PRÁCTICAS DE ANÁLISIS NUMÉRICO I

Víctor Manuel Galdámez Salguero
José Manuel Mesa Gómez

ÍNDICE GLOBAL

PRÁCTICA 1

RESOLUCIÓN DE ECUACIONES NO LINEALES.....1

PRÁCTICA 2

RESOLUCIÓN DE SISTEMAS DE ECUACIONES NO LINEALES.....24

PRÁCTICA 3

INTERPOLACIÓN.DERIVACIÓN E INTEGRACIÓN APROXIMADA.....91

PRÁCTICA 4

PROBLEMAS DE VALORES INICIALES PARA ECUACIONES DIFERENCIALES.....148

***BIBLIOGRAFÍA*.....176**

RESOLUCIÓN DE ECUACIONES NO LINEALES

-Análisis Numérico I-

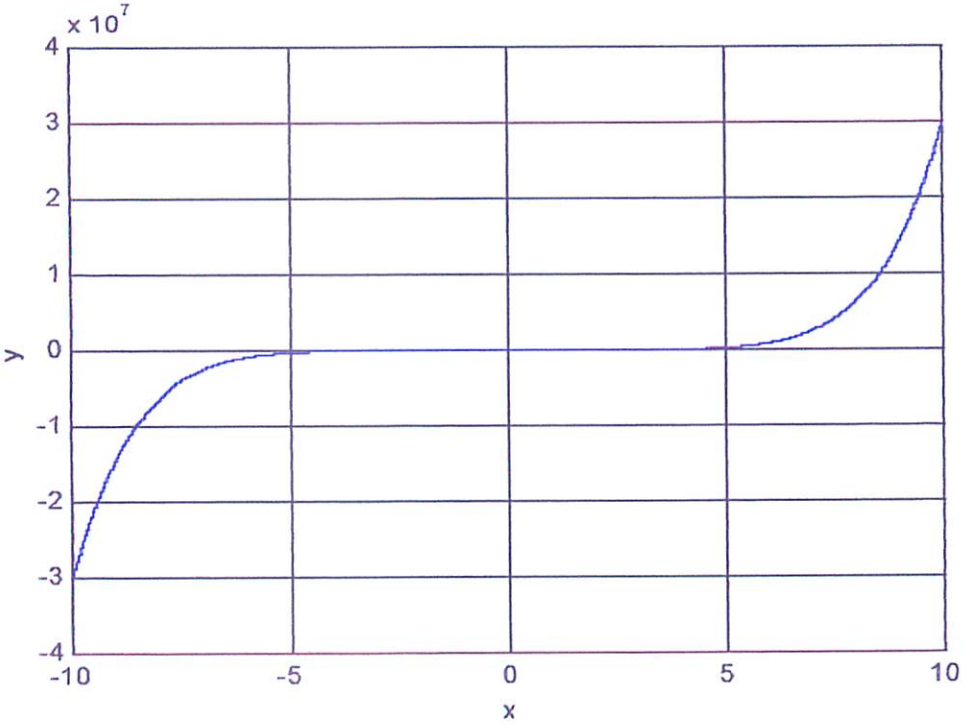
Víctor Manuel Galdámez Salguero
José Manuel Mesa Gómez

ÍNDICE

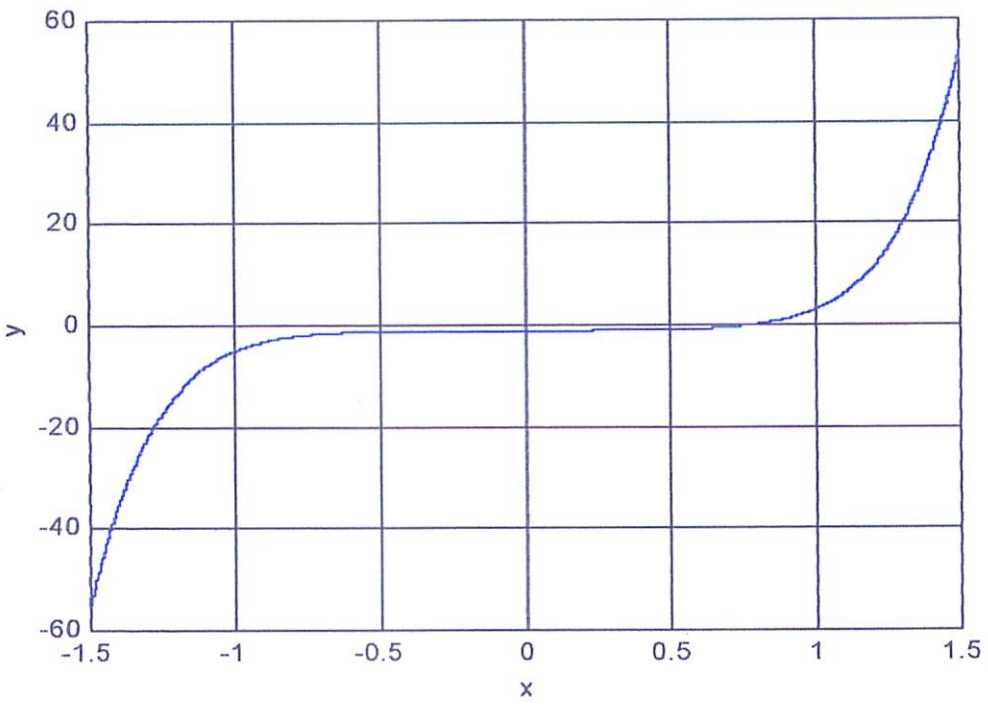
INTRODUCCIÓN.....1
BISECCIÓN.....2
ALGORITMO DEL PUNTO FIJO.....3
ALGORITMO DE AITKEN.....8
ALGORITMO DE NEWTON-RAPHSON.....8
ALGORITMO DE STEFFENSEN.....9
TABLA COMPARATIVA 1.....11
TABLA COMPARATIVA 2.....12
TABLA COMPARATIVA 3.....13
DIAGRAMAS DE FLUJO DE LOS ALGORITMOS.....14
BISECCION.....14
PUNTOFIJO.....14
AITKEN.....15
STEFFENSEN.....15
NEWTON.....15
DEFINICIÓN DE FUNCIONES UTILIZADAS.....16
IMPLEMENTACIÓN DE LOS MÉTODOS UTILIZADOS.....17

Se va a realizar un estudio de la función $f(x)=3x^7 + x^3-1$ para aproximar las soluciones de $f(x)=0$, para ello se van a usar los métodos de aproximación estudiados hasta ahora.

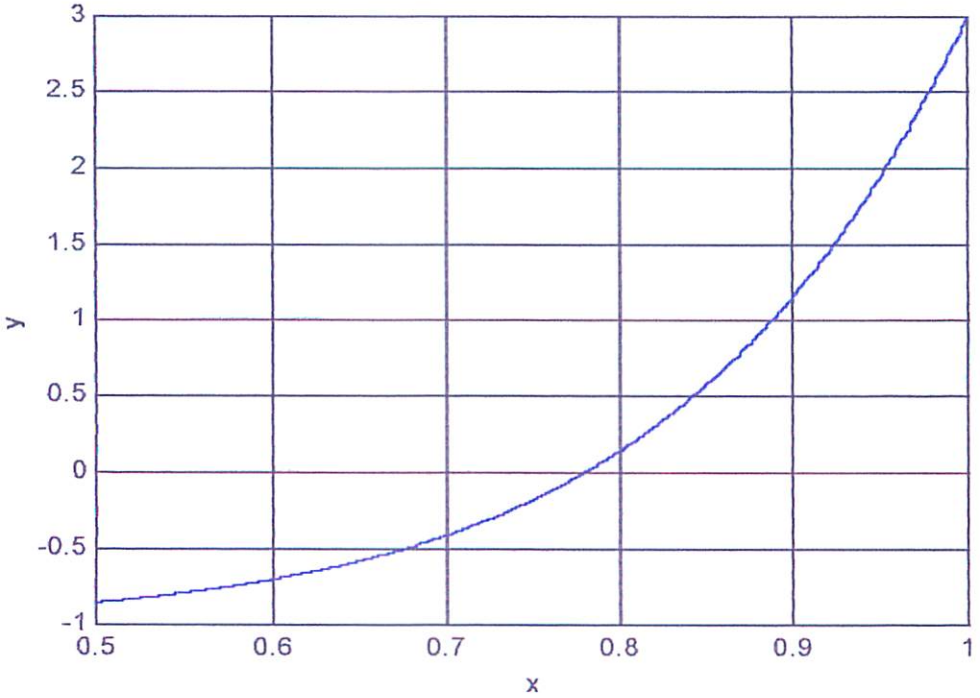
A continuación se muestra la representación gráfica de f(x) en [-10,10] para obtener una idea de la raíz de $f(x)=0$ en dicho intervalo, si es que la hay.



A simple vista no se puede obtener una idea de una solución aproximada, luego reducimos sucesivamente el intervalo hasta [-1.5, 1.5], donde ya se aprecia que una solución se encuentra en el intervalo [0, 1].



En una aproximación final se puede observar que la raíz está muy próxima a 0.8, luego representamos el intervalo [0.5, 1]:



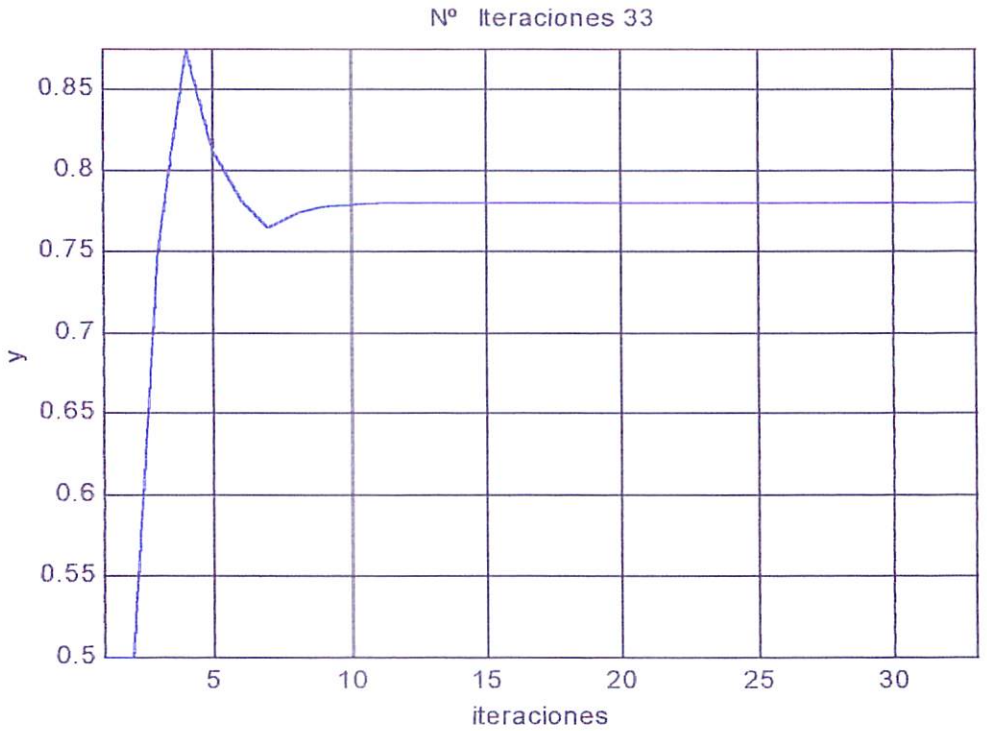
MÉTODO DE BISECCIÓN

El método de bisección lo que hace es, dado un intervalo, calcula su punto medio y estudia el signo de la función en los intervalos resultantes, quedándose con el intervalo donde la función cambia de signo, repitiendo los pasos según la tolerancia que se desee. Por ello, siempre que en el intervalo dado exista una única raíz, y solo una, el algoritmo siempre convergerá, pero de forma lenta.

La raíz resultante por este método en el intervalo [0, 1] y con una tolerancia de 10^{-10} es 0.779768068459816, utilizándose para calcularla 33 iteraciones.

N	SOLUCION	INTERVALO
1	0.500000000000000	[0.500000000000000,1.000000000000000]
5	0.781250000000000	[0.750000000000000,0.781250000000000]
9	0.779296875000000	[0.779296875000000,0.781250000000000]
13	0.779663085937500	[0.779663085937500,0.779785156250000]
17	0.779762268066406	[0.779762268066406,0.779769897460938]
21	0.779768466949463	[0.779767990112305,0.779768466949463]
25	0.779768079519272	[0.779768049716949,0.779768079519272]
29	0.779768070206046	[0.779768068343401,0.779768070206046]
33	0.779768068459816	[0.779768068459816,0.779768068576232]
Operaciones (coma flotante)	642	Tiempo(s) 0.77000000000000

El siguiente gráfico muestra la convergencia hacia la raíz respecto al numero de iteraciones:



Si se intenta usar el algoritmo en un intervalo donde no existe una única raíz, o no existen raíces, se muestra el mensaje *'En el intervalo no existe una única raíz'*.

ALGORITMO DEL PUNTO FIJO

Para aplicar el algoritmo del punto fijo hay que transformar la ecuación f(x)=0 al sistema x=g(x), siendo este paso no unívoco. Las funciones g(x), y sus derivadas, utilizadas, son las siguientes:

$$g_1(x) = x - \frac{3x^7 + x^3 - 1}{21x^6 + 3x^2} \quad , \quad g_{1'}(x) = \frac{(126x^5 + 6x)(3x^7 + x^3 - 1)}{(21x^6 + 3x^2)^2}$$

$$g_2(x) = \frac{1 - 3x^7}{x^2} \quad , \quad g_{2'}(x) = \frac{-15x^7 - 2}{x^3}$$

$$g_3(x) = 3x^7 + x^3 + x - 1 \quad , \quad g_{3'}(x) = 21x^6 + 3x^2 + 1$$

Utilizando biseccion en [0,1] con una tolerancia de 10^{-1} y una tolerancia en puntofijo de 10^{-15} resultan los siguientes datos:

O Para $g_1(x)$:

Iteración	Solución		
0 3 5	8.7500000000000000e-001 7.7977561101160131e-001 7.7976806856378933e-001		
Operaciones (coma flotante)	85	Tiempo(s)	4.999999999999716e-002

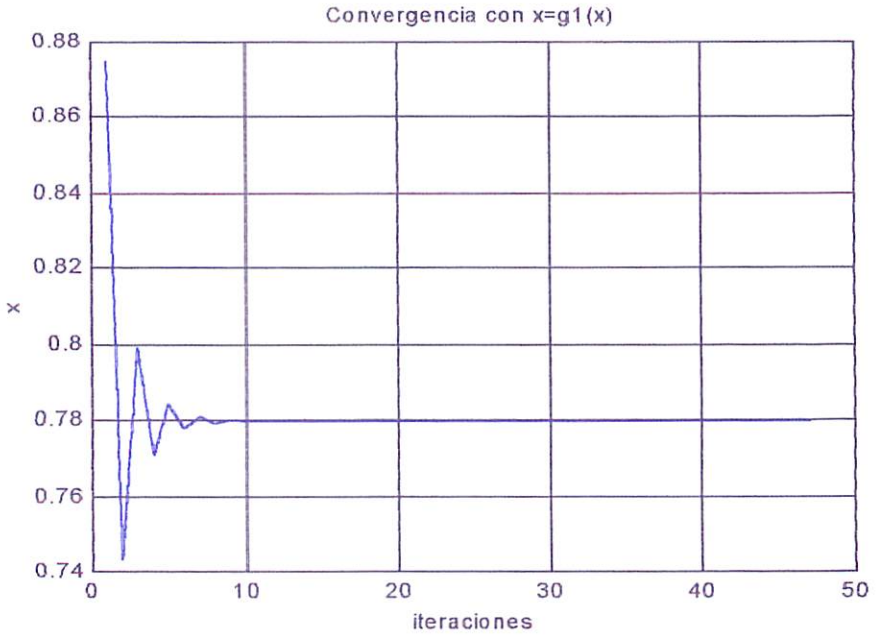
O Para $g_2(x)$:

Iteracion	Solución		
0 1 2 3 4 5	8.7500000000000000e-001 -2.3260435766103316e-001 1.8484695588939989e+001 -6.4741501725878520e+006 3.4122048781147884e+034 -1.3877037192196233e+173		
Operaciones (coma flotante)	43	Tiempo(s)	0.05000000000000

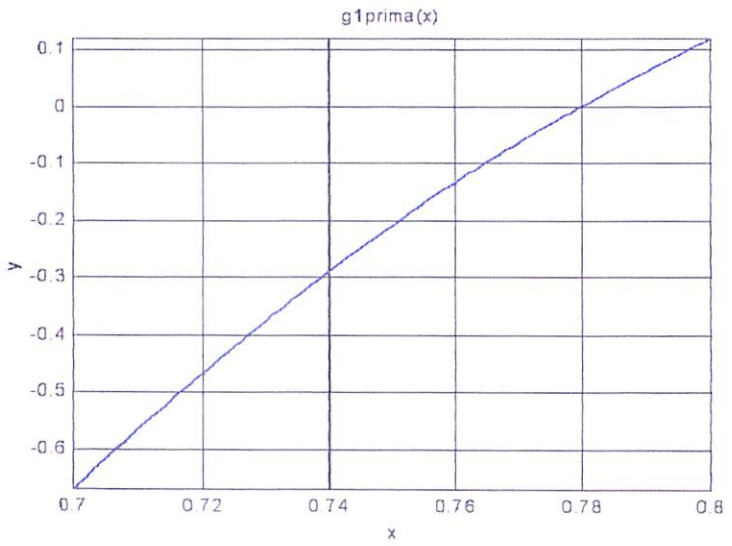
O Para $g_3(x)$:

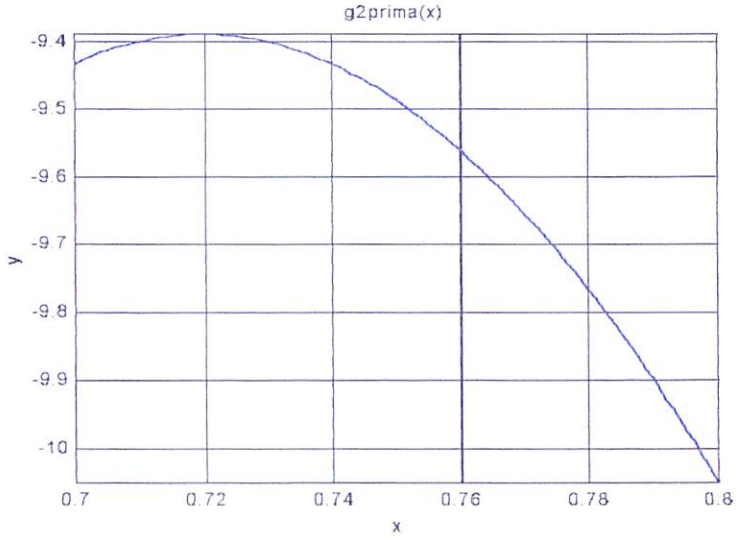
Iteración	Solución		
0 3 5	8.7500000000000000e-001 3.3384272376394255e+015 Inf		
Operaciones (coma flotante)	49	Tiempo(s)	0.06000000000000

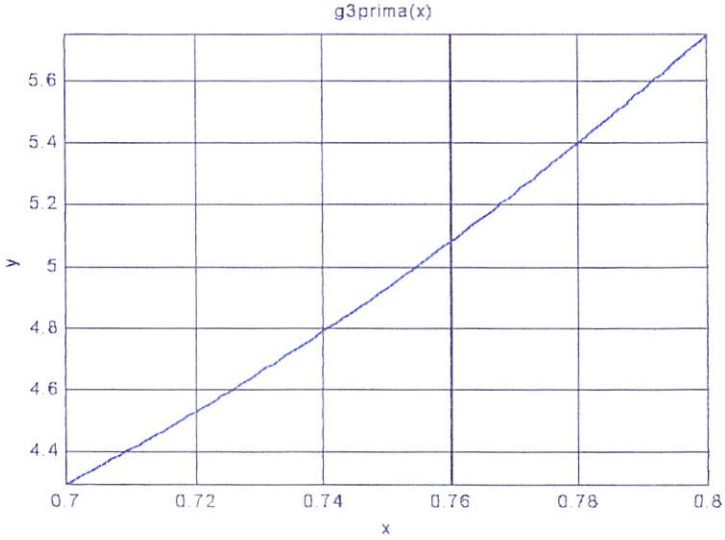
Observando los resultados se puede ver que la única g(x) que converge es $g_1(x)$ (g(x) de Newton-Raphson), pues $g_2(x)$ diverge a ∞ y $g_3(x)$ diverge a $+\infty$.



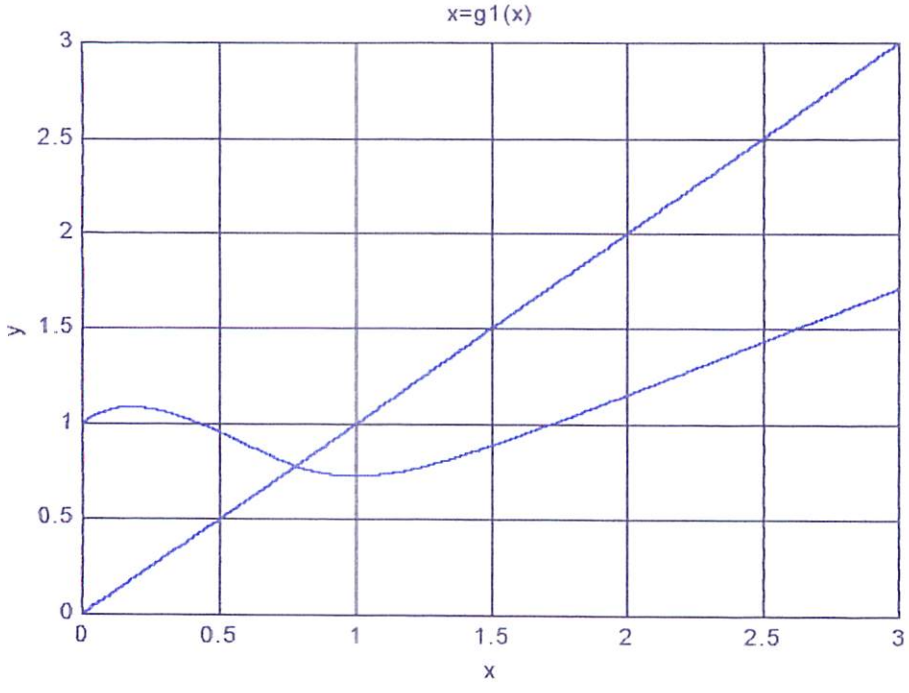
A continuación se muestran las representaciones gráficas de $g_1'(x)$, $g_2'(x)$ y $g_3'(x)$ en [0.7,0.8], para demostrar gráficamente que la única derivada que cumple **condición de convergencia $|g'(x)|<1$** es $g_1'(x)$, pues $g_1(x)$ converge para cualquier x dado, como podremos ver en la siguiente página, donde se representan los sistemas x=g(x).

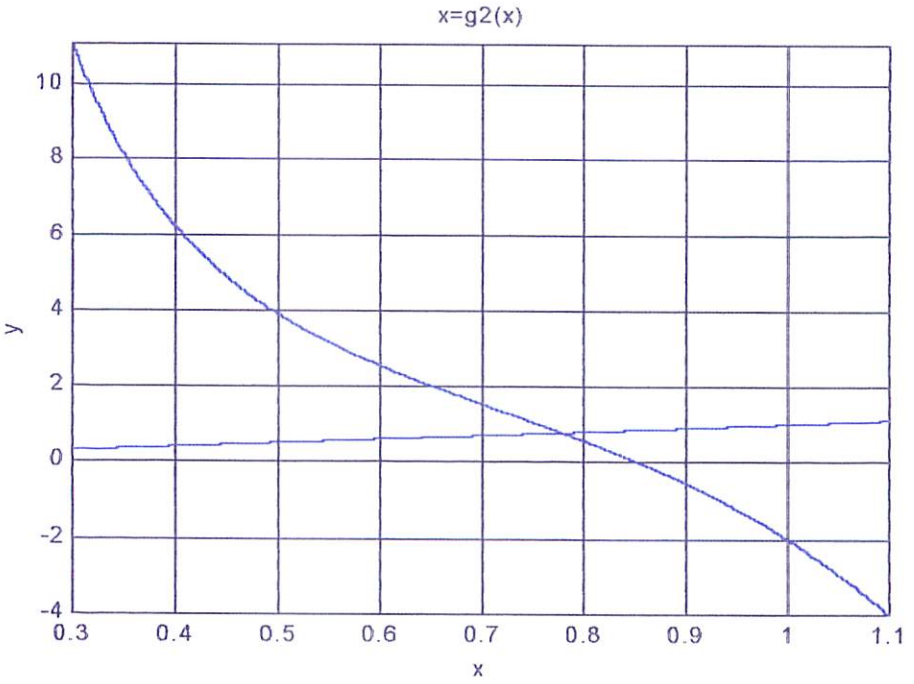


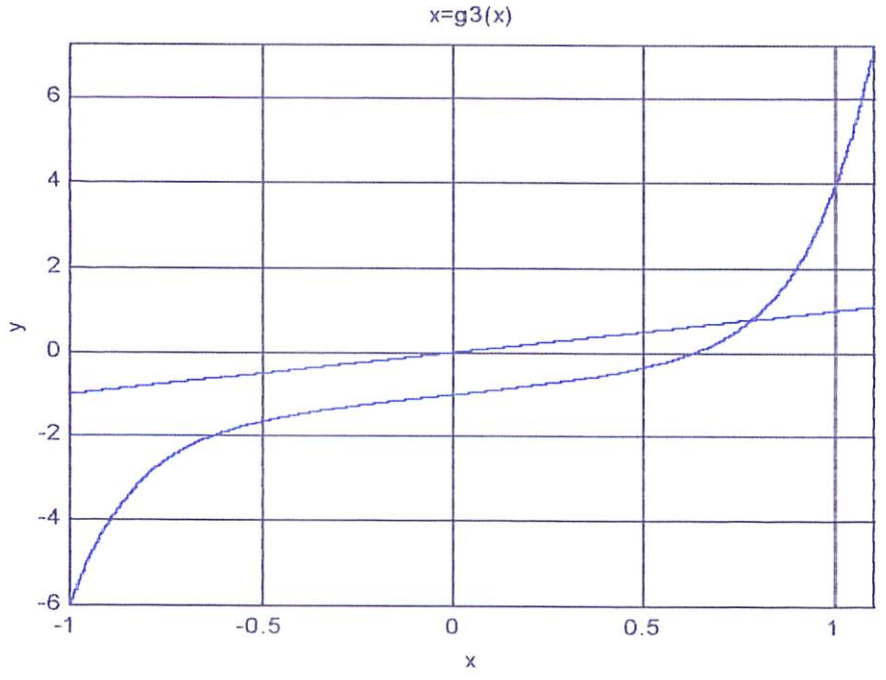




Para entender el por qué de las convergencias según los sistemas x=g(x) lo conveniente es representarlos gráficamente en entornos de la solución:







Se deduce de los gráficos que $g_1(x)$ converge para cualquier x, $g_2(x)$ diverge a $+\infty$ y $-\infty$ alternamente independientemente del x, y $g_3(x)$ diverge a $+\infty$ si el punto inicial es mayor que la solución o a $-\infty$ si es menor que la solución.

ALGORITMO DE AITKEN

Es un algoritmo cuya sucesión resultante converge superlinealmente, es decir, converge mucho más rápidamente que la sucesión lineal resultante del algoritmo del puntofijo.

It	$g_1(x)$	$g_2(x)$	$g_3(x)$
0	8.7500000000000000e-001	8.7500000000000000e-001	8.7500000000000000e-001
1	7.7965463744501284e-001	8.1311887339874978e-001	8.6980841529529296e-001
2	7.7976803179138243e-001	-2.3255024472036259e-001	1.7230095863284107e+000
3	7.7976806856378511e-001	1.8484695588939989e+001	1.4108754147129224e+002
4	7.7976806856378933e-001	-6.4741501725878520e+006	3.3384272376394255e+015
Operaciones (coma flotante) para $g_1(x)$	133	Tiempo(s) para $g_1(x)$	5.0000000000000426e-002

El comportamiento de las g(x) es igual que con el algoritmo del punto fijo pero algoritmo de Aitken converge mucho más rápidamente que el algoritmo del puntofijo.

ALGORITMO DE NEWTON-RAPHSON

El algoritmo de Newton-Raphson es un algoritmo cuadrático, de gran eficiencia y eficacia. Utilizando como dato inicial el que retorna el método de biseccion entre [0,1] con una tolerancia de 10^{-1} y newton con una tolerancia de 10^{-15} se obtienen los siguientes datos:

Iteración	Solución		
0	0.875000000000000		
1	0.802653965954176		
2	0.781322663463662		
3	0.779775611011601		
4	0.779768068741986		
5	0.779768068563789		
Operaciones (coma flotante)	83	Tiempo(s)	0.06000000000000

Se puede observar que aunque el p_0 inicial es muy malo el algoritmo obtiene la solución con 15 cifras decimales exactas en solo 5 iteraciones.

ALGORITMO DE STEFFENSEN

El algoritmo de Steffensen es también un algoritmo cuadrático pero de mayor eficiencia que newton, aunque el número de cálculos realizados es mayor.

Utilizando biseccion en [0,1] con una tolerancia de 10^{-1} y una tolerancia en steffensen de 10^{-15} resultan los siguientes datos:

O Para $g_1(x)$:

Iteración	Solución
0	8.7500000000000000e-001
1	7.7240319153723602e-001
2	7.7977203609404000e-001
3	7.7976806856378866e-001
4	7.7976806856378933e-001
Operaciones (coma flotante) 158	Tiempo(s) 7.797680685637893e-001

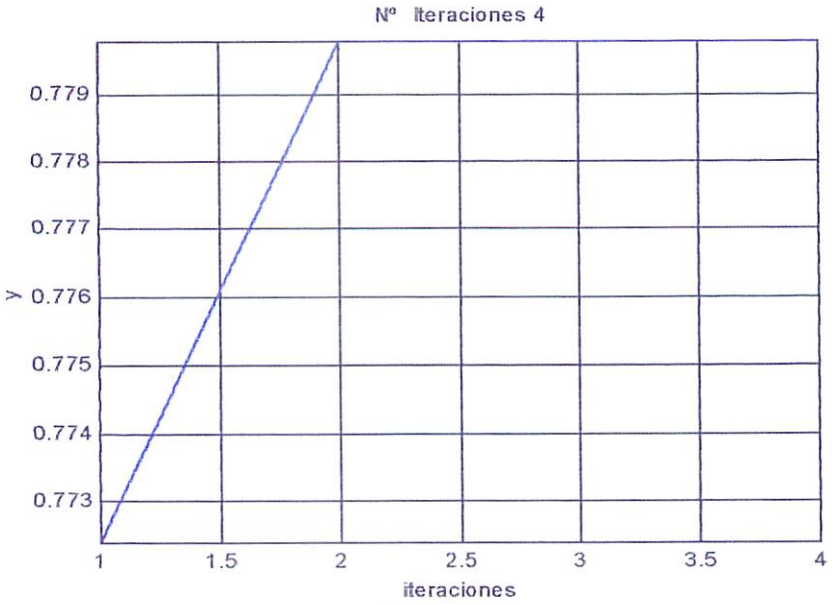
O Para $g_2(x)$:

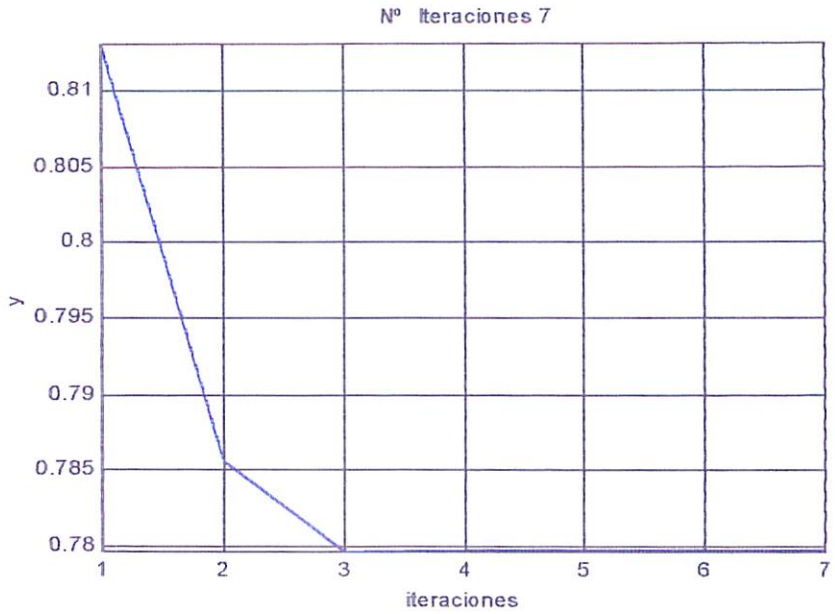
Iteración	Solución
0	8.7500000000000000e-001
1	8.1311887339874978e-001
2	7.8558917913219106e-001
3	7.7962871580987902e-001
4	7.7976796034049090e-001
5	7.7976806856372438e-001
6	7.7976806856378933e-001
Operaciones (coma flotante) 145	Tiempo(s) 0.0600000000000

O Para $g_3(x)$

Iteración	Solución
0	8.7500000000000000e-001
3	8.5727419606282407e-001
6	8.3009861216417524e-001
9	7.9144352883282387e-001
12	7.7976900693529305e-001
15	7.7976806856378933e-001
Operaciones (coma flotante) 329	Tiempo(s) 0.0500000000000

A continuación se muestran las tres gráficas de convergencia para los tres casos, $g_1(x)$, $g_2(x)$ y $g_3(x)$, respectivamente:


Nº Iteraciones 4
0.779
0.778
0.777
0.776
0.775
0.774
0.773
y
1
1.5
2
2.5
3
3.5
4
iteraciones


Nº Iteraciones 7
0.81
0.805
0.8
0.795
0.79
0.785
0.78
y
1
2
3
4
5
6
7
iteraciones

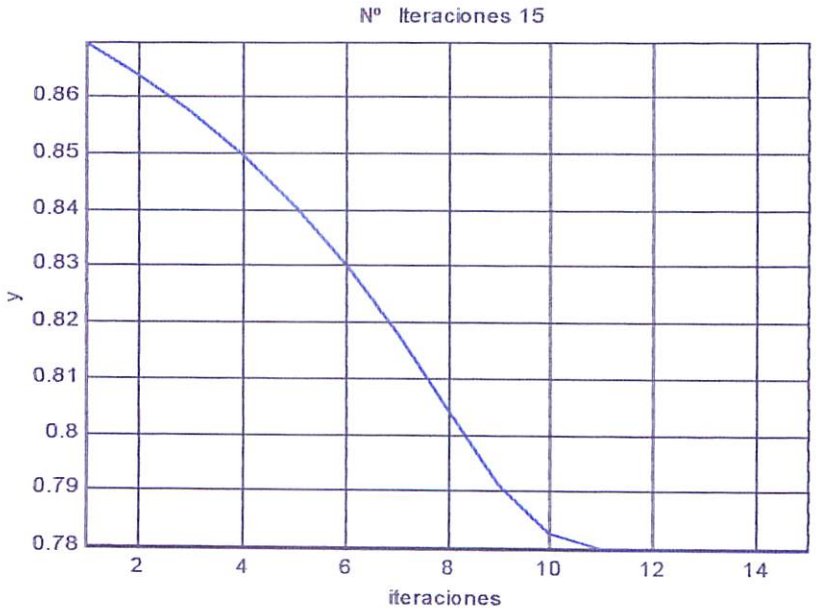

Nº Iteraciones 15
0.86
0.85
0.84
0.83
0.82
0.81
0.8
0.79
0.78
y
2
4
6
8
10
12
14
iteraciones

TABLA COMPARATIVA USANDO $g_1(x)$

Resultados obtenidos a partir del punto inicial resultante de aplicar Bisección en el intervalo [0, 1] con una tolerancia de 10^{-1}, y una tolerancia en los algoritmos de 10^{-15}

Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	8.7500000000000000e-001	8.7500000000000000e-001	8.7500000000000000e-001	8.7500000000000000e-001
3	7.7977561101160131e-001	7.7976803179138243e-001	7.7977561101160131e-001	7.7976806856378866e-001
5	7.7976806856378933e-001	7.7976806856378933e-001	7.7976806856378933e-001	7.7976806856378933e-001
OPERACIONES	85	133	83	158
TIEMPO (s)	4.999999999999716e-002	5.000000000000426e-002	0.06000000000000	7.797680685637893e-001

Resultados obtenidos a partir del punto inicial $x_0 = 2$, y una tolerancia en los algoritmos de 10^{-15}

Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	2.0000000000000000e+000	2.0000000000000000e+000	2.0000000000000000e+000	2.0000000000000000e+000
6	8.2843174757996751e-001	7.5473821270323294e-001	8.2843174757996751e-001	5.0107951786712348e+007
11	7.7976806856378933e-001	7.7976806856378933e-001	7.7976806856378933e-001	-4.1521493144484944e-002
OPERACIONES	169	259	167	DIVERGE
TIEMPO (s)	4.999999999999982e-002	6.000000000000227e-002	0.06000000000000	DIVERGE

Resultados obtenidos a partir del punto inicial $x_0 = 2$, con la diferencia de que los algoritmos de Newton-Raphson y Steffensen se ejecutan con una tolerancia de 10^{-15} y usan puntofijo con una tolerancia de 10^{-2} como punto inicial

Iteración	PUNTO FIJO (10^{-15})	PUNTO FIJO (10^{-2} con $g_1(x)$)	NEWTON-RAPHSON	STEFFENSEN
0	2.0000000000000000e+000	2.0000000000000000e+000	7.8639266884913606e-001	7.7976524628179755e-001
2	1.4636554393444523e+000	1.4636554393444523e+000	7.7976812590885869e-001	7.7976806856378933e-001
4	1.0707609344777511e+000	1.0707609344777511e+000	7.7976806856378933e-001	
6	8.2843174757996751e-001	8.2843174757996751e-001		
8	7.7990339296243294e-001	7.8639266884913606e-001		
10	7.7976806856379965e-001			
12	7.7976806856378933e-001			
OPERACIONES	169	113	69	125
TIEMPO (s)	4.999999999999982e-002	6.000000000000227e-002	0.06000000000000	6.000000000000050e-002

TABLA COMPARATIVA USANDO $g_2(x)$

Resultados obtenidos a partir del punto inicial resultante de aplicar Bisección en el intervalo [0, 1] con una tolerancia de 10^{-1}, y una tolerancia en los algoritmos de 10^{-15}

Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	8.750000000000000e-001	8.750000000000000e+001	8.750000000000000e-001	8.750000000000000e-001
3	-6.474150172587852 0e+006	1.848469558939989e+001	7.797756110116013 1e-001	7.796287158098790 2e-001
5	-1.387703719196233e+173	NaN	7.797680685637893 3e-001	7.797680685637243 8e-001
7	-Inf	NaN		7.797680685637893 3e-001
OPERACIONES	DIVERGE	DIVERGE	83	145
TIEMPO (s)	DIVERGE	DIVERGE	0.050000000000000	0.050000000000000

Resultados obtenidos a partir del punto inicial $x_0 = 2$, y una tolerancia en los algoritmos de 10^{-15}

Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	2.000000000000000e+000	2.000000000000000e+000	2.000000000000000e+000	2.000000000000000e+000
6	Inf	Inf	8.284317475799675 1e-001	1.999997625496135 1e+000
11	Inf	Inf	7.797680685637893 3e-001	1.999994855187397 4e+000
OPERACIONES	DIVERGE	DIVERGE	167	CONV. LÉNTAMENTE
TIEMPO (s)	DIVERGE	DIVERGE	0.060000000000000	CONV. LÉNTAMENTE

Resultados obtenidos a partir del punto inicial $x_0 = 2$, con la diferencia de que los algoritmos de Newton-Raphson y Steffensen se ejecutan con una tolerancia de 10^{-15} y usan puntofijo con $g_1(x)$ y una tolerancia de 10^{-2} como punto inicial

Iteración	PUNTO FIJO (10^{-15})	PUNTO FIJO (10^{-2} con $g_1(x)$)	NEWTON-RAPHSON	STEFFENSEN
0	2.000000000000000e+000	2.000000000000000e+000	7.863926688491360 6e-001	7.863926688491361 0e-001
2	2.414433053362217 7e+010	1.463655439344452 3e+000	7.797681259088586 9e-001	7.797679054388686 5e-001
4	Inf	1.070760934477751 1e+000	7.797680685637893 3e-001	7.797680685637893 3e-001
6	Inf	8.284317475799675 1e-001		
8	Inf	7.863926688491360 6e-001		
10	Inf			
OPERACIONES	DIVERGE	113	69	107
TIEMPO (s)	DIVERGE	6.000000000000227e-002	0.050000000000000	0.060000000000000

TABLA COMPARATIVA USANDO $g_3(x)$

Resultados obtenidos a partir del punto inicial resultante de aplicar Bisección en el intervalo [0, 1] con una tolerancia de 10^{-1}, y una tolerancia en los algoritmos de 10^{-15}

Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	8.7500000000000000e-001	8.6980841529529296e-001	8.7500000000000000e-001	8.7500000000000000e-001
5	Inf	Inf	7.7976806856378933e-001	8.4060441632362803e-001
10	Inf	Inf		7.8270527181257987e-001
15	Inf	Inf		7.7976806856378933e-001
OPERACIONES	DIVREGE	DIVERGE	83	329
TIEMPO (s)	DIVERGE	DIVERGE	0.05000000000000	0.05000000000000

Resultados obtenidos a partir del punto inicial $x_0 = 2$, y una tolerancia en los algoritmos de 10^{-15}

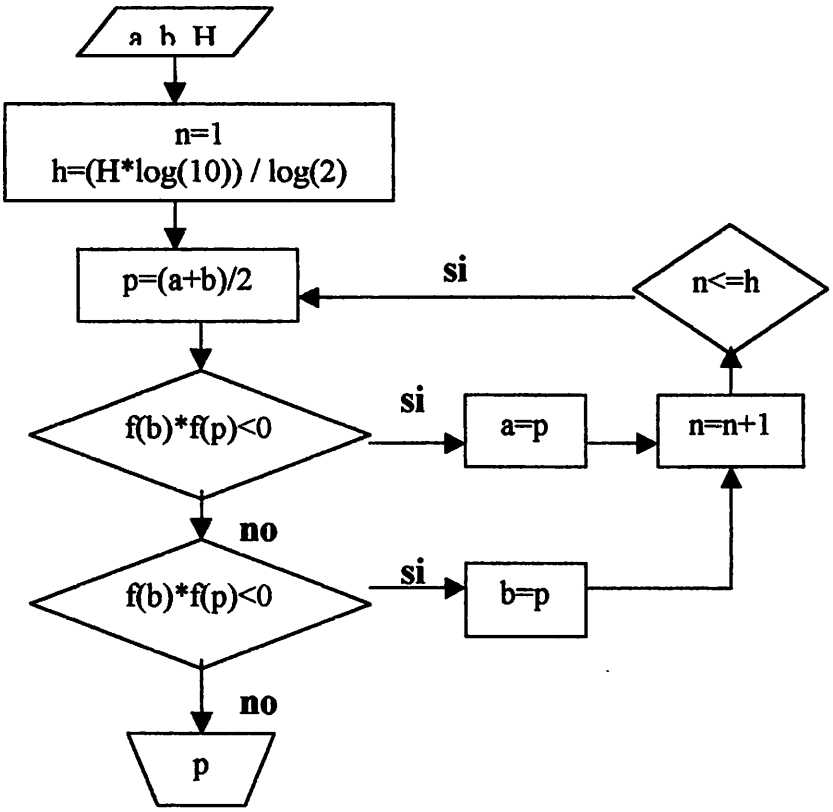
Iteración	PUNTO FIJO	AITKEN	NEWTON-RAPHSON	STEFFENSEN
0	2.0000000000000000e+000	2.0000000000000000e+000	2.0000000000000000e+000	2.0000000000000000e+000
6	Inf	Inf	8.2843174757996751e-001	1.9999999999997529e+000
11	Inf	Inf	7.7976806856378933e-001	1.9999999999995763e+000
OPERACIONES	DIVREGE	DIVREGE	167	CONV. LENTAMENTE
TIEMPO (s)	DIVERGE	DIVERGE	0.06000000000000	CONV. LENTAMENTE

Resultados obtenidos a partir del punto inicial $x_0 = 2$, con la diferencia de que los algoritmos de Newton-Raphson y Steffensen se ejecutan con una tolerancia de 10^{-15} y usan puntofijo con una tolerancia de 10^{-2} como punto inicial

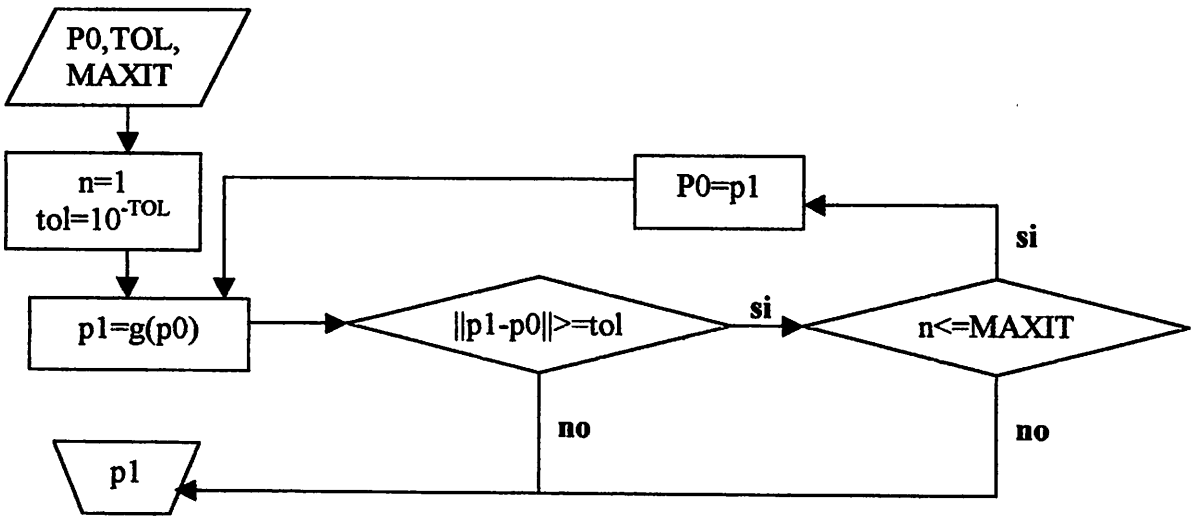
Iteración	PUNTO FIJO (10^{-15})	PUNTO FIJO (10^{-2} con $g_1(x)$)	NEWTON-RAPHSON	STEFFENSEN
0	2.0000000000000000e+000	2.0000000000000000e+000	7.8639266884913606e-001	7.8639266884913606e-001
2	2.4144330533622177e+010	1.4636554393444523e+000	7.7976812590885869e-001	7.7979085117416147e-001
4	Inf	1.0707609344777511e+000	7.7976806856378933e-001	7.7976806856379288e-001
6	Inf	8.2843174757996751e-001		7.7976806856378933e-001
8	Inf	7.8639266884913606e-001		
10	Inf			
OPERACIONES	DIVERGE	113	13	140
TIEMPO (s)	DIVERGE	6.000000000000227e-002	0.06000000000000	0.05000000000000

DIAGRAMAS DE FLUJO DE LOS ALGORITMOS

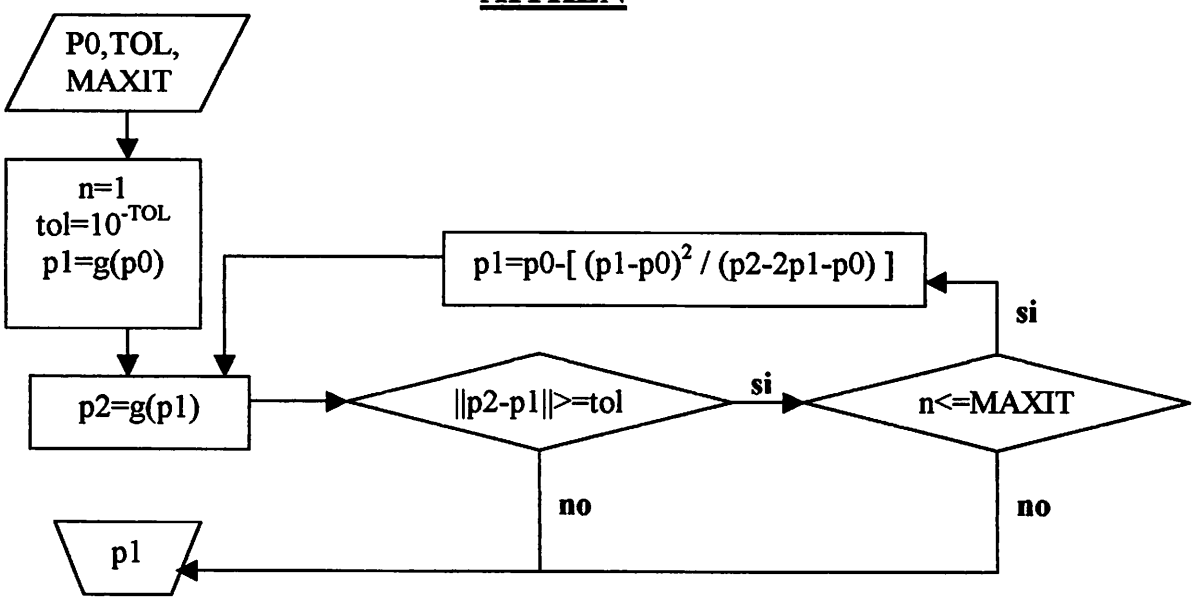
BISECCIÓN



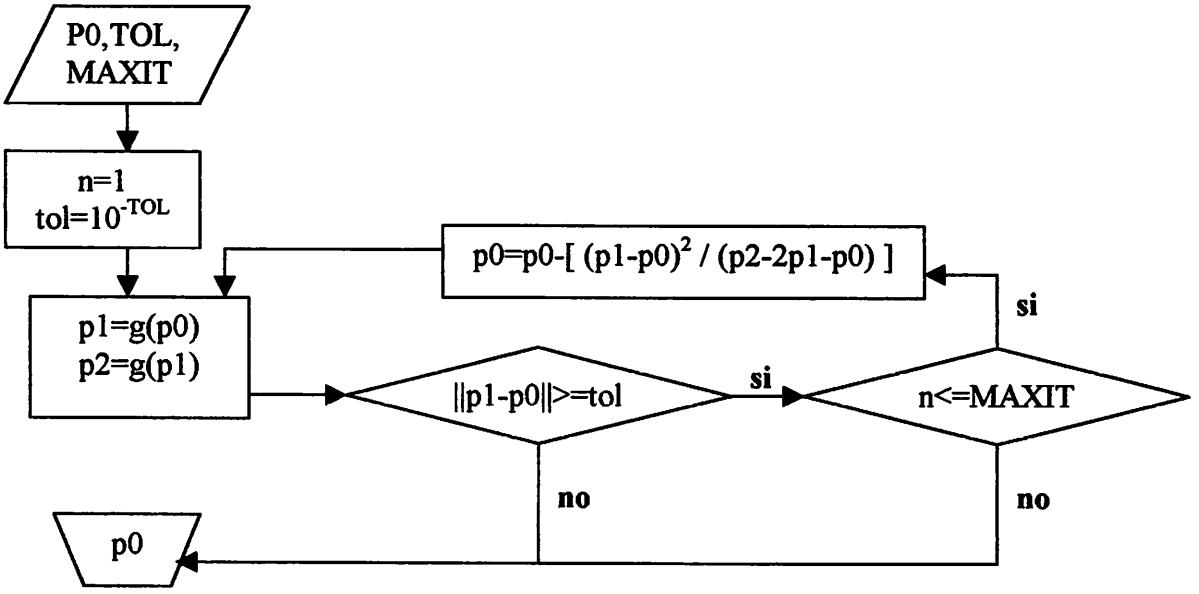
PUNTOFIJO



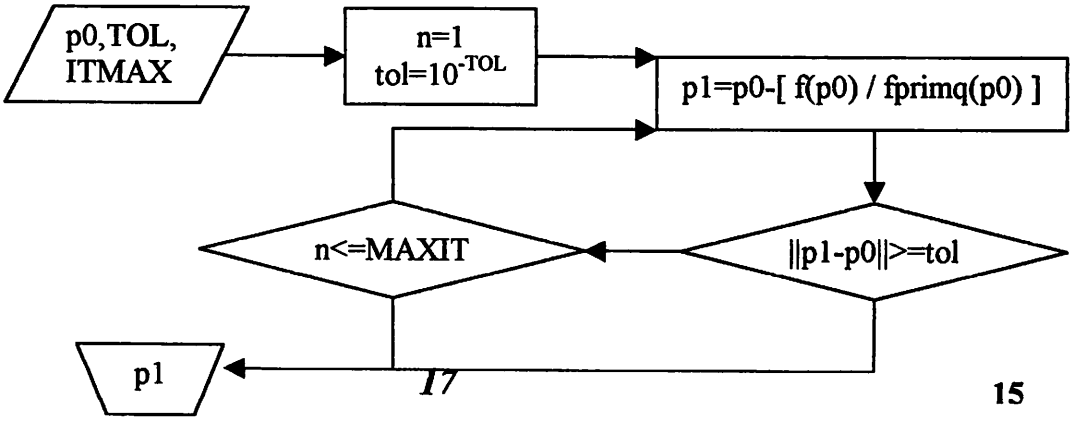
AITKEN



STEFFENSEN



NEWTON



DEFINICIÓN DE FUNCIONES QUE SE HAN UTILIZADO

Fichero f.m que define la función $f(x)=3x^7+x^3-1$.

```
function y=f(x)
format long
y=3*x.^7+x.^3-1;
```

Fichero fprima.m que define la función $fprima(x)=21x^6+3x^2$.

```
function y=fprima(x)
format long
y=21*x.^6+3*x.^2;
```

Fichero g1.m que define la función

```
function y=g1(x)
format long
y=x-(f(x)./fprima(x));
```

$$g_1(x) = x - \frac{3x^7 + x^3 - 1}{21x^6 + 3x^2}$$

Fichero g2.m que define la función

```
function y=g2(x)
format long
y=(1-3*x.^7)./x.^2;
```

$$g_2(x) = \frac{13x^7}{x^2}$$

Fichero g3.m que define la función

```
function y=g3(x)
format long
y=f(x)+x;
```

$$g_3(x) = 3x^7 + x^3 + x - 1$$

Fichero g1prima.m que define la función

```
function y=g1prima(x)
format long
y=((126*x.^5+6*x).*f(x))./fprima(x).^2;
```

$$g_{1'}(x) = \frac{(126x^5 + 6x)(3x^7 + x^3 - 1)}{(21x^6 + 3x^2).^2}$$

Fichero g2prima.m que define la función

```
function y=g2prima(x)
format long
y=(-15*x.^7-2)./x.^3;
```

$$g_{2'}(x) = \frac{-15x^7 - 2}{x^3}$$

Fichero g3prima.m que define la función

```
function y=g3prima(x)
format long
y=fprima(x)+1;
```

$$g_{3'}(x) = 21x^6 + 3x^2$$

IMPLEMENTACIÓN DE LOS MÉTODOS

```
function p=biseccion(a,b,H)
%function p=biseccion(a,b,H)
%       Metodo de biseccion para obtener un punto optimo
%       para ser usado por un mejor algoritmo de aproximacion
%       de raices en polinomios. Se calcula en el intervalo
%       [a,b] y con una cantidad de H decimales exactos.

format long
iniciomedida;
n=1;
h=round((H*log(10))/log(2));

datos(1,1)=mitad(a,b);
datos(1,2)=a;
datos(1,3)=b;
n=n+1;
while(n<=h)
  p=mitad(a,b);
        [a,b]=comprueba_signo(a,b,p);
  if(a~=b)
    datos(n,1)=p;
    datos(n,2)=a;
    datos(n,3)=b;
    n=n+1;
  else
    break
  end
end
convergencia(datos(:,1));
finmedida(h);
save c:\misdoc~1\biseccion.txt datos -ascii -double;

%--SUBFUNCIONES-----------------------------------------------------------------


function mit=mitad(a,b)
mit=(a+b)/2;

function [a,b]=comprueba_signo(a,b,p)
if (f(b)*f(p)<0)
 a=p;
elseif (f(a)*f(p)<0)
 b=p;
else
  disp('En el intervalo no existe una única raíz');
  a=b;
        break;
end
```

```
function p1=puntofijo(p0,TOLERANCIA,NUMG,MAXIT)
%function p1=puntofijo(p0,TOLERANCIA,NUMG,MAXIT)
%
% -p0 es el punto inicial
% -TOLERANCIA es el numero de digitos decimales exactos que se desean
% -NUMG selecciona la g(x) a analizar, pudiendo ser 1,2 ó 3
% -MAXIT es el máximo de iteraciones ha realizar
format long
tolerancia=10.^(-TOLERANCIA);
temp=p0+1;
n=1;
iniciomedida;
while(abs(temp-p0)>=tolerancia & n<=MAXIT | n==1)
  switch NUMG
  case 1,
        p1=g1(p0);
   case 2,
     p1=g2(p0);
  case 3,
     p1=g3(p0);
  otherwise,
     disp('FUNCION g(x) NO VALIDA');
  end
  temp=p0;
  p0=p1;
  p1=temp;
  datos(n,1)=p1;
  n=n+1;
end
finmedida(n);
convergencia(datos(:,1));
save c:\misdoc~1\puntofijo.txt datos -ascii -double
```

```
function p1=aitken(p0,TOLERANCIA,NUMG,MAXIT)
%function p1=aitken(p0,TOLERANCIA,NUMG,MAXIT)
%
% -p0 es el punto inicial
% -TOLERANCIA es el numero de digitos decimales exactos que se desean
% -NUMG selecciona la g(x) a analizar, pudiendo ser 1,2 ó 3
% -MAXIT es el máximo de iteraciones ha realizar
tolerancia=10.^(-TOLERANCIA);
iniciomedida;
switch NUMG
case 1, p1=g1(p0);
case 2, p1=g2(p0);
case 3, p1=g3(p0);
otherwise, disp('FUNCION g(x) NO VALIDA');
end
p2=p1;
n=1;                    % Indico que es la primera iteracion
while(abs(p1-p2)<tolerancia & n<=MAXIT)
  switch NUMG
        case 1, p2=g1(p2);
        case 2, p2=g2(p2);
        case 3, p2=g3(p2);
        otherwise, disp('FUNCION g(x) NO VALIDA');
  end
  temporal=(p2-2*p1+p0);
  if(temporal~=0)
    p0prima(n)=p0-(((p1-p0).^2)./temporal);
    n=n+1;
    p0=p1;p1=p2;
  else
    break;
  end
end
finmedida(n);
if(abs(p1-p2)<tolerancia)
  disp('Converge.');
elseif n>MAXIT
  disp('Se alcanzó el numero máximo de iteraciones.');
else
  disp('¡No converge!.');
end
convergencia(p0prima);
%Guardo solucion en archivo ascii
p0prima=p0prima';
save c:\misdoc~1\aitken1.txt p0prima -ascii -double
```

```
function p0=steffensen(p0,TOLERANCIA,NUMG,MAXIT)
%function p1=steffensen(p0,TOLERANCIA,NUMG,MAXIT)
%
% -p0 es el punto inicial
% -TOLERANCIA es el numero de digitos decimales exactos que se desean
% -NUMG selecciona la g(x) a analizar, pudiendo ser 1,2 ó 3
% -MAXIT es el máximo de iteraciones ha realizar
tolerancia=10.^(-TOLERANCIA);
n=1;
iniciomedida;
switch NUMG
 case 1, p1=g1(p0);
         p2=g1(p1);
 case 2, p1=g2(p0);
         p2=g2(p1);
 case 3, p1=g3(p0);
         p2=g3(p1);
 otherwise,
        disp('FUNCION g(x) NO VALIDA');
end
datos(n,1)=p1;
while(abs(p1-p2)>tolerancia & n<=MAXIT)
  switch NUMG
  case 1, p1=g1(p0);
          p2=g1(p1);
  case 2, p1=g2(p0);
          p2=g2(p1);
  case 3, p1=g3(p0);
          p2=g3(p1);
  otherwise,
          disp('FUNCION g(x) NO VALIDA');
  end
  temporal=(p2-2*p1+p0);
  if(temporal~=0)
    p0=p0-(((p1-p0).^2)./temporal);
    datos(n,1)=p0;
    n=n+1;
  else
    break;
  end
end
finmedida(n-1);
convergencia(datos(:,1));
save c:\misdoc~1\steffensen.txt datos -ascii -double
```

```
function p1=newton(p0,tol,ITMAX)
%function p1=newton(p0,tol,ITMAX)
% A partir de un p0 calcula una buena raiz aproximada para la funcion f(x)
% ITMAX marca el numero maximo de iteraciones.
%
% -tol es la tolerancia en numero de cifras decimales
n=1;

TOL=10.^-tol;
temp=p0;
p1=p0+1;
datos(1,n)=p0;
iniciomedida;
temporal=fprima(p0);
while(abs(p1-temp)>TOL & temporal~=0 & n<ITMAX)
  p1=p0-(f(p0)/temporal);
        if(p1~=p0)
    temp=p0;
    p0=p1;
    n=n+1;
    datos(n,1)=p0;
    temporal=fprima(p0);
  else
    break;      %% Si ya no se pueden representar mas cifras decimales exactas
  end
end
finmedida(n);
convergencia(datos(:,1));
save c:\misdoc~1\newton.txt datos -ascii –double

function iniciomedida
%CONTADOR DE OPERACIONES A 0 E INCIALIZO EL TEMPORIZADOR
flops(0),tic

function finmedida(h)
% IMPRIMO EL NUMERO DE OPERACIONES EMPLEADAS
disp(' ');
disp('NUMERO DE OPERACIONES:'),disp(flops);
disp('TIEMPO EMPLEADO (segundos):'),disp(toc);
disp('ITERACIONES:'),disp(h);
```

