Ejercicio 1

El metodo cerrado que hemos utilizado para el calculo de la raiz de la funcion **y=(cosx)/x,** ha sido el de la bisección. Una vez ejecutado el programa con un error de 10E−10, en el intervalo cerrado 4,7, el resultado en la iteración numero 35 ha sido el siguiente :

raiz= 4.7123889803

El listado del programa es el que mostramos a continuación:

```
program BISECCION;

uses

wincrt;

var

e,a,b,c,n,fc,fa,fb,solucion,resul:real;

cont,l:integer;

function CALCULA(a:real):real;

var

y:real;

begin

y:=cos(a)/a;

calcula:=y

end;

BEGIN

clrscr;

write('===> Introduce el error: ');

readln(e);

writeln('HEMOS TOMADO COMO VALORES a=4 y b=7');

writeln;

a:=4;
```

```
b:=7;

c:=(a+b)/2;

fc:=calcula(c);

cont:=0;

repeat

n:=b−c;

if (abs(n)<e) or (fc=0.0) then

solucion:=c

else

begin

fc:=calcula(c);

fb:=calcula(b);

resul:=fb*fc;

if (resul<0.0) then

a:=c

else

b:=c

end;

cont:=cont+1;

writeln('*** Iteraci¢n n§: ',cont,' el resultado es: ',c:12:10);

c:=(a+b)/2;

until (abs(n)<e) or (fc=0.0);

writeln;

write('===> La soluci¢n es: ',solucion:12:10);

END.
```

A continuación hemos implementado el metodo de Newton para calcular la citada raiz, tomando como valores iniciales 4,5,6 y 7. Los resultado obtenidos han sido los siguientes:

valor inicial=4 numero de iteraciones=3

4.7123880727

4.7123889804

4.7123889804

valor incial=5 número de iteraciones= 4

4.7122428734

4.7123889759

4.7123889804

4.7123889804

valor incial=6 número de iteraciones=6

−1.3625857839

−1.5454972917

−1.5703940903

−1.5707962238

−1.5707963268

−1.5707963268

valor incial=7 número de iteraciones= 4

7.8509627891

7.8539804877

7.8539816340

7.8539816340

La implementación del metodo de Newton, la mostramos a continuación:

```
program NEWTON;

uses

wincrt;

var
```

```
k,i:integer;
e,a,b,fd,n:real;
function CALCULA(a:real):real;
begin
calcula:=cos(a)/a;
end;
function DERIVADA(a:real):real;
begin
derivada:=((−sin(a)*a)−cos(a))/(a*a);
end;
BEGIN
clrscr;
write('===> Introduce el valor de a: ');
readln(a);
writeln;
write('===> Introduce el error: ');
readln(e);
writeln;
write('===> Introduce el n§ de iteraciones: ');
readln(i);
n:=a;
k:=1;
b:=a−(calcula(a)/derivada(a));
writeln;
while (abs(a−b)>e) and (i>k) do
begin
```

```
a:=b;

b:=a−(calcula(a)/derivada(a));

writeln('===> Iteracion ',k:2,' valor ',b:12:10);

k:=k+1;

end;

writeln;

if k>i then

writeln ('No converge para el valor ',n:4:2)

else

writeln(' La soluci¢n para el valor ',n:4:2,' es: ',b:12:10);

END.
```

Una vez obtenida la raiz mediante los metodos arriba citados, nos disponemos a realizar su representación gráfica, la cual, mostramos a continuación:

Ejercicio 2

En este ejercicio hemos utilizado el método cerrado de la bisección junto al método de Newton para obtener un método más efectivo. Para ello hemos empezado utilizando el método cerrado hasta conseguir una precisión determinada, y después, hemos continuado aplicando el método de Newton.

Hemos utilizado el método de Newton para calcular la raiz de la función y=cos(ex)+x, con una precisión de 10E−8, dentro del intervalo cerrado −10,10,y con un número máximo de iteraciones de 20, para ello hemos tomado distintos valores iniciales, dentro del citado intervalo, cuyos resultados mostramos a continuación:

valor inicial	***nº iteraciones***	***resultado***
−9	4	−0.9219354434
−8	4	−0.9219354434
−5	4	−0.9219354434
−1	3	−0.9219354434
3	20	3.0256902393
2	20	3.0256937006
5	20	5.9027973640
8	20	8.2183462274
9	20	9.0015929660

Como podemos observar en la tabla al utilizar como valores inciales números negativos el método converge hacia una raiz de la función, por el contrario si los valores iniciales utilizados son números positivos el método no converge.

A continuación mostramos el listado del método de Newton para la citada función:

```
Program Newton;
uses
wincrt;
var
k,i,x:integer;
error,b,fd,n,a,cal,der:real;
function CALCULA(a:real):real;
begin
calcula:=cos(exp(a))+a;
end;
function DERIVADA(a:real):real;
begin
derivada:=−(exp(a))*sin(exp(a))+1
end;
BEGIN
clrscr;
write('===> Introduce el valor de a: ');
readln(a);
writeln;
writeln('*** El error es de 10E−8 ***');
writeln;
write('===> Introduce el n§ de iteraciones: ');
readln(i);
error:=0.00000001;
n:=a;
```

```
k:=1;

b:=a−(calcula(a)/derivada(a));

writeln;

while (abs(a−b)>error) and (i>k) do

begin

a:=b;

b:=a−(calcula(a)/derivada(a));

writeln('===> Iteracion ',k:2,' valor ',b:12:10);

k:=k+1;

end;

writeln;

if k>i then

writeln ('No converge para el valor ',n:4:2)

else

writeln(' La soluci¢n para el valor ',n:4:2,' es: ',b:12:10);

END.
```

En el siguiente apartado hemos utilizado el método combinado de Newton y Bisección para la obtección de la raiz de la función antes citada. Hemos tomado como intervalo para el método de Bisección (−10,10), y hemos utilizado distintas precisiones para realizar los calculos. La precisión para el método de Newton continúa siendo 10E−8. Los resultados son los que mostramos a continuación:

====> Error: 1 Nº Iteraciones: 2 Raiz: −0.922891967

====> Error: 10E−1 Nº Iteraciones: 2 Raiz: −0.922891967

====> Error: 10E−2 Nº Iteraciones: 3 Raiz: −0.9221356048

El listado del programa es el siguiente:

```
program NEWTON_BISECCION;

uses

wincrt;

var
```

```
k,i,x:integer;
a,b,c,funa,funb,func,error,error1:real;
solucion:boolean;
ca:char;
function CALCULA(a:real):real;
begin
calcula:=cos(exp(a))+a;
end;
function DERIVADA(a:real):real;
begin
derivada:=−(exp(a))*sin(exp(a))+1
end;
BEGIN
clrscr;
solucion:=false;
repeat
writeln('−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−');
writeln(' Se pediran los valores a y b, hasta que f(a)*f(b)<0');
writeln('−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−');
writeln;
write('===> Dame el valor de a: ');
readln(a);
write('===> Dame el valor de b: ');
readln(b);
funa:=calcula(a);
funb:=calcula(b);
```

```
until (funa*funb)<0;
write('===> Dame el error para el metodo de Bisecci¢n: ');
readln(error);
writeln('===> Error para el metodo de Newton: 10E−8 ');
writeln;
write('===> Dame el n§ de iteraciones: ');
readln(i);
while not(solucion) do
begin
c:=(a+b)/2;
func:=calcula(c);
solucion:=(abs(b−c)<error) or (func=0);
if not solucion then
if funb*func<0 then
a:=c
else
b:=c;
end;
error1:=0.00000001;
solucion:=false;
k:=1;
a:=c;
b:=a−(funa/derivada(a));
while (abs(a−b)>error) and (i>k) do
begin
a:=b;
```

```
b:=a−(calcula(a)/derivada(a));

writeln('===> Iteracion ',k:2,' valor ',b:12:10);

k:=k+1;

end;

writeln;

writeln;

if k>i then

writeln('**** La funci¢n no converge ****')

else

writeln ('**** La raiz del polinomio es: ',b:10:8,' ****');

END.
```

Conclusiones: Como era de esperar el segundo caso en el cual utilizados un método combinado para el calculo de raices es bastante mejor que el primero, ya que para obtener una buena aproximación a la raiz en el primer caso hemos tenido que realizar varias pruebas tomando distintos valores iniciales, sin embargo, en el segundo tanto solo hemos tenido que introducir el intervalo de busqueda y ejecutar el programa una sola vez para conseguir los resultados deseados.

Dado que el coste computacional depende del número de iteraciones realizadas, hemos de afirmar que el segundo algoritmo es mejor que el primero ya que el número necesario de iteraciones para obtener el mismo resultado es menor.

Ejercicio 3

En la primera parte de este ejercicio se nos pedia aplicar la aceleración del punto fijo utilizando como función de iteración el método de Newton, para ello hemos utilizado el algoritmo de Steffensen que se nos suministra en el presente boletín de prácticas.

El listado del programa implementado para la resolución del citado algoritmo ha sido el siguiente:

```
program PTO_FIJO_NEWTON;

uses

wincrt;

type

vector=array[0..2] of real;

var
```

```
x,Ix,xN:vector;
R1:real;
aux:integer;
k,i:integer;
x0,x1,error,n:real;
function POT(x:real;n:integer):real;
begin
if n<>1 then
pot:=pot(x,n−1)*x
else
pot:=x;
end;
function CALCULA(x:real):real;
begin
calcula:=(pot(x,6))−x−1;
end;
function DERIVADA(x:real):real;
begin
derivada:=(6*(pot(x,5)))−1;
end;
function g(x:real):real;
begin
g:=x−(calcula(x)/derivada(x));
end;
BEGIN
clrscr;
```

```
writeln('****** METODO DE STTEFFENSEN, UTILIZANDO ITERACION DE NEWTON ******');

writeln;

write('===> Introduce el punto de partida: ');

readln(x[0]);

writeln;

write('===> Introduce el error: ');

readln(error);

writeln;

write('===> Introduce el n§ de iteraciones: ');

readln(i);

aux:=0;

repeat

aux:=aux+1;

x[1]:=g(x[0]);

x[2]:=g(x[1]);

Ix[0]:=x[1]−x[0];

Ix[1]:=x[2]−x[1];

R1:=Ix[1]/Ix[0];

xN[2]:=x[1]+Ix[1]/(1−R1);

x[0]:=xN[2];

writeln(g(xN[2]):10:8,'para xN[2]=',xN[2]:12:10);

until abs(xN[2]−x[2])<error;

writeln;

writeln('====> La raiz obtenida por el metodo de Steffensen utilizando la iteracion de newton es:
',xN[2]:10:8);

writeln;

writeln('===> El n§ iteraciones utilizadas con el metodo de Steffensen es: ',aux);
```

END.

En la segunda parte de este ejercicio hemos implementado el método de Steffensen variando la forma de construir la sucesión, o sea, obteniendo cada punto cada tres iteraciones en lugar de cada dos que es como lo hace el método original.

