AE2.CD1.I1

entrada

```
Método de Régula falsi para obtención de 3 primeras raíces
Núm.Ite Tol vertical     Tol horz
80      0.00001          0.00001
x Zurda x Diestra
0.1     0.4
0.6     0.9
1.1     1.5
Método de Newton-Rampson de segundo orden para obtención de 3 primeras raíces
        Tol Verical      Tol horz
        0.00001          0.00001
x Inicia
0.1
0.6
1.1
```

-4

Calif. 9.8

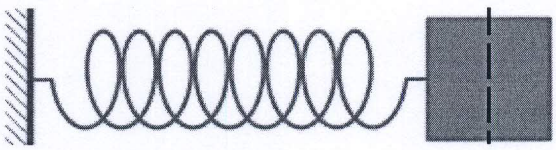
- 4

AE2.CD1.I1

*Concluciones de tus resultados

Cálculo de la posición de un bloque respecto al tiempo en un oscilador armónico

En un sistema donde no hay fricción, una masa de 2 kg está sujeta a un resorte de constante k= 72 N/m como muestra la figura. Se jala el bloque desde el punto de equilibrio unos 0.4 m y se suelta en el instante cero creándose en el bloque un movimiento armónico simple. El bloque adquiere una frecuencia angular de 6 rad/s y se calcula el ángulo de fase a 1.57080 rad. En este problema se desea saber a partir del instante 0, en cuales próximos 3 instantes el bloque pasará por el punto de equilibrio.



La ecuación para conocer la posición de un objeto en un oscilador armónico es:

$$x_{(t)} = x_{max} sin(\omega t + \emptyset)$$

Donde $x_{(t)}$ es la posición del objeto con respecto al tiempo; x_{max} es la posición más alejada del punto de equilibrio que alcanza el bloque; ω es la frecuencia angular; t es el tiempo y $\emptyset$ es el ángulo de fase. Reemplazando los datos que nos dio el problema, obtenemos la ecuación:

$$x_t = 0.4\ m\ \text{sin}(\frac{6rad}{s} t + 0.157080rad)$$

La derivada de esta ecuación es:

$$x_{\dot{t}} = 2.4\frac{m}{s}\ \cos(\frac{6rad}{s} t + 0.157080rad)$$

La segunda derivada de la ecuación sería:

$$x_{\ddot{t}} = -14.4\frac{m}{s}\sin(6\frac{rad}{s} t + 0.157080rad)$$

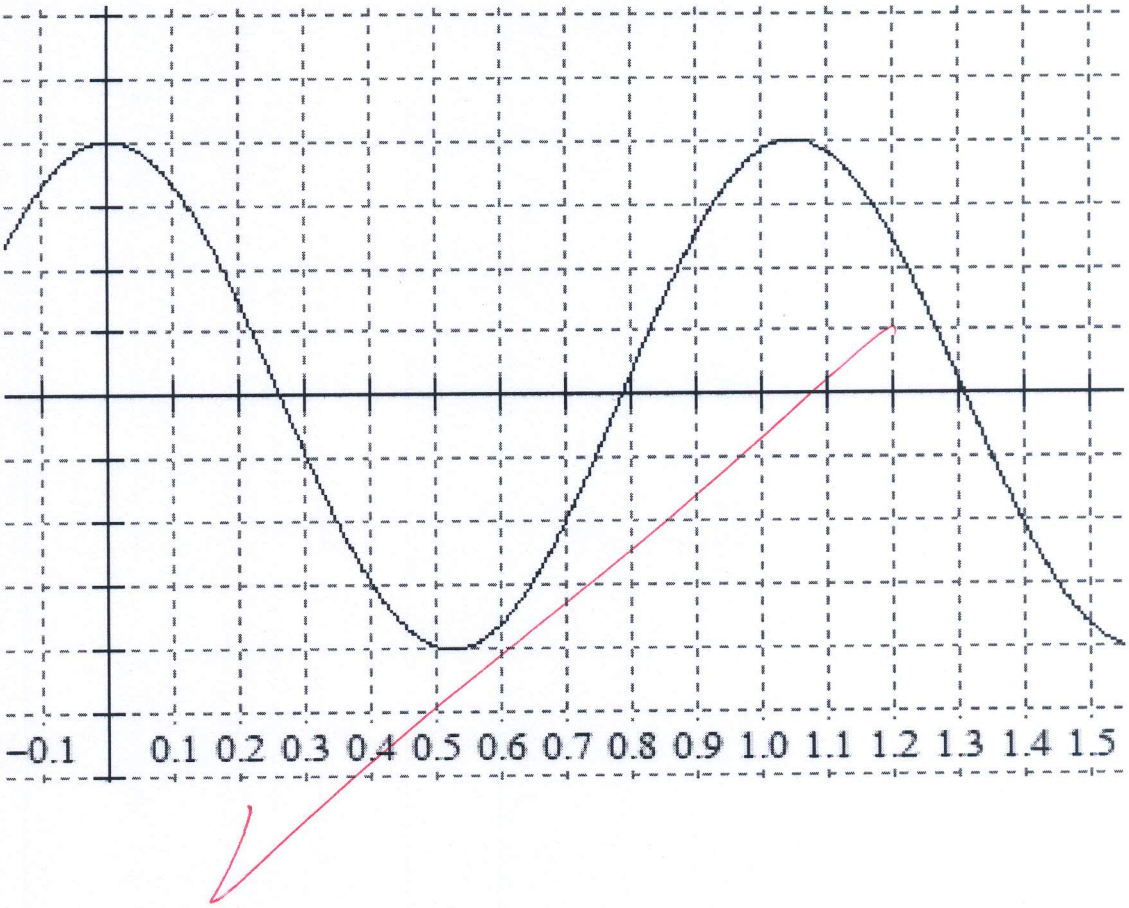
La primera y segunda derivada de la función son necesarias para la realización del método de Newton Rampson de segundo orden.

Cuando la posición del bloque respecto al tiempo es cero, podemos decir que el bloque se encuentra en el punto de equilibrio ya que el extremo desde donde se soltó con velocidad igual a cero fue 0.40 m y el bloque aumentará su velocidad hasta llegar al punto de equilibrio, donde esta será máxima ya que aquí no se presenta ninguna oposición al movimiento y una vez allí comenzará a desacelerar hasta llegar al otro extremo que sería -0.40 m y su energía cinética sería cero. Si queremos conocer los 3 instantes donde el bloque está en su punto de equilibrio debemos calcular los 3 próximos $x_{(t)} = 0$; para realizar estos cálculos podemos utilizar métodos numéricos como el de posición falsa y el de newton-raphson.

Las 3 primeras raíces son: 0.2618, 0.7854 y 1.309

Gráfica de la función:

$$x_t = 0.4\ m\ \sin(\frac{6rad}{s}t + 0.157080rad)$$



```
1   //Equipo: METHOD
2   //Integrantes: Quintero Castillo Ricardo Jesus; Fabila Ceiudo Jose Luis; Hernandez
    Narciso Jonathan
3
4   #include <stdio.h>
5   #include <stdlib.h>
6   #include <math.h>
7
8   //funcionx: Funcion del programa que determina el valor de x en la funcion
    f(x)=0.4sin(6x+1.57080) para cada x que se le asigne
9   //flotantes: xm: recibe valores de xizq[i], xder[i], xreal[i], x[i]; resreal: guarda el
    resultado de las operaciones realizadas en funcionx
10  float funcionx(float xm){
11      float resreal;
12      resreal= 0.4*sin((6*xm)+1.57080);
13      return resreal; }
14
15  //operacionizqder: Funcion del programa que determina la nueva x por medio del metodo de
    posicion falsi
16  //flotantes: xizq: recibe valores de xizq[i]; xder: recibe valores de xder[i]; fxizq:
    recibe valores de fxizq[i]; fxizq: recibe valores de fxder[i]; guarda el resultado de las
    operaciones realizadas en operacionizqder
17  float operacionizqder (float xizq, float xder, float fxizq, float fxder){
18      float xreal;
19      xreal=((xizq*fxder)-(xder*fxizq))/(fxder-fxizq);
20      return xreal; }
21
22  //errorhz: Funcion del programa que calcula el error horizontal
23  //flotantes: xvieja: recibe valores de xizq[i], xder[i], x[i]; xnueva: recibe valores de
    xreal[i], x[i]; trrorhz: guarda el resultado de las operaciones realizadas en errorhz
24  float errorhz (float xvieja, float xnueva) {
25      float trrorhz= fabs((xnueva-xvieja)*100/xnueva);
26      return trrorhz; }
27
28  //vafxreal2: Funcion del programa que calcula el valor absoluto de f(x)
29  //flotantes: fxreal: recibe valores de fxreal[i], fx[i]; valorabsoluto: guarda el
    resultado de las operaciones realizadas en vafxreal2
30  float vafxreal2(float fxreal) {
31      float valorabsoluto= fabs(fxreal);
32      return valorabsoluto;}
33
34  //funcionpx: Funcion del programa que calcula el valor para cualquier x que se asigne en
    la derivada de f(x); f´(x)=2.4cos(6x+1.57080)
35  //flotantes: x: recibe valores de x[i]; result: guarda el resultado de las operaciones
    realizadas en funcionpx
36  float funcionpx(float x) {
37      float result;
38      result= 2.4*cos((6*x)+1.57080);
39      return result;}
40
41  //funcionp2x: Funcion del programa que calcula el valor para cualquier x que se asigne en
    la segunda derivada de f(x); f´´(x)=-14.4sin(6x+1.57080)
42  //flotantes: x: recibe valores de x[i]; result: guarda el resultado de las operaciones
    realizadas en funcionp2x
43  float funcionp2x (float x){
44      float result;
45      result= -14.4*sin((6*x)+1.57080);
46      return result;}
47
48  //xnueva: Funcion del programa que determina la nueva x por medio del metodo de
    newton-ramphson de segundo orden
49  //flotantes: x: recibe valores de x[i]; fx: recibe valores de fx[i]; fpx[i]: recibe
    valores de fp2x[i]; result: guarda el resultado de las operaciones realizadas en xnueva
50  float xnueva (float x, float fx, float fpx, float fp2x) {
51      float result;
52      result= x-(fx/(fpx-((fx*fp2x)/2*fpx)));
53      return result;}
54
55  int main() {
56
57  //Variables tipo enteras: itera: guarda el numero de iteraciones que se asigne en el
    archivo de entrada; nitera: contador de numero de iteraciones que realiza el programa;
    i,j: variables iterativas
58  int itera, i, j, nitera;
59  //Variables tipo flotantes: tolhor: guarda la tolerancia horizontal asiganda en entrada;
    tolver: guarda la tolerancia vertical asignada en entrada; xder[150]: guarda el valor del
    extremo derecho en posición falsi y sus actualizaciones; xizq[150]: guarda el valor del
```