RESOLUCIÓN DE SISTEMAS DE ECUACIONES NO LINEALES

-Análisis Numérico I-

Víctor Manuel Galdámez Salguero
Jose Manuel Mesa Gómez

- ÍNDICE -

1. INTRODUCCIÓN.....*3*
2. MÉTODOS NUMÉRICOS.....*5*
2.1. PUNTO FIJO Y SEIDEL.....*5*
2.1.1. PUNTO FIJO.....**5**
2.1.2. SEIDEL.....**8**
2.1.3. COMPARATIVA PUNTO FIJO – SEIDEL.....**9**
2.2 NEWTON Y NEWTON-GLOBAL.....*11*
2.2.1. NEWTON.....**11**
2.2.2. NEWTON-GLOBAL.....**12**
2.2.3. COMPARATIVA NEWTON – NEWTON-GLOBAL..... **14**
2.3. BROYDEN.....*17*
2.4. MÁXIMO DESCENSO.....*19*
3. COMPARATIVA DE TODOS LOS MÉTODOS ESTUDIADOS.....*24*
4. SOLUCIONES OBTENIDAS EN LOS DISTINTOS MÉTODOS.....*26*

ANEXO I: GRÁFICAS DE LA DINÁMICA DE LOS MÉTODOS.....*27*
NEWTON.....**27**
NEWTON-GLOBAL($\lambda=2^{-L}$).....**29**
NEWTON-GLOBAL ($\lambda=L$).....**32**
NEWTON-GLOBAL ($\lambda=1.5^{-L}$).....**34**
BROYDEN.....**36**
MÁXIMO DESCENSO ($\lambda=2^{-L}$).....**38**
MÁXIMO DESCENSO ($\lambda=1.5^{-L}$).....**40**
MÁXIMO DESCENSO ($\lambda=3^{-L}$).....**42**

ANEXO II: GRÁFICAS INTERESANTES.....*44*
GRÁFICA DE F(X,Y) EN TRES DIMENSIONES.....**44**
GRÁFICA DE G(X,Y) EN TRES DIMENSIONES.....**44**
GRÁFICA DEL SISTEMA EN TRES DIMENSIONES.....**45**
CURVAS DE NIVEL DEL SISTEMA.....**45**
VISTA DE ALGUNAS SOLUCIONES EN TRES DIMENSIONES.....**46**

ANEXO III – IMPLEMENTACIONES.....*47*

FUNCIONES AUXILIARES.....**47**

MÉTODO DEL PUNTO FIJO.....**52**

MÉTODO DE SEIDEL.....**52**

MÉTODO DE NEWTON (CON GRÁFICAS).....**53**

MÉTODO DE NEWTON (SIN GRÁFICAS).....**55**

MÉTODO DE NEWTON GLOBAL(CON λ=CTE^{-L} Y CON GRÁFICAS).....**56**

MÉTODO DE NEWTON GLOBAL(CON λ=CTE^{-L} Y SIN GRÁFICAS).....**58**

MÉTODO DE NEWTON GLOBAL (CON λ=L Y CON GRÁFICAS).....**59**

MÉTODO DE NEWTON GLOBAL (CON λ=L Y SIN GRÁFICAS).....**60**

MÉTODO DE BROYDEN (CON GRAFICAS).....**61**

MÉTODO DE BROYDEN (SIN GRAFICAS).....**62**

MÉTODO DE MÁXIMO DESCENSO(CON λ=CTE^{-L} Y CON GRAFICAS).**63**

MÉTODO DE MÁXIMO DESCENSO(CON λ=CTE^{-L} Y SIN GRAFICAS)...**65**

FUNCIONES PARA MEDIDAS.....**66**

1. INTRODUCCIÓN

En esta práctica se plantea la resolución de sistemas de ecuaciones no lineales (S. E. N. L.) con n incógnitas. La forma general de un sistema de ecuaciones no lineales es:

$$f_1(x_1, x_2, \ldots, x_n) = 0$$
$$f_2(x_1, x_2, \ldots, x_n) = 0$$
$$\ldots$$
$$f_n(x_1, x_2, \ldots, x_n) = 0$$

Para la resolución de dichos sistemas se utilizan los siguientes métodos:

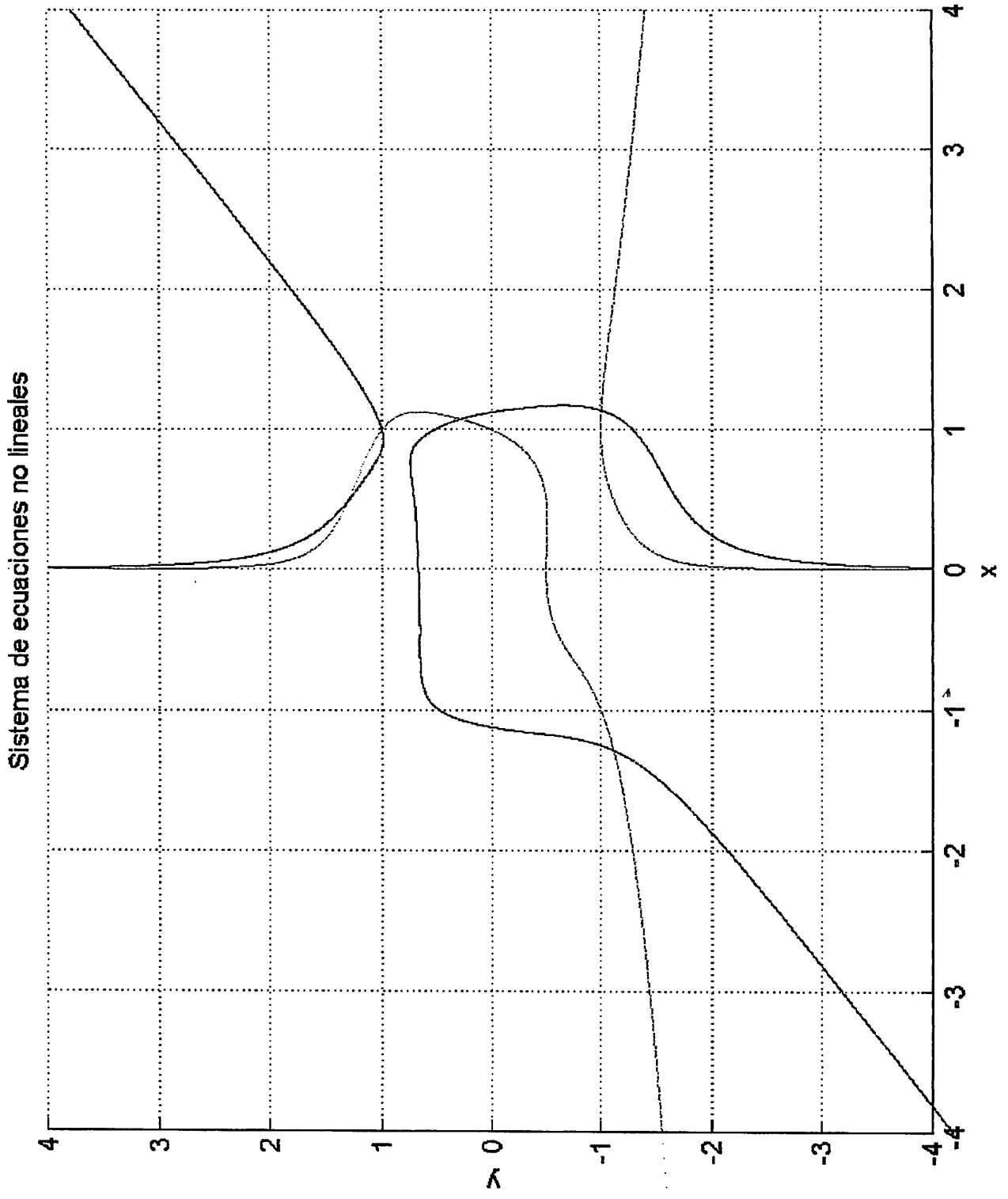
- Punto fijo
- Seidel
- Newton-Raphson
- Newton Global
- Broyden (Quasi-Newton)
- Máximo Descenso

Durante la práctica nos vamos a centrar en un sistema de ecuaciones no lineales específico, al cual aplicaremos los métodos anteriormente mencionados, que es el siguiente:

$$x^3+x^2y+y^7x-2y-1=0$$
$$x^4y-x^6+y^5x-3y+2=0$$

Un vez planteado el problema el primer paso para solucionarlo será hacer un estudio gráfico del S. E. N. L. para poder así ver las soluciones del sistema, si es que las tiene. Se nos plantea un nuevo problema que es el dibujar dichas ecuaciones, para solventarlo acudimos a uno de los métodos estudiados en la práctica anterior, como es el método de Newton para ecuaciones no lineales de una sola variable. Partiendo de un punto inicial dado el proceso consta de un doble barrido en el eje X, de modo que tomando un 'y' inicial se aplica Newton para cada 'x' del intervalo establecido obteniendo así un nuevo punto 'y' que servirá como punto 'y' inicial para el siguiente 'x'. De igual modo lo hacemos para el eje Y, con la diferencia de que lo que calculamos son puntos en 'x' y no en 'y'.

La representación gráfica obtenida se muestra en la siguiente página.


Sistema de ecuaciones no lineales
x
y
4
3
2
1
0
-1
-2
-3
-4

Como se puede observar en la gráfica anterior hay varios puntos de corte entre ambas funciones, siendo dichos puntos soluciones del sistema.

Una vez obtenida una primera aproximación gráfica de las soluciones del sistema podemos aplicar los algoritmos con diferentes puntos de inicio y ver si convergen o no hacia una de estas soluciones, y si convergen, con que velocidad lo hacen.

2. MÉTODOS NUMÉRICOS

2.1. PUNTOFIJO Y SEIDEL

2.1.1. PUNTO FIJO

Observando la forma general de un S. E. N. L. (expuesta anteriormente), puede pensarse cada función f_i como un mapeo de un vector $x = (x_1, x_2, \dots, x_n)^t$ del espacio n-dimensional R^n, en la recta real R. Alternativamente, el sistema puede representarse definiendo una función F, de R^n en R^n por

$$F(x_1, x_2, \dots, x_n) = (f_1(x_1, x_2, \dots, x_n), f_2(x_1, x_2, \dots, x_n), \dots, f_n(x_1, x_2, \dots, x_n))^t$$

Usando notación vectorial para representar las variables $x_1, x_2, \dots, x_n$, el sistema asume la forma F(x)=0. Las funciones $f_1,f_2,\dots,f_n$ se llaman las *funciones coordenadas de F*.

En la práctica anterior se desarrollo un proceso iterativo para resolver la ecuación f(x)=0 transformando primero esta ecuación en una ecuación de la forma x=g(x). La función g tiene sus puntos fijos precisamente en las soluciones de la ecuación original. A diferencia de la práctica anterior las funciones son de R^n en R^n resultando la transformación del siguiente modo (Punto fijo aplicado a n-dimensiones):

$$x_1=g_1(x_1,x_2, \dots, x_n)$$
$$x_2=g_2(x_1,x_2, \dots, x_n)$$
$$\dots$$
$$x_n=g_n(x_1,x_2, \dots, x_n)$$

Definición: Se dice que una función G de D c R^n a R^n tiene un punto fijo en p ϵ D si G(p) es igual a p.

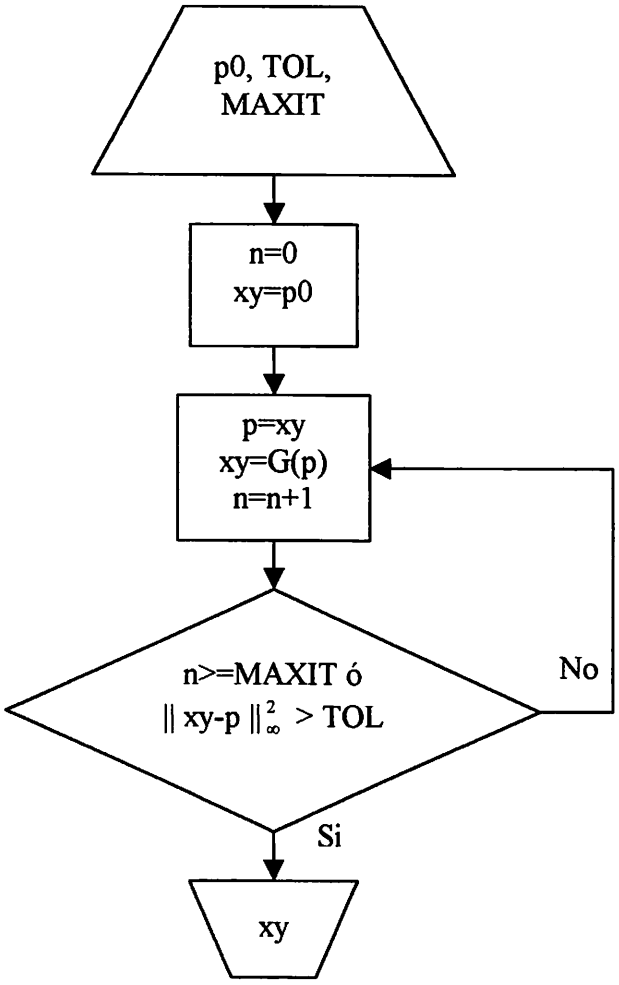
La forma de obtener las g_i no es de forma unívoca, lo que nos puede llevar a que el método no converja, o converja mas o menos rápidamente dependiendo de las g_i elegidas. Veamos a continuación un teorema que nos asegura la buena elección de las g_i.

Teorema: Sea D = { $(x_1,x_2, \ldots , x_n)^t$ | $a_i \le x_i \le b_i$ para cada i=1, 2, ..., n } para alguna colección de constantes $a_1,a_2, \ldots , a_n$ y $b_1,b_2, \ldots , b_n$. Supongamos que G es una función continua con primeras derivadas parciales continuas de D c R^n a R^n con la propiedad de que G(x) perteneciente a D para x perteneciente a D. Entonces G tiene un punto fijo en D. Además, supóngase que existe una constante k<1 con

$$\left|\frac{\partial g_i(x)}{\partial x_j}\right| \le \frac{k}{m} \quad \text{siempre que } x \in D,$$

Para cada j = 1, 2, ... , n y para cada función componente g_i . Entonces la sucesión $\{x^{(k)}\}_{k=0}^{\infty}$ definida por una $x^{(0)}$ en D seleccionada arbitrariamente y generada por $x^{(k)} = G(x^{(k-1)})$ para $k \ge 1$ converge al punto fijo único p perteneciente a D y $\|x^{(k)} - p\|_\infty \le \frac{k^k}{1-k}\|x^{(1)} - x^{(0)}\|_\infty$.

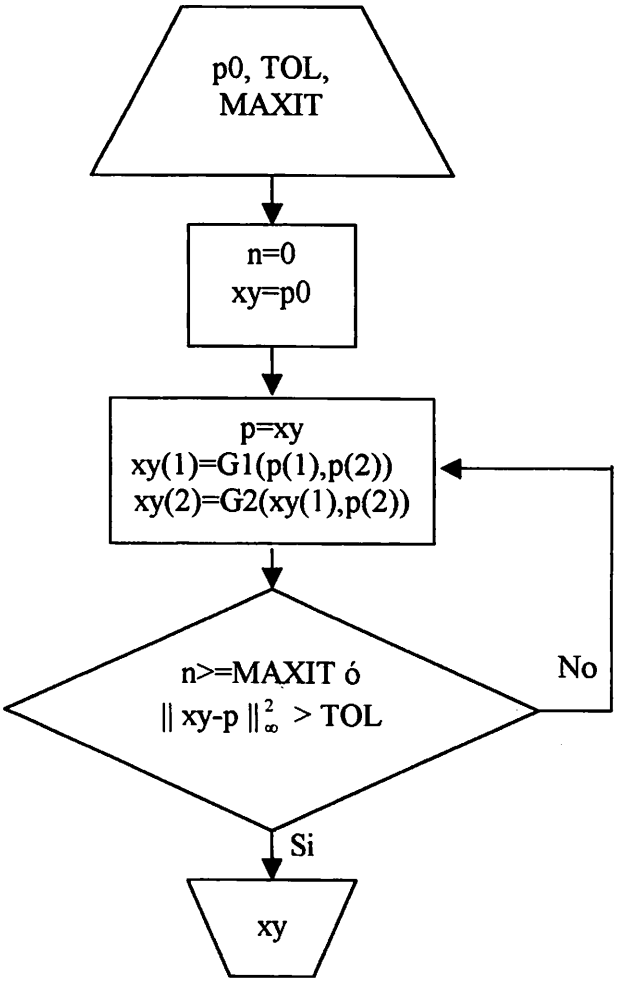
DIAGRAMA DE FLUJO DEL MÉTODO DEL PUNTO FIJO



2.1.2. SEIDEL

Este método sigue los mismos pasos que el método del punto fijo, con la diferencia de que en punto fijo se calcula un punto (x, y) en cada iteración que se utiliza para el cálculo del siguiente, mientras que Seidel en cada iteración calcula un 'x', y para el cálculo de 'y' se utiliza la 'x' hallada en la misma iteración, consiguiendo así una mejor 'y', ya que la 'x' hallada es mejor que la de entrada.

DIAGRAMA DE FLUJO DEL MÉTODO DE SEIDEL



2.1.3. COMPARATIVA PUNTOFIJO – SEIDEL

Para la comparativa vamos a utilizar el siguiente sistema, ya que habiendo intentado buscar g que hicieran que el sistema anterior convergiera no lo hemos conseguido.

$$\begin{cases} 3x - \cos(yz) - 1/2 = 0 \\ x^2 - 81(y+0.1)^2 + sen(z) + 1.06 = 0 \\ e^{-xy} + 20z + \dfrac{10\pi - 3}{3} = 0 \end{cases} \qquad (\bar{x}=g(\bar{x})) \equiv \begin{cases} x = \dfrac{1}{3}\cos(yz) + \dfrac{1}{6} \\ y = \dfrac{1}{9}\sqrt{x^2 + sen(z) + 1.06} - 0.1 \\ z = -\dfrac{1}{20}e^{-xy} - \dfrac{10\pi - 3}{60} \end{cases}$$

PUNTO FIJO			
P_0	Iteraciones	Tiempo	Operaciones
(0.1,0.1,-0.1)	11	4.999999999999716e-002	494
(-2, -1, 2)	11	5.000000000000071e-002	494
(4, 3.5, 5)	12	4.999999999999716e-002	539
(10, -10, 10)	1000	1.370000000000001	273786
(1, -1, 1)	8	6.000000000000227e-002	359
SEIDEL			
P_0	Iteraciones	Tiempo	Operaciones
(0.1,0.1,-0.1)	8	6.000000000000227e-002	359
(-2, -1, 2)	6	5.999999999999517e-	269
(4, 3.5, 5)	6	4.999999999999716e-002	269
(10, -10, 10)	7	5.999999999999872e-002	314
(1, -1, 1)	5	4.999999999999716e-002	224

PUNTO FIJO			
P_0	Tolerancia	Solución (x , y , z)	$\|p1-p0\|_2$ $\|p1-p0\|_\infty$
(0.1,0.1,-0.1)	10^{-15}	5.000000000000000e-001 2.775557561562891e-017 -5.235987755982988e-001	7.7715611723760960e-016 7.7913613603198810e-016
(-2, -1, 2)	10^{-13}	5.000000000000000e-001 -1.804112415015879e-016 -5.235987755983000e-001	4.5949355431673660e-014 4.6002974308667360e-014
(4, 3.5, 5)	10^{-15}	5.000000000000000e-001 -4.1633363423443360e-017 -5.2359877559829880e-001	7.7715611723760960e-016 7.9200718613338520e-016
(10, -10, 10)	10^{-15}	NaN NaN NaN	NaN NaN
(1, -1, 1)	10^{-8}	5.000000000000000e-001 -3.2906261049348020e-011 -5.2359877559601920e-001	6.1782867710746810e-010 6.3016785537640580e-010

SEIDEL			
P_0	Tolerancia	Solución (x , y , z)	$\|p1-p0\|_2$ $\|p1-p0\|_\infty$
(0.1,0.1,-0.1)	10^{-15}	5.000000000000000e-001 -1.3877787807814460e-017 -5.2359877559829880e-001	9.7144514654701180e-017 9.7144514654701180e-017
(-2, -1, 2)	10^{-13}	5.000000000000000e-001 -2.7755575615628910e-017 -5.2359877559829880e-001	1.1074474670635940e-014 1.1079482063497080e-014
(4, 3.5, 5)	10^{-15}	5.000000000000000e-001 -1.3877787807814460e-017 -5.2359877559829880e-001	1.2490090027033010e-016 1.2490090027033010e-016
(10, -10, 10)	10^{-15}	5.000000000000000e-001 -1.3877787807814460e-017 -5.2359877559829880e-001	1.3877787807814460e-017 1.3877787807814460e-017
(1, -1, 1)	10^{-8}	5.000000000000000e-001 1.0444423104161160e-013 -5.2359877559829630e-001	7.8041059725642020e-011 7.8065445733392160e-011

Como podemos apreciar en las tablas anteriormente expuestas el método de Seidel converge en menos iteraciones y usando menos operaciones que el de Punto Fijo, e incluso, se da el caso de que para un determinado punto Seidel converge y Punto Fijo no.

2.2 NEWTON Y NEWTON-GLOBAL

2.2.1 NEWTON

Para construir el algoritmo que nos llevó a un método apropiado de punto fijo en el caso unidimensional, tratamos de encontrar una función ϕ con la propiedad de que $g(x) = x - \phi(x)f(x)$, diera convergencia cuadrática al punto fijo p de g. De esta condición surgió el método de Newton, escogiendo ϕ(x)=1/f '(x). Usando un enfoque similar para el caso n-dimensional es necesario una matriz

$$A(x) = \begin{bmatrix} a_{11}(x) & a_{12}(x) & \dots & a_{1n}(x) \\ a_{21}(x) & a_{21}(x) & \dots & a_{2n}(x) \\ \dots & \dots & \dots & \dots \\ a_{n1}(x) & a_{n2}(x) & \dots & a_{nn}(x) \end{bmatrix}$$

donde cada una de las componentes $a_{ij}(x)$ es una función de R^n a R. El procedimiento requiere que se encuentre A(x) tal que

$$G(x) = x - A(x)^{-1}F(x)$$

de convergencia cuadrática a la solución de F(x)=0, siempre que, desde luego, A(x) sea no-singular en el punto fijo de G.

Teorema (Elección de A(x)): Supóngase que p es una solución de G(x)=x para alguna función $G=(g^1, g^2, \dots, g^n)^t$, que manda a R^n en R^n. Si existe un número $\delta > 0$ con la propiedad de que

i) $\frac{\partial g_i}{\partial x_j}$ es continua en $N_\delta = \{x \mid \| x - p \| < \delta\}$ para cada i=1,2, .. ,n y j=1,2, .. , n.

ii) $\frac{\partial^2 g_i(x)}{\partial x_j \partial x_k}$ es continua y $\left|\frac{\partial^2 g_i(x)}{\partial x_j \partial x_k}\right| \le M$ para alguna constante M., siempre que x pertenezca a N_δ para cada i = 1, 2, ... , n , j = 1, 2, ... , n y k = 1, 2, ... , n.

iii) $\frac{\partial g_i(p)}{\partial x_j} = 0$ para cada i = 1, 2, ... , n y j = 1, 2, ... , n .

entonces existe un número $\hat{\delta} < \delta$ tal que la sucesión generada por $x^{(k)} = G(x^{(k-1)})$ converge cuadráticamente a p para cualquier $x^{(0)}$ siempre que $\| x^{(0)} - p \| < \hat{\delta}$. Además

$$\left\| x^{(k)} - p \right\|_\infty \le \frac{n^2 M}{2} \left\| x^{(k-1)} - p \right\|_\infty^2 \text{ para cada } k \ge 1.$$

Una elección apropiada para A(x) es J(x) (matriz Jacobiana) ya que cumple el teorema anterior, quedando la función G definida como:

$$G(x) = x - J(x)^{-1} F(x)$$

Este método se llama, con toda razón, método de Newton para sistemas no lineales y se espera que generalmente dé convergencia cuadrática siempre y cuando se conozca un valor inicial lo suficientemente exacto y que $J(p)^{-1}$ exista.

Una debilidad clara del procedimiento del método de Newton surge de la necesidad de invertir la matriz J(x) en cada paso. En la práctica, el método se realiza generalmente en una forma de dos pasos. Primero, se encuentra un vector 'y' que satisfaga $J(x^{(k)})y = -F(x^{(k)})$. Después de que se ha logrado esto, la nueva aproximación $x^{(k+1)}$, puede obtenerse sumando 'y' a $x^{(k)}$.

2.2.2 NEWTON-GLOBAL

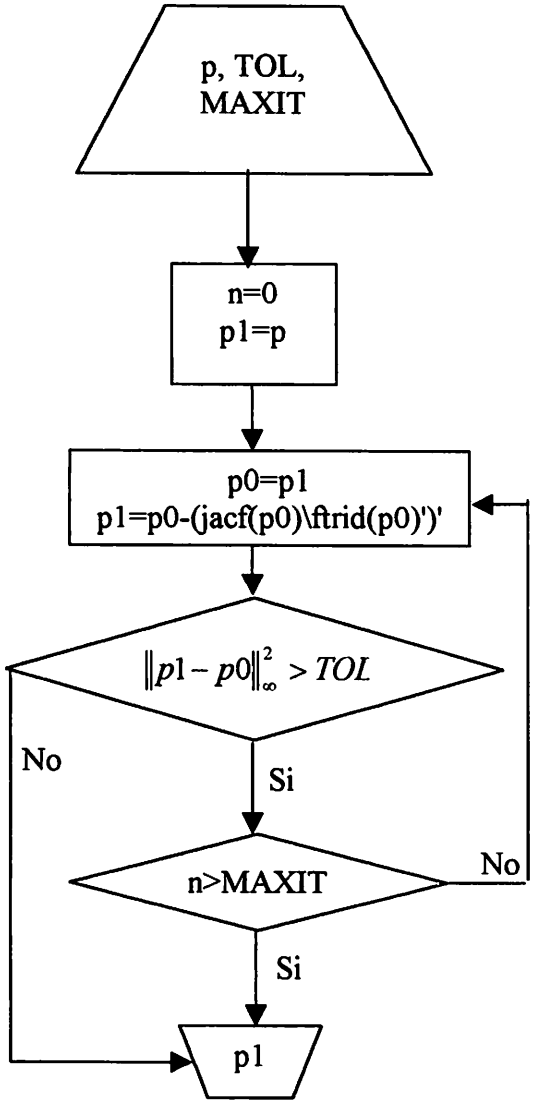
Este método es similar al anterior con la ventaja de que no necesita un punto inicial próximo a una solución del sistema para converger, esto lo consigue introduciendo un parámetro λ en el cálculo del nuevo punto: $G(x) = x - \lambda J(x)^{-1} F(x)$ con $\lambda > 0$.

Definición (altura de un punto): Se define una función $h(\bar{x})$ de R^n en R que asocia a cada punto $\bar{x} = (x_1, x_2 , \ldots , x_n)$ con una altura $h(\bar{x})$ de la siguiente manera $h(\bar{x}) = f_1^2(\bar{x}) + \ldots + f_n^2(\bar{x})$

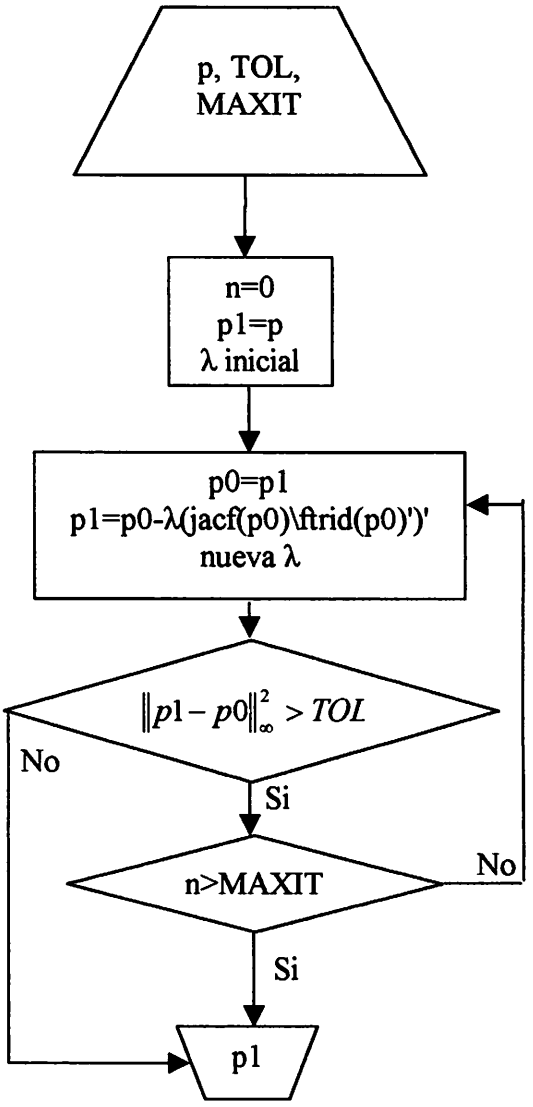
Nota: El punto que haga mínima a h es solución del sistema.

El parámetro lambda será aquel valor que minimice $h(x - \lambda J(x)^{-1} F(x))$.

DIAGRAMA DE FLUJO DEL MÉTODO NEWTON Y NEWTON-GLOBAL



Newton



Newton-Global

2.2.3. COMPARATIVA NEWTON – NEWTON-GLOBAL

Como hemos visto anteriormente la diferencia entre Newton y Newton-Global radica en el parámetro λ existente en Newton-Global para el cálculo del punto siguiente. Pasamos a comparar Newton y Newton-Global para diferentes λ.

MÉTODOS	PUNTOS	SOLUCIONES	TIEMPO (s)	Nº OPs	IT
NEWTON	(-1, -1.2)	4.8656218713152 58e-001	1.308266172673521	4.99999999999893e-002	1431	11
	(0.2, 1)	1.000000000000000	1.000000000000000	5.99999999999517e-002	707	5
	(0.35, 0.1)	1.137385442524952	-1.005461315459772	4.99999999999716e-002	1795	14
	(1, 1.1)	1.000000000000000	1.000000000000000	4.99999999999716e-002	1427	11
	(-3, 3)	-1.285218955007076	-1.113922406792850	6.00000000000050e-002	951	7
NEWTON GLOBAL ($\lambda=2^{-1}$)	(-1, -1.2)	-1.285218955526369	-1.113922406868765	1.54000000000001	75398	682
	(0.2, 1)	1.000000005149820	0.999999971263023	1.97000000000006	94120	784
	(0.35, 0.1)	1.137385443390853	-1.005461315500677	2.64000000000001	120169	1001
	(1, 1.1)	1.000000000428081	1.000000002567967	1.10000000000000	55953	466
	(-3, 3)	0.486562185262050 5	1.308266172925876	2.20000000000003	102644	855
NEWTON GLOBAL ($\lambda=L$, $L=L+0.001$)	(-1, -1.2)	-1.285218955007076	-1.113922406792850	0.60999999999999	29772	250
	(0.2, 1)	0.486562187131525 9	1.308266172673521	0.42999999999999	29941	251
	(0.35, 0.1)	1.073727421928294	0.281001205639445 8	0.42999999999999	29935	251
	(1, 1.1)	1.000000000000000	1.000000000000000	0.42999999999999	29653	249
	(-3, 3)	1.000000000000000	0. 999999999999998	0.71999999999998	44075	370
TOLERANCIA 10^{-15}						

MÉTODOS	PUNTOS	SOLUCIONES	TIEMPO (s)	Nº OPs	IT
TOLERANCIA 10^{-8}					
NEWTON	(-1, -1.2)	-1.28521895507076 -1.11392240679285	6.00000000000227e-002	831	6
	(0.2, 1)	1. 00000000000000 1. 00000000000000	5.99999999999517e-002	1311	10
	(0.35, 0.1)	1.13738544252495 2 -1.00546131545977 2	6.00000000000227e-002	1675	13
	(1, 1.1)	1.00000000000000 1. 00000000000000	6.00000000000227e-002	591	4
	(-3, 3)	4.86562187131525 8e-001 1.30826617267352 1	5.99999999999517e-002	1431	11
NEWTON GLOBAL ($\lambda=2^{-1}$)	(-1, -1.2)	-1.28522205494278 0 -1.11392285996224 3	2.69000000000000	213267	432
	(0.2, 1)	1.00000887754613 6 9.99995046242602 6	3.51000000000002	284354	601
	(0.35, 0.1)	1.13739061151456 1 -1.00546155965123 9	4.71999999999999	374196	800
	(1, 1.1)	1.00000073798791 6 1.00000442690744 7	1.82000000000000	147355	289
	(-3, 3)	4.86551027169463 5e-001 1.30826767911535 7	3.62999999999999	289337	631
NEWTON GLOBAL ($\lambda=1.5^{-1}$)	(-1, -1.2)	-1.28522205493808 5 -1.11392285996155 6	3.35000000000002	259589	581
	(0.2, 1)	1.00000887753962 2 9.99995046246239 4	4.39000000000000	347802	807
	(0.35, 0.1)	1.13739061150916 4 -1.00546155965098 3	5.88000000000003	458896	1075
	(1, 1.1)	1.00000073798711 6 1.00000442690764 4	2.25000000000000	177847	388
	(-3, 3)	4.86551027170658 0e-001 1.30826767911539 5	4.56000000000002	356599	848

Como se puede apreciar en las dos tablas anteriores Newton converge a alguna solución aún cuando el punto de inicio está alejado de la solución, sin embargo esto no tiene porque ocurrir, de hecho lo normal sería que hubiera que darle un punto cercano para que convergiera. (Depende por tanto del sistema estudiado).

En la primera tabla (tolerancia 10^{-15}) se exponen los resultados obtenidos de evaluar Newton y Newton-Global con diferentes estrategias para λ, siendo estas estrategias las siguientes:

1) $\lambda = 2^{-L}$ con L=L+1
2) $\lambda = L$ con L=L+0.001

observándose que Newton converge más rápidamente para cualquier punto, ya sea lejano ([-3, 3]) o próximo a solución. Respecto a las dos estrategias utilizadas se puede ver que la primera (1) converge más lentamente y con un número mayor de operaciones que la segunda (2). Esto es debido a que en la estrategia 1) el salto que se produce es inicialmente mayor que en la segunda estrategia, pudiendo hacer esto que se aleje más de la solución respecto a la otra estrategia.

En la segunda tabla (tolerancia 10^{-8}) se exponen los resultados obtenidos de evaluar Newton y Newton-Global con diferentes estrategias para λ, siendo estas estrategias las siguientes:

1) $\lambda = 2^{-L}$ con L=L+1
3) $\lambda = 1.5^{-L}$ con L=L+1

observándose que, al igual que en la tabla anterior, Newton converge más rápidamente. En cuanto a las dos estrategias utilizadas en esta tabla la primera converge en menos iteraciones, debido a que el salto se hace cada vez más pequeño pero más rápidamente que en el otro caso.

2.3. BROYDEN

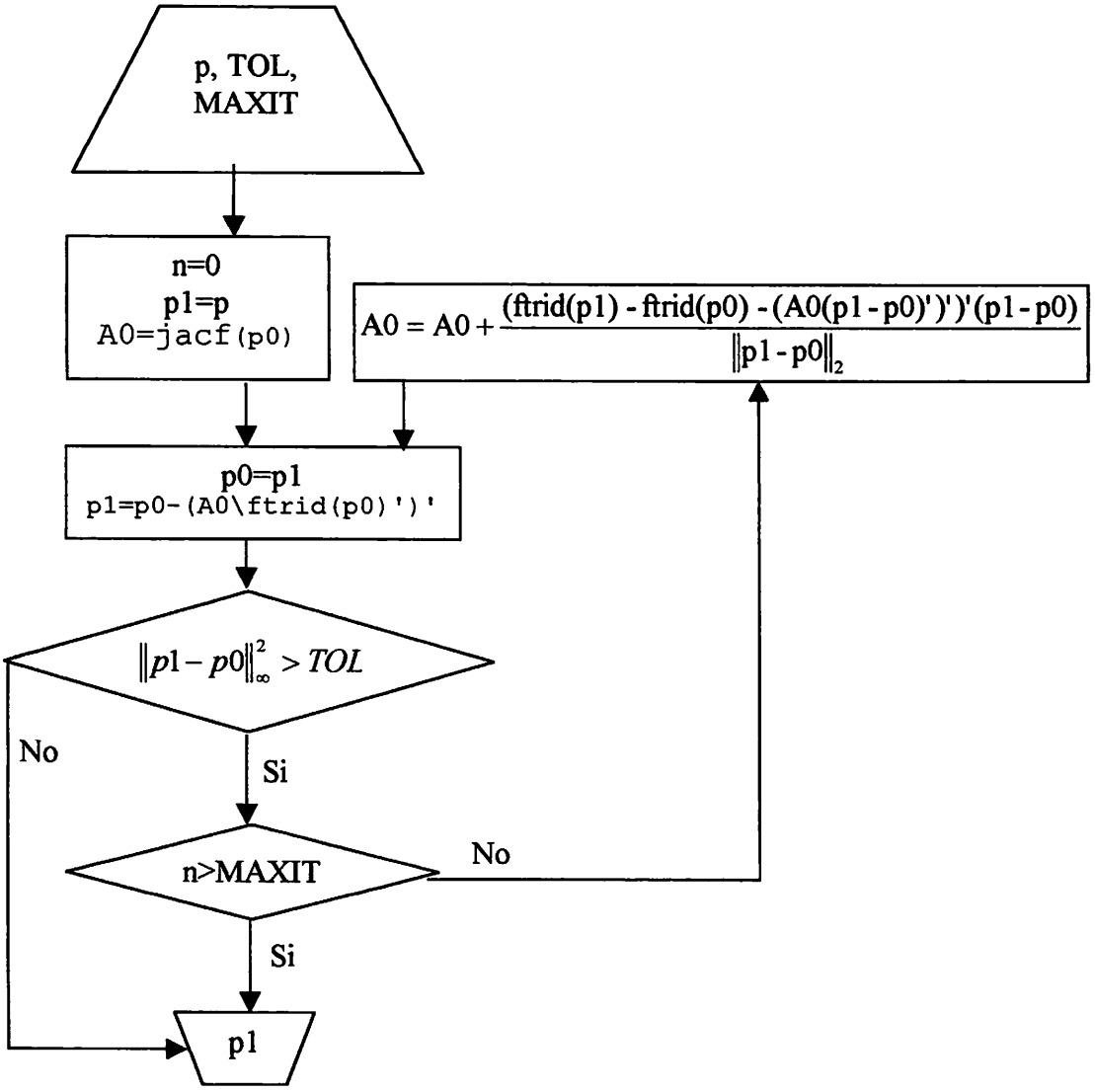
Una debilidad importante del método de Newton para resolver sistemas de ecuaciones no lineales es la necesidad de calcular la matriz Jacobiana en cada iteración y de resolver un sistema lineal de n x n asociado a esta matriz. Para ilustrar la magnitud de esta debilidad consideramos la cantidad del cálculo necesario para llevar a cabo una iteración del método de Newton. La matriz Jacobiana asociada a un sistema de n ecuaciones no lineales escritas en la forma F(x)=0 requiere de la determinación y evaluación de las n^2 derivadas parciales de las n componentes de F. En la mayoría de los casos, la evaluación exacta de las derivadas parciales es complicada y en muchas ocasiones imposible. Para sobrepasar esta dificultad se pueden usar aproximaciones de diferencias finitas a las derivadas parciales. Por ejemplo,

$$\frac{\partial f_j}{\partial x_k}(x^{(i)}) \approx \frac{f_j(x^{(i)} + e_k h) - f_j(x^{(i)})}{h}$$

donde h es pequeña en valor absoluto y e_k es el vector cuyo único elemento diferente de cero es un uno en la k-ésima coordenada. Esta aproximación, sin embargo, requiere aún de la realización de por lo menos n^2 evaluaciones funcionales escalares para aproximar el Jacobiano y no reduce la cantidad de cálculos que es en general $O(n^3)$, para resolver el sistema lineal que contiene al Jacobiano aproximado. El esfuerzo computacional total para solo una iteración del método de Newton es entonces de por lo menos, n^2+n evaluaciones funcionales escalares (n^2 para la evaluación de la matriz Jacobiana y n para la evaluación de F) junto con $O(n^3)$, operaciones aritméticas para resolver el sistema lineal. Esta cantidad de esfuerzo computacional es prohibitiva excepto para valores relativamente pequeños de n y para funciones escalares fáciles de evaluar.

Por otro lado el método de Broyden requiere solamente de n evaluaciones funcionales escalares por iteración y reduce también el número de cálculos aritméticos a $O(n^2)$. Es uno de los métodos conocidos como renovaciones de secante de mínimo cambio que producen los algoritmos llamados cuasi-Newton. Estos métodos reemplazan a la matriz Jacobiana en el método de Newton por una matriz de aproximación que se renueva en cada iteración. La desventaja de estos métodos consiste en que se pierde la convergencia cuadrática del método de Newton y se reemplaza por una convergencia llamada superlineal.

DIAGRAMA DE FLUJO DEL MÉTODO DE BROYDEN



2.4. MÁXIMO DESCENSO

La ventaja de los métodos de Newton y de los de cuasi-Newton para la solución de sistemas no lineales de ecuaciones es su rapidez de convergencia, una vez que se conoce una aproximación lo suficientemente precisa. La debilidad de estos métodos es que frecuentemente es necesario tener una aproximación inicial precisa para asegurar la convergencia. El método de máximo descenso convergerá a la solución generalmente solo en forma lineal, pero es de naturaleza global. Como resultado de esto, el método del máximo descenso se utiliza principalmente en la solución de sistemas no lineales para encontrar una aproximación inicial suficientemente precisa para ser empleada en las técnicas de Newton, de la misma manera que el método de bisección se usa para una sola ecuación.

El método de máximo descenso determina un mínimo local para una función multivariada de la forma $G: R^n \to R$. La conexión entre la minimización de una función de $R^n \to R$ y la solución de un sistema de ecuaciones no lineales se debe al hecho de un sistema de la forma

$$\begin{aligned} f_1(x_1, x_2, \ldots, x_n) &= 0 \\ f_2(x_1, x_2, \ldots, x_n) &= 0 \\ &\ldots \\ f_n(x_1, x_2, \ldots, x_n) &= 0 \end{aligned}$$

tiene una solución en $x = (x_1, x_2, \ldots, x_n)^t$ precisamente cuando la función G definida como $G(x_1, x_2, \ldots, x_n) = \sum_{i=1}^{n} [f_i(x_1, x_2, \ldots, x_n)]^2$ tiene el valor mínimo 0.

El método de máximo descenso para encontrar un mínimo local de una función $G: R^n \to R$ puede describirse intuitivamente de la siguiente manera:

i) Evaluar G en una aproximación inicial $x^{(0)} = (x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)})^t$.

ii) Encontrar una dirección a partir de $x^{(0)}$ que resulte en un descenso en el valor de G.

iii) Decidir qué tanto debe moverse en esta dirección y llamar a este nuevo valor $x^{(1)}$.

iv) Repetir los pasos i) a iii) reemplazando a $x^{(0)}$ por $x^{(1)}$.

Antes de describir la manera de elegir la dirección correcta y la distancia adecuada que debe moverse en esta dirección, necesitamos revisar algunos resultados de cálculo. El teorema del Valor Extremo implica que una función de una variable que es

diferenciable puede tener un mínimo solamente cuando la derivada es cero. Para extender este resultado a funciones multivariadas necesitamos la siguiente definición:

Definición: Si $G:R^n \to R$, el gradiente de G en $x = (x_1, x_2, \ldots, x_n)^t$ se denota por $\nabla G(x)$ y se define como

$$\nabla G(x) = \left(\frac{\partial G}{\partial x_1}(x), \frac{\partial G}{\partial x_2}(x), \ldots, \frac{\partial G}{\partial x_n}(x) \right)^t .$$

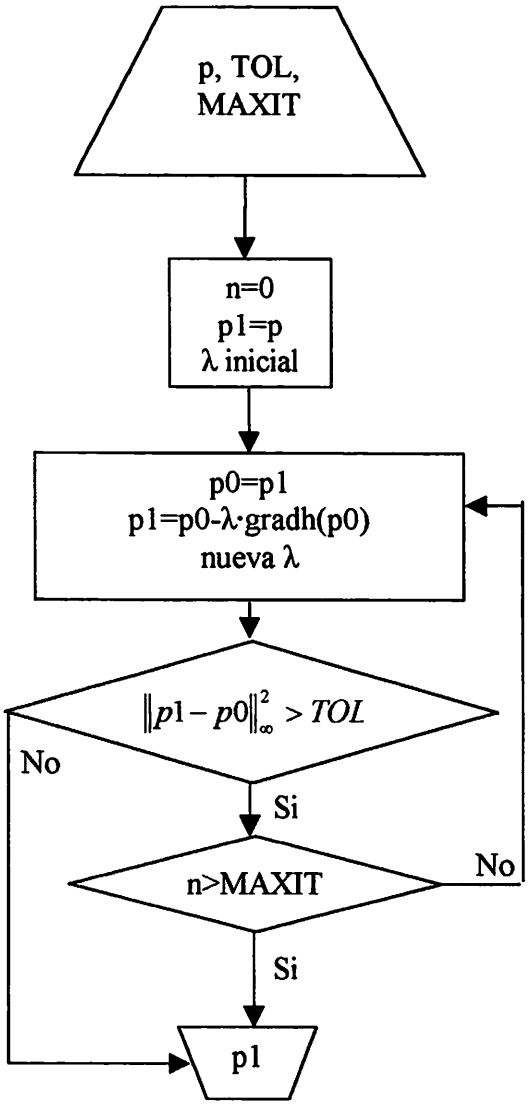
El gradiente de una función multivariada es análogo a la derivada de una función de una sola variable en el sentido de que una función multivariada diferenciable puede tener un mínimo en x solo cuando el gradiente es cero. La dirección en la que el valor de G en x decrece más es aquella dada por $-\nabla G(x)$. Entonces, una elección apropiada para x es $x^{(1)} = x^{(0)} - \alpha \nabla G(x^{(0)})$ para alguna constante $\alpha > 0$.

El problema se reduce ahora a escoger α tal que $G(x^{(1)})$ sea significativamente menor que $G(x^{(0)})$. Para encontrar como elegir el valor de α de una forma apropiada, consideramos la función de una variable $h(\alpha) = G(x^{(0)} - \alpha \nabla G(x^{(0)}))$.

El valor de α que minimiza h es valor que se necesita en la ecuación $x^{(1)} = x^{(0)} - \alpha \nabla G(x^{(0)})$. Para encontrar el mínimo de h directamente se necesitaría derivar h y encontrar los puntos críticos; éste es un procedimiento muy costoso generalmente. En su lugar podemos empezar con tres aproximaciones no negativas $\alpha_1, \alpha_2, \alpha_3$ a α, el número que produce el valor mínimo de h. Después encontramos el polinomio cuadrático que interpola h en $\alpha_1, \alpha_2, \alpha_3$. Se define α como el número que minimiza a este cuadrático. Se usa ese valor para determinar la nueva iteración para aproximar el valor mínimo de G:

$$x^{(1)} = x^{(0)} - \alpha \nabla G(x^{(0)})$$

DIAGRAMA DE FLUJO DEL MÉTODO DE MÁXIMO DESCENSO



En las siguientes tablas se muestra una comparativa del método de Máximo descenso para $\lambda = base^{-L}$, siendo base {1.5, 2, 3}, de las que se puede deducir que la mejor base es 2, pues es la que hace que el método converja más rápido y con un menor número de iteraciones:

MÉTODOS	PUNTOS	SOLUCIONES	TIEMPO (s)	Nº OPs	IT
TOLERANCIA 10^{-15}					
MÁXIMO DESCENSO (λ=2^{-L})	(-1, -1.2)	-1.285218956074 43	-1.113922408601 93	5.330000000000000	235246	457
	(0.2, 1)	1.073727420845 87	0.281001213982 88	7.690000000000000	344614	685
	(0.35, 0.1)	1.073727420316 76	0.281001218195 23	7.200000000000000	327041	667
	(1, 1.1)	0.999999999566 16	1.000000000281 83	4.230000000000000	191701	378
	(-3, 3)	1.137385441951 25	-1.005461317748 89	6.530000000000000	294652	570
MÁXIMO DESCENSO (λ=1.5^{-L})	(-1, -1.2)	NO CONVERGE EN 20000 ITERACIONES				
	(0.2, 1)	1.073727422842 11	0.281001198420 51	5.720000000000000	326735	874
	(0.35, 0.1)	0.486562202665 58	1.308266163413 35	7.250000000000000	510479	1278
	(1, 1.1)	0.999999996508 81	1.000000003106 74	2.970000000000000	211848	570
	(-3, 3)	-1.285218954622 55	-1.113922406130 25	5.060000000000000	353030	949
MÁXIMO DESCENSO (λ=3^{-L})	(-1, -1.2)	-1.285218955455 19	-1.113922407566 41	3.130000000000000	222023	323
	(0.2, 1)	1.073727419558 02	0.281001222928 39	15.440000000000000	109779	1646
	(0.35, 0.1)	1.073727419838 95	0.281001223602 85	15.770000000000000	111605	1677
	(1, 1.1)	0.999999996686 51	1.000000002073 83	5.660000000000000	404066	591
	(-3, 3)	NO CONVERGE EN 20000 ITERACIONES				

MÉTODOS	PUNTOS	SOLUCIONES		TIEMPO (s)	Nº OPs	IT
TOLERANCIA 10^{-8}						
MÁXIMO DESCENSO ($\lambda=2^{-1}$)	(-1, -1.2)	-1.28521980544432	-1.11392378223229	2.36000000000000	162688	343
	(0.2, 1)	1.07372383543626	0.28103013850028	2.80000000000000	200797	440
	(0.35, 0.1)	1.07372141374255	0.28104814829301	2.53000000000000	181648	410
	(1, 1.1)	0.99999770833677	1.00000146895266	2.03000000000000	120550	263
	(-3, 3)	1.13738955252174	-1.00545378571491	2.36000000000000	166872	361
MÁXIMO DESCENSO ($\lambda=1.5^{-1}$)	(-1, -1.2)	NO CONVERGE EN 20000 ITERACIONES				
	(0.2, 1)	1.07371985218627	0.28106082551767	2.42000000000000	174766	512
	(0.35, 0.1)	0.48653065423180	1.30828496990887	3.85000000000000	271205	743
	(1, 1.1)	0.99998422188416	1.00001406547535	1.75000000000000	125673	367
	(-3, 3)	-1.28521683789397	-1.11391875678729	4.17000000000000	288086	810
MÁXIMO DESCENSO ($\lambda=3^{-1}$)	(-1, -1.2)	-1.28521730667415	-1.11391956128727	1.98000000000000	141040	227
	(0.2, 1)	1.07371899717561	0.28106161600680	8.02000000000000	577132	966
	(0.35, 0.1)	1.07372026809089	0.28106401014297	8.35000000000000	594782	993
	(1, 1.1)	0.99997761019413	1.00001565876639	2.91000000000000	209201	342
	(-3, 3)	NO CONVERGE EN 20000 ITERACIONES				

3. COMPARATIVA DE TODOS LOS MÉTODOS ESTUDIADOS

MÉTODOS	PUNTOS	SOLUCIONES (Tolerancia 10^{-8})		TIEMPO (s)	OPERs	IT
MÁXIMO DESCENSO ($\lambda=2^{-b}$)	(-1, -1.2)	-1.28521980544432	-1.11392378223229	2.36000000000000	162688	343
	(0.2, 1)	1.07372383543626	0.28103013850028	2.80000000000000	200797	440
	(0.35, 0.1)	1.07372141374255	0.28104814829301	2.53000000000000	181648	410
	(1, 1.1)	0.99999770833677	1.00000146895266	2.03000000000000	120550	263
	(-3, 3)	1.13738955252174	-1.00545378571491	2.36000000000000	166872	361
NEWTON	(-1, -1.2)	-1.28521895514331	-1.11392240631086 3	5.00000000000426e-002	2351	24
	(0.2, 1)	1.00000159769670	9.99999915551603 4e-001	3.79999999999999e-001	39864	257
BROYDEN	(0.35, 0.1)	NO CONVERGE EN 20000 ITERACIONES				
	(1, 1.1)	1.00000000366257	1.00000000370894 5	6.00000000000227e-002	3156	29
	(-3, 3)	NO CONVERGE EN 20000 ITERACIONES				
NEWTON	(-1, -1.2)	-1.28521895500707 6	-1.11392240679285 0	6.00000000000227e-002	831	6
	(0.2, 1)	1. 00000000000000	1. 00000000000000	5.99999999999517e-002	1311	10
	(0.35, 0.1)	1.13738544252495 2	-1.00546131545977 2	6.00000000000227e-002	1675	13
	(1, 1.1)	1.00000000000000	1. 00000000000000	6.00000000000227e-002	591	4
	(-3, 3)	4.86562187131525 8e-001	1.30826617267352 1	5.99999999999517e-002	1431	11
NEWTON GLOBAL ($\lambda=2^{-b}$)	(-1, -1.2)	-1.28522205494278 0	-1.11392285996224 3	2.69000000000000	213267	432
	(0.2, 1)	1.00000887754613 6	9.99995046242602 6	3.51000000000002	284354	601
	(0.35, 0.1)	1.13739061151456 1	-1.00546155965123 9	4.71999999999999	374196	800
	(1, 1.1)	1.00000073798791 6	1.00000442691244 7	1.82000000000000	147355	289
	(-3, 3)	4.86551027169463 5e-001	1.30826767911535 7	3.62999999999999	289337	631

MÉTODOS	PUNTOS	SOLUCIONES (Tolerancia 10^{-15})	TIEMPO (s)	OPERs	IT
MÁXIMO DESCENSO	(-1, -1.2)	-1.285218955607443	-1.113922240860193	5.330000000000000	235246	457
	(0.2, 1)	1.073727420845870	0.281001213982880	7.690000000000000	344614	685
	(0.35, 0.1)	1.073727420316760	0.281001218195230	7.200000000000000	327041	667
	(1, 1.1)	0.999999999566160	1.000000000028183	4.230000000000000	191701	378
	(-3, 3)	1.137385441951250	-1.005461317748890	6.530000000000000	294652	570
	(-1, -1.2)	-1.285218955007076	-1.113922406792850	0.060000000000000	3800	33
	(0.2, 1)	1.000000000000016	0.999999999999916	1.210000000000001	84461	534
BROYDEN	(0.35, 0.1)	NO CONVERGE EN 20000 ITERACIONES				
	(1, 1.1)	1.000000000000001	1.000000000000008	0.050000000000000	6215	48
	(-3, 3)	NO CONVERGE EN 20000 ITERACIONES				
NEWTON	(-1, -1.2)	-1.285218955526369	-1.113922406868765	1.540000000000001	75398	682
	(0.2, 1)	1.000000005149820	0.9.999999971263023	1.970000000000006	94120	784
	(0.35, 0.1)	1.137385443390853	-1.005461315500677	2.640000000000001	120169	1001
	(1, 1.1)	1.000000000428081	1.000000002567967	1.100000000000000	55953	466
	(-3, 3)	0.486562185262050 5	1.308266172925876	2.200000000000003	102644	855
NEWTON GLOBAL ($\lambda=2^{-1}$)	(-1, -1.2)	-1.285218955526369	-1.113922406868765	1.540000000000001	75398	682
	(0.2, 1)	1.000000005149820	0.9.999999971263023	1.970000000000006	94120	784
	(0.35, 0.1)	1.137385443390853	-1.005461315500677	2.640000000000001	120169	1001
	(1, 1.1)	1.000000000428081	1.000000002567967	1.100000000000000	55953	466
	(-3, 3)	0.486562185262050 5	1.308266172925876	2.200000000000003	102644	855

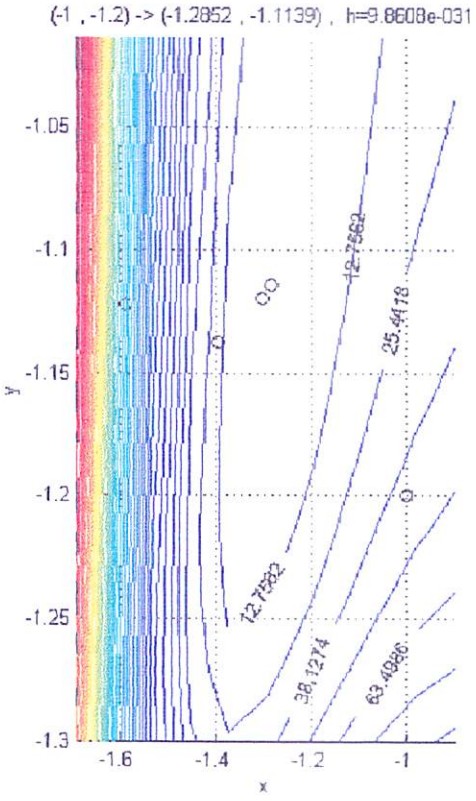
4. SOLUCIONES OBTENIDAS EN LOS DISTINTOS MÉTODOS

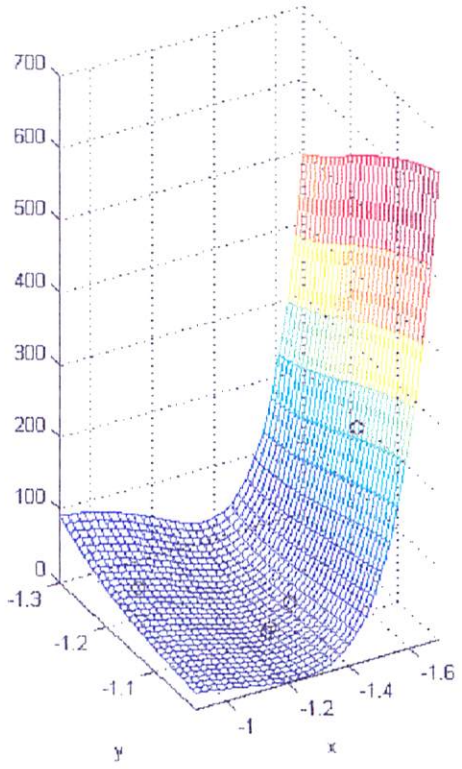
SOLUCIONES	
X	Y
1.0000000000000000+000	*1.0000000000000000+000*
-1.2852189550070761e+000	*-1.1139224067928502e+000*
1.0737274219282942e+000	*2.8100120563944564e-001*
1.1373854425249521e+000	*-1.0054613154597716e+000*
4.8656218713152560e-001	*1.3082661726735214e+000*