Listado del programa:

```
program STEFFENSEN_NEWTON;

uses

wincrt;

var

a,error,newt,steff,steff1:real;

function POT(x:real;n:integer):real;

begin

if n<>1 then

pot:=pot(x,n−1)*x

else

pot:=x;

end;

function G(x:real):real;

begin

g:=pot(x,6)−1;

end;

function STEFFENSEN(e,a:real):real;

var

x0,x1,x2,x3,Ix0,Ix1,Ix2,x2N,R1:real;

aux:integer;

begin

writeln;
```

```
writeln('******** METODO DE STEFFENSEN MODIFICADO ********');
writeln;
x0:=a;
aux:=0;
repeat
aux:=aux+1;
x1:=g(x0);
x2:=g(x1);
x3:=g(x2);
Ix0:=x1−x0;
Ix1:=x2−x1;
Ix2:=x3−x2;
R1:=Ix2/Ix1;
x2N:=x2+Ix2/(1−R1);
x0:=x2N;
writeln('Iteracion: ',aux,' Valor de x2N=',x2N:12:10);
until abs(x3−x2N)<e;
writeln;
writeln;
writeln('===> El n§ iteraciones utilizadas con el metodo de Steffensen es: ',aux);
STEFFENSEN:=x2N;
end;
BEGIN
clrscr;
write(' ====> Introduce el valor del error: ');
readln(error);
```

```
writeln;

write(' ====> Introduce el valor de x: ');

readln(a);

clrscr;

steff:=STEFFENSEN(error,a);

readln;

clrscr;

writeln;

writeln('*****************************************************************');

writeln('*****************************************************************');

writeln;

writeln('====> La raiz obtenida por el metodo de Steffensen es: ',steff:10:8);

writeln;

writeln('*****************************************************************');

writeln('*****************************************************************');

END.
```

Ejercicio 4

En este ejercicio se nos pide comparar el coste temporal de los algoritmo de Steffensen, Newton y los dos implementados en el ejercicio 3, para ello hemos utilizado la función x6−x−1, tomando como puntos de partida la siguientes valores para x0: 1, 0.7, 0.6.

Dado que al realizar el coste temporal de los citados algoritmos los tiempos obtenidos eran prácticamente despreciables, hemos decidido evaluar la eficiencia de los algoritmos basandonos en el coste computacional, o sea, el número de iteraciones que cada uno de ellos ha necesitado realizar para la obtención de los resultados.

A continuación mostramos los resultados de los distintos algoritmos, en diferentes tablas:

METODO DE STEFFENSEN−NEWTON (Ejercicio 3a)

Precisión	***x0***	***Nº Iteraciones***	***Raiz***
10E−5	1	1	1.13472414
10E−5	0.7	5	−0.7780895987
10E−5	0.6	5	−0.7780895987
10E−10	1	3	1.13472414

10E−10	0.7	6	−0.7780895987
10E−10	0.6	6	−0.7780895987

METODO DE STEFFENSEN−MODIFICADO (Ejercicio 3b)

Precisión	***x0***	***Nº Iteraciones***	***Raiz***
10E−5	1	19	−0.7780899587
10E−5	0.7	8	−0.7780895987
10E−5	0.6	14	−0.7780895987
10E−10	1	18	−0.7780895987
10E−10	0.7	6	−0.7780895987
10E−10	0.6	13	−0.7780895987

METODO DE STEFFENSEN (Ejercicio 4)

Precisión	***x0***	***Nº Iteraciones***	***Raiz***
10E−5	1	5	−0.7780895987
10E−5	0.7	6	−0.7780895987
10E−5	0.6	**7**	−0.7780895987
10E−10	1	6	−0.7780895987
10E−10	0.7	7	−0.7780895987
10E−10	0.6	8	−0.7780895987

METODO DE NEWTON (Ejercicio 4)

Precisión	***x0***	***Nº Iteraciones***	***Raiz***
10E−5	1	5	1.13472414
10E−5	0.7	33	1.13472414
10E−5	0.6	11	−0.7780895987
10E−10	1	6	1.13472414
10E−10	0.7	34	1.13472414
10E−10	0.6	12	−0.7780895987

En vista de los resultados obtenidos podemos llegar a la conclusión de que el mejor de todos es el método que utiliza el método de aceleración de Steffensen combinado con el método de Newton, y el peor el de Newton.

Ejercicio 5

Para la realización de todos los apartados que a continuación mostramos hemos utilizado el siguiente polinomio:

P(x)= 36 x6 − 373 x4 − 309 x2 +100

Apartado a: Programa que obtiene toda la información posible sobre el número y tipo de las raices de un polinomio.

Numero de raices reales: 2

Numero de multiplicidades: 2

Debido al numero de multiplicidades se pueden dar los siguientes casos:

− Una raíz real simple, otra real triple y una pareja de complejas conjugadas.

− Dos raices reales dobles y una pareja de complejas conjugadas.

