Indice.

Prólogo.

Esta práctica realiza un breve estudio sobre los temas más relevantes del análisis numérico. El objetivo de esta práctica es introducirnos, analizar y comparar las distintas técnicas que existen para resolver un determinado problema, en nuestro caso, estos serán los temas a tratar:

- **Resolución de ecuaciones.** Realizaremos un breve estudio de las técnicas más importántes para la resolver una ecuación, estudiando cada una de sus metodologías para posteriormente comparándolos y obteniendo una conclusión final.

- **Resolución de sistemas de ecuaciones.** Una vez analizado el problema de la resolución de ecuaciones, damos un paso más y comenzamos a estudiar las distintas técnicas que existen para resolver sistemas de ecuaciones, para una mejor comprensión de este tema decidimos dividirlo en dos partes: en sistemas de ecuaciones lineales y en sistemas de ecuaciones no lineales.

- **Interpolación polinomial. Derivación e integración numérica.** En esta sección de la práctica realizaremos dos tipos de análisis: la interpolación polinomial, donde además de introducirnos en este tema, realizaremos un estudio sobre la elección de los nodos, evaluación del polinomio interpolado en un determinado punto, y técnicas para elaborar polinomios de interpolación, en la otra parte del análisis nos dedicaremos a la derivación e integración numérica y estudiaremos un método de gran importancia: extrapolarización de Richardson.

- **Ecuaciones diferenciales.** Para concluir con la práctica realizaremos un análisis sobre los distintos métodos para la evaluación en un determinado punto de las ecuaciones diferenciales, además estudiaremos los sistemas de ecuaciones diferenciares.

Nuestro objetivo principal es proporcionar al lector una breve introducción sobre los temas enteriormente mencionados, suponemos que el lector tiene amplios conocimientos en cálculo y en la programación ya que todos los métodos explicados en esta práctica vienen acompañado de un diagrama de flujo con lo cual es una importante ayuda para su programación. Todos los ejemplos de esta práctica estan resueltos de antemano para confirmar que la implementación de los distintos métodos es la correcta.

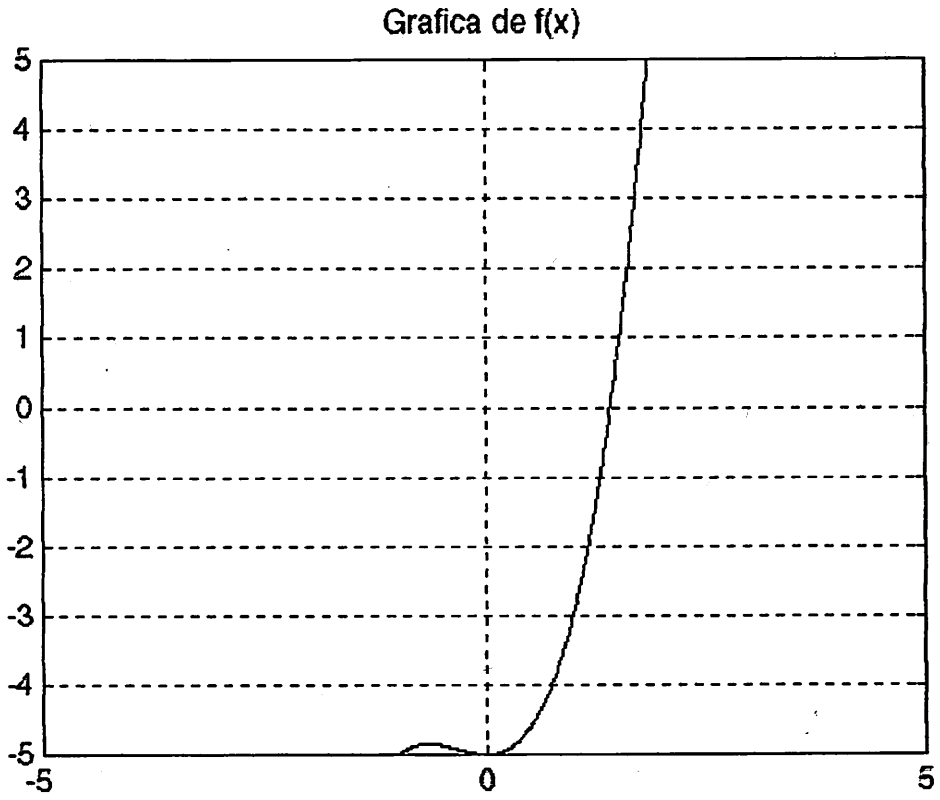
Práctica 1 Resolución de ecuaciones.

1.1 Introducción teórica.

En esta primera práctica nos ocuparemos del problema de determinar las raíces de las ecuaciones, el objetivo es encontrar las soluciones de una ecuación, calcular $p \in R$ tal que $f(p)=0$ y por tanto p sea solución. Para ello nos basaremos en los siguientes métodos:

- Método de bisección.
- Método de regula falsi.
- Método de localización aproximada de raíces.
- Método del punto fijo.
- Método de Newton-Rapshon.
- Método de la secante.
- Método de Δ^2 Aitken.
- Método de Steffesen.

Para comprender mejor cada método y compararlo posteriormente estudiaremos la siguiente ecuación: $x^3+x^2-5=0$. Antes que nada, resulta muy conveniente, realizar un análisis gráfico, para poder distinguir de manera rápida las raíces que tiene en un determinado intervalo. Utilizando las instrucciones oportunas del Matlab obtenemos la siguiente representación gráfica:

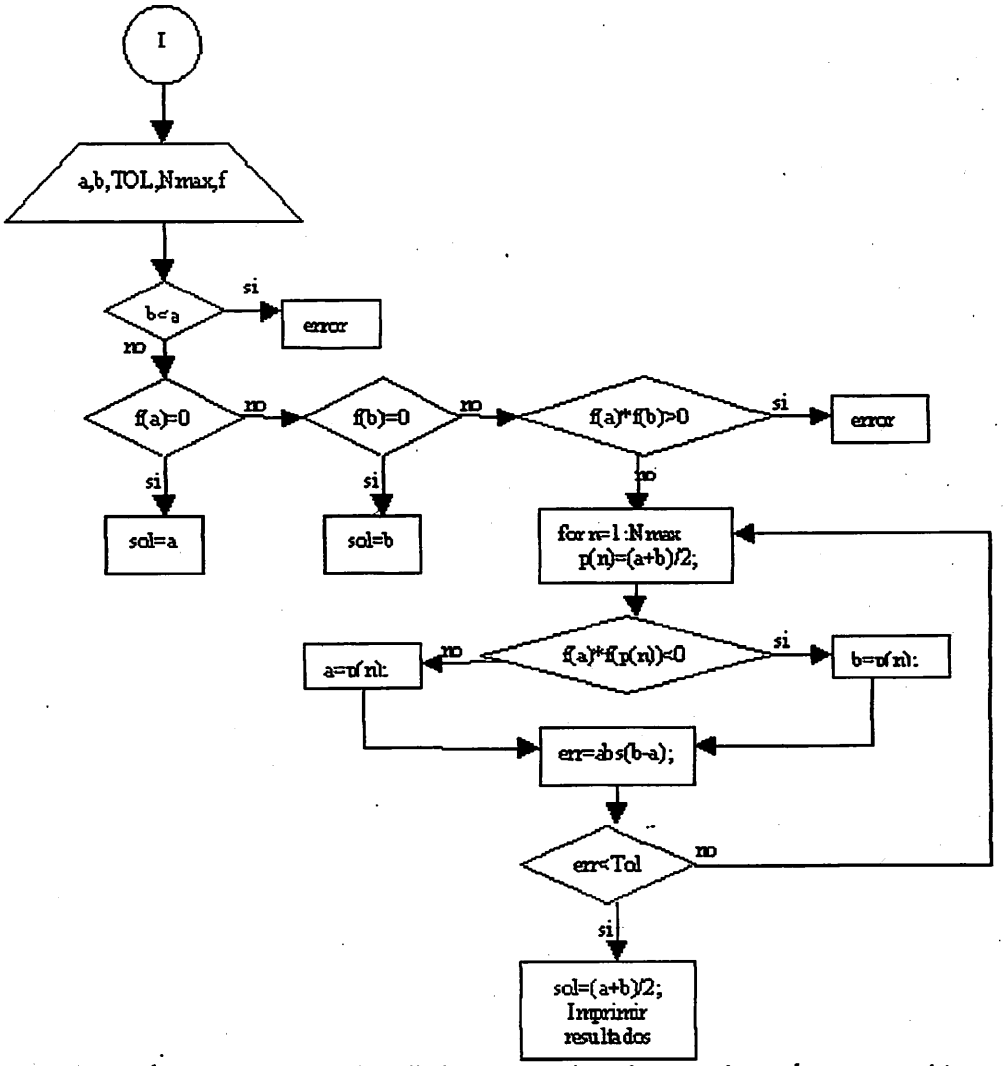


1.2 Método de bisección.

El método de bisección es de los más sencillos que estudiaremos, se basa en el teorema de Bolzano, *"f(a)·f(b)<0⇒∃ c∈[a,b] /f(c)=0"*, por lo tanto, una vez identificado un intervalo que contiene una raíz se dirige al punto medio de dicho intervalo y se queda con la parte del intervalo que siga cumpliendo con el teorema de Bolzano, así sucesivamente, de manera que iremos acotando la raíz de la ecuación.

Este algoritmo es muy deficiente pero muy seguro por eso se utilizará en algoritmos posteriores para obtener una buena aproximación, es muy raro verlo para calcular directamente la raíz de la ecuación.

Para comprenderlo mejor, diseñaremos su diagrama de flujo:



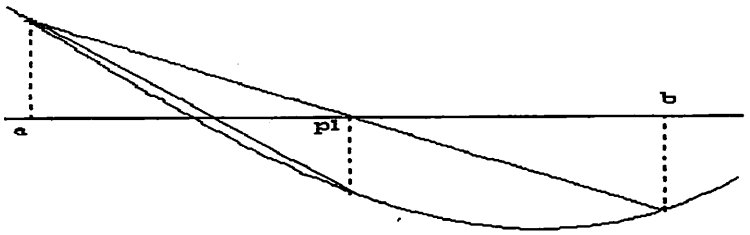
A continuación, mostraremos las distintas aproximaciones a las raíces que obtenemos al variar la tolerancia, utilizando el intervalo [1,2].

Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	10	139	0.27999999999999	1.43310546875000
10^{-5}	17	223	0.38000000000000	1.43342971801758
10^{-7}	24	307	0.33000000000000	1.43342766165733
10^{-10}	34	427	0.33000000000000	1.43342766384012

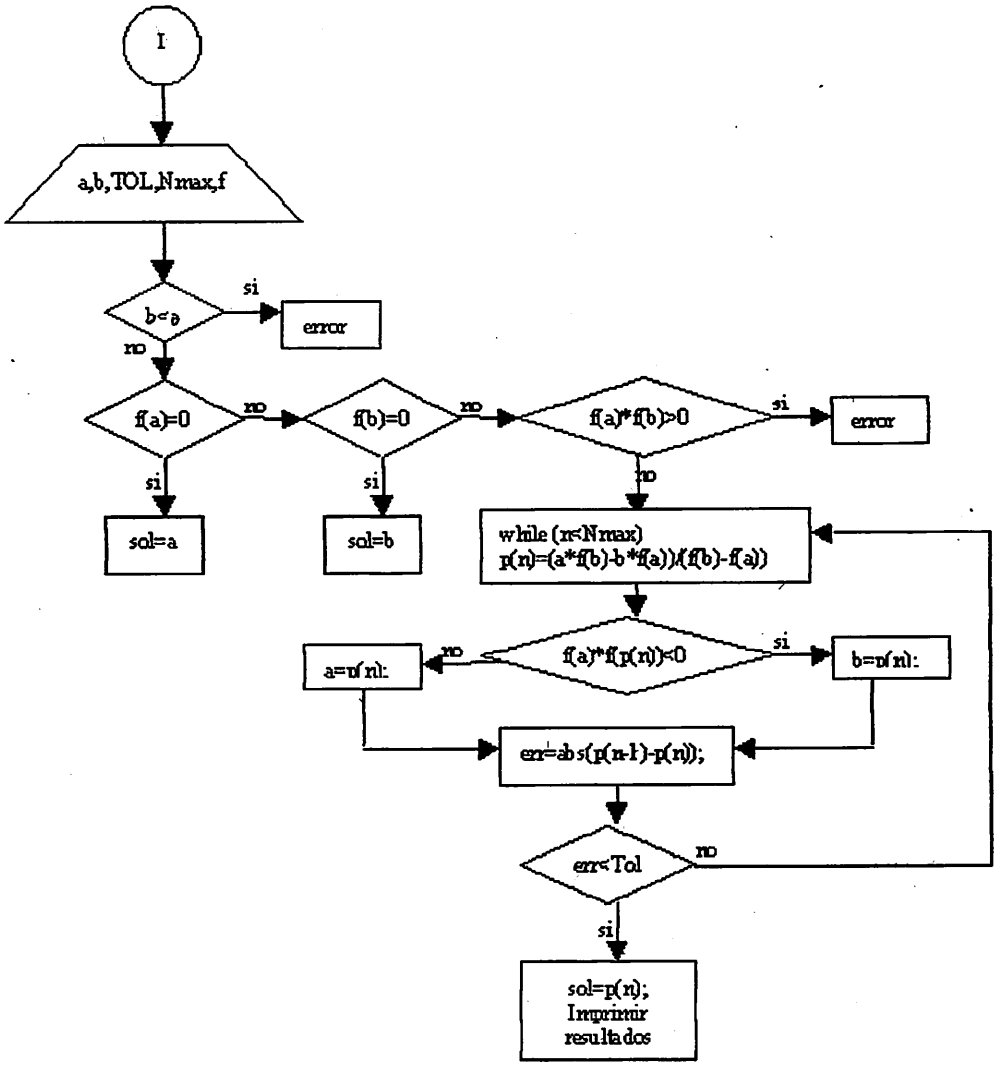
Como principal conclusión podemos obtener que al aumentar la tolerancia aumenta el número de operaciones e iteraciones, cosa que no sucede con el tiempo ya que puede verse influido por la actividad que ejerza el procesador en ese momento.

1.3 Método de regula falsi.

El método de regula falsi o de la posición falsa, al igual que el método anterior también se basa en el teorema de Bolzano, es decir, tenemos un intervalo donde el valor de la función para los extremos cambian de signos esto implica que existe un punto donde el valor de la función se anule y por tanto ese punto es raíz, pero en este caso se pretende optimizar utilizando la pendiente de la función. De manera gráfica sería:



A continuación estudiaremos su diagrama de flujo para terminar de comprender como se realiza este método:



A continuación, mostraremos los resultados obtenidos:

Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	6	287	0.21000000000000	1.43322865802549
10^{-5}	10	503	0.33000000000000	1.43342662112894
10^{-7}	13	665	0.39000000000000	1.43342764355764
10^{-10}	18	935	0.39000000000000	1.43342766388520

En el anterior método pudimos observar que a mayor exigencia, es decir, mayor tolerancia, tenemos un mayor número de iteraciones y de operaciones, pero esta regla no se le puede aplicar al tiempo, ya que el tiempo de ejecución de la función no depende de dicha función sino que depende de la actividad de procesador en el momento de ejecución de la función. Podemos observar que ejecutando la misma función con los mismos parámetros, la salida que puede sufrir modificaciones es el tiempo. De todos modos, pensamos que es interesante mostrarlo ya que a pesar de este inconveniente, resulta un buen parámetro para determinar la eficiencia de un determinado algoritmo.

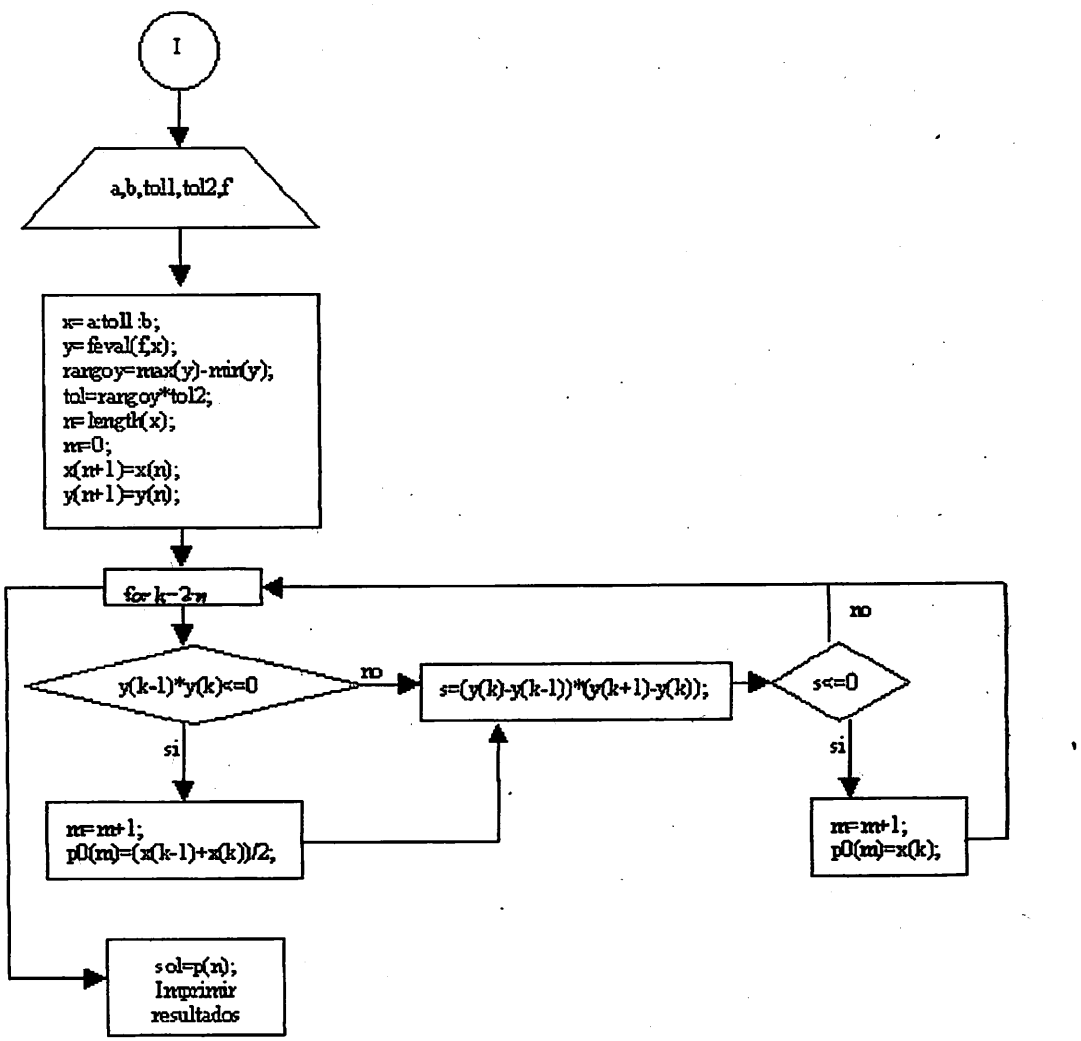
1.4 Método de localización aproximada de raíces.

Es una estimación aproximada de la localización de una raíz de la ecuación f(x)=0 en el intervalo [a, b] mediante el uso de puntos de muestra equiespaciados $(x_k, f(x_k))$ y de acuerdo con los siguientes criterios:

$\rightarrow (y_{k-1})(y_k)<0$, o

$\rightarrow |y_k| < \varepsilon$ e $(y_k - y_{k-1})(y_{k+1} - y_k)<0$

Esto es, o bien $f(x_{k-1})$ y $f(x_k)$ tienen distintos signo, o bien $|f(x_k)|$ es pequeño y la pendiente de la curva y=f(x) cambia de signo cerca de $(x_k, f(x_k))$. Su diagrama de flujo es:



La solución que obtenemos es:

Tolerancia	N º de operaciones	Tiempo	Aproximación
10^{-1}	122	0.06000000000000	1.45000000000000
10^{-2}	1112	0.28000000000000	1.43500000000000
10^{-3}	11012	0.61000000000000	1.43350000000000
$10^{-3.5}$	34794	1.37000000000000	1.43339015332608

En este caso no podemos sobrepasar la tolerancia 10^{-4} ya que trabajamos con una versión estudiantil de Matlab y no nos permite hacer un cierto número de operaciones. A pesar de eso, podemos observar claramente que este algoritmo es uno de los peores que hemos estudiados hasta ahora, realiza muchas operaciones, tarda bastante tiempo y además obtenemos una aproximación no muy adecuada, se basa en una idea compleja y en la situación en que pudiéramos decidir por un método u otro, en ningún caso escogeríamos este método.

1.5 Método del punto fijo.

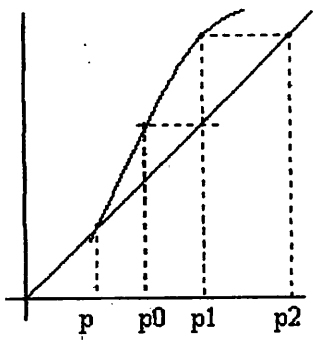
Para comprender mejor este método volvamos a la introducción teórica de la práctica, recordemos que nuestro objetivo en esta práctica es resolver una ecuación, es decir, calcular las raíces reales que anulen a la ecuación, f(x)=0.

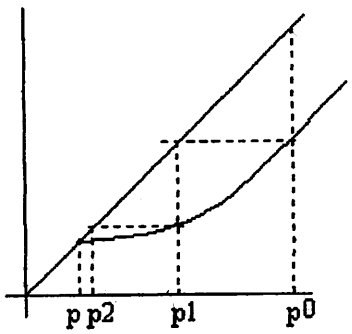
El método del punto fijo realiza un cambio sobre esta función para que resulte más sencillo de resolver:

→f(x)=0 Tenemos que calcular x tal que anule a f(x).

→x=g(x) Tenemos que calcular x tal que corte las ecuaciones: y=x e y=g(x).

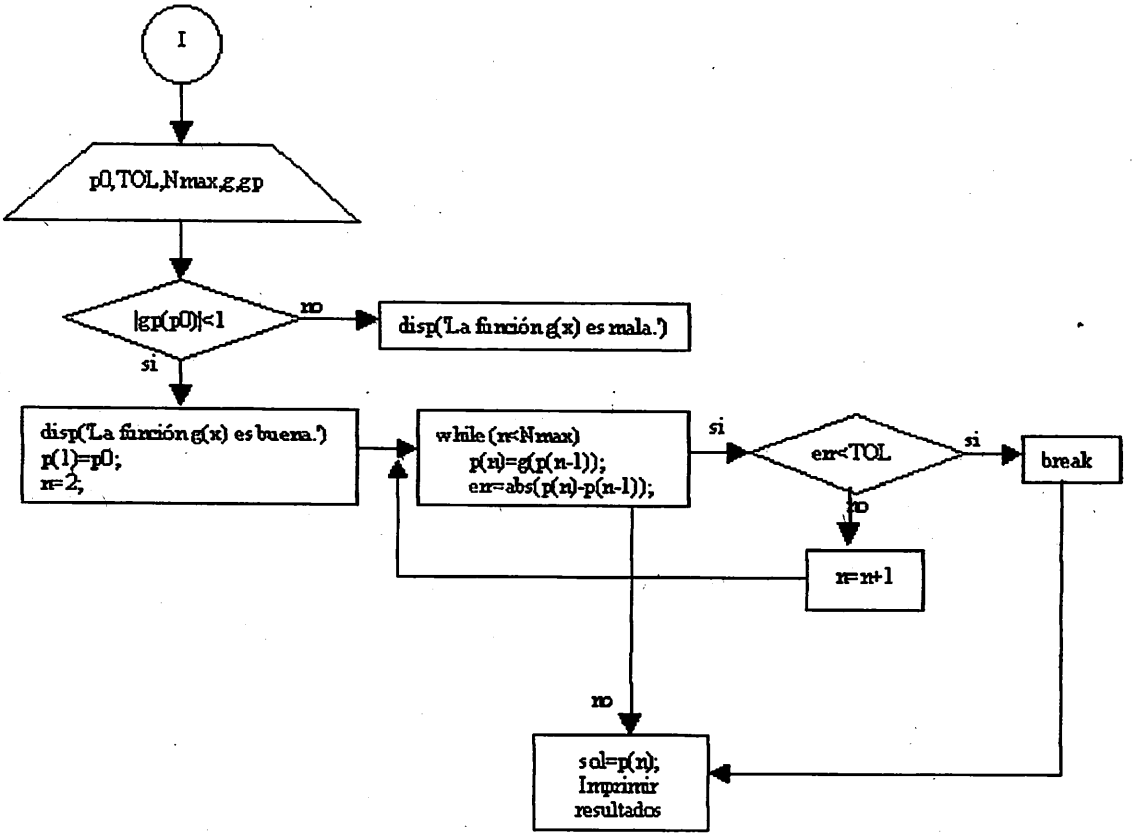
¿Pero qué g(x) tengo que utilizar? No todas las g(x) que se nos ocurran serán buenas, es decir existen algunas que en vez de converge divergen, tendremos que identificar cuales son las buenas y cuales son las malas, para ello tendremos en cuenta la pendiente de la función, es decir la derivada de la función. De manera gráfica sería:





Consideraremos como g(x) buenas aquellas cuya $|g'(x)|<1$, en otras palabras, las que están por debajo de la recta y=x. Pero dentro de las buenas existen algunas que no son tan buenas ya que puede converger localmente, es decir converger para una raíz y divergir para la otra, otra consideración que tendremos que tener en cuenta son los p0 que lo obtendremos de los métodos anteriormente vistos (bisección, regula falsi y localización de aproximación de raíces).

Para comprender mejor este método observemos el diagrama de flujo:



Los resultados que obtenemos, con un p0=1, son:

Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	5	75	0.22000000000000	1.43342767228085
10^{-5}	6	89	0.28000000000000	1.43342766386382
10^{-7}	6	89	0.27000000000000	1.43342766386382
10^{-10}	7	103	0.33000000000000	1.43342766386382

¿Por qué obtenemos idénticos resultados a partir de una cierta tolerancia? Suponemós que el principal motivo que origina esta situación es la elección del criterio de paro, también podemos observar lo eficiente que es este algoritmo cuya principal desventaja es la elección de la g(x).Debido a la imposibilidad de encontrar una buena g(x) hemos optado por la siguiente g(x):

$$g(x)=x-\frac{f(x)}{f'(x)}$$

Esta g(x) es la más eficiente y es la utilizada por el método Newton-Rapshon que explicaremos a continuación.

1.6 Método de Newton-Rapshon.

En el método del punto fijo el principal problema era saber si una g(x) es buena o mala, esta cuestión se resolvía utilizando el siguiente criterio: consideraremos g(x) como buenas si $|g'(x)|<1$. Pero dentro del conjunto de g(x) que cumple el criterio cuál es la mejor, o mejor dicho, cuál es la que converge más rápidamente. El método de Newton-Rapshon demuestra que la siguiente g(x) es la que converge más rápidamente.

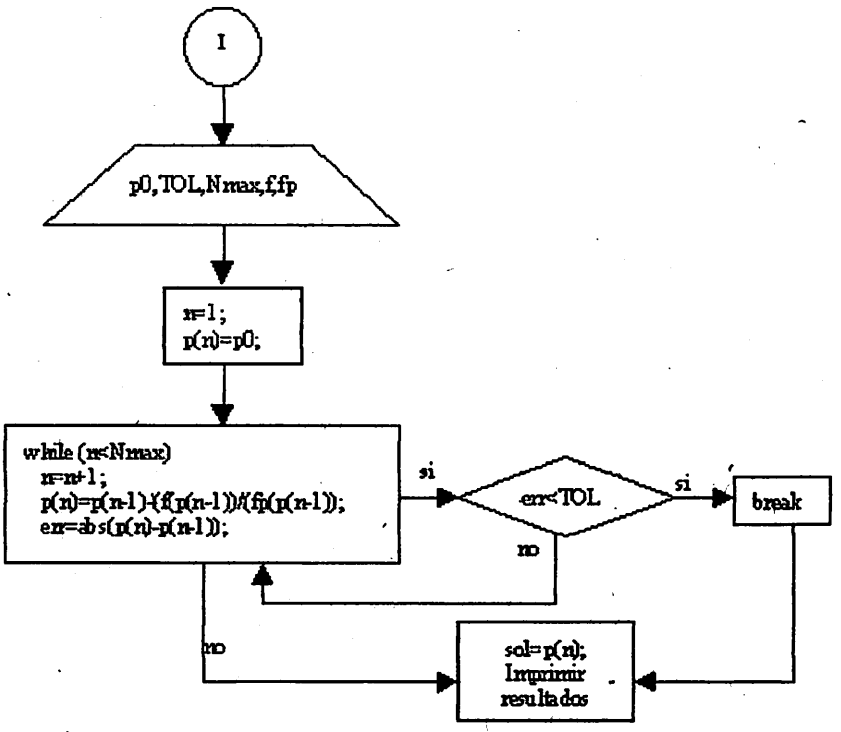
$$g(x)=x-\frac{f(x)}{f'(x)}$$

A continuación mostraremos su demostración:

$$f(x)=0 \rightarrow \frac{f(x)}{f'(x)}=0 \rightarrow x = x - \frac{f(x)}{f'(x)} = g(x)$$

$$g'(x)=1 - \frac{f'(x)f'(x)-f(x)f''(x)}{(f'(x))^2} \qquad g'(p)=1 - \frac{f'(p)f'(p)-f(p)f''(p)}{(f'(p))^2}=1-1=0$$

Su diagrama de flujo es:



Los resultados que obtenemos, con un p0=1, son:

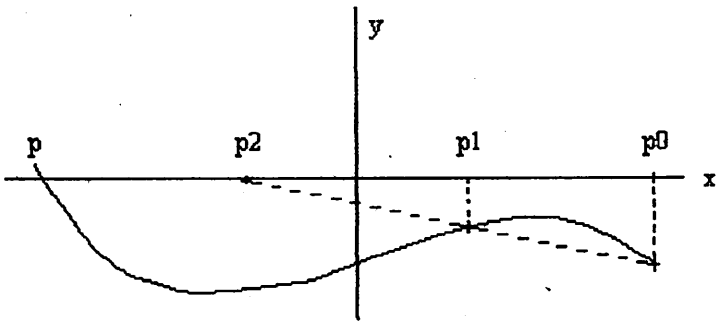
Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	5	64	0.27000000000000	1.43342767228085
10^{-5}	6	80	0.27000000000000	1.43342766386382
10^{-7}	6	80	0.28000000000000	1.43342766386382
10^{-10}	7	96	0.27000000000000	1.43342766386382

Como podemos observar obtenemos idénticos resultados con el método del punto fijo, eso es debido a que se tratan del mismo método, cuando el método del punto fijo utiliza la g(x) más eficiente, es decir, la que utiliza Newton-Rapshon. Además podemos apreciar que el método de Newton-Rapshon es más eficiente que el método del punto fijo, no sólo porque no tenemos que preocuparnos en la elección de la g(x), sino que utilizando la misma g(x) obtenemos resultados idénticos con un menor número de operaciones.

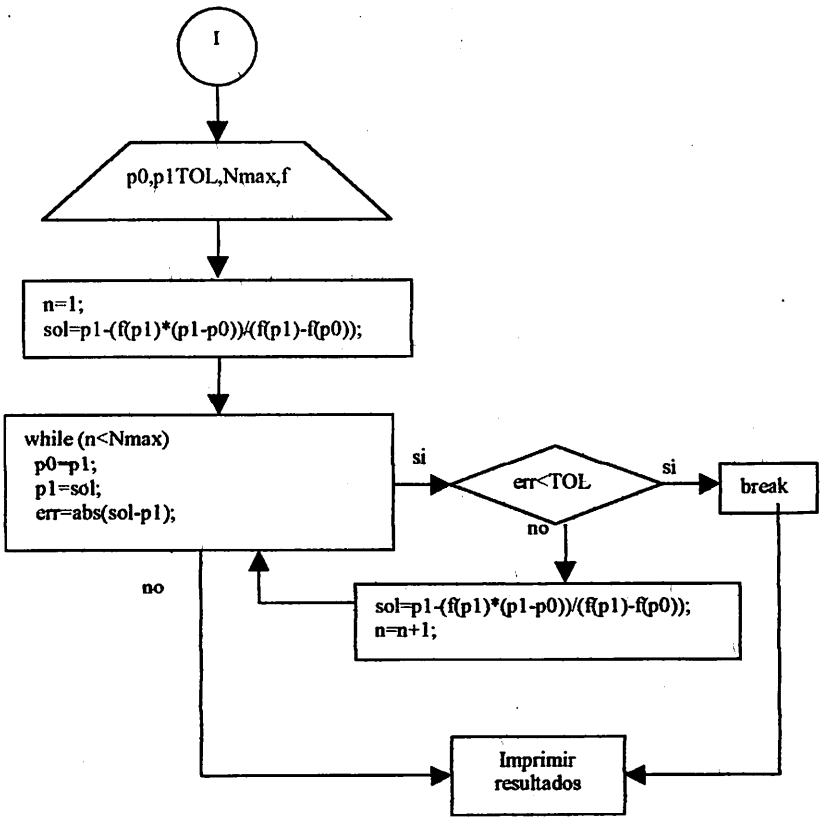
1.7 Método de la secante.

El método de la secante es un método que se basa en la pendiente de la recta para evitar de ese modo la evaluación de dos funciones: f(x) y f'(x), como sucedía con el método de Newton-Rapshon, su orden de convergencia es de 1.618033989; es casi tán eficiente como el de Newton-Rapshon cuyo orden de convergencia es de 2, por contra necesita dos aproximaciones para establecer la recta pendiente.