ANEXO I- GRÁFICAS DE LA DINÁMICA DE LOS MÉTODOS -

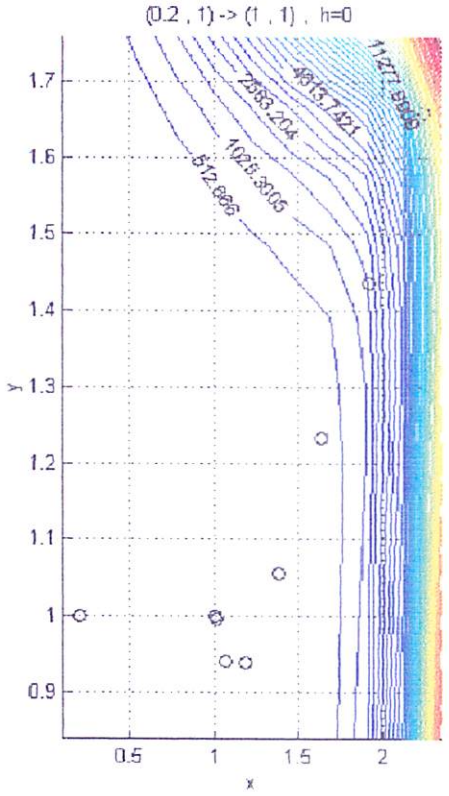
NEWTON

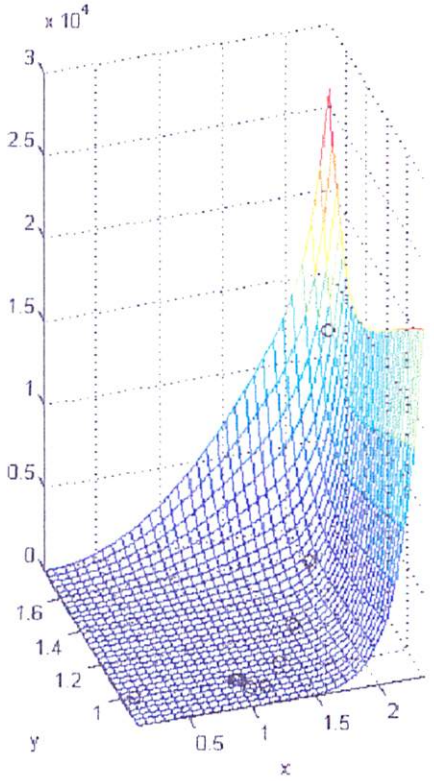
- [-1, -1.2]



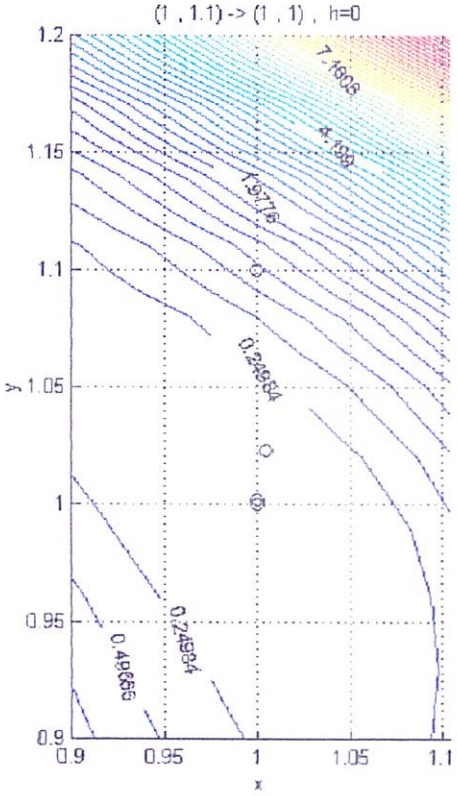


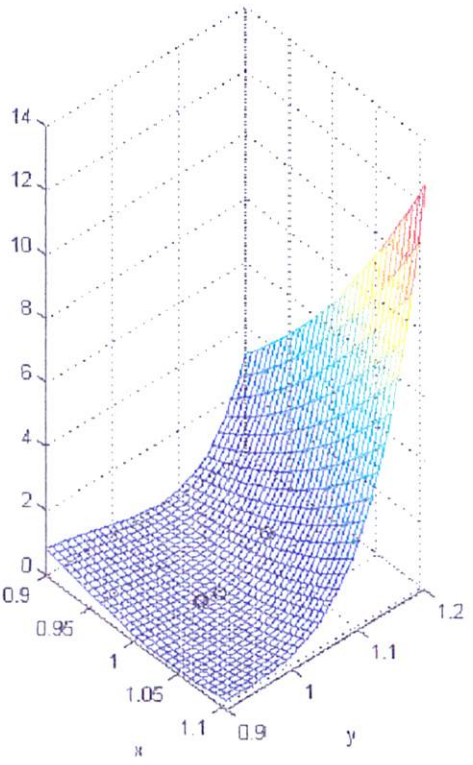
- [0.2, 1]


(0.2 , 1) -> (1 , 1) , h=0
11277.9999
4913.7421
2563.204
1025.3005
512.666
x
y

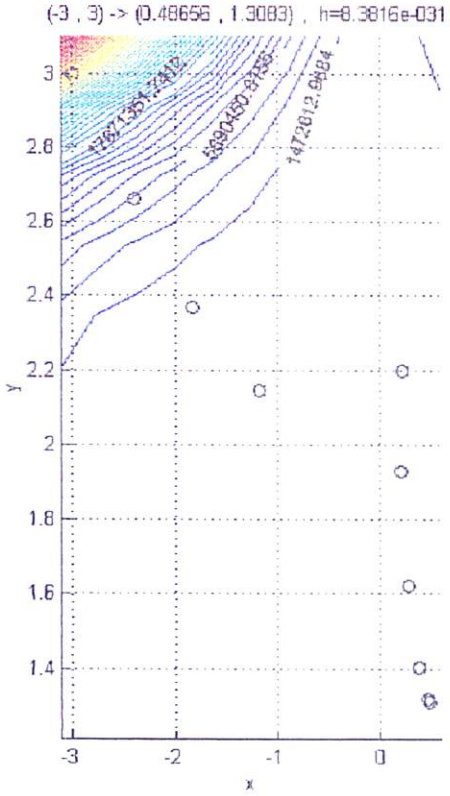

x 10^4
x
y

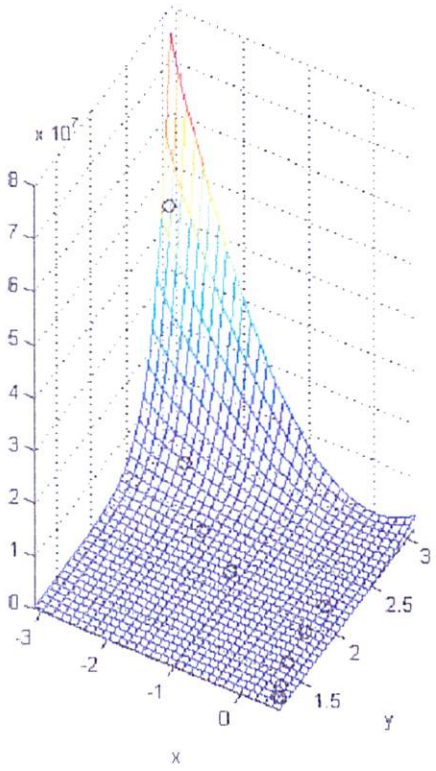
- [1, 1.1]


(1 , 1.1) -> (1 , 1) , h=0
7.1602
4.1582
1.1562
0.24984
0.24984
0.49666
x
y


x
y

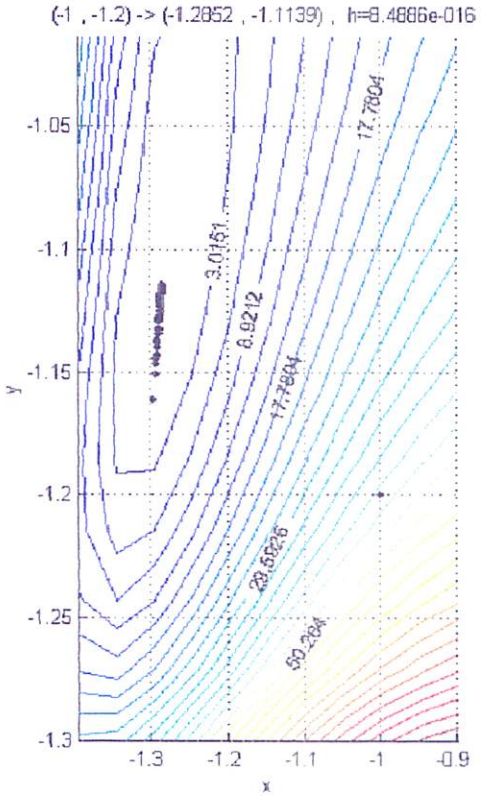
- [-3, 3]

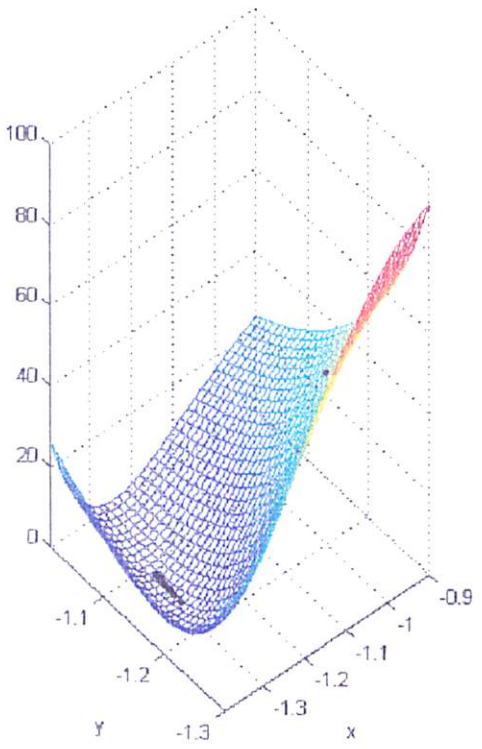




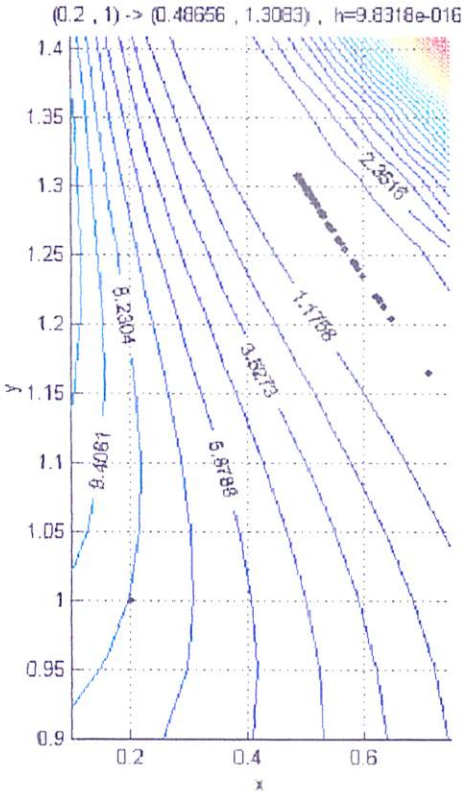
NEWTON-GLOBAL($\lambda=2^{-L}$)

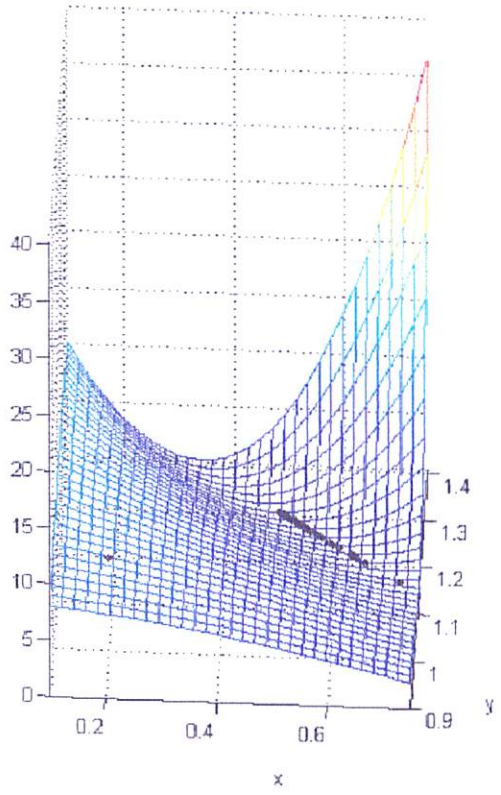
- [-1, -1.2]



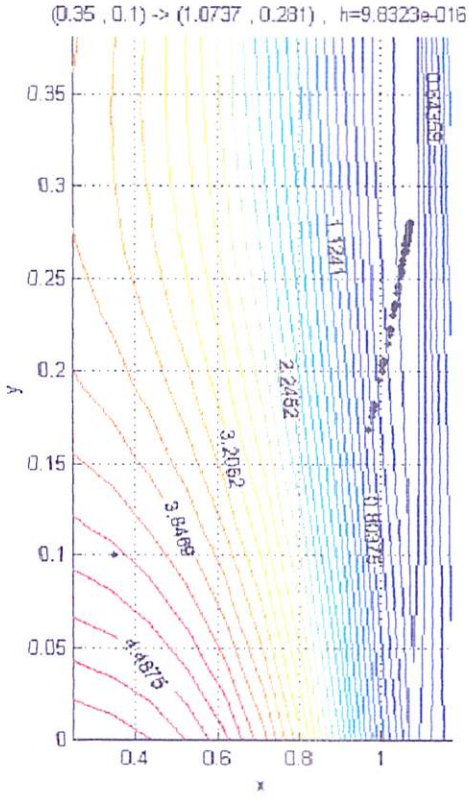


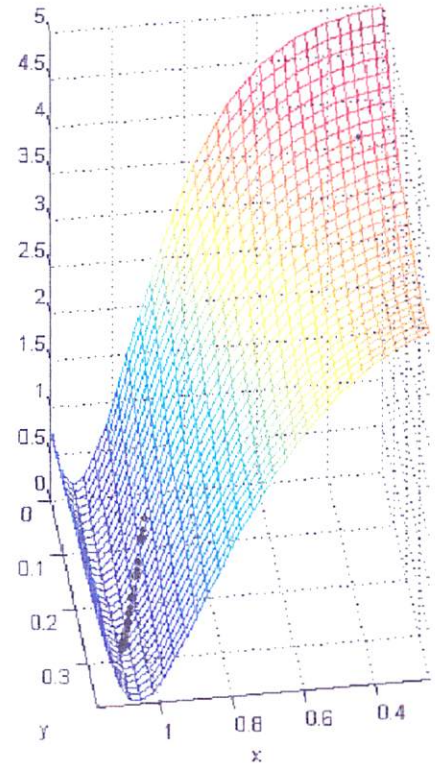
- [0.2, 1]


(0.2 , 1) -> (0.48656 , 1.3083) , h=9.8318e-016
2.2516
1.1768
0.5873
6.8788
8.4061
8.2304

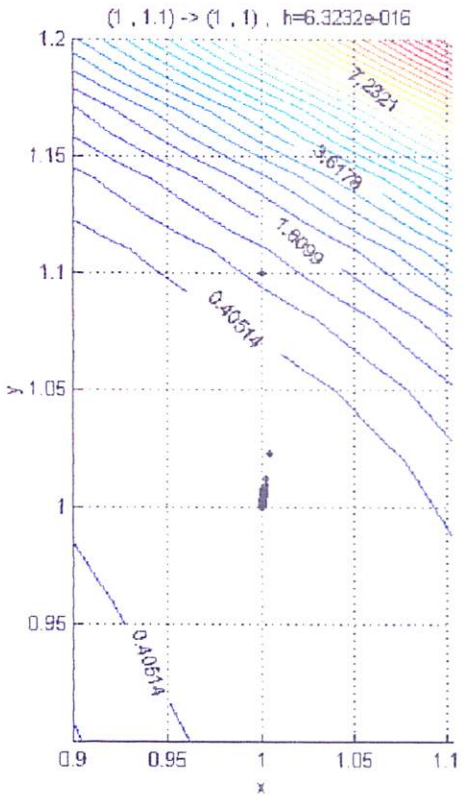


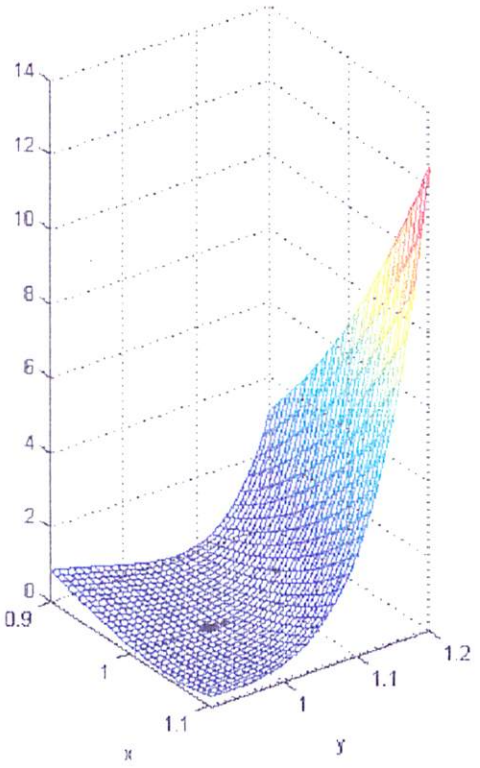
- [0.35, 0.1]


(0.35 , 0.1) -> (1.0737 , 0.281) , h=9.8323e-016
1.1241
2.2452
3.2062
3.8469
4.4875

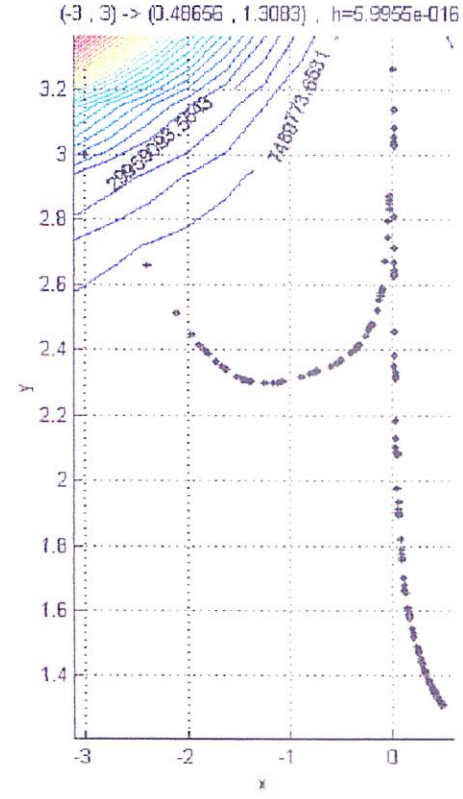


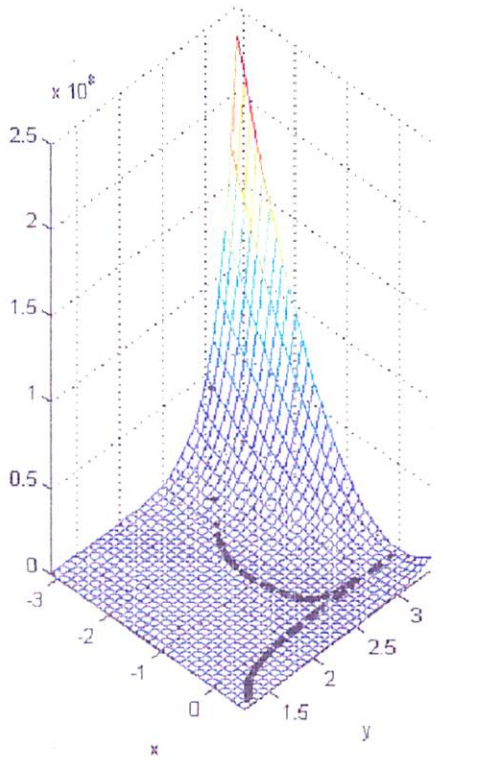
- [1, 1.1]


(1 , 1.1) -> (1 , 1) , h=6.3232e-016
7.2321
3.6179
1.8099
0.40514
0.40514
1.2
1.15
1.1
1.05
1
0.95
y
0.9
0.95
1
1.05
1.1
x


14
12
10
8
6
4
2
0
0.9
1
1.1
x
1
1.1
1.2
y

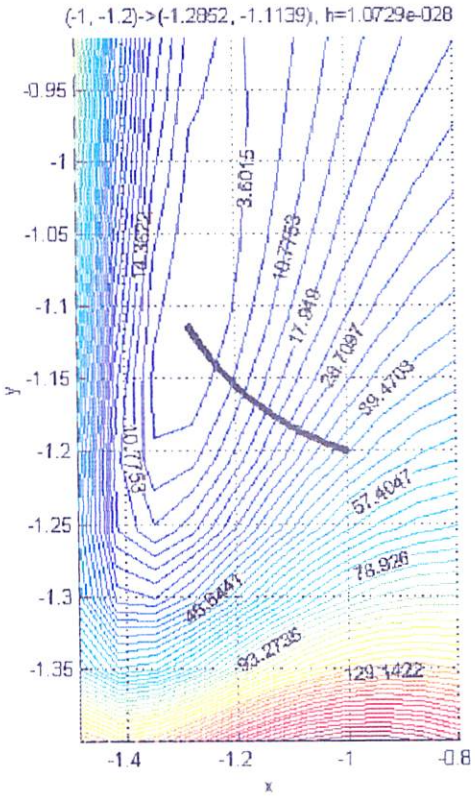
- [-3, 3]

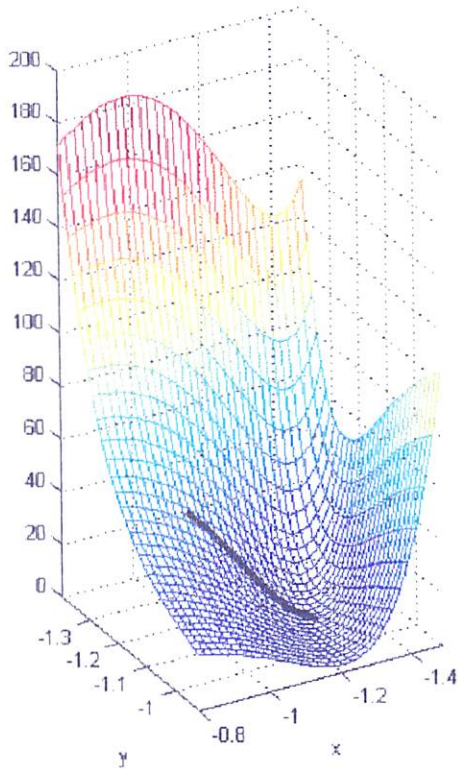

(-3 , 3) -> (0.48656 , 1.3083) , h=5.9955e-016
3.2
3
2.8
2.6
2.4
2.2
2
1.8
1.6
1.4
y
-3
-2
-1
0
x


x 10^8
2.5
2
1.5
1
0.5
0
-3
-2
-1
0
x
1.5
2
2.5
3
y

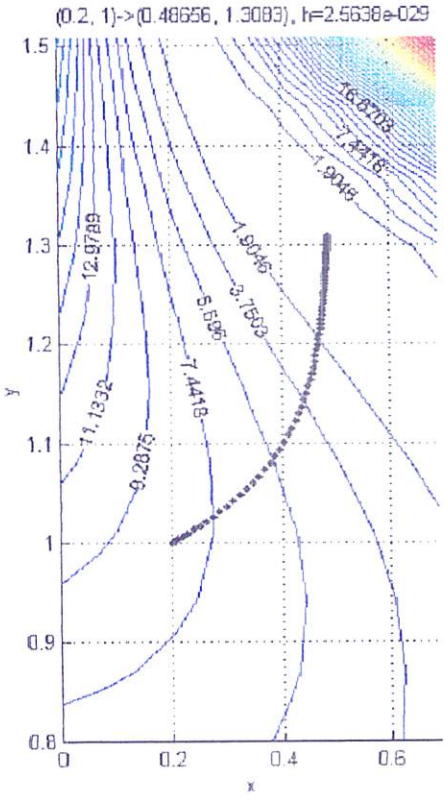
NEWTON-GLOBAL (λ=L)

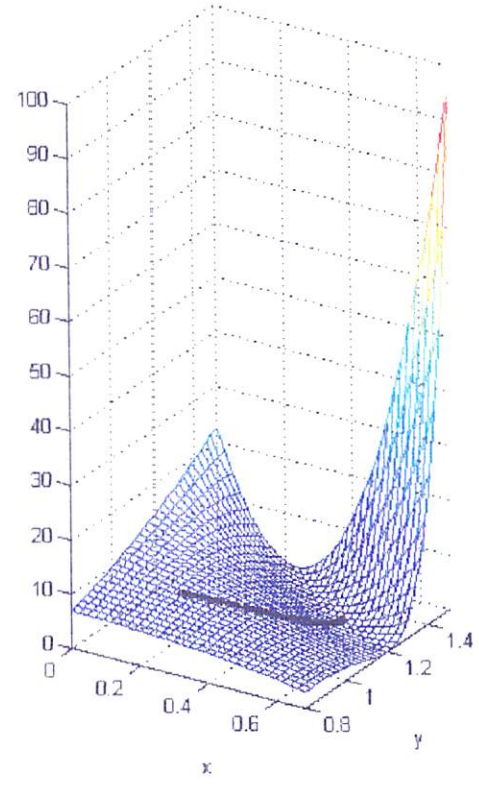
- [-1, -1.2]



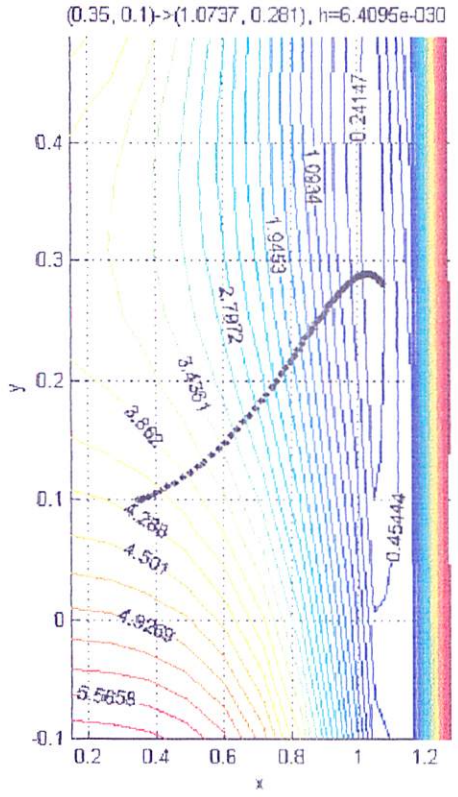


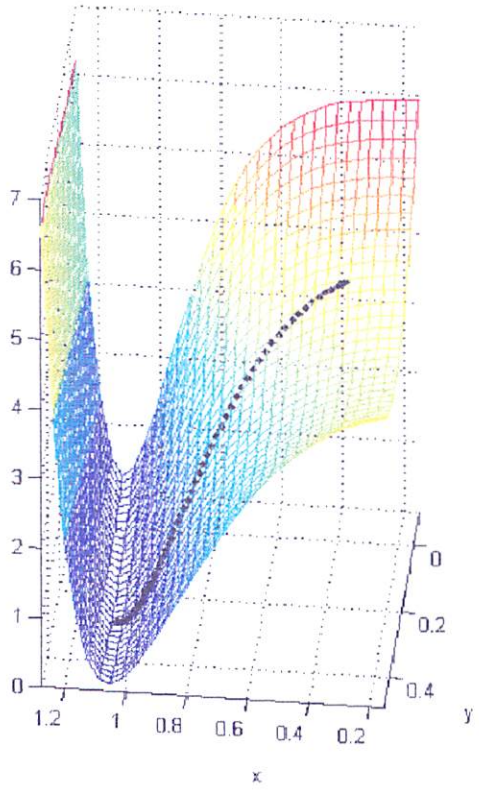
- [0.2, 1]



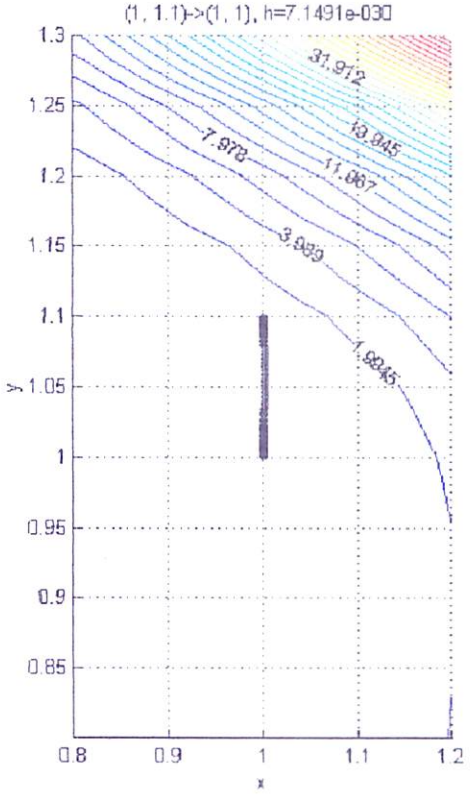


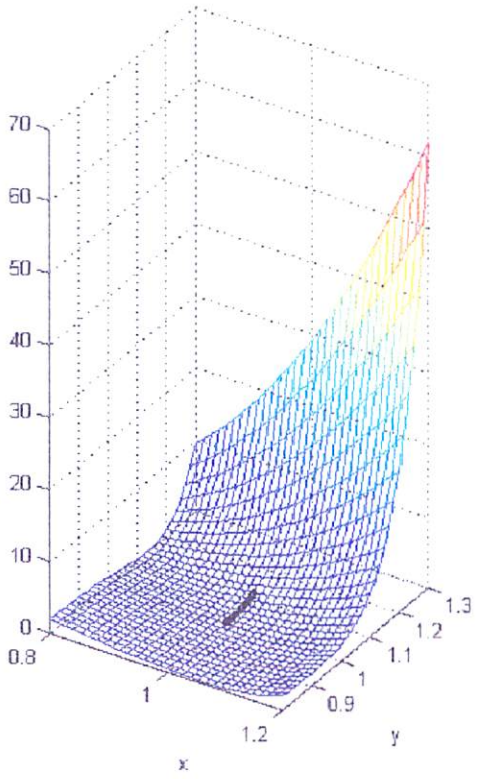
- [0.35, 0.1]


(0.35, 0.1)->(1.0737, 0.281), h=6.4095e-030
0.24147
1.0934
1.9453
2.7872
3.4361
3.862
4.2283
4.501
4.9269
5.5658
0.45444
x
y


x
y

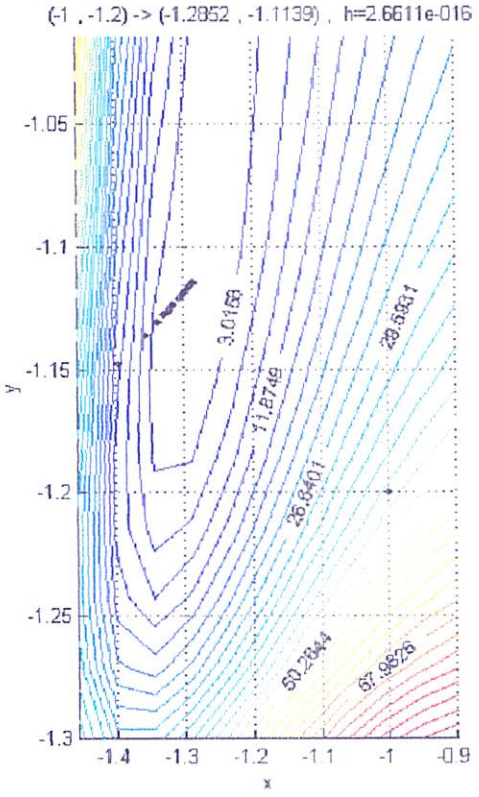
- [1, 1.1]

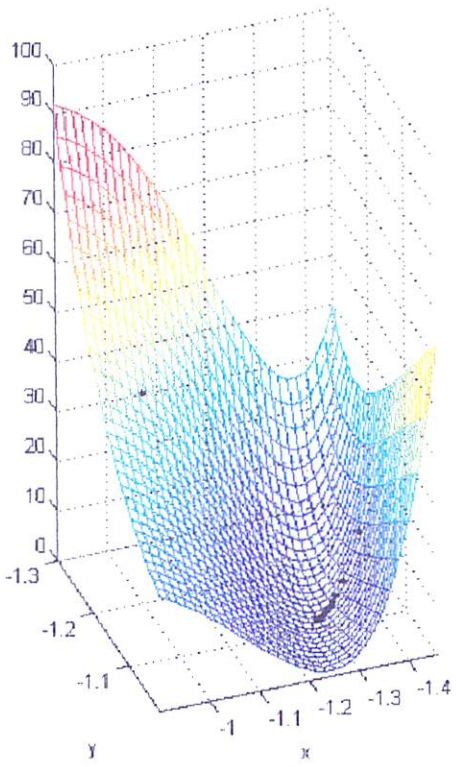

(1, 1.1)->(1, 1), h=7.1491e-030
31.912
19.945
11.067
7.978
3.989
1.9945
x
y


x
y

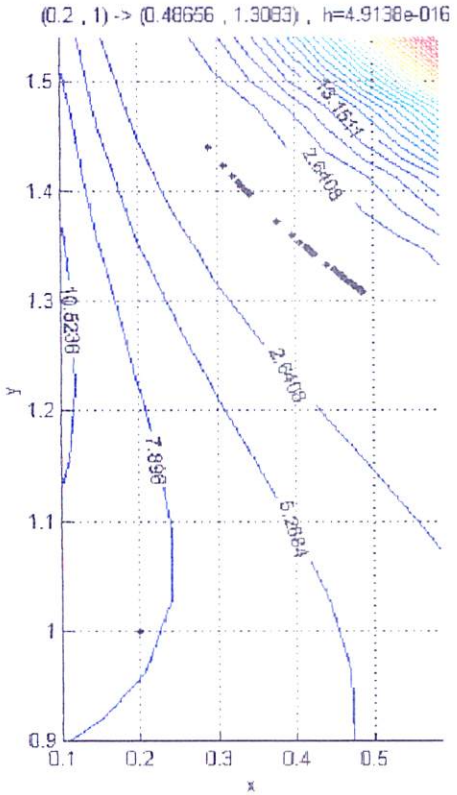
NEWTON-GLOBAL ($\lambda=1.5^{-L}$)

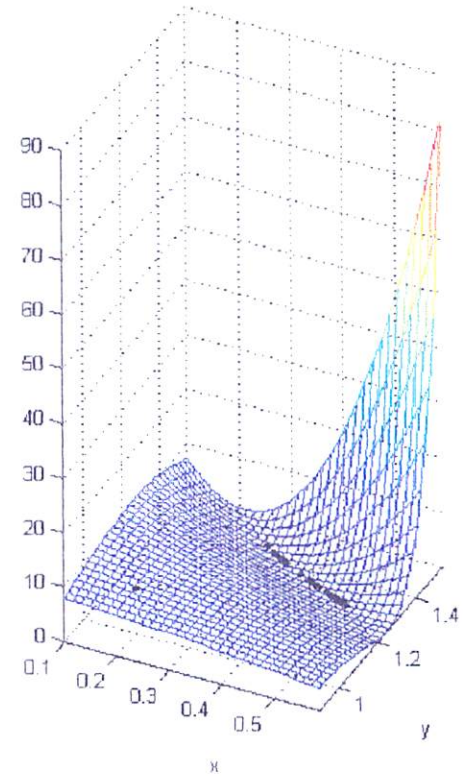
- [-1, -1.2]



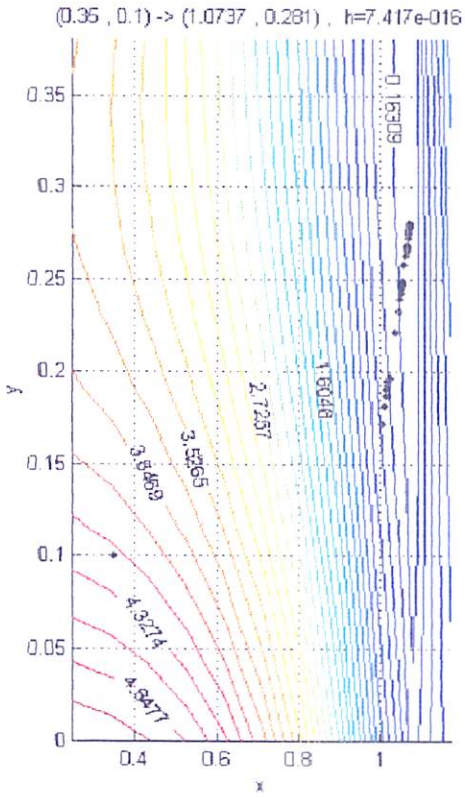


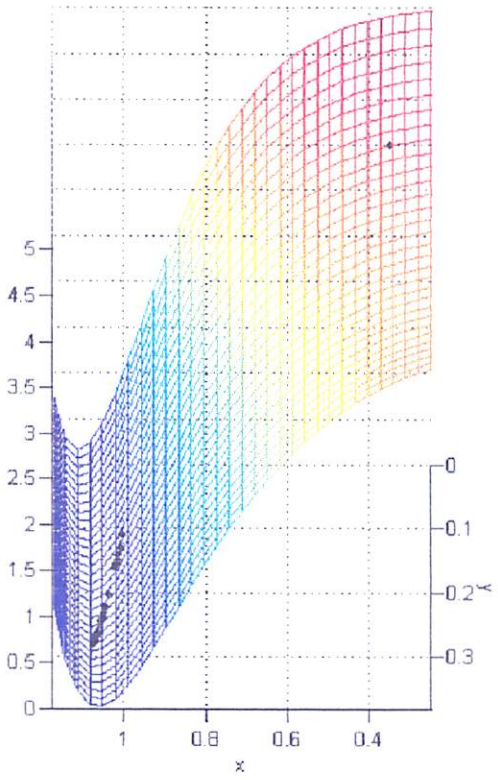
- [0.2, 1]



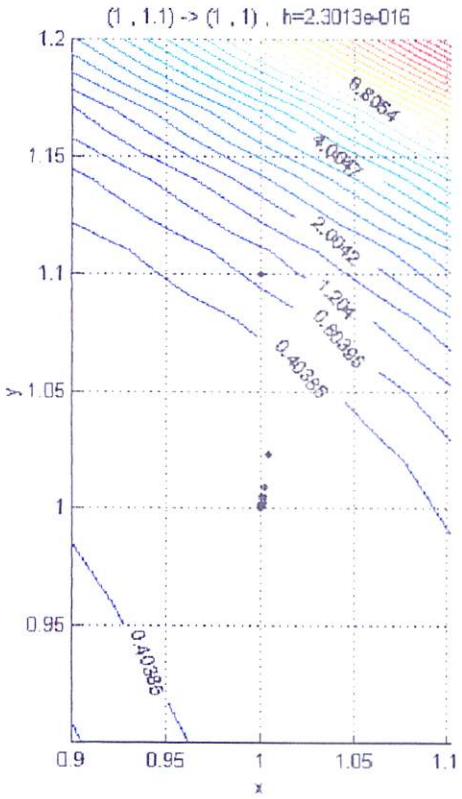


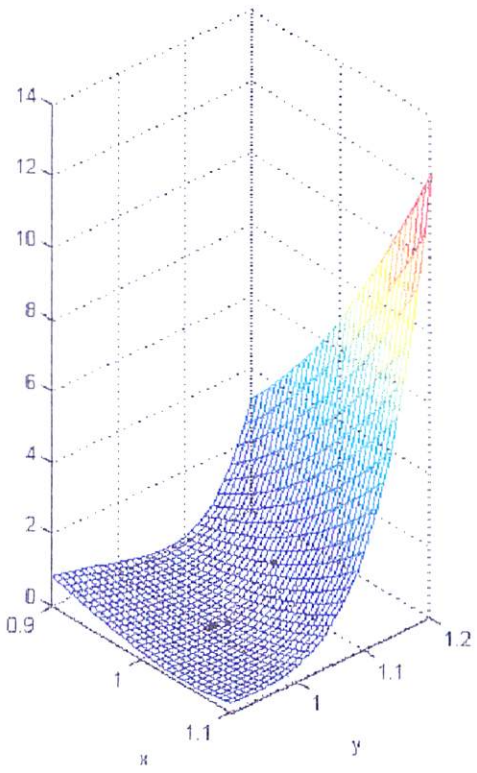
- [0.35, 0.1]


(0.35 , 0.1) -> (1.0737 , 0.281) , h=7.417e-016
0.16509
1.6048
2.7257
3.5265
3.9469
4.3214
4.9477
y
x


x
y

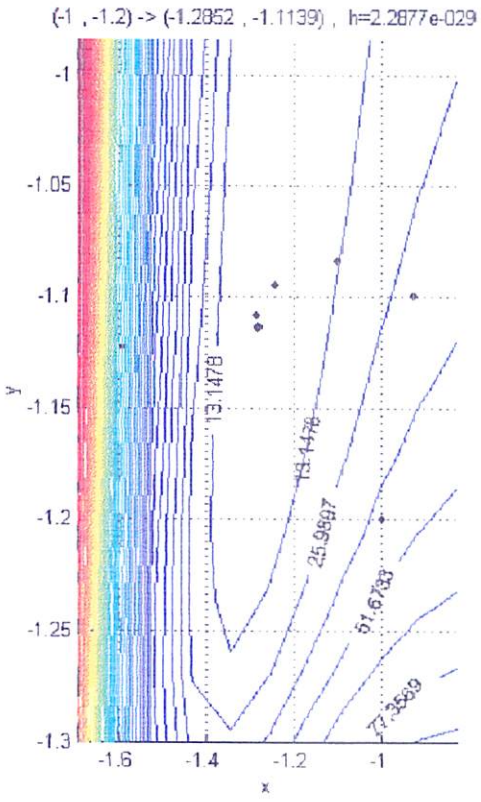
- [1, 1.1]

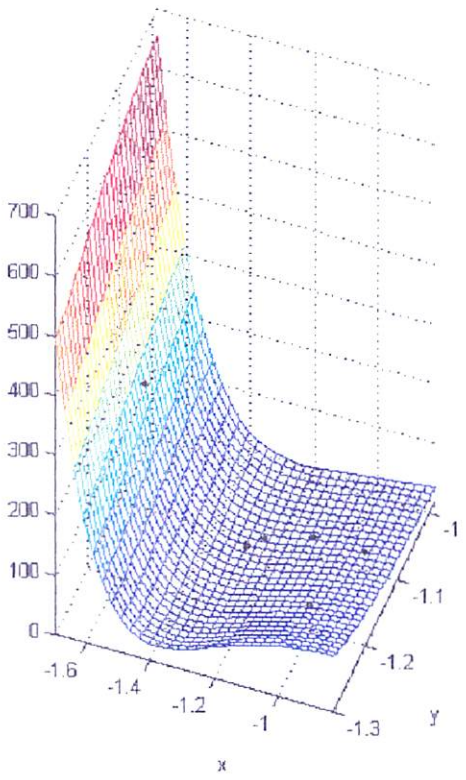

(1 , 1.1) -> (1 , 1) , h=2.3013e-016
6.8054
4.0847
2.0342
1.204
0.80395
0.40395
0.40395
y
x


x
y

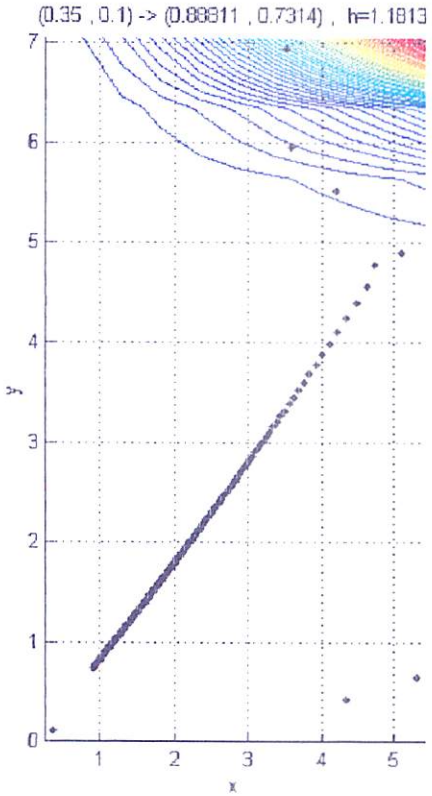
BROYDEN

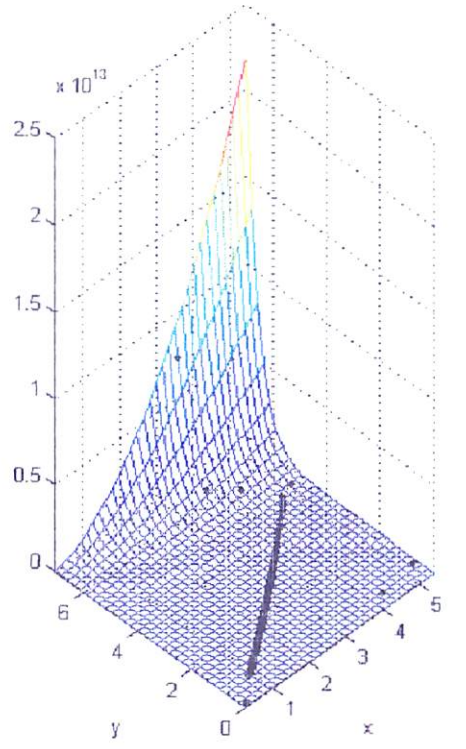
- [-1, -1.2]



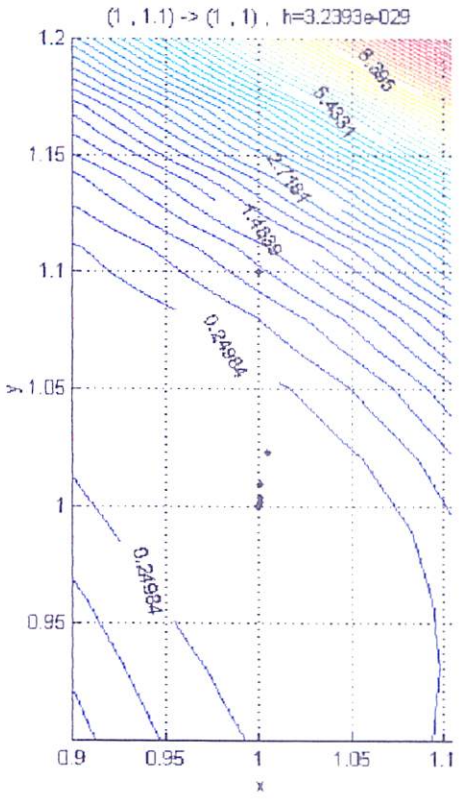


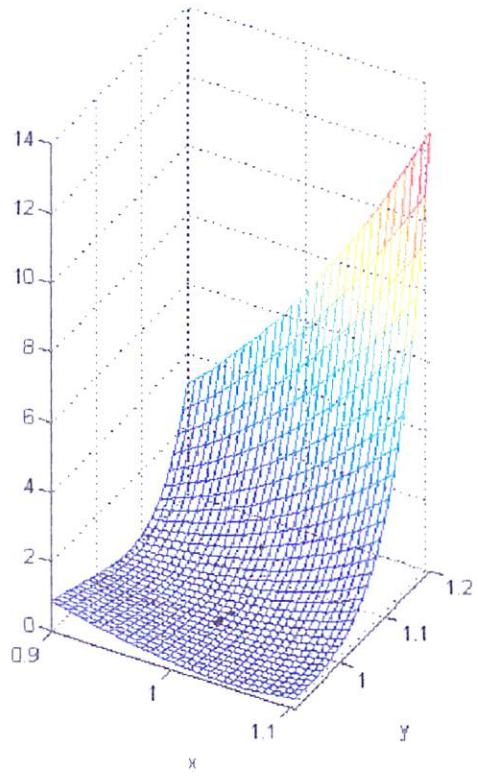
- [0.35, 0.1]



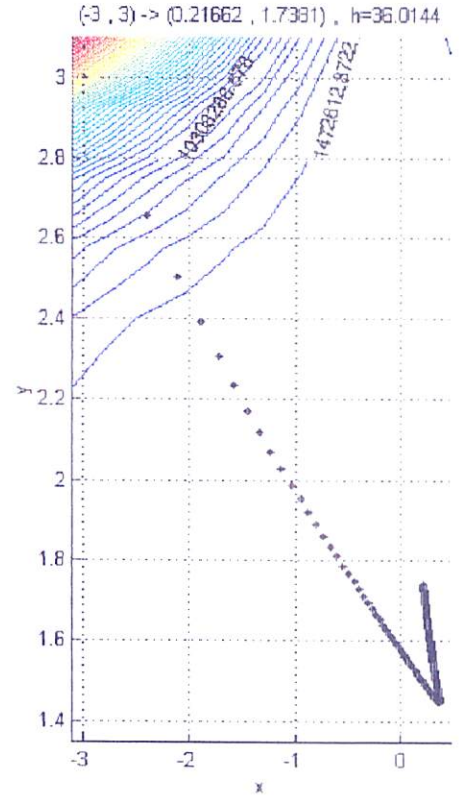


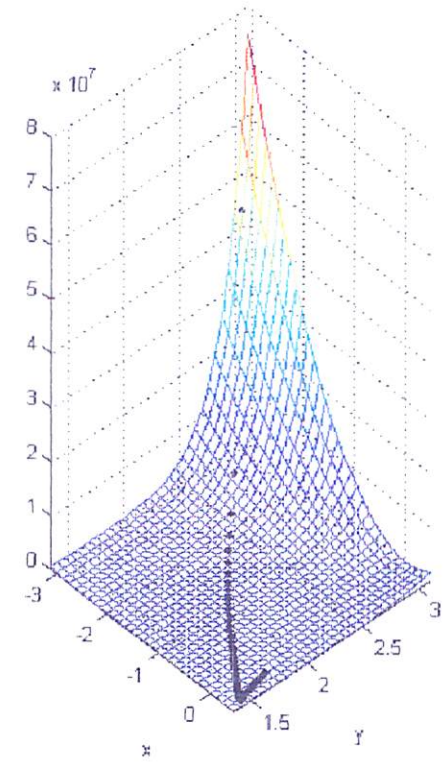
- [1, 1.1]


(1 , 1.1) -> (1 , 1) , h=3.2393e-029
8.365
5.4531
2.7191
1.4529
0.24984
0.24984
y
x


x
y

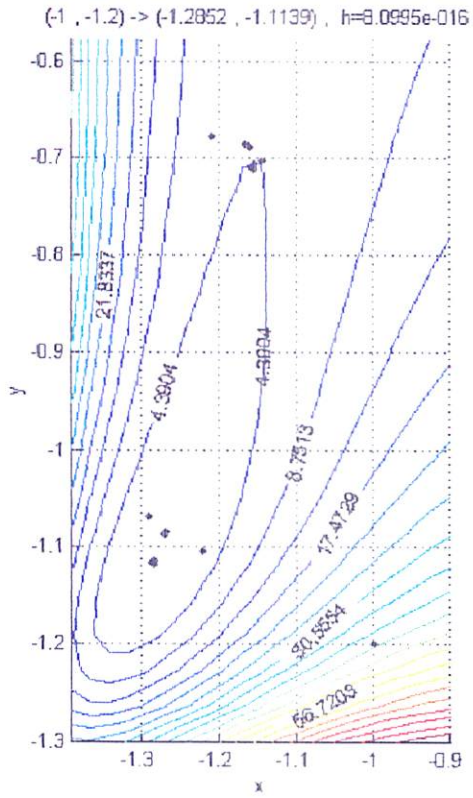
- [-3, 3]

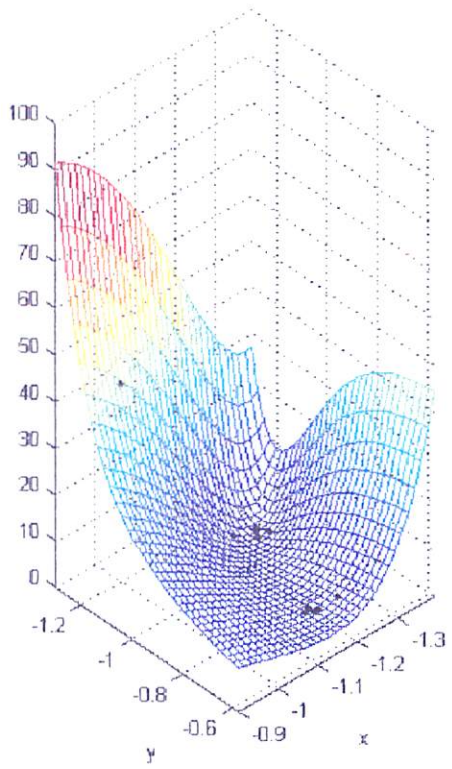

(-3 , 3) -> (0.21662 , 1.7381) , h=36.0144
14728612.8722
y
x


x 10^7
x
y

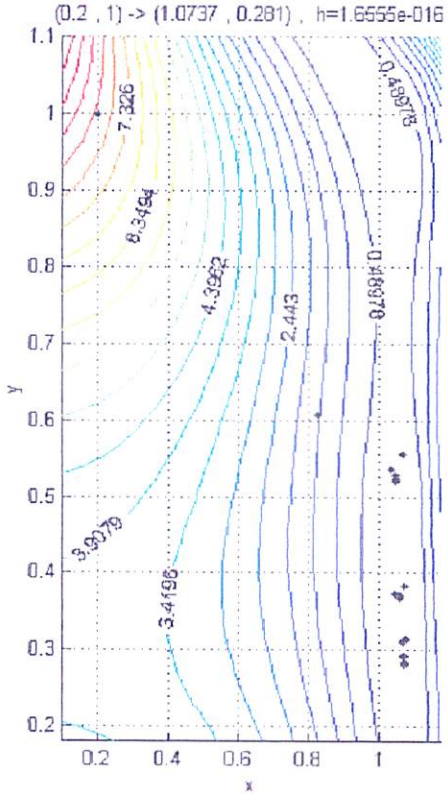
MÁXIMO DESCENSO ($\lambda=2^{-L}$)

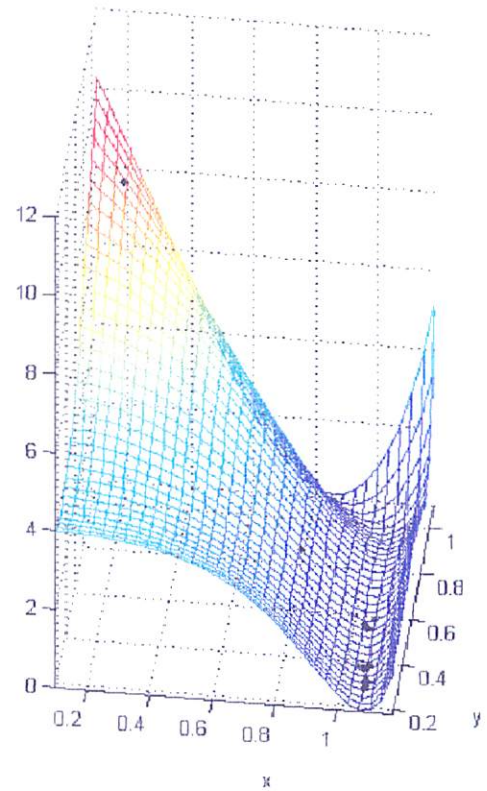
- [-1, -1.2]



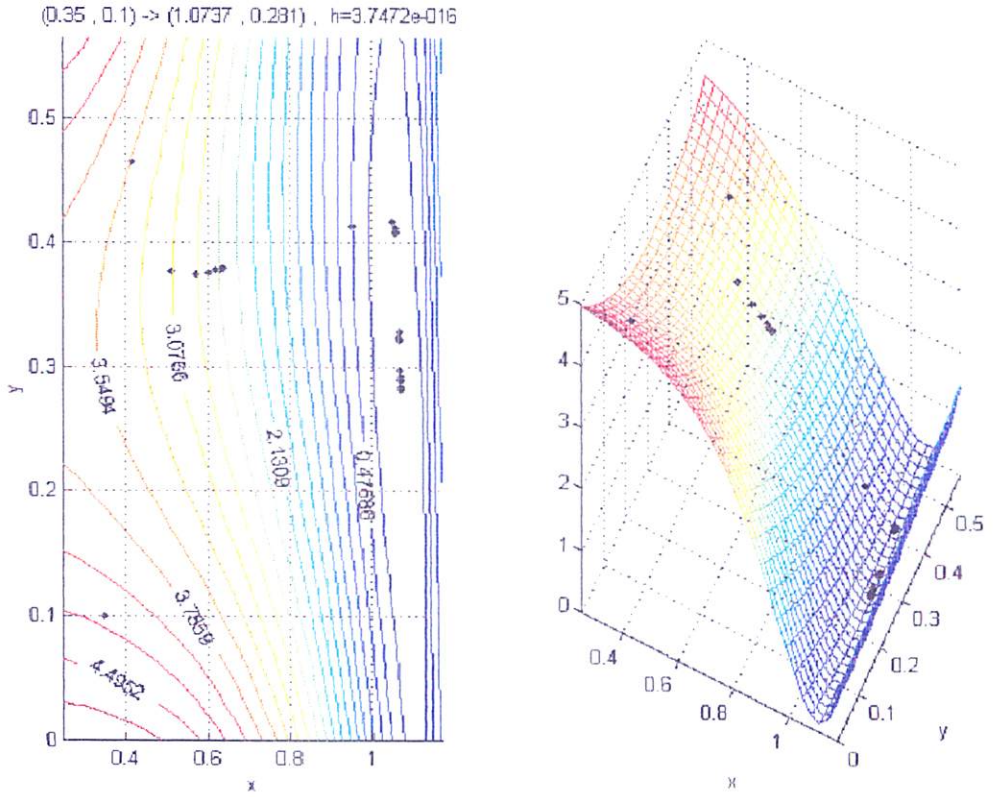


- [0.2, 0.1]

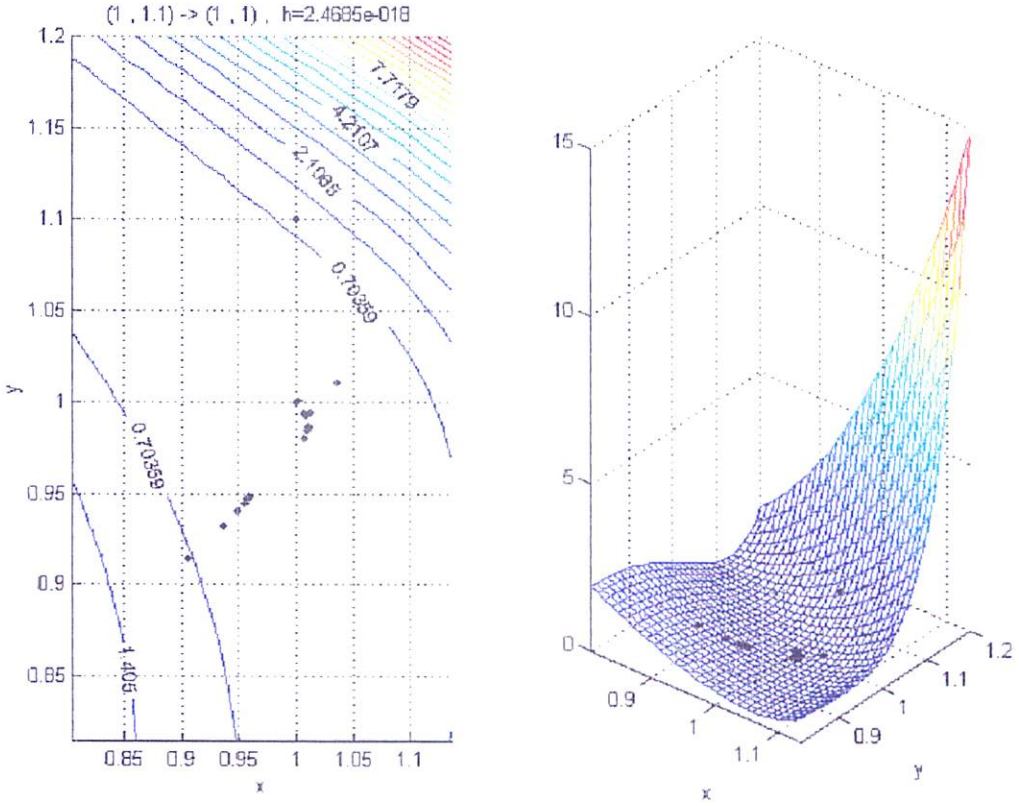




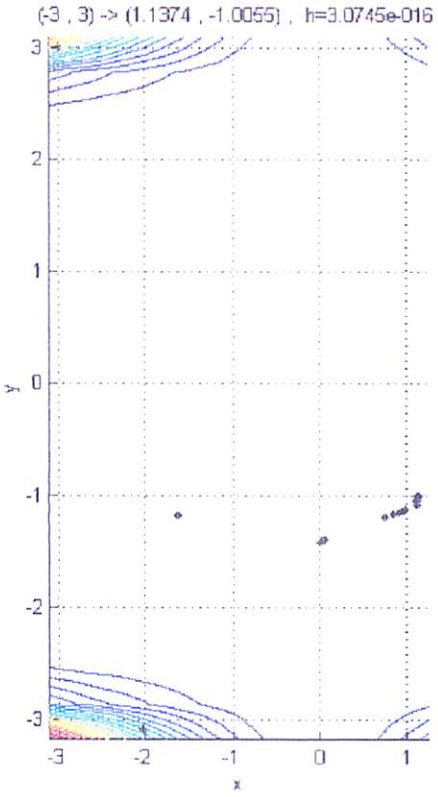
- [0.35, 0.1]