LISTADO DEL PROGRAMA

```
program raices;
uses
crt;
type
matriz=array[0..9,0..9] of integer;
vector=array[0..9] of real;
var
grado,i,num_raices,multi:integer;
m_polinomios:matriz;
ca:char;
procedure ini_matriz(var v:matriz);
{ FUNCION PARA INICIALIZAR A CEROS LA MATRIZ DONDE GUARDO LOS POLINOMIOS }
var
i,j:integer;
begin
for i:=0 to 9 do
for j:=0 to 9 do
v[i,j]:=0;
end;
procedure introduce_polinomio(var v:matriz);
{ PROCEDIMIENTO PARA INTRODUCIR LOS DATOS DEL POLINOMIO }
```

```
var
i:integer;
begin
write('Introduce el grado del polinomio :');
readln(grado);
writeln;
for i:=grado downto 0 do
begin
write('Introduce el coeficiente de x',i,': ');
readln(v[grado,i]);
end;
end;
function POT(x:real;n:integer):real;
{ FUNCION RECURSIVA PARA CALCULAR LAS POTENCIAS DE LA X DE LOS POLINOMIOS }
begin
if n<1 then n:=−n;
if n<>1 then pot:=pot(x,n−1)*x
else pot:=x;
end;
function calcula(v:matriz;pto:real;fila:integer):real;
{ FUNCION PARA CALCULAR EL VALOR DE UN POLINOMIO EN UN PUNTO }
var
i:integer;
resul:real;
begin
resul:=0;
```

```
for i:=fila downto 1 do
resul:=(resul+v[fila,i])*pto;
resul:=resul+v[fila,0];
calcula:=resul
end;
procedure derivada(var v:matriz);
{ CALULA LA DERIVADA DE UN POLINOMIO }
var
i:integer;
begin
for i:=0 to grado−1 do
v[grado−1,i]:=v[grado,i+1]*(i+1);
end;
procedure sturm(var v:matriz;grado2:integer);
{ CALCULA LOS DISTINTOS POLINOMIOS DESDE GRADO N−2 HASTA N POR
EL METODO DE LAS SUCESIONES DE STURN }
var
j:integer;
dividendo,divisor,cociente,resto:integer;
begin
for j:=grado2+2 downto 2 do
begin
dividendo:=v[grado2+2,j];
divisor:=v[grado2+1,j−1];
if divisor = 0 then
resto:=−dividendo
```

```
else
begin
cociente:=(dividendo) div (divisor);
resto:=cociente*divisor−dividendo;
end;
v[grado2,j−2]:=−resto;
end;
end;
function numero_raices(v:matriz):integer;
{ FUNCION QUE CALCULA EL NUMERO DE RAICES QUE HAY ENTRE LOS
INTERVALOS +− INFINITO }
var
i,Na,Nb:integer;
pa,pb:vector;
begin
for i:=0 to grado do
begin
pa[i]:=calcula(v,−100,i);
pb[i]:=calcula(v,100,i);
end;
Na:=0;
Nb:=0;
for i:=grado downto 1 do
begin
if ((pa[i]>0) and (pa[i−1]<0)) or ((pa[i]<0) and (pa[i−1]>0)) then
Na:=Na+1;
```

```
if ((pb[i]>0) and (pb[i−1]<0)) or ((pb[i]<0) and (pb[i−1]>0)) then
Nb:=Nb+1;
end;
numero_raices:=abs(Na−Nb);
end;
function multiplicidades(v:matriz):integer;
{ CALCULA LAS MULTIPLICIDADES DEL POLINOMIO, EN CASO DE QUE LAS TENGA }
var
i,j:integer;
mult:integer;
ceros:boolean;
begin
mult:=0;
for i:=grado−2 downto 0 do
begin
j:=0;
ceros:=true;
while (ceros) and (j<=grado−2) do
begin
if (v[i,j]<>0) then
ceros:=false;
j:=j+1;
end;
if ceros then
mult:=mult+1;
end;
```

```
multiplicidades:=mult;
end;
procedure tipo_raices(mult,n_raices:integer);
{ PROCEDIMIENTO QUE MUESTRA POR PANTALLA EL TIPO DE RAICES QUE
POSEE EL POLINOMIO, DEPENDIENDO DE LAS MULTIPLICIDADES }
begin
writeln('El numero de raices reales distintas es: ',n_raices);
writeln;
if mult=0 then
writeln('No existen multiplicidades')
else
if mult=1 then
begin
writeln('Existe una multiplicidad,');
writeln('por lo que una de las raices tendra multiplicidad dos');
end
else
begin
writeln('Existen dos multiplicidades, por lo que se pueden dar los casos:');
writeln(' − Una raiz real simple, otra triple y una pareja de complejas conjugadas.');
writeln(' − Dos raices reales dobles y una pareja de complejas conjugadas.');
end;
end;
begin
clrscr;
ini_matriz(m_polinomios);
```

```
writeln('********** CALCULO DEL NUMERO Y TIPO DE RAICES DE UN POLINOMIO **********');

writeln;

introduce_polinomio(m_polinomios);

writeln;

derivada(m_polinomios);

for i:=grado−2 downto 0 do

polinomios(m_polinomios,i);

num_raices:=numero_raices(m_polinomios);

multi:=multiplicidades(m_polinomios);

tipo_raices(multi,num_raices);

ca:=readkey;

end.
```

Apartado b−c: En este apartado hemos implementado un programa que halla intervalos de longitud 1, que contienen una única raíz, para lo cual nos hemos basado en la proposición de Cauchy, y dependiendo del número de intervalos que halla calcularemos los valores de r y s para su posterior utilización en el método d Bairstow (apartado d).

Intervalo obtenido según la proposición de Cauchy [−10.3611,10.3611

Intervalos que contienen una única raiz:

[−1.3611111111, −0.3611111111

[−0.3611111111, 0.6388888888

Valores obtenidos para r y s:

r= 0.7222222222 s= −0.1195987654

LISTADO DEL PROGRAMA

```
program raices;

uses

crt;

type

matriz=array[0..9,0..9] of integer;
```

```
vector=array[0..9] of real;
var
grado,i,k,num_raices,multi,raiz:integer;
m_polinomios:matriz;
ca:char;
a,b,numero,maximo,r1,r2,r,s:real;
rs:vector;
procedure ini_matriz(var v:matriz);
{ INICIALIZA LA MATRIZ, DONDE VOY A GUARDAR LOS POLINOMIOS, A CERO }
var
i,j:integer;
begin
for i:=0 to 9 do
for j:=0 to 9 do
v[i,j]:=0;
end;
procedure introduce_polinomio(var v:matriz);
{ PROCEDIMIENTO PARA INTRODUCIR LOS DATOS DEL POLINOMIO }
var
i:integer;
begin
write('Introduce el grado del polinomio :');
readln(grado);
writeln;
for i:=grado downto 0 do
begin
```

```
write('Introduce el coeficiente de x',i,': ');
readln(v[grado,i]);
end;
end;
function POT(x:real;n:integer):real;
{ FUNCION RECURSIVA PARA HALLAR LAS POTENCIAS DE LAS X DEL POLINOMIO }
begin
if n<1 then n:=−n;
if n<>1 then pot:=pot(x,n−1)*x
else pot:=x;
end;
function calcula(v:matriz;pto:real;fila:integer):real;
{ RESUELVE EL VALOR DEL POLINOMIO EN UN DETERMINADO PUNTO }
var
i:integer;
resul:real;
begin
resul:=0;
for i:=fila downto 1 do
resul:=(resul+v[fila,i])*pto;
resul:=resul+v[fila,0];
calcula:=resul
end;
procedure derivada(var v:matriz);
{ PROCEDIMIENTO PARA HALLAR LA DERIVADA DE UN POLINOMIO }
var
```

```
i:integer;
begin
for i:=0 to grado−1 do
v[grado−1,i]:=v[grado,i+1]*(i+1);
end;
procedure sturm(var v:matriz;grado2:integer);
{ CALCULA LOS DISTINTOS POLINOMIOS DESDE GRADO N−2 HASTA N POR
EL METODO DE LAS SUCESIONES DE STURN }
var
j:integer;
dividendo,divisor,cociente,resto:integer;
begin
for j:=grado2+2 downto 2 do
begin
dividendo:=v[grado2+2,j];
divisor:=v[grado2+1,j−1];
if divisor = 0 then
resto:=−dividendo
else
begin
cociente:=(dividendo) div (divisor);
resto:=cociente*divisor−dividendo;
end;
v[grado2,j−2]:=−resto;
end;
end;
```

```
function intervalo(v:matriz):real;
{ FUNCION QUE HALLA LOS INTERVALOS DONDE SE ENCUENTRAN LAS RAICES
DE UN POLINOMIO POR LA PROPOSICION DE CAUCHY }
var
i:integer;
max,resul:real;
begin
max:=abs(v[grado,grado−1]/v[grado,grado]);
for i:=grado−2 downto 0 do
begin
resul:=abs(v[grado,i]/v[grado,grado]);
if resul > max then
max:=resul;
end;
intervalo:=max
end;
function numero_raices(v:matriz;a,b:real):integer;
{ FUNCION QUE CALCULA EL NUMERO DE RAICES QUE HAY ENTRE LOS
INTERVALOS A,B QUE LE PASES COMO PARAMETROS }
var
i,Na,Nb:integer;
pa,pb:vector;
begin
for i:=0 to grado do
begin
pa[i]:=calcula(v,a,i);
```

```
pb[i]:=calcula(v,b,i);
end;
Na:=0;
Nb:=0;
for i:=grado downto 1 do
begin
if ((pa[i]>0) and (pa[i−1]<0)) or ((pa[i]<0) and (pa[i−1]>0)) then
Na:=Na+1;
if ((pb[i]>0) and (pb[i−1]<0)) or ((pb[i]<0) and (pb[i−1]>0)) then
Nb:=Nb+1;
end;
numero_raices:=abs(Na−Nb);
end;
begin
clrscr;
ini_matriz(m_polinomios);
writeln('********** CALCULO DEL NUMERO Y TIPO DE RAICES DE UN POLINOMIO **********');
writeln;
introduce_polinomio(m_polinomios);
writeln;
derivada(m_polinomios);
for i:=grado−2 downto 0 do
sturm(m_polinomios,i);
maximo:=intervalo(m_polinomios);
numero:=−maximo;
k:=0;
```

```
while numero<=maximo do
begin
a:=numero;
b:=numero+1;
numero:=numero+1;
num_raices:=numero_raices(m_polinomios,a,b);
if num_raices=1 then
begin
writeln('Hay una unica raiz entre el intevalo [',a:1:5,',',b:1:5,']');
raiz:=raiz+1;
rs[k]:=a;
k:=k+1;
rs[k]:=b;
k:=k+1;
end
end;
if raiz>=2 then
begin
r1:=(rs[0]+rs[1])/2;
r2:=(rs[2]+rs[3])/2;
r:=−(r1+r2);
s:=r1*r2;
end
else
begin
r:=0;
```

```
s:=0;

end;

writeln;

writeln('El valor de la r es :',r,' y el de la s es :',s);

ca:=readkey;

end.
```

Apartado d: Para la realización de este apartado hemos implementado el método de Bairstow, utilizando los apartados anteriores.

