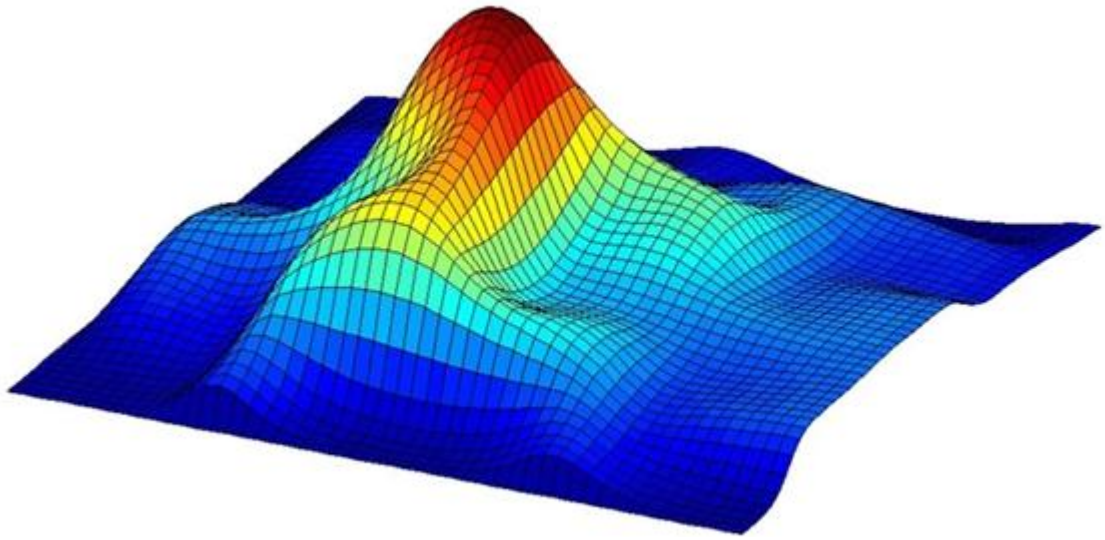
MÉTODOS NUMÉRICOS

EN JASYMCA



HERNAN PEÑARANDA V.

materias-hpv.comli.com

Sucre, 2012

1. INTRODUCCIÓN

En esta materia se estudiarán métodos para resolver problemas matemáticos complejos mediante la realización de cálculos aritméticos simples. El problema radica en el elevado número de cálculos necesarios por lo que sólo son de utilidad práctica si se aplican con la ayuda de una computadora o dispositivo programable.

En el ámbito de las computadoras se cuenta con una gran variedad de lenguajes y software que permiten automatizar los procesos de cálculo, sin embargo sólo un reducido porcentaje de estudiantes cuentan con una, por lo que no es posible emplear dicha herramienta para el desarrollo de las clases. Por ello se ha visto la necesidad de recurrir a otros medios no convencionales para la enseñanza de la materia. En ese sentido y tomando en cuenta que la mayoría de los estudiantes cuentan con un teléfono móvil, se ha optado por emplear dicha herramienta para la enseñanza de la materia.

Si bien es cierto que los teléfonos móviles han sido creados con fines muy diferentes a los del cálculo numérico, en realidad son micro-computadoras que, en muchos casos, cuentan con capacidades de almacenamiento y procesamiento superiores inclusive a los de las computadoras personales de hace no más de 10 años y en ese sentido pueden ser empleados adecuadamente en la resolución de problemas numéricos.

En este ámbito, el de los teléfonos y dispositivos móviles, se cuenta ya con aplicaciones orientadas al cálculo numérico, de ellas se ha elegido Jasymca, una aplicación con un lenguaje compatible con Matlab y Octave, así como el de Scilab y otras aplicaciones empleadas en el ámbito de las computadoras personales y que puede ser ejecutado también en una computadora.

1.1. INTRODUCCIÓN A JASYMCA

Jasymca es una aplicación libre y como tal puede ser bajada gratuitamente del sitio: http://webuser.hs-furtwangen.de/~dersch/. Cuenta con un ambiente desarrollo básico que permite ejecutar todas las funciones y operaciones incluidas, así como crear otras nuevas.

Al descomprimir Jasymca, en su interior se tienen tres carpetas: “Midlet”, “Applikation” y “Applet”, así como los manuales de la aplicación en inglés y alemán.

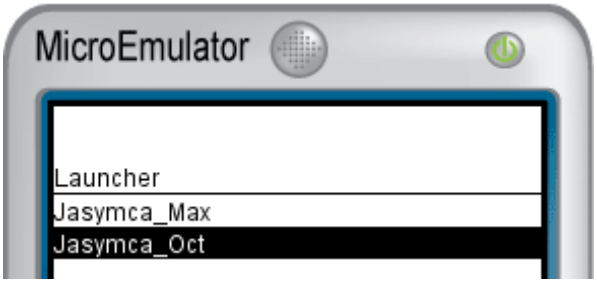
En los celulares se debe instalar el archivo “Jasymca.jar” que se encuentra en el directorio “Midlet” (algunos modelos requieren además el archivo “jasymca.jad” que también se encuentra en dicho directorio).

En el caso de las computadoras, se debe ejecutar el archivo “Jasymca.jar” que se encuentra en el directorio “Applikation” (instrucción DOS: java –jar jasymca.jar).

La carpeta “Applet” contiene la página “index.html” que puede ser abiertas en cualquier navegador y que permite utilizar Jasymca desde el navegador (no se recomienda emplear esta versión porque, por razones de seguridad, los applets no permiten guardar ni recuperar archivos).

Existe también una versión de Jasymca para el sistema operativo Android, el mismo que debe ser bajado por separado.

Al ingresar a Jasymca desde un celular se debe elegir uno de los dos modos en los que puede operar Jasymca: Octave y Maxima. En este curso emplearemos el modo Octave:



Una vez en Jasymca, se pueden comenzar a escribir órdenes en la ventana inferior (Input). En las computadoras por defecto ingresa en el modo Octave, pero dicho modo puede ser cambiado también a través del menú "Run".

Se puede ver a Jasymca como una potente calculadora que conoce todas las operaciones básicas (incluida la potenciación "^"). Conoce también la mayoría de las funciones trigonométricas, exponenciales y logarítmicas, por lo que es posible evaluar directamente expresiones matemáticas, desde las más simples hasta las más complejas.

1.1.1. Ejemplos

1. Calcule la suma de dos má dos.

En el campo de entrada se escribe:

```
2+2
```

Y se pulsa la tecla u opción Enter, con lo que se obtiene:

```
>> 2+2
ans = 4.0
```

2. Calcule el valor de la siguiente expresión.

$$\sqrt{\frac{\operatorname{sen}(9.2)-\cos(8.32)}{\tan(6.89)}}$$

En el campo de entrada se escribe:

```
sqrt((sin(9.2)-cos(8.32))/tan(6.89))
```

Con lo que se obtiene:

```
>> sqrt((sin(9.2)-cos(8.32))/tan(6.89))
ans = 0.98405
```

Como se puede ver en este ejemplo, para agrupar expresiones (en este caso separar el numerador del denominador) se emplean paréntesis.

3. Calcule el valor de la siguiente expresión mostrando todos los dígitos de la solución.

$$\ln\left(\csc(7)-\sec(9.2)+\tan(0.4)\right)^{9.3}$$

En el campo de entrada se escribe:

```
log(csc(7)-sec(9.2)+tan(0.4))^9.3
```

Con lo que se obtiene:

```
>> log(csc(7)-sec(9.2)+tan(0.4))^9.3
ans = 2.206
```

Como se puede observar, por defecto Jasymca muestra los resultados con 5 dígitos de precisión. Para ver los resultados con todos los dígitos se debe ejecutar la instrucción "format long":

Instrucción que no genera ningún resultado:

```
>> format long
```

Pero si ahora se vuelve a ejecutar la anterior instrucción se obtiene:

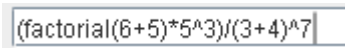
```
>> log(csc(7)-sec(9.2)+tan(0.4))^9.3
ans = 2.206014639244216
```

Aun cuando sólo se vean parte de estos dígitos, Jasymca siempre trabaja con todos los dígitos en las operaciones internas.

4. Calcule el resultado aproximado y exacto de la siguiente expresión

$$\frac{(6+5)!\cdot 5^3}{(3+4)^7}$$

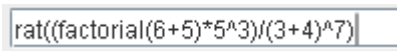
Se escribe lo siguiente en el campo de entrada:



con lo que se obtiene la solución aproximada:

```
>> (factorial(6+5)*5^3)/(3+4)^7
ans = 6058.700031449481
```

Para trabajar con números exactos (precisión infinita) se emplea la instrucción "rat":



Con lo que se obtiene:

```
>> rat((factorial(6+5)*5^3)/(3+4)^7)
ans = 60587/10
```

5. Calcule con todos los dígitos el resultado aproximado de la siguiente expresión.

$$\cos\left(\frac{\pi}{2}\right)$$

Con algunas constantes, como el número "pi", Jasymca mantiene por defecto la constante en forma simbólica, así si se escribe:

Se obtiene:

```
>> cos(pi/2)
ans = cos(0.5*pi)
```

Donde como se puede observar "pi" se mantiene como una constante. Para obtener un resultado numérico se debe emplea la instrucción "float":

Con lo que se obtiene:

```
>> cos(float(pi)/2)
ans = 6.123233995736766E-17
```

6. Calcule el valor numérico de la siguiente expresión con todos sus dígitos.

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

En el campo de entrada (estando en formato largo) se escribe:

```
(((3+factorial(4))+6^3.2)/(exp(6.5)+cos(6.7*float(pi)/180)))^(1/3)
```

Con lo que se obtiene:

```
>> (((3+factorial(4))+6^3.2)/(exp(6.5)+cos(6.7*float(pi)/180)))^(1/3)
ans = 0.7960937169762573
```

7. Calcule el cociente y el residuo de la siguiente expresión mostrando el resultado con 7 dígitos de precisión.

$$\frac{(3(1+3))!}{4^{11}}$$

Para mostrar el resultado con un número de dígitos arbitrario y en una base numérica determinada, se escribe "format base,N°_de_dígitos" o "format base N°_de_dígitos". Así para mostrar el resultado con 7 dígitos, siendo la base 10 (decimal), se escribe:

```
format 10,7
```

Que al igual que "format long" no produce ningún resultado:

```
>> format 10,7
```

Pero a partir de esta instrucción todos los resultados se muestran con 7 dígitos de precisión.

Para calcular el cociente y el residuo de una división se emplea la función "divide(numerador,denominador)", así el cociente y el residuo de la expresión es:

```
>> divide(factorial(3*(1+3)),4^11)

ans = [ 114  850944 ]
```

Como se puede ver, el resultado es un vector con dos elementos, siendo el primer elemento el cociente y el segundo el residuo. En Jasymca, para acceder a los elementos de un vector (o matriz), se escribe el índice del elemento entre paréntesis, siendo el índice del primer elemento el número 1, así para obtener sólo el cociente se escribe:

```
>> divide(factorial(3*(1+3)),4^11)(1)

ans = 114
```

Y para obtener sólo el residuo:

```
>> divide(factorial(3*(1+3)),4^11)(2)

ans = 850944
```

8. Calcule el resultado de la siguiente expresión mostrando el resultado con 12 dígitos de precisión.

$$\frac{\log(3+\sqrt{2.1+4.9})}{\sqrt[5]{7.2+2^{2.12}}}$$

Como se puede ver, en esta expresión se tiene que calcular el logaritmo en base 10, pero Jasymca sólo cuenta con la función para calcular el logaritmo natural de un número ("log"), no obstante, como se sabe por las propiedades de los logaritmos, el logaritmo en cualquier base "c" puede ser calculado si se conoce el logaritmo en cualquier otra base "b":

$$\log_c(x)=\frac{\log_b(x)}{\log_b(c)}$$

Y como se conoce el logaritmo natural (cuya base es el número de Nepper "e"), es posible calcular el logaritmo en cualquier otra base:

```
format 10,12
```

```
>> format 10,12
```

```
log(3+sqrt(2.1+4.9))/log(10)/(7.2+2^2.12)^(1/5)
```

```
>> log(3+sqrt(2.1+4.9))/log(10)/(7.2+2^2.12)^(1/5)
ans = 0.460854396527
```

1.1.2. Ejercicios

1. Calcule el valor de la siguiente expresión mostrando el resultado con 5 dígitos de precisión:

$$\frac{(4+5)(3.2*4.1)}{52.3+7.8-9.1}$$

2. Calcule el valor de la siguiente expresión mostrando el resultado con 10 dígitos de precisión:

$$\ln(6.573)e^{7.8234}$$

3. Calcule el valor de la siguiente expresión mostrando el resultado con 7 dígitos de precisión:

$$\sqrt{\frac{\log(6.75)}{6.2+1.3^{2.7}}}\cos(5.2)$$

4. Calcule el valor de la siguiente expresión mostrando el resultado con 14 dígitos de precisión:

$$\sin^{-1}\left(\frac{\sqrt{3.2}+4.67}{8.43}\right)$$

5. Calcule el valor de la siguiente expresión mostrando el resultado con todos los dígitos:

$$\ln(6.7)+\log(4.3)+\log_2(2.1)$$

6. Calcule el valor exacto de la siguiente expresión:

$$\sqrt[3]{50!4^7}$$

7. Calcule el valor aproximado (mostrando el resultado con todos los dígitos) y exacto de la siguiente expresión:

$$\sqrt{\frac{5^3-2^2+3!}{(7-3)^2}}$$

8. Calcule el valor de la siguiente expresión mostrando el resultado con 10 dígitos de precisión:

$$\left(\sin(9.2)-\frac{\sec(5.6)}{\sinh(9.2)}\right)^{0.34}$$

9. Calcule, por separado, el cociente y el residuo de la siguiente expresión:

$$\frac{5!+3^4}{4^3}$$

10. Calcule el valor aproximado de la siguiente expresión mostrando el resultado con 9 dígitos de precisión:

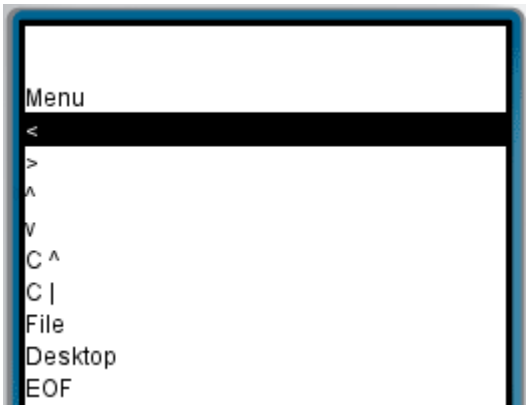
$$\log\left(\frac{4.5^{3.2}+9.8^{1.6}+e^{4.5}}{\sqrt[5]{6.32+2^{0.98}}}\right)$$

11. Calcule el valor aproximado de la siguiente expresión mostrando el resultado con 11 dígitos de precisión:

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

Jasymca no cuenta con la función hiperbólica de la tangente (ni la función inversa respectiva), sin embargo, como se verá en el siguiente acápite, dichas funciones pueden ser añadidas fácilmente.

En cuanto a la navegabilidad en los celulares se tienen las siguientes opciones en el menú de Jasymca:



La primera opción "<" permite recuperar el comando anterior, la segundo ">" el comando posterior, la tercera "^" sube la pantalla de resultados una fila, la cuarta "v" baja la pantalla de resultados una fila, "C^" inserta el símbolo de potenciación "^", "C|" inserta el símbolo del operador lógico "o", "file" ingresa a la ventana de manejo de archivos, "Desktop" permite ver el escritorio de Jasymca (que es donde se encuentran las gráficas) y "EOF" añade el carácter de fin de archivo.

No todos los celulares muestran las opciones de la figura, dependiendo del modelo algunas de dichas opciones son accesibles directamente desde teclado.

En cuanto a la versión de computadora, se cuenta también con las opciones "<" y ">" y la ventana de resultados tiene barras de desplazamiento. Además se tiene un menú para el manejo de archivos, ejecución y visualización.

1.2. AÑADIR FUNCIONES A JASYMCA

Como ya se dijo, Jasymca no cuenta con algunas funciones matemáticas, sin embargo las mismas pueden ser añadidas fácilmente. Para ello es necesario crear una función. La estructura de una función en Jasymca es la siguiente:

```
function variable_resultante=nombre_de_la_función(lista_de_parámetros)
  instrucciones;
  variable_resultante = valor_devuelto;
end
```

Las funciones en Jasymca devuelven como resultado el valor de la "variable_resultante", por eso, dicho valor debe ser asignado en alguna parte dentro de la función, frecuentemente (pero no obligatoriamente) al final.

No es aconsejable emplear nombres demasiado cortos (de una o dos letras) para nombrar a las funciones, pues los mismos pueden causar problemas al "confundirse" con las variables de los programas.

Aun cuando las funciones pueden ser creadas en Jasymca directamente en memoria (en la ventana "Input"), lo aconsejable es guardarlas en archivos, así pueden ser reutilizadas en un futuro y pueden ser corregidas y modificadas según sea necesario.

1.2.1. Ejemplos

9. Elabore una función para calcular el cubo de un número, pruebe la función calculando el cubo de 3 y de 7.9.

Lo único que debe hacer esta función es multiplicar tres veces el número recibido. Como se dijo, lo recomendable es programar la función en un archivo, para ello en los celulares se debe seguir el camino: Options → File → vfs/ → Options → New → Regular File y escribir el nombre del archivo, en este caso "cubo.m" (**todos los archivos deben tener la extensión "m"**)

En las computadoras se debe ingresar al directorio "m", dentro de applikation (si dicho directorio no existe, debe ser creado) y dentro del directorio seguir el camino: clic derecho → nuevo → documento de texto, nombrando al archivo como "cubo.m" (siempre con la extensión "m").

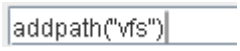
Una vez creado el archivo, se abre el mismo (tanto en el celular como en la computadora) y se escribe el código en su interior, para este ejemplo el código es:

```
function r=cubo(x)
  r=x*x*x;
end
```

Donde "x" es el parámetro donde se recibe el número y "r" es la variable donde se guarda el resultado de la multiplicación, siendo el nombre de la función "cubo".

Una vez escrito el programa se guarda (Save) y se sale del archivo (Exit) (en las computadoras se guarda el archivo y se vuelve a Jasymca).

Para emplear la función creada, en los celulares se debe ejecutar la instrucción addpath("vfs"):



Instrucción que no devuelve ningún resultado:

```
>> addpath("vfs")
```

Para verificar que el directorio "vfs" ha sido incluido en el camino, se puede ejecutar la instrucción "path":

Con lo que se obtiene:

```
>> path
vfs:m:.
```

Como se puede ver, ahora el directorio "vfs" está incluido en el camino (path), donde por defecto aparece también el directorio "m", esta es la razón por la cual en el caso de las computadoras no es necesario ejecutar la instrucción "addpath", pues el directorio "m" ya está incluido en el camino.

En los celulares, para tener acceso a los programas elaborados, la instrucción "addpath("vfs")" debe ser ejecutada una sola vez, cada vez que se ingresa a Jasymca.

Una vez añadido el directorio "vfs" al camino ("path") se pueden utilizar las funciones elaboradas.

A partir de este ejemplo, no se mostrarán las instrucciones escritas en la ventana de entrada, sino directamente los resultados obtenidos en la ventana de salida, esto porque en la salida aparecen también las instrucciones escritas para obtener los resultados.

Ahora que la función "cubo" ha sido creada y el directorio "vfs" añadido al camino, se utiliza la función cubo como cualquier otra función propia de Jasymca. Así para calcular el cubo de 3 y de 7.9 se escribe:

```
>> cubo(3)
ans = 27
>> cubo(7.9)
ans = 493.039
```

10. Elabore una función para calcular la tangente hiperbólica de un número. Luego emplee la función para calcular el valor de la siguiente expresión mostrando el resultado con 12 dígitos de precisión.

$$\sqrt[5]{\frac{\sinh(2.3)-\cosh(1.2)}{3.4\tanh(2.3)}}$$

Para calcular la tangente hiperbólica, simplemente se divide el seno hiperbólico entre el coseno hiperbólico, por lo tanto la función (que debe ser guardada en el directorio "vfs" con el nombre "tanh.m" es):

```
function r=tanh(x)
  y=sinh(x)/cosh(x);
end
```

Una vez creada y guardada la función (y habiendo ejecutado previamente la instrucción "addpath("vfs")") se puede calcular el valor de la expresión:

```
>> format 10 12
>> ((sinh(2.3)-cosh(1.2))/(3.4*tanh(2.3)))^(1/5)
ans = 0.987317333369
```

11. Elabore la función "rad" para convertir un ángulo de grados a radianes. Luego emplee la función elaborada para calcular el valor de la siguiente expresión mostrando el resultado con 7 dígitos de precisión.

$$\log\left(\frac{\sin(135°)+\cos(235°)}{3.4\tan(357°)}\right)$$

Para convertir un ángulo de grados a radianes se divide el ángulo entre 180 grados y se multiplica por π radianes, por lo tanto la función es:

```
function r=rad(x)
  r=x*float(pi)/180;
end
```

Una vez creada y guardada la función se puede calcular el valor de la expresión:

```
>> format 10 7
>> log((sin(rad(135))+cos(rad(235)))/(3.4*tan(rad(770))))/log(10)
ans = -1.482085
```

1.2.2. Ejercicios

12. Cree la función "deg" para convertir un ángulo de radianes a grado, luego emplee la función creada para convertir 5.67 y 14.23 radianes a grados, mostrando el resultado con 10 dígitos de precisión.

13. Cree la función "cot" para calcular la cotangente de un ángulo en radianes, luego emplee la función creada para calcular el valor de la siguiente expresión, mostrando el resultado con 7 dígitos de precisión.

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

14. Cree la función "log10" para calcular el logaritmo en base 10 de un número, luego emplee la función creada para calcular el valor de la siguiente expresión, mostrando el resultado con 8 dígitos de precisión.

$$\sqrt{\frac{\log(7.75)}{6.2+3^{1.7}}}\cos(30.2°)$$

15. Cree la función "cbrt" para calcular la raíz cúbica de un número, luego emplee la función creada para calcular el valor de la siguiente expresión, mostrando el resultado con todos los dígitos. Tome en cuenta que $\sqrt[3]{-x}=-\sqrt[3]{x}$, que el valor absoluto de un número "x" se calcula con la función "abs(x)" y que el signo de un número "x" (-1 o 1) se obtiene con "sign(x)".

$$\sqrt[3]{\frac{5^4-7!}{4^3+2^3}}$$

16. El arcotangente hiperbólico puede ser calculado con la expresión:

$$\tanh^{-1}(x)=\frac{1}{2}\ln\left(\frac{1+x}{1-x}\right)$$

Cree la función "atanh" para calcular el arcotangente hiperbólico, luego emplee la función creada para calcular el valor de la siguiente expresión, mostrando el resultado con 14 dígitos de precisión.

$$\frac{\sinh^{-1}(12.3)+\cosh^{-1}(10.2)+\tanh^{-1}(0.34)}{45.2+3.8^{4.9}+6!}$$

2. ESTRUCTURAS ESTÁNDAR 1

En este tema comienza el repaso de las estructuras empleadas en la programación estructurada y la manera en que se implementan en Jasymca.

2.1. SECUENCIA

El repaso comienza con la estructura básica que forma parte de todo programa: la secuencia, que simplemente es una sucesión de instrucciones separadas con comas o puntos y comas:

instrucción_1,
instrucción_2,
…
instrucción_n

o

instrucción_1;
instrucción_2;
…
instrucción_n

Si se emplean comas, Jasymca muestra los resultados de cada instrucción en pantalla, si se emplean puntos y comas las instrucciones se ejecutan pero los resultados no se muestran en pantalla. Por lo general, cuando se elaboran programas, no se quiere mostrar los resultados intermedios. Por esta razón, en esta materia, casi siempre las instrucciones estarán separadas por puntos y comas (;).

En la lógica para resolver un problema secuencial, básicamente lo único que se debe hacer es ordenar las instrucciones de manera que se ejecuten en el orden lógico correcto. Por ejemplo, si se quiere calcular el valor de la variable "w" con las siguientes expresiones y sólo se tiene como dato el valor de "x":

$$w = y^3 + z^2$$
$$z = 3x^2 - \sqrt{(y)}$$
$$y = 5x - \log(x)$$

Primero se tiene que calcular el valor de "y" (pues sólo depende de "x"), luego el valor de "z" (que depende además de "y") y finalmente el valor de "w" que (depende de "y" y "z"). Si se intenta programar las expresiones en el orden en que aparecen se produciría un error pues las variables "y" y "z" no tendrían ningún valor (estarían indefinidas).

Normalmente las secuencias en Jasymca se encierran entre la palabra que identifica la estructura y la palabra "end", como ocurre con las funciones (o módulos) que se han estudiado en el anterior tema. En esta materia Se considerará que se trata de una secuencia inclusive si la misma está formada por una sola instrucción.

2.2. GRÁFICA DE FUNCIONES EN JASYMCA

Jasymca permite graficar funciones de una sola variable en dos dimensiones. Para ello se emplea la instrucción "plot()":

plot(x,y,opciones)

Donde "x" es un vector con los valores del eje "x", "y" un vector con los respectivos valores del eje "y" y "opciones" es un parámetro opcional de

tipo texto, donde se puede establecer por ejemplo el color de la curva resultante ("r" = rojo (**r**ed), "g" = verde (**g**reen), "b" = azul (**b**lue), "y" = amarillo (**y**ellow), "m" = **m**agenta, "c" = **c**ian, "w" = blanco (**w**hite), "k" = negro (blac**k**)). Igualmente es posible mostrar los puntos que conforman la gráfica con algún símbolo: "x", "+", "*" y "o".

Estudiaremos los vectores y matrices con mayor profundidad en temas posteriores, por ahora es suficiente saber que un vector puede ser formado si se le da el límite inferior, el incremento y el límite superior separados con dos puntos (:), por ejemplo con la siguiente instrucción:

```
>> x=0:0.2:2*float(pi)
x = [ 0  0.2  0.4  0.6  0.8  1  1.2  1.4  1.6  1.8  2  2.2  2.4  2.6  2.8
3  3.2  3.4  3.6  3.8  4  4.2  4.4  4.6  4.8  5  5.2  5.4  5.6  5.8  6
6.2 ]
```

Como se ve, se forma un vector que va desde cero hasta "pi", con incrementos de 0.2 en 0.2.

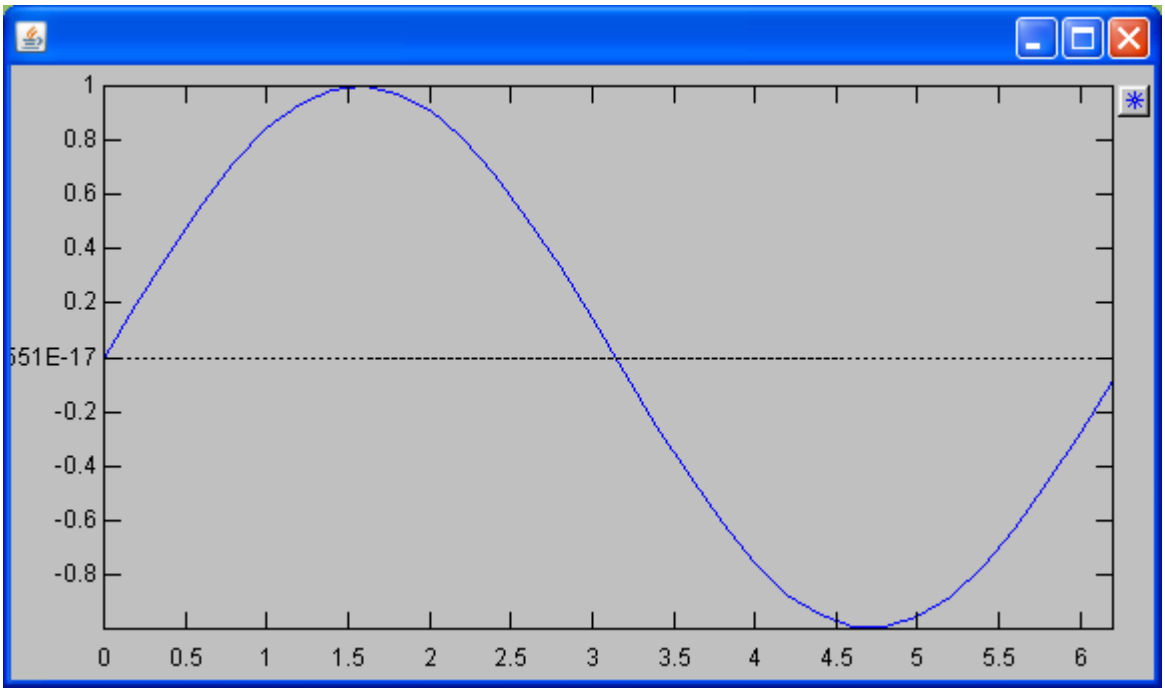
Una característica de la mayoría de las funciones de Jasymca es que pueden trabajar tanto con datos simples como con vectores, así si se manda el vector "x" a la función seno ("sin") se obtiene:

```
>> y=sin(x)
y = [ 0  0.19867  0.38942  0.56464  0.71736  0.84147  0.93204  0.98545
0.99957  0.97385  0.9093  0.8085  0.67546  0.5155  0.33499  0.14112
-5.8374E-2  -0.25554  -0.44252  -0.61186  -0.7568  -0.87158  -0.9516
-0.99369  -0.99616  -0.95892  -0.88345  -0.77276  -0.63127  -0.4646
-0.27942  -8.3089E-2 ]
```

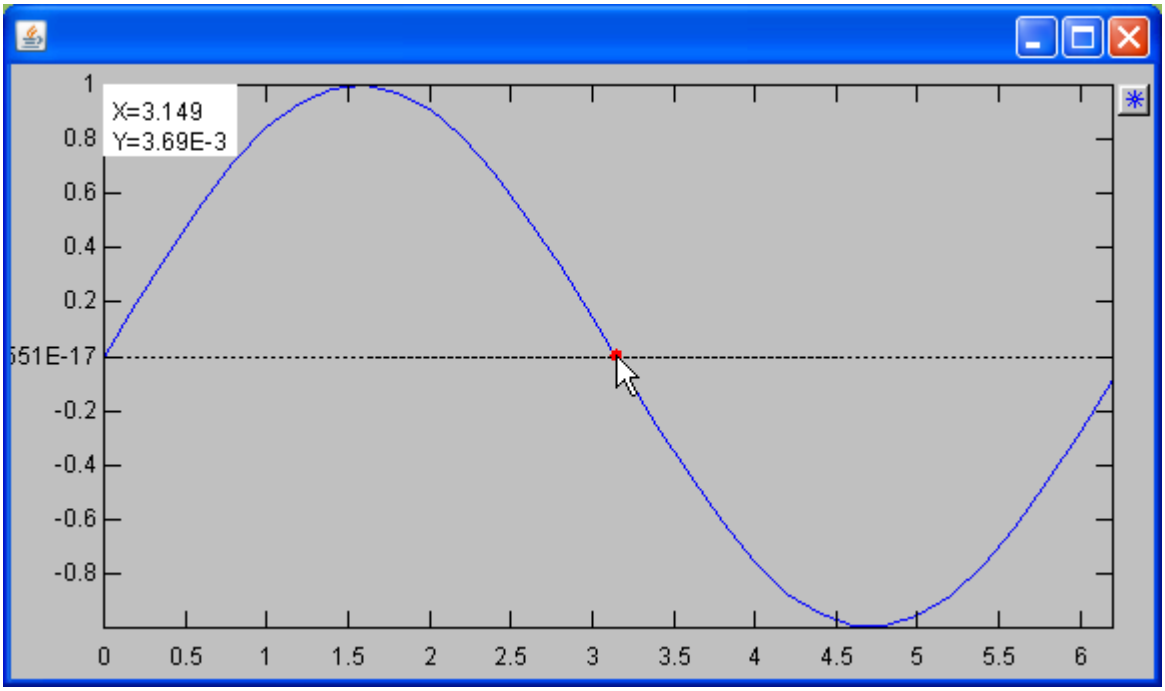
Que, como se ve, es el vector con los senos de los elementos del vector "x".

Una vez que se tienen los vectores "x" y "y" se puede elaborar la gráfica de la función:

```
>> plot(x,y)
```



Si se hace clic en un punto de la gráfica se obtienen las coordenadas:



En los celulares (dependiendo del tamaño de la pantalla) puede que sólo aparezca una parte de la gráfica, para intercambiar entre el modo normal (tamaño estándar, normalmente sólo se ve una parte de la gráfica), ajustado a la pantalla y el modo minimizado, se debe presionar repetidamente la tecla 3. Estando en el modo normal, se puede presionar repetidamente la tecla 7 para mover la gráfica de izquierda a derecha (cuando la gráfica desaparece en el lado derecho, reaparece en el izquierdo), igualmente, se puede mover la gráfica de arriba hacia abajo presionando repetidamente la tecla 9 (cuando la gráfica desaparece en la parte inferior reaparece en la parte superior).

Para ver las coordenadas de un punto se presiona el botón del joystick y para mover dicho punto las teclas de navegación del joystick.

Para salir de la pantalla gráfica (desktop) y volver a Jasymca, se pulsa la tecla 1 y se elige la opción "exit" (salir). Si sólo se quiere cerrar la gráfica (sin volver a Jasymca) se pulsa la tecla 1 y se elige la opción "close". La opción "open" debería permitir abrir una gráfica guardada previamente, pero en las versiones actuales es posible guardar gráficas en los celulares.

Si se ha salido del modo gráfico a Jasymca (sin cerrar la gráfica) es posible volver a verla seleccionando "Desktop" en el menú "Opcion." de Jasymca.

Como se dijo, es posible elegir un color para la línea de la gráfica, así por ejemplo para graficar la curva en color verde se escribe:

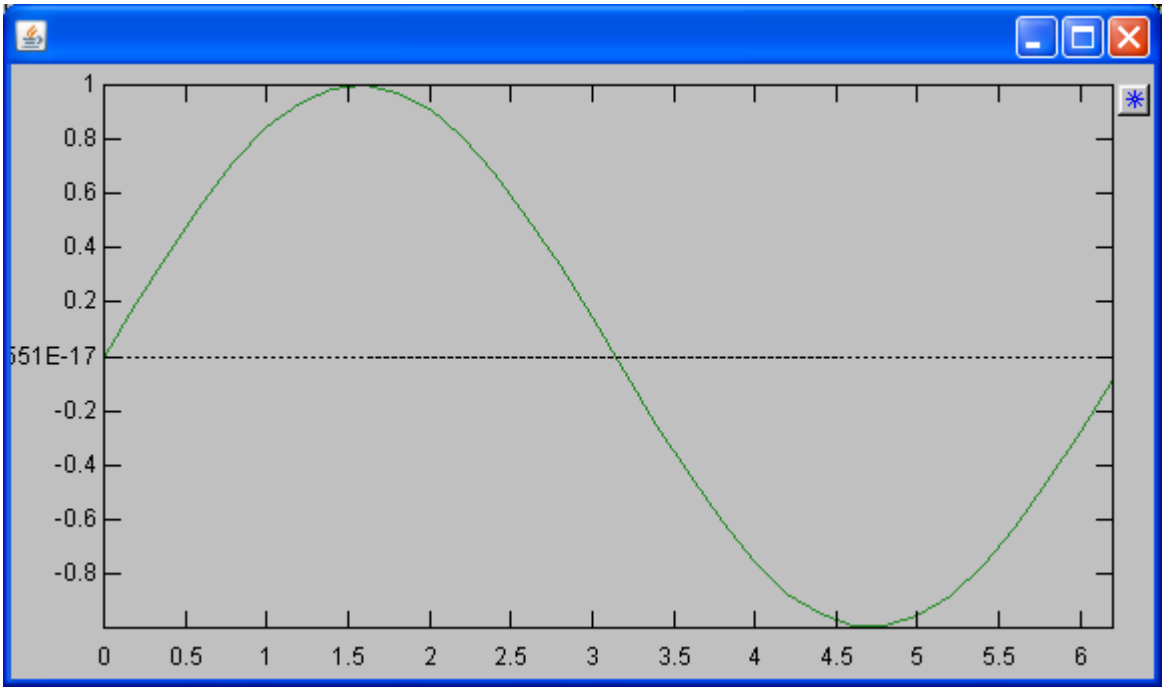
```
>> plot(x,y,"g")
```

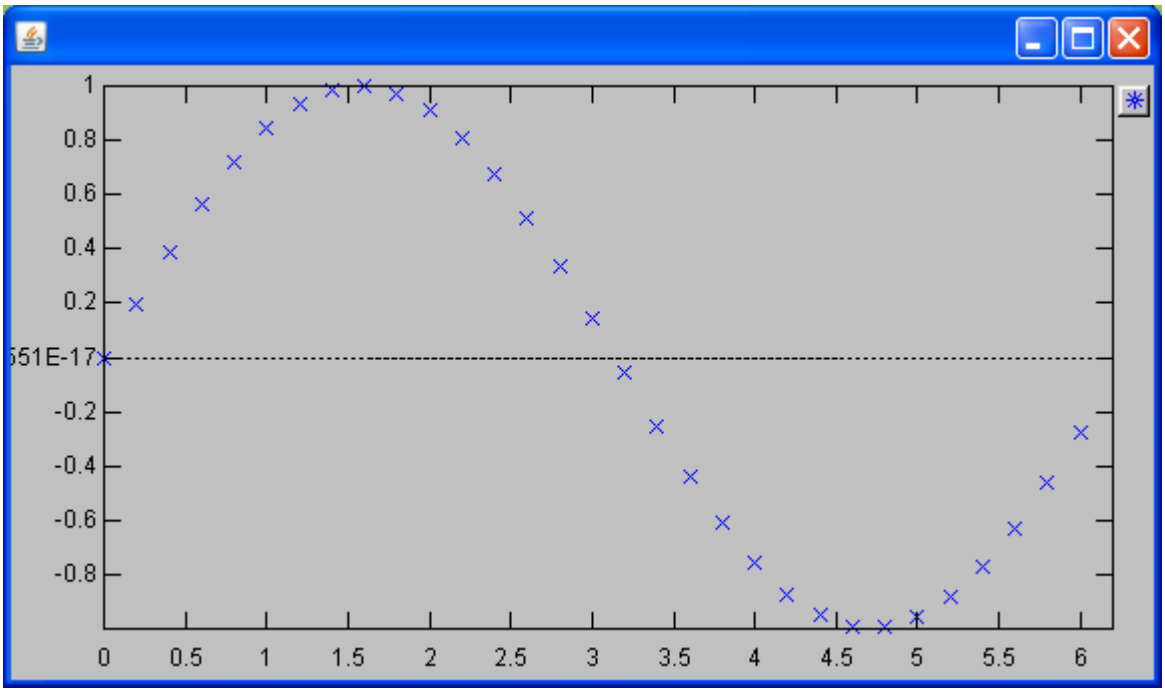
Con lo que se obtiene la misma gráfica pero estando la curva en color verde (tal como se muestra en la figura de la siguiente página).

Es posible también ver los puntos que conforman la gráfica. Por ejemplo para ver los puntos en forma de "x" se escribe:

```
>> plot(x,y,"x")
```

Con lo que se obtiene la gráfica que se muestra en la siguiente página.





Es posible también elegir un color para los puntos, así con la siguiente instrucción se obtiene la misma gráfica pero estando los puntos en color rojo:

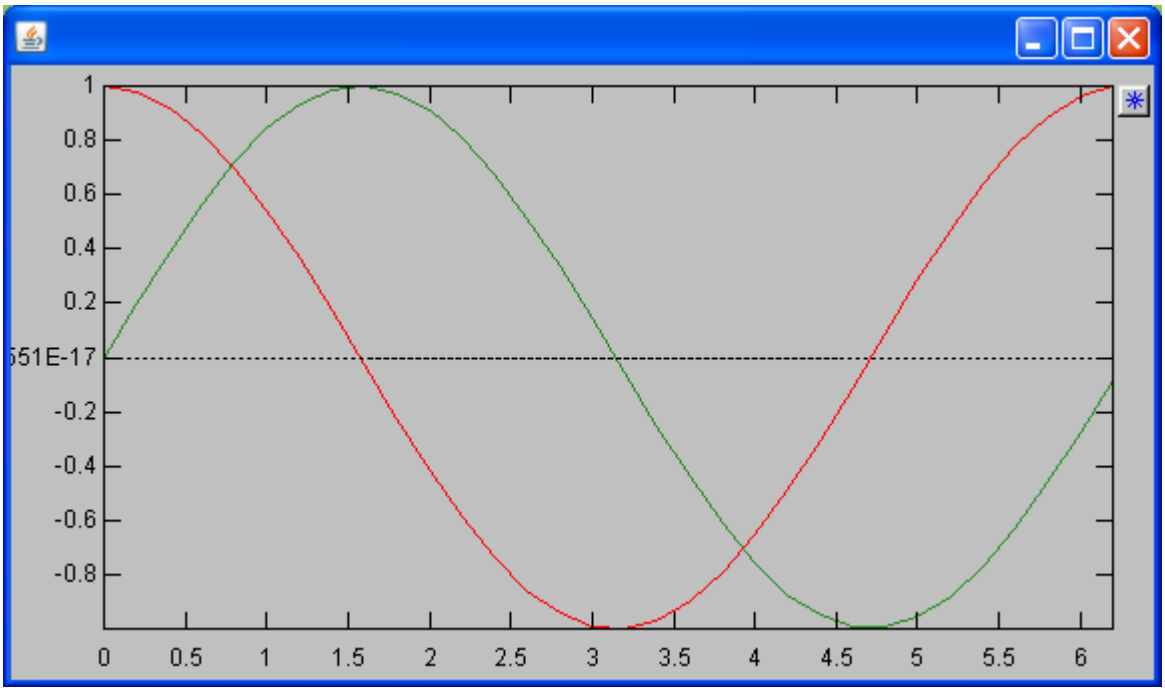
```
>> plot(x,y,"xr")
```

Finalmente, en Jasymca se pueden combinar dos o más gráficas en una sola empleando la instrucción "hold on", lamentablemente esta opción no funciona en los celulares, por lo que su uso está restringido a las computadoras portátiles y algunos otros dispositivos móviles.

Por ejemplo para graficar a la vez las funciones seno y coseno se escribe:

```
>> x=0:0.2:2*float(pi); y1=sin(x);  plot(x,y1,"g")
>> hold on; y2=cos(x); plot(x,y2,"r")
```

Donde se han empleado puntos y comas (;) para separar las instrucciones y no ver así los resultados intermedios. La gráfica resultante es:



Para volver al modo normal (una gráfica a la vez) se escribe la instrucción "hold off" o simplemente "hold". Si sólo se escribe hold el modo cambia del modo combinado al modo normal o viceversa, es decir si se encuentra en el modo combinado (held) pasa al modo simple y si se encuentra en el modo simple(released) pasa al modo combinado.

Para que la gráfica se vea lo más continua posible, al generar el vector "x" se debe emplear un incremento pequeño. No obstante si el incremento es demasiado pequeño la creación de la gráfica (sobre todo en celulares), puede demorar cierto tiempo.

2.3. EJEMPLOS

1. Elabore una función que calcule el valor de la siguiente expresión. Muestre los resultados con 9 dígitos de precisión y elabore la gráfica de la función entre x=-5 y x=5.

$$f(x)=x^3+2x^2+3x+4=0$$

Primero se crea la función (como en el anterior tema), siendo el nombre de la misma "fx1_1.m"

```
function r=fx1_1(x)
  r=x^3+2*x^2+3*x+4;
end
```

Empleando la función con algunos valores de prueba (recuerde que si aún no lo ha hecho debe ejecutar antes la instrucción "addpath("vfs")") se obtiene:

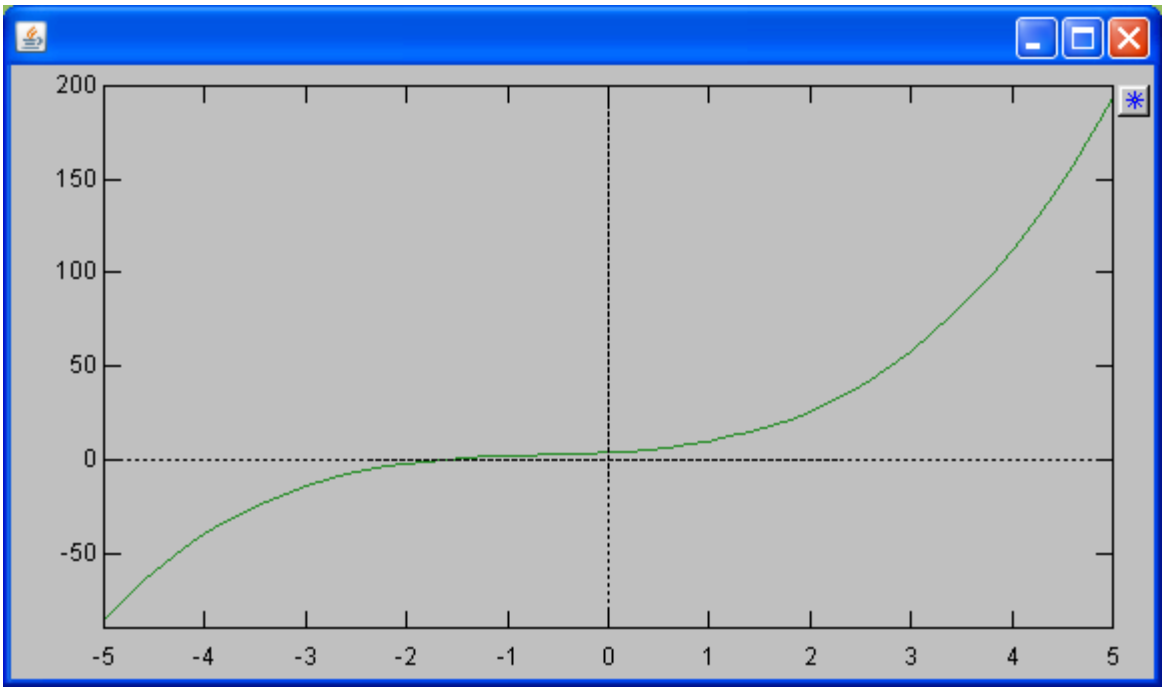
```
>> format 10,9
>> fx1_1(3.2)
```

```
ans = 66.848
>> fx1_1(-0.34)
ans = 3.171896
```

Ahora se elabora la gráfica, creando el vector "x" con un incremento igual a 0.2:

```
>> x=-5:0.2:5; y=fx1_1(x); plot(x,y,"g")
```

Con lo que se obtiene:



Como se verá posteriormente, la intersección de la gráfica con el eje "x" constituye una solución de la función y el valor aproximado de dicha solución puede ser obtenido de la gráfica elaborada.

2. Elabore una función que calcule el valor de la siguiente expresión. Muestre los resultados con 12 dígitos de precisión y elabore la gráfica de la función entre x=2 y x=4.

$$f(x)=x+y^2+z^2-40=0$$

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

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

En este caso se tiene una secuencia verdadera pues existen tres ecuaciones. Entonces se tiene que pensar en el orden en que se evaluarán las mismas. Puesto que el dato conocido es "x", y "y" sólo depende de "x", primero se calcula el valor de dicha variable, luego con "y" se puede calcular el valor de "z" (pues sólo depende de "y"), finalmente se puede calcular el valor de la función ("f(x)") pues ya se tienen todas las variables de las que depende la misma.

Con las anteriores consideraciones se programa la función, que en este caso tendrá el nombre "fx1_2.m":

```
function r=fx1_2(x)
```

```
  y=(3*x^2.1-7)/5;
  z=(14.3-y^1.2)/4;
  r=x+y^2+z^2-40;
end
```

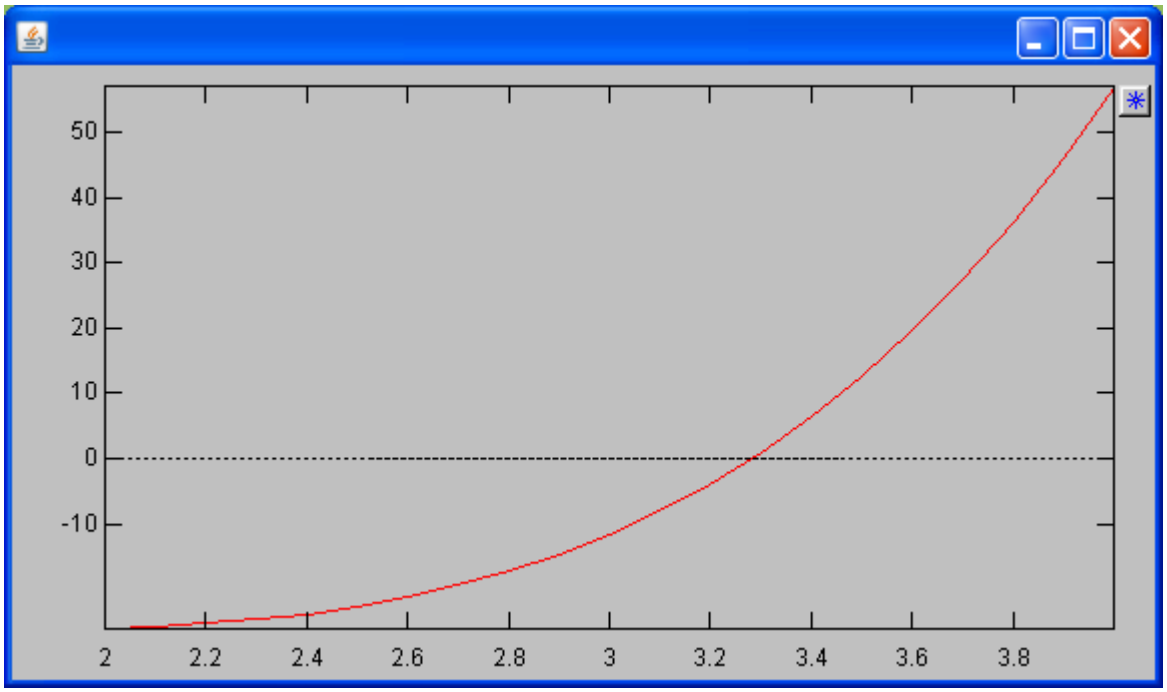
Empleando la función con algunos valores de prueba, se obtiene:

```
>> format 10,12
>> fx1_2(2.1)
ans = -25.6564167357
>> fx1_2(6.7)
ans = 1081.07356839
```

Ahora se elabora la gráfica empleando un incremento igual a 0.1 para "x".

```
>> x2=2:0.1:4; y2=fx1_2(x2); plot(x2,y2,"r")
```

Con lo que se obtiene:



Como se puede ver en este caso la función tiene una solución entre 3.2 y 3.4.

3. Elabore una función que calcule el valor de la siguiente expresión. Muestre los resultados con 10 dígitos de precisión y elabore la gráfica de la función entre x_1=0.05 y x_1=0.99.

$$f(x_1)=L_1\cdot x_1+V_1\cdot y_1-C_0=0$$

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

$$V_1=M-L_1$$

$$y_1=1.42\,x_1$$

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

$$M=L_0+V_0$$

$$C_0=L_0\,x_0+V_0\,y_0$$

$$L_0=300;\ \ x_0=0$$

$$V_0 = 100;\ y_0 = 0.2$$

Igual que en el anterior ejemplo primero se deben ordenar las operaciones de manera que el valor de una variable siempre sea calculado antes de ser empleado. En este caso además se tienen constantes, cuando eso sucede, primero se debe asignar el valor de dichas constantes a las variables respectivas. Luego, con estas constantes y el valor de "x_1" se pueden calcular las variables restantes en el orden: C_0, M, L_c, y_1, L_1, V_1 y finalmente el valor de la función. No obstante, no es el único orden válido, es igualmente correcto el orden: M, C_0, L_c, y_1, L_1, V_1 o el orden: M, y_1, L_c, C_0, L_1, V_1 o el orden: y_1, L_c, C_0, M, L_1, V_1 etc. pues M, C_0, L_c y y_1 no dependen el uno del otro y en consecuencia pueden ser calculados en cualquier orden.

La función, que tendrá el nombre "fx1_3.m" es:

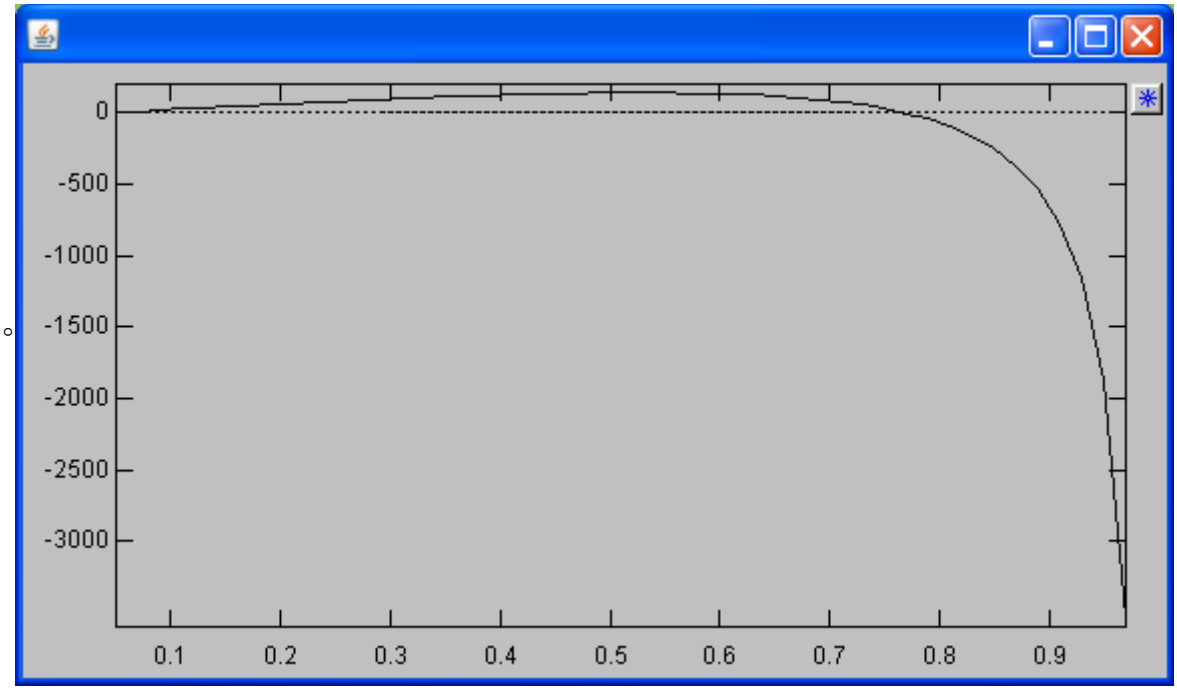
```
function r=fx1_3(x)
  L0=300; x0=0; V0=100; y0=0.2;
  C0=L0*x0+V0*y0;
  M=L0+V0;
  Lc=L0*(1-x0);
  y1=1.42*x1;
  L1=Lc/(1-x1);
  V1=M-L1;
  r=L1*x1+V1*y1-C0;
end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
format 10,10
>> fx1_3(0.2)
ans = 62.1
>> fx1_3(0.8)
ans = -69.6
```

Elaborando la gráfica con un incremento igual a 0.02, se obtiene:

```
>> x3=0.05:0.02:0.99; y3=fx1_3(x3); plot(x3,y3,"k")
```



Donde como se puede observar esta función tiene una solución entre 0.7 y 0.8.

2.4. EJERCICIOS

1. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 10 dígitos de precisión y elabore la gráfica de la función entre x=0 y x=2, siendo el color de la curva rojo. Determine el lugar (o lugares) de la solución.

$$f(x)=2x^2+1-e^x=0$$

2. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 12 dígitos de precisión y elabore la gráfica de la función entre z=0 y z=10, siendo el color de la curva verde. Determine el lugar (o lugares) de la solución.

$$f(z)=e^{z/2}+z^2-60=0$$

3. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 12 dígitos de precisión y elabore la gráfica de la función entre z=0.1 y z=1, siendo el color de la curva azul. Determine el lugar (o lugares) de la solución.

$$f(z)=\ln(z)+e^{z+3}-z^{3.1}-42=0$$

4. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 9 dígitos de precisión y elabore la gráfica de la función entre x=0 y x=4, siendo el color de la curva negro. Determine el lugar (o lugares) de la solución.

$$f(x) = x^3 - 1.95x^2 - 6.77x + 9.44 = 0$$

5. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 14 dígitos de precisión y elabore la gráfica de la función entre x=2 y x=10, siendo el color de la curva magenta. Determine el lugar (o lugares) de la solución.

$$f(x)=e^{\frac{x+y}{5}}+x^2y^2-70y-15=0$$

$$y=\frac{115-2x^2+4x}{5x}$$

6. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 7 dígitos de precisión y elabore la gráfica de la función entre x=12 y x=13, siendo el color de la curva cian. Determine el lugar (o lugares) de la solución.

$$f(x)=y^2-z^3-5x-166.81=0$$

$$y=\frac{1500-20x^2+x^3}{15}$$

$$z=\frac{300+1.5x^2-e^{\frac{x}{2}}}{9}$$

7. Elabore una función que calcule el resultado de la siguiente expresión. Muestre el resultado con 11 dígitos de precisión y elabore la gráfica de la función entre V=0.2 y V=2, siendo el color de la curva amarillo. Determine el lugar (o lugares) de la solución.

$$f(V)=\frac{P\cdot V}{R\cdot T}-\frac{V}{V-b}+\frac{\Omega_a}{\Omega_b}\frac{b}{V+b}F=0$$

$$b=\frac{\Omega_a R T_c}{P_c}$$

$$F=\frac{1}{T_r}\left[1+\left(0.480+1.574\,\omega-0.176\,\omega^2\right)\left(1-T_r^{0.5}\right)\right]^2$$

$$T_r=\frac{T}{T_c}$$

$$\Omega_a=\left[9\left(2^{1/3}-1\right)\right]^{-1}$$

$$\Omega_b=\frac{2^{1/3}-1}{3}$$

$$T=400\,;\;\; P=20\,;\;\; T_c=190.6\,;$$

$$P_c=45.4\,;\;\; \omega=0.008\,;\;\; R=0.08206$$

3. ESTRUCTURAS ESTÁNDAR 2

Continuando con el repaso de las estructuras estándar en este tema se repasará brevemente la estructura selectiva.