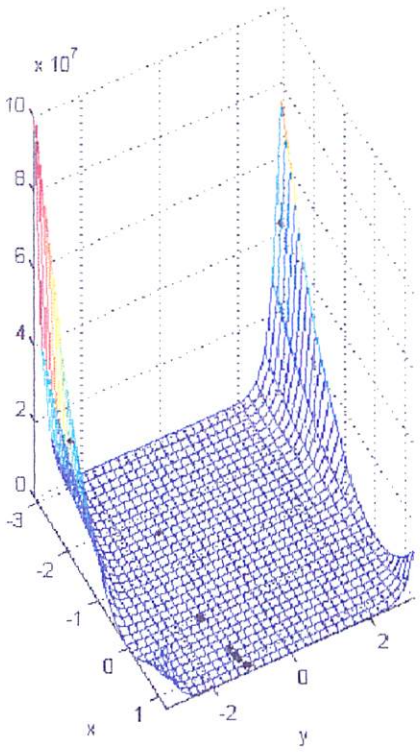

(0.35 , 0.1) -> (1.0737 , 0.281) , h=3.7472e-016
3.5494
3.0766
2.1309
0.47686
3.7859
4.4952
x
y

- [1, 1.1]


(1 , 1.1) -> (1 , 1) , h=2.4685e-018
1.7119
4.2103
2.1685
0.70359
0.70359
1.405
x
y

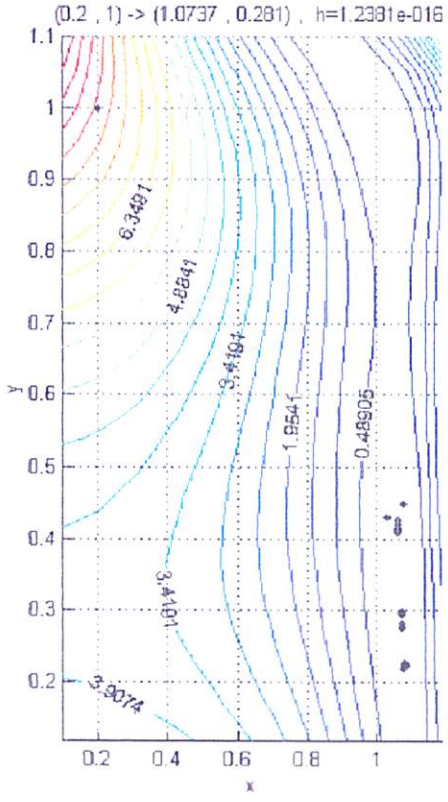
- [-3, 3]

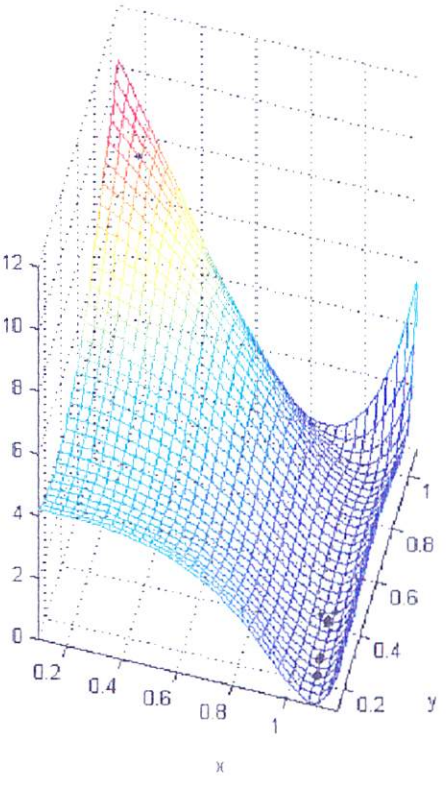




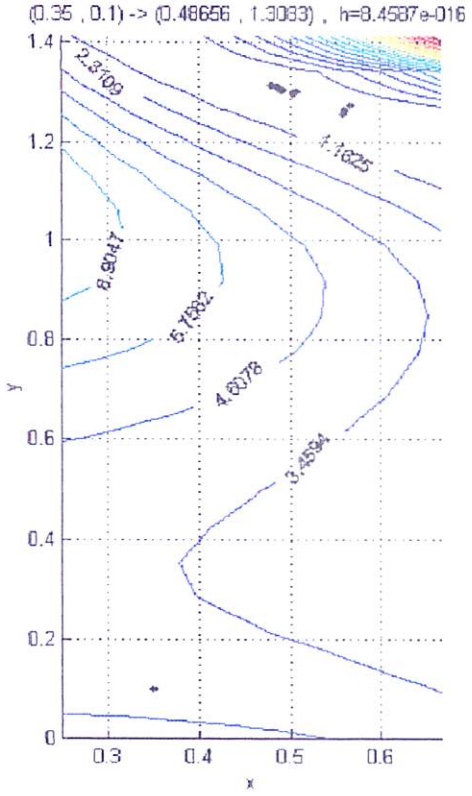
MÁXIMO DESCENSO ($\lambda=1.5^{-L}$)

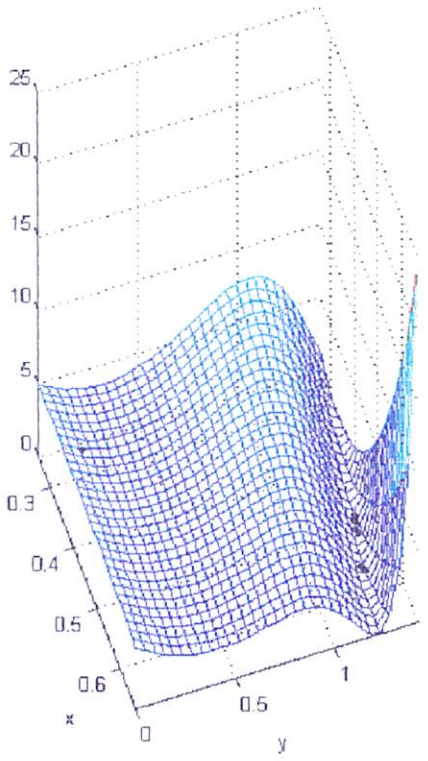
- [0.2 1]



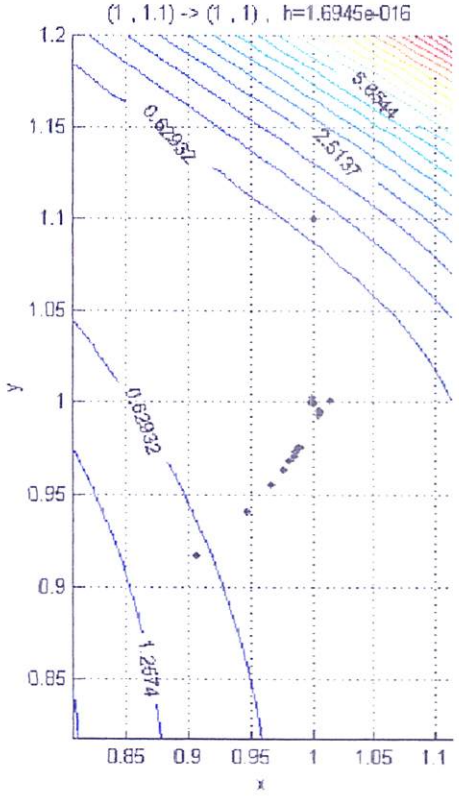


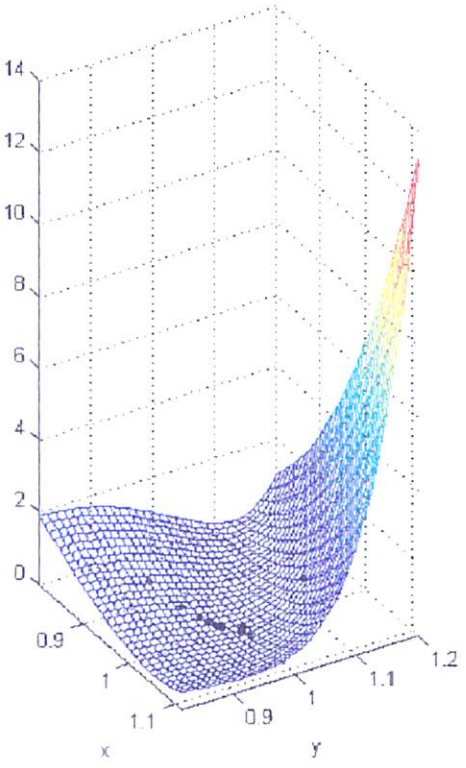
- [0.35, 0.1]


(0.35 , 0.1) -> (0.48656 , 1.3033) , h=8.4587e-016
2.3109
4.1625
6.9047
5.7592
4.6078
3.4564
x
y


x
y

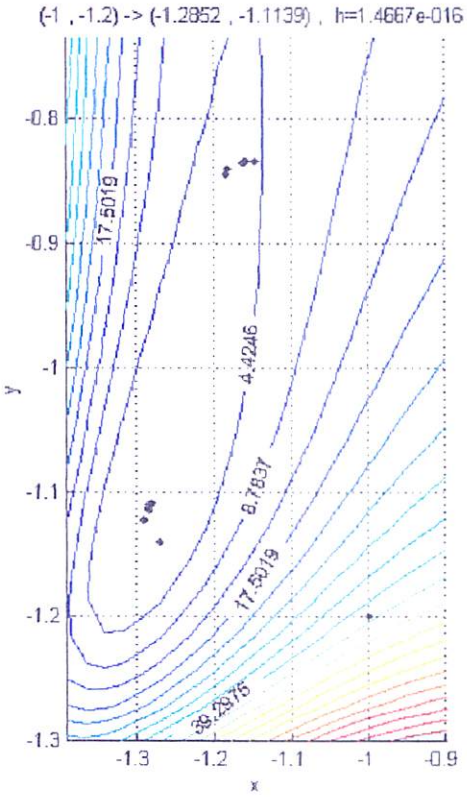
- [1, 1.1]

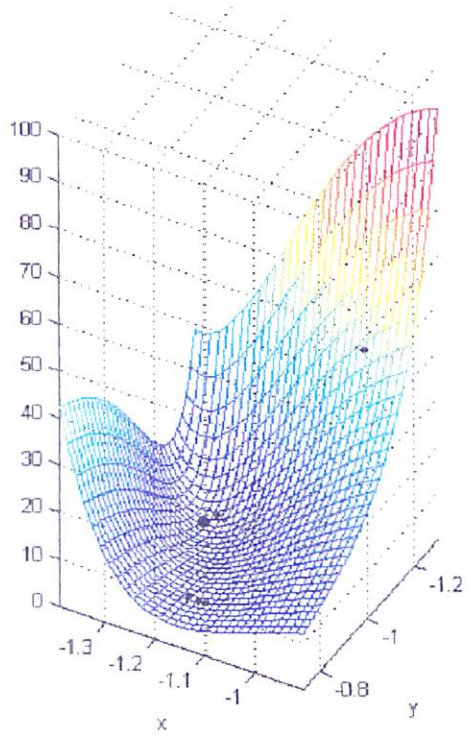

(1 , 1.1) -> (1 , 1) , h=1.6945e-016
5.2544
0.62952
2.5137
0.62932
1.2574
x
y


x
y

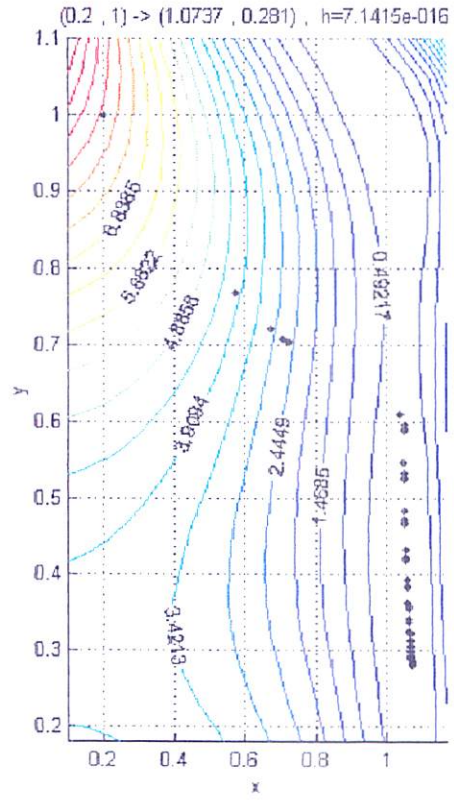
MÁXIMO DESCENSO ($\lambda=3^{-L}$)

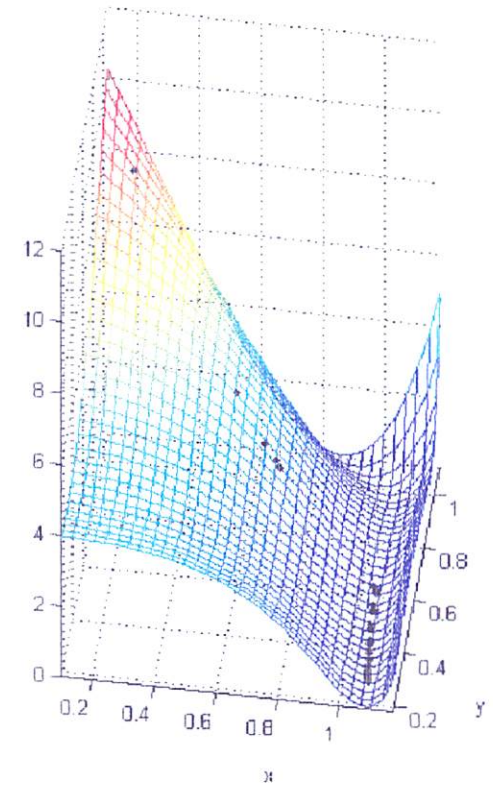
- [-1, -1.2]



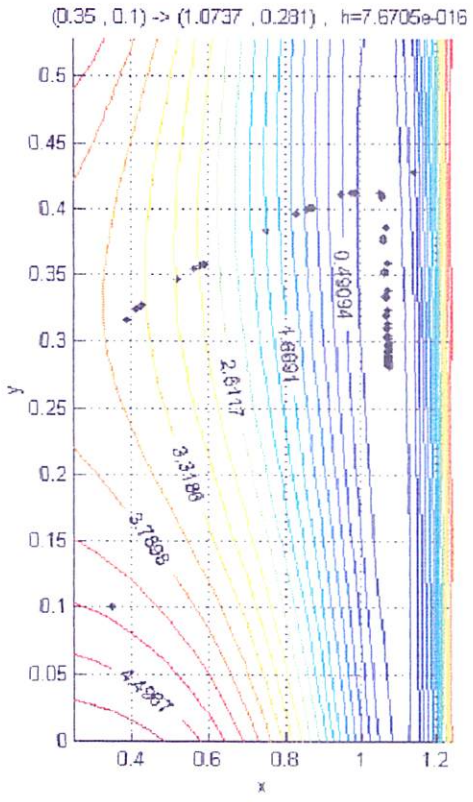


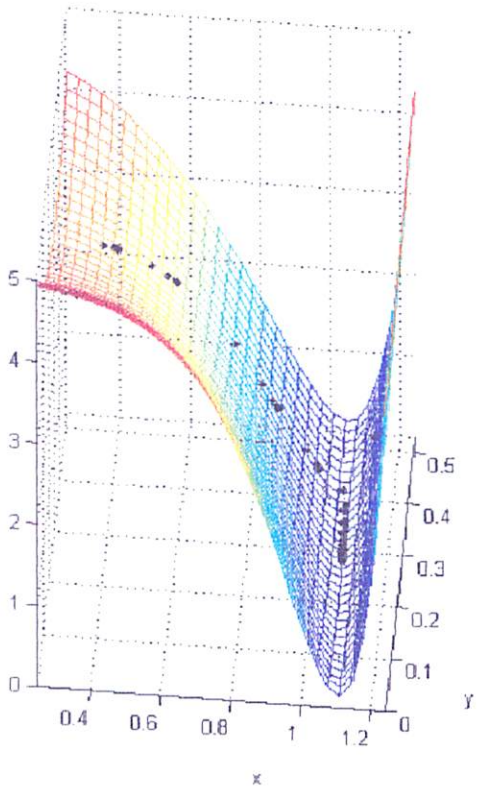
- [0.2, 1]



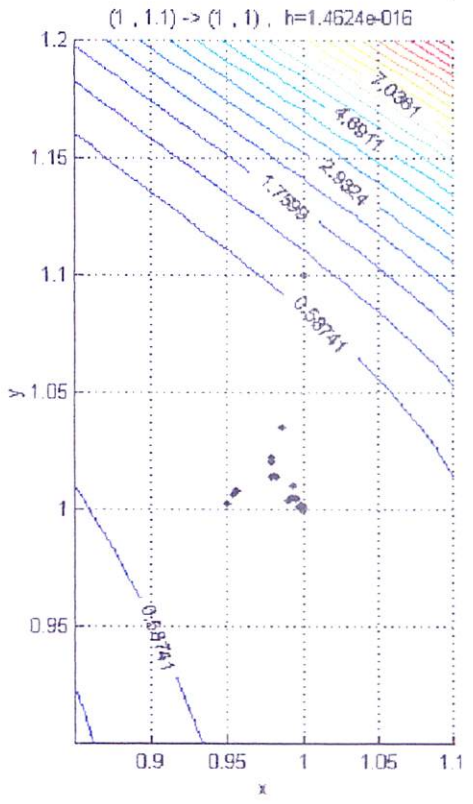


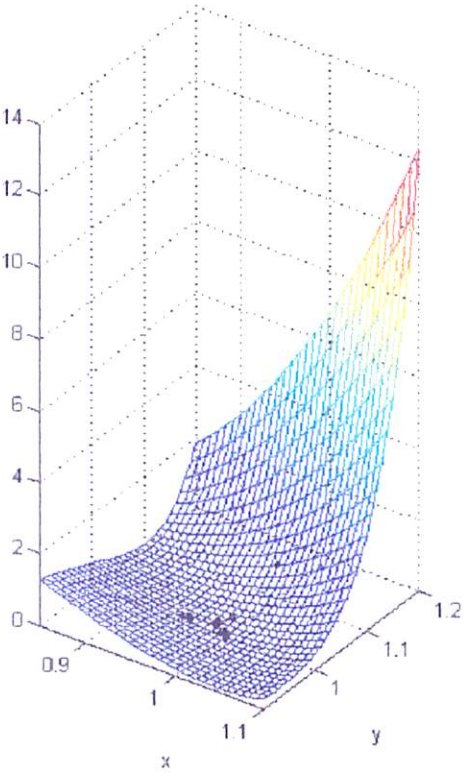
- [0.35, 0.1]


(0.35 , 0.1) -> (1.0737 , 0.281) , h=7.6705e-016
0.49094
1.6691
2.6117
3.3186
3.7868
4.4961
x
y

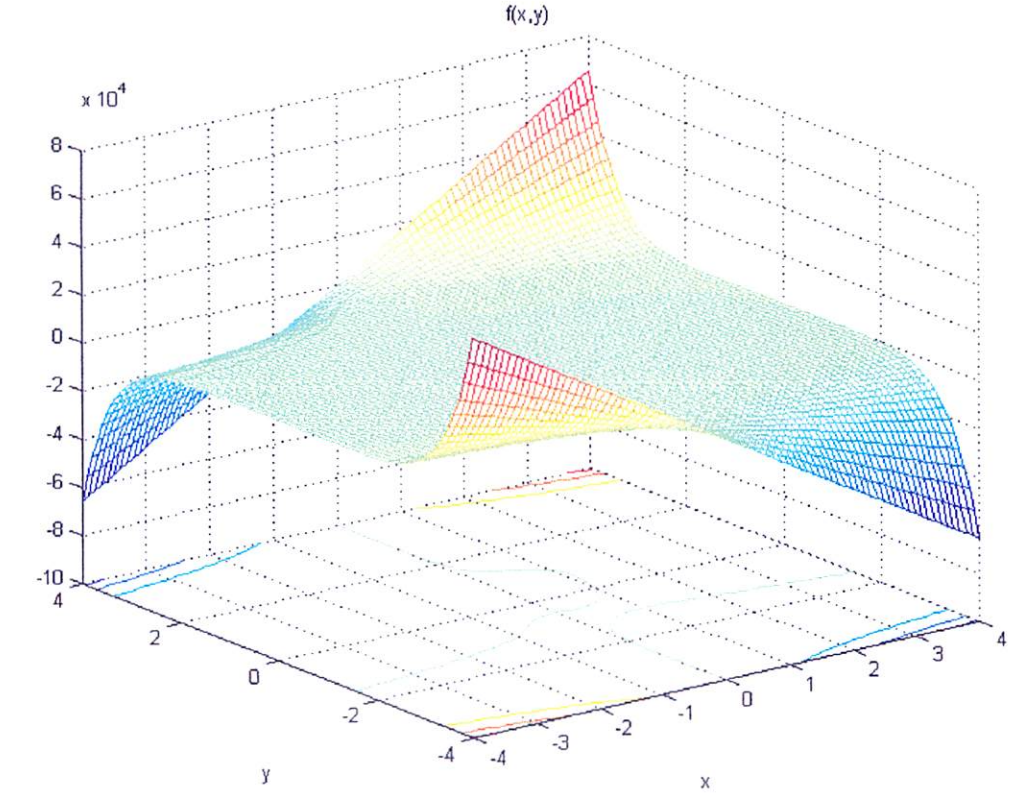

x
y

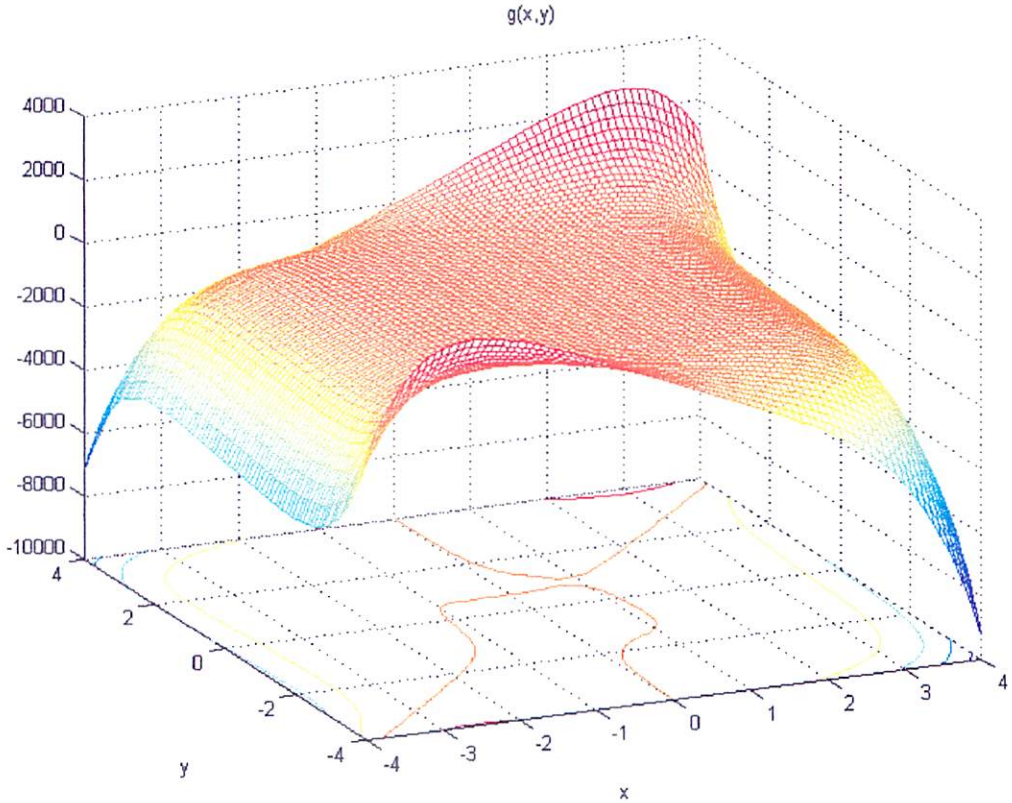
- [1, 1.1]

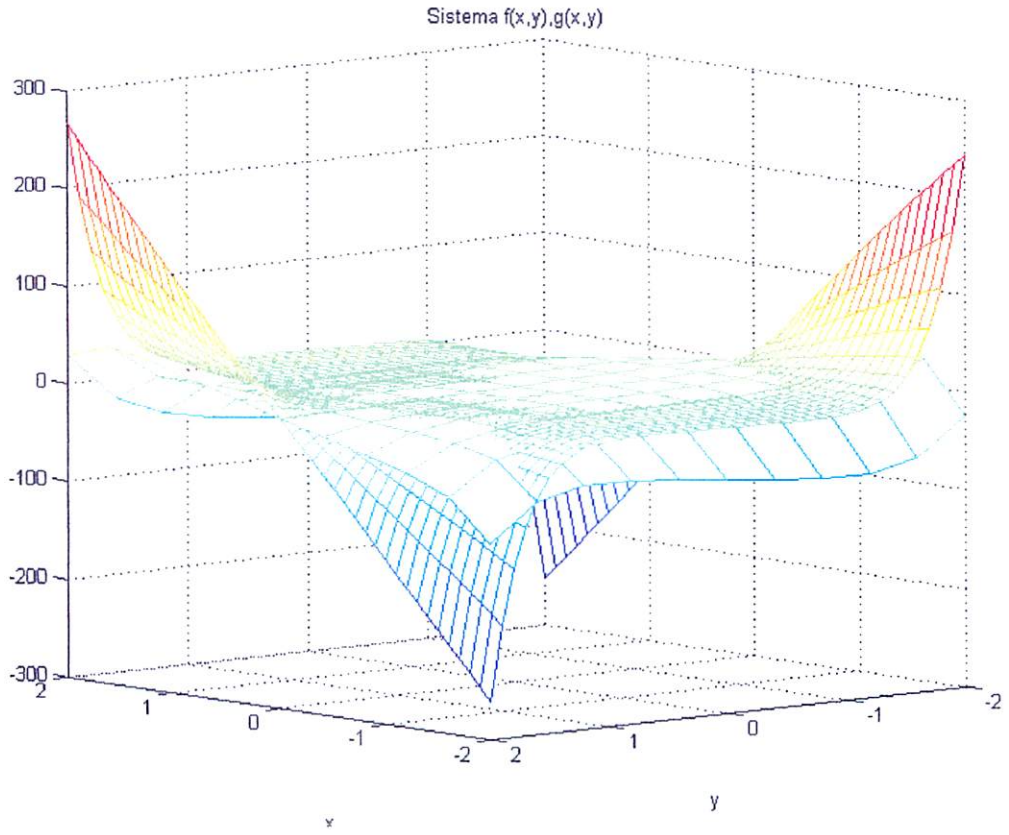

(1 , 1.1) -> (1 , 1) , h=1.4624e-016
7.0381
4.8871
2.9324
1.7593
0.58741
0.58741
x
y

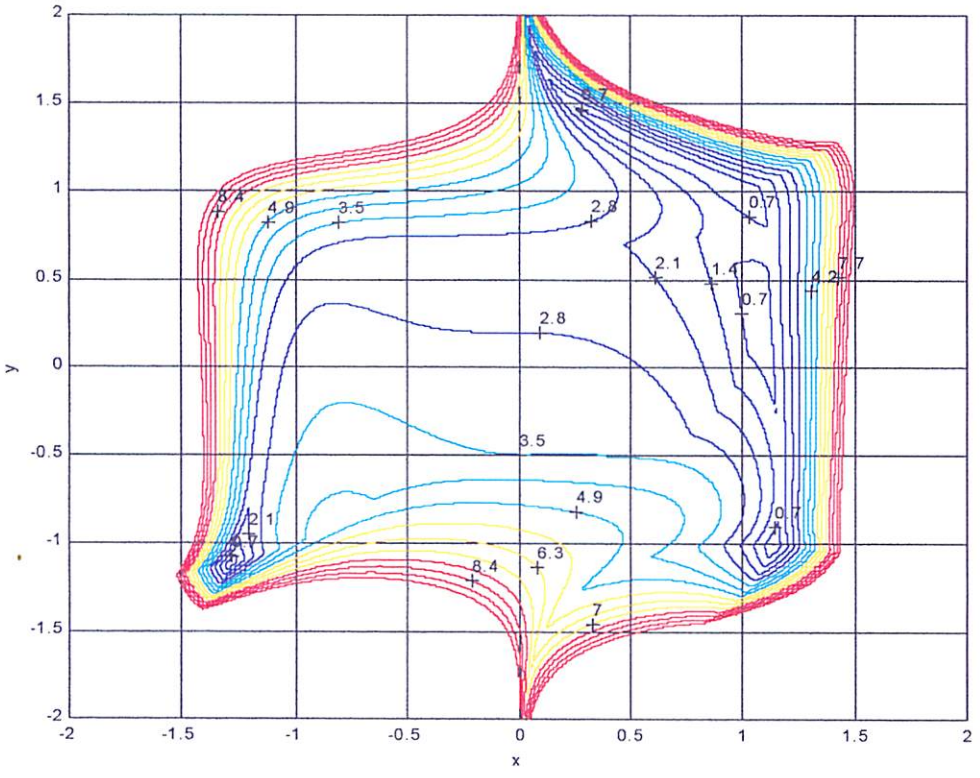

x
y

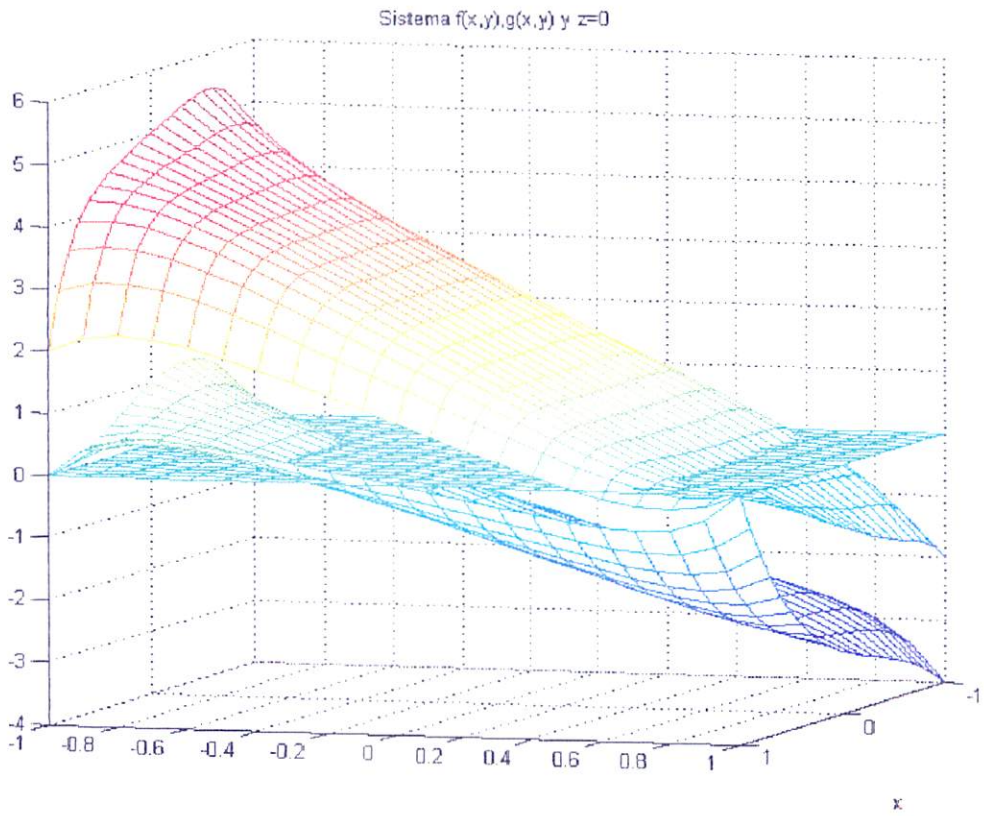
ANEXO II - GRÁFICAS INTERESANTES

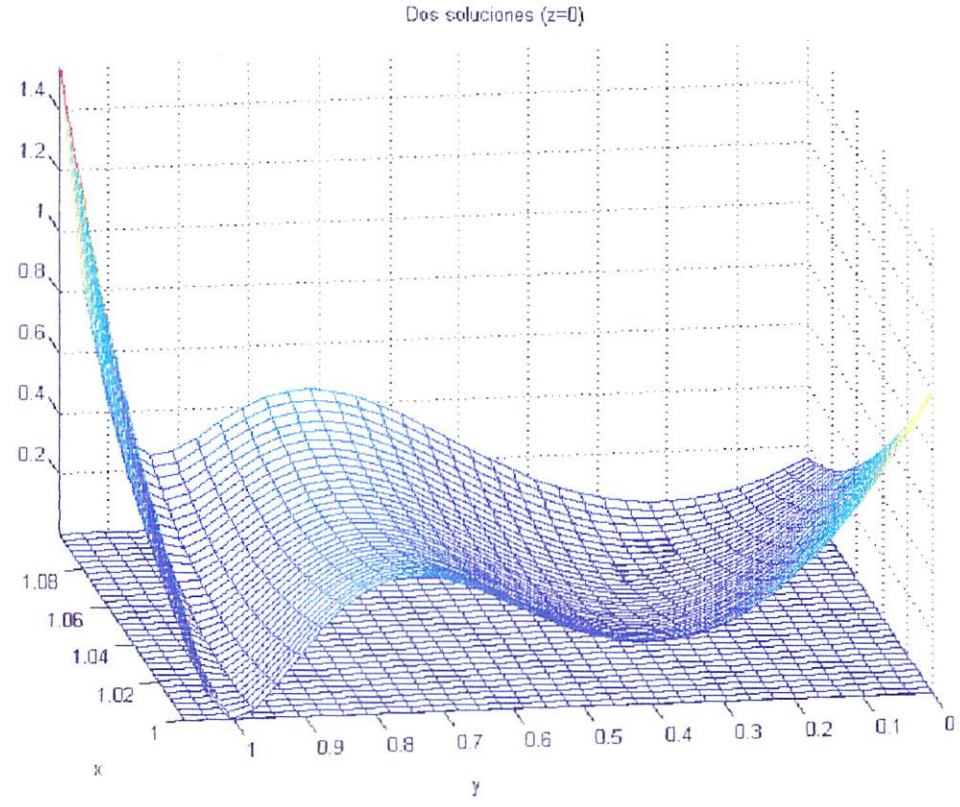





Sistema f(x,y),g(x,y)
300
200
100
0
-100
-200
-300
2
1
0
-1
-2
2
1
0
-1
-2
x
y


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1.5
-1
-0.5
0
0.5
1
1.5
2
x
y
8.4
4.9
3.5
2.8
2.1
1.4
0.7
4.2
7.7
6.3
7


Sistema f(x,y),g(x,y) y z=0
6
5
4
3
2
1
0
-1
-2
-3
-4
-1
-0.8
-0.6
-0.4
-0.2
0
0.2
0.4
0.6
0.8
1
1
0
-1
x


Dos soluciones (z=0)
1.4
1.2
1
0.8
0.6
0.4
0.2
1.08
1.06
1.04
1.02
1
x
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
y

ANEXO III – IMPLEMENTACIONES

- FUNCIONES AUXILIARES

```
function f=ftrid(p)
%Sistema f-g
format long e
x=p(1);
y=p(2);
f=zeros(1,2);
f(1,1)=x^3+x^2*y+y^7*x-2*y-1;
f(1,2)=x.^4*y-x.^6+y.^5*x-3*y+2;

function salida=f(x,y)
%funcion f del sistema f-g
salida=x.^3+x.^2.*y+y.^7.*x-2.*y-1;

function salida=fderx(x,y)
%derivada parcial de f respecto de x, del sistema f-g
salida=3*x.^2+2*x*y+y.^7;

function salida=fdery(x,y)
%derivada parcial de f respecto de y, del sistema f-g
salida=x.^2+7*y.^6*x-2;

function salida=g(x,y)
%funcion g del sistema f-g
salida=x.^4.*y-x.^6+y.^5.*x-3.*y+2;

function salida=gderx(x,y)
%derivada parcial de g respecto de x, del sistema f-g
salida=4*y*x.^3-6*x.^5+y.^5;

function salida=gdery(x,y)
%derivada parcial de g respecto de y, del sistema f-g
salida=x.^4+5*x*y.^4-3;
```

```
function g=gtrid(p,n)
%gtrid(p,n)
% -p es el punto |x,y| a sustituir en g
% -n indica la componente que se quiere calcular
%       n=1 -> x
%       n=2 -> y
%       sino-> x,y
%
% Calcula los valores correspondientes a x e y mediante
%   las funciones g1(x,y) y g2(x,y): (x,y) -> (g1,g2).
format long e
x=p(1);
y=p(2);
g=zeros(1,2);    % Inicializa g a ceros.
if(n==1)         % Calcula la primera coordenada (x=g(x,y))
  g(1,1)=(-(x.^3)-(x.^2).*y+(2.*y)+1)/(y.^7);
elseif(n==2)     % Calcula la segunda coordenada (y=g(x,y))
  g(1,2)=x.^3+x.^2.*y+y.^7.*x-2.*y-1+y;
else             % Calcula las dos coordenadas
  g(1,1)=(-(x.^3)-(x.^2).*y+(2.*y)+1)/(y.^7);
  g(1,2)=x.^3+x.^2.*y+y.^7.*x-2.*y-1+y;
end

function salida=h(x,y)
%function salida=h(x,y)
%   h es la suma de los cuadrados de las funciones f y g del sistema f-g
%
%¡¡LA FUNCION H Y SUS DERIVADAS SE ACTUALIZAN AL MODIFICAR F y G!!
salida=(f(x,y).^2)+(g(x,y).^2);

function GRAD=hderx(x,y)
%function GRAD=hderx(x,y)
%   hderx es la derivada respecto de x de h
format long
GRAD=2*f(x,y)*fderx(x,y)+2*g(x,y)*gderx(x,y);


function GRAD=hdery(x,y)
%function GRAD=hdery(x,y)
%   hdery es la derivada respecto de y de h
format long
GRAD=2*f(x,y)*fdery(x,y)+2*g(x,y)*gdery(x,y);


function GRAD=gradh(p)
%gradh(p)
%
% Calcula el gradiente de h en un punto concreto
% UTILIZA f, fderx, fdery, g, gderx, gdery
format long
GRAD=zeros(1,2);
GRAD(1)=2*f(p(1),p(2))*fderx(p(1),p(2))+2*g(p(1),p(2))*gderx(p(1),p(2));
GRAD(2)=2*f(p(1),p(2))*fdery(p(1),p(2))+2*g(p(1),p(2))*gdery(p(1),p(2));
```

```
function JA=jacf(p)
%jacf(p)
%
% Calcula el jacobiano del vector p para el sistema ftrid(p).
% (ES UNA MATRIZ DE 2x2)
format long e
x=p(1);
y=p(2);
JA=zeros(2,2);
JA=[fderx(x,y) fdery(x,y);
   gderx(x,y) gdery(x,y)];


function graficar(fun,fun_dx,fun_dy,x_f,y_f,xmin,xmax,ymin,ymax,res,tol,color)
%function graficar(fun,fun_dx,fun_dy,x_f,y_f,xmin,xmax,ymin,ymax,res,tol,color)
%
%-fun            -> funcion a graficar
%-fun_dx         -> derivada respecto de x de la funcion a derivar
%-fun_dy         -> derivada respecto de y de la funcion a derivar
%-x_f y_f        -> punto inicial
%-xmin,xmax      -> intervalo en x a graficar
%-ymin,ymax      -> intervalo en y a graficar
%-res            -> incremento en x e y');
%-tol            -> tolerancia en el calculo
%-color          -> color de linea
hold on
for n=1:2                       %Primer trazado en x e y con incrementos
    for x=xmin:res:xmax
    y_f=newtondy(x,y_f,15,20,fun,fun_dy);
    valor=feval(fun,x,y_f);
    if((abs(valor)<=10^(-tol) | tol==0))
      plot(x,y_f,color);
    end
    end
  for y=ymin:res:ymax
    x_f=newtondx(x_f,y,15,20,fun,fun_dx);
    valor=feval(fun,x_f,y);
    if((abs(valor)<=10^(-tol) | tol==0))
      plot(x_f,y,color);
    end
  end
  tmpxmin=xmin;                 %Cambios para el trazado en x e y con decrementos
  xmin=xmax;
  xmax=tmpxmin;
  tmpymin=ymin;
  ymin=ymax;
  ymax=tmpymin;
  res=-res;
  x_f=min(xmin,xmax);           %Cambio de punto inicial
  y_f=min(ymin,ymax);
end                             %Ajuste de los ejes
axis([xmin xmax ymin ymax]);
grid on
shg
hold off
```

```
function p1=newtondx(x,y,tolerancia,ITMAX,f,fprima)
%function p1=newtondx(x,y,tolerancia,ITMAX,f,fprima)
% A partir de un y calcula una buena raiz aproximada para la funcion f(y).
% tolerancia marca la tolerancia [10^(-tolerancia)].
% ITMAX marca el numero maximo de iteraciones.
format long
n=0;TOL=10.^(-tolerancia);p0=x;p1=x;

temporal=feval(fprima,x,y);
if(temporal~=0)
  p1=x-feval(f,x,y)/temporal;
end

while(temporal~=0 & abs(p1-p0)>TOL & n<ITMAX)
  p1=x-feval(f,x,y)/temporal;
  if(p1~=y)
    p0=x;
    x=p1;
    temporal=feval(fprima,x,y);
  else
    p1=x;
    break;   %% Si ya no se pueden representar mas cifras decimales exactas
  end
  n=n+1;
end

function p1=newtondy(x,y,tolerancia,ITMAX,f,fprima)
%function p1=newtondy(x,y,tolerancia,ITMAX,f,fprima)
% A partir de un y calcula una buena raiz aproximada para la funcion f(y).
% tolerancia marca la tolerancia [10^(-tolerancia)].
% ITMAX marca el numero maximo de iteraciones.
format long
n=0;TOL=10.^(-tolerancia);p0=y;p1=y;

temporal=feval(fprima,x,y);
if(temporal~=0)
  p1=y-feval(f,x,y)/temporal;
end

while(temporal~=0 & abs(p1-p0)>TOL & n<ITMAX)
  p1=y-feval(f,x,y)/temporal;
  if(p1~=y)
    p0=y;
    y=p1;
    temporal=feval(fprima,x,y);
  else
    p1=y;
    break;   %% Si ya no se pueden representar mas cifras decimales exactas
  end
  n=n+1;
end
```

```
function contornoh(xmin,xmax,ymin,ymax,incx,curvas)
%function contornoh(xmin,xmax,ymin,ymax,incx,curvas)
% -xmin y xmax intervalo en x
% -ymin y ymax intervalo en y
% -incx       incremento en x e y
% -curvas     numero de curvas de nivel
incy=(ymax-ymin)/((xmax-xmin)/incx);
[X,Y]=meshgrid(xmin:incx:xmax, ymin:incy:ymax);
Z=h(X,Y);
[c,h]=contour(X,Y,Z,curvas);
clabel(c,h,'manual');
xlabel('x');
ylabel('y');
grid on
shg
hold off

function graficarh(xmin,xmax,ymin,ymax,incx)
%function graficarh(xmin,xmax,ymin,ymax,incx)
% -xmin y xmax es el intervalo en x
% -ymin e ymax es el intervalo en y
% -incx es el incremento en x
zoom off
rotate3d on                                    %Rotacion 3D activada
incy=(ymax-ymin)/((xmax-xmin)/incx);           %Calculo del incremento en y
[X,Y]=meshgrid(xmin:incx:xmax, ymin:incy:ymax); %Conversion a matriz de x e y
Z=h(X,Y);                                      %Calculo de la altura
mesh(X,Y,Z);                                   %Representacion 3D
xlabel('x');
ylabel('y');
grid on
axis tight,shg

function graficarf(xmin,xmax,ymin,ymax,incx)
%function graficarh(xmin,xmax,ymin,ymax,incx)
% -xmin y xmax es el intervalo en x
% -ymin e ymax es el intervalo en y
% -incx es el incremento en x
zoom off
rotate3d on                                    %Rotacion 3D activada
incy=(ymax-ymin)/((xmax-xmin)/incx);           %Calculo del incremento en y
[X,Y]=meshgrid(xmin:incx:xmax, ymin:incy:ymax); %Conversion a matriz de x e y
Z1=f(X,Y);                                     %Calculo de la altura
mesh(X,Y,Z1);

function graficarg(xmin,xmax,ymin,ymax,incx)
%function graficarh(xmin,xmax,ymin,ymax,incx)
% -xmin y xmax es el intervalo en x
% -ymin e ymax es el intervalo en y
% -incx es el incremento en x
zoom off,rotate3d on                           %Rotacion 3D activada
incy=(ymax-ymin)/((xmax-xmin)/incx);           %Calculo del incremento en y
[X,Y]=meshgrid(xmin:incx:xmax, ymin:incy:ymax); %Conversion a matriz de x e y
Z1=g(X,Y);                                     %Calculo de la altura
mesh(X,Y,Z1);
```

- *MÉTODO DEL PUNTO FIJO*

```
function xy=pfijo(p,tolerancia,MAXIT)
%pfijo(p,tolerancia,MAXIT)
%
% ·p=(x,y) es el punto inicial.
% ·tolerancia es la diferencia mínima entre dos puntos consecutivos hallados.
% ·MAXIT es el máximo de iteraciones permisibles.
format long
continuar=1;
n=0;
TOL=10.^(-tolerancia);
xy=p;
iniciomedida;
while(continuar)
  n=n+1;
  p=xy;
  xy=gtrid(xy,3);
  if(n>=MAXIT | norm(xy-p,inf)<TOL | norm(xy-p,2)<TOL)
    continuar=0;
  end
end
finmedida(n);
if(n>=MAXIT)
  disp('Acabado por fin del numero de iteraciones');
elseif(norm(xy-p,2)<TOL & ~(n>MAXIT))
  disp('Acabado por norm(xy-p,2)<TOL');
elseif(norm(xy-p,inf)<TOL & ~(n>MAXIT))
  disp('Acabado por norm(xy-p,inf)<TOL');
end
```

- *MÉTODO DE SEIDEL*

```
function xy=seidel(p,tolerancia,MAXIT)
%seidel(p,tolerancia,MAXIT)
%
% ·p=(x,y) es el punto inicial.
% ·tolerancia es la diferencia mínima entre dos puntos consecutivos hallados.
% ·MAXIT es el máximo de iteraciones permisibles.
format long e
continuar=1;
n=0;
TOL=10.^(-tolerancia);
iniciomedida;
while(continuar)
  n=n+1;
  X=gtrid(p,1);
  xy(1)=X(1);
  p(1)=xy(1);
  Y=gtrid(p,2);
  xy(2)=Y(2);
```

```
  if(norm(xy-p,inf)<TOL | norm(xy-p,2)<TOL | n>=MAXIT)
    continuar=0;
  end
  p=xy;
end
finmedida(n);
if(n>=MAXIT)
  disp('Acabado por fin del numero de iteraciones');
elseif(norm(xy-p,2)<TOL & ~(n>MAXIT))
  disp('Acabado por norm(xy-p,2)<TOL');
elseif(norm(xy-p,inf)<TOL & ~(n>MAXIT))
  disp('Acabado por norm(xy-p,inf)<TOL');
end
```

- MÉTODO DE NEWTON (CON GRÁFICAS)

```
function p1=newtontrid(p,tolerancia,MAXIT)
%newtontrid(p,tolerancia,MAXIT)
%
% ·p es el vector inicial
% ·tolerancia es la separación mínima entre dos soluciones consecutivas
% ·MAXIT es el máximo de iteraciones permitidas
format long e
continuar=1;
n=0;
xmax=p(1);xmin=p(1);
ymax=p(2);ymin=p(2);
subplot(1,2,1);
hold on;
plot(p(1),p(2),'kO');
subplot(1,2,2);
hold on;
plot3(p(1),p(2),h(p(1),p(2)),'kO');
p0=p;
TOL=10.^(-tolerancia);
iniciomedida;
while(continuar)
  p1=p0-(jacf(p0)\ftrid(p0)')';
  if(p1(2)>ymax)
    ymax=p1(2);
  elseif(p1(2)<ymin)
    ymin=p1(2);
  end
  if(p1(1)>xmax)
    xmax=p1(1);
  elseif(p1(1)<xmin)
    xmin=p1(1);
  end
  subplot(1,2,1);
  plot(p1(1),p1(2),'kO');
  subplot(1,2,2);
  plot3(p1(1),p1(2),h(p1(1),p1(2)),'kO');
```

```
    if(norm(p1-p0,inf)<TOL | norm(p1-p0,2)<TOL | n>MAXIT)
    continuar=0;
  end
  datos(n+1,1)=p1(1);
  datos(n+1,2)=p1(2);
  datos(n+1,3)=norm(p1-p0,inf);
  datos(n+1,4)=norm(p1-p0,2);
  p0=p1;
  n=n+1;
end
finmedida(n-1);
if(n>=MAXIT)
  texto=sprintf('Acabado por fin del numero de iteraciones');
else
  if(norm(p1-p0,2)<TOL)
    texto=sprintf('Acabado por norm(p1-p,2)<TOL');
    elseif(norm(p1-p0,inf)<TOL)
    texto=sprintf('Acabado por norm(p1-p,inf)<TOL');
  end
end
disp(texto);
save newtontrid.txt datos -ascii -double
if(xmin>xmax)
  tmp=xmin;
  xmin=xmax;
  xmax=tmp;
elseif(xmin==xmax)
  xmax=xmax+0.1;
  xmin=xmin-0.1;
end
xmin=xmin-0.1;
xmax=xmax+0.1;
ymin=ymin-0.1;
ymax=ymax+0.1;
zoom on;
subplot(1,2,1);
title(sprintf('(%s , %s) -> (%s , %s) , ...
h=%s',num2str(p(1)),num2str(p(2)),num2str(p1(1)),num2str(p1(2)),num2str(h(p1(1),p1(2)))));
contornoh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/10),50);
axis([xmin xmax ymin ymax]);
subplot(1,2,2);
graficarh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/30));
axis([xmin xmax ymin ymax]);
hidden off
shg
hold off
```

- MÉTODO DE NEWTON (SIN GRÁFICAS)

```
function p1=newtontridsg(p,tolerancia,MAXIT)
%newtontrid(p,tolerancia,MAXIT)
%
% ·p es el vector inicial
% ·tolerancia es la separación mínima entre dos soluciones consecutivas
% ·MAXIT es el máximo de iteraciones permitidas
format long e
continuar=1;
n=0;
p0=p;
TOL=10.^(-tolerancia);
iniciomedida;
while(continuar)
  p1=p0-(jacf(p0)\ftrid(p0)')';
if(norm(p1-p0,inf)<TOL | norm(p1-p0,2)<TOL | n>MAXIT)
    continuar=0;
  end
  datos(n+1,1)=p1(1);
  datos(n+1,2)=p1(2);
  datos(n+1,3)=norm(p1-p0,inf);
  datos(n+1,4)=norm(p1-p0,2);
  p0=p1;
  n=n+1;
end
finmedida(n-1);
if(n>=MAXIT)
  texto=sprintf('Acabado por fin del numero de iteraciones');
else
  if(norm(p1-p0,2)<TOL)
    texto=sprintf('Acabado por norm(p1-p,2)<TOL');
    elseif(norm(p1-p0,inf)<TOL)
    texto=sprintf('Acabado por norm(p1-p,inf)<TOL');
  end
end
disp(texto);
save newtontrid.txt datos -ascii -double
```

- MÉTODO DE NEWTON GLOBAL (CON λ=CTE^{-L} Y CON GRÁFICAS)

```
function p1=nglob2n(p,cte,TOL)
%function nglob2n(p,cte)
%
%-p es el punto inicial.
%-TOL es la tolerancia en altura
%
%USA EL CRITERIO cte^-n PARA n=n+1 n=0,1,2,...
format long

NITER=0;
MAXIT=100;

xmax=p(1);xmin=p(1);
ymax=p(2);ymin=p(2);

L=0;
clf
p0=p;
subplot(1,2,1);
hold on;
plot(p0(1),p0(2),'k.');
subplot(1,2,2);
hold on;
plot3(p0(1),p0(2),h(p0(1),p0(2)),'k.');

p1=(p0)-(cte.^(-L)).*(jacf(p0)\ftrid(p0)')';
NITER=0;
while(~hmenor(p0,p1) & NITER<MAXIT)
  L=L+1;
  p1=(p0)-(cte.^(-L)).*(jacf(p0)\ftrid(p0)')';
  NITER=NITER+1;
end
altv=h(p(1),p(2));  %altura inicial
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
if(NITER>=MAXIT & altn>0.1)
  L=3;
end

if(p1(2)>ymax)
  ymax=p1(2);
end
if(p1(2)<ymin)
  ymin=p1(2);
end

subplot(1,2,1);
plot(p1(1),p1(2),'k.');
subplot(1,2,2);
plot3(p1(1),p1(2),h(p1(1),p1(2)),'k.');
```

```
while (altv>altn | altn>10.^-TOL)
  if(altn>10.^-TOL & altv<=altn)
    L=3;
  else
    L=L+1;
  end
  altv=altn;
  p0=p1;
  p1=(p0)-(cte.^(-L)).*(jacf(p0)\ftrid(p0)')';%[x1 y1]));
  NITER=0;
  while(~hmenor(p0,p1) & NITER<MAXIT)
    L=L+1;
    p1=(p0)-(cte.^(-L)).*(jacf(p0)\ftrid(p0)')';
    NITER=NITER+1;
  end
  altn=h(p1(1),p1(2));
  if(NITER>=MAXIT & altn>0.1)
    L=3;
  end
  subplot(1,2,1);plot(p1(1),p1(2),'k.');
  subplot(1,2,2);plot3(p1(1),p1(2),h(p1(1),p1(2)),'k.');
  if(p1(2)>ymax)
    ymax=p1(2);
  elseif(p1(2)<ymin)
    ymin=p1(2);
  end
  if(p1(1)>xmax)
    xmax=p1(1);
  elseif(p1(1)<xmin)
    xmin=p1(1);
  end
end
if(xmin>xmax)
  tmp=xmin;xmin=xmax;xmax=tmp;
elseif(xmin==xmax)
  xmax=xmax+0.1;xmin=xmin-0.1;
end
xmin=xmin-0.1;xmax=xmax+0.1;
ymin=ymin-0.1;ymax=ymax+0.1;
zoom on;subplot(1,2,1);
title(sprintf('(%s , %s) -> (%s , %s) , ...
h=%s',num2str(p(1)),num2str(p(2)),num2str(p1(1)),num2str(p1(2)),num2str(h(p1(1),p1(2)))));
contornoh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/10),30);
axis([xmin xmax ymin ymax]);subplot(1,2,2);
graficarh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/30));
axis([xmin xmax ymin ymax]);
hidden off, shg, hold off
function valor=hmenor(x1,x2)
alt1=h(x1(1),x1(2));
alt2=h(x2(1),x2(2));
if(alt2>=alt1)
  valor=0;
else
  valor=1;
end
```

- MÉTODO DE NEWTON GLOBAL (CON λ=CTE^{-L} Y SIN GRÁFICAS)

```
function p1=nglob2nsg(p,cte,TOL)
%function nglob2nsg(p,cte,TOL)
%
%-p es el punto inicial.
% -TOL tolerancia en altura
%
%USA EL CRITERIO cte^-n PARA n=n+1 n=0,1,2,...
format long
it=1;                    %Inicializacion de numero de iteraciones
NITER=0;MAXIT=100;
L=0;
p0=p;
iniciomedida;
p1=(p0)-(2.^(-L)).*(jacf(p0)\ftrid(p0)')';
datos(it,1)=p1(1);
datos(it,2)=p1(1);
altv=h(p(1),p(2));    %inicializa con una altura grande
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
while (altv>altn | altn>10.^-TOL)
  if(altn>10.^-TOL & altv<=altn)
    L=0;
  end
  L=L+1;
  altv=altn;
  p0=p1;
  p1=p0-(2.^(-L)).*(jacf(p0)\ftrid(p0)')';
  datos(it,1)=p1(1);
  datos(it,2)=p1(1);
  NITER=0;
  while(~hmenor(p0,p1) & NITER<MAXIT)
    L=L+1;
    p1=(p0)-(cte.^(-L)).*(jacf(p0)\ftrid(p0)')';
    NITER=NITER+1;
  end
  altn=h(p1(1),p1(2));
  if(NITER>=MAXIT & altn>0.1)
    L=0;
  end
  it=it+1;
end
finmedida(it);
save nglob2n.txt datos -ascii -double
function valor=hmenor(x1,x2)
alt1=h(x1(1),x1(2));
alt2=h(x2(1),x2(2));
if(alt2>=alt1)
  valor=0;
else
  valor=1;
end
```

- MÉTODO DE NEWTON GLOBAL (CON λ=L CON GRÁFICAS)

```
function nglobal(p)
%%function nglobal(p)
%
% -p es el punto inicial
format long
ymax=p(2);ymin=p(2);
L=0;
clf
subplot(1,2,1);
hold on;grid on;
subplot(1,2,2);
hold on; p0=p;
subplot(1,2,1);plot(p0(1),p0(2),'k.');
subplot(1,2,2);plot3(p0(1),p0(2),h(p0(1),p0(2)),'k.');
p1=(p0)-L.*(jacf(p0)\ftrid(p0)')';
if(p1(2)>ymax)
  ymax=p1(2);
end
if(p1(2)<ymin)
  ymin=p1(2);
end
subplot(1,2,1);plot(p1(1),p1(2),'k.');
altv=h(p(1),p(2));  %inicializa con una altura grande
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
it=1;
while (altv>altn | altn>0.1)
  if(altn>1 & altv<altn)
    L=0;
  end
  L=L+0.001; altv=altn;
  p0=p1;
  p1=p0-L.*(jacf(p0)\ftrid(p0)')';
  subplot(1,2,1);plot(p1(1),p1(2),'k.');
  subplot(1,2,2);plot3(p1(1),p1(2),h(p1(1),p1(2)),'k.');
  if(p1(2)>ymax)
    ymax=p1(2);
  end
  if(p1(2)<ymin)
    ymin=p1(2);
  end
  altn=h(p1(1),p1(2));
  it=it+1;
end
sol=p1;
xmin=p(1);xmax=sol(1);
if(xmin>xmax)
  tmp=xmin;
  xmin=xmax;
  xmax=tmp;
elseif(xmin==xmax)
  xmax=xmax+0.1;
  xmin=xmin-0.1;
end
```

```
xmin=xmin-0.2
xmax=xmax+0.2;
ymin=ymin-0.2;
ymax=ymax+0.2;
subplot(1,2,1);
title(sprintf('(%s, %s)->(%s, %s), ...
h=%s',num2str(p(1)),num2str(p(2)),num2str(sol(1)),num2str(sol(2)),num2str(h(sol(1),sol(2)))));
zoom on;
subplot(1,2,1);
axis([xmin xmax ymin ymax]);
contornoh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/10),30);
subplot(1,2,2);
axis([xmin xmax ymin ymax]);
graficarh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/30));
hidden off
shg
hold off
it
```

- MÉTODO DE NEWTON GLOBAL (CON λ=L SIN GRÁFICAS)

```
function p1=nglobal(p,TOL)
% método de newton global
% -p es el punto inicial
% -TOL tolerancia en altura
format long
it=1;     %Inicializacion de numero de iteraciones
L=0;
p0=p;
iniciomedida;
p1=(p0)-L.*(jacf(p0)\ftrid(p0)')';
datos(it,1)=p1(1);
datos(it,2)=p1(1);
altv=h(p(1),p(2));  %inicializa con una altura grande
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
while (altv>altn | altn>10.^-TOL)
  if(altn>10.^-TOL & altv<=altn)
    L=0;
    altn;
  end
    L=L+0.001;
  altv=altn;
  p0=p1;
  p1=p0-L.*(jacf(p0)\ftrid(p0)')';
  datos(it,1)=p1(1);
  datos(it,1)=p1(2);
  altn=h(p1(1),p1(2));
  it=it+1;
end
finmedida(it);
save nglobal.txt datos -ascii -double
```