LISTADO DEL PROGRAMA

```
program Metodo_Bairstow;

uses

crt;

type

matriz=array[0..9,0..9] of integer;

vector=array[0..9] of real;

soluciones=record

sreal,scomp:real;

end;

var

grado:integer;

m_polinomios:matriz;

r,s:real;

vector1:vector;

procedure ini_matriz(var v:matriz);

{ FUNCION PARA INICIALIZAR A CEROS LA MATRIZ DONDE GUARDO LOS POLINOMIOS }

var

i,j:integer;
```

```
begin
for i:=0 to 9 do
for j:=0 to 9 do
v[i,j]:=0;
end;
function POT(x:real;n:integer):real;
{ FUNCION RECURSIVA PARA CALCULAR LAS POTENCIAS DE LA X DE LOS POLINOMIOS }
begin
if n<1 then n:=−n;
if n<>1 then pot:=pot(x,n−1)*x
else pot:=x;
end;
function calcula(v:matriz;pto:real;fila:integer):real;
{ FUNCION PARA CALCULAR EL VALOR DE UN POLINOMIO EN UN PUNTO }
var
i:integer;
resul:real;
begin
resul:=0;
for i:=fila downto 1 do
resul:=(resul+v[fila,i])*pto;
resul:=resul+v[fila,0];
calcula:=resul
end;
procedure derivada(var v:matriz);
{ CALULA LA DERIVADA DE UN POLINOMIO }
```

```
var
i:integer;
begin
for i:=0 to grado−1 do
v[grado−1,i]:=v[grado,i+1]*(i+1);
end;
procedure sturm(var v:matriz;grado2:integer);
{ CALCULA LOS DISTINTOS POLINOMIOS DESDE GRADO N−2 HASTA N POR
EL METODO DE LAS SUCESIONES DE STURN }
var
j:integer;
dividendo,divisor,cociente,resto:integer;
begin
for j:=grado2+2 downto 2 do
begin
dividendo:=v[grado2+2,j];
divisor:=v[grado2+1,j−1];
if divisor = 0 then
resto:=−dividendo
else
begin
cociente:=(dividendo) div (divisor);
resto:=cociente*divisor−dividendo;
end;
v[grado2,j−2]:=−resto;
end;
```

```
end;
function intervalo(v:matriz):real;
{ FUNCION QUE HALLA LOS INTERVALOS DONDE SE ENCUENTRAN LAS RAICES
DE UN POLINOMIO POR LA PROPOSICION DE CAUCHY }
var
i:integer;
max,resul:real;
begin
max:=abs(v[grado,grado−1]/v[grado,grado]);
for i:=grado−2 downto 0 do
begin
resul:=abs(v[grado,i]/v[grado,grado]);
if resul > max then
max:=resul;
end;
intervalo:=max
end;
function numero_raices(v:matriz;a,b:real):integer;
{ FUNCION QUE CALCULA EL NUMERO DE RAICES QUE HAY ENTRE LOS
INTERVALOS A,B QUE LE PASES COMO PARAMETROS }
var
i,Na,Nb:integer;
pa,pb:vector;
begin
for i:=0 to grado do
begin
```

```
pa[i]:=calcula(v,a,i);
pb[i]:=calcula(v,b,i);
end;
Na:=0;
Nb:=0;
for i:=grado downto 1 do
begin
if ((pa[i]>0) and (pa[i−1]<0)) or ((pa[i]<0) and (pa[i−1]>0)) then
Na:=Na+1;
if ((pb[i]>0) and (pb[i−1]<0)) or ((pb[i]<0) and (pb[i−1]>0)) then
Nb:=Nb+1;
end;
numero_raices:=abs(Na−Nb);
end;
function multiplicidades(v:matriz):integer;
{ CALCULA LAS MULTIPLICIDADES DEL POLINOMIO, EN CASO DE QUE LAS TENGA }
var
i,j:integer;
mult:integer;
ceros:boolean;
begin
mult:=0;
for i:=grado−2 downto 0 do
begin
j:=0;
ceros:=true;
```

```
while (ceros) and (j<=grado−2) do
begin
if (v[i,j]<>0) then
ceros:=false;
j:=j+1;
end;
if ceros then
mult:=mult+1;
end;
multiplicidades:=mult;
end;
procedure tipo_raices(mult,n_raices:integer);
{ PROCEDIMIENTO QUE MUESTRA POR PANTALLA EL TIPO DE RAICES QUE
POSEE EL POLINOMIO, DEPENDIENDO DE LAS MULTIPLICIDADES }
begin
writeln('El numero de raices reales distintas es: ',n_raices);
writeln;
if mult=0 then
writeln('No existen multiplicidades')
else
if mult=1 then
begin
writeln('Existe una multiplicidad,');
writeln('por lo que una de las raices tendra multiplicidad dos');
end
else
```

```
begin
writeln('Existen dos multiplicidades, por lo que se pueden dar los casos:');
writeln(' − Una raiz real simple, otra triple y una pareja de complejas conjugadas.');
writeln(' − Dos raices reales dobles y una pareja de complejas conjugadas.');
end;
end;
procedure calculo_raices(v:matriz;grado:integer;var r,s:real);
{ PROCEDIMIENTO QUE UTILIZA LOS ANTERIORES PARA DAR EL NUMERO
Y TIPO DE RAICES, ASI COMO LOS INTERVALOS ENTRE LOS CUAL SE
HALLAN }
var
i,j,multi,num_raices,raiz:integer;
maximo,numero,a,b,r1,r2:real;
rs:vector;
begin
clrscr;
writeln('********** CALCULO DEL NUMERO Y TIPO DE RAICES DE UN POLINOMIO **********');
writeln;
derivada(v);
for i:=grado−2 downto 0 do
sturm(v,i);
num_raices:=numero_raices(v,−100,100);
multi:=multiplicidades(v);
tipo_raices(multi,num_raices);
maximo:=intervalo(v);
numero:=−maximo;
```

```
raiz:=0;
j:=0;
while numero<=maximo do
begin
a:=numero;
b:=numero+1;
numero:=numero+1;
num_raices:=numero_raices(v,a,b);
if num_raices=1 then
begin
writeln('Hay una unica raiz entre el intevalo [',a:1:5,',',b:1:5,']');
raiz:=raiz+1;
rs[j]:=a;
j:=j+1;
rs[j]:=b;
j:=j+1;
end
end;
if raiz>=2 then
begin
r1:=(rs[0]+rs[1])/2;
r2:=(rs[2]+rs[3])/2;
r:=−(r1+r2);
s:=r1*r2;
end
else
```

```
begin
r:=0;
s:=0;
end;
writeln;
writeln;
writeln('===> El valor de la r es :',r:0:5,' y el de la s es :',s:0:5);
readln;
end;
procedure leer_datos(var v:matriz;var grado:integer;var vector1:vector);
{ PROCEDIMIENTO PARA INTRODUCIR POR TECLADO EL POLINOMIO EN CUESTION }
var
i,j:integer;
begin
writeln('************************************************************');
writeln(' INTRODUCE LOS DATOS DEL POLINOMIO');
writeln('************************************************************');
write('===> Dame el grado del polinomio<10: ');
readln(grado);
writeln('===> Dame los coeficiente del polinoio de mayor a menor.');
writeln;
ini_matriz(v);
for i:=grado downto 0 do
