Universidad Diego Portales
Ingeniería Industrial

Tarea 1

Métodos numéricos para la resolución de problemas matemáticos

Alumnas

Profesor
Fecha

Índice de contenido

A.-Introducción.

El presente trabajo consiste en aplicar los conocimientos de métodos numéricos para encontrar soluciones de funciones mediante algoritmos e iteraciones. Se estudia además la eficiencia de cada método comparándolo con soluciones alternativas, las condiciones que deben tener los métodos y funciones a estudiar para lograr alguna solución óptima del problema.

La primera parte consiste en estudiar doce métodos iterativos para encontrar el cero de cualquier función y se estima el grado de convergencia de cada uno de estos. Para probar la eficacia de cada método se ensayan con las siete funciones dispuestas en la guia y se encuentra el cero más próximo al punto inicial de iteración.

La segunda parte consiste en el estudio de una matriz con caracteríticas particulares y la ecuación del tipo Ax=b. Se estudian métodos directos para obtener la solución como Chaloskyi o métodos iterativos como Jacobi, Gauss Siedel o Relajación Sucesiva. Además se incluye el estudio frente a perturbaciones, en el vector b, de la solución.

Dado que cada método o función esta definida en forma general, es posible trasladar a un sistema computacional y escribir funciones que se pueden invocar desde un shell y encontrar soluciones particulares en diversos casos, para probar la funcionalidad se deben tener las soluciones adecuadas a casos conocidos. Se utiliza la herramienta matlab, cuyo lenguaje de alto nivel permite una programación eficiente y simple.

B.- Desarrollo

I.a-Métodos de Convergencia.

Se establecen los distintos métodos de punto fijo, para los cuales se escribe una función por cada uno, de modo que se llama alguna de ellas para determinar el cero en cierta función determinada. La estructura es simple, primero se establece un punto de partida xo, luego se ingresa la función definida, luego la cantidad de iteraciones. El resultado de la función depende del método utilizado y la cantidad de iteraciones, algunos pueden resultar no convergentes o más lentos que otros.

Las condiciones para el estudio de la convergencia de cada método es que las funciones sean continuas y derivable la cantidad necesaria de veces que requiera cada método. De entre los 12 métodos se observa que se requiere a lo más de funciones dos veces derivables.

Primero, se establece la función Lf(x), su código es:

```
function lfx=lefx(f)

g=diff(f);
h=diff(g);
lfx=f*h/g^2;
```

A continuación el Código de cada método, con la condición límite para detener iteración:

$$\frac{|x_{n+1}-x_n|}{|x_n|} \leqslant 10^{-10}$$

1.-Método Newton

```
function [Xc,no,co,fx]=newton(xo,f,n)
g=diff(f);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-fo/go;
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
Xc=x;
syms x;
x=Xc;
fx=eval(f);
```

2.-Método Schroder

```
function [sc,no,co,fx]=schroder(xo,f,n)
g=diff(f);
h=diff(g);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    ho=eval(h);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-fo*go/(go^2-fo*ho);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
sc=x;
syms x;
x=sc;
fx=eval(f);
```

3.-Método Whittaker

```
function [wh,no,co,fx]=whittaker(xo,f,n)
g=diff(f);
h=diff(g);
lfx=lefx(f);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    ho=eval(h);
    lfxo=eval(lfx);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-(fo/(2*go))*(2-lfxo);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
wh=x;
syms x;
x=wh;
fx=eval(f);
```

4.-Método Halley

```
function [ha,no,co,fx]=halley(xo,f,n)
g=diff(f);
h=diff(g);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    ho=eval(h);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-2*fo*go/(2*go^2 - fo*ho);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
ha=x;
syms x;
x=ha;
fx=eval(f);
```

5.-Método Chebysev

```
function [ch,no,co,fx]=chebysev(xo,f,n)
g=diff(f);
lfx=lefx(f);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    lfxo=eval(lfx);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-(fo/go)*(1+lfxo/2);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
ch=x;
syms x;
x=ch;
fx=eval(f);
```

6.-Método Super Halley

```
function [sha,no,co,fx]=superhalley(xo,f,n)
g=diff(f);
lfx=lefx(f);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    lfxo=eval(lfx);
    xo=x;
    co(:,i)=xo;
    syms x;
    x=xo-(fo/(2*go))*(2-lfxo)/(1-lfxo);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
sha=x;
syms x;
x=sha;
fx=eval(f);
```

7.-Método Stirling

```
function [st,no,co,fx]=stirling(xo,f,n)
syms x;
g=diff(f);
x=xo;
for i=1:1:n
    fo=eval(f);
    xo=x;
    co(:,i)=xo;
    x1=xo;
    syms x;
    x=xo-fo;
    g1=eval(g);
    xo=x1;
    syms x;
    x=xo-(fo/g1);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
st=x;
syms x;
x=st;
fx=eval(f);
```

8.-Método Steffensen

```
function [ste,no,co,fx]=steffensen(xo,f,n)
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    xo=x;
    co(:,i)=xo;
    x1=xo;
    syms x;
    x=xo+fo;
    f1=eval(f);
    xo=x1;
    syms x;
    x=xo-(fo^2/(f1-fo));
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
ste=x;
syms x;
x=ste;
fx=eval(f);
```

9.-Método Punto Medio

```
function [pm,no,co,fx]=puntomedio(xo,f,n)
g=diff(f);
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(g);
    xo=x;
    co(:,i)=xo;
    syms x;
    x1=xo;
    x=xo-fo/(2*go);
    g1=eval(g);
    xo=x1;
    syms x;
    x=xo-(fo/g1);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
pm=x;
syms x;
x=pm;
fx=eval(f);
```

10.-Método Traub-Ostrowski

```
function [to,no,co,fx]=traubostrowski(xo,f,n)
g=diff(f);
u=f/g;
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(f);
    uo=eval(u);
    xo=x;
    syms x;
    x1=xo;
    co(:,i)=xo;
    x=xo-uo;
    f1=eval(f);
    xo=x1;
    syms x;
    x=xo-(uo)*(f1-fo)/(2*f1-fo);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
to=x;
```

```
syms x;
x=to;
fx=eval(f);
```

11.-Método Jarrat.

```
function [ja,no,co,fx]=jarrat(xo,f,n)
g=diff(f);
u=f/g;
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(f);
    uo=eval(u);
    xo=x;
    co(:,i)=xo;
    syms x;
    x1=xo;
    x=xo-(2/3)*uo;
    g1=eval(g);
    xo=x1;
    syms x;
    x=xo-(.5)*uo+fo/(go-3*g1);
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
ja=x;
syms x;
x=ja;
fx=eval(f);
```

12.-Método Newton Newton

```
function [nn,no,co,fx]=newtonnewton(xo,f,n)
g=diff(f);
u=f/g;
syms x;
x=xo;
for i=1:1:n
    fo=eval(f);
    go=eval(f);
    uo=eval(u);
    xo=x;
    co(:,i)=xo;
    syms x;
    x1=xo;
    x=xo-uo;
    f1=eval(f);
    g1=eval(g);
    xo=x1;
    syms x;
    x=xo-uo-f1/g1;
    m=abs(x-xo)/abs(x);
    no=i;
    if m<=10^-10
        break
    end
end
nn=x;
syms x;
x=nn;
fx=eval(f);
```

A continuación se expresa el algoritmo utilizado para obtener el cero.

1.- Se define la variable x.

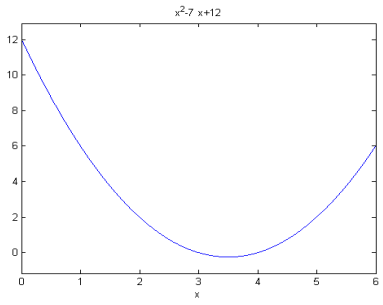
syms x;

2.- Se define una función determinada.

f =x^2-7*x+12

3.- Se grafica, para determinar un valor cercano a 0 que será el valor inicial de la iteración.

ezplot(f,[-0,6])



Dependiendo del valor escogido el método iterativo converge para una cierta cantidad de iteraciones o diverge según cada caso.

Es necesario obtener el **orden de convergencia** de cada método. De esta manera se puede deducir una cantidad adecuada de iteraciones para cada método sin adivinar o hacer numerosas iteraciones.

Si se tiene una secuencia x_k que converge al número ξ .

