7.4	6.2	1.1	3.0	5.5	7.3	2.9	3.2
Y	2.3	4.4	3.2	2.5	5.2	3.3	6.1	4.2	2.1	1.3	2.6	2.7
Z	1.71	1.36	4.51	3.44	10.4	6.93	1.03	3.72	4.68	3.94	2.86	3.26

20. Repita el ejercicio 19 empleando "Fit".

23. INTERPOLACIÓN POLINOMIAL

En el tema anterior hemos estudiado uno de los caminos que se puede seguir cuando se trabaja con datos tabulados: el ajuste de datos. En este capítulo estudiaremos otra alternativa: la interpolación de datos.

Se prefiere la interpolación sobre el ajuste, cuando los datos son confiables. Esta situación se presenta cuando dichos datos son el resultado de mediciones experimentales en equipos de alta precisión o el resultado de relaciones matemáticas, como los de la siguiente tabla:

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

En general los diferentes métodos de interpolación nos permiten trabajar con los datos tabulados de manera análoga a una función analítica: nos devuelven el valor de la variable dependiente "y" para un valor dado de la variable independiente "x", que es lo mismo que hacen las funciones analíticas. En ese sentido los diferentes métodos de interpolación se comportan como generadores de funciones arbitrarias, con la limitante de que el valor de la variable independiente (x) debe estar dentro del rango de los valores tabulados. Para predecir valores fuera del rango de valores tabulados se requiere de otra técnica conocida como "extrapolación".

Los métodos de interpolación a su vez se dividen en dos: a) Interpolación polinomial, en la cual el valor interpolado se calcula mediante una ecuación polinomial de grado (n-1) que pasa por todos los puntos y b) Interpolación segmentaria, en la cual para cada par de puntos se calculan los coeficientes de una ecuación (normalmente una ecuación cúbica) y luego se emplean dichas ecuaciones para calcular el valor interpolado. En este tema estudiaremos algunos de los métodos de interpolación polinomial, mientras que los métodos de interpolación segmentaria serán estudiado en el siguiente tema.

Sin embargo, antes de pasar a estudiar la interpolación polinomial, estudiaremos primero el método de interpolación más sencillo conocido: la interpolación lineal.

23.1. INTERPOLACIÓN LINEAL

A pesar de ser un método de poca precisión, es, por su simplicidad, uno de los métodos más empleados en la práctica, sobre todo en el área de la ingeniería donde en la mayoría de los casos sólo se requieren valores aproximados. Este método proporciona resultados satisfactorios cuando los datos están distribuidos en segmentos cercanos, o cuando, a pesar de estar espaciados, se ajustan aproximadamente a una línea recta.

En el método de interpolación lineal se busca el segmento donde se encuentra el valor de la variable independiente "x", con los dos puntos de dicho segmento se calculan los coeficientes de la ecuación lineal y con la misma se calcula el valor de la variable dependiente "y". Si el valor de "x" es mayor al último valor tabulado, o menor al menor valor tabulado, se puede emplear el último par de puntos (o el primer par de puntos) para obtener el valor de "y" por extrapolación.

Si (x_i, y_i) y (x_j, y_j) son los puntos del segmento donde se encuentra el valor conocido: "x", entonces los coeficientes de la línea recta se calculan con:

$$b = \frac{y_j - y_i}{x_j - x_i} \quad (1)$$

$$a = y_i - bx_i \quad (2)$$

El valor interpolado: "y", se calcula con la ecuación de la línea recta:

$$y = a + bx \quad (3)$$

Se debe hacer notar que debido la necesidad de ubicar el segmento donde se encuentra el valor conocido, los datos deben estar ordenados ascendentemente en función a los valores de la variable independiente.

Para ilustrar el método interpolaremos el valor de "y" para un valor de "x" igual a 0.7 aplicando el método de interpolación lineal a los siguientes datos:

X	0	0.2	0.4	0.6	0.8	1.0
Y	0	0.199	0.389	0.565	0.717	0.841

Primero guardamos los datos de la tabla y el valor conocido de "x":

```
vx=Range[0,1,0.2]
{0.,0.2,0.4,0.6,0.8,1.}

vy={0,0.199,0.389,0.565,0.717,0.841}
{0,0.199,0.389,0.565,0.717,0.841}

x=0.7
0.7
```

Ahora ubicamos el segmento en el cual se encuentra el valor de "x". Para ello simplemente recorremos el vector "vx" mientras el valor de "x" (0.7) sea mayor a los elementos del vector (y existan elementos en el mismo). Algo que podemos hacer con la estructura iterativa "While":

```
j=2;While[x>vx[[j]] && j<Length[vx],j++];j
5
```

Este número (5) nos informa que el valor de "x" se encuentra entre el cuarto ($x_4=0.6$) y quinto elemento ($x_5=0.8$), como efectivamente ocurre. Observe que la búsqueda comienza a partir del segundo elemento ($j=2$), esto para evitar que ocurra un error en caso de que el valor buscado sea menor al primero, pues en ese caso y si la búsqueda comienza en 1, el resultado sería 1, indicado que la solución se encuentra entre el elemento 0 (que no existe) y el primero.

Por supuesto también podemos encontrar el segmento con otra estructura iterativa como "For" por ejemplo:

```
For[j=2,x>vx[[j]] && j<Length[vx],j++];j
5
```

O con Do:

```
Do[If[x<vx[[i]],j=i;Break[]},{i,2,Length[vx]}};j
5
```

Una vez ubicado el segmento de interpolación (los puntos entre los cuales se encuentra el valor buscado) aplicamos las ecuaciones (1) y (2) para calcular los coeficientes:

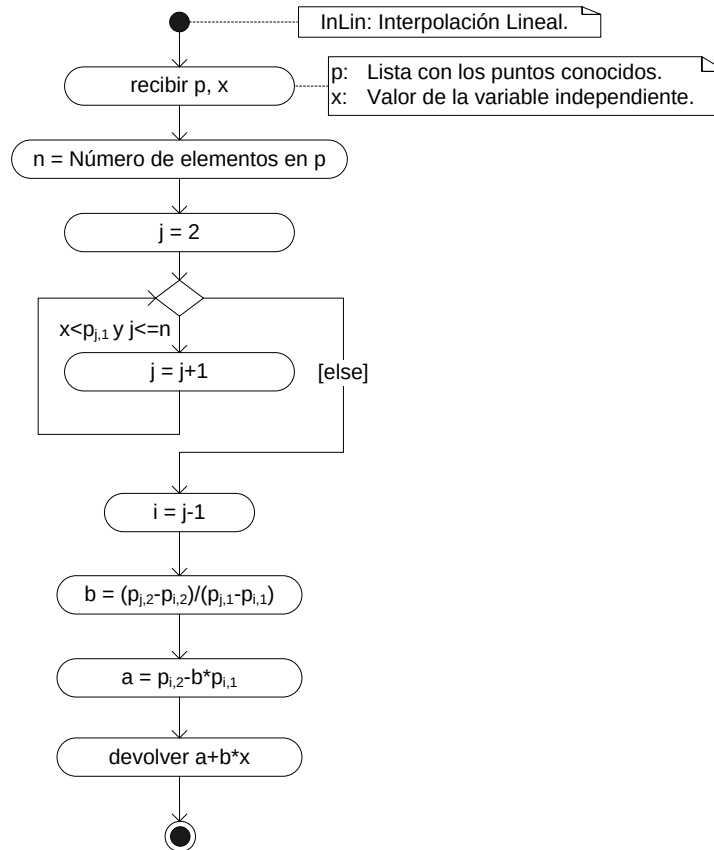
```
i=j-1;b=(vy[[j]]-vy[[i]])/(vx[[j]]-vx[[i]])
0.76
a=vy[[i]]-b*vx[[i]]
0.109
```

Finalmente el valor interpolado se calcula con la ecuación (3):

$y=a+b*x$
0.641

23.1.1. Algoritmo y código

A pesar de ser un proceso muy sencillo es conveniente automatizar el cálculo para la interpolación lineal. El algoritmo para dicho proceso es el siguiente:



Y el código elaborado en base al mismo y añadido al paquete Ajuste es:

```
BeginPackage["Ajuste`"]
```

...

```
InLin::usage="InLin[p,x], devuelve el valor de \"y\" para el valor
conocido de \"x\", a partir de la lista de puntos (x,y) contenidos
en \"p\", empleando el método de interpolación lineal."
```

```
Begin["Private`"]
```

...

```
InLin[p_List,x_?NumberQ]:=Module[{n,i,j=2,a,b},
  n=Length[p];
  While[x>p[[j,1]] && j<n,j++];
  i=j-1;
  b=(p[[j,2]]-p[[i,2]])/(p[[j,1]]-p[[i,1]]);
  a=p[[i,2]]-b*p[[i,1]];
  a+b*x]
```

```

a+b*x]

End[]

EndPackage[]

Para probar el código volvemos a realizar la interpolación del ejemplo:
Get["Ajuste`",Path->"E:\MAT205 "]

vx=Range[0,1,0.2]
{0.,0.2,0.4,0.6,0.8,1.}

vy={0,0.199,0.389,0.565,0.717,0.841}
{0,0.199,0.389,0.565,0.717,0.841}

p=Transpose[{vx,vy}]
{{0.,0},{0.2,0.199},{0.4,0.389},{0.6,0.565},{0.8,0.717},{1.,0.841}}

InLin[p,0.7]
0.641

```

Y como se ve se obtiene el mismo resultado, por lo que podemos asumir que el código es correcto.

Ahora podemos emplear el programa para interpolar otros valores, por ejemplo para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.1.

```

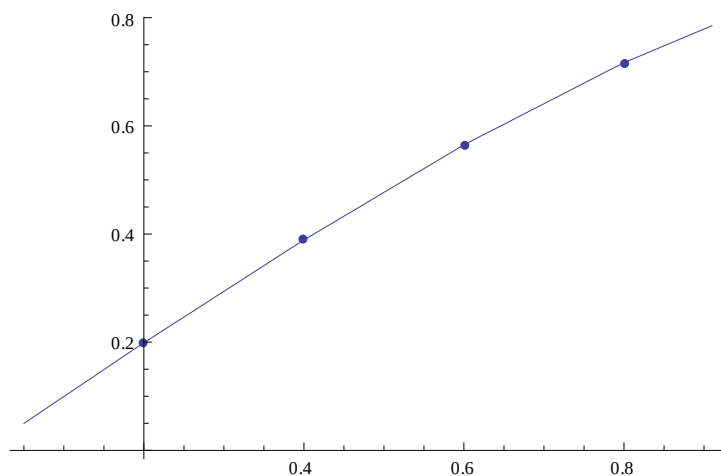
vc=Range[0.1,1.2,0.2]
{0.1,0.3,0.5,0.7,0.9,1.1}

Table[InLin[p,i],{i,vc}]
{0.0995,0.294,0.477,0.641,0.779,0.903}

Podemos analizar gráficamente si el ajuste lineal resulta o no adecuado
para este caso:

Show[Plot[InLin[p,x],{x,0.05,0.91}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]

```



Como se puede observar, en este caso la interpolación lineal representa muy bien el comportamiento de los puntos: la curva se ve continua, a pesar de tratarse de una serie de segmentos lineales.

23.1.2. Ejercicio

Elabore un módulo que empleando el método de interpolación lineal y los datos de la tabla que se presenta al final de la pregunta, calcule valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45. Grafique también los resultados de la interpolación lineal conjuntamente los puntos conocidos para determinar visualmente si la interpolación lineal resulta o no adecuado para estos datos.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

```
vx={3,5,7,11,14,16,19,21,24,25,26,27,29,33,36,40,44};
```

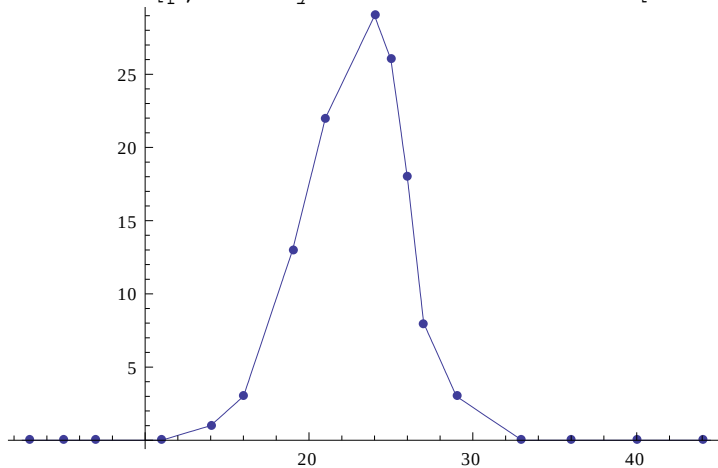
```
vy={0,0,0,0,1,3,13,22,29,26,18,8,3,0,0,0,0,0};
```

```
p=Transpose[{vx,vy}];
```

```
vcx={1,4,6,13,17,23,25.5,26.5,28,30,42,45};
```

```
Table[InLin[p,x],{x,vcx}]
{0,0,0,2/3,19/3,80/3,22.,13.,11/2,9/4,0,0}
```

```
Show[Plot[InLin[p,x],{x,2.5,44.5}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Como se puede apreciar, en algunos sectores, sobre todo en la cúspide de la curva, la interpolación lineal no devuelve en este caso resultados muy precisos, por lo que sería recomendable trabajar con un mayor número de puntos en este sector.

23.2. INTERPOLACIÓN POLINOMIAL EN MATHEMATICA

Mathematica cuenta con una función de interpolación polinomial: "InterpolatingPolynomial", que tiene la siguiente sintaxis:

InterpolatingPolynomial[datos, variables]

Donde "datos" es la matriz con los puntos a interpolar: la primera columna corresponde a la variable independiente y la segunda a la variable dependiente. Sin embargo, "InterpolatingPolynomial" no sólo permite interpolación monovariable, sino también multivariable, en cuyo caso las primeras columnas corresponden a las variables independientes y la última a la variable depen-

diente; "variables" es una lista con los nombres de las variables independientes. "InterpolatingPolynomial" devuelve una función de interpolación (en función de la o las variables independientes) que puede ser empleada directamente para encontrar los valores interpolados.

23.2.1. Ejemplos

1. Como primer ejemplo emplearemos "InterpolatingPolynomial" para calcular, con los datos de la tabla que se presenta al final del párrafo, valores de "y" para valores de "x" iguales a 2 y 5.5. Graficaremos también la función de interpolación resultante y los puntos conocidos, para juzgar visualmente la confiabilidad de los valores interpolados.

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

Primero guardamos los datos en una matriz:

```
p={{1,0},{4,1.3862944},{5,1.6094379},{6,1.7917595}};
```

Creemos la función de interpolación con "InterpolatingPolynomial":

```
f[x_]:=Evaluate[Expand[InterpolatingPolynomial[p,x]]]
```

Cuya forma es:

```
f[x]
-0.858364+0.988892 x-0.138394 x^2+0.00786554 x^3
```

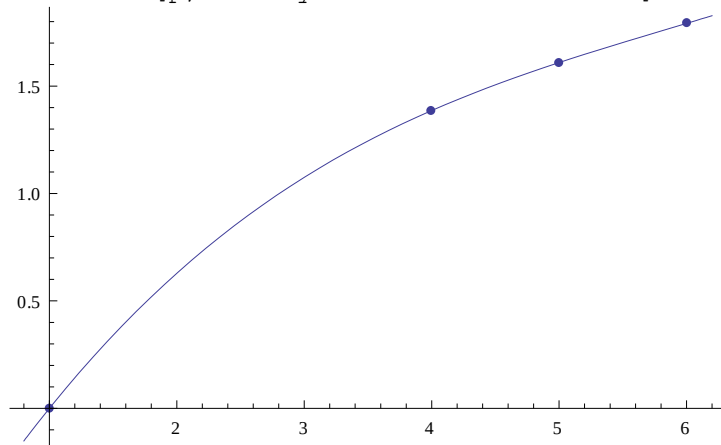
Que como era de esperar corresponde a una ecuación de tercer grado (es decir el número de puntos menos 1). Ahora empleamos la función de interpolación para calcular los valores de "y":

```
f[2]
0.628769
```

```
f[5.5]
1.70275
```

La gráfica de la función de interpolación y los puntos es:

```
Show[Plot[f[x],{x,0.8,6.2}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Podemos decir entonces que los valores interpolados son confiables, pues la curva resultante del ajuste es continua y uniforme en todo el intervalo de los puntos.

2. Como segundo ejemplo emplearemos "InterpolatingPolynomial" para calcular, con los datos de la tabla que se presenta al final del párrafo, valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.1. Al igual que el caso anterior graficaremos también la función de interpolación resultante para juzgar la confiabilidad de los valores interpolados.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

La solución de este problema es esencialmente la misma que en el caso anterior, es decir primero guardamos los datos:

```
p={{0,0},{0.2,0.199},{0.4,0.389},{0.6,0.565},{0.8,0.717},{1.0,.841}};
```

Guardamos en un vector los valores conocidos de la variable dependiente:

```
vc=Range[0.1,1.1,0.2]
{0.1,0.3,0.5,0.7,0.9,1.1}
```

Generamos la función de interpolación con "InterpolatingPolynomial":

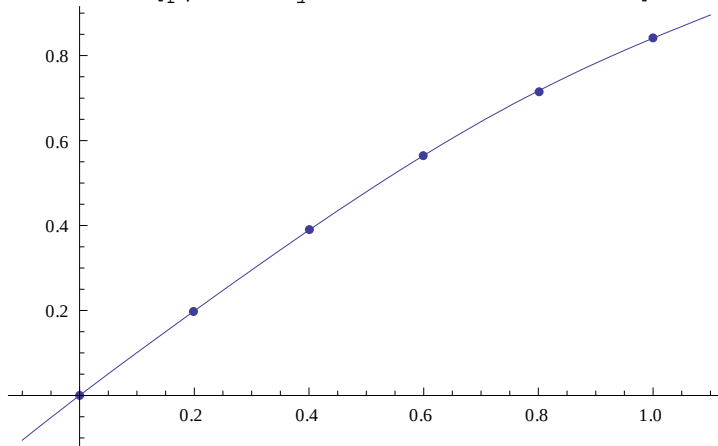
```
f[x_]:=Evaluate[Expand[InterpolatingPolynomial[p,x]]];f[x]
0. +1.02642 x-0.221875 x^2+0.453125 x^3-0.703125 x^4+0.286458 x^5
```

Y empleando la función calculamos los valores interpolados de la variable dependiente:

```
f[vc]
{0.100809,0.295191,0.479387,0.64452,0.782215,0.895598}
```

La gráfica de la función de interpolación es:

```
Show[Plot[f[x],{x,-0.1,1.1}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Y puesto que es uniforme en todo el intervalo de puntos conocidos, podemos afirmar que los valores interpolados son confiables.

3. Como tercer ejemplo emplearemos "InterpolatingPolynomial" para calcular, con los datos de la tabla que se presentan al final del párrafo, valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45. Al igual que en los dos anteriores casos, graficaremos la función de interpolación resultante para juzgar la confiabilidad de los valores interpolados.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

La solución es prácticamente la misma que con el método de Newton:

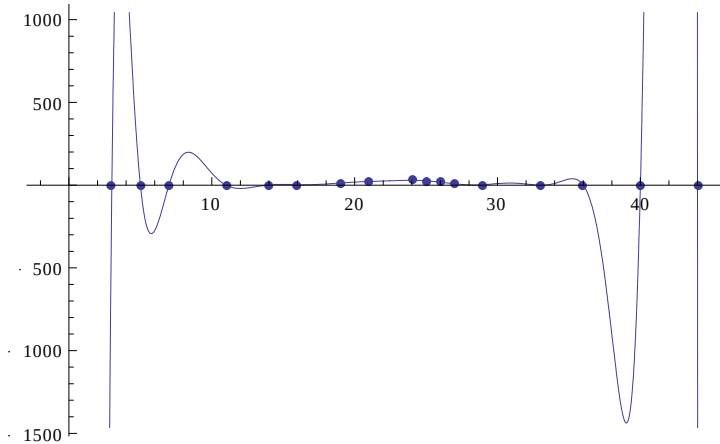
```
p={{3,0},{5,0},{7,0},{11,0},{14,1},{16,3},{19,13},{21,22},{24,29},{25,26},{26,18},{27,8},{29,3},{33,0},{36,0},{40,0},{44,0}};
```

```
vc={1,4,6,13,17,23,25.5,26.5,28,30,42,45};
```

```
f[x_]:=Evaluate[Expand[InterpolatingPolynomial[p,x]]];f[x]/N
-663163.+832434. x-455384. x^2+145434. x^3-30583.9 x^4+4519.84 x^5-488.215
x^6+39.4954 x^7-2.42746 x^8+0.114088 x^9-0.00409759 x^10+0.000111535 x^11-2.26002×10^-
6 x^12+3.30113×10^-8 x^13-3.28289×10^-10 x^14+1.98853×10^-12 x^15-5.5334×10^-15 x^16
```

```
f[vc]/N
{-167194.,1387.17,-274.195,-
10.7162,3.01998,27.8286,22.5341,12.9156,1.70822,9.34965,23896.,-298044.}
```

```
Show[Plot[f[x],{x,-2,45}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Una vez más los resultados son los mismos que con el método de Newton (y Lagrange) y se corrobora una vez más que

23.3. INTERPOLACIÓN DE LAGRANGE

Ahora veremos el primer método de interpolación polinomial: El método de Lagrange. En este método la ecuación de interpolación es:

$$y = \sum_{k=1}^n y_k L_k(x) \quad (4)$$

Donde:

$$L_k(x) = \frac{P_k(x)}{Q_k(x)} \quad \{k=1 \rightarrow n\} \quad (5)$$

$$P_k(x) = \prod_{\substack{i=1 \\ i \neq k}}^n (x - x_i) \quad \{k=1 \rightarrow n\} \quad (6)$$

$$Q_k(x_k) = \prod_{\substack{i=1 \\ i \neq k}}^n (x_k - x_i) \quad \{k=1 \rightarrow n\} \quad (7)$$

"n" es el número de datos conocidos, "x" es el valor a interpolar, " x_i " y " y_i " son los valores conocidos de las variables independiente y dependiente respectivamente.

Como se puede observar, los valores de "Q" pueden ser calculados independientemente del valor de "x", mientras que "P" debe ser calculado para cada valor de "x". Además, observe que en los datos no deben existir valores repetidos de "x" pues ello daría lugar a una división entre cero: resultante de restar dos números con el mismo valor.

Para ilustrar el método interpolaremos los datos de la tabla que se presenta al final de la pregunta, con el método de Lagrange, para valores de "x" iguales a 2 y 3.

x	0	1	4	6
y	1	-1	1	-1

Primero guardamos los datos de la tabla en dos vectores y el número de elementos en una variable:

```
vx={0,1,4,6};vy={1,-1,1,-1};n=Length[vx];
```

Calculamos los valores de "Q" con la ecuación (7):

```
Q=Table[Product[If[k!=i,vx[[k]]-vx[[i]],1],{i,n}],{k,n}]
{-24,15,-24,60}
```

Calculamos los valores de "P", para $x=2$, con la ecuación (6):

```
x=2;
```

```
P=Table[Product[If[k!=i,x-vx[[i]],1],{i,n}],{k,n}]
{8,16,-8,-4}
```

Calculamos los valores de "L" con la ecuación (5):

```
L=P/Q
{-1/3,16/15,1/3,-1/15}
```

Finalmente calculamos el valor de "y" para " $x=2$ " con la ecuación (4):

```
Sum[vy[[k]]*L[[k]],{k,n}]
-1
```

Que es el mismo resultado obtenido anteriormente. Ahora repetimos el proceso para " $x=3$ ": calculamos los valores de "P" con la (ecuación 6):

```
x=3;
```

```
P=Table[Product[If[k!=i,x-vx[[i]],1],{i,n}],{k,n}]
{6,9,-18,-6}
```

Calculamos los valores de "L" (ecuación 5):

```
L=P/Q
{-1/4,3/5,3/4,-1/10}
```

Y finalmente calculamos el valor de "y" para $x=3$, aunque en este caso recurriendo a la multiplicación de matrices, pues la ecuación (7) es justamente el resultado de dicha multiplicación:

```
vy.L
0
```

Que una vez más es el resultado correcto.

Como se puede observar, cada vez que cambia el valor de la variable independiente se deben volver a calcular los valores de "P" y "L" (y el de "Q")

si es que todavía no ha sido calculado) lo que por supuesto no es muy racional. En lugar de ello y dado que Mathematica nos permite trabajar con expresiones simbólicas resulta más eficiente trabajar directamente con la variable simbólica obteniendo así el polinomio de interpolación.

Pare ello liberamos primero la variable "x":

```
Remove[x]
```

Calculamos los valores de "P" (ecuación 6):

```
P=Table[Product[If[k!=i,x-vx[[i]],1],{i,n}],{k,n}]
{(-6+x) (-4+x) (-1+x), (-6+x) (-4+x) x, (-6+x) (-1+x) x, (-4+x) (-1+x) x}
```

Los valores de "L" (ecuación 5):

```
L=P/Q
{-(1/24) (-6+x) (-4+x) (-1+x), (1/15) (-6+x) (-4+x) x, -(1/24) (-6+x) (-1+x) x, (1/60) (-4+x) (-1+x) x}
```

Y el polinomio de interpolación (ecuación 4):

```
vy.L
-(1/24) (-6+x) (-4+x) (-1+x)-(1/15) (-6+x) (-4+x) x-1/24 (-6+x) (-1+x) x-
(1/60) (-4+x) (-1+x) x
```

Podemos realizar algunas de las operaciones de esta expresión para obtener un resultado más compacto:

```
Expand[%]
1-(10 x)/3+(3 x^2)/2-x^3/6
```

Ahora podemos ver con mayor claridad que se trata de una ecuación polinomial de tercer grado. Podemos asignar esta expresión a una función:

```
f[x_]:=Evaluate[%]
```

Y emplear la función para calcular los valores interpolados, así para "x=2" obtenemos:

```
f[2]
-1
```

Y para "x=3":

```
f[3]
0
```

Que son los mismos resultados obtenidos anteriormente. Por supuesto podemos obtener los dos resultados al mismo tiempo:

```
f[{2,3}]
{-1,0}
```

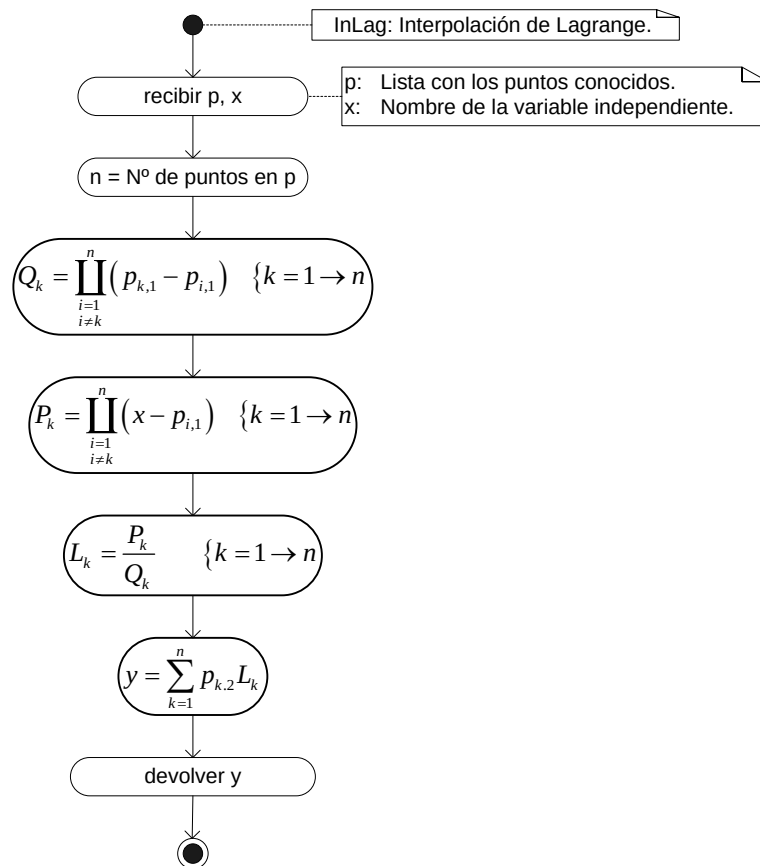
Esta es la forma que implementaremos el método de Lagrange.

23.3.1. Algoritmo y código

El algoritmo para el método de Lagrange, donde como ya se dijo se trabaja con la variable simbólica en lugar de un valor numérico específico, se presenta en la siguiente página. El código elaborado en base a dicho algoritmo y añadido al paquete "Ajuste" es el siguiente:

```
BeginPackage["Ajuste`"]
```

```
...
```



InLag::usage="InLag[p,x], devuelve, en función de \"x\", el polinomio de interpolación calculado con el método de Lagrange, donde \"p\", es la lista con los valores (x,y) conocidos."

```
Begin["Private`"]
```

```
...
```

```
InLag[p_List,x_Symbol]:=Module[{n,Q,P,L},
  n=Length[p];
  Q=Table[Product[If[k!=i,p[[k,1]]-p[[i,1]],1],{i,n}],{k,n}];
  P=Table[Product[If[k!=i,x-p[[i,1]],1],{i,n}],{k,n}];
  L=P/Q;
  Expand[p[[All,2]].L]]
```

```
End[]
```

```
EndPackage[]
```

Para probar el programa interpolamos para los datos del ejemplo:

```
Get["Ajuste`",Path->"E:\MAT205\"]
```

```
vx={0,1,4,6};vy={1,-1,1,-1};p=Transpose[{vx,vy}];
```

```
f[x_]:=Evaluate[InLag[p,x]]
```

```
f[2]
```

```
-1
```

```
f[3]
0
```

Puesto que se obtienen los mismos resultados que el ejemplo, podemos asumir que el programa ha sido correctamente elaborado.

23.3.2. Ejemplos

4. Como primer ejemplo emplearemos el programa elaborado para calcular los valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.1. Luego para analizar visualmente cuan confiables son los resultados, graficaremos los resultados devueltos por y los puntos conocidos.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

Guardamos los datos en variables:

```
p={{0,0},{0.2,0.199},{0.4,0.389},{0.6,0.565},{0.8,0.717},{1.0,.841}};
```

```
vc=Range[0.1,1.1,0.2]
{0.1,0.3,0.5,0.7,0.9,1.1}
```

Creemos la función de interpolación:

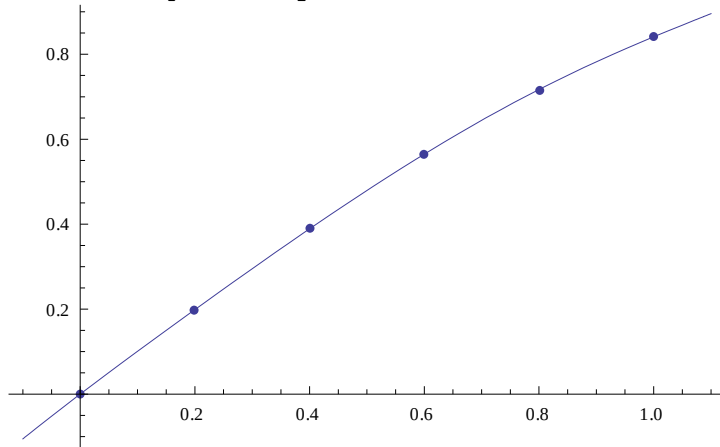
```
f[x_]:=Evaluate[InLag[p,x]];f[x]
1.02642 x-0.221875 x2+0.453125 x3-0.703125 x4+0.286458 x5
```

Interpolamos para los valores dados de "x":

```
f[vc]
{0.100809,0.295191,0.479387,0.64452,0.782215,0.895598}
```

Y graficamos la función de interpolación y los puntos conocidos:

```
Show[Plot[f[x],{x,-0.1,1.1}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Como se puede observar en este caso el polinomio se ajusta bastante bien a los valores conocidos, por lo que podemos asumir que los valores interpolados en este ejemplo son más exactos que los obtenidos con el método de interpolación lineal.

5. Como segundo ejemplo crearemos la función de interpolación para los datos que se presentan al final de la pregunta. Con la función de interpolación calcularemos los valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45. Finalmente para determinar visualmente cuán confiables son los resultados obtenidos graficaremos la función de interpolación conjuntamente

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

Al igual que en el anterior ejemplo, guardamos los datos en variables:

```
p={{3,0},{5,0},{7,0},{11,0},{14,1},{16,3},{19,13},{21,22},{24,29},{25,26},{26,18},{27,8},{29,3},{33,0},{36,0},{40,0},{44,0}};
```

```
vc={1,4,6,13,17,23,25.5,26.5,28,30,42,45};
```

Creamos la función de interpolación:

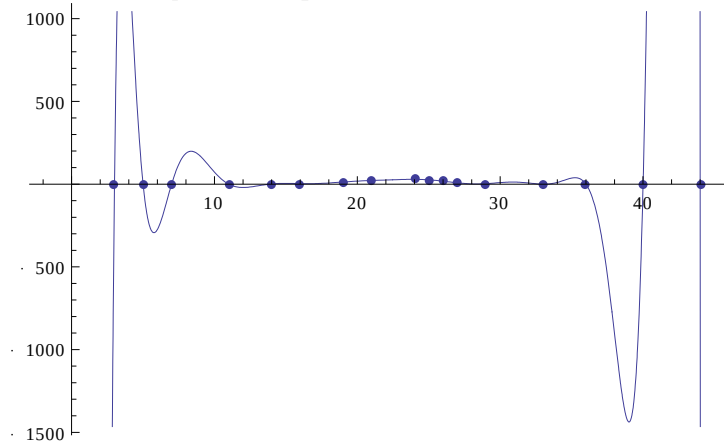
```
f[x]:=Evaluate[InLag[p,x]];f[x]/N
-663163.+832434. x-455384. x^2+145434. x^3-30583.9 x^4+4519.84 x^5-488.215 x^6+
39.4954 x^7 -2.42746 x^8+0.114088 x^9-0.00409759 x^10+0.000111535 x^11-2.26002×10-6
x^12+3.30113×10-8 x^13-3.28289×10-10 x^14+1.98853×10-12 x^15-5.5334×10-15 x^16
```

Interpolamos para los valores conocidos de "x":

```
f[vc]/N
{-167194.,1387.17,-274.195,-
10.7162,3.01998,27.8286,22.5341,12.9156,1.70822,9.34965,23896.,-298044.}
```

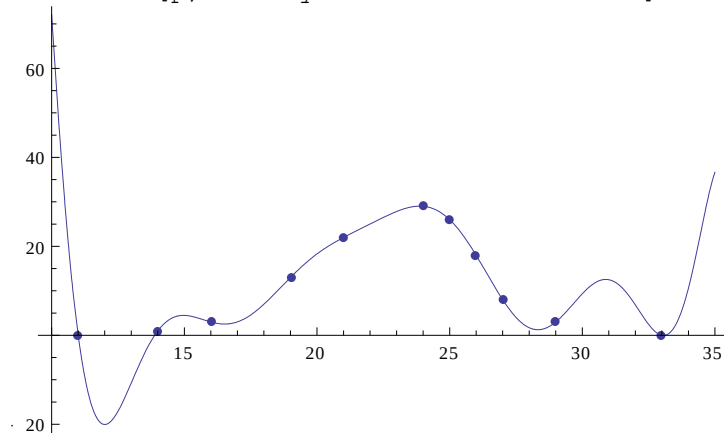
Y graficamos la función de interpolación y los puntos conocidos:

```
Show[Plot[f[x],{x,-2,45}],
ListPlot[p,PlotStyle→AbsolutePointSize[Medium]]]
```



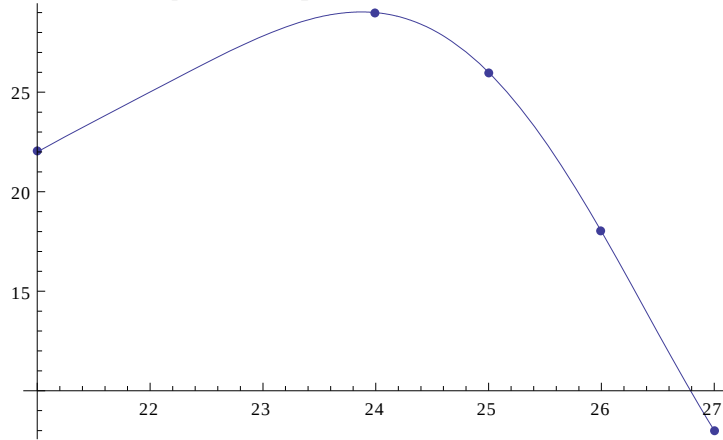
Como se puede observar, en este caso Lagrange no predice bien los datos intermedios, sobre todo a ambos extremos de la curva. Para apreciar mejor el comportamiento en la zona media graficamos entre 10 y 35:

```
Show[Plot[f[x],{x,10,35}],
ListPlot[p,PlotStyle→AbsolutePointSize[Medium]]]
```



Vemos así que el método de Lagrange sólo puede ser empleado con seguridad para valores de "x" comprendidos 21 y 27, como se puede corroborar en la siguiente gráfica:

```
Show[Plot[f[x],{x,21,27}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Para los extremos es más seguro emplear el método de interpolación lineal.