De manera gráfica sería:



A continuación mostraremos su diagrama de flujo para terminar de comprender como se realiza este método:



Los resultados que obtenemos, con un p0=1.4331 y tolerancia $=10^{-3}$, son:

p1	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
1.4332	1	18	0.27000000000000	1.43342770765452
1.4333	1	18	0.33000000000000	1.43342768841874
1.4334	1	18	0.33000000000000	1.43342766918449
1.4335	1	18	0.33000000000000	1.43342764995177

Observamos que este método la mejora de la precisión está especificado por las dos aproximaciones que necesita y no por la tolerancia, el principal inconveniente de este método es que está muy influido por las dos aproximaciones, estas aproximaciones se puede obtener por cualquier método visto anteriormente: bisección, regula falsi, etc.

1.8 Método de Δ^2 Aitken.

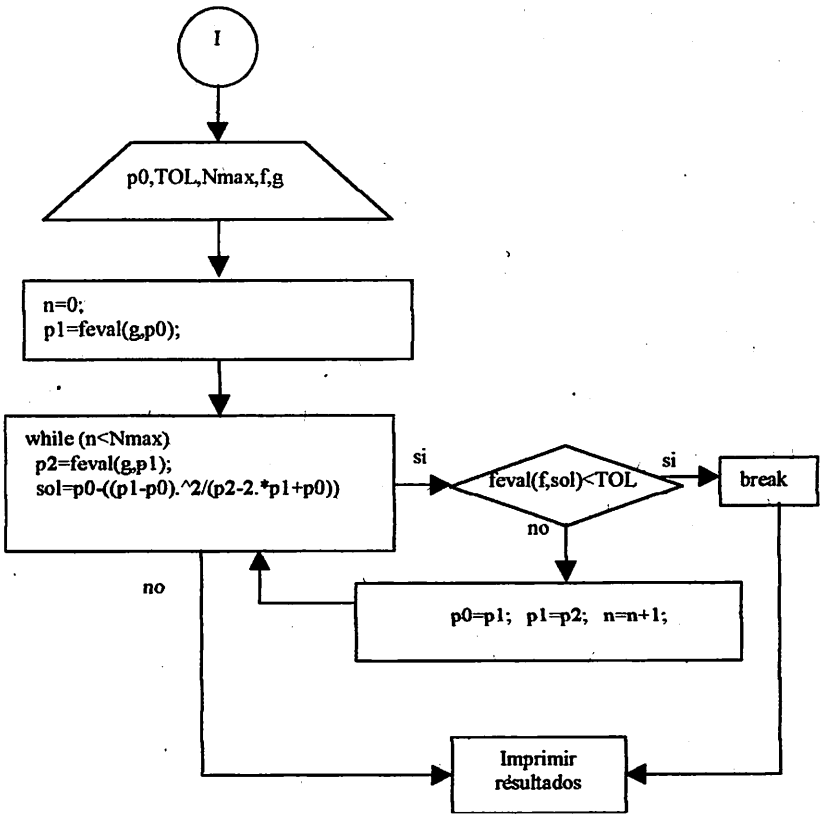
Aitken crea una sucesión que converge superlineamente a p, a partir de una sucesión $\{p_n\}$ que converge linealmente a p, esta sucesión es la siguiente:

$$P_0 \rightarrow P_1 = g(P_0) \rightarrow P_2 = g(P_1) \rightarrow \cdots \rightarrow P \quad \text{LINEAL}$$

$$\downarrow$$

$$P'_0 = P_0 - \frac{(P_1 - P_0)^2}{P_2 - 2P_1 + P_0} = P_0 - \frac{\Delta P_0^2}{\Delta^2 P_0} \rightarrow \cdots \rightarrow P \quad \text{SUPERLINEAL}$$

Como viene siendo habitual, a continuación mostraremos su diagrama de flujo:



Los resultados que obtenemos, con un p0=1 son:

Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	2	84	0.22000000000000	1.43342665713637
10^{-5}	2	84	0.44000000000000	1.43342665713637
10^{-7}	3	109	0.33000000000000	1.43342766386323
10^{-10}	3	109	0.33000000000000	1.43342766386323

Podemos observar que el método de Aitken obtiene muy buenos resultados en pocas iteraciones, esto es lo realmente soprendente de este método, que de un conjunto de aproximaciones a p obtenga mejores aproximaciones, y con mayor eficiencia, a p. El siguiente método que estudiaremos es el método de Steffensen y se basa sobretodo en el método de Aitken.

1.9 Método de Steffesen.

El método de Steffesen es una optimización del método de Aitken, recordemos que Aitken para obtener una buena aproximación necesitaba un conjunto de aproximaciones que convergan linealmente a p, este conjunto lo obtenía aplicando el método del punto fijo, utilizando este conjunto iva obteniendo aproximaciones que convergen superlinéalmente a p, Steffenssen descubre que no hace falta utilizar toltamente ese conjunto, se consigue una mejor optimización si una vez conseguido la primera aproximación de Aitken aplicamos dos veces el punto fijo, de esa manera si volvemos a aplicar Aitken con esos tres valores (1ª aprox. De Aitken y las dos aplicación del punto fijo) obtenemos una mejor solución así, sucesivamente hasta cumplir el criterio de paro seleccionado. De manera esquemática quedaría:

$P_0 \rightarrow P_1 = g(P_0) \rightarrow P_2 = g(P_1)$

↓ Aitken

$P'_0 \rightarrow P'_1 = g(P'_0) \rightarrow P'_2 = g(P'_1)$

↓ Aitken

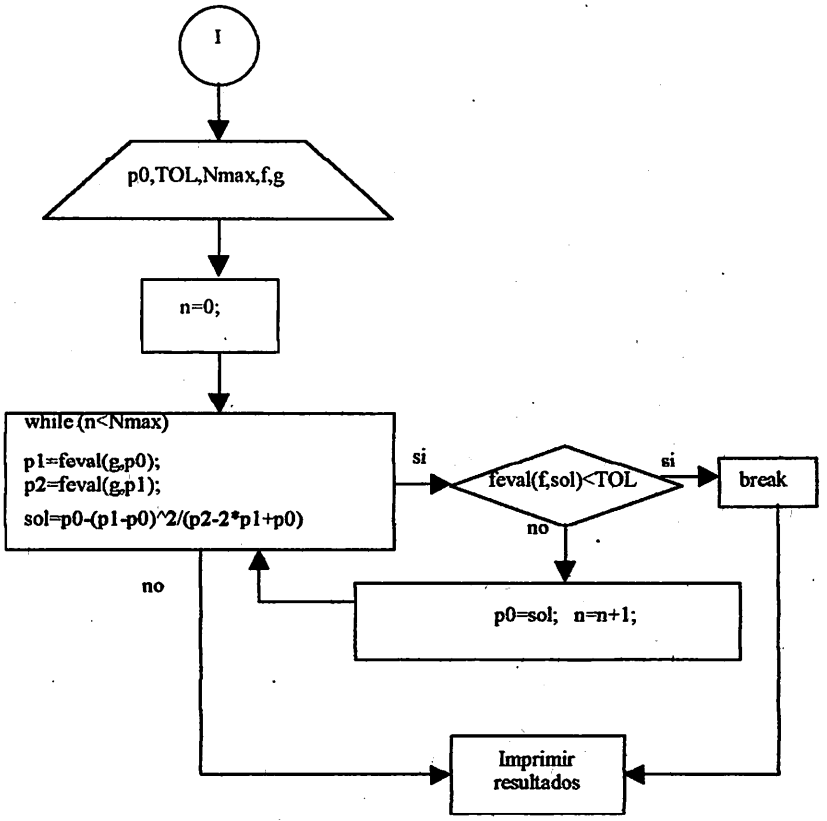
...

Los resultados que obtenemos, con un p0=1 son:

Tolerancia	N º de iteraciones	N º de operaciones	Tiempo	Aproximación
10^{-3}	2	102	0.32000000000000	1.43342766386383
10^{-5}	3	138	0.33000000000000	1.43342766386382
10^{-7}	3	138	0.27000000000000	1.43342766386382
10^{-10}	3	138	0.33000000000000	1.43342766386382

A partir de una cierta torelancia obtenemos idénticos resultados, el principal motivo de este hecho, es el desarrollo del algoritmo, existe una torelancia en el que el algoritmo no puede proporcionar mejor aproximación a p, lo que si podemos asegurar es que nuestra ecuación tiene una solución en el punto 1.43342766386382 desconocinedo el resto de los valores.

El diagrama de flujo del método de Steffensen es el siguiente:



1.10 Comparaciones y conclusiones.

		Bisección	Regula falsi	Localización de raíces *	Punto fijo	Newton.	Secante	Aitken	Steffesen
Tol: 10^{-3}	Itr.	10	6	-	5	5	1	2	2
	Op	139	287	122	75	64	18	84	102
	Sol	1.43310546875000	1.43322865802549	1.45000000000000	1.43342767228085	1.43342767228085	1.43342770765452	1.43342665713637	1.43342766386383
Tol: 10^{-5}	Itr.	17	10	-	6	6	1	2	3
	Op	223	503	1112	89	80	18	84	138
	Sol	1.43342971801758	1.43342662112894	1.43500000000000	1.43342766386382	1.43342766386382	1.43342768841874	1.43342665713637	1.43342766386382
Tol: 10^{-7}	Itr.	24	13	-	6	6	1	3	3
	Op	307	665	11012	89	80	18	109	138
	Sol	1.43342766165733	1.43342764355764	1.43350000000000	1.43342766386382	1.43342766386382	1.43342766918449	1.43342766386323	1.43342766386382
Tol: 10^{-10}	Itr.	34	18	-	7	7	1	3	3
	Op	427	935	34794	103	96	18	109	138
	Sol	1.43342766384012	1.43342766383520	1.43339015332608	1.43342766386382	1.43342766386382	1.43342764995177	1.43342766386323	1.43342766386382

*En el método de Localización de Raíces se utiliza las siguiente torelancias: 10^{-1},10^{-2},10^{-3},$10^{-3.5}$ debido a la particularidad de este método.

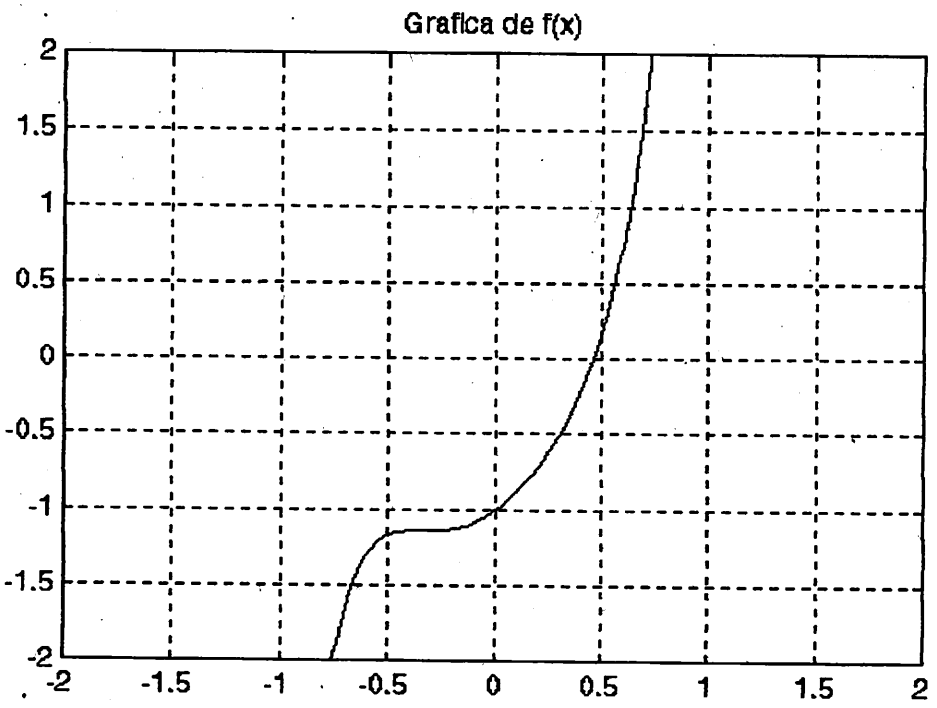
Como podemos observar existen métodos como bisección, regula falsi, etc, que no son buenos para obtener una aproximación final a p debido a la gran cantidad de iteraciones y operaciones que realizan, por contra si son muy buenos métodos para obtener una primera aproximación, aproximación que necesita los métodos: punto fijo, Newton-Rapshon, etc. Estos métodos si no permite obtener una buena aproximación final a p, todos los métodos estudiado hasta ahora tiene ventajas e inconvenientes, elegir el más eficiente no es tarea sencilla ya que dependería en gran medida de la situación que tratemos.

1.11 Ejemplos.

1.-Dado la siguiente función $f(x)=x^7+5x^5+2x^2+x-1$, estimar una raíz real para el intervalo [-2,2], utilizando los métodos anteriormente vistos, con distintas tolerancias, y utilizando en aquellos métodos que lo precisen la siguiente $g(x)=1/(x^6+5x^4+2x+1)$

Antes de nada resulta muy interesante ver la gráfica de esta función, de esa manera podremos observar de manera rápida que en el intervalo [-2,2] sólo existe una raíz, que será la que tenemos que calcular.

La gráfica de la función $f(x)=x^7+5x^5+2x^2+x-1$ en el intervalo [-2,2] es la siguiente:



Como podemos observar sólo tiene una raíz en el intervalo [-2,2] a continuación mostraremos los resultados obtenidos y además la evolución de las iteraciones para los distintos métodos.

Los resultados obtenidos son:

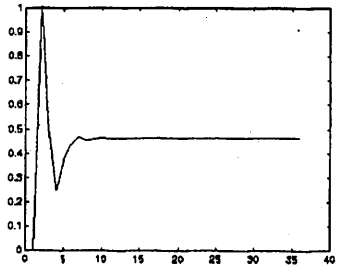
		Bisección	Regula falsi	Localización de raíces *	Punto fijo	Newton.	Secante	Aitken	Steffesen
Tol: 10^{-3}	Itr.	12	134	-	35	4	1	1	2
	Op	303	13869	657	422	75	33	35	54
	Sol	0.46240234375000	0.41311965248715	0.45000000000000	0.46277430284191	0.46234002745006	0.42953020134228	0.46243888937338	0.46234002786894
Tol: 10^{-5}	Itr.	19	357	-	70	5	1	15	3
	Op	457	37061	6419	842	100	33	427	82
	Sol	0.46233749389648	0.46188295517413	0.46500000000000	0.46233561310469	0.46234002719882	0.42953020134228	0.46234248137950	0.46234002719881
Tol: 10^{-7}	Itr.	26	573	-	105	5	1	33	3
	Op	611	59525	64019	1262	100	33	931	82
	Sol	0.46234002709389	0.46233537822217	0.46250000000000	0.46234007206663	0.46234002719882	0.42953020134228	0.46234004904621	0.46234002719881
Tol: 10^{-10}	Itr.	36	899	-	158	6	1	59	4
	Op	831	93429	6400019	1898	125	33	1659	110
	Sol	0.46234002718120	0.46234002263203	0.46234500000000	0.46234002715575	0.46234002719882	0.42953020134228	0.46234002722272	NaN

*En el método de Localización de Raíces se utiliza las siguiente torelancias: 10^{-1},10^{-2},10^{-3},10^{-5} debido a la particularidad de este método.

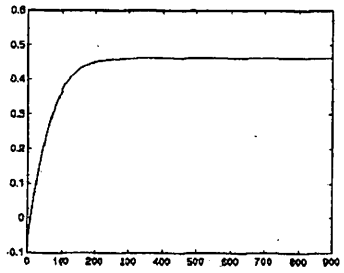
Sabemos que la solución es 0.46234002719882, de esta manera, observamos que los métodos Newton y Steffensen son los que más se aproximan y que los métodos más deficientes son: aproximación de raíces y el método de la secante, principalmente los motivos de que esto sea así son: que estén mal implementados, que le pasemos unos parámetros incorrectos o que se basan en una idea original pero poco eficiente.

Para tener una mejor idea de como evoluciona la solución para distintas iteraciones en diferentes métodos mostramos a continuación la gráfica de estas evoluciones:

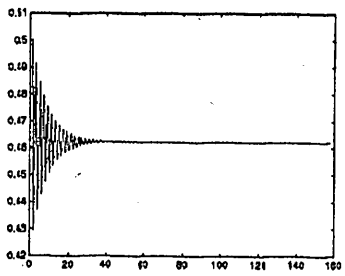
Bisección



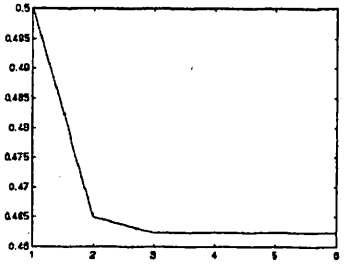
Regula falsi



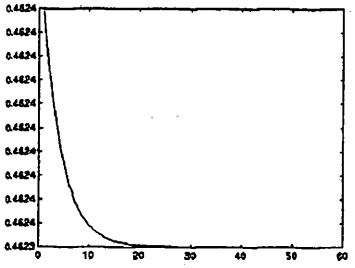
Punto fijo



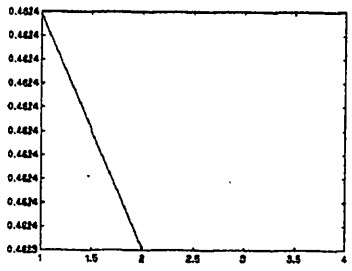
Newton



Aitken



Steffensen



Podemos observar que faltan la representaciones de los métodos localización de raíces y el método de la secante, eso es debido a que el método de localización de raíces está implementado de forma que no muestre la iteraciones, por contra el método de la secante si lo está pero no está representado ya que obtiene una aproximación con una sola iteración y la representación sería un simple punto.

Respecto a los métodos que si están representados podemos observar que los métodos bisección y punto fijo van dando saltos continuamente, que los algoritmos bisección y regula falsi no necesitan una primera aproximación y el resto si, que las técnicas Newton, Aitken y Steffensen tienen una gráfica muy parecida pero que es Steffensen el primero en converger, mientras que Aitken y Newton obtiene resultados muy próximos pero a mayor iteración.

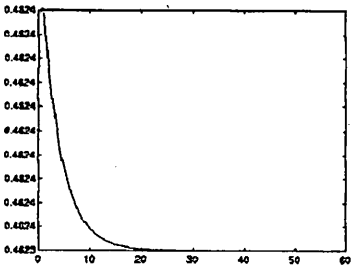
Investigación:

Creemos que resultaría más interesante investigar sobre los métodos anteriormente visto, que seguir poniendo ejemplo y sacar una vez tras otras las mismas conclusiones, nuestra investigación se dirigirá hacia los métodos de Aitken y Steffensen, sabemos que son muy eficientes ¿pero existe alguna manera de optimizarlo?, recordemos que Steffensen se basaba en Aitken y este a su vez en una sucesión de aproximaciones que se obtenía con el punto fijo, pero por qué no obtener esa sucesión de aproximaciones con newton en vez de hacerlo con el punto fijo, de esa manera, ¿podríamos obtener mejores resultados?

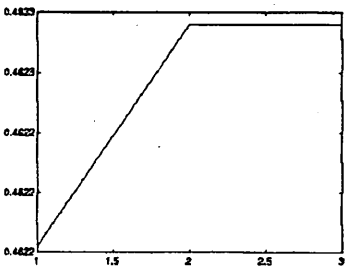
Utilizando el ejemplo anterior obtenemos los siguientes resultados:

Con tolerancia=10^{-10} y p0=0.5	Solución	Iteraciones	Operaciones
Aitken usando punto fijo	0.46234002722272	59	1659
Aitken usando Newton	0.46234002719881	3	135
Con tolerancia=10^{-7} y p0=0.5	Solución	Iteraciones	Operaciones
Steffesen usando Aitken con punto fijo	0.46234002719881	3	82
Steffesen usando Aitken con Newton	0.46234002719882	3	148

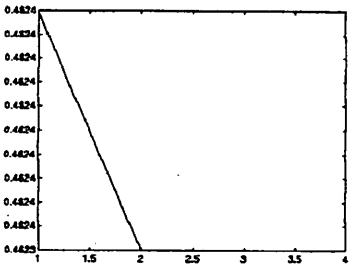
La evolucion de las iteraciones son las siguientes:



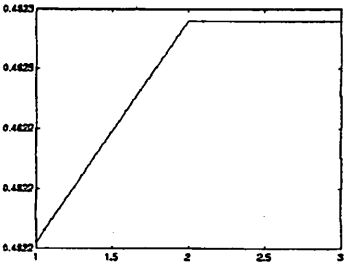
Aitken con punto fijo



Aitken con Newton



Steffensen con punto fijo



Steffensen con Newton

Es fácil observar que Aitken utilizando Newton es mucho más eficiente ya que obtenemos mejor aproximación en menores iteraciones y con un menor número de operaciones, que utilizandolo con el punto fijo, no ocurre lo mismo con Steffensen, eso es debido a lo realmente eficiente que es este método.

Para concluir esta primera práctica creemos que resultaría conveniente mostrar las ventajas y desventajas de los métodos estudiados hasta ahora.

METODOS	VENTAJAS	DESVENTAJAS
Bisección.	-Muy rentable para obtener una primera aproximación.	-Muy lento. -Necesita especificarle un intervalo que contenga una raiz.
Regula falsi.	-Rentable para obtener una primera aproximación.	-Muchas iteraciones. -Muchas operaciones. -Necesita especificarle un intervalo que contenga una raiz.
Localización aprox.de raíces.	-No encontramos ventajas.	-Es muy lento. -Muchas iteraciones. -Muchas operaciones. -Es muy deficiente.
Punto fijo.	-Converge.	-Elección de p0. -Necesita de g(x). -Rápido.
Newton-Rapshon.	-Converge cuadráticamente.	-Elección de p0. -Rápido.
Secante.	-Converge casi cuadrácicamente.	-Elección de p0 y p1. -Rápido.
Aitken con punto fijo.	-Converge superlineal.	-Elección de p0. -Elección de g(x). -Rápido.
Aitken con Newton-Rapshon.	-Converge cuadrática.	-Elección de p0. -Muy eficiente.
Steffensen.	-Converge cuadrática.	-Elección de p0. -Elección de g(x). -Muy eficiente.

Práctica 2 Resolución de sistemas de ecuaciones.

2.1 Introducción teórica.

Si la practica anterior se nos hizo un mundo esta se nos hará un universo, ya que en esta práctica estudiaremos los distintos métodos que existen para resolver un sistema de ecuación, para comprender mejor esta práctica creemos que es necesario estudiar de forma independiente tanto los sistemas de ecuaciones lineales como los sistemas de ecuaciones no lineales.

Recordemos que antes teniamos una ecuación a resolver:

$$a_1x^n+a_2x^{n-1}+...+ a_nx^0 = 0$$

Siendo la solucion aquellos valores de x donde se anulaba la función. Pero ahora tenemos un sistemas de ecuaciones:

$$a_{11}x_1+a_{12}x_2+...+a_{1n}x_n= b_1$$
$$a_{21}x_1+a_{22}x_2+...+a_{2n}x_n= b_2$$

$$a_{n1}x_1+a_{n2}x_2+...+a_{nn}x_n= b_n$$

Donde la solución del sistema es el punto donde cortan todas las ecuaciones que constituyen nuestro sistema. En esta práctica no realizaremos análisis gráfico alguno, ya que la implementación de este algoritmo es compleja y la información que aporta carece de gran importancia.

2.2 Resolución de sist. de ec. lineales.

Para la resolución de sistemas de ecuaciones lineales analizaremos los siguientes métodos:

- Triangularización superior seguido de sustitución regresiva.
- Factorización LU.
- Método de Jacobi.
- Método de Gauss-Seidel.

Para una mejor compresión y comparación de los métodos nos basaremos en el siguiente ejemplo:

$$5x+y+z=5$$
$$x+4y+z=4$$
$$x+y+3z=3$$

Escogemos este ejemplo porque ya conocemos la solución (x=0'76,y=0'68,z=0'52) de esta manera podemos comprobar que los métodos están correctamente implementados.

2.2.1 Método por triangularización superior seguido de sustitución regresiva.

Dado el siguiente sistema de ecuaciones lineales:

$$a_{11}x_1+a_{12}x_2+...+a_{1n}x_n= b_1$$
$$a_{21}x_1+a_{22}x_2+...+a_{2n}x_n= b_2$$
$$\vdots$$
$$a_{n1}x_1+a_{n2}x_2+...+a_{nn}x_n= b_n$$

Podemos expresarlo de forma matricial:

$$\begin{vmatrix} a_{11} & a_{12} \dots & a_{1n} \\ a_{21} & a_{22} \dots & a_{2n} \\ & \vdots & \\ a_{n1} & a_{n2} \dots & a_{nn} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{vmatrix} = \begin{vmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{vmatrix}$$
$$A \qquad X \qquad B$$

De forma que si conseguimos, mediante operaciones elementales (sumar filas, restar filas, etc.), que la matriz ampliada [A|B] sea triangular superior.

$$\left|\begin{array}{cccc|c} a'_{11} & a'_{12} \dots & a'_{1n} & & b'_1 \\ 0 & a'_{22} \dots & a'_{2n} & & b'_2 \\ & & \vdots & & \\ 0 & 0 \dots & a'_{nn} & & b'_n \end{array}\right|$$

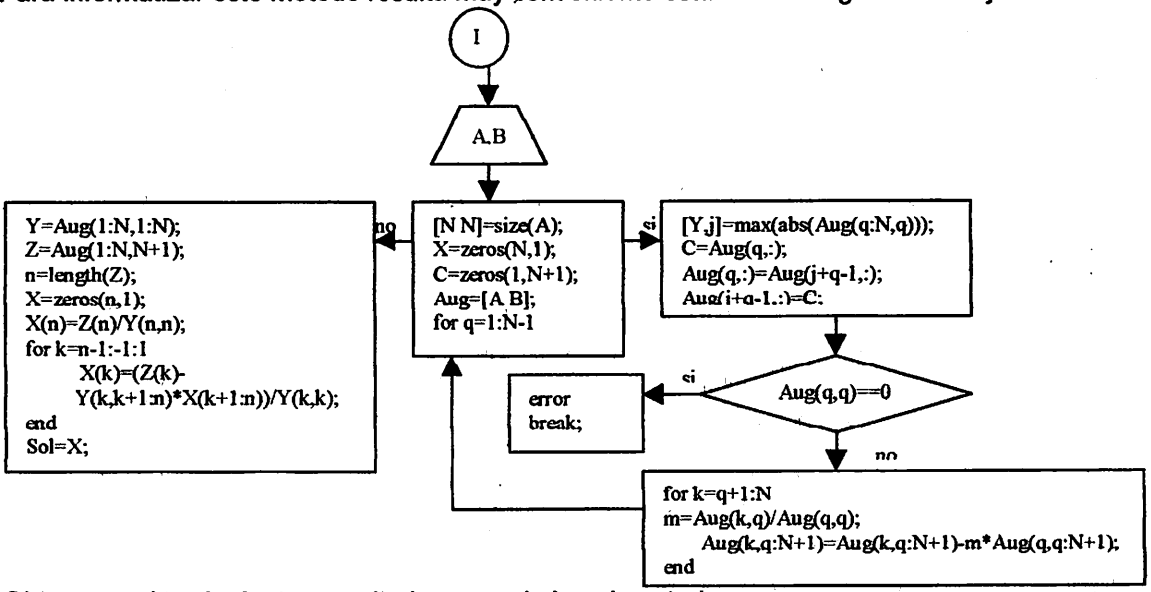
Mediante la sustitución regresiva obtenemos la solución del sistema.

$$\begin{vmatrix} a'_{11} & a'_{12} \dots & a'_{1n} \\ 0 & a'_{22} \dots & a'_{2n} \\ & \vdots & \\ 0 & 0 \dots & a'_{nn} \end{vmatrix} \cdot \begin{vmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{vmatrix} = \begin{vmatrix} b'_1 \\ b'_2 \\ \vdots \\ b'_n \end{vmatrix}$$

$$a'_{11}x_1+a'_{12}x_2 \dots +a'_{1n}x_n=b'_1$$
$$0 \quad a'_{22}x_2 \dots +a'_{2n}x_n=b'_2$$
$$\vdots \qquad \vdots \qquad \vdots$$
$$0 \quad 0 \dots +a'_{nn}x_n=b'_n$$

Como a'_{nn} b'_n son constantes obtenemos el valor x_n sustituyéndolo en la ecuación superior obtenemos x_{n-1}, así sucesivamente vamos obteniendo los valores de todas las variables.

Para informatizar este metodo resulta muy conveniente estudiar su diagrama de flujo:



Obtenemos los siguientes resultados, con el ejemplo anterior:

x=0.7600
y=0.6800
z=0.5200

2.2.2 Método de factorización LU.

Cuando la matriz A es invertible, es decir, admite una factorización LU o factorización triangular podemos expresarla de la siguiente manera: A=LU, siendo L una matriz triangular inferior, donde los elementos diagonales son todos 1, y U una matriz triangular superior, de forma matricial quedaría:

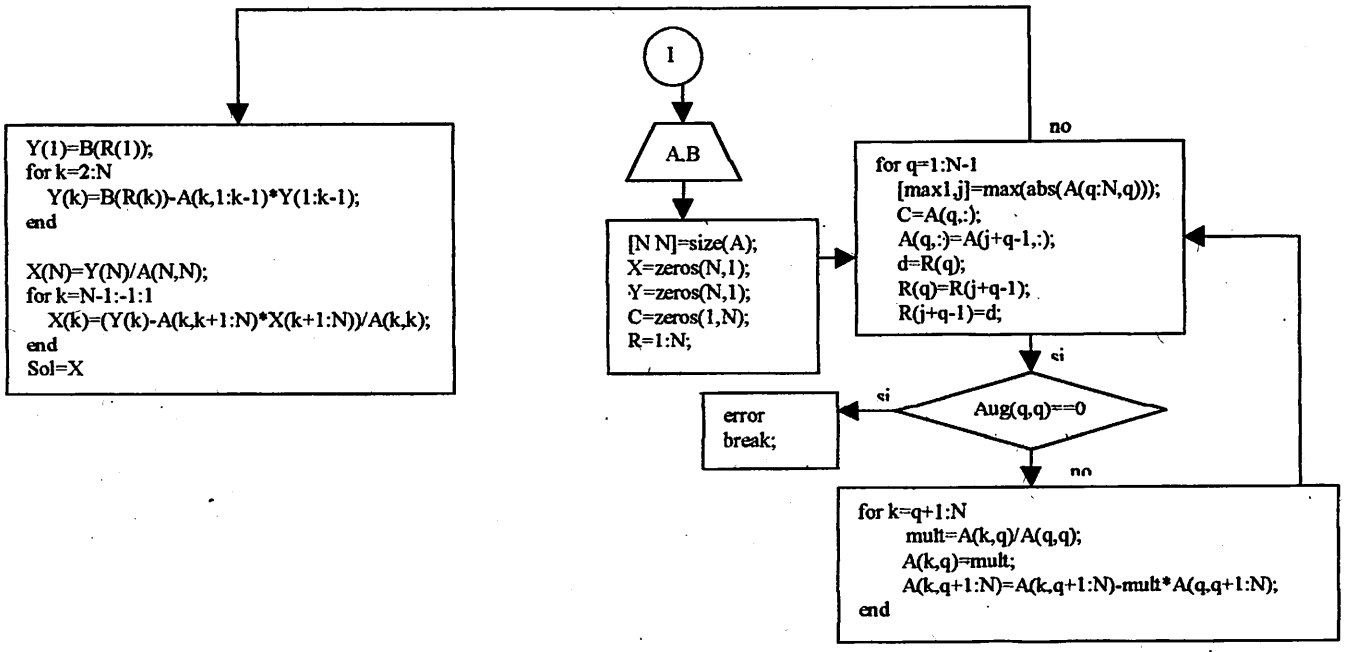
$$\begin{vmatrix} a_{11} & a_{12} \dots & a_{1n} \\ a_{21} & a_{22} \dots & a_{2n} \\ \vdots & & \\ a_{n1} & a_{n2} \dots & a_{nn} \end{vmatrix} = \begin{vmatrix} 1 & 0 \dots & 0 \\ m_{11} & 1 \dots & 0 \\ \vdots & & \\ m_{n1} & m_{n2} \dots & 1 \end{vmatrix} \begin{vmatrix} u_{11} & u_{12} \dots & u_{1n} \\ 0 & u_{22} \dots & u_{2n} \\ \vdots & & \\ 0 & 0 \dots & u_{nn} \end{vmatrix}$$

De forma que nuestro sistema de ecuaciones lineales puede quedar de la siguiente manera: AX=B ->LUX=B. Definiendo Y=UX y resolviendo el sistema LY=B podemos resolver a continuación el sistema UX=Y, para comprenderlo mejor lo veremos en forma desarrollada:

$$\begin{vmatrix} y_1 = b_1 \\ m_{21}y_1+m_{22}y_2 = b_2 \\ \vdots \\ m_{n1}y_1+m_{n2}y_2+\dots+m_{nn}y_{n-1}+y_n = b_n \end{vmatrix}$$

$$\begin{vmatrix} u_{11}x_1+u_{12}x_2+\dots+u_{1n}x_n=y_1 \\ u_{22}x_2+\dots+u_{2n}x_n=y_2 \\ \vdots \\ u_{nn}x_n=y_n \end{vmatrix}$$

Para una mejor comprensión a la hora de informatizar este método estudiaremos su diagrama de flujo:



Obtenemos los siguientes resultados:
x=0.7600
y=0.6800
z=0.5200

2.2.3 Método iterativo de Jacobi.

Dado el siguiente sistema de ecuaciones lineales:

$$a_{11}x+a_{12}y+...+a_{1n}z= b_1$$
$$a_{21}x+a_{22}y+...+a_{2n}z= b_2$$
$$\vdots$$
$$a_{n1}x+a_{n2}y+...+a_{nn}z= b_n$$