Se dice entonces que la sucesión **converge con orden q** a ξ si:

$$\lim_{k\to\infty} \frac{|X_{k+1}-\xi|}{|x_k-\xi|^q} = \mu \, con \, \mu > 0.$$

El número **q** es llamado **orden de convergencia.**

Siendo, para q=1 llamda lineal, q=2 cuadrática, q=3 cúbica.

Definido un método en particular, para obtener una sucesión es necesario reemplazar los f o f' genéricos por una función determinada.

Si utilizamos como f:

$$f = x^2 - 7*x + 12$$

Su punto de convergencia, es 3 o 4.

Para cada método se tiene una sucesión del tipo

$$x_{n+1} = F(x_n)$$

Para definir el orden de convergencia hay que obtener:

$$g(\bar{x}) = \bar{x}\,,\ g'(\bar{x}) = \cdots = g^{(p-1)}(\bar{x}) = 0\,,\ \text{y}\ \ g^{(p)}(\bar{x}) \neq 0.$$

Dada esta forma se escribe un código que calcula desde orden 1 hasta 4 los límites de cada método iterativo en el valor del cero de función, donde se toma la definición de cada método como una función continua y derivable p veces:

```
syms x;
f=x^2-7*x+12;
g=diff(f);
h=diff(g);
%grado conv newton
syms x;
fn=x-f/g;
fn1=diff(fn);
fn2=diff(fn1);
fn3=diff(fn2);
fn4=diff(fn4);
OC(1,1)=limit(fn1,3);
```

```
%steffensen
syms x;
syms t;
t=x+f;
x=t;
g1=eval(g);
syms x;
fste=x-f^2/(g1-f);
fste1=diff(fste);
fste2=diff(fste1);
fste3=diff(fste2);
fste4=diff(fste3);
```

```
OC(1,2)=limit(fn2,3);
OC(1,3)=limit(fn3,3);
OC(1,4)=limit(fn4,3);
%grado conv schröder
syms x;
fsc=x-f*g/(g^2-f*h);
fsc1=diff(fsc);
fsc2=diff(fsc1);
fsc3=diff(fsc2);
fsc4=diff(fsc3);
OC(2,1)=limit(fsc1,3);
OC(2,2)=limit(fsc2,3);
OC(2,3)=limit(fsc3,3);
OC(2,4)=limit(fsc4,3);
% GC whittaker
syms x;
fw=x-f/(2*g)*(2-lefx(f));
fw1=diff(fw);
fw2=diff(fw1);
fw3=diff(fw2);
fw4=diff(fw3);
OC(3,1)=limit(fw1,3);
OC(3,2)=limit(fw2,3);
OC(3,3)=limit(fw3,3);
OC(3,4)=limit(fw4,3);
%Halley
syms x;
fh=x-f*(2*g)/(2*g^2-f*h);
fh1=diff(fh);
fh2=diff(fh1);
fh3=diff(fh2);
fh4=diff(fh3);
OC(4,1)=limit(fh1,3);
OC(4,2)=limit(fh2,3);
OC(4,3)=limit(fh3,3);
OC(4,4)=limit(fh4,3);
%Chebysev
syms x;
fch=x-(f/g)*(1+lefx(f)/2);
fch1=diff(fch);
fch2=diff(fch1);
fch3=diff(fch2);
fch4=diff(fch3);
OC(5,1)=limit(fch1,3);
OC(5,2)=limit(fch2,3);
OC(5,3)=limit(fch3,3);
OC(5,4)=limit(fch4,3);
%super halley
syms x;
fsh=x-f/(2*g)*(2-lefx(f))/(1-lefx(f));
fsh1=diff(fsh);
fsh2=diff(fsh1);
fsh3=diff(fsh2);
fsh4=diff(fsh3);
OC(6,1)=limit(fsh1,3);
OC(6,2)=limit(fsh2,3);
OC(8,1)=limit(fste1,3);
OC(8,2)=limit(fste2,3);
OC(8,3)=limit(fste3,3);
OC(8,4)=limit(fste4,3);
% Punto medio
syms x;
syms t;
t=x-.5*f/g;
x=t;
g1=eval(g);
syms x;
fpm=x-f/g1;
fpm1=diff(fpm);
fpm2=diff(fpm1);
fpm3=diff(fpm2);
fpm4=diff(fpm3);
OC(9,1)=limit(fpm1,3);
OC(9,2)=limit(fpm2,3);
OC(9,3)=limit(fpm3,3);
OC(9,4)=limit(fpm4,3);
%traub ostrowski
u=f/g;
syms t;
t=x-u;
x=t;
f1=eval(f);
syms x;
fto=x-u*(f1-f)/(2*f1-f);
fto1=diff(fto);
fto2=diff(fto1);
fto3=diff(fto2);
fto4=diff(fto3);
OC(10,1)=limit(fto1,3);
OC(10,2)=limit(fto2,3);
OC(10,3)=limit(fto3,3);
OC(10,4)=limit(fto4,3);
%Jarrat
u=f/g;
syms t;
t=x-(2/3)*u;
x=t;
g1=eval(g);
syms x;
fj=x-u*(.5)+f/(g-3*g1);
fj1=diff(fj);
fj2=diff(fj1);
fj3=diff(fj2);
fj4=diff(fj3);
OC(11,1)=limit(fj1,3);
OC(11,2)=limit(fj2,3);
OC(11,3)=limit(fj3,3);
OC(11,4)=limit(fj4,3);
%newton newton
u=f/g;
syms t;
t=x-u;
```

```
OC(6,3)=limit(fsh3,3);
OC(6,4)=limit(fsh4,3);
%Stirleing
syms x;
syms t;
t=x-f;
x=t;
g1=eval(g);
syms x;
fst=x-f/g1;
fst1=diff(fst);
fst2=diff(fst1);
fst3=diff(fst2);
fst4=diff(fst3);
OC(7,1)=limit(fst1,3);
OC(7,2)=limit(fst2,3);
OC(7,3)=limit(fst3,3);
OC(7,4)=limit(fst4,3);
```

```
x=t;
g1=eval(g);
f1=eval(f);
syms x;
fnn=x-u-f1/g1;
fnn1=diff(fnn);
fnn2=diff(fnn1);
fnn3=diff(fnn2);
fnn4=diff(fnn3);
OC(12,1)=limit(fnn1,3);
OC(12,2)=limit(fnn2,3);
OC(12,3)=limit(fnn3,3);
OC(12,4)=limit(fnn4,3);
```

La matriz de resultados va ordenada de la siguiente forma: por fila cada método distinto, por columna el grado de convergencia. Si el valor obtenido en una columna p es distinto de 0, entonces el grado (estimado) del método es p.

Método	OC =	Grado de convergencia estimado.
1,-Newton	[0, -2, -12, 0]	2
2,-Schroder	[0, 2, 12, 48]	2
3,-Whittaker	[0, -4, -36, -408]	2
4,-Halley	[0, 0, 6, 72]	3
5,-Chebysev	[0, 0, 12, 216]	3
6,-SuperHalley	[0, 0, 0, -24]	4
7,-Stirlign	[0, -6, -60, -816]	2
8,-Steffenson	[1, 2, -6, 24]	1
9,-Punto Medio	[0, 0, 6, 72]	3
10,-Traub Ostrowski	[0, 0, 0, -24]	4
11,-Jarrat	[0, 0, 0, -24]	4
12,-Newton Newton	[0, 0, 0, -24]	4