Este comportamiento es bastante frecuente en los métodos de interpolación: cuando existen cambios bruscos en el sentido de la curva, como en este caso.

Por ello en estos es aconsejable emplear un método de ajuste segmentario (como los que estudiaremos en el siguiente tema) o en su caso dividir la curva en segmentos de comportamiento uniforme, calcular los polinomios de interpolación para cada uno de ellos y con los mismos crear una función general con Piecewise.

23.4. INTERPOLACIÓN DE NEWTON

El segundo método de interpolación polinomial que estudiaremos es el de Newton, en el cual el valor interpolado se calcula con la ecuación:

$$y = b_1 + b_2(x - x_1) + b_3(x - x_1)(x - x_2) + \dots + b_n(x - x_1)(x - x_2)\dots(x - x_{n-1}) = \sum_{k=1}^n \left[b_k \prod_{i=1}^{k-1} (x - x_i) \right] \quad (8)$$

Los valores de "b" se calculan con las siguientes relaciones:

$$\begin{aligned} f &= y \\ b_k &= f_1 \\ f_i &= (f_{i+1} - f_i) / (x_{i+k} - x_i) \quad \{i=1 \rightarrow n-k\} \quad \left\{ \begin{array}{l} k=1 \rightarrow n-1 \end{array} \right. \\ b_n &= f_1 \end{aligned} \quad (9)$$

En estas ecuaciones "n" es el número de datos, "x_i" y "y_i" son los datos para la variable independiente y dependiente respectivamente y "x" es el valor a interpolar.

Como los coeficientes "b" sólo dependen de los valores conocidos pueden ser calculados por separado (tal como sucede con "Q" en el método de Lagrange). De esta manera es posible reducir el número de cálculos cuando es necesario interpolar varios valores.

Al igual que sucede con el método de Lagrange, el método de Newton no puede ser empleado cuando existen 2 o más valores repetidos de la variable independiente, pues como se puede observar en las ecuaciones 3, tendría lugar una división entre cero. Al igual que en todos los métodos estudiados hasta el momento es necesario que los valores de la variable independiente estén ordenados ascendentemente.

Para ilustrar el proceso de cálculo en el método de Newton calcularemos, con los datos de la tabla, los valores de "y" para valores de "x" iguales a 2 y 5.5:

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

Primero guardamos los valores de la tabla en dos vectores (o en matriz):

```
vx={1,4,5,6};
```

```
vy={0,1.3862944,1.6094379,1.7917595};
```

Leemos el número de elementos en el vector "x" (número de datos):

```
n=length[vx];
```

Creamos el vector "b" con "n" elementos cualesquiera (en este caso valores consecutivos), el objetivo es simplemente contar con dicho vector para llenar luego los datos respectivos:

```
b=Range[n];
```

Tal como se especifica en las ecuaciones (9), inicializamos la variable "f" con los valores del vector "y":

```
f=vy;
```

Calculamos los "n-1" valores de "b" con las dos expresiones intermedias de las ecuaciones (9):

```
Do[b[[k]]=f[[1]];
  Do[f[[i]]=(f[[i+1]]-f[[i]])/(vx[[i+k]]-vx[[i]]),
    {i,n-k}],{k,n-1}]
```

Y el ultimo valor de "b" con la última expresión de las ecuaciones (9):

```
b[[n]]=f[[1]];

```

Los resultados obtenidos son:

```
b
{0,0.462098,-0.0597387,0.00786554}
```

Finalmente calculamos el polinomio de interpolación con la ecuación (8), sin embargo, la aplicación directa de dicha ecuación implica la repetición innecesaria de operaciones, así para calcular el tercer término se realiza la operación: $(x-1)(x-2)$, para calcular el cuarto término se realiza la operación: $(x-1)(x-2)(x-3)$, donde como se puede observar se repite el cálculo de: $(x-1)(x-2)$, para calcular el cuarto término se realiza la operación: $(x-1)(x-2)(x-3)(x-4)$, volviéndose a repetir el cálculo de: $(x-1)(x-2)(x-3)$ y así sucesivamente.

Se puede evitar la inútil repetición de cálculos guardando el resultado de la operación anterior en una variable, en este caso "p", y empleando dicho resultado para calcular el nuevo, tal como se ha procedido en las siguientes instrucciones:

```
p=1;
```

```
s=b[[1]];
```

```
Do[p=p*(x-vx[[k-1]]);s=s+b[[k]]*p,{k,2,n}]
```

Siendo el polinomio resultante (almacenado en la variable "s"):

```
Expand[s]
-0.858364+0.988892 x-0.138394 x^2+0.00786554 x^3
```

Con este resultado podemos crear la función de interpolación:

```
fx[x_]:=Evaluate[%]
```

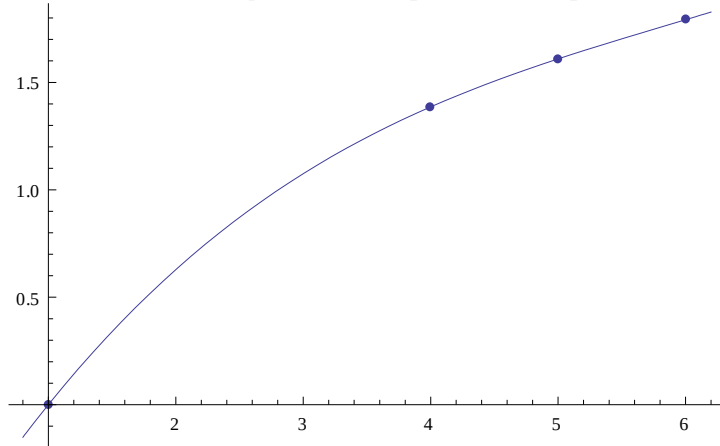
Y con la función de interpolación calculamos los valores de "y" para "x=2" y "x=5.5":

```
fx[2]
0.628769
```

```
fx[5.5]
1.70275
```

Para apreciar visualmente cuan confiables son los valores interpolados podemos graficar la función de interpolación y los puntos conocidos:

```
Show[Plot[fx[x],{x,0.8,6.2}],
ListPlot[Transpose[{vx,vy}],PlotStyle->AbsolutePointSize[Medium]]]
```



Vemos que en este caso la función de interpolación describe muy bien el comportamiento de los puntos, por lo que los valores interpolados deben ser correctos.

23.4.1. Algoritmo y código

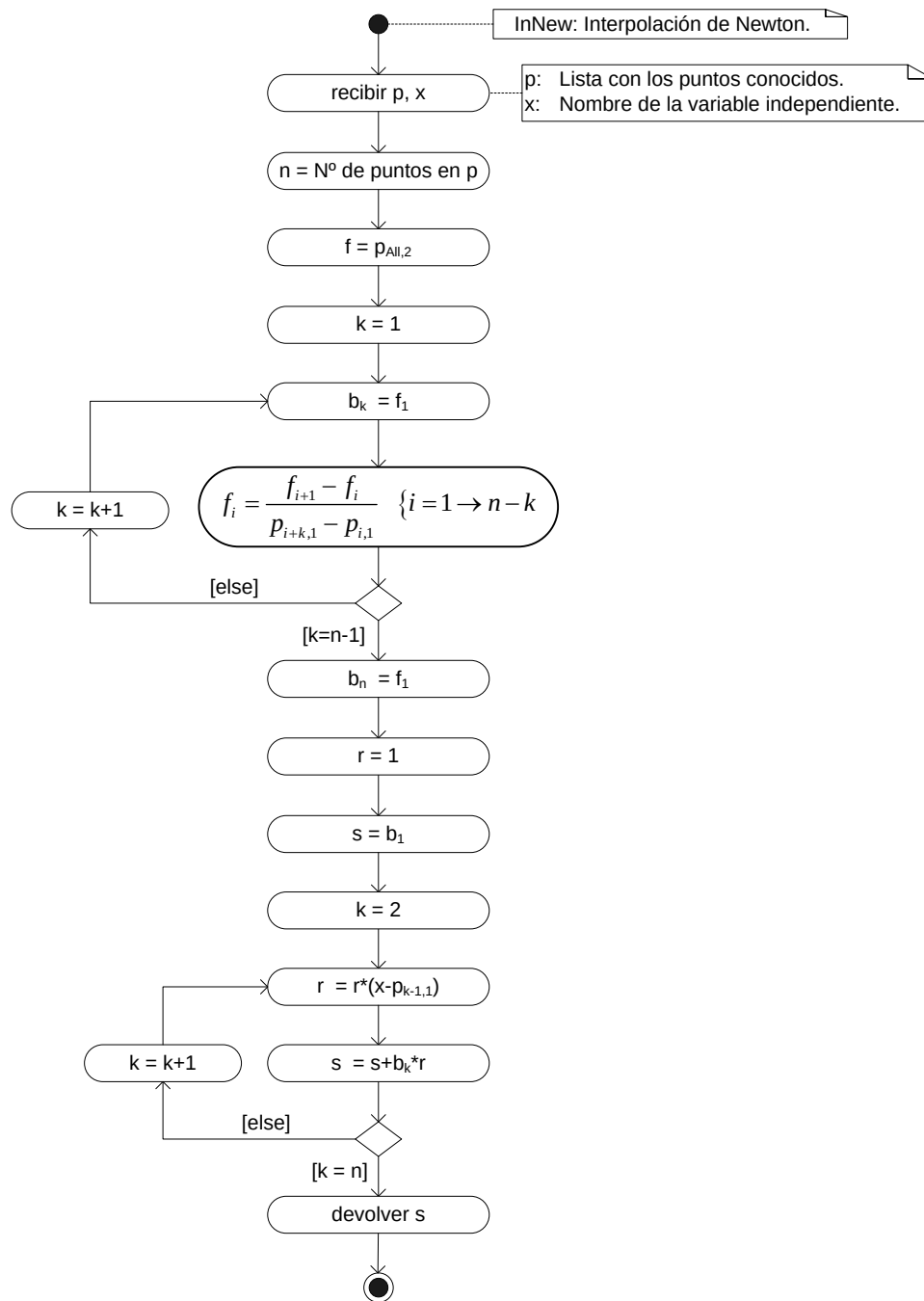
El algoritmo para el método de Newton, que básicamente consiste en la aplicación de las ecuaciones (9) y (8), cuidando de evitar la repetición de cálculos, se presenta en el diagrama de actividades de la siguiente página.

El código elaborado en base al mismo y añadido al paquete "Ajuste`" es el siguiente:

```
BeginPackage["Ajuste`"]
```

```
...
```

```
InNew::usage="InNew[p,x], devuelve, en función de \"x\", el polinomio
de interpolación calculado con el método de Lagrange, donde \"p\",
es la lista con los valores (x,y) conocidos."
```



```
Begin["Private`"]
```

```
...
```

```

InNew[p_List,x_Symbol]:=Module[{n,f,s,b,r},
  n=Length[p]; b=Range[n];
  f=p[[All,2]];
  Do[b[[k]]=f[[1]];
    Do[f[[i]]=(f[[i+1]]-f[[i]])/(p[[i+k,1]]-p[[i,1]]),{i,n-k}],
    {k,n-1}];
  b[[n]]=f[[1]];
  r=1;

```

```

s=b[[1]];
Do[r=r*(x-p[[k-1,1]]);s=s+b[[k]]*r,{k,2,n}];
Expand[s]]
End[]

EndPackage[]

```

Para probar el programa volvemos a resolver el ejemplo, para ello primero cargamos la librería:

```

Get["Ajuste`",Path->"E:\MAT205"]

Guardamos los puntos conocidos en una variable:
p={{1,0},{4,1.3862944},{5,1.6094379},{6,1.7917595}};

Creamos la función de interpolación con el programa:
f[x_]:=Evaluate[InNew[p,x]]

```

```

f[x]
-0.858364+0.988892 x-0.138394 x^2+0.00786554 x^3

```

Y con la función de interpolación calculamos los valores de "y" para "x=2" y "x=5.5":

```

f[2]
0.628769

```

```

f[5.5]
1.70275

```

Como se obtienen los mismos resultados que en el ejemplo, se puede asumir que el programa ha sido correctamente elaborado.

23.4.2. Ejemplos

6. Como primer ejemplo volveremos a resolver el ejemplo 1 del método de Lagrange, pero ahora empleando el método de Newton. Es decir crearemos una función de interpolación para la tabla que se presenta al final de la pregunta, calcularemos los valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.1 y graficaremos la función de interpolación conjuntamente los datos conocidos para poder juzgar la confiabilidad de los resultados obtenidos.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

El procedimiento de cálculo es el mismo que en el ejemplo 1, sólo que en lugar de llamar a "InLag" se llama a "InNew":

```

Get["Ajuste`",Path->"E:\MAT205"]

p={{0,0},{0.2,0.199},{0.4,0.389},{0.6,0.565},{0.8,0.717},{1.0,.841}};

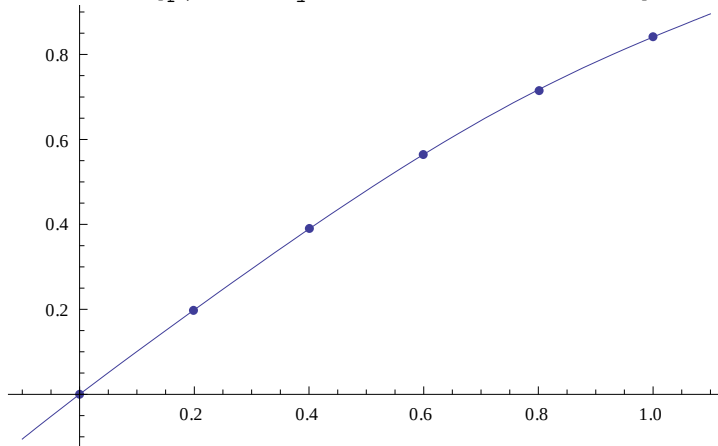
vc=Range[0.1,1.1,0.2]
{0.1,0.3,0.5,0.7,0.9,1.1}

f[x_]:=Evaluate[InNew[p,x]];f[x]
1.02642 x-0.221875 x^2+0.453125 x^3-0.703125 x^4+0.286458 x^5

f[vc]
{0.100809,0.295191,0.479387,0.64452,0.782215,0.895598}

```

```
Show[Plot[f[x],{x,-0.1,1.1}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```



Como se puede observar, en este caso se obtienen exactamente los mismos resultados que con el método de Lagrange.

7. Como segundo ejemplo resolveremos el ejemplo 2 del método de Lagrange, es decir crearemos una función de interpolación para los datos de la tabla, calcularemos los valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45 y juzgar gráficamente la exactitud de los resultados graficaremos la función de interpolación conjuntamente los datos conocidos.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

Una vez más, el procedimiento es el mismo que en el método de Lagrange sólo que en lugar de emplear "InLag" se emplea "InNew":

```
p={{3,0},{5,0},{7,0},{11,0},{14,1},{16,3},{19,13},{21,22},{24,29},{25,26},{26,18},{27,8},{29,3},{33,0},{36,0},{40,0},{44,0}};
```

```
vc={1,4,6,13,17,23,25.5,26.5,28,30,42,45};
```

```
f[x_]:=Evaluate[InNew[p,x]];
```

```
f[x]//N
```

```
-663163.+832434. x-455384. x^2+145434. x^3-30583.9 x^4+4519.84 x^5-488.215
x^6+39.4954 x^7-2.42746 x^8+0.114088 x^9-0.00409759 x^10+0.000111535 x^11-2.26002×10^-
6 x^12+3.30113×10^-8 x^13-3.28289×10^-10 x^14+1.98853×10^-12 x^15-5.5334×10^-15 x^16
```

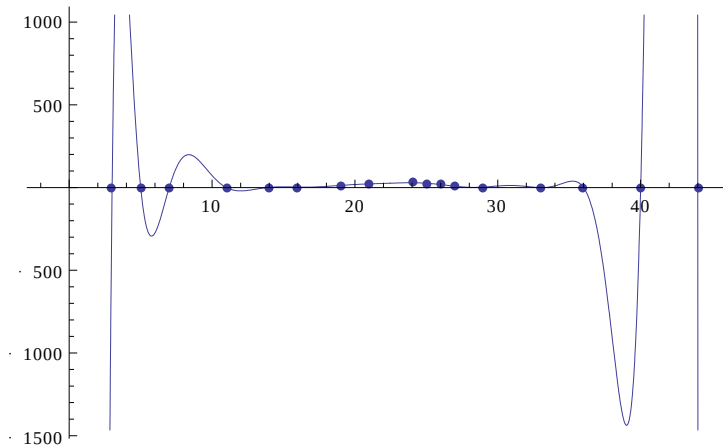
```
f[vc]//N
```

```
{-167194.,1387.17,-274.195,-
10.7162,3.01998,27.8286,22.5341,12.9156,1.70822,9.34965,23896.,-298044.}
```

La gráfica de la función de interpolación y de los puntos conocidos se obtiene con:

```
Show[Plot[f[x],{x,-2,45}],
ListPlot[p,PlotStyle->AbsolutePointSize[Medium]]]
```

El resultado se presenta en la siguiente página y como se puede ver, en este caso se obtienen también los mismos resultados que con el método de Lagrange, por lo tanto, en este caso no se debe emplear ni el método de Newton ni de Lagrange para interpolar datos.



23.5. Ejercicios

1. Empleando los datos de la siguiente tabla y el método de interpolación lineal, cree una función de interpolación, calcule con la misma valores de "y" para valores de "x" iguales a 3.9, 4.8, 6.1, 6.57, 6.7, 7.1 y grafique la función de interpolación conjuntamente los puntos conocidos para determinar visualmente la confiabilidad de los resultados.

x	3.5	4	4.5	5	5.5	6	6.2	6.5	6.8
y	0.002	0.030	0.155	0.396	0.658	0.903	0.933	0.975	0.993

2. Resuelva el ejercicio 1 empleando el método de Lagrange.
3. Resuelva el ejercicio 1 empleando el método de Newton.
4. Resuelva el ejercicio 1 empleando InterpolatingPolynomial.
5. Empleando los datos de la siguiente tabla y el método de interpolación lineal, cree una función de interpolación, calcule con la misma valores de "y" para valores de "x" iguales a 0.5, 2.5, 3.2, 4.3, 5.5, 6.5, 8.5, 10.5, 13.5, 15.5, 18.5, 20.5, 22.5, 24 y grafique la función de interpolación conjuntamente los puntos conocidos para determinar visualmente la confiabilidad de los resultados.

x	1	2	3	3.5	4	4.5	5	6	7	8	9	10	12	13	14	15	16	17	18	19	20	21	22	23
y	0	0	0.5	1.5	3	5	9	11.8	12	12.5	12.5	12.5	12.5	12.4	12.0	11.5	10.5	9.5	8.3	7.0	5.5	4.0	3.0	2.5

6. Resuelva el ejercicio 5 empleando el método de Lagrange.
7. Resuelva el ejercicio 5 empleando el método de Newton.
8. Resuelva el ejercicio 5 empleando InterpolatingPolynomial.
9. Elabore un módulo que calcule y muestre los valores de " x_i ", " y_i ", " k_x " y " k_y ", resolviendo, con FindRoot, la función implícita, que se muestra a continuación (empleando un valor inicial igual a 0.5). Para obtener los valores de " y_i " debe crear una función de interpolación, empleando los datos de la tabla y el método de interpolación lineal.

$$f(x_i) = \frac{k_x}{k_y} + \frac{y - y_i}{x - x_i} = 0$$

$$k_y = \frac{k'_y}{(1-y)_{im}}; \quad (1-y)_{im} = \frac{(1-y_i) - (1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

$$k_x = \frac{k'_x}{(1-x)_{im}}; \quad (1-x)_{im} = \frac{(1-x) - (1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)}$$

$$k'_x = 1.967 \times 10^{-3}; \quad k'_y = 1.465 \times 10^{-3}; \quad y = 0.39; \quad x = 0.1$$

x_i	0	0.05	0.10	0.15	0.20	0.25	0.30	0.35
y_i	0	0.022	0.052	0.087	0.131	0.187	0.265	0.385

10. Resuelva el ejercicio 9 empleando "NewtonDN" y "Lagrange".
11. Resuelva el ejercicio 9 empleando "Incret1" (valor inicial 0.1, incremento 0.01), "RegulaFalsi" y "Newton".
12. Resuelva el ejercicio 9 empleando "Incret2" (valor inicial 0.1, incremento 0.01) e "InterpolatingPolynomial".

8. INTERPOLACIÓN SEGMENTARIA

8.1. INTERPOLACIÓN SEGMENTARIA LINEAL (SPLINE LINEAL)

El primer método de interpolación segmentaria que estudiaremos es el lineal (Spline Lineal). En este método cada par de puntos están unidos por una línea recta. En consecuencia se trata del método de interpolación lineal, excepto que ahora, para crear la función de interpolación, se calculan los coeficientes para las "n-1" ecuaciones lineales que unen los "n" puntos.

La ecuación para cada segmento es:

$$y = s_j(x) = a_j + b_j(x - x_j) \quad \{j=1 \rightarrow n-1\} \quad (1)$$

Donde n es el número de datos (x-y) con los que se cuenta. Puesto que en la interpolación segmentaria se tiene una ecuación para cada par de puntos y cada ecuación pasa exactamente a través de dos puntos, en el punto (x_{j+1}, y_{j+1}) , los valores devueltos por las ecuaciones que se encuentra a ambos lados de dicho punto ($s_j(x)$ y $s_{j+1}(x)$) son iguales, es decir:

$$\begin{aligned} s_j(x_{j+1}) &= s_{j+1}(x_{j+1}) = y_{j+1} \\ a_j + b_j(x_{j+1} - x_j) &= a_{j+1} + b_{j+1}(x_{j+1} - x_{j+1}) = y_{j+1} \\ a_j + b_j(x_{j+1} - x_j) &= a_{j+1} = y_{j+1} \end{aligned} \quad (2)$$

En consecuencia:

$$a_j = y_j \quad \{j=1 \rightarrow n\} \quad (3)$$

Denominando h_j a la diferencia:

$$h_j = x_{j+1} - x_j \quad \{j=1 \rightarrow n-1\} \quad (4)$$

La ecuación (2) puede ser reescrita como:

$$a_j + b_j h_j = a_{j+1} \quad \{j=1 \rightarrow n-1\} \quad (5)$$

De donde podemos despejar b_j :

$$b_j = (a_{j+1} - a_j) / h_j \quad \{j=1 \rightarrow n-1\} \quad (6)$$

Por lo tanto, con las ecuaciones 3, 4 y 6 podemos calcular todos los coeficientes a_j y b_j de los "n-1" segmentos.

8.1.1. Ejemplo Manual

Para comprender mejor el mecanismo de cálculo aplicaremos la interpolación segmentaria lineal para calcular valores de "y" para valores de "x" iguales a 2 y 5.5 con los datos de la tabla. Para apreciar visualmente cuan buenos son los resultados intermedios calculados graficaremos también la función de interpolación y los puntos conocidos.

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

Primero guardamos los puntos conocidos en una matriz:

```
p={{1,0},{4,1.3862944},{5,1.6094379},{6,1.7917595}};
```

El número de puntos en la matriz es:

```
n=length[p];
```

De acuerdo a la ecuación (3), los valores de "a" son simplemente los valores de la variable dependiente "y":

```
a=p[[All,2]];
```

Los valores de "h" se calcula aplicando la ecuación (4):

```
h=Table[p[[j+1,1]]-p[[j,1]],{j,n-1}]
{3,1,1}
```

Ahora, con los valores de "h" podemos calcular los valores de "b" se aplicando con la ecuación (6):

```
b=Table[(a[[j+1]]-a[[j]])/h[[j]],{j,n-1}]
{0.462098,0.223144,0.182322}
```

Con ello contamos ya con todos los coeficientes necesarios para la interpolación. Ahora se puede crear una función de interpolación, que básicamente ubica el segmento "j" en el cual se encuentra el valor a interpolar ("x") y una vez ubicado emplea la ecuación (1) para obtener el correspondiente valor de "y" (el valor interpolado):

```
f[x_]:=Module[{j=1},While[ x>p[[j+1,1]] && j<n-1 ,j++];
a[[j]]+b[[j]]*(x-p[[j,1]])]
```

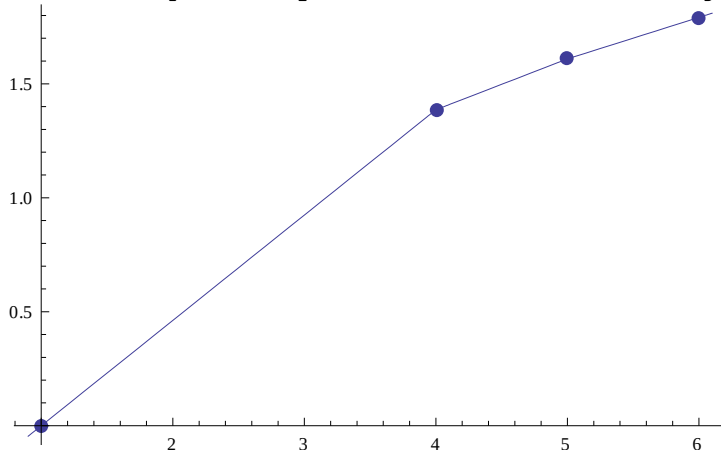
Con la función de interpolación calculamos los valores de "y" para "x" igual a 2 y 5.5:

```
f[2]
0.462098
```

```
f[5.5]
1.7006
```

Ahora podemos emplear la función de interpolación para graficar los valores interpolados y los puntos conocidos:

```
Show[Plot[f[x],{x,0.9,6.1}],
ListPlot[p,PlotStyle->AbsolutePointSize[Large]]]
```

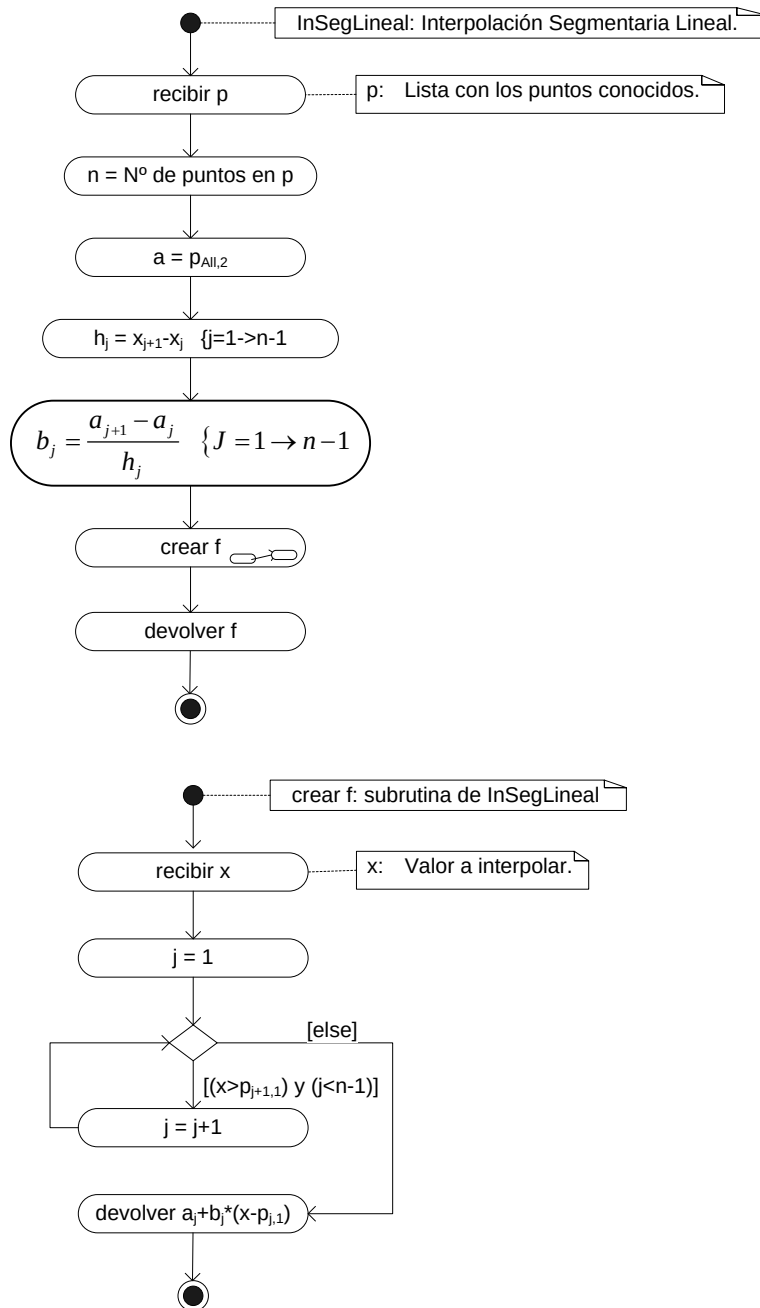


Como se puede observar, la función de interpolación resulta adecuada para los dos últimos segmentos, pero muy probablemente no para el primero, donde es casi seguro que los datos no sigan un comportamiento lineal.

3.1.2. Algoritmo y Código

El algoritmo para el método de interpolación lineal, que básicamente sigue los pasos descritos en el ejemplo manual, se presenta en las dos figuras de la siguiente página.

El código elaborado en base a dicho algoritmo y añadido al paquete "Ajuste" es el siguiente:



8.1.2.1. Ejemplo 2

Como segundo ejemplo resolveremos el ejemplo 2 del método de Newton, es decir elaboraremos un módulo que, empleando los datos de la siguiente tabla, cree una función de interpolación aplicando interpolación segmentaria lineal, grafique la función conjuntamente los datos conocidos y calcule los valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.2.

X	0	0.2	0.4	0.6	0.8	1.0
Y	0	0.199	0.389	0.565	0.717	0.841

El módulo que resuelve el problema es el siguiente:

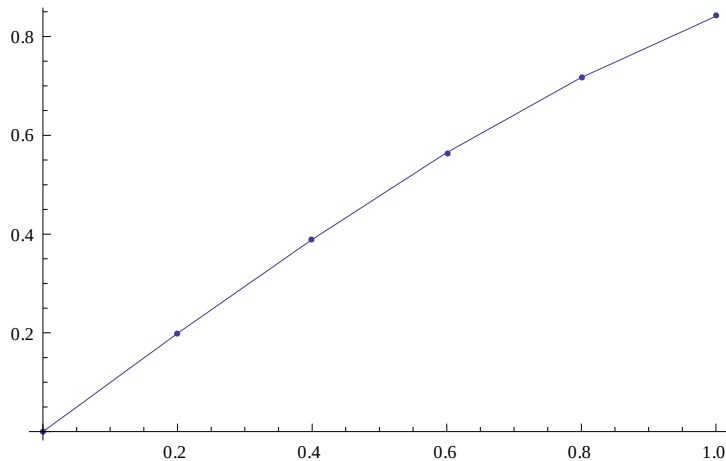
Module[{n, InLin, a, h, b, g1, g2, p, vx,