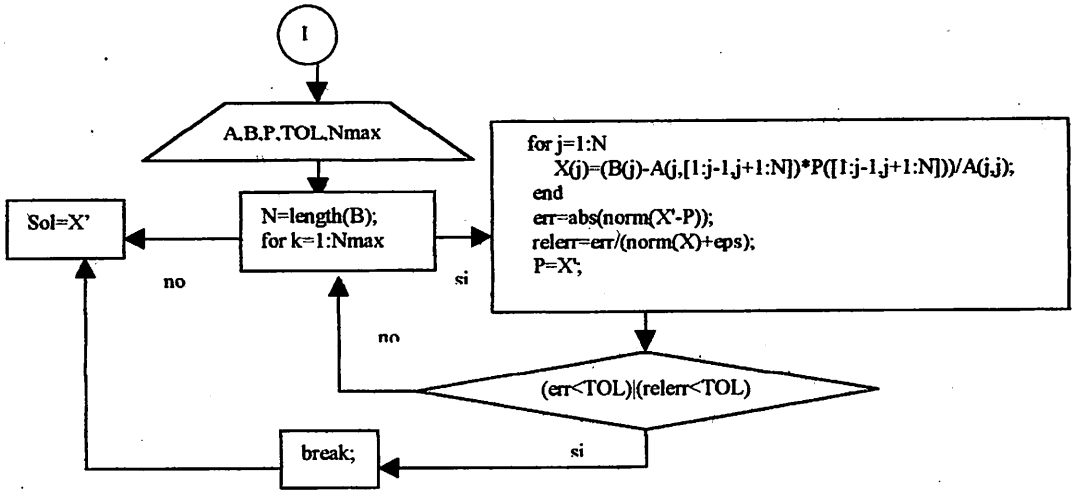
Lo podemos expresar de la siguiente manera:

$$x=(b_1-a_{12}y-...-a_{1n}z)/a_{11}$$
$$y=(b_2-a_{21}x-...-a_{2n}z)/a_{22}$$
$$\vdots$$
$$z=(b_n-a_{n1}x-a_{n2}y-...-a_{nn-1}t)/a_{nn}$$

Esta es la base del siguiente proceso iterativo:

$$x_{k+1}=(b_1-a_{12}y_k-...-a_{1n}z_k)/a_{11}$$
$$y_{k+1}=(b_2-a_{21}x_k-...-a_{2n}z_k)/a_{22}$$
$$\vdots$$
$$z_{k+1}=(b_n-a_{n1}x_k-a_{n2}y_k-...-a_{nn-1}t_k)/a_{nn}$$

Un proceso iterativo es aquel que obtiene la solución a partir de continuas iteraciones, de forma que x_{k+1} está más próximo a la solución que x_1, a este método iterativo se le llama método de Jacobi, y su diagrama de flujo es el siguiente:



Obtenemos los siguientes resultados:

x=0.7600
y=0.6800
z=0.5200

2.2.4 Método iterativo Gauss-Seidel.

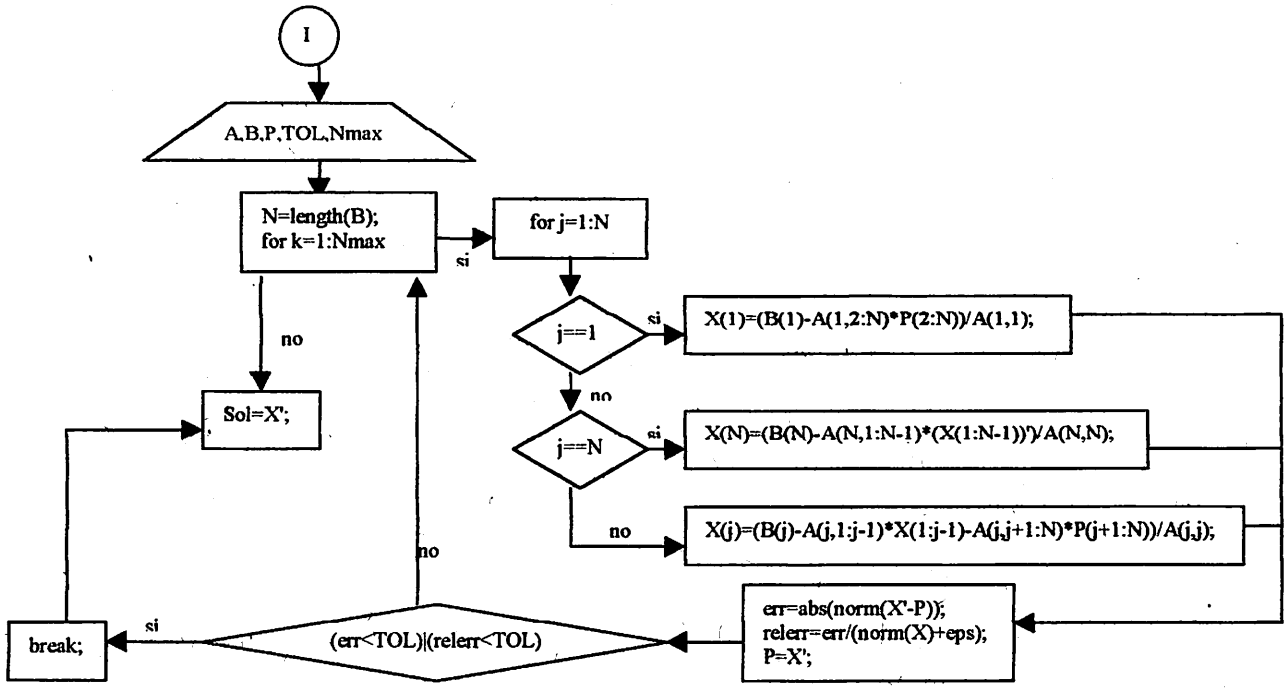
El método Gauss-Seidel es una optimización del método iterativo de Jacobi, recordemos que la base del método de Jacobi era la siguiente:

$$x_{k+1}=(b_1-a_{12}y_k-\ldots-a_{1n}z_k)/a_{11}$$
$$y_{k+1}=(b_2-a_{21}x_k-\ldots-a_{2n}z_k)/a_{22}$$
$$\vdots$$
$$z_{k+1}=(b_n-a_{n1}x_k-a_{n2}y_k-\ldots-a_{nn-1}t_k)/a_{nn}$$

Probablemente x_{k+1} converga más rápido que x_k, entonces por qué no utilizamos x_{k+1} en vez de utilizar x_k para calcular y_{k+1} de esa manera conseguiremos más eficiencia, esta es la idea fundamental donde se basa el método de Gauss-Seidel.

$$x_{k+1}=(b_1-a_{12}y_k-\ldots-a_{1n}z_k)/a_{11}$$
$$y_{k+1}=(b_2-a_{21}x_{k+1}-\ldots-a_{2n}z_k)/a_{22}$$
$$\vdots$$
$$z_{k+1}=(b_n-a_{n1}x_{k+1}-a_{n2}y_{k+1}-\ldots-a_{nn-1}t_{k+1})/a_{nn}$$

Para tenerlo más claro a la hora de informatizar este método, mostramos a continuación su diagrama de flujo:



Obtenemos los siguientes resultados:
x=0.7600
y=0.6800
z=0.5200

2.2.5 Comparaciones y conclusiones.

METODOS	VENTAJAS	DESVENTAJAS
Triangularización superior	-Sencillo de implementar. -Fácil comprensión.	-Necesita A y B.Matrices que determina al sistema.
Factorización LU	-Resulta más complicado de implementar que el anterior. -Fácil comprensión.	-Necesita A y B.
Jacobi	-Es muy eficiente. -Es iteractivo. -Fácil comprensión.	-Requiere muchas operaciones para obtener un resultado muy próximo. -Necesita A, B y p0.
Gauss-Seidel	-Es el más eficiente. -Es iteractivo. -Fácil comprensión.	-Necesita A, B y p0.

En esta práctica lo realmente importante son los métodos que existen para resolver sistemas de ecuaciones no lineales ya que resultan más interesantes de estudiar que estos métodos. No obstantes estas técnicas que hemos analizado anteriormente, son técnicas muy eficientes para resolver sistemas de ecuaciones lineales.

2.2.6 Ejemplos.

1.-Dado la siguiente sistema de ecuaciones lineales resolverlo por los métodos anteriormente visto.

4x-y=15
x+5y=9

	x	y
Triangularización superior	4	1
Factorización LU	4	1
Jacobi	3.99999843750000	1.00000500000000
Gauss-Seidel	3.99999843750000	1.00000031250000

2.-Dado la siguiente sistema de ecuaciones lineales resolverlo por los métodos anteriormente visto.

2x+8y-z=11
5x-y+z=10
-x+y+4z=3

	x	y	z
Triangularización superior	2.00000000000000	1.00000000000000	1.00000000000000
Factorización LU	-Inf	-Inf	Inf
Jacobi	9.08703957294244	-0.99500863769602	0.52819015475198
Gauss-Seidel	-0.99613445627197	-5.02937897619247	1.00831112998012

Los resultados que obtengo en el método de factorización LU son debido a que se produce una división por cero, mientras que en los métodos Jacobi y Gauss-Seidel los resultados que obtengo no son los correctos el principal motivo de esta cause es que se ha tomado un mal p0.

2.3 Resolución de sist. de ec. no lineales.

Esta es la parte más interesante de esta segunda práctica, en esta sección estudiaremos los siguientes métodos para la resolución de sistemas de ecuaciones no lineales:

- Método del punto fijo para sistemas.
- Método de Newton para sistemas.
- Método de Seidel.
- Método de Broyden.
- Método del máximo descenso.
- Método del Newton global.

Como hemos hecho hasta ahora vamos a introducirnos en cada método, expondremos su diagrama de flujo, para que la implementación sea lo más sencilla posible y para finalizar mostraremos los resultado obtenidos con el siguiente sistema de ecuaciones no lineales:

$x^2-2x-y+0.5=0$ Es una parábola.
$x^2+4y^2-4=0$ Es una elipse.

Como hicimos en la sección anterior escogemos este ejemplo porque ya conocemos la solución, se cortan en los puntos (-0.2,1.0) y (1.9,0.3) aunque nuestro estudio será orientado a obtener una aproximación al primer punto de corte, de esta manera podemos comprobar que los métodos están correctamente implementados.

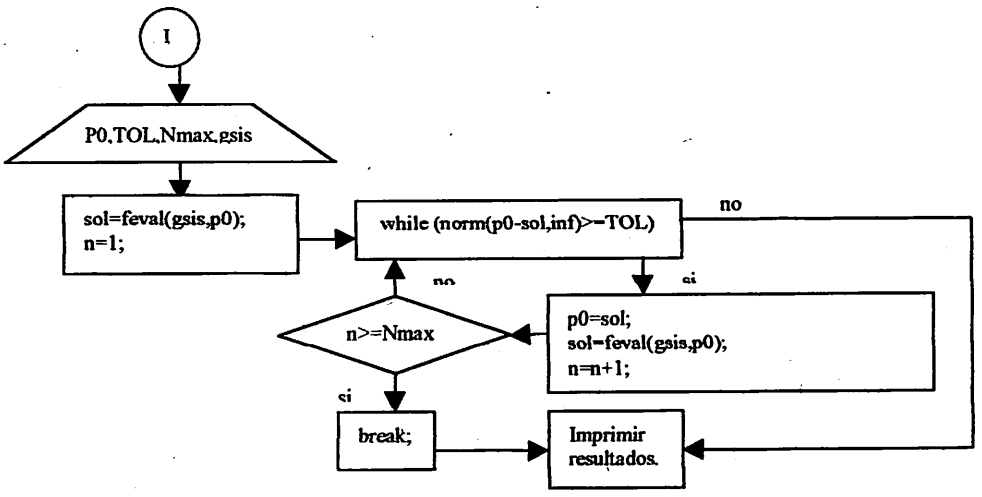
2.3.1 Método del punto fijo.

Se basa en la misma idea que en la práctica anterior, lo que sucede es que ahora tenemos un vector de incognitas en vez de una sola incognita y la solución será ahora un vector de forma esquemática queda:

$$\begin{array}{lcl} f_1(x_1,\dots,x_n) & & x_1=g_1(x_1,\dots,x_n) \\ f_2(x_1,\dots,x_n) & = & x_2=g_2(x_1,\dots,x_n) \\ \vdots & & \vdots \\ f_n(x_1,\dots,x_n) & & x_n=g_n(x_1,\dots,x_n) \end{array}$$

$$p_0 \in R^n \rightarrow p_1=g(p_0) \rightarrow p/f(p)=0$$

Para comprender mejor este método mostraremos a continuación su diagrama de flujo:



Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	5	74	(-0.22205699037635,0.99381941082883)
10^{-5}	8	119	(-0.22221447747710,0.99380866003302)
10^{-7}	11	164	(-0.22221455741294,0.99380842009442)
10^{-10}	15	224	(-0.22221455505876,0.99380841859887)

Como principal conclusión que obtenemos es que el método del punto fijo es un metodo iterativo muy eficiente pero con el gran inconveniente de tener que elegir una g(x), en nuestro caso un sistema g(x), además está muy influido por la primera aproximación que determina en gran parte el éxito o fracaso de los resultados obtenidos.

2.3.2 Método de Newton.

Si hacemos un pequeño ejercicio de memoria, podremos recordar que en la primera práctica además de utilizar el método del punto fijo existía otro método que optimizaba el punto fijo ya que encontraba la "g(x) ideal", este método era el método de Newton-Rapshon y si anteriormente aplicamos el método del punto fijo para sistemas por qué no hacerlo con el método de Newton-Rapshon. De forma esquemática quedaría:

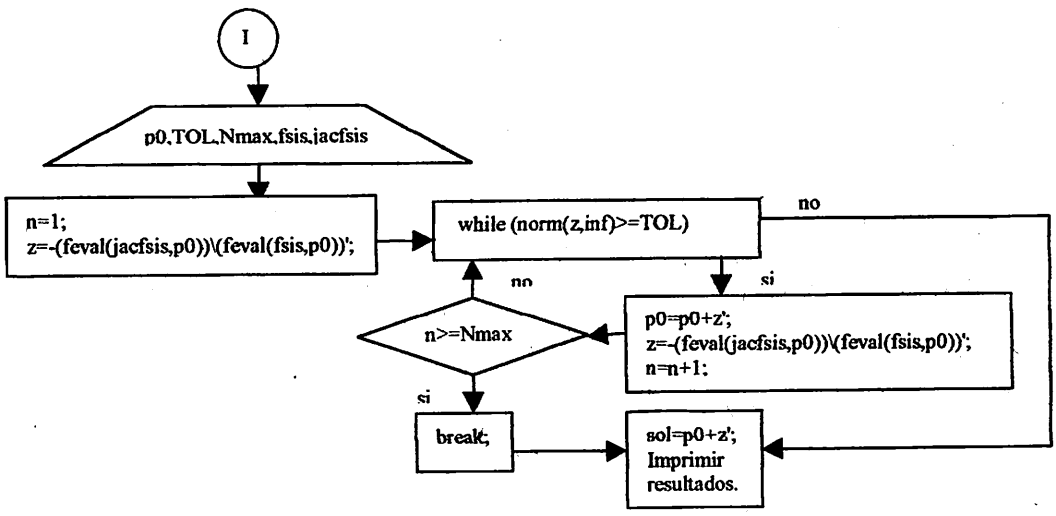
$$p_0 \approx p$$
$$Df(p_n)\, Z_n = -f(p_n)$$
$$p_{n+1} = p_n + z_n$$

Donde p es la solución, p_0 es la primera aproximación y $Df(p_n)$ es la matriz jacobiana o diferencial:

$$Df(p_n)=\begin{vmatrix} \text{Derivada de } 1^a \text{ ec. con respecto a la } 1^a \text{ incog.} & \dots & \text{Derivada de } 1^a \text{ ec. con respecto a la } n^a \text{ incog.} \\ \vdots & & \\ \text{Derivada de } n^a \text{ ec. con respecto a la } 1^a \text{ incog.} & \dots & \text{Derivada de } n^a \text{ ec. con respecto a la } n^a \text{ incog.} \end{vmatrix}$$

$$\begin{vmatrix} p_0 \in R^n \\ p_{n+1} = p_n - (Df(p_n))^{-1} \cdot f(p_n) \qquad z_n = Df(p_n) \backslash f(p_n) \\ p_{n+1} = p_n - z_n \end{vmatrix}$$

Para que sea más fácil la compresión, mostramos a continuación el diagrama de flujo:



Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	7	378	(1.90067672669776,0.31121856535121)
10^{-5}	8	437	(1.90067672636707,0.31121856541929)
10^{-7}	8	437	(1.90067672636707,0.31121856541929)
10^{-10}	8	437	(1.90067672636707,0.31121856541929)

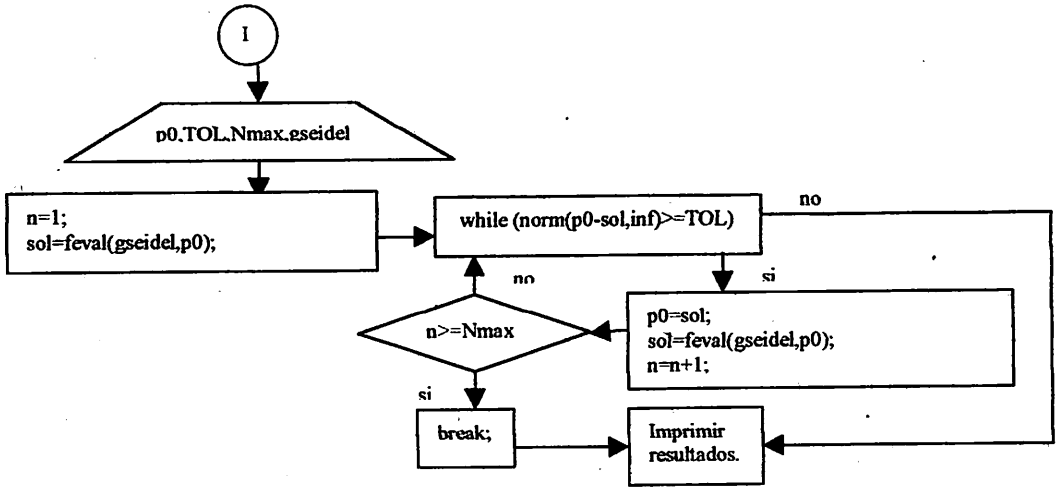
Como podemos observar obtenemos una aproximación más cercana del segundo punto que del primer punto, el principal motivo de este hecho es la primera aproximación que le aportamos.

2.3.3 Método de Seidel.

En la primera sección de esta práctica hemos estudiado el método de Gauss-Seidel, el método de Seidel para sistemas de ecuaciones no lineales se basa en la misma idea, utilizar x_{k+1} para calcular x_{k+2} ,de esta forma conseguimos que converga más rápido que el método del punto fijo, de forma esquemática quedaría:

$$x_1=g_1(x_0,y_0)$$
$$x_2=g_2(x_1,y_0)$$
$$\vdots$$
$$x_n=g_n(x_{n-1},y_0)$$

Su diagrama de flujo sería:



Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	5	74	(-0.22232719087891,0.99380231098083)
10^{-5}	8	119	(-0.22221280911430,0.99380851324262)
10^{-7}	12	179	(-0.22221454831373,0.99380841896552)
10^{-10}	17	254	(-0.22221455506622,0.99380841859948)

Como podemos observar obtenemos una mejor aproximación al primer punto que el método del punto fijo, con menor número de operaciones e iteraciones. Además es muy sencillo de implementar pero su gran inconveniente es uqe depende de gran medida la g(x) que tomemos y la primera aproximación que indiquemos.

2.3.4 Método de Broyden.

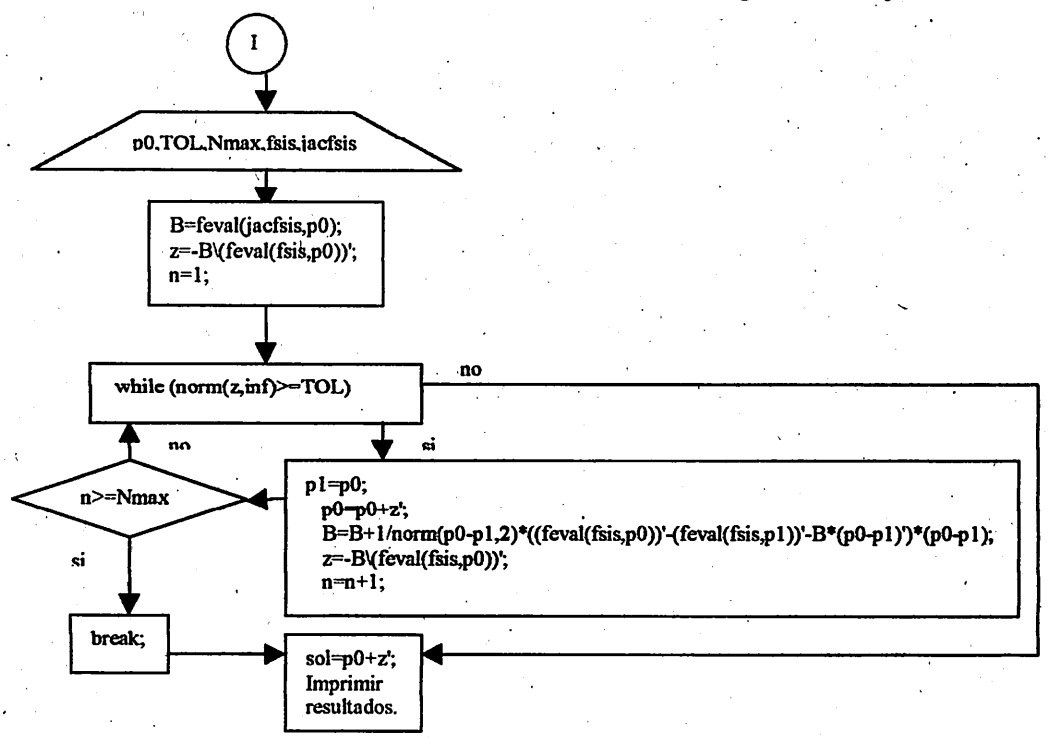
El método de Broyden o tambien conocido como quasi-Newton, pretende optimizar el método de Newton, intentando evitar calcular tantas veces la diferencial, los pasos en que podemos dividr este método son los siguientes:

- Tomar $p_0 \approx p$
- Definir $A_0=Df(p_0)$
- Definir $p_1=p_0-A_0^{-1}(f(p_0))$ (Primera iteración de Newton.) En Matlab: $p_1=p_0-A_0\backslash$
- Para n>1

$$A_n = A_{n-1} + \frac{[f(p_n)-f(p_{n-1})-A_{n-1}(p_n-p_{n-1})][p_n-p_{n-1}]'}{[[p_n-p_{n-1}]]_2}$$

- Resolver $A_n z_n=-f(p_n)$ En Matlab:$z_n=A_n\backslash f(p_n)$
- $p_{n+1}=p_n+z_n$

Para comprenderlo mejor, mostramos a continuación su diagrama de flujo:



Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	16	1764	(1.90119428120588,0.31045506158076)
10^{-5}	24	2692	(1.90068248331377,0.31121034936584)
10^{-7}	32	3620	(1.90067678849615,0.31121847682049)
10^{-10}	44	5012	(1.90067672643665,0.31121856532007)

Como podemos observar el métod de Broyden es un claro ejemplo de "quiero pero no puedo" ya que no consigue optimizar ni el número de iteraciones, ni el número de operaciones, ni tampoco obtiene una mejor aproximación que el método de Newton.

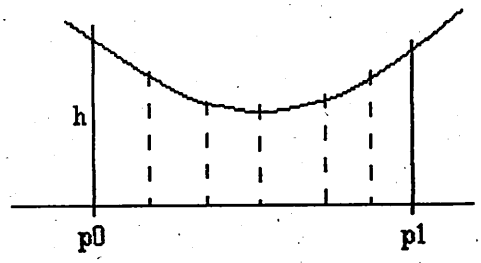
2.3.5 Método de máximo descenso.

Dado el siguiente sistema de ecuaciones no lineales:

$f(x,y)=0$

$g(x,y)=0$

Podemos obtener la altura del sistema: $h:R^2 \rightarrow R$ siendo $h(x,y)=f^2(x,y)+g^2(x,y)$, de forma que cuando $h(x,y)\approx 0$ que implica que $f(x,y)\approx 0$ y $g(x,y)\approx 0$, por lo tanto, cuando $h(x,y)$ sea minimo e igual a cero indicaría que x e y son solución del sistema.De manera gráfica:



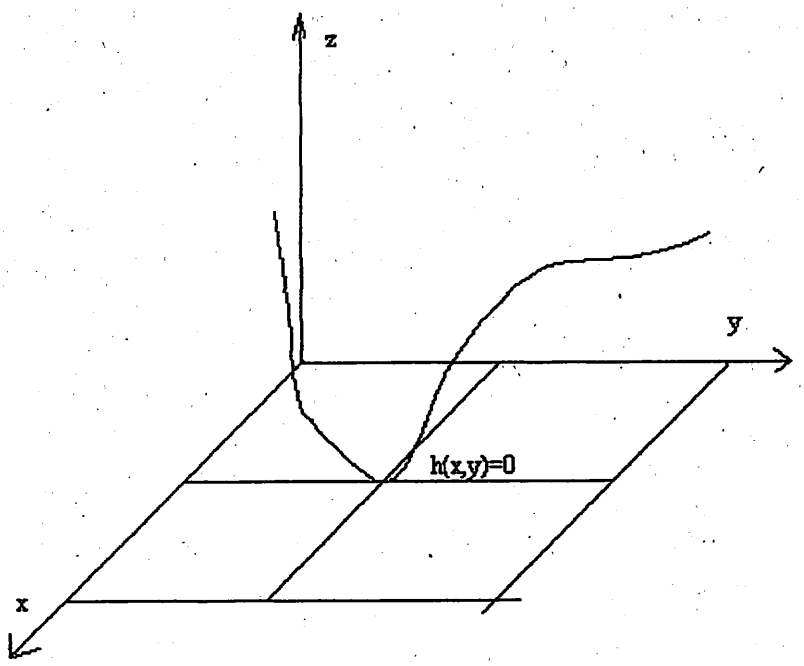
Ya sabemos que encontraremos una solución cuando $h(x,y)$ sea minimo e igual a cero, pero cuál es la dirección que debemos tomar para que $h(x,y)=0$, la dirección de máximo ascenso es $\nabla h(x,y)$,gradiente de $h(x,y)$, por lo tanto la dirección de máximo descenso será $-\nabla h(x,y)$, partiendo de p_0 nos dirigiremos a la mejor aproximación $p_0-\lambda\nabla h(p_0)$.

$$\nabla h(x,y)=(h_x(x,y);\ h_y(x,y))$$

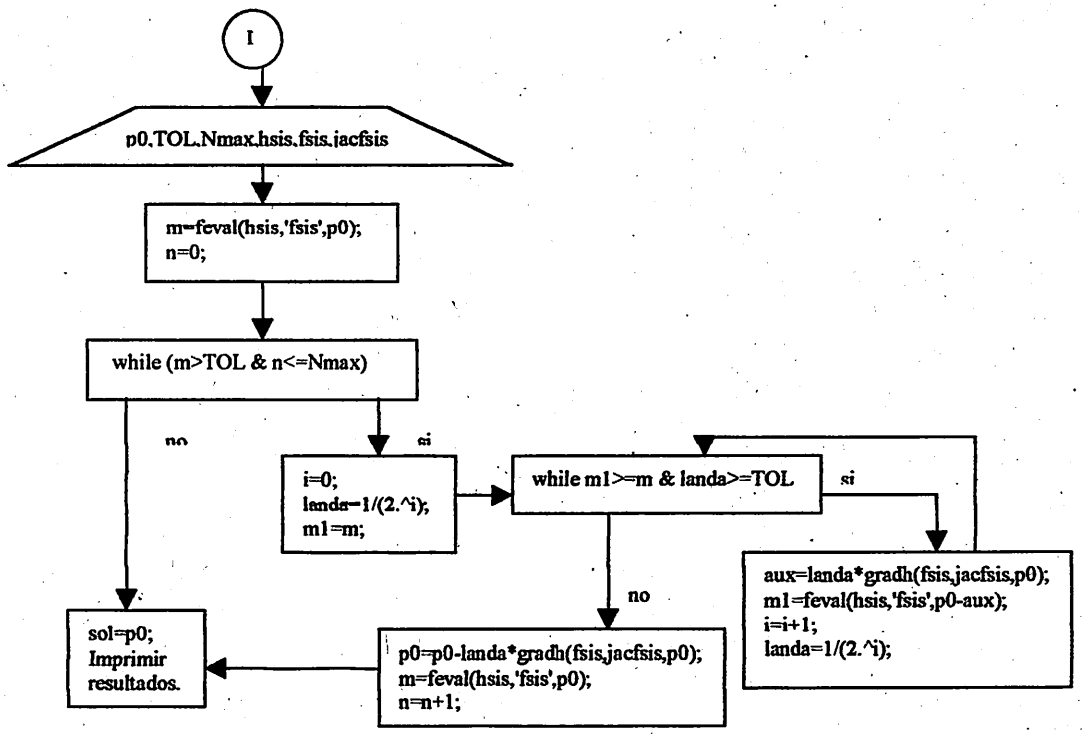
$h_x(x,y)$=Derivadas parciales respecto a x.

$h_y(x,y)$=Derivadas parciales respecto a y.

De manera gráfica:



Para que sea más sencillo la implementación en el ordenador del método de máximo descenso ofrecemos a continuación el diagrama de flujo:



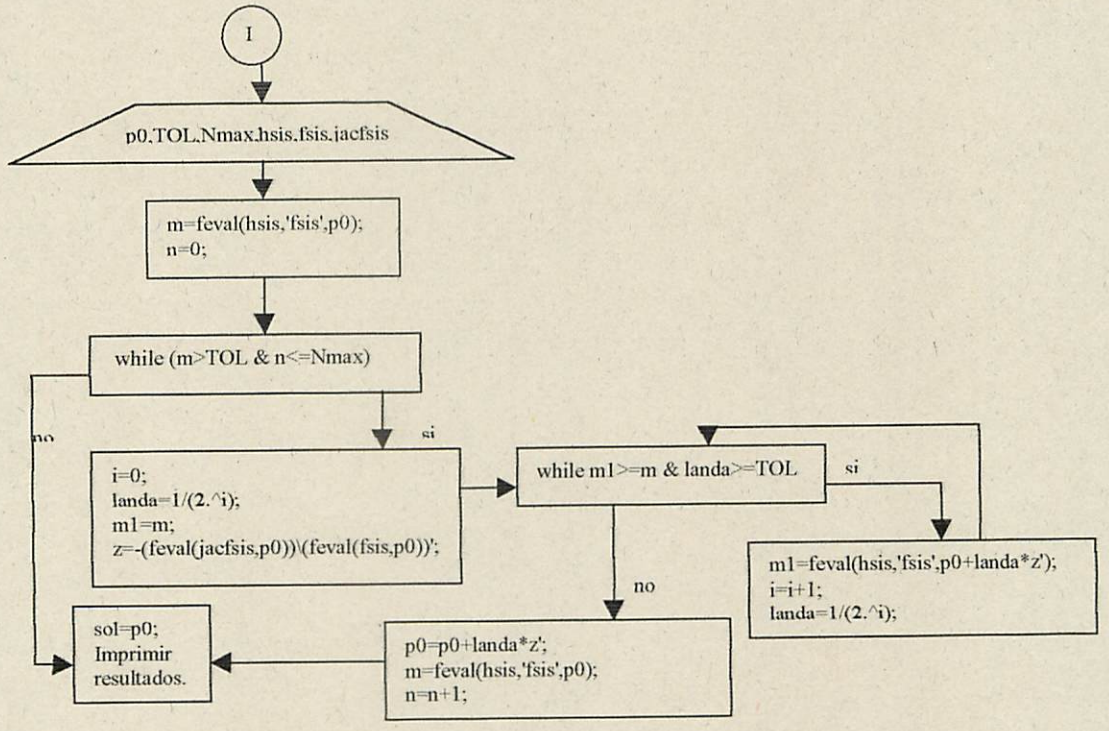
Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	76	31660	(-0.21299257866394,0.99224900875499)
10^{-5}	El método no converge para 500 it. Sol: (0.93866006054550,0.87203530689861)		
10^{-7}	El método no converge para 500 it. Sol: (0.93752047754919,0.84400942528485)		
10^{-10}	El método no converge para 500 it. Sol: (0.93750002009406,0.84375025366345)		

A partir de la torelancia 10^{-5} el método no converge, apesar que aumentemos el número máximo de iteraciones, el resultado que mostramos es la última aproximación como podemos observar está muy lejos del resultado correcto, y es que este método además de ser muy lento requiere de muchas operaciones para alcanzar una solución aceptable.

2.3.6 Método de Newton global.

El método de Newton global se basa en la misma idea que el método de máximo descenso, la principal diferencia es que este método no toma la dirección de máximo descenso sino que toma la dirección de Newton $z_0=Df(p_0)\backslash f(p_0)$. Para comprenderlo mejor ofrecemos a continuación el diagrama de flujo:



Los resultados que obtenemos, con un p0=[1 1], son:

Tolerancia	N º de iteraciones	N º de oper.	Aproximación
10^{-3}	8	808	(1.90050270750661,0.32326420057654)
10^{-5}	15	1466	(1.90067645262974,0.31131462731114)
10^{-7}	12	1184	(1.90067403888314,0.31198616424034)
10^{-10}	20	1936	(1.90067671808248,0.31122156783925)

Los resultados que obtenemos son para el segundo punto de corte, el motivo principal por que obtenemos esta aproximación y no la del primer corte puede venir determinado por: elección del p0, implementación de hsis o de jacfsis, ya que podría influir en la solución. Para solucionar esto tendremos que tomar p0 más próximo a la primera solución que a la del segundo de esa manera evitaremos en cierto modo que nuestro algoritmo muestre la información que no deseamos.