A continuación se muestra el código programado para obtener en forma fácil el punto de convergencia y la cantidad de iteraciones por método, valores por iteración y comprobación.

```
function [A,B,C,F]=analisisf(f,xo,N)
%1
[Xc,no1,co1,fx1]=newton(xo,f,N);
%2
[sc,no2,co2,fx2]=schroder(xo,f,N);
%3
[wh,no3,co3,fx3]=whittaker(xo,f,N);
%4
[ha,no4,co4,fx4]=halley(xo,f,N);
%5
[ch,no5,co5,fx5]=chebysev(xo,f,N);
%6
[sha,no6,co6,fx6]=superhalley(xo,f,N);
%7
[st,no7,co7,fx7]=stirling(xo,f,N);
%8
[ste,no8,co8,fx8]=steffensen(xo,f,N);
%9
[pm,no9,co9,fx9]=puntomedio(xo,f,N);
%10
[to,no10,co10,fx10]=traubostrowski(xo,f,N);
%11
[ja,no11,co11,fx11]=jarrat(xo,f,N);
%12
[nn,no12,co12,fx12]=newtonnewton(xo,f,N);
A(:,1)=[Xc;sc;wh;ha;ch;sha;st;ste;pm;to;ja;nn];
B(:,1)=[no1,no2,no3,no4,no5,no6,no7,no8,no9,no10,no11,no12];
F(:,1)=[fx1,fx2,fx3,fx4,fx5,fx6,fx7,fx8,fx9,fx10,fx11,fx12];
l(:,1)=length(co1);
l(:,2)=length(co2);
l(:,3)=length(co3);
l(:,4)=length(co4);
l(:,5)=length(co5);
l(:,6)=length(co6);
l(:,7)=length(co7);
l(:,8)=length(co8);
l(:,9)=length(co9);
l(:,10)=length(co10);
l(:,11)=length(co11);
l(:,12)=length(co12);
lm=max(l);

Cx=[zeros(1,lm-l(1)) co1 ;zeros(1,lm-l(2)) co2 ;zeros(1,lm-l(3)) co3 ;zeros(1,lm-
l(4)) co4 ;zeros(1,lm-l(5)) co5 ;zeros(1,lm-l(6)) co6 ;zeros(1,lm-l(7)) co7
;zeros(1,lm-l(8)) co8;zeros(1,lm-l(9)) co9 ; zeros(1,lm-l(10)) co10; zeros(1,lm-
l(11)) co11;zeros(1,lm-l(12)) co12 ];
C=Cx;
```

Seleccionando xo=2,5, los resultados obtenidos, del ejemplo, para cada método iterativo son:

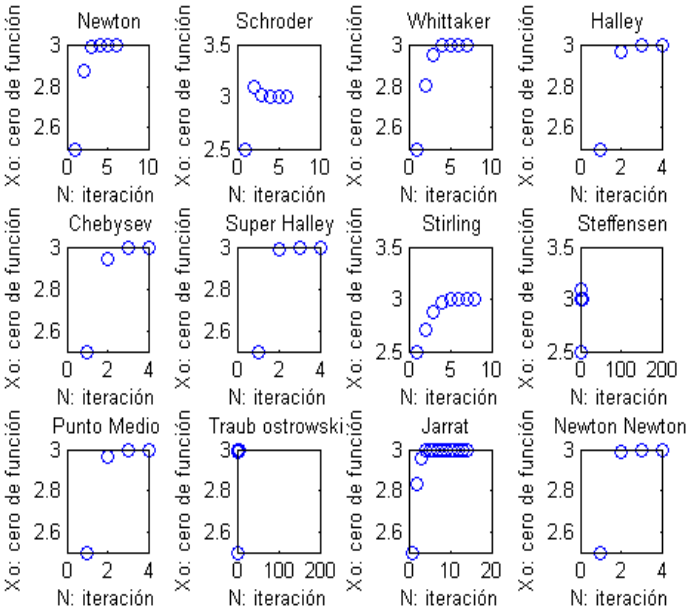
Método	A: Son los valores de Xo cero de función	B: Son los valores de las iteraciones suficientes por método.	F =Son los valores de F obtenidos con el cero encontrado
1,-Newton	3.0000	6	0
2,-Schroder	3.0000	6	-1,77635683940025e-15
3,-Whittaker	3.0000	7	-1,77635683940025e-15
4,-Halley	3.0000	4	-1,77635683940025e-15
5,-Chebysev	3.0000	4	0
6,-SuperHalley	3.0000	4	-1,77635683940025e-15
7,-Stirlign	3.0000	8	1,77635683940025e-15
8,-Steffenson	NaN	50	NaN
9,-Punto Medio	3.0000	4	-1,77635683940025e-15
10,-Traub ostrowski	NaN	50	NaN
11,-Jarrat	3.0000	14	2,15365503208886e-11
12,-Newton Newton	3.0000	4	0

Se observa convergencia todos los métodos, excepto 8 y 10, con cantidad de iteraciones menores a 50 siendo el método más rápido para este caso el Halley, entre otros.

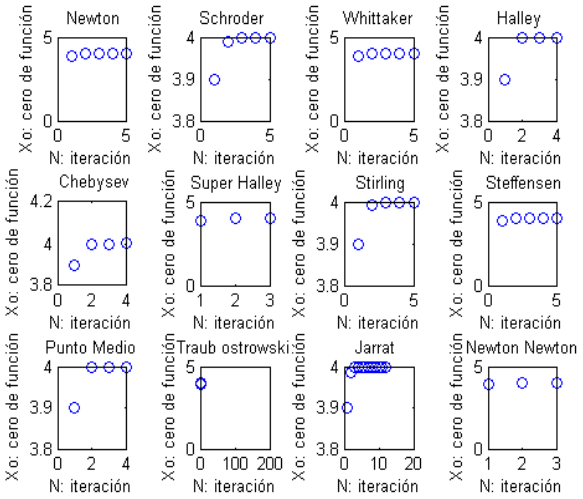
Para facilitar el estudio se desarrolla un código para graficar todos los métodos:

```
function p=plotearf(B,C)
metodos=['     Newton     ';'   Schroder     ';'   Whittaker   ';'    Halley     ';
    '    Chebysev    ';'  Super Halley ';'    Stirling    ';'   Steffensen   ';'
Punto Medio  ';
    'Traub ostrowski';'    Jarrat     ';' Newton Newton '];
l=length(B)
p=figure;
for i=1:l
    n=i;
    co=length(C(i,:));
    bo=co-B(i,:)+1;
    B(i,1);
    no=[1:1:B(i,1)];
    subplot(3,4,i);
    c=C(i,bo:co);
    plot(no,c,'o');
   title(metodos(i,:));
   xlabel('N: iteración');
   ylabel('Xo: cero de función');
end
```

Las gráficas del ejemplo resultan:



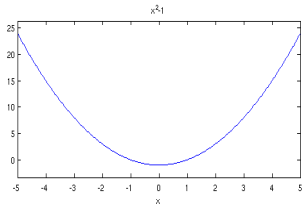
Esta iteración se hizo para el punto inicial xo=2,5, con cero en 3, para el valor inicial xo=3,9, el cero a encontrar será 4. Como se observa, ahora, además de los métodos que convergen para condición inicial xo=2.5, converge el método Steffenson.



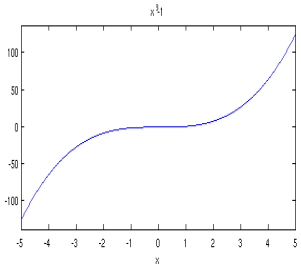
Es de notar también la importancia de escoger un valor cercano, de esta manera es posible evitar grandes cantidades de iteraciones y se puede asegurar la convergencia con métodos que no lo harían si el valor inicial está más alejado.

A continuación se definen las funciones a las cuales se debe calcular el cero, su gráfica y el valor inicial de iteración.

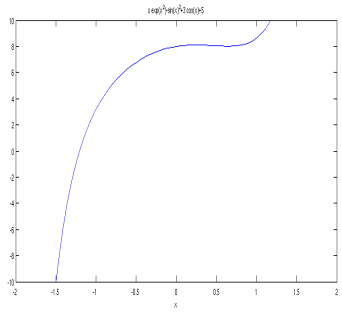
```
syms x;
f1=x^2-1;
```



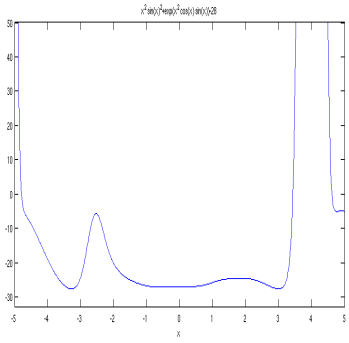
```
f2=x^3-1;
```



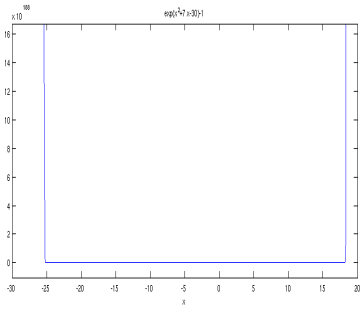
```
f3=x*exp(x^2)-sin(x)^2+3*cos(x)+5;
```



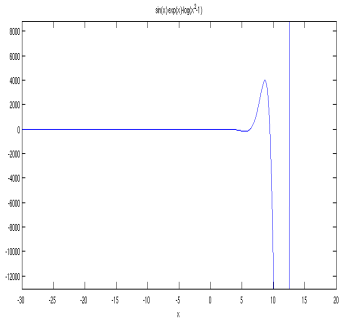
```
f4=x^2*sin(x)^2+exp(x^2*cos(x)*sin(x))-28;
```



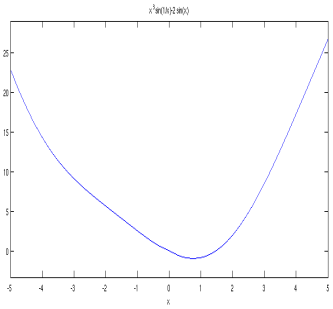
%menor cero positivo
f5=exp(x^2+7*x-30)-1;



f6=sin(x)*exp(x) -log(x^2-1);



f7=x^3 * sin(1/x) -2*sin(x); %continua en 0, equivalente a definida en guía



Se define el vector de condiciones iniciales:

$$x_0 = \begin{pmatrix} 1.039 \\ 1.026 \\ -1.22 \\ 3.395 \\ 2.979 \\ 3.025 \\ 1.455 \end{pmatrix}$$

Se define la cantidad de iteraciones máxima es: N=50

Se definen las siglas:

CR: convergencia rápida, en orden descendente.