```

vy={0,0.199,0.389,0.565,0.717,0.841},
vcx={0.1,0.3,0.5,0.7,1.2}},
vx=Range[0,1,0.2];n=Length[vx];p=Transpose[{vx,vy}];
a=vy;h=Table[vx[[j+1]]-vx[[j]],{j,n-1}];
b=Table[(a[[j+1]]-a[[j]])/h[[j]],{j,n-1}];
InLin[x_]:= (For[j=1,x>vx[[j+1]] && j<n-1,j++];a[[j]]+b[[j]]*(x-vx[[j]]));
g1=Plot[InLin[x],{x,0,1}]; g2=ListPlot[p];
Print[Show[g1,g2]];
Print["Valores de y interpolados: "];
Do[Print["x = ",vcx[[i]]," y = ",InLin[vcx[[i]]]],{i,Length[vcx]}];
]

```

Ejecutando el módulo se obtiene:



Valores de y interpolados:

```

x = 0.1    y = 0.0995
x = 0.3    y = 0.294
x = 0.5    y = 0.477
x = 0.7    y = 0.641
x = 1.2    y = 0.965

```

Como era de esperar estos resultados concuerdan exactamente con los del ejemplo 2 del método de interpolación lineal.

8.1.2.2. Ejemplo 3

Como tercer ejemplo resolveremos el ejemplo 3 de Newton, es decir elaboraremos un módulo que empleando los datos de la siguiente tabla, cree una función que devuelva el valor "y" correspondiente a un determinado valor de "x", aplicando el método de interpolación segmentaria lineal. El módulo debe graficar la función de interpolación conjuntamente los datos conocidos y emplear la función de interpolación para calcular valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

El módulo que resuelve el problema es el siguiente:

```

Module[{InLin,a,h,b,g1,g2,n,p,
  vx={3,5,7,11,14,16,19,21,24,25,26,27,29,33,36,40,44},
  vy={0,0,0,0,1,3,13,22,29,26,18,8,3,0,0,0,0,0},
  vcx={1,4,6,13,17,23,25.5,26.5,28,30,42,45}},
n=Length[vx];p=Transpose[{vx,vy}];

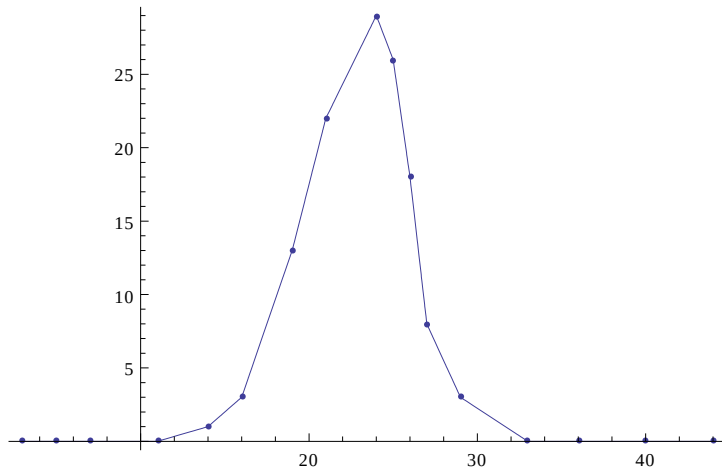
```

```

a=vy;h=Table[vx[[j+1]]-vx[[j]],{j,n-1}];
b=Table[(a[[j+1]]-a[[j]])/h[[j]],{j,n-1}];
InLin[x_] := (For[j=1,x>vx[[j+1]] && j<n-1,j++];a[[j]]+b[[j]]*(x-vx[[j]]));
g1=Plot[InLin[x],{x,3,44}];
g2=ListPlot[p];
Print[Show[g1,g2]];
Print["Valores de y interpolados: "];
Do[Print["x = ",vcx[[i]]," y = ",InLin[vcx[[i]]]],{i,Length[vcx]}];
]

```

Evaluando el módulo se obtiene:



Valores de y interpolados:

```

x = 1      y = 0
x = 4      y = 0
x = 6      y = 0
x = 13     y = 2/3
x = 17     y = 19/3
x = 23     y = 80/3
x = 25.5   y = 22.
x = 26.5   y = 13.
x = 28     y = 11/2
x = 30     y = 9/4
x = 42     y = 0
x = 45     y = 0

```

Y como era de esperar, estos resultados concuerdan exactamente con los del ejemplo 3 del método de interpolación lineal.

8.2. INTERPOLACIÓN SEGMENTARIA CÚBICA (SPLINE CÚBICO)

En el spline cúbico la ecuación que une cada par de puntos es la ecuación cúbica:

$$y = s_j(x) = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3 \quad \{j=1 \rightarrow n-1\} \quad (7)$$

Al igual que en spline lineal, las ecuaciones a ambos lados de un punto deben devolver el mismo resultado en dicho punto, es decir:

$$\begin{aligned}
 s_j(x_{j+1}) &= s_{j+1}(x_{j+1}) = y_{j+1} \\
 a_j + b_j(x_{j+1}-x_j) + c_j(x_{j+1}-x_j)^2 + d_j(x_{j+1}-x_j)^3 &= \\
 a_{j+1} + b_{j+1}(x_{j+1}-x_{j+1}) + c_{j+1}(x_{j+1}-x_{j+1})^2 + d_{j+1}(x_{j+1}-x_{j+1})^3 &= y_{j+1} \\
 a_j + b_j h_j + c_j h_j^2 + d_j h_j^3 &= a_{j+1} = y_{j+1} \quad \{j=1 \rightarrow n-1\} \quad (8)
 \end{aligned}$$

Donde:

$$h_j = x_{j+1} - x_j \quad \{j=1 \rightarrow n-1\} \quad (9)$$

Por lo tanto los valores de "a" pueden ser calculados con:

$$a_j = y_j \quad \{j=1 \rightarrow n\} \quad (10)$$

Para que las curvas a ambos lados de un punto tengan la misma pendiente en el punto y sean continuas, se igualan las derivadas primera y segunda de las ecuaciones a ambos lados del punto:

$$\begin{aligned} s'_j(x_{j+1}) &= s'_{j+1}(x_{j+1}) \\ b_j + 2c_j h_j + 3d_j h_j^2 &= b_{j+1} + 2c_{j+1}(x_{j+1} - x_j) + 3d_{j+1}(x_{j+1} - x_j)^2 \\ b_j + 2c_j h_j + 3d_j h_j^2 &= b_{j+1} \quad \{j=1 \rightarrow n-1\} \end{aligned} \quad (11)$$

y

$$\begin{aligned} s''_j(x_{j+1}) &= s''_{j+1}(x_{j+1}) \\ 2c_j + 6d_j h_j &= 2c_{j+1} + 6d_{j+1}(x_{j+1} - x_j) \\ c_j + 3d_j h_j &= c_{j+1} \quad \{j=1 \rightarrow n-1\} \end{aligned} \quad (12)$$

Despejando d_j de esta ecuación:

$$d_j = (c_{j+1} - c_j) / (3h_j) \quad \{j=1 \rightarrow n-1\} \quad (13)$$

y reemplazando esta ecuación en la ecuación 20:

$$\begin{aligned} b_j + 2c_j h_j + 3(c_{j+1} - c_j) / (3h_j) h_j^2 &= b_{j+1} \\ b_j + 2c_j h_j + c_{j+1} h_j - c_j h_j &= b_{j+1} \\ b_j + c_j h_j + c_{j+1} h_j &= b_{j+1} \\ b_j + (c_j + c_{j+1}) h_j &= b_{j+1} \end{aligned} \quad (14)$$

Reemplazando la ecuación 22 en la ecuación 17:

$$\begin{aligned} a_j + b_j h_j + c_j h_j^2 + (c_{j+1} - c_j) / (3h_j) h_j^3 &= a_{j+1} \\ a_j + b_j h_j + (3c_j h_j^2 + c_{j+1} h_j^2 - c_j h_j^2) / 3 &= a_{j+1} \\ b_j &= (a_{j+1} - a_j) / h_j - (2c_j h_j + c_{j+1} h_j) / 3 \\ b_j &= (a_{j+1} - a_j) / h_j - (2c_j + c_{j+1}) h_j / 3 \quad \{j=1 \rightarrow n-1\} \end{aligned} \quad (15)$$

Reemplazando 24 en 23 y reajustando índices:

$$\begin{aligned} (a_{j+1} - a_j) / h_j - (2c_j + c_{j+1}) h_j / 3 + (c_j + c_{j+1}) h_j &= (a_{j+2} - a_{j+1}) / h_{j+1} - (2c_{j+1} + c_{j+2}) h_{j+1} / 3 \\ (-2c_j h_j - c_{j+1} h_j + 3c_j h_j + 3c_{j+1} h_j) / 3 + (2c_{j+1} + c_{j+2}) h_{j+1} / 3 &= \\ 3((a_{j+2} - a_{j+1}) / h_{j+1} - (a_{j+1} - a_j) / h_j) & \\ (c_j h_j + 2c_{j+1} h_j + 2c_{j+1} h_{j+1} + c_{j+2} h_{j+1}) &= 3((a_{j+2} - a_{j+1}) / h_{j+1} - (a_{j+1} - a_j) / h_j) \\ (h_j c_j + 2(h_j + h_{j+1}) c_{j+1} + h_{j+1} c_{j+2}) &= 3((a_{j+2} - a_{j+1}) / h_{j+1} - (a_{j+1} - a_j) / h_j) \quad \{j=1 \rightarrow n-2\} \end{aligned} \quad (16)$$

La ecuación (25) constituye un sistema de $n-2$ ecuaciones con n incógnitas (los valores de "c"), por consiguiente se requieren 2 valores conocidos de "c" para resolver el sistema. Como no se tiene ninguna información con relación al comportamiento de los datos antes del primer o después del último punto, podemos asumir que la función en el segmento 1 tiene la forma: $s_1(x) = a_1 + b_1(x - x_1) + d_1(x - x_1)^3$ y en el segmento "n" (en realidad inexistente) tiene la misma forma: $s_n(x) = a_n + b_n(x - x_n) + d_n(x - x_n)^3$, o lo que es lo mismo asumimos que los valores de c_1 y c_n son cero.

Con ello tenemos los dos valores de "c" que nos permiten resolver el sistema de ecuaciones lineales (25). Para comprender mejor la solución de este sistema escribamos el mismo en forma matricial:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \dots & 0 & 0 & 0 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \dots & 0 & 0 & 0 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ \dots \\ c_{n-1} \\ c_n \end{bmatrix} = \begin{bmatrix} 0 \\ p_2 \\ p_3 \\ p_4 \\ \dots \\ p_{n-1} \\ 0 \end{bmatrix} \quad (17)$$

Donde la primera y última fila corresponde a los valores asumidos (c_1 y c_n). Los valores de "p" es la simbología elegida para representar el lado derecho de la ecuación (9), es decir:

$$p_j = 3((a_{j+1}-a_j)/h_j - (a_j-a_{j-1})/h_{j-1}) \quad \{j=1 \rightarrow n-1\} \quad (18)$$

La matriz aumentada del sistema de ecuaciones 9 es la siguiente:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ h_1 & 2(h_1+h_2) & h_2 & 0 & 0 & \dots & 0 & 0 & 0 & p_2 \\ 0 & h_2 & 2(h_2+h_3) & h_3 & 0 & \dots & 0 & 0 & 0 & p_3 \\ 0 & 0 & h_3 & 2(h_3+h_4) & h_4 & \dots & 0 & 0 & 0 & p_4 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & h_{n-2} & 2(h_{n-2}+h_{n-1}) & h_{n-1} & p_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix} \quad (19)$$

Para resolver el sistema de ecuaciones lineales, por un método de eliminación como el de Gauss, debemos llevar la matriz aumentada (28) a la forma:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 \\ 0 & 1 & r_2 & 0 & 0 & \dots & 0 & 0 & 0 & s_2 \\ 0 & 0 & 1 & r_3 & 0 & \dots & 0 & 0 & 0 & s_3 \\ 0 & 0 & 0 & 1 & r_4 & \dots & 0 & 0 & 0 & s_4 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 1 & r_{n-1} & s_{n-1} \\ 0 & 0 & 0 & 0 & 0 & \dots & 0 & 0 & 1 & 0 \end{bmatrix} \quad (20)$$

Donde r_j y s_j es la simbología elegida para representar los nuevos valores de los elementos de la matriz aumentada una vez llevada a esta forma. Con la matriz aumentada en forma de la ecuación 29, podemos calcular las incógnitas (los valores de c_2 a c_{n-1}) por sustitución hacia atrás.

Para llevar la matriz aumentada 28 la forma 29 se deben efectuar las siguientes operaciones: Los elementos de la primera columna se convierten en cero simplemente restando a la segunda fila la primera fila multiplicada por h_1 , puesto que todos los otros elementos de la primera fila son cero, esta operación no modifica ninguno de los otros elementos de la segunda fila. Posteriormente para convertir en uno el segundo elemento de la segunda fila, dividimos esta fila entre " $2(h_1+h_2)$ ", llamando q_2 a esta expresión las operaciones que se realizan son:

$$q_2 = 2(h_1+h_2)$$

$$r_2 = h_2/q_2$$

$$s_2 = p_2/h_2$$

Con estas operaciones la segunda fila queda como en la ecuación 29.

En la tercera fila debemos eliminar el elemento h_2 , esto se consigue restando a la tercera fila la segunda fila multiplicada por $-h_2$, con ello cambia los valores del tercer y último elemento de la tercera fila:

$$q_3 = 2(h_2+h_3) - h_2r_2$$

$$p'_3 = p_3 - h_2s_2$$

Ahora convertimos en uno el tercer elemento de la tercera fila, para ello dividimos la fila entre q_3 . Con esta operación cambian los valores del cuarto y último elemento de esta fila:

$$r_3 = h_3/q_3$$

$$s_3 = p'_3/q_3 = (p_3 - h_2s_2)/q_3$$

Con lo que la tercera fila queda como en la ecuación 12.

Procedemos de manera similar con la cuarta fila. Para eliminar el tercer elemento restamos a la cuarta fila la tercera fila multiplicada por $-h_3$, con ello cambian los valores del cuarto y último elementos de la siguiente manera:

$$q_4 = 2(h_3+h_4) - h_3r_3$$

$$p'_4 = p_4 - h_3s_3$$

Ahora convertimos en uno el cuarto elemento de la cuarta fila dividiendo esta fila entre q_4 , con ello cambian los valores del quinto y último elemento de esta fila:

$$r_4 = h_4/q_4$$

$$s_4 = p'_4/q_4 = (p_4 - h_3s_3)/q_4$$

Con las anteriores ecuaciones se pueden deducir las relaciones generales para llevar la matriz aumentada de la forma 28 a la forma 29:

$$q_j = 2(h_{j-1}+h_j) - h_{j-1}r_{j-1} \quad \{j = 2 \rightarrow n-1\} \quad (21)$$

$$r_j = h_j/q_j \quad \{j = 2 \rightarrow n-1\} \quad (22)$$

$$s_j = (p_j - h_{j-1}s_{j-1})/q_j \quad \{j = 2 \rightarrow n-1\} \quad (23)$$

Como se puede observar en la matriz aumentada 29, r_1 y s_1 son iguales a cero. Estando la matriz aumentada en la forma 29, los valores de "c" pueden ser calculados por sustitución hacia atrás:

$$c_j = s_j - r_j c_{j+1} \quad \{j = n-1 \rightarrow 2\} \quad (24)$$

Con los valores de "c" calculados se calculan los valores de "d" empleando la ecuación 22 y los valores de "b" empleando la ecuación 24:

$$d_j = (c_{j+1} - c_j)/(3h_j) \quad \{j=1 \rightarrow n-1\} \quad (22)$$

$$b_j = (a_{j+1} - a_j)/h_j - (2c_j + c_{j+1})h_j/3 \quad \{j=1 \rightarrow n-1\} \quad (24)$$

Recordemos que los valores de "h" se calculan con la ecuación 18, los de "a" con la ecuación 19, los valores de "p" con la ecuación (27) y el valor de "y" (valor interpolado) con la ecuación 16:

$$h_j = x_{j+1} - x_j \quad \{j=1 \rightarrow n-1\} \quad (18)$$

$$a_j = y_j \quad \{j=1 \rightarrow n\} \quad (19)$$

$$p_j = 3((a_{j+1} - a_j)/h_j - (a_j - a_{j-1})/h_{j-1}) \quad \{j=1 \rightarrow n-1\} \quad (27)$$

$$y = a_j + b_j(x-x_j) + c_j(x-x_j)^2 + d_j(x-x_j)^3 \quad (16)$$

Para reducir el número de cálculos, esta ecuación también puede ser escrita en la forma:

$$y = a_j + (b_j + (c_j + d_j * h) * h) * h; \quad h = x - x_j \quad (16)$$

8.2.1. Ejemplos

8.2.1.1. Ejemplo 1

Como primer ejemplo emplearemos la interpolación segmentaria cúbica para resolver el ejemplo 1 de Newton, es decir calcular valores de "y" para valores de "x" iguales a 2 y 5.5 con los datos de la tabla:

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

Primero guardamos los datos en dos vectores y el número de elementos en una variable:

```
vx={1,4,5,6};vy={0,1.386294,1.6094379,1.7917595};n=Length[vx];
```

Luego calculamos los valores de "h", "a" y "p" (ecuaciones 18, 19 y 27):

```
h=Table[vx[[j+1]]-vx[[j]],{j,n-1}]
{3,1,1}
a=vy
{0,1.38629,1.60944,1.79176}
p=Table[3*((a[[j+1]]-a[[j]])/h[[j]]-(a[[j]]-a[[j-1]])/h[[j-1]]),{j,n-1}]
{4.38629,-0.716862,-0.122467}
```

Calculamos los valores de "r" y "s" (ecuaciones 27, 30, 31 y 32):

```
q=ConstantArray[0,n-1];r=q;s=q;
Do[q[[j]]=2*(h[[j-1]]+h[[j]])-h[[j-1]]*r[[j-1]];
r[[j]]=h[[j]]/q[[j]];
s[[j]]=(p[[j]]-h[[j-1]]*s[[j-1]])/q[[j]],{j,2,n-1}]
Print[q,r,s]
{0,8,31/8} {0,1/8,8/31} {0,-0.0896078,-0.00847977}
```

Calculamos los valores de "c" (ecuación 33):

```
c=ConstantArray[0,n];
Do[c[[j]]=s[[j]]-r[[j]]*c[[j+1]],{j,n-1,2,-1}];c
{0,-0.0885478,-0.00847977,0}
```

Calculamos los valores de "d" y "b" (ecuaciones 22 y 24):

```
d=Table[(c[[j+1]]-c[[j]])/(3*h[[j]]),{j,n-1}]
{-0.00983865,0.0266893,0.00282659}
b=Table[(a[[j+1]]-a[[j]])/h[[j]]-(2*c[[j]]+c[[j+1]])*h[[j]]/3,{j,n-1}]
{0.550646,0.285002,0.187975}
```

Creamos la función de interpolación (ecuación 16):

```
InCub[x_]:= (For[j=1,x>vx[[j+1]] && j<n-1,j++];h=x-vx[[j]];
a[[j]]+(b[[j]]+(c[[j]]+d[[j]]*h)*h)*h)
```

Y calculamos los valores de "y" para "x=2" y "x=5.5" con esta función:

```
InCub[2]
0.540807
InCub[5.5]
1.70166
```

8.2.1.2. Ejemplo 2

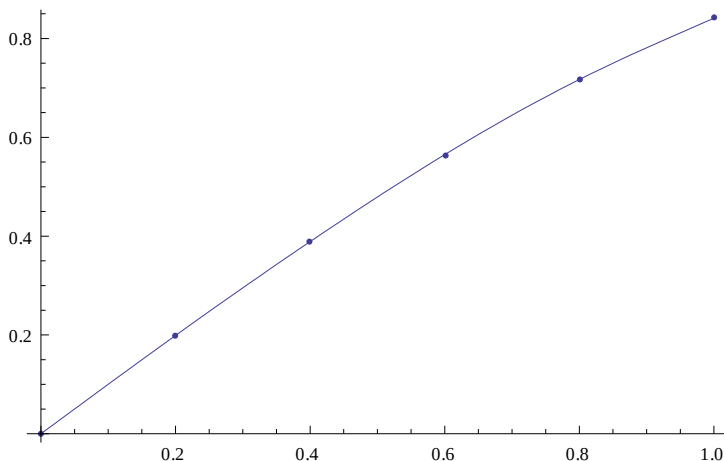
Como segundo ejemplo resolveremos el ejemplo 2 del método de Newton, es decir elaboraremos un módulo que, empleando los datos de la siguiente tabla, cree una función de interpolación aplicando interpolación segmentaria cúbica, grafique la función conjuntamente los datos conocidos y calcule los valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.2.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

El módulo que resuelve el problema es el siguiente:

```
Module[{n, InCub, puntos, a, h, b, q, r, s, c, d, j, g1, g2, p, vx,
  vy={0,0.199,0.389,0.565,0.717,0.841},
  vcx={0.1,0.3,0.5,0.7,1.2}},
  vx=Range[0,1,0.2];
  n=Length[vx];puntos=Transpose[{vx,vy}];
  h=Table[vx[[j+1]]-vx[[j]],{j,n-1}];a=vy;
  p=Table[3*((a[[j+1]]-a[[j]])/h[[j]]-(a[[j]]-a[[j-1]])/h[[j-1]]),{j,n-1}];
  q=ConstantArray[0,{n-1}];r=q;s=q;
  Do[q[[j]]=2*(h[[j-1]]+h[[j]])-h[[j-1]]*r[[j-1]];
    r[[j]]=h[[j]]/q[[j]];
    s[[j]]=(p[[j]]-h[[j-1]]*s[[j-1]])/q[[j]],{j,2,n-1}];
  c=ConstantArray[0,n];
  Do[c[[j]]=s[[j]]-r[[j]]*c[[j+1]],{j,n-1,2,-1}];
  d=Table[(c[[j+1]]-c[[j]])/(3*h[[j]]),{j,n-1}];
  b=Table[(a[[j+1]]-a[[j]])/h[[j]]-(2*c[[j]]+c[[j+1]])*h[[j]]/3,{j,n-1}];
  InCub[x_]:= (For[j=1,x>vx[[j+1]] && j<n-1,j++;h=x-vx[[j]];
    a[[j]]+(b[[j]]+(c[[j]]+d[[j]]*h)*h)*h);
  g1=Plot[InCub[x],{x,0,1}];
  g2=ListPlot[puntos];
  Print[Show[g1,g2]];
  Print["Valores de y interpolados: "];
  Do[Print["x = ",vcx[[i]]," y = ",InCub[vcx[[i]]]],{i,Length[vcx]}];
]
```

Ejecutando el módulo se obtiene:



Valores de y interpolados:

```
x = 0.1  y = 0.10015
x = 0.3  y = 0.295426
x = 0.5  y = 0.47927
x = 0.7  y = 0.644745
x = 1.2  y = 0.965
```

Valores que concuerdan bastante bien con los métodos de interpolación polinomial.

8.2.1.3. Ejemplo 3

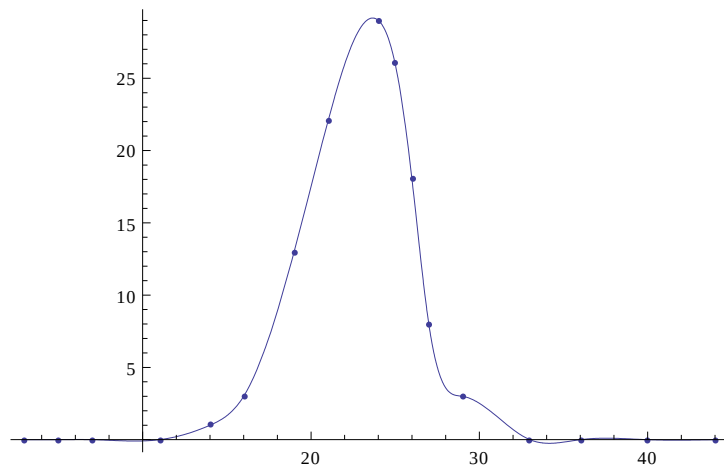
Como tercer ejemplo resolveremos el ejemplo 3 de Newton, es decir elaboraremos un módulo que empleando los datos de la siguiente tabla, cree una función que devuelva el valor "y" correspondiente a un determinado valor de "x", aplicando el método de interpolación segmentaria cúbica. El módulo debe graficar la función de interpolación conjuntamente los datos conocidos y emplear la función de interpolación para calcular valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

El módulo que resuelve el problema es el siguiente:

```
Module[{n, InCub, puntos, a, h, b, q, r, s, c, d, j, g1, g2, p,
  vx={3,5,7,11,14,16,19,21,24,25,26,27,29,33,36,40,44},
  vy={0,0,0,0,1,3,13,22,29,26,18,8,3,0,0,0,0,0},
  vcx={1,4,6,13,17,23,25.5,26.5,28,30,42,45}},
n=Length[vx]; puntos=Transpose[{vx,vy}];
h=Table[vx[[j+1]]-vx[[j]],{j,n-1}]; a=vy;
p=Table[3*((a[[j+1]]-a[[j]])/h[[j]]-(a[[j]]-a[[j-1]])/h[[j-1]]),{j,n-1}];
q=ConstantArray[0,{n-1}]; r=q; s=q;
Do[q[[j]]=2*(h[[j-1]]+h[[j]])-h[[j-1]]*r[[j-1]];
  r[[j]]=h[[j]]/q[[j]];
  s[[j]]=(p[[j]]-h[[j-1]]*s[[j-1]])/q[[j]],{j,2,n-1}];
c=ConstantArray[0,n];
Do[c[[j]]=s[[j]]-r[[j]]*c[[j+1]],{j,n-1,2,-1}];
d=Table[(c[[j+1]]-c[[j]])/(3*h[[j]]),{j,n-1}];
b=Table[(a[[j+1]]-a[[j]])/h[[j]]-(2*c[[j]]+c[[j+1]])*h[[j]]/3,{j,n-1}];
InCub[x_]:= (For[j=1,x>vx[[j+1]] && j<n-1,j++]; h=x-vx[[j]];
  a[[j]]+(b[[j]]+(c[[j]]+d[[j]]*h)*h)*h);
g1=Plot[InCub[x],{x,3,44}];
g2=ListPlot[puntos];
Print[Show[g1,g2]];
Print["Valores de y interpolados: "];
Do[Print["x = ",vcx[[i]]," y = ",InCub[vcx[[i]]]/N],{i,Length[vcx]}];
]
```

Ejecutando el módulo se obtiene:



Valores de y interpolados:

```
x = 1      y = 0.
x = 4      y = -0.00288212
x = 6      y = 0.00864635
x = 13     y = 0.546934
x = 17     y = 5.45266
x = 23     y = 28.5916
x = 25.5   y = 22.5904
x = 26.5   y = 12.7058
x = 28     y = 3.58993
x = 30     y = 2.51768
x = 42     y = -0.0315977
x = 45     y = 0.0197486
```

Como se puede observar, la interpolación segmentaria cúbica se ajusta mucho mejor a los datos que los métodos de interpolación polinomial. Aún existen ligeras variaciones en los extremos de la curva, sin embargo, es probable que dichas variaciones se deban a la inexactitud de los datos y no al método en sí.

8.3. Interpolation

Mathematica cuenta con una función para la interpolación segmentaria: "Interpolation". La sintaxis de esta función es la siguiente:

Interpolation[datos]

Donde "datos", al igual que en "InterpolatingPolynomial", son los valores conocidos en forma de una matriz, estando en la primera columna los valores de la variable independiente (o las variables independientes en forma de listas) y en la segunda los valores de la variable dependiente.

"Interpolation" devuelve una función de interpolación (InterpolatingFunction), la cual puede ser empleada directamente para calcular los valores de la variable dependiente.

Por defecto la ecuación que emplea "Interpolation" en los segmentos es la ecuación cúbica (igual que en la interpolación segmentaria cúbica), pero la misma puede ser cambiada con la opción "InterpolationOrder", así con "InterpolationOrder->1" se emplea una ecuación lineal (interpolación segmentaria lineal), con "InterpolationOrder->2" una ecuación de segundo grado, etc.

8.3.1. Ejemplos

8.3.1.1. Ejemplo 1

Como primer ejemplo emplearemos "Interpolation" para resolver el ejemplo 1 de Newton, es decir calcular valores de "y" para valores de "x" iguales a 2 y 5.5 con los datos de la tabla:

x	1	4	5	6
y	0	1.3862944	1.6094379	1.7917595

Primero guardamos los datos en dos vectores y el número de elementos en una variable:

```
vx={1,4,5,6};vy={0,1.386294,1.6094379,1.7917595};n=Length[vx];
```

Creemos la matriz de datos:

```
puntos=Transpose[{vx,vy}]
{{1,0},{4,1.38629},{5,1.60944},{6,1.79176}}
```

Creemos la función de interpolación con "Interpolation":

```
f3=Interpolation[puntos]
InterpolatingFunction[{{1.,6.}},<>]
```

Y calculamos los valores de "y" para "x=2" y "x=5.5":

```
f3[{2,5.5}]
{0.628768,1.70275}
```

Estos valores no concuerdan exactamente con los obtenidos en la interpolación segmentaria cúbica. Esto se debe a que las ecuaciones que emplea Mathematica, no son las mismas que hemos empleado en el acápite anterior. Es posible llegar a diferentes ecuaciones en función a los valores asumidos para " c_1 " y " c_n ".

Crearemos ahora la función de interpolación pero empleando ecuaciones lineales en lugar de cúbicas:

```
f1=Interpolation[puntos,InterpolationOrder->1]
InterpolatingFunction[{{1.,6.}},<>]
```

Y calculamos los valores de "y" para "x=2" y "x=5.5":

```
f1[{2,5.5}]
{0.462098,1.7006}
```

Como se puede observar estos valores concuerdan exactamente con los obtenidos en la interpolación segmentaria lineal, esto debido a que en una ecuación lineal no es necesario hacer ninguna suposición.

8.3.1.2. Ejemplo 2

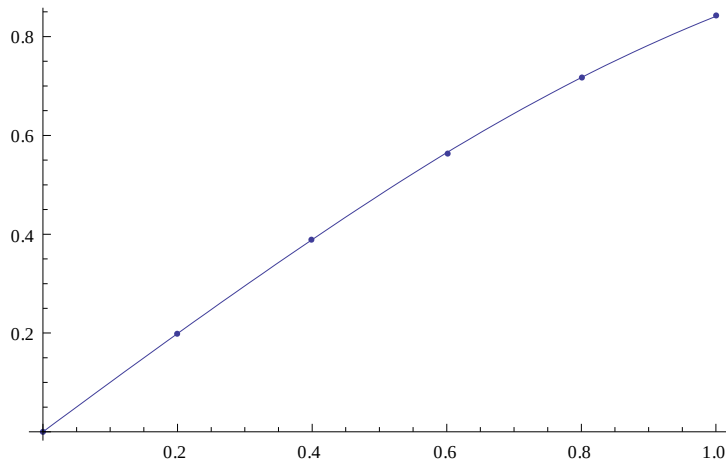
Como segundo ejemplo resolveremos el ejemplo 2 del método de Newton, es decir elaboraremos un módulo que, empleando los datos de la siguiente tabla, cree una función de interpolación "Interpolation", grafique la función conjuntamente los datos conocidos y calcule los valores de "y" para valores de "x" iguales a 0.1, 0.3, 0.5, 0.7, 0.9 y 1.2.

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

El módulo que resuelve el problema es el siguiente:

```
Module[{n,Inter,g1,g2,puntos,vx,
  vy={0,0.199,0.389,0.565,0.717,0.841},
  vcx={0.1,0.3,0.5,0.7,1.2}},
  vx=Range[0,1,0.2];
  n=Length[vx];puntos=Transpose[{vx,vy}];
  Inter=Interpolation[puntos];
  g1=Plot[Inter[x],{x,0,1}];
  g2=ListPlot[puntos];
  Print[Show[g1,g2]];
  Print["Valores de y interpolados: "];
  Do[Print["x = ",vcx[[i]]," y = ",Inter[vcx[[i]]],{i,Length[vcx]}];
]
```

Ejecutando el módulo se obtienen los resultados:



Valores de y interpolados:

```
x = 0.1    y = 0.100313
x = 0.3    y = 0.295437
x = 0.5    y = 0.479375
x = 0.7    y = 0.64425
```

InterpolatingFunction::dmval : Input value {1.2} lies outside the range of data in the interpolating function. Extrapolation will be used. >>

```
x = 1.2    y = 0.933
```

Como se puede observar, en este caso "Interpolation" nos muestra un mensaje indicando que el valor de "y" para "x=1.2" está fuera del rango, por lo que su cálculo involucra una extrapolación en lugar de una interpolación. Este mensaje puede ser evitado apagando el mismo con la instrucción Off:

```
Off[InterpolatingFunction::dmval]
```

La cual debe ser escrita antes de llamar a la función de interpolación.

8.3.1.3. Ejemplo 3

Como tercer ejemplo resolveremos el ejemplo 3 de Newton, es decir elaboraremos un módulo que empleando los datos de la siguiente tabla, cree una función que devuelva el valor "y" correspondiente a un determinado valor de "x", aplicando "Interpolation". El módulo debe graficar la función de interpolación conjuntamente los datos conocidos y emplear la función de interpolación para calcular valores de "y" para valores de "x" iguales a 1, 4, 6, 13, 17, 23, 25.5, 26.5, 28, 30, 42 y 45.

x	3	5	7	11	14	16	19	21	24	25	26	27	29	33	36	40	44
y	0	0	0	0	1	3	13	22	29	26	18	8	3	0	0	0	0

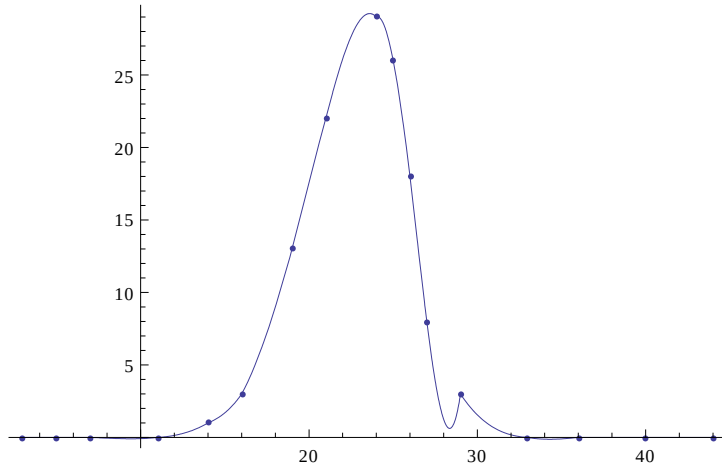
El módulo que resuelve el problema es el siguiente y en el mismo se evita la presentación de los mensajes: "InterpolatingFunction::dmval", apagándolo con "Off":