2.3.7 Comparaciones y conclusiones.

METODOS	VENTAJAS	DESVENTAJAS
Punto fijo	-Fácil de implementar y de comprender. -Obtenemos una buena aproximación con muy pocas iteraciones.	-Necesita p0. -Depende en gran medida la elección de la g(x).(Siendo g(x) el sistema donde en cada ecuación se despeja una incognita).
Newton	-Fácil de implementar y de comprender.	-Realiza muchos cálculos. -Necesita p0, que condiciona en gran mediada la solución obtenida.
Seidel	-Fácil de implementar y de comprender.	-Necesita p0. -Necesita g(x), pero distinta a la del punto fijo, por lo tanto, tenemos que volver a implementar un nuevo sistema.
Broyden	-Fácil de comprender.	-Necesita p0. -No consigue lo que se pretende, optimizar los cálculos.
Máximo descenso	-No se ha encontrado ventajas.	-Dificil su comprensión. -Dificil de implementar. -Realiza muchos cálculos para después obtener una pesima aproximación. -Necesita p0. -No resulta rentable utilizarlo.
Newton Global	-Se basa en una idea más fácil de interpretar que el método de máximo descenso.	-Necesita p0. -Realiza muchos cálculos. -Depende del p0 obtendremos una solución u otra.

Como conclusión final, es que los métodos que se basan en ideas más sencillas son los que a la postre son más fáciles de implementar, y si además obtenemos resultados aceptables, resultan idóneos para nuestro problema, resolver un sistema de ecuaciones no lineales, estos métodos son: el método del punto fijo, el de newton, el de seidel, el de Broyden e incluso el método de Newton global, a excepción del método del máximo descenso que no llegamos a comprender que sea tan ineficiente, esperemos que haya sido un error de implementación y que lo resolvamos antes del exámen, porque tanta ineficiencia es inexplicable.

2.3.8 Ejemplos.

1.-Dado el siguiente sistema de ecuaciones no lineales resolverlo por los métodos anteriormente estudiado.(TOL=10^{-10}, iteraciones máximas 500, p0=[0.1,0.1,–0.1]).

$$3x-\cos(yz)-1/2=0$$
$$x^2-81(y+0.1)^2+\operatorname{sen}(z)+1.06=0$$
$$e^{-xy}+20z+(10\pi-3)/3=0$$

Métodos	Iter.	Op.	Aprox. x	Aprox. y	Aprox. z
Punto fijo	8	207	0.50000000000000	0.00000000000006	-0.52359877559775
Newton	8	1087	0.48912719903498	-0.00066695945166	-0.52361508965963
Seidel	5	129	0.50000000000000	0.00000000000007	-0.52359877559830
Broyden	29	6855	0.48912719903795	-0.00066695939024	-0.52361508965696
Máximo desc.	223	201937	0.48912405477440	-0.00066710544554	-0.52361523485688
Newton global	20	4025	0.48912725295282	-0.00066675803874	-0.52361467939014

Como podemos deducir los métodos del punto fijo, Newton y Seidel son de los más eficientes, ya que obtienen una buena aproximación con iteraciones pequeñas, también podemos observar que a diferencia de la resolución de ecuaciones el método de Newton realiza mayor operaciones que el método del punto fijo, el principal motivo de este hecho es que Newton utiliza la matriz jacobiana para encontrar una solución al sistema con lo cual utiliza un mayor número de operaciones.

Del resto de los métodos podemos confirmar lo que se preveía,que el método de Broyden quiere pero no puede, que los métodos de máximo descenso y Newton global resultarían más eficientes para obtener una primera aproximación y no la última como pretendemos en el ejercicio.

A continuación, estudiaremos unos particulares ejemplos.

2.-Dado el siguiente sistema de ecuaciones no lineales resolverlo a partir de los siguientes puntos iniciales: (0,0), (5,5), (-3,3), (-1,-1). ¿Qué sucede? (TOL=10^{-10}, iteraciones máximas 500).

$$x^2y+x^2-y=1$$
$$xy^2+2y^2-4x=8$$

p0=(0,0)	Iter.	Op.	Aprox. x	Aprox. y
Punto fijo	2	5	1	2
Newton	2	73	-2	-1
Seidel	2	5	1	2
Broyden	2	130	-2	-1
Máximo descenso	19	7981	-1.00000078993748	1.99999993743349
Newton global	20	1639	-1.00000000000000	-1.99999846971386

p0=(5,5)	Iter.	Op.	Aprox. x	Aprox. y
Punto fijo	2	5	1	2
Newton	8	507	1	2
Seidel	2	5	1	2
Broyden	77	9798	1.00000000026111	1.99999999976299
Máximo descenso	No conv. x=-2.01704696959879 y=-0.48345117167676			
Newton global	26	2980	1.00000103842608	2.00000045747688

p0=(-3,3)	Iter.	Op.	Aprox. x	Aprox. y
Punto fijo	2	5	1	2
Newton	8	533	-1	2
Seidel	2	5	1	2
Broyden	El método no converge para 500 iteraciones.			
Máximo descenso	18	7391	-1.00000085265381	1.99999999994529
Newton global	23	2688	-1.00000163015978	2.00000002368130

p0=(-1,-1)	Iter.	Op.	Aprox. x	Aprox. Y
Punto fijo	2	5	1	2
Newton	Warning: Matrix is singular to working precision.			
Seidel	2	5	1	2
Broyden	Warning: Matrix is singular to working precision.			
Máximo descenso	No conv. x=-1.60661914270691 y=-1.46279372387135			
Newton global	Warning:Matrix is singular to working precision.			

Una vez comprobado para los distintos puntos iniciales podemos sacar como conclusión:

- Los métodos punto fijo y Seidel obtenemos idénticos resultados para distintos puntos iniciales, debido a que la 'g' del sistema y la 'g' de seidel poseen valores constantes con lo cual agilizan el proceso de resolución del sistema de ecuaciones no lineales.

- El resto de los métodos nos dará una solución u otra dependiendo del punto inicial indicado, de forma que obtenemos como posibles solución los siguientes puntos: (1,2), (-2,-1) y (-1,2). A diferencia de los métodos del punto fijo y de Seidel no estan influidos por la 'g' del sistema o por la 'g' de Seidel.

- En el punto (-1,-1) obtenemos en los métodos: Newton, Broyden y Newton global el siguiente mensaje: Warning:Matrix is singular to working precision. Lo que indica que se trata de un vector 'malo', no muy bueno para trabajar con precisión, el vector se considera no muy correcto para trabajar con él ya que una fila de la matriz jacobiana es nula y podríamos obtener resultados incorrectos.

- Y por último las mismas conclusiones de siempre: lo más eficientes: Newton, punto fijo y Seidel, para obtener primeras aproximaciones: máximo descenso y newton global y sobre el método de Broyden, sin comentario.

3.-Dado el siguiente sistema de ecuaciones no lineales resolverlo utilizando distintos criterios de paros, (norma euclidea y norma infinito) ¿obtenemos idénticos resultados?¿cuál es el criterio de paro más eficiente?(TOL=10^{-10}, iteraciones máximas 500,p0=(0,0,0)).

6x+y+z-1=0
x+4y+z-2=0
x+y+5z=0

Norma Euclidea	Iter.	Op.	Aprox. x	Aprox. y	Aprox. z
Punto fijo	26	623	0.10280373830813	0.50467289718365	-0.12149532711372
Newton	2	237	0.10280373831776	0.50467289719626	-0.12149532710280
Seidel	26	623	0.10280373830813	0.50467289718365	-0.12149532711372
Broyden	2	335	0.10280373831776	0.50467289719626	-0.12149532710280
Máximo descenso	26	10570	0.10280507558240	0.50467195758529	-0.12149388782774
Newton global	18	2732	0.10280334615262	0.50467097202194	-0.12149486363491

Norma infinita	Iter.	Op.	Aprox. x	Aprox. y	Aprox. z
Punto fijo	26	311	0.10280373830813	0.50467289718365	-0.12149532711372
Newton	2	213	0.10280373831776	0.50467289719626	-0.12149532710280
Seidel	26	311	0.10280373830813	0.50467289718365	-0.12149532711372
Broyden	2	311	0.10280373831776	0.50467289719626	-0.12149532710280
Máximo descenso	26	10570	0.10280507558240	0.50467195758529	-0.12149388782774
Newton global	18	2732	0.10280334615262	0.50467097202194	-0.12149486363491

Como podemos deducir tanto la norma Euclídea como la norma infinita, ambas son buenas herramientas para utilizar como criterio de paro, ya que obtenemos casi idénticos resultados, por lo tanto, elegir entre norma Euclídea y norma infinita no influye de forma desmesurada los resultados obtenidos.

De cada algoritmo obtenemos conclusión final lo siguiente:

- Método del punto fijo, es de lo más eficientes estudiado en esta sección, el principal inconveniente es que depende notablemente de la 'g' del sistema, de manera que si esta 'g' del sistema tienen valores constantes obtendremos una única solución para distintos puntos iniciales, teniendo la posibilidad de descartar soluciones, como sucedía en el ejemplo 2.

- Método de Newton, necesita utilizar la matriz jacobiana con lo cual es natural que exista un incremento en el número de operaciones en comparación con el método del punto fijo, esto no sucede en el tercer ejemplo ya que obtenemos una matriz jacobiana con elementos constantes, con lo cual agiliza el procedimiento. No obstante, el método de Newton resulta muy eficiente.

- Método de Seidel, el principal inconveniente de este método es que tenemos que definir una 'g.m' especial para Seidel, pretende optimizar el método del punto fijo, y en ocasiones no lo consigue, a pesar de eso, es un buen método.

- Método de Broyden, pretende optimizar el método de Newton, para ello, pretende evitar el uso de la matriz jacobiana, pero el rodeo que da no resulta tan eficiente como se pensaba en un principio, y por tanto obtiene mayor iteraciones y mayor operaciones que el método de Newton.

- Método de máximo descenso, es deficiente para obtener una aproximación final, en cambio para obtener una aproximación inicial es muy útil, el principal inconveniente que tiene este método es que se basa en una idea difícil de comprender.

- Método de Newton global, al igual que el método de máximo descenso es bastante deficiente para obtener una aproximación final pero muy útil para conseguir una primera aproximación, esta primera aproximación la utilizará métodos como el de Newton, punto fijo, Seidel y Broyden.

Práctica 3 Interpolación polinomial. Derivación e integración numérica.

3.1 Introducción teórica.

En esta penúltima práctica estudiaremos dos bloques fundamentales de cualquier temario de análisis numérico, estos dos bloques son: interpolación polinomial y derivación e integración numérica.

En la parte de interpolación polinomial el problema general se basa que a partir de n+1 nodos de interpolación o puntos de medidas (x_1,x_2, ... , x_n) de un intervalo [a,b], y n+1 números reales (y_1, y_2, ... , y_n) llamados valores de interpolación se trata de obtener una función tal que $f(x_i)=y_i$,para i=1,2, ..., n+1.

En la parte de derivación e integración numérica nos dedicaremos a analizar técnicas para resolver los siguiente problemas: f '(a) o $\int f(x)dx$.

3.2 Interpolación polinomial.

El objetivo de la interpolación polinomial consiste que a partir de dos vectores X, vector de abcisas, donde todas sus componentes son diferentes, e Y vector de ordenadas, tenemos que obtener una función, en nuestro caso un polinomio, que 'simule' el comportamiento de la función que pasa por esos puntos, recordemos que sólo existe uno y sólo un polinomio de interpolación asociado a los vectores X e Y, (esta demostración fue desarrollada por Vandermonde) , nos encontraremos con los siguientes problemas:

- Si no poseemos el vector de abcisas X, ¿qué puntos debo de tomar para obtener el polinomio de interpolación?
- ¿Cómo podemos evaluar el polinomio en un punto determinado?
- ¿Cómo podemos obtener el polinomio de interpolación?

A estas cuestiones y a otras tantas intentaremos darle respuesta en este apartado de la tercera práctica.

3.2.1 Elección de nodos.

Cuando queremos medir un evento o una trayectoria de un objeto los nodos o puntos de medidas no pueden ser colocado de manera aleatoria ya que corremos el peligro de obtener parámetros que carecen de importancia, nosotros estudiaremos dos maneras de elegir nodos:

-Nodos igualmente espaciados.

-Nodos de Chebyshev.

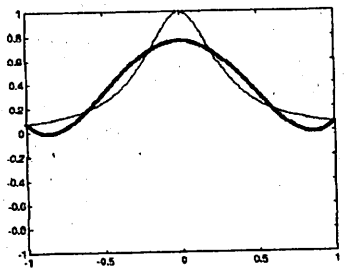
Los nodos igualmente espaciados son aquellos cuya distancia entre dos nodos consecutivos es constante para todos los nodos, los nodos de Chebyshev que se obtienen aplicando la siguiente formula:

$$x_i = \cos\left(\frac{(2i-1)\pi}{2(n+1)}\right) i = 1,2,\ldots,n+1 \; para [-1,1]$$

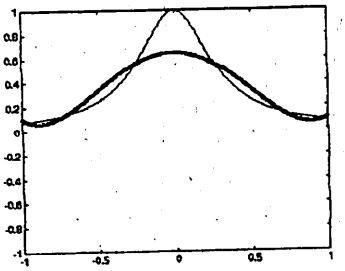
$$z(x) = \frac{b-a}{2}x + \frac{b+a}{2} \; para [a,b]$$

¿Cuáles son mejores nodos: los nodos igualmente espaciados o los nodos de Chebyshev? Para responder esta cuestión analizaremos el fenómeno de Runge (1901), supongamos la siguiente función $f(x)=1/(1+12x^2)$, si para esta función se construyen polinomios de interpolación p_n usando nodos igualmente espaciados se tiene que el error ($e_n(x)=f(x)-p_n$) no tiende a cero cuando $n\to\infty$, es decir que la sucesión p_n no converge, esta falta de convergencia se denomina fenómeno de Runge. Este problema ocurre porque usamos nodos igualmente espaciados, si usamos nodos de Chebyshev el error que se comete es mucho menor, como se muestra en las siguientes representaciones, esta situación tan tambolesca sucede más veces de la que nos imaginamos, de ahí la gran importancia de los nodos de Chebyshev.

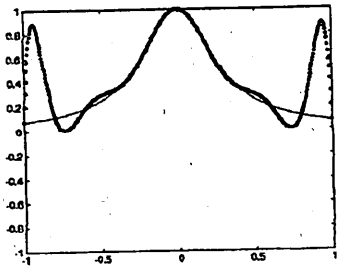
Nodos igualmente espaciados n=5



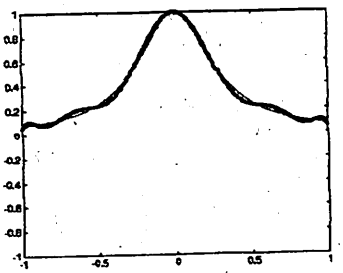
Nodos Chebyshev n=5



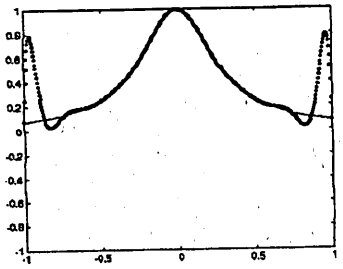
Nodos igualmente espaciados n=10



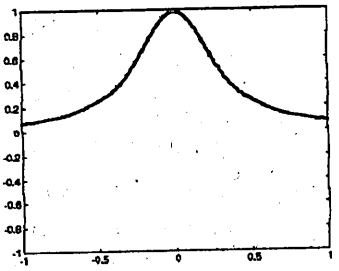
Nodos Chebyshev n=10



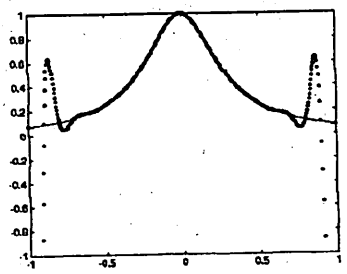
Nodos igualmente espaciados n=15



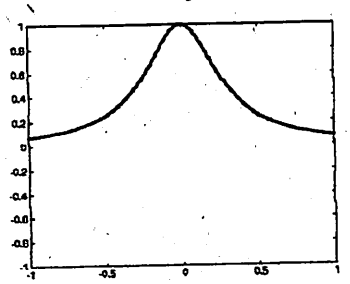
Nodos Chebyshev n=15



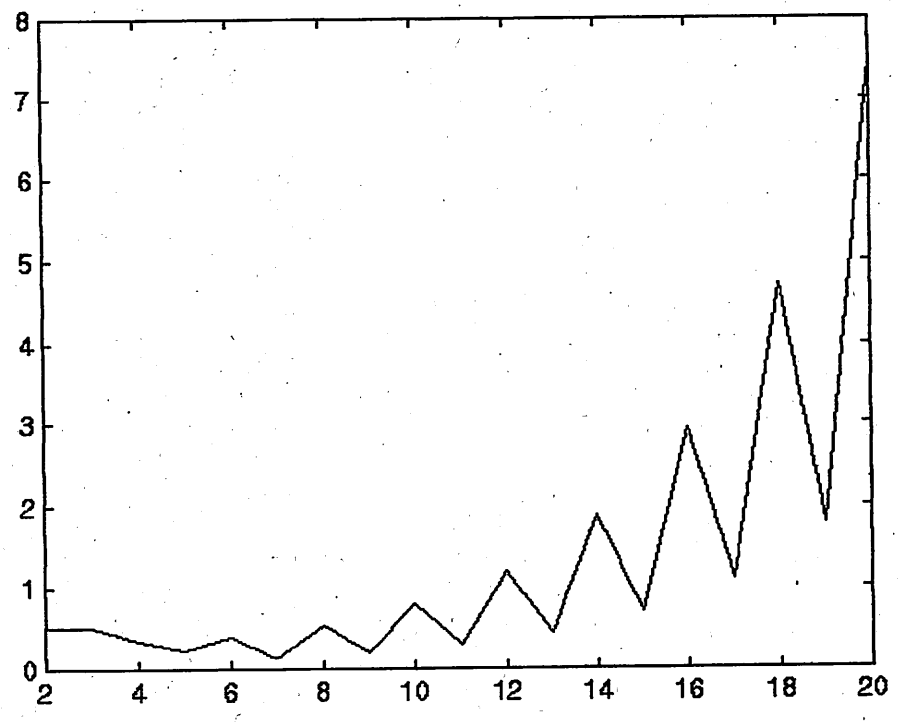
Nodos igualmente espaciados n=20



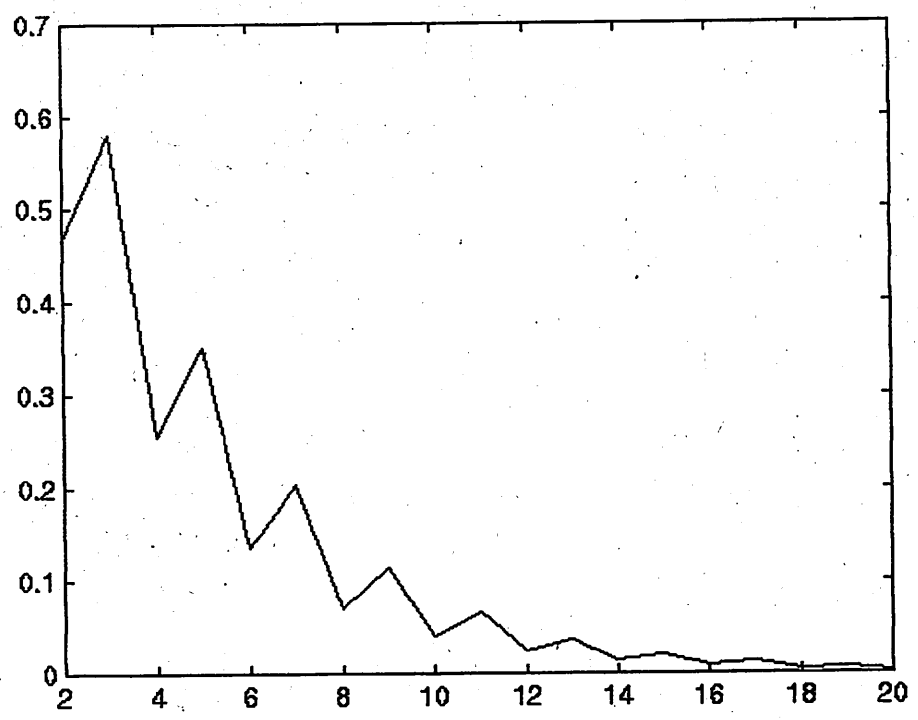
Nodos Chebyshev n=20



Evolución del error de nodos igualmente espaciados:



Evolución del error de nodos Chebyshev:



3.2.2 Evaluación de un punto en el polinomio de interpolación.

Supongamos que obtenemos el polinomio de interpolación mediante alguna técnica que posteriormente trataremos su análisis y deseamos calcular el valor del polinomio interpolado en un punto determinado, Matlab posee una función que realiza todo este trabajo esa función a la que nos referimos es polyval, (t=polyval(p,n), donde 't' es el resultado de evaluar 'n' en el polinomio 'p') pero existe otra manera de evaluar un punto en el polinomio de interpolación, es utilizando el algoritmo de Neville.

El algoritmo de Neville, se basa en la idea de empezar por los polinomios de interpolación de grado cero, es decir, por los valores de interpolación e ir calculando de forma iterada, los valores de polinomios interpoladores de grados cada vez más altos hasta llegar a n.

A continuación mostraremos los resultados obtenidos utilizando la función polyval y el algoritmo de Neville, a partir del vector de los nodos x=[1.0 1.3 1.6 1.9 2.2] y del vector de los valores y=[.76 .62 .45 .28 .11] obtendremos el polinomio de interpolación con la instrucción polyfit de Matlab, y deseamos evaluar el polinomio obtenido para el punto p=1.5.

	Operaciones realizadas	Aproximación
Polyval	716	0.50753086419753
Algoritmo de Neville	105	0.50753086419753

Como podemos observar el algoritmo de Neville es mucho más eficiente que la función polyval ya que requiere menos operaciones para encontrar la misma aproximación que polyval, por el contrario, la función polyval ya está implementada en Matlab.

3.2.3 Interpolación de Lagrange.

Una vez visto los apartados anteriores, estudiaremos los métodos que existen para estimar polinomios de interpolación, el primer método que vamos a estudiar es la interpolación de Lagrange.

La interpolación de Lagrange obtenemos un polinomio de interpolación a partir de una tabla de datos (x_i,y_i), con $0\le i\le n$, recordemos que existe **uno y sólo un** polinomio de interpolación de grado$\le n$ asociado a la tabla de datos.

El método de Lagrange expresa el polinomio de interpolación de la siguiente manera:

$$p(x) = y_0 L_0(x) + y_1 L_1(x) + \ldots + y_n L_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

En este caso $L_0, L_1, \ldots, L_n$ son polinomios que depende de los nodos $x_0, x_1, \ldots, x_n$, y no de las ordenadas $y_0, y_1, \ldots, y_n$. En vista de que las ordenadas podrían ser todas iguales a 0, excepto por un 1 que ocupa la i-ésima posición tenemos:

$$\delta_{ij} = p_n(x_j) = \sum_{k=0}^{n} y_k L_k(x_j) = \sum_{k=0}^{n} \delta_{ki} L_k(x_j) = L_i(x_j)$$

Recordemos que $\delta_{ki}=1$ si k=i y $\delta_{ki}=0$ si k≠i. Podemos fácilmente obtener un conjunto de polinomios que tengan esta propiedad. Consideremos L_0 , éste será un polinomio de grado n que tome el valor 0 en $x_1, x_2, \ldots, x_n$, y 1 en x_0, L_0 deberá adoptar la forma:

$$L_0(x) = c(x-x_1)(x-x_2)\ldots(x-x_n) = c\prod_{j=1}^{n}(x-x_j)$$

El valor de c se obtiene haciendo $x=x_0$, de manera que:

$$1 = c\prod_{j=1}^{n}(x_0 - x_j)$$

$$c = \prod_{j=1}^{n}(x_0 - x_j)^{-1}$$

Por lo tanto:

$$L_0(x) = \prod_{j=1}^{n}\frac{(x - x_j)}{(x_0 - x_j)}$$

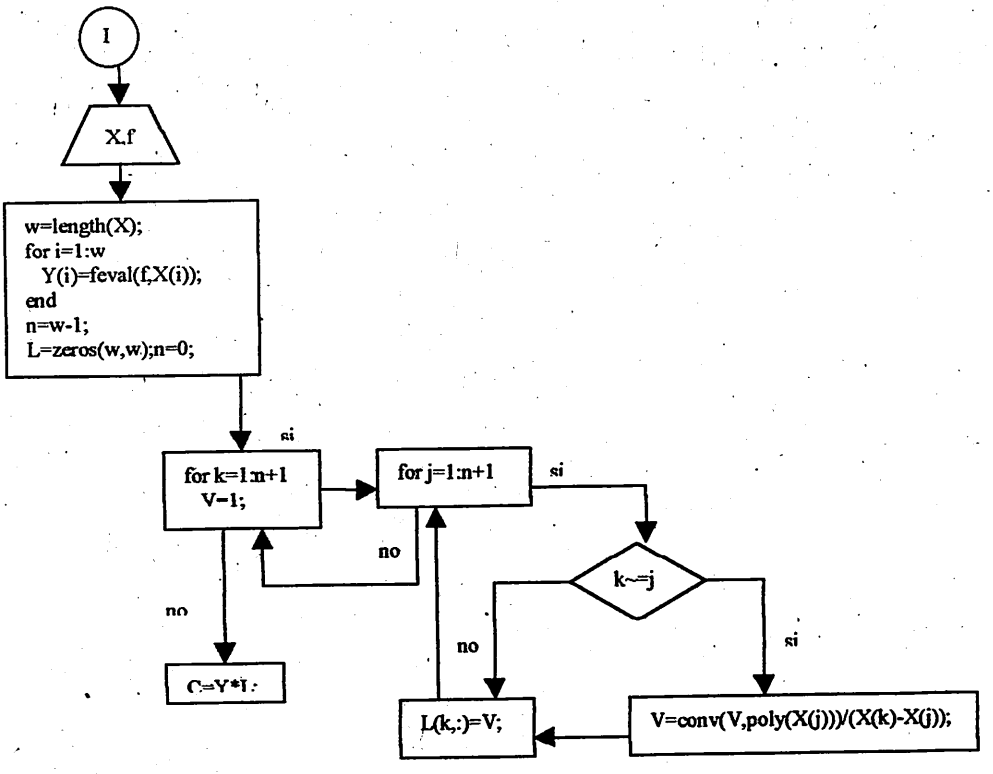
Un razonamiento similar nos permite obtener el resto de las L_i:

$$L_i(x) = \prod_{\substack{j=0 \\ j\neq i}}^{n}\frac{(x - x_j)}{(x_i - x_j)} = \frac{(x - x_1)\cdot\ldots\cdot(x - x_n)}{(x_i - x_1)\cdot\ldots\cdot(x_i - x_n)}$$

Por ejemplo (1,3) y (2,4):

$$p(x) = 3\frac{x-2}{1-2} + 4\frac{x-1}{2-1} = x + 2$$

Para que sea más fácil de implementar el método de Lagrange ofrecemos a continuación su diagrama de flujo:

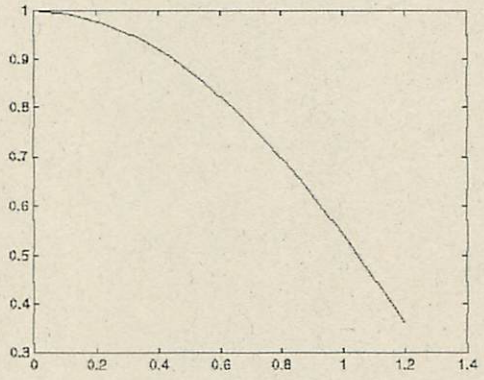


Para finalizar con el método de Lagrange vamos a obtener el polinomio de interpolación de la función f(x)=cos(x), para el intervalo [0,1.2], con nodos igualmente espaciados y con los nodos de Chebyshev, en ambos caso mostrar el error producido.(Error=f(x)-p(x),n=12)

Con nodos igualmente espaciados			
x_i	$f(x_i)=\cos(x_i)$	$P(x_i)$	$\lvert f(x_i)-P(x_i)\rvert$
0.0	1.00000000000000	1.00000000000000	0
0.1	0.99500416527803	0.99500416527788	0.00000000000014732660
0.2	0.98006657784124	0.98006657783955	0.00000000000168831615
0.3	0.95533648912561	0.95533648911573	0.00000000000987765425
0.4	0.92106099400289	0.92106099396278	0.00000000004010680676
0.5	0.87758256189037	0.87758256175294	0.00000000013743572946
0.6	0.82533561490968	0.82533561447028	0.00000000043939762939
0.7	0.76484218728449	0.76484218594102	0.00000000134347255543
0.8	0.69670670934717	0.69670670545555	0.00000000389161725067
0.9	0.62160996827066	0.62160995771053	0.00000001056013920753
1.0	0.54030230586814	0.54030227911499	0.00000002675315391443
1.1	0.45359612142558	0.45359605797191	0.00000006345366576621
1.2	0.36235775447667	0.36235761277388	0.00000014170279005743

Con nodos de Chebyshev			
x_i	$f(x_i)=\cos(x_i)$	$P(x_i)$	$\lvert f(x_i)-P(x_i)\rvert$
1.19562532445883	0.36643164270711	0.36643158790417	0.000000054802938409537
1.16100974561125	0.39841358806760	0.39841354591957	0.000000042148031065281
1.09379031953619	0.45912145345766	0.45912142872580	0.000000024731868497607
0.99787359494448	0.54209039116131	0.54209038028846	0.000000010872849065535
0.87883390322626	0.63804946692179	0.63804946346451	0.000000003457283792585
0.74358939857253	0.73604351873029	0.73604351799218	0.000000000738113903331
0.60000000000000	0.82533561490968	0.82533561482487	0.000000000084812934453
0.45641060142747	0.89764022853884	0.89764022854034	0.000000000001495026325
0.32116609677374	0.94886795773772	0.94886795773974	0.000000000002022382262
0.20212640505552	0.97964191095729	0.97964191095756	0.000000000000271782596
0.10620968046381	0.99436505196109	0.99436505196110	0.000000000000015432100
0.03899025438875	0.99923997632353	0.99923997632353	0.000000000000000999201
0.00437467554117	0.99999043112222	0.99999043112222	0.000000000000000444089

Hemos estudiado de forma independiente la formula de Lagrange obtenida con los nodos igualmente espaciados y con los nodos de chebyshev para comprobar el comportamiento del polinomio de interpolación, ambos polinomios de interpolación tanto lo que generamos por Lagrange con nodo igualmente espaciados como el de Lagrange con nodos de Chebyshev tiene un comportamiento idéntico a la función que pretendiamos aproximar, de manera gráfica:



3.2.4 Diferencias divididas.

El polinomio de interpolación p, para los datos $\{(x_i,y_i)$: i=1,2,...,n+1$\}$, viene dado por:

$$p(x)=f[x_1]+f[x_1,x_2]\cdot(x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2)+\ldots+f[x_1,\ldots,x_n](x-x_1)\ldots(x-x_n)$$

Donde las diferencias divididas $f[x_1,\ldots,x_k]$ para k=1,2,...,n+1, están definidas según el siguiente esquema recursivo:

$$f[x_i]=y_i \quad i=1,2,\ldots,n+1$$

$$f[x_i,\ldots,x_{i+k}]=(f[x_{i+1},\ldots,x_{i+k}]-f[x_i,\ldots,x_{i+k-1}])/(x_{i+k}-x_i)$$

Para calcular las diferencias divididas se utiliza la siguiente tabla:

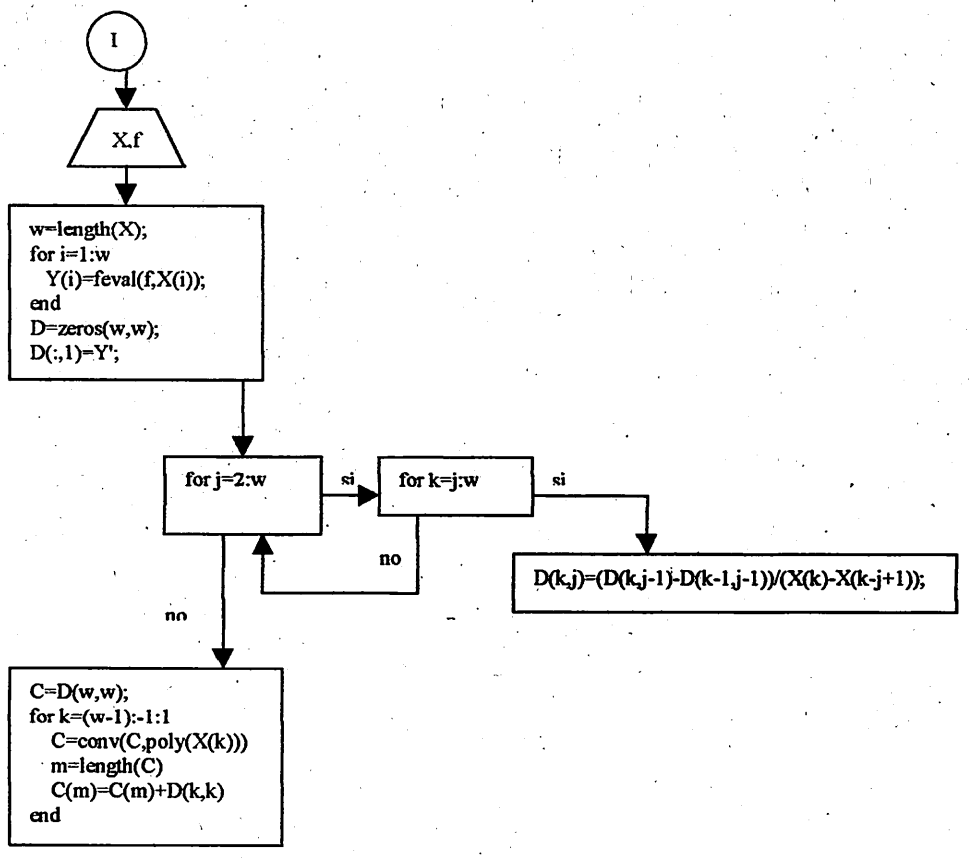
x	f(x)=y	1ª dif. div.	2ª dif. div.
x_1	$f[x_1]=y_1$		
x_2	$f[x_2]=y_2$	$f[x_1,x_2]=(f[x_2]-f[x_1])/(x_2-x_1)$	
x_3	$f[x_3]=y_3$	$f[x_2,x_3]=(f[x_3]-f[x_2])/(x_3-x_2)$	$f[x_1,x_2,x_3]=(f[x_2,x_3]-f[x_1,x_2])/(x_3-x_1)$

Ejemplo: x=[0,1,2] y=[1,2,5]

x	f(x)=y	1ª dif. div.	2ª dif. div.
0	$f[x_1]=1$		
1	$f[x_2]=2$	$f[x_1,x_2]=(f[x_2]-f[x_1])/(x_2-x_1)=1$	
2	$f[x_3]=5$	$f[x_2,x_3]=(f[x_3]-f[x_2])/(x_3-x_2)=3$	$f[x_1,x_2,x_3]=(f[x_2,x_3]-f[x_1,x_2])/(x_3-x_1)=1$

$$p_2(x)=f[x_1]+f[x_1,x_2]\cdot(x-x_1)+f[x_1,x_2,x_3](x-x_1)(x-x_2)=1+1(x-0)+1(x-0)(x-1)=x^2+1$$

Como viene siendo habitual a continuación mostraremos su diagrama de flujo:



Para concluir con el método de diferencia divididas, como hicimos anteriormente con el método de Lagrange, vamos a obtener el polinomio de interpolación de la función f(x)=cos(x), para el intervalo [0,1.2], con nodos igualmente espaciados. (Error=f(x)-p(x),n=12)

Con nodos igualmente espaciados			
x_i	$f(x_i)=\cos(x_i)$	$P(x_i)$	$\lvert f(x_i)-P(x_i)\rvert$
0.0	1.00000000000000	1.00000000000000	0
0.1	0.99500416527803	0.99500416527803	0
0.2	0.98006657784124	0.98006657784124	0
0.3	0.95533648912561	0.95533648912561	0
0.4	0.92106099400289	0.92106099400289	0
0.5	0.87758256189037	0.87758256189037	0
0.6	0.82533561490968	0.82533561490968	0
0.7	0.76484218728449	0.76484218728449	0
0.8	0.69670670934717	0.69670670934717	0
0.9	0.62160996827066	0.62160996827066	0
1.0	0.54030230586814	0.54030230586814	0
1.1	0.45359612142558	0.45359612142558	0
1.2	0.36235775447667	0.36235775447667	0

Es realmente eficiente, al igual que el método de Lagrange obtiene unos polinomios de interpolación muy próximos a la función.