CN: No alcanza a converger en la cantidad de iteraciones dispuestas.

D:diverge

Con estas siglas se determinará en forma rápida el nivel de convergencia de cada método por cada función, clasificándolos adecuadamente.

A continuación los resultados y las gráficas de cada función con cada método.

Se definen

A: Valor del cero de la función.

B: Cantidad iteraciones para obtenerla.

F: Comprobación que el x* sea 0, evaluando en función.

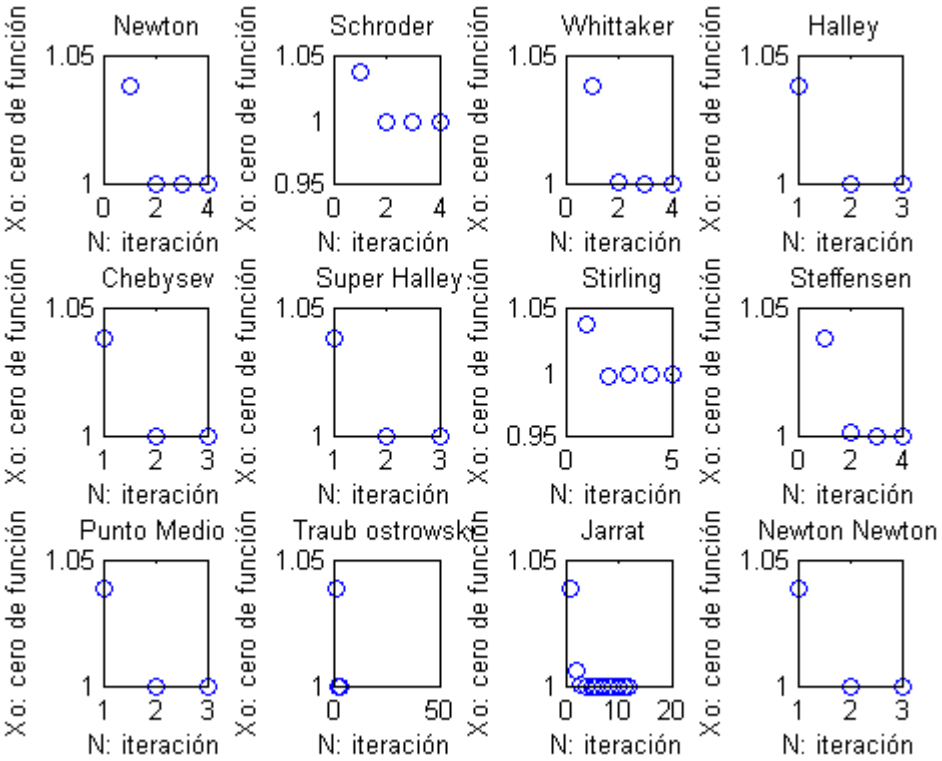
I.b Estudio de funciones con métodos de convergencia

En esta sección se estudian las siete funciones definidas en la guia, utilizando los métodos programados y evaluando su convergencia, su rapidez en el caso que lo haga o bien, explicando su no convergencia o divergencia.

Como pasos previos se comprueba la continuidad y derivabilidad suficiente para poder aplicar los métodos iterativos. Como se dijo anteriormente, el requerimiento máximo es que sean dos veces derivables.

f1=x^2-1;

Método	A	B	F
1, - Newton	1.0000	4	0
2, - Schroder	1.0000	4	0
3, - Whittaker	1.0000	4	0
4, - Halley	1.0000	3	0
5, - Chebysev	1.0000	3	0
6, - SuperHalley	1.0000	3	0
7, - Stirlign	1.0000	5	0
8, - Steffenson	1.0000	4	0
9, - Punto Medio	1.0000	3	0
10, - Traub ostrowski	NaN	50	NaN
11, - Jarrat	1.0000	12	3,66129349060884e-11
12, - Newton Newton	1.0000	3	0



CR:{4,5,6,9,12,1,2,3,8,7,11}

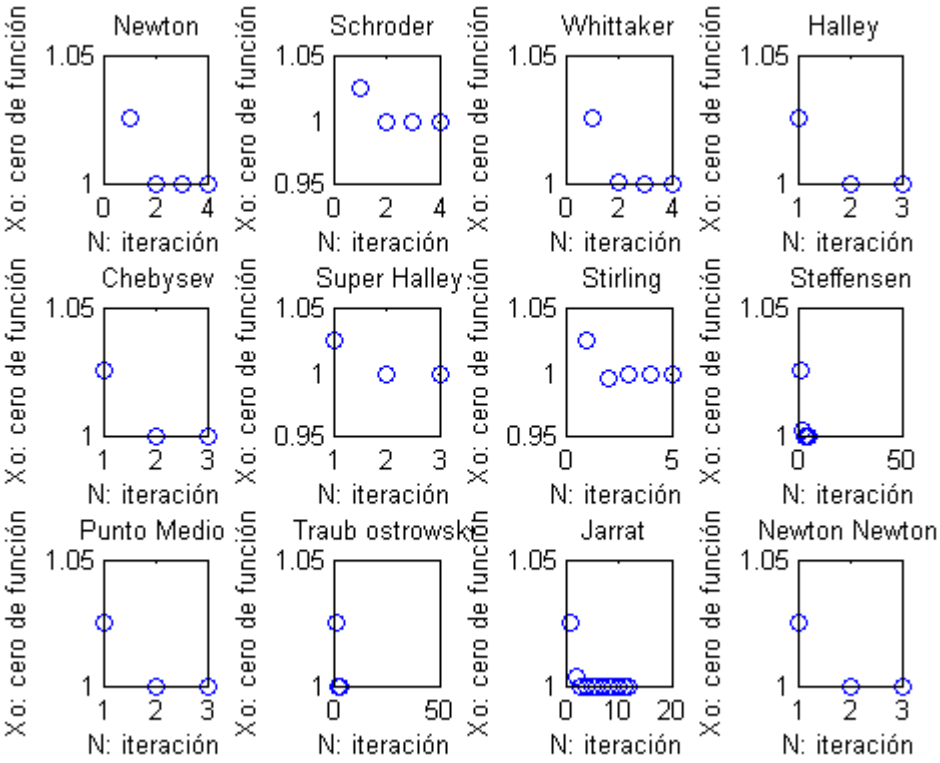
CN:{}

D:{10}

Se observa convergencia para todos los métodos excepto Trub Ostrowski, el método más lento de convergencia es Jarrat. El orden de rapidez está indicado en CR.

f2=x^3-1;

Método	A	B	F
1, - Newton	1.0000	4	0
2, - Schroder	1.0000	4	0
3, - Whittaker	1.0000	4	0
4, - Halley	1.0000	3	0
5, - Chebysev	1.0000	3	0
6, - SuperHalley	1.0000	3	0
7, - Stirlign	1.0000	5	0
8, - Steffenson	NaN	50	NaN
9, - Punto Medio	1.0000	3	0
10, - Traub ostrowski	NaN	50	NaN
11, - Jarrat	1.0000	12	3,76758624298645e-11
12, - Newton Newton	1.0000	3	0