```
Module[{Inter,g1,g2,n,puntos,
  vx={3,5,7,11,14,16,19,21,24,25,26,27,29,33,36,40,44},
  vy={0,0,0,0,1,3,13,22,29,26,18,8,3,0,0,0,0},
  vcx={1,4,6,13,17,23,25.5,26.5,28,30,42,45}},
n=Length[vx];puntos=Transpose[{vx,vy}];
Off[InterpolatingFunction::dmval];
Inter=Interpolation[puntos];
g1=Plot[Inter[x],{x,3,44}];
g2=ListPlot[puntos];
```



```
Print[Show[g1,g2]];
Print["Valores de y interpolados: "];
Do[Print["x = ",vcx[[i]]," y = ",Inter[vcx[[i]]]/N,{i,Length[vcx]}];
]
```

Ejecutando el módulo se obtiene:



Valores de y interpolados:

```
x = 1    y = 0.
x = 4    y = 0.
x = 6    y = 0.
x = 13   y = 0.457143
x = 17   y = 5.6
x = 23   y = 28.7333
x = 25.5 y = 22.4375
x = 26.5 y = 12.9219
x = 28   y = 1.25
x = 30   y = 1.55952
x = 42   y = 0.
x = 45   y = 0.
```

Como se puede ver y al igual que ocurre con la interpolación segmentaria cúbica, los valores calculados con "Interpolating" se ajustan mucho mejor a los datos que los métodos de interpolación polinomial. En este caso "Interpolating" predice mejor los valores en los extremos, pero presenta una discontinuidad más marcada en el segmento comprendido entre 27 y 29. Como se dijo lo más probable es que se deba a errores en los datos originales y no al método en sí.

8.4. Ejercicios

1. Elabore un módulo que empleando los datos de la siguiente tabla y el método de interpolación lineal, cree una función de interpolación, grafique la función de interpolación conjuntamente los datos y emplee la función de interpolación para calcular valores de "y" para valores de "x" iguales a 3.9, 4.8, 6.1, 6.57, 6.7 y 7.1.

x	3.5	4	4.5	5	5.5	6	6.2	6.5	6.8
y	0.002	0.030	0.155	0.396	0.658	0.903	0.933	0.975	0.993

2. Resuelva el ejercicio 1 empleando el método de Lagrange.
3. Resuelva el ejercicio 1 empleando el método de Newton.
4. Resuelva el ejercicio 1 empleando InterpolatingPolynomial.

5. Resuelva el ejercicio 1 empleando interpolación segmentaria lineal.
6. Resuelva el ejercicio 1 empleando interpolación segmentaria cúbica.
7. Resuelva el ejercicio 1 empleando Interpolation.
8. Elabore un módulo que empleando los datos de la siguiente tabla y el método de interpolación lineal, cree una función de interpolación, grafique la función de interpolación conjuntamente los datos y emplee la función de interpolación para calcular valores de "y" para valores de "x" iguales 0.5, 2.5, 3.2, 4.3, 5.5, 6.5, 8.5, 10.5, 13.5, 15.5, 18.5, 20.5, 22.5 y 24.

x	1	2	3	3.5	4	4.5	5	6	7	8	9	10	12	13	14	15	16	17	18	19	20	21	22	23
y	0	0	0.5	1.5	3	5	9	11.8	12	12.5	12.5	12.5	12.5	12.4	12.0	11.5	10.5	9.5	8.3	7.0	5.5	4.0	3.0	2.5

9. Resuelva el ejercicio 8 empleando el método de Lagrange.
10. Resuelva el ejercicio 8 empleando el método de Newton.
11. Resuelva el ejercicio 8 empleando InterpolatingPolynomial.
12. Resuelva el ejercicio 8 empleando interpolación segmentaria lineal.
13. Resuelva el ejercicio 8 empleando interpolación segmentaria cúbica.
14. Resuelva el ejercicio 8 empleando Interpolation.
15. Elabore un módulo que calcule y muestre los valores de " x_i ", " y_i ", " k_x " y " k_y ", resolviendo, con FindRoot, la función implícita, que se muestra a continuación (empleando un valor inicial igual a 0.5). Para obtener los valores de " y_i " debe crear una función de interpolación, que empleando los datos de la tabla y el método de interpolación lineal, calcule valores de " y_i " para valores dados de " x_i ".

$$f(x_i) = \frac{k_x}{k_y} + \frac{y - y_i}{x - x_i} = 0$$

$$k_y = \frac{k'_y}{(1-y)_{im}}; \quad (1-y)_{im} = \frac{(1-y_i) - (1-y)}{\ln\left(\frac{1-y_i}{1-y}\right)}$$

$$k_x = \frac{k'_x}{(1-x)_{im}}; \quad (1-x)_{im} = \frac{(1-x) - (1-x_i)}{\ln\left(\frac{1-x}{1-x_i}\right)}$$

$$k'_x = 1.967 \times 10^{-3}; \quad k'_y = 1.465 \times 10^{-3}; \quad y = 0.39; \quad x = 0.1$$

x_i	0	0.05	0.10	0.15	0.20	0.25	0.30	0.35
y_i	0	0.022	0.052	0.087	0.131	0.187	0.265	0.385

16. Resuelva el ejercicio 15 empleando "NewtonDN" y "Lagrange".
17. Resuelva el ejercicio 15 empleando "SegSol" (valor inicial 0.1, incremento 0.01), "RegulaFalsi" y "Lagrange".
18. Resuelva el ejercicio 15 empleando "SegSol" (valor inicial 0.1, incremento 0.01), "Bisección" y "Newton".
19. Resuelva el ejercicio 15 empleando "Incremental" (valor inicial 0.1, incremento 0.01) e "InterpolatingPolynomial".

20. Resuelva el ejercicio 15 empleando "Secante" (valores iniciales 0.1 y 0.2) e "Interpolación segmentaria lineal".
21. Resuelva el ejercicio 15 empleando "SegSol" (valor inicial 0.1, incremento 0.01), "Secante" e "Interpolación segmentaria cúbica".
22. Resuelva el ejercicio 15 empleando "FindRoot" (valor inicial 0.24), e "Interpolation".

25. INTEGRACIÓN NUMÉRICA

La integración numérica se emplea cuando las funciones a integrar se encuentran en forma tabular, gráfica o son demasiado complejas como para integrarlas analíticamente. Sin embargo, aún en situaciones donde las funciones no son muy complejas se recurre a la integración numérica, simplemente porque es más sencillo integrar una función numéricamente.

Existen muchos métodos que nos permiten integrar funciones o tablas numéricamente. En este capítulo estudiaremos los métodos del *Trapecio*, *Romberg* y *Simpson*, además estudiaremos la función "NIntegrate" de Mathematica.

25.1. Método del trapecio

La integral se define como el área bajo la curva de la función (ver figura);

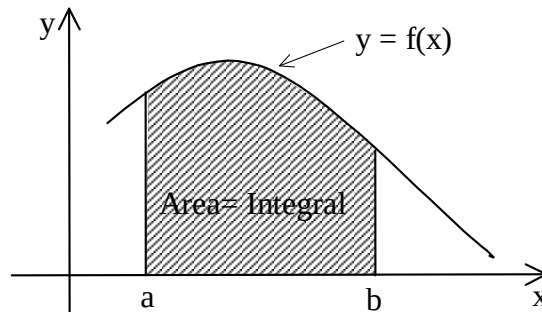


Figura 25.1. Representación gráfica de la integral

Todos los métodos numéricos tratan de calcular esta área empleando para ello diferentes aproximaciones. Si se logra calcular esta área se tiene entonces el valor de la integral:

$$\int_a^b f(x) * dx = \text{Area bajo la curva} \quad (1)$$

Con el fin de facilitar los cálculos, el área bajo la curva puede ser dividido en "n" segmentos, tal como se muestra en la Figura 25.2.

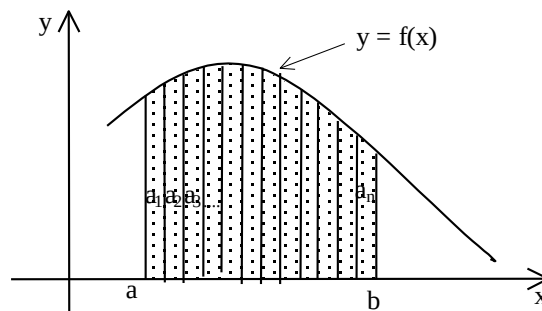


Figura 25.2. La integral (área bajo la curva) dividida en "n" segmentos

Entonces la integral (área bajo la curva) se calcula sumamos las áreas de cada uno de los segmentos:

$$\int_a^b f(x) * dx = \text{Area bajo la curva} = \sum_{i=1}^n a_i \quad (2)$$

En el método del trapecio se asume que cada uno de estos segmentos tiene la forma de un trapecio. En otras palabras se asume que la curva entre dos puntos consecutivos es una línea recta.

Como se puede observar en los segmentos ampliados de la Figura 25.3, el asumir que los segmentos están unidos por una línea recta ocasiona un error (por exceso o por defecto) en el cálculo de las áreas de los segmentos. Este error es más grande cuanto más grande es el ancho del segmento y menor cuanto más pequeño es el ancho (dentro de los límites de precisión que nos permite la computadora o calculadora con la que estemos trabajando).

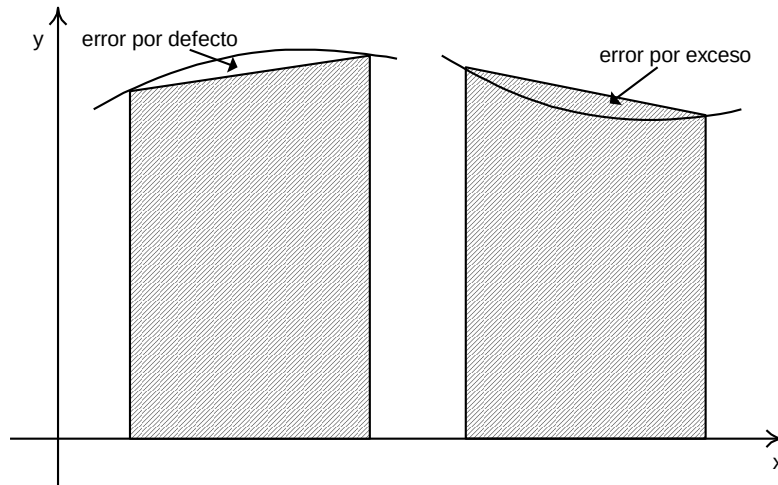


Figura 25.3. Vista ampliada de dos segmentos.

Como se ha asumido que los segmentos son trapecios, el área de cualquier segmento i , puede ser calculado con la siguiente ecuación:

$$a_i = \frac{(x_{i+1} - x_i)(y_{i+1} + y_i)}{2} \quad (3)$$

Y si llamamos " h_i " al ancho del segmento:

$$h_i = x_{i+1} - x_i \quad (4)$$

La ecuación (4) se transforma en:

$$a_i = \frac{h_i}{2}(y_i + y_{i+1}) \quad (5)$$

Aplicando esta ecuación (o la ecuación 3) podemos calcular el área de cualquier segmento y con estas áreas y la ecuación (2) la integral (el área bajo la curva).

25.1.1. Integración de datos tabulados

Para integrar datos tabulados simplemente se aplica la ecuación (3) o (4), conjuntamente la ecuación (2), a los puntos (x_i, y_i) de la tabla.

25.1.2. Ejemplos para la forma tabular

25.1.2.1. Ejemplo 1

Como primer ejemplo calcularemos el valor de la siguiente integral en base a los datos de la siguiente tabla empleando el método del Trapecio.

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

X	Y
2.0	4.800
2.5	6.625
2.7	7.509
3.0	9.00
3.3	10.689
3.8	13.944
4.2	16.944
4.5	19.425
5.1	24.981
5.7	31.329
6.0	34.800
6.4	39.736
6.9	46.401
7.2	50.664
7.5	55.125

Primero guardamos los datos de esta tabla en dos vectores y el número de elementos en una variable:

```
vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,34.800,
39.736,46.401,50.664,55.125};
n=Length[vx]-1;
```

Ahora calculamos la integral, aplicando las ecuaciones (2) y (3) a los segmentos comprendidos entre 2.5 y 7.2 (entre el segundo y penúltimo puntos):

```
Sum[(vx[[i+1]]-vx[[i]])*(vy[[i+1]]+vy[[i]])/2,{i,2,n-1}]
115.77
```

Por lo tanto el valor de la integral es 115.77.

25.1.2.2. Ejemplo 2

Como segundo ejemplo calcularemos la siguiente integral empleando los mismos datos del ejemplo anterior:

$$\int_{2.2}^{7.4} f(x)dx = \int_{2.2}^{7.4} ydx$$

En este caso los límites no corresponden exactamente a los puntos de la tabla, por lo que es necesario calcular los puntos inicial y final por interpolación. En este ejemplo emplearemos "Interpolation" para calcular dichos valores:

```
puntos=Transpose[{vx,vy}];f=Interpolation[puntos]
InterpolatingFunction[{{2.,7.5}},<>]
```

Copiamos los datos del vector "vx" en otra variable y reemplazamos el primer y último datos por 2.2 y 7.4 respectivamente:

```
vxn=vx;vxn[{{1,n}}]={2.2,7.4};vxn
{2.2,2.5,2.7,3.,3.3,3.8,4.2,4.5,5.1,5.7,6.,6.4,6.9,7.2,7.4}
```

Copiamos los datos del vector "vy" en otra variable y reemplazamos el primer y último datos por los valores de "y" interpolados para 2.2 y 7.4:

```
vyn=vy;vyn[[{1,n}]]={f[2.2],f[7.4]};vyn
{5.464,6.625,7.509,9.,10.689,13.944,16.944,19.425,24.981,31.329,34.8,39.736,
46.401,50.664,53.616}
```

Ahora simplemente aplicamos las ecuaciones (2) y (3) a todos estos puntos:

```
Sum[(vxn[[i+1]]-vxn[[i]])*(vyn[[i+1]]+vyn[[i]])/2,{i,n}]
128.011
```

Por lo tanto el valor de la integral es 128.011.

25.1.2.3. Ejemplo 3

Como tercer ejemplo emplearemos el método del trapecio para calcular el valor de la siguiente integral:

$$\int_{5.1}^{8.4} y dx = \int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1}) dx$$

Cuando se tiene una función, en lugar de datos tabulados, simplemente se crea una tabla con la función. Para ello se calculan valores de "x" igualmente espaciados y con los mismos se calculan los respectivos valores de "y". El espaciamiento entre los sucesivos valores de "x", al cual llamaremos "h", se calcula con la siguiente expresión:

$$h = \frac{\text{Límite superior} - \text{Límite inferior}}{\text{Número de segmentos}} \quad (6)$$

En teoría, mientras mayor es el número de segmentos (menor el valor de "h") más exacto será el valor de la integral, sin embargo, en la práctica no es conveniente emplear un número demasiado grande de segmentos, porque con ello comienzan a crecer los errores de redondeo, consiguiéndose el efecto contrario al deseado.

Para este ejemplo fijaremos en 30 el número de segmentos, por lo tanto los valores de "x" son:

```
a=5.1;b=8.4;n=30;h=(b-a)/n;x=a-h;
vx=Table[x+=h,{n+1}]
{5.1,5.21,5.32,5.43,5.54,5.65,5.76,5.87,5.98,6.09,6.2,6.31,6.42,6.53,6.64,6.
75,6.86,6.97,7.08,7.19,7.3,7.41,7.52,7.63,7.74,7.85,7.96,8.07,8.18,8.29,8.4}
```

Los respectivos valores de "y" calculados con la función son:

```
fx[x_]:=3-6*x^2+1.1*x^3.1
vy=fx[vx]
{18.6756,23.6158,28.9433,34.6704,40.8095,47.3731,54.3736,61.8236,69.7354,78.
1217,86.9949,96.3677,106.253,116.662,127.609,139.107,151.166,163.802,177.025
,190.85,205.288,220.352,236.056,252.412,269.432,287.131,305.52,324.612,344.4
21,364.96,386.24}
```

Entonces el área bajo la curva (la integral) es:

```
Sum[(vx[[i+1]]-vx[[i]])*(vy[[i+1]]+vy[[i]])/2,{i,n}]
528.874
```

25.1.3. Integración de funciones analíticas

Como se ha visto en el último ejemplo, el método del trapecio en su forma tabular puede ser empleado también para resolver funciones analíticas. Sin embargo, puesto que en estos casos el espaciamiento entre puntos (*h*) es uniforme, se puede simplificar la expresión factorizando dichos valores:

$$\begin{aligned}
\int_a^b f(x) dx &= \sum_{i=1}^n a_i = \frac{h}{2}(y_1 + y_2) + \frac{h}{2}(y_2 + y_3) + \frac{h}{2}(y_3 + y_4) + \dots + \frac{h}{2}(y_{n-1} + y_n) + \frac{h}{2}(y_n + y_{n+1}) \\
\int_a^b f(x) dx &= \frac{h}{2}(y_1 + y_2 + y_2 + y_3 + y_3 + y_4 + \dots + y_{n-1} + y_n + y_n + y_{n+1}) \\
\int_a^b f(x) dx &= \frac{h}{2}(y_1 + 2y_2 + 2y_3 + 2y_4 + \dots + 2y_{n-1} + 2y_n + y_{n+1}) \\
\int_a^b f(x) dx &= \frac{h}{2} \left(y_1 + 2 \sum_{i=2}^n y_i + y_{n+1} \right) \quad (7)
\end{aligned}$$

Esta ecuación es conocida como la *regla del trapecio* y es la forma que se emplea para integrar funciones analíticas.

25.1.4. Ejemplos para la forma analítica

25.1.4.1. Ejemplo 1

Como primer ejemplo volveremos a calcular la siguiente integral, pero ahora aplicando la ecuación (7):

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1}) dx$$

Primero programamos la función:

```
fx[x_]:=3-6*x^2+1.1*x^3.1
```

Asignamos a variables los límites, el número de segmentos y calculamos el valor de "h":

```
a=5.1;b=8.4;n=30;h=(b-a)/n;x=a;
```

Entonces calculamos el valor de la integral aplicando la ecuación (7):

```
h/2*(fx[a]+2*Sum[x+=h;fx[x],{i,2,n}]+fx[b])
528.874
```

Y como era de esperar, obtenemos el mismo resultado que con la forma tabular. Podemos incrementar el número de segmentos a 100 para obtener un resultado más preciso:

```
n=100;h=(b-a)/n;x=a;
h/2*(fx[a]+2*Sum[x+=h;fx[x],{i,2,n}]+fx[b])
528.733
```

25.1.4.2. Ejemplo 2

Como segundo ejemplo, y con el propósito de demostrar la integración de datos tabulados empleando la forma analítica, volveremos a resolver el ejemplo 1 de la forma tabular:

$$\int_{2.5}^{7.2} f(x) dx = \int_{2.5}^{7.2} y dx$$

Para integrar datos tabulados con la forma analítica se debe crear una función de interpolación. Para ello se puede recurrir a alguno de los métodos estudiados en el anterior capítulo. En este ejemplo construiremos la función de interpolación con "Interpolation" (empleando los datos del mencionado ejemplo). El módulo que resuelve el problema es el siguiente:

```
Module[{vx,vy,puntos,f,a=2.5,b=7.2,n=20,h,x,vi},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,
    34.800,39.736,46.401,50.664,55.125};
  puntos=Transpose[{vx,vy}];
  f=Interpolation[puntos];
  h=(b-a)/n;x=a;
  vi=h/2*(f[a]+2*Sum[x+=h;f[x],{i,2,n}]+f[b]);
  Print["Valor de la Integral: ",vi]
]
```

Ejecutando el módulo se obtiene:

Valor de la Integral: 115.643

Resultado que concuerda bastante bien con el calculado empleando la forma tabular. Por supuesto es posible emplear cualquiera de los métodos de interpolación estudiados en el anterior capítulo, por ejemplo, la solución con "InterpolatingPolynomial" es:

```
Module[{vx,vy,puntos,f,a=2.5,b=7.2,n=20,h,x,vi},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,
    34.800,39.736,46.401,50.664,55.125};
  puntos=Transpose[{vx,vy}];
  f[x_]:=Evaluate[InterpolatingPolynomial[puntos,x]];
  h=(b-a)/n;x=a;
  vi=h/2*(f[a]+2*Sum[x+=h;f[x],{i,2,n}]+f[b]);
  Print["Valor de la Integral: ",vi]
]
```

Con el cual obtenemos el mismo resultado:

Valor de la Integral: 115.643

25.2. Método de Simpson

En el método del trapecio, se asume que los puntos están unidos por líneas rectas, en el método de Simpson se mejora esta aproximación pues se asume que cada tres puntos (2 segmentos) están unidos por una ecuación cuadrática, tal como se muestra en la Figura 25.4.

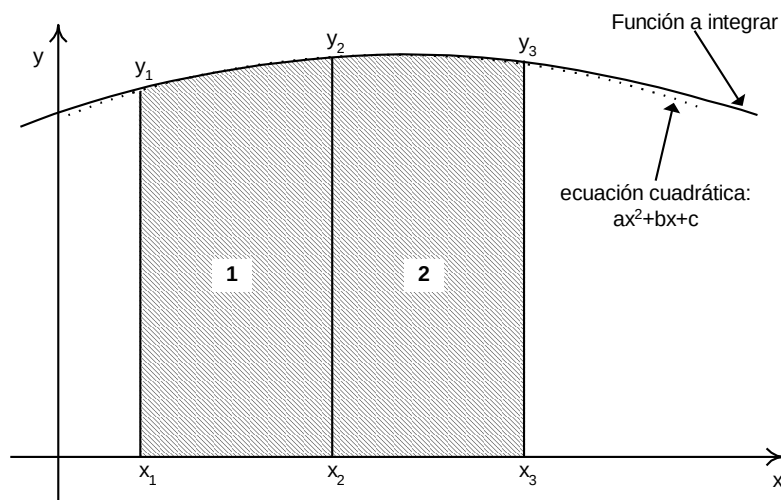


Figura 25.4. Área de dos segmentos en el método de Simpson

Entonces el área de 2 segmentos puede ser calculada integrando la ecuación cuadrática ax^2+bx+c :

$$\text{Área de dos segmentos} = \int_{x_1}^{x_3} (ax^2 + bx + c)dx = \frac{a}{3}(x_3^3 - x_1^3) + \frac{b}{2}(x_3^2 - x_1^2) + c(x_3 - x_1) \quad (8)$$

Si reemplazamos los tres puntos del segmento (x_1, y_1) , (x_2, y_2) y (x_3, y_3) en esta ecuación, se forma un sistema de 3 ecuaciones con 3 incógnitas:

$$\begin{aligned} y_1 &= ax_1^2 + bx_1 + c \\ y_2 &= ax_2^2 + bx_2 + c \\ y_3 &= ax_3^2 + bx_3 + c \end{aligned} \quad (9)$$

De este sistema podemos despejar los coeficientes a , b y c de la ecuación cuadrática:

$$\begin{aligned} a &= \frac{(y_1 - y_3)(x_1 - x_2) - (y_1 - y_2)(x_1 - x_3)}{(x_1^2 - x_3^2)(x_1 - x_2) - (x_1^2 - x_2^2)(x_1 - x_3)} \\ b &= \frac{(y_1 - y_2) - a(x_1^2 - x_2^2)}{(x_1 - x_2)} \\ c &= y_1 - ax_1^2 - bx_1 \end{aligned} \quad (10)$$

Si en lugar de los puntos 1, 2 y 3 tomamos en cuenta tres puntos consecutivos cualesquiera: i , j y k , la ecuación (8) toma la forma general:

$$\text{Área de los segmentos } i, j = \int_{x_i}^{x_k} (ax^2 + bx + c)dx = \frac{a}{3}(x_k^3 - x_i^3) + \frac{b}{2}(x_k^2 - x_i^2) + c(x_k - x_i) \quad (11)$$

Igualmente las ecuaciones (10) toman la forma general:

$$\begin{aligned} a &= \frac{(y_i - y_k)(x_i - x_j) - (y_i - y_j)(x_i - x_k)}{(x_i^2 - x_k^2)(x_i - x_j) - (x_i^2 - x_j^2)(x_i - x_k)} \\ b &= \frac{(y_i - y_j) - a(x_i^2 - x_j^2)}{(x_i - x_j)} \\ c &= y_i - ax_i^2 - bx_i \end{aligned} \quad (12)$$

Entonces el área total bajo la curva (la integral) puede ser calculada sumando las áreas de los "n" segmentos existentes entre el límite inferior (a) y superior (b):

$$\text{Área bajo la curva} = \int_a^b f(x)dx = \sum_{i=1,3,5}^{n-1} \text{área de los segmentos } i, i+1 \quad (13)$$

Para aplicar la ecuación (13) el número de segmentos "n" debe ser par, o lo que es lo mismo, el número de puntos debe ser impar.

Las ecuaciones (11) a (13) son las ecuaciones generales del método de Simpson. Estas ecuaciones permiten obtener resultados más exactos que el método del trapecio, porque, tal como se puede ver en la Figura 25.4, la ecuación cuadrática se ajusta mejor a los datos que la ecuación lineal.

25.2.1. Integración de datos tabulados

Cuando se integran datos tabulados puede suceder que el número de puntos

comprendidos entre los límites de integración sea par, o lo que es lo mismo el número de segmentos sea impar. Entonces no se pueden aplicar directamente las ecuaciones (11) a (13) pues las mismas han sido deducidas para un número par de segmentos.

En estos casos lo que se hace es calcular los coeficientes de la ecuación cuadrática con los tres primeros puntos, aplicando las ecuaciones (10). Los coeficientes calculados se emplean entonces para calcular el área del primer segmento:

$$a_1 = \int_{x_1}^{x_2} (ax^2 + bx + c)dx = \frac{a}{3}(x_2^3 - x_1^3) + \frac{b}{2}(x_2^2 - x_1^2) + c(x_2 - x_1) \quad (14)$$

Una vez calculada el área del primer segmento queda un número par de segmentos y como ya es par se pueden emplear las ecuaciones (11) a (13) para calcular el área de los segmentos restantes (comenzando con el segundo punto). Por lo tanto el área total bajo la curva (la integral) es:

$$\text{Area bajo la curva (n impar)} = \int_a^b f(x)dx = a_1 + \sum_{i=2,4,6,\dots}^{n-1} \text{area de los segmentos } i, i+1 \quad (15)$$

25.2.2. Ejemplos para la forma tabular

25.2.2.1. Ejemplo 1

Como primer ejemplo volveremos a calcular la integral:

$$\int_{2.5}^{7.2} f(x)dx$$

x	2.0	2.5	2.7	3.0	3.3	3.8	4.2	4.5	5.1	5.7	6.0	6.4	6.9	7.2	7.5
y	4.800	6.625	7.509	9.000	10.689	13.944	16.944	19.425	24.981	31.329	34.800	39.736	46.401	50.664	55.125

En este caso, entre los límites 2.5 y 7.2 existe un total de 13 puntos (12 segmentos), por lo que podemos aplicar directamente las ecuaciones (11) a (13). El módulo que resuelve el problema es el siguiente:

```
Module[{vx,vy,coef,as,a,b,c,n,vi},
vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
vx=Rest[vx];vx=Most[vx];
vy={4.800,6.625,7.509,9.000,10.689,13.944,16.944,19.425,24.981,31.329,
34.800,39.736,46.401,50.664,55.125};
vy=Rest[vy];vy=Most[vy];
n=Length[vx]-1;
coef[i_]:=Module[{j=i+1,k=i+2},
a=((vy[[i]]-vy[[k]])*(vx[[i]]-vx[[j]])-
(vy[[i]]-vy[[j]])*(vx[[i]]-vx[[k]]))/
((vx[[i]]^2-vx[[k]]^2)*(vx[[i]]-vx[[j]])-
(vx[[i]]^2-vx[[j]]^2)*(vx[[i]]-vx[[k]]));
b=((vy[[i]]-vy[[j]])-a*(vx[[i]]^2-vx[[j]]^2))/(vx[[i]]-vx[[j]]);
c=(vy[[i]]-a*vx[[i]]^2-b*vx[[i]]);
{a,b,c}];
as[a_,b_,c_,i_,k_]:=a/3*(vx[[k]]^3-vx[[i]]^3)+
b/2*(vx[[k]]^2-vx[[i]]^2)+c*(vx[[k]]-vx[[i]]);
vi=Sum[{a,b,c}=coef[i];as[a,b,c,i,i+2],{i,1,n,2}];
Print["Valor de la integral: ",vi]
]
```

Como se puede observar, en este módulo se quitan los primeros y últimos datos de los dos vectores (pues están fuera de los límites de integración),

se calculan los coeficientes (ecuación 12) con la función "coef", el área de dos segmentos (ecuación 11) con "as" y se calcula la sumatoria (la integral) aplicando la ecuación 13. Ejecutando el módulo se obtiene:

Valor de la integral: 115.595

25.2.2.2. Ejemplo 2

Como segundo ejemplo calcularemos el valor de la integral, empleando los mismos datos del ejemplo 1:

$$\int_{2.0}^{7.2} f(x)dx$$

En este caso el número de puntos comprendidos entre el límite inferior y superior es 14 (13 segmentos), por lo tanto se deben aplicar primero las ecuaciones (10) y (14) y luego las ecuaciones (11), (12) y (15), ello simplemente implica cambiar los índices para el primer segmento. El módulo que resuelve el problema es el siguiente:

```
Module[{vx,vy,coef,as,a,b,c,n,al,vi},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vx=Most[vx];
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,
    34.800,39.736,46.401,50.664,55.125};
  vy=Most[vy];
  n=Length[vx]-1;
  coef[i_]:=Module[{j=i+1,k=i+2},
    a=((vy[[i]]-vy[[k]])*(vx[[i]]-vx[[j]])-
      (vy[[i]]-vy[[j]])*(vx[[i]]-vx[[k]]))/
      ((vx[[i]]^2-vx[[k]]^2)*(vx[[i]]-vx[[j]])-
      (vx[[i]]^2-vx[[j]]^2)*(vx[[i]]-vx[[k]]));
    b=((vy[[i]]-vy[[j]])-a*(vx[[i]]^2-vx[[j]]^2))/(vx[[i]]-vx[[j]]);
    c=(vy[[i]]-a*vx[[i]]^2-b*vx[[i]]);
    {a,b,c}};
  as[a_,b_,c_,i_,k_]:=a/3*(vx[[k]]^3-vx[[i]]^3)+
    b/2*(vx[[k]]^2-vx[[i]]^2)+c*(vx[[k]]-vx[[i]]);
  {a,b,c}=coef[1];
  al=as[a,b,c,1,2];
  vi=al+Sum[{a,b,c}=coef[i];as[a,b,c,i,i+2]},{i,2,n-1,2}];
  Print["Valor de la integral: ",vi]
]
```

Ejecutando el módulo se obtiene:

Valor de la integral: 118.428

25.2.2.3. Ejemplo 3

Como tercer ejemplo volveremos a calcular el valor de la siguiente integral, empleando el método de Simpson en su forma tabular:

$$\int_{5.1}^{8.4} ydx = \int_{5.1}^{8.4} (3-6x^2+1.1x^{3.1})dx$$

Igual que con el método del trapecio, lo que se hace es generar una tabla con la función (cuidando en este caso que el número de puntos sea impar). Una vez generados los puntos simplemente se aplican las ecuaciones del método de Simpson. El módulo que resuelve el problema, donde se generan 21 puntos, es el siguiente:

```

Module[{li=5.1,ls=8.4,n=21,h,x,fx,vx,vy,coef,as,a,b,c,vi},
h=(ls-li)/n;
x=li-h;vx=Table[x+=h,{n+1}];
fx[x_]:=3-6*x^2+1.1*x^3.1;
vy=fx[vx];
coef[i_]:=Module[{j=i+1,k=i+2},
a=((vy[[i]]-vy[[k]])*(vx[[i]]-vx[[j]])-
(vy[[i]]-vy[[j]])*(vx[[i]]-vx[[k]]))/
((vx[[i]]^2-vx[[k]]^2)*(vx[[i]]-vx[[j]])-
(vx[[i]]^2-vx[[j]]^2)*(vx[[i]]-vx[[k]]));
b=((vy[[i]]-vy[[j]])-a*(vx[[i]]^2-vx[[j]]^2))/(vx[[i]]-vx[[j]]);
c=(vy[[i]]-a*vx[[i]]^2-b*vx[[i]]);
{a,b,c}};
as[a_,b_,c_,i_,k_]:=a/3*(vx[[k]]^3-vx[[i]]^3)+
b/2*(vx[[k]]^2-vx[[i]]^2)+c*(vx[[k]]-vx[[i]]);
vi=Sum[{a,b,c}=coef[i];as[a,b,c,i,i+2],{i,1,n,2}];
Print["Valor de la integral: ",vi]
]

```

Ejecutando el módulo se obtiene:

Valor de la integral: 528.719

Que es la solución exacta.

25.2.3. Integración de funciones analíticas

Como se ha demostrado en el último ejemplo, el método de Simpson, en su forma tabular, puede ser empleado también para resolver funciones analíticas. Sin embargo, cuando se tiene una función analítica, debido a que el espacio entre los puntos es constante, las ecuaciones pueden ser simplificadas. Así el cálculo de los coeficientes (para tres puntos consecutivos i, j, k) se simplifica a:

$$\begin{aligned}
 a &= \frac{y_i - 2y_j + y_k - y_j}{2h^2} \\
 b &= \frac{y_k - y_i}{2h} \\
 c &= y_j
 \end{aligned} \tag{16}$$

Donde h es el espaciamiento (o incremento de la variable independiente), tal como fue definida en la ecuación (6). Entonces, el cálculo del área de dos segmentos consecutivos se reduce a:

$$a_{i,j} = \frac{2a}{3}h^3 + 2ch = \frac{h}{3}(y_i + 4y_j + y_k) \tag{17}$$

Reemplazando la ecuación (17) en la ecuación (13):

$$\begin{aligned}
 \int_a^b f(x) * dx &= \sum_{i=1,3,5,\dots}^{n-1} a_{ij} = \frac{h}{3}(y_1 + 4y_2 + y_3 + y_3 + 4y_4 + y_5 + y_5 + 4y_6 + y_7 + \dots + y_{n-1} + 4y_n + y_{n+1}) \\
 \int_a^b f(x) * dx &= \frac{h}{3}(y_1 + 4(y_2 + y_4 + y_6 + \dots + y_n) + 2(y_3 + y_5 + y_7 + \dots + y_{n-1}) + y_{n+1})
 \end{aligned}$$

La ecuación (13) se reduce a

$$\int_a^b f(x) * dx = \frac{h}{3}(y_1 + 4 \sum_{i=2,4,6,\dots}^n y_i + 2 \sum_{i=1,3,5,\dots}^{n-1} y_i + y_{n+1}) \tag{18}$$

Que es la ecuación conocida como la *regla de Simpson* y es la que se emplea cuando se integran ecuaciones analíticas. En esta ecuación el número de segmentos siempre tiene que ser par, pero ello no es problema cuando se integran ecuaciones analíticas pues siempre es posible fijar un número impar de puntos.

Al igual que en el método del trapecio, " y_1 " es el valor de la función para " $x=a$ " (límite inferior), " y_2 " el valor de la función para " $x=x+h$ ", y_3 el valor de la función para " $x=x+2h$ " y así sucesivamente siendo " y_{n+1} " el valor de la función para " $x=b$ " (límite superior).

25.2.4. Ejemplos para la forma analítica

25.2.4.1. Ejemplo 1

Como primer ejemplo volveremos a resolver la integral:

$$\int_{5.1}^{8.4} (3-6x^2+1.1x^{3.1})dx$$

El módulo que resuelve el problema, fijando en 31 el número de puntos es el siguiente:

```
Module[{li=5.1,ls=8.4,n=30,f,h,x,sp,si,vi},
  f[x_]:=3-6*x^2+1.1*x^3.1;
  h=(ls-li)/n;
  sp=(x=li-h;Sum[x+=2*h;f[x],{i,2,n,2}]);
  si=(x=li;Sum[x+=2*h;f[x],{i,3,n-1,2}]);
  vi=h/3*(f[li]+4*sp+2*si+f[ls]);
  Print["Valor de la integral: ",vi]
]
```

Ejecutando (Shift+Enter) el módulo se obtiene:

Valor de la integral: 528.719

Que como ya vimos es el resultado exato.

25.2.4.2. Ejemplo 2

Como segundo ejemplo, volveremos a calcular el valor de integral:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

x	2.0	2.5	2.7	3.0	3.3	3.8	4.2	4.5	5.1	5.7	6.0	6.4	6.9	7.2	7.5
y	4.800	6.625	7.509	9.000	10.689	13.944	16.944	19.425	24.981	31.329	34.800	39.736	46.401	50.664	55.125

Como en el método del trapecio, para integrar datos tabulados con la forma analítica, se debe crear una función de interpolación, recurriendo para ello a cualquiera de los métodos estudiados en el anterior capítulo. En este ejemplo volveremos a emplear "Interpolation".