- MÉTODO DE BROYDEN (CON GRAFICAS)

```
function p1=broyden(p,tolerancia,MAXIT)
%broyden(p,tolerancia,MAXIT)
%
% -p es el vector inicial
% -tolerancia es la distancia mínima entre dos soluciones consecutivas (10^-tolerancia)
% -MAXIT es el máximo de iteraciones permisibles
format long
xmax=p(1);xmin=p(1);
ymax=p(2);ymin=p(2);
continuar=1;
n=10;
p0=p;
subplot(1,2,1);
hold on;
plot(p0(1),p0(2),'k.');
subplot(1,2,2);
hold on;
plot3(p0(1),p0(2),h(p0(1),p0(2)),'k.');
TOL=10.^(-tolerancia);
iniciomedida;
A0=jacf(p0);
while(continuar)
  p1=p0-(A0\ftrid(p0)')';
  datos(n-9,1)=p1(1);
  datos(n-9,2)=p1(1);
  if(norm(p1-p0,inf)<TOL | norm(p1-p0,2)<TOL | n>=MAXIT)
    continuar=0;
  else
    A0=A0+((ftrid(p1)-ftrid(p0)-(A0*(p1-p0)')')'*(p1-p0))/(norm(p1-p0,2));
  end
  subplot(1,2,1);plot(p1(1),p1(2),'k.');
  subplot(1,2,2);plot3(p1(1),p1(2),h(p1(1),p1(2)),'k.');
  if(p1(2)>ymax)
    ymax=p1(2);
  elseif(p1(2)<ymin)
    ymin=p1(2);
  end
  if(p1(1)>xmax)
    xmax=p1(1);
  elseif(p1(1)<xmin)
    xmin=p1(1);
  end
  p0=p1;
  n=n+1;
end
finmedida(n-1);
if(n>=MAXIT)
  texto=sprintf('Acabado por fin del numero de iteraciones %d',n-1);
elseif(norm(p1-p0,2)<TOL & ~(n>MAXIT))
  texto=sprintf('Acabado por norm(p1-p,2)<TOL, Iteraciones: %d',n-1);
elseif(norm(p1-p0,inf)<TOL & ~(n>MAXIT))
  texto=sprintf('Acabado por norm(p1-p,inf)<TOL, Iteraciones: %d',n-1);
end
```

```
disp(texto);
save broyden.txt datos -ascii -double
if(xmin>xmax)
  tmp=xmin;
  xmin=xmax;
  xmax=tmp;
elseif(xmin==xmax)
  xmax=xmax+0.1;
  xmin=xmin-0.1;
end
xmin=xmin-0.1;
xmax=xmax+0.1;
ymin=ymin-0.1;
ymax=ymax+0.1;
zoom on;
subplot(1,2,1);
title(sprintf('(%s , %s) -> (%s , %s) , ...
h=%s',num2str(p(1)),num2str(p(2)),num2str(p1(1)),num2str(p1(2)),num2str(h(p1(1),p1(2)))));
contornoh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/10),50);
axis([xmin xmax ymin ymax]);
subplot(1,2,2);
graficarh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/30));
axis([xmin xmax ymin ymax]);
hidden off
shg
hold off
```

- MÉTODO DE BROYDEN (SIN GRAFICAS)

```
function p1=broyden(p,tolerancia,MAXIT)
%broyden(p,tolerancia,MAXIT)
%
% ·p es el vector inicial
% ·tolerancia es la distancia mínima entre dos soluciones consecutivas (10^-tolerancia)
% ·MAXIT es el máximo de iteraciones permisibles
format long
continuar=1;
n=10;
p0=p;
TOL=10.^(-tolerancia);
iniciomedida;
A0=jacf(p0);
while(continuar)
  p1=p0-(A0\ftrid(p0)')';
  datos(n-9,1)=p1(1);
    datos(n-9,2)=p1(1);
  if(norm(p1-p0,inf)<TOL | norm(p1-p0,2)<TOL | n>=MAXIT)
    continuar=0;
  else
    A0=A0+((ftrid(p1)-ftrid(p0)-(A0*(p1-p0)')')'*(p1-p0))/(norm(p1-p0,2));
  end
  p0=p1;
  n=n+1;
end
```

```
finmedida(n-1);
if(n>=MAXIT)
  texto=sprintf('Acabado por fin del numero de iteraciones %d',n-1);
elseif(norm(p1-p0,2)<TOL & ~(n>MAXIT))
  texto=sprintf('Acabado por norm(p1-p,2)<TOL, Iteraciones: %d',n-1);
elseif(norm(p1-p0,inf)<TOL & ~(n>MAXIT))
  texto=sprintf('Acabado por norm(p1-p,inf)<TOL, Iteraciones: %d',n-1);
end
disp(texto);
save broyden.txt datos -ascii -double
```

- MÉTODO DE MÁXIMO DESCENSO (CON $\lambda=CTE^{-L}$ CON GRAFICAS)

```
function p1=md2(p,cte,TOL)
%function md2(p,cte,TOL)  -METODO DE MAXIMO DESCENSO CON GRAFICOS-
%
%-p es el punto inicial.
%-TOL tolerancia en altura
%USA EL CRITERIO CTE^-n PARA n=n+1 n=0,1,2,...
format long
tol=10.^-TOL;
NITER=0;MAXIT=100;
xmax=p(1);xmin=p(1);
ymax=p(2);ymin=p(2);
L=0;
clf
p0=p;
p1=p0;
subplot(1,2,1);
hold on;
plot(p0(1),p0(2),'k.');
subplot(1,2,2);
hold on;
plot3(p0(1),p0(2),h(p0(1),p0(2)),'k.');
NITER=0;
altv=h(p(1),p(2));  %altura inicial
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
continuar=1;
while (altv>altn | altn>tol | continuar)
  if(altn>tol & altv<=altn)
    L=3;
  else
    L=L+1;
  end
  altv=altn;
  p0=p1;
  p1=(p0)-(cte.^(-L)).*gradh(p0);
  NITER=0;
  while(~hmenor(p0,p1) & NITER<MAXIT)
    L=L+1;
    p1=(p0)-(cte.^(-L)).*gradh(p0);
    NITER=NITER+1;
  end
  altn=h(p1(1),p1(2));
```

```
  if(NITER>=MAXIT & altn>0.1)
    L=3;
  end
  subplot(1,2,1);plot(p1(1),p1(2),'k.');
  subplot(1,2,2);plot3(p1(1),p1(2),h(p1(1),p1(2)),'k.');
  if(p1(2)>ymax)
    ymax=p1(2);
  elseif(p1(2)<ymin)
    ymin=p1(2);
  end
  if(p1(1)>xmax)
    xmax=p1(1);
  elseif(p1(1)<xmin)
    xmin=p1(1);
  end
  continuar=0;
end
if(xmin>xmax)
  tmp=xmin;
  xmin=xmax;
  xmax=tmp;
elseif(xmin==xmax)
  xmax=xmax+0.1;
  xmin=xmin-0.1;
end
xmin=xmin-0.1;
xmax=xmax+0.1;
ymin=ymin-0.1;
ymax=ymax+0.1;
zoom on;
subplot(1,2,1);
title(sprintf('(%s , %s) -> (%s , %s) , ...
h=%s',num2str(p(1)),num2str(p(2)),num2str(p1(1)),num2str(p1(2)),num2str(h(p1(1),p1(2)))));
contornoh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/20),20);
axis([xmin xmax ymin ymax]);
subplot(1,2,2);
graficarh(xmin,xmax,ymin,ymax,(abs(xmax-xmin)/30));
axis([xmin xmax ymin ymax]);
hidden off
shg
hold off
NITER
function valor=hmenor(x1,x2)
alt1=h(x1(1),x1(2));
alt2=h(x2(1),x2(2));
if(alt2>=alt1)
  valor=0;
else
  valor=1;
end
```

- MÉTODO DE MÁXIMO DESCENSO (CON $\lambda=CTE^{-L}$ SIN GRAFICAS)

```
function p1=md2sg(p,cte,TOL)
%function md2sg(p,cte,TOL)     -METODO DE MAXIMO DESCENSO SIN GRAFICOS-
%
%-p es el punto inicial.
%-cte base del criterio.
%-TOL es la tolerancia en altura
%
%USA EL CRITERIO cte.^-n CON INCREMENTOS DE n LA UNIDAD
format long
tol=10.^-TOL;
ittotal=1;
NITER=0;MAXIT=100;
L=0;p0=p;p1=p0;
altv=h(p(1),p(2));   %altura inicial
altn=h(p1(1),p1(2)); %altura de la funcion en (x1,y1)
continuar=1;
iniciomedida;
while (altv>altn | altn>tol | continuar)
  if(altn>tol & altv<=altn)
    L=3;
  else
    L=L+1;
  end
  altv=altn;
  p0=p1;
  p1=(p0)-(cte.^(-L)).*gradh(p0);%[x1 y1]));
  datos(ittotal,1)=p1(1);
  datos(ittotal,2)=p1(2);
  NITER=0;
  while(~hmenor(p0,p1) & NITER<MAXIT)
    L=L+1;
    p1=(p0)-(cte.^(-L)).*gradh(p0);
    datos(ittotal,1)=p1(1);
    datos(ittotal,2)=p1(2);
    NITER=NITER+1;
  end
  altn=h(p1(1),p1(2));
  if(NITER>=MAXIT & altn>0.1)
    L=3;
  end
  continuar=0;
  ittotal=ittotal+1;
end
finmedida(ittotal);
save md.txt datos -ascii -double
function valor=hmenor(x1,x2)
alt1=h(x1(1),x1(2));
alt2=h(x2(1),x2(2));
if(alt2>=alt1)
  valor=0;
else
  valor=1;
end
```

-FUNCIONES PARA MEDIDAS

```
function iniciomedida
%CONTADOR DE OPERACIONES A 0 E INCIALIZO EL TEMPORIZADOR
flops(0),tic

function finmedida(h)
% IMPRIMO EL NUMERO DE OPERACIONES EMPLEADAS
disp(' ');
disp('NUMERO DE OPERACIONES:'),disp(flops);
disp('TIEMPO EMPLEADO (segundos):'),disp(toc);
disp('ITERACIONES:'),disp(h);
```

INTERPOLACIÓN . DERIVACIÓN E INTEGRACIÓN APROXIMADA

-Análisis Numérico I-

Víctor Manuel Galdámez Salguero
Jose Manuel Mesa Gómez

- ÍNDICE -

1. INTERPOLACIÓN....3
1.1.INTRODUCCIÓN....3
1.2. ALGORITMOS DE INTERPOLACIÓN....4
1.2.1. LAGRANGE....4
1.2.2. NEVILLE....5
1.2.3. DIFERENCIAS DIVIDIDAS....7
1.2.4. DIFERENCIAS PROGRESIVAS DE NEWTON....10
2. DERIVACIÓN E INTEGRACIÓN....12
2.1.INTRODUCCIÓN....12
2.2. DERIVACIÓN NUMÉRICA....12
2.2.1.DIFERENCIAS PROGRESIVAS Y REGRESIVAS PRIMERAS....12
2.2.2.DIFERENCIA CENTRAL....13
2.2.3. EXTRAPOLACIÓN DE RICHARDSON....13
2.3. INTEGRACIÓN NUMÉRICA....15
2.3.1. REGLA DEL TRAPECIO....15
2.3.2. ALGORITMO DE ROMBERG....15
2.4. EVALUACIÓN DE LOS MÉTODOS....17
2.4.1. COMPARATIVAS . INTERPOLACIÓN....17
2.4.2. DERIVACIÓN E INTEGRACIÓN....39
2.4.2.1 DERIVACIÓN (h_0=0.1)....39
2.4.2.2. DERIVACIÓN DE POLINOMIOS INTERPOLADORES....40
2.4.2.3. INTEGRACIÓN....42
2.4.2.4. INTEGRACIÓN DE POLINOMIOS INTERPOLADORES....43

2.5. IMPLEMENTACIÓN DE LOS MÉTODOS.....44

FUNCIONES AUXILIARES DE INTERPOLACIÓN.....44

MÉTODOS DE INTERPOLACIÓN.....47

FUNCIONES AUXILIARES DE INTEGRACIÓN.....51

MÉTODOS DE DERIVACIÓN E INTEGRACIÓN.....53

FUNCIONES PARA MEDIDAS.....56

1. INTERPOLACIÓN

1.1.INTRODUCCIÓN

Una de las más útiles y más conocidas clases de funciones reales de variable real es la clase de los polinomios algebraicos, o sea, el conjunto de funciones de la forma $P_n(x) = a_0 + a_1 x + ... + a_n x^n$, donde n es un entero no negativo y a_0, ... , a_n son constantes reales. Una razón primordial de su importancia es que aproximan uniformemente funciones continuas; esto es, dado una función definida y continua en un intervalo cerrado, existe un polinomio que está tan "cerca" de la función dada como se desee. Este resultado se expresa más precisamente en el teorema siguiente.

Teorema (Aproximación de Weierstrass): Si f está definida y es continua en [a, b], dado un $\varepsilon > 0$, existe un polinomio P, definido en [a, b], con la propiedad de que $|f(x) - P(x)| < \varepsilon$ para toda x $\in$ [a, b].

Otro aspecto importante para considerar la clase de polinomios en la aproximación de funciones es que es sencillo determinar la derivada y la integral indefinida de cualquier polinomio y el resultado es otra vez un polinomio. Por estas razones, la clase de polinomios se usa con frecuencia para aproximar otras funciones que se conoce o se supone son continuas.

1.2. ALGORITMOS DE INTERPOLACIÓN

1.2.1. LAGRANGE

Considérese el problema de determinar un polinomio de grado uno que pase por los puntos distintos (x_0, y_0) y (x_1, y_1). Este problema es el mismo que el de aproximar una función f , para la cual $f(x_0)= y_0$ y $f(x_1)= y_1$, por medio de un polinomio de primer grado, interpolado entre, o coincidiendo con, los valores de f en los puntos dados.

Consideremos el polinomio $P(x)=\frac{(x-x_1)}{(x_0-x_1)}y_0+\frac{(x-x_1)}{(x_0-x_1)}y_1$

Cuando $x=x_0$, $P(x_0)=1\cdot y_0+0\cdot y_1=y_0=f(x_0)$

y cuando $x=x_1$, $P(x_1)=0\cdot y_0+1\cdot y_1=y_0=f(x_1)$,

así que P tiene las propiedades requeridas.

La técnica usada para a P es el método de interpolación usado con frecuencia en las tablas trigonométricas o logarítmicas. Lo que puede ser no tan obvio es que P es el único polinomio de grado 1 o menor con la propiedad de interpolación.

Para generalizar el concepto de interpolación lineal, consideramos la construcción de un polinomio de grado a lo más n que pase por los n+1 puntos $(x_0,f(x_0))$, $(x_1, f(x_1))$, ... , $(x_n, f(x_n))$. El polinomio lineal que pasa por $(x_0, f(x_0))$ y $(x_1, f(x_1))$ se construye usando los cocientes

$$L_0(x)=\frac{(x-x_1)}{(x_0-x_1)} \text{ y } L_1(x)=\frac{(x-x_1)}{(x_0-x_1)}$$

Cuando $x=x_0$, $L_0(x_0)=1$ mientras que $L_1(x_0)=0$. Cuando $x=x_1$, $L_0(x_1)=0$ mientras que $L_1(x_1)=1$.

Para el caso general necesitamos construir, para cada k=0, 1, ... , n un cociente $L_{n,k}(x)$ con la propiedad de que $L_{n,k}(x_i)=0$ cuando $y\neq k$ y $L_{n,k}(x_k)=1$. Para satisfacer que $L_{n,k}(x_i)=0$ para cada $i\neq k$ se requiere que el numerador de $L_{n,k}$ contenga el termino $(x-x_0)\ (x-x_1)\cdots(x-x_{k-1})\ (x-x_{k-1})\cdots(x-x_n)$.

Para satisfacer $L_{n,k}(x_k)=1$, el denominador de L_k debe ser igual a $(x-x_0)\ (x-x_1)\cdots(x-x_{k-1})\ (x-x_{k-1})\cdots(x-x_n)$ cuando $x=x_k$. Por lo tanto,

$$L_{n,k}(x)=\frac{(x-x_0)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)}=\prod_{\substack{i=0\\ i\neq k}}^{n}\frac{(x-x_i)}{(x_k-x_i)}$$

Ahora que se conoce la forma de $L_{n,k}$, es fácil describir al polinomio interpolante. Este polinomio se llama polinomio interpolante de Lagrange y se define en el teorema siguiente.

Teorema (Polinomio de Lagrange): Si $x_0, x_1, \ldots, x_n$ son (n+1) números diferentes y f es una función cuyos valores están dados en estos puntos, entonces existe un único polinomio P de grado a lo más n con la propiedad de que $f(x_k)=P(x_k)$ para k=0,1,...,n. Este polinomio está dado por

$$P(x) = f(x_9)L_{n,0}(x) + \ldots + f(x_n)L_{n,n}(x) = \sum_{k=0}^{n} f(x_k)L_{n,k}(x)$$

1.2.2. NEVILLE

Una dificultad que surge al usar el método de Lagrange es que es difícil trabajar con el término de error dado por el teorema que se ha denominado polinomio de Lagrange definido en la página anterior, no se sabe generalmente el grado del polinomio necesario para lograr la precisión deseada hasta que los cálculos han sido completados. La práctica usual consiste en comparar los resultados obtenidos de varios polinomios hasta que se obtiene una concordancia apropiada. Comparando el resultado de estos polinomios se ve que el trabajo realizado para calcular la aproximación mediante un polinomio de segundo grado no reduce el trabajo necesario para calcular la aproximación de tercer grado; tampoco es más fácil de obtener la aproximación de cuarto grado ya conocida la de tercer grado. El propósito de Neville es la derivación de estos polinomios aproximantes de tal manera que se utilicen con mayor ventaja los cálculos anteriores, y lo hace con el siguiente esquema recursivo:

$$Q_{i,0} = y_i$$

$$Q_{i,k} = \frac{(x - x_{i-k})Q_{i,k-1} - (x - x_i)Q_{i-1,k-1}}{x_i - x_{i-k}} \text{ para i=1, 2, ... , n+1 y k=0, 1, ... , n.}$$

DIAGRAMA DE FLUJO DE LAGRANGE

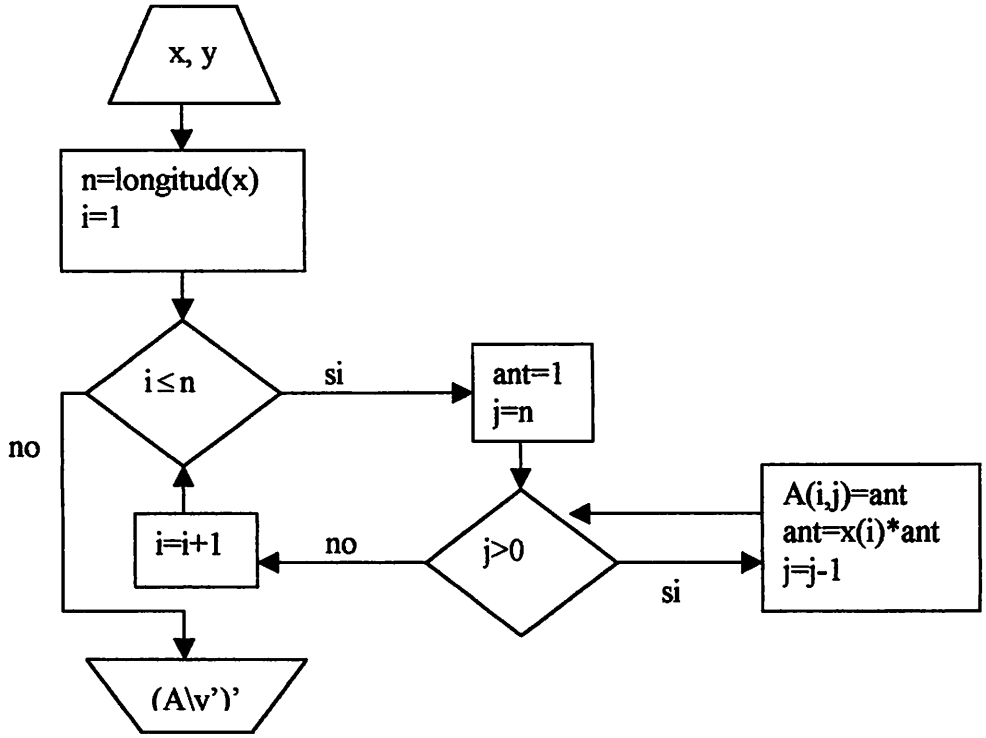
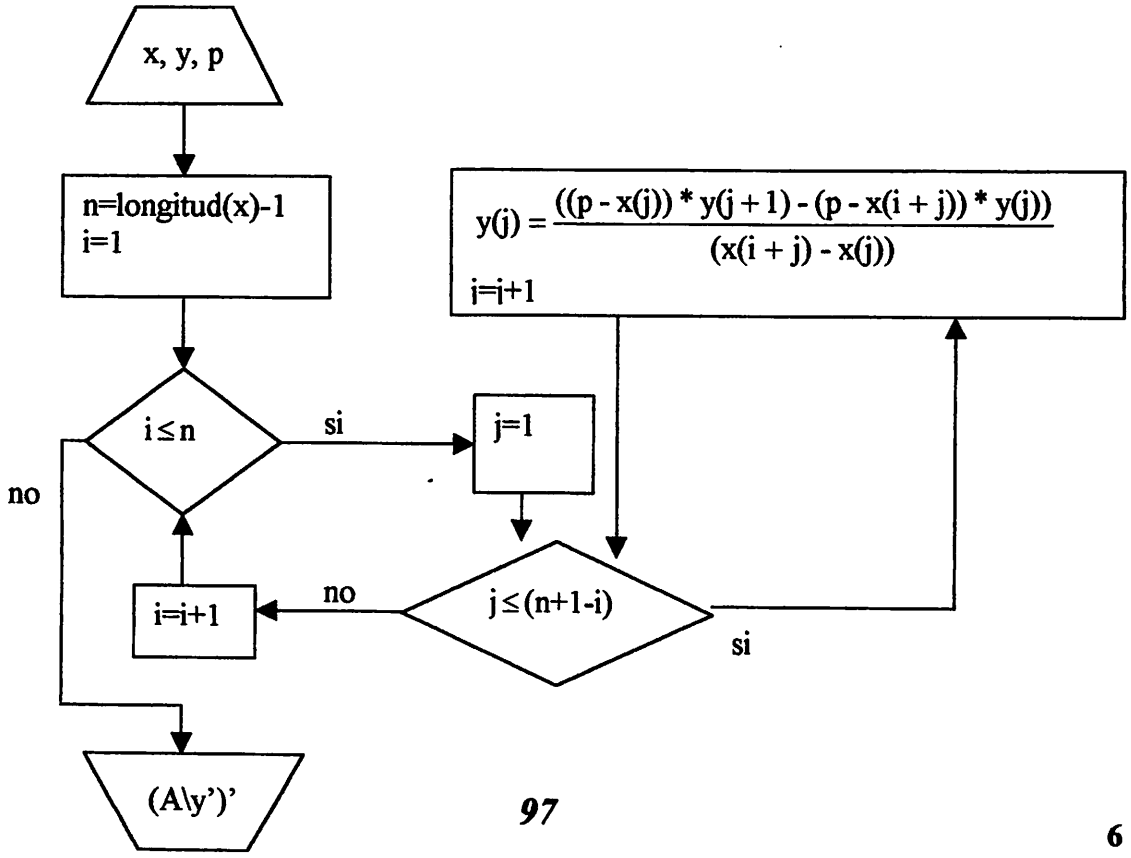


DIAGRAMA DE FLUJO DE NEVILLE



1.2.3. DIFERENCIAS DIVIDIDAS

La técnica de interpolación iterada (Neville) discutida anteriormente es útil para determinar los valores de los polinomios interpolantes de grado sucesivamente mayor en un punto particular. Sin embargo, cada uno de los elementos en la tabla de interpolación depende del punto que se está evaluando, así que la tabla no se puede usar para encontrar una representación explícita del polinomio interpolante.

Los métodos para determinar la representación explícita de un polinomio interpolante a partir de datos tabulados se conocen como métodos de diferencia dividida . Estos métodos se usaron más con propósito de cómputo antes de que el equipo de cómputo digital llegara a ser fácilmente disponible. Sin embargo, los métodos pueden usarse también para derivar técnicas para aproximar las derivadas y las integrales de funciones, así como para aproximar las soluciones de ecuaciones diferenciales.

Supongamos que P_n es el polinomio de Lagrange de grado a lo mas n que coincide con la función f en los números distintos $x_0, x_1, \ldots, x_n$. Las diferencias divididas de f con respecto a $x_0, x_1, \ldots, x_n$, se pueden derivar demostrando que P_n tiene la representación

$$P_n(x)=a_0+a_1(x-x_0)+a_2(x-x_0)(x-x_1)+\ldots+a_n(x-x_0)(x-x_1)\ldots+(x-x_{n-1})$$

con constants apropiadas $a_0, a_1, \ldots, a_n$.

Para determinar la primera de estas constantes, a_0, note que si $P_n(x)$ puede escribirse de la forma de la ecuación anterior, entonces evaluando P_n en x_0 deja solamente el término constante a_0; esto es, $a_0 = P_n(x_0) = f(x_0)$.

Similarmente, cuando P_n se evalúa en x_1, los únicos términos distintos de 0 en la evaluación de $P_n(x_1)$ son la constante y el término lineal,

$$f(x_0)+a_1(x_1-x_0) = P_n(x_1) = f(x_1), \text{ así que}$$

$$a_1 = \frac{f(x_1)-f(x_0)}{x_1-x_0}.$$

Aquí introducimos lo que se conoce como notación de diferencia dividida. La diferencia dividida 0 de la función f con respecto a x_i , se denota por $f[x_i]$ y es simplemente la evaluación de f en x_i, $f[x_i]=f(x_i)$. Las diferencias divididas restantes de definen inductivamente; la primera diferencia dividida de f con respecto a x_i y x_{i+1} , se denota por $f[x_i, x_{i+1}]$ y está definida como

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}]-f[x_i]}{x_{i+1}-x_i}.$$

Cuando las (k-1) diferencias divididas

$$f[x_i, x_{i+1},\ldots,x_{i+k-1}] \qquad \text{y} \qquad f[x_i, x_{i+1},\ldots,x_{i+k}]$$

han sido determinadas, la k-ésima diferencia dividida de f relativa a $x_i, x_{i+1},\ldots,x_{i+k}$ está dada por

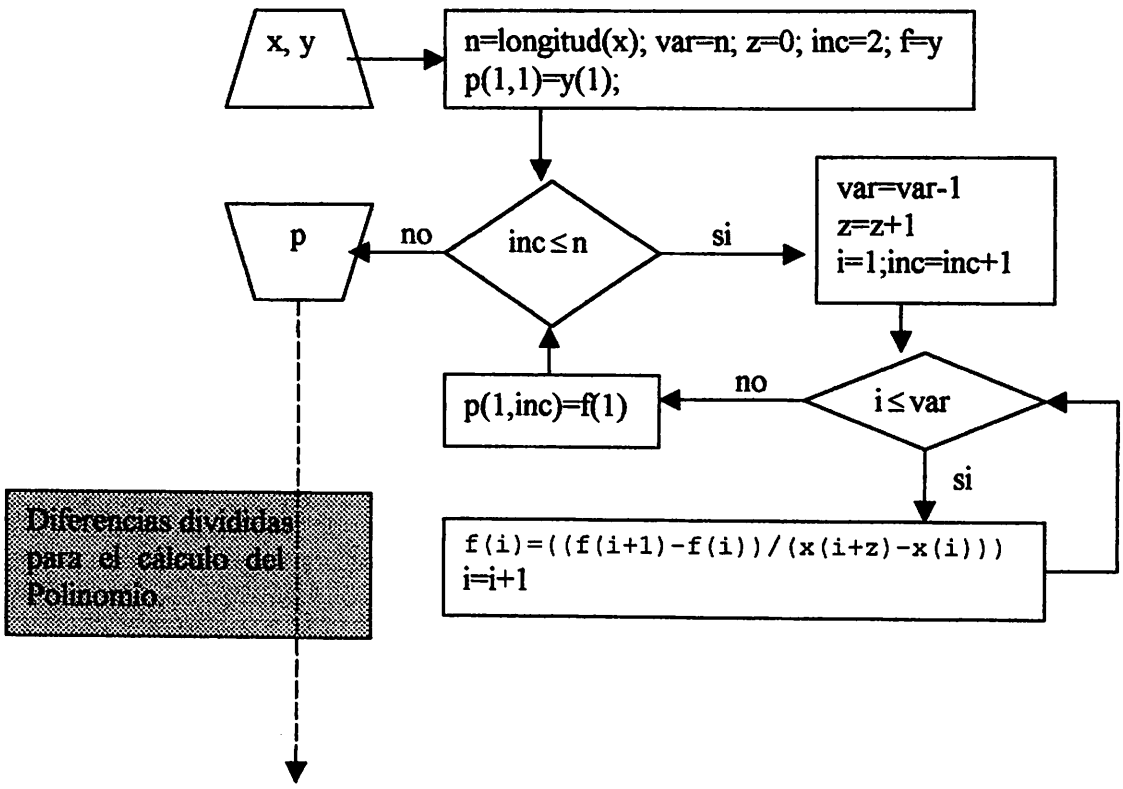
$$f[x_i, x_{i+1},\ldots,x_{i+k-1}, x_{i+k}] = \frac{f[x_i, x_{i+1},\ldots,x_{i+k}] - f[x_i, x_{i+1},\ldots,x_{i+k-1}]}{x_{i+k} - x_i}.$$

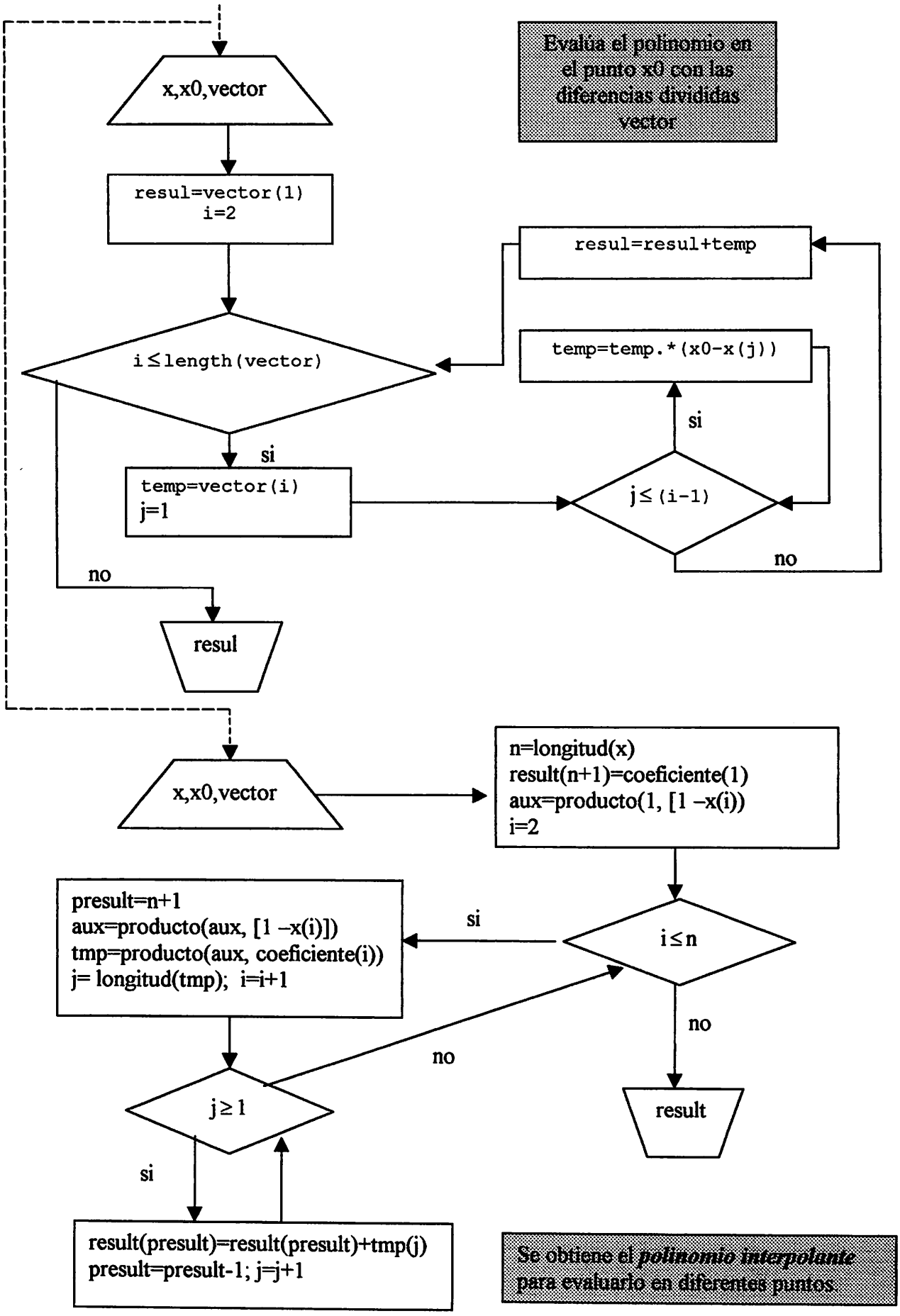
Con esta notación podemos expresar a_1 como $a_1=f[x_0,x_1]$ y el polinomio interpolante queda

$$P_n(x)= f[x_0]+ f[x_0,x_1](x- x_0)+ a_2(x- x_0)(x- x_1)+\ldots + a_n(x- x_0)(x- x_1)\ldots + (x- x_{n-1}).$$

Las otras constantes $a_2, a_3, \ldots, a_n$, en P_n se pueden obtener consecutivamente de una manera similar a la evaluación de a_0 y a_1, pero las manipulaciones algebraicas se vuelven tediosas. Como puede esperarse de la evaluación de a_0 y a_1, las constantes requeridas son: $a_k=f[x_0,x_1,x_2,\ldots,x_n]$, para cada k=0,1,...,n ; así, P_n puede rescribirse como $P_n(x)=f[x_0]+f[x_0,x_1](x- x_0)+f[x_0,x_1,x_2](x-x_0)(x-x_1)+\ldots+f[x_0,..,x_n](x- x_0)(x- x_1)\cdot\ldots\cdot(x-x_{n-1})$ conociéndose como la fórmula de diferencia dividida interpolante de Newton.

DIAGRAMA DE FLUJO DE DIFERENCIAS DIVIDIDAS




Evalúa el polinomio en el punto x0 con las diferencias divididas vector
x,x0,vector
resul=vector(1)
i=2
i ≤ length(vector)
si
temp=vector(i)
j=1
j ≤ (i-1)
si
temp=temp.*(x0-x(j))
resul=resul+temp
no
no
resul
x,x0,vector
n=longitud(x)
result(n+1)=coeficiente(1)
aux=producto(1, [1 –x(i))
i=2
i ≤ n
si
presult=n+1
aux=producto(aux, [1 –x(i)])
tmp=producto(aux, coeficiente(i))
j= longitud(tmp); i=i+1
no
result
j ≥ 1
no
si
result(presult)=result(presult)+tmp(j)
presult=presult-1; j=j+1
Se obtiene el *polinomio interpolante* para evaluarlo en diferentes puntos

1.2.4. DIFERENCIAS PROGRESIVAS DE NEWTON

Este método necesita que los puntos a interpolar estén igualmente espaciados. Diremos que n+1 puntos $(x_0, f(x_0))$, $(x_1, f(x_1))$, ... , $(x_n, f(x_n))$ están igualmentes espaciados si tienen sus abscisas igualmente espaciadas, es decir,

$x_1=x_0+h$, $x_2=x_0+2h$, ... , $x_n=x_0+nh$. A h lo llamaremos longitud de paso.

Se definen las diferencias progresivas primeras o diferencias hacia delante, por la siguiente expresión:

$$\Delta f_k = f_{k+1} - f_k$$

Al símbolo Δ se le denomina operador de las diferencias hacia delante o progresivas.

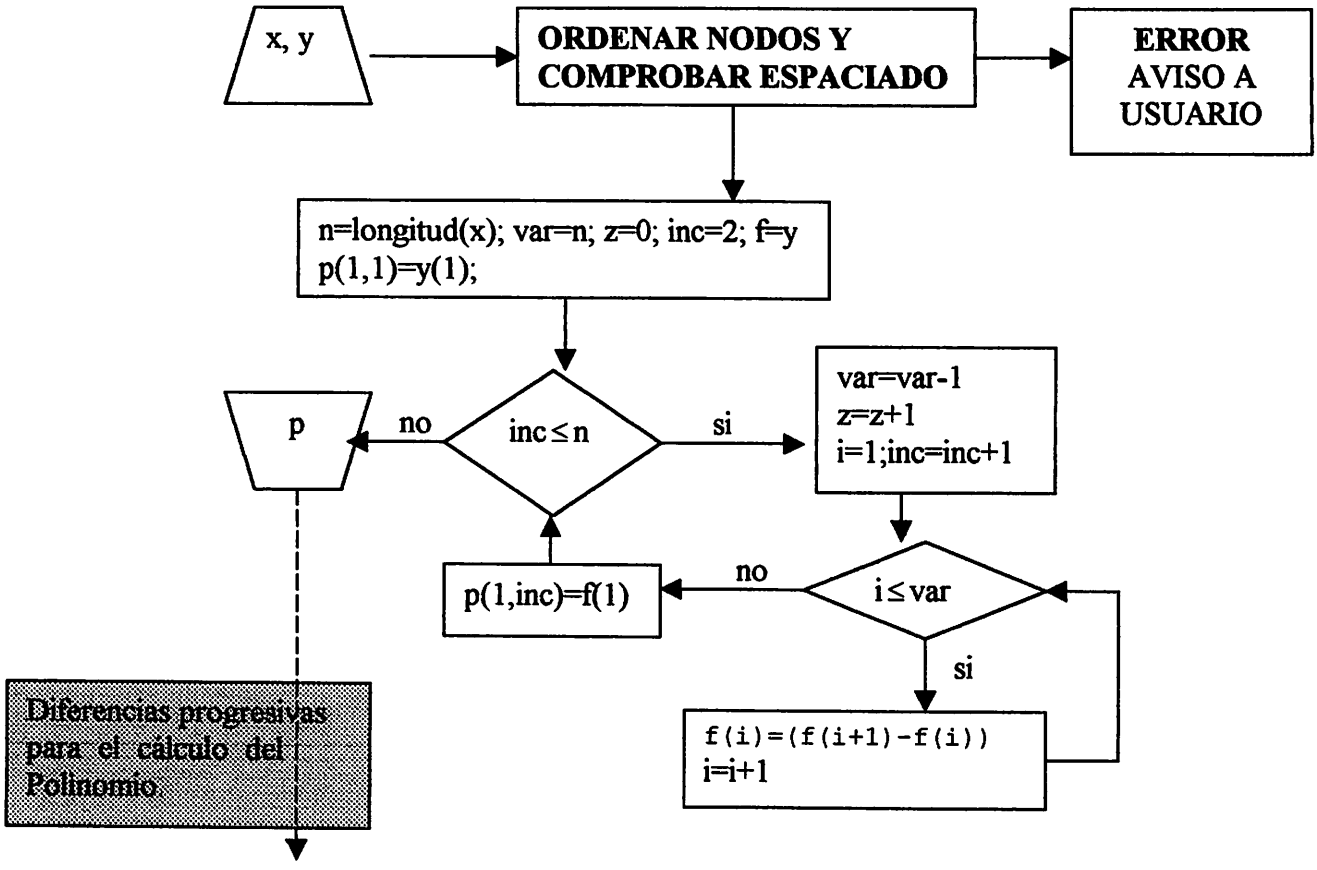
De forma análoga, las diferencias progresivas segundas se definen por

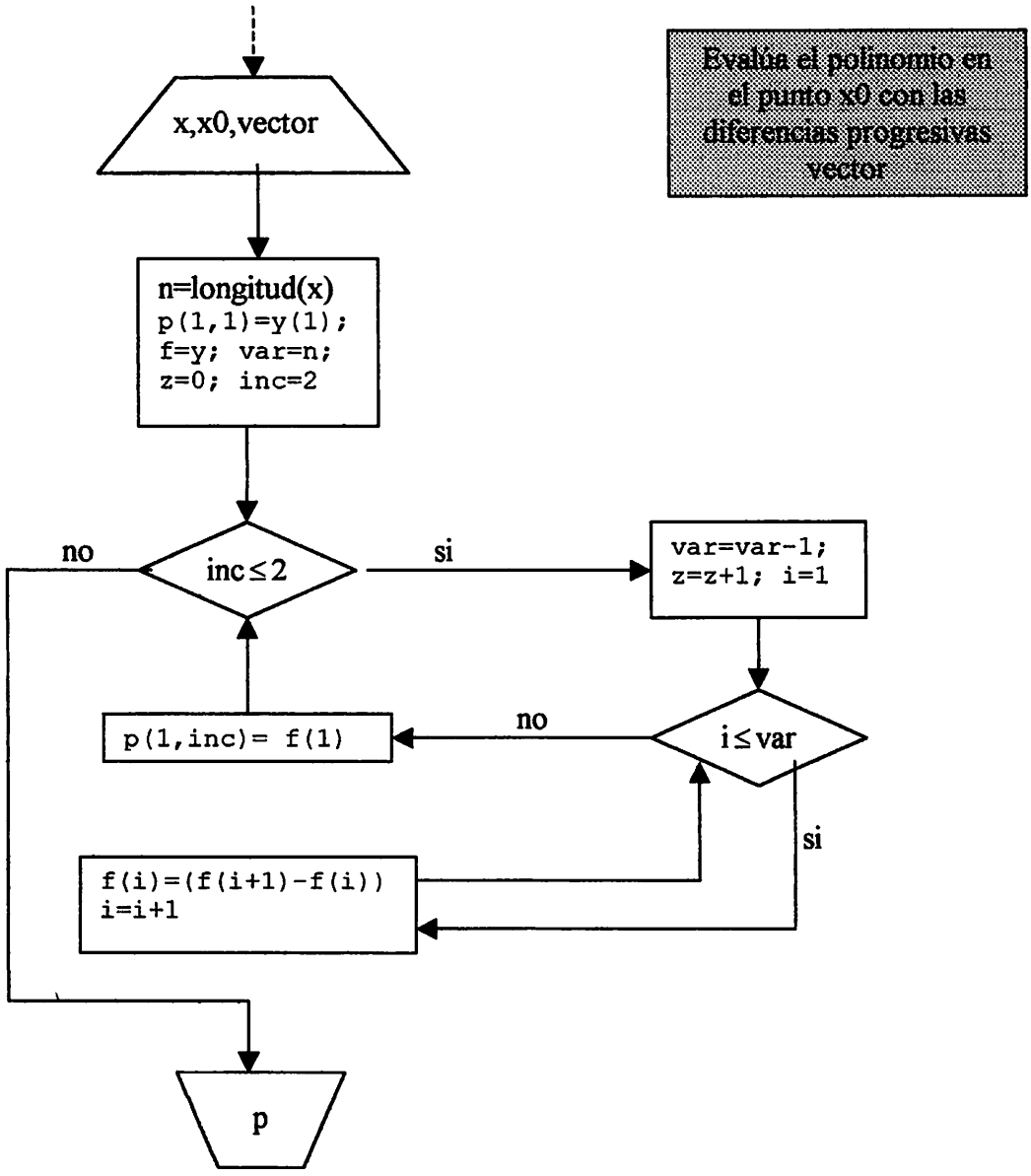
$$\Delta^2 f_k = \Delta(\Delta f_k) = \Delta(f_{k+1} - f_k) = \Delta f_{k+1} - \Delta f_k .$$

En general,

$$\Delta^n f_k = \sum_{i=0}^{n} (-1)^i \cdot \binom{n}{i} \cdot f_{n-1+k} .$$

DIAGRAMA DE FLUJO DE DIFERENCIAS PROGRESIVAS




Evalúa el polinomio en el punto x0 con las diferencias progresivas vector
x,x0,vector
n=longitud(x)
p(1,1)=y(1);
f=y; var=n;
z=0; inc=2
no
inc≤2
si
var=var-1;
z=z+1; i=1
p(1,inc)= f(1)
no
i≤var
si
f(i)=(f(i+1)-f(i))
i=i+1
p

2. DERIVACIÓN E INTEGRACIÓN

2.1.INTRODUCCIÓN

Muchos problemas de la vida real se reducen a la resolución de una derivada o una integral . Debido a las muchas aplicaciones en las que se usan las derivadas y las integrales de funciones, es de esperarse que las aproximaciones de estos conceptos sean de interés.

En el apartado de interpolación mencionamos que una de las razones para usar la clase de polinomios algebraicos para aproximar a un conjunto arbitrario de datos, es el hecho de que, dada una función continua definida en un intervalo cerrado, existe un polinomio que está arbitrariamente cerca de la función en cada punto del intervalo. Una propiedad adicional que posee esta clase de polinomios es que sus derivadas e integrales son bastante fáciles de obtener y de evaluar. No debe ser sorprendente, entonces, encontrar que la mayoría de los procedimientos para aproximar integrales o derivadas se inicien aproximando a la función con polinomios algebraicos.

2.2. DERIVACIÓN NUMÉRICA

2.2.1.DIFERENCIAS PROGRESIVAS Y REGRESIVAS PRIMERAS

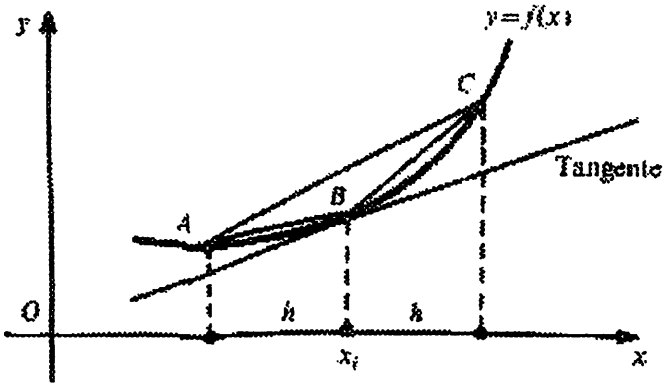
Se define como derivada f' de una función y=f(x) en un punto x_i a la expresión

$$f'(x_i) = \lim_{h \to 0} \frac{f(x_i + h) - f(x_i)}{h}$$

de acuerdo con esta fórmula, una aproximación a la derivada en un punto cualquiera de x_i puede conseguirse utilizando un valor pequeño para h. Así se obtiene la fórmula aproximada

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i)}{h}$$

denominada primera diferencia progresiva, que representa la pendiente de la recta BC de la siguiente figura.



Otra aproximación a la derivada se consigue empleando la primera diferencia regresiva

$$f'(x_i) \approx \frac{f(x_i) - f(x_i - h)}{h}$$

que proporciona el valor de la pendiente de la recta AB.

2.2.2.DIFERENCIA CENTRAL

Por último, otra aproximación útil es la obtenida empleando la diferencia central, es decir,

$$f'(x_i) \approx \frac{f(x_i + h) - f(x_i - h)}{2h}$$

que corresponde a la pendiente de la recta AC.

La mejor aproximación se obtiene empleando esta última fórmula.

2.2.3. EXTRAPOLACIÓN DE RICHARDSON

Este algoritmo se emplea para generar resultados de alta precisión usando fórmulas de bajo orden.

Para examinar la técnica de extrapolación, suponga N(h) es una fórmula que produce aproximación $O(h^2)$ para un valor desconocido N. Además que la forma para el error en las aproximaciones de N(h) para N se puede expresar como $M=N(h)+k_1h^2+O(h^4)$, donde k_1 es constante. Reemplazando h por h/2 en la ecuación anterior obtenemos una nueva, y probablemente más precisa, aproximación de N(h/2) que satisface

$$M = N\left(\frac{h}{2}\right) + k_1 \frac{h^2}{4} + O\left(\left(\frac{h}{2}\right)^4\right)$$

Multiplicando por 4 y restando $M=N(h)+k_1h^2+O(h^4)$ da:

$$M = \frac{4N\left(\frac{h}{2}\right) - N(h) + O(h^4)}{3}$$

Para facilitar la discusión, definiremos $N_1(h)=N(h)$ y

$$N_2(h) = \frac{4N_1\left(\frac{h}{2}\right) - N_1(h)}{3}.$$

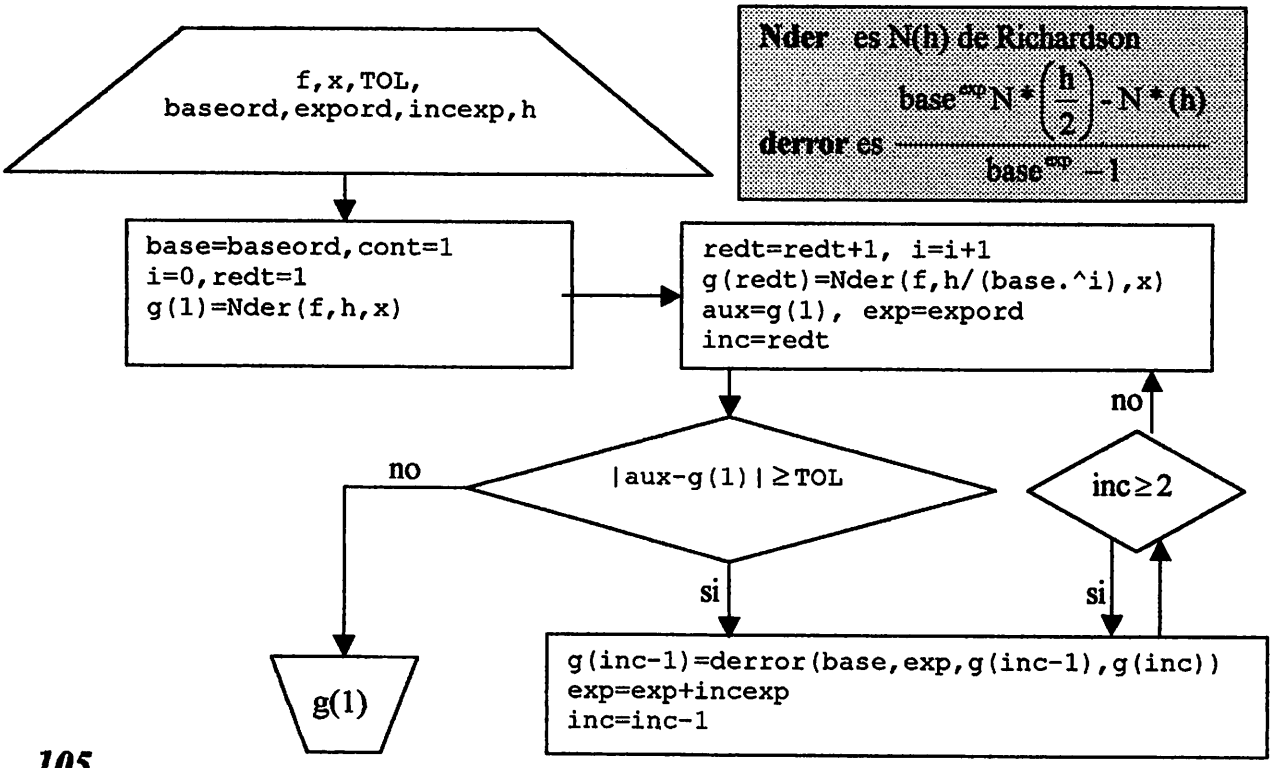
De forma general, se puede expandir a m de estas columnas siempre y cuando la forma del error de la aproximación N(h) para N puede ser expresada con

$$M = N(h) + \sum_{j=1}^{m-1} k_j h^{2j} + O\left(h^{2m}\right)$$

Para alguna colección de constantes k_j. Las aproximaciones de $O(h^{2j})$ son generadas de manera recursiva de la fórmula

$$N_j(h) = \frac{4^{j-1} N_{j-1}\left(\frac{h}{2}\right) - N_{j-1}(h)}{4^{j-1} - 1} \quad \text{para j=2, 3, ... , m.}$$

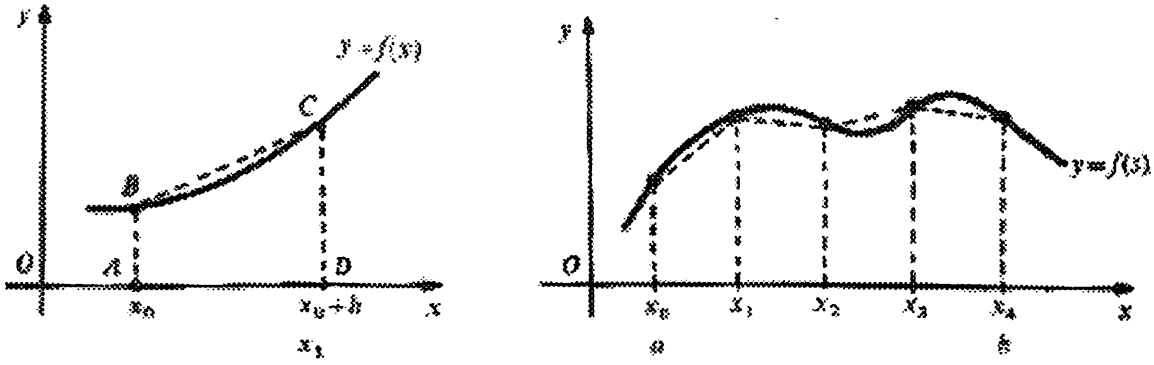
DIAGRAMA DE FLUJO DE LA DERIVADA



2.3. INTEGRACIÓN NUMÉRICA

2.3.1. REGLA DEL TRAPECIO

La función f(x) se aproxima por la cuerda de la curva dentro del intervalo [x_0, x_0+h]. Se trata de una aproximación lineal. El arco BC se sustituye por la cuerda BC.



El área para un solo intervalo valdrá:

$$\int_{x_0}^{x_0+h} f(x)dx \approx \text{Área trapecio ABCD} = \frac{1}{2}h(f_0 + f_1)$$

para n=4 será $\int_a^b f(x)dx \approx \frac{1}{2}h(f_0 + 2f_1 + 2f_2 + 2f_3 + f_4) = h\left[\frac{1}{2}E + I\right]$

representándose por E la suma de las ordenadas extremas f_0 y f_4, y por I la suma de las restantes ordenadas intermedias f_1, f_2 y f_3. La fórmula anterior puede aplicarse al caso general de n intervalos de anchura h.

El error máximo cometido al emplear la regla del trapecio o cota del error absoluto es de orden de h^2.

2.3.2. ALGORITMO DE ROMBERG

El cálculo numérico de integrales basada en la regla del trapecio puede aún mejorarse aplicando el algoritmo de Romberg. Este produce una convergencia hacia el verdadero valor de forma más rápida y con menos operaciones.

El procedimiento de Romberg consiste en la elaboración de una tabla triangular a partir de la tabla definitoria de la función y=f(x) a integral. La primera columna de la tabla triangular consta de los valores resultantes de aplicar la regla del trapecio: T_n, T_{2n}, T_{4n}, ...

La segunda columna está formada por los valores $T_n^{(1}$, $T_{2n}^{(1}$, ... obtenidos de la columna anterior mediante las expresiones

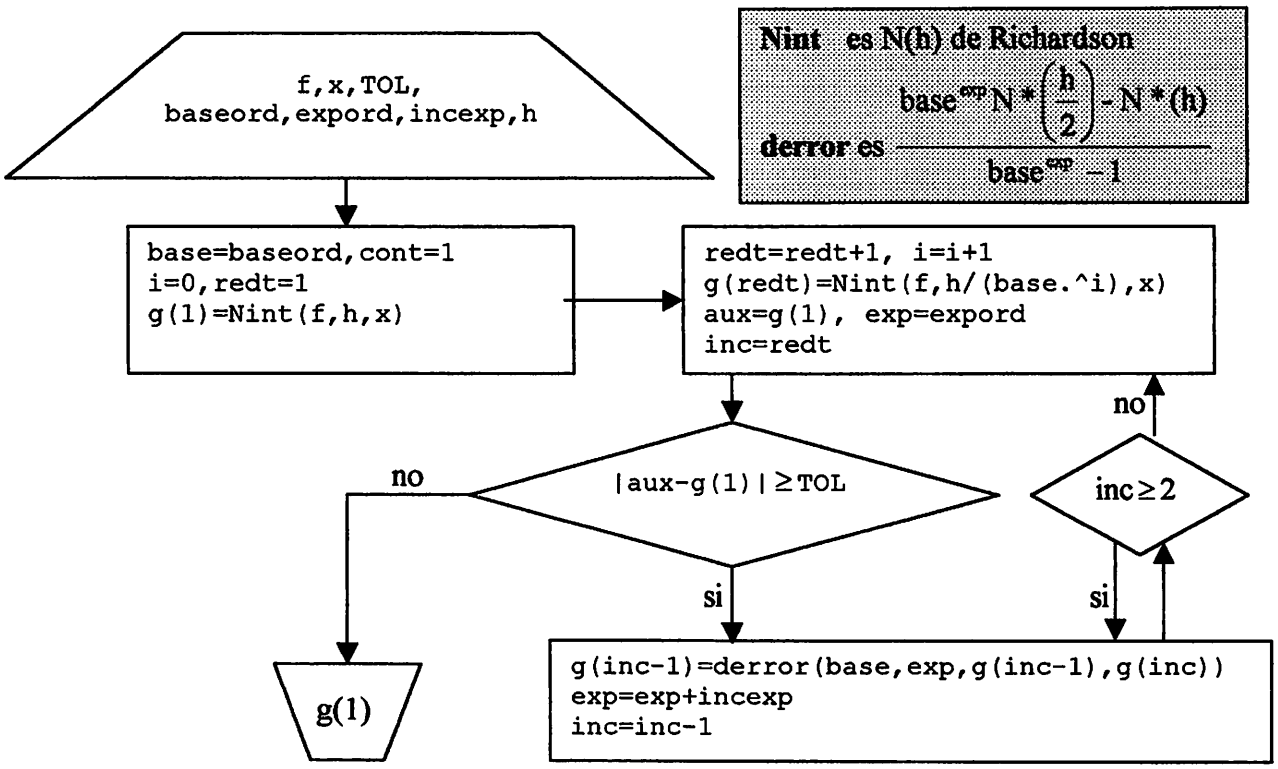
$$T_n^{(1} = \frac{4T_{2n} - T_n}{3} \quad ; \quad T_{2n}^{(1} = \frac{4T_{4n} - T_{2n}}{3} \quad ; \ \ldots$$

En general, cada elemento $T_n^{(m}$ de la columna (m-1)-enésima, vale:

$$T_n^{(m} = \frac{4T_{2n}^{\ (m-1} - T_n^{\ (m-1}}{4^{m-1}}$$

lo que significa que la aplicación sucesiva de cuatro veces la regla del trapecio, T_n, T_{2n}, T_{4n}, T_{8n}, constituyen la base para obtener la aproximación de Romberg, $T_n^{(3}$.

DIAGRAMA DE FLUJO DE LA INTEGRAL



2.4. EVALUACIÓN DE LOS MÉTODOS

Para evaluar todos los métodos estudiados en esta practica vamos a utilizar las siguientes funciones $f(x) = x\cos x$, $g(x) = sen(x) + \cos(x)$ y $h(x) = 1/(1+12x^2)$.

En un primer paso aproximaremos estas funciones a partir de una serie de puntos iniciales a través de los diferentes algoritmos de interpolación, y posteriormente pasaremos a derivarlas e integrarlas.