CR:{4,5,6,9,12,1,2,3,7,11}

CN:{}

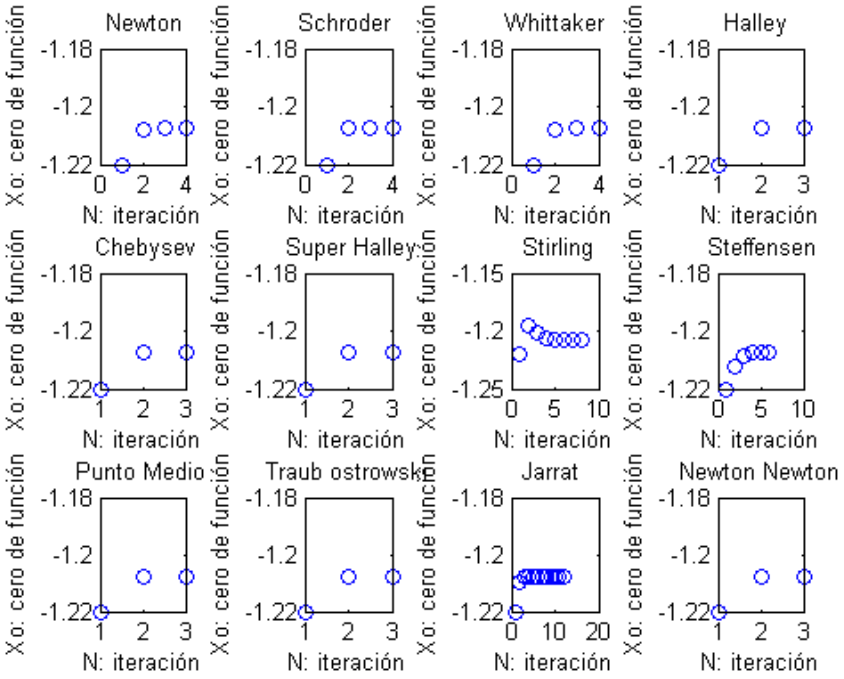
D:{8,10}

Se observa convergencia para todos los métodos excepto Trub Ostrowski y Steffenson, el método más lento de convergencia es Jarrat. El orden de rapidez está indicado

en CR.

f3=x*exp(x^2)-sin(x)^2+3*cos(x)+5;

Método	A	B	F
1, - Newton	- 1.2076	4	- 2,66453525910038e - 15
2, - Schroder	- 1.2076	4	3,55271367880050e - 15
3, - Whittaker	- 1.2076	4	3,55271367880050e - 15
4, - Halley	- 1.2076	3	3,55271367880050e - 15
5, - Chebysev	- 1.2076	3	- 2,66453525910038e - 15
6, - SuperHalley	- 1.2076	3	3,55271367880050e - 15
7, - Stirlign	- 1.2076	8	- 2,66453525910038e - 15
8, - Steffenson	- 1.2076	6	3,55271367880050e - 15
9, - Punto Medio	- 1.2076	3	3,55271367880050e - 15
10, - Traub ostrowski	- 1.2076	3	- 2,66453525910038e - 15
11, - Jarrat	- 1.2076	12	- 1,22551746528643e - 10
12, - Newton Newton	- 1.2076	3	- 2,66453525910038e - 15



CR:{4,5,6,9,10,12,1,2,3,8,7,11}

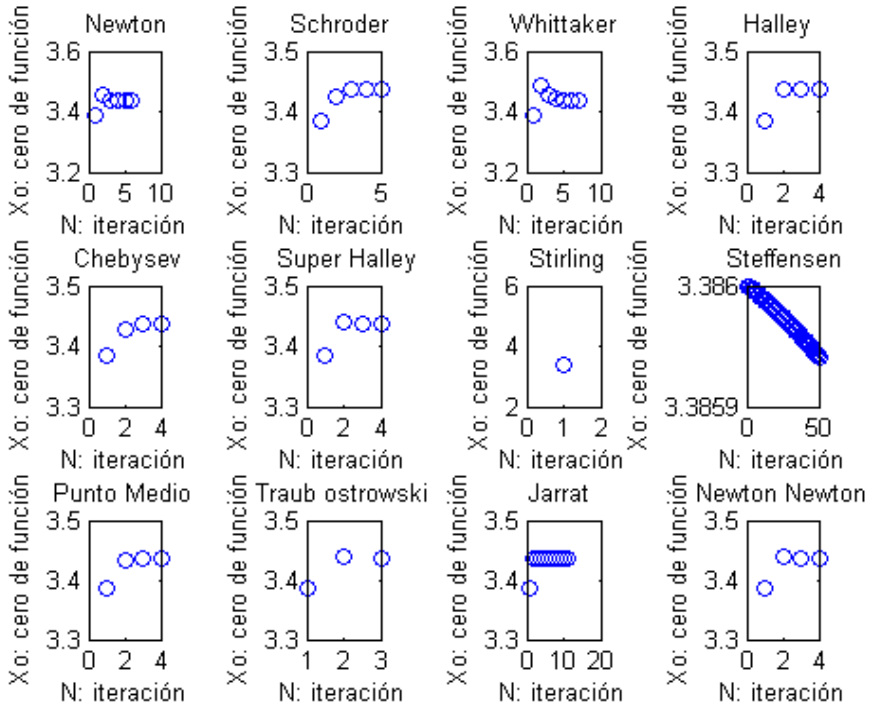
CN:{7..12}

D:{}

Todos los métodos convergen al cero de la función. El orden de la rapidez de convergencia está indicado en CR.

f4=x^2*sin(x)^2+exp(x^2*cos(x)*sin(x))-28

Método	A	B	F
1, - Newton	3.4375	6	0.0000
2, - Schroder	3.4375	5	0.0000
3, - Whittaker	3.4375	7	0.0000
4, - Halley	3.4375	4	0.0000
5, - Chebysev	3.4375	4	0.0000
6, - SuperHalley	3.4375	4	0.0000
7, - Stirlign	3.3860	1	- 12.5695
8, - Steffenson	3.3860	50	- 12.5749
9, - Punto Medio	3.4375	4	0.0000
10, - Traub ostrowski	3.4375	3	0.0000
11, - Jarrat	3.4375	11	- 0.0000
12, - Newton Newton	3.4375	4	0.0000



CR:{10,4,5,6,12,2,1,3,11}

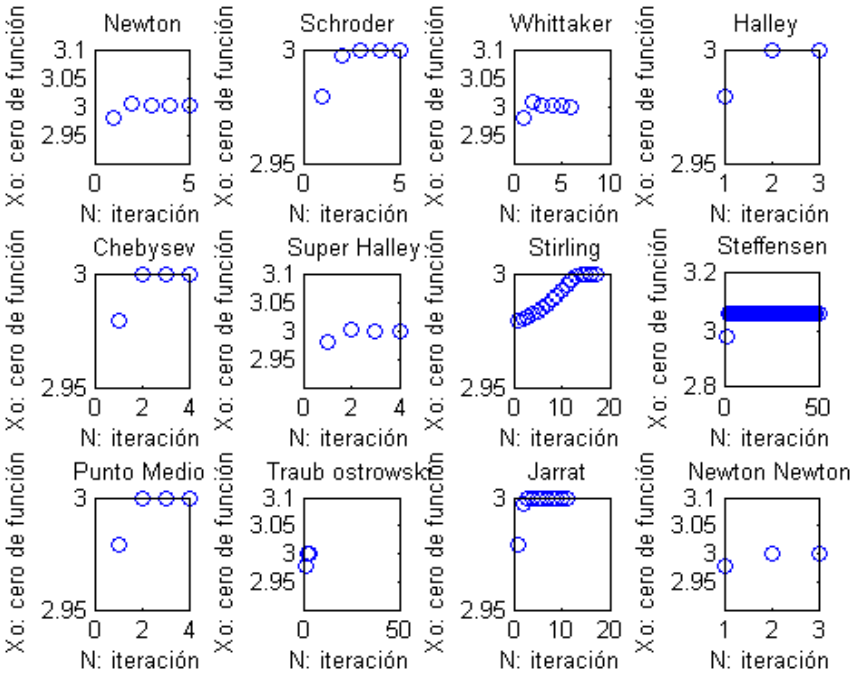
CN:{7,8}

D:{}

Convergen todos los métodos excepto Stirling y Steffensen, se hizo un estudio cambiando el xo y aumentando N y no es posible la convergencia con estos métodos debido a que el grado de convergencia no es compatible con la función.

f5=exp(x^2+7*x-30)-1;

Método	A	B	F
1, - Newton	3.0000	5	0
2, - Schroder	3.0000	5	0
3, - Whittaker	3.0000	6	0
4, - Halley	3.0000	3	0
5, - Chebysev	3.0000	4	0
6, - SuperHalley	3.0000	4	0
7, - Stirlign	3.0000	17	0
8, - Steffenson	3.0575	50	1.1193
9, - Punto Medio	3.0000	4	0
10, - Traub ostrowski	NaN	50	NaN
11, - Jarrat	3.0000	11	- 0.0000
12, - Newton Newton	3.0000	3	0