```
Module[{vx,vy,puntos,f,li=2.5,ls=7.2,n=20,h,x,vi},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,34.800,
  39.736,46.401,50.664,55.125};
  puntos=Transpose[{vx,vy}];
  f=Interpolation[puntos];
  h=(ls-li)/n;
```

```

sp=(x=li-h;Sum[x+=2*h;f[x],{i,2,n,2}]);
si=(x=li;Sum[x+=2*h;f[x],{i,3,n-1,2}]);
vi=h/3*(f[li]+4*sp+2*si+f[ls]);
Print["Valor de la integral: ",vi]
]

```

Ejecutando el módulo se obtiene:

Valor de la integral: 115.595

Que una vez más es el resultado exacto.

25.3. Método de Romberg

El método de Romberg es básicamente la regla del trapecio a la cual se aplica el método de extrapolación de Richardson.

En este método se aplica el método del trapecio para calcular inicialmente dos áreas: una tomando toda el área como un solo segmento ($I_{0,1}$) y la segunda dividiendo el área en dos segmentos ($I_{0,2}$). Entonces con la fórmula de Richardson se extrapola un valor a partir de estos dos valores calculados.

La fórmula de extrapolación de Richardson es la siguiente:

$$I_{n,k} = \frac{4^n I_{n-1,k+1} - I_{n-1,k}}{4^n - 1} \quad (19)$$

Donde "n" es el nivel de extrapolación, "k" es el k-ésimo valor dentro del nivel al que pertenece (n o $n-1$), $I_{n-1,k+1}$ es el valor de la integral en el nivel de extrapolación $n-1$ y en la posición $k+1$, $I_{n-1,k}$ es el valor de la integral en el nivel de extrapolación $n-1$ y en la posición k .

Aplicando la ecuación (19) a los dos valores calculados con el método del trapecio (que como no son extrapolados pertenecen al nivel 0), se tiene:

$$I_{1,1} = \frac{4^1 I_{0,2} - I_{0,1}}{4^1 - 1}$$

Si este valor extrapolado ($I_{1,1}$) es igual, en un determinado número de dígitos, al último valor del nivel anterior ($I_{0,2}$), entonces el proceso concluye, siendo el valor de la integral el valor extrapolado ($I_{1,1}$). Caso contrario se vuelve a aplicar la regla del trapecio para calcular el valor de la integral ($I_{0,3}$), pero con un número de segmentos igual al doble del número de segmentos del cálculo anterior ($I_{0,2}$). Como en el último cálculo se emplearon 2 segmentos, en este nuevo cálculo se emplean 4.

Con el nuevo valor de la integral y el penúltimo valor de este nivel ($I_{0,2}$) se realiza otra extrapolación:

$$I_{1,2} = \frac{4^1 I_{0,3} - I_{0,2}}{4^1 - 1}$$

Ahora contamos con dos valores en el nivel de extrapolación 1 ($I_{1,1}$ e $I_{1,2}$), por lo que hacer una extrapolación de segundo nivel:

$$I_{2,1} = \frac{4^2 I_{1,2} - I_{1,1}}{4^2 - 1}$$

Se comprueba entonces si este valor es igual, en un determinado número de dígitos, al último valor del nivel anterior ($I_{1,2}$) y de ser así el proceso concluye, siendo el valor de la integral el último valor extrapolado ($I_{2,1}$).

Caso contrario se vuelve a calcular un nuevo valor de la integral con la regla del Trapecio ($I_{0,4}$), tomando un número de segmentos igual al doble del número de segmentos del cálculo anterior. Como en el cálculo anterior se emplearon 4 segmentos, ahora se emplean 8.

Con el nuevo valor de la integral y el penúltimo valor de este nivel ($I_{0,3}$) se realiza una nueva extrapolación:

$$I_{1,3} = \frac{4^1 I_{0,4} - I_{0,3}}{4^1 - 1}$$

Ahora en el nivel 1 tenemos tres valores, entonces con los dos últimos valores de este nivel ($I_{1,2}$ e $I_{1,3}$) realizamos una nueva extrapolación:

$$I_{2,2} = \frac{4^2 I_{1,3} - I_{1,2}}{4^2 - 1}$$

Con lo que tenemos dos valores en el nivel 2 ($I_{2,1}$ e $I_{2,2}$). Entonces podemos hacer una extrapolación de tercer nivel:

$$I_{3,1} = \frac{4^3 I_{2,2} - I_{2,1}}{4^3 - 1}$$

Ahora comparamos este valor con el último valor del nivel anterior ($I_{2,2}$) y si son iguales en un determinado número de dígitos el proceso concluye, siendo el resultado el último valor extrapolado, caso contrario se vuelve a repetir el proceso antes descrito, calculando la integral con un número de segmentos igual al doble del número de segmentos del cálculo anterior.

Puesto que cada nueva aplicación del método requiere un número de segmentos igual al doble de la iteración anterior, el método de Romberg no puede ser aplicado directamente a datos tabulados, sin embargo y al igual que en los métodos del Trapecio y Simpson, el método de Romberg puede ser empleado para integrara datos tabulados conjuntamente un método de interpolación.

En este método sólo se utilizan los dos últimos valores de cualquiera de los niveles de extrapolación, porque los valores anteriores, una vez utilizados, no vuelven a ser empleados. En consecuencia sólo es necesario guardar los dos últimos valores de todos los niveles de extrapolación.

El procedimiento que se sigue en el método de Romberg puede ser resumido de la siguiente manera:

- a) Se calcula el incremento inicial para el método del trapecio: $h = l_s - l_i$.
- b) Se inicializan dos contadores: "m" en 1 para el número de divisiones y "n" en 1 para el nivel de extrapolación.
- c) Se calcula el primer valor de la integral (una división) por el método del trapecio: $R_1 = (h/2) * (f(l_i) + f(l_s))$.
- d) Se inicializa el valor de "x" en " $l_i + h/2$ " y se calcula un nuevo valor de la integral por el método del Trapecio (modificado para aprovechar los valores calculados en la iteración previa):

$$S_1 = \frac{\left(R_1 + h \sum_{i=1}^m f(x + (i-1)h) \right)}{2} \quad (20)$$

- e) Se calculan los valores de extrapolación con la ecuación de Richardson (modificada para trabajar con los vectores "R" y "S"):

$$S_{i+1} = \frac{4^i S_i - R_i}{4^i - 1} \quad \{i=1 \rightarrow n\} \quad (21)$$

- f) Se compara S_{n+1} con S_n y si son aproximadamente iguales el proceso concluye, siendo la respuesta S_{n+1} , caso contrario se realizan los siguientes cambios en las variables: $h=h/2$, $m=2*m$, $n=n+1$, $R=S$ y se repite el proceso desde el paso (d).

Como no se sabe de antemano el número de veces que debe repetirse el procedimiento, no se sabe tampoco el número de elementos para los vectores "R" y "S". Sin embargo muy rara vez se requieren más de 10 iteraciones, por lo que es seguro fijar su dimensión en 20 elementos.

25.3.1. Ejemplos

25.3.1.1. Ejemplo 1

Como primer ejemplo volveremos a calcular el valor de la siguiente integral con 9 dígitos de precisión:

$$\int_{5.1}^{8.4} (3-6x^2+1.1x^{3.1})dx$$

Programamos la función e inicializamos variables siguiendo el procedimiento antes descrito:

```
f[x_]:=3-6*x^2+1.1*x^3.1;
R=ConstantArray[0,20];S=R;li=5.1; ls=8.4; h=ls-li;m=1;n=1;
Calculamos el valor de "R1":
R[[1]]=h/2*(f[li]+f[ls])
668.111
```

Inicializamos "x" y calculamos el valor de "S₁":

```
x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2
563.582
```

Calculamos (y mostramos) los valores de extrapolación:

```
Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];S
{563.582,528.738,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

Como podemos ver los dos últimos valores calculados son iguales sólo en el primer dígito (y requerimos que sean iguales en los primeros 9 dígitos), por lo que debemos repetir el proceso, realizando previamente los respectivos cambios en las variables:

```
h=h/2;m=2*m;n=n+1;R=S;
```

Calculamos el nuevo valor de "S₁":

```
x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2
537.436
```

Calculamos (y mostramos) los valores de extrapolación:

```
Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];S
{537.436,528.72,528.719,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

Ahora los dos últimos valores calculados son iguales en 4 dígitos, pero todavía no en 9, por lo que debemos repetir el proceso:

```
h=h/2;m=2*m;n=n+1;R=S;
x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2;
Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];S
```

```
{530.898,528.719,528.719,528.719,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
```

Ahora los dos últimos valores calculados son iguales en los 6 dígitos, que muestra Mathematica, para ver si son iguales en los 9 dígitos requeridos, empleamos la función "InputForm":

```
%//InputForm
{530.8982229992894, 528.7191030770164, 528.7190270412317,
 528.7190268545451, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
```

Vemos entonces que los dos últimos valores son iguales en los primeros 8 dígitos (no en los 9 requeridos), por lo que debemos repetir el proceso una vez más:

```
h=h/2;m=2*m;n=n+1;R=S;
x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2;
Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];S//InputForm
{529.2638294567184, 528.7190316091948, 528.7190268446733,
 528.7190268415533, 528.7190268415023, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
0, 0, 0, 0}
```

Ahora los dos últimos valores son iguales en los primeros 13 dígitos, por lo que podemos concluir el proceso, siendo entonces el valor de la integral: 528.7190268415023.

Por supuesto que en lugar de repetir el proceso manualmente "n" veces es más eficiente emplear una estructura iterativa tal como se muestra en el siguiente módulo donde se vuelve a resolver el problema (con una precisión de 12 dígitos):

```
Module[{f,R,S,li=5.1,ls=8.4,m=1,n=1,h,x},
  f[x_]:=3-6*x^2+1.1*x^3.1;
  R=ConstantArray[0,20];S=R; h=ls-li;
  R[[1]]=h/2*(f[li]+f[ls]);
  While[True,
    x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2;
    Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];
    If[Abs[S[[n]]/S[[n+1]]-1]<10^-12,Break[]];
    h=h/2;m=2*m;n=n+1;R=S;
  ];
  Print["Valor de la integral: ",S[[n+1]]//InputForm]
]
```

Ejecutando el módulo se obtiene, por supuesto, el mismo resultado:

Valor de la integral: 528.7190268415023

25.3.1.2. Ejemplo 2

Como segundo ejemplo volveremos a resolver la integral:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

x	2.0	2.5	2.7	3.0	3.3	3.8	4.2	4.5	5.1	5.7	6.0	6.4	6.9	7.2	7.5
y	4.800	6.625	7.509	9.000	10.689	13.944	16.944	19.425	24.981	31.329	34.800	39.736	46.401	50.664	55.125

Tal como sucede con los métodos del Trapecio y Simpson en su forma analítica, para integrar datos tabulados se crea una función de interpolación. En este caso se creará la función de interpolación con "InterpolatingPolynomial". El módulo que resuelve el problema (con 12 dígitos de precisión) es el siguiente:

```
Module[{vx,vy,puntos,f,li=2.5,ls=7.2,m=1,n=1,R,S,h,x},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,
    34.800,39.736,46.401,50.664,55.125};
  puntos=Transpose[{vx,vy}];
  f[x_]=InterpolatingPolynomial[puntos,x];
  R=ConstantArray[0,20];S=R; h=ls-li;
  R[[1]]=h/2*(f[li]+f[ls]);
  While[True,
    x=li+h/2;S[[1]]=(R[[1]]+h*Sum[f[x+(i-1)*h],{i,m}])/2;
    Do[S[[i+1]]=(4^i*S[[i]]-R[[i]])/(4^i-1),{i,n}];
    If[Abs[S[[n]]/S[[n+1]]-1]<10^-12,Break[]];
    h=h/2;m=2*m;n=n+1;R=S;
  ];
  Print["Valor de la integral: ",S[[n+1]]//InputForm]
]
```

Ejecutando el módulo se obtiene:

Valor de la integral: 115.59493333333333

Que una vez más es el resultado exacto.

25.4. NIntegrate

Para encontrar el valor numérico de ecuaciones integrales, Mathematica cuenta con "NIntegrate" (e "Integrate" para funciones simbólicas). "NIntegrate" es una función muy completa que tiene varias opciones y reglas adicionales. En su forma básica la sintaxis de esta función es la siguiente:

$$NIntegrate[\text{función}, \{\text{variable}, li, ls\}]$$

Donde "función" es la función a integrar, "variable" es la variable independiente, "li" el límite inferior y "ls" el límite superior. "NIntegrate" permite resolver también integrales múltiples, en cuyo caso la sintaxis es la siguiente:

$$NIntegrate[\text{función}, \{\text{variable 1}, li, ls\}, \{\text{variable 2}, li, ls\}, \dots]$$

Donde "variable 1", "variable 2", etc., son los nombres de las variables independientes.

25.4.1. Ejemplos

25.4.1.1. Ejemplo 1

Como primer ejemplo volveremos a calcular el valor de la siguiente integral:

$$\int_{5.1}^{8.4} (3 - 6x^2 + 1.1x^{3.1}) dx$$

El módulo que resuelve el problema es el siguiente:

```
Module[{f,vi},
  f[x_]:=3-6*x^2+1.1*x^3.1;
  vi=NIntegrate[f[x],{x,5.1,8.4}];
  Print["Valor de la integral: ",vi//InputForm]
]
```

Ejecutando el módulo se obtiene:

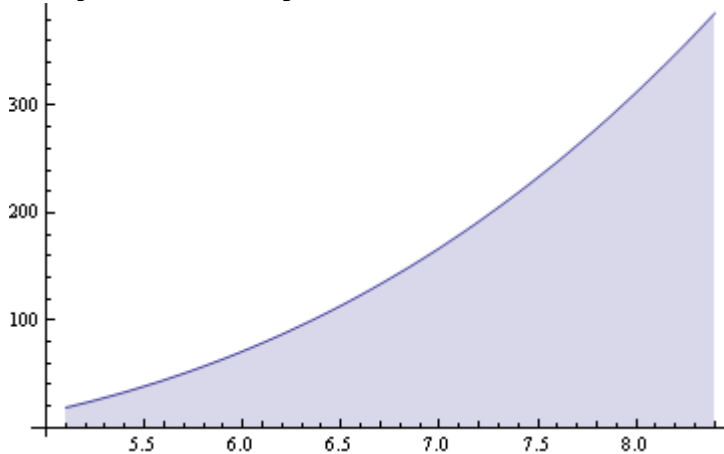
Valor de la integral: 528.7190268414938

Podemos visualizar gráficamente la integral podemos graficar la función con la opción "Filling->Bottom", que rellena el área bajo la curva (el área que es equivalente a la integral de la función). El anterior módulo con esta instrucción añadida es:

```
Module[{f,vi},
  f[x_]:=3-6*x^2+1.1*x^3.1;
  Print["Integral (área bajo la curva): "];
  Print[Plot[f[x],{x,5.1,8.4},AxesOrigin->{5,0},Filling->Bottom]];
  vi=NIntegrate[f[x],{x,5.1,8.4}];
  Print["Valor de la integral: ",vi//InputForm]
]
```

Ejecutando el módulo se obtiene:

Integral (área bajo la curva):



Valor de la integral: 528.7190268414938

25.4.1.2. Ejemplo 2

Como segundo ejemplo volveremos a resolver la integral:

$$\int_{2.5}^{7.2} f(x)dx = \int_{2.5}^{7.2} ydx$$

x	2.0	2.5	2.7	3.0	3.3	3.8	4.2	4.5	5.1	5.7	6.0	6.4	6.9	7.2	7.5
y	4.800	6.625	7.509	9.000	10.689	13.944	16.944	19.425	24.981	31.329	34.800	39.736	46.401	50.664	55.125

Al igual que con los otros métodos estudiados, para integrar datos tabulados debemos crear una función de interpolación. En este ejemplo crearemos la función de interpolación con "Interpolation":

```
Module[{vx,vy,puntos,f,li=2.5,ls=7.2,vi},
  vx={2.0,2.5,2.7,3.0,3.3,3.8,4.2,4.5,5.1,5.7,6.0,6.4,6.9,7.2,7.5};
  vy={4.800,6.625,7.509,9.00,10.689,13.944,16.944,19.425,24.981,31.329,
    34.800,39.736,46.401,50.664,55.125};
  puntos=Transpose[{vx,vy}];
  f=Interpolation[puntos];
  vi=NIntegrate[f[x],{x,li,ls}];
  Print["Valor de la integral: ",vi]
]
```

Ejecutando el módulo se obtiene:

Valor de la integral: 115.595

Que es el mismo resultado obtenido con Simpson y Romberg.

25.5. Ejercicios

1. Elabore un módulo que calcule el valor de la siguiente integral empleando el método del trapecio en su forma analítica.

$$\int_0^{2\pi} \frac{x dx}{1 + \cos^2 x}$$

2. Repita el ejercicio 1 empleando el método del trapecio en su forma tabular generando 30 puntos.
3. Repita el ejercicio 1 empleando el método de Romberg.
4. Repita el ejercicio 1 empleando el NIntegrate.
5. Elabore un módulo que calcule el valor de la siguiente integral empleando el método del trapecio en su forma tabular:

$$\int_2^{42} f(x) dx$$

X	1	2	5	7	10	13	15	18	22	26	30	35	42
Y	0.3125	0.3333	0.3775	0.3901	0.4005	0.4065	0.4093	0.4124	0.4152	0.4172	0.4187	0.4200	0.4214

6. Repita el ejercicio 5 empleando el método de Simpson en su forma tabular.
7. Repita el ejercicio 5 empleando el método de Romberg y el método de interpolación segmentaria cúbica.
8. Repita el ejercicio 5 empleando NIntegrate e InterpolatingPolynomial.
9. Elabore un módulo que calcule el valor de la siguiente integral empleando el método del Trapecio en su forma tabular y el método de interpolación lineal para los límites:

$$\int_{0.1}^{0.9} f(x) dx$$

x	0	0.2	0.4	0.6	0.8	1.0
y	0	0.199	0.389	0.565	0.717	0.841

10. Repita el ejercicio 9 empleando el método de Simpson en su forma tabular e Interpolation.
11. Con los datos del ejercicio 9 calcule el valor de la siguiente integral empleando el método de Simpson en su forma tabular y el método de interpolación de Newton.

$$\int_{0.24}^{0.98} f(x) dx$$

12. Repita el ejercicio 11 empleando el método del Trapecio e InterpolatingPolynomial.
13. Elabore un módulo que calcule el valor de la siguiente integral empleando el método del Trapecio en su forma analítica (con 200 segmentos). El módulo debe mostrar una gráfica de la integral (área bajo la curva).

$$\int_0^{12.4} \frac{x dx}{\sqrt{x^2 + 2x + 5}}$$

14. Repita el ejercicio 13 empleando el método de Simpson en su forma analítica (con 50 segmentos).
15. Repita el ejercicio 13 empleando el método de Romberg (con una precisión de 14 dígitos).
16. Repita el ejercicio 13 empleando NIntegrate.

26. ECUACIONES DIFERENCIALES ORDINARIAS

Con frecuencia en la solución de problemas en el campo de la ingeniería es necesario resolver una o más ecuaciones diferenciales, tanto ordinarias como entre derivadas parciales.

En este capítulo estudiaremos algunos de los métodos que nos permiten resolver ecuaciones y sistemas de ecuaciones diferenciales ordinarias.

Una ecuación diferencial ordinaria de enésimo grado puede ser representada en la siguiente forma:

$$y^n = f(x, y, y', y'', \dots, y^{n-1}) \quad (1)$$

Donde "x" es la variable independiente, "y" la variable dependiente y y' , y'' a y^{n-1} , son las derivadas: $y' = dy/dx$; $y'' = d^2y/dx^2$; $y^{n-1} = d^{n-1}y/dx^{n-1}$, etc.

Las ecuaciones diferenciales surgen al momento de modelar matemáticamente la solución de problemas en el campo de la ingeniería y deben ser resueltas porque es necesario calcular valores de la variable dependiente "y" para valores de la variable independiente "x". Este cálculo no es directo porque se desconoce la forma que tiene la función $y = f(x)$. En realidad es esta función la que constituye la solución de la ecuación diferencial y existen métodos analíticos que permiten encontrar algunas de dichas formas (métodos que han sido estudiados en la materia de ecuaciones diferenciales). Sin embargo, la mayoría de las ecuaciones diferenciales que se forman en la solución de problemas prácticos son demasiado complejas como para resolverlas analíticamente, esto sumado a lo tedioso y moroso del proceso analítico ha hecho que la solución de la mayoría de las ecuaciones diferenciales en el campo de la ingeniería, sea llevada a cabo casi exclusivamente con métodos numéricos.

Para ilustrar como una función puede dar lugar a un número infinito de ecuaciones diferenciales tomemos por ejemplo la siguiente función:

$$y = f(x) = 3x^3 + 2x^2 + 5x \quad (2)$$

Las derivadas primera, segunda y tercera de esta función son:

$$y' = 9x^2 + 4x + 5 \quad (3)$$

$$y'' = 18x + 4 \quad (4)$$

$$y''' = 18 \quad (5)$$

Con estas derivadas y la función original podemos generar por ejemplo las siguientes ecuaciones diferenciales:

$$2xy' + 3y'' = 2x(9x^2 + 4x) + 3(18x + 4) = 18x^3 + 8x^2 + 10x + 54x + 12 \quad (6)$$

$$2xy' + 3y'' - 18x^3 - 64x - 12 = 0$$

$$3yy''' - 2y'' = 3y(18) - 2(18x + 4) = 54y - 36x - 8 \quad (7)$$

$$3yy''' - 2y'' - 54y + 36x + 8 = 0$$

La solución de cualquiera de estas dos ecuaciones diferenciales es la ecuación (2). De manera similar podemos generar un número infinito de ecuaciones diferenciales y la solución de todas ellas seguiría siendo la ecuación (2).

En un caso real no se conoce la función con la cual hemos generado estas ecuaciones diferenciales (la ecuación 2), sino sólo las ecuaciones diferen-

ciales (ecuaciones 6 y 7) y con las mismas se deben calcular valores de la variable dependiente "y" para valores de la variable independiente "x".

El procedimiento antes ilustrado es útil para deducir ecuaciones diferenciales y con ellas comprobar la exactitud de un determinado método.

26.1. Problemas del valor inicial y problemas del valor límite

En general al resolver ecuaciones diferenciales ordinarias se presentan dos tipos de problemas:

Problemas del valor inicial: cuando se conoce el valor de la variable dependiente y de todas sus derivadas, para un valor *inicial* dado de la variable independiente.

Problemas del valor límite: Cuando algunos valores de la variable dependiente y de sus derivadas se conocen para el valor inicial y otros para el valor final de la variable independiente.

Los métodos que estudiemos en este capítulo permiten resolver problemas del valor inicial. Estos métodos combinados con otros métodos iterativos (como el de Newton - Raphson) permiten resolver también los problemas del valor límite, no obstante nosotros no estudiaremos este tipo de problemas.

De los muchos métodos disponibles para resolver los problemas del valor inicial estudiaremos los métodos de *Euler* y *Runge-Kutta*. El primero porque dada su sencillez permite comprender el procedimiento general que se sigue en la resolución de ecuaciones diferenciales y el segundo porque es uno de los métodos más utilizados en la práctica. Veremos además como se resuelven ecuaciones diferenciales ordinarias empleando la función *NDSolve* de Mathematica.

26.2. NDSolve

Para resolver ecuaciones diferenciales Mathematica cuenta con la función "NDSolve". "NDSolve" es una función muy completa que no sólo permite resolver ecuaciones diferenciales ordinarias con valores iniciales (como los métodos que hemos estudiado), sino también ecuaciones diferenciales ordinarias con valores límites conocidos (problemas del valor límite) e inclusive ecuaciones diferenciales entre derivadas parciales.

Con "NDSolve" se pueden resolver ecuaciones diferenciales de primer orden, de enésimo orden, sistemas de ecuaciones diferenciales de primer orden o de cualquier orden, e inclusive sistemas de ecuaciones diferenciales combinadas con ecuaciones algebraicas.

La sintaxis básica de esta función es la siguiente:

NDSolve[ecuaciones, var. dep., var. indep.]

Donde "ecuaciones" es una lista con la ecuación diferencial a resolver (o las ecuaciones a resolver) y los valores conocidos. "var. dep." es la variable dependiente o una lista con las variables dependientes. "var. indep." es el nombre de la variable independiente y los límites de integración.

Las ecuaciones se escriben como igualdades (con ==) y en todas ellas se debe especificar el nombre de la variable independiente escribiéndola entre corchetes, así por ejemplo: "y[x]", "z[x]", "v[x]", hace referencia a las variables dependientes "y", "z" y "v", siendo la variable independiente "x".

"NDSolve" devuelve él o los resultados como una matriz de funciones de interpolación en forma de reglas, empleando los nombres de las variables dependientes para cada una de ellas.

26.2.1. Ejemplos

26.2.1.1. Ejemplo 1

Como primer ejemplo resolveremos y graficaremos la siguiente ecuación diferencial:

$$3y' + 4xy - 8x^3 + 4x = 0 \quad \begin{cases} y = 4 \text{ para } x = 2 \\ y = ? \text{ para } x = 2.5, 3.1, 4.7, 5.3, 5.8, 6 \end{cases}$$

En este caso la variable dependiente es "y", la variable independiente "x" y el valor conocido de "y" es "y[2]=4", pues se sabe que para "x=2" el valor de "y" es 4:

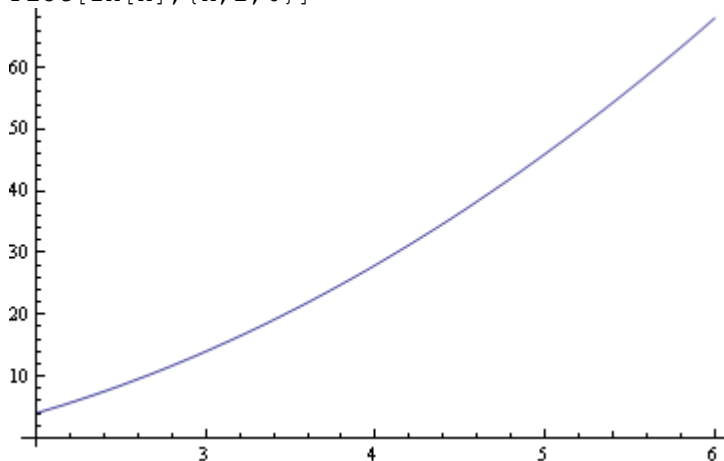
```
r=NDSolve[{3*y'[x]+4*x*y[x]-8*x^3+4*x==0,y[2]==4},y,{x,2,6}]
{{y->InterpolatingFunction[{{2.,6.}},<>]}}
```

Como se puede observar el resultado está en una matriz, en forma de una regla para la variable dependiente "y". Por comodidad y claridad extraemos la función de interpolación de este resultado:

```
fx=y/.r[[1]]
InterpolatingFunction[{{2.,6.}},<>]
```

Ahora con "fx" podemos graficar la función:

```
Plot[fx[x],{x,2,6}]
```



Y calcular los valores de "y" para los valores conocidos de "x":

```
vcx={2.5,3.1,4.7,5.3,5.8,6.0};
Do[Print["Para      x=      ",vcx[[i]],"      el      valor      de      y      es: ",fx[vcx[[i]]],{i,Length[vcx]}]]
Para x=  2.5  el valor de y es:  8.5
Para x=  3.1  el valor de y es: 15.22
Para x=  4.7  el valor de y es: 40.18
Para x=  5.3  el valor de y es: 52.18
Para x=  5.8  el valor de y es: 63.28
Para x=  6.   el valor de y es: 68.
```

26.2.1.2. Ejemplo 2

Como segundo ejemplo resolveremos la siguiente ecuación diferencial:

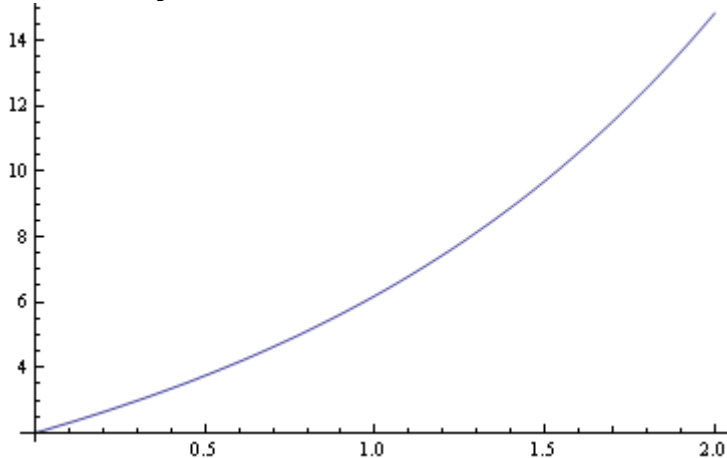
$$y' = 4e^{0.8x} - 0.5y \quad \begin{cases} y = 2 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.3, 0.7, 0.9, 1.2, 1.5, 1.7, 2.0 \end{cases}$$

Prácticamente la única diferencia con relación al anterior ejemplo es que en este caso resolveremos el problema en un módulo:

```
Module[{x0=0,y0=2.0,xn=2.0,r,fx,y,
  vcx={0.3,0.7,0.9,1.2,1.5,1.7,2.0}},
  r=NDSolve[{y'[x]==4*Exp[0.8*x]-0.5*y[x],y[x0]==y0},y,{x,x0,xn}];
  fx=y/.r[[1]];
  Print["Solución gráfica:"];
  Print[Plot[fx[x],{x,x0,xn}]];
  Do[Print["Para x = ",vcx[[i]]," el valor de y es: ",
    fx[vcx[[i]]]],{i,Length[vcx]}]
]
```

Ejecutando el módulo se obtiene:

Solución gráfica:



```
Para x = 0.3 el valor de y es: 2.98462
Para x = 0.7 el valor de y es: 4.62779
Para x = 0.9 el valor de y es: 5.63466
Para x = 1.2 el valor de y es: 7.44496
Para x = 1.5 el valor de y es: 9.70704
Para x = 1.7 el valor de y es: 11.528
Para x = 2. el valor de y es: 14.8439
```

Resultados que concuerdan exactamente con los obtenidos con Runge-Kutta.

26.2.1.3. Ejemplo 3

Como tercer ejemplo volveremos a resolver y a graficar la solución de la ecuación diferencial de tercer orden:

$$3y''' - 3xy'' + y' + 1.8711x^{-1.9} + 14.49x^{1.1} - 5 = 0$$

$$\begin{cases} y = 4, y' = 11.3, y'' = 6.93, y''' = 0.693 \text{ para } x = 1 \\ y = ?, y' = ?, y'' = ?, y''' = ? \text{ para } x = 1.3, 1.5, 1.8, 2.0, 2.4, 2.9, 3.3, 3.7, 3.8, 4.0 \end{cases}$$

Que como sabemos tiene la solución: $y = 3x^{2.1} + 5x - 4$.

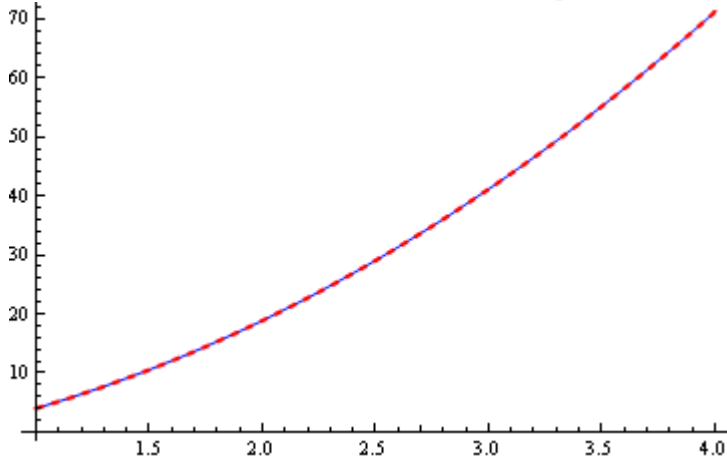
El módulo que resuelve el problema es el siguiente:

```
Module[{x0=1,xn=4.0,y0=4,y1=11.3,y2=6.93,y3=0.693,r,fx,fc,
  y,vcx={1.3,1.5,1.8,2.0,2.4,2.9,3.3,3.7,3.8,4.0}},
  r=NDSolve[{3*y'''[x]-3*x*y''[x]+y'[x]+1.8711*x^-1.9+14.49*x^1.1-5==0,
    y[x0]==y0,y'[x0]==y1,y''[x0]==y2,y'''[x0]==y3},y,{x,x0,xn}];
  fx=y/.r[[1]];
  fc[x_]:=3*x^2.1+5*x-4;
```

```
Print["Gráfica de la solución encontrada y conocida"];
Print[Plot[{fx[x],fc[x]},{x,x0,xn},PlotStyle->{Blue,{Red,Dashed,Thick}}]];
Do[Print["Para x = ",vcx[[i]]," el valor de y es: ",
  fx[vcx[[i]]],{i,Length[vcx]}]
]
```

Ejecutando el módulo se obtiene:

Gráfica de la solución encontrada y conocida



```
Para x = 1.3 el valor de y es: 7.70478
Para x = 1.5 el valor de y es: 10.5293
Para x = 1.8 el valor de y es: 15.3085
Para x = 2. el valor de y es: 18.8613
Para x = 2.4 el valor de y es: 26.861
Para x = 2.9 el valor de y es: 38.5645
Para x = 3.3 el valor de y es: 49.3129
Para x = 3.7 el valor de y es: 61.3107
Para x = 3.8 el valor de y es: 64.507
Para x = 4. el valor de y es: 71.1375
```

26.2.1.4. Ejemplo 4

Como cuarto ejemplo volveremos a resolver la ecuación diferencial:

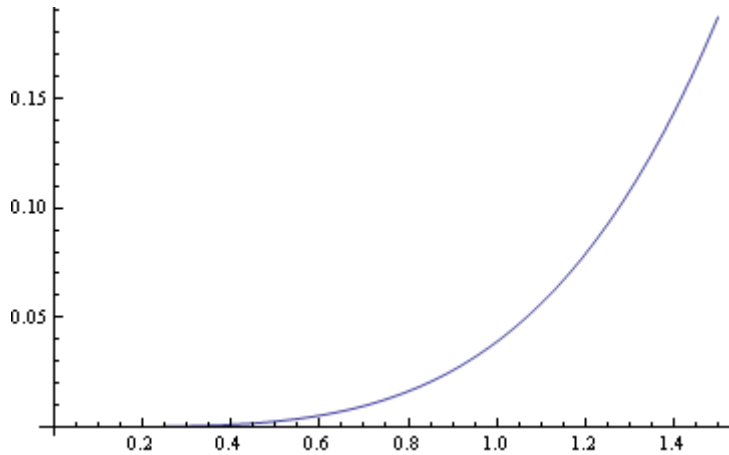
$$y''' + ty'' - ty' - 2y = t \quad \begin{cases} y = 0; y' = 0; y'' = 0 \text{ para } t = 0 \\ y = ? \text{ para } t = 0.2, 0.3, 0.4, 0.8, 1.1, 1.4, 1.5 \end{cases}$$

El módulo que resuelve el problema es el siguiente:

```
Module[{t0=0,tn=1.5,y0=0,y1=0,y2=0,r,ft,y,
  vct={0.2,0.3,0.4,0.8,1.1,1.4,1.5}},
r=NDSolve[{y'''[t]+t*y''[t]-t*y'[t]-2*y[t]==t,
  y[t0]==y0,y'[t0]==y1,y''[t0]==y2},y,{t,t0,tn}];
ft=y/.r[[1]];
Print["Gráfica de la solución"];
Print[Plot[ft[t],{t,t0,tn}]];
Do[Print["Para x = ",vct[[i]]," el valor de y es: ",
  ft[vct[[i]]],{i,Length[vct]}]
]
```

Ejecutando el módulo se obtiene:

Gráfica de la solución



Para x = 0.2 el valor de y es: 0.0000664152
 Para x = 0.3 el valor de y es: 0.000334743
 Para x = 0.4 el valor de y es: 0.00105175
 Para x = 0.8 el valor de y es: 0.0162641
 Para x = 1.1 el valor de y es: 0.0563542
 Para x = 1.4 el valor de y es: 0.143458
 Para x = 1.5 el valor de y es: 0.187332

26.2.1.5. Ejemplo 5

Como quinto ejemplo volveremos a resolver y a graficar las soluciones del siguiente sistema de ecuaciones diferenciales:

$$u' - v + 2w + t^2 - 2t^{1.5} - 1.2t^{0.2} + 10 = 0$$

$$v' + 5w - 5t^{1.5} - 2t + 35 = 0$$

$$w' - 2u + v - t^2 + 2t^{1.2} - 1.5t^{0.5} + 10 = 0$$

$$\begin{cases} u = 3, v = -4, w = -7 \text{ para } t = 0 \end{cases}$$

$$\begin{cases} u = ?, v = ?, w = ? \text{ para } t = 0.05, 0.1, 0.14, 0.2, 0.23, 0.26, 0.30, 0.33, 0.35 \end{cases}$$

Donde u , v y w son las variables dependientes y t la variable independiente. Las soluciones analíticas de este sistema son: $u = t^{1.2} + 3$; $v = t^2 - 4$ y $w = t^{1.5} - 7$.

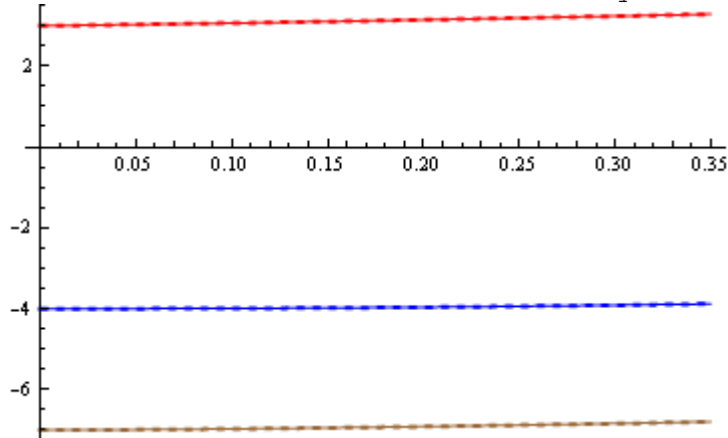
El módulo que resuelve el problema es el siguiente:

```
Module[{t0=0,tn=0.35,u0=3,v0=-4,w0=-7,r,fu,fv,fw,u,v,w,g1,g2,
  fcu,fcv,fcw,vct={0.05,0.1,0.14,0.2,0.23,0.26,0.30,0.33,0.35}},
r=NDSolve[{u'[t]-v[t]+2*w[t]+t^2-2*t^1.5-1.2*t^0.2+10==0,
  v'[t]+5 w[t]-5*t^1.5-2*t+35==0,
  w'[t]-2*u[t]+v[t]-t^2+2*t^1.2-1.5*t^0.5+10==0,
  u[t0]==u0,v[t0]==v0,w[t0]==w0}
,{u,v,w},{t,t0,tn}];
{fu,fv,fw}={u,v,w}/.r[[1]];
g1=Plot[{fu[t],fv[t],fw[t]},{t,t0,tn},PlotStyle->
{Red,Blue,Brown}];
fcu[t_]:=t^1.2+3;fcv[t_]:=t^2-4;fcw[t_]:=t^1.5-7;
g2=Plot[{fcu[t],fcv[t],fcw[t]},{t,t0,tn},PlotStyle->
{{Red,Dashed,Thick},{Blue,Dashed,Thick},{Brown,Dashed,Thick}}];
Print["Gráfica de las soluciones encontradas y conocidas"];
Print[Show[g1,g2]];
Do[Print["Para t= ",vct[[i]]];
Print["El valor de u es: ",fu[vct[[i]]]];
Print["El valor de v es: ",fv[vct[[i]]]];
```

```
Print["El valor de w es: ",fw[vct[[i]]],{i,Length[vct]}]
]
```

Ejecutando el módulo se obtiene:

Gráfica de las soluciones encontradas y conocidas



```
Para t= 0.05
El valor de u es: 3.02746
El valor de v es: -3.9975
El valor de w es: -6.98882
Para t= 0.1
El valor de u es: 3.0631
El valor de v es: -3.99
El valor de w es: -6.96838
Para t= 0.14
El valor de u es: 3.09448
El valor de v es: -3.9804
El valor de w es: -6.94762
Para t= 0.2
El valor de u es: 3.14496
El valor de v es: -3.96
El valor de w es: -6.91056
Para t= 0.23
El valor de u es: 3.17142
El valor de v es: -3.9471
El valor de w es: -6.8897
Para t= 0.26
El valor de u es: 3.19859
El valor de v es: -3.9324
El valor de w es: -6.86743
Para t= 0.3
El valor de u es: 3.2358
El valor de v es: -3.91
El valor de w es: -6.83568
Para t= 0.33
El valor de u es: 3.26437
El valor de v es: -3.8911
El valor de w es: -6.81043
Para t= 0.35
El valor de u es: 3.28371
El valor de v es: -3.8775
El valor de w es: -6.79294
```

26.2.1.6. Ejemplo 6

Como sexto ejemplo volveremos a resolver el siguiente sistema de ecuaciones diferenciales:

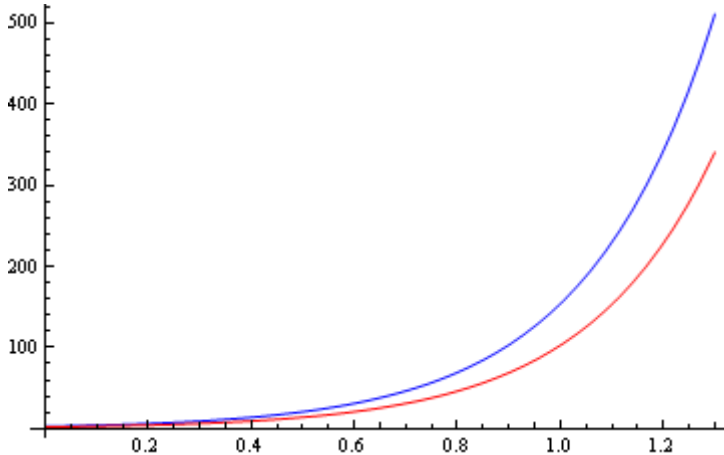
$$\begin{aligned}\frac{dx}{dt} &= x' = 2x + 3y \\ \frac{dy}{dt} &= y' = 2x + y \\ \begin{cases} x = 2.7; y = 2 \text{ para } t = 0 \\ x = ?; y = ? \text{ para } t = 0.2, 0.5, 0.7, 0.9, 1.2, 1.3 \end{cases}\end{aligned}$$

El módulo que resuelve el problema es el siguiente:

```
Module[{t0=0,tn=1.3,x0=2.7,y0=2,fx,fy,r,x,y,vct={0.2,0.5,0.7,0.9,1.2,1.3}},
  r=NDSolve[{x'[t]==2*x[t]+3*y[t],
    y'[t]==2*x[t]+y[t],
    x[t0]==x0,y[t0]==y0},
    {x,y},{t,t0,tn}];
  {fx,fy}={x,y}/.r[[1]];
  Print["Gráfica de las soluciones: "];
  Print[Plot[{fx[t],fy[t]},{t,t0,tn},PlotStyle->{Blue,Red}]];
  Do[Print["Para t= ",vct[[i]]];
    Print["El valor de x es: ",fx[vct[[i]]]];
    Print["El valor de y es: ",fy[vct[[i]]]],{i,Length[vct]}]
]
```

Ejecutando el módulo se obtiene:

Gráfica de las soluciones:



```
Para t= 0.2
El valor de x es: 6.17778
El valor de y es: 4.28226
Para t= 0.5
El valor de x es: 20.7644
El valor de y es: 13.9642
Para t= 0.7
El valor de x es: 46.3143
El valor de y es: 30.9755
Para t= 0.9
El valor de x es: 103.158
El valor de y es: 68.8534
Para t= 1.2
El valor de x es: 342.623
```

El valor de y es: 228.475
 Para t= 1.3
 El valor de x es: 511.155
 El valor de y es: 340.825

26.3. Ejercicios

1. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial:

$$y' = y(x^2 - 1) \quad \begin{cases} y = 1 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.3, 0.6, 0.8, 1.0, 1.3, 1.5, 1.7, 2 \end{cases}$$

2. Repita el ejercicio 1 empleando el método de Runge Kutta.
3. Repita el ejercicio 1 empleando NDSolve.
4. Elabore un programa que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial:

$$y' = \frac{5x}{y} - xy \quad \begin{cases} y = 2 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.1, 0.2, 0.3, 0.4, 0.5 \end{cases}$$

5. Repita el ejercicio 4 empleando el método de Runge Kutta.
6. Repita el ejercicio 4 empleando el NDSolve.
7. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial:

$$y' = x^3 + y^2 \quad \begin{cases} y = 0 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.2, 0.5, 0.7, 1.0, 1.2, 1.3, 1.4 \end{cases}$$

8. Repita el ejercicio 7 empleando el método de Runge Kutta.
9. Repita el ejercicio 7 empleando el NDSolve.
10. Elabore un programa que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial de primer orden:

$$y' = y^2 + t^2 \quad \begin{cases} y = 2 \text{ para } t = 1 \\ y = ? \text{ para } t = 2 \end{cases}$$

11. Repita el ejercicio 10 empleando Runge Kutta.
12. Repita el ejercicio 10 empleando NDSolve.
13. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial:

$$\frac{1}{2}q'' + 6q' + 50q = 24\text{sen}(10t) \quad \begin{cases} q = 0; q' = 0 \text{ para } t = 0 \\ q = ? \text{ para } t = 0.4, 0.6, 0.8, 1.2, 1.4, 1.6, 1.9, 2.1 \end{cases}$$

14. Repita el ejercicio 13 empleando Runge Kutta.
15. Repita el ejercicio 13 empleando NDSolve.
16. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique la siguiente ecuación diferencial:

$$x'' + 64x = 16\cos(8t) \quad \begin{cases} x = 0; x' = 0 \text{ para } t = 0 \\ x = ? \text{ para } t = 0.1, 0.2, 0.5, 0.6, 0.7, 0.8 \end{cases}$$

17. Repita el ejercicio 16 empleando Runge Kutta.

18. Repita el ejercicio 16 empleando NDSolve.
19. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique el siguiente sistema de ecuaciones diferenciales:

$$\frac{dx}{dt} = xy + t$$

$$\frac{dy}{dt} = x - t$$

$$x = 0; y = 1 \text{ para } t = 0$$

$$x = ?; y = ? \text{ para } t = 0.1, 0.2, 0.4, 0.6, 0.7, 0.8$$

20. Repita el ejercicio 19 empleando Runge Kutta.
21. Repita el ejercicio 19 empleando NDSolve.
22. Elabore un módulo que empleando el método de Euler e Interpolation, resuelva y grafique el siguiente sistema de ecuaciones diferenciales:

$$u' - 2v + w + t^2 - 2t^{1.5} + 1.2t^{0.2} + 7 = 0$$

$$v' + 3w - 4t^{1.5} - 2t + 25 = 0$$

$$w' - 3u + 2v - t^2 + t^{1.2} - 1.5t^{0.5} + 8 = 0$$

$$\begin{cases} u = 3, v = -4, w = -7 \text{ para } t = 0.3 \end{cases}$$

$$\begin{cases} u = ?, v = ?, w = ? \text{ para } t = 0.33, 0.35, 0.4, 0.42, 0.45, 0.47, 0.5 \end{cases}$$

23. Repita el ejercicio 22 empleando Runge Kutta.
24. Repita el ejercicio 22 empleando NDSolve.

27. EULER Y RUNGE KUTTA

27.1. Método de Euler

El método de Euler, como todos los métodos numéricos existentes sólo permite resolver ecuaciones diferenciales de primer orden, pues como demostraremos posteriormente, una ecuación diferencial de n ésimo orden puede ser transformada en un sistema de " n " ecuaciones diferenciales de primer orden.

Por esta razón estudiaremos en primer lugar la solución de ecuaciones diferenciales de primer orden.

27.1.1. Ecuaciones diferenciales de primer orden

Consideremos la siguiente ecuación diferencial de primer orden:

$$y' = f(x, y) \quad (1)$$

Para los problemas del valor inicial, se conoce el valor de la variable dependiente (y_0) para un valor inicial de la variable independiente (x_0).

El problema radica en calcular el valor de la variable dependiente (y_n) para un determinado valor de la variable independiente (x_n). Si se conoce la forma de la curva y' versus $f(x, y)$ la solución del problema se encuentra calculando el área bajo la curva comprendida entre el valor inicial (x_0) y el valor final (x_n), para lo cual se puede emplear uno de los métodos de integración estudiados en el anterior capítulo:

$$\int_{y_0}^{y_n} dy = y_n - y_0 = \int_{x_0}^{x_n} f(x, y) dx \Rightarrow y_n = \int_{x_0}^{x_n} f(x, y) dx + y_0 \quad (2)$$

Como en la práctica no se conoce la forma de la curva, lo que hacen los diferentes métodos es asumir una forma para la curva y emplearla para calcular el área bajo la curva (la integral). La forma de curva asumida es lo que diferencia a un método de otro.

Así en el método de Euler se divide el área comprendida entre x_0 y x_n en un número de segmentos igualmente espaciados y se asume que el área de cada uno de los segmentos es un rectángulo (ver Figura 10.1).

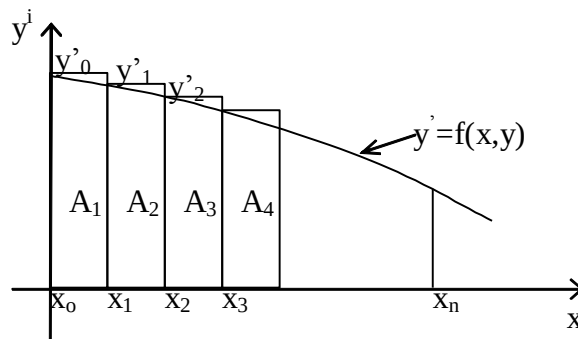


Figura 10.1. Cálculo del área con el método de Euler

Como el ancho de los segmentos es conocido ($h = (x_n - x_0)/n$) el área del primer segmento (A_1) se calcula multiplicando la altura del rectángulo (y'_0) por su ancho (h). La altura (y'_0) se calcula sustituyendo los valores iniciales (x_0, y_0) en la ecuación (8):

$$y'_0 = f(x_0, y_0) \quad (3)$$

Entonces el valor de la variable dependiente en el segundo punto (y_1), se calcula con:

$$\int_{y_0}^{y_1} dy = y_1 - y_0 \approx A_1 = y'_0 * h \Rightarrow y_1 = y_0 + y'_0 * h \quad (4)$$

El valor de la variable independiente en este punto (x_1) es $x_1 = x_0 + h$. Ahora con este nuevo punto y la ecuación (8), calculamos el valor de y'_1 :

$$y'_1 = f(x_1, y_1) \quad (5)$$

Entonces podemos calcular el área del segundo segmento (A_2) y con este el valor de y_2 :

$$\int_{y_1}^{y_2} dy = y_2 - y_1 \approx A_2 = y'_1 * h \Rightarrow y_2 = y_1 + y'_1 * h \quad (6)$$

Siendo el valor de $x_2 = x_1 + h$. Con este nuevo punto podemos calcular A_3 y con este valor y_3 prosiguiendo de esta manera hasta llegar al segmento A_n y consiguientemente el valor de y_n .

Entonces para cualquier punto conocido (x_i, y_i), podemos calcular los valores de un nuevo punto (x_{i+1}, y_{i+1}) con:

$$\begin{aligned} y'_i &= f(x_i, y_i) \\ y_{i+1} &= y_i + y'_i * h \\ x_{i+1} &= x_i + h \end{aligned} \quad (7)$$

Que son las ecuaciones del método de Euler. Una vez calculados los valores de " y_i " (y los correspondientes valores de " x_i "), se pueden emplear dichos valores para crear una función de interpolación (recurriendo a alguno de los métodos estudiados en el capítulo 8). Entonces dicha función constituye la solución de la ecuación diferencial y puede ser empleada de manera similar a la solución analítica, pues con la misma podemos calcular valores de " y " para valores conocidos de " x ".

Puesto que en el método de Euler se asumen áreas rectangulares para los segmentos se comete un error apreciable en los cálculos. Por esta razón, el método de Euler no se emplea normalmente en la solución de problemas reales, sin embargo, la lógica que se sigue en este método es esencialmente la misma que en otros métodos más elaborados.

Por ejemplo, si en lugar de asumir áreas rectangulares para los segmentos se asumen áreas trapezoidales, la exactitud del método de Euler mejora considerablemente. Cuando se lleva a cabo esta modificación, las ecuaciones resultantes se conocen con el nombre del *método de Euler modificado*.

27.1.2. Ejemplos: Ecuaciones diferenciales de primer orden

27.1.2.1. Ejemplo 1

Como primer ejemplo emplearemos el método de Euler para resolver la siguiente ecuación diferencial:

$$3y' + 4xy - 8x^3 + 4x = 0 \quad \begin{cases} y = 4 \text{ para } x = 2 \\ y = ? \text{ para } x = 2.5, 3.1, 4.7, 5.3, 5.8, 6 \end{cases}$$

Como se puede observar, en este caso se sabe que para " $x=2$ " el valor de " y " es 4, es decir se conocen los valores iniciales: $x_0=2$, $y_0=4$. Entonces guardamos primero los valores conocidos en variables:

```
x0=2.;y0=4.;xn=6.;
```

Para este ejemplo fijamos el número de segmentos "n" en 50 y con el mismo calculamos el valor de "h":

```
n=50;h=(xn-x0)/n;
```

Con este valor calculamos los valores de "y_i" y "x_i" (en los vectores "vx" y "vy") aplicando las ecuaciones 14:

```
vx=ConstantArray[0,{n+1}];vy=vx;vx[[1]]=x0;vy[[1]]=y0;
Do[vy[[i+1]]=vy[[i]]+f[vx[[i]],vy[[i]]]*h;vx[[i+1]]=vx[[i]]+h,{i,n}]
```

Con estos vectores creamos la tabla con los puntos:

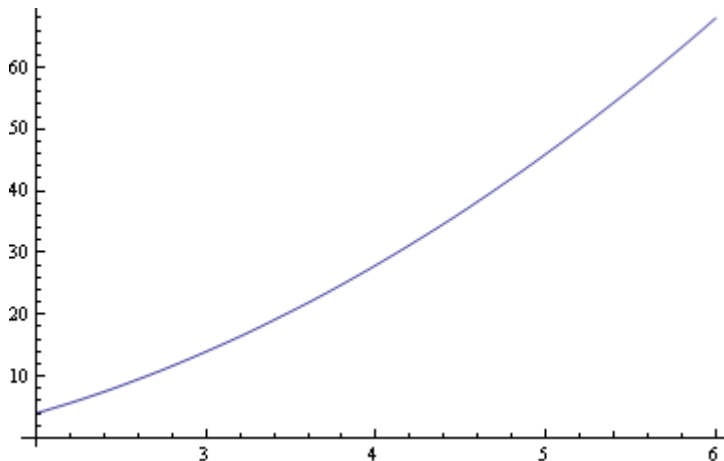
```
puntos=Transpose[{vx,vy}]
{{2.,4.},{2.08,4.64},{2.16,5.30844},{2.24,6.00488},{2.32,6.72893},{2.4,7.480
2},{2.48,8.25839},{2.56,9.06321},{2.64,9.89442},{2.72,10.7518},{2.8,11.6353}
,{2.88,12.5446},{2.96,13.4798},{3.04,14.4407},{3.12,15.4273},{3.2,16.4395},{
3.28,17.4773},{3.36,18.5407},{3.44,19.6297},{3.52,20.7443},{3.6,21.8844},{3.
68,23.0501},{3.76,24.2413},{3.84,25.4581},{3.92,26.7005},{4.,27.9684},{4.08,
29.2619},{4.16,30.5809},{4.24,31.9256},{4.32,33.2958},{4.4,34.6915},{4.48,36
.1129},{4.56,37.5598},{4.64,39.0323},{4.72,40.5304},{4.8,42.0541},{4.88,43.6
034},{4.96,45.1782},{5.04,46.7786},{5.12,48.4046},{5.2,50.0562},{5.28,51.733
4},{5.36,53.4362},{5.44,55.1645},{5.52,56.9185},{5.6,58.698},{5.68,60.5032},
{5.76,62.3339},{5.84,64.1902},{5.92,66.0721},{6.,67.9796}}
```

Con estos puntos creamos la función de interpolación y para este ejemplo emplearemos "Interpolation":

```
fx=Interpolation[puntos];
```

Esta función es la solución de la ecuación diferencial que podemos por ejemplo graficarla:

```
Plot[fx[x],{x,x0,xn}]
```



Y emplearla para calcular los valores de "y" requeridos:

```
vcx={2.5,3.1,4.7,5.3,5.8,6.};
Do[Print["Para x = ",vcx[[i]]," el valor de y es : ",fx[vcx[
i]]],{i,Length[vcx]}]
Para x = 2.5 el valor de y es : 8.45711
Para x = 3.1 el valor de y es : 15.1782
Para x = 4.7 el valor de y es : 40.1535
Para x = 5.3 el valor de y es : 52.1567
Para x = 5.8 el valor de y es : 63.2588
Para x = 6. el valor de y es : 67.9796
```

27.1.2.2. Ejemplo 2

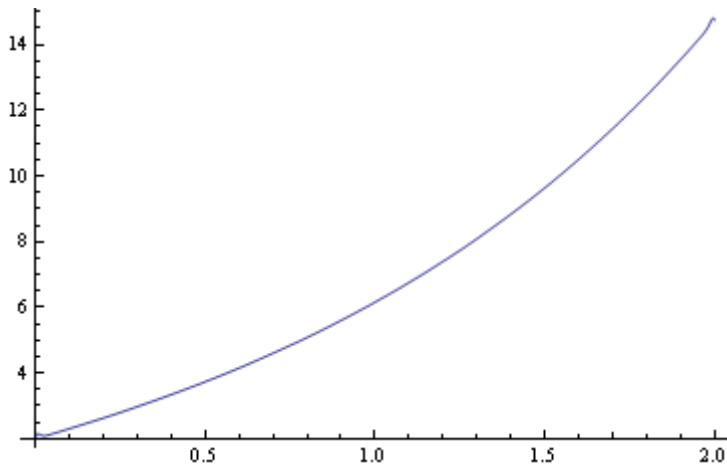
Como segundo ejemplo resolveremos la siguiente ecuación diferencial:

$$y' = 4e^{0.8x} - 0.5y \quad \begin{cases} y = 2 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.3, 0.7, 0.9, 1.2, 1.5, 1.7, 2.0 \end{cases}$$

Básicamente la única diferencia con el anterior ejemplo son los valores y la forma de la ecuación diferencial. En este caso resolveremos el problema en un módulo, fijando el número de segmentos en 60 y empleando el método de interpolación de Newton:

```
Module[{x0,y0,xn,n,h,vx,vy,fx,f,nd,b,fx,vcx={0.3,0.7,0.9,1.2,1.5,1.7,2.0}},
  x0=0;y0=2.0;xn=2.0;n=60;h=(xn-x0)/n;
  vx=ConstantArray[0,{n+1}];vy=vx;vx[[1]]=x0;vy[[1]]=y0;
  fxy[x_,y_]:=4*Exp[0.8*x]-0.5*y;
  Do[vy[[i+1]]=vy[[i]]+fxy[vx[[i]],vy[[i]]]*h;vx[[i+1]]=vx[[i]]+h,{i,n}];
  f=vy;nd=n+1;b=Range[nd];
  Do[b[[k]]=f[[1]];Do[f[[i]]=(f[[i+1]]-f[[i]])/(vx[[i+k]]-vx[[i]]),
    {i,nd-k},{k,nd-1}];
  b[[nd]]=f[[1]];
  fx[x_]:=Sum[b[[k]]*Product[(x-vx[[i]]),{i,k-1}},{k,nd}];
  Print[Plot[fx[x],{x,x0,xn}]];
  Do[Print["Para x = ",vcx[[i]]," el valor de y es: ",
    fx[vcx[[i]]]],{i,Length[vcx]}]
]
```

Ejecutando el módulo se obtiene:



```
Para x = 0.3 el valor de y es: 2.97539
Para x = 0.7 el valor de y es: 4.60355
Para x = 0.9 el valor de y es: 5.6014
Para x = 1.2 el valor de y es: 7.39576
Para x = 1.5 el valor de y es: 9.63823
Para x = 1.7 el valor de y es: 11.4436
Para x = 2.0 el valor de y es: 14.7318
```

27.1.3. Ecuaciones diferenciales de enésimo orden

Como se mencionó previamente, una ecuación de enésimo orden puede ser transformada en un sistema de ecuaciones diferenciales de primer orden. Para ilustrar el procedimiento consideremos la siguiente ecuación diferencial:

$$3x \frac{d^4 y}{dx^4} + (x-y) \frac{d^3 y}{dx^3} + 5 \frac{d^2 y}{dx^2} - \frac{dy}{dx} - \frac{x^2 - y^2}{xy} = 3xy''' + (x-y)y''' + 5y'' - y' - \frac{x^2 - y^2}{xy} \quad (8)$$

Esta ecuación de cuarto orden puede ser transformada en un sistema de 4 ecuaciones diferenciales de primer orden realizando los siguientes cambios de variable:

$$\begin{aligned} \frac{dy}{dx} &= \frac{dy_1}{dx} = y'_1 = y_1 \\ \frac{d^2 y}{dx^2} &= \frac{d}{dx} \left(\frac{dy}{dx} \right) = \frac{d}{dx} (y_1) = y'_2 = y_2 \\ \frac{d^3 y}{dx^3} &= \frac{d}{dx} \left(\frac{d^2 y}{dx^2} \right) = \frac{d}{dx} (y_2) = y'_3 = y_3 \\ \frac{d^4 y}{dx^4} &= \frac{d}{dx} \left(\frac{d^3 y}{dx^3} \right) = \frac{d}{dx} (y_3) = y'_4 = y_4 \end{aligned} \quad (9)$$

En consecuencia la ecuación (15) puede ser rescrita de la siguiente manera:

$$\begin{aligned} y'_1 &= y_1 \\ y'_2 &= y_2 \\ y'_3 &= y_3 \\ y'_4 &= f_4(x, y, y_1, y_2, y_3) = \frac{(y-x)y_3 - 5y_2 + y_1 + (x^2 - y^2)/xy}{3x} = y_4 \end{aligned} \quad (10)$$

Como estamos resolviendo los problemas del valor inicial, son datos los valores iniciales de y , y_2 , y_3 y y_4 para un valor inicial de " x ".

Las ecuaciones de Euler (ecuaciones 14) pueden ser aplicadas a cada una de las ecuaciones de este sistema tomando en cuenta las nuevas variables (ecuaciones 16 y 17) y empleando en todos los casos el mismo número de segmentos. Entonces aplicando las ecuaciones de Euler a un punto " j " cualquiera se tiene:

$$\begin{aligned} y_{4,j} &= f(x_j, y_j, y_{1,j}, y_{2,j}, y_{3,j}) = \frac{(y_j - x_j)y_{3,j} - 5y_{2,j} + y_{1,j} + (x_j^2 - y_j^2)/x_j y_j}{3x_j} \\ y_{j+1} &= y_j + y'_j * h = y_j + y_{1,j} * h \\ y_{1,j+1} &= y_{1,j} + y'_{1,j} * h = y_{1,j} + y_{2,j} * h \\ y_{2,j+1} &= y_{2,j} + y'_{2,j} * h = y_{2,j} + y_{3,j} * h \\ y_{3,j+1} &= y_{3,j} + y'_{3,j} * h = y_{3,j} + y_{4,j} * h \\ x_{j+1} &= x_j + h \end{aligned} \quad (11)$$

Como se puede observar, la única función que realmente necesita ser evaluada es la correspondiente a la derivada de enésimo grado (y_5 en el ejemplo), pues las otras funciones son simplemente los valores de las derivadas, las mismas que se conocen del punto anterior y en el primer cálculo son los valores iniciales.

Generalizando el procedimiento para una ecuación de enésimo orden, en un punto " j " cualquiera se tiene:

$$\left. \begin{aligned} y_{j+1} &= y_j + y'_j * h = y_j + y_{1,j} * h \\ y_{m,j} &= f(x_j, y_j, y_{1,j}, y_{2,j}, \dots, y_{m-1,j}) \\ y_{i,j+1} &= y_{i,j} + y'_{i,j} * h = y_{i,j} + y_{i+1,j} * h \\ x_{j+1} &= x_j + h \end{aligned} \right\} \begin{matrix} j=1 \rightarrow n \\ \\ \{i=1 \rightarrow m-1\} \end{matrix} \quad (12)$$

Donde "m" es el orden de la ecuación diferencial y las variables y_i son las derivadas de la ecuación:

$$\begin{aligned} \frac{d^i y}{dx^i} &= y_i \\ y'_i &= y_i \end{aligned} \quad (13)$$

Estos valores son sólo intermedios y necesitan ser guardados en vectores.

27.1.4. Ejemplos: Ecuaciones diferenciales de enésimo orden

27.1.4.1. Ejemplo 1

Como primer ejemplo resolveremos la siguiente ecuación diferencial de cuarto orden:

$$\begin{aligned} 3y'''' - 3xy'' + y' + 1.8711x^{-1.9} + 14.49x^{1.1} - 5 &= 0 \\ \begin{cases} y = 4, y' = 11.3, y'' = 6.93, y''' = 0.693 \text{ para } x = 1 \\ y = ?, y' = ?, y'' = ?, y''' = ? \text{ para } x = 1.3, 1.5, 1.8, 2.0, 2.4, 2.9, 3.3, 3.7, 3.8, 4.0 \end{cases} \end{aligned}$$

En este caso se sabe que la solución analítica de esta ecuación es: $y = 3x^{2.1} + 5x - 4$.

Guardamos los valores iniciales y constantes (donde "nd" es el número de puntos):