3.1. SELECCIÓN

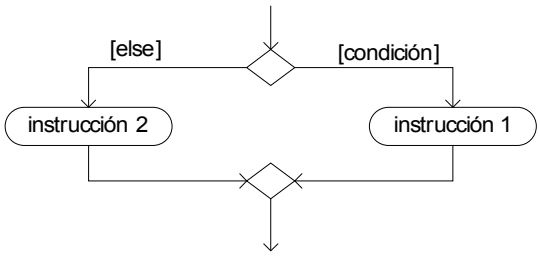
Los problemas reales no pueden ser resueltos empleando únicamente la secuencia, la mayoría requieren por lo menos una estructura selectiva y con frecuencia una o más estructuras iterativas.

En Jasymca se cuenta con la estructura selectiva "if-else" que tiene la siguiente sintaxis:

```
if condición
  instrucciones 1;
else
  instrucciones 2;
end
```

Donde, como se recordará, si la condición es verdadera se ejecutan las "instrucciones 1" y si es falsa las "instrucciones 2". En cualquier caso el programa continua con las instrucciones que se encuentran después de la palabra "end".

El diagrama de actividades de esta estructura (una forma de representar el algoritmo) es el siguiente:



El bloque "else" es opcional, es decir que puede ser omitido, en ese caso si la condición es falsa el programa continua directamente con las instrucciones que se encuentran después de "end".

3.1.1. Expresiones lógicas

Las condiciones se crean escribiendo expresiones lógicas. Una expresión lógica es aquella que devuelve sólo uno de dos resultados "verdadero" (que en Jasymca es el número 1) y "falso" (que en Jasymca es el número 0).

Las expresiones lógicas se crean empleando operadores relacionales y operadores lógicos. Los operadores relacionales disponibles en Jasymca son:

Operador	Relación
==	Igual
~=	Diferente
>	Mayor
>=	Mayor o igual
<	Menor
<=	Menor o igual

Todos estos operadores comparan dos valores y devuelven verdadero (1) si se cumple la relación (es decir si son iguales, diferentes, mayores, etc.) y falso (0) en caso contrario.

Los operadores relacionales pueden ser empleados conjuntamente los operadores relacionales. Los operadores lógicos disponibles en Jasymca son:

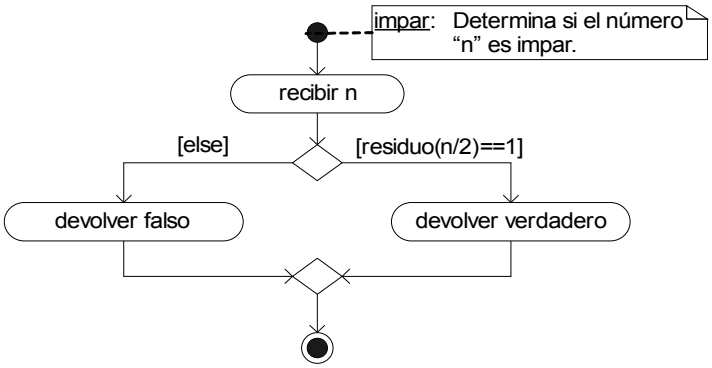
Operador	Operación
~	No lógico
&	Y lógico
|	O lógico

Todos estos operadores trabajan con valores lógicos (1 = verdadero y 0 = falso) y devuelven otro valor lógico. El operador de negación "~" (el operador No) cambia el valor lógico al que precede de verdadero a falso y viceversa. El operador "&" (Y lógico) devuelve verdadero sólo si los dos valores sobre los que opera son verdaderos, en cualquier otro caso devuelve falso. El operador "|" (O lógico) devuelve falso sólo si los dos valores sobre los que opera son falsos, en cualquier otro caso devuelve verdadero.

3.2. EJEMPLOS

1. Elabore la función "impar" que reciba un número y devuelva verdadero si el mismo es impar y falso en caso contrario.

Un número es impar si no es divisible entre 2, es decir si el residuo de la división es 1. El algoritmo que resuelve el problema es el siguiente:



Debido a un error en Jasymca, cuando se emplea la instrucción "divide" dentro de una función es necesario que la misma reciba por lo menos dos parámetros, pues de lo contrario no calcula correctamente el cociente y el residuo.

Esta es la razón por la cual en la codificación del algoritmo se reciben dos parámetros cuando en realidad sólo se requieres uno:

```
function r=impar(n,x)
  if divide(n,2)(2)==1 r=1; else r=0; end
end
```

Haciendo correr esta función (que como se recordará debe ser guardada con el nombre "impar.m"), se obtiene:

```
>> impar(15,0)
r = 1
>> impar(16,0)
r = 0
```

Que son los resultados correctos, es decir verdadero (1) si es impar y falso (0) si es par (el segundo parámetro "0" es necesario por el error antes mencionado)

No obstante, en este caso, no es necesario emplear la estructura "if", pues el resultado buscado (falso o verdadero) es devuelto directamente por la expresión lógica. Con esta consideración el programa se reduce a:

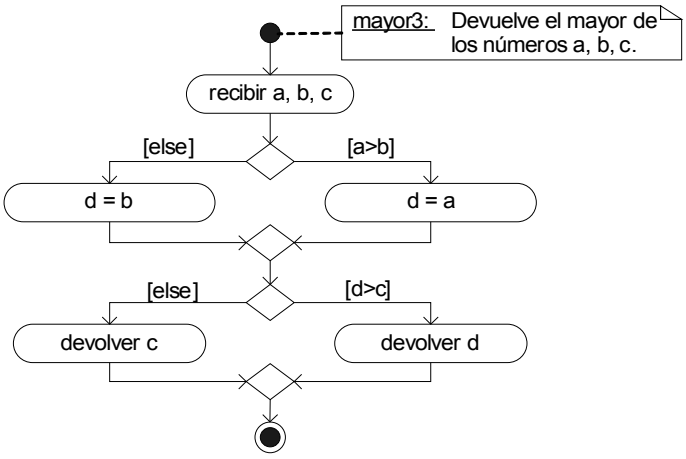
```
function r=impar(n,x)
  r = divide(n,2)(2)==1;
end
```

Con el cual, por supuesto, se obtienen los mismos resultados.

2. Elabore la función "mayor3" que reciba tres números y devuelva el mayor de ellos.

Para determinar el mayor de tres números primero se comparan dos de ellos, luego el mayor de ellos se compara con el tercero, siendo el mayor de los tres el resultado de esta última comparación.

El algoritmo que resuelve el problema es:



Siendo el código respectivo:

```
function r=mayor3(a,b,c)
  if a>b d=a; else d=b; end
  if d>c r=d; else r=c; end
end
```

Probando la función con algunos valores se obtiene:

```
>> mayor3(1,2,3)
r = 3
>> mayor3(1,3,2)
r = 3
>> mayor3(3,1,2)
r = 3
```

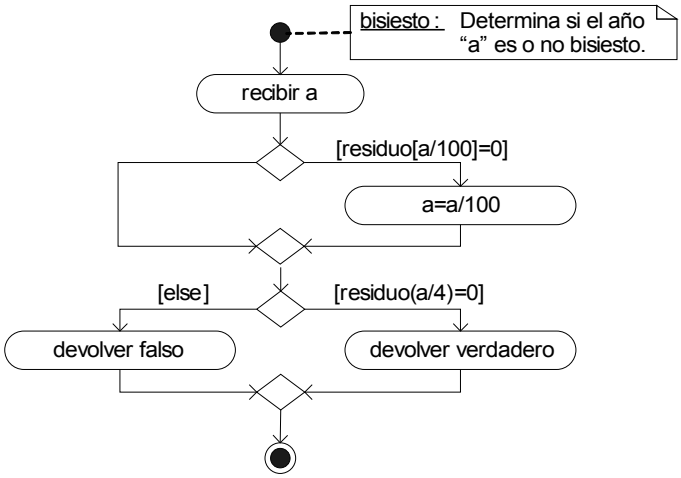
3. Elabore la función "bisiesto" que reciba un año y devuelva verdadero si el mismo es bisiesto y falso en caso contrario.

Un año es bisiesto si es divisible entre cuatro, con excepción de los años que terminan en dos ceros (como 1900), a los cuales se les debe quitar los dos ceros antes de comprobar si son divisibles o no entre cuatro.

Entonces lo primero que se debe hacer, para determinar si un año es bisiesto, es verificar si termina en dos ceros (es decir si es divisible entre 100). De ser así se eliminan dichos ceros (dividiendo el número entre

100), caso contrario se deja el año sin modificación. Finalmente se verifica si el año es divisible entre 4.

El algoritmo que resuelve el problema es:



Al igual que en el ejemplo 1, no es necesario emplear la estructura "if" para codificar la segunda condición del algoritmo: la expresión relacional ("residuo(a/4)=0") devuelve de hecho un resultado lógico (verdadero o falso).

Por la misma razón que en el ejemplo 1 (debido al error con la función "divide"), en el código la función recibe 2 parámetros cuando en realidad sólo requiere uno (el año).

Con estas consideración el código es:

```
function r=bisiesto(a,x)
  if divide(a,100)(2)==0 a=a/100; end
  r = divide(a,4)(2)==0;
end
```

Probando la función con algunos valores se obtiene:

```
>> bisiesto(1988,0)
r = 1
>> bisiesto(1998,0)
r = 0
>> bisiesto(2000,0)
r = 1
>> bisiesto(1900,0)
r = 0
```

4. Elabore la función "cuad" que encuentre las dos soluciones de la ecuación cuadrática: $ax^2+bx+c=0$.

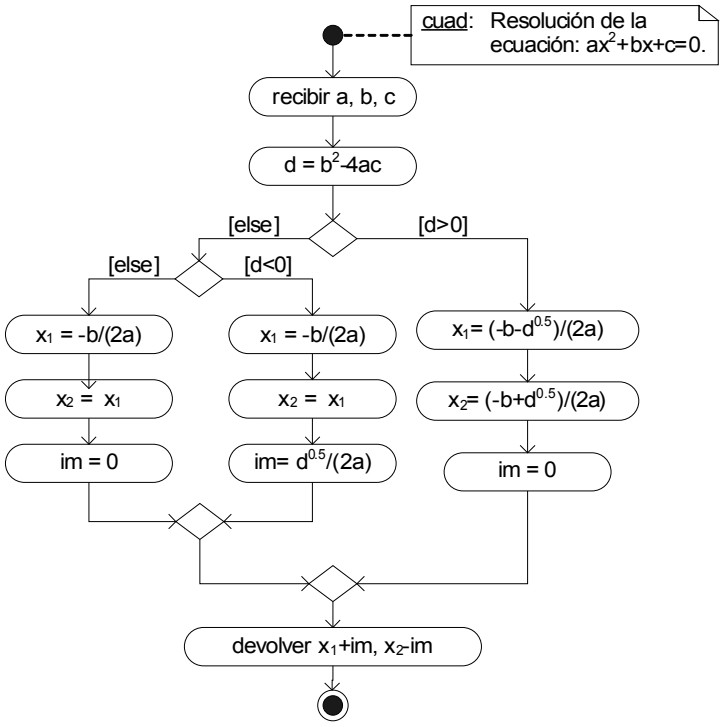
La ecuación general para resolver la ecuación cuadrática es:

$$x_{1,2}=\frac{-b\pm\sqrt{b^2-4\cdot a\cdot c}}{2\cdot a} \qquad (3.1)$$

Al resolver esta ecuación se presentan tres casos que dependen del valor del discriminante (b^2-4ac). Si el discriminante es positivo las soluciones son reales y diferentes, si es cero las soluciones son reales e iguales y si es negativo las soluciones son complejas (un complejo conjugado).

Por lo tanto para resolver la ecuación cuadrática se debe calcular primero el discriminante y en base a su valor calcular las soluciones de acuerdo a al caso que corresponda.

El algoritmo que resuelve el problema es:



El código respectivo es:

```
function r=cuad(a,b,c)
  d=b^2-4*a*c;
  if d>0
    x1=(-b-sqrt(d))/(2*a);
    x2=(-b+sqrt(d))/(2*a);
    im=0;
  else
    if d<0
      x1=-b/(2*a);
      x2=x1;
      im=sqrt(d)/(2*a);
    else
      x1=-b/(2*a);
      x2=x1;
      im=0;
    end
  end
  r=[x1+im,x2-im];
end
```

Donde el resultado se devuelve en un vector (que en Jasymca se crea simplemente escribiendo los elementos entre corchetes).

Probando la función con algunos valores se obtiene:

```
>> format 10,12
>> cuad(1,-7,12)
r = [ 3  4 ]
```

```
>> cuad(1,-14,49)
r = [ 7  7 ]
>> cuad(1,2,3)
r = [ -1+1.41421356237i  -1-1.41421356237i ]
```

Es necesario aclarar que como Jasymca puede trabajar con números complejos, el problema de la ecuación cuadrática puede ser resuelto directamente aplicando la solución general (ecuación 3.1).

3.3. EJERCICIOS

1. Elabore la función "positivo" que reciba un número y devuelva verdadero si es positivo y falso en caso contrario.
2. Elabore la función "mayor" que reciba dos números y devuelva verdadero si el primero es mayor que el segundo y falso en caso contrario.
3. Elabore la función "entero" que reciba un número y devuelva verdadero si es entero y falso en caso contrario (el residuo de "divide" es 0 si alguno de sus argumentos es real).
4. Elabore la función "rcub_", que sin emplear "sign" ni "abs", devuelva la raíz cúbica de un número.
5. Elabore la función "menor3" que reciba tres números y devuelva el menor de ellos.
6. Elabore la función "triangulo" que reciba tres lados lados y devuelva verdadero si los mismos conforman un triángulo y falso en caso contrario. Tres lados conforman un triángulo si en todos los casos posibles la suma de dos de ellos es siempre mayor al tercero.
7. Elabore la función "cuad_" que resuelva el problema de la ecuación cuadrática tomando en cuenta primero el caso imaginario, luego el caso iguales y finalmente el caso reales (y diferentes).

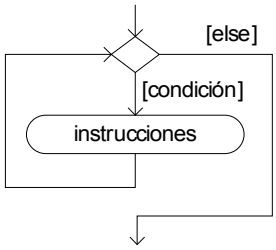
4. ESTRUCTURAS ESTÁNDAR 3

En este tema continúa el breve repaso de las estructuras estándar, pasando a repasar la primera estructura iterativa disponible en Jasymca.

Las estructuras iterativas son aquellas que permiten repetir un determinado proceso una o más veces. En Jasymca se cuenta con dos estructuras iterativas: la estructura "while" y la estructura "for".

4.1. ESTRUCTURA WHILE

La lógica de la estructura "while" es la siguiente:



Es decir que las "instrucciones" se repiten "mientras" que la "condición" sea verdadera. Cuando la condición es falsa ("else") el programa continúa con la instrucción que exista después de la estructura "while".

La forma de codificar esta estructura en Jasymca es:

```
while condición
  instrucciones;
end
```

La forma normal en que finaliza la estructura "while" (cuando la "condición" toma el valor falso (0)) puede ser modificada mediante las instrucciones "break" y "return".

La instrucción "break" ocasiona la salida inmediata de la estructura "while" y el programa continua con la instrucción que se encuentre después del ciclo.

La instrucción "return" (que en realidad funciona tanto dentro de un ciclo como en cualquier otra parte dentro de una función) ocasiona la inmediata finalización de la función. Cuando esta instrucción se ejecuta dentro de un ciclo las instrucciones del ciclo dejan de ejecutarse y el flujo salta hasta el final de la función, devolviendo como resultado el valor que tenía asignado en ese momento la variable resultante.

Adicionalmente, el comportamiento normal de la estructura "while" puede ser modificado con la instrucción "continue". Esta instrucción provoca un salto inmediato a la siguiente iteración, es decir que las instrucciones del ciclo dejan de ejecutarse y el flujo del programa salta a verificar la "condición" del ciclo y, como es usual, se ejecuta el siguiente ciclo si la misma es verdadera o finaliza el ciclo en caso contrario.

Con frecuencia la lógica que resuelve los problemas numéricos requiere de un ciclo infinito, del cual se sale con la instrucción "break" o "return". Como es de suponera, para crear un ciclo infinito, simplemente se hace que la condición del ciclo sea verdadera (1):

```
while 1
  instrucciones;
end;
```

Por supuesto, dentro de las "instrucciones" debe existir al menos una instrucción similar a la siguiente:

```
if condición break;
```

O a la siguiente:

```
if condición return;
```

Pues de lo contrario el ciclo sería realmente infinito y en consecuencia la función nunca terminaría.

4.1.1. Ejemplos

1. Elabore la función "rcuad" que reciba un número "n" y devuelva su raíz cuadrada, calculando la misma con la ecuación de Newton:

$$x_2 = \frac{1}{2} \cdot \left(x_1 + \frac{n}{x_1} \right)$$

Para calcular la raíz cuadrada con esta ecuación se comienza asumiendo un valor inicial para la raíz "x_1", con este valor y la ecuación de Newton se calcula "x_2", entonces se comparan "x_1" y "x_2" y si son aproximadamente iguales el proceso concluye (siendo la raíz x_2), caso contrario el proceso se repite haciendo que "x_1" tome el valor de "x_2". Es decir que el proceso se repite mientras "x_1" y "x_2" no son aproximadamente iguales.

Para decidir si dos valores numéricos dados son aproximadamente iguales se empleará en este y en los subsiguientes temas la expresión:

$$\left| \frac{x_1}{x_2} - 1 \right| < 10^{-d}$$

Donde "x_1" y "x_2" son los valores numéricos a comparar y "d" es el mínimo número de dígitos que deben ser iguales en "x_1" y "x_2" para que la relación sea verdadera.

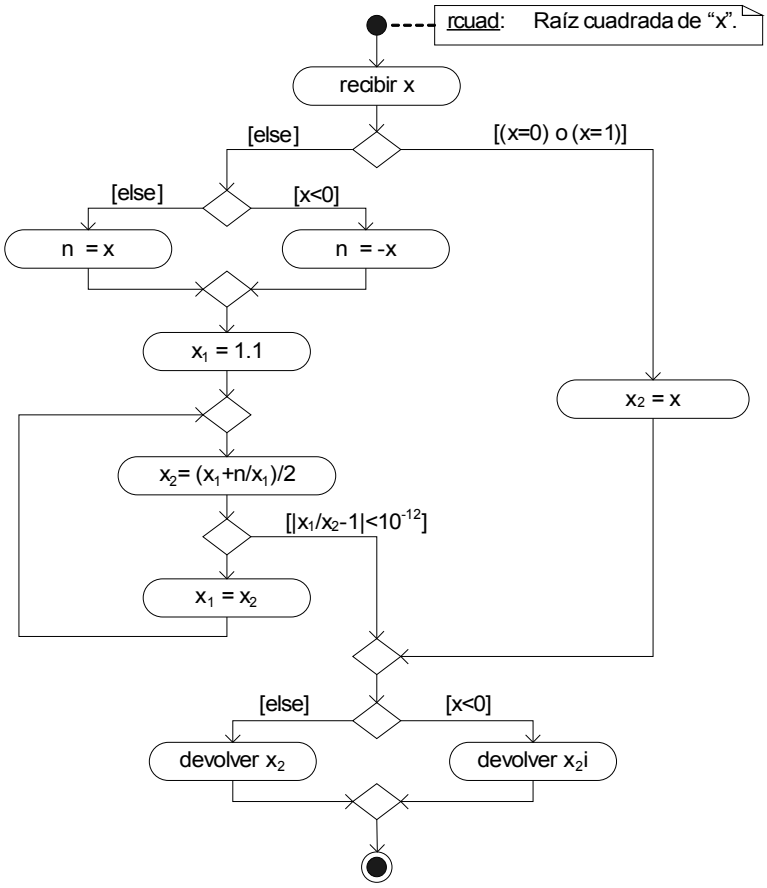
Si los valores que se compara son los correspondientes a una solución (como ocurre frecuentemente), al cumplirse la relación (cuando es verdadera) se habrá encontrado la solución con "d-1" dígitos de precisión. Es decir que se podrá afirmar que la solución es exacta en los primeros "d-1" dígitos (el último dígito puede no ser correcto debido al redondeo).

Con esta relación se puede implementar el procedimiento para calcular la raíz cuadrada de un número. El valor inicial asumido "x_1" puede ser cualquier valor, aunque el número de iteraciones (repeticiones) requeridas, para encontrar la solución, es menor cuanto más cercano es el valor asumido de la solución final. En este ejemplo se asumirá directamente un valor cualquiera "1.1".

El algoritmo que resuelve el problema se presenta en la siguiente página y como se puede observar en el mismo se comprueba primero si el número es 1 o 0, porque como se sabe la raíz cuadrada de cero es cero y la de uno es uno.

Además se trabaja siempre con el valor positivo del número, porque con valores negativos no se logra convergencia (los valores de x_1 y x_2 nunca son aproximadamente iguales) porque como se sabe, la raíz cuadrada de números negativos es imaginaria.

Como también se puede observar, en este ejemplo se repite el proceso hasta que los valores son iguales en los primeros 12 dígitos, por lo tanto el resultado será exacto en los primeros 11 dígitos.



El código respectivo es:

```
function x2=rcuad(x)
  if x==0 | abs(x)==1 x2=x;
  else
    if x<0 n=-x; else n=x; end
    x1=1.1;
    while 1
      x2=(x1+n/x1)/2;
      if abs(x1/x2-1)<1e-12 break; end
      x1=x2;
    end
  end
  if x<0 x2=x2*i;
end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
>> format 10,11
>> rcuad(2)
x2 = 1.4142135624
>> rcuad(-2)
x2 = 1.4142135624i
>> rcuad(64)
x2 = 8
>> rcuad(-23.43)
x2 = 4.8404545241i
```

Los resultados se muestran con 11 dígitos porque esa es la exactitud con la que se encuentra la solución en la función elaborada.

2. Elabore la función "expo" que calcule el exponente de un número empleando la serie de Taylor respectiva.

La serie de Taylor para el exponente es:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + ... + \infty$$

Como esta serie va hasta el infinito no es posible en la práctica calcular la sumatoria (pues se requeriría un tiempo infinito). Entonces, lo que se hace es sumar términos hasta que los resultados de dos sumas sucesivas sean aproximadamente iguales.

Por otra parte, los términos de una serie se calculan en base al término anterior, empleando la razón (o regla) que rige la serie. Por ejemplo, en este caso se puede ver que el nuevo término se calcula multiplicando el término anterior por "x" y dividiendo entre el número que representa la posición del término (siendo el primer término el número 0). Por ejemplo, para calcular el cuarto término (en base al tercero) las operaciones necesarias son:

$$\frac{x^3}{3!} \cdot \frac{x}{4} = \frac{x^4}{4!}$$

Probablemente lo más difícil al programar una serie sea este paso, es decir identificar la razón o regla que rige la misma. Luego prácticamente todas las series siguen la misma lógica: a) se inicializan variables; b) se calcula el nuevo término (aplicando la razón o regla); c) se compara el nuevo término con el anterior y si son aproximadamente iguales el proceso concluye, siendo la solución la última sumatoria; d) se actualizan variables y se repite el proceso desde el paso "b".

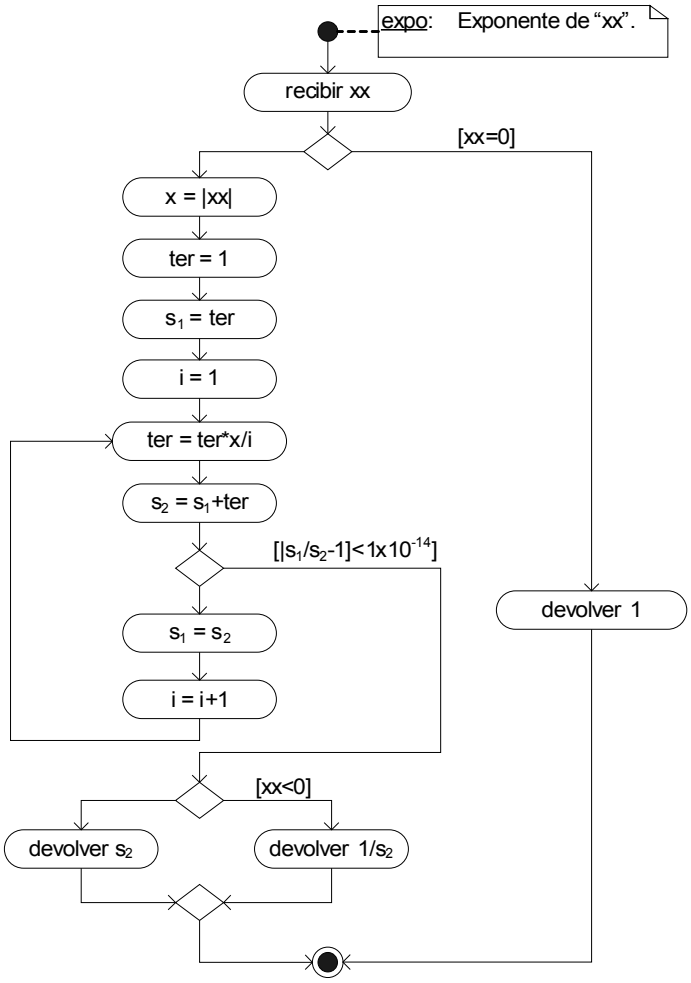
El algoritmo que resuelve el problema, siguiendo los pasos antes indicados, se muestra en la siguiente página. Como se puede ver en el mismo, además de los pasos indicados se verifica primero si el número es cero, porque el exponente de "0" es "1". Además, se trabaja con el valor absoluto del número porque al ser $e^{-x}=1/e^x$, se puede calcular el resultado negativo conociendo el positivo, esto es muy importante en este caso porque de esa manera se evitan las restas que aparecerían si se trabajara con el número negativo.

En general, **en todos los métodos numéricos, se debe evitar trabajar con restas de números muy grandes, porque generan errores de redondeo que llevan a resultados erróneos.**

En este algoritmo el proceso se repite hasta que las dos últimas sumatorias son iguales en 14 dígitos, es decir que el resultado será exacto en 13 dígitos.

El código en Jasymca es:

```
function s2=expo(xx)
  if xx==0 s2=1; return; end
  x=abs(xx);
  ter=1;
  s1=ter;
  i=1;
  while 1
    ter*=x/i;
    s2=s1+ter;
    if abs(s1/s2-1)<1e-14 break; end
    s1=s2;
```




```
    i++;
  end
  if xx<0 s2=1/s2; end
end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
>> format 10,13
>> expo(2)
s2 = 7.389056098931
>> expo(12)
s2 = 162754.791419
>> expo(50)
s2 = 5.184705528587E21
>> expo(-50)
s2 = 1.928749847964E-22
```

Resultados que pueden ser corroborados con "exp".

En el código elaborado existen algunas cosas nuevas. En el cálculo del nuevo término se emplea un operador compuesto:

```
ter*=x/i;
```

Esta operación es equivalente a:

```
ter=ter*x/i;
```

Los operadores compuestos que se pueden emplear en Jasymca son:

Operador compuesto	Operación equivalente
y+=x	y=y+x
y-=x	y=y-x
y*=x	y=y*x
y/=x	y=y/x

El contador "i" empleando un operador de incremento:

```
i++;
```

Que es equivalente a:

```
i=i+1;
```

Los operadores de incremento y decremento que se pueden emplear son:

Operador	Operación equivalente
x++	x=x+1
x--	x=x-1
++x	x=x+1
--x	x=x-1

Si la variable sobre la que actúa el operador forma parte de una expresión, cuando el operador se emplea después de la variable, primero se realizan las operaciones con la variable y luego se incrementa su valor, por el contrario cuando se emplea el operador antes de la variable primero se incrementa su valor y luego se emplea la variable.

Por ejemplo con las siguientes instrucciones se obtiene:

```
>> x=5; y=x++ + 3
y = 8
>> x
x = 6
```

En este caso, primero se suma el valor de "x" (5) a 3 y luego se incrementa su valor en 1. Por el contrario en las siguientes instrucciones:

```
>> x=5;y=++x + 3
y = 9
>> x
x = 6
```

Primero se incrementa el valor de "x" (a 6) y luego se le suma 3.

3. Elabore la función "seno" que calcule el seno de un ángulo en radianes empleando la serie de Taylor respectiva.

La serie de Taylor del seno es:

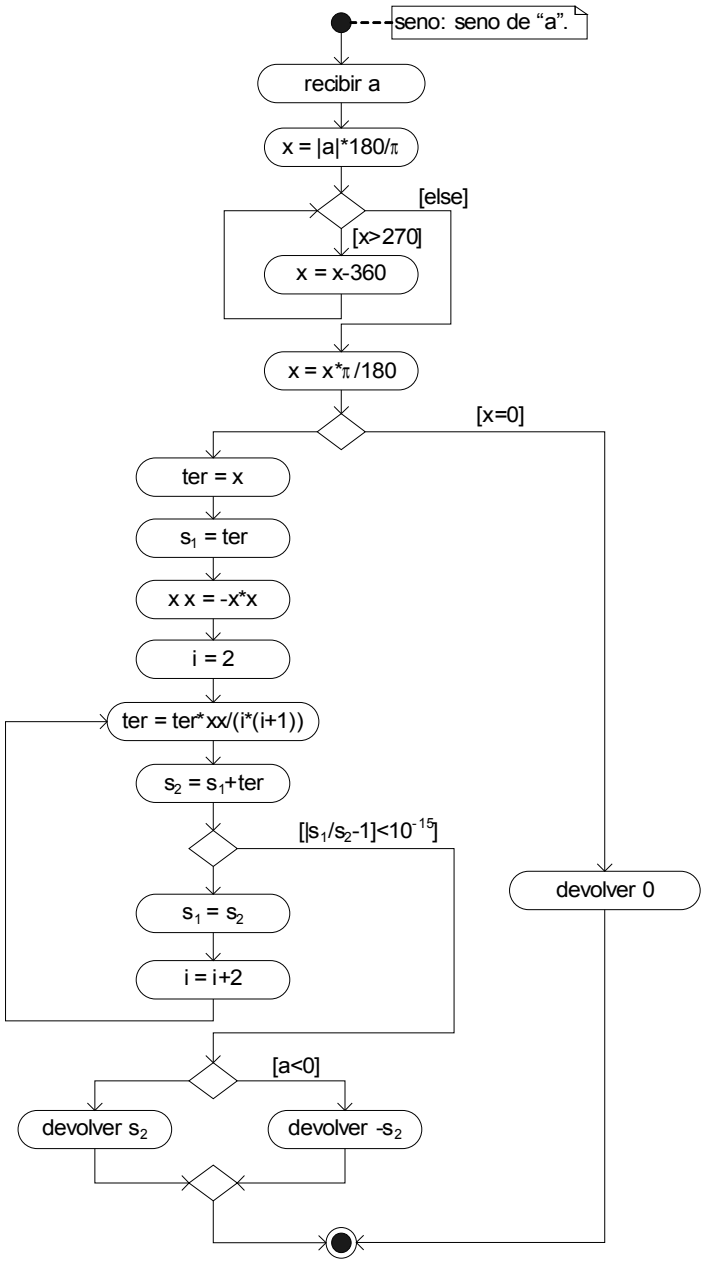
$$\operatorname{sen}(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots\infty$$

En este caso el nuevo término se calcula multiplicando el anterior por $-x^2$ y dividiendo entre un contador y su valor siguiente. Dicho contador comienza en 0 e incrementa de 2 en 2, por ejemplo para calcular el cuarto término a partir del tercero las operaciones son:

$$\frac{x^5}{5!}\cdot\frac{-x^2}{6\cdot 7}=-\frac{x^7}{7!}$$

En este caso la serie misma tiene restas, por lo que esta operación no puede ser evitada, pero sí se puede evitar trabajar con números muy grandes pues como se sabe todo ángulo superior a 3*π/2 puede ser reducido a su equivalente entre -3*π/2 y 3*π/2 simplemente restando al ángulo 2*π hasta que sea menor o igual a 3*π/2. En este proceso, sin embargo, es conveniente trabajar con grados (en lugar de radianes) para minimizar los errores de redondeo, además, como se cumple que sin(-α)=-sin(α), se puede trabajar con el ángulo sin singo y cambiar el signo al devolver el resultado en el caso de que el ángulo sea negativo.

El algoritmo que resuelve el problema es:



El código respectivo es:

```
function s2=seno(a)
  x=abs(a)*180/float(pi);
  while x>270 x-=360; end
```

```
    x*=float(pi)/180;
    if x==0 s2=0; return; end
    ter=x;
    s1=ter;
    xx=-x*x;
    i=2;
    while 1
      ter*=xx/(i*(i+1));
      s2=s1+ter;
      if abs(s1/s2-1)<1e-15 break; end
      s1=s2;
      i+=2;
    end
    if a<0 s2=-s2; end
  end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
>> format 10,14
>> seno(2.56)
s2 = 0.54935543642713
>> seno(67.3)
s2 = -0.97032089132789
>> seno(-120.2)
s2 = -0.7307904049279
```

Resultados que pueden ser corroborados con "sin".

4.1.2. Ejercicios

1. Elabore la función "rcub_" que calcule, con 11 dígitos de exactitud, la raíz cúbica de un número "n" empleando la ecuación de Newton:

$$x_2=\frac{1}{3}\cdot\left(2x_1+\frac{n}{x_1^2}\right)$$

2. Elabore la función "expo_" que modifique la función "expo", de manera que no se considere como caso especial los números negativos. Pruebe luego la función calculando el exponente de 50 y de -50.

3. Elabore la función "senoh" que calcule, con 14 dígitos de exactitud, el seno hiperbólico de un número "x" empleando la serie de Taylor respectiva.

$$\operatorname{senh}(x)=x+\frac{x^3}{3!}+\frac{x^5}{5!}+\frac{x^7}{7!}+\ldots+\infty$$

4. Elabore la función "cose" que calcule, con 12 dígitos de exactitud, el coseno de un ángulo "x" en radianes empleando la serie de Taylor respectiva.

$$\cos(x)=1-\frac{x^2}{2!}+\frac{x^4}{4!}-\frac{x^6}{6!}+\ldots\infty$$

5. Elabore la función "atanh_" que calcule, con 13 dígitos de exactitud, la tangente hiperbólica de un número "x" empleando la serie de Taylor respectiva.

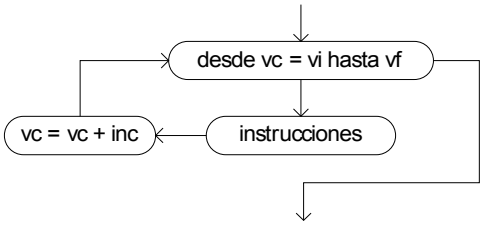
$$\tanh^{-1}(x)=x+\frac{x^3}{3}+\frac{x^5}{5}+\frac{x^7}{7}+\ldots\infty\,;\qquad |x|<1$$

5. ESTRUCTURAS ESTÁNDAR 4

En este tema termina el repaso de las estructuras estándar con la segunda estructura iterativa disponible en Jasyma, la estructura "for".

5.1. ESTRUCTURA FOR

La lógica de la estructura "for" es la siguiente:



En esta estructura las "instrucciones" se repiten desde que la variable de control "vc" toma un valor inicial "vi" hasta que alcanza un valor final "vf", incrementando en cada repetición del ciclo el valor de la variable de control en el valor "inc".

Cuando la variable de control alcanza (o supera) el valor final el ciclo termina y el programa continua con la instrucción que se encuentre después de la estructura.

En realidad en Jasymca la variable de control "vc" toma valores sucesivos de un vector hasta que llega al último elemento. Por lo tanto en Jasymca la variable de control toma valores en el orden en que se encuentren en el vector, pudiendo en consecuencia incrementar, disminuir o mantener su valor.

No obstante, la mayoría de las aplicaciones prácticas implican el incremento o decremento constante de la variable de control (tal como se muestra en el algoritmo). En Jasymca, para lograr un incremento o decremento constante, se debe crear un vector con elementos uniformemente espaciados y la forma más sencilla de lograrlo es con el operador de rango ":" (empleado ya en el tema 2 para crear gráficos).

Por lo tanto, si bien en Jasymca la forma general de codificar la estructura "for" es:

```
for vc=vector
  instrucciones;
end
```

La forma más usual es:

```
for vc=vi:inc:vf
  instrucciones;
end
```

Donde "vc", "vi", "inc" y "vf" son, al igual que en el algoritmo, la variable de control, el valor inicial, el incremento y el valor final respectivamente. En esta segunda forma, si al principio el valor inicial es menor al valor final las instrucciones del ciclo no se ejecutan ni una vez.

Se prefiere la estructura "for", sobre la estructura "while", cuando se conoce el número de veces que debe repetirse un determinado proceso.

5.2. EJEMPLOS/EJERCICIOS

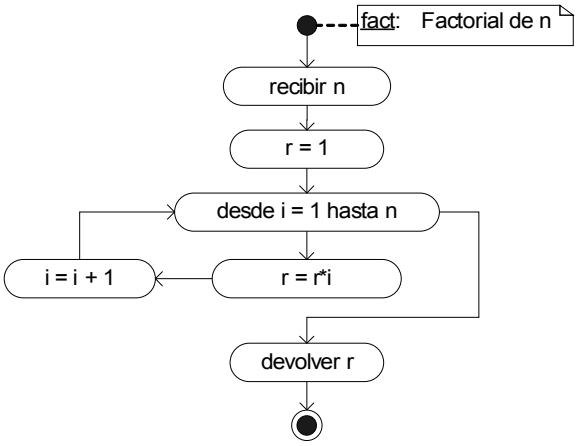
1. Elabore la función "fact" que calcule el factorial de un número "n".

El factorial de un número es el resultado de multiplicar los números sucesivos desde 1 hasta el número cuyo factorial se quiere calcular, siendo por definición el factorial de 0 igual a 1, es decir:

$$n!=\prod_{i=1}^{n} i=1*2*3*4*\cdots*n$$

$$0!=1$$

Para resolver este problema simplemente se debe repetir el ciclo desde 1 hasta "n" multiplicando en cada repetición el resultado anterior por el nuevo valor de la variable de control:



El código respectivo es:

```
function r=fact(n)
  r=1;
  for i=1:n r*=i; end
end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
>> format 10,16
>> fact(5)
r = 120
>> fact(12)
r = 479001600
>> fact(18)
r = 6402373705728000
```

Resultados que pueden ser corroborados con la función "factorial" de Jasymca.

2. Elabore la función "sum" que calcule la sumatoria de los números enteros existentes entre 1 y el número "n" introducido.

$$\text{sum}(n)=\sum_{i=1}^{n} i$$

3. Elabore la función "sumi" que calcula la sumatoria de los números impares existentes entre 1 y el número introducido.

$$\text{sumi}(n)=\sum_{i=1,3,5,\ldots}^{n} i$$

4. Elabore la función "prodp" que calcula la productoria de los números pares existentes entre 2 y el número introducido.

$$\text{prodp}(n)=\prod_{i=2,4,6,\ldots}^{n} i$$

5. Elabore la función "cheb" que calcula el chebyshev enésimo "n" de un número real "x".

La ecuación de definición del Chebyshev es:

$$Ch_{n+1}(x)=2\cdot x\cdot Ch_n(x)-Ch_{n-1}(x)$$
$$Ch_0(x)=1$$
$$Ch_1(x)=x$$

Como se puede ver se conoce el Chebyshev para "n=0" y "n=1". Con estos valores se puede calcular el Chebyshev para "n=2", luego con este valor y el anterior se calcula el Chebyshev para "n=3" y así sucesivamente hasta llegar al Chebyshev enésimo "n" que se quiere calcular.

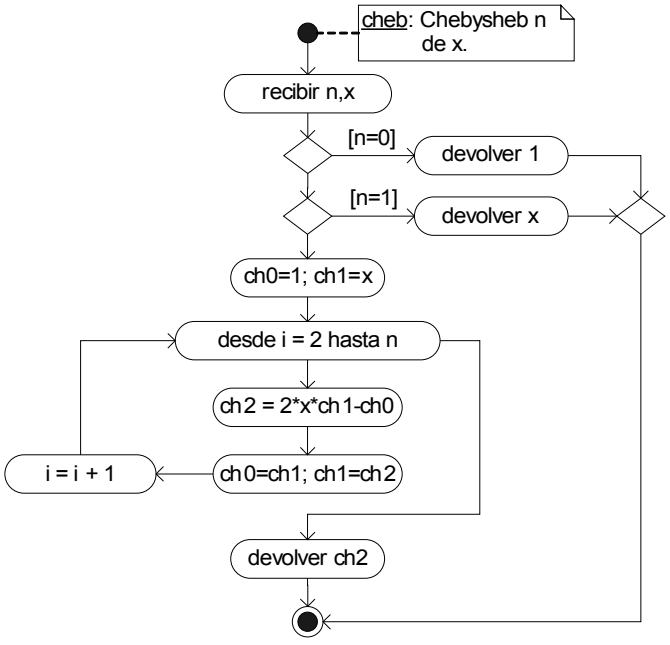
Por ejemplo para calcular el Chebyshev 7 (n=7) de "x" las operaciones a realizar son:

$$Ch_2(x)=2\cdot x\cdot Ch_1(x)-Ch_0(x)$$
$$Ch_3(x)=2\cdot x\cdot Ch_2(x)-Ch_1(x)$$
$$Ch_4(x)=2\cdot x\cdot Ch_3(x)-Ch_2(x)$$
$$Ch_5(x)=2\cdot x\cdot Ch_4(x)-Ch_3(x)$$
$$Ch_6(x)=2\cdot x\cdot Ch_5(x)-Ch_4(x)$$
$$Ch_7(x)=2\cdot x\cdot Ch_6(x)-Ch_5(x)$$

Por lo tanto, para calcular el Chebyshev de 7 se calculan los valores desde 2 hasta 7. Entonces, como se conoce el número de veces que se deben repetir los cálculos la estructura más adecuada para resolver el problema es la estructura "for".

Para calcular un nuevo valor sólo se requieren los dos anteriores, por lo que en el proceso sólo es necesario guardar los dos últimos valores.

El algoritmo que resuelve el problema es:



Siendo el código respectivo:

```
function ch2=cheb(n,x)
  if n==0 ch2=1; return; end
  if n==1 ch2=x; return; end
  ch0=1; ch1=x;
  for i=2:n
    ch2=2*x*ch1-ch0;
    ch0=ch1; ch1=ch2;
  end
end
```

Haciendo correr el programa con algunos valores de prueba se obtiene:

```
>> format 10,14
>> cheb(5,1.2)
ch2 = 11.25312
>> cheb(7,1.2)
ch2 = 38.9997312
>> cheb(6,3.5)
ch2 = 51841
>> cheb(2,7.1)
ch2 = 99.82
```

6. Elabore la función "fibo" que calcula el Fibonacci enésimo "n". La ecuación de definición del Fibonacci es:

$$F_n = F_{n-1} + F_{n-2}$$
$$F_1 = F_2 = 1$$

7. Elabore la función "leg" que calcula el Legendre enésimo "n" de un número real "x". La ecuación de definición del Legendre es:

$$\mathrm{Le}_n(x) = \left(2 - \frac{1}{n}\right) \cdot x \cdot \mathrm{Le}_{n-1}(x) - \left(1 - \frac{1}{n}\right) \cdot \mathrm{Le}_{n-2}(x)$$
$$\mathrm{Le}_0(x) = 1$$
$$\mathrm{Le}_1(x) = x$$

6. ECUACIONES ALGEBRAICAS 1

En este tema comienza propiamente el estudio de los métodos numéricos que son parte del contenido de la materia.

Se inicia dicho estudio con algunos de los métodos existentes para la resolución de ecuaciones algebraicas con una incógnita, es decir encontrar el valor de la variable independiente que cumple con la igualdad expresada en la ecuación.

En general son dos las formas en las cuales suele ordenarse la ecuación a resolver:

$$x=g(x) \qquad (6.1)$$

y

$$f(x)=0 \qquad (6.2)$$

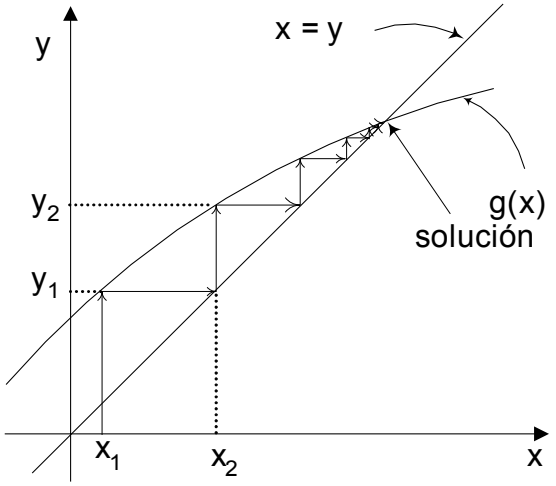
Es decir se puede despejar una de las incógnitas e igualarla al resto de la ecuación, o agrupar todos los miembros en un lado de la ecuación e igualarla a cero.

En este tema se estudia el método más sencillo que existe para resolver ecuaciones escritas en la primera forma.

6.1. MÉTODO DE SUSTITUCIÓN DIRECTA

El método de Sustitución Directa es el método más simple que existe. En el mismo simplemente se asume un valor de "x", se reemplaza en la ecuación despejada (g(x)) y si el resultado es igual al valor asumido, el proceso concluye, caso contrario el valor calculado se convierte en el nuevo valor asumido y el proceso se repite.

Gráficamente la solución se encuentra en la intersección entre la recta "x=y" y la función "g(x)".



Gráficamente, el método sigue el proceso mostrado en la figura: Con el valor asumido, x_1, se obtiene el valor de la función "y_1" en la intersección con la curva de la función, luego "y_1" se convierte en el nuevo valor de prueba "x_2" (en la intersección con la recta x=y), con "x_2" se obtiene el nuevo valor de la función "y_2" y el mismo se convierte en el nuevo valor de prueba "x_3" y así sucesivamente hasta que el valor de la función "y_n" es

prácticamente igual al nuevo valor "x_n" (lo que ocurre en la intersección de la curva con la recta "x=y").

Este método es lo suficientemente sencillo como para ser aplicado sin necesidad de programar el método. Por ejemplo, para resolver la siguiente función (es decir encontrar el valor de "x" que cumple con la igualdad):

$$f(x)=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}-x=0 \qquad (6.3)$$

Se la coloca primero en la forma x=g(x):

$$x=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}=g(x)$$

Se programa la función despejada:

```
function r=gx(x)
  r=(52+3*sqrt(x)-8*x^0.8)^0.36;
end
```

Y comenzando con un valor inicial asumido (en este caso 1.1) se calcula el valor de la función:

```
>> format 10:10
>> gx(1.1)
ans = 3.984055388
```

Como el valor asumido "1.1" no es igual al calculado "3.984...", se repite el proceso, pero ahora el valor asumido es el valor calculado:

```
>> gx(ans)
ans = 3.552012132
```

Una vez más el valor asumido "3.984..." es diferente al calculado "3.552..", por lo que el proceso se repite hasta que los valores asumido y calculado son iguales (en los 10 dígitos de precisión con los que se muestra el resultado:

```
>> gx(ans)
ans = 3.618479599
>> gx(ans)
ans = 3.608323565
>> gx(ans)
ans = 3.609876926
>> gx(ans)
ans = 3.609639377
>> gx(ans)
ans = 3.609675705
>> gx(ans)
ans = 3.609670149
>> gx(ans)
ans = 3.609670999
>> gx(ans)
ans = 3.609670869
>> gx(ans)
ans = 3.609670889
>> gx(ans)
ans = 3.609670886
>> gx(ans)
ans = 3.609670886
```

En este punto el valor asumido y calculado son iguales, por lo que se concluye que la solución es "x=3.609670886" (con 10 dígitos de precisión).

Cuando como en este caso, el resultado se acerca a la solución en cada repetición, se dice que el proceso es **convergente**, por el contrario si el resultado se aleja de la solución en cada repetición se dice que es **divergente**. Puede ser también **oscilatorio** cuando los resultados se van alternando entre dos valores que se repiten indefinidamente. Se pueden presentar igualmente procesos convergentes oscilatorios y divergentes oscilatorios.

6.1.1. Ejercicios

1. **Encuentre la solución de la siguiente ecuación, con 7 dígitos de precisión, aplicando manualmente el método de sustitución directa (valor inicial 1.1).**

$$f(x)=x^3+2x^2+3x+4=0 \tag{6.4}$$

2. **Encuentre la solución de la siguiente ecuación, con 9 dígitos de precisión, aplicando manualmente el método de sustitución directa (valor inicial 1.1).**

$$f(x)=\cos(x)+\left(1+x^2\right)^{-1}=0 \tag{6.5}$$

6.1.2. Algoritmo y Código

El algoritmo del método básicamente es el descrito en el anterior acápite, sin embargo, antes de elaborar formalmente el algoritmo, es necesario tomar en cuenta algunos aspectos.

Por lo general, en los métodos numéricos existen dos criterios por los cuales se termina el proceso iterativo. El primero ya ha sido empleado en temas anteriores y consiste en terminar el proceso cuando los dos últimos valores calculados son iguales en un determinado número de dígitos, es decir, cuando se cumple que:

$$\left|\frac{x_1}{x_2}-1\right|<10^{-d} \tag{6.6}$$

Como se sabe en estos casos se puede asegurar que el resultado es exacto en "d-1" dígitos (siempre y cuando "d" no sea muy cercano a la precisión con la que trabaja el lenguaje, en el caso de Jasymca 16 dígitos).

El segundo criterio consiste en terminar el proceso cuando prácticamente se logra la igualdad de la ecuación, es decir cuando se cumple que:

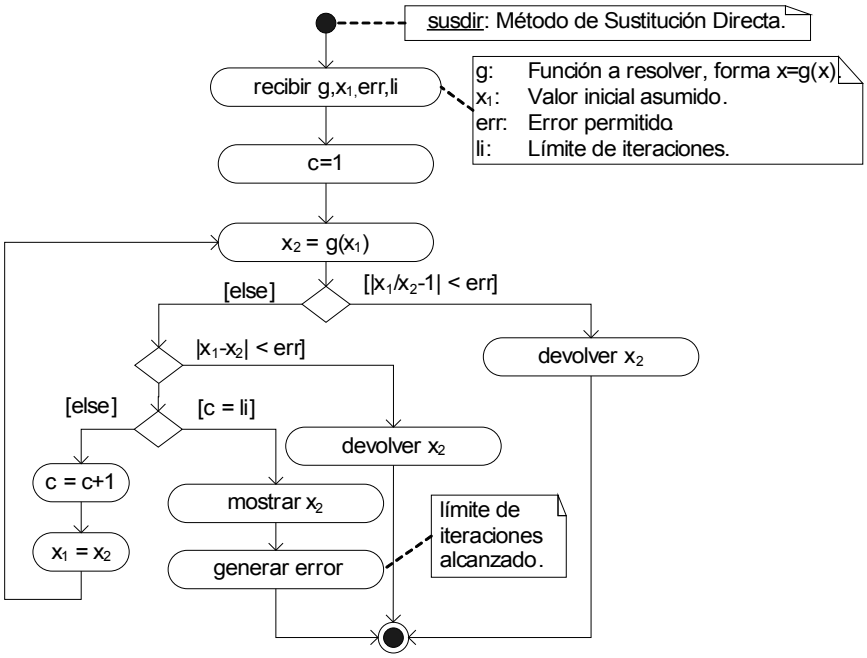
$$|diff|<10^{-d} \tag{6.7}$$

Donde "diff" es la diferencia entre los valores a ambos lados de la ecuación (si la ecuación está igualada a cero es directamente el valor de la ecuación). Con este criterio se sabe que la igualdad expresada en la ecuación se cumple con "d" dígitos de exactitud.

Por lo general ambos criterios se cumplen para los mismos valores, sin embargo, en ocasiones puede ocurrir que se cumpla sólo uno de ellos, razón por la cual es conveniente incluir ambos criterios al momento de programar un método numérico.

Por otra parte (y como ya se explicó) no siempre el proceso es convergente (llega al resultado), por lo que se debe incluir también un contador, para terminar el proceso en caso de llegar a un límite de iteraciones dado.

El algoritmo del método de Sustitución Directa, tomando en cuenta las anteriores consideraciones es el siguiente:



El código respectivo es:

```
function x2=susdir(g,x1,err,li)
  c=1;
  while 1
    x2=g(x1);
    if abs(x1/x2-1)<err break; end
    if abs(x1-x2)<err break; end
    if c++ == li error("¡Error! x= %f\n",x2); end
    x1=x2;
  end
end
```

En este programa se ha empleado la instrucción "error". Esta instrucción ocasiona la inmediata conclusión de la función y muestra el mensaje de error que se escribe entre comillas. Su formato es similar al de "printf", excepto que "error" termina la ejecución de la función mientras que "printf" sólo muestra la información pero no concluye la función:

```
error("formato",valores)
```

```
printf("formato",valores)
```

Donde "valores" son los valores a mostrar separados con comas y "formato" es la información y la forma en que se muestran dicha información.

"Formato" es una cadena de caracteres que a más de texto, acepta algunas combinaciones especiales de letras como "%f" que al momento de presentar la información es reemplazado por el valor respectivo en la lista de valores y "\n" que es la instrucción para un salto de línea. Así en el programa la instrucción "error" muestra el mensaje "¡Error! x= ", seguido del valor correspondiente al último valor calculado y terminando con un salto de línea.

Probando el programa con la ecuación del ejemplo manual (ecuación 6.1), asumiendo que la misma sigue programada en la función "gx", empleando un valor inicial igual a 1.1, un error permitido igual a 1e-11 (10 dígitos de exactitud) y un límite de 50 iteraciones se obtiene:

```
>> format 10,10
>> susdir($gx,1.1,1e-11,50)
```

```
x2 = 3.609670886
```

Que es el mismo resultado obtenido manualmente. Observe que para mandar la función como un dato se ha precedido el nombre de la misma con el símbolo "$". Este símbolo le instruye a Jasymca que no evalúe la función, de esta manera se manda al método la función en sí y no un resultado obtenido con la función.

Si se reduce el número de iteraciones a 10 se obtiene:

```
susdir($gx,1.1,1e-11,10)
¡Error! x= 3.609670869
```

Lo que informa que el método de Sustitución Directa no ha encontrado la solución y que el último valor calculado es: 3.609670869. En este caso la solución no ha sido encontrada porque se requieren más de 10 iteraciones para encontrar la solución con 10 dígitos de exactitud.

La instrucción "printf" nos permite visualizar los valores intermedios de una función, algo que es de gran utilidad para analizar el comportamiento de funciones, comprender la lógica de un programa, encontrar posibles errores y corregirlos.

Por ejemplo si se quieren ver los valores intermedios que se obtienen en el método de Sustitución Directa se puede añadir la siguiente línea después de la instrucción "x2=g(x1);":

```
printf("%f: %f\n",c,x2);
```

Que muestra en cada iteración el valor del contador de ciclos ("c") y el nuevo valor calculado ("x2"). Haciendo correr el programa con esta modificación se obtiene:

```
susdir($gx,1.1,1e-11,50)
 1: 3.984055388
 2: 3.552012132
 3: 3.618479599
 4: 3.608323565
 5: 3.609876926
 6: 3.609639377
 7: 3.609675705
 8: 3.609670149
 9: 3.609670999
 10: 3.609670869
 11: 3.609670889
 12: 3.609670886
 13: 3.609670886
 14: 3.609670886
 15: 3.609670886
x2 = 3.609670886
```

Donde se puede ver que el método requiere de 15 iteraciones para llegar a la solución con 10 dígitos de exactitud y que en cada una de ellas el valor calculado se aproxima más a la solución, por lo que se puede afirmar que el proceso es convergente.

Si luego se quieren suprimir la impresión de valores intermedios simplemente se borra la línea añadida o, más simple aún, se transforma la línea en un comentario. En Jasymca, para transformar una línea en un comentario se escribe el símbolo de porcentaje (%) delante de la instrucción que se quiere transformar en comentario. Todas las instrucciones que se encuentren después del "%" y hasta el final de la línea se transforman en un comentario.

Como ocurre en todos los lenguajes de programación, las líneas que son comentarios no se ejecutan: para el programa es como si no existieran. Los comentarios sirven para documentar el programa o, como en este caso, anular temporalmente una o más líneas de código.

Para transformar en comentario la línea añadida previamente se escribe:

```
% printf("%f: %f\n",c,x2);
```

Es importante recalcar que el "%" convierte en comentario toda la línea, no sólo una instrucción, por lo que si se ha escrito todo el programa en una sola línea, todo el programa, a partir del "%", se convierte en un comentario y en consecuencia el programa no funcionará.

Como es de suponer, para que la línea de código vuelva a ejecutarse (vuelva a ser reconocida por Jasymca) simplemente se borra el porcentaje.

El método de Sustitución Directa es el más sencillo de los métodos pero a la vez el menos estable. La convergencia en este método depende no sólo del valor inicial asumido, sino también del término despejado para igualar la variable independiente al resto de la ecuación.