CR:{12,4,5,6,9,1,2,3,11,7}

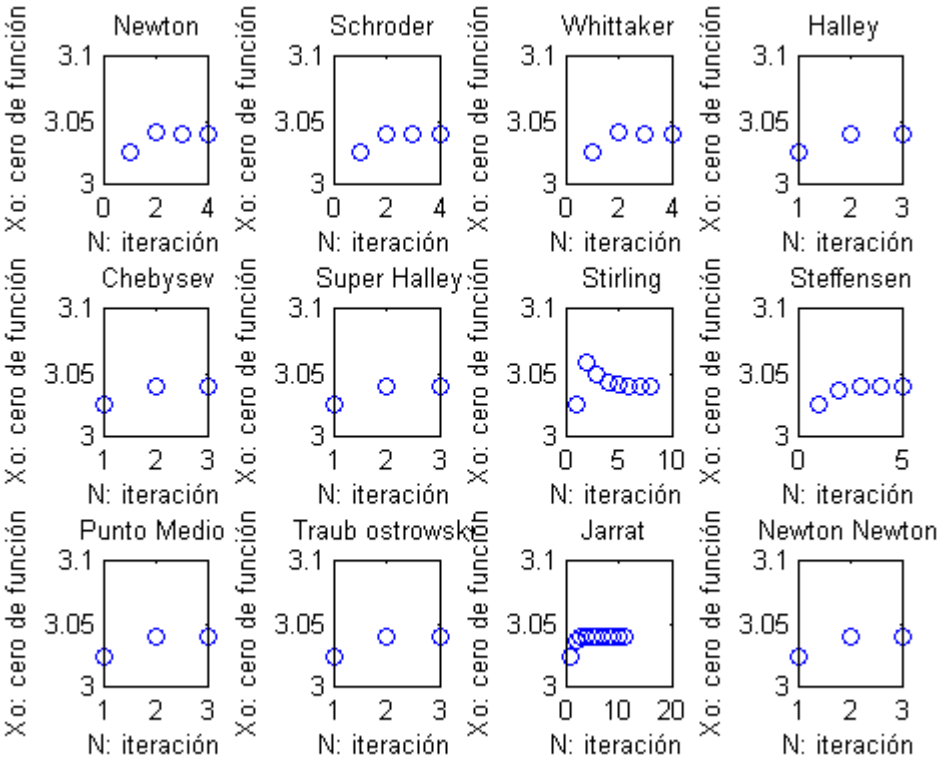
CN:{8}

D:{10}

Hay convergencia para todos los métodos excepto Steffenson que queda en un punto estable y Traub Ostroski que presenta divergencia.

f6=sin(x)*exp(x) -log(x^2-1)

Método	A	B	F
1, - Newton	3.0406	4	2,22044604925031e-15
2, - Schroder	3.0406	4	2,22044604925031e-15
3, - Whittaker	3.0406	4	2,22044604925031e-15
4, - Halley	3.0406	3	2,22044604925031e-15
5, - Chebysev	3.0406	3	2,22044604925031e-15
6, - SuperHalley	3.0406	3	2,22044604925031e-15
7, - Stirlign	3.0406	8	2,22044604925031e-15
8, - Steffenson	3.0406	5	2,22044604925031e-15
9, - Punto Medio	3.0406	3	2,22044604925031e-15
10, - Traub ostrowski	3.0406	3	2,22044604925031e-15
11, - Jarrat	3.0406	11	8,05530753211770e-10
12, - Newton Newton	3.0406	3	2,22044604925031e-15



CR:{4,5,6,9,10,12,1,2,3,8,7,11}

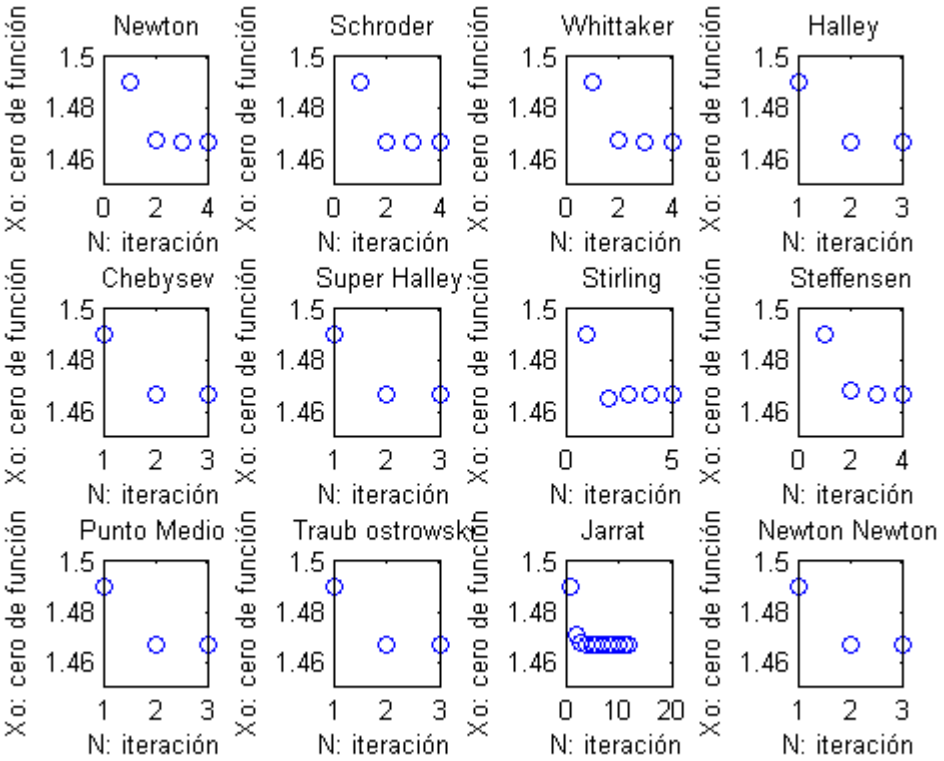
CN:{}

D:{}

Hay convergencia para todos los métodos. La rapidez está determinada por CR.

f7=x^3 * sin(1/x) -2* sin(x)

Método	A	B	F
1, - Newton	1.4670	4	0
2, - Schroder	1.4670	4	0
3, - Whittaker	1.4670	4	-4,44089209850063e-16
4, - Halley	1.4670	3	0
5, - Chebysev	1.4670	3	-4,44089209850063e-16
6, - SuperHalley	1.4670	3	0
7, - Stirlign	1.4670	5	0
8, - Steffenson	1.4670	4	0
9, - Punto Medio	1.4670	3	0
10, - Traub ostrowski	1.4670	3	0
11, - Jarrat	1.4670	12	2,95719004839157e-11
12, - Newton Newton	1.4670	3	0



CR:{4,5,6,9,10,12,1,2,3,8,7,11}

CN:{}

D:{}

Hay convergencia para todos los métodos. La rapidez está determinada por CR.

II Estudio de álgebra lineal.

Sea $A \in M_n(\mathbb{R})$ con $n = 70$ y considere el siguiente sistema de ecuaciones lineales $Ax = b$ donde:

$$A = \begin{pmatrix} 4 & 1 & 0 & 0 & \cdots & 0 \\ 1 & 2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 4 & 1 \\ 0 & 0 & 0 & \cdots & 1 & 2 \end{pmatrix} \quad \text{y} \quad b = \begin{pmatrix} 1 \\ 2 \\ 1 \\ \vdots \\ 2 \end{pmatrix}$$

Se genera el código que permite definir la matriz y el vector con las características indicadas.

```
function A=matriza(n)
a=0;
for i=1:n
    for j=1:n
        if j==i
            if a==0
            A(i,j)=4;
            a=i+1;%hace referencia al i siguiente
            elseif a==i
            A(i,j)=2;
            a=0;
            end
        elseif j==i-1
            A(i,j)=1;
        elseif j==i+1
            A(i,j)=1;
        else
            A(i,j)=0;
        end

    end
end
```

Como ejemplo, se obtiene para n=4:

```
matriza(4)

ans =

     4     1     0     0
     1     2     1     0
     0     1     4     1
     0     0     1     2
```

Para n=70, es similar, solo que no se mostrará en el texto, basta con la función.