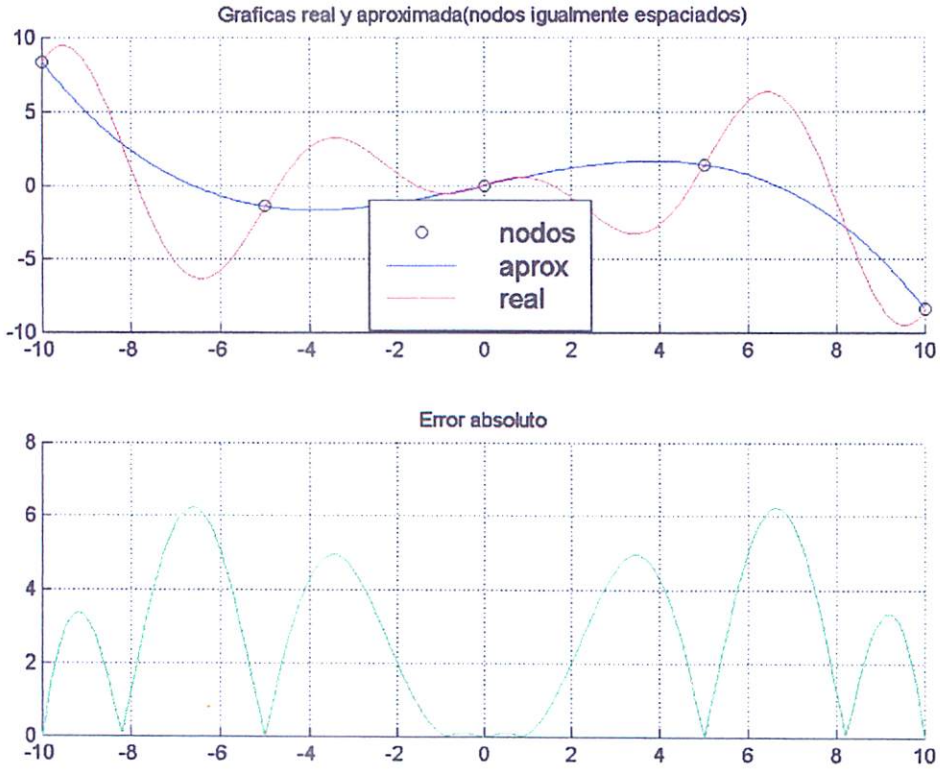
2.4.1. COMPARATIVAS . INTERPOLACIÓN

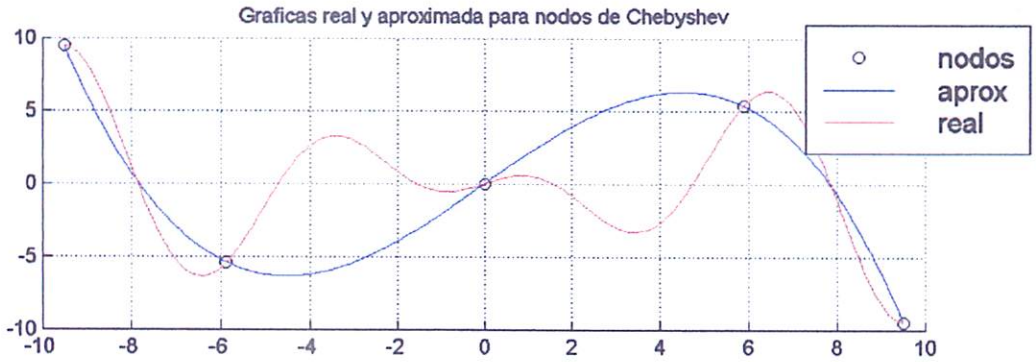
Teniendo en cuenta que todos los métodos ante los mismos nodos deben producir el mismo polinomio interpolador, para no ser reiterativos, se va a comparar únicamente el Método de Diferencias Divididas con nodos igualmente espaciados y con nodos de Chebyshev para las distintas funciones.

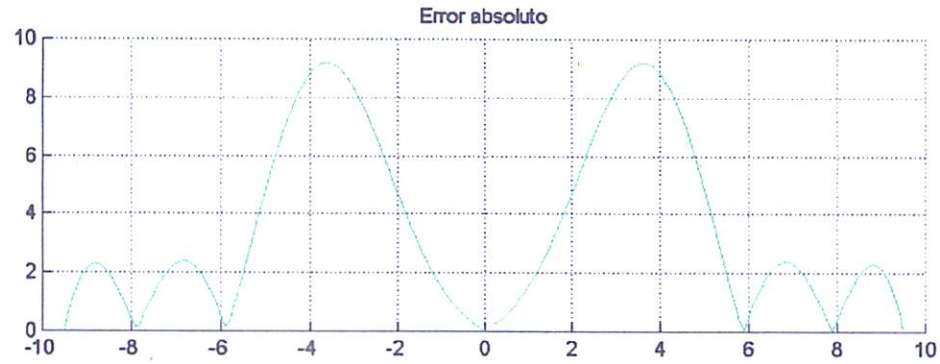
CON ERROR ABSOLUTO

• ***Gráfica de f(x) y su error absoluto en el intervalo [-10, 10] y con 5, 10, 15 y 20:***

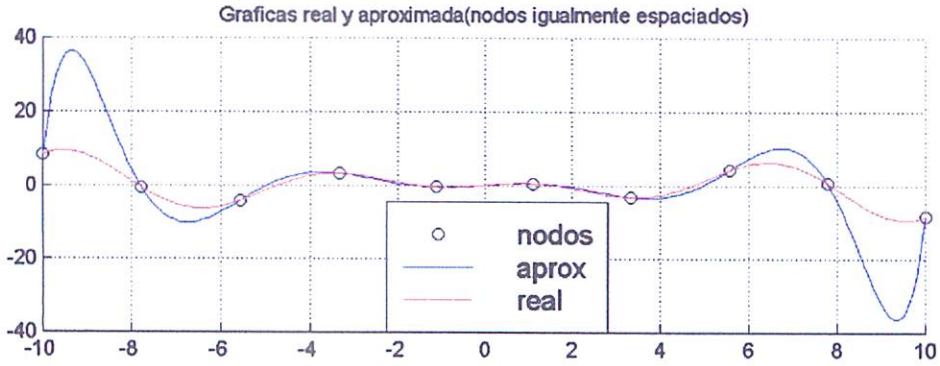
- *Para 5 nodos:*

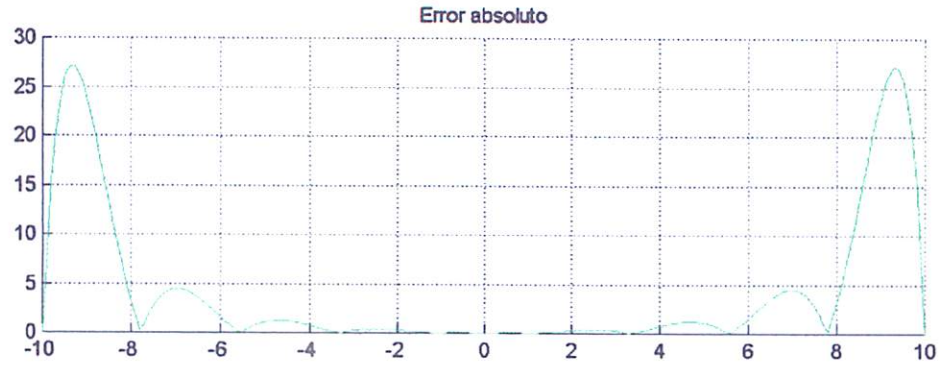


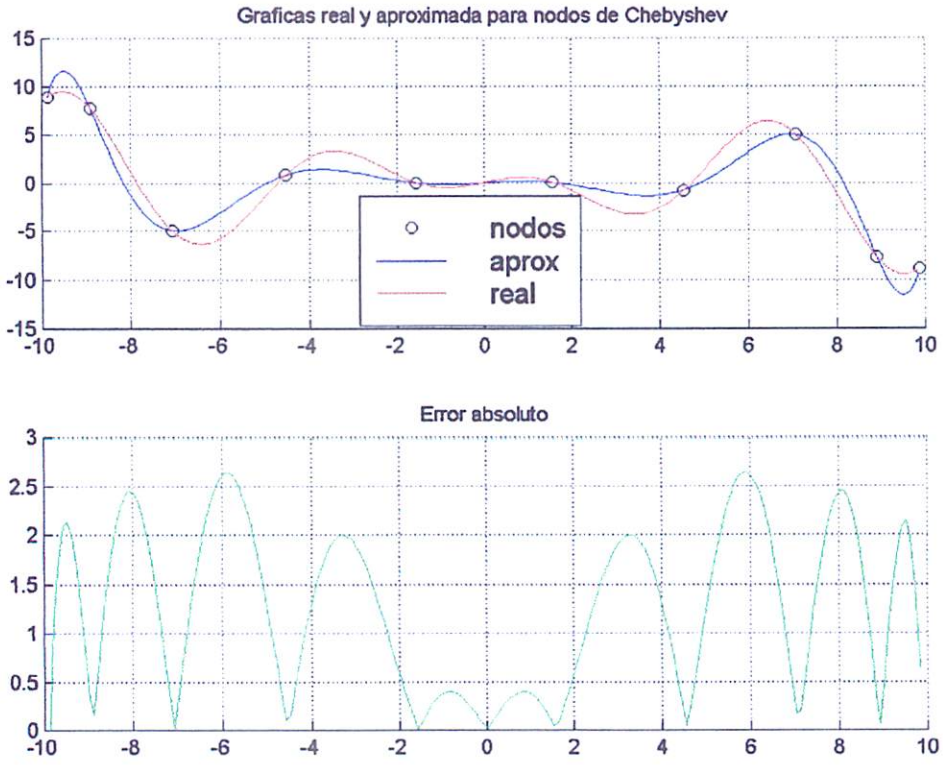




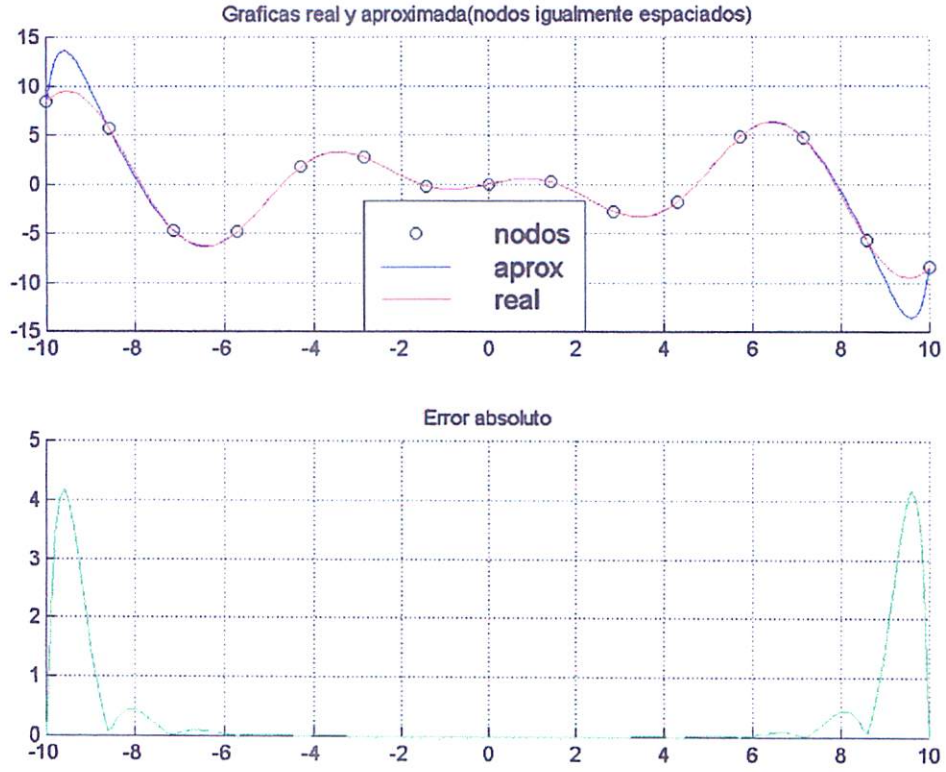
- *Para 10 nodos:*

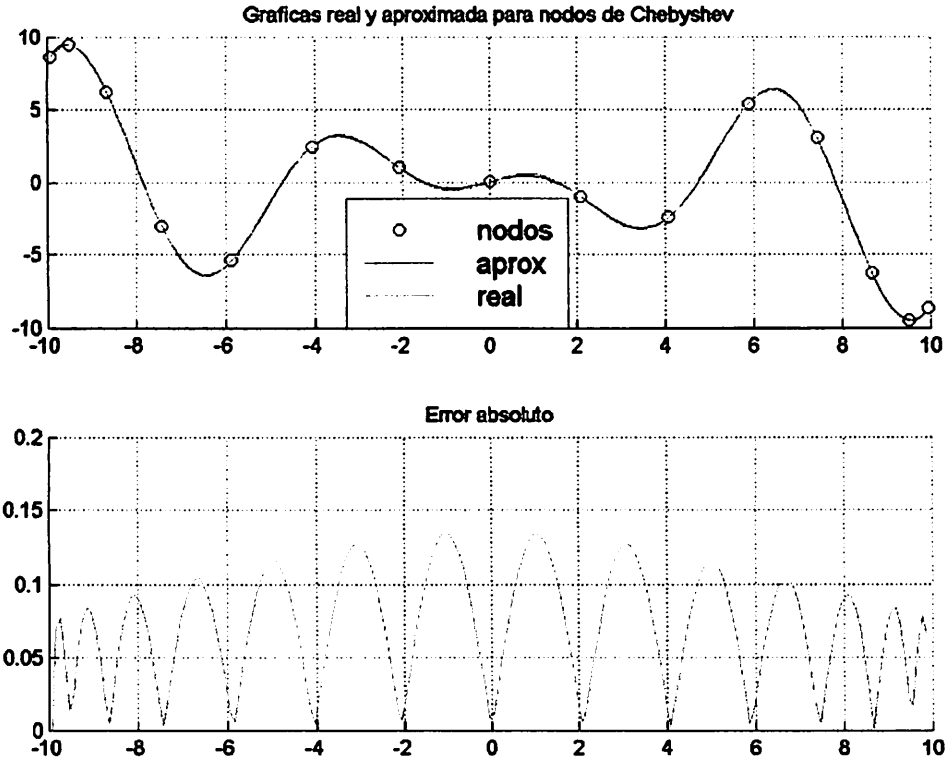




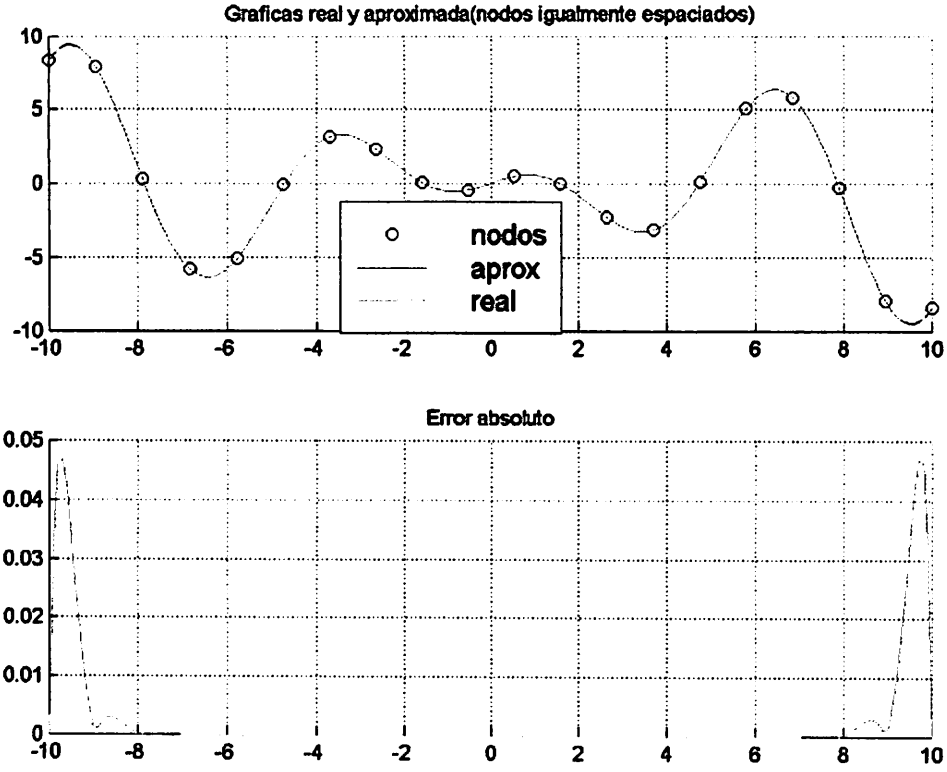


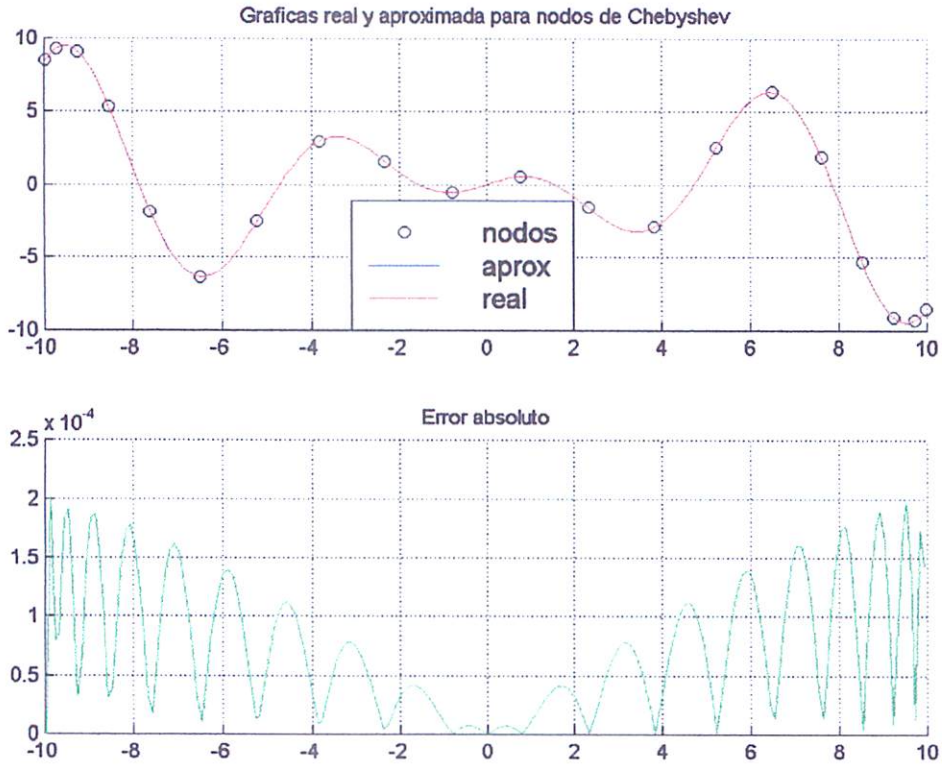
- *Para 15 nodos:*





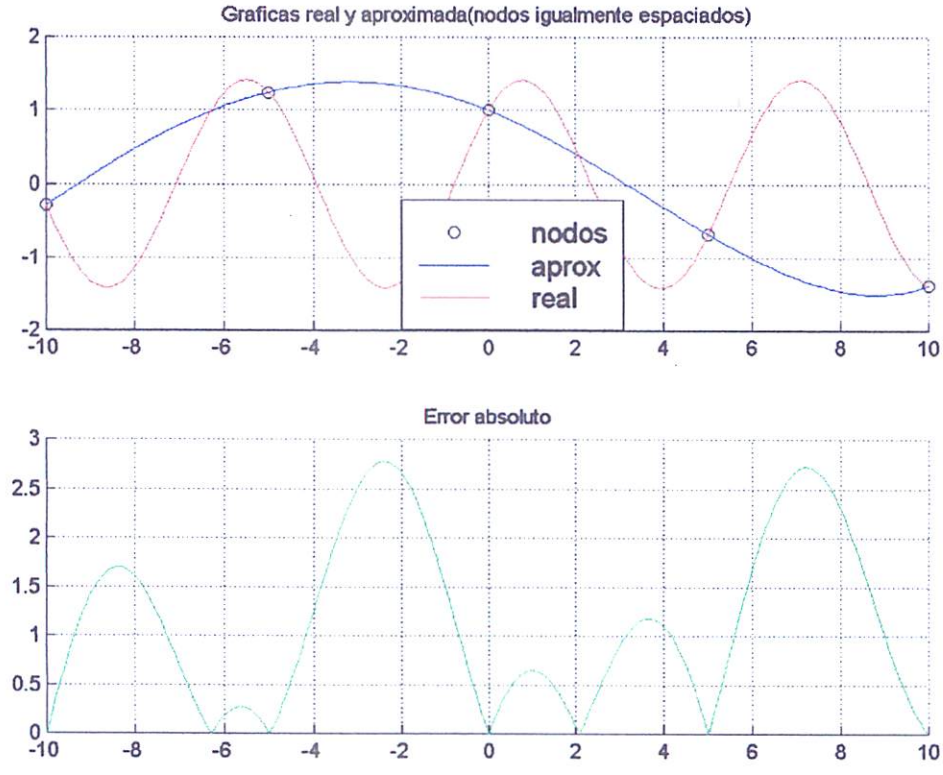
-Para 20 nodos:

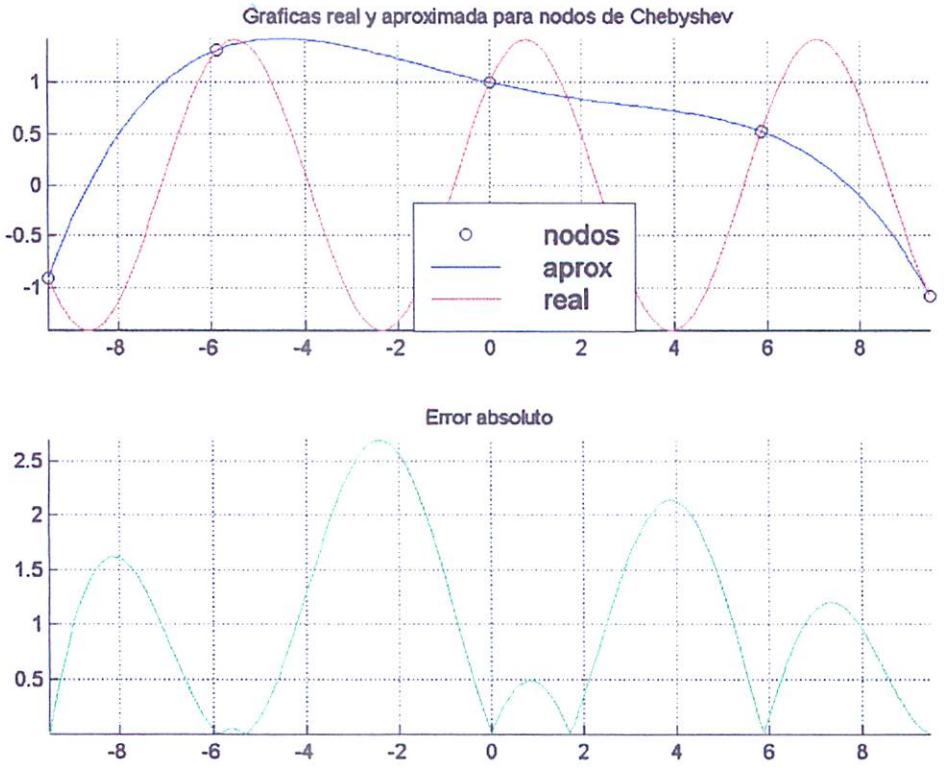




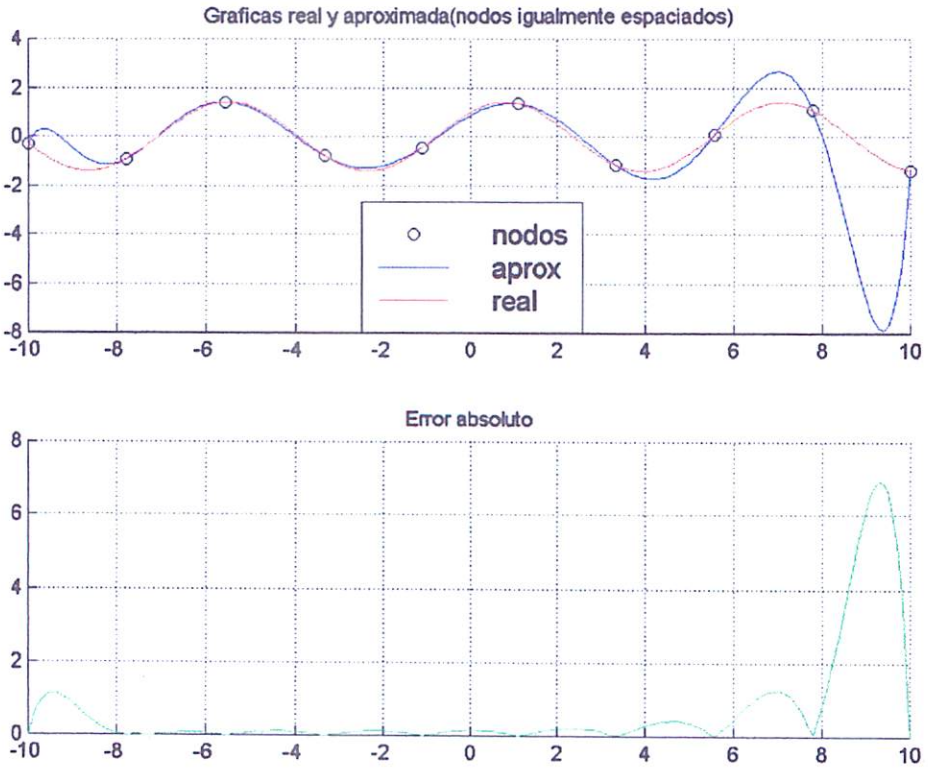
• ***Gráfica de g(x) y su error absoluto en el intervalo [-10, 10] y con 5, 10, 15 y 20:***

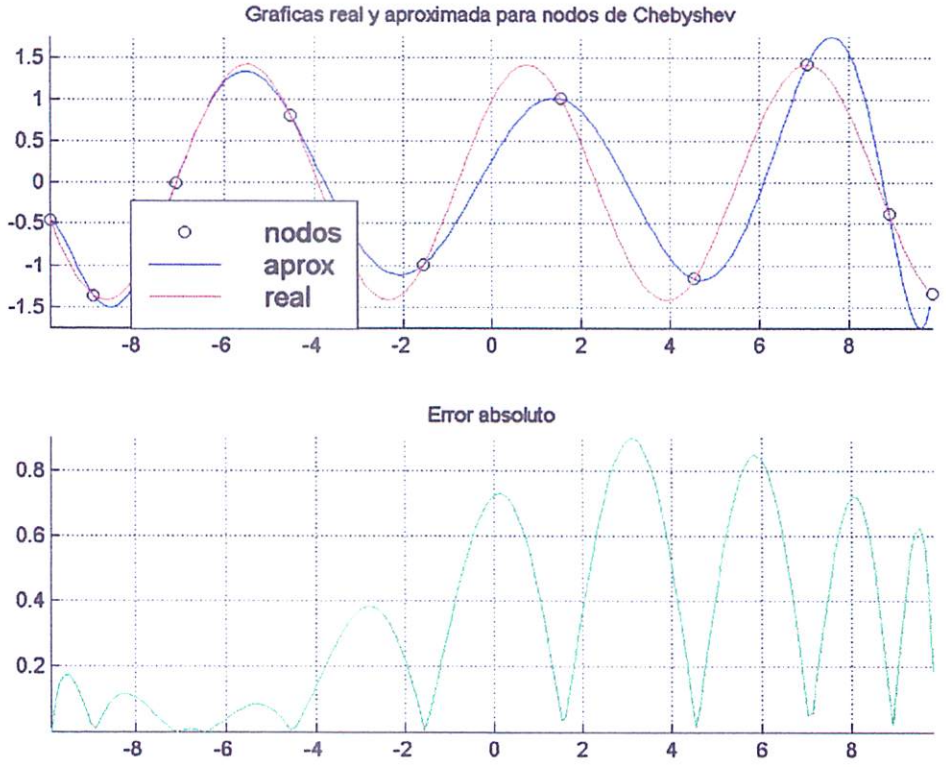
-Para 5 nodos:



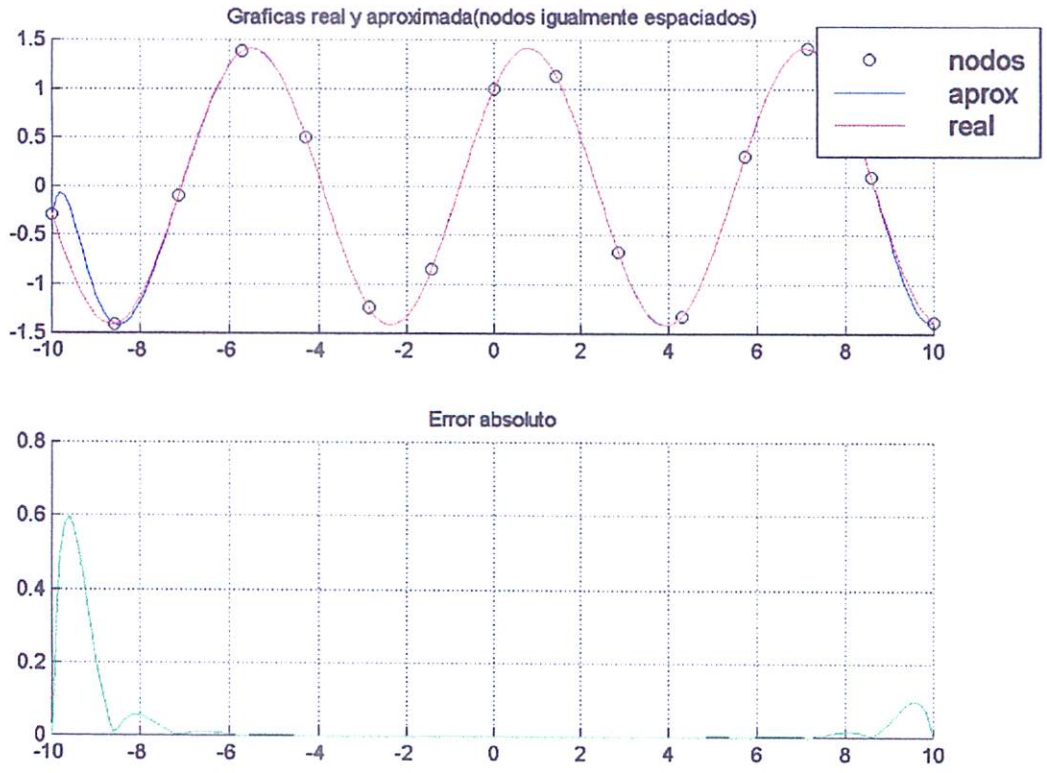


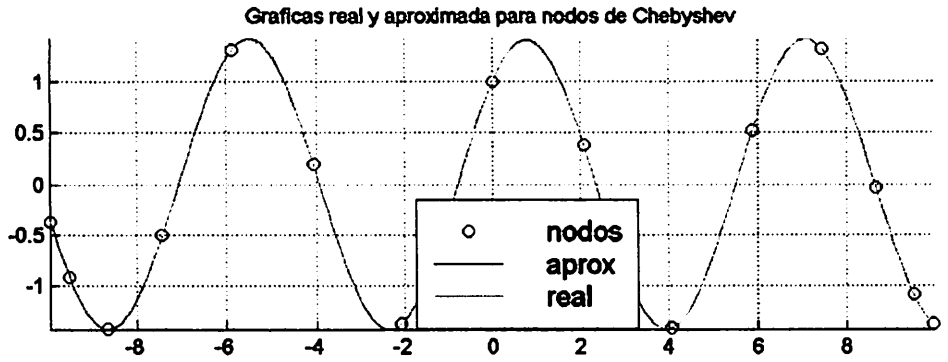
-Para 10 nodos:

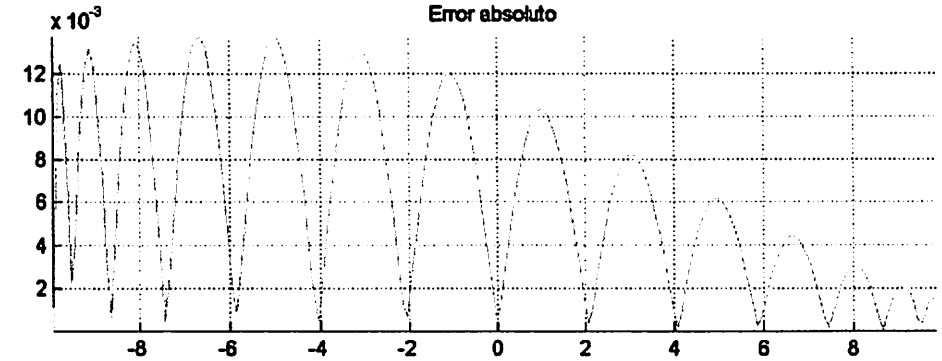




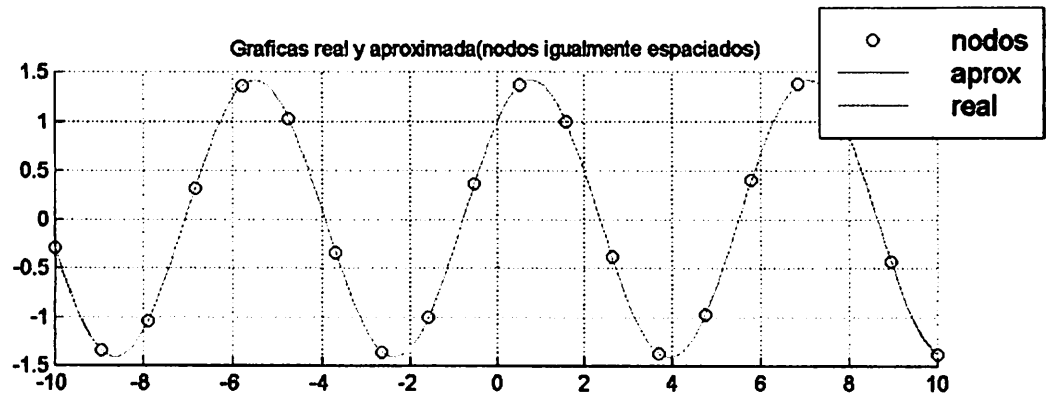
-Para 15 nodos:

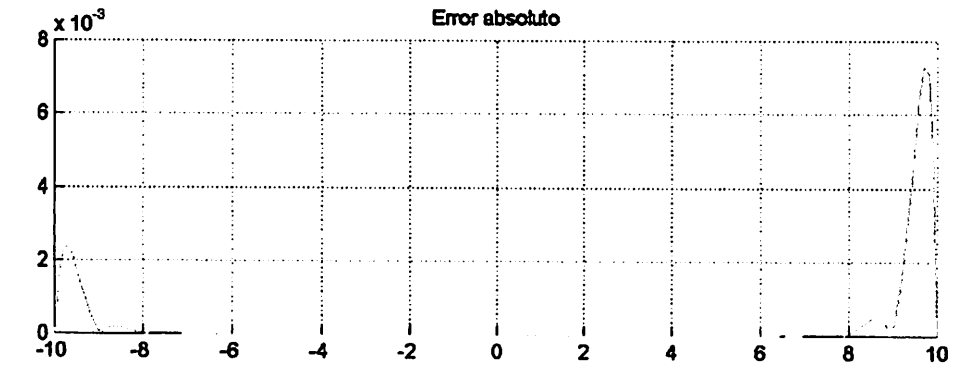


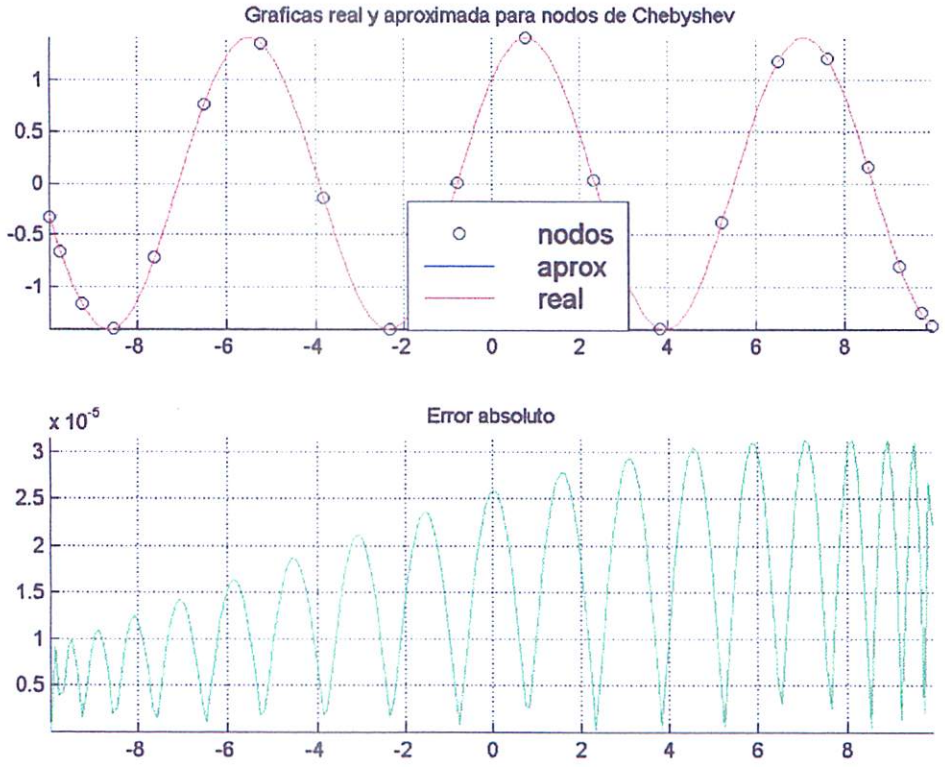




-Para 20 nodos:

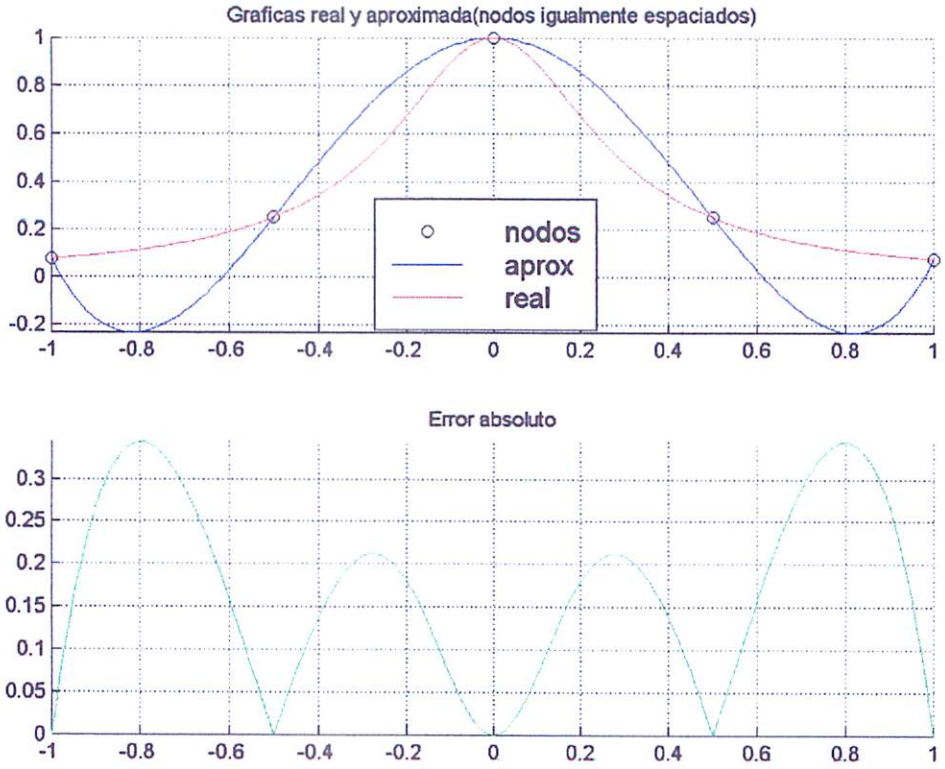


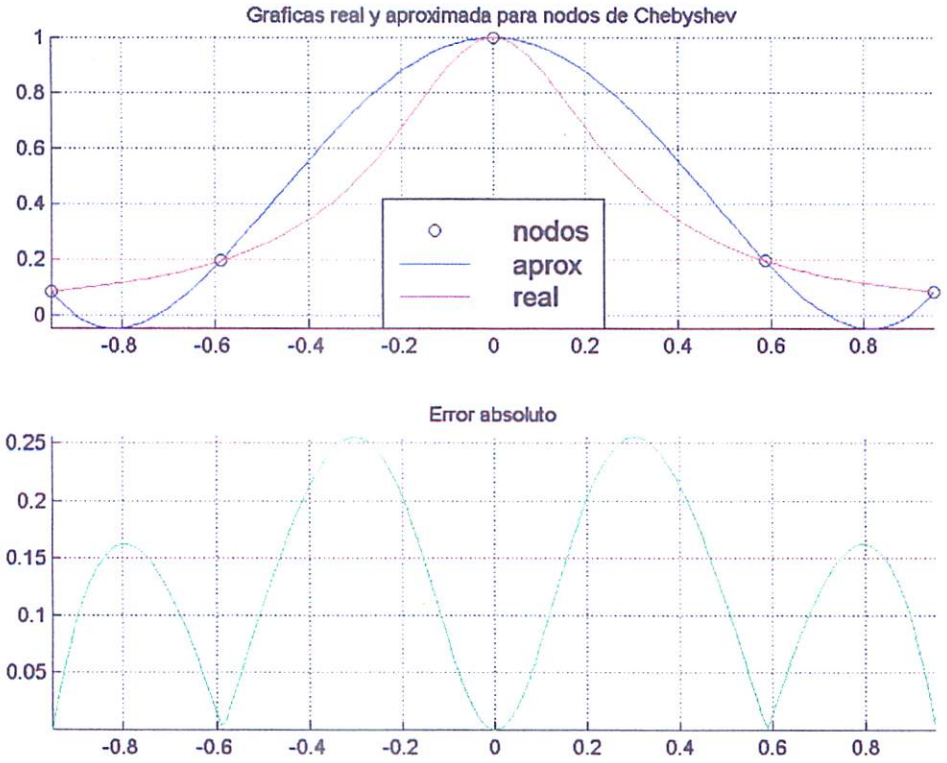




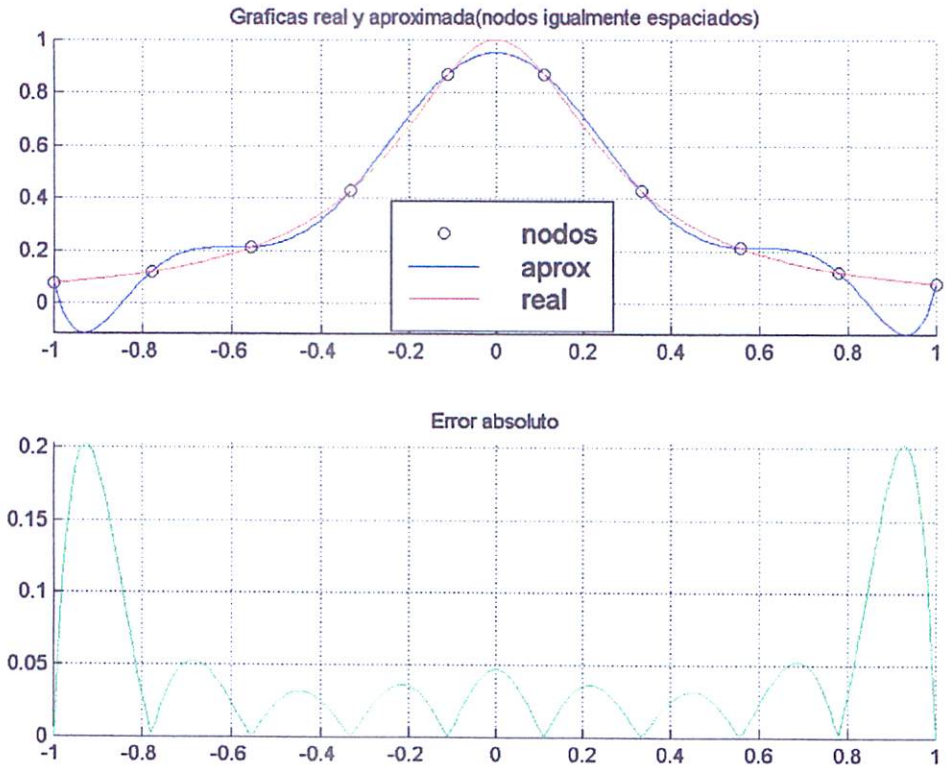
• ***Gráfica de h(x) y su error absoluto en el intervalo [-1, 1] y con 5, 10, 15 y 20:***

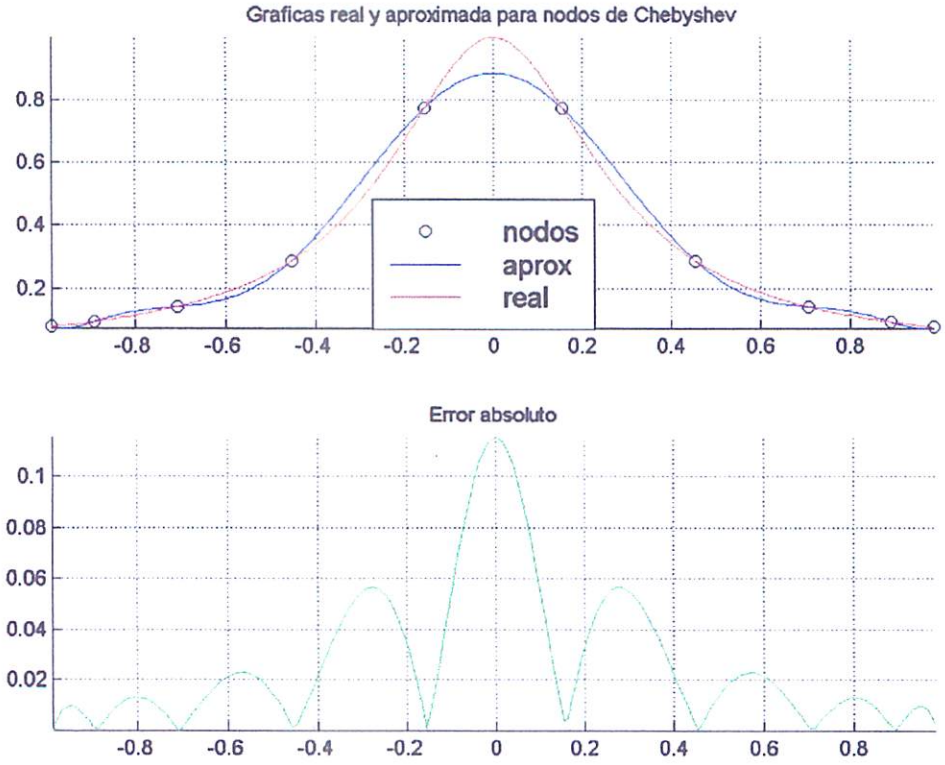
-Para 5 nodos:



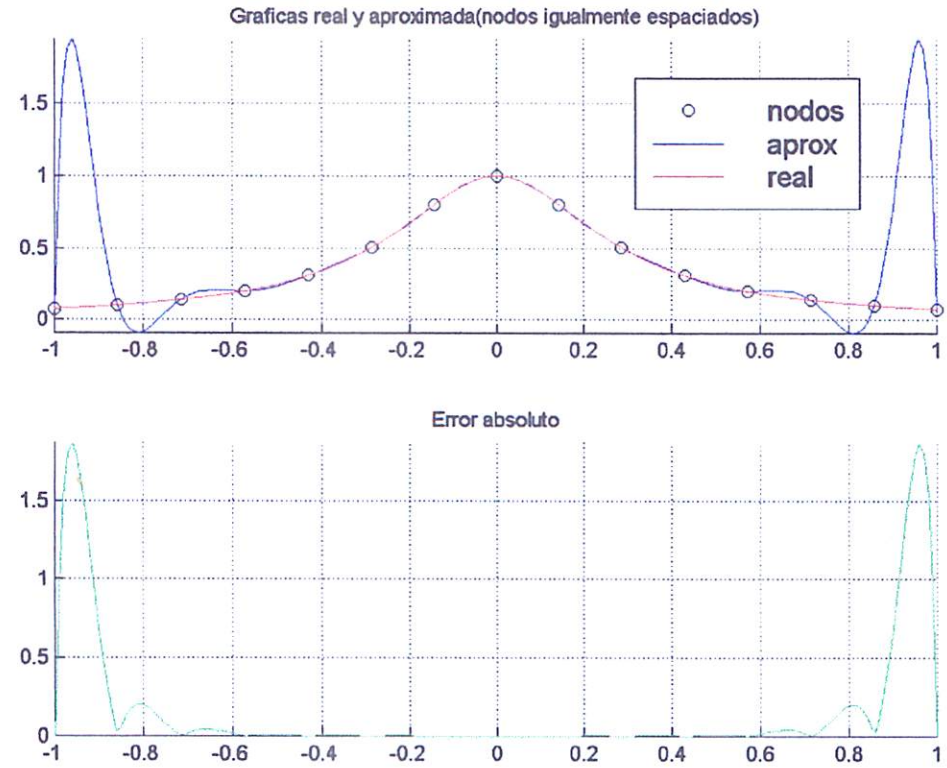


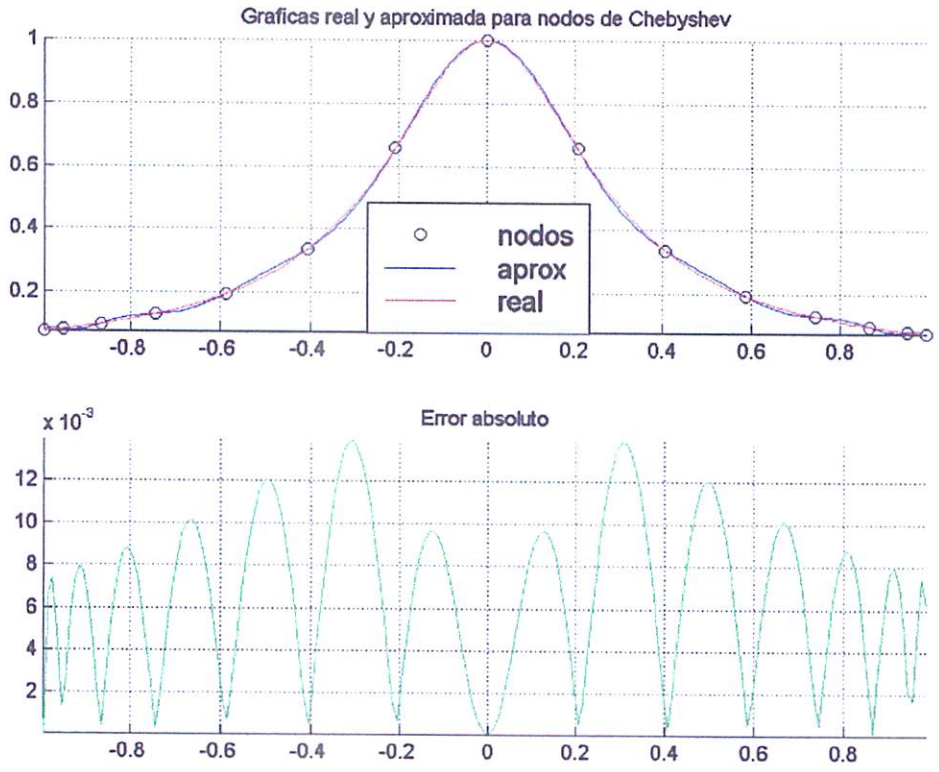
-Para 10 nodos:



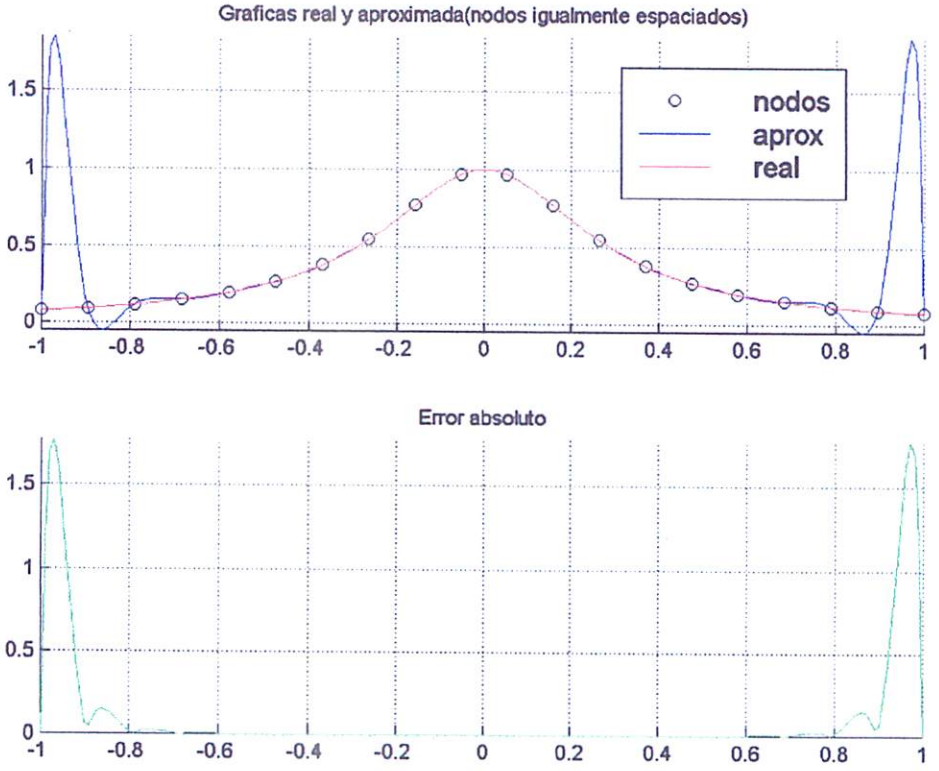


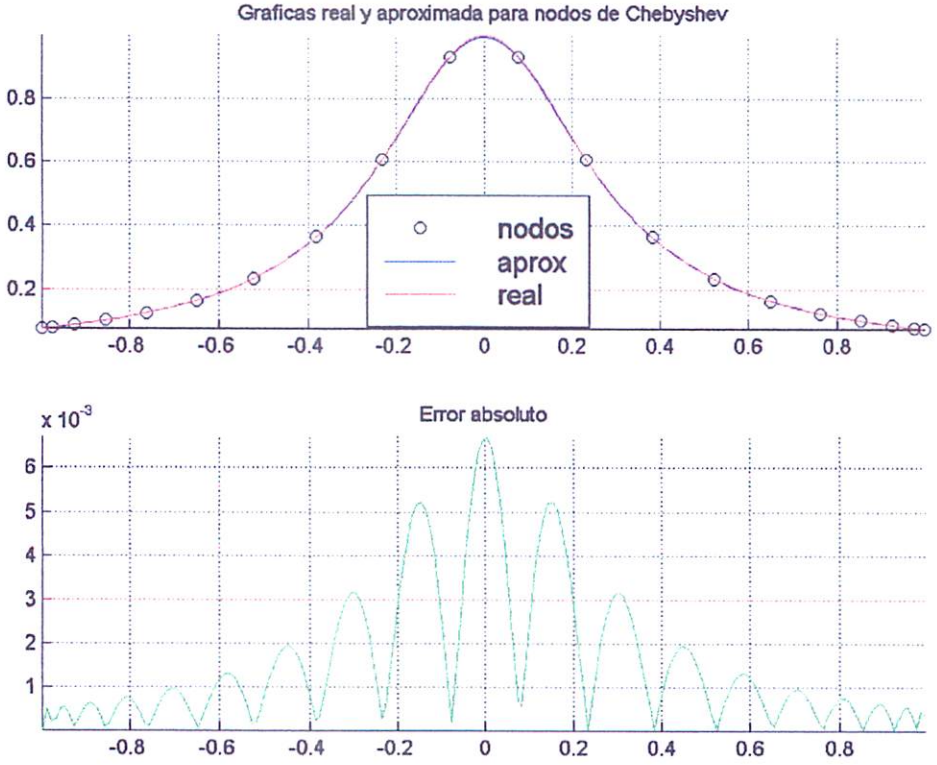
-Para 15 nodos:





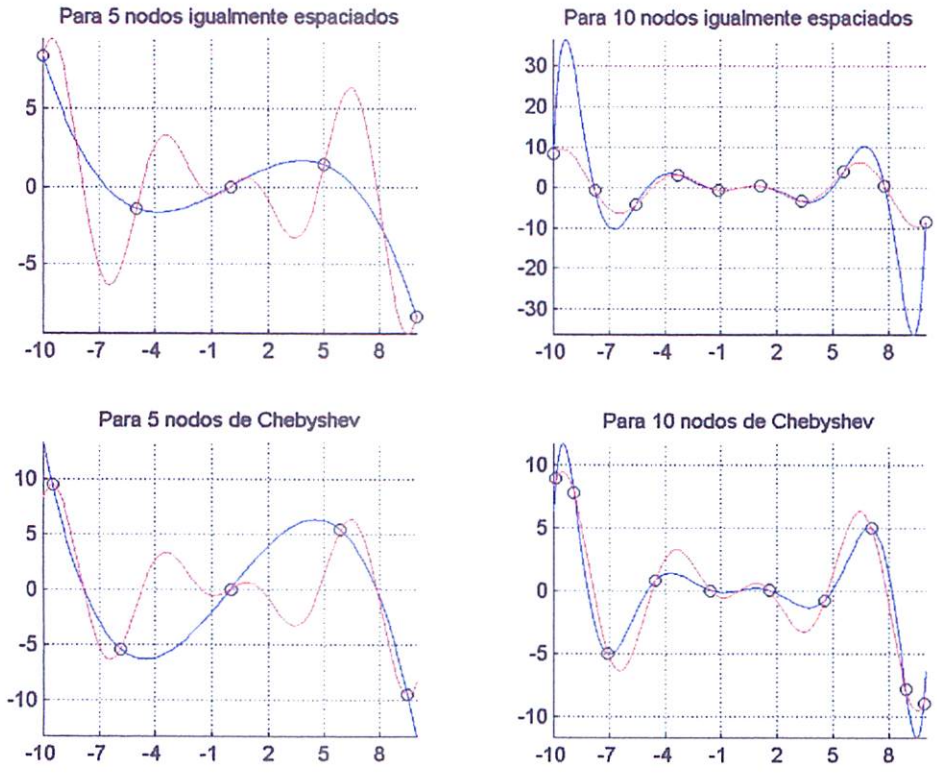
-Para 20 nodos:





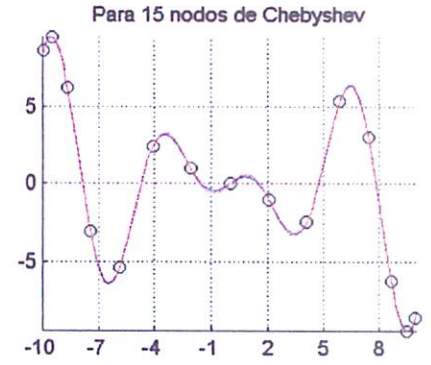
ERROR DISCRETO

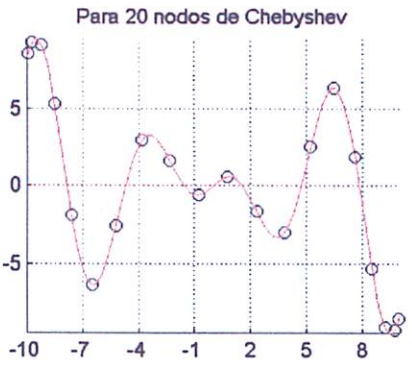
- Para f(x):



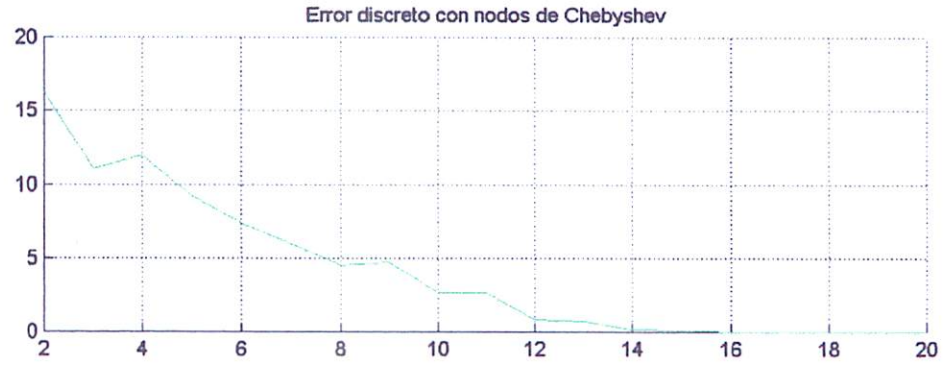

Para 15 nodos igualmente espaciados
10
5
0
-5
-10
-10 -7 -4 -1 2 5 8


Para 20 nodos igualmente espaciados
5
0
-5
-10 -7 -4 -1 2 5 8


Para 15 nodos de Chebyshev
5
0
-5
-10 -7 -4 -1 2 5 8


Para 20 nodos de Chebyshev
5
0
-5
-10 -7 -4 -1 2 5 8


Error discreto con nodos igualmente espaciados
30
25
20
15
10
5
0
2 4 6 8 10 12 14 16 18 20