```
x0=1.0;xn=4.0;n=100;nd=n+1;h=(xn-x0)/n;y0=4;y1=11.3;y2=6.93;
y3=0.693;vcx={1.3,1.5,1.8,2.0,2.4,2.9,3.3,3.7,3.8,4.0};
```

Escribimos la función correspondiente a la derivada de cuarto orden:

```
f4[x_,y_,y1_,y2_,y3_]:= (3*x*y2-y1-1.8711*x^-1.9-14.49*x^1.1+5)/3;
```

Observe que aún cuando en la ecuación no existen las variables "y" y "y2", se incluyen como parámetros de la función para hacerla más general. Ahora calculamos los valores de "x" y "y" aplicando las ecuaciones 19:

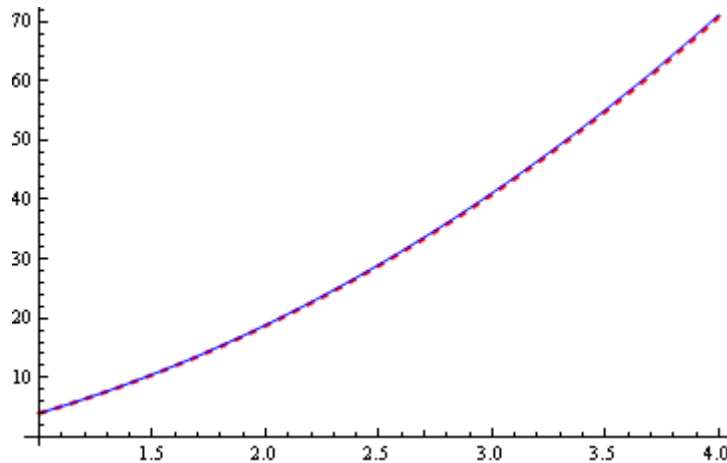
```
vx=ConstantArray[0,{nd}];vy=vx;vx[[1]]:=x0;vy[[1]]:=y0;
Do[vy[[j+1]]:=vy[[j]]+y1*h;y4=f4[vx[[j]],vy[[j]],y1,y2,y3];y1=y1+y2*h;y2=y2+y3*h;y3=y3+y4*h;vx[[j+1]]:=vx[[j]]+h,{j,n}]
```

Con los valores de "vx" y "vy" construimos la función de interpolación, empleando en este ejemplo "Interpolation" (como el número de puntos es muy grande no resultan adecuados los métodos de interpolación polinomial):

```
puntos=Transpose[{vx,vy}];
fx=Interpolation[puntos];
```

Graficamos el resultado conocido (en rojo y con guiones gruesos) y la función de interpolación (en azul) para ver cuan bien se ajustan los resultados calculados a los correctos:

```
fs[x_]:=3*x^2.1+5*x-4;
g1=Plot[fx[x],{x,x0,xn},PlotStyle->{Red,Dashed,Thick}];
g2=Plot[fs[x],{x,x0,xn},PlotStyle->Blue];Show[g1,g2]
```



Como se puede ver la función de interpolación se ajusta bastante bien a los valores correctos. Finalmente, con la función de interpolación, calculamos los valores de "y" requeridos:

```
Do[Print["Para x = ", vcx[[i]], " el valor de y es: ", fx[vcx[[i]]],
  {i, Length[vcx]}]
Para x = 1.3 el valor de y es: 7.67281
Para x = 1.5 el valor de y es: 10.4752
Para x = 1.8 el valor de y es: 15.2203
Para x = 2. el valor de y es: 18.7499
Para x = 2.4 el valor de y es: 26.702
Para x = 2.9 el valor de y es: 38.3442
Para x = 3.3 el valor de y es: 49.0425
Para x = 3.7 el valor de y es: 60.9894
Para x = 3.8 el valor de y es: 64.1731
Para x = 4. el valor de y es: 70.7782
```

27.1.4.2. Ejemplo 2

Como segundo ejemplo resolveremos la siguiente ecuación diferencial de tercer orden:

$$y''' + ty'' - ty' - 2y = t \quad \begin{cases} y = 0; y' = 0; y'' = 0 \text{ para } t = 0 \\ y = ? \text{ para } t = 0.2, 0.3, 0.4, 0.8, 1.1, 1.4, 1.5 \end{cases}$$

En este ejemplo resolveremos el problema en un módulo y para crear la función de interpolación emplearemos el método de interpolación segmentaria cúbica:

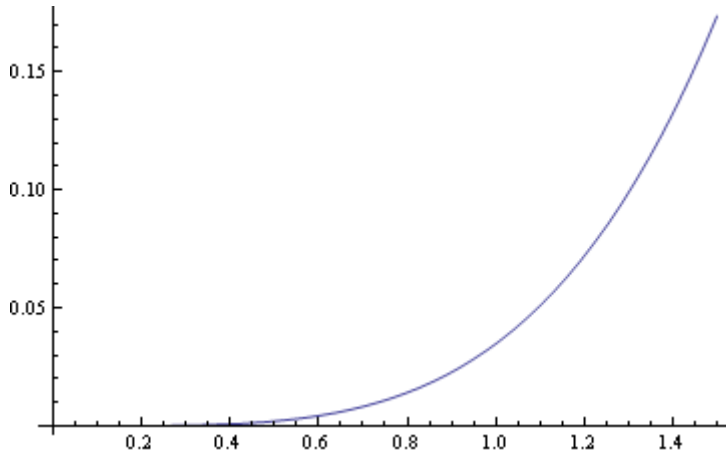
```
Module[{t0=0,tn=1.5,y0=0,y1=0,y2=0,y3,n=80,nd,f3,h,vt,vy,
  a,b,c,d,p,q,r,s,ft,j,vct={0.2,0.3,0.4,0.8,1.1,1.4,1.5}},
h=(tn-t0)/n;nd=n+1;
f3[t_,y_,y1_,y2_]:=t+2 y+t*y1-t*y2;
vt=ConstantArray[0,{nd}];vy=vt;vt[[1]]=t0;vy[[1]]=y0;
Do[vy[[j+1]]=vy[[j]]+y1*h;y3=f3[vt[[j]],vy[[j]],y1,y2];
  y1=y1+y2*h;y2=y2+y3*h;vt[[j+1]]=vt[[j]]+h,{j,n}];
h=Table[vt[[j+1]]-vt[[j]],{j,nd-1}];
a=vy;
p=Table[3*((a[[j+1]]-a[[j]])/h[[j]]-(a[[j]]-a[[j-1]])/h[[j-1]]),{j,nd-1}];
r=ConstantArray[0,nd-1];s=r;
Do[q=2*(h[[j-1]]+h[[j]])-h[[j-1]]*r[[j-1]];
  r[[j]]=h[[j]]/q;
  s[[j]]=(p[[j]]-h[[j-1]]*s[[j-1]])/q,{j,2,nd-1}];
```

```

c=ConstantArray[0,nd];
Do[c[[j]]=s[[j]]-r[[j]]*c[[j+1]],{j,nd-1,2,-1}];
d=Table[(c[[j+1]]-c[[j]])/(3*h[[j]]),{j,nd-1}];
b=Table[(a[[j+1]]-a[[j]])/h[[j]]-(2*c[[j]]+c[[j+1]])*h[[j]]/3,{j,nd-1}];
ft[t_]:= (For[j=1,t>vt[[j+1]] && j<nd-1,j++;h=t-vt[[j]];
  a[[j]]+(b[[j]]+(c[[j]]+d[[j]]*h)*h)*h);
Print[Plot[ft[t],{t,t0,tn}]];
Do[Print["Para t= ",vct[[i]]," el valor de y es: ", ft[vct[[i]]]],
  {i,Length[vct]}]
]

```

Ejecutando el módulo se obtiene:



```

Para t= 0.2 el valor de y es: 0.0000352044
Para t= 0.3 el valor de y es: 0.000223604
Para t= 0.4 el valor de y es: 0.000782861
Para t= 0.8 el valor de y es: 0.0141231
Para t= 1.1 el valor de y es: 0.050906
Para t= 1.4 el valor de y es: 0.132329
Para t= 1.5 el valor de y es: 0.17362

```

27.1.5. Sistemas de ecuaciones diferenciales

Cuando en lugar de una ecuación diferencial se debe resolver un sistema de ecuaciones diferenciales y en dicho sistema existen ecuaciones de segundo o mayor orden, las mismas deben ser transformadas previamente en ecuaciones diferenciales de primer orden (siguiendo el procedimiento descrito en el anterior acápite).

De esta manera es siempre posible transformar un sistema de ecuaciones diferenciales de cualquier orden en un sistema de "m" ecuaciones diferenciales de primer orden:

$$\begin{aligned}
 u' &= f_1(t, u, v, w, \dots, z) \\
 v' &= f_2(t, u, v, w, \dots, z) \\
 w' &= f_3(t, u, v, w, \dots, z) \\
 &\dots \\
 z' &= f_m(t, u, v, w, \dots, z)
 \end{aligned} \tag{14}$$

En este sistema "t" es la variable independiente y u, v, w, ..., z son las variables dependientes.

Aplicando la ecuación de Euler (13) a cada una de las ecuaciones del sistema, en un punto "j" cualquiera, se tiene:

$$\left. \begin{aligned} u_{j+1} &= u_j + u'_j * h = u_j + f_1(t, u, v, w, \dots, z) * h \\ v_{j+1} &= v_j + v'_j * h = v_j + f_2(t, u, v, w, \dots, z) * h \\ w_{j+1} &= w_j + w'_j * h = w_j + f_3(t, u, v, w, \dots, z) * h \\ &\dots \\ z_{j+1} &= z_j + z'_j * h = z_j + f_m(t, u, v, w, \dots, z) * h \\ t_{j+1} &= t_j + h \end{aligned} \right\} j = 1 \rightarrow n \quad (15)$$

Estas ecuaciones pueden ser empleadas también para resolver una ecuación diferencial de enésimo orden (transformándola en un sistema de ecuaciones diferenciales de primer orden).

Cuando se resuelven sistemas de ecuaciones diferenciales se obtienen "m" vectores (uno por cada variable) con los cuales se pueden crear "m" funciones de interpolación, mismas que son las soluciones de las "m" ecuaciones del sistema.

27.1.6. Ejemplos: Sistemas de ecuaciones diferenciales

27.1.6.1. Ejemplo 1

Como primer ejemplo calcularemos los valores de u , v , w del siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned} u' - v + 2w + t^2 - 2t^{1.5} - 1.2t^{0.2} + 10 &= 0 \\ v' + 5w - 5t^{1.5} - 2t + 35 &= 0 \\ w' - 2u + v - t^2 + 2t^{1.2} - 1.5t^{0.5} + 10 &= 0 \\ \begin{cases} u = 3, v = -4, w = -7 \text{ para } t = 0 \\ u = ?, v = ?, w = ? \text{ para } t = 0.05, 0.1, 0.14, 0.2, 0.23, 0.26, 0.30, 0.33, 0.35 \end{cases} \end{aligned}$$

Donde u , v y w son las variables dependientes y t la variable independiente. Las soluciones analíticas de este sistema son: $u = t^{1.2} + 3$; $v = t^2 - 4$ y $w = t^{1.5} - 7$.

Primero asignamos los valores conocidos a variables y calculamos el valor de "h" fijando el número de segmentos en 40:

```
t0=0;tn=0.35;u0=3;v0=-4;w0=-7;
vct={0.05,0.1,0.14,0.2,0.23,0.26,0.30,0.33,0.35};n=40;
nd=n+1;h=(tn-t0)/n;
```

Creamos las funciones correspondientes a las tres ecuaciones diferenciales de primer orden del sistema:

```
f1[t_,u_,v_,w_]:=v-2*w-t^2+2*t^1.5+1.2*t^0.2-10;
f2[t_,u_,v_,w_]:=-5*w+5*t^1.5+2*t-35;
f3[t_,u_,v_,w_]:=2*u-v+t^2-2*t^1.2+1.5*t^0.5-10;
```

Aplicando las ecuaciones de Euler (ecuación 22) calculamos los valores de los vectores de solución ("vu", "vv", "vw" y "vt"):

```
vt=vu=vv=vw=ConstantArray[0,nd];vt[[1]]=t0;vu[[1]]=u0;
vv[[1]]=v0;vw[[1]]=w0;
Do[vu[[j+1]]=vu[[j]]+f1[vt[[j]],vu[[j]],vv[[j]],vw[[j]]]*h;
vv[[j+1]]=vv[[j]]+f2[vt[[j]],vu[[j]],vv[[j]],vw[[j]]]*h;
vw[[j+1]]=vw[[j]]+f3[vt[[j]],vu[[j]],vv[[j]],vw[[j]]]*h;
vt[[j+1]]=vt[[j]]+h,{j,n}]
```

Con los vectores calculados e "Interpolation" creamos las funciones de interpolación:

```
fu=Interpolation[Transpose[{vt,vu}]];
fv=Interpolation[Transpose[{vt,vv}]];
fw=Interpolation[Transpose[{vt,vw}]];
```

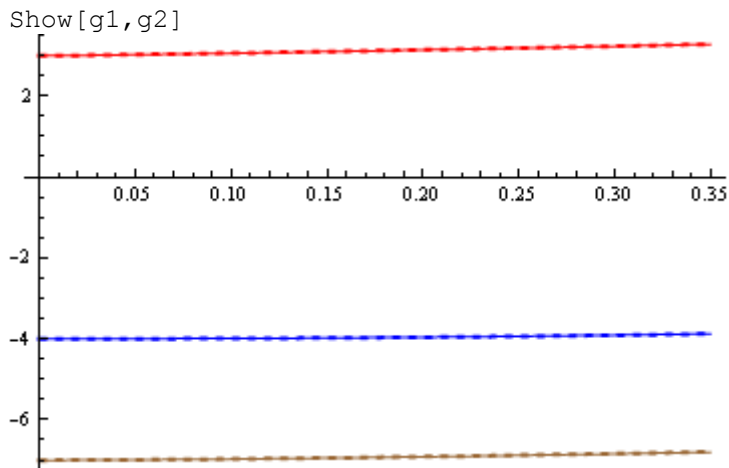
Graficamos las tres funciones empleando los colores: rojo, azul y café.

```
g1=Plot[{fu[t],fv[t],fw[t]},{t,t0,tn},PlotStyle->{Red,Blue,Brown}];
```

Programamos las tres soluciones conocidas y las graficamos con guiones gruesos empleando los colores rojo, azul y café:

```
fcu[t_]:=t^1.2+3;fcv[t_]:=t^2-4;fcw[t_]:=t^1.5-7;
g2=Plot[{fcu[t],fcv[t],fcw[t]},{t,t0,tn},PlotStyle->
{{Red,Dashed,Thick},{Blue,Dashed,Thick},{Brown,Dashed,Thick}}};
```

Mostramos las dos gráficas:



Como se puede observar las funciones de interpolación calculadas concuerdan bastante bien con las soluciones conocidas.

Finalmente, empleando las funciones de interpolación, calculamos los valores requeridos:

```
Do[Print["Para t= ",vct[[i]]];
Print["El valor de u es: ",fu[vct[[i]]]];
Print["El valor de v es: ",fv[vct[[i]]]];
Print["El valor de w es: ",fw[vct[[i]]]],{i,Length[vct]}]
Para t= 0.05
El valor de u es: 3.02328
El valor de v es: -3.99766
El valor de w es: -6.99083
Para t= 0.1
El valor de u es: 3.05872
El valor de v es: -3.98998
El valor de w es: -6.97143
Para t= 0.14
El valor de u es: 3.09014
El valor de v es: -3.98007
El valor de w es: -6.9514
Para t= 0.2
El valor de u es: 3.1409
El valor de v es: -3.95892
El valor de w es: -6.91537
```

```

Para t= 0.23
El valor de u es: 3.16759
El valor de v es: -3.94554
El valor de w es: -6.895
Para t= 0.26
El valor de u es: 3.19505
El valor de v es: -3.93028
El valor de w es: -6.87321
Para t= 0.3
El valor de u es: 3.23272
El valor de v es: -3.90702
El valor de w es: -6.84208
Para t= 0.33
El valor de u es: 3.2617
El valor de v es: -3.8874
El valor de w es: -6.81728
Para t= 0.35
El valor de u es: 3.28134
El valor de v es: -3.87328
El valor de w es: -6.80008

```

27.1.6.2. Ejemplo 2

Como segundo ejemplo resolveremos el siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned}\frac{dx}{dt} &= x' = 2x + 3y \\ \frac{dy}{dt} &= y' = 2x + y \\ \begin{cases} x = 2.7; y = 2 & \text{para } t = 0 \\ x = ?; y = ? & \text{para } t = 0.2, 0.5, 0.7, 0.9, 1.2, 1.3 \end{cases}\end{aligned}$$

En este caso resolveremos el problema en un módulo:

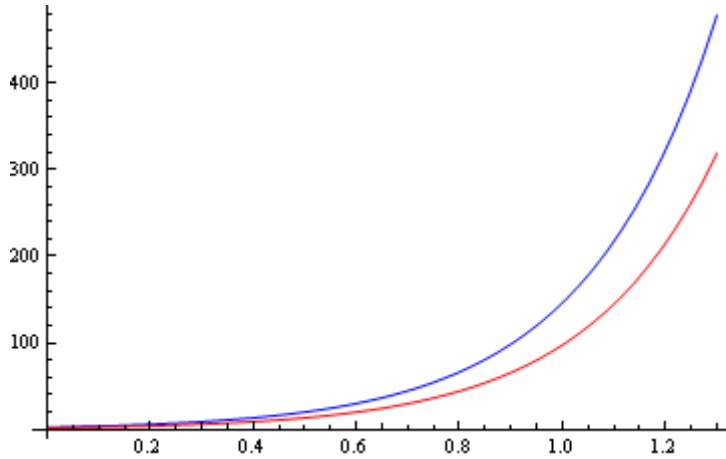
```

Module[{t0=0,tn=1.3,x0=2.7,y0=2,n=200,h,f1,f2,vt,vx,vy,fx,
  fy,vct={0.2,0.5,0.7,0.9,1.2,1.3}},
h=(tn-t0)/n;
f1[t_,x_,y_]:=2 x+3 y;
f2[t_,x_,y_]:=2 x+y;
vt=vx=vy=ConstantArray[0,n+1];
vt[[1]]=t0;vx[[1]]=x0;vy[[1]]=y0;
Do[vx[[j+1]]=vx[[j]]+f1[vt[[j]],vx[[j]],vy[[j]]]*h;
  vy[[j+1]]=vy[[j]]+f2[vt[[j]],vx[[j]],vy[[j]]]*h;
  vt[[j+1]]=vt[[j]]+h,{j,n}];
fx=Interpolation[Transpose[{vt,vx}]];
fy=Interpolation[Transpose[{vt,vy}]];
Print["Gráfica de las soluciones: "];
Print[Plot[{fx[t],fy[t]},{t,t0,tn},PlotStyle->{Blue,Red}]];
Do[Print["Para t= ",vt[[i]]];
  Print["El valor de x es: ",fx[vct[[i]]];
  Print["El valor de y es: ",fy[vct[[i]]]],{i,Length[vct]}]
]

```

Ejecutando el módulo se obtiene:

Gráfica de las soluciones:



Para $t= 0.2$
 El valor de x es: 6.11401
 El valor de y es: 4.23964
 Para $t= 0.5$
 El valor de x es: 20.2387
 El valor de y es: 13.6136
 Para $t= 0.7$
 El valor de x es: 44.6845
 El valor de y es: 29.8887
 Para $t= 0.9$
 El valor de x es: 98.518
 El valor de y es: 65.7597
 Para $t= 1.2$
 El valor de x es: 322.237
 El valor de y es: 214.884
 Para $t= 1.3$
 El valor de x es: 478.29
 El valor de y es: 318.915

27.2. Métodos de Runge - Kutta

Los métodos de *Runge-Kutta*, son aquellos que tienen la forma general:

$$y_{i+1} = y_i + a_1 k_1 + a_2 k_2 + a_3 k_3 + \dots + a_n k_n \quad (16)$$

Donde los valores de k tienen la forma:

$$\begin{aligned}
 k_1 &= hf(x_i, y_i) \\
 k_2 &= hf(x_i + p_1 h, y_i + q_{1,1} k_1) \\
 k_3 &= hf(x_i + p_2 h, y_i + q_{2,1} k_1 + q_{2,2} k_2) \\
 k_4 &= hf(x_i + p_3 h, y_i + q_{3,1} k_1 + q_{3,2} k_2 + q_{3,3} k_3) \\
 &\dots \\
 k_n &= hf(x_i + p_{n-1} h, y_i + q_{n-1,1} k_1 + q_{n-1,2} k_2 + \dots + q_{n-1,n-1} k_{n-1})
 \end{aligned} \quad (17)$$

Donde $f(x,y)$ es la función correspondiente a la derivada primera de la variable dependiente, x_i , y_i son los valores conocidos de las variables independiente y dependiente para un punto dado i ; y_{i+1} es el valor de la variable dependiente en el siguiente punto y h es el incremento de la variable independiente. Para todas las formas, el valor de la variable independiente del siguiente punto se calcula como en el método de *Euler*:

$$x_{i+1} = x_i + h \quad (18)$$

En base a la ecuación (23), estableciendo el número de evaluaciones funcionales (valores de k) y calculando los parámetros " a ", " p " y " q ", se puede deducir un número prácticamente infinito de ecuaciones de *Runge-Kutta*.

No todas las formas de Runge - Kutta predicen correctamente los valores de la variable dependiente, razón por la cual una vez deducidas deben ser probadas para determinar su confiabilidad.

Muchos investigadores han deducido y puesto a prueba varias formas de *Runge-Kutta*. Estas ecuaciones se clasifican generalmente como métodos (m,n) , donde " m " es el número de parámetros " a_i " y " n " es el número de valores " k " que deben ser calculados.

Algunas de las ecuaciones de las formas de *Runge-Kutta* que más se utilizan en la práctica son las siguientes:

La ecuación (2,2) de *Runge-Kutta* con una exactitud comparable con la del método modificado de *Euler*:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{2}(k_1 + k_2) \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf(x_i + h, y_i + k_1) \end{aligned} \quad (19)$$

La ecuación (3,3) de *Runge-Kutta* con un error de orden h^4 :

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6}(k_1 + 4k_2 + k_3) \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\ k_3 &= hf(x_i + h, y_i - k_1 + 2k_2) \end{aligned} \quad (20)$$

La ecuación (4,4) de *Runge-Kutta*, conocida como la forma clásica:

$$\begin{aligned} y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= hf(x_i, y_i) \\ k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\ k_3 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) \\ k_4 &= hf(x_i + h, y_i + k_3) \end{aligned} \quad (21)$$

La ecuación (4,4) de *Runge-Kutta* conocida como la forma de *Gill*, la cual minimiza los errores debido a redondeos:

$$\begin{aligned}
y_{i+1} &= y_i + \frac{1}{6} \left(k_1 + (2 - \sqrt{2})k_2 + (2 + \sqrt{2})k_3 + k_4 \right) \\
k_1 &= hf(x_i, y_i) \\
k_2 &= hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) \\
k_3 &= hf\left(x_i + \frac{h}{2}, y_i + (\sqrt{2} - 1)\frac{k_1}{2} + (2 - \sqrt{2})\frac{k_2}{2}\right) \\
k_4 &= hf\left(x_i + h, y_i - \sqrt{2}\frac{k_2}{2} + \left(1 + \frac{\sqrt{2}}{2}\right)k_2\right)
\end{aligned} \tag{22}$$

Y una ecuación (5,6) conocida como la forma de *Butcher*, que suele ser empleada cuando se requiere gran exactitud en los cálculos:

$$\begin{aligned}
y_{i+1} &= y_i + \frac{1}{90} (7k_1 + 32k_3 + 12k_4 + 35k_5 + 7k_6) \\
k_1 &= hf(x_i, y_i) \\
k_2 &= hf\left(x_i + \frac{h}{4}, y_i + \frac{k_1}{4}\right) \\
k_3 &= hf\left(x_i + \frac{h}{4}, y_i + \frac{k_1}{8} + \frac{k_2}{8}\right) \\
k_4 &= hf\left(x_i + \frac{h}{2}, y_i - \frac{k_2}{2} + k_3\right) \\
k_5 &= hf\left(x_i + \frac{3}{4}h, y_i + \frac{3}{16}k_1 + \frac{9}{16}k_4\right) \\
k_6 &= hf\left(x_i + h, y_i - \frac{3}{7}k_1 + \frac{2}{7}k_2 + \frac{12}{7}k_3 - \frac{12}{7}k_4 + \frac{8}{7}k_5\right)
\end{aligned} \tag{23}$$

En este capítulo emplearemos la forma clásica (ecuación 28), por ser la forma que más se emplea en la práctica. Por lo tanto cuando hagamos referencia al método de *Runge Kutta* estaremos haciendo referencia a esta ecuación.

27.2.1. Resolución de una ecuación diferencial de primer orden

Todas las ecuaciones de *Runge Kutta* permiten resolver sólo ecuaciones diferenciales de primer orden y el procedimiento de cálculo es esencialmente el mismo que en el método de *Euler*, sólo que en lugar de calcular el valor de la variable dependiente con la ecuación 14 ($y_{i+1} = y_i + y'_i h$) se emplea la ecuación 27.

27.2.2. Ejemplos: Ecuaciones de primer orden

27.2.2.1. Ejemplo 1

Como primer ejemplo emplearemos el método de *Runge Kutta* para resolver la ecuación diferencial resuelta con el método de *Euler*:

$$3y' + 4xy - 8x^3 + 4x = 0 \quad \begin{cases} y = 4 \text{ para } x = 2 \\ y = ? \text{ para } x = 2.5, 3.1, 4.7, 5.3, 5.8, 6 \end{cases}$$

Guardamos primero los valores conocidos en variables:

$x0=2.; y0=4.; xn=6.;$

Fijamos el número de segmentos en 50 y calculamos el valor de "h":

```
n=50;h=(xn-x0)/n;
```

Programamos la ecuación diferencial:

```
f[x_,y_]:= (8*x^3-4*x-4*x*y)/3
```

Empleamos las ecuaciones de Runge - Kutta (ecuaciones 28) para calcular los vectores "vx" y "vy":

```
vx=vy=ConstantArray[0,{n+1}];vx[[1]]=x0;vy[[1]]=y0;
```

```
Do[k1=h*f[vx[[i]],vy[[i]]];
```

```
k2=h*f[vx[[i]]+h/2,vy[[i]]+k1/2];
```

```
k3=h*f[vx[[i]]+h/2,vy[[i]]+k2/2];
```

```
k4=h*f[vx[[i]]+h,vy[[i]]+k3];
```

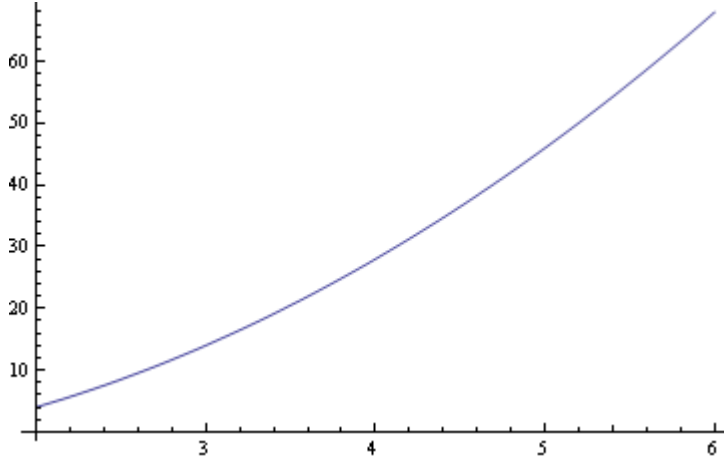
```
vy[[i+1]]=vy[[i]]+1/6*(k1+2*k2+2*k3+k4);vx[[i+1]]=vx[[i]]+h,{i,n}]
```

Creamos la función de interpolación con estos vectores e "Interpolation":

```
fx=Interpolation[Transpose[{vx,vy}]];
```

Graficamos la función de interpolación para ver la forma que tiene la solución:

```
Plot[fx[x],{x,x0,xn}]
```



Finalmente calculamos los valores de "y" para los valores conocidos de "x":

```
vcx={2.5,3.1,4.7,5.3,5.8,6.};
```

```
Do[Print["Para x = ",vcx[[i]]," el valor de y es : ",fx[vcx[[i]]],
```

```
{i,Length[vcx]}]
```

```
Para x = 2.5 el valor de y es : 8.50002
```

```
Para x = 3.1 el valor de y es : 15.22
```

```
Para x = 4.7 el valor de y es : 40.1801
```

```
Para x = 5.3 el valor de y es : 52.1801
```

```
Para x = 5.8 el valor de y es : 63.2801
```

```
Para x = 6. el valor de y es : 68.0001
```

En este caso las soluciones son prácticamente exactas.

27.2.2.2. Ejemplo 2

Como segundo ejemplo resolveremos la siguiente ecuación diferencial:

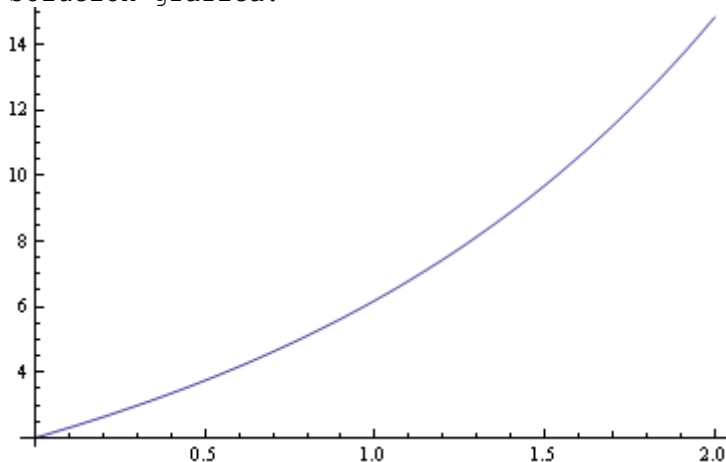
$$y' = 4e^{0.8x} - 0.5y \quad \begin{cases} y = 2 \text{ para } x = 0 \\ y = ? \text{ para } x = 0.3, 0.7, 0.9, 1.2, 1.5, 1.7, 2.0 \end{cases}$$

Prácticamente la única diferencia con relación al anterior ejemplo es que en este caso resolveremos el problema en un módulo:

```
Module[{x0,y0,xn,n,h,vx,vy,f,fx,k1,k2,k3,k4,
  vcx={0.3,0.7,0.9,1.2,1.5,1.7,2.0}},
  x0=0;y0=2.0;xn=2.0;n=60;h=(xn-x0)/n;
  vx=vy=ConstantArray[0,{n+1}];vx[[1]]=x0;vy[[1]]=y0;
  f[x_,y_]:=4*Exp[0.8*x]-0.5*y;
  Do[k1=h*f[vx[[i]],vy[[i]]];
    k2=h*f[vx[[i]]+h/2,vy[[i]]+k1/2];
    k3=h*f[vx[[i]]+h/2,vy[[i]]+k2/2];
    k4=h*f[vx[[i]]+h,vy[[i]]+k3];
    vy[[i+1]]=vy[[i]]+1/6*(k1+2*k2+2*k3+k4);vx[[i+1]]=vx[[i]]+h,{i,n}];
  fx=Interpolation[Transpose[{vx,vy}]];
  Print["Solución gráfica:"];
  Print[Plot[fx[x],{x,x0,xn}]];
  Do[Print["Para x = ",vcx[[i]]," el valor de y es: ",
    fx[vcx[[i]]]],{i,Length[vcx]}]
]
```

Ejecutando el módulo se obtiene:

Solución gráfica:



```
Para x = 0.3 el valor de y es: 2.98462
Para x = 0.7 el valor de y es: 4.62779
Para x = 0.9 el valor de y es: 5.63466
Para x = 1.2 el valor de y es: 7.44496
Para x = 1.5 el valor de y es: 9.70704
Para x = 1.7 el valor de y es: 11.528
Para x = 2.0 el valor de y es: 14.8439
```

27.2.3. Ecuaciones diferenciales de enésimo orden

Al igual que con el método de *Euler*, los métodos de *Runge-Kutta* pueden ser empleados para resolver ecuaciones diferenciales de enésimo orden. Para ello se sigue el mismo procedimiento que en el método de *Euler*, es decir se transforma la ecuación diferencial en un sistema de ecuaciones diferenciales de primer orden y se aplica la ecuación de *Runge-Kutta* (ecuación 28) a cada una de las ecuaciones resultantes. Ello implica el cálculo de los cuatro parámetros de la ecuación de *Runge-Kutta* para cada una de dichas ecuaciones, así por ejemplo si se tiene una ecuación diferencial de segundo orden, el sistema de ecuaciones diferenciales de primer orden equivalente a dicha ecuación sería:

$$y'_1 = f_1(x, y, y_1) = y_1$$

$$y'_2 = f_2(x, y, y_1)$$

Entonces las ecuaciones de Runge-Kutta son:

$$k_1 = hf_1(x_i, y_i, y_{1_i}) = h(y_{1_i})$$

$$k_{1_1} = hf_2(x_i, y_i, y_{1_i})$$

$$k_2 = hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}\right) = h\left(y_{1_i} + \frac{k_{1_1}}{2}\right)$$

$$k_{2_1} = hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}\right)$$

$$k_3 = hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}\right) = h\left(y_{1_i} + \frac{k_{2_1}}{2}\right)$$

$$k_{3_1} = hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}\right)$$

$$k_4 = hf_1(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}) = h(y_{1_i} + k_{3_1})$$

$$k_{4_1} = hf_2(x_i + h, y_i + k_3, y_{1_i} + k_{3_1})$$

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

$$y_{1_{i+1}} = y_{1_i} + \frac{1}{6}(k_{1_1} + 2k_{2_1} + 2k_{3_1} + k_{4_1})$$

Donde como se puede observar, primero se calculan los parámetros k_1 , luego los parámetros k_2 , los parámetros k_3 , los parámetros k_4 y finalmente los valores de "y" y "y1". Como también se puede observar, en el caso de la función " f_1 ", que es igual a una variable, el cálculo es directo.

Si en lugar de una ecuación de segundo orden tenemos una de cuarto, el sistema de ecuaciones diferenciales de primer orden, equivalente a dicha ecuación sería:

$$y'_1 = f_1(x, y, y_1, y_2, y_3) = y_1$$

$$y'_2 = f_2(x, y, y_1, y_2, y_3) = y_2$$

$$y'_3 = f_3(x, y, y_1, y_2, y_3) = y_3$$

$$y'_4 = f_4(x, y, y_1, y_2, y_3)$$

Y las ecuaciones de Runge-Kutta para este sistema son:

$$\begin{aligned}
k_1 &= hf_1(x_i, y_i, y_{1_i}, y_{2_i}, y_{3_i}) = h(y_{1_i}) \\
k_{1_1} &= hf_2(x_i, y_i, y_{1_i}, y_{2_i}, y_{3_i}) = h(y_{2_i}) \\
k_{1_2} &= hf_3(x_i, y_i, y_{1_i}, y_{2_i}, y_{3_i}) = h(y_{3_i}) \\
k_{1_3} &= hf_4(x_i, y_i, y_{1_i}, y_{2_i}, y_{3_i}) \\
k_2 &= hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, y_{3_i} + \frac{k_{1_3}}{2}\right) = h\left(y_{1_i} + \frac{k_{1_1}}{2}\right) \\
k_{2_1} &= hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, y_{3_i} + \frac{k_{1_3}}{2}\right) = h\left(y_{2_i} + \frac{k_{1_2}}{2}\right) \\
k_{2_2} &= hf_3\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, y_{3_i} + \frac{k_{1_3}}{2}\right) = h\left(y_{3_i} + \frac{k_{1_3}}{2}\right) \\
k_{2_3} &= hf_4\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, y_{3_i} + \frac{k_{1_3}}{2}\right) \\
k_3 &= hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, y_{3_i} + \frac{k_{2_3}}{2}\right) = h\left(y_{1_i} + \frac{k_{2_1}}{2}\right) \\
k_{3_1} &= hf_2\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, y_{3_i} + \frac{k_{2_3}}{2}\right) = h\left(y_{2_i} + \frac{k_{2_2}}{2}\right) \\
k_{3_2} &= hf_3\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, y_{3_i} + \frac{k_{2_3}}{2}\right) = h\left(y_{3_i} + \frac{k_{2_3}}{2}\right) \\
k_{3_3} &= hf_4\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, y_{3_i} + \frac{k_{2_3}}{2}\right) \\
k_4 &= hf_1(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, y_{3_i} + k_{3_3}) = h(y_{1_i} + k_{3_1}) \\
k_{4_1} &= hf_2(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, y_{3_i} + k_{3_3}) = h(y_{2_i} + k_{3_2}) \\
k_{4_2} &= hf_3(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, y_{3_i} + k_{3_3}) = h(y_{3_i} + k_{3_3}) \\
k_{4_3} &= hf_4(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, y_{3_i} + k_{3_3}) \\
y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
y_{1_{i+1}} &= y_{1_i} + \frac{1}{6}(k_{1_1} + 2k_{2_1} + 2k_{3_1} + k_{4_1}) \\
y_{2_{i+1}} &= y_{2_i} + \frac{1}{6}(k_{1_2} + 2k_{2_2} + 2k_{3_2} + k_{4_2}) \\
y_{3_{i+1}} &= y_{3_i} + \frac{1}{6}(k_{1_3} + 2k_{2_3} + 2k_{3_3} + k_{4_3})
\end{aligned}$$

En general, para una ecuación de orden "m", el sistema de ecuaciones diferenciales de primer orden equivalente a dicho sistema es:

$$\begin{aligned}
y'_1 &= f_1(x, y, y_1, y_2, y_3, \dots, y_{m-1}) = y_1 \\
y'_2 &= f_2(x, y, y_1, y_2, y_3, \dots, y_{m-1}) = y_2 \\
y'_3 &= f_3(x, y, y_1, y_2, y_3, \dots, y_{m-1}) = y_3 \\
&\dots \\
y'_{m-1} &= f_{m-1}(x, y, y_1, y_2, y_3, \dots, y_{m-1}) = y_{m-1} \\
y'_m &= f_m(x, y, y_1, y_2, y_3, \dots, y_{m-1})
\end{aligned}$$

Y las ecuaciones de Runge-Kutta para dicho sistema son:

$$\begin{aligned}
 k_1 &= hf_1(x_i, y_i, y_{1_i}, y_{2_i}, \dots, y_{m-1_i}) = h(y_{1_i}) \\
 k_{1_j} &= hf_{j+1}(x_i, y_i, y_{1_i}, y_{2_i}, \dots, y_{m-1_i}) = h(y_{j+1_i}) \quad \{j = 1 \rightarrow m-2\} \\
 k_{1_{m-1}} &= hf_4(x_i, y_i, y_{1_i}, y_{2_i}, \dots, y_{m-1_i}) \\
 k_2 &= hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, \dots, y_{m-1_i} + \frac{k_{1_{m-1}}}{2}\right) = h\left(y_{1_i} + \frac{k_{1_1}}{2}\right) \\
 k_{2_j} &= hf_{j+1}\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, \dots, y_{m-1_i} + \frac{k_{1_{m-1}}}{2}\right) = h\left(y_{j+1_i} + \frac{k_{1_{j+1}}}{2}\right) \quad \{j = 1 \rightarrow m-2\} \\
 k_{2_{m-1}} &= hf_m\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}, y_{1_i} + \frac{k_{1_1}}{2}, y_{2_i} + \frac{k_{1_2}}{2}, \dots, y_{m-1_i} + \frac{k_{1_{m-1}}}{2}\right) \\
 k_3 &= hf_1\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, \dots, y_{m-1_i} + \frac{k_{2_{m-1}}}{2}\right) = h\left(y_{1_i} + \frac{k_{2_1}}{2}\right) \\
 k_{3_j} &= hf_{j+1}\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, \dots, y_{m-1_i} + \frac{k_{2_{m-1}}}{2}\right) = h\left(y_{j+1_i} + \frac{k_{2_{j+1}}}{2}\right) \quad \{j = 1 \rightarrow m-2\} \\
 k_{3_{m-1}} &= hf_m\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}, y_{1_i} + \frac{k_{2_1}}{2}, y_{2_i} + \frac{k_{2_2}}{2}, \dots, y_{m-1_i} + \frac{k_{2_{m-1}}}{2}\right) \\
 k_4 &= hf_1(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, \dots, y_{m-1_i} + k_{3_{m-1}}) = h(y_{1_i} + k_{3_1}) \\
 k_{4_j} &= hf_{j+1}(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, \dots, y_{m-1_i} + k_{3_{m-1}}) = h(y_{j+1_i} + k_{3_{j+1}}) \quad \{j = 1 \rightarrow m-2\} \\
 k_{4_{m-1}} &= hf_m(x_i + h, y_i + k_3, y_{1_i} + k_{3_1}, y_{2_i} + k_{3_2}, \dots, y_{m-1_i} + k_{3_{m-1}}) \\
 y_{i+1} &= y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
 y_{j+1} &= y_{j_i} + \frac{1}{6}(k_{1_j} + 2k_{2_j} + 2k_{3_j} + k_{4_j}) \quad \{j = 1 \rightarrow m-1\}
 \end{aligned} \tag{24}$$

Como de costumbre el valor de la variable independiente se calcula con:

$$x_{j+1} = x_j + h \tag{25}$$

27.2.4. Ejemplos: Ecuaciones diferenciales de enésimo orden

27.2.4.1. Ejemplo 1

Como primer ejemplo volveremos a resolver la ecuación diferencial:

$$\begin{aligned}
 3y''' - 3xy'' + y' + 1.8711x^{-1.9} + 14.49x^{1.1} - 5 &= 0 \\
 \begin{cases} y = 4, y' = 11.3, y'' = 6.93, y''' = 0.693 \text{ para } x = 1 \\ y = ?, y' = ?, y'' = ?, y''' = ? \text{ para } x = 1.3, 1.5, 1.8, 2.0, 2.4, 2.9, 3.3, 3.7, 3.8, 4.0 \end{cases}
 \end{aligned}$$

Que como sabemos tiene la solución: $y = 3x^{2.1} + 5x - 4$.

Primero guardamos los valores conocidos y calculamos el incremento "h" fijando el número de segmentos en 100:

```
x0=1.0;xn=4.0;y0=4;y1=11.3;y2=6.93;y3=0.693;
vcx={1.3,1.5,1.8,2.0,2.4,2.9,3.3,3.7,3.8,4.0};n=100;h=(xn-x0)/n;
```

Programamos la función correspondiente a la derivada de cuarto orden:

```
f4[x_,y_,y1_,y2_,y3_] := (3*x*y2-y1-1.8711*x^-1.9-14.49*x^1.1+5)/3;
```

Inicializamos los vectores de resultados "vx" y "vy":

```
vx=vy=ConstantArray[0,n+1];vx[[1]]:=x0;vy[[1]]=y0;
```

Calculamos los vectores de resultados "vx" y "vy" aplicando las ecuaciones de Runge-Kutta:

```
Do[k1=h*y1;k11=h*y2;k12=h*y3;
  k13=h*f4[vx[[j]],vy[[j]],y1,y2,y3];
  k2=h*(y1+k11/2);k21=h*(y2+k12/2);k22=h*(y3+k13/2);
  k23=h*f4[vx[[j]]+h/2,vy[[j]]+k1/2,y1+k11/2,y2+k12/2,y3+k13/2];
  k3=h*(y1+k21/2);k31=h*(y2+k22/2);k32=h*(y3+k23/2);
  k33=h*f4[vx[[j]]+h/2,vy[[j]]+k2/2,y1+k21/2,y2+k22/2,y3+k23/2];
  k4=h*(y1+k31);k41=h*(y2+k32);k42=h*(y3+k33);
  k43=h*f4[vx[[j]]+h,vy[[j]]+k3,y1+k31,y2+k32,y3+k33];
  vy[[j+1]]=vy[[j]]+1/6*(k1+2*k2+2*k3+k4);
  vx[[j+1]]=vx[[j]]+h;
  y1=y1+1/6*(k11+2*k21+2*k31+k41);
  y2=y2+1/6*(k12+2*k22+2*k32+k42);
  y3=y3+1/6*(k13+2*k23+2*k33+k43),{j,n}]
```

Creamos la función de interpolación con "Interpolation":

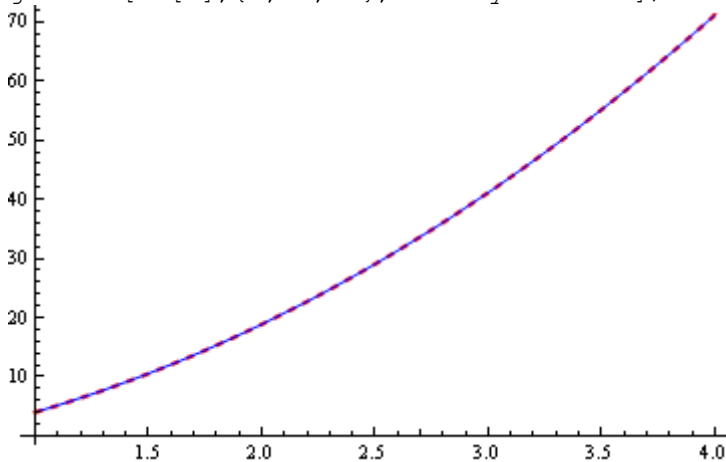
```
fx=Interpolation[Transpose[{vx,vy}]];
```

Programamos la función correspondiente a la solución:

```
fs[x_]:=3*x^2.1+5*x-4;
```

Graficamos la función de interpolación conjuntamente la función solución:

```
g1=Plot[fx[x],{x,x0,xn},PlotStyle->{Red,Dashed,Thick}];
g2=Plot[fs[x],{x,x0,xn},PlotStyle->Blue];Show[g1,g2]
```



Como era de esperar, debido a la mayor exactitud del método, en este caso la función de interpolación concuerda casi perfectamente con la solución.

Finalmente calculamos los valores requeridos:

```
Do[Print["Para x = ",vcx[[i]]," el valor de y es: ",fx[vcx[[i]]],
  {i,Length[vcx]}]
Para x = 1.3 el valor de y es: 7.70478
Para x = 1.5 el valor de y es: 10.5293
Para x = 1.8 el valor de y es: 15.3085
Para x = 2. el valor de y es: 18.8613
Para x = 2.4 el valor de y es: 26.861
Para x = 2.9 el valor de y es: 38.5645
Para x = 3.3 el valor de y es: 49.3129
Para x = 3.7 el valor de y es: 61.3107
```

Para x = 3.8 el valor de y es: 64.507

Para x = 4. el valor de y es: 71.1375

27.2.4.2. Ejemplo 2

Como segundo ejemplo volveremos a resolver la ecuación diferencial:

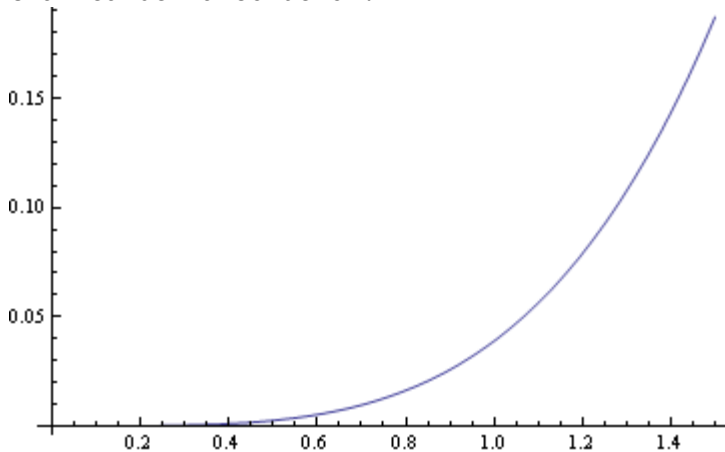
$$y''' + ty'' - ty' - 2y = t \quad \begin{cases} y = 0; y' = 0; y'' = 0 \text{ para } t = 0 \\ y = ? \text{ para } t = 0.2, 0.3, 0.4, 0.8, 1.1, 1.4, 1.5 \end{cases}$$

En este caso resolvemos el problema en un módulo:

```
Module[{t0=0,tn=1.5,y0=0,y1=0,y2=0,n=80,f3,h,vt,vy,
  k1,k11,k12,k2,k21,k22,k3,k31,k32,k4,k41,k42,ft,
  vct={0.2,0.3,0.4,0.8,1.1,1.4,1.5}},
h=(tn-t0)/n;
f3[t_,y_,y1_,y2_]:=t+2 y+t*y1-t*y2;
vt=vy=ConstantArray[0,n+1];vt[[1]]=t0;vy[[1]]=y0;
Do[k1=h*y1;k11=h*y2;
  k12=h*f3[vt[[j]],vy[[j]],y1,y2];
  k2=h*(y1+k11/2);k21=h*(y2+k12/2);
  k22=h*f3[vt[[j]]+h/2,vy[[j]]+k1/2,y1+k11/2,y2+k12/2];
  k3=h*(y1+k21/2);k31=h*(y2+k22/2);
  k32=h*f3[vt[[j]]+h/2,vy[[j]]+k2/2,y1+k21/2,y2+k22/2];
  k4=h*(y1+k31);k41=h*(y2+k32);
  k42=h*f3[vt[[j]]+h,vy[[j]]+k3,y1+k31,y2+k32];
  vy[[j+1]]=vy[[j]]+1/6*(k1+2*k2+2*k3+k4);
  vt[[j+1]]=vt[[j]]+h;
  y1=y1+1/6*(k11+2*k21+2*k31+k41);
  y2=y2+1/6*(k12+2*k22+2*k32+k42),{j,n}];
ft=Interpolation[Transpose[{vt,vy}]];
Print["Gráfica de la solución: "];
Print[Plot[ft[t],{t,t0,tn}]];
Do[Print["Para t= ",vct[[i]]," el valor de y es: ",ft[vct[[i]]]],
  {i,Length[vct]}]
]
```

Ejecutando el módulo se obtiene:

Gráfica de la solución:



Para t= 0.2 el valor de y es: 0.0000664137

Para t= 0.3 el valor de y es: 0.000334744

Para t= 0.4 el valor de y es: 0.00105175

Para t= 0.8 el valor de y es: 0.016264

Para t= 1.1 el valor de y es: 0.0563542

Para $t = 1.4$ el valor de y es: 0.143457

Para $t = 1.5$ el valor de y es: 0.187332

27.2.5. Sistemas de ecuaciones diferenciales

Al igual que el método de *Euler*, los métodos de *Runge-Kutta* pueden ser empleados para resolver sistemas de ecuaciones diferenciales. En caso de que en el sistema existan ecuaciones diferenciales de segundo orden o superior, dichas ecuaciones deben ser transformadas en ecuaciones diferenciales de primer orden (tal como se mostró en el acápite anterior). De esta manera siempre es posible transformar un sistema de ecuaciones diferenciales con ecuaciones de cualquier orden en un sistema de " m " ecuaciones diferenciales de primer orden de la forma:

$$\begin{aligned} y'_1 &= f_1(x, y_1, y_2, y_3, \dots, y_m) \\ y'_2 &= f_2(x, y_1, y_2, y_3, \dots, y_m) \\ y'_3 &= f_3(x, y_1, y_2, y_3, \dots, y_m) \\ &\dots \\ y'_m &= f_m(x, y_1, y_2, y_3, \dots, y_m) \end{aligned} \quad (26)$$

Donde, por comodidad y claridad, todas las variables dependientes inician su nombre con " y ". Aplicando las ecuaciones de *Runge Kutta* a este sistema de ecuaciones se tiene:

$$\begin{aligned} k_{1j} &= h * f_j(x_i, y_{1i}, y_{2i}, \dots, y_{mi}) \quad \{j = 1 \rightarrow m\} \\ k_{2j} &= h * f_j\left(x_i + \frac{h}{2}, y_{1i} + \frac{k_{1,1}}{2}, y_{2i} + \frac{k_{1,2}}{2}, \dots, y_{mi} + \frac{k_{1,m}}{2}\right) \quad \{j = 1 \rightarrow m\} \\ k_{3j} &= h * f_j\left(x_i + \frac{h}{2}, y_{1i} + \frac{k_{2,1}}{2}, y_{2i} + \frac{k_{2,2}}{2}, \dots, y_{mi} + \frac{k_{2,m}}{2}\right) \quad \{j = 1 \rightarrow m\} \\ k_{4j} &= h * f_j(x_i + h, y_{1i} + k_{3,1}, y_{2i} + k_{3,2}, \dots, y_{mi} + k_{3,m}) \quad \{j = 1 \rightarrow m\} \\ y_{ji+1} &= y_{ji} + \frac{1}{6}(k_{1j} + 2k_{2j} + 2k_{3j} + k_{4j}) \quad \{j = 1 \rightarrow m\} \end{aligned} \quad (27)$$

Donde los subíndices " i " hacen referencia a los valores del anterior punto. Como se muestra, y como ya se vio al resolver ecuaciones diferenciales de enésimo orden, primero se calculan los valores " k_1 " de todas las ecuaciones, luego los valores " k_2 ", luego " k_3 ", " k_4 " y finalmente los valores de las variables dependientes en el nuevo punto.

27.2.6. Ejemplos: Sistemas de ecuaciones diferenciales

27.2.6.1. Ejemplo 1

Como primer ejemplo volveremos a resolver el siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned} u' - v + 2w + t^2 - 2t^{1.5} - 1.2t^{0.2} + 10 &= 0 \\ v' + 5w - 5t^{1.5} - 2t + 35 &= 0 \\ w' - 2u + v - t^2 + 2t^{1.2} - 1.5t^{0.5} + 10 &= 0 \\ \begin{cases} u = 3, v = -4, w = -7 & \text{para } t = 0 \\ u = ?, v = ?, w = ? & \text{para } t = 0.05, 0.1, 0.14, 0.2, 0.23, 0.26, 0.30, 0.33, 0.35 \end{cases} \end{aligned}$$

Donde u , v y w son las variables dependientes y t la variable independiente. Las soluciones analíticas de este sistema son: $u = t^{1.2} + 3$; $v = t^2 - 4$ y $w = t^{1.5} - 7$.

Como ya vimos en el método de Euler, cuando se resuelve un sistema de ecuaciones diferenciales se obtienen "m" vectores con los valores de las variables dependientes, con los cuales se pueden crear "m" funciones de interpolación, mismas que constituyen las soluciones del sistema.

Primero asignamos los valores conocidos a variables y calculamos el valor de "h" fijando el número de segmentos en 40:

```
t0=0;tn=0.35;u0=3;v0=-4;w0=-7;
vct={0.05,0.1,0.14,0.2,0.23,0.26,0.30,0.33,0.35};
n=40;h=(tn-t0)/n;
```

Creamos las funciones correspondientes a las derivadas primeras de las tres variables del sistema:

```
f1[t_,u_,v_,w_]:=v-2*w-t^2+2*t^1.5+1.2*t^0.2-10;
f2[t_,u_,v_,w_]:=-5*w+5*t^1.5+2*t-35;
f3[t_,u_,v_,w_]:=2*u-v+t^2-2*t^1.2+1.5*t^0.5-10;
```

Inicializamos los vectores donde guardaremos las soluciones: "vt", "vu", "vv", "vw":

```
vt=vu=vv=vw=ConstantArray[0,n+1];vt[[1]]=t0;vu[[1]]=u0;
vv[[1]]=v0;vw[[1]]=w0;
```

Calculamos los valores de los vectores de solución aplicando las ecuaciones de Runge-Kutta:

```
Do[k11=h*f1[vt[[i]],vu[[i]],vv[[i]],vw[[i]]];
k12=h*f2[vt[[i]],vu[[i]],vv[[i]],vw[[i]]];
k13=h*f3[vt[[i]],vu[[i]],vv[[i]],vw[[i]]];
k21=h*f1[vt[[i]]+h/2,vu[[i]]+k11/2,vv[[i]]+k12/2,vw[[i]]+k13/2];
k22=h*f2[vt[[i]]+h/2,vu[[i]]+k11/2,vv[[i]]+k12/2,vw[[i]]+k13/2];
k23=h*f3[vt[[i]]+h/2,vu[[i]]+k11/2,vv[[i]]+k12/2,vw[[i]]+k13/2];
k31=h*f1[vt[[i]]+h/2,vu[[i]]+k21/2,vv[[i]]+k22/2,vw[[i]]+k23/2];
k32=h*f2[vt[[i]]+h/2,vu[[i]]+k21/2,vv[[i]]+k22/2,vw[[i]]+k23/2];
k33=h*f3[vt[[i]]+h/2,vu[[i]]+k21/2,vv[[i]]+k22/2,vw[[i]]+k23/2];
k41=h*f1[vt[[i]]+h,vu[[i]]+k31,vv[[i]]+k32,vw[[i]]+k33];
k42=h*f2[vt[[i]]+h,vu[[i]]+k31,vv[[i]]+k32,vw[[i]]+k33];
k43=h*f3[vt[[i]]+h,vu[[i]]+k31,vv[[i]]+k32,vw[[i]]+k33];
vu[[i+1]]=vu[[i]]+1/6*(k11+2*k21+2*k31+k41);
vv[[i+1]]=vv[[i]]+1/6*(k12+2*k22+2*k32+k42);
vw[[i+1]]=vw[[i]]+1/6*(k13+2*k23+2*k33+k43);
vt[[i+1]]=vt[[i]]+h,{i,n}]
```

Con los vectores de solución e "Interpolation" creamos las tres funciones de interpolación:

```
fu=Interpolation[Transpose[{vt,vu}]];
fv=Interpolation[Transpose[{vt,vv}]];
fw=Interpolation[Transpose[{vt,vw}]];
```

Graficamos las tres funciones de interpolación empleando los colores rojo, azul y café:

```
g1=Plot[{fu[t],fv[t],fw[t]},{t,t0,tn},PlotStyle->
{Red,Blue,Brown}];
```

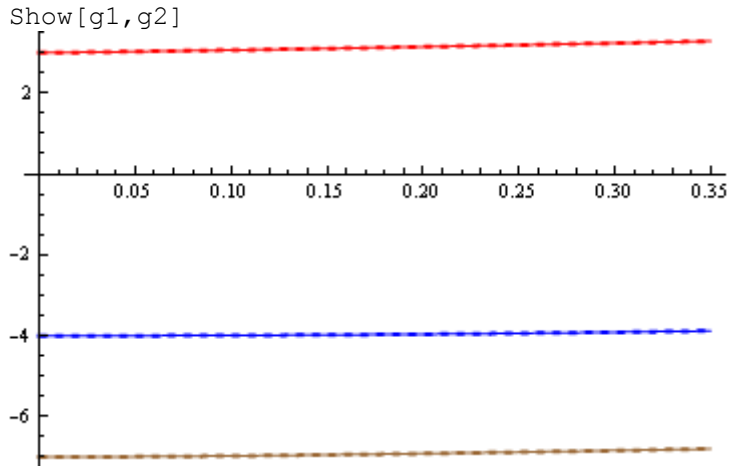
Programamos las soluciones conocidas:

```
fcu[t_]:=t^1.2+3;fcv[t_]:=t^2-4;fcw[t_]:=t^1.5-7;
```

Graficamos las soluciones en los colores rojo, azul y café con guiones gruesos:

```
g2=Plot[{fcu[t],fcv[t],fcw[t]},{t,t0,tn},PlotStyle->
  {{Red,Dashed,Thick},{Blue,Dashed,Thick},{Brown,Dashed,Thick}}}];
```

Mostramos la gráfica de las funciones de interpolación (las soluciones encontradas) conjuntamente las funciones de las soluciones conocidas:



Como se puede observar las funciones de interpolación (las soluciones encontradas) se ajustan bastante bien a las soluciones conocidas.

Finalmente calculamos los valores de las variables dependientes para los valores de "t" dados:

```
Do[Print["Para t= ",vct[[i]]];
  Print["El valor de u es: ",fu[vct[[i]]]];
  Print["El valor de v es: ",fv[vct[[i]]]];
  Print["El valor de w es: ",fw[vct[[i]]]],{i,Length[vct]}]
Para t= 0.05
El valor de u es: 3.02712
El valor de v es: -3.99749
El valor de w es: -6.98889
Para t= 0.1
El valor de u es: 3.06276
El valor de v es: -3.98996
El valor de w es: -6.96848
Para t= 0.14
El valor de u es: 3.09416
El valor de v es: -3.98034
El valor de w es: -6.94775
Para t= 0.2
El valor de u es: 3.14466
El valor de v es: -3.95989
El valor de w es: -6.91073
Para t= 0.23
El valor de u es: 3.17114
El valor de v es: -3.94697
El valor de w es: -6.88989
Para t= 0.26
El valor de u es: 3.19833
El valor de v es: -3.93223
El valor de w es: -6.86764
Para t= 0.3
El valor de u es: 3.23556
```



```

El valor de v es: -3.90979
El valor de w es: -6.83593
Para t= 0.33
El valor de u es: 3.26415
El valor de v es: -3.89085
El valor de w es: -6.8107
Para t= 0.35
El valor de u es: 3.28351
El valor de v es: -3.87722
El valor de w es: -6.79322

```

27.2.6.2. Ejemplo 2

Como segundo ejemplo volveremos a resolver el siguiente sistema de ecuaciones diferenciales:

$$\begin{aligned}\frac{dx}{dt} &= x' = 2x + 3y \\ \frac{dy}{dt} &= y' = 2x + y \\ \begin{cases} x = 2.7; y = 2 & \text{para } t = 0 \\ x = ?; y = ? & \text{para } t = 0.2, 0.5, 0.7, 0.9, 1.2, 1.3 \end{cases}\end{aligned}$$

El procedimiento es el mismo que en el anterior ejemplo, sólo que ahora se tienen 2 ecuaciones en lugar de 3 y el problema es resuelto en un módulo:

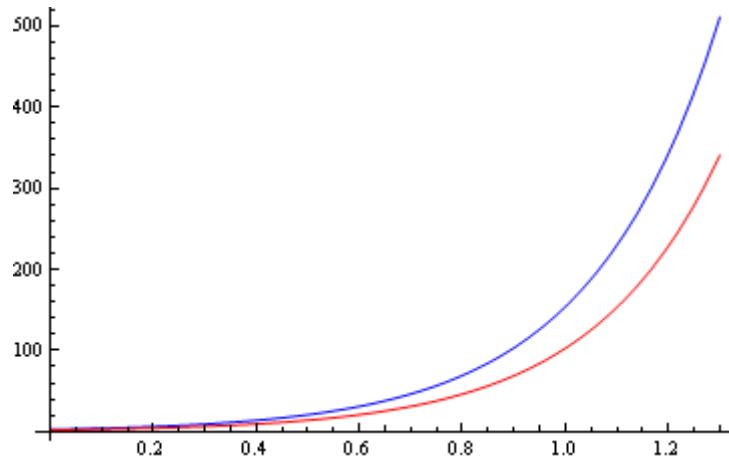
```

Module[{t0=0,tn=1.3,x0=2.7,y0=2,n=40,h,f1,f2,vt,vx,vy,fx,
  fy,k11,k12,k21,k22,k31,k32,k41,k42,
  vct={0.2,0.5,0.7,0.9,1.2,1.3}},
h=(tn-t0)/n;
f1[t_,x_,y_]:=2 x+3 y;
f2[t_,x_,y_]:=2 x+y;
vt=vx=vy=ConstantArray[0,n+1];
vt[[1]]=t0;vx[[1]]=x0;vy[[1]]=y0;
Do[k11=h*f1[vt[[i]],vx[[i]],vy[[i]]];
  k12=h*f2[vt[[i]],vx[[i]],vy[[i]]];
  k21=h*f1[vt[[i]]+h/2,vx[[i]]+k11/2,vy[[i]]+k12/2];
  k22=h*f2[vt[[i]]+h/2,vx[[i]]+k11/2,vy[[i]]+k12/2];
  k31=h*f1[vt[[i]]+h/2,vx[[i]]+k21/2,vy[[i]]+k22/2];
  k32=h*f2[vt[[i]]+h/2,vx[[i]]+k21/2,vy[[i]]+k22/2];
  k41=h*f1[vt[[i]]+h,vx[[i]]+k31,vy[[i]]+k32];
  k42=h*f2[vt[[i]]+h,vx[[i]]+k31,vy[[i]]+k32];
  vx[[i+1]]=vx[[i]]+1/6*(k11+2*k21+2*k31+k41);
  vy[[i+1]]=vy[[i]]+1/6*(k12+2*k22+2*k32+k42);
  vt[[i+1]]=vt[[i]]+h,{i,n}];
fx=Interpolation[Transpose[{vt,vx}]];
fy=Interpolation[Transpose[{vt,vy}]];
Print["Gráfica de las soluciones: "];
Print[Plot[{fx[t],fy[t]},{t,t0,tn},PlotStyle->{Blue,Red}]];
Do[Print["Para t= ",vct[[i]]];
  Print["El valor de x es: ",fx[vct[[i]]];
  Print["El valor de y es: ",fy[vct[[i]]]],{i,Length[vct]}]
]

```

Ejecutando el módulo se obtienen los siguientes resultados:

Gráfica de las soluciones:



Para $t= 0.2$

El valor de x es: 6.17775

El valor de y es: 4.28224

Para $t= 0.5$

El valor de x es: 20.7641

El valor de y es: 13.9641

Para $t= 0.7$

El valor de x es: 46.3137

El valor de y es: 30.9751

Para $t= 0.9$

El valor de x es: 103.157

El valor de y es: 68.8526

Para $t= 1.2$

El valor de x es: 342.619

El valor de y es: 228.473

Para $t= 1.3$

El valor de x es: 511.149

El valor de y es: 340.821