3.2.5 Diferencias progresivas.

Supongamos que los nodos están igualmente espaciados $x_i-x_{i-1}=h$ para i=2,3,...,n+1. Sea $S=x_1+Sh$, entonces el polinomio interpolado de los puntos (x_i,y_i) para i=1,2,...,n+1 es:

$$p(x)=\sum\binom{s}{k}\Delta^k f(x) \qquad \binom{s}{k}=\frac{s(s-1)\ldots(s-k+1)}{k!}$$

Ejemplo: x=[1,2,3] y=[2,1,3]

x	$f(x)=y=\Delta^0$	Δ^1	Δ^2
1	$f[x_1]=2$		
2	$f[x_2]=1$	1-2=-1	
3	$f[x_3]=3$	3-1=2	2-(-1)=3

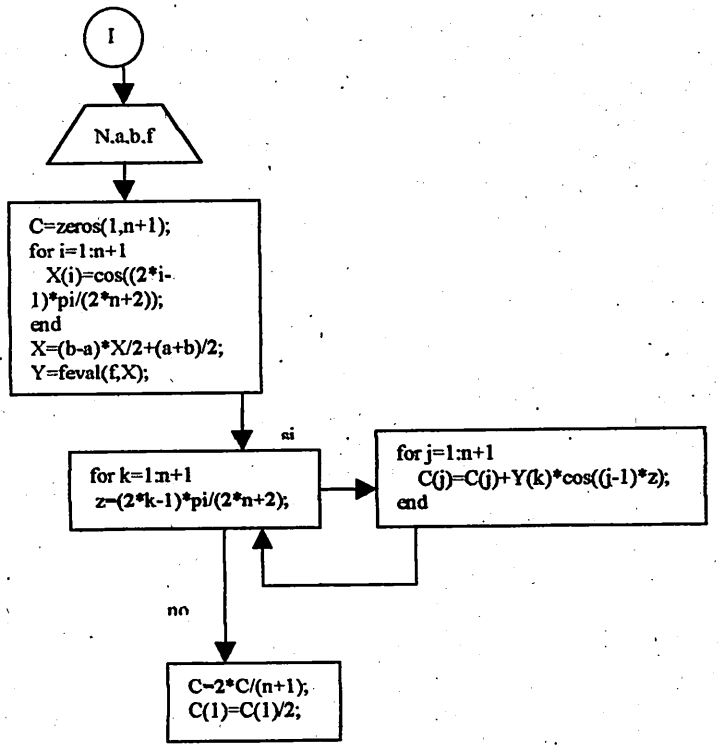
$$p(x)=1\cdot 2+S\cdot(-1)+\frac{S(S-1)}{2}\cdot 3 \qquad x=x_1+S\cdot h$$
$$p(x)=2-(1-x)+\frac{(1-x)(-x)}{2}\cdot 3 \qquad h=1$$
$$p(x)=\frac{3}{2}x^2-\frac{11}{2}x+6 \qquad x_1=1$$
$$S=1-x$$

3.2.6 Polinomios de Chebyshev.

Anteriormente estudiamos la elección de los nodos , en ese apartado hicimos mención de los nodos de Chebyshev, los nodos de Chebyshev son las raíces reales del polinomio de Chebyshev, un polinomio que nos sirve para interpolar cualquier función en el intervalo [-1,1], el polinomio de Chebyshev se define recurrentemente de la siguiente forma:

$$T_0(x) = 1$$
$$T_1(x) = x$$
$$T_n(x) = 2x \cdot T_{n-1} - T_{n-2}$$

A continuación, mostraremos su diagrama de flujo:



Tanto en el método de diferenciación progresiva como el método de interpolación de Chebyshev no mostramos los resultados obtenidos con el ejemplo f(x)=cos(x), para el intervalo (0,1.2), el principal motivo de este hecho es que obtenemos resultado ilógicos y tras analizarlo detenidamente no soy capaz de averiguar el fallo que origina esta situación. No obstante, a continuación estudiaremos las ventajas y deventajas que tienen los métodos estudiado hasta ahora para interpolar un polinomio (método de Lagrange, método de diferencias divididas, método de diferencias progresivas y el método de interpolación de Newton).

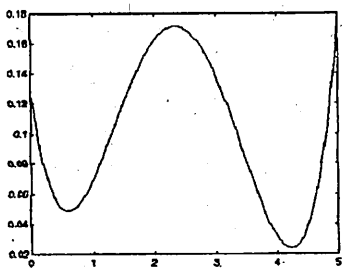
3.2.7 Comparaciones y conclusiones.

METODOS	VENTAJAS	DESVENTAJAS
Lagrange	-Es bastante eficiente. -Proporciona un buen polinomio de interpolación tanto usando nodos de Chebyshev como nodos igualmente espaciados.	
Diferencias divididas	-Al igual que el método de Lagrange, resulta bastante eficiente.	
Diferencias progresivas		-Posee un gran coste en cálculo computacional, ya que emplea la operación factorial. -Difícil de programar.
Chebyshev	-Genera el polinomio de interpolación a partir de reglas de recurrencia, con lo cual se optimiza el método. -Los ceros o raíces del polinomio de interpolación de Chebyshev son los nodos de Chebyshev.	-Necesita los nodos de Chebyshev.

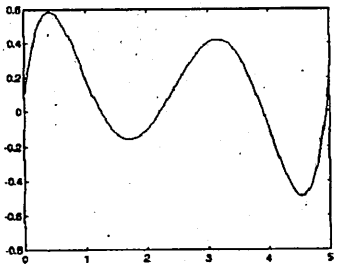
3.2.8 Ejemplos.

1.-Dado los puntos del plano x=[0 1 2 2 4 5] e y=[0.1 0.2 –0.1 0.4 –0.1 0.2] representar gráficamente los polinomios de interpolación con los métodos anteriormente estudiado.

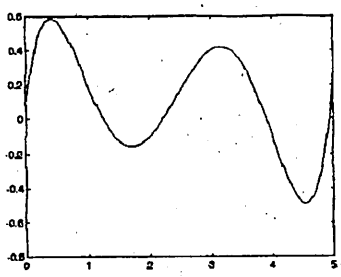
Polyfit



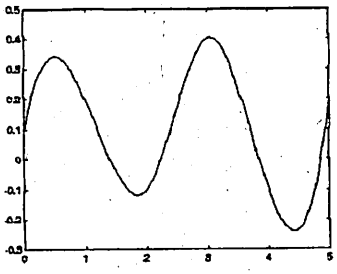
Lagrange



Diferencias divididas



Splines



Como podemos observar obtenemos una gráficas muy semejantes con los métodos de Lagrange y con el método de diferencias divididas, el principal motivo de este fenómeno es debido a que obtenemos polinomios de interpolación muy parecidos, con la técnica del polyfit, implementado por Matlab, obtenemos una gráfica que no se parece a ninguna del resto, esto surge porque le hemos especificado un grado para el polinomio muy pequeño, con lo cual obtiene un mal polinomio de interpolación y obtenemos una gráfica que representa poco la realidad.

Para concluir hemos representado la gráfica que obtenemos si aplicamos spline, un spline es una función que está formada por varios polinomios, cada uno definido sobre un subintervalo, que se unen entre sí obedeciendo a ciertas condiciones de continuidad. Para una definición formal procedemos como sigue: suponga que se han especificado n+1 puntos $t_0 < t_1 < \ldots < t_n$. A estos puntos se llaman nudos o nodos. Supongamos además que se ha fijado un entero $k \geq 0$. Una función spline de grado k con nudos $t_0, t_1, \ldots, t_n$ es una función S que satisface las siguientes condiciones:

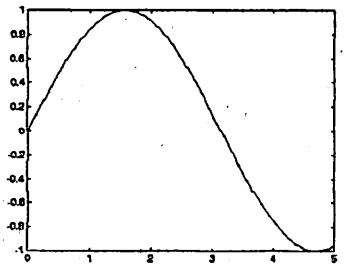
-En cada intervalo $[t_{i-1}, t_i)$, S es un polinomio de grado $\leq k$.

-S tiene una derivada de orden (k-1) continua en $[t_0, t_n]$.

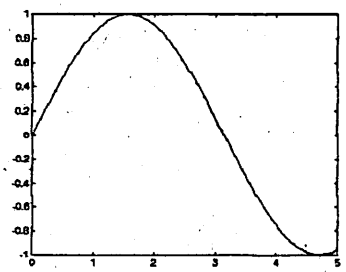
Por consiguiente S es un polinomio continuo por pedazos de a lo más grado k, que tiene derivadas contínuas de orden hasta (k-1).

2.-Realizar un análisis entre los nodos igualmente espaciados y los nodos de Chebyshev, utilizando para ello, los métodos anteriormente vistos, representando gráficamente los polinomios de interpolación que obtengamos en cada caso, utilizando la función f(x)=sen (x).

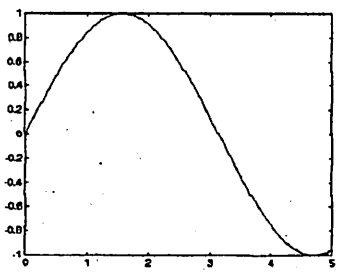
Polyfit con nodos igualmente espaciados



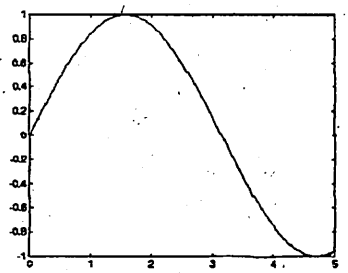
Polyfit con nodos de Chebyshev



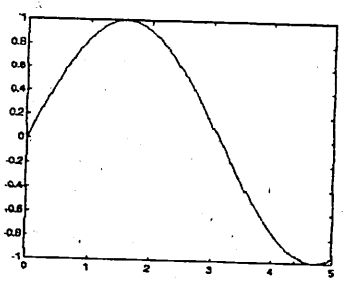
Lagrange con nodos igualmente espaciados



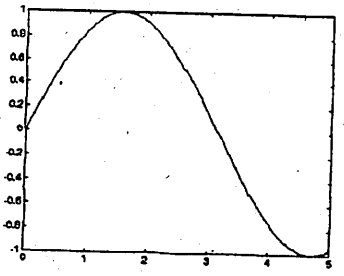
Lagrange con nodos de Chebyshev



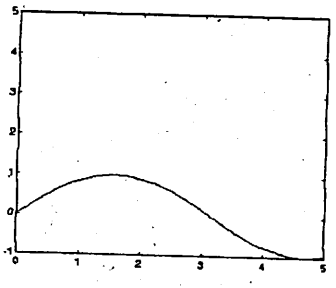
Dif. divididas con nodos igualmente espaciados



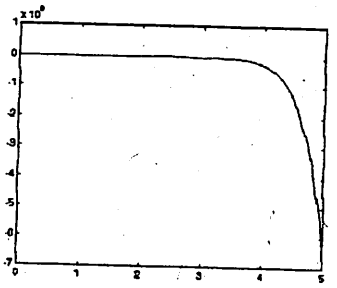
Dif. divididas con nodos de Chebyshev



Spline con nodos igualmente espaciados



Polinomio de Chebyshev



Como podemos observar obtenemos los mismos reustados para los métodos del polyfit, el método de Lagrange y con el método de diferencias divididas para los nodos igualmente espaciados y para los nodos de Chebyshev, con lo cual no influye tanto la elección por uno o por otro tipo de nodos.

En principio podriamos pensar que con el método de spline obtenemos un resultado incorrecto pero si observamos con detenimiento nos daremos cuenta que está representado en otro tipo de coordenadas.

En cambio podemos observar que para el método del polinomio de Chebyshev obtenemos un resultado totalmente distinto si lo comparamos con los métodos anteriores, desconocemos el principal motivo que genera este fenómeno ya que anteriormente comprobamos la eficiencia de este método y pensamos que es debido a un error de implementación.

3.3 Derivación e integración numérica.

Si se nos proporcionan los valores de una función f en ciertos puntos, digamos $x_0,x_1,...,x_n$, podemos usar esta información para obtener una estimación de cómo es su derivada o una integral. Tanto la derivación como la integración tiene numerosas aplicaciones, en esta sección nos dedicaremos hacer un breve análisis de las técnicas más relevantes para estimar derivadas en un punto de una determinada función o calcular la integral definida de una función.

Para estudiar la derivación nos basaremos en las fórmulas progresivas,regresivas y centradas, para la integración nos basaremos sobretodo en casos particulares de la fórmula cerrada de Newton-Côrtes.

Además estudiaremos el método de extrapolación de Richardson, un método que lo utilizaremos para obtener mayor exactitud en algunas formulas numéricas.

3.3.1 Derivación:Fórmulas progresivas,regresivas y centradas.

Sea f una función y $\{x_1, x_2,..., x_n\}$ puntos distintos en un intervalo [a,b] y basandonos en los polinomios elementales de Lagrangre, si $x_1=x$, $x_2=x+h$, $x_3i=x+2h$ para aproximar f '(x) tenemos:

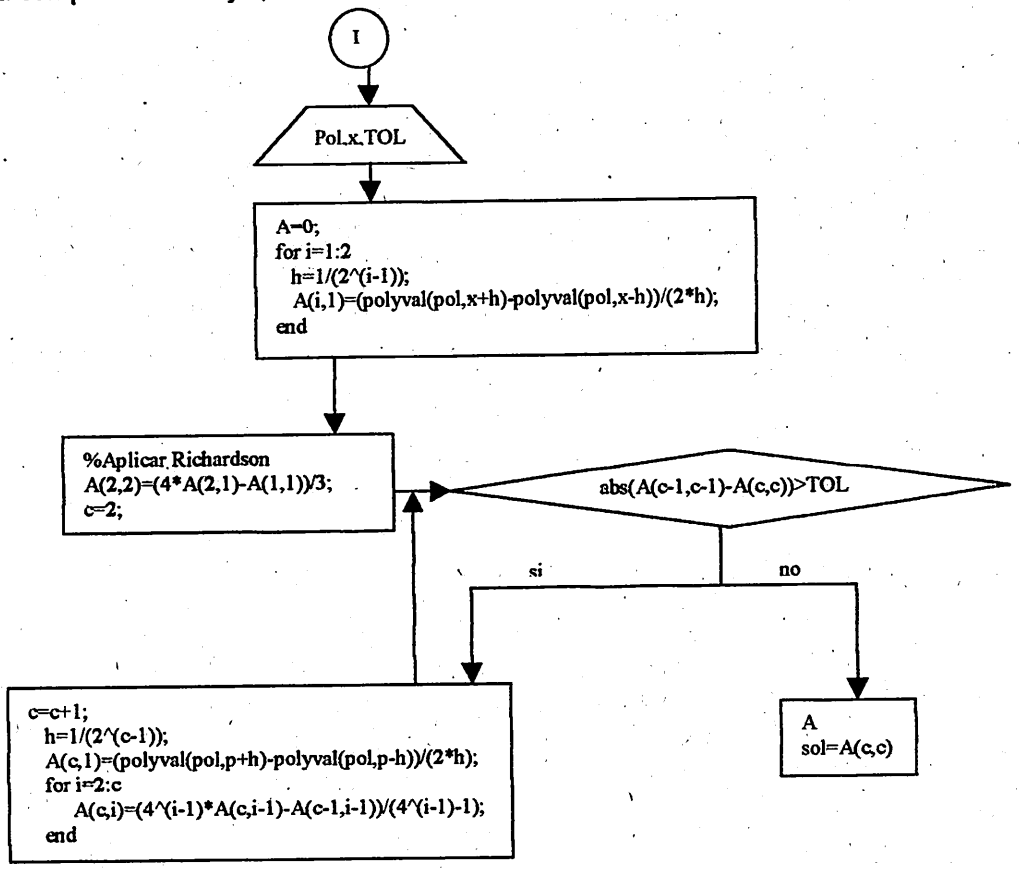
$$f'(x)= \frac{-3f(x)+4f(x+h)-f(x+2h)}{2h} + \frac{h^2}{3} f^{3)}(\alpha)$$

De tal forma que si h>0 se conoce como la fórmula progresiva de tres puntos y para h<0 se conoce con el nombre de formula regresiva de tres puntos. Si tomamos $x_1=x-h$, $x_2=x$ y $x_3=x+h$ para aproximar f '(x) obtenemos:

$$f'(x)= \frac{f(x+h)-f(x-h)}{2h} - \frac{h^2}{6} f^{3)}(\alpha)$$

A la fórmula anterior se la conoce cómo fórmula centrada de tres puntos, y es la que emplearemos para calcular la derivada de una función en un determinado punto, y elegimos esta porquela cota de error es menor y además necesita menos evaluaciones de f.

Para comprenderlo mejor, ofrecemos a continuación sun diagrama de flujo:



3.3.2 Extrapolación de Richardson.

Mostraremos ahora como puede usarse, para obtener más exactitud en algunas fórmulas numéricas, un procedimiento conocido como extrapolación de Richardson.

$M=N(h)+Error(h)$ M=Valor exacto, N(h)=Valor aproximado

$M=N(h)+k1h^2+k2h^4+k3h^6+...$ Conocemos N(h) y el tipo de error que se produce.

Para que sea más preciso tendremos que ir quitando componente al error, en este caso h^2.

$M=N(h)+k1h^2+k2h^4+k3h^6+...$

h=h/2 $M=N(h/2)+k1(h^2/4)+k2(h^4/16)+k3(h^6/64)+...$

$$-(M=N(h)+k1h^2+k2h^4+k3h^6+...)$$
$$+\ 4(M=N(h/2)+k1(h^2/4)+k2(h^4/16)+k3(h^6/64)+...)$$
$$3M=4N(h/2)-N(h)+k2h^4+k3h^6+...$$

$$N^{'}=M=\frac{4N(h/2)-N(h)+k2h^4+k3h^6+...}{3}$$ Siendo k2 y k3 nuevas constantes

De manera recursiva podemos y eliminando componente a componente del error, expresado en forma tabular quedaría:

M	$O(h^2)$	$O(h^4)$	$O(h^6)$
h=0.1	N(h)		
h/2	N(h/2)	$N^{'}(h)=(4N(h/2)-N(h))/3$	
h/4	N(h/4)	$N^{'}(h/2)=(4N(h/4)-N(h/2))/3$	$N^{''}(h)=(16N^{'}(h/2)-N^{'}(h))/15$
h/8	N(h/8)	$N^{'}(h/4)=(4N(h/8)-N(h/4))/3$	$N^{''}(h/2)=(16N^{'}(h/4)-N^{'}(h/2))/15$

Aplicaremos Richardson siemple que:
-Conozcamos el valor de M.
-Conozcamos la formula N().

Un aspecto de gran importancia que hemos pasado por alto es que tendremos que tener en cuenta el tipo de error que posee la relación, a continuación mostraremos a Richardson lo más genérico posible:

M	$O(h^l)$	$O(h^j)$	$O(h^k)$
h=0.1	N(h)		
h/l	N(h/l)	$N^{'}(h)=(l^jN(h/l)-N(h))/(l^j-1)$	
h/m	N(h/m)	$N^{'}(h/l)=(m^jN(h/m)-N(h/l))/(m^j-1)$	$N^{''}(h)=(m^jN^{'}(h/2)-N^{'}(h))/(m^j-1)$
h/n	N(h/n)	$N^{'}(h/m)=(n^jN(h/n)-N(h/m))/(n^j-1)$	$N^{''}(h/l)=(m^jN^{'}(h/4)-N^{'}(h/2))/(m^j-1)$

3.3.3 Integración:Regla del trapecio.

La regla del trapecio es un ejemplo particular de fórmulas cerradas donde n=1 y los nodos x_0=a, x_1=b. Entonces los polinomios fundamentales para la interpolación son:

$$l_0(x) = \frac{b-x}{b-a} \qquad l_1(x) = \frac{x-a}{b-a}$$

Y como consecuencia de lo anterior,

$$A_0 = \int_a^b l_0(x)dx = \tfrac{1}{2}(b-a) = \int_a^b l_1(x)dx = A_1$$

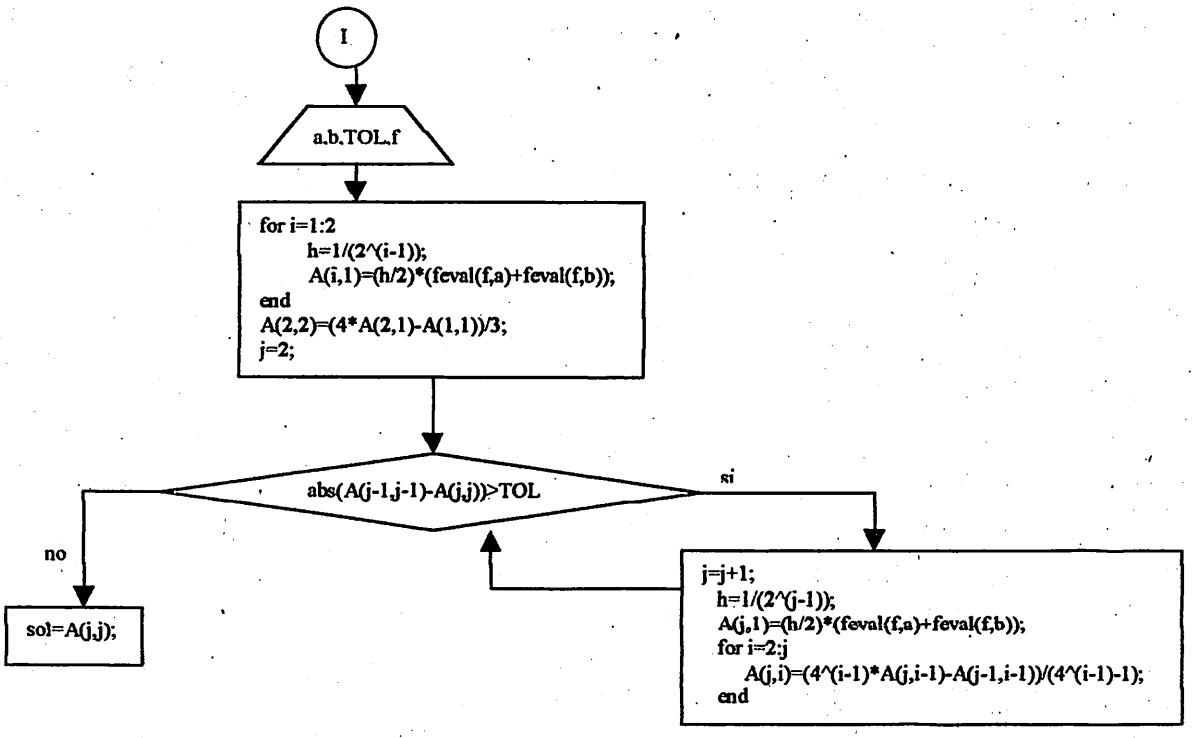
La fórmula de cuadratura correspondiente es: (h=b-a)

$$\int_a^b f(x)dx = \frac{h}{2}[f(a)+f(b)] - \frac{h^3}{12}f''(\alpha), a < \alpha < b$$

La regla del trapecio proporciona un resultado exacto para todos los polinomios de grado a lo sumo 1.Siendo el error que se comete:

$$-\frac{h^3}{12}f''(\alpha)$$

Su diagrama de flujo es el siguiente, utilizando Richardson para optimizar el resultado:

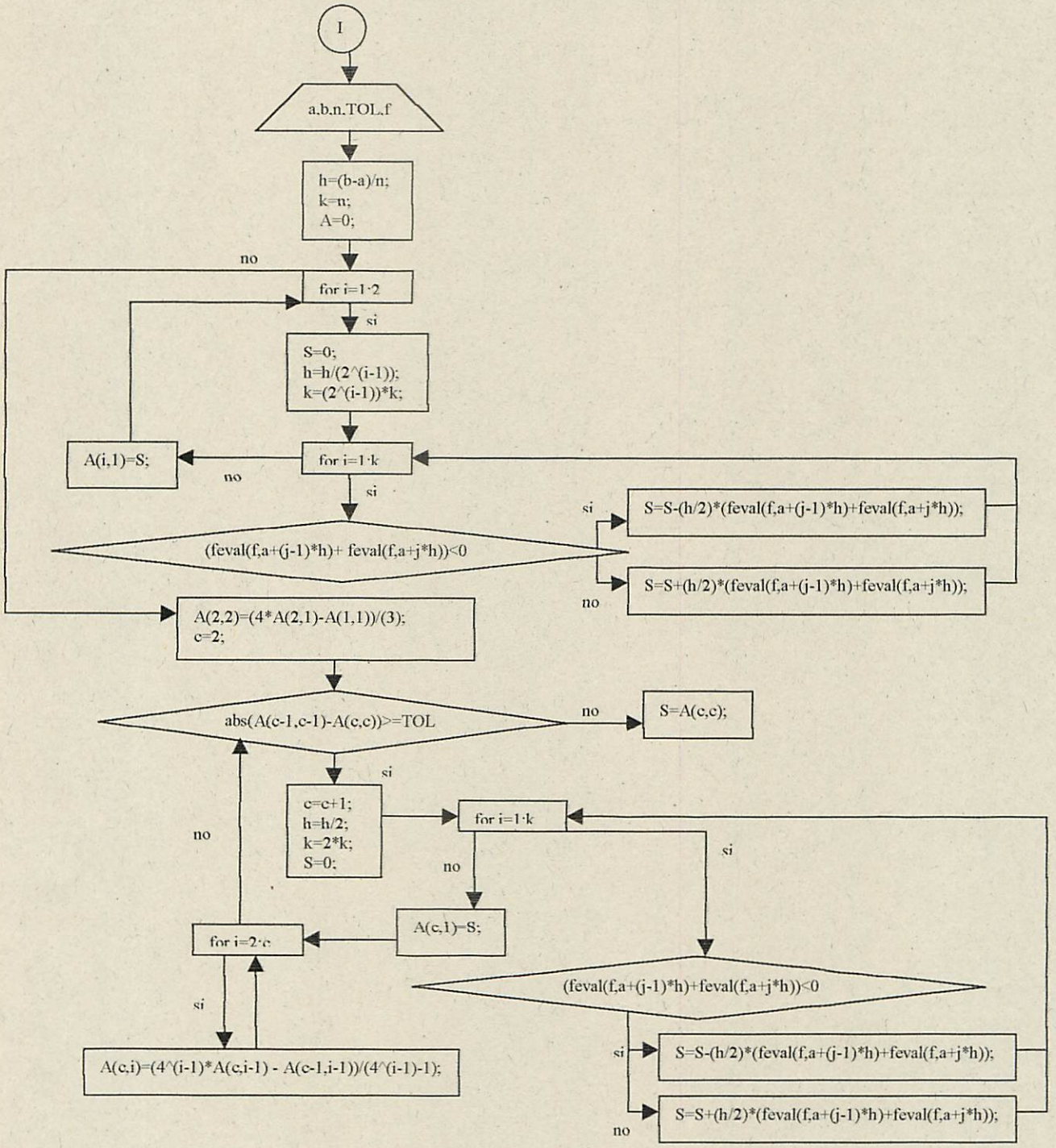


3.3.4 Integración:Regla del trapecio compuesta.

Si en el intervalo [a,b] donde se desea calcular la integral, se hace la siguiente partición: $a=x_0<x_1...<x_n=b$, entonces podemos aplicar la regla del trapecio a cada uno de los subintervalos. En este caso los nodos no se encuentran necesariamente espaciados de manera uniforme. Es así como se obtiene la regla del trapecio compuesta.

$$\int_a^b f(x)dx = T_h(f) - \frac{b-a}{12}h^2 f''(\alpha) \qquad T_h(f) = \frac{h}{2}\left[f(a) + 2\sum_{k=1}^{n-1} f(x_k) + f(b) \right]$$

Su diagrama de flujo es, utilizando Richardson para optimizar el resultado es el siguiente:

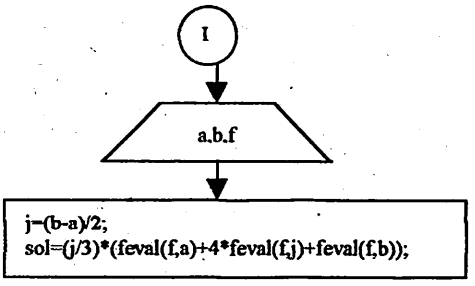


3.3.5 Integración:Regla de Simpson.

La regla de Simpson al igual que la regla del trapecio es un ejemplo particular de fórmulas cerradas donde n=2, la formula de la regla de Simpson es la siguiente:

$$\int_a^c f(x)dx = \frac{h}{3}[f(a)+4f(b)+f(c)]-\frac{h^5}{90}f^{4)}(\alpha), a<\alpha<c$$

Sabemos que es exacta para todos los polinomios de grado 2, e incluso, obtenemos solución exacta para los polinomios de grado 3. Su diagrama de flujo es, sin utilizar Richardson para su optimización es el siguiente:

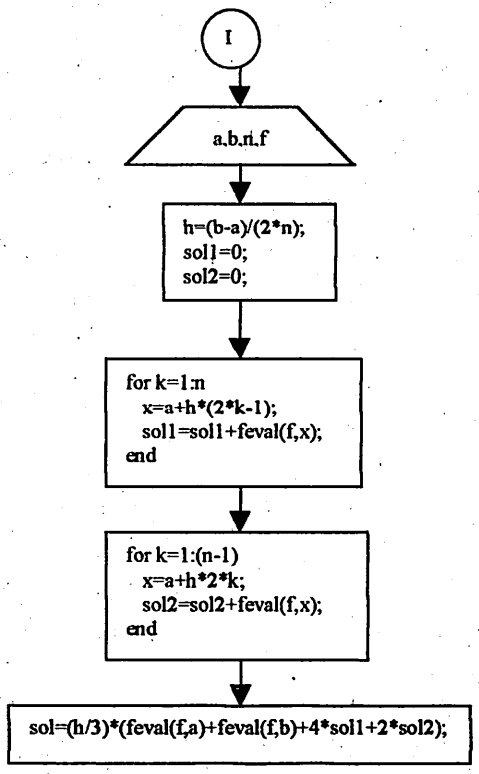


3.3.6 Integración:Regla de Simpson compuesta.

La regla de Simpson para los nodos x_k=a+kh (k=0,1,...,n), siendo n=2m par y h=(b-a)/n, puede expresarse como:

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(a)+2\sum_{k=1}^{m-1}f(x_{2k})+4\sum_{k=1}^{m}f(x_{2k-1})+f(b)\right]-\frac{b-a}{180}h^4 f^{4)}(\alpha)$$

Su diagrama de flujo es, sin utilizar Richardson para su optimización es:



3.3.7 Integración:Método de Romberg.

Es conocido que el término del error en la fórmula del trapecio compuesta, viene dado por, para h=(b-a)/n:

$$\int_a^b f(x)dx = T_h(f) + \sum_{i=1}^{k} l_{2i} h^{2i} + O(h^{2k+2})$$

Aplicando la extrapolación de Richardson a la fórmula anterior se obtiene la integración de Romberg.