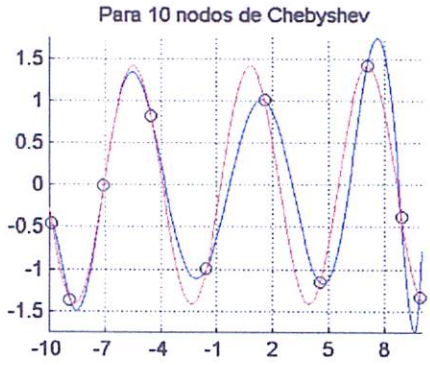

Error discreto con nodos de Chebyshev
20
15
10
5
0
2 4 6 8 10 12 14 16 18 20

- *Para g(x):*


Para 5 nodos igualmente espaciados
1
0.5
0
-0.5
-1
-1.5
-10 -7 -4 -1 2 5 8


Para 10 nodos igualmente espaciados
2
0
-2
-4
-6
-10 -7 -4 -1 2 5 8

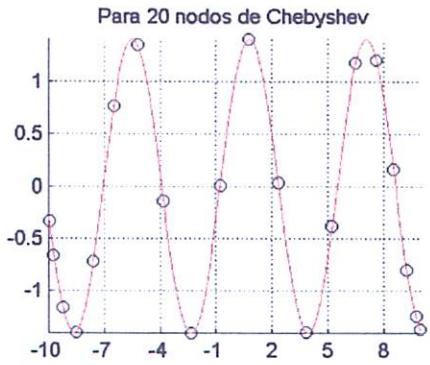

Para 5 nodos de Chebyshev
1
0.5
0
-0.5
-1
-1.5
-10 -7 -4 -1 2 5 8

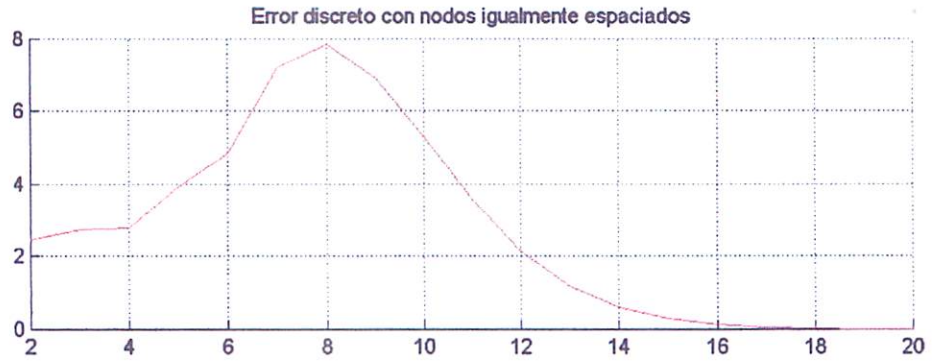

Para 10 nodos de Chebyshev
1.5
1
0.5
0
-0.5
-1
-1.5
-10 -7 -4 -1 2 5 8


Para 15 nodos igualmente espaciados
1
0.5
0
-0.5
-1
-10 -7 -4 -1 2 5 8


Para 20 nodos igualmente espaciados
1
0.5
0
-0.5
-1
-10 -7 -4 -1 2 5 8

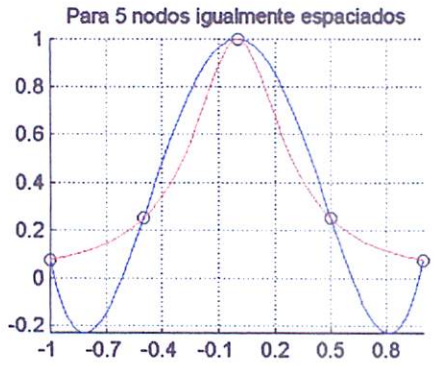

Para 15 nodos de Chebyshev
1
0.5
0
-0.5
-1
-10 -7 -4 -1 2 5 8

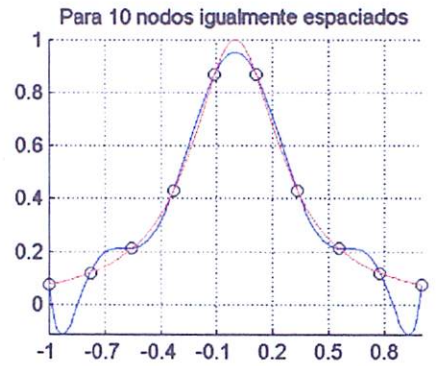

Para 20 nodos de Chebyshev
1
0.5
0
-0.5
-1
-10 -7 -4 -1 2 5 8

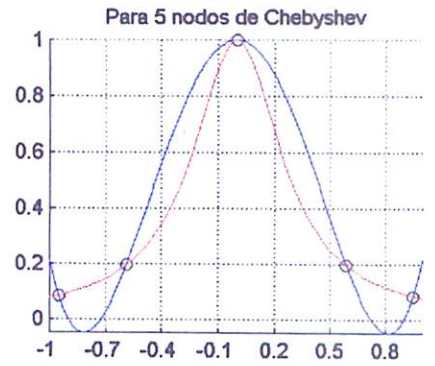

Error discreto con nodos igualmente espaciados
8
6
4
2
0
2
4
6
8
10
12
14
16
18
20

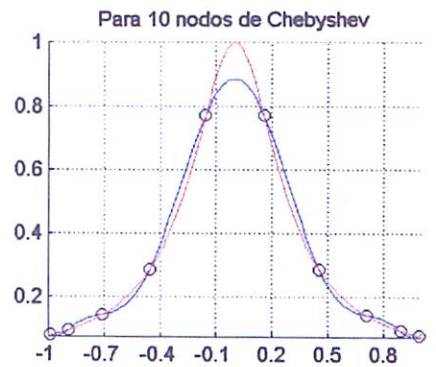

Error discreto con nodos de Chebyshev
4
3
2
1
0
2
4
6
8
10
12
14
16
18
20

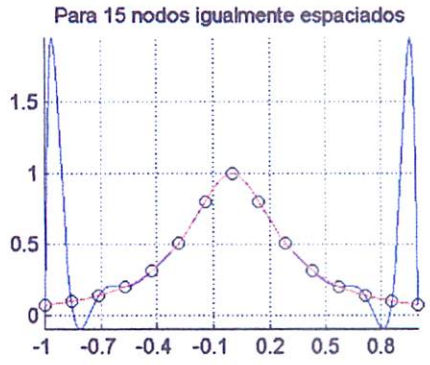
- *Para h(x):*

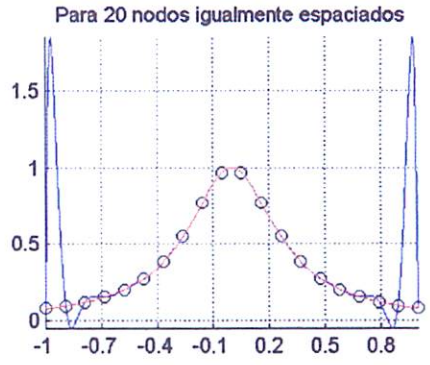

Para 5 nodos igualmente espaciados
1
0.8
0.6
0.4
0.2
0
-0.2
-1
-0.7
-0.4
-0.1
0.2
0.5
0.8

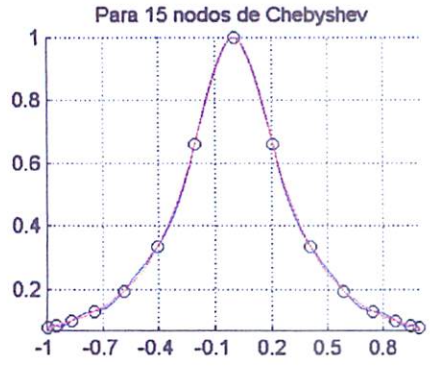

Para 10 nodos igualmente espaciados
1
0.8
0.6
0.4
0.2
0
-1
-0.7
-0.4
-0.1
0.2
0.5
0.8

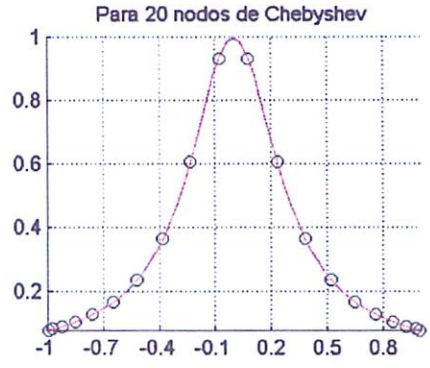

Para 5 nodos de Chebyshev
1
0.8
0.6
0.4
0.2
0
-1
-0.7
-0.4
-0.1
0.2
0.5
0.8

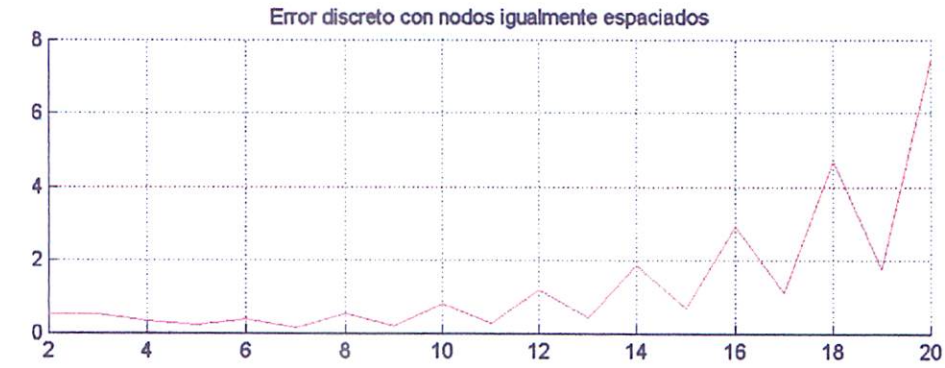

Para 10 nodos de Chebyshev
1
0.8
0.6
0.4
0.2
-1
-0.7
-0.4
-0.1
0.2
0.5
0.8

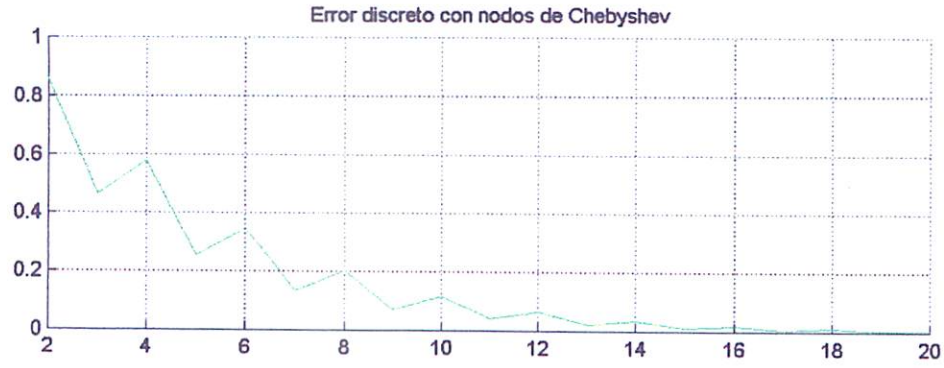

Para 15 nodos igualmente espaciados
1.5
1
0.5
0
-1 -0.7 -0.4 -0.1 0.2 0.5 0.8


Para 20 nodos igualmente espaciados
1.5
1
0.5
0
-1 -0.7 -0.4 -0.1 0.2 0.5 0.8


Para 15 nodos de Chebyshev
1
0.8
0.6
0.4
0.2
-1 -0.7 -0.4 -0.1 0.2 0.5 0.8


Para 20 nodos de Chebyshev
1
0.8
0.6
0.4
0.2
-1 -0.7 -0.4 -0.1 0.2 0.5 0.8


Error discreto con nodos igualmente espaciados
8
6
4
2
0
2 4 6 8 10 12 14 16 18 20


Error discreto con nodos de Chebyshev
1
0.8
0.6
0.4
0.2
0
2 4 6 8 10 12 14 16 18 20

Como se puede apreciar en las gráficas anteriores el número de nodos que se utilizan influyen en el error que se comete al interpolar. También depende de cómo estén situados dichos nodos, viendo que con nodos igualmente espaciados se obtiene un error algo mayor, en cualquier caso, que con nodos no igualmente espaciados (nodos de Chebyshev) siempre que el número de nodos sea lo suficientemente alto.

Se puede observar en las gráficas del error discreto que el error depende además de la función en estudio, pues, para las diferentes funciones estudiadas hemos obtenido en algunas, una diferencia grande entre interpolación a partir de nodos igualmente espaciados e interpolación a partir de nodos de Chebyshev (para un mismo número de nodos) y para otras gráficas el error es muy similar.

Pasamos ahora a realizar una comparativa respecto al número de operaciones y tiempo empleado en realizar los cálculos. Para ello vamos a evaluar cada uno de los algoritmos en varios puntos. Además, vamos a aprovechar que se ha implementado el algoritmo de Horner para hacer una comparativa entre éste y polyval cuando obtenemos un polinomio interpolador y queremos evaluarlo en un punto determinado.(Los algoritmos que tenemos implementados que devuelven el polinomio interpolador son Lagrange y Diferencias Divididas).

Tabla comparativa para f(x) en el intervalo [-10, 10] y nodos igualmente espaciados

METODO	PUNTO	Nº NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	5.3689	5	1.20263342380320	0.05000000000000	297
		10	2.92121432189999	0.05000000000000	1351
		15	3.39810332802459	0.05000000000000	3877
		20	3.27696020901269	0.06000000000000	7829
NEVILLE	5.3689	5	1.20263342380320	0.06000000000000	109
		10	2.92121432189999	0.05000000000000	469
		15	3.39810332802456	0.06000000000000	1079
		20	3.37436051616393	0.06000000000000	1939
DIFERENCIAS DIVIDIDAS	5.3689	5	1.20263342380320	0.05000000000000	86
		10	2.92121432189984	0.06000000000000	351
		15	3.39810332802577	0.05000000000000	791
		20	3.37436051616334	0.06000000000000	1406
DIFERENCIAS PROGRESIVAS	5.3689	5	1.20263342380320	0.06000000000000	59
		10	2.92121432189983	0.06000000000000	174
		15	3.39810332802507	0.06000000000000	339
		20	3.37436051615519	0.06000000000000	554

Tabla comparativa para g(x) en el intervalo [-10, 10] y nodos igualmente espaciados

MÉTODO	PUNTO	Nº NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	3.5	5	-0.12536752933211	0.06000000000000	296
		10	-1.35813343289685	0.05000000000000	1410
		15	-1.28795496580631	0.05000000000000	3915
		20	-1.28723952718203	0.05000000000000	7884
NEVILLE	3.5	5	-0.12536752933211	0.06000000000000	109
		10	-1.35813343289685	0.05000000000000	469
		15	-1.28795496580631	0.06000000000000	1079
		20	-1.28723952718203	0.05000000000000	1939
DIFERENCIAS DIVIDIDAS	3.5	5	-0.12536752933211	0.06000000000000	86
		10	-1.35813343289687	0.06000000000000	351
		15	-1.28795496580644	0.05000000000000	791
		20	-1.28723952718192	0.05000000000000	1406
DIFERENCIAS PROGRESIVAS	3.5	5	-0.12536752933211	0.05000000000000	59
		10	-1.35813343289680	0.06000000000000	174
		15	-1.28795496580626	0.05000000000000	339
		20	-1.28723952718198	0.05000000000000	554

Tabla comparativa para h(x) en el intervalo [-1, 1] y nodos igualmente espaciados

MÉTODO	PUNTO	Nº NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	0.3	5	0.69012307692308	0.06000000000000	298
		10	0.49562005630056	0.05000000000000	1525
		15	0.48244563498770	0.05000000000000	3709
		20	0.48151213916252	0.05000000000000	8027
NEVILLE	0.3	5	0.69012307692308	0.06000000000000	109
		10	0.49562005630056	0.05000000000000	469
		15	0.48244563498770	0.05000000000000	1079
		20	0.48151213916252	0.06000000000000	1939
DIFERENCIAS DIVIDIDAS	0.3	5	0.69012307692308	0.06000000000000	86
		10	0.49562005630056	0.05000000000000	351
		15	0.48244563498770	0.06000000000000	791
		20	0.48151213916251	0.05000000000000	1406
DIFERENCIAS PROGRESIVAS	0.3	5	0.69012307692308	0.05000000000000	59
		10	0.49562005630056	0.06000000000000	174
		15	0.48244563498770	0.05000000000000	339
		20	0.48151213916252	0.05000000000000	554

Tabla comparativa para f(x) en el intervalo [-10, 10] y nodos de Chebyshev

MÉTODO	PUNTO	N°NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	5.3689	5	5.96928459970383	0.05000000000000	311
		10	1.20924666646413	0.05000000000000	1551
		15	3.28613072548098	0.05000000000000	3785
		20	3.37429617584682	0.06000000000000	7679
NEVILLE	5.3689	5	5.96928459970383	0.06000000000000	109
		10	1.20924666646412	0.05000000000000	469
		15	3.28613072548100	0.06000000000000	1079
		20	3.37429617584685	0.06000000000000	1939
DIFERENCIAS DIVIDIDAS	5.3689	5	5.96928459970383	0.05000000000000	86
		10	1.20924666646413	0.06000000000000	351
		15	3.28613072548099	0.05000000000000	791
		20	3.37429617584686	0.06000000000000	1406

Tabla comparativa para g(x) en el intervalo [-10, 10] y nodos de Chebyshev

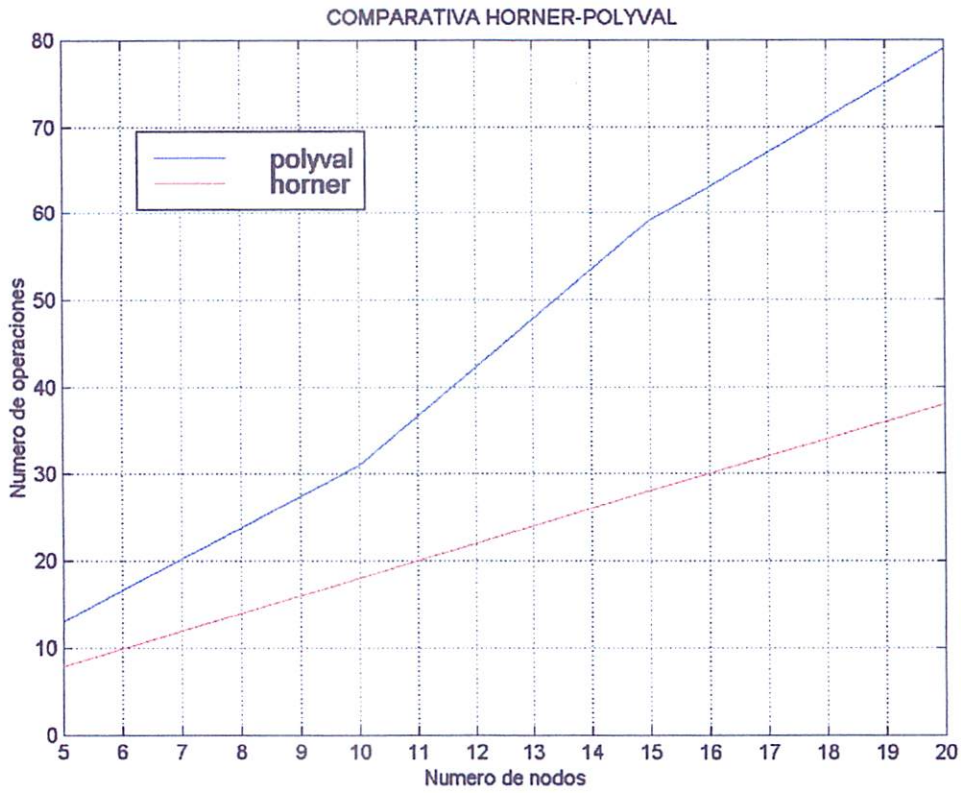
MÉTODO	PUNTO	N°NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	3.5	5	0.75177137327014	0.06000000000000	301
		10	-0.46892915105535	0.05000000000000	1551
		15	-1.28104839198411	0.05000000000000	3785
		20	-1.28725917421774	0.05000000000000	7799
NEVILLE	3.5	5	0.75177137327014	0.06000000000000	109
		10	-0.46892915105535	0.05000000000000	469
		15	-1.28104839198411	0.06000000000000	1079
		20	-1.28725917421774	0.05000000000000	1939
DIFERENCIAS DIVIDIDAS	3.5	5	0.75177137327014	0.06000000000000	86
		10	-0.46892915105535	0.06000000000000	351
		15	-1.28104839198411	0.05000000000000	791
		20	-1.28725917421773	0.05000000000000	1406

Tabla comparativa para h(x) en el intervalo [-1, 1] y nodos de Chebyshev

MÉTODO	PUNTO	N°NODOS	RESULTADO	TIEMPO (s)	OPERs
LAGRANGE	0.3	5	0.99641215944151	0.06000000000000	305
		10	0.03996865934418	0.05000000000000	1560
		15	0.96744486679044	0.05000000000000	3660
		20	0.14202440711041	0.05000000000000	7799
NEVILLE	0.3	5	0.99641215944151	0.06000000000000	109
		10	0.03996865934418	0.05000000000000	469
		15	0.96744486679044	0.05000000000000	1079
		20	0.14202440711043	0.06000000000000	1939
DIFERENCIAS DIVIDIDAS	0.3	5	0.99641215944151	0.06000000000000	86
		10	0.03996865934418	0.05000000000000	351
		15	0.96744486679044	0.06000000000000	791
		20	0.14202440711043	0.05000000000000	1406

A continuación pasamos a comparar la eficiencia del comando polyval de Matlab con nuestro algoritmo de Horner de evaluación de polinomios para la función f(x) en el intervalo [-10, 10]:

MÉTODO	PUNTO	N°NODOS	RESULTADO	OPERs
POLYVAL	5.3689	5	1.21552680131571	13
		10	2.77580030650017	31
		15	3.30153179229586	59
		20	3.27696020901269	79
HORNER	5.3689	5	1.21552680131571	8
		10	2.77580030650017	18
		15	3.30153179229586	28
		20	3.27696020901269	38



Como se puede observar en la tabla y en la gráfica anterior Horner siempre realiza un número de operaciones inferior, para un mismo número de nodos, que polyval. A su vez se puede ver que Horner presenta un crecimiento lineal en el número de operaciones con respecto al número de nodos, esto no ocurre sin embargo con polyval, ya que cuando aumenta el número de nodos el número de operaciones se 'dispara'.

2.4.2. DERIVACIÓN E INTEGRACIÓN

2.4.2.1 DERIVACIÓN (h_0=0.1)

$f(x) = x \cos(x) \Rightarrow f'(x) = \cos(x) - x \sin(x)$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-6.231	10^{-3}	1.32365773689103	64
	10^{-6}	1.32365773691367	102
	10^{-12}	1.32365773691359	148
-2.425	10^{-3}	-2.34683464015800	64
	10^{-6}	-2.34683464017935	102
	10^{-12}	-2.34683464017935	102
3.398	10^{-3}	-0.10555076893486	64
	10^{-6}	-0.10555076893486	64
	10^{-12}	-0.10555076895314	148
9.875	10^{-3}	3.39690839670789	64
	10^{-6}	3.39690839670789	64
	10^{-12}	3.39690839670121	148

$g(x) = \sin(x) + \cos(x) \Rightarrow g'(x) = \cos(x) - \sin(x)$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-4.789	10^{-3}	-0.92053071080132	70
	10^{-6}	-0.92053071080132	70
	10^{-12}	-0.92053071080407	158
-0.001	10^{-3}	1.00099949983027	70
	10^{-6}	1.00099949983027	70
	10^{-12}	1.00099949983338	158
6.023	10^{-3}	1.22360194914516	70
	10^{-6}	1.22360194914516	70
	10^{-12}	1.22360194914881	158
9.986	10^{-3}	-0.31438410267252	38
	10^{-6}	-0.31438416814876	70
	10^{-12}	-0.31438416814968	110

$h(x) = 1/(1+12x^2) \Rightarrow h'(x) = -24x/(1+12x^2)^2$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-0.7983	10^{-3}	0.25621633723211	76
	10^{-6}	0.25621633135333	118
	10^{-12}	0.25621633135211	226
-0.3423	10^{-3}	1.41910664572333	76
	10^{-6}	1.41910826797953	168
	10^{-12}	1.41910826797920	226
0.2123	10^{-3}	-2.14603806792141	118
	10^{-6}	-2.14603806715657	168
	10^{-12}	-2.14603806716216	292
0.9985	10^{-3}	-0.14258684681556	76
	10^{-6}	-0.14258684422925	118
	10^{-12}	-0.14258684422940	168

2.4.2.2. DERIVACIÓN DE POLINOMIOS INTERPOLADORES

Para conseguir los polinomios interpoladores hemos utilizado el algoritmo de Lagrange y 20 nodos de Chebyshev en el intervalo [-10, 10] para f(x) y g(x), y [-1, 1] para h(x). Debido a ello se obtienen resultados muy aproximados a los vistos anteriormente. *Recuérdese que h_0 es 0.1.*

$f(x) = x \cos(x) \Rightarrow f'(x) \approx P_{19}(x)$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-6.231	10^{-3}	1.32393639382526	538
	10^{-6}	1.32393639385005	734
	10^{-10}	1.32393639385005	734
-2.425	10^{-3}	-2.34696481986591	538
	10^{-6}	-2.34696481988731	734
	10^{-10}	-2.34696481988731	734
3.398	10^{-3}	-0.10545932127129	538
	10^{-6}	-0.10545932127129	538
	10^{-10}	-0.10545932128957	734
9.875	10^{-3}	3.39698146460621	538
	10^{-6}	3.39698146522040	734
	10^{-9}	3.39698146522040	734

$g(x) = \sin(x) + \cos(x) \Rightarrow g'(x) = Q_{19}(x)$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-4.789	10^{-3}	-0.92050702319300	538
	10^{-6}	-0.92050702319300	538
	10^{-10}	-0.92050702319570	734
-0.001	10^{-3}	1.00099802692529	538
	10^{-6}	1.00099802692529	538
	10^{-10}	1.00099802692838	734
6.023	10^{-3}	1.22362930055879	538
	10^{-6}	1.22362930055879	538
	10^{-10}	1.22362930056207	734
9.986	10^{-3}	-0.31541800061170	350
	10^{-6}	-0.31541868715901	538
	10^{-9}	-0.31541868706028	1150

$h(x) = 1/(1+12x^2) \Rightarrow h'(x) = R_{19}(x)$			
PUNTO	TOLERANCIA	RESULTADO	OPERs
-0.7983	10^{-3}	0.26369821400681	734
	10^{-6}	0.26369832373387	938
	10^{-10}	0.26369832368894	1150
-0.3423	10^{-3}	1.37337192269287	734
	10^{-6}	1.37337191634883	938
	10^{-10}	1.37337191634829	1150
0.2123	10^{-3}	-2.23305745441147	734
	10^{-6}	-2.23305745540005	938
	10^{-10}	-2.23305745540017	1150
0.9985	10^{-3}	-0.31045258334942	938
	10^{-6}	-0.31045258297976	1150
	10^{-9}	-0.31045258297976	1150

2.4.2.3. INTEGRACIÓN

$f(x) = x \cos(x) \Rightarrow \int f(x)dx = x \sin(x) + \cos(x)$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-5, 2]	10^{-3}	5.91340720495146	2506
	10^{-6}	5.91340720495146	2506
	10^{-10}	5.91340720495669	5342
[2.5, 7]	10^{-3}	4.65777170066429	1631
	10^{-6}	4.65777170066429	1631
	10^{-10}	4.65777170066187	3467

$g(x) = \sin(x) + \cos(x) \Rightarrow \int g(x)dx = -\cos(x) + \sin(x)$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-3, 1]	10^{-3}	-0.54770382862398	750
	10^{-6}	-0.54770380960054	1739
	10^{-10}	-0.54770380960082	3696
[7, 9]	10^{-3}	1.42016440275021	899
	10^{-6}	1.42016440275021	899
	10^{-10}	1.42016440275095	1896

$h(x) = 1/(1+12x^2) \Rightarrow \int h(x)dx = \frac{arctg(\sqrt{12}x)}{\sqrt{12}}$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-0.7, -0.3]	10^{-3}	0.10825850462897	258
	10^{-6}	0.10825850462897	258
	10^{-10}	0.10825850804109	1014
[-0.2, 0.4]	10^{-3}	0.44789497797777	356
	10^{-6}	0.44789497120145	730
	10^{-10}	0.44789497122171	1448

2.4.2.4. INTEGRACIÓN DE POLINOMIOS INTERPOLADORES

Para conseguir los polinomios interpoladores hemos utilizado el algoritmo de Lagrange y 20 nodos de Chebyshev en el intervalo [-10, 10] para f(x) y g(x), y [-1, 1] para h(x). Debido a ello se obtienen resultados muy aproximados a los vistos anteriormente. *Recuérdese que h_0 es 0.1.*

$f(x) = x \cos(x) \Rightarrow \int P_{19}dx$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-5, 2]	10^{-3}	5.91338324066809	41453
	10^{-6}	5.91338324066809	41453
	10^{-10}	5.91338324067333	88608
[2.5, 7]	10^{-3}	4.65773189849867	26753
	10^{-6}	4.65773189849867	26753
	10^{-10}	4.65773189849623	57108

$g(x) = \sin(x) + \cos(x) \Rightarrow \int Q_{19}dx$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-3, 1]	10^{-3}	-0.54771363668284	10266
	10^{-6}	-0.54771361765299	23813
	10^{-10}	-0.54771361765327	50808
[7, 9]	10^{-3}	1.42016922112887	12053
	10^{-6}	1.42016922112887	12053
	10^{-10}	1.42016922112957	25608

$h(x) = 1/(1+12x^2) \Rightarrow \int R_{19}dx$			
INTERVALO	TOLERANCIA	RESULTADO	OPERs
[-0.7, -0.3]	10^{-3}	0.05369540052318	1194
	10^{-6}	0.05369539899634	2645
	10^{-10}	0.05369539899639	5448
[-0.2, 0.4]	10^{-3}	0.08662191952854	1698
	10^{-6}	0.08662191662102	3821
	10^{-10}	0.08662191662111	7968

2.5. IMPLEMENTACIÓN DE LOS MÉTODOS

-FUNCIONES AUXILIARES DE INTERPOLACIÓN

```
function y=faux(x)
% Se selecciona manualmente la funcion a estudiar
y=x.*cos(x);              %f(x)
%y=sin(x)+cos(x);         %g(x)
%y=1./(1+12.*x.^2);       %h(x)


function dibujar(x,y,x0,y0,sierror)
%function dibujar(x,y,x0,y0,sierror)
%
% -x  son los nodos iniciales a interpolar.
% -y  son las imagenes de los nodos iniciales.
% -x0 son los nodos a interpolar (evaluando en polinomio resultante).
% -y0 imagenes de los nodos obtenidas por un algoritmo de interpolacion.
% -sierror vble. booleana que indica si se quiere graficar el error absoluto.
if(sierror)
  subplot(2,1,1)
  %ejes(min(x0),max(x0));
  %title('Graficas real y aproximada(nodos igualmente espaciados)');
    title('Graficas real y aproximada para nodos de Chebyshev');
end
ejes(-10,10),hold on,grid on;
plot(x,y,'k O')          %Nodos
plot(x0,y0,'b')          %Aproximacion
plot(x0,faux(x0),'r') %Real
axis tight
if(sierror)
  legend('nodos','aprox','real',0);
  subplot(2,1,2),hold on,grid on;
  ejes(-10,10);  % Esto hay que cambiarlo para cada grafico
  %ejes(min(x0),max(x0));
  title('Error absoluto');
  ydex0=abs(faux(x0)-y0);
  plot(x0,ydex0,'g');
  axis tight
end
```

```
function chebygraf(a,b,n1,n2)
%function chebygraf(a,b,n1,n2)
%
% -a     es el intervalo a interpolar [a, b]
% -b
% -ni son los numeros de nodos en cada ventana
%
% SI UN VECTOR DE NODOS SE INTERPOLAN MEDIANTE
% CUALQUIER METODO ESTUDIADO, ESTE DEBE DEVOLVER
% EL MISMO POLINOMIO INTERPOLADOR QUE OTRO METODO,
% DEBIDO A ELLO SE USA PARA LAS GRAFICAS DIF_DIV.
x(1)=n1;x(2)=n2;
for i=1:2
  switch(i)
    case 1, nodo=linspace(a,b,x(1));xche=chebyshev(x(1),a,b);
    case 2, nodo=linspace(a,b,x(2));xche=chebyshev(x(2),a,b);
  end
  ynodo=faux(nodo);
  yche=faux(xche);
  x0=a:(b-a)/200:b;
  subplot(2,2,i);
  buffer=sprintf('Para %d nodos igualmente espaciados',x(i));
  title(buffer);
  dibujar(nodo,ynodo,x0,polidifdiv(nodo,x0,difdiv(nodo,ynodo,0)),0)
  subplot(2,2,i+2);
  buffer=sprintf('Para %d nodos de Chebyshev',x(i));
  title(buffer);
  dibujar(xche,yche,x0,polidifdiv(xche,x0,difdiv(xche,yche,0)),0)
end

function compara(x)
%function compara(x)
 % Compara Horner con polyval
%
% -x es el punto donde se quiere evaluar
iniciomedida;
horner([1 5 3 2 1 11 1212 1212],x)
finmedida(0);
iniciomedida;
polyval([1 5 3 2 1 11 1212 1212],x)
finmedida(0);
```

```
function cuatro(a,b,n1,n2,n3,n4)
%function cuatro(a,b,n1,n2,n3,n4)
%
% -a     es el intervalo a interpolar [a, b]
% -b
% -ni son los numeros de nodos en cada ventana
%
% SI UN VECTOR DE NODOS SE INTERPOLAN MEDIANTE
% CUALQUIER METODO ESTUDIADO, ESTE DEBE DEVOLVER
% EL MISMO POLINOMIO INTERPOLADOR QUE OTRO METODO,
% DEBIDO A ELLO SE USA PARA LAS GRAFICAS DIF_DIV.
x(1)=n1;x(2)=n2;
x(3)=n3;x(4)=n4;
for i=1:4
  switch(i)
    case 1, nodo=linspace(a,b,x(1));
    case 2, nodo=linspace(a,b,x(2));
    case 3, nodo=linspace(a,b,x(3));
    case 4, nodo=linspace(a,b,x(4));
  end
  ynodo=faux(nodo);
  x0=a:(b-a)/200:b;
  subplot(2,2,i);
  buffer=sprintf('Para %d nodos',x(i));
  title(buffer);
  dibujar(nodo,ynodo,x0,polidifdiv(nodo,x0,difdiv(nodo,ynodo,0)),0)
end

function errores(a,b,m)
%function errores(a,b,m)
%
% Grafica el error discreto desde 2 hasta m nodos en el intervalo [a, b]
h=(b-a)/200;
clf
for i=1:2
    c=0;
  subplot(2,1,i),grid on,hold on
  for n=2:1:m
    switch(i)
    case 1,nodos=linspace(a,b,n+1);color='r';title('Error discreto con nodos igualmente ...
        espaciados')
    case 2,nodos=chebyshev(n,a,b);color='g';title('Error discreto con nodos de Chebyshev')
    end
    div=difdiv(nodos,faux(nodos),0);
    c=c+1;
    absc(c)=n;
    Max(c)=0;
    for x0=a:h:b% Calcula el error maximo en todo el intervalo para n nodos
      if abs(faux(x0)-(polidifdiv(nodos,x0,div)))>Max(c)
        Max(c)=abs(faux(x0)-(polidifdiv(nodos,x0,div)));
      end
    end
  end
  plot(absc,Max,color);
end
```

```
function ejes(a,b)
% Controla GRID en ejes
set(gca,'YTickMode','auto');
set(gca,'XTickMode','manual');
set(gca,'XTick',a:2:b);

function y=horner1(p,x)
%function y=horner1(p,x)
%
% -p polinomio en forma de vector
% -x valor x a evaluar en p
%
% HORNER HACE MENOS OPERACIONES QUE POLYVAL
y=p(1);
for k=2:length(p)
  y=p(k)+x*y;
end
```

-MÉTODOS DE INTERPOLACIÓN

```
function pol=lagrange(x,y,grafica)
%function pol=lagrange(x,y,grafica)
%
% Siendo (x,y) los puntos a interpolar
% -x        vector de entrada
% -y        vector de entrada
% -grafica   indica si se quiere ver graficas '1' o no '0'.
% -pol       vector con los coeficientes del polinomio interpolador.
%
% Se muestra polinomio evaluado con polyval y puntos
format long
if (length(x)~=length(y))
  ('El vector x y el vector y tienen longitudes diferentes.')
else
  n=length(x);
    A=zeros(n,n);
  iniciomedida;
  ant=1;
    for i=1:1:n
      ant=1;
      for j=n:-1:1
        A(i,j)=ant;
      ant=x(i)*ant;
        end
    end
  p=A\y';
  finmedida(n);
  pol=p';
  if(grafica)
    x0=min(x):0.01:max(x);
    dibujar(x,y,x0,polyval(p',x0),1);
  end
end
```

```
function neville(x,y,p)
%function neville(x,y,p)
%
% -x vector de entrada
% -y vector de entrada
% -p punto a evaluar en el polinomio
iniciomedida
n=length(x)-1;
for i=1:n
  for j=1:n+1-i
    y(j)=((p-x(j))*y(j+1)-(p-x(i+j))*y(j))/(x(i+j)-x(j));
  end
end
y(1)
finmedida(n);

function p=difdiv(x,y,grafica)
%function p=difdiv(x,y,grafica)
%
% -x        vector de entrada
% -y        vector de entrada
% -grafica  indica si se quiere graficas '1' o no '0'
%
% SE PUEDE ACTIVAR MEDIDAS DE TIEMPOS EN EL INTERIOR
format long
if (length(x)~=length(y))
  ('El vector x y el vector y tienen longitudes diferentes.')
else
    n=length(x);
    p=zeros(1,n);
    p(1,1)=y(1);
    f=y;
    var=n;
    z=0;
  iniciomedida;
    for inc=2:1:n
     var=var-1;
     z=z+1;
     for i=1:1:var
       f(i)=((f(i+1)-f(i))/(x(i+z)-x(i)));
     end
     p(1,inc)= f(1);
    end
  if(grafica)
    %REPRESENTACION GRAFICA
    x0=min(x):0.1:max(x);
    dibujar(x,y,x0,polidifdiv(x,x0,p),1);
  end
end
```

```
function resul=polidifdiv(x,x0,vector)
%function resul=polidifdiv(x,x0,vector)
%
% -x       vector de las x de entrada
% -x0      punto a evaluar el polinomio
% -vector  diferencias divididas para unos puntos x e y
%
% Evalua un punto con las diferencias divididas dadas
format long
resul=vector(1);
for i=2:length(vector)
  temp=vector(i);
  for j=1:(i-1)
    temp=temp.*(x0-x(j));
  end
  resul=resul+temp;
end
finmedida(length(vector));

function result=polinomio(x,coeficiente)
%function result=polinomio(x,coeficiente)
%
% -x              conjunto de coordenadas x para las que se hayo las dif. divididas
% -coeficiente    diferencias divididas para unos puntos (x, y)
format long
n=length(x);
result=zeros(1,n);
result(n)=coeficiente(1);
aux=1;
tmp2=zeros(1,n);
for i=2:1:n
  presult=n;
  aux=producto(aux,[1 -x(i-1)]);
  tmp=producto(aux,coeficiente(i));
  for j=length(tmp):-1:1
    result(presult)=result(presult)+tmp(j);
    presult=presult-1;
  end
end
function salida=producto(pol1,pol2)
%function salida=producto(pol1,pol2)
%
% Realiza el producto vectorial de dos polinomios pol1 y pol2
lpol1=length(pol1);
lpol2=length(pol2);
salida=zeros(1,lpol1+lpol2-1);
cont=0;
for inc1=1:1:lpol2
  for inc2=1:1:lpol1
    salida(inc2+cont)=(pol2(inc1)*pol1(inc2))+salida(inc2+cont);
  end
  cont=cont+1;
end
```

```
function p=difprog(x,y,grafica)
%function p=difprog(x,y,grafica)
%
% -x        vector x de entrada
% -y        vector y de entrada
% -grafica    indica si se quieren mostrar graficas '1' o no '0'
%
% SE PUEDE ACTIVAR MEDIDAS DE TIEMPOS EN EL INTERIOR
% NO HACE FALTA ORDENAR LOS VECTORES DE ENTRADA
format long
if (length(x)~=length(y))
  disp('El vector x y el vector y tienen longitudes diferentes.');
else
%   [x,y]=ordenar(x,y);
%   if(~espacios_iguales(x))
%     disp('El vector x no esta igualmente espaciado');
%   end
  n=length(x);
     p=zeros(1,n);
     p(1,1)=y(1);
     f=y;
     var=n;
     z=0;
  iniciomedida;
  for inc=2:1:n
       var=var-1;
       z=z+1;
       for i=1:1:var
         f(i)=(f(i+1)-f(i));
       end
       p(1,inc)= f(1);
  end
  if(grafica)
    x0=min(x):0.01:max(x);
    dibujar(x,y,x0,polidifprog(x,x0,p),1);
  end
end
function bool=espacios_iguales(x)
n=length(x);
espv=x(2)-x(1);
espn=espv;
i=2;
while(espn==espv & i<n)
  espn=x(i+1)-x(i);
  i=i+1;
end
if(espn~=espv)
  bool=0;
else
  bool=1;
end
```

```
function [x,y]=ordenar(x,y)
n=length(x);
for i=1:n-1
  pmin=i;
  for j=i+1:n
    if(x(j)<x(pmin))
      pmin=j;
    end
  end
  % Intercambio de posiciones en x e y
  if(i~=pmin)
    tmp=x(i);
    x(i)=x(pmin);
    x(pmin)=tmp;
    tmp=y(i);
    y(i)=y(pmin);
    y(pmin)=tmp;
  end
end

function resul=polidifprog(x,x0,vector)
%function resul=polidifprog(x,x0,vector)
%
% -x              vector de las x de entrada
% -x0       punto a evaluar
% -vector   diferencias progresivas
format long
h=(abs(x(1)-x(2)));
s=(x0-x(1))/h;
temp=1;
resul=vector(1);
for i=2:length(vector)
  temp=(((s-(i-2)))/(i-1)).*temp;
  resul=resul+(temp.*vector(i));
end
finmedida(length(vector));
```

- FUNCIONES AUXILIARES DE INTEGRACIÓN

```
function y=f(x)
format long
y=x.*cos(x);

function y=g(x)
format long
y=sin(x)+cos(x);

function y=h(x)
format long
y=1./(1+12.*x.^2);
```

```
function y=Nder(f,h,x)
%
%Nder(f,h,x)
% -f  ---> funcion
% -x  ---> valor de x donde se quiere calcular la derivada.
% -h  ---> valor de h .
%
format long
y=(feval(f,x+h)-feval(f,x-h))/(2*h);

function y=Nderp(vector,h,x)

% -vector  ---> polinomio interpolador
% -x       ---> valor de x donde se quiere calcular la derivada.
% -h       ---> valor de h .
%
format long
y=(polyval(vector,x+h)-polyval(vector,x-h))/(2*h);

function tmp=Nint(f,a,b,h,n)
% -f     ---> funcion
% -a, b  ---> intervalo de integracion
% -h     ---> valor de h
format long
tmp=feval(f,a+h);
for i=2:n-1
  tmp=tmp+feval(f,a+i*h);
end
tmp=(feval(f,a)+2*tmp+feval(f,b))*h/2;

function tmp=Nintp(vector,a,b,h,n)
% -vector    ---> polinomio interpolador.
% -a, b      ---> intervalo de integracion
% -h         ---> valor de h
format long
tmp=polyval(vector,a+h);
for i=2:n-1
  tmp=tmp+polyval(vector,a+i*h);
end
tmp=(polyval(vector,a)+2*tmp+polyval(vector,b))*h/2;
```

- MÉTODOS DE DERIVACIÓN E INTEGRACIÓN

```
function [resul]=derivada(f,x,TOL,baseord,expord,incexp,h)

% derivada(f,x,TOL,baseord,expord,incexp,h)
% ejemplo apuntes: derivada('f',1,8,2,2,2,0.1)
%
%
% -f        ---> calcula derivada de la función f
%
% -x        ---> punto en el que se quiere calcular la derivada.
%
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
%
% -       M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
%
% -       N(h)= A + (A-B)/((baseord.^expord)-1)
%
% -baseord   ---> base del orden del error.
%
% -expord    ---> exponente del orden del error.
%
% -incexp    ---> incremento del exponente del error
%
% -h         ---> h incial.
%

format long
base=baseord;
TOL=10.^(-TOL);
cont=1;
i=0;
redt=1;                              %reducciones + tamaño vector
continuar=1;
g(1)=Nder(f,h,x);
aux=g(1)+2;
while (abs(aux-g(1))>=TOL  )
  redt=redt+1;
  i=i+1;
  g(redt)=Nder(f,h/(base.^i),x);
  aux=g(1);                          %sirve para comparación .
  exp=expord;
  for inc=redt:-1:2
    g(inc-1)=derror(base,exp,g(inc-1),g(inc));
    exp=exp+incexp;
  end

end
resul=g(1) ;
```

```
function [resul]=derivada(vector,x,TOL,baseord,expord,incexp,h)
% derivada(vector,x,TOL,baseord,expord,incexp,h)
% ejemplo apuntes: derivada(vector,1,8,2,2,2,0.1)
%
% -vector    ---> polinomio interpolador.
% -x         ---> punto en el que se quiere calcular la derivada.
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
% -       M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
% -       N(h)= A + (A-B)/((baseord.^expord)-1)
% -baseord   ---> base del orden del error.
% -expord    ---> exponente del orden del error.
% -incexp    ---> incremento del exponente del error
% -h         ---> h incial.
%
format long
base=baseord;
TOL=10.^(-TOL);
cont=1;
i=0;
redt=1;                                  %reducciones + tamaño vector
continuar=1;
g(1)=Nderp(vector,h,x);
aux=g(1)+2;
 while (abs(aux-g(1))>=TOL  )
   redt=redt+1;
   i=i+1;
   g(redt)=Nderp(vector,h/(base.^i),x);
   aux=g(1);                             %sirve para comparación .
   exp=expord;
   for inc=redt:-1:2
     g(inc-1)=derror(base,exp,g(inc-1),g(inc));
     exp=exp+incexp;
   end
end
resul=g(1) ;


function [resul]=integral(f,a,b,n,h,TOL,baseord,expord,incexp)
% integral(f,a,b,n,h,TOL,baseord,expord,incexp)
% ejemplo apuntes: integral('f',0,1,10,0,10,2,2,2)
% ejemplo apuntes: integral('f',0,1,0,0.1,10,2,2,2)
%
% -f         ---> calcula derivada de la función f
% -a         ---> punto inferior de integracion
% -b         ---> punto superior de integracion
% -n         ---> numero de intervalos que queremos (a 0 para intro. h)
% -h         ---> h inicial (a 0 si se quiere introducir n)
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
% -       M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
% -       N(h)= A + (A-B)/((baseord.^expord)-1)
% -baseord   ---> base del orden del error.
% -expord    ---> exponente del orden del error.
% -incexp    ---> incremento del exponente del error
```

```
format long
if(h==0)
  h=(b-a)/n;
else
  n=(b-a)/h;
end
base=basecord;
TOL=10.^(-TOL);
cont=1;
i=0;
redt=1;                                   %reducciones + tamaño vector
continuar=1;
g(1)=Nint(f,a,b,h,n);
aux=g(1)+2;
 while (abs(aux-g(1))>=TOL  )
  redt=redt+1;
  i=i+1;
  g(redt)=Nint(f,a,b,h/(base.^i),(b-a)/(h/(base.^i)));
  aux=g(1);                               %sirve para comparación .
  exp=expord;
  for inc=redt:-1:2
    g(inc-1)=derror(base,exp,g(inc-1),g(inc));
    exp=exp+incexp;
  end
end
resul=g(1) ;
```

```
function [resul]=integral(vector,a,b,n,h,TOL,baseord,expord,incexp)
% integral(vector,a,b,n,h,TOL,baseord,expord,incexp)
% ejemplo apuntes: integral(vector,0,1,10,0,10,2,2,2)
% ejemplo apuntes: integral(vector,0,1,0,0.1,10,2,2,2)
%
% -vector    ---> polinomio interpolador.
% -a         ---> punto inferior de integración.
% -b         ---> punto superior de integración.
% -n         ---> numero de intervalos que queremos (a 0 para intro. h).
% -h         ---> h inicial (a 0 si se quiere introducir n).
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
% -       M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
% -       N(h)= A + (A-B)/((baseord.^expord)-1).
% -baseord   ---> base del orden del error.
% -expord    ---> exponente del orden del error.
% -incexp    ---> incremento del exponente del error.
format long
if(h==0)
  h=(b-a)/n;
else
  n=(b-a)/h;
end
base=baseord;
TOL=10.^(-TOL);
cont=1;
i=0;
redt=1;%reducciones + tamaño vector
continuar=1;
g(1)=Nintp(vector,a,b,h,n);
aux=g(1)+2;
 while (abs(aux-g(1))>=TOL )
  redt=redt+1;
  i=i+1;
  g(redt)=Nintp(vector,a,b,h/(base.^i),(b-a)/(h/(base.^i)));
  aux=g(1);%sirve para comparación .
  exp=expord;
  for inc=redt:-1:2
    g(inc-1)=derror(base,exp,g(inc-1),g(inc));
    exp=exp+incexp;
  end
end
resul=g(1) ;
```

-FUNCIONES PARA MEDIDAS

```
function iniciomedida
%CONTADOR DE OPERACIONES A 0 E INCIALIZO EL TEMPORIZADOR
flops(0),tic

function finmedida(h)
% IMPRIMO EL NUMERO DE OPERACIONES EMPLEADAS
disp(' ');disp('NUMERO DE OPERACIONES:'),disp(flops);
disp('TIEMPO EMPLEADO (segundos):'),disp(toc);
disp('ITERACIONES:'),disp(h);
```

PROBLEMAS DE VALORES INICIALES PARA ECUACIONES DIFERENCIALES

-Análisis Numérico I-

Víctor Manuel Galdámez Salguero
Jose Manuel Mesa Gómez

ÍNDICE

1. INTRODUCCIÓN.....2

2. MÉTODO DE EULER O DE LAS TANGENTES.....2

3. MÉTODO DE GRAGG.....5

4. MÉTODO DE RUNGE-KUTTA.....7

5. MÉTODOS MULTIPASO.....8

6. EVALUACIÓN DE LOS MÉTODOS.....12

6.1. EULER (Tolerancia de 10^{-10}).....12

6.2. GRAGG (Tolerancia de 10^{-5}).....13

6.3. RUNGE-KUTTA.....14

6.4. ADAMS-BASHFORTH.....15

6.5. ADAMS-MOULTON.....16

6.6. PREDICTOR-CORRECTOR.....17

7. IMPLEMENTACIÓN DE LOS MÉTODOS.....21

FUNCIONES AUXILIARES.....21

MÉTODOS PARA LA RESOLUCIÓN DE PROBLEMAS DE VALOR INICIAL.....22

FUNCIONES PARA MEDIDAS.....27

1. INTRODUCCIÓN

Se trata de resolver un problema de valores iniciales para ecuaciones diferenciales del siguiente tipo:

$$\begin{cases} y' = f(x,y) \\ y(a) = \alpha \end{cases} \quad x \in [a,b]$$

Este problema en la realidad no se resuelve tan fácilmente, e incluso existen muchos que no se pueden resolver.

Nuestro objetivo es hallar 'y' en un intervalo [a, b] de forma aproximada, y se hará dividiendo dicho intervalo en partes iguales y hallando en cada punto una aproximación a la imagen de 'y' respecto a cada punto. Finalmente se aplicará a los puntos calculados un algoritmo de interpolación, obteniendo así una aproximación a 'y'.

2. MÉTODO DE EULER O DE LAS TANGENTES

Se desea aproximar la solución de la ecuación diferencial $y'=f(x,y)$, $y(x_0)=y_0$. Si $h>0$, entonces sobre la recta tangente en el punto (x_0, y_0) a la curva solución desconocida, encontramos el punto $(x_1, y_1)= (x_0+h, y_1)$. Pues bien, la recta que pasa por los puntos (x_0, y_0) y (x_1, y_1) será

$$y_0' = \frac{y_1 - y_0}{(x_0 + h) - x_0} \qquad \text{ó} \qquad y_1=y_0+hy_0' \text{ donde } y_0'=f(x_0, y_0).$$

Si consideramos que h toma un valor constante, se puede obtener una sucesión de puntos $(x_1, y_1),(x_2, y_2),\ldots,(x_n, y_n)$, que serán aproximaciones de los puntos $(x_1, y(x_1))$, $(x_2, y(x_2))$, ... , $(x_n, y(x_n))$.

Utilizando (x_1, y_1) se puede obtener el valor de y_2 sobre la nueva recta tangente. Así,

$$y_1 = \frac{y_2 - y_1}{h} \text{ o bien } y_2=y_1+hy_i'$$

y en general

$$y_{n+1} = y_n + hy'_n = y_n + hf(x_n, y_n) \text{ siendo } x_n= x_0+nh$$

expresión conocida como método de Euler o de las tangentes, siendo el error de la aproximación de la forma:

$M=N(h)+k_1h+k_2h^2+k_3h^3+\ldots$, siendo N(h) la aproximación al valor real M.

DIAGRAMA DE FLUJO DE EULER

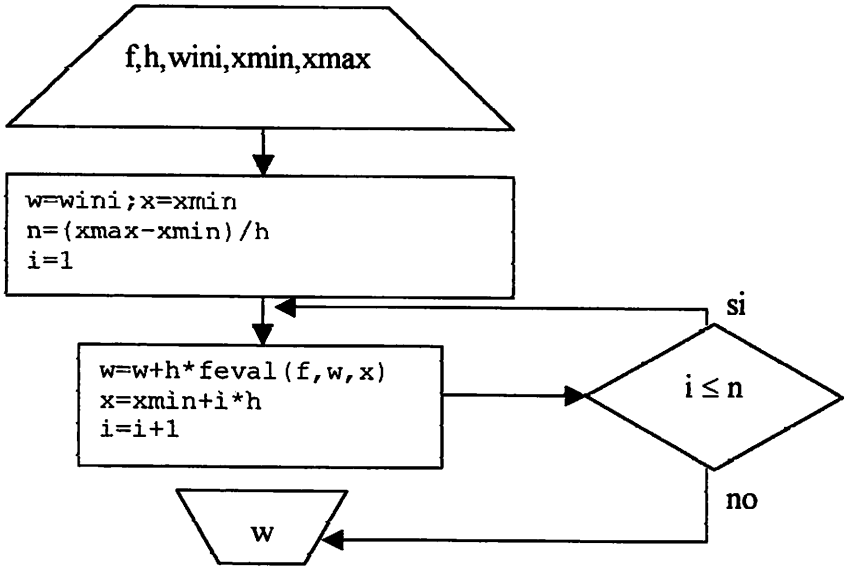
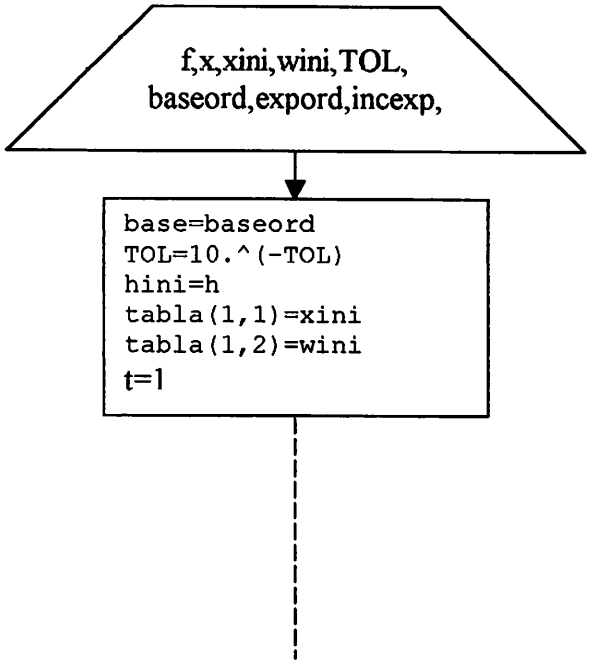
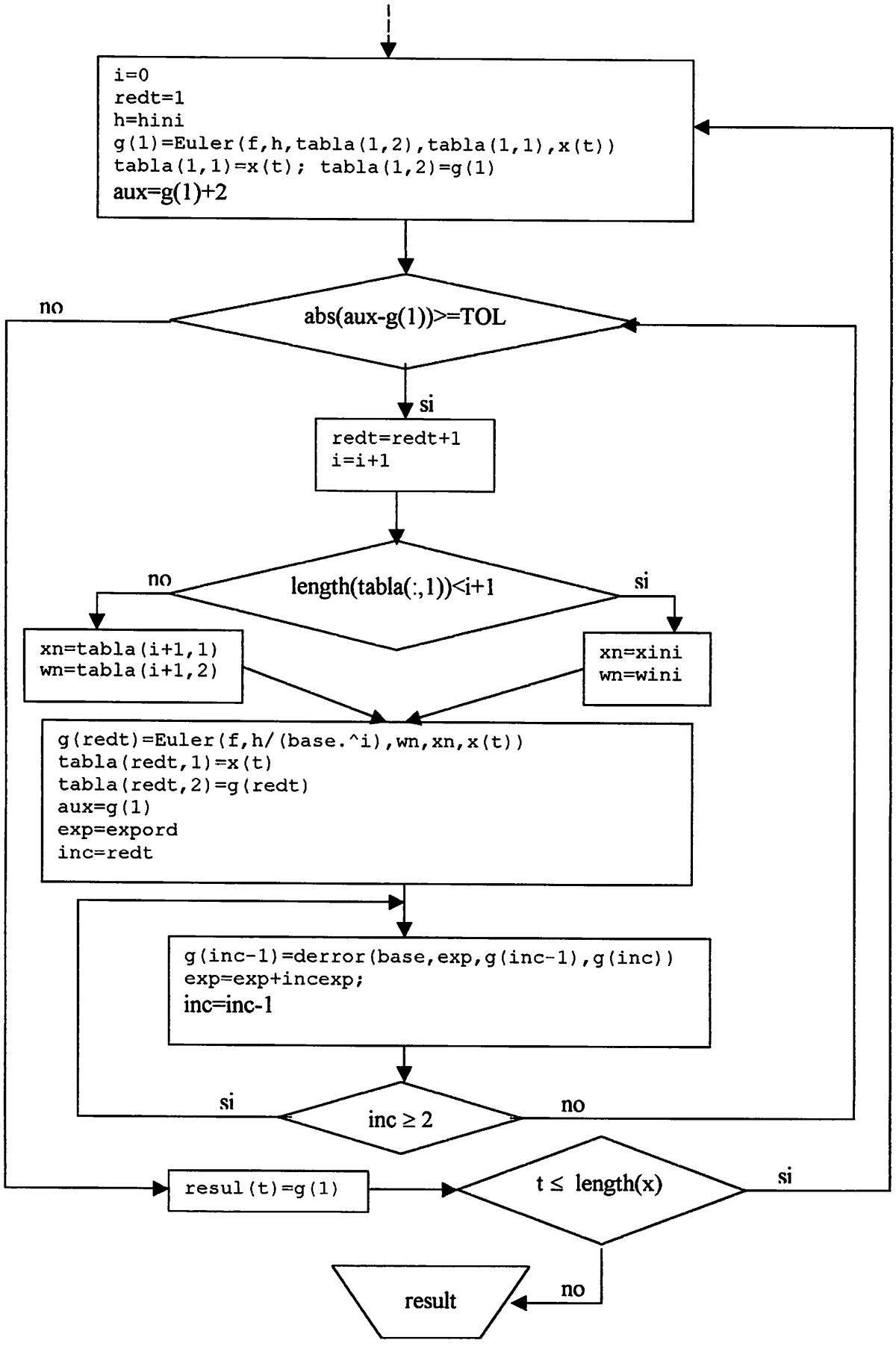


DIAGRAMA DE FLUJO DE EULER CON RICHARDSON PARA MÚLTIPLES PUNTOS




i=0
redt=1
h=hini
g(1)=Euler(f,h,tabla(1,2),tabla(1,1),x(t))
tabla(1,1)=x(t); tabla(1,2)=g(1)
aux=g(1)+2
no
abs(aux-g(1))>=TOL
si
redt=redt+1
i=i+1
no
length(tabla(:,1))<i+1
si
xn=tabla(i+1,1)
wn=tabla(i+1,2)
xn=xini
wn=wini
g(redt)=Euler(f,h/(base.^i),wn,xn,x(t))
tabla(redt,1)=x(t)
tabla(redt,2)=g(redt)
aux=g(1)
exp=expord
inc=redt
g(inc-1)=derror(base,exp,g(inc-1),g(inc))
exp=exp+incexp;
inc=inc-1
si
inc ≥ 2
no
resul(t)=g(1)
t ≤ length(x)
si
result
no

3. MÉTODO DE GRAGG

El método de Gragg es una mejora del de Euler. Gragg utiliza el método de Euler para una primera aproximación, y las siguientes aproximaciones se harán usando las dos anteriores.

$$\begin{cases} w_0 = \alpha \\ w_1 = w_0 + hf(x_0, w_0) \rightarrow \text{Euler} \\ w_{i+1} = w_{i-1} + 2hf(x_i, w_i) \end{cases} \qquad \text{Siendo } w_i \approx f(x_i).$$

Se demuestra que la forma del error es:

$M=N(h)+k_1h^2+k_2h^4+k_3h^6+...$ Siendo N(h) la aproximación al valor real M.