De manera similar, el codigo para generar b es:

```
function b=vectorb(n)
a=0;
for i=1:n
            if a==0
            b(i,:)=1;
            a=i+1;%hace referencia al i siguiente
            elseif a==i
            b(i,:)=2;
            a=0;
            end
end
```

Y para n=4, b es lo pedido.

```
b =

     1
     2
     1
     2
```

II.a Método Cholesky

Demostrar que A es definida positiva. Encontrar descomposición Cholesky, resolver sistema usando método Cholesky.

Para que una matriz sea DP sus valores propios deben ser positivos. Utilizando el comando 'eig(A)' y el codigo siguiente que compara cada valor propio, se determina que es DP.

```
function DP=defpos(A)
E=eig(A);
l=length(E);
for i=1:l
   if E(i,:)>0
       DP='La matriz es Definida Positiva';
   elseif E(i,:)<=0
       DP='La matriz no es Definida Positiva';
       break

       end
   end
```

El resultado obtenido es:

```
DP=defpos(A)

DP =

La matriz es Definida Positiva
```

El método de resulución del sistema viene dado por:

Con el siguiente comando se obtuvo la matriz L, que es parte de la descomposición de Cholesky.

'L = chol(A,'lower');'

Luego se escribe la solucion del sistema:

L *y=*b;

Y a continuación la solución para cada y_i

$$y_1 = \frac{b_1}{L_1}$$

Y, en forma general:

$$y_k = \frac{1}{l_{kk}} \left[b_k - \sum_{j=1}^{k-1} l_{kj}\, y_j \right],\ k = 2, 3, \ldots, n$$

Una vez se resulve y, se procede a resolver el sistema triangular superior.

Para el último valor de x $x_n = \frac{y_n}{l_{nn}}$

Para el resto de los valores:

$$x_k = \frac{1}{l_{kk}} \left[y_k - \sum_{j=k+1}^{n} l_{jk}\, x_j \right],\ k = n-1, n-2, \ldots, 1$$

El código para obtener la solución es:

```
function chole=cholesky(A,b)
    L = chol(A,'lower');
    [m,n] = size(L);
    Lt=L';
%se resuelve L*y=b
y(1,:)=b(1)/L(1,1);

for i=2:m
    c=0;
    for j=1:n-1
        if i~=j & i>j
     c=c+L(i,j)*y(j,:);
        end
    end
   y(i,:)=(1/(L(i,i)))*(b(i)-c);
end
%se resuelbe L'x=y
x(n,:)=y(n,:)/L(n,n);

for i=n-1:-1:1
    c=0;
    for j=n:-1:2
        if i~=j & i<j
     c=c+Lt(i,j)*x(j);
        end
    end
   x(i,:)=(1/(Lt(i,i)))*(y(i,:)-c);
end
chole=x;
```

La solución de Xo (solución directa) es:

```
x'

ans =

 Columns 1 through 9

  -0.0607   1.2426  -0.4246   1.4558  -0.4871   1.4924  -0.4978   1.4987  -0.4996

 Columns 10 through 18

   1.4998  -0.4999   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000

 Columns 19 through 27

  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000

 Columns 28 through 36

   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000

 Columns 37 through 45

  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000

 Columns 46 through 54

   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000

 Columns 55 through 63

  -0.5000   1.5000  -0.5000   1.5000  -0.5000   1.5000  -0.4999   1.4997  -0.4996

 Columns 64 through 70

   1.4985  -0.4975   1.4914  -0.4853   1.4497  -0.4142   1.2071
```

Se obtiene la solución por método Cholesky escribiendo:

"chole=cholesky(A,b);"

Luego, se calcula el error entre la solución directa y "chole":

"EE=norm(xx-chole)"

"EE = 3.7494e-015"

Se observa que el error (EE) siendo tan pequeño, es suficientemente adecuado como para aceptar a "chole" como solución del sistema.

II.b- Obtener la solución con método de Jacobi, Gauss Siedel, SOR.

Se deduce que el método que puede converger más rápido es Gauss Siedel. El tiempo de iteración es relativamente pequeño para Jacobi y Gauss Siedel, son necesarias N^2 multiplicaciones y N(N-1) restas por iteracion. En princiio Gauss Siedel debiera ser más rápido, ya que se va adaptando el vector x_n en cada subiteración que ocurre por iteración.

b.1.- Método Jacobi.

Primero es necesario descomponer la matriz A en LDU, tal que:

"A=L+D+U"

Se escribe el código que descompone cualquier matriz en esa forma:

```
function [L D U]=jacobildu(A)
[m,n]=size(A);
for i=1:n
    for j=1:m
        if i==j
            L(i,j)=0;
            D(i,j)=A(i,j);
            U(i,j)=0;
        elseif j<i
            L(i,j)=A(i,j);
            D(i,j)=0;
            U(i,j)=0;
        elseif j>i
            L(i,j)=0;
            D(i,j)=0;
            U(i,j)=A(i,j);
        end
    end
end
```

EL algoritmo de Jacobi es el siguiente:

$$D\,x+(L+U)\,x=b$$
$$x=D^{-1}*[b-(L+U)*x]$$
$$La\ sucesión\ de\ Jacobi\ es\ la\ siguiente:$$
$$X^{(k+1)}=D^{-1}*[b-(L+U)*x^{k}]$$

El código que encuentra la solución con el algoritmo de Jacobi es:

```
function jx=jacobi(A,b,xo,n)
[L D U]=jacobildu(A);
for i=1:n
    x=D^(-1)*[b-(L+U)*xo];
    xo=x;
end
jx=xo;
```

Haciendo la prueba para una matriz de dimension N=5., se tiene:

```
%se define matriz de dim 5
A=matriza(5)
A =  4     1     0     0     0
     1     2     1     0     0
     0     1     4     1     0
     0     0     1     2     1
     0     0     0     1     4
% se encuentra LDU
>> [L D U]=jacobildu(A)
L =  0     0     0     0     0
     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0

D =  4     0     0     0     0
     0     2     0     0     0
     0     0     4     0     0
     0     0     0     2     0
     0     0     0     0     4

U =  0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
     0     0     0     0     0
%se define el vectro b
b=vectorb(5)
b =[1,2,1,2,1]';
```

```
%se encuentra solucion sin iteracion,
directa
>> sol=A^(-1)*b
sol =
   -0.0500
    1.2000
   -0.3500
    1.2000
   -0.0500
%se encuentra solucion para 10
iteraciones, no es suficiente
jx=jacobi(A,b,b,10)

jx =

   -0.0441
    1.2059
   -0.3381
    1.2059
   -0.0441
%solucion para 50 iteraciones, es
suficiente
>> jx=jacobi(A,b,b,50)
jx = -0.0500
    1.2000
   -0.3500
    1.2000
   -0.0500
```

Ahora bien, se debe incluir la condición

$$\frac{\left\|x^{(n+1)}-x^{(n)}\right\|_2}{\left\|x^{(n)}\right\|_2}\leq 10^{-7}$$

Entonces el codigo, con esta condición, sería:

```
function [jx,no]=jacobi(A,b,xo,n)
[L D U]=jacobildu(A);
for i=1:n
    x=D^(-1)*[b-(L+U)*xo];
    x1=xo;
    xo=x;
    m=norm(xo-x1)/norm(x1);
    no=i;
    if m<=10^-7
        break
    end
end
jx=xo;
```

Para el ejemplo se cumple la condición en la iteración no=36.

b.2.- Gauss Siedel

Es un método indirecto, lo que significa que se parte de una aproximación inicial y se repite el proceso hasta llegar a una solución con un margen de error tan pequeño como se quiera. Buscamos la solución a un sistema de ecuaciones lineales, en notación matricial:

$$Ax = b.$$

El método de iteración Gauss-Seidel es, en código matlab:

```
function [gs,no]=gausssiedel(A,b,xo,N)
[m,n]=size(A)
for k=1:N
    x1=xo;
    for i=1:m
        for j=1:n
            Ax=[A(j,1:j-1) 0 A(j,j+1:n)];
            if j==i
            xo(j)=(b(j)-Ax*xo)/A(j,j);
            else
            xo(j)=xo(j);% subiteracion xi^n=xi^(n-1)
            end
        end
    end
    x2=xo;%una iteracion completa
    mx=norm(x2-x1)/norm(x1);%se obtiene valor que compara y detiene si cumple
condicion
    no=i;
    if mx<=10^-7
        break
    end
end
gs=xo;
```

La solución para n=5 es:

```
[gs,no]=gausssiedel(A,b,xo,N)
gs =-0.0500
    1.2000
   -0.3500
    1.2000
   -0.0500
no =  18
```

Lógicamente es correcta, la cantidad de iteraciones es 18.

b.3.- Relajación Sucesiva

Si A es simétrica, definida positiva y tridiagonal, el valor de ω óptimo se encuentra en:

$$\omega = \frac{2}{(1+\sqrt{(1-\rho^2)})}$$

Donde ρ es el radio espectral del método de Jacobi.