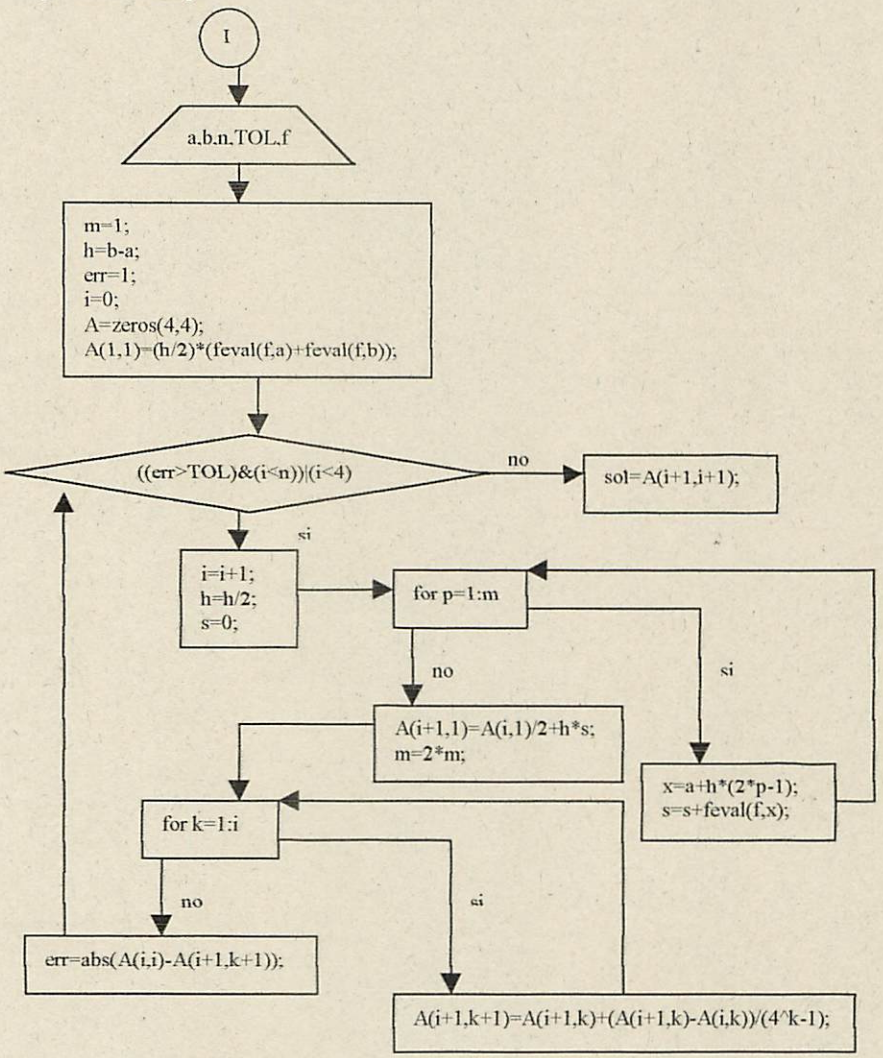
$$N_1(h) = T_h(f)$$

$$N_j(h) = \frac{4^{j-1} N_{j-1}(\frac{h}{2}) - N_{j-1}(h)}{4^{j-1} - 1}, j >= 2$$

Se cumple:

$$\int_a^b f(x)dx = N_j(h) + O(h^{2j})$$

Su diagrama de flujo es:



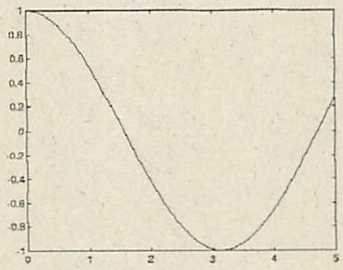
3.3.8 Comparaciones y conclusiones.

MÉTODOS	VENTAJAS	DESVENTAJAS
Regla del Trapecio	-Es sencillo de implementar el método, ya que se basa en una idea muy simple para obtener el valor de la integral. -Si se emplea Richardson resulta bastante eficiente. -Proporciona un resultado exacto para todos los polinomios de grado a lo sumo 1.	-Necesita Richardson para mejorar el resultado. -Sólo obtenemos resultados exactos para polinomios de grado 1.
Regla del Trapecio Compuesta	-Resulta más eficiente que el método de la regla del trapecio. -Utilizando la técnica de Richardson obtenemos mejores resultados. - Se basa en una idea no muy difícil de entender.	-Al igual que el método anterior, la regla del trapecio, necesita la técnica de Richardson para optimizar el resultado obtenido.
Regla de Simpson	-Se basa en una idea no muy compleja con lo cual no resulta tan difícil de implementar. -No necesita Richardson para obtener un resultado eficiente. -Proporciona un resultado exacto para todos los polinomios de grado 2 e incluso para los polinomios de grado 3.	-Sólo obtenemos resultados exactos para polinomios de grado 3.
Regla de Simpson Compuesta	-Es sencillo de implementar el método, ya que se basa en una idea muy simple para obtener el valor de la integral. -No necesita Richardson para obtener un resultado eficiente.	-No resulta tan eficiente como la regla de Romberg.
Regla de Romberg	-Es el método más eficiente a la hora de calcular integrales definidas.	-Necesita aplicar la técnica de Richardson.

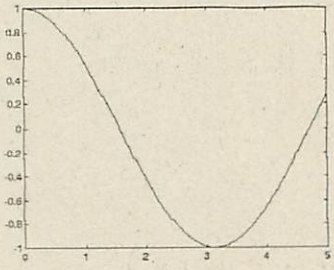
3.3.9 Ejemplos.

1.-Utilizando los métodos de interpolación vistos anteriormente (método de diferencias divididas, método de Laplace,...) representar gráficamente f'(x) siendo f(x)=sen(x), para calcular su derivada utilizar el método visto en la práctica.

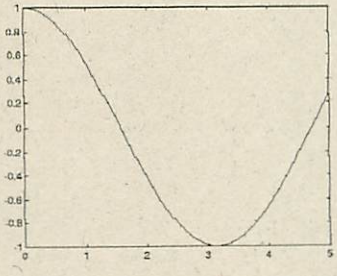
Con polyfit



Con Lagrange



Con Diferencia divididas



Es fácil darse cuenta de que obtenemos los mismos resultados, anteriormente comprobamos la eficiencia de los métodos del polyfit, el método de Lagrange y el método de diferencias divididas, este ejercicio es más bien un ejercicio de interpolación que de derivación numérica, con la particularidad de que el vector de abcisas que le pasamos a Lagrange o a polyfit lo obtenemos tras aplicar el algoritmo de derivación numérica que hemos implementado.

A continuación estudiaremos un ejemplo para estudiar la derivada primera, la derivada segunda e incluso la derivada tercera de la función $f(x)=x^3+ x^2+x$.

2.-Obtener la derivada primera, la derivada segunda y la derivada tercera de la función $f(x)=x^3+ x^2+x$ en el punto x=2. Explica como obtienes la derivada segunda y la derivada tercera.

Para obtener la derivada primera, nos basamos en la formula básica de Taylor, recordemos que obteniamos la derivada de la siguiente manera:

(a) $f(x_0+h)= f(x_0)+ f\,'(x_0)\cdot h+ (f\,''(x_0)\cdot h^2)/2! + (f\,'''(x_0)\cdot h^3)/3! + ...$

(b) $f(x_0-h)= f(x_0) - f\,'(x_0)\cdot h+ (f\,''(x_0)\cdot h^2)/2! - (f\,'''(x_0)\cdot h^3)/3! + ...$

Si realizamos la siguiente operación (a)-(b) obtenemos:

$$f(x_0+h)-f(x_0-h)= 2\cdot f\,'(x_0)\cdot h + (2\cdot f\,'''(x_0)\cdot h^3)/3! + ...$$

Despejando $f\,'(x_0)$ obtenemos:

$$f\,'(x_0)= [f(x_0+h)-f(x_0-h)] / (2\cdot h) + ERROR$$

Si realizamos la operación (a)+(b) obtendremos:

$$f(x_0+h)+f(x_0-h)= 2\cdot f(x_0) + f\,''(x_0)\cdot h^2 + ...$$

Despejando $f\,''(x_0)$ obtenemos:

$$f\,''(x_0)= [f(x_0+h)+f(x_0-h)-2\cdot f(x_0)] / h^2 + ERROR$$

Para obtener la derivada tercera realizaremos la siguiente operación (a)-(b):

$$f(x_0+h)-f(x_0-h)= 2\cdot f\,'(x_0)\cdot h + (2\cdot f\,'''(x_0)\cdot h^3)/3! + ...$$

Despejando $f\,'''(x_0)$ obtenemos:

$$f\,'''(x_0)= 3\cdot[f(x_0+h)-f(x_0-h)- 2\cdot f\,'(x_0)\cdot h] / h^3 + ERROR$$

El principal inconveniente que presenta la derivada tercera con este planteamiento es que también soportamos los errores producidos al calcular la primera derivada.Los resultados que obtenemos son: $f\,'(2)=17$, $f\,''(2)=14$, $f\,'''(2)=6$.

3.- Realizar una comparativa entre los métodos estudiados para realizar la integración numérica de la función $f(x)=x^3+ x^2+x$ en el intervalo [0,2]. Comenta los resultados obtenidos.

Métodos	Resultados
La regla del trapecio	5.205136717093937e-004
La regla del trapecio compuesto	8.66667199999999
La regla del trapecio compuesto con Richardson	8.66666666666668
La regla de Simpson	8.66666666666667
La regla de Simpson compuesto	8.66666666666667
La regla de Romberg	8.66666666666667

Podemos observar que obtenemos resultados muy similares, a excepción de la regla de trapecio, que nos proporciona un resultado que se acerca poco a la realidad, este hecho suele pasar en los métodos que se basan en ideas muy simples, si hacemos un pequeño ejercicio de memoria recordaremos que con la regla del trapecio obteniamos resultados exactos con polinomios a lo sumo de grado uno, en este caso, el polinomio es de grado tres, no podemos esperar más de este método si sobrepasamos su límite, también podemos hacer mención a la optimización que realiza la técnica de Richardson en el caso de la regla del trapecio compuesto, con los mismos datos obtenemos una aproximación muy eficiente.

Del resto de los métodos podemos observar que obtenemos idénticos resultados esto es debido a que la función a integrar no es muy compleja.

4.- Realizar una comparativa entre los métodos estudiados para realizar la integración numérica de la función f(x)=sin(x) en el intervalo [0,2]. Comenta los resultados obtenidos.

Métodos	Resultados
La regla del trapecio	5.409169620541486e-004
La regla del trapecio compuesto	1.41614636449817
La regla del trapecio compuesto con Richardson	1.41614683654715
La regla de Simpson	1.42506045535242
La regla de Simpson compuesto	1.41614683654715
La regla de Romberg	1.41614683651391

Como conclusión podemos obtener:

- Que la regla del trapecio no resulta tan eficiente como pensabamos en un principio, resulta muy sencillo de implementar porque se basa en una idea muy simple y es muy poco porbable que basandose en una idea tan simple obtenga buenos valores.
- Que la regla del trapecio compuesto, sin aplicar la técnica del Richardson para la optimización, no resulta tan deficiente.
- Que si empleamos la regla del trapecio compuesto con Richardson conseguimos un idéntico resultado que con la regla del Simpson compuesta, es simplemente casualidad ya que en el ejemplo anterior obteniamos resultados distintos.
- Que la regla de Simpson obtiene una muy buena primera aproximación.
- Que la regla de Simpson compuesto es un algoritmo muy eficiente, tenemos que tener en cuenta que tanto la regla de Simpson como la regla de Simpson compuesta utilizan Richardson para optimizar su resultado.
- Que la regla de Romberg que también utiliza la técnica de Richardson para su optimización consigue un valor muy cercano a la realidad, la regla de Romberg es realmente eficiente ya que obtiene casisiempre resultados muy buenos.

5.- Realizar una comparativa entre los métodos estudiados para realizar la integración numérica de la función $f(x)=(sen(2x))/(1+x^5)$ en el intervalo [0,3]. Comenta los resultados obtenidos.

Métodos	Resultados
La regla del trapecio	-1.908575807369712e-004
La regla del trapecio compuesto	0.67175637195780
La regla del trapecio compuesto con Richardson	0.70893098097077
La regla de Simpson	0.03226990186081
La regla de Simpson compuesto	0.67175786463067
La regla de Romberg	0.67175036070002

Podemos observar que la regla del trapecio resulta mejor no emplearla porque el resultado que obtenemos está muy lejos de la realidad, en este caso podemos observar que la optimización de Richardson en la regla del trapecio no resulta tán convenienten como se esperaba y resulta ilógico este hecho ya que siempre la técnica de Richardson optimizaba el resultado. También podemos sacar como conclusión que la ragla de Simpson en este caso no resulta tan próxima al valor real, recordemos que con la regla de Simspon obteniamos valores exactos con polinomios de grado tres y en este caso actua un polinomio de grado cinco. Para finalizar podemos observar lo eficiente que son los métodos de la regla de Simpson compuesto y la regla de Romberg.

6.- Realizar una comparativa entre los métodos estudiados para realizar la integración numérica de la función f(x)=sin(x) en el intervalo [0,2]. Comenta los resultados obtenidos.

Métodos	Resultados
La regla del trapecio	5.948729538181567e-004
La regla del trapecio compuesto	3.14155546691103
La regla del trapecio compuesto con Richardson	3.14158751891228
La regla de Simpson	2.97606774342517
La regla de Simpson compuesto	3.14158751891228
La regla de Romberg	3.14132476566914

Como conclusión general de los métodos para la integración numérica podemos obtener: que el método de la regla de trapecio es sencillo de implementar y de desarrollar pero que sirve para poco ya que en los ejemplos puestos anteriormente en ninguno nos ha dado una aproximación cercana al resultado real. Que la regla del trapecio compuesto no resulta tan deficiente si no aplicamos la técnica de Richardson para su optimización. Que con la regla de Simpson obtenemos una muy buena primera aproximación pero sin duda alguna tenemos que volver a resaltar la eficiencia de los métodos de la regla de Simpson compuesto y la regla de Romberg. Son algoritmos que obtenemos muy buenas aproximaciones y no resulta muy complicados de desarrollar.

Práctica 4 Ecuaciones diferenciales.

4.1 Introducción teórica.

Esta última práctica se ocupa del estudio de las ecuaciones diferenciales, las ecuaciones diferenciales se usan para construir modelos matemáticos de problemas de ciencia e ingeniería. Las ecuaciones diferenciales son aquellas funciones donde están involucrada la derivada y el termino independiente.

Nuestro modelo es un problema de valor inicial presentado de la siguiente manera:

$$\begin{cases} y' = f(t,y), t \in [a,b] \\ y(a) = \alpha \end{cases}$$

La primera ecuación proporciona la pendiente de la curva y en cualquier punto t, la segunda de las dos ecuaciones especifica un valor particular de la función y(t).

¿Tendrá solución todo problema de valor inicial que se puede expresar de la forma anterior? No, no todos los problemas del valor inicial tendrá solución, tienen que cumplir:

"Si f es contínua en un rectángulo centrado en (t_0,y_0), digamos: $R=\{(t,y):|t-t_0|\leq\alpha,|y-y_0|\leq\beta,\}$, entonces el problema del valor inicial tiene una solución y(t) para $|t-t_0|\leq min(\alpha,\beta/M)$, donde M es el máximo de $|f(t,y)|$ en el rectángulo R."

Ya sabemos que si puede tener solución si cumple ciertas premisas, pero será única, para que la solución encontrada sea única tiene que cumplir:

"Si f y $\delta f/\delta y$ son contínuas en el rectángulo R, entonces el problema del valor inicial tiene una solución única en el intervalo que incluye a las t que satisfacen $|t-t_0|\leq min(\alpha,\beta/M)$."

Hemos mostrados anteriormente teoremas que especifican que un determinado problema de valor inicial tenga una única solución, a continuación mostraremos la condición de Lipschitz, otro teorema que implica la existencia y la unicidad de una solución al problema del valor inicial.

"Si f es contínua en la franja $a\leq t\leq b$, $-\infty<y<\infty$ y satiface la desigualdad:

$$|f(t,y_1)-f(t,y_2)| \leq L|y_1-y_2|$$

Entonces el problema de valor inicial tiene solución única en el intervalo [a,b]."

4.2 Método de la serie de Taylor.

Para explicar el método de la serie de Taylor nos detendremos antes en el teorema de Taylor que indica; *Supongamos que* $y(t) \in C^{N+1}[t_0,b]$ *y que* $y(t)$ *tiene el desarrollo de Taylor de orden* N *alrededor de un punto* $t=t_k \in [t_0,b]$ *dado por:* $y(t_k+h)=y(t_k)+hT_N(t_k,y(t_k))+O(h^{N+1})$, *donde* $T_N(t_k,y(t_k))=y\cdot t_k\cdot h+ ((y^2\cdot t_k)/2!)h^2+...+((y^n\cdot t_k)/2!)h^n$. *Siendo* $y^{(j)}(t)= f^{(j-1)}(t,y(t))$ *denota la derivada (j-1)-ésima de la función* $f(t,y(t))$ *con respecto a t. Las fórmulas de esta derivadas pueden calcularse recursivamente usando la regla de la cadena:*

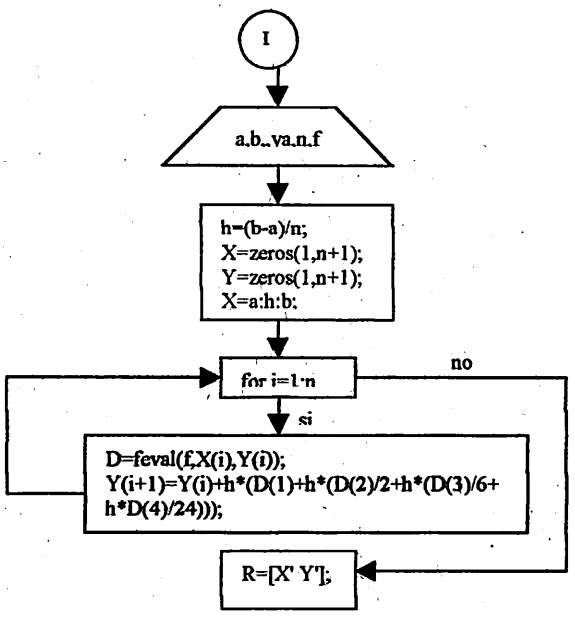
$$y'(t)=f$$
$$y''(t)=f_t+f_y y' = f_t+f_y f$$
$$y'''(t)=f_{tt}+2f_{ty}y'+f_y y''+f_{yy}(y')^2= f_{tt}+2f_{ty}f+f_{yy}f^2+f_y(f_t+f_y f)$$
$$\ldots$$

El valor numérico aproximado de la solución del problema de valor inicial $y'(t)=f(t,y)$ en $[t_0,t_M]$ se calcula usando la fórmula $y(t_k+h)=y(t_k)+hT_N(t_k,y(t_k))+O(h^{N+1})$ en cada subintervalo $[t_k,t_{k+1}]$, de manera que el paso general del método de Taylor de orden N es:

$$y_{k+1}= y_k+ d_1h+ (d_2h^2)/2!+...+ (d_Nh^N)/N!$$
$$d_i= y^{(i)}(t)$$

El método de la serie de Taylor posee una sencillez conceptual acompañado de su potencial para obtener una precisión muy alta en los resultados (el método de Taylor de orden N tiene la propiedad de que el error global final es de orden $O(h^{N+1})$, por tanto, se pueden elegir N de manera que este error sea tan pequeño como queramos) hacen que este método sea realmente eficiente, por contra, los cálculos en cada paso se vuelven más engorrosos.

Como viene siendo habitual, mostraremos a continuación su diagrama de flujo:



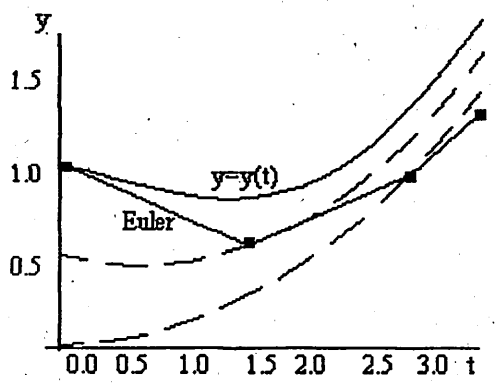
4.3 Método de Euler.

El método de Euler es el método de la serie de Taylor para n=1. Su expresión formal es la siguiente:

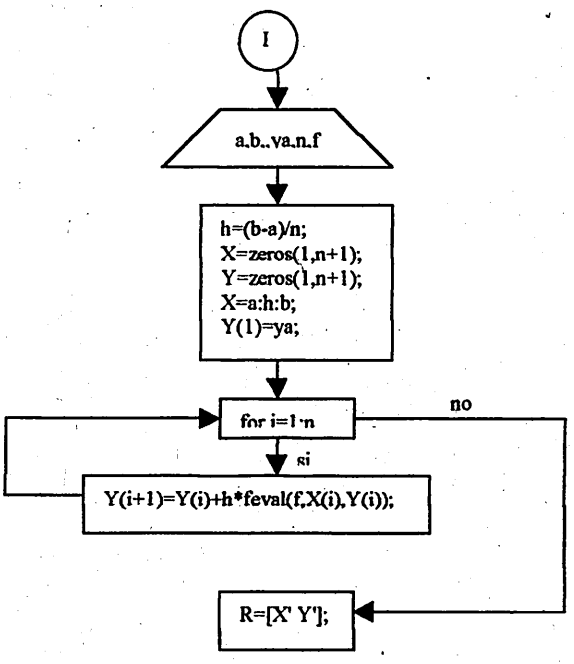
$$w_0=\alpha$$
$$w_{i+1}= w_i+hf(t_i,w_i)\ i=1,2,...n$$

Esta fórmula tiene la ventaja de que no necesita tomar ninguna derivada de f, sin embargo dihca ventaja se neutraliza por la necesidad de tomar pequeños valores de h para obtener una precisión aceptable.

Para comprenderlo mejor describiremos geométricamente el método de Euler. Si partimos del punto $[t_0,y_0]$, calculamos el valor de la pendiente $m_0=f(t_0,y_0)$, nos movemos horizontalmente una distancia h y verticalmente una distancia $hf(t_0,y_0)$, entonces lo que hacemos es desplazarnos al punto (t_1,y_1), usaremos este valor para calcular el siguiente punto, así sucesivamente, gráficamente sería:



Su diagrama de flujo es el siguiente:

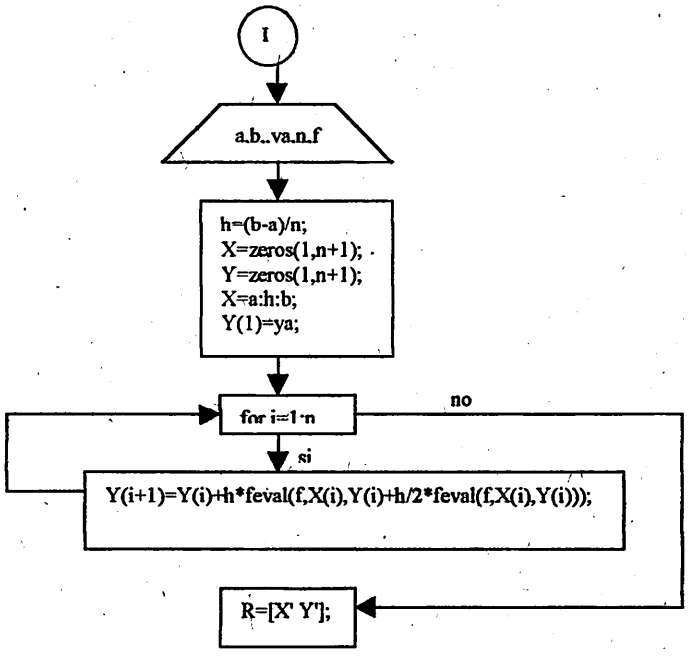


4.4 Método del punto medio.

Sea h=(b-a)/n y t_i=a+ih para (i=0,1,...,n). El método del punto medio construye $w_i=w(t_i,h)\approx y(t_i)$ para i=1,2,...,n donde el error de truncamiento local es de $O(h^2)$.

$$\begin{cases} w_0 = \alpha \\ w_{i+1} = w_i + hf\left(t_i + \frac{h}{2}, w_i + \frac{h}{2} f(t_i, w_i)\right) \\ i = 1,2,\ldots,n \end{cases}$$

Su diagrama de flujo es:



El método del punto medio es un método muy similar al método de Euler, en cambio no resulta tan eficiente como este ya que realiza un mayor número de operaciones y consigue una precisión inferior a la que obteníamos con el método de Euler.

4.4 Método de Heun.

El método de Heun introduce una idea nueva en la construcción de un algoritmo para resolver el problema del valor inicial, se basa en el teorema fundamental del cálculo e integrar, el método de Heun tiene error de truncamiento local es de $O(h^2)$ y se especifica de la siguiente manera:

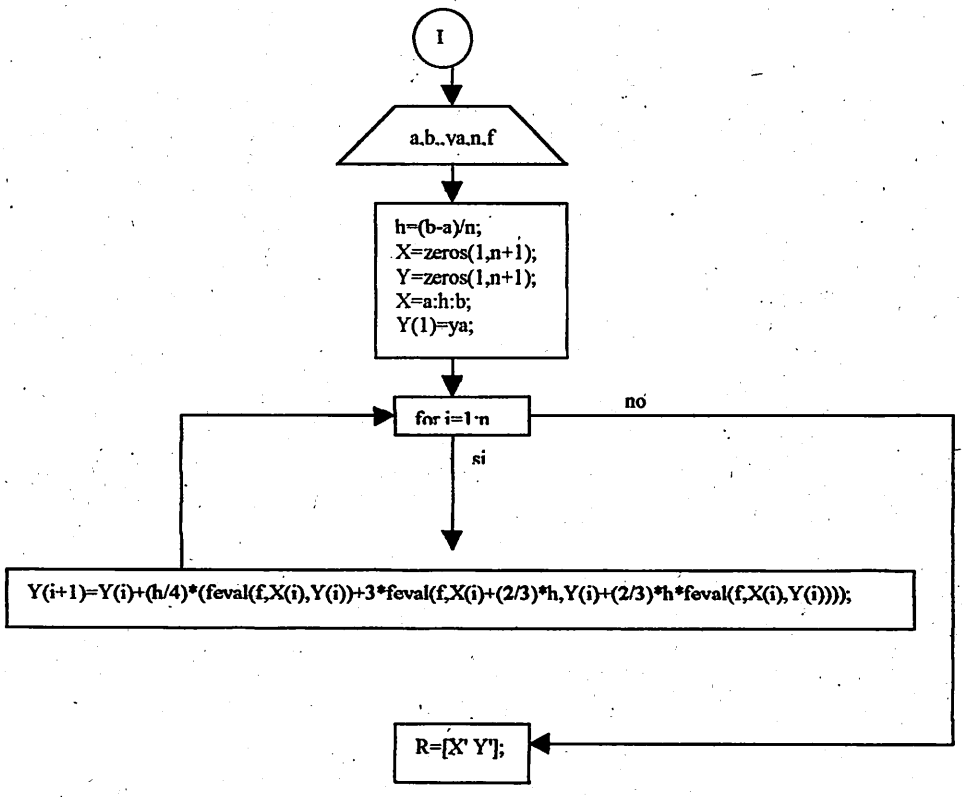
Sea $h=(b-a)/n$ y $t_i=a+ih$ para $(i=0,1,...,n)$. El método de Heun construye $w_i=w(t_i,h)\approx y(t_i)$.

$$w_0=\alpha$$

$$w_{i+1}= w_i+\frac{h}{4}\cdot[f(t_i,w_i)+3f\cdot(t_i+\frac{2h}{3}, w_i+\frac{2h}{3} f(t_i,w_i))]$$

$$i=1,2,...n$$

Su diagrama de flujo es:



4.5 Método de Runge-Kutta clásico.

Los métodos de Runge-Kutta se construyen a partir de un método de Taylor, de tal manera que el error global final sea del mismo orden $O(h^n)$, aunque estos métodos se pueden construir para cualquier orden N, nosotros estudiaremos el del orden N=4, también conocido como Runge-Kutta clásico. A continuación presentamos las expresiones del método de Runge-Kutta de orden 4 o clásico:

$$w_0=\alpha$$

$$w_{i+1}= w_i+\frac{k_1+2k_2+2k_3+k_4}{6}$$

$$k_1=hf(x_i,w_i),$$

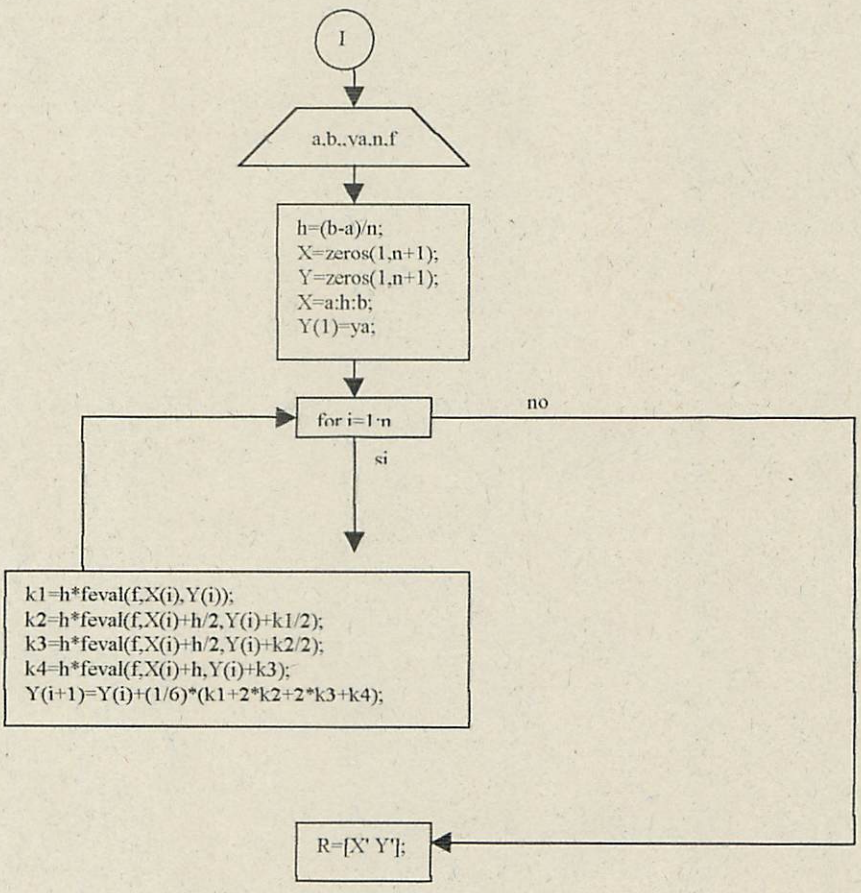
$$k_2=hf(x_i+\frac{h}{2},w_i+\frac{k_1}{2}),$$

$$k_3=hf(x_i+\frac{h}{2},w_i+\frac{k_2}{2}),$$

$$k_4=hf(x_{i+1},w_i+k_3),$$

$$i=1,2,...n$$

Su diagrama de flujo es el siguiente:



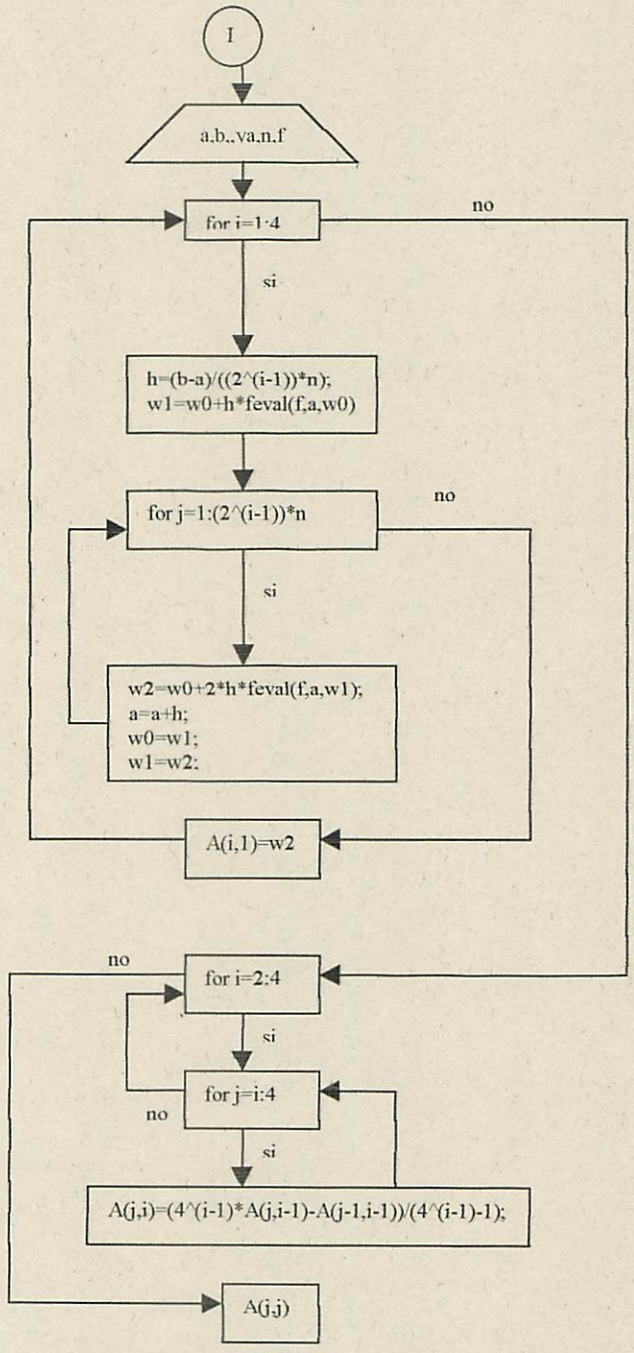
4.6 Método de Gragg.