6.1.3. Ejercicios

3. Encuentre la solución positiva de la siguiente función con 9 dígitos de precisión. Estime el valor inicial elaborando la gráfica de la función (igualada a cero).

$$f(x)=2x+1-e^{x}=0$$

4. Encuentre las soluciones negativa y positiva de la siguiente función con 7 dígitos de exactitud. Estime los valores iniciales elaborando la gráfica de la función (¿es posible encontrar ambas soluciones?).

$$f(z)=\ln(z)+e^{z+3}-z^{3.1}-42=0$$

5. Encuentre las dos soluciones positivas del siguiente sistema de ecuaciones (como una función de x) con 11 dígitos de precisión. Estime los valores iniciales elaborando la gráfica de la función. ¿En cuántas iteraciones se encuentra la segunda solución?

$$\begin{aligned} 3x^{2.1}-5y&=7.0\\ y^{1.2}+4z&=14.3\\ x+y^{2}+z^{2}&=14.0 \end{aligned}$$

6. Encuentre la solución negativa y positiva del siguiente sistema de ecuaciones (como una función de x) con 8 dígitos de precisión. Estime los valores iniciales elaborando la gráfica de la función, fije el límite de iteraciones en 15 y muestre los resultados de dichas iteraciones.

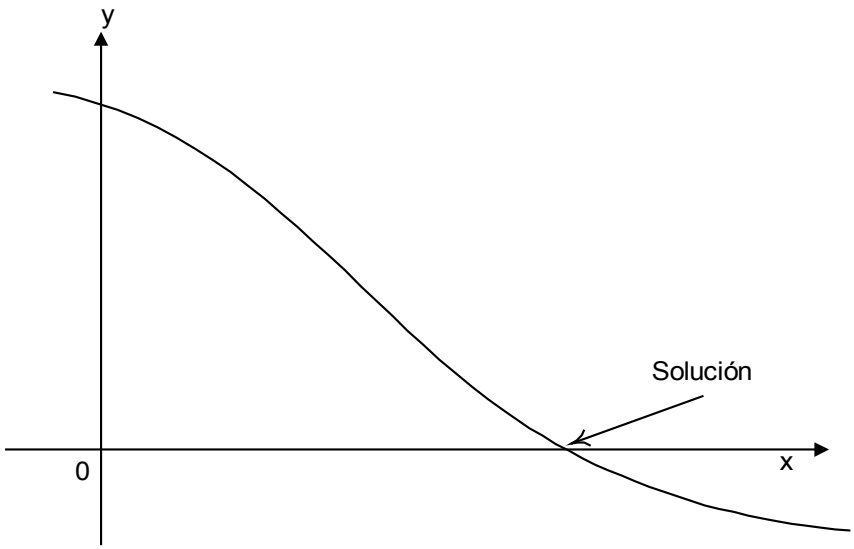
$$\begin{aligned} 15y+20x^{2}-x^{3}&=1500\\ 9z-1.5x^{2}+e^{\frac{x}{2}}&=300\\ y^{2}-z^{3}-5x&=166.81 \end{aligned}$$

7. ECUACIONES ALGEBRAICAS 2

Además del método de Sustitución Directa, existen otros métodos para resolver ecuaciones en la forma x=g(x). Con ellos mejora la probabilidad de encontrar el resultado buscado, sin embargo persiste la necesidad de despejar una de las variables de la ecuación, además, la convergencia sigue dependiendo de la variable despejada.

Por esta razón en la práctica se emplean casi exclusivamente los métodos que resuelven ecuaciones en la forma f(x)=0. Muchas ecuaciones están ya en esta forma y de no ser así el igualar una ecuación a cero es mucho más sencillo que despejar una de sus variables, además la convergencia es más estable en la forma f(x)=0 que en la forma x=g(x).

Como ya se ha visto anteriormente, cuando la ecuación está en la forma f(x)=0, la solución se encuentra en la intersección de la curva "f(x)" y la recta "x=0":



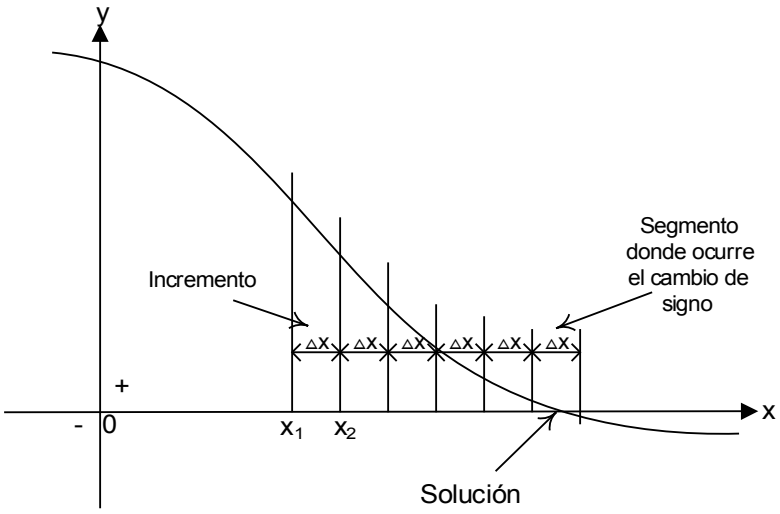
En este tema comienza el estudio de los métodos para resolver ecuaciones algebraicas que se encuentra en esta forma.

7.1. MÉTODO DE SUSTITUCIÓN DIRECTA

El método incremental es, conceptualmente, el método más sencillo que existe para resolver ecuaciones en la forma f(x)=0. Se basa en el hecho de que a ambos lados de la solución la función cambia de signo, entonces el método simplemente busca este cambio incrementando el valor de la variable independiente hasta que ocurre el mismo. Se sabe que ha ocurrido un cambio de signo cuando al multiplicar dos valores consecutivos de la función el resultado es negativo (menor que cero).

Gráficamente el método incremental sigue el procedimiento mostrado en la figura de la siguiente página: se asume un valor inicial "x_1" y un incremento "Δx". Con este incremento se van calculando valores sucesivos de la variable independiente ($x_{i+1}=x_i+\Delta x$) hasta que el valor de la función (la intersección con la curva) cambia de signo.

El método en sí concluye cuando se encuentra el segmento donde ocurre el cambio de signo, sin embargo, puede ser repetido cuantas veces se requiera, empleando un incremento igual a la décima parte del incremento original, para encontrar un segmento más cercano a la solución.



Aunque el proceso es muy simple, para comprender mejor como funciona se encontrará el segmento de solución para la siguiente función, siendo el valor inicial asumido x_1=1.1 y el incremento Δx=0.1:

$$f(x)=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}-x=0 \qquad (7.1)$$

Como se encuentra en la forma f(x)=0, no es necesario realizar ninguna operación algebraica, por lo que la función puede ser programada directamente:

```
function r=fx(x)
  r=(52+3*sqrt(x)-8*x^0.8)^0.36-x;
end
```

Entonces se asignan los valores iniciales asumidos y se calcula el valor de la función (mostrándola en este ejemplo con 12 dígitos de precisión):

```
>> format 10,12
>> x=1.1; dx=0.1; fx(x)
ans = 2.88405538779
```

Ahora se incrementa el valor de la variable independiente y se calcula el valor de la función:

```
>> x=x+dx, fx(x)
x = 1.2
ans = 2.76912747776
```

Como la función no cambia de signo (sigue siendo positiva) se continúa incrementando los valores de "x", y calculando los valores respectivos de la función, hasta que ocurra un cambio de signo:

```
>> x=x+dx, fx(x)
x = 1.3
ans = 2.65424229656
>> x=x+dx, fx(x)
x = 1.4
ans = 2.53939166098
>> x=x+dx, fx(x)
x = 1.5
ans = 2.42456823296
>> x=x+dx, fx(x)
x = 1.6
ans = 2.30976538517
```

```
>> x=x+dx, fx(x)
x = 1.7
ans = 2.19497709079
>> x=x+dx, fx(x)
x = 1.8
ans = 2.0801978327
>> x=x+dx, fx(x)
x = 1.9
ans = 1.96542252857
>> x=x+dx, fx(x)
x = 2
ans = 1.85064646856
>> x=x+dx, fx(x)
x = 2.1
ans = 1.73586526314
>> x=x+dx, fx(x)
x = 2.2
ans = 1.62107479931
>> x=x+dx, fx(x)
x = 2.3
ans = 1.50627120335
>> x=x+dx, fx(x)
x = 2.4
ans = 1.39145080915
>> x=x+dx, fx(x)
x = 2.5
ans = 1.27661013098
>> x=x+dx, fx(x)
x = 2.6
ans = 1.16174583997
>> x=x+dx, fx(x)
x = 2.7
ans = 1.04685474369
>> x=x+dx, fx(x)
x = 2.8
ans = 0.931933768265
>> x=x+dx, fx(x)
x = 2.9
ans = 0.816979942659
>> x=x+dx, fx(x)
x = 3
ans = 0.701990384751
>> x=x+dx, fx(x)
x = 3.1
ans = 0.586962288954
>> x=x+dx, fx(x)
x = 3.2
ans = 0.471892915111
>> x=x+dx, fx(x)
x = 3.3
ans = 0.356779578498
>> x=x+dx, fx(x)
x = 3.4
ans = 0.241619640759
>> x=x+dx, fx(x)
x = 3.5
ans = 0.126410501638
>> x=x+dx, fx(x)
```

```
x = 3.6
ans = 1.11495913895E-2
>> x=x+dx, fx(x)
x = 3.7
ans = -0.104165636232
```

En este último incremento se produce un cambio de signo, por lo que el proceso concluye, estando el segmento de solución entre 3.6 y 3.7, es decir que la solución se encuentra en algún lugar entre 3.6 y 3.7.

7.1.1. Ejercicio

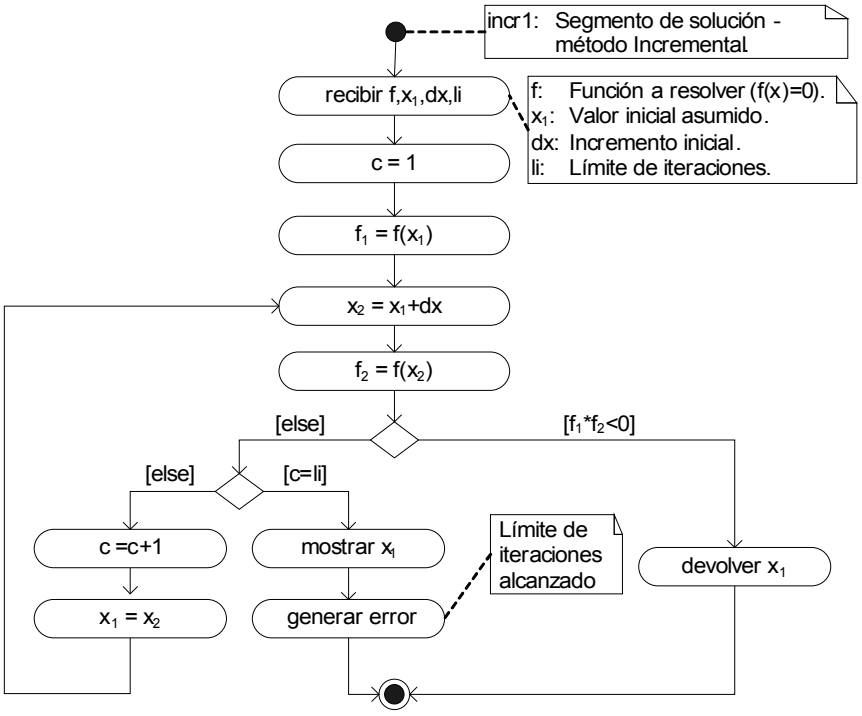
1. Encuentre el segmento de solución de la siguiente ecuación aplicando manualmente el método incremental (valor inicial -2.1, incremento 0.1).

$$f(x)=x^3+2x^2+3x+4=0 \tag{7.2}$$

7.1.2. Algoritmo y código

Como se hace evidente en el ejemplo y en los ejercicios, el proceso es simple, pero moroso, por lo que es conveniente su automatización a través de un programa.

Al momento de implementar el algoritmo se toma en cuenta el límite de iteraciones, pues puede suceder que no exista una raíz en el lugar donde se hace la búsqueda, o que el incremento sea muy pequeño (o muy grande) como para encontrar el segmento de solución. En esos casos se genera un mensaje de error y se muestra el penúltimo valor calculado para la variable independiente. El algoritmo tomando en cuenta estas consideraciones es:



Y el código respectivo es:

```
function x1=incr1(f,x1,dx,li)
  c=1; f1=f(x1);
  while 1
    x2=x1+dx;
```

```
      f2=f(x2);
      if f1*f2<0 break; end
      if c++ == li error("¡Error! x1= %f\n",x1); end
      x1=x2;
    end
  end
```

Haciendo correr el programa con la ecuación resuelta en el ejemplo manual se obtiene:

```
>> incr1($fx,1.1,0.1,50)
x1 = 3.6
```

Es decir que la solución se encuentra entre x_1=3.6 y x_1+dx=3.6+0.1=3.7, que es el mismo segmento encontrado en el ejemplo manual.

Si se quieren ver los resultados intermedios, se debe añadir una instrucción "printf" después de la instrucción "x2=x1+dx":

```
printf("%f: x= %f\n",c,x2);
```

Como ya se vio, para anular la impresión de resultados intermedios, simplemente se añade un "%" delante de esta línea.

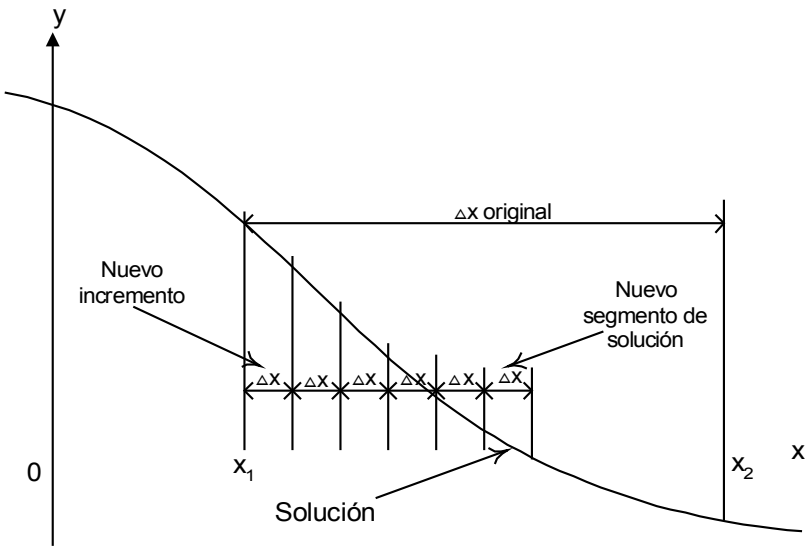
7.1.3. Ejercicio

2. Encuentre el segmento de solución positivo de la siguiente ecuación con el método incremental empleando un incremento igual a 0.1.

$$f(x)=\cos(x)+\left(1+x^2\right)^{-1}=0 \tag{7.3}$$

7.1.4. Cálculo de la solución con el método Incremental

La aplicación principal del método incremental es la que se vio en el anterior acápite, es decir encontrar el segmento de solución, sin embargo, es posible repetir el proceso, empleando incrementos más pequeños, hasta que el incremento sea lo suficientemente pequeño como para asumir que el valor de la variable (a uno u otro lado del segmento) constituye la solución:



Para mejorar en un dígito, en cada repetición, la exactitud del resultado obtenido, el nuevo incremento debe ser igual a la décima parte del incremento anterior.

Por ejemplo, para la ecuación resuelta en el ejemplo manual (ecuación 7.1), comenzando con un incremento inicial igual a 0.1 se obtiene:

```
>> x1=1.1; dx=0.1, incr1($fx,x1,dx,50)
dx = 0.1
x1 = 3.6
```

Por lo tanto (y como ya se vio) la solución exacta se encuentra entre 3.6 y 3.7. Ahora se repite el proceso siendo el valor inicial 3.6 y el incremento la décima parte del incremento original:

```
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-2
x1 = 3.6
```

Aparentemente no cambia nada, sin embargo el incremento ahora es 0.01, por lo tanto, se sabe que la solución se encuentre entre 3.6 y 3.61.

Repitiendo el proceso unas cuantas veces más se obtiene:

```
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-3
x1 = 3.609
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-4
x1 = 3.6096
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-5
x1 = 3.60967
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-6
x1 = 3.60967
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-7
x1 = 3.609671
>> x1=ans; dx=dx/10, incr1($fx,x1,dx,10)
dx = 1.0E-8
x1 = 3.609671
```

En los 7 dígitos con los que se muestra el resultado, el valor se repite cuando el incremento es igual a 10^{-8}, por lo tanto se puede firmar que la solución es exacta en esos 7 dígitos. Observe que en estos cálculos el límite de iteraciones se ha fijado en 10, porque al ser el incremento la décima parte del anterior, no es posible que la solución se encuentre más allá del décimo segmento.

7.1.5. Ejercicios

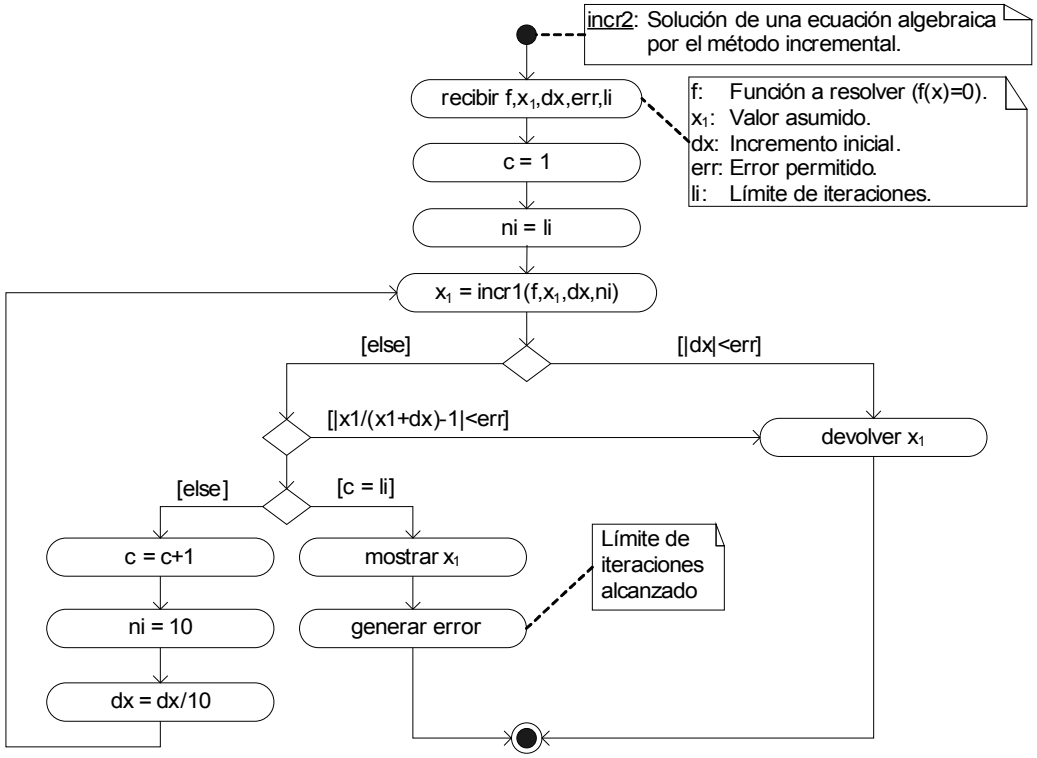
3. Encuentre, con 9 dígitos de exactitud, la solución positiva de la siguiente ecuación aplicando manualmente el método incremental, comenzando con un valor inicial igual a 0.1 y un incremento igual a 0.1.

$$f(x)=2x+1-e^{x}=0 \tag{7.4}$$

7.1.6. Algoritmo y código para el cálculo de la solución

El algoritmo que automatiza el cálculo de la raíz (solución) de una ecuación con el método incremental se presenta en la siguiente página. Como se puede ver el proceso iterativo concluye cuando los valores a ambos lados del segmento son casi iguales o cuando el incremento es casi cero. El número de dígitos de exactitud (y/o exactitud de la solución encontrada) se

fija mediante el error permitido. Se ha incluido también un límite de iteraciones porque puede suceder que el lugar donde ocurre el cambio de signo sea en realidad una discontinuidad y no una solución.



El código respectivo en Jasymca es:

```
function x1=incr2(f,x1,dx,err,li)
  c=1; ni=li;
  while 1
    x1=incr1(f,x1,dx,ni);
    if abs(dx)<err return; end
    if abs(x1/(x1+dx)-1)<err return; end
    if c++ == li error("¡Error!, x=%f\n",x1); end
    ni=10; dx/=10;
  end
end
```

Haciendo correr el programa con la ecuación del ejemplo manual con un error permitido igual a 1e-12 y un límite de 100 iteraciones, se obtiene:

```
>> format 10,11
>> incr2($fx,1.1,0.1,1e-12,100)
x1 = 3.6096708861
```

Que es el resultado correcto con 11 dígitos de exactitud.

Como en "incr1" y "susdir" para ver resultados intermedios se debe incluir una instrucción "printf" después de calcular el nuevo valor de "x" (después de x1=incr1(f,x1,dx,ni);).

7.1.7. Ejercicio

4. Encuentre la primera solución positiva de la siguiente función con 12 dígitos de exactitud empleando un incremento inicial igual a 1.

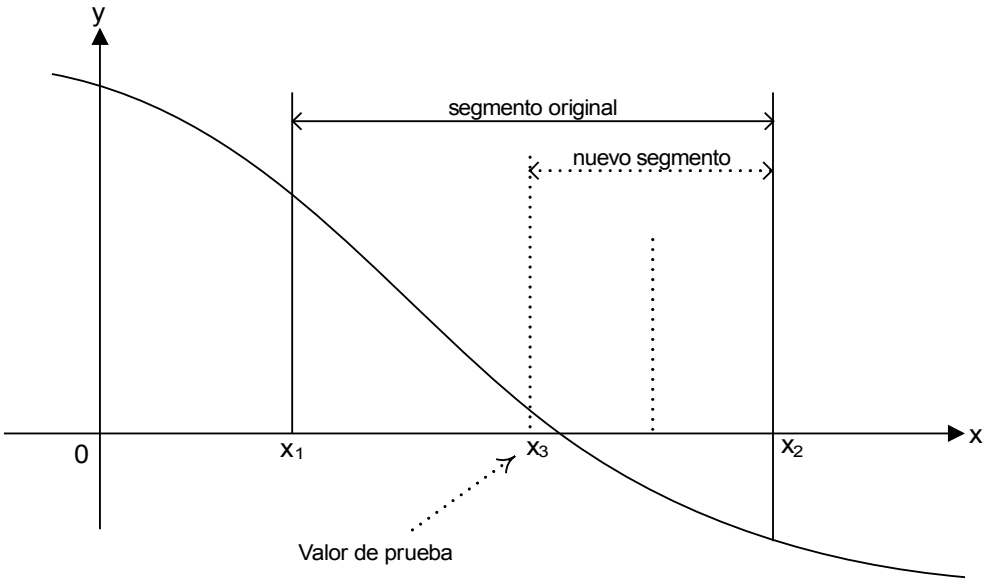
$$
\begin{aligned}
&f(p)=\cos(k\cdot l)\cdot\cosh(k\cdot l)+1=0\\
&k^2=\frac{p}{a}\\
&a^2=\frac{E\cdot I\cdot g}{A\cdot y}\\
&l=120;\quad I=170.6;\quad E=3\times10^6\\
&y=0.066;\quad A=32;\quad g=386;
\end{aligned}
\qquad (7.5)
$$

7.2. MÉTODO DE LA BISECCIÓN (BOLTZANO)

Como ya se dijo en el anterior acápite, el método incremental (incr1) se emplea sobre todo para encontrar el segmento de solución. Una vez encontrado dicho segmento se recurre a otros métodos como el de la Bisección. Por lo general no se continúa con el método incremental (incr2) porque requiere demasiadas iteraciones (hasta 10 iteraciones por cada llamada a "incr1").

El método de la Bisección simplemente comprueba si la solución se encuentra a la mitad del segmento. De ser así el proceso concluye, caso contrario continúa la búsqueda en la mitad del segmento donde ocurre el cambio de signo.

Gráficamente el proceso es el que se muestra en la figura:



El lugar donde se encuentra la solución se determina comparando los signos de $f(x_1)$ y $f(x_3)$. Si tienen signos diferentes (si el resultado de multiplicarlos es negativo), la solución se encuentra en la mitad izquierda (entre x_1 y x_3), caso contrario se encuentra en la mitad derecha (entre x_3 y x_2).

La ecuación para el cálculo del nuevo valor de prueba ("x_3") se deduce a partir de la gráfica:

$$x_3=\frac{x_2-x_1}{2}+x_1$$

$$x_3=\frac{x_2-x_1+2x_1}{2}$$

$$x_3=\frac{x_1+x_2}{2} \tag{7.6}$$

Para comprender mejor el proceso se resolverá manualmente la ecuación (7.1). Se comienza por encontrar el segmento de solución con "incr1", que como se recordará se hace con:

```
>> format 10,9
>> incr1($fx,1.1,0.1,100)
x1 = 3.6
```

Siendo el valor de la función en este punto positivo:

```
>> f1=fx(x1)
f1 = 1.11495914E-2
```

La solución se encuentra entre x_1=3.6 y x_2=3.7. Con estos valores y la ecuación (7.6) se calcular el nuevo valor de prueba y el valor de la función para el mismo:

```
>> x1=3.6; x2=3.7; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.65
f3 = -4.65010743E-2
```

Como f1 y f3 tienen signos diferentes la solución se encuentra entre x1 y x3, por lo tanto x2 toma el valor de x3:

```
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.625
f3 = -1.76740242E-2
```

Una vez más f1 y f3 tienen signos diferentes, por lo que x2 toma nuevamente el valor de x3:

```
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.6125
f3 = -3.26178957E-3
```

Nuevamente f1 y f3 tienen signos diferentes:

```
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60625
f3 = 3.94400731E-3
```

Ahora f1 y f3 tienen el mismo signo, por lo que la solución se encuentra entre x2 y x3, entonces x1 toma el valor de x3:

```
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.609375
f3 = 3.41135506E-4
```

Una vez más f1 y f3 tienen el mismo signo, entonces x1 toma el valor de x3:

```
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.6109375
f3 = -1.46032037E-3
```

Ahora f1 y f3 tienen signos diferentes, por lo que x2 toma el valor de x3:

```
>> x2=x3; f1=f3, x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.61015625
f3 = -5.59590767E-4
```

Siguiendo este razonamiento se obtienen los siguientes resultados:

```
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60976563
f3 = -1.09227214E-4
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60957031
f3 = 1.1595425E-4
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60966797
f3 = 3.36354389E-6
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.6097168
f3 = -5.29318286E-5
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60969238
f3 = -2.47841407E-5
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60968018
f3 = -1.0710298E-5
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967407
f3 = -3.67337696E-6
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967102
f3 = -1.54916505E-7
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60966949
f3 = 1.6043137E-6
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967026
f3 = 7.24698599E-7
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967064
f3 = 2.84891047E-7
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967083
f3 = 6.49872711E-8
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967093
f3 = -4.49646169E-8
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967088
f3 = 1.00113273E-8
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.6096709
f3 = -1.7476645E-8
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967089
f3 = -3.73265907E-9
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967088
f3 = 3.13933413E-9
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967089
f3 = -2.96663138E-10
>> x2=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967088
f3 = 1.42133683E-9
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
```

```
x3 = 3.60967089
f3 = 5.62336844E-10
>> x1=x3; x3=(x1+x2)/2, f3=fx(x3)
x3 = 3.60967089
f3 = 1.32836409E-10
```

En este punto, los dos últimos valores calculados iguales en los 9 dígitos, por lo que el proceso puede concluir en este punto siendo esa la solución con 9 dígitos de exactitud (observe que el valor de la función es también es cercano a cero con 9 dígitos de exactitud, es menor a 1e-9).

Aunque a primera vista se podría pensar que el método de la bisección requiere un mayor número de iteraciones que el método incremental, no es así, lo que sucede es que en el método incremental no se ven las iteraciones intermedias que ocurren cuando se llama a "incr2".

7.2.1. Ejercicio

5. Encuentre la primera solución positiva de la siguiente función con 7 dígitos de exactitud, ubicando el segmento de solución con el método incremental (incremento igual a 0.1).

$$f(z)=\ln(z)+e^{z+3}-z^{3.1}-42=0$$

7.2.2. Algoritmo y código

Con el ejemplo y ejercicio del anterior acápite se hace evidente que la aplicación manual del método de la bisección no es práctica. Además de requerir varias iteraciones, cuando se comete algún error en una de ellas se debe repetir todo el proceso.

El algoritmo que automatiza el proceso se presenta en la siguiente página y en el mismo se han hecho las mismas consideraciones que en los anteriores métodos, añadiéndose además la condición para determinar si "x1" o "x2" toman el valor de "x3".

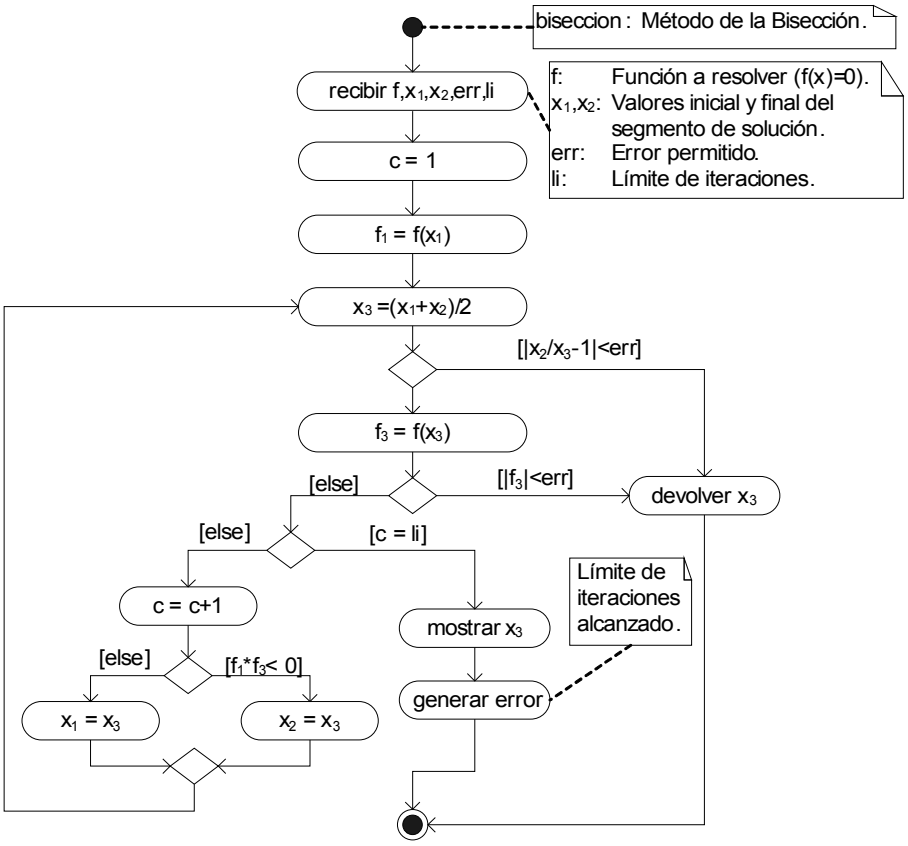
El código respectivo en Jasymca es:

```
function x3=biseccion(f,x1,x2,err,li)
  c=1; f1=f(x1);
  while 1
    x3=(x1+x2)/2;
    if abs(x2/x3-1)<err return; end
    f3=f(x3);
    if abs(f3)<err return; end
    if c++ == li error("¡Error! x=%f\n",x3); end
    if f1*f3<0 x2=x3; else x1=x3; end
  end
end
```

Haciendo correr el programa con la función resuelta en el ejemplo manual se obtiene:

```
>> format 10,9
>> incr1($fx,0.1,0.1,100)
x1 = 3.6
>> biseccion($fx,3.6,3.7,1e-10,100)
x3 = 3.60967089
```

Que es el mismo resultado obtenido en dicho ejemplo. Como de costumbre para mostrar resultados intermedios se añade un "printf" después de calcular el último varo de "x" (después de: x3=(x1+x2)/2;).



7.2.3. Ejercicios

6. Encuentre con el método de la bisección las dos soluciones positivas del siguiente sistema de ecuaciones (como una función de x), con 10 dígitos de exactitud, siendo 0.1 el incremento del método incremental y empleando como valor inicial para la segunda solución la primera solución encontrada más 0.1.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

7. Encuentre la solución negativa (comenzando la búsqueda en -25, con un incremento igual a 1) y positiva (comenzando la búsqueda en 1, con un incremento igual a 1) del siguiente sistema de ecuaciones (como una función de x) con 9 dígitos de exactitud.

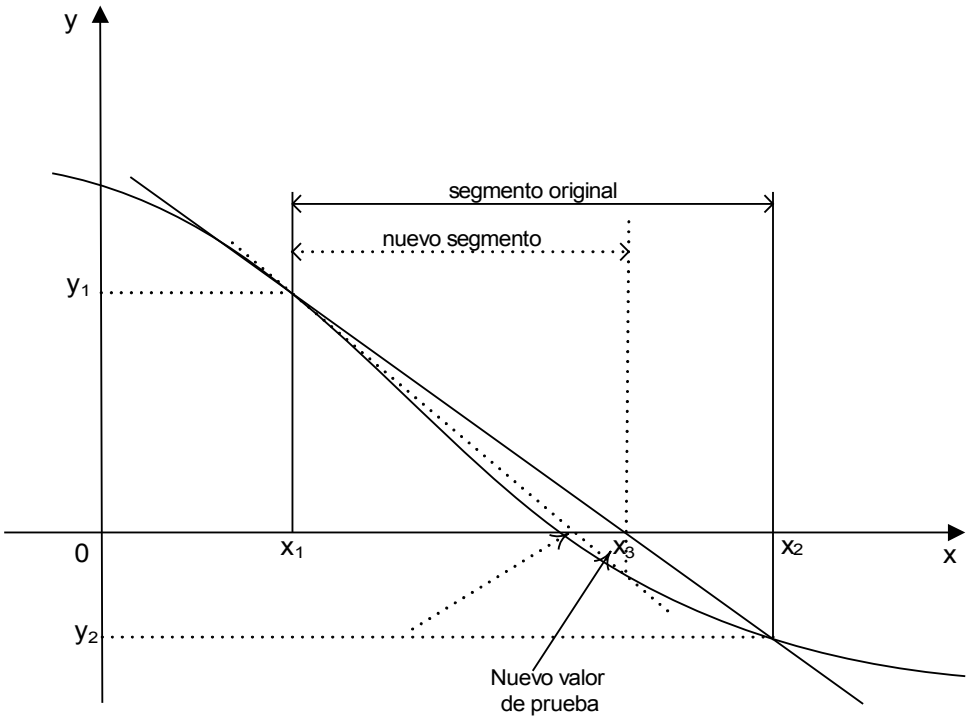
$$15y + 20x^2 - x^3 = 1500$$
$$9z - 1.5x^2 + e^{\frac{x}{2}} = 300$$
$$y^2 - z^3 - 5x = 166.81$$

8. ECUACIONES ALGEBRAICAS 3

Continuando con el estudio de los métodos para resolver ecuaciones algebraicas con una incógnita, en este tema se estudian tres métodos más para la forma f(x)=0.

8.1. MÉTODO DE REGULA FALSI

Al igual que el método de la Bisección, este método comienza con un segmento de solución conocido, sin embargo, en este caso el nuevo valor de prueba (x_3) se calcula por interpolación lineal, tal como se muestra en la figura:



En esencia difiere del método de la Bisección por la forma en que se encuentra el nuevo valor de prueba. La ecuación para el cálculo del valor de prueba "x3", se deduce de la intersección entre la línea que pasa por los puntos (x_1,f_1), (x_2,f_2) y la recta x=0.

Reemplazando dichos puntos en la ecuación de la línea recta, de obtiene:

$$y_1 = a + bx_1$$

$$y_2 = a + bx_2$$

De donde resulta:

$$y_1 - y_2 = b(x_1 - x_2)$$

$$b = \frac{y_1 - y_2}{x_1 - x_2} \qquad (8.1)$$

Y con "b" se puede calcular el valor de "a":

$$y_1 = a + \frac{y_1 - y_2}{x_1 - x_2} x_1$$

$$a=\frac{y_1x_1-y_1x_2-y_1x_1+y_2x_1}{x_1-x_2}$$

$$a=\frac{y_2x_1-y_1x_2}{x_1-x_2} \qquad (8.2)$$

Reemplazando "a" y "b" en la ecuación general de la línea recta (y=a+bx), se obtiene:

$$y=\frac{y_2x_1-y_1x_2}{x_1-x_2}+\frac{y_1-y_2}{x_1-x_2}x$$

Pero en la intersección "y" es cero y "x" es "x_3", de donde resulta:

$$x_3=\frac{y_2x_1-y_1x_2}{y_2-y_1} \qquad (8.3)$$

Que es la ecuación del método de Regula Falsi. Para comprender mejor el proceso se volverá a resolver, manualmente, la siguiente ecuación:

$$f(x)=\left(52+3\sqrt{x}-8x^{0.8}\right)^{0.36}-x=0 \qquad (8.4)$$

Como se recordará, el programa para esta ecuación es:

```
function r=fx(x)
  r=(52+3*sqrt(x)-8*x^0.8)^0.36-x;
end
```

El segmento de solución se encuentra con el método incremental:

```
>> format 10,12

>> incr1($fx,1.1,1,50)

x1 = 3.1
```

Entonces los valores iniciales son:

```
>> x1=ans, x2=ans+1, y1=fx(x1), y2=fx(x2)
x1 = 3.1
x2 = 4.1
y1 = 0.586962288954
y2 = -0.566020176438
```

Con estos valores se calcula el nuevo valor de prueba y el valor correspondiente de la función:

```
>> x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60908171336
y3 = 6.79273504388E-4
```

Como se puede ver, el valor de la función (y_3) se acerca bastante a 0 en una sola iteración, sin embargo, todavía no es lo suficientemente cercano y tampoco ninguno de los valores anteriores de "x" (ni x_1 ni x_2) son aproximadamente iguales al nuevo valor calculado (x_3), por lo que el proceso debe ser repetido.

Para ello, se deben comparar los signos de y_1 y y_3 para saber donde se encuentra la solución. Como en esta iteración son del mismo signo se sabe que la solución se encuentra en el lado derecho, entre x_2 y x_3. Entonces x_1 toma el valor de x_3 y y_1 toma el valor de y_3. Es necesario que y_1 (o y_2) tome el valor de y_3 (el nuevo valor de la función) porque este valor se emplea en el cálculo del nuevo valor de "x" (x_3):

```
>> x1=x3; y1=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967015188
y3 = 8.46552651979E-7
```

Como se puede ver, el valor de la función se aproxima más aún a cero y ahora el nuevo valor de "x" (x_3) es igual al anterior en 4 dígitos, no obstante no son aún lo suficientemente cercanos, por lo que se debe repetir el proceso unas cuantas veces más:

```
>> x1=x3; y1=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967088522
y3 = 1.05511244186E-9
>> x1=x3; y1=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967088614
y3 = 1.31583632879E-12
>> x1=x3; y1=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967088614
y3 = 1.33226762955E-15
```

En esta última iteración los dos últimos valores de "x" (x_3) son iguales en los 12 dígitos que se muestran en los resultados, además, el valor de la función (y_3) es muy cercano a cero (tiene 14 ceros), por lo que el proceso concluye, siendo la solución el último valor de "x" (3.60967088614).

Por lo general, como sucede en el ejemplo, el método de Regula-Falsi requiere menos iteraciones que el método de la Bisección. No obstante, como también se puede ver en el ejemplo, el método suele acercarse a la solución por un solo lado (en el ejemplo todos los valores de la función son siempre positivos) lo que incrementa innecesariamente el número de iteraciones (este aspecto puede ser mejorado modificando ligeramente el método).

8.1.1. Ejercicio

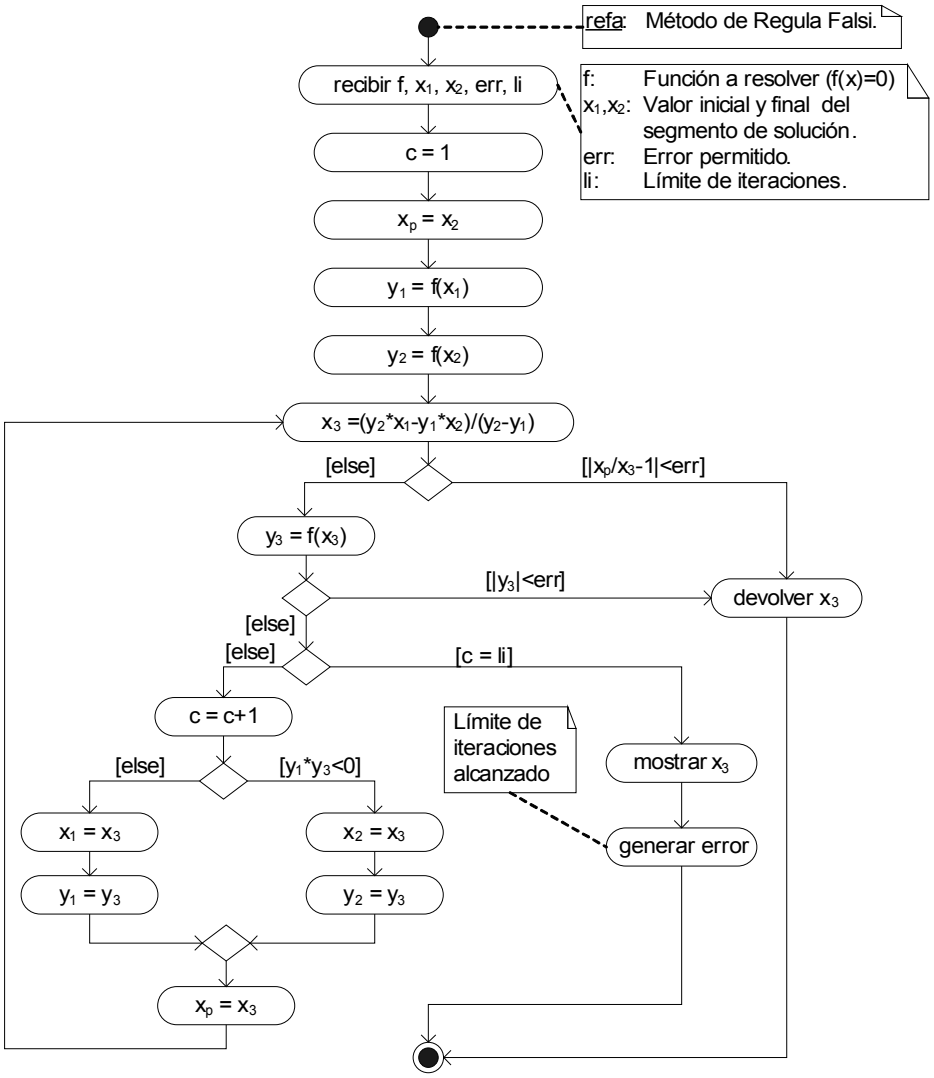
1. **Encuentre la solución de la siguiente ecuación, con 10 dígitos de exactitud, aplicando manualmente el método de Regula-Falsi. Encuentre el segmento de solución con el método incremental (valor inicial -12.1, incremento 1).**

$$f(x)=x^3+2x^2+3x+4=0 \tag{8.5}$$

8.1.2. Algoritmo y código

El algoritmo que automatiza el proceso se presenta en la siguiente página y el código elaborado en base al mismo es:

```
function x3=refa(f,x1,x2,err,li)
  c=1;
  xp=x2;
  y1=f(x1);
  y2=f(x2);
  while 1
    x3=(y2*x1-y1*x2)/(y2-y1);
    %printf("%f: x= %f\n",c,x3);
    if abs(x3/xp-1)<err break; end
    y3=f(x3);
    if abs(y3)<err break; end
    if c++ == li error("¡Error! x=%f\n",x3); end
    if y1*y3<0 x2=x3;y2=y3; else x1=x3;y1=y3; end
    xp=x3;
  end
end
```



Haciendo correr el programa con la ecuación resuelta en el ejemplo manual se obtiene:

```
>> incr1($fx,1.1,1,50)
x1 = 3.1
>> refa($fx,ans,ans+1,1e-13,50)
x3 = 3.60967088614
```

Que es la misma solución obtenida en el ejemplo manual.

8.1.3. Ejercicio

2. Encuentre, con 14 dígitos de exactitud, la primera solución negativa de la siguiente ecuación con el método de Regula-Falsi. Encuentre el segmento de solución con el método incremental (valor inicial 0, incremento -0.5).

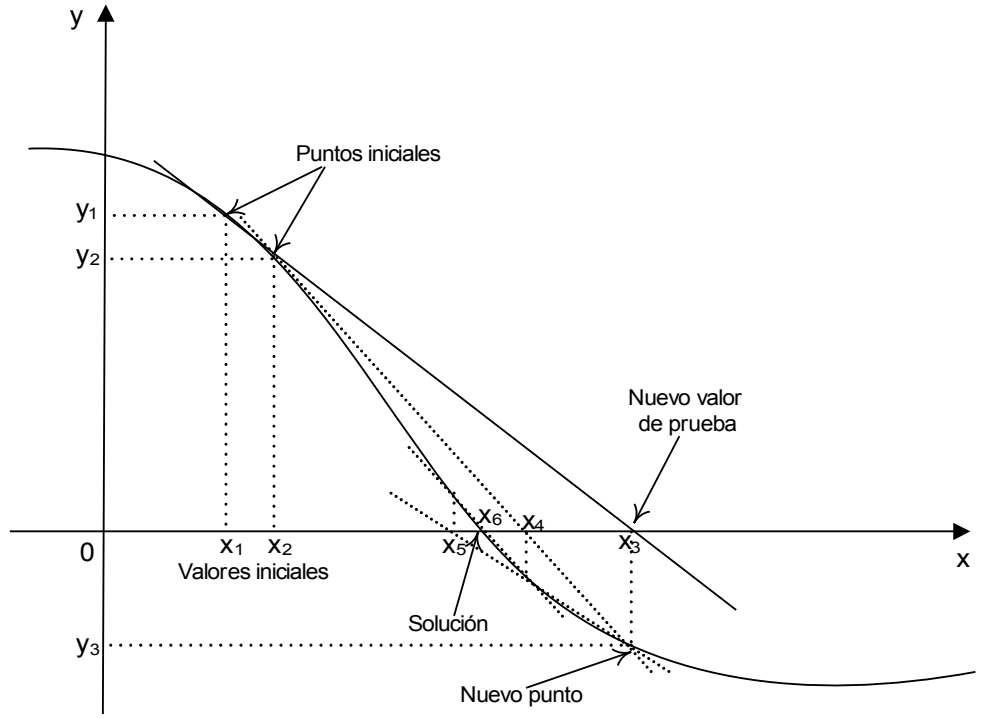
$$f(x)=\cos(x)+\left(1+x^2\right)^{-1}=0 \qquad (8.6)$$

8.2. MÉTODO DE LA SECANTE

Este método, al igual que los dos anteriores, requiere dos valores iniciales, sin embargo, dichos valores pueden no ser los valores inicial y fi-

nal de segmento de solución. Es decir que en este método no se requiere conocer el segmento de solución.

Por lo general los valores iniciales asumidos suelen ser dos valores consecutivos. Con estos valores se calcula un nuevo valor de prueba x_3, interpolando o extrapolando a través de estos dos puntos hasta la intersección con la recta "y=0" (tal como se puede ver en la siguiente figura). Luego se emplean los dos últimos puntos para calcular un nuevo valor de prueba y así sucesivamente hasta que el valor de la función es cercano a cero (o hasta que los dos últimos valores de "x" son casi iguales).



La ecuación para el cálculo del nuevo valor de prueba "x_3" es la misma que la de Regula Falsi (ecuación 8.3).

Para comprender mejor el proceso se resolverá manualmente, una vez más, la ecuación (8.4), asumiendo valores iniciales iguales a 1.1 y 1.2:

```
format 10,12
>> x1=1.1, x2=1.2; y1=fx(x1), y2=fx(x2)
x1 = 1.1
y1 = 2.88405538779
y2 = 2.76912747776
```

Entonces se calcula el nuevo valor de "x" (x_3) y de la función (y_3):

```
>> x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60944734573
y3 = 2.57726112391E-4
```

Como se puede ver, aún en la primera iteración la función se aproxima a cero (tiene 3 ceros), pero aún está lejos de los 12 ceros que debería tener (de acuerdo al número de dígitos que se muestran en los resultados). Entonces se calculan nuevos valores realizando previamente el cambio de variables respectivo:

```
>> x1=x2; x2=x3; y1=y2; y2=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967161684
```

```
y3 = -8.42444866311E-7
```

Ahora la función tiene 6 ceros y los dos últimos valores calculados son iguales en 4 dígitos, por lo que el proceso se repite:

```
>> x1=x2; x2=x3; y1=y2; y2=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967088614
y3 = 4.4542147748E-13
>> x1=x2; x2=x3; y1=y2; y2=y3; x3=(y2*x1-y1*x2)/(y2-y1), y3=fx(x3)
x3 = 3.60967088614
y3 = 4.4408920985E-16
```

En realidad la solución se encuentra ya en la iteración previa, pues la función tenía ya 12 ceros.

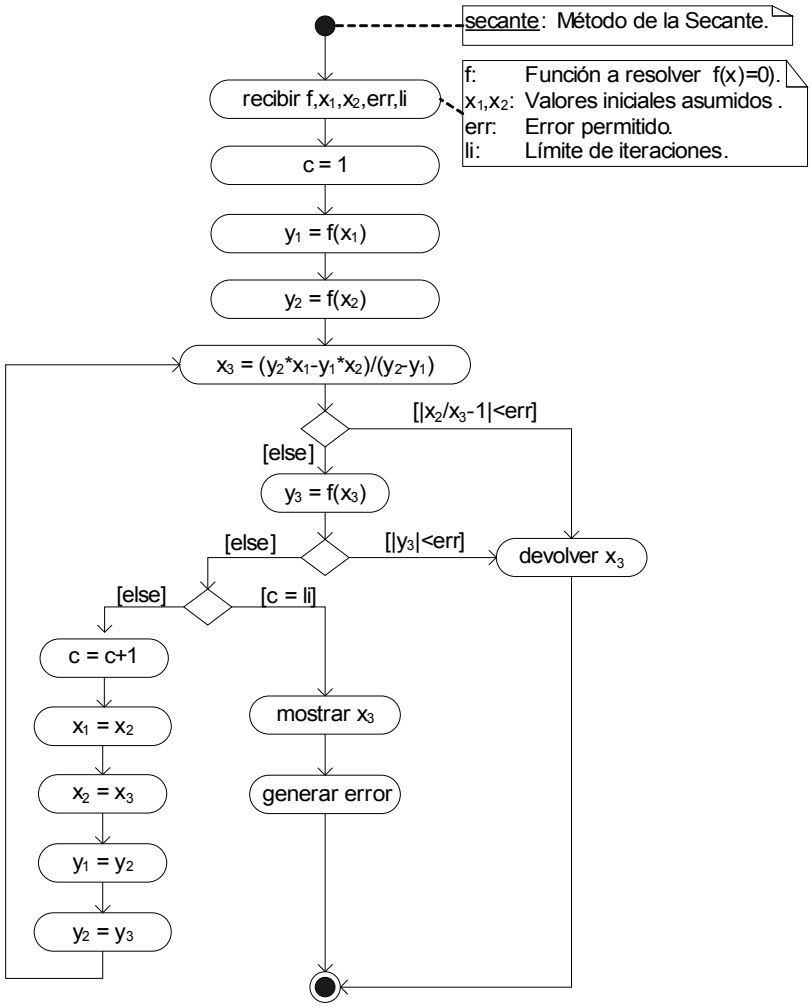
8.2.1. Ejercicios

3. Encuentre, con 11 dígitos de exactitud, la solución de la siguiente ecuación aplicando manualmente el método de Regula-Falsi. Emplee como valores iniciales 10.1 y 10.2.

$$f(x)=2x+1-e^{x}=0 \tag{8.7}$$

8.2.2. Algoritmo y código

El algoritmo que automatiza el proceso es:



Y el código elaborado en base al mismo es:

```
function x3=secante(f,x1,x2,err,li)
  c=1;
  y1=f(x1);
  y2=f(x2);
  while 1
    x3=(y2*x1-y1*x2)/(y2-y1);
    %printf("%f: x= %f\n",c,x3);
    if abs(x2/x3-1)<err break; end
    y3=f(x3);
    if abs(y3)<err break; end
    if c++ == li error("¡Error! x=%f\n",x3); end
    x1=x2;
    x2=x3;
    y1=y2;
    y2=y3;
  end
end
```

Haciendo correr el programa con la ecuación del ejemplo manual se obtiene:

```
secante($fx,1.1,1.2,1e-13,50)
x3 = 3.60967088614
```

Que es el mismo resultado obtenido en el ejemplo manual.

8.2.3. Ejercicio

4. Encuentre la solución de la siguiente función con 13 dígitos de exactitud empleando como valores iniciales 1.1 y 1.2. Muestre los resultados intermedios obtenidos.

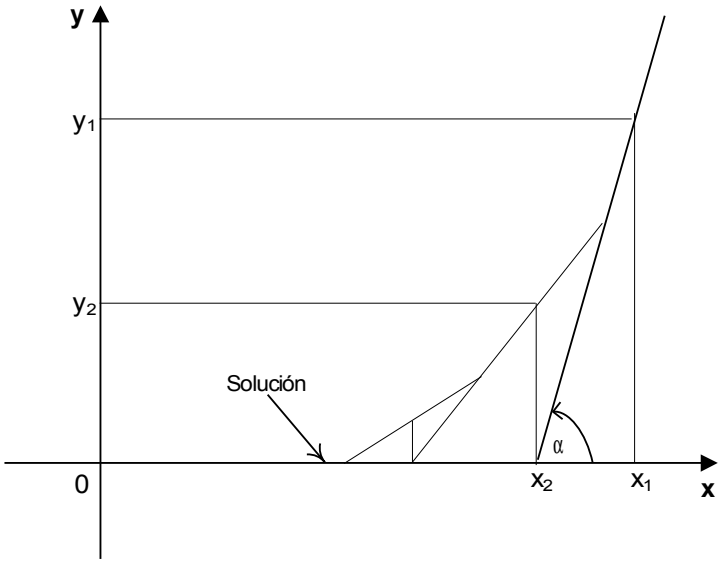
$$
\begin{aligned}
&f(p)=\cos(k\cdot l)\cdot\cosh(k\cdot l)+1=0\\
&k^2=\frac{p}{a}\\
&a^2=\frac{E\cdot I\cdot g}{A\cdot y}\\
&l=120;\quad I=170.6;\quad E=3\times10^6\\
&y=0.066;\quad A=32;\quad g=386;
\end{aligned}
\qquad (8.8)
$$

8.3. MÉTODO DE LA NEWTON - RAPHSON

El método de Newton – Raphson es probablemente el método más empleado en la práctica para resolver ecuaciones algebraicas con una incógnita.

El proceso es el que se presenta en la gráfica de la siguiente página: 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 *y=0*, se encuentra el nuevo valor de prueba *(x_2)*. Si para x_2 la función es casi cero, o si x_2 es igual aproximadamente igual a x_1, se ha encontrado la solución y el proceso concluye, caso contrario se repite.

La ecuación para el cálculo del nuevo valor de prueba *(x_2)* puede ser deducida del triángulo que forman la tangente con los segmentos, donde se cumple que:



$$\tan(\alpha) = f'(x_1) = \frac{f(x_1) - 0}{x_1 - x_2}$$

De donde resulta:

$$x_1 - x_2 = \frac{f(x_1)}{f'(x_1)}$$

$$x_2 = x_1 - \frac{y_1}{f'(x_1)} \qquad (8.9)$$

Que es la ecuación del método de Newton – Raphson. Como se puede ver, esta ecuación implica el cálculo de la derivada de la función.

Si bien en ocasiones el cálculo de la derivada analítica puede ser simple, en otros es muy complejo e incluso imposible (en funciones compuestas por ejemplo), por lo que la mejor alternativa radica en el cálculo numérico de la derivada.

En este método se empleará la fórmula de diferencia central de segundo orden (cuya deducción se verá en un tema posterior):

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} \qquad (8.10)$$

Donde "h" es el incremento de "x" (Δx). En teoría mientras más pequeño sea el valor de "h" más exacto será el resultado obtenido. En la práctica, sin embargo, no es así (debido a errores de redondeo). En este tema se empleará un valor de "h" igual a $x_1 * 10^{-6}$.

El programa que automatiza el cálculo de la ecuación (8.10) es:

```
function r=df(f,x)
  h=x*1e-6;
  r=(f(x+h)-f(x-h))/(2*h);
end
```

Así la derivada de la ecuación (8.4) para x=2.1 es:

```
>> df($fx,2.1)
```

```
r = -1.14785170205
```

Una vez más, para comprender mejor el proceso se resolverá manualmente la ecuación (8.4).