begin
write('Introduce el coeficiente de x',i,': ');
readln(v[grado,i]);
```

```
vector1[i]:=v[grado,i];
end;
end;
{ ********************************************************************** }
{ ********************** metodo de Bairstow ************************** }
{ ********************************************************************** }
procedure Ruffini(var v:vector;var r,s:real);
{ PROCEDIMIENTO QUE RESUELVE LA DIVISION DE POLINOMIOS POR EL
METODO DE RUFINI DE ORDEN 2 }
var
aux:vector;
i:integer;
begin
aux[grado]:=v[grado];
aux[grado−1]:=v[grado−1]−(r*v[grado]);
for i:=(grado−2) downto 1 do
aux[i]:=v[i]−(r*aux[i+1])−(s*aux[i+2]);
aux[0]:=v[0]−(s*aux[2]);
for i:=0 to grado do
v[i]:=aux[i];
end;
procedure escribe_solucion(sx:soluciones);
{ MUESTRA POR PANTALLA LAS RAICES DEL POLINOMIO, CUANDO
EL METODO DE BAIRSTOW LAS ENCUENTRA }
begin
write('Soluci¢n = ');
```

```
if (sx.scomp<>0) AND (sx.sreal<>0) then
begin
write(sx.sreal:0:5);
if sx.scomp>0 then
write('+');
writeln(sx.scomp:0:5,'i');
end
else
if (sx.scomp=0) then
writeln(sx.sreal:0:5)
else
writeln(sx.scomp:0:5,'i');
end;
procedure calcular_raices(c,r,s:real;var x1,x2:soluciones);
{ CALCULA LAS RAICES DE UN POLINOMIO DE GRADO 2 }
var
raiz,discriminante:real;
begin
discriminante:=r*r−4*s*c;
if discriminante>=0 then
begin
raiz:=sqrt(discriminante);
x1.sreal:=(−r+raiz)/(2*c);
x2.sreal:=(−r−raiz)/(2*c);
x1.scomp:=0;
x2.scomp:=0;
```

```
end
else
begin
x1.sreal:=−r/(2*c);
x2.sreal:=x1.sreal;
discriminante:=−discriminante;
raiz:=sqrt(discriminante);
x1.scomp:=raiz/(2*c);
x2.scomp:=−raiz/(2*c);
end;
end;
procedure polinomio2(var aux:vector);
{ PROCEDIMIENTO QUE MUESTRA POR PANTALLA EL POLINOMIO DE GRADO 2 }
var
num:integer;
begin
for num:=grado downto 2 do
writeln(' *** Qn−2[',num−2,']=',aux[num]);
end;
procedure cramer(b1r,b1s,b0r,b0s,b1,b0:real;var Ar,As:real;cont:integer);
{ RESUELVE UN SISTEMA DE ECUACIONES POR LA REGLA DE CRAMER }
type
matriz=array[1..2,1..2] of real;
vec=array[1..2] of real;
var
m:matriz;
```

```
v1:vec;
det:real;
ca:char;
begin
det:=b1r*b0s−b1s*b0r;
Ar:=−(b1*b0s−b0*b1s)/det;
As:=−(b0*b1r−b1*b0r)/det;
writeln('===> Los coeficiente del resto son:');
writeln;
writeln(' b1= ',b1:12:10,' b0= ',b0:12:10);
writeln;
writeln(' &b1/&r= ',b1r:12:10,' &b0/&r= ',b0r:12:10);
writeln(' &b1/&s= ',b1s:12:10,' &b0/&s= ',b0s:12:10);
writeln;
writeln('===> Incremento de r= ',Ar:12:10);
writeln('===> Incremento de s= ',As:12:10);
writeln;
end;
procedure Bairstow(var v:vector;r,s:real);
{ PROCEDIMIENTO PARA LA RESOLUCION DE LAS RAICES DE UN POLINOMIO
POR EL METODO DE BAIRSTOW }
var
i,j,iter_max,cont,x,num:integer;
v1,v2,v3:vector;
b1r,b1s,b1,error:real;
b0r,b0s,b0:real;
```

```
Ar,As,sol1,sol2,c:real;
x1,x2:soluciones;
grado_dos,superior,solucion:boolean;
begin
clrscr;
writeln('*****************************************************************');
writeln(' APLICACION DEL METODO BAIRSTOW PARA LA OBTENCION DE RAICES');
writeln('*****************************************************************');
write('===> Dame el error admitido: ');
readln(error);
write('===> Introduce el numero de iteraciones: ');
readln(iter_max);
for j:=0 to grado do
begin
v1[j]:=0;
v2[j]:=0;
v3[j]:=0;
end;
cont:=1;
grado_dos:=false;
superior:=false;
repeat
if not superior then
begin
for j:=0 to grado do v1[j]:=v[j];
ruffini(v1,r,s);
```

```
b1:=v1[1];
b0:=v1[0];
end;
for j:=0 to (grado−2) do v2[j]:=v1[j+2];
for j:=1 to (grado−1) do v3[j]:=v1[j+1];
v3[0]:=0;
if (grado−1)>=2 then
begin
ruffini(v3,r,s);
b1r:=−v3[1];
b0r:=−v3[0];
end
else
begin
b1r:=v3[1];
b0r:=v3[0];
end;
if (grado−2)>=2 then
begin
ruffini(v2,r,s);
b1s:=−v2[1];
b0s:=−v2[0];
end
else
begin
b1s:=v2[1];
```

```
b0s:=v2[0];
end;
writeln;
writeln('*****************************************************************');
writeln('*****************************************************************');
writeln;
writeln('===> Los valores de r y s son: r= ', r:12:10, ' s= ',s:12:10);
writeln('===> Los coeficiente del cociente de menor a mayor son los siguiente:');
writeln;
polinomio2(v1);
writeln;
cramer(b1r,b1s,b0r,b0s,b1,b0,Ar,As,cont);
writeln('ITERACION ======= ',cont);
readln;
clrscr;
r:=r+Ar;
s:=s+As;
for j:=0 to grado do v1[j]:=v[j];
ruffini(v1,r,s);
b1:=v1[1];
b0:=v1[0];
if (abs(b1)+abs(b0))<error then
begin
solucion:=true;
calcular_raices(1,r,s,x1,x2);
escribe_solucion(x1);
```

```
escribe_solucion(x2);
readln;
grado:=grado−2;
cont:=1;
if grado<3 then
begin
grado:=grado+2;
c:=v1[grado];
r:=v1[grado−1];
s:=v1[grado−2];
if c<>0 then
begin
calcular_raices(c,r,s,x1,x2);
escribe_solucion(x1);
escribe_solucion(x2);
end
else
begin
sol1:=−(s/r);
writeln(sol1);
readln;
end;
grado_dos:=true;
end
else
for j:=2 to grado+2 do
```

```
v[j−2]:=v1[j];
superior:=false;
end
else superior:=true;
cont:=cont+1;
until (grado_dos) or (cont>iter_max);
if cont>iter_max then
writeln('===> El numero de iteraciones es superior al permitido');
if (not solucion) then
writeln('===> El metodo de Bairstow diverge');
end;
BEGIN
clrscr;
leer_datos(m_polinomios,grado,vector1);
calculo_raices(m_polinomios,grado,r,s);
bairstow(vector1,r,s);
readln;
END.
```