El método de Gragg es un método multipaso donde el primer paso es la primera aproximación y el segundo paso se obtiene empleando el método de Euler, el método de Gragg viene definido por la siguiente expresión:

$w_0=\alpha$
$w_1= w_0+h\ f(t_0,w_0)$ (Euler)
$w_{i+1}= w_{i-1}+2h\ f(t_i,w_i)$
$i=1,2,...n$

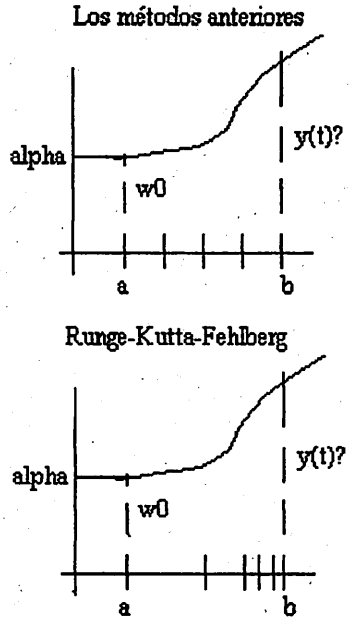
Si aplicamos Richardson conseguiremos una mejor optimización ya que conocemos el tipo de error ($k_1h^2+ k_2h^4+k_3h^6+...$)que comete el método de Gragg.

A continuación y como viene siendo habitual mostraremos el diagrama de flujo:



4.7 Método de Runge-Kutta-Fehlberg.

El método de Runge-Kutta-Fehlberg supone un esfuerzo por diseña un procedimiento para que ajuste automáticamente la longitud del paso, a diferencia de los métodos anteriormente vistos este porpone no tomar un paso constante sino irlo modificando según las necesidades.



Fehlberg en 1969 recurrió a un método de cuarto orden con cinco evaluaciones de la función y a un método de quinto orden con seis evaluaciones de la función. Podemos suponer que no resultaría muy eficiente ya que su método requería una gran cantidad de cálculo pero Fehlberg se la ingenió para elegir los parámetros en estos métodos de manera que obtenía dos fórmulas de distinto orden utilizando los mismos puntos para las evaluaciones de la función. Con ello sólo necesita seis evaluaciones de la función, estas evaluaciones son las siguientes:

$$k_1 = hf(t_k, y_k),$$

$$k_2 = hf\left(t_k + \frac{1}{4}h, y_k + \frac{1}{4}k_1\right),$$

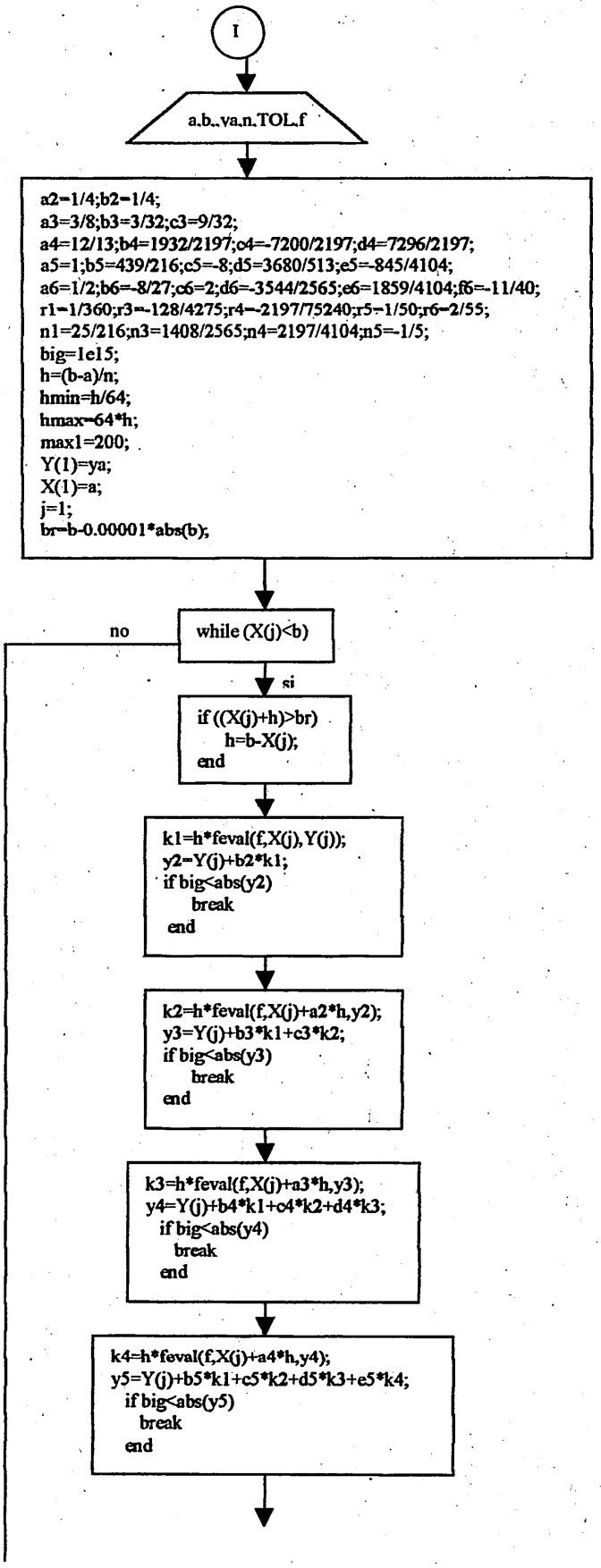
$$k_3 = hf\left(t_k + \frac{3}{8}h, y_k + \frac{3}{32}k_1 + \frac{9}{32}k_2\right),$$

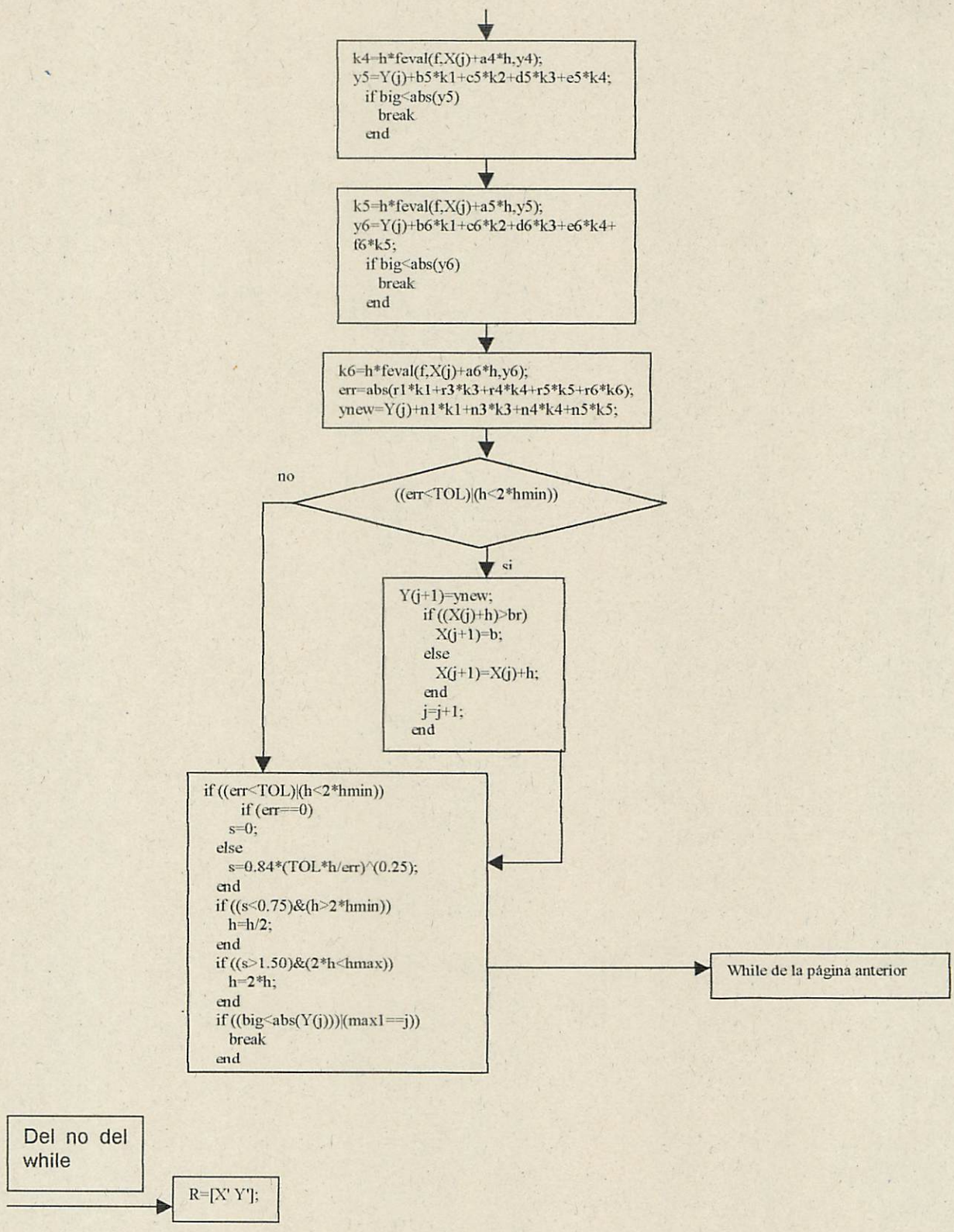
$$k_4 = hf\left(t_k + \frac{12}{13}h, y_k + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right),$$

$$k_5 = hf\left(t_k + h, y_k + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right),$$

$$k_6 = hf\left(t_k + \frac{1}{2}h, y_k - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right),$$

Su diagrama de flujo es el siguiente:




k4=h*feval(f,X(j)+a4*h,y4);
y5=Y(j)+b5*k1+c5*k2+d5*k3+e5*k4;
if big<abs(y5)
break
end
k5=h*feval(f,X(j)+a5*h,y5);
y6=Y(j)+b6*k1+c6*k2+d6*k3+e6*k4+
f6*k5;
if big<abs(y6)
break
end
k6=h*feval(f,X(j)+a6*h,y6);
err=abs(r1*k1+r3*k3+r4*k4+r5*k5+r6*k6);
ynew=Y(j)+n1*k1+n3*k3+n4*k4+n5*k5;
no
((err<TOL)|(h<2*hmin))
si
Y(j+1)=ynew;
if ((X(j)+h)>br)
X(j+1)=b;
else
X(j+1)=X(j)+h;
end
j=j+1;
end
if ((err<TOL)|(h<2*hmin))
if (err==0)
s=0;
else
s=0.84*(TOL*h/err)^(0.25);
end
if ((s<0.75)&(h>2*hmin))
h=h/2;
end
if ((s>1.50)&(2*h<hmax))
h=2*h;
end
if ((big<abs(Y(j)))|(max1==j))
break
end
While de la página anterior
Del no del while
R=[X' Y'];

4.8 Método de Adams-Bashforth-Moulton.

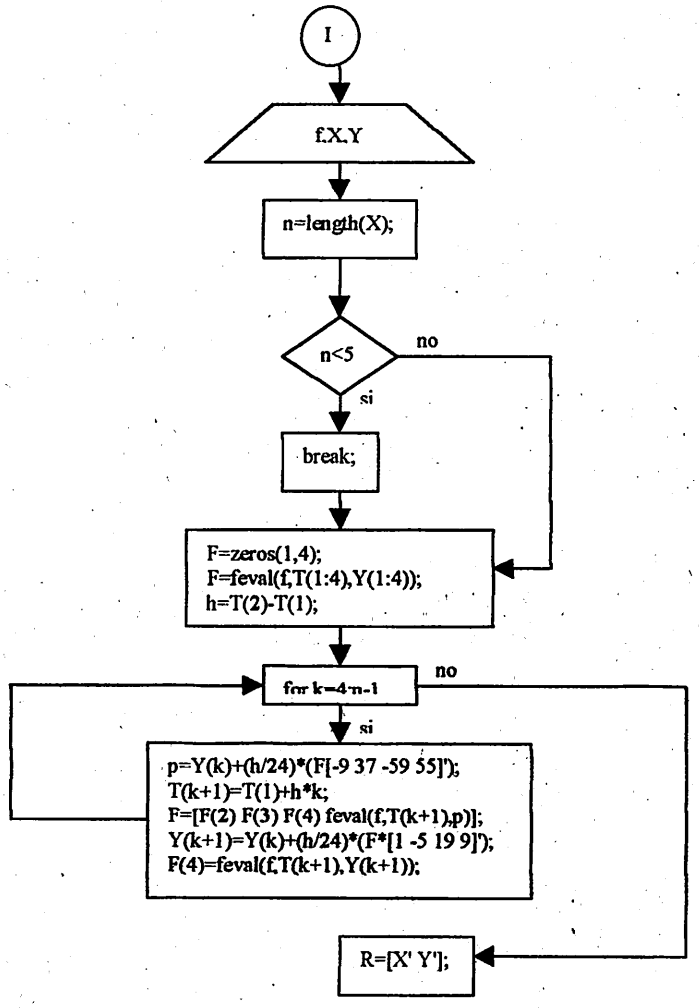
El método de Adams-Bashforth-Moulton es un método multipaso, al igual que el método de Gragg visto anteriormente, el método de Adams-Bashforth-Moulton construye las aproximaciones a la solución del problema de valor inicial $y'=f(t,y)$ con $y(\alpha)=y_0$ en [a,b] usando el método de Adams-Bashforth para obtener el valor predictor:

$$p_{k+1}=y_k+\frac{h}{24}(-9\,f_{k-3}+37\,f_{k-2}-59\,f_{k-1}+55\,f_k)$$

Y utilizando el método de Adams-Moulton para obtener el valor corrector:

$$y_{k+1}=y_k+\frac{h}{24}(-9\,f_{k-2}-5\,f_{k-1}+19\,f_k+9\,f_{k+1})$$

Su diagrama de flujo es el siguiente:



4.9 Comparaciones y conclusiones.

MÉTODOS	CARACTERÍSTICAS
Método de la serie de Taylor	-Se basa en el teorema de Taylor. -Utiliza la derivada de la función con lo que implica una gran cantidad de operaciones. -Su error es de orden $O(h^{N+1})$.
Método de Euler	-Es el método de la serie de Taylor para N=1. -No necesita tomar ninguna derivada de f. -Necesita tomar pequeños valores de h para obtener una precisión aceptable.
Método del punto medio	-No requiere de una gran cantidad de operaciones para obtener unas aceptables aproximaciones. -Su error es de orden $O(h^2)$.
Método de Heun	-Se basa en el teorema fundamental de cálculo y en el teorema fundamental de la integral. -Su error es del mismo orden que el del método del punto medio, es decir, del orden $O(h^2)$.
Método de Runge-Kutta clásico	-Se construye a partir del método de la serie de Taylor. -Se pueden construir de cualquier orden aunque el método de Runge-Kutta de orden cuatro es el más clásico y es el que hemos analizado en esta práctica.
Método de Gragg.	-Método multipaso, es decir, necesita más de una aproximación (pasos), para construir la aproximación final. -El segundo paso lo obtiene a través del método de Euler.
Método de Runge-Kutta-Fehlberg	-Es el único método estudiado en esta práctica donde la longitud del paso varía. -Se basa en el método de Runge-Kutta. -Resulta un poco complicado de desarrollar.
Método de Adams-Bashforth-Moulton	-Método multipaso, al igual que el método de Gragg. -Utiliza el método de Adams-Bashforth para obtener el valor predictor y utiliza el método de Adams-Moulton para obtener el valor corrector.

4.10 Ejemplos.

1.- Dado el siguiente problema: $y'=-y+t^2+1$, $y(0)=1$ $0\leq t\leq 1$. Calcular y(1) con h=0.01, utilizando para ello, los métodos anteriomente vistos.

A continuación mostraremos una tabla con los resultados obtenidos:

Método	Nº de Op.	Número de pasos	y(1)
Taylor	1512	100	1.25842580257264
Euler	612	100	1.26159564086627
Punto medio	1212	100	1.26056334382853
Runge-Kutta	3512	100	1.26424111772764
Heun	2212	100	1.26424993892880
Gragg	10618	100	1.25693177952969
Runge-Kutta-Fehlberg	781	9	1.26423523100905
Adams-Bashforth-Moulton	3409	100	1.26424111785766

A primera vista podemos observar que los resultados que obtenemos son muy parecidos, también podemos ver que el método de Euler y el método de Runge-Kutta-Fehlberg son los que realizan menos operaciones que el resto, en cambio hay que resaltar que el método de Gragg es el proceso que más operaciones realiza, unas 10618, para conseguir una aproximación no tan buena como el resto de los métodos.

Otro aspecto a resaltar es el número de pasos, ya vimos que salvo el método de Runge-Kutta-Fehlberg todos tomaban un h constante y por lo tanto no modificaban la longuitud de su pasos, en cambio el método de Runge-Kutta-Fehlberg se atreve a modificarlo y obtiene una muy buena aproximación con tan sólo nueve pasos.

Elegir el método más eficiente es realmente complicado ya que cada uno tiene su pos y sus contra, por ejemplo Taylor es un buen método pero necesita hacer uso de las derivadas, cosa que no necesita el método de Euler, si para elegir el método más óptimo nos basaramos en el número de operaciones que realiza para conseguir la aproximación, la elección estaría entre el método de Euler y el método de Runge-Kutta-Fehlberg, pero considerando otros aspectos crremos que el método más eficiente de los estudiados en esta sección es el método de Runge-Kutta-Fehlberg ya que no sólo realiza pocas operaciones para obtener la aproximación y(1) sino que tán sólo realiza nueve pasos en comparación con el resto que realizan cien pasos.

2.- Dado el siguiente problema: $y'=t^2-y$, $y(0)=1$ $y(t)=-e^{-t}+t^2-2t+2$ $0\leq t\leq 1$. Calcular y(1) con h=0.01 utilizando para ello, los métodos anteriomente vistos.

A continuación mostraremos una tabla con los resultados obtenidos:

Método	Nº de Op.	Número de pasos	y(1)
Taylor	1412	100	0.62445814384587
Euler	1212	100	0.62528642928524
Punto medio	2412	100	0.62528642928524
Runge-Kutta	5912	100	0.63212055883075
Heun	4012	100	0.63212056175601
Gragg	19642	100	0.61846154847537
Runge-Kutta-Fehlberg	1069	9	0.63212051002010
Adams-Bashforth-Moulton	4594	100	0.63212055898893

Al igual que antes los métodos obtienen aproximaciones muy parecidas, al igual que antes los métodos de Euler y Runge-Kutta-Fehlberg son los métodos que menos operaciones realizan, por contra sigue siendo el método de Gragg el que más requiere, un aspecto importante que tenemos que resaltar es que el método de la serie de Taylor en este caso realiza muy pocas operaciones, el principal motivo de este hecho es que las derivadas tercera y cuarta son nulas.

3.- Dado el siguiente sistema de ecuaciones diferenciales, resolverlo por Runge-Kutta mostrando las 5 primeras iteraciones.

x'(t)=x+2y x(0)=6

y'(t)=3x+2 y(0)=4

A continuación mostraremos una tabla con los resultados obtenidos:

k	t_k	$x_k \cdot 10000$	$y_k \cdot 10000$
0	0.00	0.00060000000000	0.00040000000000
1	0.02	0.00062935455067	0.00045393248933
2	0.04	0.00066156221225	0.00051194859878
3	0.06	0.00069685252789	0.00057439652500
4	0.08	0.00073547431874	0.00064165330487
5	0.10	0.00077769728646	0.00071412722062

Para aplicar el método de Runge-Kutta hemos tenido que hacer unas pequeñas modificaciones, para el tratamiento de un sistema de ecuaciones diferenciales lo hemos tratado igual que si fuera una ecuación diferencial teniendo en cuenta que la solución no era un escalar valor sino un vector.

4.- Dado la siguiente ecuación diferencial de orden superior, resolverlo por Runge-Kutta, explicar como se ha resuelto, mostrar las 5 primeras iteraciones.

x''(t)+4x'(t)+5x(t)=0 x(0)=3 x'(0)=-5

Para resolver esta ecuación la vamos a transformar en dos ecuaciones de primer orden, que si sabemos como se resuelve.

y=x' y'=x''=-5x-4y

A continuación mostraremos una tabla con los resultados obtenidos:

t_k	x_k	y_k
0	3.00000000000000	-5.00000000000000
0.10000000000000	2.52564583333333	-4.48185416666667
0.20000000000000	2.10402783255208	-3.95059997647569
0.30000000000000	1.73506268505788	-3.43236151975128
0.40000000000000	1.41653369346863	-2.94411599507803
0.50000000000000	1.14488509380768	-2.49601256867580
1.00000000000000	0.33324302192576	-0.93498293635178
2.00000000000000	-0.00620684405292	-0.04516717443284
3.00000000000000	-0.00701079126808	0.01051787885985
4.00000000000000	-0.00091163015844	0.00236540065916
5.00000000000000	-0.00000492598244	0.00015330540923

Aunque no era necesario también hemos mostrado $y_k=x'$, en este caso no era necesario modificar el método de Runge-Kutta ya que hemos transformado la ecuación diferencial de orden superior en un sistema de ecuaciones de primer orden de esa manera si lo podemos resolver. Teniendo en cuenta que como resultado obtenemos un vector es la primera coordenada la que nos interesa, aunque en este caso hemos mostrado todo el vector.

Apéndice A Código fuente.

PRÁCTICA Nº1:RESOLUCIÓN DE ECUACIONES.

Método de bisección:

```
function sol=bisec(a,b,TOL,Nmax,f);
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
if (b<=a)
  error('El extremo superior no puede ser menor o igual que extremo inferior.');
end
%Comprobación de que los extremos no son solución.
if (feval(f,a)==0)
  sol=a;
  break;
end
if (feval(f,b)==0)
  sol=b;
  break;
end
%Comprobación de si se satisface la condicion f(a)*f(b)<0.
if (feval(f,a)*feval(f,b) > 0)
  error('El intervalo podría no tener solución');
end
%Buscamos la solución.
for n=1:Nmax
   p(n)=(a+b)/2;
   if (feval(f,a)*feval(f,p(n)) < 0)
     b=p(n);
   else
     a=p(n);
   end
   err=abs(b-a);
   if (err<TOL)  %Criterios de paro
     break;
   end
end
sol=(a+b)/2;
%Imprimir resultados.
disp('Para bisección')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol
  disp('Iteraciones')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p);
```

Método de regula:

```
function sol=regula(a,b,TOL,Nmax,f);
format long %Para obtener más números decimales.
flops(0);  %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
if (b<=a)
  error('El extremo superior no puede ser menor o igual que extremo inferior.');
end
%Comprobación de que los extremos no son solución.
if (feval(f,a) == 0)
  disp('Se ha encontrado la raíz, siendo esta')
  ,a
  break;
end
if (feval(f,b) == 0)
  disp('Se ha encontrado la raíz, siendo esta')
  ,b
  break;
end
%Comprobación de si se satisface la condicion f(a)*f(b)<0.
if (feval(f,a)*feval(f,b) > 0)
  error('El intervalo podría no tener solución');
end
%Buscamos la solución.
n=1;
while (n<Nmax)
   p(n)=(a*feval(f,b)-b*feval(f,a))/(feval(f,b)-feval(f,a));
   if (feval(f,a)*feval(f,p(n))<0)
     b=p(n);
   else
     a=p(n);
   end
   n=n+1;
   p(n)=(a*feval(f,b)-b*feval(f,a))/(feval(f,b)-feval(f,a));
   err=abs(p(n-1)-p(n));
   if (err<TOL)
     break;
   end
end
sol=p(n);
%Imprimir resultados.
disp('Para regula falsi')
if (n>=Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol
  disp('Iteraciones')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p);
```

Método de aproximación a las raíces:

```
function sol=aprox(a,b,tol1,tol2,f)
format long        %Para obtener más números decimales.
flops(0);          %Inicializa el contador de nº de operaciones por segundo.
```

```
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
x=a:tol1:b;
y=feval(f,x);
rangoy=max(y)-min(y);
tol=rangoy*tol2;
n=length(x);
m=0;
x(n+1)=x(n);
y(n+1)=y(n);
for k=2:n,
  if (y(k-1)*y(k)<=0)
    m=m+1;
    p0(m)=(x(k-1)+x(k))/2;
  end
  s=(y(k)-y(k-1))*(y(k+1)-y(k));
  if (abs(y(k))<tol) & (s<=0)
    m=m+1;
    p0(m)=x(k);
  end
end
sol=p0(m);
%Imprimir resultados.
disp('Para localización de aproximación de raíces')
disp('La raíz más aproximada es')
,sol
disp('Número de operaciones realizadas'),flops
t=etime(clock,tiempo_inicial);
disp('Tiempo transcurrido')
,t
plot(p0);
```

Método del punto fijo:

```
function sol=pfijo(p0,TOL,Nmax,g,gp)
%PUNTO FIJO: Algoritmo para calcular la raiz de f(x) necesita una g(x).
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Comprobación de la calidad de g(x).
%La función gp(x) es la derivada de g(x).
if (abs(feval(gp,p0))<1)
  disp('La función g(x) es buena.')
else
  disp('La función g(x) es mala.')
  break;
end
%Buscamos la solución.
p(1)=p0;
n=2;
while (n<=Nmax)
    p(n)=feval(g,p(n-1));
    err=abs(p(n)-p(n-1)); %Error relativo relerr=2*err/(abs(p)+TOL);
    if (err<TOL)
          break
    end
    n=n+1;
end
%Imprimir resultados.
disp('Para punto fijo')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
```

```
 disp('La raíz más aproximada es')
 ,sol=p(n)
 disp('Iteraciones:')
 ,n
 disp('Número de operaciones realizadas')
 ,flops
 t=etime(clock,tiempo_inicial);
 disp('Tiempo transcurrido')
 ,t
end
plot(p);
```

Método del Newton:

```
function sol=newton(p0,TOL,Nmax,f,fp)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Buscamos la solución.
%La función fp(x) es la derivada de f(x).
n=1;
p(n)=p0;
while (n<Nmax)
  n=n+1;
  p(n)=p(n-1)-feval(f,p(n-1))/feval(fp,p(n-1));
  err=abs(p(n)-p(n-1));
  if (err<TOL)
    break;
  end
end
%Imprimir resultados.
disp('Para Newton-Raphson')
if (n>=Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol=p(n)
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p);
```

Método de la secante:

```
function sol=secante(p0,p1,TOL,Nmax,f)
format long %Para obtener más números decimales.
flops(0);  %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Buscamos la solución.
n=1;
p(n)=p1-(feval(f,p1)*(p1-p0))/(feval(f,p1)-feval(f,p0));
while (n<Nmax)
  p0=p1;
  p1=p(n);
  err=abs(p(n)-p1);
  if (err<TOL)
    break;
  end
  n=n+1;
```

```
 p(n)=p1-(feval(f,p1)*(p1-p0))/(feval(f,p1)-feval(f,p0));
end
%Imprimir resultados.
disp('Para el método secante')
if (n>=Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol=p(n)
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p)
```

Método de Aitken:

```
function sol=aitken(p0,TOL,Nmax,f,g)
format long; %Para obtener más números decimales.
flops(0);  %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Buscamos la solución.
n=1;
p1=feval(g,p0);%Aplico punto fijo.
while (n<=Nmax)
  p2=feval(g,p1);%Aplico punto fijo.
  %Prevención de las divisiones entre cero.
  if (abs(p2-2.*p1+p0)<TOL)
    disp('El denominador es casi nulo.')
    p(n)=p0-((p1-p0).^2/(p2-2.*p1+p0));
    break;
  end
  p(n)=p0-((p1-p0).^2/(p2-2.*p1+p0));
  if (abs(feval(f,p(n)))<TOL)
    break;
  end
  p0=p1;
  p1=p2;
  n=n+1;
end;
%Imprimir resultados.
disp('Para Aitken')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol=p(n)
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p);
```

Método de Steffensen:

```
function sol=steffens(p0,TOL,Nmax,f,g)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Buscamos la solución.
n=1;
while (n<=Nmax)
  p1=feval(g,p0);%Aplico punto fijo.
  p2=feval(g,p1);%Aplico punto fijo.
  %Prevención de las divisiones entre cero.
  if (abs(p2-2*p1+p0)<TOL)
    disp('El denominador es casi nulo.')
    p(n)=p0-(p1-p0)^2/(p2-2*p1+p0);
    break;
  end
  p(n)=p0-(p1-p0)^2/(p2-2*p1+p0);%Aplico Aitken.
  if (abs(p(n)-p0)<TOL & abs(feval(f,p(n)))<TOL)
     %Solución encontrada.
     break;
  end
  p0=p(n);
  n=n+1;
end
%Imprimir resultados.
disp('Para Steffensen')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol=p(n)
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
plot(p);
```

PRÁCTICA Nº2:RESOLUCIÓN DE SISTEMAS DE ECUACIONES.

Método de triangularización superior seguido de sustitución regresiva:

```
function X=trisusureg(A,B)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
%Inicializamos X y una matriz C que sirve de almacén temporal
[N N]=size(A);
X=zeros(N,1);
C=zeros(1,N+1);
%Calculo de la matriz ampliada Aug=[AIB]
Aug=[A B];
for q=1:N-1
  %Pivoteo parcial en la columna q-ésima
  [Y,j]=max(abs(Aug(q:N,q)));
  %Intercambiamos las filas q-ésima y (j+q-1)-ésima
  C=Aug(q,:);
  Aug(q,:)=Aug(j+q-1,:);
  Aug(j+q-1,:)=C;
  if Aug(q,q)==0
```

```
    disp('A es singular.No hay solución o no es única.')
    break;
  end
  %Proceso de eliminación en la columna q-ésima
  for k=q+1:N
    m=Aug(k,q)/Aug(q,q);
    Aug(k,q:N+1)=Aug(k,q:N+1)-m*Aug(q,q:N+1);
  end
end
%Sustitución regresiva
Y=Aug(1:N,1:N);
Z=Aug(1:N,N+1);
n=length(Z);
X=zeros(n,1);
X(n)=Z(n)/Y(n,n);
for k=n-1:-1:1
  X(k)=(Z(k)-Y(k,k+1:n)*X(k+1:n))/Y(k,k);
end
disp('Número de operaciones realizadas')
,flops
t=etime(clock,tiempo_inicial);
disp('Tiempo transcurrido')
,t
disp('La raíz más aproximada es')
```

Método de factorización LU:

```
function X=lufatc(A,B)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
[N,N]=size(A);
X=zeros(N,1);
Y=zeros(N,1);
C=zeros(1,N);
R=1:N;
for q=1:N-1
  %Determinación de la fila pivote para la columna q-ésima
  [max1,j]=max(abs(A(q:N,q)));
  %Intercambio de las filas q-ésima y j-ésima
  C=A(q,:);
  A(q,:)=A(j+q-1,:);
  d=R(q);
  R(q)=R(j+q-1);
  R(j+q-1)=d;
  if A(q,q)==0
    disp('A es singular, no hay solución o no es única');
    break;
  end
  %Cálculo del multiplicador, que se guarda en la parte subdiagonal de A
  for k=q+1:N
    mult=A(k,q)/A(q,q);
    A(k,q)=mult;
    A(k,q+1:N)=A(k,q+1:N)-mult*A(q,q+1:N);
  end
end
%Resolución para hallar Y
Y(1)=B(R(1));
for k=2:N
  Y(k)=B(R(k))-A(k,1:k-1)*Y(1:k-1);
end
%Resolución para hallar X
X(N)=Y(N)/A(N,N);
for k=N-1:-1:1
```

```
   X(k)=(Y(k)-A(k,k+1:N)*X(k+1:N))/A(k,k);
end
disp('Número de operaciones realizadas')
,flops
t=etime(clock,tiempo_inicial);
disp('Tiempo transcurrido')
,t
disp('La raíz más aproximada es')
```

Método de jacobi:

```
function X=jacobi(A,B,P,TOL,Nmax)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
N=length(B);
for k=1:Nmax
   for j=1:N
      X(j)=(B(j)-A(j,[1:j-1,j+1:N])*P([1:j-1,j+1:N]))/A(j,j);
   end
   err=abs(norm(X'-P));
   relerr=err/(norm(X)+eps);
   P=X';
   if (err<TOL)|(relerr<TOL)
     break;
   end;
end
disp('Número de operaciones realizadas')
,flops
t=etime(clock,tiempo_inicial);
disp('Tiempo transcurrido')
,t
disp('La raíz más aproximada es')
,X=X';
```

Método de gauss-seidel:

```
function X=gseid(A,B,P,TOL,Nmax)
format long;        %Para obtener más números decimales.
flops(0);           %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
N=length(B);
for k=1:Nmax
   for j=1:N
      if j==1
        X(1)=(B(1)-A(1,2:N)*P(2:N))/A(1,1);
      elseif j==N
        X(N)=(B(N)-A(N,1:N-1)*(X(1:N-1))')/A(N,N);
      else
        X(j)=(B(j)-A(j,1:j-1)*X(1:j-1)-A(j,j+1:N)*P(j+1:N))/A(j,j);
      end
   end
   err=abs(norm(X'-P));
   relerr=err/(norm(X)+eps);
   P=X';
   if (err<TOL)|(relerr<TOL)
     break;
   end;
end;
disp('Número de operaciones realizadas')
,flops
t=etime(clock,tiempo_inicial);
disp('Tiempo transcurrido')
,t
disp('La raíz más aproximada es')
,X=X';
```

Método de punto fijo para sistemas:

```
function pfijosis(p0,TOL,Nmax,gsis)
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
sol=feval(gsis,p0);
n=1;
while (norm(p0-sol,2)>=TOL) %norm(p0-sol,inf)>=TOL)
  p0=sol;
  sol=feval(gsis,p0);
  n=n+1;
  if n>=Nmax
     break
  end
end
%Imprimir resultados.
disp('Para punto fijo')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol'
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
```

Método de newton para sistemas:

```
function newsis(p0,TOL,Nmax,fsis,jacfsis)
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
n=1;
z=-(feval(jacfsis,p0))\(feval(fsis,p0))';
while (norm(z,2)>=TOL) %(norm(z,inf)>=TOL)
  p0=p0+z';
  z=-(feval(jacfsis,p0))\(feval(fsis,p0))';
  n=n+1;
  if n>=Nmax
     break
  end
end
sol=p0+z';
%Imprimir resultados.
disp('Para newton sistemas')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol'
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
```

```
  disp('Tiempo transcurrido')
  ,t
end
```

Método de seidel:

```
function seidel(p0,TOL,Nmax,gseidel)
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
sol=feval(gseidel,p0);
n=1;
while (norm(p0-sol,2)>=TOL) %norm(p0-sol,inf)>=TOL)
  p0=sol;
  sol=feval(gseidel,p0);
  n=n+1;
  if n>=Nmax
    break
  end
end
%Imprimir resultados.
disp('Para seidel')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol'
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
```

Método de broyden:

```
function broyden(p0,TOL,Nmax,fsis,jacfsis)
format long          %Para obtener más números decimales.
flops(0);            %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
B=feval(jacfsis,p0);
z=-B\(feval(fsis,p0))';
n=1;
while (norm(z,2)>=TOL) %(norm(z,inf)>=TOL)
  p1=p0;
  p0=p0+z';
  B=B+1/norm(p0-p1,2)*((feval(fsis,p0))'-(feval(fsis,p1))'-B*(p0-p1)')*(p0-p1);
  z=-B\(feval(fsis,p0))';
  n=n+1;
  if n>=Nmax
    break
  end
end
sol=p0+z';
%Imprimir resultados.
disp('Para broyden')
if (n>=Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
```

```
 ,sol'
 disp('Iteraciones:')
 ,n
 disp('Número de operaciones realizadas')
 ,flops
 t=etime(clock,tiempo_inicial);
 disp('Tiempo transcurrido')
 ,t
end
```

Método de máximo descenso:

```
function sol=maxdes(p0,TOL,Nmax,hsis,fsis,jacfsis);
format long        %Para obtener más números decimales.
flops(0);          %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
m=feval(hsis,'fsis',p0);
n=0;
while (m>TOL & n<=Nmax)
  i=0;
  landa=1/(2.^i);
  m1=m;
  % ahora se busca la direccion de maximo descenso
  while m1>=m & landa>=TOL
    aux=landa*gradh(fsis,jacfsis,p0);
    m1=feval(hsis,'fsis',p0-aux);
    i=i+1;
    landa=1/(2.^i);
  end;
  %el nuevo p0 es el que hizo h(p0-lambda*gradh(p0)) menor que m
  p0=p0-landa*gradh(fsis,jacfsis,p0);
  m=feval(hsis,'fsis',p0);
  n=n+1; % cuenta el numero de veces que intenta busca el p0
end;
sol=p0;
%Imprimir resultados.
disp('Para máximo descenso')
if (n>=Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol=sol'
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
```

Método de newton global:

```
function nglobal(p0,TOL,Nmax,hsis,fsis,jacfsis)
format long        %Para obtener más números decimales.
flops(0);          %Inicializa el contador de nº de operaciones por segundo.
tiempo_inicial=clock; %Necesario para saber el tiempo transcurrido.
m=feval(hsis,'fsis',p0);
n=0;
while m>TOL & n<=Nmax
  i=0;
  landa=1/(2.^i);
  m1=m;
  z=-(feval(jacfsis,p0))\(feval(fsis,p0))';
```

```
  while m1>=m & landa>=TOL
    m1=feval(hsis,'fsis',p0+landa*z');
    i=i+1;
    landa=1/(2.^i);
  end;
  %El nuevo x0 es el que hizo h(p0-lambda*gradh(p0)) menor que m
  p0=p0+landa*z';
  m=feval(hsis,'fsis',p0);
  n=n+1;
end;
sol=p0;
%Imprimir resultados.
disp('Para newton sistemas')
if (n>Nmax)
  disp('El método no converge para ')
  ,Nmax
  disp('iteraciones con la condición inicial y TOL dada.')
else
  disp('La raíz más aproximada es')
  ,sol'
  disp('Iteraciones:')
  ,n
  disp('Número de operaciones realizadas')
  ,flops
  t=etime(clock,tiempo_inicial);
  disp('Tiempo transcurrido')
  ,t
end
```

PRÁCTICA Nº3:INTERPOLACIÓN POLINOMIAL.DERIVACIÓN E INTEGRACIÓN NUMÉRICA.