Si A es simétrica y definida positiva, el método de relajación converge ssi 0<ω <2.

Si ω <1 entonces el método es subrelajación y si ω >2 es sobrerrelajación.

Como en efecto A es cumple todas las carácterísticas, falta solamente ρ.

Radio espectral es el mayor valor de los valores propios de la matriz Q.

Para esta matriz Q= D^(-1)*(L+U) . Entonces

```
Q =

         0    0.2500         0         0         0
    0.5000         0    0.5000         0         0
         0    0.2500         0    0.2500         0
         0         0    0.5000         0    0.5000
         0         0         0    0.2500         0

rho=max(eig(Q))
rho = 0.6124
w= 2/(1+sqrt(1-rho^2))
```

$$\omega = 1{,}117$$

Por teorema el método debe converger ya que el valor está entre 0 y 2.

El algoritmo para obtener la solución se representa por el siguiente código:

```
function [sor,no]=relajacionsucesiva(A,b,xo,n)
[L,D,U]=jacobildu(A);
Q=D^(-1)*(L+U);
rho=max(eig(Q));
w =2/(1+(1-rho^2)^(1/2));
D1=(1/w)*D;
D2=(w-1)/w*D;
%A=D1+L+D2+U
```

```
M=D1+L;
N=D2+U;
%M*x^(n+1)=N*x^(n)+b
Mn=M^(-1)*N;
c=M^(-1)*b;
for i=1:n
    x=-Mn*xo+c;
    x1=xo;
    xo=x;
    m=norm(xo-x1)/norm(x1);
    no=i;
    if m<=10^-7
        break
    end
end
sor=x;
```

En efecto, para n=5, la solución es correcta.

```
[sor,no]=relajacionsucesiva(A,b,b,20)

sor =

   -0.0500
    1.2000
   -0.3500
    1.2000
   -0.0500
no = 11
```

Por último, se calcula la solución para los tres algoritmos, con n=70.

Los resultados no se mostrarán, sin embargo el error entre la solución directa e iterativa si, además de mostrar los valores para las iteraciones, a continuación:

	N° iteraciones	**||Xo-X* ||=norm(EE(:,i),2);**
Gauss Siedel	26	7.7795e-007
SOR	20	1.7341e-007
Jacobi	49	4.7523e-007

Se observa que, dado que los errores son pequeños, las soluciones que presentan los tres métodos son aceptables como solución numérica. Donde la cantidad de iteraciones por GS es notablemente menor que Jacobi, pero es más rápida SOR, que se adapta mejor a matrices con las condiciones enunciadas en b.3, cuya matriz A es un ejemplar.

II.c Perturbaciones en el sistema Ax=b.

Se estudia la solución del sistema con el vector "b" perturbado. El codigo generador de b se modifica para el caso.

```
function b=vectorbp(n,p)
a=0;
for i=1:n
            if a==0
            b(i,:)=1;
            a=i+1;%hace referencia al i siguiente
            elseif a==i
            b(i,:)=p;
            a=0;
            end
end
```

Dado que se trata de un sistema lineal y continuo, la perturbación no afectará notablemente la solución al sistema.

c.1) condicionamiento de la matriz A.

Se A matriz no singular. Y "x" solución de Ax=b, $\hat{x}$ solución al problema perturbado $A*\hat{x}=b$. Restando estos dos sistemas de ecuaciones se tiene:

$$A*(x-\hat{x})=b-\hat{b}$$
$$x-\hat{x}=A^{-1}*(b-\hat{b})$$

Luego, la submultiplicidad de la norma Matricial se obtiene:

$$\left\|x-\hat{x}\right\|=\left\|A^{-1}(b-\hat{b})\right\|\leq\left\|A^{-1}\right\|\left\|(b-\hat{b})\right\|$$

Dividiendo po $\|x\|$ se tiene que

$$\frac{\left\|x-\hat{x}\right\|}{\|x\|}\leq\frac{\left\|A^{-1}\right\|\left\|(b-\hat{b})\right\|}{\|x\|}=\frac{\|A\|*\left\|A^{-1}\right\|*\left\|(b-\hat{b})\right\|}{\|A\|\|x\|}$$

Usando la desigualdad , obtenemos:

$$\frac{\left\|x-\hat{x}\right\|}{\|x\|}\leq\frac{\left\|A^{-1}\right\|\left\|(b-\hat{b})\right\|}{\|x\|}=\frac{\|A\|*\left\|A^{-1}\right\|*\left\|(b-\hat{b})\right\|}{\|b\|}$$

EL condicionamiento de A es:

$$cond\,(A)=\|A\|\left\|A^{-1}\right\|$$

Este valor es importante, ya que determina la sensibilidad de la solución frente a las perturbaciones de b. SI cond(A) es grande, el sistema esta *mal condicionado,* en cambio, si es pequeño el sistema está *bien condicionado.*

Se obtiene de la matriz A:

```
 A=matriza(70);
a1=norm(A);
 a2=norm(A^-1);
condA=a1*a2
condA =  6.8361
```

Condicionamiento toma valores de 1 a infinito. Cuanto mayor es condA, peor es el comportamiento de la matriz.

Se deduce que, dado que condA=6,8361, el condicionamiento de la matriz es bastante bueno, por lo que una perturbación en b no debiera afectar significativamente el resultado.

C.2) Solución para el sistema perturbado.

Se calcula la solución por método Cholesky para el sistema normal, luego el perturbado, posteriormente se obtiene el la norma del error (EE) que es la diferencia entre ambos, este valor indica cuanto cambia el sistema al perturbar b.

```
chole1=cholesky(A,b);%solución sistema normal

bp=vectorbp(70,1.9998);

chole2=cholesky(A,bp);%solución sistema perturbado.

Error=norm(chole1-chole2)=0.0013
```

Se observa que el sistema perturbado provoca un error en la solución de 0.0013. Que es un valor pequeño. Así, dependiendo del problema, puede considerarse despreciable.

C.- Conclusiones.

A lo largo del desarrollo de este trabajo se observó la eficacia de los métodos iterativos para obtener soluciones que, en problemas complejos, llegan a ser la mejor alternativa para obtener resultados de buena calidad y que, en forma directa o manual, resulta complejo obtenerlos.

La primera parte consistió en un estudio de doce métodos de iteración del tipo $x_{n+1} = F(x_n)$ en que la función F tiene incluida variables del tipo

$$\{x_n, f(x_n), f'(x_n), f''(x_n)\}$$

donde $f'(x_n)$ es la función a la cual se debe encontrar el cero. Estos son métodos iterativos que requieren los valores del paso anterior para obtener resultados, por ende, se necesita un solo valor inicial para comenzar a iterar. Según el grado de convergencia del método y la condición inicial, la el método converge en unos pasos (en general sucedió que siempre era una cantidad inferior a 50) o, en caso contrario diverge.

Gracias a que fúe posible programar los métodos y la obtención de los resultados, se logró en forma rápida ordenar, clasificar, obtener resultados y graficar todos los métodos y funciones en estudio. En general, los métodos Halley, Chebisev y Super Halley aseguran la convergencia en pocos pasos.

La segunda parte consistió en el estudio de una matriz particular A definida por la guia y el vector b, que son parte del sistema de ecuaciones AX=b, la matriz A resulta ser del tipo A es simétrica, definida positiva y tridiagonal, por lo que es posible hacer varios estudios sobre ella. En primer lugar se buscó solución con el método Cholesky, luego se programaron métodos iterativos de Jacobi, Gauss Siedel y Relajación Sucesiva (SOR), resultaron ser métodos efectivos para encontrar la solución al sistema, destacando SOR, cuya convergencia fúe en 20 pasos.

Por último, para estudiar las perturbaciónes el vector b, se obtuvo un condicionamiento de A bastante pequeño, lo que aseguró que el sistema fuese estable ante perturbaciones, cuestión comprobada en la parte c.

D.- Referencias.

1. Wikipedia
2. Manual de Matlab
3. www.google.com
4. Apuntes de Cálculo Numérico. MA33a.