Apartado e: Traza de la ejecución del programa del método de Bairstow.

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = 2.600000000E+01

Qn−2[0 = −3.49916666E+02

Los coeficientes del resto son b1= 413.4409686 y b0= 36.47456897

&b1/&r= 904.429098 &b0/&r= −55.71789

&b1/&s= −465.873456 &b0/&s= 567.96512

incremento de r= −0.5163021416

incremento de s= −0.1148762834

ITERACIÓN ========= 1

r=0.2059200806 s= −0.344750488

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = −7.41312290E+00

Qn−2[0 = −3.63032387E+02

Los coeficientes del resto son b1= 101.37458092 y b0=4.0627050364

&b1/&r= 520.81096092 &b0/&r= −33.27901297

&b1/&s= −141.92986902 &b0/&s= 491.58475086

incremento de r= −0.200600595

incremento de s= −0.021844646

ITERACIÓN ========= 2

r=0.005319485511 s= −0.256319648

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.6000000000E+01

Qn−2[1 = −1.900000000E−01

Qn−2[0 = −3.6377147723E+02

Los coeficientes del resto son b1= 2.6231895286 y b0=−3.10506508

&b1/&r= 493.14744623 &b0/&r= −0.94166753

&b1/&s= −3.6738009308 &b0/&s= 491.1279035

incremento de r= −0.0052724470

incremento de s= −0.0062866047

ITERACIÓN ========= 3

r=0.000470385 s= 0.2500330901

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 =−1.6933858641E−03

Qn−2[0 =−3.6399880868E+02

Los coeficientes del resto son b1= 0.0229911413 y b0=−0.0161733611

&b1/&r= 488.77290047 &b0/&r= −0.0081386

&b1/&s= −0.0325503020 &b0/&s= 488.77289

incremento de r= −0.000470363

incremento de s= 0.0000330889

Las raices obtenidas son:

x= 0.5000000

x=−0.5000000

ITERACIÓN ========= 1

r=0.0000000022 s= −0.2500000012

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = −1.5866881808E−07

Qn−2[0 = −3.5499999999E+02

Los coeficientes del resto son b1= 0.000001525 y b0= −488.75000081

&b1/&r= 345.99999987 &b0/&r= 0.0002117327

&b1/&s= 0.0008469706 &b0/&s= 343.74999985

incremento de r= −0.0000034849

incremento de s= 1.4218181848

ITERACIÓN ========= 2

r= −0.0000034827 s= 1.1718181836

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = 1.2529645678E−04

Qn−2[0 = −4.0618545457E+02

Los coeficientes del resto son b1= −0.0015666485 y b0= 75.975501165

&b1/&r= 448.37090915 &b0/&r= 0.0049443143

&b1/&s= −0.0039256105 &b0/&s= 398.93722637

incremento de r= 0.0000018133

incremento de s= −0.1904447521

ITERACIÓN ========= 3

r= −0.0000016693 s= 0.9813734315

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.60000000E+01

Qn−2[1 = 6.0017203993E−05

Qn−2[0 = −3.9932944349E+02

Los coeficientes del resto son b1= −0.0007247369 y b0= −8.1086942464

&b1/&r= 434.65888702 &b0/&r= 0.0033796552

&b1/&s= −0.0032028845 &b0/&s= 399.98750979

incremento de r= 0.0000018168

incremento de s= 0.0202723684

ITERACIÓN ========= 4

r= 0.0000001474 s= 1.0016457999

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = −5.3858298727E−06

Qn−2[0 = −4.0005924875E+02

Los coeficientes del resto son b1= 0.0000651468 y b0= 0.7176658059

&b1/&r= 4.3611849755 &b0/&r= 0.0033872188

&b1/&s= −0.0032065856 &b0/&s= 399.99990245

incremento de r= −0.0000001626

incremento de s= −0.001794165

ITERACIÓN ========= 5

r= −0.0000000152 s= 0.9998516349

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = 4.6670438524E−07

Qn−2[0 = −3.9999465882E+02

Los coeficientes del resto son b1= −0.0000057513 y b0= −0.064686832

&b1/&r= 435.98931767 &b0/&r= 0.0033867793

&b1/&s= 0.0032061302 &b0/&s= 399.99999917

incremento de r= 0.0000000144

incremento de s= 0.0001617171

ITERACIÓN ========= 6

r= −0.0000000008 s= 1.000013352

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = −5.0999233624E−08

Qn−2[0 = −4.000008063E+02

Los coeficientes del resto son b1= 0.0000005185 y b0= 0.0058210297

&b1/&r= 436.0009613 &b0/&r= 0.0033868394

&b1/&s= −0.0032061025 &b0/&s= 399.99999995

incremento de r= −0.0000000013

incremento de s= −0.0000145526

ITERACIÓN ========= 7

r= −0.0000000021 s= 0.9999987994

Los coeficientes del cociente de mayor a menor grado son los siguientes:

Qn−2[2 = 3.600000000E+01

Qn−2[1 = −4.3358457771E−09

Qn−2[0 = −3.9999995674E+02

Los coeficientes del resto son b1= −0.0000000467 y b0= −0.0005238992

&b1/&r= 435.99991352 &b0/&r= 0.0033868341

&b1/&s= −0.0032060276 &b0/&s= 399.99999996

incremento de r= 0.0000000001

incremento de s= 0.0000013097

Las raices obtenidas son:

x=0.0 + 1.0i

x=0.0 − 1.0i

x=3.33333

x=−3.33333