DIAGRAMA DE FLUJO DE GRAGG

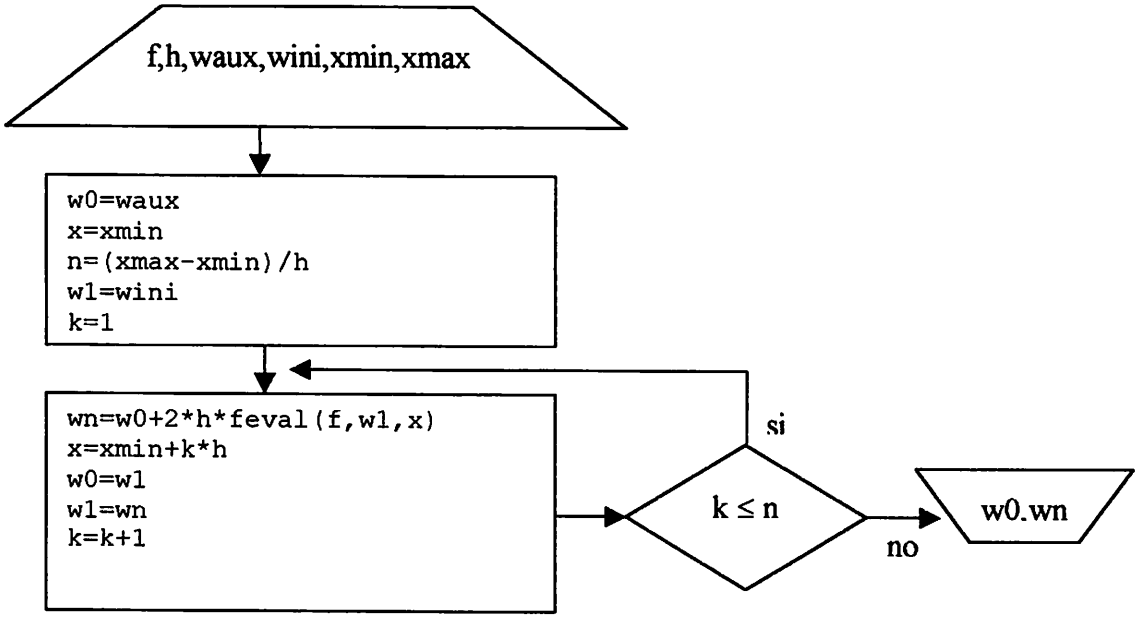
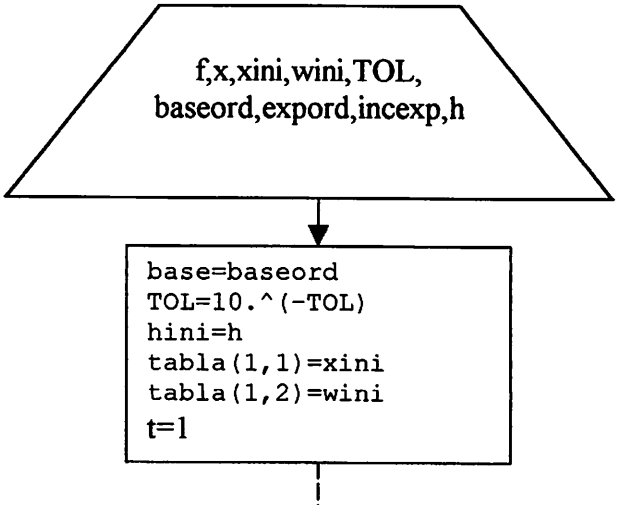
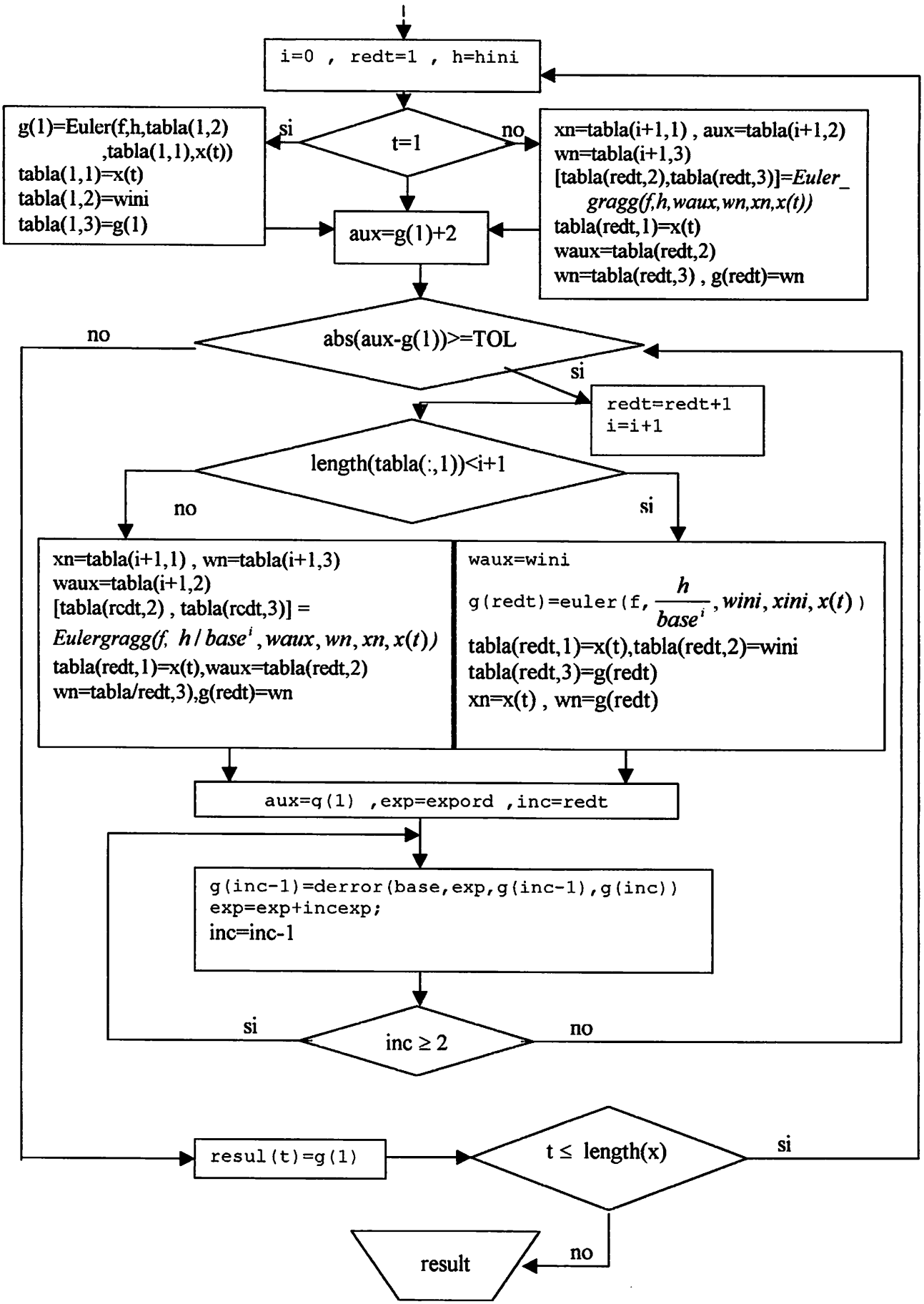


DIAGRAMA DE FLUJO DE GRAGG CON RICHARDSON




i=0 , redt=1 , h=hini
t=1
si
no
g(1)=Euler(f,h,tabla(1,2)
,tabla(1,1),x(t))
tabla(1,1)=x(t)
tabla(1,2)=wini
tabla(1,3)=g(1)
xn=tabla(i+1,1) , aux=tabla(i+1,2)
wn=tabla(i+1,3)
[tabla(redt,2),tabla(redt,3)]=Euler_
gragg(f,h,waux,wn,xn,x(t))
tabla(redt,1)=x(t)
waux=tabla(redt,2)
wn=tabla(redt,3) , g(redt)=wn
aux=g(1)+2
abs(aux-g(1))>=TOL
no
si
redt=redt+1
i=i+1
length(tabla(:,1))<i+1
no
si
xn=tabla(i+1,1) , wn=tabla(i+1,3)
waux=tabla(i+1,2)
[tabla(redt,2) , tabla(redt,3)] =
Eulergragg(f, h / base^i , waux, wn, xn, x(t))
tabla(redt,1)=x(t),waux=tabla(redt,2)
wn=tabla/redt,3),g(redt)=wn
waux=wini
g(redt)=euler(f, h/base^i , wini, xini, x(t))
tabla(redt,1)=x(t),tabla(redt,2)=wini
tabla(redt,3)=g(redt)
xn=x(t) , wn=g(redt)
aux=q(1) ,exp=expord ,inc=redt
g(inc-1)=derror(base,exp,g(inc-1),g(inc))
exp=exp+incexp;
inc=inc-1
si
inc ≥ 2
no
resul(t)=g(1)
t ≤ length(x)
si
result
no

4. MÉTODO DE RUNGE-KUTTA

Quizás uno de los métodos numéricos más exactos para encontrar soluciones aproximadas de ecuaciones diferenciales sea el método de Runge-Kutta.

Consiste en determinar las constantes apropiadas de forma que la fórmula $y_{n+1}=y_n+ak_1+bk_2+ck_3+dk_4$ coincida con el desarrollo de Taylor de 5 términos. Obteniéndose la expresión general $y_{n+1}=y_n+1/6(k_1+2k_2+2k_3+k_4)$ siendo

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2})$$

$$k_3 = hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2})$$

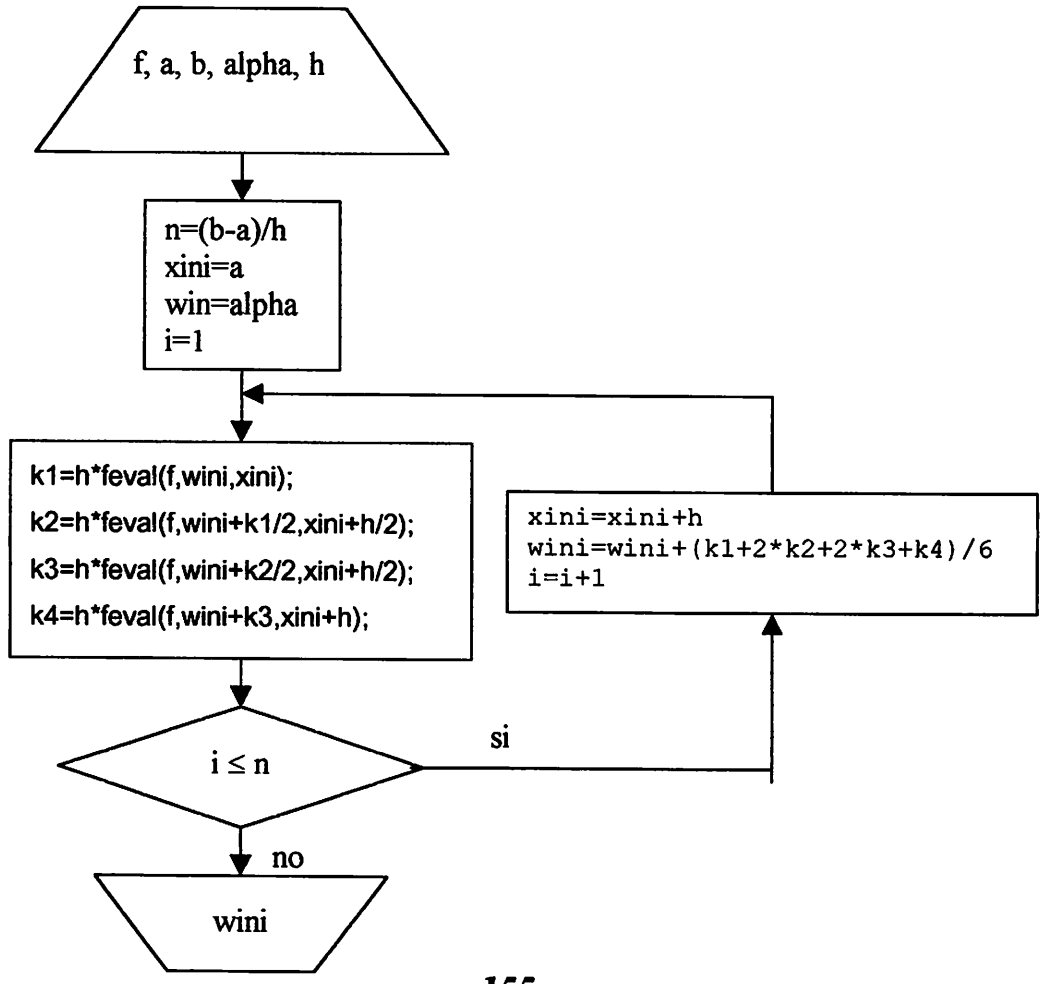
$$k_4 = hf(x_n + h, y_n + k_3)$$

Obsérvese que k_2 depende de k_1; k_3 depende de k_2 y k_4 de k_3.

Se demuestra que la forma del error es:

$M=N(h)+k_1h^4+k_2h^5+k_3h^6+...$ Siendo N(h) la aproximación al valor real M.

DIAGRAMA DE FLUJO DE RUNGE-KUTTA



5. MÉTODOS MULTIPASO

Los métodos vistos hasta ahora se llaman métodos de un paso, ya que la aproximación para el punto de red t_{i+1} involucra solo información de unos de los puntos de red anteriores t_i. A pesar de que estos métodos generalmente usan información de evaluación funcional en puntos entre t_i y t_{i+1} , no retienen esta información para usarlas en aproximaciones futuras. Consecuentemente toda la información utilizada por estos métodos se obtiene dentro del intervalo en el cual se está aproximando la solución.

Como se dispone de la solución aproximada en cada uno de los puntos de red t_0, t_1, ... , t_i antes de que se obtenga la aproximación en t_{i+1} y debido a que el error $|w_j-y(t_j)|$ tiende a incrementar con j, parece razonable desarrollar métodos que usen estos datos anteriores, que son más precisos cuando se aproxima la solución en t_{i+1}.

Los métodos que utilizan la aproximación en más de uno de los puntos anteriores de red para determinar la aproximación en el punto siguiente se llaman métodos multipaso.

Los métodos multipaso que vamos a estudiar son el método de Adams-Bashforth (4 pasos) y el de Adams-Moulton (3 pasos), que combinados conforman un método denominado *predictor-corrector*. Estos son los siguientes:

Adams-Bashforth

$$\begin{cases} w_0 = \alpha \\ w_1, w_2, w_3, \quad se \quad calculan \quad por \quad Runge-Kutta \quad de \quad orden \quad 4 \\ w_{i+1} = w_i + \dfrac{h}{24}\left[55f_i - 59f_{i-1} + 37f_{i-2} - 9f_{i-3}\right], i = 3,4,\ldots,n \end{cases}$$

Adams-Moulton

$$\begin{cases} w_0 = \alpha \\ w_1, w_2, \quad se \quad calculan \quad por \quad Runge-Kutta \quad de \quad orden \quad 4 \\ w_{i+1} = w_i + \dfrac{h}{24}\left[9f_{i+1} + 19f_i - 5f_{i-1} + f_{i-2}\right], i = 2,3,4,\ldots,n \end{cases}$$

Siendo $f_j=f(t_j,w_j)$ para j=i, i-1, i-2, i-3.

DIAGRAMA DE FLUJO DE ADAMS-BASHFORTH

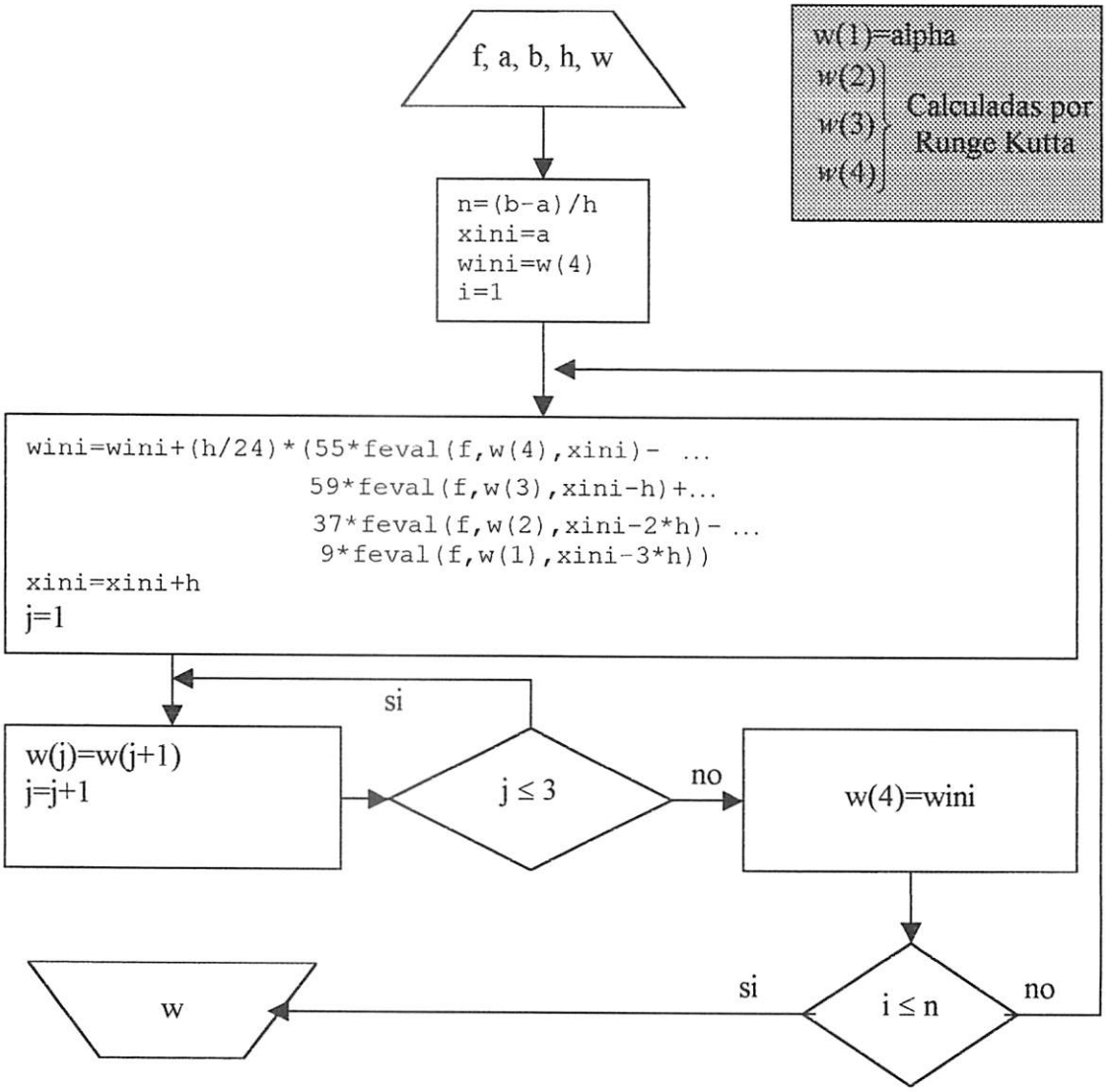


DIAGRAMA DE FLUJO DE ADAMS-MOULTON

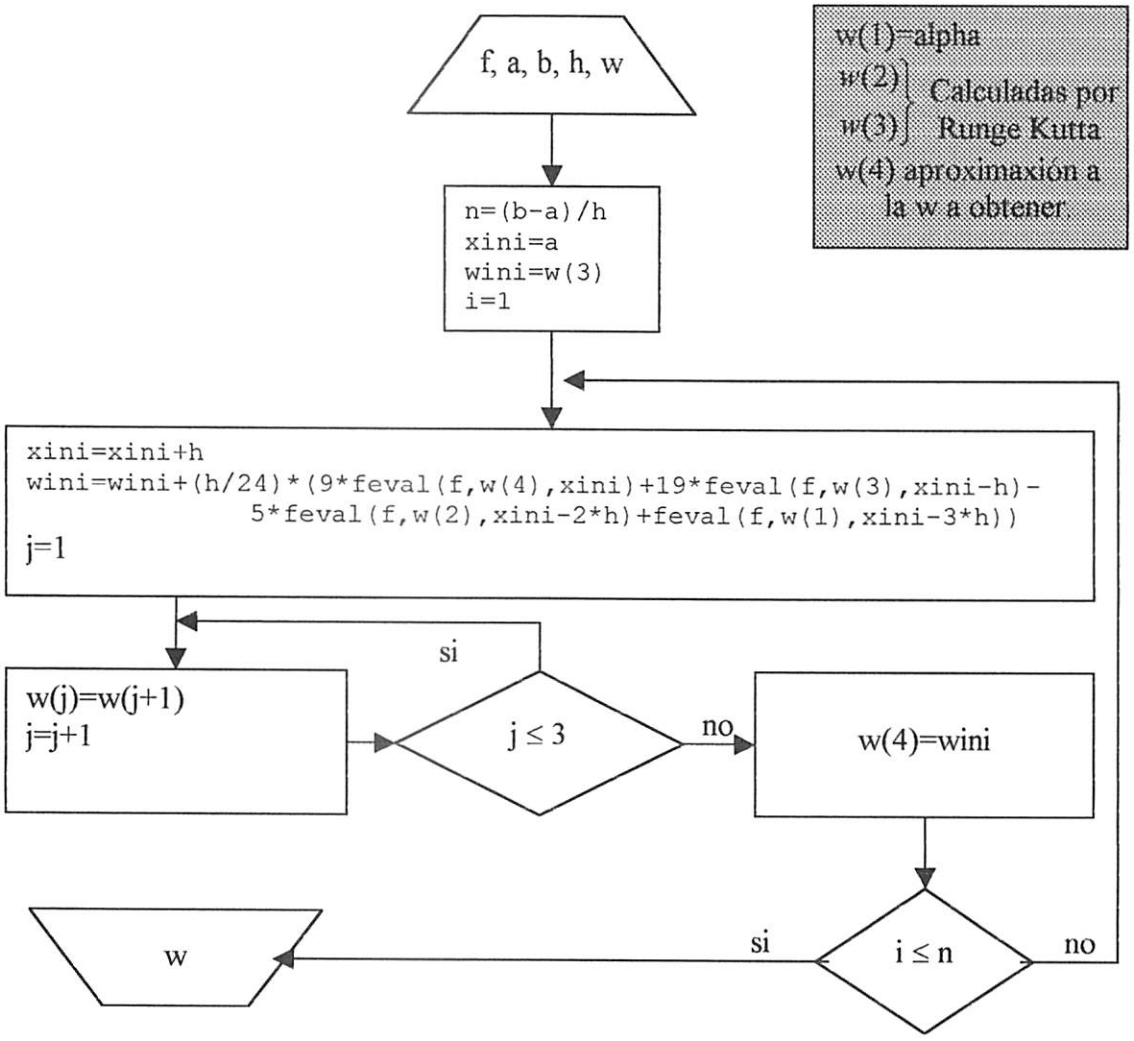
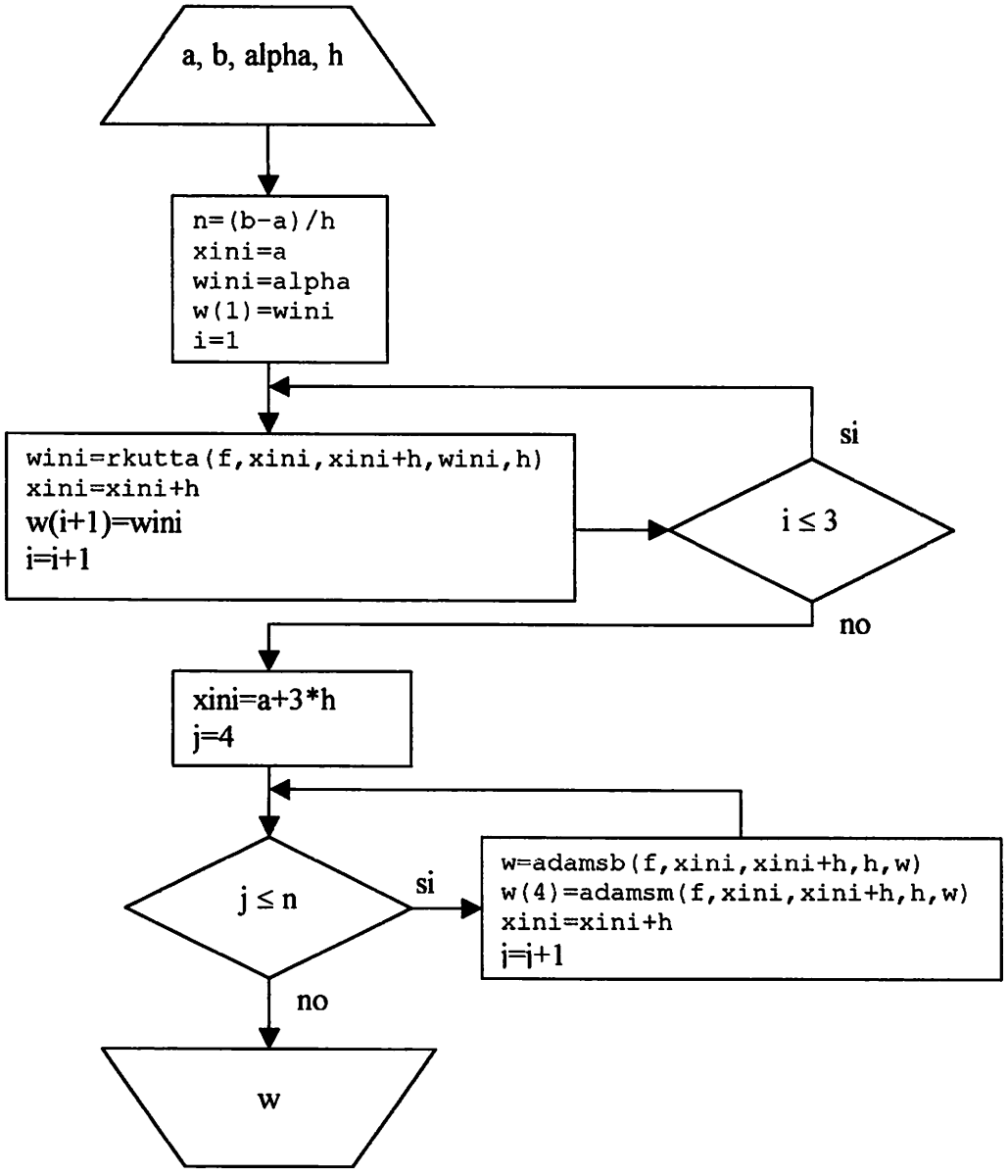


DIAGRAMA DE FLUJO DE PREDICTOR-CORRECTOR



6. EVALUACIÓN DE LOS MÉTODOS

Pasamos a evaluar los distintos métodos para los problemas de valor inicial siguientes:

- $y'=e^x$, $y(0) = 1$
- $y'= -y+x^2+1$, $y(0) = 1$

pudiendo comparar en el primer caso la soluciones obtenidas con la función real ya que conocemos la f(x) buscada (e^x).

Como podremos ver en las siguientes páginas, en las que se estudian los distintos métodos para h diferentes, las soluciones obtenidas son mejores cuando la h es más pequeña (*si la h es demasiado pequeña la computadora puede cometer errores de cálculo, que harán que la solución no sea correcta*).

Con las soluciones obtenidas intentaremos reconstruir la funciones originales interpolando los resultados. Para el primer caso podremos compararla gráficamente con la función real, mientras que para la segunda solo podremos poner la reconstrucción ya que no conocemos la real.

6.1. EULER (Tolerancia de 10^{-10})

- **$y'= e^x$, $y(0) = 1$**

PUNTO	H	RESULTADO	OPERACIONES
0.5	0.1	1.64872127070012	1739
	0.01	1.64872127070027	3826
1	0.1	2.71828182845903	3314
	0.01	2.71828182845943	7576
1.5	0.1	4.48168907033806	4889
	0.01	4.48168907033884	11326
2	0.1	7.38905609893062	6464
	0.01	7.38905609893205	15076

- $y' = -y+x^2+1, \quad y(0) = 1$

PUNTO	H	RESULTADO	OPERACIONES
0.6	0.1	1.06237672781195	5554
	0.01	1.06237672781194	13136
1	0.1	1.26424111765698	4574
	0.01	1.26424111765637	10576
1.6	0.1	1.95620696401078	7220
	0.01	1.95620696401068	34836
2	0.1	2.72932943352687	8984
	0.01	2.72932943352678	43516

6.2. GRAGG (Tolerancia de 10^{-5})

- $y' = e^x, \quad y(0) = 1$

PUNTO	H	RESULTADO	OPERACIONES
0.5	0.1	1.64871162303396	102995
	0.01	1.64871355252856	128106
1	0.1	2.71827543996491	819986
	0.01	2.71827671766211	1024024
1.5	0.1	4.48168259795672	2458481
	0.01	4.48168389243277	3071870
2	0.1	7.38905016037525	6554584
	0.01	7.38904659724129	4095620

- $y' = -y+x^2+1, \quad y(0) = 1$

PUNTO	H	RESULTADO	OPERACIONES
0.6	0.1	1.06236947622840	86498
	0.01	1.06237092642893	107384
1	0.1	1.26423325816429	287270
	0.01	1.26423483003138	358056
1.6	0.1	1.95619989886952	918116
	0.01	1.95620131189069	1146196
2	0.1	2.72932059975303	1147464
	0.01	2.72932236649894	1432636

6.3. RUNGE-KUTTA

- $y' = e^x$, $y(0) = 1$

PUNTO	h	RESULTADO	OPERACIONES
0.5	0.1	1.64872129321847	130
	0.01	1.64872127070238	1300
1	0.1	2.71828188810386	260
	0.01	2.71828182846501	2600
1.5	0.1	4.48168919119408	390
	0.01	4.48168907035016	3900
2	0.1	7.38905632070687	520
	0.01	7.38905609895284	5200

- $y' = -y+x^2+1$, $y(0) = 1$

PUNTO	H	RESULTADO	OPERACIONES
0.6	0.1	1.06237711900530	204
	0.01	1.06237672785108	2040
1	0.1	1.26424183503644	340
	0.01	1.26424111772764	3400
1.6	0.1	1.95620812601111	544
	0.01	1.95620696412350	5440
2	0.1	2.72933083611249	680
	0.01	2.72932943366233	6800

6.4. ADAMS-BASHFORTH

- **$y' = e^x$, $y(0) = 1$**

PUNTO	h	RESULTADO	OPERACIONES
0.5	0.1	1.64871235374760	124
	0.01	1.63846807465130	1059
1	0.1	2.71824095630664	239
	0.01	2.70802862873905	2309
1.5	0.1	4.48159551296792	354
	0.01	4.47143586456536	3459
2	0.1	7.38887567832137	469
	0.01	7.37880288317870	4609

- **$y' = -y+x^2+1$, $y(0) = 1$**

PUNTO	H	RESULTADO	OPERACIONES
0.6	0.1	1.06236330516179	195
	0.01	1.06237319209741	1869
1	0.1	1.26422016549419	319
	0.01	1.26423874655473	3109
1.6	0.1	1.95618559417231	505
	0.01	1.95620566186427	4969
2	0.1	2.72931069581489	629
	0.01	2.72932856028858	6209

6.5. ADAMS-MOULTON

- $y' = e^x$, $y(0) = 1$

PUNTO	h	APROXIMACIÓN Al a+3h	RESULTADO	OPERs
0.5	0.1	*1.34985881972026*	1.64872232103133	168
	0.01	*1.33960561255299*	1.64872127086475	1158
1	0.1	*1.34985881972026*	2.71828548848979	278
	0.01	*1.33960561255299*	2.71828182890370	2258
1.5	0.1	*1.34985881972026*	4.48169703303595	388
	0.01	*1.33960561255299*	2.71828182890370	3358
2	0.1	*1.34985881972026*	7.38907115552736	498
	0.01	*1.33960561255299*	7.38905610059819	4458

- $y' = -y+x^2+1$, $y(0) = 1$

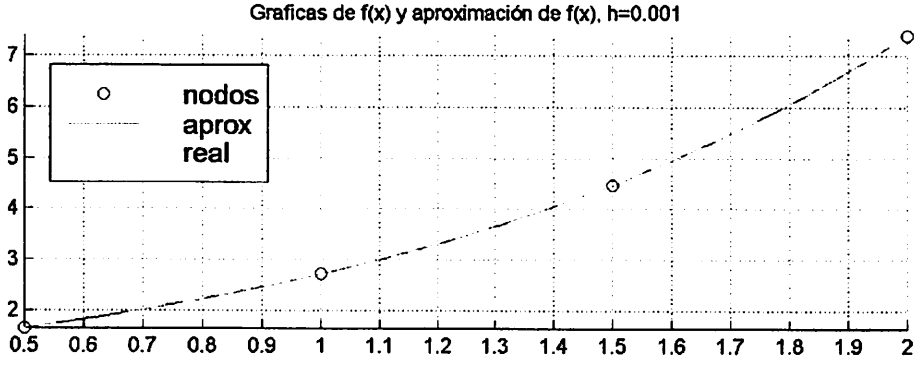
PUNTO	h	APROXIMACIÓN	RESULTADO	OPERs
0.6	0.1	*1.00836372340773*	1.06631638925664	246
	0.01	*1.00835878749499*	1.06285984043730	1866
1	0.1	*1.00836372340773*	1.28339036654497	366
	0.01	*1.00835878749499*	1.26627170095737	3066
1.6	0.1	*1.00836372340773*	2.02079833634653	546
	0.01	*1.00835878749499*	1.96272191058320	4866
2	0.1	*1.00836372340773*	2.83789095325320	666
	0.01	*1.00835878749499*	2.74014988798163	6066

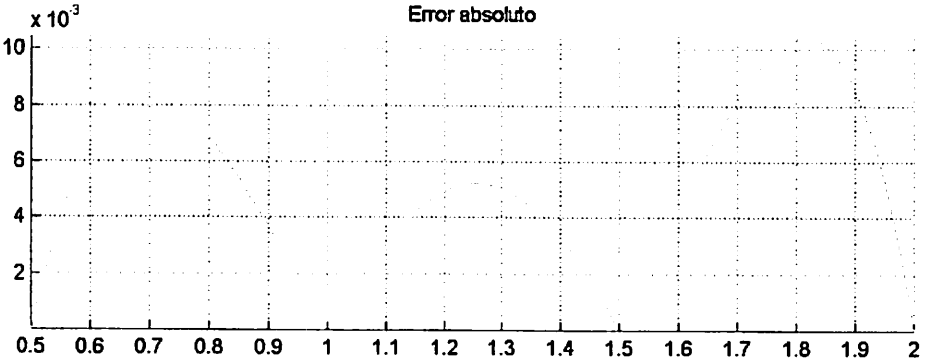
6.6. PREDICTOR-CORRECTOR

Como comentamos anteriormente vamos a reconstruir la función original (aproximadamente) y para ello usaremos los resultados obtenidos por predictor corrector por ser un buen método y además rápido. Primero lo hacemos en el intervalo [0, 2] con los nodos [0.5, 1, 1.5, 2] para f y [0.6, 1, 1.6, 2] para g, y posteriormente se hará para el intervalo [0, 10] con nodos de Chebyshev para las dos funciones.

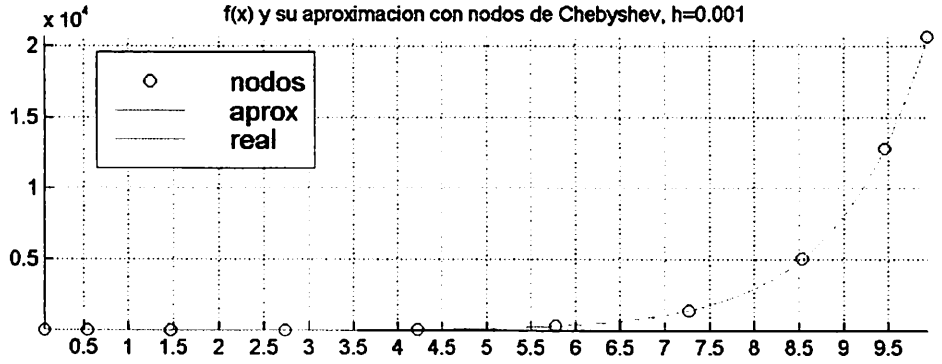
- **$y' = e^x$, $y(0) = 1$**

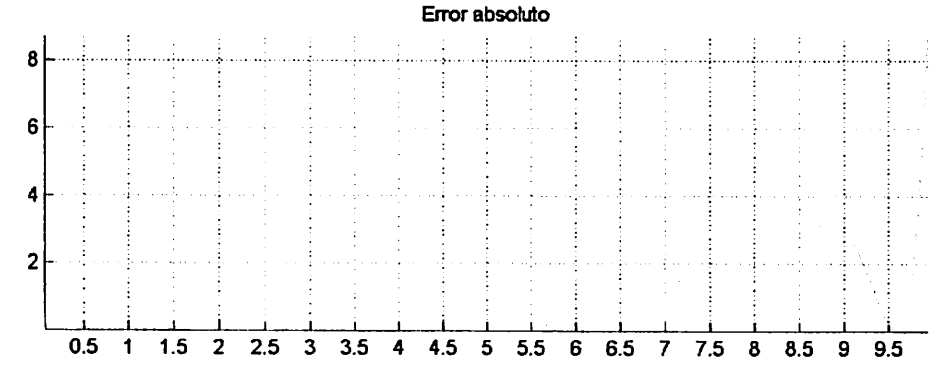
PUNTO	h	RESULTADO	OPERACIONES
0.5	0.1	1.64872201206091	199
	0.001	1.64872127070014	25939
1	0.1	2.71828517951937	459
	0.001	2.71828182845909	51939
1.5	0.1	4.48169672406553	719
	0.001	4.48168907033810	77939
2	0.1	7.38907084655694	979
	0.001	7.38905609893053	103939





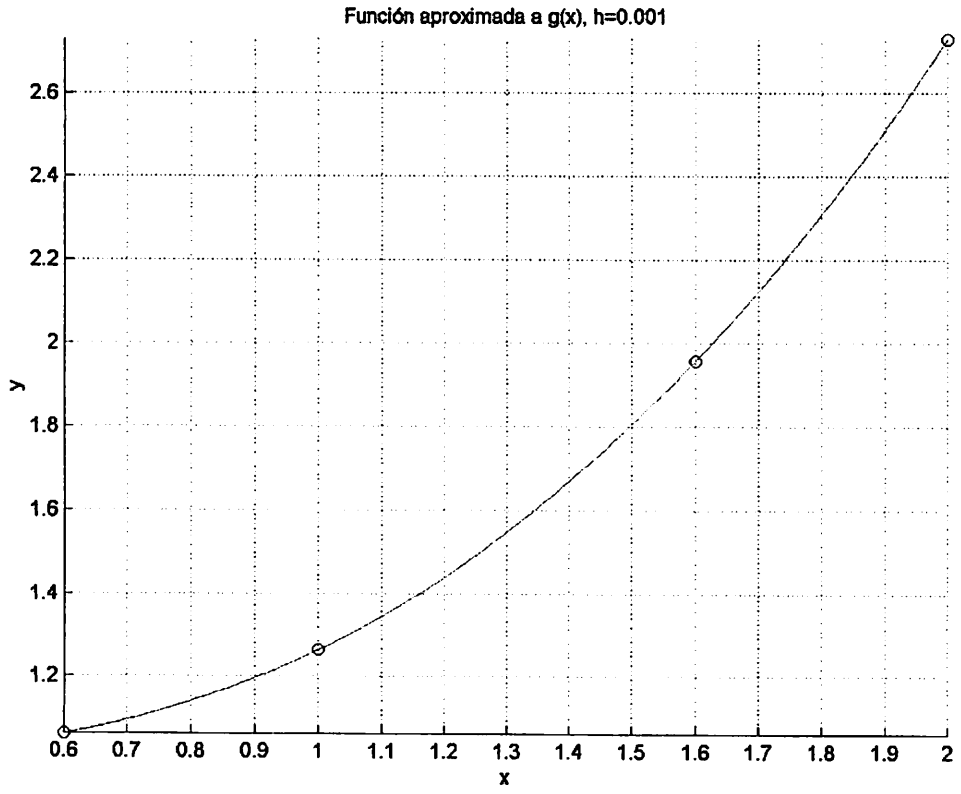
NODOS DE CHEBYSHEV	IMÁGENES APROXIMADAS
0.06155829702431	0.00010628989142e+004
0.54496737905816	0.00017228846360e+004
1.46446609406726	0.00043232178601e+004
2.73004750130227	0.00153328870199e+004
4.21782767479885	0.00678296896813e+004
5.78217232520116	0.03244073567484e+004
7.26995249869773	0.14351146206201e+004
8.53553390593274	0.50898314752120e+004
9.45503262094184	1.27718652071422e+004
9.93844170297569	2.07022982570361e+004



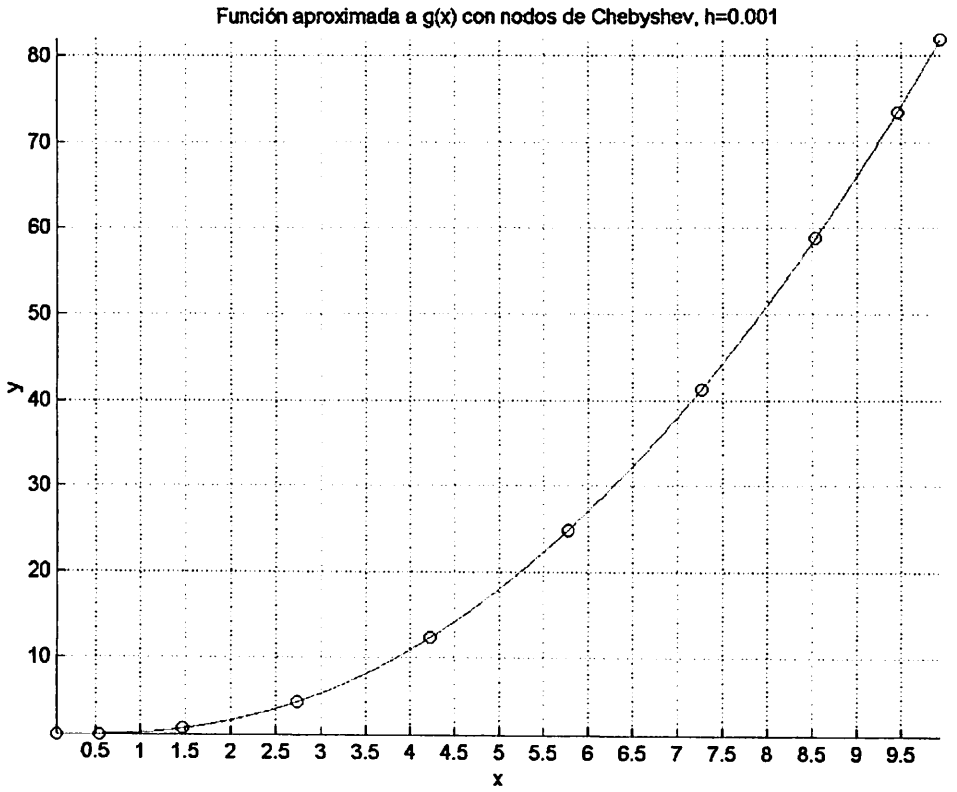


- $y' = -y+x^2+1, \quad y(0) = 1$

PUNTO	H	RESULTADO	OPERACIONES
0.6	0.1	1.06237835391835	323
	0.001	1.06237672781196	40715
1	0.1	1.26424354882897	595
	0.001	1.26424111765713	67915
1.6	0.1	1.95620940231879	1003
	0.001	1.95620696401066	108715
2	0.1	2.72933156135155	1275
	0.001	2.72932943352666	135915



NODOS DE CHEBYSHEV	IMÁGENES APROXIMADAS
0.06155829702431	1.00007452044798
0.54496737905816	1.04709216971853
1.46446609406726	1.75267762990734
2.73004750130227	4.86246142066330
4.21782767479885	12.31960338670131
5.78217232520116	24.86135891169885
7.26995249869773	41.29896738309826
8.53553390593274	58.77583205969336
9.45503262094184	73.48686840581027
9.93844170297569	81.88774739237451



7. IMPLEMENTACIÓN DE LOS MÉTODOS

- FUNCIONES AUXILIARES

```
function salida=f(w,x)
%function salida=f(w,x)
%
% funcion f'=exp(x)
format long
salida=exp(x);

function salida=g(w,x)
%function salida=g(w,x)
%
% funcion g'=-w+x.^2+1
format long
salida=-w+x.^2+1;

function dibujar(x,y,x0,y0,sierror)
%function dibujar(x,y,x0,y0,sierror)
%
% -x  son los nodos iniciales a interpolar.
% -y  son las imagenes de los nodos iniciales.
% -x0 son los nodos a interpolar (evaluando en polinomio resultante).
% -y0 imagenes de los nodos obtenidas por un algoritmo de interpolacion.
% -sierror vble. booleana que indica si se quiere graficar el error absoluto.
if(sierror)
  subplot(2,1,1)
  ejes(min(x0),max(x0));
  title('Graficas real y aproximada(nodos igualmente espaciados)');
  %title('Graficas real y aproximada para nodos de Chebyshev');
end
ejes(min(x0),max(x0));
hold on,grid on;
plot(x,y,'k O')          %Nodos
plot(x0,y0,'b')          %Aproximacion
plot(x0,faux(x0),'r')    %Real
axis tight
if(sierror)
  legend('nodos','aprox','real',0);
  subplot(2,1,2),hold on,grid on;
  ejes(min(x0),max(x0));
  title('Error absoluto');
  ydex0=abs(faux(x0)-y0);
  plot(x0,ydex0,'g');
  axis tight
end


function ejes(a,b)
%function ejes(a,b)
%
% En su interior se cambia el incremento visual en eje x (grid)
set(gca,'YTickMode','auto');
set(gca,'XTickMode','manual');
set(gca,'XTick',a:0.1:b);
```

```
function y=derror(base,exp,n1,n2)
%function y=derror(base,exp,n1,n2)
%
%n1 N(h)
%n2 N(h/2)
%
format long
%y= (( base.^exp)*n2-n1)/(base.^(exp)-1);
y=n2+((n2-n1)/((base.^(exp))-1));


function pinta(a,b,xvar)
%function pinta(a,b,xvar)
%
% -a  intervalo [a, b]
% -b
% -xvar son los nodos
for i=1:length(xvar)
  w=predcorr('f',0,xvar(i),1,0.001);
  yaprox(i)=w(4);
end
lagrange(xvar,yaprox,1);
yaprox
```

- MÉTODOS PARA LA RESOLUCIÓN DE PROBLEMAS DE VALOR INICIAL

```
function y=Euler(f,h,wini,xmin,xmax)
%function y=Euler(f,h,wini,xmin,xmax)
%
% - f       ---> f'
% -h        ---> valor de h .
% -wini     ---> w inicial para el cálculo de Euler.
% -xini     ---> valor de x minimo que corresponda con el wini .
% -xmax     ---> valor de x máximo(valor al que se quiere aproximar).
format long
w=wini;x=xmin;
n=(xmax-xmin)/h;
for i=1:n
  w=w+h*feval(f,w,x);
  x=xmin+i*h;
end
y=w;
```

```
function [resul]=ec_dif_euler(f,x,xini,wini,TOL,baseord,expord,incexp,h)
% function [result]=ec_dif_euler(f,x,xini,wini,TOL,baseord,expord,incexp,h)
%
% -f        ---> calcula derivada de la función f
% -x        ---> vector con los puntos donde se quiere evaluar.
% -xini ,wini---> punto inicial y w inicial en dicho punto
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
% -       M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
% -       N(h)= A + (A-B)/((baseord.^expord)-1)
% -baseord   ---> base del orden del error.
% -expord    ---> exponente del orden del error.
% -incexp    ---> incremento del exponente del error(1)
% -h         ---> h incial.
format long
base=baseord;TOL=10.^(-TOL);
hini=h;
tabla(1,1)=xini;
tabla(1,2)=wini;
%iniciomedida;
for t=1:length(x)
  i=0;
  redt=1;%reducciones + tamaño vector
  h=hini;
  g(1)=Euler(f,h,tabla(1,2),tabla(1,1),x(t));
  tabla(1,1)=x(t);
  tabla(1,2)=g(1);
  aux=g(1)+2;.
  while (abs(aux-g(1))>=TOL  )
    redt=redt+1;
    i=i+1;
     if(length(tabla(:,1))<i+1)
       xn=xini;wn=wini;
    else
       xn=tabla(i+1,1);wn=tabla(i+1,2);
    end
    g(redt)=Euler(f,h/(base.^i),wn,xn,x(t));
    tabla(redt,1)=x(t);
    tabla(redt,2)=g(redt);
    aux=g(1);%sirve para comparación .
    exp=expord;
    for inc=redt:-1:2
      g(inc-1)=derror(base,exp,g(inc-1),g(inc));
      exp=exp+incexp;
    end
  end
  resul(t)=g(1);
  g(1)
end
%finmedida(0);
```

```
function [w0,wn]=Euler_gragg(f,h,waux,wini,xmin,xmax)
%function [w0,wn]=Euler_gragg(f,h,waux,wini,xmin,xmax)
%
% - f       ---> función.
% -h        ---> valor de h .
% -waux     ---> En la primera it --> w inicial (alpha)...En las demas it --> w auxiliar
% -wini     --->                          primer w  (euler)...              w inicial
% -xini     ---> valor de x minimo que corresponda con el wini .
% -xmax     ---> valor de x máximo(valor al que se quiere aproximar).
%
format long
k=1;
w0=waux;%wini
x=xmin;
n=(xmax-xmin)/h;
w1=wini;
for k=1:n
  wn=w0+2*h*feval(f,w1,x);
  x=xmin+k*h;
  w0=w1;
  w1=wn;
end

function [resul]=ec_dif_gragg(f,x,xini,wini,TOL,baseord,expord,incexp,h)
% function [result]=ec_dif_gragg(f,x,xini,wini,TOL,baseord,expord,incexp,h)
%
% -f        ---> calcula derivada de la función f
% -x        ---> vector con los puntos donde se quiere evaluar.
% -xini ,wini---> punto inicial y w inicial en dicho punto
% -10.-^tol  ---> tolerancia exigida entre [N(h),N*(h)];[N(h)*,N**(h)], ...
%
% -      M= N(h) + k1*h.^expord + k2*h.^(expord+incexp)+ ...
%
% -      N(h)= A + (A-B)/((baseord.^expord)-1)
%
% -baseord   ---> base del orden del error.
% -expord    ---> exponente del orden del error.
% -incexp    ---> incremento del exponente del error(2)
% -h         ---> h incial.
format long
base=baseord;
TOL=10.^(-TOL);
hini=h;
tabla(1,1)=xini;
tabla(1,2)=wini;
iniciomedida;
for t=1:length(x)
  i=0;
  redt=1;%reducciones + tamaño vector
  h=hini;
%  'Otro x(t)'
  if (t==1)
    %'Euler'
    g(1)=Euler(f,h,tabla(1,2),tabla(1,1),x(t));
    tabla(1,1)=x(t);
    tabla(1,2)=wini;  %Guardo w auxiliar (alpha)
    tabla(1,3)=g(1);  %Guardo w de x(t), o sea, w inicial para euler-gragg
```

```
  else
    xn=tabla(i+1,1);
    waux=tabla(i+1,2);
    wn=tabla(i+1,3);
    [tabla(redt,2),tabla(redt,3)]=Euler_gragg(f,h,waux,wn,xn,x(t));
    tabla(redt,1)=x(t);
    waux=tabla(redt,2);
    wn=tabla(redt,3);
    g(redt)=wn;
  end
  aux=g(1)+2;%para entrar en bucle.
  while (abs(aux-g(1))>=TOL)
    redt=redt+1;
    i=i+1;
    if(length(tabla(:,1))<i+1)
%       'Nueva linea --- Euler'
      waux=wini;
      g(redt)=Euler(f,h/(base.^i),wini,xini,x(t));% Euler para la nueva altura
      tabla(redt,1)=x(t);   %Nuevo punto
      tabla(redt,3)=g(redt); %Guardo w de x(t), o sea, nuevo punto
      tabla(redt,2)=wini;   %Guardo w auxiliar (alpha)
      xn=x(t);
      wn=g(redt);
     else
      xn=tabla(i+1,1);     %Si hay valores en la tabla de x, w y waux se usan y
      wn=tabla(i+1,3);      %no se empieza desde el principio.
      waux=tabla(i+1,2);
      [tabla(redt,2),tabla(redt,3)]=Euler_gragg(f,h/(base.^i),waux,wn,xn,x(t));
      tabla(redt,1)=x(t);    % Euler_gragg te devuelve 2 w, el inicial y el anterior
      waux=tabla(redt,2);   % waux es el w anterior (para posterior uso con gragg)
      wn=tabla(redt,3);      % wn es el w inicial (para posterior uso con gragg)
      g(redt)=wn;
    end
    aux=g(1);%sirve para comparación . Es el ultimo valor mejor aproximado.
    exp=expord;
    for inc=redt:-1:2
      g(inc-1)=derror(base,exp,g(inc-1),g(inc));
      exp=exp+incexp;
    end
  end
  resul(t)=g(1);   g(1)
end
finmedida(0);

function wini=rkutta(f,a,b,alpha,h)
% function wini=rkutta(f,a,b,alpha,h)
format long
n=(b-a)/h;
xini=a;wini=alpha;
%iniciomedida
for i=1:1:n
  k1=h*feval(f,wini,xini);
    k2=h*feval(f,wini+k1/2,xini+h/2);
    k3=h*feval(f,wini+k2/2,xini+h/2);
    k4=h*feval(f,wini+k3,xini+h);
  xini=xini+h;
  wini=wini+(k1+2*k2+2*k3+k4)/6;
end
%finmedida(0);
```

```
function wf=adamsb(f,a,b,h,w)
%function wf=adamsb(f,a,b,h,w)
%
% -f es la funcion derivada conocida
% -a es el punto del que conocemos alpha=f*(a)
% -b es el punto del que queremos conocer su w(b) ~ f*(b)
% -h es la altura fija
% -w es un vector con las 4 aproximaciones anteriores (wi i=0,1,2 y 3)
%
% Siendo f* es la funcion que queremos conocer, y f su ec.diferencial
n=(b-a)/h;
xini=a;wini=w(4);
%iniciomedida;
for i=1:n
  wini=wini+(h/24)*(  55*feval(f,w(4),xini)-59*feval(f,w(3),xini-h)+37*feval(f,w(2),xini-2*h)...
     -9*feval(f,w(1),xini-3*h));
  xini=xini+h;
  for j=1:3
       w(j)=w(j+1);
  end
  w(4)=wini;
end
wf=w;
%finmedida(0);

function wb=ejeadamsb(f,a,b,alpha,h)
%function wb=ejeadamsb(f,a,b,alpha,h)
%
% -f es la funcion derivada conocida
% -a es el punto del que conocemos alpha=f*(a)
% -b es el punto del que queremos conocer su w(b) ~ f*(b)
% -h es la altura fija
%
% Siendo f* es la funcion que queremos conocer, y f su ec.diferencial
n=(b-a)/h;  xini=a;   wini=alpha;   w(1)=wini;
for i=1:3
  wini=rkutta(f,xini,xini+h,wini,h);   xini=xini+h;   w(i+1)=wini;
end
wb=adamsb(f,a+3*h,b,h,w);

function wini=adamsm(f,a,b,h,w)
%function wini=adamsm(f,a,b,h,w)
%
% -f es la funcion derivada conocida
% -a es el punto del que conocemos alpha=f*(a)
% -b es el punto del que queremos conocer su w(b) ~ f*(b)
% -h es la altura fija
% -w vector con w0, w1, w2 y waproximada a w3
%
% Siendo f* es la funcion que queremos conocer, y f su ec.diferencial
n=(b-a)/h;  xini=a;  wini=w(3);
for i=1:n
  xini=xini+h;
  wini=wini+(h/24)*(9*feval(f,w(4),xini)+19*feval(f,w(3),xini-h)-5*feval(f,w(2),xini...
     -2*h)+feval(f,w(1),xini-3*h));
  for j=1:3
       w(j)=w(j+1);
  end
  w(4)=wini;
end
```

```
function wb=ejeadamsm(f,a,b,alpha,h,waprox)
%function wb=ejeadamsm(f,a,b,alpha,h,waprox)
%
% -f es la funcion derivada conocida
% -a es el punto del que conocemos alpha=f*(a)
% -b es el punto del que queremos conocer su w(b) ~ f*(b)
% -h es la altura fija
% -waprox al punto a+3*h
%
% Siendo f* es la funcion que queremos conocer, y f su ec.diferencial
n=(b-a)/h;
xini=a; wini=alpha; w(1)=wini;
% iniciomedida;
for i=1:2
  wini=rkutta(f,xini,xini+h,wini,h);   xini=xini+h;   w(i+1)=wini;
end
w(4)=waprox;
wb=adamsm(f,a+2*h,b,h,w);
%finmedida(0);

function w=predcorr(f,a,b,alpha,h)
%function w=predcorr(f,a,b,alpha,h)
%
% -f es la funcion derivada conocida
% -a es el punto del que conocemos alpha=f*(a)
% -b es el punto del que queremos conocer su w(b) ~ f*(b)
% -h es la altura fija
%
% Siendo f* es la funcion que queremos conocer, y f su ec.diferencial
n=(b-a)/h;
xini=a;
wini=alpha;
w(1)=wini;
%iniciomedida;
for i=1:3
  wini=rkutta(f,xini,xini+h,wini,h);
  xini=xini+h;
  w(i+1)=wini;
end
xini=a+3*h;
for i=4:n
  w=adamsb(f,xini,xini+h,h,w);
  w(4)=adamsm(f,xini,xini+h,h,w);
  xini=xini+h;
end
%finmedida(0);
```

- FUNCIONES PARA MEDIDAS

```
function iniciomedida
%CONTADOR DE OPERACIONES A 0 E INCIALIZO EL TEMPORIZADOR
flops(0),tic

function finmedida(h)
% IMPRIMO EL NUMERO DE OPERACIONES EMPLEADAS
disp(' ');
disp('NUMERO DE OPERACIONES:'),disp(flops);
disp('TIEMPO EMPLEADO (segundos):'),disp(toc);
disp('ITERACIONES:'),disp(h);
```

BIBLIOGRAFÍA

ANÁLISIS NUMÉRICO
Richard L. Burden
J. Douglas Faires

MÉTODOS NUMÉRICOS
Félix García Merayo
Antonio Nevot Luna
Publicaciones de la Universidad Pontificia de Comillas

APUNTES DE ANÁLISIS NUMÉRICO I
DE LA UNIVERSIDAD DE HUELVA