Los valores iniciales y el primer valor de "x" calculado es:

```
>> format 10,12
>> x1=1.1, y1=fx(x1), x2=x1-y1/df($fx,x1)
x1 = 1.1
y1 = 2.88405538779
x2 = 3.60891610415
```

Como x_1 y x_2 no son aproximadamente iguales (y tampoco y_1 es cercano a cero), se repite el proceso hasta que se cumpla una de las dos condiciones:

```
>> x1=x2; y1=fx(x1), x2=x1-y1/df($fx,x1)
y1 = 8.70208574404E-4
x2 = 3.60967088749
>> x1=x2; y1=fx(x1), x2=x1-y1/df($fx,x1)
y1 = -1.55385171396E-9
x2 = 3.60967088614
>> x1=x2; y1=fx(x1), x2=x1-y1/df($fx,x1)
y1 = 4.4408920985E-16
x2 = 3.60967088614
```

Como se ve en esta iteración los dos últimos valores de "x" son iguales (en los 12 dígitos con que se muestran los resultados) y la función es muy cercana a cero (tiene 15 ceros). Entonces la solución es: 3.60967088614.

8.3.1. Ejercicio

5. Encuentre la solución de la siguiente ecuación, con 11 dígitos de exactitud, aplicando manualmente el método de Newton-Raphson (x_1 = 5.1).

$$f(z)=\ln(z)+e^{z+3}-z^{3.1}-42=0$$

8.3.2. Algoritmo y código

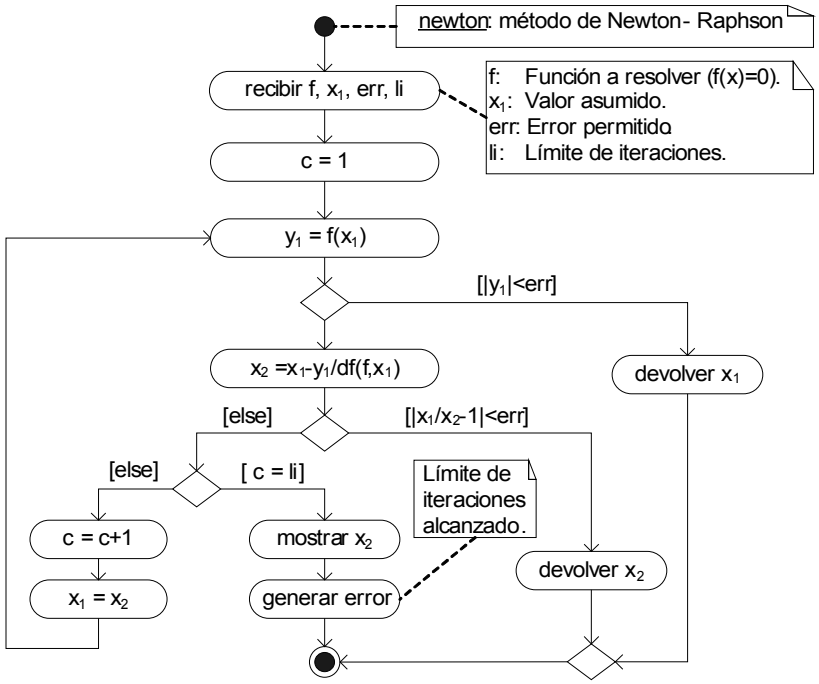
El algoritmo que automatiza el proceso se presenta en la siguiente página y el código elaborado en base al mismo es:

```
function x2=newton(f,x1,err,li)
  c=1; x2=x1;
  while 1
    y1=f(x1);
    if abs(y1)<err break; end
    x2=x1-y1/df(f,x1);
    if abs(x1/x2-1)<err break; end
    if c++ == li error("¡Error! x=%f\n",x2); end
    x1=x2;
  end
end
```

Haciendo correr el programa con la función resuelta en el ejemplo manual se obtiene:

```
>> newton($fx,1.1,1e-13,30)
x2 = 3.60967088614
```

Que es el resultado obtenido en dicho ejemplo. Como de costumbre para mostrar resultados intermedios se añade un "printf" después de calcular el último varo de "x" (después de: x2=x1-y1/df($fx,x1);).



8.3.3. Ejercicios

6. Encuentre las dos soluciones positivas del siguiente sistema de ecuaciones (como una función de x), con 12 dígitos de exactitud (valores iniciales 1.8 y 5.1). Debe mostrar los resultados intermedios.

$$3x^{2.1} - 5y = 7.0$$
$$y^{1.2} + 4z = 14.3$$
$$x + y^2 + z^2 = 14.0$$

7. Encuentre las dos soluciones del siguiente sistema de ecuaciones (como una función de x), con 10 dígitos de exactitud (valores iniciales: -30.11 y 7.1).

$$15y + 20x^2 - x^3 = 1500$$
$$9z - 1.5x^2 + e^{\frac{x}{2}} = 300$$
$$y^2 - z^3 - 5x = 166.81$$

9. ECUACIONES POLINOMIALES 1

Con los métodos estudiados hasta ahora se puede encontrar (una a la vez) las soluciones reales de una ecuación polinomial de la forma:

$$P_n(x)=a_1x^n+a_2x^{n-1}+a_3x^{(n-2)}+\cdots+a_nx+a_{n+1}=0 \tag{9.1}$$

Sin embargo, para encontrar todas las soluciones, tanto reales como imaginarias, existen otros métodos, algunos de los cuales serán estudiados en el presente tema.

9.1. ECUACIÓN CUADRÁTICA

La ecuación polinomial más simple es la ecuación cuadrática:

$$P_2(x)=a_1x^2+a_2x+a_3=0 \tag{9.2}$$

Que puede ser resuelta directamente, como se ha hecho en el tema 3, aplicando la solución general:

$$x_{1,2}=\frac{-a_2\pm\sqrt{a_2^2-4\cdot a_1\cdot a_3}}{2\cdot a_1} \tag{9.3}$$

En este tema se vuelve a aplicar esta solución, pero aprovechando el hecho de que Jasymca trabaja tanto con números reales como con imaginarios y que en la ecuación cuadrática se cumple que:

$$x_1\cdot x_2=\frac{a_3}{a_1} \tag{9.4}$$

Entonces, si se calcula el valor de la expresión:

$$q=\frac{a_2+signo(a_2)\cdot\sqrt{a_2^2-4*a_1*a_3}}{-2} \tag{9.5}$$

Que corresponde a la suma de los términos en el numerador de la ecuación (9.3) dividido entre 2, resulta claro que la primera solución puede ser calculada con:

$$x_1=\frac{q}{a_1} \tag{9.6}$$

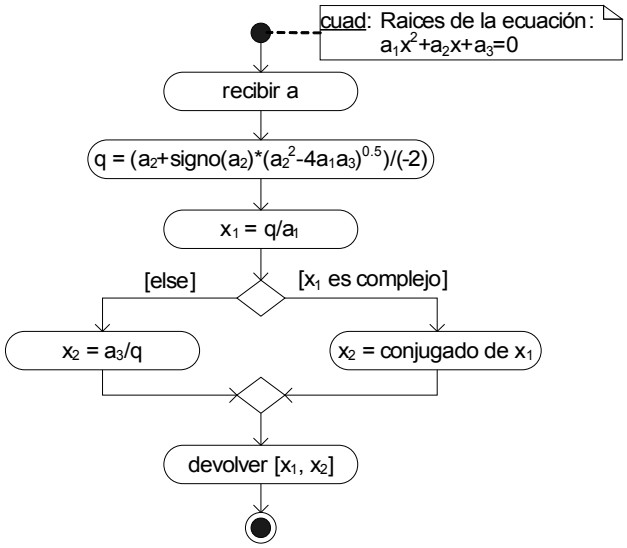
Entonces, la segunda solución se calcula a partir de las ecuaciones (9.4) y (9.5):

$$x_2=\frac{a_3}{a_1\cdot x_1}=\frac{a_3}{\frac{q}{a_1}\cdot a_1}$$

$$x_2=\frac{a_3}{q} \tag{9.7}$$

El algoritmo que aplica estas expresiones se presenta en la siguiente página, siendo el código respectivo:

```
function r=cuad(a)
  q=(a(2)+sign(a(2))*sqrt(a(2)^2-4*a(1)*(a(3))))/(-2);
  x1=q/a(1);
  if (imagpart(x1)~=0) x2=conj(x1); else x2=a(3)/q; end
```




```
  r=[x1 x2];
end
```

Para resolver una ecuación cuadrática con este programa, simplemente se le manda un vector con los coeficientes de la ecuación. Por ejemplo, para resolver la ecuación:

$$x^2-8x+7=0$$

Se escribe:

```
>> cuad([1,-8,7])
r = [ 7.0  1.0 ]
```

Por lo tanto, las soluciones son x_1=7.0 y x_2=1.0. Estos resultados pueden ser corroborados con la función "roots" de Jasymca:

```
>> roots([1,-8,7])
ans = [ 7  1 ]
```

9.1.1. Ejercicio

1. Encuentre las soluciones de las siguientes ecuaciones empleando tanto "cuad" como "roots".

$$x^2+2x+3=0$$
$$x^2-15x+50=0$$
$$x^2+14x+49=0$$

9.2. ECUACIÓN CÚBICA

Las raíces de la ecuación cúbica pueden ser calculadas aplicando la solución propuesta por primera vez, en el año 1545, por Cardano, razón por la cual se conoce también como el método de Cardano. Para ello es necesario que la ecuación general:

$$a_1x^3+a_2x^2+a_3x+a_4=0 \qquad (9.8)$$

Sea llevada a la forma:

$$x^3+a_1x^2+a_2x+a_3=0 \qquad (9.9)$$

Lo que se consigue simplemente dividiendo todos los coeficientes entre el

primero (a_1). Una vez hecha esta operación se calculan los siguientes valores:

$$Q=\frac{3a_2-a_1^2}{9} \tag{9.10}$$

$$R=\frac{9a_1a_2-27a_3-2a_1^3}{54} \tag{9.11}$$

$$S=\sqrt[3]{R+\sqrt{Q^3+R^2}} \tag{9.12}$$

$$T=\sqrt[3]{R-\sqrt{Q^3+R^2}} \tag{9.13}$$

Con estos valores se calculan las tres soluciones:

$$x_1=S+T-\frac{1}{3}a_1 \tag{9.14}$$

$$x_2=-\frac{1}{2}(S+T)-\frac{1}{3}a_1+\frac{1}{2}\sqrt{-3}(S-T) \tag{9.15}$$

$$x_2=-\frac{1}{2}(S+T)-\frac{1}{3}a_1-\frac{1}{2}\sqrt{-3}(S-T) \tag{9.16}$$

Una vez más se aprovecha el hecho de que Jasymca trabaja tanto con números reales como imaginarios, por lo que estas expresiones pueden ser programadas casi directamente, con excepción del cálculo de la raíz cúbica, pues Jasymca no cuenta con una función para la misma, por lo que primero se crea una función para dicho fin.

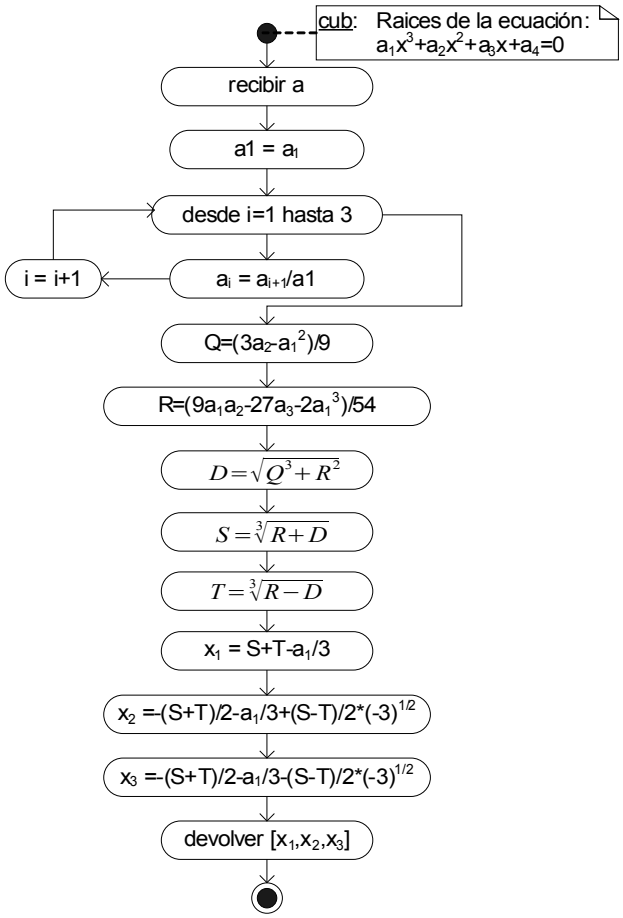
El algoritmo para el cálculo de la raíz cúbica es:

Siendo el código respectivo:

```
function r=cbrt(x)
  if realpart(x)~=0 & x<0
    r=-(-x)^(1/3);
  else
    r=x^(1/3);
  end
end
```

Una vez que se cuenta con la función para el cálculo de la raíz cúbica las ecuaciones del método pueden ser programadas directamente tal como se muestra en el algoritmo de la siguiente página, siendo el código respectivo:

```
function r=cub(a)
  a1=a(1);
  for i=1:3 a(i)=a(i+1)/a1; end
```




```
  Q=(3*a(2)-a(1)^2)/9;
  R=(9*a(1)*a(2)-27*a(3)-2*a(1)^3)/54;
  D=sqrt(Q^3+R^2);
  S=cbrt(R+D);
  T=cbrt(R-D);
  x1=S+T-a(1)/3; C=-(S+T)/2-a(1)/3; D=(S-T)/2*sqrt(-3);
  x2=C+D;
  x3=C-D;
  r=[x1,x2,x3];
end
```

Así, para encontrar las soluciones de la ecuación:

$$x^3+3x^2+7x+9=0$$

Se escribe:

```
>> cub([1,3,7,9])

r = [ -1.8477075981395665  -0.5761462009302167+2.1304826047066574i
      -0.5761462009302167-2.1304826047066574i ]
```

Por lo tanto esta ecuación tienen una solución real (-1.8477075981395665) y dos imaginarias (-0.5761462009302167+/-2.1304826047066574i). Al igual que con "cuad", los resultados que pueden ser corroborados con "roots":

```
>> roots([1,3,7,9])
ans = [ -1.8477075981395665  -0.5761462009302167-2.1304826047066658i
        -0.5761462009302167+2.1304826047066658i ]
```

9.2.1. Ejercicio

2. Encuentre las soluciones de las siguientes ecuaciones cúbicas empleando tanto "cub" como "roots".

$$x^3+2x^2+3x+4=0$$
$$x^3-6x^2+11x-6=0$$
$$x^3-22x^2+145x-300=0$$
$$x^3+21x^2+147+343=0$$

9.3. MÉTODO QD (QUOTIENT DIVISOR)

Este es propiamente el primer método numérico que se estudia para la resolución de ecuaciones polinomiales. El método "QD", permite encontrar todas las soluciones (reales e imaginarias) sin requerir valores iniciales, lo que constituye una ventaja cuando se trabaja con métodos iterativos.

Sin embargo tiene la desventaja de requerir un elevado número de iteraciones (pudiendo superar fácilmente las 1000) para encontrar resultados con un error aceptable. Por ello en la práctica, este método se emplea casi exclusivamente para encontrar iniciales, luego, una vez encontrados dichos valores, se recurre a otros métodos para encontrar soluciones más exactas.

Tomando en cuenta el polinomio general (9.1), en este método se calculan primero los parámetros "p" y "d" con las siguientes expresiones:

$$p_1=-\frac{a_2}{a_1};\ p_i=0 \quad \{i=2\rightarrow n \tag{9.17}$$

$$d_i=\frac{a_{i+2}}{a_{i+1}} \quad \{i=1\rightarrow n-1 \tag{9.18}$$

Luego, con estos parámetros, se calculan otros dos parámetros "q" y "e":

$$\begin{aligned} q_1&=d_1+p_1 \\ q_i&=d_i-d_{i-1}+p_i \quad \{i=2\rightarrow n-1 \\ q_n&=p_n-d_{n-1} \end{aligned} \tag{9.19}$$

$$e_i=\frac{q_{i+1}}{q_i}d_i \quad \left\{i=1\rightarrow n-1 \right. \tag{9.20}$$

Si los parámetros "e" son cercanos a cero, el proceso concluye, caso contrario se repite haciendo que "p=q" y "d=e". Cuando el proceso concluye las soluciones aproximadas se encuentran en el parámetro "q". No obstante, si existen soluciones complejas algunos de los valores de "e" no convergen hacia cero, en ese caso las soluciones complejas se encuentran resolviendo la ecuación cuadrática:

$$x^2-rx-s=0 \tag{9.21}$$

Donde "r" y "s" se calculan con:

$$\begin{aligned} r&=q_i+q_{i+1} \\ s&=-p_iq_{i+1} \end{aligned} \tag{9.22}$$

Siendo "i" el índice del elemento "e_i" que no converge hacia cero.

El proceso en sí es muy sencillo: una vez calculados los parámetros "p" y "d" (con las ecuaciones 9.17 y 9.18), se aplica repetidamente las ecuaciones (9.19) y (9.20) hasta que los valores de "e" (o algunos de ellos) sean cercanos a cero.

Para comprender mejor el proceso, se encontrarán las soluciones del siguiente polinomio aplicando el método manualmente:

$$P_4(x) = 128x^4 - 256x^3 + 160x^2 - 32x + 1 = 0$$

El vector de coeficientes y el grado del polinomio son:

```
>> a=[128,-256,160,-32,1]; n=4
```

Se comienza calculando los parámetros "p" y "d" (ecuaciones 9.17 y 9.18):

```
>> p(1)=-a(2)/a(1); for i=2:n p(i)=0; end p,
   for i=1:n-1 d(i)=a(i+2)/a(i+1); end d
ans = [ 2  0  0  0 ]
ans = [ -0.625  -0.2  -3.125E-2 ]
```

Con estos valores se calculan los parámetros "q" y "e" (ecuaciones 9.19 y 9.20):

```
>> q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1.375  0.425  0.16875  3.125E-2 ]
ans = [ -0.19318  -7.9412E-2  -5.787E-3 ]
```

Como los valores de "e" no son cercanos a cero, se realiza el cambio de variables y se repite el proceso:

```
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1.1818  0.53877  0.24237  3.7037E-2 ]
ans = [ -8.8068E-2  -3.5725E-2  -8.8431E-4 ]
```

Ahora los valores de "e" comienzan a acercarse a cero. Repitiendo el proceso unas cuantas veces más se obtiene:

```
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1.0938  0.59111  0.27722  3.7921E-2 ]
ans = [ -4.7596E-2  -1.6754E-2  -1.2097E-4 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1.0462  0.62196  0.29385  3.8042E-2 ]
ans = [ -2.8297E-2  -7.9155E-3  -1.5661E-5 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1.0179  0.64234  0.30175  3.8058E-2 ]
ans = [ -1.7857E-2  -3.7184E-3  -1.9752E-6 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 1  0.65648  0.30546  3.806E-2 ]
ans = [ -1.1723E-2  -1.7302E-3  -2.4611E-7 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.98828  0.66647  0.30719  3.806E-2 ]
ans = [ -7.9055E-3  -7.9751E-4  -3.0492E-8 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.98037  0.67358  0.30799  3.806E-2 ]
```

```
ans = [ -5.4316E-3  -3.6466E-4  -3.768E-9 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.97494  0.67864  0.30836  3.806E-2 ]
ans = [ -3.7809E-3  -1.6569E-4  -4.6509E-10 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.97116  0.68226  0.30852  3.806E-2 ]
ans = [ -2.6561E-3  -7.4927E-5  -5.7375E-11 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.9685  0.68484  0.3086  3.806E-2 ]
ans = [ -1.8782E-3  -3.3763E-5  -7.0762E-12 ]
>> p=q(:); d=e(:); q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i);
   end; q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 0.96662  0.68668  0.30863  3.806E-2 ]
ans = [ -1.3342E-3  -1.5175E-5  -8.7263E-13 ]
```

Como se puede ver, el último valor de "e" es casi cero (tiene 12 puntos después del cero), mientras que el primero sólo tiene dos ceros después del punto. Este es el comportamiento normal de método, los últimos valores de "e" convergen rápidamente pero los primeros requieren decenas, cientos e inclusive miles de iteraciones para lograr convergencia y esta es justamente la razón por la cual el método se emplea sólo para obtener valores aproximados.

Deteniendo el proceso en este punto las soluciones (raíces) aproximadas del polinomio son los valores del vector "q", es decir:

```
[ 0.96662  0.68668  0.30863  3.806E-2 ]
```

Valores que pueden ser comparados con los resultados más exactos obtenidos con "roots":

```
>> roots(a)
ans = [ 0.69134  0.96194  3.806E-2  0.30866 ]
```

Como se puede ver, sólo la última solución es exacta, no obstante, las otras soluciones son lo suficientemente cercanas como para constituir muy buenos valores iniciales.

Como otro ejemplo se encontrarán las soluciones del siguiente polinomio:

$$P_4(x)=x^4-6x^3+12x^2-19x+12=0$$

El vector de coeficientes y el orden son:

```
>> a=[1,-6,12,-19,12]; n=4;
```

Se calculan los valores de "p" y "d":

```
>> p(1)=-a(2)/a(1); for i=2:n p(i)=0; end p,
   for i=1:n-1 d(i)=a(i+2)/a(i+1); end d
ans = [ 6  0  0  0 ]
ans = [ -2  -1.5833  -0.63158 ]
```

Y con estos valores se calculan los correspondientes de "q" y "e":

```
>> q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4  0.41667  0.95175  0.63158 ]
ans = [ -0.20833  -3.6167  -0.41911 ]
```

Y como los valores de "e" todavía no se acercan a cero, se repite el proceso (haciendo los cambios de variables respectivos) unas cuantas veces más:

```
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 3.7917  -2.9917  4.1493  1.0507 ]
ans = [ 0.16438  5.0162  -0.10613 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 3.956  1.8601  -0.97298  1.1568 ]
ans = [ 7.7289E-2  -2.6238  0.12618 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4.0333  -0.841  1.777  1.0306 ]
ans = [ -1.6116E-2  5.5441  7.3182E-2 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4.0172  4.7192  -3.6939  0.95746 ]
ans = [ -1.8932E-2  -4.3396  -1.8969E-2 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 3.9983  0.3986  0.62669  0.97642 ]
ans = [ -1.8874E-3  -6.8229  -2.9554E-2 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 3.9964  -6.4224  7.4201  1.006 ]
ans = [ 3.0331E-3  7.8828  -4.0068E-3 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 3.9994  1.4573  -0.4667  1.01 ]
ans = [ 1.1052E-3  -2.5245  8.6712E-3 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4.0005  -1.0683  2.0664  1.0013 ]
ans = [ -2.9513E-4  4.8832  4.2017E-3 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4.0002  3.8152  -2.8125  0.99711 ]
ans = [ -2.8147E-4  -3.5999  -1.4896E-3 ]
>> p=q(:);d=e(:);q(1)=d(1)+p(1); for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end;
   q(n)=p(n)-d(n-1); q, for i=1:n-1 e(i)=q(i+1)*d(i)/q(i); end; e
ans = [ 4  0.2156  0.78584  0.9986 ]
ans = [ -1.5171E-5  -13.121  -1.8929E-3 ]
```

Como se puede ver en ese caso dos de los valores de "e" (el primero y el tercero) convergen lentamente hacia cero, mientras que uno (el segundo) no. Como se dijo, cuando esto sucede se sabe que existen soluciones complejas, que en este caso serían la segunda y tercera (puesto que es el segundo elemento el que no converge hacia cero), por lo tanto las dos soluciones reales son el primer y cuarto valores de "q", es decir:

```
4  0.9986
```

Las dos soluciones complejas se calculan aplicando las ecuaciones (9.21) y (9.22), donde el valor de "i" es 2:

```
>> i=2; r=q(i)+q(i+1); s=-p(i)*q(i+1); cuad([1,-r,-s])
r = [ 0.50072+1.6575i  0.50072-1.6575i ]
```

Estas cuatro soluciones pueden ser comparadas con los resultados más exactos obtenidos con "root":

```
>> roots(a)
ans = [ 0.5-1.6583i  0.5+1.6583i  1  4 ]
```

Una vez más sólo una de las soluciones es exacta, pero las otras constituyen muy buenos valores iniciales.

9.3.1. Ejercicio

3. Encuentre las soluciones reales e imaginarias de las siguientes ecuaciones aplicando manualmente el método QD. El proceso debe concluir cuando por lo menos uno de los valores de "e" tenga 6 ceros, emplee "root" para juzgar la exactitud de los resultados obtenidos.

$$x^4-27x^3+246x^2-883x+999=0 \qquad (9.23)$$
$$x^4+x^3-3x^2+x-5=0$$

9.3.2. Algoritmo y código

Como queda claro en los ejemplos y ejercicios del anterior acápite, la aplicación manual del método QD es bastante sencilla, sin embargo, su automatización no lo es tanto.

Para programar el método QD, además del proceso propiamente, es necesario tomar en cuenta algunos casos.

En primer lugar se deben evitar las divisiones entre cero y algunas situaciones en las cuales se presentan dichas divisiones son: a) Uno de los coeficientes es cero, b) Tres o más coeficientes consecutivos son iguales y el grado del polinomio es menor a 5, c) Tres o más coeficientes están igualmente espaciados y d) El espaciamiento de dos coeficientes es el cuadrado de los dos coeficientes anteriores.

En segundo lugar, y como se ha visto en los ejemplos, la convergencia hacia cero depende de que las soluciones sean reales o complejas, además las últimas raíces convergen más rápidamente que las primeras por lo que deben tomarse en cuenta estos factores para determinar si se ha encontrado o no la solución con el error permitido.

Para resolver el primer problema se debe realizar un cambio de variable. Por comodidad, el cambio de variable más utilizado es y=x-1. Por supuesto, cuando se realiza un cambio de variable, a las soluciones calculadas se les debe sumar la constante que se restó (normalmente 1) para obtener las soluciones correctas.

Antes de pasar al algoritmo y escribir el código es necesario hacer notar que en Jasymca los vectores (y matrices) son punteros, es decir direcciones de memoria. Para comprender el por qué es importante este aspecto, considere lo que sucede con el siguiente vector:

```
>> v1=[1,3,7,9,12,15];
```

Del cual se quiere hacer una copia en otras dos variables:

```
>> v3=v2=v1;
```

Entonces se asigna un nuevo valor al quinto elemento del "nuevo" vector "v3":

```
>> v3(5)=-26;
```

Sin embargo, si se imprimen los valores de "v1", "v2" y "v3" se obtiene:

```
>> v1,v2,v3
v3 = [ 1  3  7  9  -26  15 ]
v3 = [ 1  3  7  9  -26  15 ]
ans = [ 1  3  7  9  -26  15 ]
```

Como se puede ver “v1”, “v2” y “v3” son en realidad el mismo vector, sólo que tiene tres nombres. Cuando se asigna un vector a otro lo único que en realidad se asigna es la dirección de memoria donde se encuentran los elementos del vector, por lo tanto, la nueva variable apunta a los mismo datos que la variable original, por eso cualquier cambio en la nueva variable es también un cambio en la variable original (y viceversa) porque en realidad se trata de los mismos datos.

En ocasiones se quiere el anterior comportamiento, es decir darle un nuevo nombre a los mismos datos, pero si lo que se quiere es crear una copia del vector, entonces se deben extraer los valores del vector (o matriz) y asignar dichos valores a la variable. Como se sabe, para extraer un elemento simplemente se escribe el índice del mismo entre paréntesis, pero para extraer todos los elementos se escribe dos puntos (“:”) en lugar del índice:

```
>> v1(:)
ans = [ 1  3  7  9  -26  15 ]
```

Por lo tanto para crear una copia del vector “v1” en la variable “v2” se escribe:

```
>> v2=v1(:);
```

Si ahora se cambia uno de los datos del vector “v2”:

```
>> v2(2)=-45;
```

Dicho cambio no afecta al vector original:

```
>> v1,v2
v3 = [ 1  3  7  9  -26  15 ]
ans = [ 1  -45  7  9  -26  15 ]
```

También es posible extraer partes del vector, por ejemplo los elementos del 2 al 5:

```
>> v1(2:5)
ans = [ 3  7  9  -26 ]
```

O elementos específicos, por ejemplos los elementos 2, 4 y 6:

```
>> v1([2,4,6])
ans = [ 3  9  15 ]
```

Habiendo hecho este breve repaso se procede a automatizar el método QD. El algoritmo general se presenta en la siguiente página y en el mismo, los casos antes comentados, se han agrupado en actividades las que se presentan en diagramas separados. En el diagrama una actividad se identifica con el icono de una acción llamando a otra.

Como se puede ver en el algoritmo se ha definido la variable “const”, con un valor igual a 1, para realizar el cambio de variable, normalmente con este valor se consiguen buenos resultados, pero esta variable permite probar otros valores en los casos donde con 1 no se consigan buenos resultados.

El código es el siguiente:

```
function x=qd(a,err,li)
  n=length(a)-1; c=1; const=1;
  %Se determina si es necesario o no el cambio de variable
  while 1
    scv=0;
    for i=1:n if abs(a(i))<1e-9 scv=1; break; end end
    if scv break; end
    if abs(a(1)-a(2))<1e-3 scv=1; break; end
    if scv break; end
```

```
    for i=1:n-1
      if abs(a(i)-a(i+1))<1e-3 & abs(a(i+1)-a(i+2))<1e-3 scv=1; break;
end
    end
    if scv break; end
    d1=a(2)-a(1);
    for i=2:n+1
      d2=a(i)-a(i-1);
      if abs(d1-d2)<1e-3 & abs(d1^2-d2)<1e-3 scv=1; break; end
      d1=d2;
    end
    break;
  end
  %Se realiza el cambio de variable si "scv" es 1
  if scv
    for k=n+1:-1:2
      for i=2:k a(i)=a(i)+a(i-1); end
    end
  end
  p=0; q=0; x=0; p(1)=-a(2)/a(1);
  for i=2:n p(i)=0; end
  for i=1:n-1 d(i)=a(i+2)/a(i+1); end
  while 1
    q(1)=d(1)+p(1);
    for i=2:n-1 q(i)=d(i)-d(i-1)+p(i); end
    q(n)=p(n)-d(n-1);
    for i=1:n-1 e(i)=q(i+1)/q(i)*d(i); end
    %Se determina si ya se ha alcanzado la exactitud requerida
    ss=0;
    for i=1:2 if abs(e(i))<err ss=1; break; end end
    if ss break; end
    %Se determina si se ha alcanzado el límite de iteraciones
    if c==li error("¡Error! e=%f\n",e); end
    c++; p=q(:); d=e(:);
  end
  %Se cálculan los resultados
  i=0;
  while ++i <= n-1
    if abs(e(i))>1e-2
      r=q(i)+q(i+1); s=-p(i)*q(i+1);
      x([i,i+1])=cuad([1,-r,-s]); i++;
    else
      x(i)=q(i);
    end
  end
  if i==n x(i)=q(i); end
  %Se ajustan los resultados si se ha hecho un cambio de variable
  if scv for i=1:n x(i)=x(i)+const; end end
```

```
end
```

Haciendo correr el programa con las ecuaciones de los ejemplos manuales, y para el mismo error permitido, se obtiene:

```
qd([128,-256,160,-32,1],2e-5,100)
x = [ 0.96662  0.68668  0.30863  3.806E-2 ]
>> qd([1,-6,12,-19,12],2e-5,100)
x = [ 4  0.50072+1.6575i  0.50072-1.6575i  0.9986 ]
```

Que son los mismos resultados obtenidos en dichos ejemplos.

9.3.3. Ejercicio

4. Encuentre las soluciones reales y complejas de las siguientes ecuaciones con un error permitido igual a 10^{-9}, 10^{-10} y 10^{-8} respectivamente. Compare los resultados con los obtenidos con "root".

$$x^8-7.3x^7-41.44x^6+373.474x^5+115.908x^4-4822.15x^3+6985.17x^2+5589.66x-10901.5=0$$

$$x^8-7x^7+11x^6+41x^5-183x^4+231x^3-21x^2-265x-150=0$$

$$x^8-60x^7+1554x^6-22680x^5+203889x^4-1155420x^3+4028156x^2-7893840x+6652800=0$$

10. ECUACIONES POLINOMIALES 2

En este tema se estudian dos métodos más para resolver ecuaciones polinomiales de la forma:

$$P_n(x)=a_1x^n+a_2x^{n-1}+a_3x^{(n-2)}+\cdots+a_nx+a_{n+1}=0 \qquad (10.1)$$

10.1. MÉTODO DE NEWTON-RAPHSON

El método de *Newton – Raphson* (al igual que los otros métodos estudiados en temas anteriores) permite encontrar una de las soluciones reales de una ecuación polinomial y en el caso de Jasymca, es posible encontrar también una solución imaginaria (debido a que Jasymca permite trabajar con números complejos).

Si bien es posible emplear directamente el programa elaborado previamente, en el caso específico de las ecuaciones polinomiales resulta más eficiente calcular el valor del polinomio y el de su derivada mediante la división sintética.

10.1.1. División sintética

La división sintética es el modo más eficiente para calcular el valor de un polinomio y el de su derivada. Por ejemplo, si se quiere calcular el valor del siguiente polinomio y el de su derivada para "x=2" (x_1=2):

$$P_3(x)=x^3+x^2-3x-3=0$$

Se divide el polinomio entre "x-2" y el polinomio resultante se vuelve a dividir entre "x-2":

$$\begin{array}{l|llll} x_1=2 & 1 & 1 & -3 & -3 \\ & & 2 & 6 & 6 \\ \hline & 1 & 3 & 3 & 3 \leftarrow \text{valor de la función: f(2)} \\ & & 2 & 10 & \\ \hline & 1 & 5 & 13 & \leftarrow \text{derivada de la función: f'(2)} \end{array}$$

De esa manera se obtienen el valor de la función y el de su derivada, lo que resulta muy conveniente para el método de Newton que requiere de ambos valores.

Como se puede deducir fácilmente del anterior ejemplo, si "*a*" son los coeficientes del polinomio original, los coeficientes resultantes de la división sintética: "*b*" se calculan con:

$$\begin{aligned} b_1&=a_1 \\ b_i&=a_i+b_{i-1}\cdot x_1 \quad \{i=2\rightarrow n+1 \end{aligned} \qquad (10.2)$$

Donde "n" es el grado del polinomio. El programa que automatiza este cálculo es:

```
function b=divsin1(a,x1)
  b=0; b(1)=a(1);
  for i=2:length(a) b(i)=a(i)+b(i-1)*x1; end
end
```

Llamando al módulo con los datos del ejemplo manual se obtiene:

```
>> divsin1([1,1,-3,-3],2)
```

```
b = [ 1  3  3  3 ]
```

Que son los mismos resultados obtenidos en el ejemplo manual. Llamando nuevamente a "divsin1" con los primeros tres elementos del vector resultante se obtiene:

```
>> divsin1(ans(1:3),2)
b = [ 1  5  13 ]
```

Que también concuerda con los resultados del ejemplo manual. En Jasymca, el valor del polinomio puede ser calculado también con "polyval":

```
>> polyval([1,1,-3,-3],2)
y = 3
```

Es posible también calcular el cociente y el residuo con "divide", creando una variable simbólica con "syms":

```
>> syms x
>> divide(x^3+x^2-3*x-3,x-2)
ans = [ x^2+3*x+3  3 ]
```

En este resultado el residuo es el número 3 y el cociente es "$x^3+3x+3=0$", que es una solución analítica y cuyos coeficientes concuerdan con los obtenidos por división sintética:

```
>> coeff(ans(1),x,2:-1:0)
ans = [ 1  3  3 ]
```

Donde "coeff" es la función que permite extraer los coeficientes de una ecuación.

Una vez repasado el procedimiento de la división sintética, se aplicará el método de Newton Raphson para encontrar, con 8 dígitos de precisión, una de las soluciones de la ecuación:

$$P_4(x)=x^4-6x^3+12x^2-19x+12=0$$

Se empleará como valor inicial "x1=3.1". Como se dijo, el método en si no cambia, sólo cambia la forma en que se calculan el valor de la función y el de su derivada.

Los valores conocidos son:

```
>> format 10,8
>> a=[1,-6,12,-19,12]; n=length(a)-1; x1=3.1;
```

Con el valor inicial asumido se calcula el valor de la función y el de su derivada:

```
>> b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n)
ans = -17.9739
ans = 1.584
```

Y con estos valores se calcula el nuevo valor de "x":

```
>> x2=x1-b(n+1)/c(n)
x2 = 14.447159
```

Como ocurre frecuentemente en la primera iteración, ni el valor de la función (b_{n+1}) es cercano a cero, ni el nuevo valor de "x" (x2) es igual al valor asumido (x1), por lo que el proceso se repite hasta que se cumpla una de estas dos condiciones:

```
>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 27713.855
>> x2=x1-b(n+1)/c(n)
```

```
x2 = 11.236725

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 8743.5023
>> x2=x1-b(n+1)/c(n)
x2 = 8.8432805

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 2748.7668
>> x2=x1-b(n+1)/c(n)
x2 = 7.0720304

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 856.96959
>> x2=x1-b(n+1)/c(n)
x2 = 5.7838868

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 261.73048
>> x2=x1-b(n+1)/c(n)
x2 = 4.8863647

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 75.750423
>> x2=x1-b(n+1)/c(n)
x2 = 4.3259685

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 18.851429
>> x2=x1-b(n+1)/c(n)
x2 = 4.0633989

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 3.0002159
>> x2=x1-b(n+1)/c(n)
x2 = 4.0030158

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 0.13603897
>> x2=x1-b(n+1)/c(n)
x2 = 4.0000073

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 3.2639687E-4
>> x2=x1-b(n+1)/c(n)
x2 = 4

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 1.893925E-9
>> x2=x1-b(n+1)/c(n)
x2 = 4
```

En esta última iteración el valor de "x" se repiten (en los 8 dígitos empleados en el cálculo), igualmente el valor de la función tiene 8 ceros después del punto, por lo que el proceso concluye siendo la solución "x=4".

Como Jasymca permite trabajar con números complejos, es posible encontrar también soluciones imaginarias. Así es posible encontrar una solución imaginaria empleando como valor inicial x1=1+2i:

```
>> a=[1,-6,12,-19,12]; n=length(a)-1; x1=1+2i;
>>  b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 16-2i
>> x2=x1-b(n+1)/c(n)
x2 = 0.75660528+1.6140913i

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 4.2797153-2.6892125i
>> x2=x1-b(n+1)/c(n)
x2 = 0.47183979+1.5828856i

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = -0.72988056-1.517537i
>> x2=x1-b(n+1)/c(n)
x2 = 0.50686324+1.6610642i

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 0.16039207+3.6536288E-2i
>> x2=x1-b(n+1)/c(n)
x2 = 0.50002118+1.6582594i

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = 2.9020127E-4-1.2363929E-3i
>> x2=x1-b(n+1)/c(n)
x2 = 0.5+1.6583124i

>> x1=x2; b=divsin1(a,x1); c=divsin1(b(1:n),x1); b(n+1), c(n);
ans = -1.6357438E-8+7.3944828E-8i
>> x2=x1-b(n+1)/c(n)
x2 = 0.5+1.6583124i
```

En esta última iteración los valores de "x" se repiten (y la función tiene 7 ceros) por lo que el proceso concluye siendo la solución x=0.5+1.6583124i.

10.1.2. Ejercicios

1. **Encuentre, con 6 dígitos de precisión, una de las soluciones de las siguientes ecuación, aplicando manualmente el método de Newton Raphson (valor inicial asumido: 1.1).**

$$P_4 = x^4 - 2x^3 + 3x^2 + 4x - 5 = 0$$

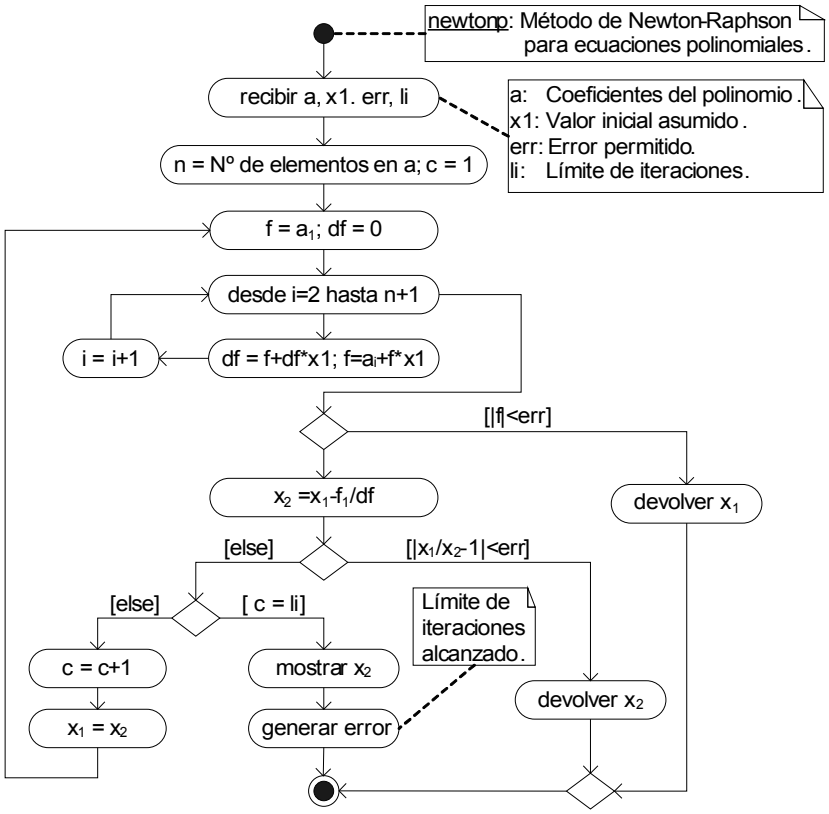
2. **Encuentre, con 8 dígitos de precisión, una de las soluciones de la ecuación del anterior ejercicio, aplicando manualmente el método de Newton Raphson (valor inicial asumido: 1+2i).**

10.1.3. Algoritmo y código

Como se ha visto en los anteriores ejemplos y ejercicios, en el método de Newton-Raphson sólo se emplean los residuos de la división sintética, por lo que no es necesario guardar los demás coeficientes.

El algoritmo que resuelve el problema, tomando en cuenta la anterior consideración, se presenta en la siguiente página y el código respectivo es:

```
function x2=newtonp(a,x1,err,li)
  c=1; n=length(a);
  while 1
    f=a(1); df=0;
```




```
    for i=2:n df=f+df*x1; f=a(i)+f*x1; end
    if abs(f)<err x2=x1; return end
    x2=x1-f/df;
    %printf("%f: %f\n",c,x2);
    if abs(x1/x2-1)<err return end
    if c++==li error("error-newtonp: %f\n",x2); end
    x1=x2;
  end
end
```

Se puede probar el programa encontrando las soluciones de la siguiente ecuación, estimando los valores iniciales con "qd":

$$P_3(x)=x^3+2\cdot x^2+3\cdot x+4=0$$

Los valores iniciales (con un error igual a 1e-3) son:

```
>> a=[1,2,3,4]; xi=qd(a,1e-3,100)
xi = [ -1.6422  -0.17888-1.5359i  -0.17888+1.5359i ]
```

Entonces se pueden calcular resultados más exactos (con 14 dígitos de exactitud) empleando cada uno de estos valores como valores iniciales del método de Newton:

```
>> format 10,14;
>> newtonp(a,xi(1),1e-15,30)
x2 = -1.6506291914394
>> newtonp(a,xi(2),1e-15,30)
x2 = -0.17468540428031-1.5468688872314i
>> newtonp(a,xi(3),1e-15,30)
x2 = -0.17468540428031+1.5468688872314i
```

Los tres resultados pueden ser calculados con una instrucción empleando la estructura "for" (guardando los resultados en la variable "r"):

```
>> for i=1:3 r(i)=newtonp(a,xi(i),1e-15,30); end r
   ans = [ -1.6506291914394  -0.17468540428031-1.54686888723l4i
           -0.17468540428031+1.54686888723l4i ]
```

En la práctica, puesto que las soluciones complejas siempre vienen en pares conjugados, sólo es necesario calcular una de ellas, la otra simplemente es el conjugado ("conj") de la primera. Por ejemplo, el par conjugado de la segunda solución:

```
>> conj(r(2))
ans = -0.17468540428031+1.54686888723l4i
```

Es en realidad la tercera solución encontrada.

10.1.4. Ejercicios

3. Encuentre las soluciones de las siguiente ecuación estimando los valores iniciales con "qd" (error 1e-3), calculando las soluciones más exactas con "newtonp" (error 1e-14) y comprando los resultados con "roots".

$$x^4-3x^2+x-5=0$$

4. Encuentre las soluciones de las siguiente ecuación estimando los valores iniciales con "qd" (error 1e-6), calculando las soluciones más exactas con "newtonp" (error 1e-15) y comparando los resultados con "roots".

$$x^8-7x^7+11x^6+41x^5-183x^4+231x^3-21x^2-265x-150=0$$

10.2. 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 (x^2-rx-s) y el método va cambiando los coeficientes del divisor ("r" y "s") hasta que el residuo se hace cero, es decir hasta que se cumple que:

$$\frac{P_n(x)}{x^2-r\cdot x-s}=Q_{n-2}(x)\cdot\left(x^2-r\cdot x-s\right)=0 \qquad (10.3)$$

Entonces la ecuación de segundo grado puede ser resuelta obteniéndose así dos de las raíces del polinomio.

Luego se puede repetir el proceso con el polinomio residual (Q_{n-2}) y continuar así hasta encontrar todas las soluciones, sin embargo y al igual que en el método de Newton Raphson, este procedimiento puede dar lugar a resultados erróneos (debido a la propagación del error), por lo que en general se prefiere encontrar las soluciones con el polinomio original, empleando valores iniciales adecuados para converger hacia cada para de soluciones.

La relación entre los valores iniciales asumidos (x_1 y x_2) y la ecuación cuadrática x^2-rx-s, se puede deducir fácilmente de la igualdad:

$$\left(x-x_1\right)\cdot\left(x-x_2\right)=x^2-r\cdot x-s$$
$$x^2-\left(x_1+x_2\right)\cdot x+x_1\cdot x_2=x^2-r\cdot x-s$$

En consecuencia se cumple que:

$$\begin{aligned} r&=x_1+x_2\\ s&=-x_1\cdot x_2 \end{aligned} \qquad (10.4)$$

Para llevar a cabo la división de la ecuación (10.3), el método más eficiente es, una vez más, el de la división sintética, por lo que primero se repasa dicho procedimiento.

10.2.1. División sintética de segundo grado

En la división sintética de segundo grado se divide el polinomio entre la ecuación cuadrática: $x^2-rx-s=0$.

Para recordar el procedimiento que se sigue se dividirá el siguiente polinomio entre x^2+x+1 (r=-1, s=-1):

$$P_4(x)=x^4-1.1\cdot x^3+2.3\cdot x^2+0.5\cdot x+3.3=0$$

$$\begin{array}{l|rrrrr} r=1 & 1 & -1.1 & 2.3 & 0.5 & 3.3 \\ \underline{s=1} & & -1.0 & 2.1 & -3.4 & 0.8 \\ & & & -1.0 & 2.1 & -3.4 \\ \hline & 1 & -2.1 & 3.4 & -0.8 & 0.7 \end{array}$$

Si se denomina "*b*" a los coeficientes resultantes de la primera división, entonces el residuo es: $b_n(x-r)+b_{n+1}$, donde "n" es el grado del polinomio. Con los resultados del ejemplo el residuo es: -0.8*(x-1)+0.7 = -0.8*x-0.1.

Una segunda división sintética permite calcular las derivadas parciales de la función con relación a los coeficientes "r" y "s" de la ecuación cuadrática (estos valores son de utilidad en el método de Bairstow):

$$\begin{array}{l|rrrr} r=1 & 1 & -2.1 & 3.4 & -0.8 \\ \underline{s=1} & & -1.0 & 3.1 & -5.5 \\ & & & -1.0 & 3.1 \\ \hline & 1 & -3.1 & 5.5 & -3.2 \end{array}$$

Si se denomina "c" a los coeficientes resultantes de esta segunda división se tiene: $\partial b_4/\partial r = c_3 = 5.5$; $\partial b_4/\partial s = c_2 = -3.1$. En general se cumple que:

$$\begin{aligned} \frac{\partial b_i}{\partial r} &= c_{i-1} \\ \frac{\partial b_i}{\partial s} &= c_{i-2} \end{aligned} \qquad (10.5)$$

Analizando las operaciones realizadas, se puede deducir fácilmente las ecuaciones generales para el cálculo de los coeficientes "b" resultantes de la división sintética:

$$\begin{aligned} b_1 &= a_1 \\ b_2 &= a_2+b_1\cdot r \\ b_i &= a_i+b_{i-1}\cdot r+b_{i-2}\cdot x \quad \{i=3\rightarrow n+1 \end{aligned} \qquad (10.6)$$

Donde "n" es el grado del polinomio, siendo el residuo de la división:

$$residuo=b_n\cdot(x-r)+b_{n+1}=b_n\cdot x+\left(b_{n+1}-b_n\cdot r\right) \qquad (10.7)$$

La función que automatiza este cálculo es:

```
function b=divsin2(a,r,s)
  b=0;
  b(1)=a(1);
  b(2)=a(2)+b(1)*r;
   for i=3:length(a) b(i)=a(i)+b(i-1)*r+b(i-2)*s; end
end
```

Llamando a esta función con los datos del ejemplo manual, se obtiene:

```
>> divsin2([1,-1.1,2.3,0.5,3.3],-1,-1)
b = [ 1  -2.1  3.4  -0.8  0.7 ]
```

Y para la segunda:

```
>> divsin2(ans(1:4),-1,-1)
b = [ 1  -3.1  5.5  -3.2 ]
```

Que concuerdan con los resultados calculados manualmente.

Habiendo repasado el procedimiento para realizar la división sintética de segundo grado, se prosigue con el desarrollo del método.

Para que la ecuación (10.3) se cumpla, el residuo de la división (ecuación 10.7)debe ser cero. Ello se puede lograr expandiendo los coeficientes "b_n" y "b_{n+1}" en series de Taylor, como funciones de los coeficientes "r" y "s":

$$b_n(r+\Delta r, s+\Delta S)=b_n(r,s)+\frac{\partial}{\partial r}\left(b_n(r,s)\right)\Delta r+\frac{\partial}{\partial s}\left(b_n(r,s)\right)\Delta s+\cdots+\infty=0$$

$$b_{n+1}(r+\Delta r, s+\Delta S)=b_{n+1}(r,s)+\frac{\partial}{\partial r}\left(b_{n+1}(r,s)\right)\Delta r+\frac{\partial}{\partial s}\left(b_{n+1}(r,s)\right)\Delta s+\cdots+\infty=0$$

Donde Δr y Δs son los valores que deben sumarse a "r" y "s" para que tanto "b_n" como "b_{n+1}" sean cero. Por supuesto, no es posible en la práctica resolver este sistema (que es más complejo aún que el problema original), por eso sólo se toman los términos de primer orden con lo que el sistema queda en la forma:

$$\begin{aligned} b_n+\frac{\partial b_n}{\partial r}\Delta r+\frac{\partial b_n}{\partial s}\Delta s=0 \\ b_{n+1}+\frac{\partial b_{n+1}}{\partial r}\Delta r+\frac{\partial b_{n+1}}{\partial s}\Delta s=0 \end{aligned} \qquad (10.8)$$

Las derivadas parciales de este sistema son los coeficientes resultantes de la segunda división sintética.

Reemplazando las igualdades de la ecuación (10.5), en la ecuación (10.8), se obtiene:

$$\begin{aligned} c_{n-1}\Delta r+c_{n-2}\Delta s=-b_n \\ c_n\Delta r+c_{n-1}\Delta s=-b_{n+1} \end{aligned} \qquad (10.9)$$

Que es un sistema lineal de dos ecuaciones con dos incógnitas (Δr y Δs), cuya solución es:

$$\begin{aligned} \Delta r=\frac{b_{n+1}\cdot c_{n-2}-b_n\cdot c_{n-1}}{c_{n-1}^2-c_n\cdot c_{n-2}} \\ \Delta s=\frac{-b_n-c_{n-1}\cdot\Delta r}{c_{n-2}} \end{aligned} \qquad (10.10)$$

Al sumar estos valores a "r" y "s" no se consigue que "b_n" y "b_{n+1}" sean cero, pues al haber truncado la serie de Taylor en los términos de primer orden, los valores de "Δr" y "Δs" son solo aproximados. Por eso el cálculo de los valores de "r" y "s" que hacen "b_n" y "b_{n+1}" cero, se convierte en un proceso iterativo, es decir se deben ir calculando nuevos valores de "r" y "s" (r=r+Δr y s=s+Δs) hasta que los valores de "b_n"y "b_{n+1}" sean casi cero o hasta que los valores de "r" y "s", de dos iteraciones consecutivas, sean casi iguales.

Para comprender mejor el procedimiento, se encontrarán dos de la soluciones de la siguiente ecuación con 7 dígitos de precisión. Los valores iniciales asumidos son: x1=0.1+0.1i, x2=0.1-0.1i.

$$P_4(x)=x^4-1.1\cdot x^3+2.3\cdot x^2+0.5\cdot x+3.3=0$$

Los valores conocidos y los valores de "r" y "s" (calculados con la ecuación 10.4) son:

```
>> format 10,7
>> a=[1,-1.1,2.3,0.5,3.3]; x1=0.1+0.1i; x2=conj(x1); n=length(a)-1;
   r=x1+x2, s=-x1*x2
r = 0.2
s = -2.0E-2
```

Se calculan los valores de "b" y "c" por división sintética:

```
>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -0.9  2.1  0.938  3.4456 ]
c = [ 1  -0.7  1.94  1.34 ]
```

Como era de esperar, al ser la primera iteración, ni b_n (b_4) ni b_{n+1} (b_5) son cercanos a cero. Por lo tanto se calculan Δr y Δs (ecuación 10.10):dr=(b

```
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)),
   ds=(-b(n)-c(n-1)*dr)/c(n-2)
dr = -0.9000425
ds = -1.154404
```

Por lo tanto, los nuevos valores de "r" y "s" son:

```
>> r+=dr, s+=ds
r = -0.7000425
s = -1.174404
```

Como estos valores no son iguales a los iniciales el proceso se repite:

```
>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -1.800043  2.385703  0.943883  -0.1625362 ]
c = [ 1  -2.500085  2.961465  1.80684 ]
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)); ds=(-b(n)-c(n-
1)*dr)/c(n-2); r+=dr, s+=ds
r = -0.8798293
s = -1.009829
```

Como no se cumple ninguna de las condiciones: b_n y b_{n+1} no son cercanos a cero y los nuevos valores de "r" y "s" no son iguales a los anteriores, el proceso se repite hasta que se cumpla alguna de las condiciones:

```
>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -1.979829  3.032083  -0.1684267  0.3863017 ]
c = [ 1  -2.859659  4.538266  -1.27356 ]
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)); ds=(-b(n)-c(n-
1)*dr)/c(n-2); r+=dr, s+=ds
r = -0.8999029
s = -1.100583

>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -1.999903  2.999135  2.128613E-3  -2.713253E-3 ]
c = [ 1  -2.899806  4.508096  -0.863243 ]
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)); ds=(-b(n)-c(n-
1)*dr)/c(n-2); r+=dr, s+=ds
r = -0.8999999
```

```
s = -1.1

>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -2  3  -1.488659E-7  8.343186E-7 ]
c = [ 1  -2.9  4.51  -0.8690002 ]
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)); ds=(-b(n)-c(n-
1)*dr)/c(n-2); r+=dr, s+=ds
r = -0.9
s = -1.1
>> b=divsin2(a,r,s), c=divsin2(b(1:n),r,s)
b = [ 1  -2  3  3.108624E-15  -4.440892E-14 ]
c = [ 1  -2.9  4.51  -0.869 ]
>> dr=(b(n+1)*c(n-2)-b(n)*c(n-1))/(c(n-1)^2-c(n)*c(n-2)); ds=(-b(n)-c(n-
1)*dr)/c(n-2); r+=dr, s+=ds
r = -0.9
s = -1.1
```

En esta última iteración se cumplen las dos condiciones: los valores de b_n y b_{n+1} son cercanos a cero (tienen 14 y 13 ceros) y los nuevos valores de "r" y "s" son iguales a los anteriores (en los 7 dígitos empleados en el cálculo), en consecuencia el proceso concluye y con los valores de "r" y "s" (y la función "cuad") se calculan dos de las cuatro soluciones:

```
>> cuad([1,-r,-s])
r = [ -0.45-0.9473648i  -0.45+0.9473648i ]
```

10.2.2. Ejercicios

5. Encuentre, con 8 dígitos de precisión, dos de las soluciones de las siguiente ecuación aplicando manualmente el método de Bairstow (valores asumidos -1, 5).

$$x^8-5x^7+x^4-8x^3+2x-3=0$$

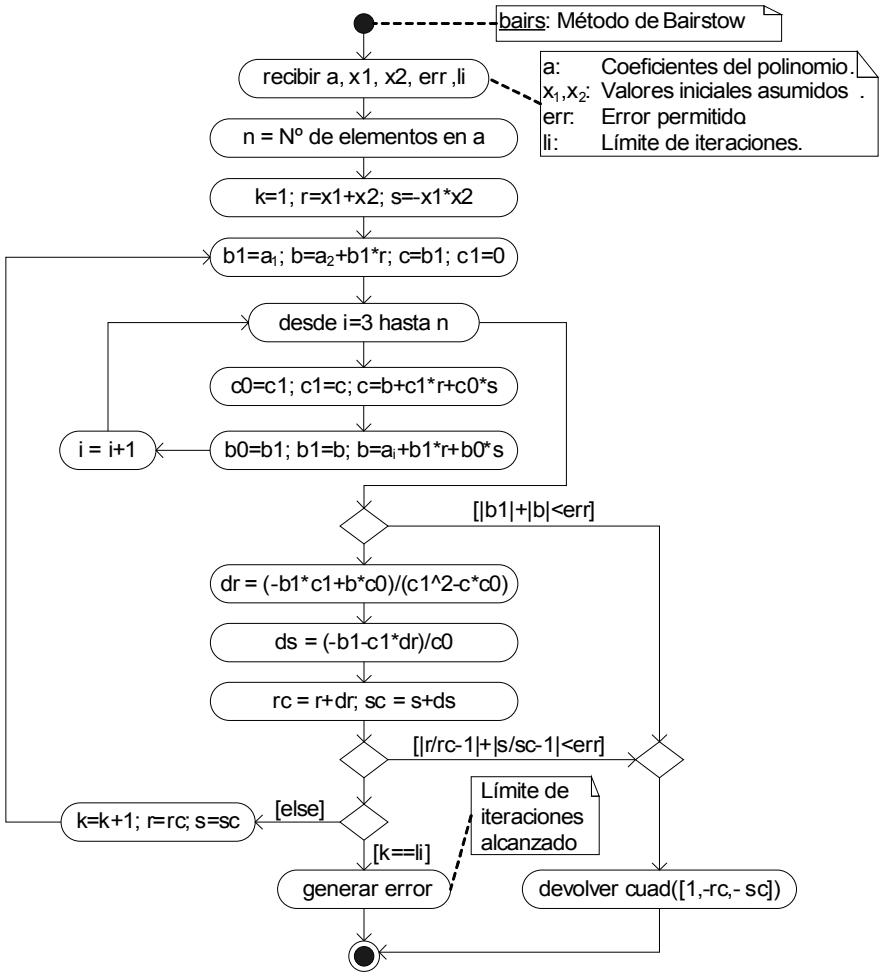
6. Encuentre, con 12 dígitos de precisión, dos de las soluciones de la ecuación del ejercicio anterior aplicando manualmente el método de Bairstow (valores asumidos -0.8-0.9i, -0.8+0.9i).