Método de lagrange:

```
function C=lagrange(X,f)
w=length(X);
%Obtengo el vector de ordenadas, el de los valores.
for i=1:w
  Y(i)=feval(f,X(i));
end
n=w-1;
L=zeros(w,w);
%Formación de los polinomios coeficientes de Lagrange.
for k=1:n+1
  V=1;
  for j=1:n+1
    if k~=j
      V=conv(V,poly(X(j)))/(X(k)-X(j));
    end
  end
  L(k,:)=V;
end
%Cálculo de los coeficientes del polinomio interpolador de Lagrange.
C=Y*L;
```

Método de diferencias divididas:

```
function C=difdiv(X,f)
%DIFERENCIA DIVIDIDA DE NEWTON.
format long;
w=length(X);
%Obtengo el vector de ordenadas.
for i=1:w
  Y(i)=feval(f,X(i));
end
D=zeros(w,w);
D(:,1)=Y';
%Cálculo de las diferencias divididas
```

```
for j=2:w
   for k=j:w
      D(k,j)=(D(k,j-1)-D(k-1,j-1))/(X(k)-X(k-j+1));
   end
end
%Cálculo del vector que contiene los coeficientes del polinomio interpolado
C=D(w,w);
for k=(w-1):-1:1
   C=conv(C,poly(X(k)));
   m=length(C);
   C(m)=C(m)+D(k,k);
end
```

Método de diferencias progresivas:

```
function C=difpro(X,f)
w=length(X);
%Obtengo el vector de ordenadas.
for i=1:w
  Y(i)=feval(f,X(i));
end
V=Y(1);
% Calculo de las diferencias progresivas
for i=1:w
   coeficientes(i)=V(i);
   for k=(1+i):w
      V(k)=Y(k)-Y(k-1);
   end
   Y=V;
end
%Parte combinatoria del polinomio interpolador mediante NEWTON
polinom=zeros(1,w);
auxpolin=[1 -1]; % equivale a S=(x-x1)/h
for i=1:w
   if (i==1)
      polinom(w)=coeficientes(i);
   else
      for j=1:w-i
         auxpolin=conv(auxpolin,[1 (-j+1)]);
      end
      for k=1:length(auxpolin)
         if (length(polinom)<length(auxpolin) & k==length(auxpolin))
            polinom(k)=auxpolin(k);
         else
            polinom(k)=(auxpolin(k)+polinom(k));
         end
      end
   end
end
C=polinom;
```

Método de chebyshev:

```
function C=chebyshev(n,a,b,f)
C=zeros(1,n+1);
%Estimación de X.
for i=1:n+1
   X(i)=cos((2*i-1)*pi/(2*n+2));
end
X=(b-a)*X/2+(a+b)/2;
Y=feval(f,X);
for k=1:n+1
   z=(2*k-1)*pi/(2*n+2);
   for j=1:n+1
      C(j)=C(j)+Y(k)*cos((j-1)*z);
```

```
  end
end
C=2*C/(n+1);
C(1)=C(1)/2;
```

Método de diferenciación numérica basado en Taylor:

```
function sol=der1(pol,x,TOL)
format long;
flops(0);
A=0;
for i=1:2
   h=1/(2^(i-1));
   A(i,1)=(feval(pol,x+h)-feval(pol,x-h))/(2*h);
end
A(2,2)=(4*A(2,1)-A(1,1))/3;
f=2;
while abs(A(f-1,f-1)-A(f,f))>TOL
  f=f+1;
  h=1/(2^(f-1));
     A(f,1)=(feval(pol,x+h)-feval(pol,x-h))/(2*h);
  for i=2:f
      A(f,i)=(4^(i-1)*A(f,i-1)-A(f-1,i-1))/(4^(i-1)-1);
  end
end
sol=A(f,f);
```

Método del trapecio:

```
function sol=trapecio(a,b,TOL,f)
for i=1:2
  h=1/(2^(i-1));
     A(i,1)=(h/2)*(feval(f,a)+feval(f,b));
end
A(2,2)=(4*A(2,1)-A(1,1))/3;
j=2;
while abs(A(j-1,j-1)-A(j,j))>TOL
  j=j+1;
  h=1/(2^(j-1));
  A(j,1)=(h/2)*(feval(f,a)+feval(f,b));
     for i=2:j
      A(j,i)=(4^(i-1)*A(j,i-1)-A(j-1,i-1))/(4^(i-1)-1);
  end
end
sol=A(j,j);
```

Método del trapecio compuesto:

```
function sol=trapeciocomp(a,b,n,f)
h=(b-a)/n;
sol=0;
for k=1:(n-1)
   x=a+h*k;
   sol=sol+feval(f,x);
end
sol=h*(feval(f,a)+feval(f,b))/2+h*sol;
```

Método del trapecio compuesto utilizando Richardson para su optimización:

```
function S=trcpri(a,b,n,TOL,f)
h=(b-a)/n;
k=n;
A=0;
for i=1:2
  S=0;
  h=h/(2^(i-1));
  k=(2^(i-1))*k;
  for j=1:k
    if (feval(f,a+(j-1)*h)+ feval(f,a+j*h))<0
      S=S-(h/2)*(feval(f,a+(j-1)*h)+feval(f,a+j*h));
```

```
    else
      S=S+(h/2)*(feval(f,a+(j-1)*h)+feval(f,a+j*h));
    end
  end
  A(i,1)=S;
end
A(2,2)=(4*A(2,1)-A(1,1))/(3);
c=2;
while abs(A(c-1,c-1)-A(c,c))>=TOL
  c=c+1;
  h=h/2;
  k=2*k;
  S=0;
  for j=1:k
    if (feval(f,a+(j-1)*h)+feval(f,a+j*h))<0
      S=S-(h/2)*(feval(f,a+(j-1)*h)+feval(f,a+j*h));
    else
      S=S+(h/2)*(feval(f,a+(j-1)*h)+feval(f,a+j*h));
    end
  end
  A(c,1)=S;
  for i=2:c
    A(c,i)=(4^(i-1)*A(c,i-1) - A(c-1,i-1))/(4^(i-1)-1);
  end
end
A
S=A(c,c);
```

Método de Simpson:

```
function sol=simpson(a,b,f)
j=(b-a)/2;
sol=(j/3)*(feval(f,a)+4*feval(f,j)+feval(f,b));
```

Método de Simpson compuesto:

```
function sol=simpsoncomp(a,b,n,f)
h=(b-a)/(2*n);
sol1=0;
sol2=0;
for k=1:n
  x=a+h*(2*k-1);
  sol1=sol1+feval(f,x);
end
for k=1:(n-1)
  x=a+h*2*k;
  sol2=sol2+feval(f,x);
end
sol=(h/3)*(feval(f,a)+feval(f,b)+4*sol1+2*sol2);
```

Método de Romberg:

```
function sol=romberg(a,b,n,TOL,f)
m=1;
h=b-a;
err=1;
i=0;
A=zeros(4,4);
A(1,1)=(h/2)*(feval(f,a)+feval(f,b));
while((err>TOL)&(i<n))|(i<4)
  i=i+1;
  h=h/2;
  s=0;
  for p=1:m
    x=a+h*(2*p-1);
    s=s+feval(f,x);
  end
  A(i+1,1)=A(i,1)/2+h*s;
```

```
  m=2*m;
  for k=1:i
    A(i+1,k+1)=A(i+1,k)+(A(i+1,k)-A(i,k))/(4^k-1);
  end
  err=abs(A(i,i)-A(i+1,k+1));
end
sol=A(i+1,i+1);
```

PRÁCTICA Nº4:ECUACIONES DIFERENCIALES.

Método de Euler:

```
function R=euler(a,b,ya,n,f);
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(1,n+1);
X=a:h:b;
Y(1)=ya;
for i=1:n
  Y(i+1)=Y(i)+h*feval(f,X(i),Y(i));
end
R=[X' Y'];
E=abs(-2*exp(-b)+b^2-2*b+3-ya)
,flops
```

Método del punto medio:

```
function R=puntomedio(a,b,ya,n,f)
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(1,n+1);
X=a:h:b;
Y(1)=ya;
for i=1:n
   Y(i+1)=Y(i)+h*feval(f,X(i),Y(i)+h/2*feval(f,X(i),Y(i)));
end
R=[X' Y'];
E=abs(-2*exp(-b)+b^2-2*b+3-ya)
,flops
```

Método de Heun:

```
function R=heun(a,b,ya,n,f);
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(1,n+1);
X=a:h:b;
Y(1)=ya;
for i=1:n
Y(i+1)=Y(i)+(h/4)*(feval(f,X(i),Y(i))+3*feval(f,X(i)+(2/3)*h,Y(i)+(2/3)*h*feval(f,X(i),Y(i))));
end
R=[X' Y'];
E=abs(-2*exp(-b)+b^2-2*b+3-ya);
,flops
```

Método de Gragg:

```
function A=gragg(a,b,w0,n,f);
format long
flops(0);
for i=1:4
  h=(b-a)/((2^(i-1))*n);
  w1=w0+h*feval(f,a,w0);
  for j=1:(2^(i-1))*n
      w2=w0+2*h*feval(f,a,w1);
```

```
      a=a+h;
      w0=w1;
      w1=w2;
  end
  A(i,1)=w2;
end
for i=2:4
  for j=i:4
    A(j,i)=(4^(i-1)*A(j,i-1)-A(j-1,i-1))/(4^(i-1)-1);
  end
end
A(j,j)
,flops
```

Método de Taylor:

```
function R=taylor(a,b,ya,n,f)
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(1,n+1);
X=a:h:b;
Y(1)=ya;
for i=1:n
  D=feval(f,X(i),Y(i));
  Y(i+1)=Y(i)+h*(D(1)+h*(D(2)/2+h*(D(3)/6+h*D(4)/24)));
end
R=[X' Y'];
E=abs(-2*exp(-b)+b^2-2*b+3-ya)
,flops
```

Método de Runge:

```
function R=runge(a,b,ya,n,f);
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(1,n+1);
X=a:h:b;
Y(1)=ya;
for i=1:n
   k1=h*feval(f,X(i),Y(i));
   k2=h*feval(f,X(i)+h/2,Y(i)+k1/2);
   k3=h*feval(f,X(i)+h/2,Y(i)+k2/2);
   k4=h*feval(f,X(i)+h,Y(i)+k3);
   Y(i+1)=Y(i)+(1/6)*(k1+2*k2+2*k3+k4);
end
R=[X' Y'];
E=abs(-2*exp(-b)+b^2-2*b+3-ya)
,flops
```

Método de Runge-Kutta-Fehlberg:

```
function R=rkf(a,b,ya,n,TOL,f)
format long;
flops(0);
a2=1/4;b2=1/4;
a3=3/8;b3=3/32;c3=9/32;
a4=12/13;b4=1932/2197;c4=-7200/2197;d4=7296/2197;
a5=1;b5=439/216;c5=-8;d5=3680/513;e5=-845/4104;
a6=1/2;b6=-8/27;c6=2;d6=-3544/2565;e6=1859/4104;f6=-11/40;
r1=1/360;r3=-128/4275;r4=-2197/75240;r5=1/50;r6=2/55;
n1=25/216;n3=1408/2565;n4=2197/4104;n5=-1/5;
big=1e15;
h=(b-a)/n;
hmin=h/64;
```

```
hmax=64*h;
max1=200;
Y(1)=ya;
X(1)=a;
j=1;
br=b-0.00001*abs(b);
while (X(j)<b)
  if ((X(j)+h)>br)
    h=b-X(j);
  end
  k1=h*feval(f,X(j),Y(j));
  y2=Y(j)+b2*k1;
  if big<abs(y2)
    break
  end
  k2=h*feval(f,X(j)+a2*h,y2);
  y3=Y(j)+b3*k1+c3*k2;
  if big<abs(y3)
    break
  end
  k3=h*feval(f,X(j)+a3*h,y3);
  y4=Y(j)+b4*k1+c4*k2+d4*k3;
  if big<abs(y4)
    break
  end
  k4=h*feval(f,X(j)+a4*h,y4);
  y5=Y(j)+b5*k1+c5*k2+d5*k3+e5*k4;
  if big<abs(y5)
    break
  end
  k5=h*feval(f,X(j)+a5*h,y5);
  y6=Y(j)+b6*k1+c6*k2+d6*k3+e6*k4+f6*k5;
  if big<abs(y6)
    break
  end
  k6=h*feval(f,X(j)+a6*h,y6);
  err=abs(r1*k1+r3*k3+r4*k4+r5*k5+r6*k6);
  ynew=Y(j)+n1*k1+n3*k3+n4*k4+n5*k5;
  if ((err<TOL)|(h<2*hmin))
    Y(j+1)=ynew;
    if ((X(j)+h)>br)
      X(j+1)=b;
    else
      X(j+1)=X(j)+h;
    end
    j=j+1;
  end
  if (err==0)
    s=0;
  else
    s=0.84*(TOL*h/err)^(0.25);
  end
  if ((s<0.75)&(h>2*hmin))
    h=h/2;
  end
  if ((s>1.50)&(2*h<hmax))
    h=2*h;
  end
  if ((big<abs(Y(j)))|(max1==j))
    break
  end
  n=j;
```

```
  if (b>X(j))
    n=j+1;
  else
    n=j;
  end
end
R=[X' Y'];
,flops
```

Método de Adams-Bashforth-Moulton:

```
function R=adbamo(f,X,Y);
format long;
flops(0);
n=length(X);
if n<5
  break
end;
F=zeros(1,4);
F=feval(f,T(1:4),Y(1:4));
h=T(2)-T(1);
for k=4:n-1
  p=Y(k)+(h/24)*(F[-9 37 -59 55]');
  T(k+1)=T(1)+h*k;
  F=[F(2) F(3) F(4) feval(f,T(k+1),p)];
  Y(k+1)=Y(k)+(h/24)*(F*[1 -5 19 9]');
  F(4)=feval(f,T(k+1),Y(k+1));
end
R=[X' Y'];
,flops
```

Método de Runge para sistemas:

```
function R=rungesis(a,b,YA,n,fsis);
format long;
flops(0);
h=(b-a)/n;
X=zeros(1,n+1);
Y=zeros(n+1,length(YA));
X=a:h:b;
Y(1,:)=YA;
for i=1:n
   k1=h*feval(fsis,X(i),Y(i,:));
   k2=h*feval(fsis,X(i)+h/2,Y(i,:)+k1/2);
   k3=h*feval(fsis,X(i)+h/2,Y(i,:)+k2/2);
   k4=h*feval(fsis,X(i)+h,Y(i,:)+k3);
   Y(i+1,:)=Y(i,:)+(1/6)*(k1+2*k2+2*k3+k4);
end
R=[X'Y];
```

Apéndice B Comandos de Matlab.

Introducción.

Desde el principio de la práctica los algoritmos que hemos implementado y los ejercicios que hemos resueltos han sidos realizado en Matlab, un programa con un gran potencial matemático entre otras cualidades, por este motivo creemos que es importante repasar de manera breve los comandos más importantes de este programa.

Operaciones aritméticas.

Sumar	+
Restar	-
Multiplicar	*
Dividir	/
Elevar a una potencia	^

Operaciones con matrices.

- Definición de matrices desde el teclado: >> A = [1 2 3 ; 4 5 6 ; 7 8 9]

"A" es un nombre de variable que sirve para almacenar una matriz , ';' separa las filas de la matriz. Los elementos de cada fila se separan con espacios en blanco.

-Matriz trapuesta:>> A'

-Para acceder a los elementos de un vector se añade al nombre de variable que almacena el vector un paréntesis con el valor de un índice. Para acceder a los elementos de una matriz hay que proporcionar en principio dos índices. Las matrices se introducen por filas pero se almacenan por columnas permitiéndose el acceso a un elemento utilizando un solo índice.

"\" : "X = A\b", calcula un vector X, tal que, X = inv(A)*b. Si A no tiene inversa calcula la solución mínimos cuadrados de A*X=b.Estos operadores son mixtos, en el sentido que pueden admitir en determinados casos, que un operando sea una matriz y otro operando sea un escalar. En estos casos, el escalar se aplica según el comando, a cada elemento de la matriz.

Tipos de datos.

-Números reales de doble precisión:
Constantes: "inf"=infinito
"NaN"=valor no numérico.
Funciones: "eps", "realmin", "realmax".

-Números complejos:Los complejos se representan "a + b·i" con "i" o "j" la unidad imaginaria.
Función "complex".

-Cadenas de caracteres.Van encerradas entre '.

Variables y expresiones matriciales.

Una variable es un nombre que denota el almacenamiento de una matriz, un vector o un escalar.Operador de asignación: "=".Expresión: conjunto de comandos y argumentos.En una línea pueden ir varias expresiones, separadas por "," o ";", una expresión, o bien, una expresión puede ocupar más de una línea, para lo cual se deben especificar "..." en la parte del comando que cambia de línea.";" al final de una expresión, hace que el resultado no se displaye.

En los nombres de variables se distinguen las mayúsculas de las minúsculas. El nombre de una variable debe comenzar por una letra y puede tener hasta 31 caracteres entre letras, números y algunos caracteres especiales (".", "-").

Las variables se almacenan en el "Workspace" y se pueden consultar con los comandos "who" y "whos".

Características generales de las funciones Matlab.

Función Matlab: nombre, valor de retorno y argumentos.Las funciones pueden ser llamadas desde expresiones o introducidas como comandos. Las funciones usuario se escriben en ficheros de texto del tipo .m.Las funciones pueden tener valores de retorno matriciales múltiples. Puede haber funciones sin argumentos.

Tipos de funciones según su finalidad:

1.- Funciones matemáticas elementales.
2.- Funciones especiales.
3.- Funciones matriciales elementales.
4.- Funciones matriciales específicas.
5.- Funciones para la descomposición y/o factorización de matrices.
6.- Funciones para el análisis estadístico de datos.
7.- Funciones para el análisis de polinomios.
8.- Funciones para la integración de ecuac. diferenciales ordinarias.
9.- Resolución de ecuac. no lineales y optimización.
10.- Integración numérica.
11.- Funciones para procesamiento de señal.

Características generales de las funciones Matlab:

- Los argumentos pueden ser expresiones o llamadas a otra función.
- Nunca se modifican las variables que se usan como argumentos.
- Se admiten valores d retorno matriciales múltiples; >> [v, d] = eig(A)
- "Help nombre de función": ofrece información sobre la función con ese nombre.

"Help Desk" contiene los enlaces "matlab function by subject" y "matlab function by index" donde aparece la relación completa de las funciones Matlab.

Bifurcaciones y Bucles.

- Sentencia IF.

```
if condición sentencias end
if condición-1 bloque-1
elseif cndición-2 bloque-2
  ...
else bloque  end
```

- Sentencia SWITCH.

```
switch  switch_expresión
case case_exp-1 bñoque-1
case {case_exp-2, case_exp-3,...} bloque-2
 ...
otherwise bloque
end
```

- Sentencia FOR.

```
for valor-inicial : incremento : valor-final
        sentencias
end
```

- Sentencia WHILE.

```
While condición
      sentencias
end
```

- Sentencia Break.

Termina la ejecución del bucle más interno de los que contiene a la instrucción.

Ficheros *.m

Un fichero *.m, es un fichero de texto (creado por un editor de texto) que contiene un conjunto de comandos Matlab. Hay dos tipos de ficheros *.m, los ficheros de comandos y las funciones.

Ficheros de comandos (Scripts).

Se ejecutan, escribiendo su nombre en la ventana de comandos. Pueden hacer llamadas a otros ficheros*.m o incluso a sí mismos de forma recursiva. Las variables de trabajo creadas por estos ficheros están en el Workspace.

El comando 'echo' hace que se escriban los nombres de cada comando que es ejecutado.Un ejemplo de fichero *.m creado por el propio Matlab es el fichero 'starup.m' eal que se le pueden añadir comandos que modifiquen el path.

Definición de Funciones.

Las funciones se escriben con editores (archivos de texto). La primera línea (excluyendo líneas de comentario) es de la forma:

Function [valores de retorno,...] = nombre-de-función (argumentos,...)

Una función no modifica nunca los argumentos que recibe. Las variables definidas dentro de una función son variables locales y pertenecen al espacio de trabajo de la función. Para que una función tenga acceso a variables que no han sido declaradas como argumentos, éstas han de ser variables globales.

Existen dos variables que se definen de modo automático cada vez que se defne una función, las variables 'nargin', 'nargout' que representan el número de argumentos y el número de valores de retorno. Estas variables se pueden utilzar dentro de la función.

La sentencia 'return', abandona la ejecución de la función, antes de llegar al últmo comando que la define.

Subfunciones.

Las subfunciones son funciones adicionales definidas en el mismo fichero *.m que la función principal con la que están relacionadas. Tienen sus propios nombres y solo pueden ser llamadas desde funciones del fichero *.m donde están escritas.

Funciones gráficos 2D elementales.

Estas funciones se diferencian en el tipo de escala que utilizan en los ejes de abscisas y ordenadas.

"plot()" crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales en ambos ejes. En los casos más sencillos, la función plot toma como argumentos vectores. Si el argumento es una matriz, se considerará como un conjunto de vectores columna (en algunos casos también de vectores fila).

"loglog()": idem con escala logarítmica en ambos ejes.

"semilogx()": idem con escala lineal en el eje de ordenadas y logarítmica en el de abscisa.

"semilogy()": idem escala lineal en abscisa y logarítmica en ordenadas.

Existen también otras funciones auxiliares:

"title('título')": añade título a un gráfico.

"xlabel('tal')": añade una etiqueta al eje de abscisas. Se desactiva con "xlabel off". Análogamente "ylabel('cual'), ylabel off".

"text(x,y,'texto')": añade 'texto' a las coordenadas especificadas por x, y. Si x e y son vectores, el texto se añade a todo par de elementos.

"gtext('texto')".

"legen()".

"grid".

Función Plot.

El elemento básico de los gráficos es el vector. Se puede decir que la función plot dibuja vectores.

Ejemplo:

```
>> x = [1 3 2 4 5 3]
X =
  1 3 2 4 5 3
>> plot(x)
```

En este caso, se dibuja las componentes de x en el eje de ordenadas para valores de abscisa 1, 2, 3, 4, 5, 6.

Ejemplo:

```
>> x =[1 6 5 2 1]; y = [1 0 4 3 1],
>> plot(x,y)
```

Dibuja las entre las componentes de y para valores de abscisa de x. De forma más general:
Ejemplo:

```
>> x = 0:pi/25:6+pi;
>> y = sin(x); z = cos(x);
>> plot(x,y,x,z)
```

Si los argumentos son varios vectores complejos, se ignoran las partes imaginarias. Si el argumento es un solo vector complejo se representa la parte imaginaria (ordenadas) sobre la parte real (abscisas).
Ejemplo:

```
>> plot (eig (rand(20,20)),'+')
```

'+' hace que no se escriba con línea continua.

El comando "plot" puede tambhién tomar matrices como argumentos:

```
Plot(A), plot(x,A), plot(A,x), plot(A,B)
```

Estilos de línea y marcadores en la función plot. Estos aspectos se especifican en la función plot como una cadena de 1, 2, 3 o 4 caracteres al final de los vectores y/o matrices que toma como argumento.

Símbolo	Color	Símbolo	Marcadores
Y	Amarillo	.	puntos
M	Magenta	o	Círculos
C	Cyan	X	Marcas en x
R	Rojo	+	Marcas en +
G	Verde	*	Marcas en *
B	Azul	S	Marcas cuadradas
W	Blanco	D	Marcas diamante
K	Negro	^	Triángulo Δ
		V	Triángulo ∇
Símbolo	Estilo de línea	>	Triángulo >
-	Línea continua	<	Triángulo <
:	Línea a puntos	P	Estrella 5 puntas
-.	Línea barra-punto	H	Estrella 6 puntas
--	Línea a trazos		

Apéndice C Distribución de los ficheros.

Práctica 1:Resolución de ecuaciones.

Auxiliares:
1-f.m: Donde está implementada la ecuación.
2-fp.m: Donde está implementada la derivada de la ecuación.
3-g.m: Donde está implementada la ecuación x=g(x).
4-gp.m: Donde está implementada la derivada de la ecuación x=g(x).
5-dibujafx.m: Dibuja gráficamente la función implementada en f.m.
6-dibujagp.m: Dibuja gráficamente la función implementada en gp.m.
Algoritmos:
1-bisec.m:Bisección.
2-aprox.m:Aproximación a las raíces.
3-regula.m:Regula falsi.
4-pfijo.m: Punto fijo.
5-newton.m:Newton-Rapshon.
6-secante.m: Secante.
7-aitken.m:Aitken.
8-steffensen.m:Steffensen.

Práctica 2:Resolución de sist. de ecuaciones.

SisLin:
Algoritmos:
1-trisusureg.m:Triangularización sup. y sust.
2-lufact.m:Factorización LU.
3-jacobi.m:Jacobiano.
4-gseid.m:Gauss-Seidel.

SisNoLin:
Auxiliares:
1-fsis.m: Donde está implementado el sist. de ecuaciones.
2-gsis.m: Donde está implementado el sist. de ecuaciones x=g(x).
3-gseidel.m: Donde está implementado el sist. de ecuaciones para Seidel.
4-jacfsis.m:Jacobiano de f sistema.
5-hsis.m: Se guarda la altura.
6-gradh.m: Se guarda el gradiente.
Algoritmos:
1-pfijosis.m:Punto fijo para sistemas.
2-newsis.m:Newton para sistemas.
3-seidel.m:Seidel.
4-broyden.m:Broyden.
5-maxdes.m:Máximo descenso.
6-nglobal.m:Newton global.

Práctica 3:Interpolación polinomial.Derivación e integración numérica.

Interpolación polinomial:
Auxiliares:
1-f.m: Donde está implementado la función a interpolar.
2-nodche.m: Devuelve los nodos de chebyshev para un determinado intervalo.
3-fact.m: Devuelve el factorial de un número.
4-comparar.m:Compara entre nodos igualmente espaciado y de Chebyshev.
Algoritmos:
1-Interpolación de Lagrange.
2-Diferencias divididas.
3-Diferencias progresivas.
4-Polinomios de Chebyshev.

Derivación e integración numérica:
Auxiliares:
1-f.m:Donde está la función a integral o derivar.
2-der2.m:Derivada segunda.
3-der3.m:Derivada tercera.

Algoritmos:
1-Der1.m:Diferencial por Taylor.
2-Difftext.m:Diferencial con extrapolación.
3-Diflim.m:Diferencial por límites.
4-Trapecio.m:Integración.Regla del trapecio.
5-Trapeciocomp.m:Integración.Regla del trapecio compuesta.
6-Trcpri.m:Integración.Regla del trapecio compuesta optimizado por Richardson.
7-Simpson.m:Integración.Regla de Simpson.
8-Simpsoncomp.m:Integración.Regla de Simpson compuesta.
9-Romberg.m:Integración.Método de Romberg.

Práctica 4:Ecuaciones diferenciales.

Auxiliares:
1-f.m:Donde está la ecuación diferencial.
2-fsis.m:Sistema de ecuaciones diferenciales.
3-df.m:Ecuaciones diferenciales de grado superior.

Algoritmos:
1-euler.m:Método de Euler.
2-puntomedio.m:Método del punto medio.
3-heun.m:Método de Heun.
4-gragg.m:Método de Gragg.
5-taylor.m:Método de la serie de Taylor.
6-runge.m:Método de Runge-Kutta clásico.
7-rkf.m:Método de Runge-Kutta-Fehlberg.
8-adbamo.m:Método de Adams-Bashforth-Moulton.
9-rungesis.m:Método de Runge-Kutta clásico para sistemas de ec. dif.

Apéndice D Bibliografía.

Práctica 1:Resolución de ecuaciones.
1-Análisis numérico.
David Kincaid, Ward Cheney.
Ed. Adilson-Wesley. Capítulo 3.

2-Métodos numéricos con MATLAB.
John H.Mathews, Kurtis D. Fink.
Ed. Prentice Hall. Capítulo 2.

3-Análisis numérico y visualización gráfica con Matlab.
Shoichiro Nakamura.
Ed. Hispanoamericana. Capítulo 7.

Práctica 2:Resolución de sistemas de ecuaciones.
1-Métodos numéricos con MATLAB.
John H.Mathews, Kurtis D. Fink.
Ed. Prentice Hall. Capítulo 3.

2-Análisis numérico.
Richard L.Burden, J. Douglas Faires.
Ed. Iberoamericana. Capítulo 2.

Práctica 3:Interpolación polinomial. Derivación e integración numérica.
1-Métodos numéricos con MATLAB.
John H.Mathews, Kurtis D. Fink.
Ed. Prentice Hall. Capítulo 4,6 y 7.

2-Análisis numérico.
David Kincaid, Ward Cheney.
Ed. Adilson-Wesley. Capítulo 6 y 7.

Práctica 4:Ecuaciones diferenciales.
1-Métodos numéricos con MATLAB.
John H.Mathews, Kurtis D. Fink.
Ed. Prentice Hall. Capítulo 9.

2-Análisis numérico.
David Kincaid, Ward Cheney.
Ed. Adilson-Wesley. Capítulo 8.

Anexo B:Comandos de Matlab.
1-Métodos numéricos con MATLAB.
John H.Mathews, Kurtis D. Fink.
Ed. Prentice Hall. Apéndice Matlab.

2-Apuntes de la Universidad de Navarra.