10.2.3. Algoritmo y código

Como se ha podido ver en los anteriores ejemplos y ejercicios, en el método de Bairstow sólo se emplean los dos últimos elementos de la primera división sintética (el residuo) y los tres últimos elementos de la segunda (las derivadas parciales), por lo tanto no es necesario guardar los otros elementos resultantes.

El algoritmo del método, que toma en cuenta la anterior consideración, se presenta en la siguiente página y el código respectivo en Jasymca es:

```
function x=bairs(a,x1,x2,err,li)
  n=length(a);
  k=1; r=x1+x2; s=-x1*x2;
  while 1
    b1=a(1); b=a(2)+b1*r; c=b1; c1=0;
    for i=3:n
      c0=c1; c1=c; c=b+c1*r; c+=c0*s;
      b0=b1; b1=b; b=a(i); b+=b1*r; b+=b0*s;
    end
    if abs(b1)+abs(b)<err break; end
    dr=(-b1*c1+b*c0); dr/=(c1*c1-c*c0);
```




```
    ds=(-b1-c1*dr)/c0;
    rc=r+dr;
    sc=s+ds;
    %printf("%f: r=%f; s=%f;\n",k,rc,sc);
    if abs(r/rc-1)+abs(s/sc-1)<err break; end
    if k++ == li error("¡Error!: %f\n",cuad([1,-rc,-sc])); end
    r=rc; s=sc;
  end
  x=cuad([1,-rc,-sc]);
end
```

Haciendo correr este programa con los datos del ejemplo manual se obtiene:

```
>> format 10,7
>> bairs([1,-1.1,2.3,0.5,3.3],0.1+0.1i,0.1-0.1i,1e-8,100)
x = [ -0.45-0.9473648i  -0.45+0.9473648i ]
```

Que son los mismos resultados obtenidos en el ejemplo.

10.2.4. Ejercicios

7. Encuentre las soluciones (error 1e-11) de la siguientes ecuación empleando el método de Bairstow. Obtenga los valores iniciales con el método QD (error 1e-6). Compare los resultados con los obtenidos con "roots".

$$2x^{20}+3x^{17}-4x^{15}+12x^{12}-9x^{8}+6x^{6}-2x^{4}-4x^{3}+3x^{2}-x+9=0$$

8. Encuentre las soluciones (error 1e-13) de la siguientes ecuación empleando el método de Bairstow para las soluciones complejas y el método de Newton para la solución real. Estime los valores iniciales con el método QD (error 1e-4). Compare los resultados con los obtenidos con "roots".

$$x^{5}-5x^{4}-6x^{3}+12x^{2}+34x-48=0$$

11. SISTEMAS DE ECUACIONES LINEALES 1

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 grupos: a) *No iterativos o directos*, con los cuales se pueden resolver sistemas con 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 ocurre con los métodos iterativos, la convergencia no está garantizada.

11.1. RESOLUCIÓN DE SISTEMAS DE ECUACIONES LINEALES EN JASYMCA

Cuando se trabaja con sistemas de ecuaciones lineales, la forma más eficiente de operar con dichos sistemas es colocarlos en forma matricial. Por ejemplo, si el sistema de ecuaciones a resolver es:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 14 \\ x_1 + 6x_2 - x_3 &= 15 \\ 2x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad (11.1)$$

En forma matricial se representa como:

$$\begin{bmatrix} 2 & 8 & 2 \\ 1 & 6 & -1 \\ 2 & -1 & 2 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 14 \\ 15 \\ 5 \end{bmatrix}$$

Donde la primera matriz es la matriz de los coeficientes, la segunda el vector de las variables y la tercera (a la derecha del signo =) el de las constantes.

Otra forma frecuente de representar sistemas de ecuaciones lineales es mediante la matriz aumentada, esto es la matriz de los coeficientes aumentada (a la derecha) con la columna de las constantes. Así, la matriz aumentada del sistema (11.1) es:

$$\begin{bmatrix} 2 & 8 & 2 & 14 \\ 1 & 6 & -1 & 15 \\ 2 & -1 & 2 & 5 \end{bmatrix}$$

Antes de pasar a estudiar los métodos de este tema se repasarán brevemente algunas de las operaciones básicas con matrices en Jasymca.

Las matrices se crean separando las filas con puntos y comas (";"), por ejemplo para crear la matriz de constantes del sistema (11.1), se escribe:

```
>> b=[2,8,2;1,6,-1;2,-1,2]
b =
  2   8   2
  1   6   -1
  2   -1  2
```

Para emplear o modificar los elementos de una matriz se escriben los índices entre paréntesis, así para recuperar el valor del elemento $b_{2,3}$, se escribe:

```
>> b(2,3)
ans = -1
```

Y para cambiar el valor del elemento $b_{3,1}$ a 10, se escribe:

```
>> b(3,1)=10
ans =
  2   8   2
  1   6   -1
  10  -1  2
```

Como ocurre con los vectores, para acceder a grupos de elementos se emplea el operador ":", por ejemplo para obtener todos los elementos de la segunda fila se escribe:

```
>> b(2,:)
ans = [ 1  6  -1 ]
```

Para obtener todos los elementos de la tercera columna:

```
>> b(:,3)
ans =
  2
  -1
  2
```

Para obtener los elementos de las filas 1 y 3:

```
>> b([1,3],:)
ans =
  2   8   2
  10  -1  2
```

Y para obtener todos los elementos de las columnas 1 y 3:

```
>> b(:,[1,3])
ans =
  2   2
  1   -1
  10  2
```

Como en vectores, se puede acceder también a dos o más elementos separados, en una misma fila o columna. Por ejemplo para obtener los elementos 1 y 3 de la fila 2 se escribe:

```
>> b(2,[1,3])
ans = [ 1  -1 ]
```

Y para obtener el primer y tercer elemento de la segunda columna:

```
>> b([1,3],2)
ans =
  8
  -1
```

Se debe tener presente que, al igual que en vectores, las matrices en Jasymca son punteros y que por lo tanto la operación de asignación no crea una copia de la matriz, sino simplemente le da un nombre adicional a la misma. Por ejemplo si se hace la asignación:

```
>> d=b;
```

Y luego se cambia el elemento 2,2 de la matriz "d" a 10:

```
>> d(2,2)=10;
```

No sólo cambia el elemento 2,2 de la matriz "d", sino también el de "b":

```
>> b
d =
  2   8   2
  1   10  -1
  10  -1  2
```

Porque, con la asignación, se ha hecho que "b" y "d" sean la misma matriz. Si lo que se quiere es crear una copia de la matriz, se deben recuperar todos los elementos de la misma con el operador ":", por ejemplo, la siguiente instrucción crea una copia "real" de la matriz b:

```
>> d=b(:,:);
```

Ahora si se cambia el elemento 1,3 de la matriz "d" a 10:

```
>> d(1,3)=10;
```

Se puede comprobar que sólo cambia la matriz "d":

```
> d,b
ans =
  2   8   10
  1   10  -1
  10  -1  2
d =
  2   8   2
  1   10  -1
  10  -1  2
```

Pues ahora son realmente dos matrices (aunque Jasymca reporta erróneamente la matriz "b" con el nombre de la matriz "d").

Se pueden añadir filas o columnas a una matriz simplemente colocando el índice (o índices) de las filas o columnas a añadir. Por ejemplo, para añadir el vector de coeficientes de la matriz (11.1), a la matriz "d", se escribe:

```
>> c=[14;15;5]; d(:,4)=c
ans =
  2   8   10  14
  1   10  -1  15
  10  -1  2   5
```

Observe que el vector "c" ha sido introducido como un vector columna (una matriz con una columna). Si se quiere transformar un vector fila en un vector columna, o en general, si se quiere transponer una matriz, se emplea el operador "'". Por ejemplo la transpuesta de la matriz "d" se obtiene con:

```
>> d'
ans =
  2   1   10
  8   10  -1
  10  -1  2
  14  15  5
```

Además, Jasymca, como se verá luego, puede sumar, multiplicar, dividir y restar matrices.

11.1.1. Ejercicio

1. Tomando en cuenta la matriz que se presenta al final de la pregunta haga las siguientes operaciones: Cree la matriz de coeficientes "b" y el vector columna de las constantes "c", con "b" y "c" cree la matriz aumentada "a", guarde en "d" la primera y tercera fila de la matriz "a", guarde

en "e" la segunda y cuarta columna de la matriz "a", añada como columnas de la matriz "d" la transpuesta de la matriz "e", asigne al elemento de la fila 2, columna 5 de la matriz "d" el valor 33, asigne al elemento de la fila 1, columna 6 de la matriz "d" el valor 44.

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 10 \end{aligned} \qquad (11.2)$$

Habiendo repasado las operaciones básicas con matrices se pasa a la solución de sistemas de ecuaciones lineales. Para resolver sistemas de ecuaciones lineales, Jasymca cuenta con la función "linsolve":

linsolve(matriz_coeficientes, matriz_constantes)

Por ejemplo, para resolver el sistema de ecuaciones lineales (11.1), se escribe:

```
>> b=[2,8,2;1,6,-1;2,-1,2]; c=[14;15;5]; linsolve(b,c)
ans =
  6
  1
  -3
```

Por lo tanto, las soluciones son x_1=6, x_2=1, x_3=-3.

Otra forma de resolver sistemas de ecuaciones lineales es calculando la inversa ("inv") de la matriz de los coeficientes y multiplicarla por la matriz de las constantes, es decir:

soluciones = inversa_matriz_coeficientes * matriz_constantes

Así para resolver el sistema (11.1) se escribe:

```
>> inv(b)*c
ans =
  6
  1
  -3
```

O alternativamente:

```
>> b\c
ans =
  6
  1
  -3
```

Donde el operador "\" es el operador de división matricial.

Cuando se tienen dos o más sistemas de ecuaciones lineales que sólo difieren en sus constantes, se pueden resolver todos los sistemas a la vez escribiendo los vectores de constantes en una matriz. Por ejemplo, para resolver los siguientes sistemas de ecuaciones:

$$\begin{aligned} x_1 + 2x_2 + 3x_3 &= 21 \\ 5x_1 - 9x_2 + 2x_3 &= 12 \\ 3x_1 + x_2 - 4x_3 &= 15 \end{aligned}$$

$$\begin{aligned} y_1 + 2y_2 + 3y_3 &= 11 \\ 5y_1 - 9y_2 + 2y_3 &= 2 \\ 3y_1 + y_2 - 4y_3 &= 4 \end{aligned} \qquad (11.3)$$

$$\begin{aligned} z_1 + 2z_2 + 3z_3 &= 1 \\ 5z_1 - 9z_2 + 2z_3 &= 2 \\ 3z_1 + z_2 - 4z_3 &= 3 \end{aligned}$$

Se escribe:

```
>> b=[1,2,3; 5,-9,2;3,1,-4]; c=[21,11,1;12,2,2;15,4,3]; linsolve(b,c)
ans =
  7.2033      2.8571      0.81868
  3.2143      1.7143      0.21429
  2.456       1.5714      -8.2418E-2
```

Donde la primera columna contiene las soluciones del primer sistema de ecuaciones, la segunda del segundo y la tercera del tercero. Estas soluciones pueden ser obtenidas también con la inversa:

```
>> inv(b)*c
ans =
  7.2033      2.8571      0.81868
  3.2143      1.7143      0.21429
  2.456       1.5714      -8.2418E-2
```

O:

```
>> b\c
ans =
  7.2033      2.8571      0.81868
  3.2143      1.7143      0.21429
  2.456       1.5714      -8.2418E-2
```

11.1.2. Ejercicio

2. Dados los siguientes sistemas de ecuaciones, guarde la matriz de coeficientes en las matriz "b", la de los coeficientes en la matriz "c", encuentre las soluciones del primer sistema con "linsolve", las del segundo con "inv", las del tercero con el operador "\" y todas las soluciones a la vez con "linsolve".

$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\ x_1 + 4x_2 + + 2x_4 &= 0 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

$$\begin{aligned} 3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\ y_1 + 4y_2 + + 2y_4 &= 2 \\ 2y_1 + y_2 + 2y_3 - y_4 &= 3 \\ y_1 + y_2 - y_3 + 3y_4 &= 4 \end{aligned} \qquad (11.4)$$

$$\begin{aligned} 3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\ z_1 + 4z_2 + + 2z_4 &= 2 \\ 2z_1 + z_2 + 2z_3 - z_4 &= 1 \\ z_1 + z_2 - z_3 + 3z_4 &= 0 \end{aligned}$$

11.2. MÉTODO DE ELIMINACIÓN DE GAUSS

El método de eliminación de Gauss, reduce el sistema de ecuaciones lineales en un sistema triangular superior. Así al aplicar el método al siguiente sistema de 4 ecuaciones lineales:

$$\begin{array}{rcrcrcrcl} a_{11}x_1 & + & a_{12}x_2 & + & a_{13}x_3 & + & a_{14}x_4 & = & c_1 \\ a_{21}x_1 & + & a_{22}x_2 & + & a_{23}x_3 & + & a_{24}x_4 & = & c_2 \\ a_{31}x_1 & + & a_{32}x_2 & + & a_{33}x_3 & + & a_{34}x_4 & = & c_3 \\ a_{41}x_1 & + & a_{42}x_2 & + & a_{43}x_3 & + & a_{44}x_4 & = & c_4 \end{array} \qquad (11.5)$$

Se reduce a:

$$\begin{array}{rcrcrcrcl} x_1 & + & a'_{12}x_2 & + & a'_{13}x_3 & + & a'_{14}x_4 & = & c'_1 \\ 0 & + & x_2 & + & a'_{23}x_3 & + & a'_{24}x_4 & = & c'_2 \\ 0 & + & 0 & + & x_3 & + & a'_{34}x_4 & = & c'_3 \\ 0 & + & 0 & + & 0 & + & x_4 & = & c'_4 \end{array} \qquad (11.6)$$

Donde los apostrofes se emplean solo para denotar que son valore diferentes a los originales. Como se puede observar en la última ecuación, la cuarta solución "x_4" es igual a "c'_4". Entonces, el valor de la tercera solución "x_3" puede ser calculado con la penúltima ecuación:

$$x_3 = c'_3 - a'_{34}x_4$$

Ahora, con "x_3" y "x_4", se puede calcular "x_2" con la segunda ecuación:

$$x_2 = c'_2 - a'_{23}x_3 - a'_{24}x_4$$

Finalmente con "x_2", "x_3" y "x_4" se calcula "x_1" con la primera ecuación:

$$x_1 = c'_1 - a'_{12}x_2 - a'_{13}x_3 - a'_{14}x_4$$

En forma matricial (trabajando con la matriz aumentada), la aplicación del método de Gauss da lugar al siguiente cambio:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix} \xrightarrow{gauss} \begin{bmatrix} 1 & a'_{12} & a'_{13} & a'_{14} & a'_{15} \\ 0 & 1 & a'_{23} & a'_{24} & a'_{25} \\ 0 & 0 & 1 & a'_{34} & a'_{35} \\ 0 & 0 & 0 & 1 & a'_{45} \end{bmatrix} \qquad (11.7)$$

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 se denominará "*p*" *(pivote)* al contador que determina la fila y columna que se está reduciendo, "*n*" al número de ecuaciones del sistema y "m" al número de columnas de la matriz aumentada.

Para reducir los elementos de la diagonal principal a 1, simplemente se

divide la fila pivote (a_p) entre el elemento que se encuentra en la diagonal principal (a_{pp}):

$$a_p = \frac{a_p}{a_{pp}} \tag{11.8}$$

Luego, para reducir a cero los elementos de la columna "p", debajo del elemento a_{pp}, se resta a cada fila (a_i) el resultado de multiplicar el elemento que se encuentra en la columna pivote (a_{ip}) por la fila pivote (a_p):

$$a_i = a_i - a_p \cdot a_{ip} \quad \{i = p+1 \rightarrow n \tag{11.9}$$

Las ecuaciones (11.8) y (11.9) se aplican una tras otra de forma intercalada para valores de "p" que van desde 1 hasta "m", sin embargo, la ecuación (11.9), sólo se aplica hasta "m-1", porque cuando "p" es igual a "m" el límite inferior "p+1" es de hecho mayor a "m" y esto ocurre porque en ese momento ya no existen más filas por debajo de la columna pivote.

Una vez reducida la matriz a la forma triangular superior, las soluciones del sistema se calculan por sustitución inversa y se almacenan en la columna de las constantes:

$$a_{ij} = a_{ij} - \sum_{k=i+1}^{n} a_{ik} \cdot a_{kj} \quad \begin{cases} i = n-1 \rightarrow 1 \\ j = n+1 \rightarrow m \end{cases} \tag{11.10}$$

Para comprender mejor el método de Gauss, se resolverá manualmente el sistema de ecuaciones (11.1).

Los datos son:

```
>> a=[2,8,2,14;1,6,-1,15;2,-1,2,5]; n=3; m=4; p=1;
```

Entonces se reduce la fila "p" (1), aplicando la ecuación (11.8):

```
>> a(p,:)=a(p,:)/a(p,p)
ans =
  1   4   1   7
  1   6   -1  15
  2   -1  2   5
```

Y se reduce la columna "p" (1), aplicando la ecuación (11.9):

```
>> for i=p+1:n a(i,:)=a(i,:)-a(p,:).*a(i,p); end; a
ans =
  1   4   1   7
  0   2   -2  8
  0   -9  0   -9
```

Con ello se ha reducido la primera fila y la primera columna, ahora se incrementa el valor del pivote "p" y se repite el proceso:

```
>> p++; a(p,:)=a(p,:)/a(p,p); for i=p+1:n a(i,:)=a(i,:)-a(p,:).*a(i,p);
end; a
ans =
  1   4   1   7
  0   1   -1  4
  0   0   -9  27
```

Finalmente se incrementa el valor de "p" una vez más y se reduce la última fila ecuación (11.8):

```
>> p++; a(p,:)=a(p,:)/a(p,p)
ans =
  1   4   1   7
```

```
   0   1   -1   4
   0   0   1   -3
```

Así la matriz queda reducida a la forma triangular superior. Ahora, para calcular las soluciones se aplica la ecuación (11.10):

```
>> for i=n-1:-1:1 for j=n+1:m s=0; for k=i+1:n s+=a(i,k)*a(k,j); end;
   a(i,j)=a(i,j)-s; end end a
ans =
   1   4   1   6
   0   1   -1  1
   0   0   1   -3
```

Por lo tanto las soluciones (que se encuentran en la columna 4) son:

```
>> a(:,m)
ans =
   6
   1
   -3
```

Que son los resultados calculados perviamente con "linsolve".

11.2.1. Ejercicios

3. Encuentre las soluciones del siguiente sistema aplicando manualmente el método de eliminación de Gauss. Verifique los resultados con "\".

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 2 \end{aligned} \qquad (11.11)$$

4. Encuentre las soluciones del siguiente sistema aplicando manualmente el método de eliminación de Gauss. Verifique los resultados con "linsolve".

$$\begin{aligned} 2x_1 - 2x_2 + 5x_3 &= 13 \\ 2x_1 + 3x_2 + 3x_3 &= 20 \\ 3x_1 - x_2 + 3x_3 &= 10 \end{aligned} \qquad (11.12)$$

11.2.2. 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, lo que da lugar a un error por división entre cero. Para evitar este tipo de errores, se debe realizar un intercambio de filas y/o columnas de manera que el elemento pivote sea siempre diferente de cero. A este proceso se conoce con el nombre de "pivotaje".

El pivotaje no sólo es útil para evitar una división entre cero, sino también para reducir los errores de redondeo: cuanto mayor sea el valor del elemento pivote (en valor absoluto) menores serán los errores de redondeo.

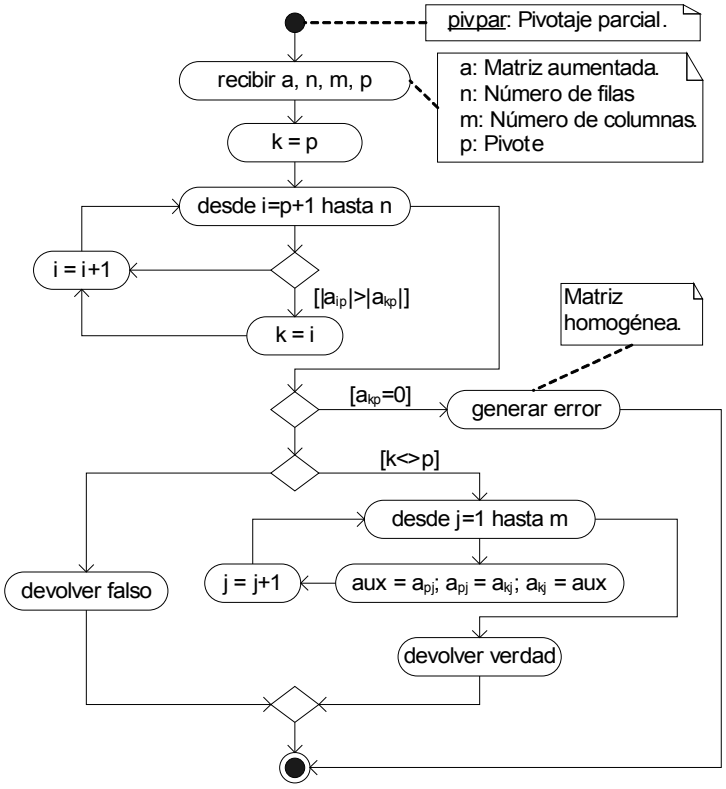
En general es posible aplicar dos tipos de pivotaje: a) "Parcial", cuando el mayor valor absoluto se busca sólo en la columna pivote y b) "Total" cuando el mayor valor absoluto se busca en todas las filas y columnas no reducidas.

11.2.3. Pivotaje Parcial

El proceso para llevar a cabo 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 el sistema homogéneo y por lo tanto no tiene soluciones únicas, por lo que se debe genera un error.

El algoritmo es el siguiente:



El código respectivo es el siguiente:

```
function r=pivpar(a,n,m,p)
  k=p;
  for i=p+1:n if abs(a(i,p))>abs(a(k,p)) k=i; end end
  if a(k,p)==0 error("¡Error! Matriz Homogénea\n"); end
  if k ~= p
    for j=1:m aux=a(p,j); a(p,j)=a(k,j); a(k,j)=aux; end
    r=1;
  else
    r=0;
  end
end
```

Se puede probar esta 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 =
  3    2    4    6    3
  2    2    6    4    1
  8    10   0    2    8
  1    -13  0    4    3
```

Si el pivote es 2 (p=2), deberían intercambiarse las filas 2 y 4:

```
>> a=[3,2,4,6,3;2,2,6,4,1;8,10,0,2,8;1,-13,0,4,3]; pivpar(a,4,5,2), a
```

```
r = 1
a =
  3    2    4    6    3
  1    -13  0    4    3
  8    10   0    2    8
  2    2    6    4    1
```

Como efectivamente sucede.

Para p=3, el programa debería generar un error, pues los elementos restantes en esa columna son cero:

```
>> a=[3,2,4,6,3;2,2,6,4,1;8,10,0,2,8;1,-13,0,4,3]; pivpar(a,4,5,3),
¡Error! Matriz Homogénea
```

11.2.4. 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 realizan intercambios de filas y/o 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 columnas implica también un intercambio en la posición de los resultados, por ejemplo, si se intercambian las columnas 2 y 4, el segundo resultado x_2 se encontrará en la fila 4 (y no en la fila 2), mientras que el cuarto resultado x_4 se encontrará en la fila 2 (y no en la fila 4).

El algoritmo se presenta en la siguiente página. En el mismo "v" es el vector de posiciones, esto es un vector cuyos elementos iniciales son números secuenciales de 1 al número de filas del sistema y que al final contiene el orden en que se encuentran los resultados. El código respectivo es:

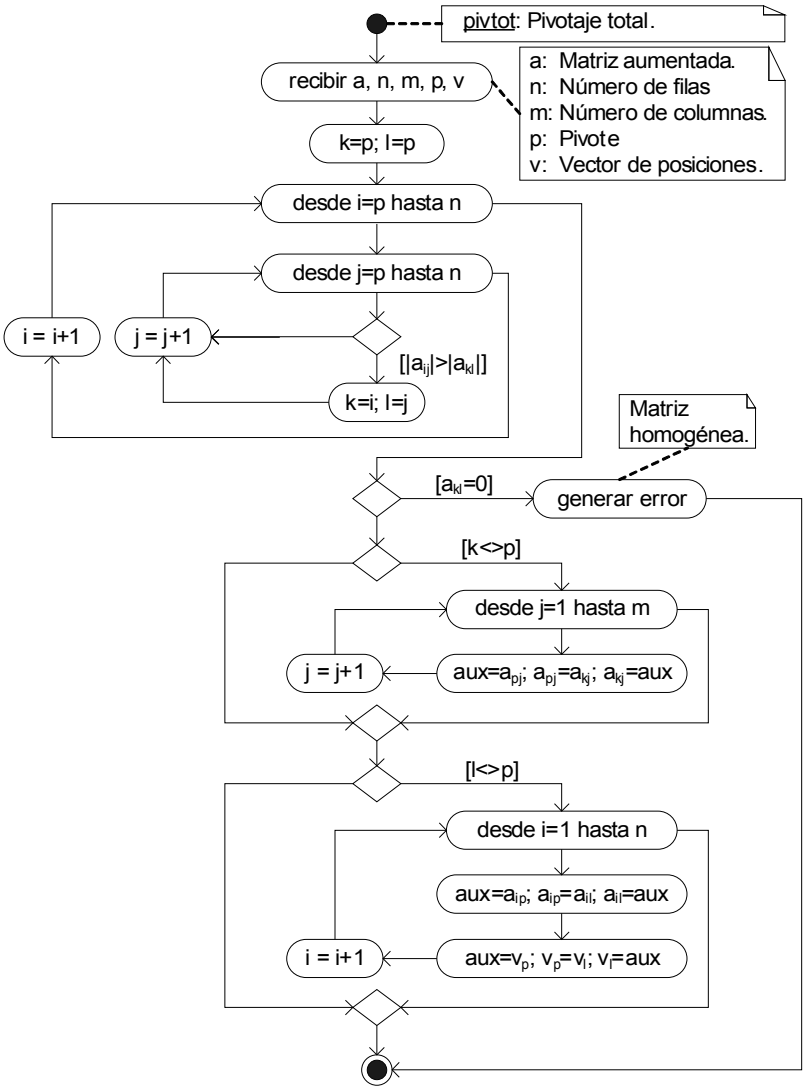
```
function r=pivtot(a,n,m,p,v)
  k=p; l=p;
  for i=p:n
    for j=p:n
      if abs(a(i,j))>abs(a(k,l)) k=i; l=j; end
    end
  end
  if a(k,l)==0 error("¡Error! Matriz Homogénea\n"); end
  if k ~= p
    for j=1:m aux=a(p,j); a(p,j)=a(k,j); a(k,j)=aux; end
  end
  if l ~= p
    for i=1:n aux=a(i,p); a(i,p)=a(i,l); a(i,l)=aux; end
    aux=v(p); v(p)=v(l); v(l)=aux;
  end
end
```

Se puede probar el programa con la misma matriz empleada en el pivotaje parcial. Como el sistema tiene 4 ecuaciones el vector de posiciones tendrá cuatro elementos (del 1 al 4):

```
a=[3,2,4,6,3;2,2,6,4,1;8,10,0,2,8;1,-13,0,4,3]; v=1:4;
```

Con "p=1", el programa debería ubicar el mayor elemento (-13) e intercambiar las filas 1 con 4 y las columnas 1 con 2, intercambiando también los elementos 1 y 2 (columnas 1 y 2) del vector de posiciones:

```
>> pivtot(a,4,5,1,v); a,v
```




```
a =
  -13  1    0    4    3
  2    2    6    4    1
  10   8    0    2    8
  2    3    4    6    3
v = [ 2  1  3  4 ]
```

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 al no existir intercambio de columnas, no se intercambian los elementos del vector de posiciones:

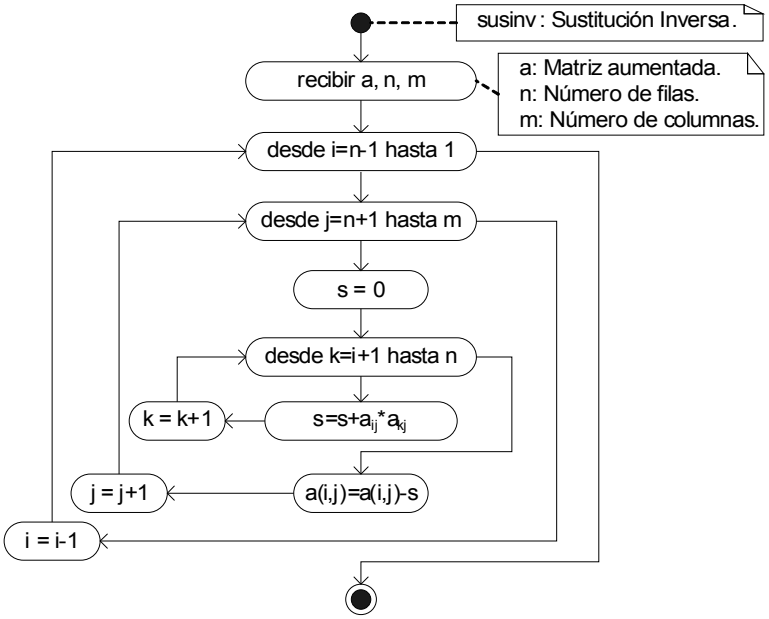
```
>> a=[3,2,4,6,3;2,2,6,4,1;8,10,0,2,8;1,-13,0,4,3]; v=1:4;
>> pivtot(a,4,5,2,v),a,v
a =
  3    2    4    6    3
  1    -13  0    4    3
  8    10   0    2    8
  2    2    6    4    1
v = [ 1  2  3  4 ]
```

Finalmente para "p=3" el mayor valor es 4, por lo que deberían un intercambiarse las filas 3 y 4 y de las columnas 3 y 4:

```
>> a=[3,2,4,6,3;2,2,6,4,1;8,10,0,2,8;1,-13,0,4,3]; v=1:4;
>> pivtot(a,4,5,3,v),a,v
a =
   3    2    6    4    3
   2    2    4    6    1
   1   -13   4    0    3
   8   10    2    0    8
v = [ 1  2  4  3 ]
```

11.2.5. Sustitución Inversa

Una vez reducido el sistema de ecuaciones las soluciones deben ser encontradas por sustitución hacia atrás (ecuación 11.10). El algoritmo es el siguiente:



Siendo el código respectivo:

```
function r=susinv(a,n,m)
  for i=n-1:-1:1
    for j=n+1:m
      s=0;
      for k=i+1:n s+=a(i,k)*a(k,j); end
      a(i,j)=a(i,j)-s;
    end
  end
end
```

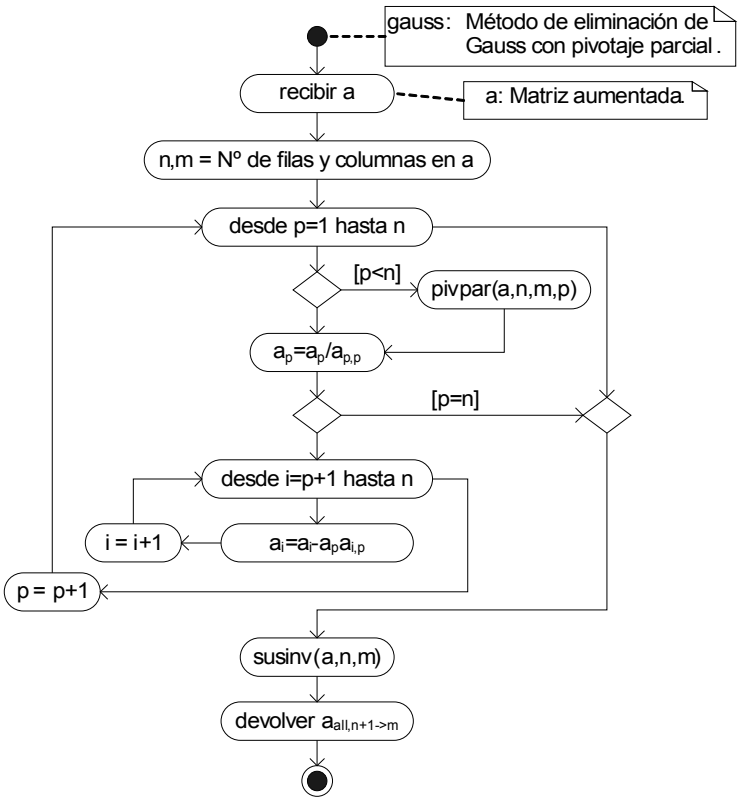
Probando el programa con la matriz reducida del ejemplo manual se obtiene:

```
>> a=[1,4,1,7;0,1,-1,4;0,0,1,-3]; susinv(a,3,4);a
a =
  1   4   1   6
  0   1   -1  1
  0   0   1   -3
```

Que son los mismos resultados obtenidos en el ejemplo.

11.2.6. Método de Eliminación de Gauss con pivotaje parcial

Con las funciones elaboradas, se puede programar el método de Gauss con pivotaje parcial. El algoritmo del mismo es:



Siendo el código respectivo:

```
function r=gauss(b)
  a=0; a=b(:,:); [n,m]=size(a);
  for p=1:n
    if p<n pivpar(a,n,m,p); end
    a(p,:)=a(p,:)/a(p,p);
    if p==n break; end
    for i=p+1:n a(i,:)=a(i,:)-a(p,:).*a(i,p); end
  end
  susinv(a,n,m);
  r=a(:,n+1:m);
end
```

Probando el programa con la matriz del ejemplo manual se obtiene:

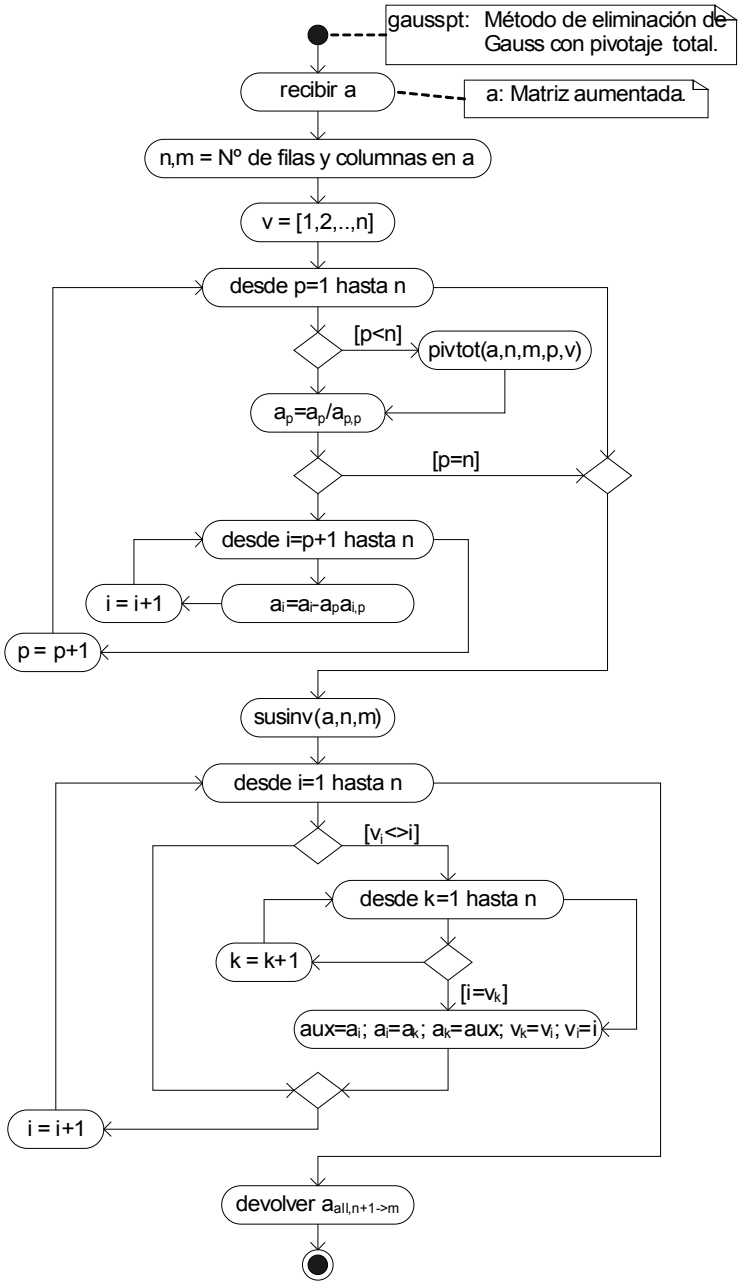
```
>> a=[2,8,2,14;1,6,-1,15;2,-1,2,5]; gauss(a)
r =
  6
  1
  -3
```

Que concuerda con los resultados del ejemplo.

11.2.7. Método de eliminación de Gauss con pivotaje total

La diferencia principal con relación al pivotaje total es la necesidad de ordenar los resultados en caso de existir intercambios de columnas.

El algoritmo es:



Siendo el código respectivo:

```
function r=gausspt(b)
  a=0; a=b(:,:); [n,m]=size(a); v=1:n;
  for p=1:n
    if p<n pivtot(a,n,m,p,v); end
    a(p,:)=a(p,:)/a(p,p);
    if p==n break; end
    for i=p+1:n a(i,:)=a(i,:)-a(p,:).*a(i,p); end
  end
  susinv(a,n,m);
  for i=1:n
    if v(i) ~= i
      for k=i:n if i==v(k) break; end end
```

```
        aux=a(i,:); a(i,:)=a(k,:); a(k,:)=aux; v(k)=v(i); v(i)=i;
      end
    end
    r=a(:,n+1:m);
  end
```

Probando el programa con la matriz del ejemplo manual se obtiene:

```
>> a=[2,8,2,14;1,6,-1,15;2,-1,2,5]; gausspt(a)
r =
  6
  1
  -3
```

Que son los resultados obtenidos en el ejemplo manual.

Gauss y gausspt, permiten calcular también la matriz inversa, para ello simplemente se añade a la matriz de coeficientes (o cualquier matriz cuadrada) la matriz unidad. Por ejemplo, para calcular la inversa de la matriz de coeficientes del ejemplo manual, se escribe:

```
>> a=[2,8,2;1,6,-1;2,-1,2];
>> a(:,4:6)=eye(3)
ans =
  2   8   2   1   0   0
  1   6   -1  0   1   0
  2   -1  2   0   0   1
>> gauss(a)
r =
  -0.30556  0.5       0.55556
  0.11111   0         -0.11111
  0.36111   -0.5      -0.11111
```

Donde "eye" genera una matriz unidad con "n" filas y columnas. El mismo resultado se obtiene con "gausspt". Estos resultados pueden ser corroborados con "inv":

```
>> a=[2,8,2;1,6,-1;2,-1,2];
>> inv(a)
ans =
  -0.30556  0.5       0.55556
  0.11111   0         -0.11111
  0.36111   -0.5      -0.11111
```

11.2.8. Ejercicios

5. Cree la matriz aumentada "a" del siguiente sistema de ecuaciones y encuentre las soluciones con "gauss", "gausspt", corroborando los resultados con "\".

$$\begin{aligned}
30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12\\
2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24\\
x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17\\
3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37\\
4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23\\
5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24
\end{aligned} \qquad (11.13)$$

6. Encuentre la inversa de los coeficientes del sistema de ecuaciones (11.13) con "gauss" y "gausspt", corrobore los resultados con "inv".

7. Cree la matriz de coeficientes "b", la matriz de constantes "c" de los siguientes sistemas de ecuaciones y encuentre, simultáneamente, las soluciones con "gauss", "gausspt", corroborando los resultados con "linsolve".

$$\begin{aligned}
3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\
x_1 + 4x_2 + \quad + 2x_4 &= 0 \\
2x_1 + x_2 + 2x_3 - x_4 &= 1 \\
x_1 + x_2 - x_3 + 3x_4 &= 0
\end{aligned}$$

$$\begin{aligned}
3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\
y_1 + 4y_2 + \quad + 2y_4 &= 2 \\
2y_1 + y_2 + 2y_3 - y_4 &= 3 \\
y_1 + y_2 - y_3 + 3y_4 &= 4
\end{aligned} \qquad (11.14)$$

$$\begin{aligned}
3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\
z_1 + 4z_2 + \quad + 2z_4 &= 2 \\
2z_1 + z_2 + 2z_3 - z_4 &= 1 \\
z_1 + z_2 - z_3 + 3z_4 &= 0
\end{aligned}$$

12. SISTEMAS DE ECUACIONES LINEALES 2

En este tema continúa el estudio de algunos de los métodos para resolver sistemas de ecuaciones lineales.

12.1. MÉTODO DE ELIMINACIÓN DE GAUSS JORDÁN

El método de eliminación de Gauss - Jordán, es muy similar 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 se aplica el método de eliminación de Gauss-Jordán al siguiente sistema:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= c_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= c_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= c_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= c_4 \end{aligned} \qquad (12.1)$$

Se transforma en:

$$\begin{aligned} x_1 + 0 + 0 + 0 &= c'_1 \\ 0 + x_2 + 0 + 0 &= c'_2 \\ 0 + 0 + x_3 + 0 &= c'_3 \\ 0 + 0 + 0 + x_4 &= c'_4 \end{aligned} \qquad (12.2)$$

Por lo tanto las soluciones se encuentran directamente en la columna de las constantes. En forma matricial, la aplicación del método de Gauss-Jordán da lugar a la siguiente transformación:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix} \xrightarrow{gauss-jordán} \begin{bmatrix} 1 & 0 & 0 & 0 & a'_{15} \\ 0 & 1 & 0 & 0 & a'_{25} \\ 0 & 0 & 1 & 0 & a'_{35} \\ 0 & 0 & 0 & 1 & a'_{45} \end{bmatrix} \qquad (12.3)$$

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. Por lo tanto, para la reducción de las filas se emplea la misma ecuación que en el método de Gauss:

$$a_p = \frac{a_p}{a_{pp}} \qquad (12.4)$$

Y para la reducción de columnas se cambia el límite inferior a 1:

$$a_i = a_i - a_p \cdot a_{ip} \quad \begin{cases} i = 1 \rightarrow n \\ i \neq p \end{cases} \qquad (12.5)$$

Para comprender mejor el método, se resolverá manualmente el siguiente sistema:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 14 \\ x_1 + 6x_2 - x_3 &= 13 \\ 2x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad (12.6)$$

Los datos son:

```
>> a=[2,8,2,14;1,6,-1,13;2,-1,2,5]; n=3; m=4; p=0;
```

Como el pivote se ha inicializado en 0, se incrementa su valor y se reduce la fila y la columna 1:

```
>> p++;a(p,:)=a(p,:)/a(p,p); for i=1:n if(i ~= p) a(i,:)=a(i,:)-
a(p,:).*a(i,p); end; end; a
ans =
  1   4   1   7
  0   2   -2  6
  0   -9  0   -9
```

Ahora se incrementa el valor del pivote (a 2) y se reduce la segunda fila y la segunda columna:

```
>> p++;a(p,:)=a(p,:)/a(p,p); for i=1:n if(i ~= p) a(i,:)=a(i,:)-
a(p,:).*a(i,p); end; end; a
ans =
  1   0   5   -5
  0   1   -1  3
  0   0   -9  18
```

Finalmente se incrementa una vez más el valor del pivote (a 3) y se reduce la tercera fila y tercera columna:

```
>> p++;a(p,:)=a(p,:)/a(p,p); for i=1:n if(i ~= p) a(i,:)=a(i,:)-
a(p,:).*a(i,p); end; end; a
ans =
  1   0   0   5
  0   1   0   1
  0   0   1   -2
```

Siendo las soluciones los valores de la última columna:

```
>> a(:,4)
ans =
  5
  1
  -2
```

12.1.1. Ejercicio

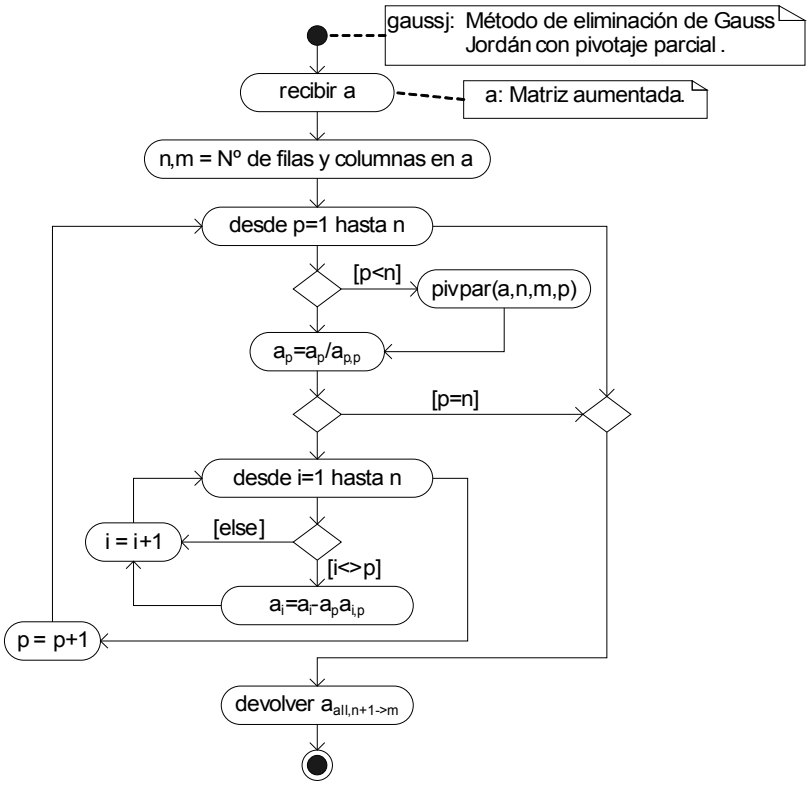
1. Encuentre las soluciones del siguientes sistema aplicando manualmente el método de Gauss – Jordán.

$$\begin{aligned} 3x_1 - x_2 + 2x_3 &= 12 \\ x_1 + 2x_2 + 3x_3 &= 11 \\ 2x_1 - 2x_2 - x_3 &= 10 \end{aligned} \qquad (12.7)$$

12.1.2. Algoritmo y código, Gauss-Jordán con pivotaje parcial

El algoritmo para el método de Gauss-Jordán con pivotaje parcial se presenta en la siguiente página y el código respectivo es:

```
function r=gaussj(b)
  a=0; a=b(:,:); [n,m]=size(a);
  for p=1:n
    if p<n pivpar(a,n,m,p); end
    a(p,:)=a(p,:)/a(p,p);
    for i=1:n
```




```
      if i ~= p a(i,:)=a(i,:)-a(p,:).*a(i,p); end
    end
  end
  r=a(:,n+1:m);
end
```

Haciendo correr el programa con la matriz del ejemplo manual se obtiene:

```
>> a=[2,8,2,14;1,6,-1,13;2,-1,2,5]; gaussj(a)
r =
  5
  1
  -2
```

Que son los resultados obtenidos con el programa manual.

12.1.3. Ejercicio

2. Dados los siguientes sistemas de ecuaciones, cree la matriz aumentada "a" correspondiente a los tres sistemas. Empleando "gaussj" obtenga las soluciones del segundo sistema y luego todas las soluciones a la vez.

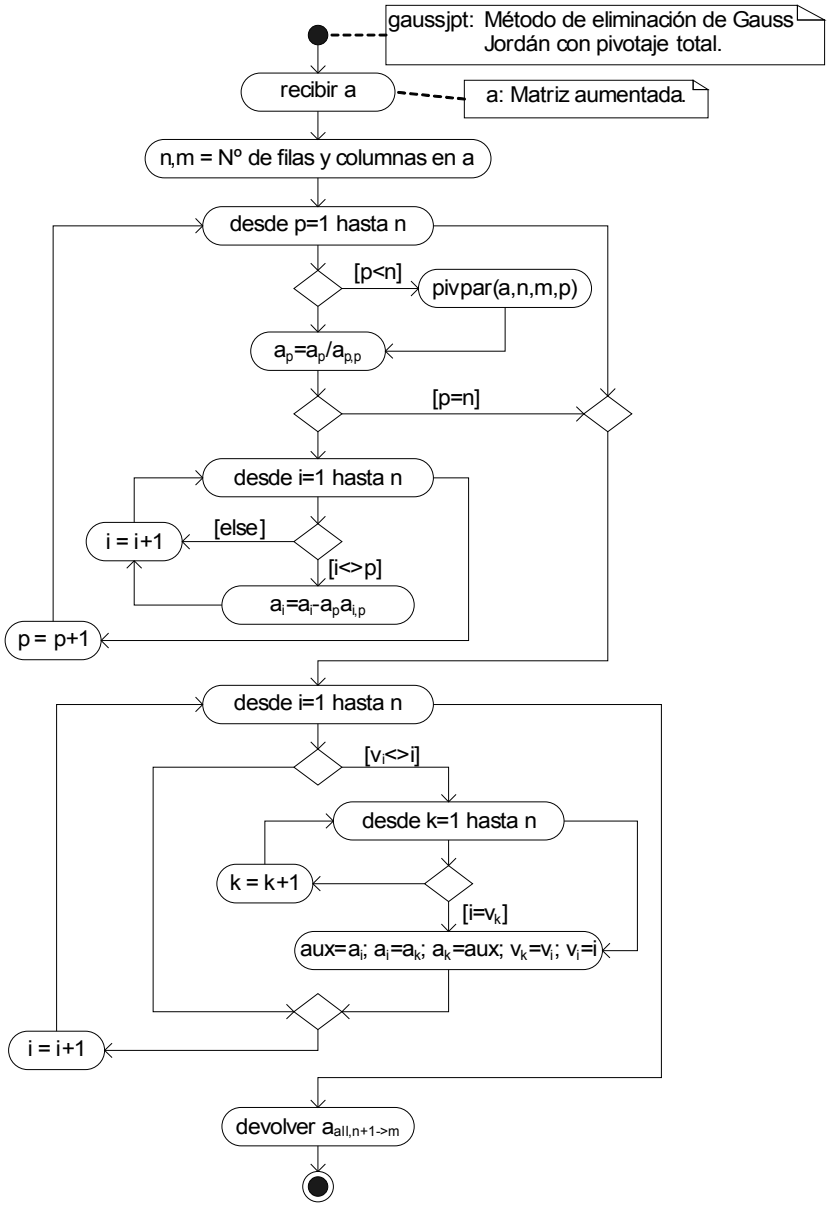
$$\begin{aligned} 3x_1 + 2x_2 - x_3 + 2x_4 &= 0 \\ x_1 + 4x_2 + \quad + 2x_4 &= 0 \\ 2x_1 + x_2 + 2x_3 - x_4 &= 1 \\ x_1 + x_2 - x_3 + 3x_4 &= 0 \end{aligned}$$

$$\begin{aligned} 3y_1 + 2y_2 - y_3 + 2y_4 &= -2 \\ y_1 + 4y_2 + \quad + 2y_4 &= 2 \\ 2y_1 + y_2 + 2y_3 - y_4 &= 3 \\ y_1 + y_2 - y_3 + 3y_4 &= 4 \end{aligned} \qquad (12.8)$$

$$\begin{aligned} 3z_1 + 2z_2 - z_3 + 2z_4 &= 2 \\ z_1 + 4z_2 + + 2z_4 &= 2 \\ 2z_1 + z_2 + 2z_3 - z_4 &= 1 \\ z_1 + z_2 - z_3 + 3z_4 &= 0 \end{aligned}$$

12.1.4. Algoritmo y código, Gauss-Jordán con pivotaje total

El algoritmo del método de Gauss-Jordan con pivotaje total es:



Siendo el código respectivo:

```
function r=gaussjpt(b)
  a=0; a=b(:,:); [n,m]=size(a); v=1:n;
  for p=1:n
    if p<n pivtot(a,n,m,p,v); end
    a(p,:)=a(p,:)/a(p,p);
    for i=1:n
```

```
      if i ~= p a(i,:)=a(i,:)-a(p,:).*a(i,p); end
    end
  end
  for i=1:n
    if v(i) ~= i
      for k=i:n if i==v(k) break; end end
      aux=a(i,:); a(i,:)=a(k,:); a(k,:)=aux; v(k)=v(i); v(i)=i;
    end
  end
  r=a(:,n+1:m);
end
```

Haciendo correr el programa con el sistema de ejemplo manual se obtiene:

```
>> a=[2,8,2,14;1,6,-1,13;2,-1,2,5]; gaussjpt(a)
r =
  5
  1
  -2
```

Que son los resultados correctos.

12.1.5. Ejercicios

3. Cree la matriz de coeficientes "b", la matriz de constantes "c" y calcule las soluciones del siguiente sistema con "gaussjpt".

$$\begin{aligned} 30x_1 + x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12 \\ 2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24 \\ x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17 \\ 3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37 \\ 4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23 \\ 5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24 \end{aligned} \qquad (12.9)$$

12.2. MÉTODO DE CROUT

El método de Crout, al igual que los métodos de Doolitle y Cholesky forman parte de los denominados métodos de factorización o descomposición. El fundamento de estos métodos es el siguiente:

Dada la matriz A:

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix} \qquad (12.10)$$

Se puede demostrar que la misma puede ser expresada como el producto de dos matrices triangulares, una inferior "L" y otra superior "U":

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}=\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}\times\begin{bmatrix} 1 & u_{12} & u_{13} & u_{14} \\ 0 & 1 & u_{23} & u_{24} \\ 0 & 0 & 1 & u_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}=L\times U \qquad (12.11)$$

A este proceso, el de calcular los elementos de las matrices triangulares "L" y "U" para una matriz cuadrada "A", se conoce como *descomposición LU* o *factorización LU* y es el proceso en el que se basan los métodos de Crout, Doolitle y Cholesky.

El proceso de descomposición no es único, las posibles combinaciones de "L" y "U" son infinitas. En general, sin embargo, son tres las formas de descomposición más empleadas en la práctica: La descomposición de **Crout**, que corresponde a la descomposición mostrada en el ejemplo, es decir aquella en la cual los elementos de la diagonal principal en "U" son unos. La descomposición de **Doolittle**, en la cual los elementos de la diagonal principal en "L" son unos, es decir:

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}=\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{bmatrix}\times\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{bmatrix}=L\times U \quad (12.12)$$

Y la descomposición de **Cholesky**, en la cual U es la transpuesta de L, es decir:

$$A=\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{bmatrix}=\begin{bmatrix} l_{11} & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix}\times\begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{34} \\ 0 & l_{22} & l_{32} & l_{34} \\ 0 & 0 & l_{33} & l_{43} \\ 0 & 0 & 0 & l_{44} \end{bmatrix}=L\times U \quad (12.13)$$

Si la matriz A, es la matriz de los coeficientes de un sistema de ecuaciones lineales, como el siguiente:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x_4 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x_4 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x_4 &= b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x_4 &= b_4 \end{aligned} \quad (12.14)$$

Una vez realizada la factorización, las soluciones pueden ser calculadas, empleando las matrices triangulares obtenidas:

$$A\cdot x=L\cdot U\cdot x=b \quad (12.15)$$

Multiplicando ambos lados de la ecuación por L^{-1}, se obtiene:

$$L^{-1}\cdot L\cdot U\cdot x=I\cdot U\cdot x=U\cdot x=L^{-1}\cdot b=c \quad (12.16)$$

De donde se obtienen las igualdades:

$$U\cdot x=c \quad (12.17)$$

$$L^{-1}\cdot b=c \quad (12.18)$$

Primero se calculan los valores de "c" y para ello se multiplican ambos lados de la ecuación (12.18) por L:

$$L\cdot L^{-1}\cdot b=I\cdot b=b=L\cdot c \quad (12.19)$$

En consecuencia el valor de "c" puede ser calculado resolviendo el sistema de ecuaciones lineales:

$$L\cdot c=b \quad (12.20)$$

Este sistema puede ser resuelto porque es un sistema triangular inferior, en el cual las soluciones se encuentran por sustitución hacia adelante.

Por ejemplo para el sistema (12.14) el sistema triangular superior que se forma (asumiendo la descomposición de Crout) es:

$$\begin{aligned} l_{11}c_1 &= b_1 \\ l_{21}c_1 + l_{22}c_2 &= b_2 \\ l_{31}c_1 + l_{32}c_2 + l_{33}c_3 &= b_3 \\ l_{41}c_1 + l_{42}c_2 + l_{43}c_3 + l_{44}c_4 &= b_4 \end{aligned} \qquad (12.21)$$

Como se puede ver, c_1 se calcula con la primera ecuación, c_2 con la segunda y así sucesivamente. En general, para un sistema con "n" ecuaciones los valores de "c" se calculan con:

$$c_i = \frac{b_i - \sum_{k=1}^{i-1} l_{ik} \cdot c_k}{l_{ii}} \quad \{ i = 1 \rightarrow n \qquad (12.22)$$

Esta ecuación puede ser generalizada para el caso en el que "b" es una matriz en lugar de un vector (que es lo que ocurre cuando se resuelven simultáneamente sistemas de ecuaciones lineales o se calcula la inversa):

$$c_{ij} = \frac{b_{ij} - \sum_{k=1}^{i-1} l_{ik} \cdot c_{kj}}{l_{ii}} \quad \begin{cases} i = 1 \rightarrow n \\ j = 1 \rightarrow m \end{cases} \qquad (12.23)$$

Donde "m" es el número de columnas de la matriz "b" y "n" es el número de filas del sistema de ecuaciones.

Una vez calculado el vector "c", se encuentran las soluciones resolviendo el sistema de ecuaciones lineales (12.17). En este caso se trata de un sistema triangular superior y en consecuencia puede ser resuelto por sustitución hacia atrás (igual que en el método de Gauss). Por ejemplo para el sistema (12.14) el sistema triangular superior que se forma (asumiendo la descomposición de Crout) es:

$$\begin{aligned} x_1 + u_{12}x_2 + u_{13}x_3 + u_{14}x_4 &= c_1 \\ x_2 + u_{23}x_3 + u_{24}x_4 &= c_2 \\ x_3 + u_{34}x_4 &= c_3 \\ x_4 &= c_4 \end{aligned} \qquad (12.24)$$

Por lo tanto x_4 se calcula con la última ecuación, x_3 con la penúltima y así sucesivamente. En general, para un sistema con "n" ecuaciones los valores de "x_i" se calculan con:

$$x_i = \frac{c_i - \sum_{k=i+1}^{n} u_{ik} \cdot x_k}{u_{ii}} \quad \{ i = n \rightarrow 1 \qquad (12.25)$$

Para el caso general en el que "c" es una matriz, se tiene:

$$x_{ij} = \frac{c_{ij} - \sum_{k=i+1}^{n} u_{ik} \cdot x_{kj}}{u_{ii}} \quad \begin{cases} i = n \rightarrow 1 \\ j = 1 \rightarrow m \end{cases} \qquad (12.26)$$

Donde "m" es el número de columnas de la matriz "c" y "n" es el número de filas.

La factorización (o descomposición) matricial es particularmente útil cuando en un problema dado sólo cambian las constantes del sistema. Cuando ello ocurre se calculan las matrices "L" y "U" una sola vez y luego, cada vez que las constantes cambian, las soluciones se obtienen aplicando las ecuaciones (12.23) y (12.26) (sin necesidad de recalcular "L" y "U").

Las ecuaciones que permiten calcular las matrices "L" y "U" en la descomposición de Crout, pueden ser deducidas tomando en cuenta que la multiplicación de las filas de la matriz "L" por la primera columna de la matriz "U" es igual a la primera columna de la matriz "A"; que la multiplicación de la primera fila de la matriz "L" por las columnas 2 a "n" de la matriz "U" es igual a los elementos 2 a "n" de la primera fila de la matriz "A"; que la multiplicación de las filas de la matriz "L" por la segunda columna de "U" es igual a la segunda columna de "A"; que la multiplicación de la segunda fila de la matriz "L" por las columnas 3 a "n" de la matriz "U" es igual a los elementos 3 a "n" de "A" y así sucesivamente.

De esa manera se deducen las siguientes expresiones:

$$\left\{\begin{array}{ll} l_{ip}=a_{ip}-\sum_{k=1}^{p-1} l_{ik}\cdot u_{kp} & \{i=p\rightarrow n \\ u_{pp}=1 & \\ u_{pj}=\dfrac{a_{pj}-\sum_{k=1}^{p-1} l_{pk}\cdot u_{kj}}{l_{pp}} & \{j=p+1\rightarrow n \end{array}\right. \quad p=1\rightarrow n \qquad (12.27)$$

Si no se almacenan los unos de la matriz "U", las matrices "L" y "U" pueden ser almacenadas en una sola matriz, así las dos matrices de la ecuación (12.11), pueden ser almacenadas como:

$$\begin{bmatrix} l_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & l_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & l_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \qquad (12.28)$$

Donde se sabe que la matriz triangular "U" tiene unos en su diagonal principal. Así se ahorra memoria y, más importante aún, se simplifican los cálculos. Si estos elemento se almacenan en la matriz "A", las ecuaciones (12.27) se transforman en:

$$\left\{\begin{array}{ll} a_{ip}=a_{ip}-\sum_{k=1}^{p-1} a_{ik}\cdot a_{kp} & \{i=p\rightarrow n \\ a_{pj}=\dfrac{a_{pj}-\sum_{k=1}^{p-1} a_{pk}\cdot a_{kj}}{a_{pp}} & \{j=p+1\rightarrow n \end{array}\right. \quad p=1\rightarrow n \qquad (12.29)$$

Las ecuaciones (12.23) y (12.26), con las cuales se calculan las soluciones del sistema, deben ser reescritas para tomar en cuenta este cambio, pero si además las soluciones se almacenan en la matriz de constantes "b" y se toma en cuenta que los elementos de la diagonal principal en "U" son unos, se transforman en:

$$b_{ij}=\frac{b_{ij}-\sum_{k=1}^{i-1} a_{ik}\cdot b_{kj}}{a_{ii}} \quad \begin{cases} i=1\rightarrow n \\ j=1\rightarrow m \end{cases} \tag{12.30}$$

$$b_{ij}=b_{ij}-\sum_{k=i+1}^{n} a_{ik}\cdot b_{kj} \quad \begin{cases} i=n\rightarrow 1 \\ j=1\rightarrow m \end{cases} \tag{12.31}$$

A esta forma se denomina esquema compacto y es la forma más empleada en la práctica.

Para comprender mejor el método se aplicará manualmente el mismo a la matriz de coeficientes del sistema (12.6). Los datos con los que se cuenta son:

```
>> a=[2,8,2;1,6,-1;2,-1,2]; n=3; p=1;
```

Ahora se calculan los elementos de la primera columna (de "l"):

```
>> for i=p:n s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end a(i,p)=a(i,p)-s; end
a
ans =
  2   8   2
  1   6   -1
  2   -1  2
```

Entonces se calculan los elementos de la primera fila (de "u"):

```
>> for j=p+1:n s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end a(p,j)=(a(p,j)-
s)/a(p,p); end a
ans =
  2   4   1
  1   6   -1
  2   -1  2
```

Para calcular la segunda columna (de "l") y la segunda fila (de "u") se incrementa el valor de "p" (a 2) y se repite el proceso:

```
>> p++;
>> for i=p:n s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end a(i,p)=a(i,p)-s; end
a
ans =
  2   4   1
  1   2   -1
  2   -9  2
>> for j=p+1:n s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end a(p,j)=(a(p,j)-
s)/a(p,p); end a
ans =
  2   4   1
  1   2   -1
  2   -9  2
```

Finalmente se incrementa el valor de "p" (a 3) y se calcula la última columna (de "l"):

```
>> p++;
>> for i=p:n s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end a(i,p)=a(i,p)-s; end
a
ans =
  2   4   1
  1   2   -1
  2   -9  -9
```

De esa manera se obtienen las matrices triangulares "l" (los elementos de la diagonal principal e inferiores) y "u" (los elementos encima de la diagonal principal).

Ahora se calcula las soluciones aplicando primero la ecuación (12.30) (con j=1 pues en este caso "b" es un vector columna):

```
>>  b=[14;13;5]; j=1;
>> for i=1:n s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end b(i,j)=(b(i,j)-
s)/a(i,i); end b
ans =
  7
  3
  -2
```

Y luego la ecuación (12.31):

```
>> for i=n:-1:1 s=0; for k=i+1:n s+=a(i,k).*b(k,j); end b(i,j)=b(i,j)-s;
end b
ans =
  5
  1
  -2
```

Con lo que se obtienen las soluciones del sistema.

12.2.1. Ejercicio

4. Encuentre las soluciones del sistema (12.7) aplicando manualmente el método de factorización de Crout.

12.2.2. Pivotaje parcial en los métodos de factorización

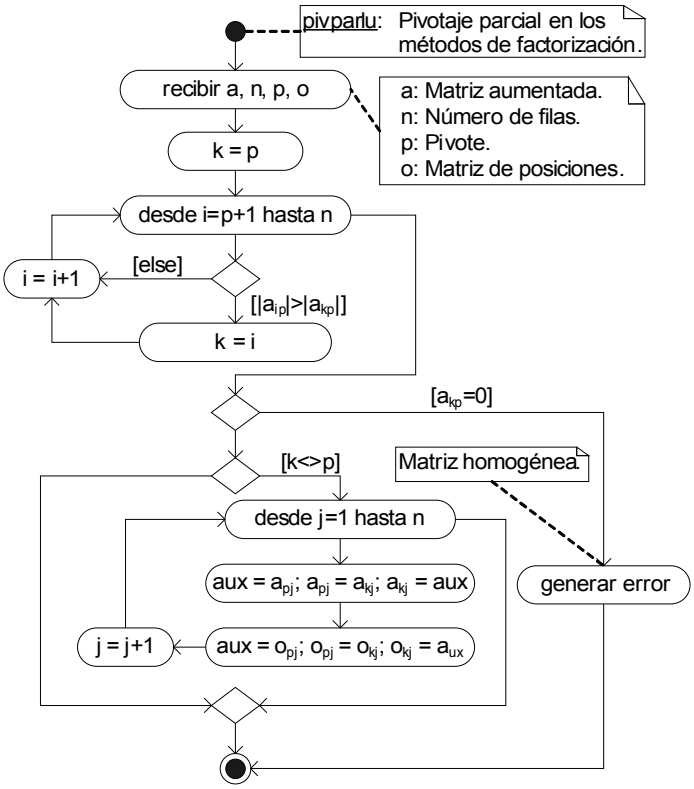
Al igual que en los métodos de Gauss y Gauss Jordán en los métodos de factorización se debe evitar que el elemento pivote sea cero y con ese fin es necesario realizar un pivotaje parcial.

En estos métodos, sin embargo, se debe mantener un registro de los intercambios de filas realizados, pues es necesario ordenar la matriz de las constantes ("b") en función a dichos intercambios. Alternativamente se puede trabajar con la matriz aumentada, no obstante, con ello se pierde la principal ventaja de estos métodos: la de calcular nuevas soluciones empleando las matrices "l" y "u" ya calculadas (o la matriz compacta).

Para hacer el seguimiento de los intercambios realizados, se empleará en este caso una matriz de permutaciones "o", que inicialmente será igual a la matriz unidad. Cada vez que se intercambien dos filas en la matriz "A", se intercambiarán las mismas filas en la matriz de permutaciones "o", entonces, los intercambios de filas hechos en la matriz "A" podrán ser replicados en la matriz "b" simplemente multiplicándola por la matriz "o".

El algoritmo del pivotaje parcial en los métodos de factorización se presenta en la siguiente página y el código respectivo es:

```
function r=pivparlu(a,n,p,o)
  k=p;
  for i=p+1:n
    if abs(a(i,p))>abs(a(k,p)) k=i; end
  end
  if a(k,p)==0 error("Matriz Homogénea\n"); end
  if k ~= p
    for j=1:n
```




```
      aux=a(p,j); a(p,j)=a(k,j); a(k,j)=aux;
      aux=o(p,j); o(p,j)=o(k,j); o(k,j)=aux;
    end
  end
end
```

Haciendo correr el programa con la matriz:

```
>> a=[0,2,1;1,0,0;3,0,1]
a =
  0  2  1
  1  0  0
  3  0  1
```

Donde se deben intercambiar las filas 1 y 3, se obtiene:

```
>> o=eye(3);
>> pivparlu(a,3,1,o)
>> a,o
a =
  3  0  1
  1  0  0
  0  2  1
o =
  0  0  1
  0  1  0
  1  0  0
```

Que corresponde al intercambio esperado, el mismo que se refleja también la matriz "o". Si se aplica pivotaje parcial a la matriz resultante, siendo el pivote 2, se obtiene:

```
>> pivparlu(a,3,2,o)
>> a,o
```

```
a =
  3  0  1
  0  2  1
  1  0  0
o =
  0  0  1
  1  0  0
  0  1  0
```

Que una vez más corresponde a los intercambios de filas esperados (filas 2 y 3) y el respectivo intercambio en la matriz de permutaciones ("o").

Simplemente para demostrar que la matriz de permutaciones "o" permite reproducir los intercambios de filas realizados, se multiplica la matriz de permutaciones por la matriz original:

```
>> o*[0,2,1;1,0,0;3,0,1]
ans =
  3  0  1
  0  2  1
  1  0  0
```

Obteniéndose la matriz resultante de los intercambios.

12.2.3. Factorización LU

El algoritmo para factorizar (descomponer) una matriz por el método de Crout, se presenta en la siguiente página.

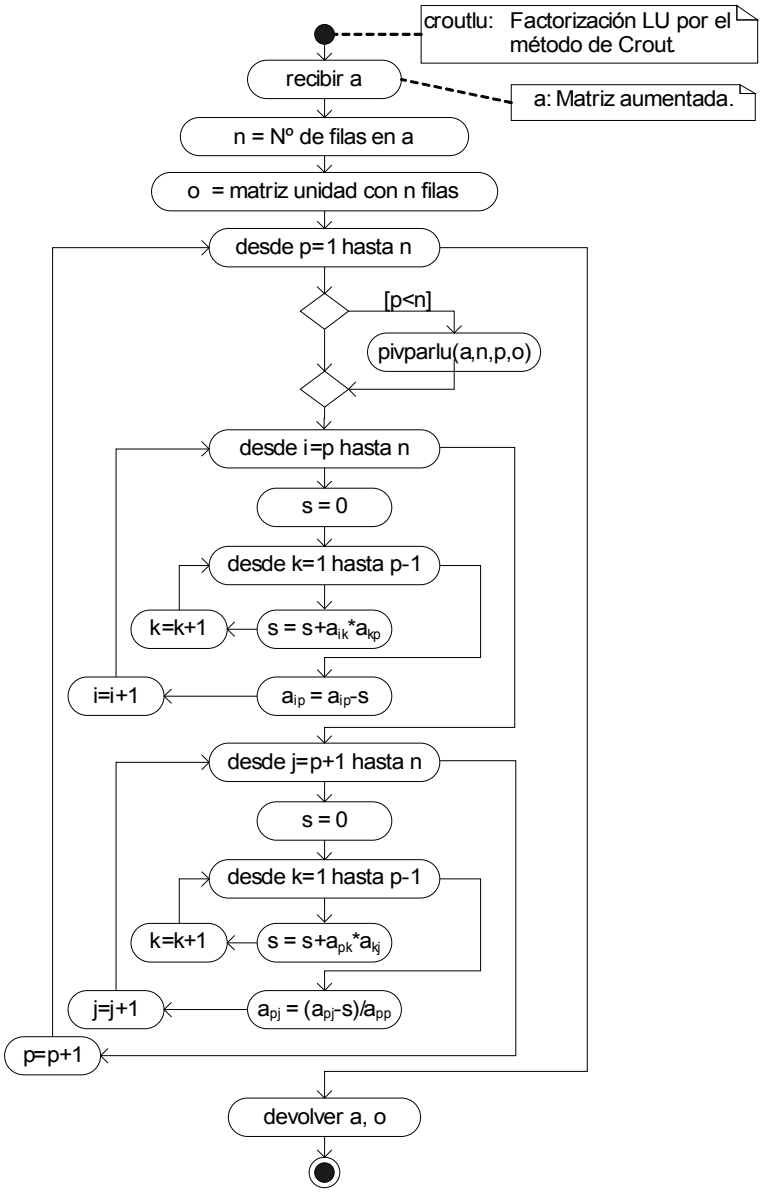
Al momento de codificar el algoritmo se hace una modificación porque Jasymca no permite devolver más de un resultado. Por esa razón las matrices "a" y "o" se devuelven en una sola, en la matriz "a", la cual se devuelve aumentada con las columnas de la matriz "o":

```
function a=croutlu(b)
  a=0; a=b(:,:); n=size(a)(1); o=eye(n);
  for p=1:n
    if p<n pivparlu(a,n,p,o); end
    for i=p:n
      s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end
      a(i,p)=a(i,p)-s;
    end
    for j=p+1:n
      s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end
      a(p,j)=(a(p,j)-s)/a(p,p);
    end
  end
  a(:,n+1:2*n)=o;
end
```

Haciendo correr el programa con la matriz del ejemplo manual (sistema (12.6), se obtienen los mismos resultados que en el ejemplo:

```
>> a=[2,8,2;1,6,-1;2,-1,2];
>> lu=croutlu(a)
lu =
  2   4   1   1   0   0
  1   2   -1  0   1   0
  2   -9  -9  0   0   1
```

Como Las 3 últimas columnas corresponden a la matriz de permutaciones "o".

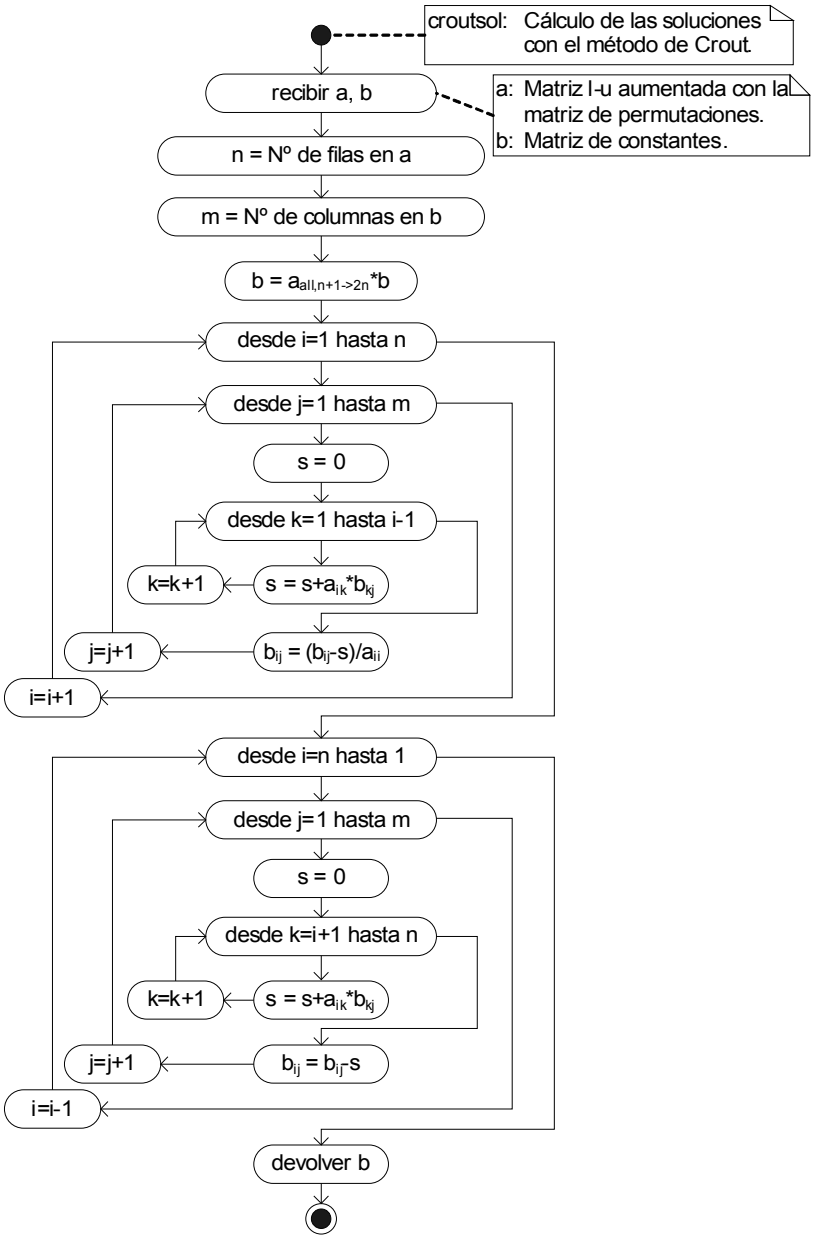


12.2.4. Cálculo de las soluciones

Una vez que se tienen las matrices triangulares inferior, superior y la matriz de permutaciones (en una matriz aumentada), se pueden calcular los resultados aplicando los procesos de sustitución hacia adelante y hacia atrás (ecuaciones (12.30) y (12.31)).

El algoritmo se presenta en la siguiente página y el código respectivo es:

```
function b=croutsol(a,c)
  b=0; b=c(:,:); n=size(a)(1); m=size(b)(2);
  b=a(:,n+1:2*n)*b;
  for i=1:n
    for j=1:m
      s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end
      b(i,j)=(b(i,j)-s)/a(i,i);
    end
  end
```




```
  for i=n:-1:1
    for j=1:m
      s=0; for k=i+1:n s+=a(i,k).*b(k,j); end
      b(i,j)=b(i,j)-s;
    end
  end
end
```

Haciendo correr el programa con los resultados de "croutlu" se obtiene:

```
>> b=[14;13;5];
>> croutsol(lu,b)
b =
  5
  1
  -2
```

Que son las soluciones correctas.

12.2.5. Ejercicio

5. Encuentre la inversa de la matriz de coeficientes del sistema de ecuaciones (12.9) empleando el método de Crout.

12.3. MÉTODO DE DOOLITTLE

Como ya se dijo, cuando en la factorización L-U, la matriz triangular inferior queda con unos en la diagonal principal, el proceso se conoce como el método de Doolittle (ecuación (12.12)).

Las ecuaciones para el cálculo de las matrices "L" y "U" son:

$$\left.\begin{array}{l} u_{pj}=a_{pj}-\sum_{k=1}^{p-1} l_{pk}\cdot u_{kj} \quad \{j=p\rightarrow n \\ l_{pp}=1 \\ l_{ip}=\dfrac{a_{ip}-\sum_{k=1}^{p-1} l_{ik}\cdot u_{kp}}{u_{pp}} \quad \{i=p+1\rightarrow n \end{array}\right\} p=1\rightarrow n \qquad (12.32)$$

Y se deducen de manera similar al método de Crout, sólo que ahora los elementos de la matriz original "A" se calculan en el orden: fila 1, columna 1, fila 2, columna 2, fila 3, columna 3 y así sucesivamente.

Sin embargo, y al igual que en el método de Crout, no es necesario guardar los ceros ni los unos de las matrices, por lo que las dos matrices ("L" y "U") pueden ser guardadas en una:

$$\begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ l_{21} & u_{22} & u_{23} & u_{24} \\ l_{31} & l_{32} & u_{33} & u_{34} \\ l_{41} & l_{42} & l_{43} & u_{44} \end{bmatrix} \qquad (12.33)$$

Si esta matriz se guarda en la matriz "A", las ecuaciones (12.32) se transforman en:

$$\left.\begin{array}{l} a_{pj}=a_{pj}-\sum_{k=1}^{p-1} a_{pk}\cdot a_{kj} \quad \{j=p\rightarrow n \\ a_{ip}=\dfrac{a_{ip}-\sum_{k=1}^{p-1} a_{ik}\cdot a_{kp}}{a_{pp}} \quad \{i=p+1\rightarrow n \end{array}\right\} p=1\rightarrow n \qquad (12.34)$$

Las ecuaciones para el cálculo de las soluciones, tomando en cuenta que ahora "l" tiene la diagonal con unos y guardando las soluciones en la matriz de constantes "b", son:

$$b_{ij}=b_{ij}-\sum_{k=1}^{i-1} a_{ik}\cdot b_{kj} \quad \begin{cases} i=1\rightarrow n \\ j=1\rightarrow m \end{cases} \qquad (12.35)$$

$$b_{ij}=\frac{b_{ij}-\sum_{k=i+1}^{n} a_{ik}\cdot b_{kj}}{a_{ii}} \quad \begin{cases} i=n\rightarrow 1 \\ j=1\rightarrow m \end{cases} \qquad (12.36)$$

Para comprender mejor el proceso se aplicará manualmente el método de Doolittle al sistema (12.6).

Primero se crea la matriz "a" y se inicializan las variables "n" y "p":

```
>> a=[2,8,2;1,6,-1;2,-1,2]; n=length(a); p=1;
```

Ahora se aplican las ecuaciones (12.34) para calcular la primera fila y columna de "u" y "l":

```
>> for j=p:n s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end a(p,j)=a(p,j)-s; end
a
ans =
  2   8   2
  1   6   -1
  2   -1  2
>> for i=p+1:n s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end a(i,p)=(a(i,p)-
s)/a(p,p); end; a
ans =
  2    8    2
  0.5  6    -1
  1    -1   2
```

Ahora se procede con la segunda fila y columna de "u" y "l":

```
>> p++;
>> for j=p:n s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end a(p,j)=a(p,j)-s; end
a
ans =
  2    8    2
  0.5  2    -2
  1    -1   2
>> for i=p+1:n s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end a(i,p)=(a(i,p)-
s)/a(p,p); end; a
ans =
  2     8     2
  0.5   2     -2
  1     -4.5  2
```

Finalmente se calcula la última fila de "u":

```
>> p++;
>> for j=p:n s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end a(p,j)=a(p,j)-s; end
a
ans =
  2     8     2
  0.5   2     -2
  1     -4.5  -9
```

Ahora se calculan las soluciones con las ecuaciones (12.35) y (12.36):

```
>> b=[14;13;5]; j=1;
>> for i=1:n s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end b(i,j)=b(i,j)-s; end
b
ans =
  14
  6
  18
>> for i=n:-1:1 s=0; for k=i+1:n s+=a(i,k).*b(k,j); end b(i,j)=(b(i,j)-
s)/a(i,i); end b
ans =
  5
  1
```

-2

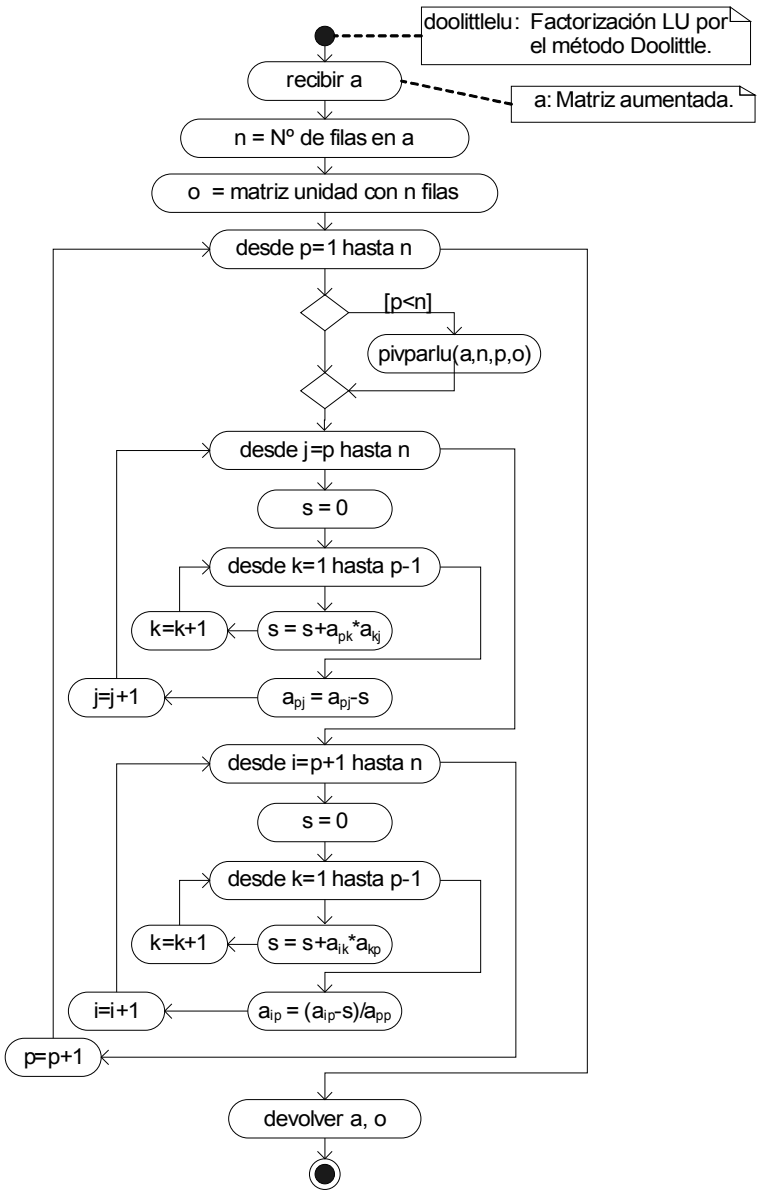
Que son las soluciones correctas.

12.3.1. Ejercicio

6. Encuentre las soluciones del sistema (12.7) aplicando manualmente el método de factorización de Doolittle.

12.3.2. Fatorización LU

El algoritmo para calcular las matrices "L" y "U" por el método de Doolittle, el cual, con excepción del pivotaje parcial, se automatiza el proceso seguido en el ejemplo manual, es el siguiente:



Siendo el código respectivo:

```
function a=doolittlelu(b)
  a=0; a=b(:,:); n=size(a)(1); o=eye(n);
```

```
    for p=1:n
      if p<n pivparlu(a,n,p,o); end
      for j=p:n
        s=0; for k=1:p-1 s+=a(p,k).*a(k,j); end
        a(p,j)=a(p,j)-s;
      end
      for i=p+1:n
        s=0; for k=1:p-1 s+=a(i,k).*a(k,p); end
        a(i,p)=(a(i,p)-s)/a(p,p);
      end
    end
    a(:,n+1:2*n)=o;
  end
```

Donde, al igual que en el método de Crout, se ha añadido a la matriz l-u la matriz de permutaciones. Haciendo correr el programa con la matriz del ejemplo manual se obtiene:

```
>> a=[2,8,2;1,6,-1;2,-1,2];
>> r=doolittlelu(a)
r =
  2     8     2     1     0     0
  0.5   2     -2    0     1     0
  1     -4.5  -9    0     0     1
```

Que son los resultados obtenidos también en el ejemplo manual.

Jasymca cuenta también con una función “lu”, que lleva a cabo la descomposición por el método de Doolittle. Con esta función se obtiene:

```
>> [l,u,o]=lu(a)
l =
  1         0         0
  0.5       -0.22222  1
  1         1         0
u =
  2   8   2
  0   -9  0
  0   0   -2
o =
  1  0  0
  0  0  1
  0  1  0
```

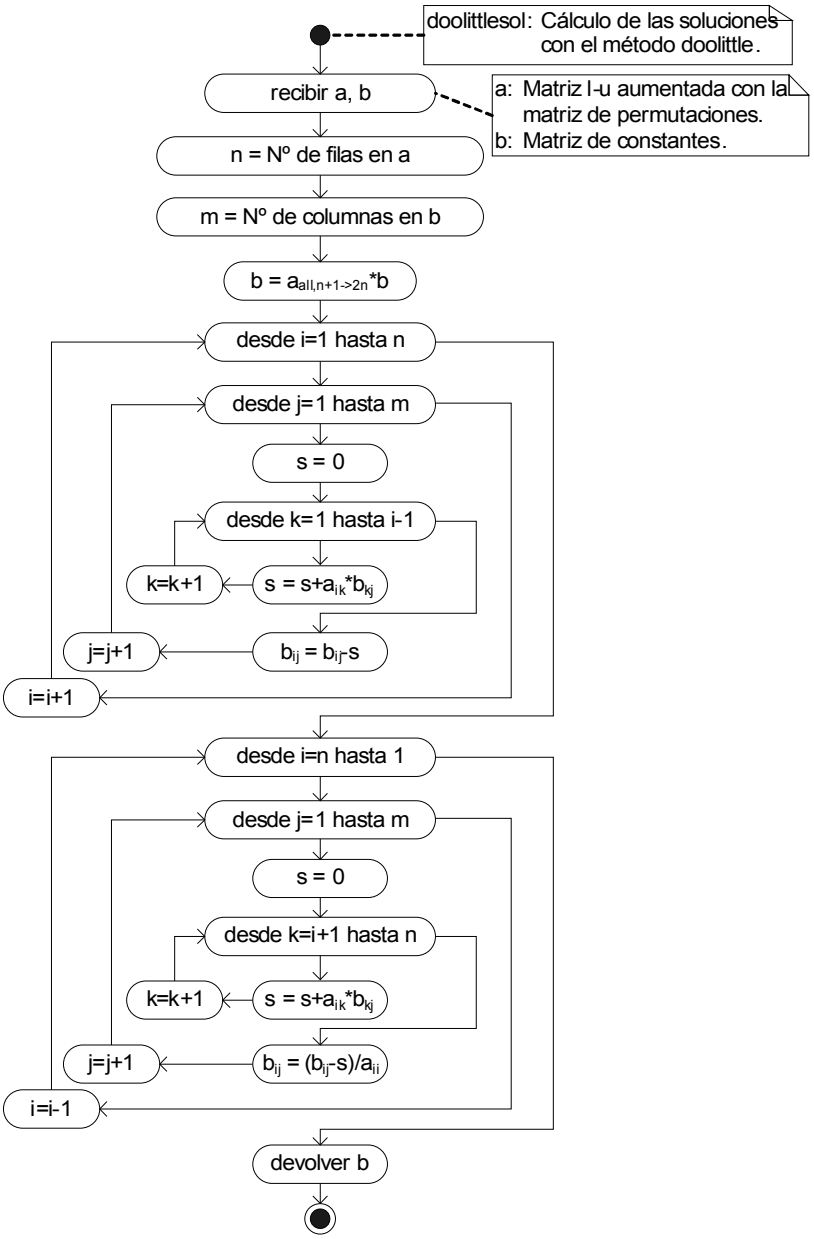
Que, a diferencia del programa elaborado, devuelve los resultados en tres matrices y lleva a cabo un pivotaje total en lugar de uno parcial.

12.3.3. Cálculo de las soluciones

Con las matrices triangulares inferior, superior y la matriz de permutaciones (en una matriz aumentada), se pueden calcular las soluciones del sistema aplicando los procesos de sustitución hacia adelante y hacia atrás (ecuaciones (12.35) y (12.36)).

El algoritmo se presenta en la siguiente página y el código respectivo es:

```
function b=doolittlesol(a,c)
  b=0; b=c(:,:); n=size(a)(1); m=size(b)(2);
  b=a(:,n+1:2*n)*b;
  for i=1:n
    for j=1:m
```




```
      s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end
      b(i,j)=b(i,j)-s;
    end
  end
  for i=n:-1:1
    for j=1:m
      s=0; for k=i+1:n s+=a(i,k).*b(k,j); end
      b(i,j)=(b(i,j)-s)/a(i,i);
    end
  end
end
```

Haciendo correr el programa con los resultados de "doolittlelu" se obtienen las soluciones encontradas en el ejemplo manual:

```
>> b=[14;13;5];
>> doolittlesol(r,b)
b =
```

```
5
1
-2
```

12.3.4. Ejercicios

7. Cree la matriz de coeficientes "a" y la matriz de constantes "b" del sistema de ecuaciones (12.8). Luego, empleando el método de Doolittle, encuentre primero las soluciones de cada uno de los sistemas independientemente y luego todas las soluciones a la vez.

12.4. MÉTODO DE CHOLESKY

Como ya se vio, cuando la matriz es simétrica, la descomposición "L" "U" puede ser simplificada y al proceso se conoce como el método de Cholesky (ecuación (12.13). Puesto que la matriz a factorizar debe ser simétrica, el método de Cholesky no admite pivotaje.

A pesar de estas limitaciones, la forma de Cholesky resulta de utilidad práctica, pues los sistemas con matrices simétricas aparecen en la resolución de varios problemas prácticos en el campo de la ingeniería.

Las ecuaciones que permiten calcular los valores de "L" (y en consecuencia los de "U"), se deducen obteniendo los términos de la matriz triangular inferior de la matriz original "A", columna por columna y son:

$$\begin{aligned} l_{pp} &= \sqrt{a_{pp} - \sum_{k=1}^{p-1} l_{pk}^{\,2}} \\ l_{ip} &= \frac{a_{ip} - \sum_{k=1}^{p-1} l_{ik} \cdot l_{pk}}{l_{pp}} = l_{pi} \quad \{i = p+1 \rightarrow n \end{aligned} \left\{ p = 1 \rightarrow n \right. \qquad (12.37)$$

Como se puede ver, el cálculo de los elementos de la diagonal principal involucra el cálculo de la raíz cuadrada, por lo que dicho valor debe ser positivo. Al igual que en los métodos de Crout y Doolittle, no es necesario almacenar los ceros y dado que en este caso la matriz transpuesta y la original tienen los mismos elementos en la diagonal principal, los resultados de la factorización se pueden almacenar en una matriz:

$$\begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} \\ l_{21} & l_{22} & l_{32} & l_{42} \\ l_{31} & l_{32} & l_{33} & l_{43} \\ l_{41} & l_{42} & l_{43} & l_{44} \end{bmatrix} \qquad (12.38)$$

Si la matriz donde se guardan estos valores es la matriz "A", las ecuaciones (12.37) pueden ser reescritas como:

$$\begin{aligned} a_{pp} &= \sqrt{a_{pp} - \sum_{k=1}^{p-1} a_{pk}^{\,2}} \\ a_{ip} &= \frac{a_{ip} - \sum_{k=1}^{p-1} a_{ik} \cdot a_{pk}}{a_{pp}} = a_{pi} \quad \{i = p+1 \rightarrow n \end{aligned} \left\{ p = 1 \rightarrow n \right. \qquad (12.39)$$

Una vez factorizada la matriz de los coeficientes los resultados pueden ser calculados con las ecuaciones (12.35) y (12.36).

Para comprender mejor el proceso se resolverá el siguiente sistema de ecuaciones con este método:

$$\begin{aligned} 4x_1 + x_2 + 2x_3 &= 5 \\ x_1 + 7x_2 + 3x_3 &= 8 \\ 2x_1 + 3x_2 + 9x_3 &= 3 \end{aligned} \qquad (12.40)$$

Primero se crea la matriz de coeficientes y se inicializan "p" y "n":

```
>> a=[4,1,2;1,7,3;2,3,9]; n=size(a)(1); p=1;
```

Ahora se calcula el primer elemento de la diagonal principal y la primera columna (y por consiguiente la fila 1):

```
>> s=0; for k=1:p-1 s+=a(p,k)^2; end a(p,p)=sqrt(a(p,p)-s);  a
ans =
  2  1  2
  1  7  3
  2  3  9
>> for i=p+1:n s=0; for k=1:p-1 s+=a(i,k).*a(p,k); end a(i,p)=(a(i,p)-
s)/a(p,p); a(p,i)=a(i,p); end  a
ans =
  2    0.5  1
  0.5  7    3
  1    3    9
```

Se incrementa "p" y se repite el proceso para la segunda columna y fila:

```
>> p++;
>> s=0; for k=1:p-1 s+=a(p,k)^2; end a(p,p)=sqrt(a(p,p)-s);  a
ans =
  2       0.5     1
  0.5     2.5981  3
  1       3       9
>> for i=p+1:n s=0; for k=1:p-1 s+=a(i,k).*a(p,k); end a(i,p)=(a(i,p)-
s)/a(p,p); a(p,i)=a(i,p); end  a
ans =
  2        0.5      1
  0.5      2.5981   0.96225
  1        0.96225  9
```

Finalmente se calcula el último valor de la diagonal principal:

```
>> p++;
>> s=0; for k=1:p-1 s+=a(p,k)^2; end a(p,p)=sqrt(a(p,p)-s);  a
ans =
  2        0.5      1
  0.5      2.5981   0.96225
  1        0.96225  2.6597
```

Ahora las soluciones se calculan con las ecuaciones (12.35) y (12.36):

```
>> b=[5;8;3]; j=1;
>> for i=1:n s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end b(i,j)=(b(i,j)-
s)/a(i,i); end b
ans =
  2.5
  2.5981
  -0.75196
```

```
>> for i=n:-1:1 s=0; for k=i+1:n s+=a(i,k).*b(k,j); end b(i,j)=(b(i,j)-
s)/a(i,i); end b
ans =
  1.1152
  1.1047
  -0.28272
```

Que son las soluciones correctas.

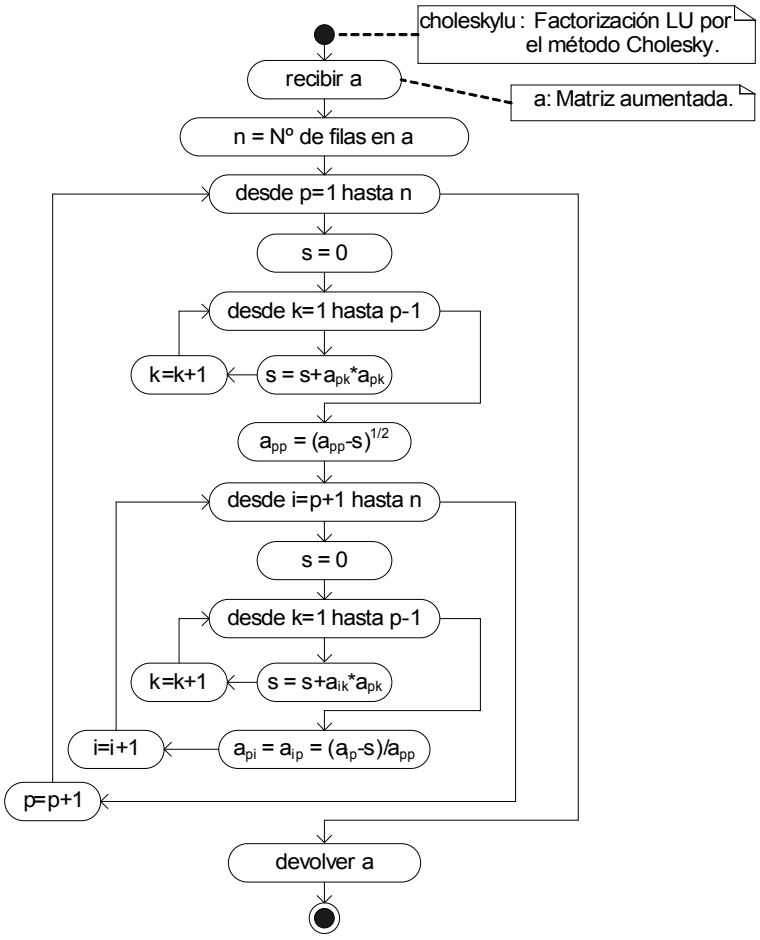
12.4.1. Ejercicio

8. Encuentre las soluciones del siguiente sistema de ecuaciones aplicando manualmente el método de Cholesky.

$$\begin{aligned} 10x_1 + x_2 + 2x_3 &= 44 \\ x_1 + 10x_2 + x_3 &= 51 \\ 2x_1 + x_2 + 10x_3 &= 61 \end{aligned} \qquad (12.41)$$

12.4.2. Factorización LU

El algoritmo que automatiza el proceso de cálculo es el siguiente:



Siendo el código respectivo:

```
function a=choleskylu(b)
  a=0; a=b(:,:); n=size(a)(1);
  for p=1:n
```

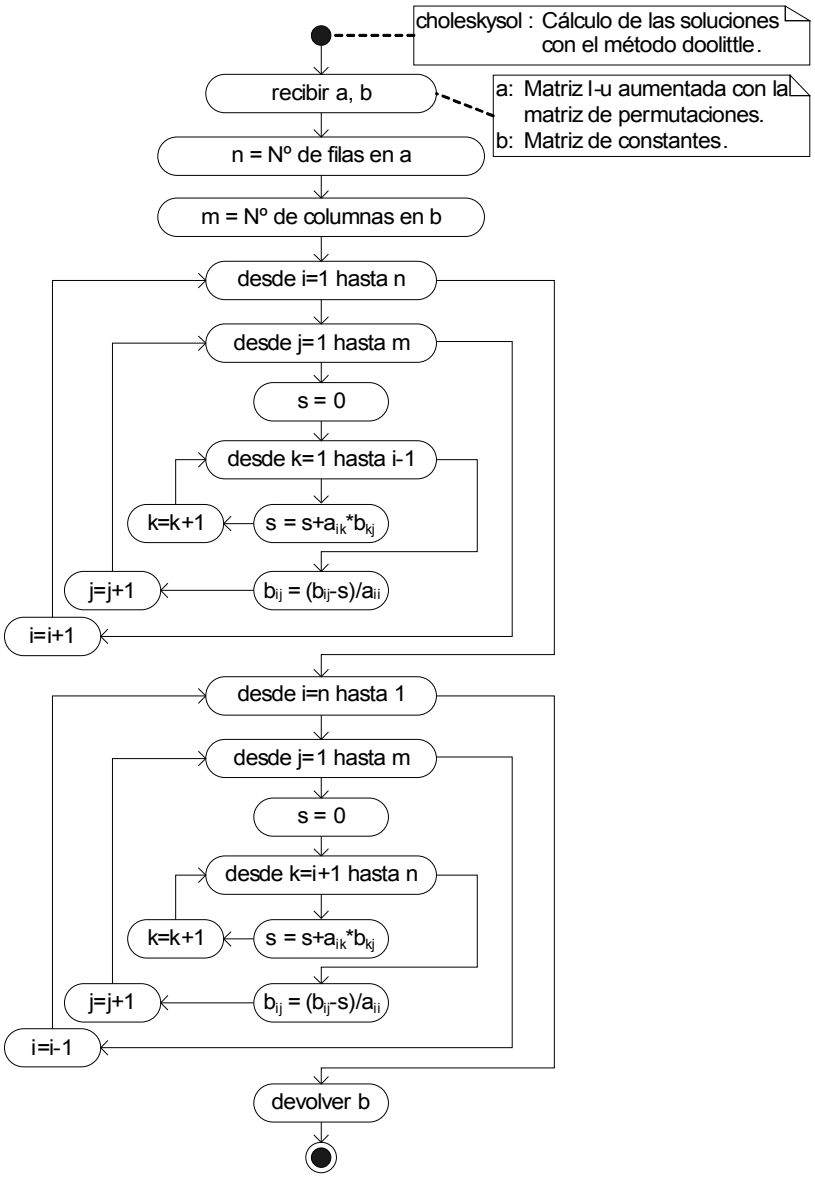
```
    s=0; for k=1:p-1 s+=a(p,k).*a(p,k); end
    a(p,p)=sqrt(a(p,p)-s);
    for i=p+1:n
      s=0; for k=1:p-1 s+=a(i,k).*a(p,k); end
      a(i,p)=(a(i,p)-s)/a(p,p); a(p,i)=a(i,p);
    end
  end
end
```

Haciendo correr el programa con la matriz del ejemplo manual se obtiene:

```
>> a=[4,1,2;1,7,3;2,3,9]; r=choleskylu(a)
r =
  2        0.5      1
  0.5      2.5981   0.96225
  1        0.96225  2.6597
```

12.4.3. Cálculo de las soluciones

El algoritmo que automatiza el proceso es:



Siendo el código respectivo:

```
function b=choleskysol(a,c)
  b=0; b=c(:,:); n=size(a)(1); m=size(b)(2);
  for i=1:n
    for j=1:m
      s=0; for k=1:i-1 s+=a(i,k).*b(k,j); end
      b(i,j)=(b(i,j)-s)/a(i,i);
    end
  end
  for i=n:-1:1
    for j=1:m
      s=0; for k=i+1:n s+=a(i,k).*b(k,j); end
      b(i,j)=(b(i,j)-s)/a(i,i);
    end
  end
end
```

Haciendo correr el programa con los resultados de "choleskylu" se obtiene:

```
>> b=[5;8;3];
>> choleskysol(r,b)
b =
  1.1152
  1.1047
  -0.28272
```

12.4.4. Ejercicio

9. Encuentre las soluciones del siguiente sistema de ecuaciones aplicando el método de Cholesky.

$$\begin{aligned} 7x_1 + x_2 + 2x_3 - x_4 &= 3 \\ x_1 + 9x_2 + 3x_3 + 2x_4 &= 2 \\ 2x_1 + 3x_2 + 12x_3 + 3x_4 &= 7 \\ -x_1 + 2x_2 + 3x_3 + 11x_4 &= 5 \end{aligned} \qquad (12.42)$$

13. SISTEMAS DE ECUACIONES LINEALES 3

Cuando el número de ecuaciones en un sistema es elevado (con cientos o miles de ecuaciones) los métodos directos estudiados en temas previos no son de utilidad práctica y ello se debe a que en dichos métodos 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. Esto da origen a la propagación del error, donde los errores de redondeo se van acumulando en cada nuevo cálculo y cuando el número de ecuaciones es muy elevado el error acumulado es tan grande que los resultados son de hecho erróneos.

En estos casos se debe recurrir a 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. En este tema se estudian dos de dichos métodos: Jacobi y Gauss-Seidel.

Por otra parte, en la resolución de problemas en el campo de la ciencia y la ingeniería suelen aparecer, con cierta frecuencia, sistemas que sólo tienen elementos en la diagonal principal y las diagonales adyacentes. Dichos sistemas se denominan sistemas tridiagonales y para los mismos se han adaptado los métodos de manera que no sea necesario almacenar los ceros (que en este caso son la mayoría). En este tema se han adaptado dos métodos para este fin: Gauss y Gauss-Seidel.

13.1. MÉTODO DE JACOBI

Uno de los métodos iterativos más sencillos es el de ***Jacobi*** (que es equivalente al método de sustitución directa pero para sistemas de ecuaciones lineales).

En este método se asumen valores iniciales para todas las incógnitas (generalmente 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, de ser así el proceso concluye y se tienen las soluciones del sistema (los valores calculados), caso contrario los valores calculados se convierten en valores asumidos y se repite el proceso.

Para comprender mejor el método y contar con valores de referencia para elaborar el programa, se resolverá el siguiente sistema de ecuaciones lineales:

$$\begin{aligned} 10x_1 + x_2 + 2x_3 &= 44 \\ 2x_1 + 10x_2 + x_3 &= 51 \\ x_1 + 2x_2 + 10x_3 &= 61 \end{aligned} \qquad (13.1)$$

Se asumen valores iniciales iguales a cero. Con estos ceros, se calculan los nuevos valores, a los cuales se denominarán "y", empleando la primera ecuación para calcular el primer valor (y_1), la segunda para el segundo y la tercera para el tercero:

$$y_1 = \frac{44 - x_2 - 2x_3}{10} = \frac{44 - 0 - 2*0}{10} = 4.4$$

$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*0-0}{10}=5.1$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-0-2*0}{10}=6.1$$

Como los valores iniciales son diferentes a los valores asumidos (ceros) el proceso se repite. Para ellos los valores calculados se convierten en nuevos valores de prueba, es decir: $x_1=y_1=4.4$, $x_2=y_2=5.1$ y $x_3=y_3=6.1$:

$$y_1=\frac{44-x_2-2x_3}{10}=\frac{44-5.1-2*6.1}{10}=2.67$$
$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*4.4-6.1}{10}=3.61$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-4.4-2*5.1}{10}=4.64$$

Una vez más los valores asumidos y calculados son diferentes, por lo que es necesario repetir el proceso empleando los valores calculados como nuevos valores de prueba:

$$y_1=\frac{44-x_2-2x_3}{10}=\frac{44-3.61-2*4.64}{10}=3.111$$
$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*2.67-4.64}{10}=4.102$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-2.67-2*3.61}{10}=5.111$$

Como los valores asumidos y calculados siguen siendo diferentes, se repite el proceso una vez más:

$$y_1=\frac{44-x_2-2x_3}{10}=\frac{44-4.102-2*5.111}{10}=2.9676$$
$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*3.111-5.111}{10}=3.9667$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-3.111-2*4.102}{10}=4.9685$$

Ahora los valores de las dos últimas iteraciones son parecidos, pero todavía existen diferencias apreciables entre ellos, por lo que el proceso debe ser repetido:

$$y_1=\frac{44-x_2-2x_3}{10}=\frac{44-3.9667-2*4.9685}{10}=3.00963$$
$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*2.9676-4.9685}{10}=4.00963$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-2.9676-2*3.9667}{10}=5.0099$$

Repitiendo el proceso una vez más se obtiene:

$$y_1=\frac{44-x_2-2x_3}{10}=\frac{44-4.00963-2*5.0099}{10}=2.997057$$

$$y_2=\frac{51-2x_1-x_3}{10}=\frac{51-2*3.00963-5.0099}{10}=3.997084$$
$$y_3=\frac{61-x_1-2x_2}{10}=\frac{61-3.00963-2*4.00963}{10}=4.997111$$

Si sólo se trabaja con tres dígitos, los valores asumidos y calculados son iguales. Por lo tanto, las soluciones, con 3 dígitos de precisión, son: x_1=3.00, x_2=4.00 y x_3=5.00.

13.1.1. Ejercicio

1. Encuentre, con 5 dígitos de precisión, las soluciones del siguientes sistema aplicando manualmente el método de Jacobi.

$$\begin{aligned} 22x_1 - 2x_2 + 5x_3 + 2x_4 &= 13 \\ x_1 + 31x_2 + 3x_3 + 3x_4 &= 20 \\ 3x_1 - x_2 + 43x_3 + 5x_4 &= 12 \\ 4x_1 + 3x_2 + x_3 + 21x_4 &= 15 \end{aligned} \qquad (13.2)$$

13.1.2. Algoritmo y código

Como se hace evidente al aplicar el método manualmente, y como ocurre con la mayoría de los métodos iterativos, la principal desventaja del método de Jacobi es el elevado número de cálculos requeridos, por lo que sólo se justifica su uso si el método es programado.

Al ser un proceso iterativo es probable que en ocasiones no se logre convergencia, por lo que es necesario incluir un contador para terminar el proceso cuando se supere cierto límite de iteraciones.

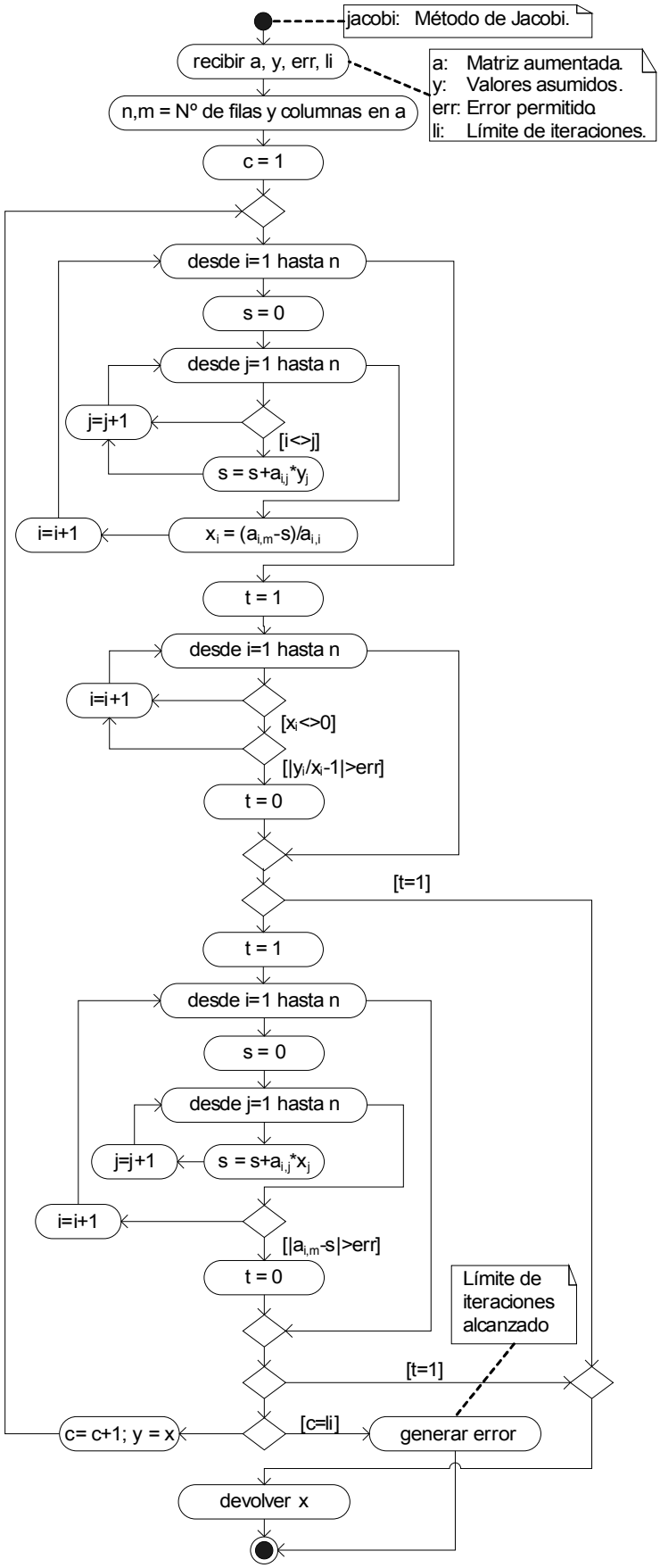
Es importante tomar en cuenta que l**a convergencia de este método está asegurada para sistemas con una *diagonal dominante***. Un sistema de ecuaciones lineales tiene una diagonal dominante cuando los valores absolutos de los elementos de la diagonal principal son mayores a la suma de los valores absolutos de los otros coeficientes de la misma fila (tal como ocurre en el ejemplo y en los ejercicios manuales).

En ocasiones es posible formar un sistema con una diagonal dominante ordenando las filas, si ello no es posible la convergencia del método no está asegurada. Lo bueno es que en muchos problemas en el campo de la ingeniería se forman sistemas con una diagonal dominante, por lo que el método de Jacobi resulta de utilidad práctica.

La ecuación general para el cálculo de los nuevos valores de "x", que puede ser deducida fácilmente analizando las ecuaciones empleadas en el ejemplo manual es:

$$x_i=\frac{a_{i,m}-\sum_{j=1}^{n} a_{i,j}\cdot y_j \{ j\neq i}{\{ i=1\rightarrow n} \qquad (13.3)$$

Donde "a" es la matriz aumentada del sistema de ecuaciones lineales, "n" es el número de filas y "m" el número de columnas. Con esta ecuación se calculan los nuevos valores de "x" hasta que son aproximadamente iguales a los asumidos ("y") o hasta que, al reemplazar los valores calculados, las ecuaciones son casi cero. El algoritmo que automatiza el proceso se presenta en la siguiente página:


jacobi: Método de Jacobi.
recibir a, y, err, li
a: Matriz aumentada.
y: Valores asumidos.
err: Error permitido.
li: Límite de iteraciones.
n,m = Nº de filas y columnas en a
c = 1
desde i=1 hasta n
s = 0
desde j=1 hasta n
j=j+1
[i<>j]
s = s+a_ij*y_j
i=i+1
x_i = (a_i,m-s)/a_i,i
t = 1
desde i=1 hasta n
i=i+1
[x_i<>0]
[|y_i/x_i-1|>err]
t = 0
[t=1]
t = 1
desde i=1 hasta n
s = 0
desde j=1 hasta n
j=j+1
s = s+a_i,j*x_j
i=i+1
[|a_i,m-s|>err]
t = 0
Límite de iteraciones alcanzado
[t=1]
[c=li]
generar error
c= c+1; y = x
devolver x

Siendo el código respectivo:

```
function x=jacobi(a,y,err,li)
  [n,m]=size(a); c=1; x=zeros(1,n);
  while 1
    for i=1:n
      s=0; for j=1:n if i ~= j s+=a(i,j).*y(j); end end
      x(i)=(a(i,m)-s)/a(i,i);
    end
    t=1;
    for i=1:n
      if x(i) ~= 0 if abs(y(i)/x(i)-1)>err t=0; break; end end
    end
    if t==1 return; end
    t=1;
    for i=1:n
      s=0; for j=1:n s+=a(i,j).*x(j); end;
      if abs(a(i,m)-s)>err t=0; break; end;
    end
    if t==1 return; end
    if c++ == li error("Jacobi no converge\n"); end
    y=x(:);
  end
end
```

Donde "zeros" devuelve una matriz con el número de filas y columnas especificado, en este caso, puesto que el número de filas es 1, se trata de un vector. Haciendo correr el programa con la matriz aumentada del sistema resuelto manualmente, con un error permitido igual a 1e-6, se obtiene:

```
>> a=[10,1,2,44;2,10,1,51;1,2,10,61]; y=zeros(1,3);
>> jacobi(a,y,1e-6,100)
x = [ 3  4  5 ]
```

Que son las soluciones del sistema con 6 dígitos de precisión.

13.1.3. Ejercicio

2. Encuentre, con 12 dígitos de precisión, las soluciones del siguiente sistema de ecuaciones lineales con la función "jacobi".

$$\begin{aligned}
2x_1 + 25x_2 + x_3 - x_4 + 2x_5 - 5x_6 &= 24\\
30x_1 + 1x_2 + 2x_3 + 6x_4 + 5x_5 - 3x_6 &= 12\\
4x_1 + 3x_2 + 2x_3 + 3x_4 + 40x_5 - 4x_6 &= 23\\
x_1 + 2x_2 + 33x_3 + 5x_4 - x_5 + 2x_6 &= 17\\
5x_1 + 6x_2 - 3x_3 - 4x_4 + 3x_5 + 36x_6 &= 24\\
3x_1 + 5x_2 + 7x_3 - 45x_4 + 4x_5 + x_6 &= 37
\end{aligned} \qquad (13.4)$$

13.2. MÉTODO DE GAUSS-SEIDEL

El método de Gauss – Seidel es muy similar al método de Jacobi, prácticamente la única diferencia es que los nuevos valores se calculan empleando siempre los últimos valores calculados y no sólo los asumidos (como ocurre en el método de *Jacobi*). Así la primera incógnita se calcula con los valores asumidos, pero la segunda se calcula con los valores asumidos y el valor calculado para la primera incógnita, la tercera se calcula con los valores asumidos y los valores calculados para la primera y segunda incógnita y así sucesivamente.

Para comprender mejor el método se resolverá manualmente el sistema de ecuaciones lineales (13.1).

Como de costumbre se comienza por introducir los datos:

```
>> a=[10,1,2,44;2,10,1,51;1,2,10,61]; y=zeros(1,3); x=zeros(1,3);  m=4;
```

Entonces se calculan los tres valores de x:

```
> x(1)=(a(1,m)-(a(1,2).*y(2)+a(1,3).*y(3)))/a(1,1)
ans = [ 4.4  0  0 ]
>> x(2)=(a(2,m)-(a(2,1).*x(1)+a(2,3).*y(3)))/a(2,2)
ans = [ 4.4  4.22  0 ]
>> x(3)=(a(3,m)-(a(3,1).*x(1)+a(3,2).*x(2)))/a(3,3)
ans = [ 4.4  4.22  4.816 ]
```

Como los valores asumidos (ceros) no son iguales a los calculados, se repite el proceso con los valores calculados como valores asumidos:

```
>> y=x(:);
>> x(1)=(a(1,m)-(a(1,2).*y(2)+a(1,3).*y(3)))/a(1,1)
ans = [ 3.0148  4.22  4.816 ]
>> x(2)=(a(2,m)-(a(2,1).*x(1)+a(2,3).*y(3)))/a(2,2)
ans = [ 3.0148  4.0154  4.816 ]
>> x(3)=(a(3,m)-(a(3,1).*x(1)+a(3,2).*x(2)))/a(3,3)
ans = [ 3.0148  4.0154  4.9954 ]
```

Se sigue de esta manera hasta que los valores sean iguales (en este caso en los 5 dígitos mostrados). Las iteraciones realizadas, sin mostrar valores intermedios, son:

```
>> y=x(:);
>> x(1)=(a(1,m)-(a(1,2).*y(2)+a(1,3).*y(3)))/a(1,1);
>> x(2)=(a(2,m)-(a(2,1).*x(1)+a(2,3).*y(3)))/a(2,2);
>> x(3)=(a(3,m)-(a(3,1).*x(1)+a(3,2).*x(2)))/a(3,3)
ans = [ 2.9994  4.0006  4.9999 ]
>> y=x(:);
>> x(1)=(a(1,m)-(a(1,2).*y(2)+a(1,3).*y(3)))/a(1,1);
>> x(2)=(a(2,m)-(a(2,1).*x(1)+a(2,3).*y(3)))/a(2,2);
>> x(3)=(a(3,m)-(a(3,1).*x(1)+a(3,2).*x(2)))/a(3,3)
ans = [ 3  4  5 ]
>> y=x(:);
>> x(1)=(a(1,m)-(a(1,2).*y(2)+a(1,3).*y(3)))/a(1,1);
>> x(2)=(a(2,m)-(a(2,1).*x(1)+a(2,3).*y(3)))/a(2,2);
>> x(3)=(a(3,m)-(a(3,1).*x(1)+a(3,2).*x(2)))/a(3,3)
ans = [ 3  4  5 ]
```

Como se hace evidente en el ejemplo, el método de Gauss – Seidel requiere menos iteraciones que el método de Jacobi, no obstante, el método de Jacobi puede ser programado en paralelo, por lo que en computadoras con dos o más núcleos, puede ser más rápido que el método de Gauss-Seidel.

13.2.1. Ejercicio

3. Encuentre, con 5 dígitos de precisión, las soluciones del siguiente sistema aplicando manualmente el método de Gauss-Seidel.

$$\begin{aligned} 13x_1 + 18x_2 + x_3 &= 23 \\ 21x_1 + 3x_2 + 2x_3 &= 45 \\ 2x_1 + 4x_2 + 19x_3 &= 32 \end{aligned} \qquad (13.5)$$

13.2.2. Algoritmo y código

La ecuación general para calcular los nuevos valores en el método de *Gauss – Seidel*, que puede ser deducida fácilmente del ejemplo manual, es:

$$x_i = \frac{a_{i,m} - \sum_{j=1}^{i-1} a_{i,j} \cdot x_j - \sum_{j=i+1}^{n} a_{i,j} \cdot y_j}{a_{i,i}} \quad \{i = 1 \rightarrow n \tag{13.6}$$

Donde "a" es la matriz aumentada, "n" el número de filas, "m" el de columnas, "y" los valores asumidos y "x" los valores calculados.

Puesto que en el método de Gauss-Seidel, siempre se emplean los últimos valores calculados, es posible trabajar directamente con el vector "x" (los nuevos valores calculados), conservando el vector "y" simplemente para efectos de comparación. Procediendo así la ecuación (13.6) se convierte en:

$$x_i = \frac{a_{i,m} - \sum_{j=1}^{n} a_{i,j} \cdot x_j \quad \{j \neq i}{a_{i,i}} \quad \{i = 1 \rightarrow n \tag{13.7}$$

El algoritmo que automatiza el cálculo de esta ecuación se presenta en la siguiente página y el código respectivo es:

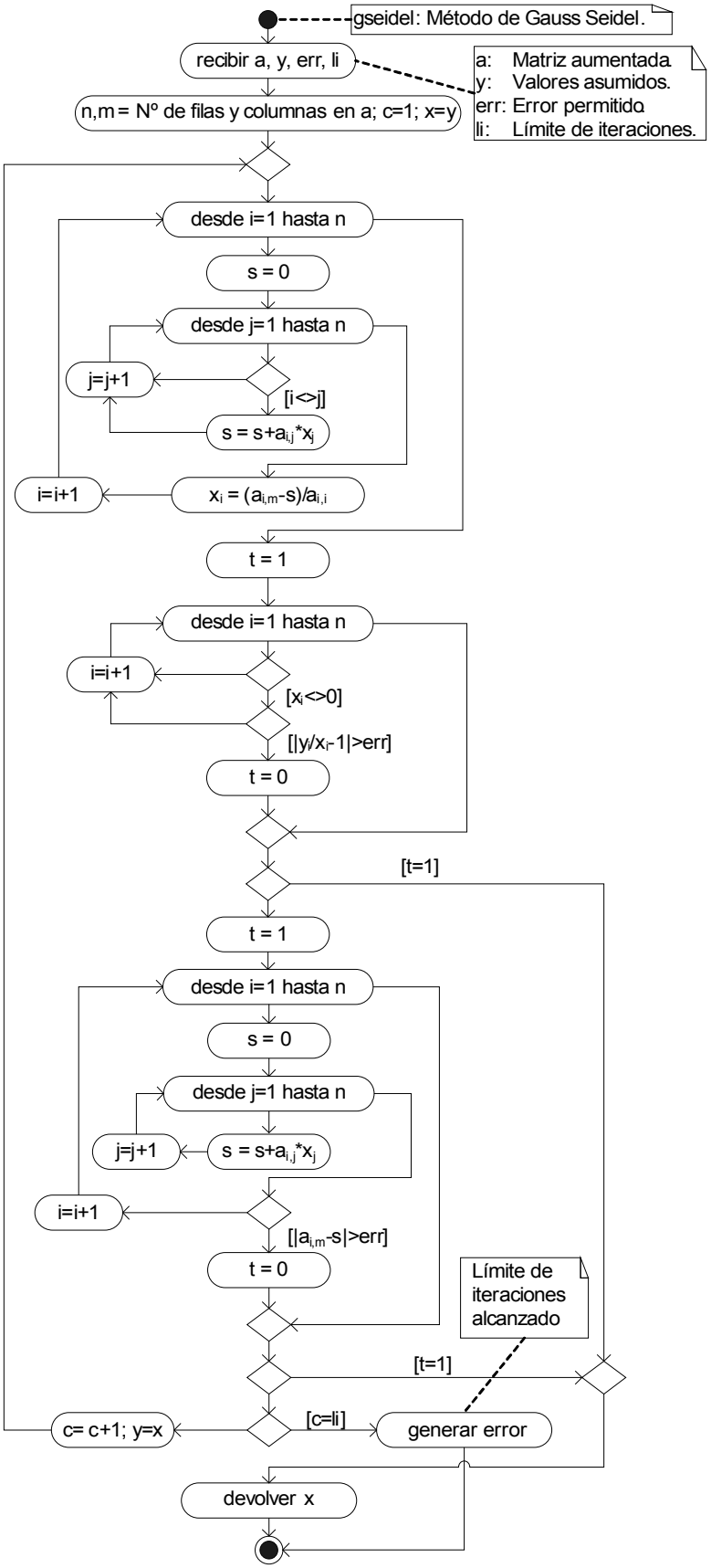
```
function x=gseidel(a,y,err,li)
  [n,m]=size(a); c=1; x=y(:);
  while 1
    for i=1:n
      s=0; for j=1:n if i ~= j s+=a(i,j).*x(j); end end
      x(i)=(a(i,m)-s)/a(i,i);
    end
    t=1;
    for i=1:n
      if x(i) ~= 0 if abs(y(i)/x(i)-1)>err t=0; break; end end
    end
    if t==1 return; end
    t=1;
    for i=1:n
      s=0; for j=1:n s+=a(i,j).*x(j); end;
      if abs(a(i,m)-s)>err t=0; break; end;
    end
    if t==1 return; end
    if c++ == li error("gseidel no converge\n"); end
    y=x(:);
  end
end
```

Haciendo correr el programa par el sistema resuelto manualmente, con un error permitido igual a 10^{-10}, se obtiene:

```
>> a=[10,1,2,44;2,10,1,51;1,2,10,61]; y=zeros(1,3);
>> format 10 10
>> gseidel(a,y,1e-10,100)
x = [ 3  4  5 ]
```

13.2.3. Ejercicio

4. Encuentre, con 11 dígitos de precisión, las soluciones de los siguientes sistema de ecuaciones con la función "gseidel".


gseidel: Método de Gauss Seidel.
recibir a, y, err, li
a: Matriz aumentada.
y: Valores asumidos.
err: Error permitido.
li: Límite de iteraciones.
n,m = Nº de filas y columnas en a; c=1; x=y
desde i=1 hasta n
s = 0
desde j=1 hasta n
j=j+1
[i<>j]
s = s+a_i,j*x_j
i=i+1
x_i = (a_i,m-s)/a_i,i
t = 1
desde i=1 hasta n
i=i+1
[x_i<>0]
[|y_i/x_i-1|>err]
t = 0
[t=1]
t = 1
desde i=1 hasta n
s = 0
desde j=1 hasta n
j=j+1
s = s+a_i,j*x_j
i=i+1
[|a_i,m-s|>err]
t = 0
Límite de iteraciones alcanzado
[t=1]
[c=li]
generar error
c= c+1; y=x
devolver x

$$
\begin{array}{rcrcrcrcr}
19x_1 & + & 2x_2 & + & x_3 & & & = & 0 \\
2x_1 & + & 15x_2 & + & 3x_3 & + & x_4 & = & 0 \\
x_1 & + & 3x_2 & + & 20x_3 & + & 2x_4 & = & 1 \\
 & & x_2 & + & 2x_3 & + & 34x_4 & = & 0 \\
 & & & & & & & & \\
19y_1 & + & 2y_2 & + & y_3 & & & = & -2 \\
2y_1 & + & 15y_2 & + & 3y_3 & + & y_4 & = & 2 \\
y_1 & + & 3y_2 & + & 20y_3 & + & 2y_4 & = & 3 \\
 & & y_2 & + & 2y_3 & + & 34y_4 & = & 4 \\
 & & & & & & & & \\
19z_1 & + & 2z_2 & + & z_3 & & & = & 2 \\
2z_1 & + & 15z_2 & + & 3z_3 & + & z_4 & = & 2 \\
z_1 & + & 3z_2 & + & 20z_3 & + & 2z_4 & = & 1 \\
 & & z_2 & + & 2z_3 & + & 34z_4 & = & 0
\end{array}
\qquad (13.8)
$$

13.3. MÉTODO DE GAUSS PARA 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 forma general:

$$
\begin{array}{rcrcrcrcrcr}
a_{1,2}x_1 & + & a_{1,3}x_2 & + & 0x_3 & + & 0x_4 & +\cdots+ & 0x_{n-1} & + & 0x_n & = & a_{1,m} \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & + & 0x_4 & +\cdots+ & 0x_{n-1} & + & 0x_n & = & a_{2,m} \\
0x_1 & + & a_{3,2}x_2 & + & a_{3,3}x_3 & + & a_{3,4}x_4 & +\cdots+ & 0x_{n-1} & + & 0x_n & = & a_{3,m} \\
0x_1 & + & 0x_2 & + & a_{4,3}x_3 & + & a_{4,4}x_4 & +\cdots+ & 0x_{n-1} & + & 0x_n & = & a_{4,m} \\
0x_1 & + & 0x_2 & + & 0x_3 & + & a_{5,4}x_4 & +\cdots+ & 0x_{n-1} & + & 0x_n & = & a_{5,m} \\
\cdots & + & \cdots & + & \cdots & + & \cdots & +\cdots+ & \cdots & + & \cdots & = & \cdots \\
0x_1 & + & 0x_2 & + & 0x_3 & + & 0x_4 & +\cdots+ & a_{n,n-1}x_{n-1} & + & a_{n,n}x_n & = & a_{n,m}
\end{array}
\qquad (13.9)
$$

Donde *"n"* es el número de filas y *"m"* es el número de columnas.

Si sólo se guardan los valores diferentes de cero, el sistema tridiagonal puede ser reescrito como:

$$
\begin{array}{rcrcrcr}
0 & + & a_{1,2}x_2 & + & a_{1,3}x_3 & = & a_{1,4} \\
a_{2,1}x_1 & + & a_{2,2}x_2 & + & a_{2,3}x_3 & = & a_{2,4} \\
a_{3,1}x_2 & + & a_{3,2}x_3 & + & a_{3,3}x_4 & = & a_{3,4} \\
a_{4,1}x_3 & + & a_{4,2}x_4 & + & a_{4,3}x_5 & = & a_{4,4} \\
a_{5,1}x_4 & + & a_{5,2}x_5 & + & a_{5,3}x_6 & = & a_{5,4} \\
\cdots & + & \cdots & + & \cdots & = & \cdots \\
a_{n,1}x_{n-1} & + & a_{n,2}x_n & + & 0 & = & a_{5,4}
\end{array}
\qquad (13.10)
$$

Con lo que se consigue un gran ahorro de memoria, pues en lugar de reservar espacio para n*m elementos sólo se requiere espacio para n*4 elementos.

La mayoría de los métodos estudiados en este y los anteriores temas pueden ser adaptados de manera que trabajen con los elementos de la ecuación (13.12) en lugar de los elementos de la ecuación (13.11).

Para deducir las ecuaciones generales para el método de *Gauss*, se considera un sistema tridiagonal de 5 ecuaciones con 5 incógnitas. Con el fin de facilitar la deducción de las ecuaciones se escriben también los ceros:

$$\begin{aligned}
a_{1,2}x_1 &+ a_{1,3}x_2 &+ 0x_3 &+ 0x_4 &+ 0x_5 &= a_{1,4}\\
a_{2,1}x_1 &+ a_{2,2}x_2 &+ a_{2,3}x_3 &+ 0x_4 &+ 0x_5 &= a_{2,4}\\
0x_1 &+ a_{3,1}x_2 &+ a_{3,2}x_3 &+ a_{3,3}x_4 &+ 0x_5 &= a_{3,4}\\
0x_1 &+ 0x_2 &+ a_{4,1}x_3 &+ a_{4,2}x_4 &+ a_{4,3}x_5 &= a_{4,4}\\
0x_1 &+ 0x_2 &+ 0x_3 &+ a_{5,1}x_4 &+ a_{5,2}x_5 &= a_{5,4}
\end{aligned} \qquad (13.11)$$

Que al aplicar el método de Gauss se convierte en:

$$\begin{aligned}
a_{1,2}x_1 &+ a_{1,3}x_2 &+ 0x_3 &+ 0x_4 &+ 0x_5 &= a_{1,4}\\
0x_1 &+ a_{2,2}x_2 &+ a_{2,3}x_3 &+ 0x_4 &+ 0x_5 &= a_{2,4}\\
0x_1 &+ 0x_2 &+ a_{3,2}x_3 &+ a_{3,3}x_4 &+ 0x_5 &= a_{3,4}\\
0x_1 &+ 0x_2 &+ 0x_3 &+ a_{4,2}x_4 &+ a_{4,3}x_5 &= a_{4,4}\\
0x_1 &+ 0x_2 &+ 0x_3 &+ 0x_4 &+ a_{5,2}x_5 &= a_{5,4}
\end{aligned} \qquad (13.12)$$

Por lo tanto sólo es necesario convertir en ceros los elementos que se encuentran debajo de la diagonal principal. Así para convertir en cero el elemento $a_{2,1}$, se resta a la segunda fila la primera multiplicada por $a_{2,1}/a_{1,2}$. Como no es necesario guardar ni calcular los ceros, se puede aprovechar el elemento $a_{2,1}$ para guardar 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):

$$\begin{aligned}
a_{2,1} &= \frac{a_{2,1}}{a_{1,2}}\\
a_{2,2} &= a_{2,2} - a_{2,1}\cdot a_{1,3}\\
a_{2,4} &= a_{2,4} - a_{2,1}\cdot a_{1,4}
\end{aligned}$$

De la misma forma, para convertir el elemento $a_{3,1}$ en cero se resta 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 $a_{3,1}$:

$$\begin{aligned}
a_{3,1} &= \frac{a_{3,1}}{a_{2,2}}\\
a_{3,2} &= a_{3,2} - a_{3,1}\cdot a_{2,3}\\
a_{3,4} &= a_{3,4} - a_{3,1}\cdot a_{2,4}
\end{aligned}$$

De manera similar, las operaciones para la cuarta y quinta fila son:

$$\begin{aligned}
a_{4,1} &= \frac{a_{4,1}}{a_{3,2}}\\
a_{4,2} &= a_{4,2} - a_{4,1}\cdot a_{3,3}\\
a_{4,4} &= a_{4,4} - a_{4,1}\cdot a_{3,4}
\end{aligned}$$

$$\begin{aligned}
a_{5,1} &= \frac{a_{5,1}}{a_{4,2}}\\
a_{5,2} &= a_{5,2} - a_{5,1}\cdot a_{4,3}\\
a_{5,4} &= a_{5,4} - a_{5,1}\cdot a_{4,4}
\end{aligned}$$

Analizando estas ecuaciones se pueden deducir la forma general de las

ecuaciones para reducir a cero los elementos que se encuentran debajo de la diagonal principal:

$$\left.\begin{aligned} a_{i,1} &= \frac{a_{i,1}}{a_{i-1,2}} \\ a_{i,2} &= a_{i,2} - a_{i,1} \cdot a_{i-1,3} \\ a_{i,4} &= a_{i,4} - a_{i,1} \cdot a_{i-1,4} \end{aligned}\right\} \; i = 2 \rightarrow n \qquad (13.13)$$

Una vez que el sistema ha quedado reducido a la forma (13.16), los resultados se calculan por sustitución inversa y pueden ser almacenados en la última columna (la columna 4) del sistema tridiagonal. Así el valor de la última variable *(x_5)* se calcula con:

$$x_5 = a_{5,4} = \frac{a_{5,4}}{a_{5,2}}$$

De manera similar se calculan los valores de las otras variables:

$$x_4 = a_{4,4} = \frac{a_{4,4} - a_{4,3} \cdot x_5}{a_{4,2}} = \frac{a_{4,4} - a_{4,3} \cdot a_{5,4}}{a_{4,2}}$$

$$x_3 = a_{3,4} = \frac{a_{3,4} - a_{3,3} \cdot x_4}{a_{3,2}} = \frac{a_{3,4} - a_{3,3} \cdot a_{4,4}}{a_{3,2}}$$

$$x_2 = a_{2,4} = \frac{a_{2,4} - a_{2,3} \cdot x_3}{a_{2,2}} = \frac{a_{2,4} - a_{2,3} \cdot a_{3,4}}{a_{2,2}}$$

$$x_1 = a_{1,4} = \frac{a_{1,4} - a_{1,3} \cdot x_2}{a_{1,2}} = \frac{a_{1,4} - a_{1,3} \cdot a_{2,4}}{a_{1,2}}$$

De donde se deducen las ecuaciones generales para el cálculo de las soluciones por sustitución inversa:

$$\left.\begin{aligned} x_n &= a_{n,4} = \frac{a_{n,4}}{a_{n,2}} \\ x_i &= a_{i,4} = \frac{a_{i,4} - a_{i,3} \cdot x_{i+1}}{a_{i,2}} = \frac{a_{i,4} - a_{i,3} \cdot a_{i+1,4}}{a_{i,2}} \end{aligned}\right\} \; i = n-1 \rightarrow 1 \qquad (13.14)$$

Para comprender mejor como se aplican las ecuaciones (13.17) y (13.14) se resolverá manualmente el siguiente sistema de ecuaciones:

$$\begin{aligned} 2x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 &= 6 \\ x_1 + 4x_2 + x_3 + 0x_4 + 0x_5 &= 36 \\ 0x_1 + x_2 + 4x_3 + x_4 + 0x_5 &= 72 \\ 0x_1 + 0x_2 + x_3 + 4x_4 + x_5 &= 108 \\ 0x_1 + 0x_2 + 0x_3 + x_4 + 2x_5 &= 66 \end{aligned} \qquad (13.15)$$

Primero se crea la matriz tridiagonal y se determina el número de filas:

```
>> a=[0,2,1,6;1,4,1,36;1,4,1,72;1,4,1,108;1,2,0,66]; n=length(a);
```

Luego, se aplican las ecuaciones (13.17) para reducir el sistema:

```
>> i=2; 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); a(i,:)
ans = [ 0.5  3.5  1  33 ]
>> i=3; 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); a(i,:)
```

```
ans = [ 0.28571  3.7143  1  62.571 ]
>> i=4; 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); a(i,:)
ans = [ 0.26923  3.7308  1  91.154 ]
>> i=5; 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); a(i,:)
ans = [ 0.26804  1.732  0  41.567 ]
```

Una vez reducido el sistema se aplican las ecuaciones (13.14) para calcular las soluciones por sustitución inversa:

```
>> a(n,4)=a(n,4)/a(n,2);a(n,4)
ans = 24
>> i=n-1; a(i,4)=(a(i,4)-a(i,3).*a(i+1,4))/a(i,2); a(i,4)
ans = 18
>> i--; a(i,4)=(a(i,4)-a(i,3).*a(i+1,4))/a(i,2); a(i,4)
ans = 12
>> i--; a(i,4)=(a(i,4)-a(i,3).*a(i+1,4))/a(i,2); a(i,4)
ans = 6
>> i--; a(i,4)=(a(i,4)-a(i,3).*a(i+1,4))/a(i,2); a(i,4)
ans = 0
```

Por lo tanto, las soluciones del sistema son: $x_1=0$; $x_2=6$; $x_3=12$; $x_4=18$; $x_5=24$.

13.3.1. Ejercicio

5. Encuentre las soluciones del siguiente sistema de ecuaciones aplicando manualmente el método de Gauss para sistemas tridiagonales.

$$\begin{aligned} 4x_1 + 2x_2 + 0x_3 + 0x_4 + 0x_5 &= 8 \\ 3x_1 + 8x_2 + x_3 + 0x_4 + 0x_5 &= 14 \\ 0x_1 + 2x_2 + 12x_3 + 3x_4 + 0x_5 &= 21 \\ 0x_1 + 0x_2 + x_3 + 9x_4 + 2x_5 &= 34 \\ 0x_1 + 0x_2 + 0x_3 + 3x_4 + 14x_5 &= 37 \end{aligned} \qquad (13.16)$$

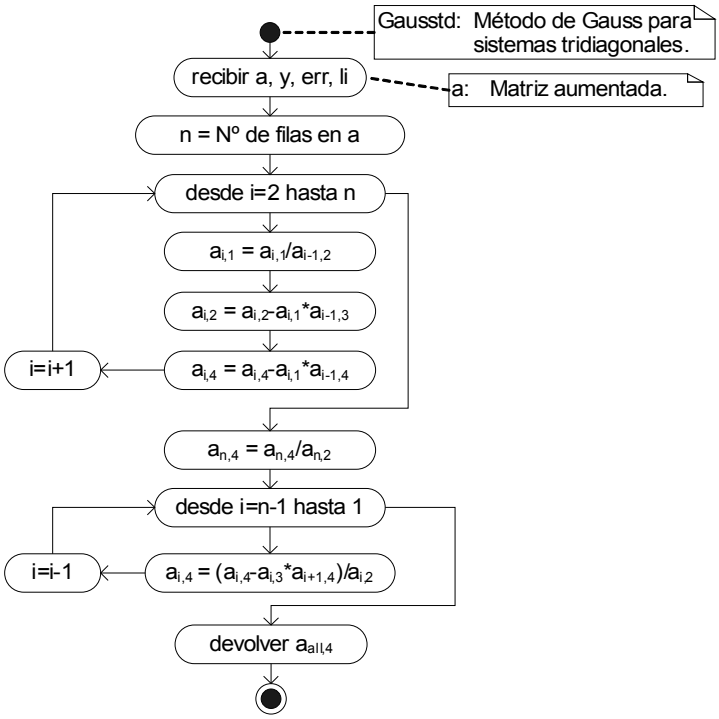
13.3.2. Algoritmo y código

El algoritmo que automatiza la aplicación del método se presenta en la siguiente página, siendo el código respectivo:

```
function x=gausstd(b)
  a=b(:,:); n=length(a);
  for i=2:n
    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);
  end
  a(n,4)=a(n,4)/a(n,2);
  for i=n-1:-1:1
    a(i,4)=(a(i,4)-a(i,3).*a(i+1,4))/a(i,2);
  end
  x=a(:,4);
end
```

Haciendo correr el programa con la matriz del ejemplo manual se obtiene:

```
>> a=[0,2,1,6;1,4,1,36;1,4,1,72;1,4,1,108;1,2,0,66];
>> gausstd(a)
```




```
x =
   0
   6
   12
   18
   24
```

Que son los resultados obtenidos en dicho ejemplo.

13.3.3. Ejercicio

6. Encuentre las soluciones del siguiente sistema con la función "gausstd".

$$\begin{aligned}
0.6x_1 + 0.1x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 &= -1.23 \\
0.1x_1 + 0.4x_2 + 0.1x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 &= 3.96 \\
0x_1 + 0.1x_2 + 0.4x_3 + 0.1x_4 + 0x_5 + 0x_6 + 0x_7 &= 9.60 \\
0x_1 + 0x_2 + 0.1x_3 + 0.4x_4 + 0.1x_5 + 0x_6 + 0x_7 &= 11.52 \\
0x_1 + 0x_2 + 0x_3 + 0.1x_4 + 0.4x_5 + 0.1x_6 + 0x_7 &= -21.42 \\
0x_1 + 0x_2 + 0x_3 + 0x_4 + 0.1x_5 + 0.4x_6 + 0.1x_7 &= -4.50 \\
0x_1 + 0x_2 + 0x_3 + 0x_4 + 0x_5 + 0.1x_6 + 0.6x_7 &= 0.60
\end{aligned} \qquad (13.17)$$

13.4. MÉTODO DE GAUSS SEIDEL PARA SISTEMAS TRIDIAGONALES

Generalmente los sistemas tridiagonales tienen además una diagonal dominante. Por ello, los métodos iterativos como *Jacobi* y *Gauss – Seidel* resultan particularmente adecuados para resolver estos sistemas (como ya se dijo, cuando la diagonal es dominante la convergencia está asegurada).

Al igual que en el método de *Gauss* se empleará el sistema tridiagonal de 5 ecuaciones (ecuación 13.11) para deducir las ecuaciones generales del método.

Si "x" es el vector con los valores iniciales asumidos, entonces los nue-

vos valores se calculan con:

$$x_1 = \frac{a_{1,4} - a_{1,3} \cdot x_2}{a_{1,2}}$$
$$x_2 = \frac{a_{2,4} - a_{2,1} \cdot x_1 - a_{2,3} \cdot x_3}{a_{2,2}}$$
$$x_3 = \frac{a_{3,4} - a_{3,1} \cdot x_2 - a_{3,3} \cdot x_4}{a_{3,2}}$$
$$x_4 = \frac{a_{4,4} - a_{4,1} \cdot x_3 - a_{4,3} \cdot x_5}{a_{4,2}}$$
$$x_5 = \frac{a_{5,4} - a_{5,1} \cdot x_4}{a_{5,2}}$$

De donde se deducen las ecuaciones generales del método:

$$x_1 = \frac{a_{1,4} - a_{1,3} \cdot x_2}{a_{1,2}}$$
$$x_i = \frac{a_{i,4} - a_{i,1} \cdot x_{i-1} - a_{i,3} \cdot x_{i+1}}{a_{i,2}} \quad \left\{ i = 2 \rightarrow n-1 \right. \qquad (13.18)$$
$$x_n = \frac{a_{n,4} - a_{n,1} \cdot x_{n-1}}{a_{n,2}}$$

Para comprender mejor la aplicación de estas ecuaciones se resolverá manualmente el sistema de ecuaciones (13.15).

Primero se crea la matriz tridiagonal y se inicializan variables (siendo los valores sumidos ceros):

```
>> a=[0,2,1,6;1,4,1,36;1,4,1,72;1,4,1,108;1,2,0,66]; n=length(a);
   x=zeros(1,n);
```

Entonces se calculan los nuevos valores de "x" aplicando las ecuaciones (13.18):

```
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ 3  8.25  15.938  23.016  21.492 ]
```

Y se vuelven a aplicar las ecuaciones (13.18) hasta que los dos últimos valores sean aproximadamente iguales (en los 3 dígitos de precisión con los que se muestran los resultados):

```
>> format 10 3
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -1.13  5.3  10.9  18.9  23.6 ]
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ 0.352  6.18  11.7  18.2  23.9 ]
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -9.08E-2  6.09  11.9  18  24 ]
```

```
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -4.5E-2  6.03  12  18  24 ]
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -1.41E-2  6.01  12  18  24 ]
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -3.86E-3  6  12  18  24 ]
>> x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2); for i=2:n-1 x(i)=(a(i,4)-
   a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2); end x(n)=(a(n,4)-a(n,1)
   .*x(n-1))/a(n,2); x
ans = [ -1.01E-3  6  12  18  24 ]
```

Los últimos valores calculados son casi iguales (con excepción del primer término que es casi cero), por lo que el proceso puede concluir en este punto, siendo las soluciones $x_1 \approx 0$; $x_2=6$; $x_3=12$, $x_4=18$; $x_5=24$.

13.4.1. Ejercicio

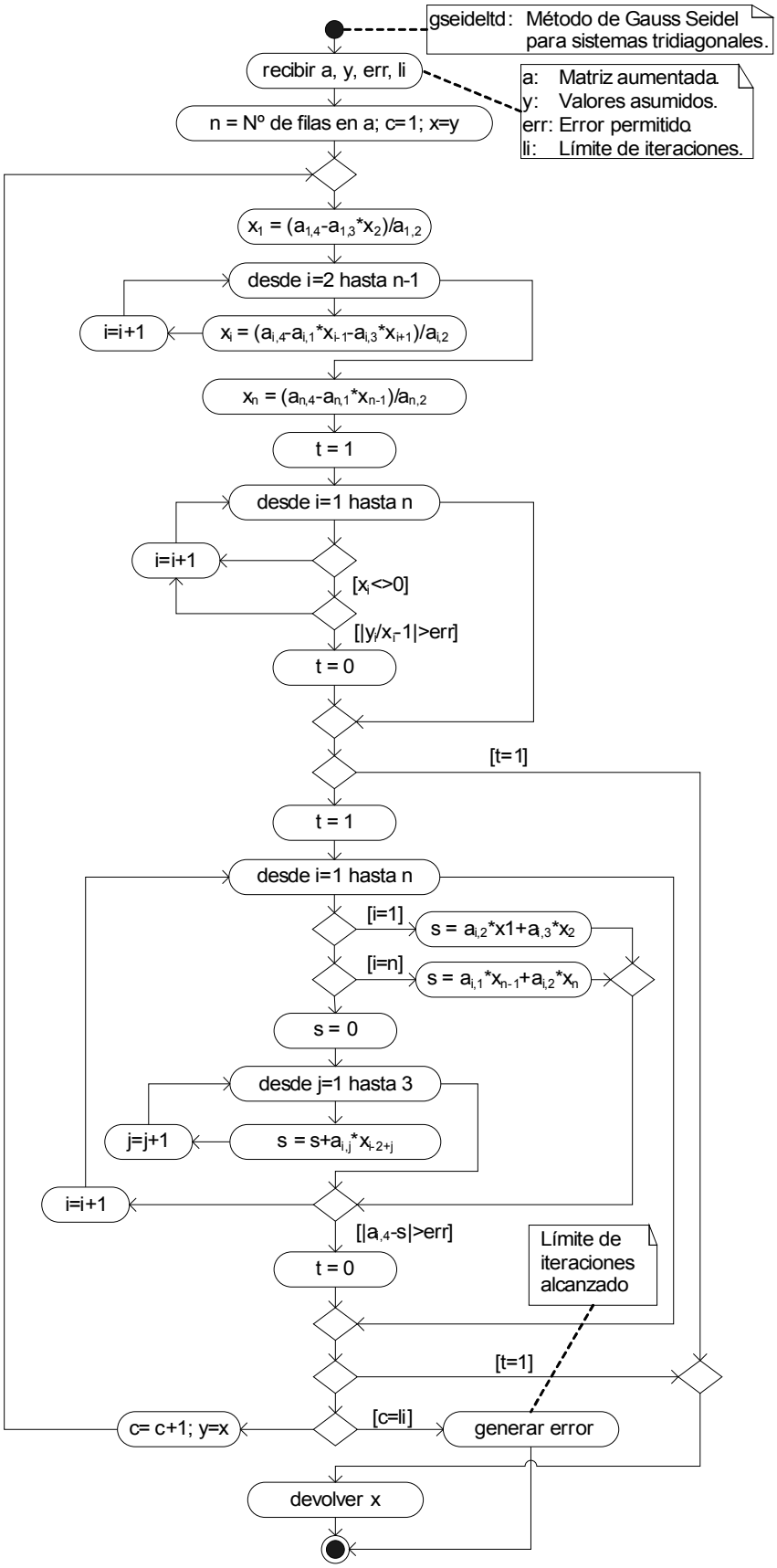
7. Encuentre las soluciones del sistema de ecuaciones (13.16) aplicando manualmente el método de Gauss-Seidel para sistemas tridiagonales.

13.4.2. Algoritmo y código

El algoritmo del método de Gauss Seidel para sistemas tridiagonales es similar al algoritmo normal del método, excepto que ahora se emplean las ecuaciones (13.18) y para comprobar la igualdad se suman sólo las tres columnas del sistema tridiagonal.

El algoritmo que automatiza el proceso se presenta en la siguiente página y el código respectivo es:

```
function x=gseideltd(a,y,err,li)
  n=length(a); c=1; x=y(:);
  while 1
    x(1)=(a(1,4)-a(1,3).*x(2))/a(1,2);
    for i=2:n-1
     x(i)=(a(i,4)-a(i,1).*x(i-1)-a(i,3).*x(i+1))/a(i,2);
    end
    x(n)=(a(n,4)-a(n,1).*x(n-1))/a(n,2);
    t=1;
    for i=1:n
      if x(i) ~= 0 if abs(y(i)/x(i)-1)>err t=0; break; end end
    end
    if t==1 return; end
    t=1;
    for i=1:n
      if i==1 s=a(i,2).*x(1)+a(i,3).*x(2);
      else if i==n s=a(i,1).*x(n-1)+a(i,2).*x(n);
      else s=0; for j=1:3 s+=a(i,j).*x(i-2+j); end; end; end;
      if abs(a(i,4)-s)>err t=0; break; end;
    end
    if t==1 return; end
    if c++ == li error("gseideltd no converge\n"); end
    y=x(:);
```


gseideltd: Método de Gauss Seidel para sistemas tridiagonales.
recibir a, y, err, li
a: Matriz aumentada.
y: Valores asumidos.
err: Error permitido.
li: Límite de iteraciones.
n = Nº de filas en a; c=1; x=y
x1 = (a1,4-a1,3*x2)/a1,2
desde i=2 hasta n-1
i=i+1
xi = (ai,4-ai,1*xi-1-ai,3*xi+1)/ai,2
xn = (an,4-an,1*xn-1)/an,2
t = 1
desde i=1 hasta n
i=i+1
[xi<>0]
[|yi/xi-1|>err]
t = 0
[t=1]
t = 1
desde i=1 hasta n
[i=1]
s = ai,2*x1+ai,3*x2
[i=n]
s = ai,1*xn-1+ai,2*xn
s = 0
desde j=1 hasta 3
j=j+1
s = s+ai,j*xi-2+j
i=i+1
[|a,4-s|>err]
t = 0
Límite de iteraciones alcanzado
[t=1]
[c=li]
c= c+1; y=x
generar error
devolver x

end

```
end
```

Haciendo correr el programa para los datos del ejemplo manual, con un error permitido igual a 10^{-10} y un límite de 100 iteraciones, se obtiene:

```
   >> a=[0,2,1,6;1,4,1,36;1,4,1,72;1,4,1,108;1,2,0,66]; x=zeros(1,5);
>> gseideltd(a,x,1e-10,100)
x = [ -6.2754E-11   6   12   18   24 ]
```

Que esencialmente son los mismos resultados del ejemplo manual.

13.4.3. Ejercicio

8. Encuentre las soluciones del siguiente sistema de ecuaciones empleando la función "gseideltd" con 8 dígitos de precisión.

$$
\begin{aligned}
-2.0304x_1 + x_2 + 0x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 &= -1.952\\
x_1 - 2.0288x_2 + x_3 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 &= 0.056\\
0x_1 + x_2 - 2.0272x_3 + x_4 + 0x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 &= 0.064\\
0x_1 + 0x_2 + x_3 - 2.0256x_4 + x_5 + 0x_6 + 0x_7 + 0x_8 + 0x_9 &= 0.072\\
0x_1 + 0x_2 + 0x_2 + x_4 - 2.0240x_5 + x_6 + 0x_7 + 0x_8 + 0x_9 &= 0.080\\
0x_1 + 0x_2 + 0x_2 + 0x_4 + x_5 - 2.0224x_6 + x_7 + 0x_8 + 0x_9 &= 0.088\\
0x_1 + 0x_2 + 0x_2 + 0x_4 + 0x_5 + x_6 - 2.0208x_7 + x_8 + 0x_9 &= 0.096\\
0x_1 + 0x_2 + 0x_2 + 0x_4 + 0x_5 + 0x_6 + x_7 - 2.0192x_8 + x_9 &= 0.104\\
0x_1 + 0x_2 + 0x_2 + 0x_4 + 0x_5 + 0x_6 + 0x_7 + x_8 - 2.0176x_9 &= 1.112
\end{aligned}
\qquad (13.19)
$$

13.5. DETERMINANTE DE UNA MATRIZ

Tradicionalmente para el cálculo del determinante se emplea el método de Cramer, sin embargo, dicho método sólo resulta de utilidad práctica para un sistema de 2 ecuaciones lineales con 2 incógnitas. Para sistemas con un mayor número de ecuaciones el método de Cramer es sumamente ineficiente, así por ejemplo para resolver un sistema de 10 ecuaciones lineales con 10 incógnitas se requieren aproximadamente 70 millones de operaciones, un número exorbitante, inclusive para las computadoras actuales.

Por ello el cálculo del determinante se realiza en la práctica recurriendo a uno de los métodos de eliminación estudiados, siendo un tanto más eficiente el método de Gauss.

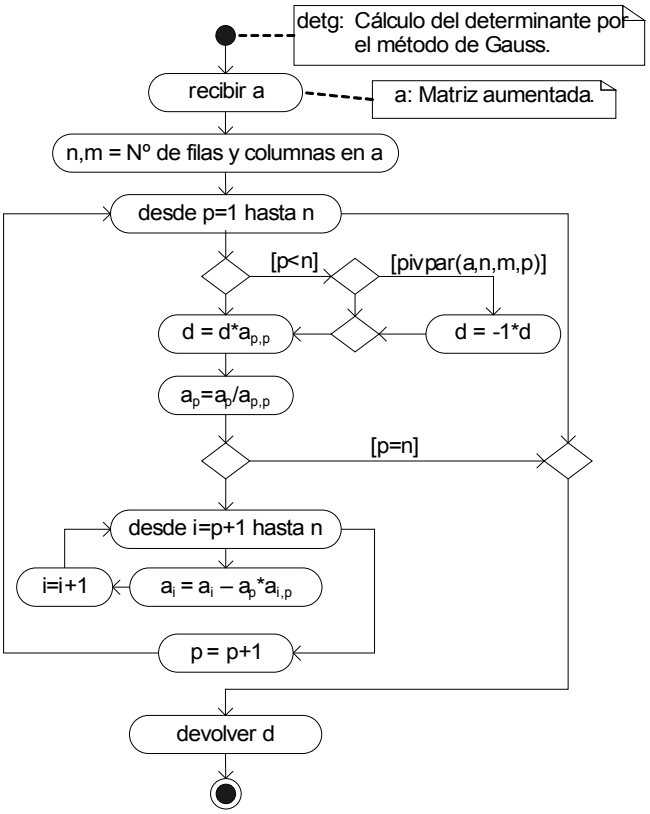
Para calcular el determinante con uno de estos métodos simplemente se multiplican los elementos de la diagonal principal (antes de que sean convertidos en unos).

El único cuidado adicional que se requiere en este proceso es el de cambiar el signo del determinante con cada intercambio de filas (o lo que es lo mismo, cambiar el signo del determinante si el número de intercambios es impar).

Jasymca cuenta con una función para el cálculo de determinantes "det", pero para demostrar cómo se efectúa dicho cálculo, se elaborará un programa que calcule el determinante por el método de Gauss.

El algoritmo de dicho programa se presenta en la siguiente página y el código respectivo es el siguiente:

```
function d=detg(b)
  a=0; a=b(:,:); [n,m]=size(a); d=1;
```




```
  for p=1:n
    if p<n if pivpar(a,n,m,p) d*=-1; end end
    d*=a(p,p);
    a(p,:)=a(p,:)/a(p,p);
    if p==n break; end
    for i=p+1:n a(i,:)=a(i,:)-a(p,:).*a(i,p); end
  end
end
```

Para probar el programa elaborado se encontrará el determinante de la matriz de coeficientes del siguiente sistema de ecuaciones:

$$\begin{aligned} 2x_1 + 8x_2 + 2x_3 &= 14 \\ x_1 + 6x_2 - x_3 &= 13 \\ 2x_1 - x_2 + 2x_3 &= 5 \end{aligned} \qquad (13.20)$$

```
>> a=[2,8,2;1,6,-1;2,-1,2];
>> detg(a)
d = -36
```

Resultado que puede ser corroborado con "det":

```
>> det(a)
ans = -36
```

Como se sabe, el determinante puede ser empleado para encontrar las soluciones de un sistema de ecuaciones (aunque probablemente sea el método menos eficiente de todos).

Para encontrar una de las soluciones se reemplaza, en la matriz de coeficientes, la columna de las constantes por la columna correspondiente a la solución que se quiere encontrar (la columna uno para la solución uno, la columna dos para la solución dos, etc.). Luego se divide el determinante de

de esta matriz (la matriz modificada) entre el determinante de la matriz de los coeficientes.

Por ejemplo para encontrar la primera solución del sistema de ecuaciones (13.20) se escribe:

```
>> a=[2,8,2;1,6,-1;2,-1,2]; b=[14;15;5];
>> c=a(:,:); c(:,1)=b; detg(c)/detg(a)
ans = 6
```

De manera similar para encontrar la segunda y tercera solución se escribe:

```
>> c=a(:,:); c(:,2)=b; detg(c)/detg(a)
ans = 1
>> c=a(:,:); c(:,3)=b; detg(c)/detg(a)
ans = -3
```

En lugar de calcular tres veces el determinante de "a", es más eficiente guardar el determinante en una variable y emplear luego dicha variable:

```
>> a=[2,8,2;1,6,-1;2,-1,2]; b=[14;15;5]; d=det(a);
>> c=a(:,:); c(:,1)=b; detg(c)/d
ans = 6
>> c=a(:,:); c(:,2)=b; detg(c)/d
ans = 1
>> c=a(:,:); c(:,3)=b; detg(c)/d
ans = -3
```

13.5.1. Ejercicio

9. Encuentre las soluciones del siguiente sistema de ecuaciones mediante el método del determinante.

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= -1 \\ 4x_1 + 5x_2 + 6x_3 + 7x_4 &= 0 \\ 6x_1 + 10x_2 + 15x_3 + 21x_4 &= 10 \\ 12x_1 + 30x_2 + 60x_3 + 105x_4 &= 10 \end{aligned} \qquad (13.21)$$

14. SISTEMAS DE ECUACIONES NO LINEALES

Para la resolución de sistemas de ecuaciones no lineales (con excepción de los polinomios) no existen métodos analíticos. Por lo tanto dichos sistemas sólo pueden ser resueltos con la ayuda de los métodos numéricos.

Como sucede con la mayoría de los métodos numéricos, en estos métodos se requieren valores iniciales asumidos (valores de prueba) para comenzar el proceso y dado que un sistema de ecuaciones no lineales tiene un conjunto de soluciones (y no sólo una) es importante que dichos valores sean lo más cercanos posibles a las soluciones finales.

Cuando el sistema de ecuaciones no lineales puede ser expresado en función de una sola variable, la mejor forma de resolver el problema es aplicar los métodos estudiados en los temas 2 y 3, tal como se ha hecho en algunos de los ejemplos resueltos en dichos temas.

14.1. MÉTODO DE SIMPLEX

En realidad, el método Simplex es un método de optimización, es decir un método que permite encontrar un máximo o un mínimo y en consecuencia puede ser empleado para resolver una gran variedad de problemas, tales como optimizar ganancias (es decir maximizar los beneficios), optimizar gastos (esto es minimizar los costos), ajustar datos (minimizar residuos), optimizar el aprovechamiento de recursos (maximizar el tiempo de uso), etc.

En este sentido puede ser empleado también para resolver un sistema de ecuaciones no lineales. Esto es posible porque los sistemas de ecuaciones no lineales pueden ser expresados como un problema de optimización, siendo el problema el minimizar las diferencias entre los lados izquierdo y derecho de las ecuaciones.

Supongamos que la función a resolver es:

$$f(x_1, x_2, x_3, \cdots, x_n) = \alpha \tag{14.1}$$

Como en toda función, la solución se encuentra cuando al reemplazar los valores de las variables independientes (x_1 a x_n), la igualdad se cumple. Una forma conveniente de conseguirlo es colocando la función en forma de un problema de optimización (más propiamente de minimización):

$$g(x_1, x_2, x_3, \cdots, x_n) = f(x_1, x_2, x_3, \cdots, x_n) - \alpha = 0 \tag{14.2}$$

Sin embargo, esta forma tiene un inconveniente pues la diferencia puede ser positiva o negativa y si es negativa, es menor a cero, entonces se podría concluir erróneamente que se ha encontrado el mínimo.

Este problema puede ser resuelto si la diferencia se eleva al cuadrado:

$$\left[g(x_1, x_2, x_3, \cdots, x_n)\right]^2 = \left[f(x_1, x_2, x_3, \cdots, x_n) - \alpha\right]^2 = 0 \tag{14.3}$$

De esa manera nunca se tienen valores negativos. Cuando la función se coloca en esta forma, se conoce como la ***función de error***, y es la forma en que deben ser colocadas las expresiones para que puedan ser resueltas por el método Simplex:

$$e(x_1, x_2, x_3, \cdots, x_n) = \left[f(x_1, x_2, x_3, \cdots, x_n) - \alpha\right]^2 = 0 \tag{14.4}$$

Es importante recordar que el objetivo de esta transformación es simplemente la de asegurar que la función siempre devuelva valores positivos por lo que también podría emplearse el valor absoluto de la función.

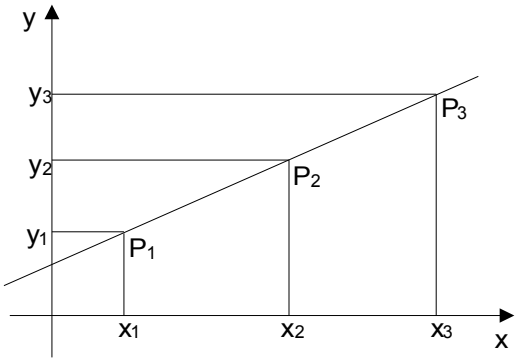
Ahora bien, partiendo de un conjunto de valores iniciales asumidos (plan inicial Simplex), el método encuentra nuevos valores proyectando los valores asumidos en dirección al mínimo. Dicha proyección se la realiza mediante interpolación o extrapolación lineal de puntos "n-dimensionales".

14.1.1. Proyección lineal n-dimensional

En un sistema de ecuaciones no lineales, cada valor asumido constituye un vector, o, lo que es lo mismo un punto "n-dimensional", que tiene tantas dimensiones como variables tiene el sistema.

La deducción de las ecuaciones que permiten la proyección de un punto a través de otros dos, se la hará empleando puntos en el plano (vectores con dos dimensiones), sin embargo, las ecuaciones resultantes tendrán aplicación general.

Tomando en cuenta la siguiente figura:



Donde se conocen los puntos P_1 y P_2, siendo el objetivo calcular el punto P_3. Ello es posible si se conocen las distancias que existen entre los puntos P_1, P_2 y P_1, P_3, es decir si se conoce la relación:

$$k=\frac{\overline{P_1P_3}}{\overline{P_1P_2}} \tag{14.5}$$

Entonces los elementos del punto P_3 (x_3,y_3), se calculan con:

$$\begin{aligned} k&=\frac{x_3-x_1}{x_2-x_1}\\ x_3&=x_1+k\cdot(x_2-x_1)\\ x_3&=k\cdot x_2+(1-k)\cdot x_1 \end{aligned} \tag{14.6}$$

$$\begin{aligned} k&=\frac{y_3-y_1}{y_2-y_1}\\ y_3&=k\cdot y_2+(1-k)\cdot y_1 \end{aligned} \tag{14.7}$$

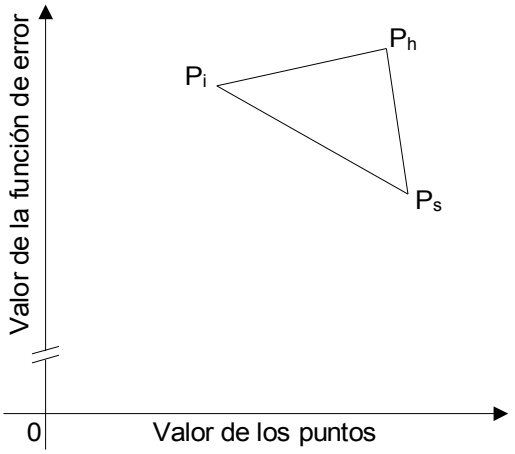
O en general, para cualquier punto P (vector), con "n" dimensiones:

$$P_3=k\cdot P_2+(1-k)\cdot P_1 \tag{14.8}$$

14.1.2. Plan inicial Simplex

Como ya se dijo, para comenzar el método se requiere un conjunto de valores iniciales asumidos, dicho conjunto debe estar conformado por n+1 valores (puntos) siendo "n" el número de dimensiones (variables) del sistema.

Por ejemplo, para dos dimensiones se requieren 3 puntos y si los puntos son los que se muestran en la figura:



Al punto que se encuentra más cerca del mínimo (más cerca de 0) se denomina P_s (s de smallest) mientras que al que se encuentra más alejado se denomina P_h (h de highest). Los otros puntos se identifican con un número.

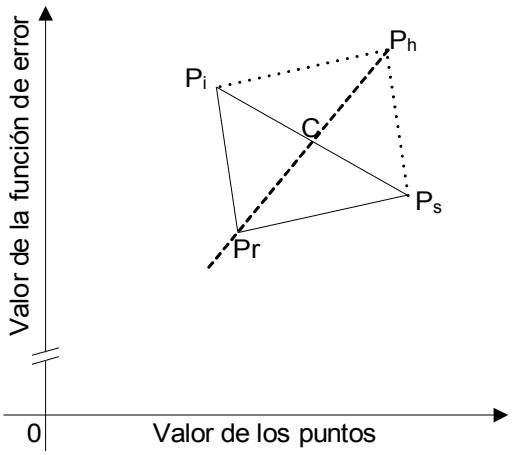
14.1.3. Centroide

En el método Simplex se busca reemplazar el peor valor asumido (o el peor valor del plan) por uno más cercano al mínimo. Con este propósito se calcula el promedio de todos los puntos del plan Simplex, pero sin incluir el punto P_h. Al punto resultante se denomina centroide "C" y se calcula con:

$$C=\frac{1}{n}\sum_{i=1}^{n+1} P_i \quad \{i\neq h \tag{14.9}$$

14.1.4. Reflexión

Para reemplazar el punto que se encuentra más alejado del mínimo (P_h), se realiza una proyección lineal del punto P_h a través del centroide C:



De manera que el punto proyectado sea el reflejo del punto P_h (siendo P_i-P_s el plano de proyección). Si se denomina "a" a la relación entre las distancias $(P_h-P_r)/(P_h-C)$ (el valor de "k" en las ecuaciones (14.5) y (14.8)), la ecuación de proyección para el cálculo del punto reflejado P_r es:

$$P_r=a\cdot C+(1-a)\cdot P_h \tag{14.10}$$

Donde el valor de "a" es 2. En la práctica, sin embargo, se emplea un valor un tanto diferente (por lo general 1.95), porque un valor entero puede dar lugar a un comportamiento oscilatorio.

Si se denomina e_r al valor de la función de error en el punto P_r ($e(P_r)$), e_h al valor de la función de error en el punto P_h ($e(P_h)$) y e_s al valor de la función de error en el punto P_s ($e(P_s)$), al realizar la reflexión se pueden presentar los siguientes casos:

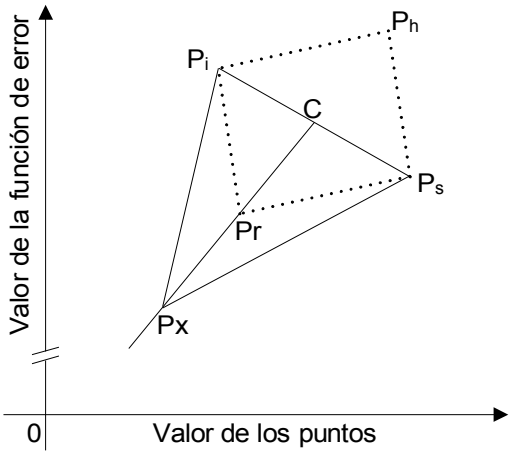
a) Si e_r es menor e_s, entonces la reflexión ha sido muy buena, pues se ha conseguido un punto más cercano aún al mínimo. Entonces se procede con la expansión (que se estudiará más adelante).

b) Si e_r es menor e_h pero no menor a e_s, entonces la reflexión es buena. En este caso se reemplaza el punto P_h por el punto P_r.

c) Si e_r es mayor a e_h, entonces la reflexión es mala. En este caso se procede con la contracción (que se estudiará más adelante).

14.1.5. Expansión

Como se dijo, cuando la reflexión es muy buena ($e_r<e_s$), se procede con la expansión, que simplemente consiste en proyectar el punto P_r al doble de la distancia que existe entre P_r y C:



Si se denomina "b" a la relación entre las distancias $(P_x-C)/(P_r-C)$, la ecuación de proyección (ecuación (14.8)) se transforma en:

$$P_x = b \cdot P_r + (1-b) \cdot C \qquad (14.11)$$

Donde "b" es igual a 2, sin embargo y como sucede en la reflexión, se emplea un valor cercano a 2 (generalmente 1.95).

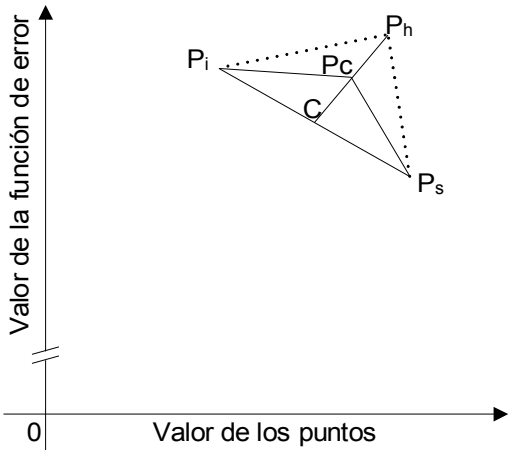
Si se denomina e_x al valor de la función de error en P_x ($e(P_x)$), entonces se pueden presentar los siguientes casos:

a) Si e_x es menor a e_r, el nuevo punto se aproxima más aún al mínimo, por lo tanto P_x reemplaza a P_h.

b) Si e_x es mayor a e_r, la expansión no ha sido buena, por lo tanto se mantiene el punto de reflexión, es decir que P_r reemplaza a P_h.

14.1.6. Contracción

Como se dijo, cuando la reflexión es mala, se procede con la contracción, que consiste en recorrer el punto reflejado a la mitad de la distancia que existe entre C y P_h:



Si se denomina "d" a la relación entre las distancias $(P_c-C)/(P_h-C)$, la ecuación de proyección (ecuación (14.8)) se transforma en:

$$P_c = d \cdot C + (1-d) \cdot P_h \qquad (14.12)$$

Donde "d" es 0.5, pero al igual que en los otros casos (para evitar un comportamiento oscilatorio) en la práctica se toma un valor cercano a 0.5 (generalmente 0.45).

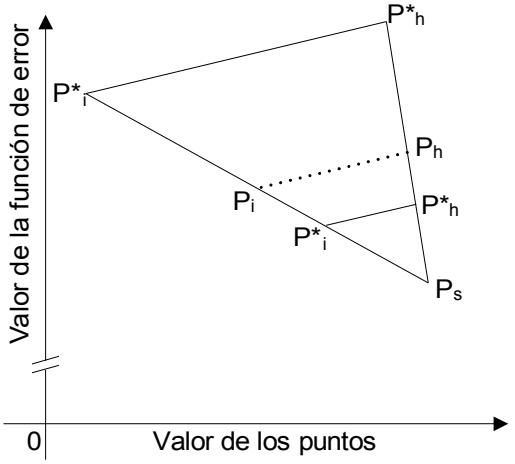
Si se denomina e_c al valor de la función de error en el punto P_c ($e(P_c)$), entonces se pueden presentar los siguientes casos:

a) Si e_c es menor a e_h, la contracción ha sido buena, entonces se reemplaza P_h por P_c.

b) Si e_c es mayor a e_h, la contracción ha sido mala, se procede con el escalamiento.

14.1.7. Escalamiento

Como se dijo, cuando la contracción es mala se procede con el escalamiento. En el escalamiento se proyectan los puntos del plan Simplex (con excepción de P_s) a través del punto P_s:



El escalamiento puede hacerse hasta la mitad de la distancia que existe entre los puntos P_i y P_s (como se muestra en la figura) o al doble de esta distancia. Si se denomina P* a los puntos resultantes de la proyección, la ecuación ecuación para el cálculo de estos puntos es:

$$P^*_i = k \cdot P_s + (1-k) \cdot P_i \quad \{i = 1 \rightarrow n+1;\ i \neq s \tag{14.13}$$

Donde el valor de k puede ser 2 (en la práctica 1.95) o 0.5 (en la práctica 0.45).

Como el escalamiento cambia todos los puntos del plan (con excepción del punto P_s), es necesario recalcular los valores de la función de error para los nuevos puntos del plan.

En el método Simplex, las operaciones de0 reflexión y las consiguientes operaciones de expansión, contracción y/o escalamiento se repiten hasta que se encuentra un valor de la función de error aproximadamente igual a cero (con un margen de error predeterminado). Dado que la convergencia no está asegurada, se debe emplear también un contador de iteraciones para terminar el proceso cuando se alcance un límite dado.

14.1.8. Plan inicial Simplex

Como ya se señaló previamente un plan Simplex está conformado por n+1 puntos, siendo "n" el número de dimensiones (variables del sistema). Si por ejemplo se tiene un sistema con 5 dimensiones (o variables), se requieren 6 puntos para el plan inicial Simplex, ello implica asumir los valores de 30 variables (5 variables para cada punto), una labor tediosa que puede desalentar inclusive el empleo del método.

Por esta razón se han desarrollado algunos procedimientos para generar los puntos del plan inicial Simplex en base a un solo punto asumido.

Uno de dichos métodos es el siguiente: Sea P_1 el punto asumido, entonces el segundo punto (P_2) se genera multiplicando el primer elemento del punto asumido por 1.1, el tercer punto (P_3) multiplicando el segundo elemento del punto asumido por 1.1, el cuarto punto (P_4) multiplicando el tercer elemento del punto asumido por 1.1 y así sucesivamente hasta generar los n+1 puntos del plan.

Por ejemplo, si el sistema tiene 3 dimensiones (o variables) y el punto asumido es P1=(15,15,150), los otros puntos del plan Simplex se obtienen de la siguiente manera:

P2 = (15*1.1, 15, 150) = (16.5, 15, 150)

P3 = (15, 15*1.1, 150) = (15, 16.5, 150)

P4 = (15, 15, 150*1.1) = (15, 15, 165)

14.1.9. Ejemplo manual

Como este método, y los otros que se estudiarán en este tema, requiere demasiadas iteraciones para encontrar un resultado, en la práctica no se emplea manualmente, sin embargo, para comprender mejor el proceso y así poder automatizar el mismo, se realizarán algunas iteraciones manuales para el siguiente sistema (que por cierto puede ser resuelto con los métodos estudiados en los temas 6, 7 y 8):

$$\begin{aligned} x^2 + 2y^2 &= 22 \\ -2x^2 + xy - 3y &= -11 \end{aligned} \tag{14.14}$$

Para resolver este sistema con el método Simplex, primero se programa la función de error y dado que la misma debe recibir un vector con "n" elementos (siendo "n" el número de variables del sistema, en este caso 2), se reescribe el sistema de ecuaciones en forma vectorial:

$$\begin{aligned} x_1^2+2x_2^2 &= 22 \\ -2x_1^2+x_1x_2-3x_2 &= -11 \end{aligned} \qquad (14.15)$$

Para crear la función de error (un problema de minimización) las ecuaciones se igualan a cero, se elevan al cuadrado y se suman:

$$e(x)=\left[x_1^2+2x_2^2-22\right]^2+\left[-2x_1^2+x_1x_2-3x_2+11\right]^2=0$$

Siendo esta la función de error "fe" que se programa para el método:

```
>> function r=fe(x) r=(x(1)^2+2*x(2)^2-22)^2+(-2*x(1)^2+x(1)*x(2)-3*x(2)+
11)^2 end
```

Ahora se inicializan variables (las constantes de las ecuaciones), se asume un vector de valores iniciales, para este ejemplo [1,2], se determina el número de dimensiones (o variables), se genera una matriz de (n+1)*n ceros, para el plan simplex y un vector de (n+1) elementos para los valores de las funciones de error:

```
>> a=1.95; b=1.95; d=0.45; k=1.95; x=[1,2]; n=length(x); p=zeros(n+1,n);
   e=zeros(1,n+1);
```

Con el procedimiento descrito en "plan inicial Simplex", se calculan los elementos de la matriz "p":

```
>> p(1,:)=x(:); for i=2:n+1 p(i,:)=x; p(i,i-1)=x(i-1)*1.1; end; p
ans =
  1    2
  1.1  2
  1    2.2
```

Donde cada fila representa uno de los puntos del plan (en este caso 3, pues el número de dimensiones es 2). Para cada uno de estos puntos se calcula el valor de la función de error:

```
>> for i=1:n+1 e(i)=fe(p(i,:)); end; e
ans = [ 194  186.43  149.3 ]
```

Se determinan los índices del menor ("s") y mayor ("h") valor de la función de error, que como se pude ver son 3 y 1 respectivamente):

```
>> s=3; h=1;
```

Se calcula el valor del centroide (c), aplicando la ecuación (14.9):

```
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n
c = [ 1.05  2.1 ]
```

Con el centroide se calcula el punto de reflexión ("pr") con la ecuación (14.7):

```
>> pr=a*c+(1-a)*p(h,:)
pr = [ 1.0975  2.195 ]
```

Y el respectivo valor de la función de error:

```
>> er=fe(pr)
er = 144.03
```

Como se puede ver, este es el menor valor de las funciones de error ("er=144.03" es menor a "e_s=149.3"), por lo tanto la reflexión ha sido muy buena. Se procede en consecuencia con la expansión (ecuación (14.6)):

```
>> px=b*pr+(1-b)*c
px = [ 1.1426  2.2852 ]
>> ex=fe(px)
```

```
ex = 122.23
```

Y dado que este valor es menor a "er=144.03", la expansión ha sido buena, por lo tanto se reemplaza el punto "p_h" del plan con el punto "px":

```
>> p(h,:)=px(:)
ans =
  1.1426  2.2852
  1.1     2
  1       2.2
```

Siendo este el nuevo plan Simplex. Se debe reemplazar igualmente el valor de la función de error respectivo:

```
>> e(h)=ex
ans = [ 122.23  186.43  149.3 ]
```

Como el menor valor de la función de error (122.23) es aún alejado de cero (el mínimo), se repite el proceso determinando primero los nuevos índices del menor ("s") y mayor ("h") valor de la función de error, que como se pude ver son 1 y 2 respectivamente:

```
>> s=1;h=2;
```

Entonces se calcula el nuevo centroide ("c"), el nuevo punto de reflexión ("pr") y el respectivo valor de la función de error ("er"):

```
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 91.157
```

Una vez más la reflexión es muy buena ("er=91.157" es menor a "e_s=122.23"), por lo que se procede con la expansión:

```
>> px=b*pr+(1-b)*c; ex=fe(px)
ex = 54.742
```

Y una vez más la expansión es buena, por lo tanto se reemplaza el punto "p_h" del plan por el punto "px":

```
>>  p(h,:)=px(:); e(h)=ex(:)
ans = [ 122.23  54.742  149.3 ]
```

Como el menor valor (54.742) es todavía lejano a cero, se repite el proceso. Los nuevos valores de "s" y "h" son:

```
>> s=2; h=3;
```

Y el nuevo valor de la función de error en el punto reflejado es:

```
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 39.542
```

Una vez más la reflexión es muy buena ("ex" es menor a "e_s"), por lo que se procede con la expansión:

```
>> px=b*pr+(1-b)*c; ex=fe(px)
ex = 11.754
```

Como la expansión es buena, "px" reemplaza a "p_h":

```
>> p(h,:)=px(:);e(h)=ex
ans = [ 122.23  54.742  11.754 ]
```

Y se repite el proceso:

```
>> s=3; h=1;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 10.05
```

Como "er" es menor a "e_s", se procede con la expansión:

```
>> px=b*pr+(1-b)*c; ex=fe(px)
ex = 99.153
```

Sin embargo, en este caso la expansión no es buena ("ex" es mayor a "er"), por lo que "pr" reemplaza a "p_h":

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  54.742  11.754 ]
```

Pero como aún el menor valor está lejos de cero, se repite el proceso:

```
>> s=1; h=2;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 53.51
```

Como la reflexión es buena ("er" es ligeramente menor a "e_h") "pr" reemplaza a "p_h" y se repite el proceso:

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  53.51  11.754 ]
>> s=1; h=2;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 46.707
```

Como la reflexión es buena "pr" reemplaza a "p_h" y se repite el proceso:

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  46.707  11.754 ]
>> s=1; h=2;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 43.603
```

Como la reflexión es buena "pr" reemplaza a "p_h" y se repite el proceso:

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  43.603  11.754 ]
>> s=1; h=2;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
er = 39.967
```

Una vez más la reflexión es buena, por lo que "pr" reemplaza a "p_h" y se repite el proceso:

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  39.967  11.754 ]
>> s=1; h=2;
>> c=zeros(1,n); for i=1:n+1 if i ~= h c+=p(i,:); end; end; c/=n;
>> pr=a*c+(1-a)*p(h,:);
>> er=fe(pr)
```

```
er = 35.735
```

Como la reflexión es buena, "pr" reemplaza a "p_h" y se repite el proceso:

```
>> p(h,:)=pr(:); e(h)=er(:)
ans = [ 10.05  34.323  11.754 ]
```

Como se pude ver, a pesar de haber repetido 10 veces el proceso, las funciones de error están todavía bastante lejos de cero, por lo que el proceso debe repetirse varias veces más (por lo menos 30 veces más para obtener una solución con un error aceptable). Hasta el momento la mejor aproximación que se tiene es el valor de p_1, es decir:

```
>> p(1,:)
ans = [ 1.1058  3.4016 ]
```

Si el proceso concluiría en esta iteración, estas serían las soluciones que debería devolver el sistema, es decir x_1=x=1.1058 y x_2=y=3.4016, valores que todavía están lejos de las soluciones correctas (x=2, y=3). Esta es la razón por la cual los métodos numéricos se aplican con la ayuda de un programa y no manualmente.

14.1.10. Ejercicio

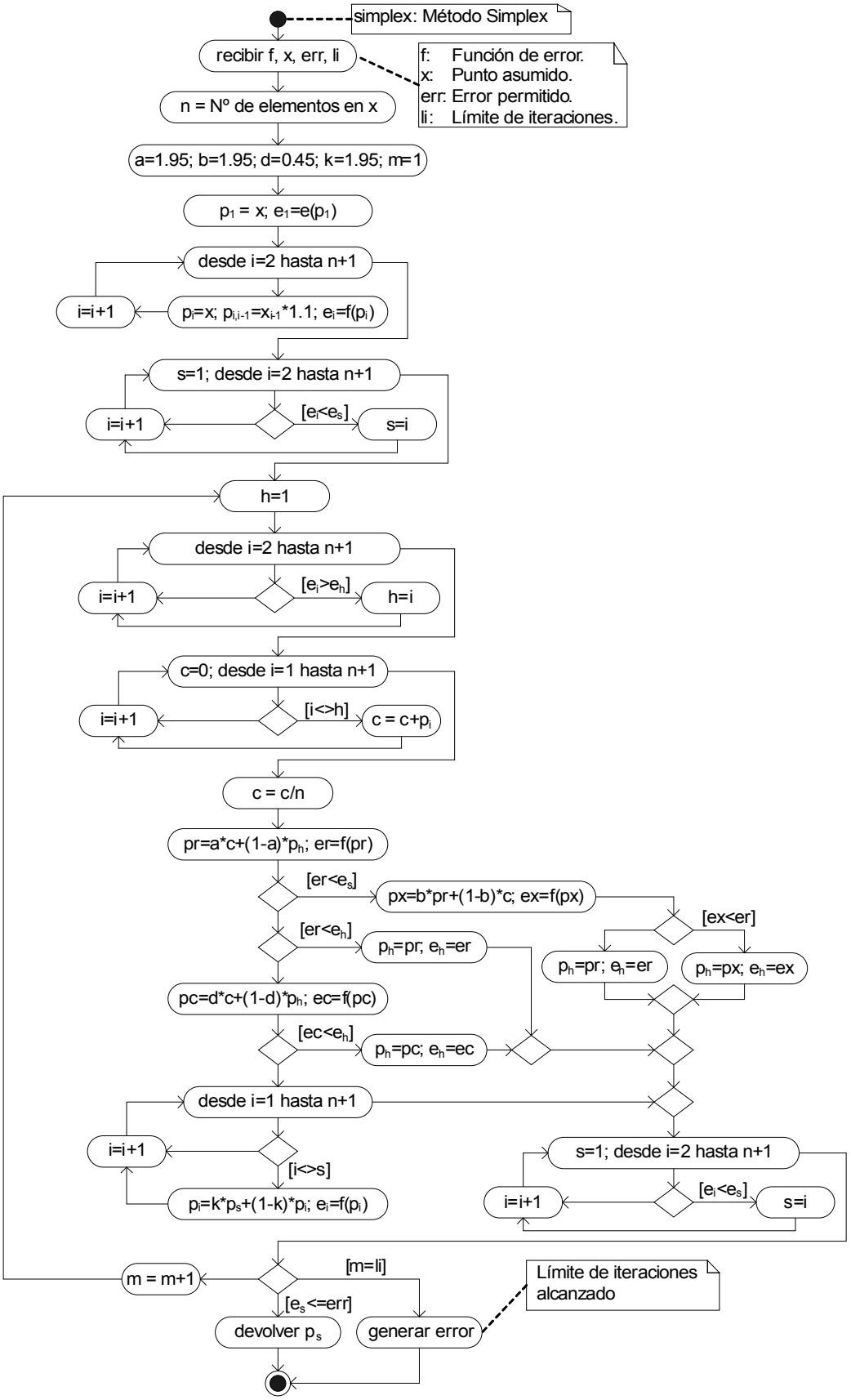
1. Calcule las soluciones aproximadas del siguiente sistema de ecuaciones no lineales, aplicando manualmente el método Simplex durante 10 iteraciones (valores iniciales asumidos: x=-7.5; y=1.5).

$$\begin{aligned} 2x^2+5xy-4x &= 115 \\ e^{\frac{x+y}{5}}+x^2y^2-70y &= 15 \end{aligned} \qquad (14.16)$$

14.1.11. Algoritmo y código

Una vez que el método ha sido comprendido, puede ser programado. El algoritmo elaborado se presenta en la siguiente página y el código respectivo es el siguiente:

```
function r=simplex(f,x,err,li)
  n=length(x); a=1.95; b=1.95; d=0.45; k=1.95; m=1;
  p=zeros(n+1,n); e=zeros(1,n+1);
  p(1,:)=x(:); e(1)=f(p(1,:));
  for i=2:n+1 p(i,:)=x; p(i,i-1)=x(i-1)*1.1; e(i)=f(p(i,:)); end
  s=1;
  for i=2:n+1 if e(i)<e(s) s=i; end end
  while 1
    h=1;
    for i=2:n+1 if e(i)>e(h) h=i; end end
    if sqrt(abs(e(h)/e(s)-1))<err r=p(s,:); return; end
    c=zeros(1,n);
    for i=1:n+1 if i ~= h  c+=p(i,:); end end
    c/=n;
    pr=a*c+(1-a)*p(h,:); er=f(pr);
    if er<e(s)
      px=b*pr+(1-b)*c; ex=f(px);
      if ex<er p(h,:)=px(:);e(h)=ex(:); else p(h,:)=pr(:);e(h)=er(:); end
    else
      if er<e(h)
        p(h,:)=pr(:); e(h)=er(:);
      else
        pc=d*c+(1-d)*p(h,:); ec=f(pc);
```


simplex: Método Simplex
recibir f, x, err, li
f: Función de error.
x: Punto asumido.
err: Error permitido.
li: Límite de iteraciones.
n = Nº de elementos en x
a=1.95; b=1.95; d=0.45; k=1.95; m=1
p1 = x; e1=e(p1)
desde i=2 hasta n+1
pi=x; pi,i-1=xi-1*1.1; ei=f(pi)
i=i+1
s=1; desde i=2 hasta n+1
[ei<es]
s=i
h=1
desde i=2 hasta n+1
[ei>eh]
h=i
c=0; desde i=1 hasta n+1
[i<>h]
c = c+pi
c = c/n
pr=a*c+(1-a)*ph; er=f(pr)
[er<es]
px=b*pr+(1-b)*c; ex=f(px)
[ex<er]
[er<eh]
ph=pr; eh=er
ph=px; eh=ex
pc=d*c+(1-d)*ph; ec=f(pc)
[ec<eh]
ph=pc; eh=ec
desde i=1 hasta n+1
s=1; desde i=2 hasta n+1
[i<>s]
pi=k*ps+(1-k)*pi; ei=f(pi)
[ei<es]
m = m+1
[m=li]
[es<=err]
devolver ps
generar error
Límite de iteraciones alcanzado

```
          if ec<e(h)
            p(h,:)=pc(:); e(h)=ec(:);
          else
            for i=1:n+1
              if i ~= s p(i,:)=k*p(s,:)+(1-k)*p(i,:); e(i)=f(p(i,:)); end
            end
          end
        end
      end
      s=1;
      for i=2:n+1 if e(i)<e(s) s=i; end end
      if sqrt(e(s)) < err break; end
      if m++ == li error("Error simplex:\n x=%f; fe=%f\n",p(s,:),e(s)); end
    end
    r=p(s,:);
  end
```

Haciendo correr el programa con la ecuación del ejemplo manual (con un error igual a 10^{-7}) se obtiene:

```
>> function r=fe(x) r=(x(1)^2+2*x(2)^2-22)^2+(-2*x(1)^2+x(1)*x(2)-3*x(2)+
   11)^2 end; simplex($fe,[1,2],1e-7,200)
r = [ 2  3 ]
```

Que son las soluciones exactas (con 6 dígitos de precisión).

14.1.12. Ejercicios

2. Encuentre las soluciones del siguiente sistema de ecuaciones con "simplex", error permitido 10^{-6} (valores iniciales x=1.1, y=1.2, z=1.3).

$$\begin{aligned} x^2+y^2+z^2 &= 17 \\ x\,y\,z &= 2 \\ x+y-z^2 &= -4 \end{aligned} \qquad (14.17)$$

3. Encuentre las soluciones del siguiente sistema de ecuaciones con "simplex" error permitido 10^{-6} (valores iniciales x_1=0.6, x_2=0.4, x_3=0.3).

$$\begin{aligned} 3x_1-\cos(x_2x_3) &= 0.5 \\ x_1^2-81(x_2+0.1)^2+\sin(x_3) &= -1.06 \\ e^{-x_1x_2}+20x_3 &= -9.472 \end{aligned} \qquad (14.18)$$

14.2. MÉTODO DEL DESCENSO ACELERADO

Al igual que en Simplex, en el método del descenso acelerado se trabaja con la función de error, pero la dirección a seguir se determina mediante el gradiente de la función:

$$\vec{z}=\nabla fe(\vec{x})=\begin{bmatrix} \dfrac{\partial fe(\vec{x})}{\partial x_1} \\ \dfrac{\partial fe(\vec{x})}{\partial x_2} \\ \cdots \\ \dfrac{\partial fe(\vec{x})}{\partial x_n} \end{bmatrix} \qquad (14.19)$$

Para aproximarse al mínimo, se resta al vector asumido el gradiente de la función multiplicado por un factor α:

$$\vec{x}_n = \vec{x} - \alpha \cdot \vec{z} \tag{14.20}$$

La forma en la que se determina α es lo que diferencia las variantes del método. En esta implementación este valor se determina normalizando primero los valores del vector "z" (para que su suma sea 1) y probando luego valores de alfa que comienzan en 1 y que en iteraciones sucesivas disminuyen a la mitad del valor anterior hasta que el valor de la función de error sea menor al valor obtenido con los valores asumidos.

Posteriormente, se verifica si es posible obtener un mejor valor de α realizando una extrapolación. Si la extrapolación genera un mejor valor, se elije el mismo, caso contrario se mantiene el valor calculado previamente.

Para el cálculo del gradiente, el principal problema radica en el cálculo de las derivadas parciales. Si bien con funciones sencillas es posible trabajar con las derivadas analíticas, en la mayoría de los casos esa alternativa no es práctica, por un lado porque consume mucho tiempo y por otro porque sería necesario repetir dicho procedimiento para cada nuevo sistema de ecuaciones.

Para una implementación que sea de utilidad práctica, el cálculo de las derivadas parciales debe ser numérico y para ello se empleará la fórmula de diferencia central de segundo orden, estudiada previamente, adaptada para el cálculo de la derivada parcial:

$$\frac{\partial\, fe(\vec{x})}{\partial\, x_i} = \frac{fe(x_1, x_2, \cdots, x_i + h, \cdots, x_{n-1}, x_n) - fe(x_1, x_2, \cdots, x_i - h, \cdots, x_{n-1}, x_n)}{2h} \tag{14.21}$$

Donde el valor de "h" se calcula con: $h = x_i \cdot 10^{-6}$.

El proceso comienza calculando el valor de la función de error para los valores iniciales asumidos (vector "x"):

$$fe_1 = fe(\vec{x}) \tag{14.22}$$

Para determinar la dirección a seguir se calcula "z" y con el mismo el valor absoluto de "z":

$$z_0 = |\vec{z}| \tag{14.23}$$

Si este valor es cero, el proceso concluye, siendo las soluciones los valores del vector "x", caso contrario, se normalizan los valores de "z":

$$\vec{z} = \frac{\vec{z}}{z_0} \tag{14.24}$$

Se fija a_3 (valor de α) en y con el mismo (y el valor del vector "z") se calcula el nuevo vector de prueba y los respectivos valores de la función de error:

$$\begin{aligned} \vec{x}_3 &= \vec{x} - a_3 \vec{z} \\ fe_3 &= fe(\vec{x}_3) \end{aligned} \tag{14.25}$$

Si "fe_3" no es menor a "fe_1", "a_3" se divide entre dos ($a_3 = a_3/2$) y se vuelven a calcular los valores de "x_3" y "fe_3", repitiendo si es necesario hasta que "fe_3" sea menor a "fe_1" o hasta que "a_3" sea menor al error permitido, en cuyo caso el proceso concluye, siendo las soluciones los valores actuales de "x".

Una vez que "fe_3" es menor a "fe_1", se calcula un nuevo valor de α extrapolado "a0" y para ello se calculan los siguientes parámetros:

$$\begin{aligned} \vec{x}_2 &= \vec{x} - a_2 \vec{z} \\ fe_2 &= fe(\vec{x}_2) \end{aligned} \tag{14.26}$$

$$h_1 = \frac{fe_2 - fe_0}{a_2} \tag{14.27}$$

$$h_2 = \frac{fe_3 - fe_2}{a_3 - a_2} \tag{14.28}$$

$$h_3 = \frac{h_2 - h_1}{a_3} \tag{14.29}$$

Y con los mismos se calcula el valor del parámetro "a_0":

$$a_0 = 0.5 \cdot \left(a_2 - \frac{h_1}{h_3} \right) \tag{14.30}$$

Con este valor (α) se calcula un nuevo valor de prueba y el correspondiente valor de la función de error:

$$\begin{aligned} \vec{x}_0 &= \vec{x} - a_0 \vec{z} \\ fe_0 &= fe(\vec{x}_0) \end{aligned} \tag{14.31}$$

Si "fe_0" es menor a "fe_3", los nuevos valores de prueba son los valores del vector "x_0", caso contrario los nuevos valores de prueba son los valores del vector "x_3".

14.2.1. Ejemplo manual

Para comprender mejor el proceso se aplicará el método a la ecuación (14.15), por lo tanto, la función de error es la misma que para el método Simplex:

```
>> function r=fe(x) r=(x(1)^2+2*x(2)^2-22)^2+(-2*x(1)^2+x(1)*x(2)-3*x(2)+
   11)^2 end
```

Puesto que en este método se deben calcular las derivadas parciales, se crea una función para ese fin:

```
>> function r=dp1(f,x,i) h=x(i)*1e-6; x(i)=x(i)+h; f2=f(x); x(i)=x(i)-
   2*h; f1=f(x); x(i)=x(i)+h; r=(f2-f1)/(2*h); end
```

Los valores iniciales asumidos, serán los mismos que en el método Simplex es decir $x=x_1=1$ y $y=x_2=2$:

```
x=[1,2]; n=length(x); z=zeros(1,n);
```

Con el vector asumidos se calcula el valor de la función de error:

```
>> fe1=fe(x)
fe1 = 194
```

Como este valor no es cercano a cero se determina la dirección a seguir con el gradiente de la función de error:

```
>> for i=1:n z(i)=dp1($fe,x,i); end z
ans = [ -72  -228 ]
```

Y se calcula su valor absoluto:

```
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 239.1
```

Con el mismo se normalizan los valores de "z":

```
>> for i=1:n z(i)=z(i)/z0; end z
ans = [ -0.30113  -0.95358 ]
```

Ahora se fija a3 en 1 y se calculan el vector x3 y el correspondiente valore de la función de error:

```
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 14.919
```

Como "fe3" es menor a "fe1", el proceso continúa con el cálculo del parámetro "a0" y los correspondientes valores de "x0" y "fe0":

```
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 33.885
```

Como "fe3" es menor a "fe0", los nuevos valores de prueba son los valores del vector "x3":

```
>> x=x3(:), fe1=fe3
x = [ 1.3011  2.9536 ]
fe1 = 14.919
```

Como el valor de la función de error está todavía lejos de cero, se repite el proceso con los nuevos valores de "x":

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 80.883
```

Como "z0" no es cero, el proceso continúa:

```
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 122.13
```

Como "fe3" no es menor a "fe1" a3 toma el valor de a3/2 y se vuelven a calcular los valores "x3" y "fe3":

```
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 15.201
```

Y como "fe3" no es menor aún a "fe1" se vuelve a repetir el cálculo:

```
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 4.1338
```

Ahora "fe3" es menor a "fe1", por lo tanto se puede continuar con el proceso calculando el nuevo valor de "a0" y "fe0":

```
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 4.1305
```

Como "fe0" es menor a "fe3" los nuevos valores de prueba son los correspondientes al vector "x0":

```
>> x=x0(:), fe1=fe0
x = [ 1.3874  3.2016 ]
fe1 = 4.1305
```

Y dado que la función de error está aún lejos de cero, se repite el proceso:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 8.2914
```

Como "z0" no es aún cero, el proceso continúa:

```
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 9.2711
```

Como "fe3" no es menor a "fe1" a3 toma el valor de a3/2 y se repite el cálculo hasta que "fe3" sea menor a "fe1":

```
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 3.3472
```

Entonces se calculan los valores de "a0" y "fe0":

```
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 2.9185
```

Ahora "fe0" es menor a "fe3", por lo tanto los nuevos valores de prueba son los del vector "x0":

```
>> x=x0(:), fe1=fe0
x = [ 1.6504  3.0323 ]
fe1 = 2.9185
```

Pero aún la función de error no es cero, por lo que el proceso se repite (ahora sin mayor explicación):

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 30.218
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 167.32
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 30.689
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 5.3963
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 1.5655
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 1.436
>> x=x0(:), fe1=fe0
x = [ 1.7025  3.1138 ]
fe1 = 1.436
```

Y dado que "fe1" no es todavía cero, se repite el proceso:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 7.8457
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 14.833
```

```
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 1.9453
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.53351
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 0.52595
>> x=x0(:), fe1=fe0
x = [ 1.8971  2.99 ]
fe1 = 0.52595
```

Como "fe1" todavía no es cero el proceso se repite:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 16.067
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 186.67
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 37.191
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 6.9865
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 1.0519
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.14485
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2;h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 0.12003
>> x=x0(:), fe1=fe0
x = [ 1.9239  3.0324 ]
fe1 = 0.12003
```

Como "fe1" no es cero, el proceso se repite:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 2.8456
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 27.559
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 4.7668
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.79908
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 9.9873E-2
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2; h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 2.4419E-2
>> x=x0(:), fe1=fe0
x = [ 1.9798  2.9969 ]
fe1 = 2.4419E-2
```

Como "fe1" no es aún cero, el proceso se repite:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
```

```
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 3.5948
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 203.43
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 44.089
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 9.8904
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 2.1808
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 7.099E-2
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 7.8139E-3
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2; h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 4.4593E-3
>> x=x0(:), fe1=fe0
x = [ 1.9858  3.0062 ]
fe1 = 4.4593E-3
```

Como “fe1” no es aún cero, el proceso se repite:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 0.57912
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 32.717
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 6.3917
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 1.3837
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.29873
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 5.8372E-2
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 8.7088E-3
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 9.7569E-4
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2; h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 7.7841E-4
>> x=x0(:), fe1=fe0
x = [ 1.9547  3.0315 ]
fe1 = 7.7841E-4
```

Repitiendo el proceso una vez más se obtiene:

```
>> for i=1:n z(i)=dp1($fe,x,i); end;
>> z0=0; for i=1:n z0+=z(i)^2; end z0=sqrt(z0)
z0 = 0.64893
>> for i=1:n z(i)=z(i)/z0; end
>> a3=1; x3=x-a3.*z; fe3=fe(x3)
fe3 = 207.73
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
```

```
fe3 = 45.871
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 10.684
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 2.5453
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.60604
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 0.14064
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 3.051E-2
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 5.656E-3
>> a3/=2; x3=x-a3.*z; fe3=fe(x3)
fe3 = 7.2781E-4
>> a2=a3/2; x2=x-a2.*z; fe2=fe(x2);
>> h1=(fe2-fe1)/a2; h2=(fe3-fe2)/(a3-a2); h3=(h2-h1)/a3;
>> a0=(a2-h1/h3)/2; x0=x-a0.*z; fe0=fe(x0)
fe0 = 1.3145E-4
>> x=x0(:), fe1=fe0
x = [ 1.9976  3.0011 ]
fe1 = 1.3145E-4
```

Si se detiene el proceso en este punto, las soluciones serían: "x=1.9976" y "y=3.0011", que son bastante cercanos a las soluciones exactas ("x=2", "y=3"). Estos resultados se consiguen en 10 iteraciones, mientras que con Simplex en 10 iteraciones los resultados apenas empiezan a acercarse a las soluciones correctas.

El método del descenso acelerado requiere, en general, menos iteraciones que el método Simplex, en contrapartida los cálculos en el método Simplex son más sencillos que los del descenso acelerado (donde se calcula el gradiente de la función y se realizan varias evaluaciones funcionales por iteración), además, el método Simplex es más estable que el método del descenso acelerado (es decir que el método Simplex logra convergencia en la mayoría de los casos, mientras que el descenso acelerado no).

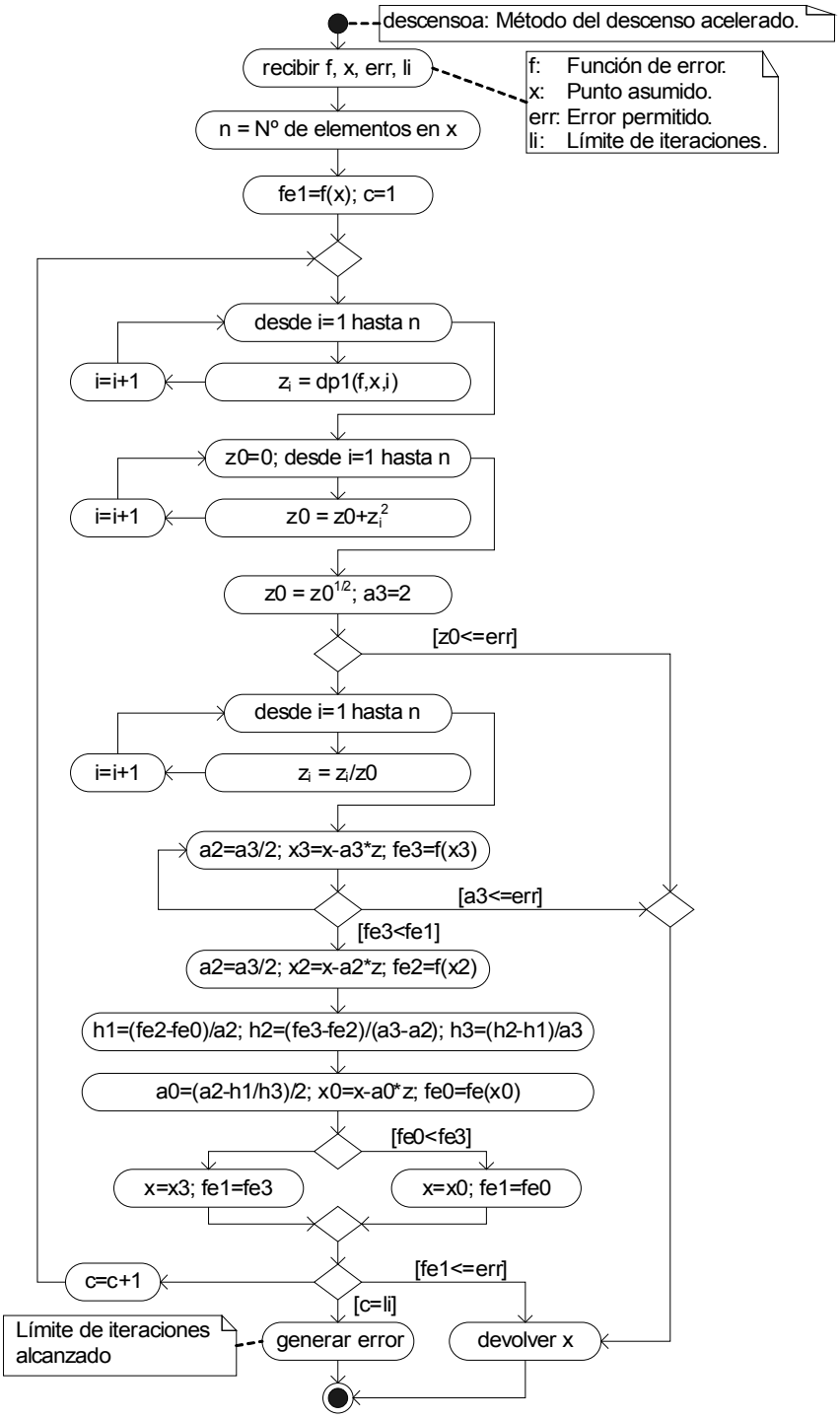
14.2.2. Ejercicio

4. Encuentre manualmente las soluciones aproximadas del sistema de ecuaciones no lineales (14.16), aplicando el método del descenso acelerado durante 10 iteraciones (valores iniciales x=-7.5; y=1.5)

14.2.3. Algoritmo y código

El algoritmo que automatiza el proceso se presenta en la siguiente página, siendo el código respectivo:

```
function x=descensoa(f,r,err,li)
  x=r(:); n=length(x); z=zeros(1,n);
  fe1=f(x); c=1;
  while 1
    fep=fe1;
    for i=1:n z(i)=dp1(f,x,i); end
    z0=0; for i=1:n z0+=z(i)^2; end
    z0=sqrt(z0); a3=2;
    if z0<=err return; end
    for i=1:n z(i)=z(i)/z0; end
    while 1
```




```
  a3/=2;
  x3=(x-a3.*z); fe3=f(x3);
  if fe3<fe1 break; end
  if a3<=err*err return; end
end
a2=a3/2; x2=x-a2.*z; fe2=f(x2);
h1=((fe2-fe1)/a2); h2=((fe3-fe2)/(a3-a2)); h3=((h2-h1)/a3);
a0=((a2-h1/h3)/2); x0=(x-a0*z); fe0=f(x0);
if fe0<fe3 x=x0(:); fe1=fe0; else x=x3(:); fe1=fe3; end
if sqrt(fe1)<err return; end
```

```
      if sqrt(abs(fep/fe1-1))<err return; end
      if c++ == li error("Error descensoa:\n x=%f; fe=%f\n",x,fe1); end
    end
  end
```

Donde "dp1" es la función implementada en el ejemplo manual para el cálculo de la derivada parcial, con una ligera modificación para tomar en cuenta los casos en los cuales el valor de la variable "x_i" es cero.

```
function r=dp1(f,x,i)
  if x(i) ~= 0  h=x(i)*1e-6; else h=1e-6; end
  x(i)=x(i)+h; f2=f(x); x(i)=x(i)-2*h; f1=f(x); x(i)=x(i)+h;
  r=(f2-f1)/(2*h);
end
```

Haciendo correr el programa para el sistema del ejemplo manual, con un error igual a 1e-9 y un límite de 100 iteraciones, se obtiene:

```
>> function r=fe(x) r=(x(1)^2+2*x(2)^2-22)^2+(-2*x(1)^2+x(1)*x(2)-3*x(2)+
   11)^2 end
>> descensoa($fe,[1,2],1e-9,100)
x = [ 2  3 ]
```

Que son los resultados exactos.

14.2.4. Ejercicios

5. Encuentre las soluciones del siguiente sistema de ecuaciones con el método del descenso acelerado (error 10^{-2}, valores iniciales x=1,y=2,z=3).

$$\begin{aligned} x\,y\,z - x^2 + y^2 &= 134 \\ x\,y - z^2 &= 0.09 \\ e^x - e^y + z &= 0.41 \end{aligned} \qquad (14.32)$$

6. Encuentre las soluciones del siguiente sistema de ecuaciones con el método del descenso acelerado (error 10^{-8}, val. iniciales x_1=1.1, x_2=1.2).

$$\begin{aligned} \sin(4\pi x_1 x_2) - 2x_2 - x_1 &= 0 \\ \left(\frac{4\pi-1}{4\pi}\right)(e^{2x_1} - e) + 4e\,x_2^2 - 2e\,x_1 &= 0 \end{aligned} \qquad (14.33)$$

14.3. MÉTODO DE NEWTON PARA SISTEMAS DE ECUACIONES NO LINEALES

Los métodos Simplex y del descenso acelerado se emplean generalmente sólo para obtener valores iniciales (debido al elevado número de iteraciones que requieren).

Una vez que se cuenta con valores iniciales adecuados se puede proseguir con un método más rápido. Uno de dichos métodos es el método de Newton para sistemas de ecuaciones no lineales.

En este método se expande el sistema de ecuaciones lineales empleando series de Taylor. Por ejemplo, si en el sistema se tienen 3 ecuaciones no lineales con 3 incógnitas:

$$\begin{aligned} f_1(x_1, x_2, x_3) &= 0 \\ f_3(x_1, x_2, x_3) &= 0 \\ f_2(x_1, x_2, x_3) &= 0 \end{aligned} \qquad (14.34)$$

La expansión en series de Taylor resulta:

$$f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)=f_1+\frac{\partial f_1}{\partial x_1}\Delta x_1+\frac{\partial f_1}{\partial x_2}\Delta x_2+\frac{\partial f_1}{\partial x_3}\Delta x_3+\cdots\infty=0$$
$$f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)=f_2+\frac{\partial f_2}{\partial x_1}\Delta x_1+\frac{\partial f_2}{\partial x_2}\Delta x_2+\frac{\partial f_2}{\partial x_3}\Delta x_3+\cdots\infty=0 \quad (14.35)$$
$$f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)=f_3+\frac{\partial f_3}{\partial x_1}\Delta x_1+\frac{\partial f_3}{\partial x_2}\Delta x_2+\frac{\partial f_3}{\partial x_3}\Delta x_3+\cdots\infty=0$$

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 $y_3 = x_3+\Delta x3$.

Por supuesto al tratarse de series infinitas, no es posible en la práctica calcular dichos valores, sin embargo, sí es posible obtener una aproximación de los mismos tomando en cuenta sólo los términos de primero orden:

$$f_1(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)\approx f_1+\frac{\partial f_1}{\partial x_1}\Delta x_1+\frac{\partial f_1}{\partial x_2}\Delta x_2+\frac{\partial f_1}{\partial x_3}\Delta x_3=0$$
$$f_2(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)\approx f_2+\frac{\partial f_2}{\partial x_1}\Delta x_1+\frac{\partial f_2}{\partial x_2}\Delta x_2+\frac{\partial f_2}{\partial x_3}\Delta x_3=0$$
$$f_3(x_1+\Delta x_1, x_2+\Delta x_2, x_3+\Delta x_3)\approx f_3+\frac{\partial f_3}{\partial x_1}\Delta x_1+\frac{\partial f_3}{\partial x_2}\Delta x_2+\frac{\partial f_3}{\partial x_3}\Delta x_3=0$$

$$\frac{\partial f_1}{\partial x_1}\Delta x_1+\frac{\partial f_1}{\partial x_2}\Delta x_2+\frac{\partial f_1}{\partial x_3}\Delta x_3=-f_1$$
$$\frac{\partial f_2}{\partial x_1}\Delta x_1+\frac{\partial f_2}{\partial x_2}\Delta x_2+\frac{\partial f_2}{\partial x_3}\Delta x_3=-f_2 \quad (14.36)$$
$$\frac{\partial f_3}{\partial x_1}\Delta x_1+\frac{\partial f_3}{\partial x_2}\Delta x_2+\frac{\partial f_3}{\partial x_3}\Delta x_3=-f_3$$

Como se ve, se forma un sistema de 3 ecuaciones lineales con 3 incógnitas: Δx_1, Δx_2 y Δx_3. No obstante estos valores son sólo aproximados, por lo que el proceso de cálculo se torna iterativo: comenzando con los valores asumidos x_1 a x_n, se calculan los valores de Δx_1, Δx_2 y Δx_3 (resolviendo el sistema de ecuaciones lineales). Si estos valores son diferentes de cero, se calculan nuevos valores de prueba: $x_1=x_1+\Delta x_1$, $x_2=x_2+\Delta x_2$; $x_3=x_3+\Delta x_3$ y el proceso se repite, empleando estos nuevos valores, hasta que los valores Δx_i son cero (o casi cero) o hasta que los valores de "x" de dos iteraciones sucesivas son iguales (o casi iguales).

El cálculo de las derivadas parciales se realiza con la ecuación (14.21) (función "dp1"), tomando en cuenta que ahora en cada llamada a la función se calculan las derivadas parciales de las funciones con respecto a x_i.

14.3.1. Ejemplo manual

Para comprender mejor el método se resolverá el sistema de ecuaciones (14.15), con el método de la Newton, empleando como valores iniciales $x=x_1=1.5$, $y=x_2=2$:

Como de costumbre, primero se programa la función, en este caso cada ecuación se evalúa por separado y los resultados de todas las ecuaciones del sistema se devuelven en una matriz:

```
>> function r=fe(x)  f1=(x(1)^2+2*x(2)^2-22); f2=(-2*x(1)^2+x(1)*x(2)-
   3*x(2)+ 11); r=[f1;f2]; end
```

Y se inicializan variables:

```
>> x=[1.5,2]; n=length(x); a=zeros(n,n); b=zeros(n,1);
```

Entonces se calcula la matriz de coeficientes (la matriz de derivadas parciales):

```
>> for i=1:n a(:,i)=dp1($fe,x,i); end; a
a =
  3     8
  -4    -1.5
```

Y el vector de las constantes (el vector de funciones con signo cambiado):

```
>> b=-fe(x)
b =
  11.75
  -3.5
```

Con la matriz de coeficientes y el vector de constantes se encuentran las soluciones del sistema con "linsolve" (o cualquiera de los métodos estudiados en los temas previos:

```
>> dx=linsolve(a,b)'
dx = [ 0.37727  1.3273 ]
```

Como estos valores no son cercanos a cero, se calculan nuevos valores de prueba:

```
>> x=x+dx
x = [ 1.8773  3.3273 ]
```

Y se repite el proceso:

```
>> for i=1:n a(:,i)=dp1($fe,x,i); end; b=-fe(x); dx=linsolve(a,b)'
dx = [ 0.13591  -0.31376 ]
>> x=x+dx
x = [ 2.0132  3.0135 ]
```

Y como los valores "dx" no son todavía cero, se repite el proceso:

```
>> for i=1:n a(:,i)=dp1($fe,x,i); end; b=-fe(x); dx=linsolve(a,b)'
dx = [ -1.3155E-2  -1.3473E-2 ]
>> x=x+dx
x = [ 2  3 ]
```

Ahora los valores "dx" son cercanos a cero, pero todavía tienen un error apreciable, por lo que se repite el proceso una vez más:

```
>> for i=1:n a(:,i)=dp1($fe,x,i); end; b=-fe(x); dx=linsolve(a,b)'
dx = [ -2.6614E-5  -3.5802E-5 ]
>> x=x+dx
x = [ 2  3 ]
```

En este punto se puede detener el proceso, pues por una parte los valores de "dx" tienen 4 ceros después del punto y por otra los valores de "x", de dos iteraciones sucesivas, se repiten (en los 5 dígitos de precisión con los que se muestran los resultados). Por lo tanto las soluciones son x=2, y=3 (que son las soluciones exactas).

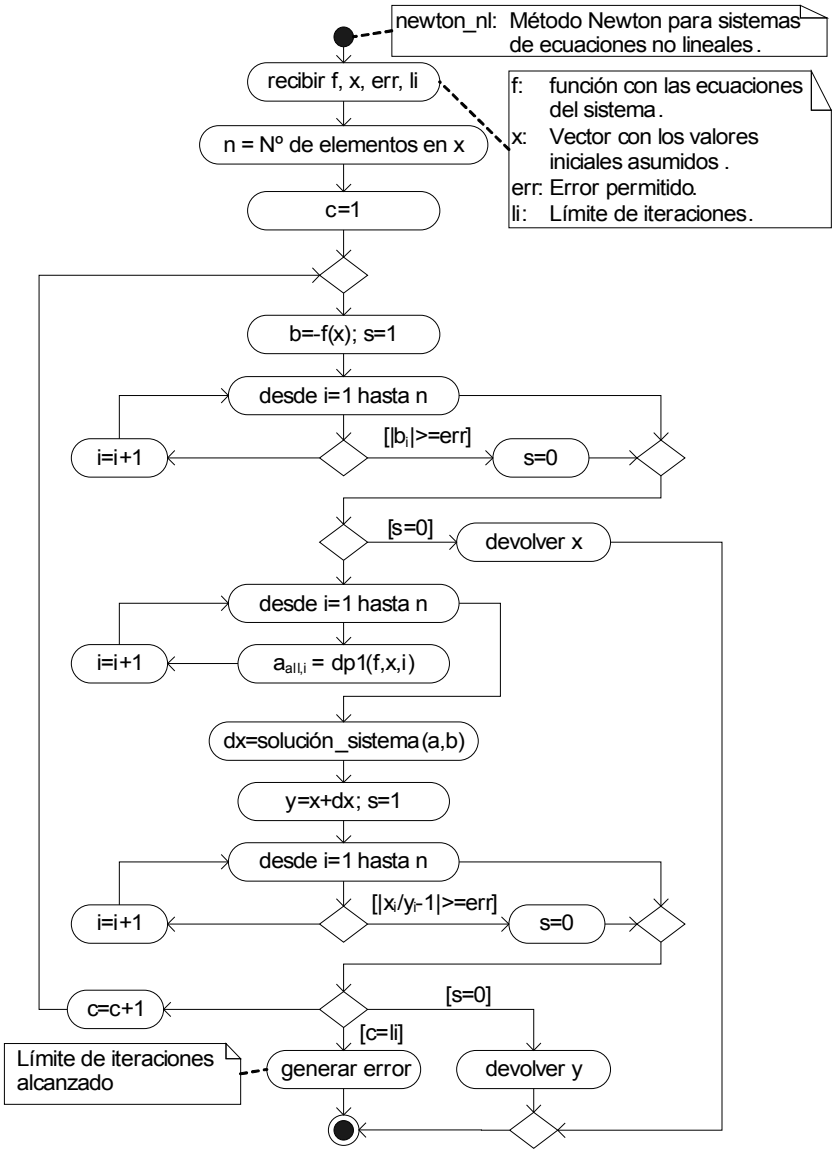
Como se puede apreciar en este ejemplo, el método de Newton converge mucho más rápidamente que Simlex y el descenso acelerado.

14.3.2. Ejercicio

7. Calcule las soluciones aproximadas del sistema de ecuaciones no lineales (14.16) (con 5 dígitos de precisión) aplicando manualmente el método de Newton (valores iniciales asumidos: x=-7.5; y=1.5).

14.3.3. Algoritmo y código

El algoritmo que automatiza el proceso de cálculo es el siguiente:



El código respectivo es:

```
function x=newton_nl(f,r,err,li)
  x=r(:); n=length(x); a=zeros(n,n); b=zeros(n,1); c=1;
  while 1
    b=-f(x); s=1;
    for i=1:n if abs(b(i,1))>=err s=0; break; end end
    if s return; end
    for i=1:n a(:,i)=dp1(f,x,i); end
    a(:,n+1)=b; dx=gauss(a)';
```

```
      y=x+dx; s=1;
      for i=1:n if abs(x(i)/y(i)-1)>=err; s=0; break; end end
      if s x(:)=y(:); return; end
      if c++ == li error("Error-newton_nd:\n x=%f; fe=%f\n",y,b'); end
      x(:)=y(:);
    end
  end
```

Haciendo correr el programa con el sistema del ejemplo manual se obtiene:

```
>> function r=fe(x) f1=(x(1)^2+2*x(2)^2-22); f2=(-2*x(1)^2+x(1)*x(2)-
   3*x(2)+11); r=[f1;f2];end
>> newton_nl($fe,[1.5,2],1e-15,100)
x = [ 2  3 ]
```

Que son las soluciones exactas.

14.3.4. Ejercicios

8. Encuentre las soluciones del siguiente sistema de ecuaciones con el método del descenso acelerado (error 10^{-15}, val. iniciales x_1=1.5, x_2=2).

$$\begin{aligned} \ln\left(x_1^2+x_2^2\right)-\sin\left(x_1 x_2\right) &= \ln(2)+\ln(\pi) \\ e^{x_1-x_2}+\cos\left(x_1 x_2\right) &= 0 \end{aligned} \qquad (14.37)$$

9. Encuentre las soluciones del siguiente sistema de ecuaciones con el método de Newton (error: 1e-9, valores iniciales: 1.1,1.1,1.1,1.1,1.1,1.1, 1.1,1.1,1.1,1.1,11.1).

$$\begin{aligned} x_1+x_4 &= 3 \\ 2x_1+x_2+x_4+x_7+x_8+x_9+2x_{10} &= 10+r \\ x_2+2x_5+x_6+x_7 &= 8 \\ 2x_3+x_5 &= 4r \\ x_1 x_5 &= a_1 x_2 x_4 \\ x_6\sqrt{x_2} &= a_2\sqrt{x_2 x_4 x_{11}} \\ x_7\sqrt{x_4} &= a_3\sqrt{x_1 x_4 x_{11}} \\ x_8 x_4 &= a_4 x_2 x_{11} \\ x_9 x_4 &= a_5 x_1\sqrt{x_3 x_{11}} \\ x_{10}x_4^2 &= a_6 x_4^2 x_{11} \\ x_{11} &= x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8+x_9+x_{10} \end{aligned} \qquad (14.38)$$

$$a_1=0.193;\ a_2=0.002597;\ a_3=0.003448;\ a_4=0.00001799;$$
$$a_5=0.0002155;\ a_6=0.00003846;\ r=4.056734$$