AE2.CD1.I1

AE2.CD1.I1

AE2.CD1.I1

AE2.CD1.I1

AE2.CD1.I1

AE2.CD1.I1

```
     extremo izquierdo en posición falsi y sus actualizaciones; fxder[150]: guarda el valor de
     cada de cada xder[i] evaluada en la funcion f; fxizq[150]: guarda el valor de cada
     xizq[i] evaluada en la funcion f; xreal [150]: guarda cada nueva x obtenida mediante el
     metodo de posicion falsi; fxreal [150]: guarda el valor de cada xreal[i] evaluada en la
     funcion; errhz [150]: guarda el error horizontal calculado en cada iteracion;
     vafxreal[150]: guarda el error vertical calculado en cada iteracion
60   float tolhor, tolver, xder[150], xizq[150], fxder[150], fxizq[150], xreal [150], fxreal
     [150], errhz [150], varrhz [150], vafxreal[150];
61   //Variables tipo caracter: letrero[120]: guarda cada renglon que se desee del archivo de
     entrada
62   char letrero[120];
63
64   //Apuntadores de archivo
65   FILE* entrada;
66   FILE* salida;
67
68   //Asignacion de archivo al apuntador
69   entrada= fopen("entrada.txt", "r");
70   salida= fopen ("salida.txt", "w");
71
72   if(entrada==NULL) {
73       printf("File: error al abrir el archivo de entrada");
74       exit(-1); }
75
76   //Lee las 2 primeras filas del archivo de entrada
77   fgets (letrero,120,entrada);
78   fgets (letrero,120,entrada);
79
80   //Lee del archivo de entrada respectivamente: el numero de iteraciones, tolerancia
     horizontal, tolerancia vertical para el metodo de regula falsi
81   fscanf(entrada,"%d %f %f\n",&itera, &tolhor, &tolver);
82
83   //Lee la 4 fila del archivo de entrada
84   fgets (letrero,120,entrada);
85
86   //Lee los 2 valores extremos con los que iniciara calculos el metodo de posicion falsi
87   fscanf(entrada,"%f %f\n",&xizq[0], &xder[0]);
88
89   printf("Metodo de Regula falsi\nPrimera raiz:\n");
90   printf("#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
91   fprintf(salida, "Metodo de Regula falsi\nPrimera raiz:\n");
92   fprintf(salida, "#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
93
94   nitera=0; i=0;
95
96   /*Actualizacion y calculo de cada tipo de variable y parametro requerido para la
     realizacion del metodo de regula falsi mediante el uso de funciones y un ciclo do/while*/
97   do {
98       fxizq[i]= funcionx(xizq[i]);
99       fxder[i]= funcionx(xder[i]);
100      xreal[i]= operacionizqder(xizq[i],xder[i],fxizq[i],fxder[i]);
101      fxreal[i]= funcionx(xreal[i]);
102      vafxreal[i]= vafxreal2(fxreal[i]);
103
104  //Contadador del numero de iteraciones
105      nitera=nitera+1;
106
AE2.CD1.I1
107  //Esta parte del programa determina cual de los dos extremos se tomara como punto fijo,
     mientras que el otro se actualizara con el nuevo valor de x obtenido por regula falsi en
     cada iteracion y en consecuencia este se utilizara para calcular el error horizontal en
     cada actualizacion
108      if (xreal[i]<0){
109          xizq[i+1]=xreal[i];
110          xder[i+1]=xder[i];
111          errhz[i]= errorhz(xizq[i],xreal[i]);}
112      if (xreal[i]>0) {
113          xder[i+1]=xreal[i];
114          xizq[i+1]=xizq[i];
115          errhz[i]= errorhz(xder[i],xreal[i]);}
116  //Esta parte imprime en el programa y archivo de salida los resultados obtenidos para
     cada parametro por cada iteracion usando el metodo de regula falsi
117      printf("%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
     fxreal[i], errhz[i], vafxreal[i]);
118      fprintf(salida,"%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
     fxreal[i], errhz[i], vafxreal[i]);
119      j=i;
120      i=i+1;
```

```
121   //Las condiciones para que se cumpla el while respectivamente son: el error horizontal
      debe ser menor a la tolerancia horizontal o el error vertical debe ser menor a la
      tolerancia vertical y las iteraciones realizadas por el programa deben ser menor al
      numero de estas asignadas en el archivo de entrada
122       }while ((errhz[j]>tolhor||vafxreal[j]>tolver)&&(itera>nitera));
123
124   printf("El valor de la primera raiz es aproximadamente: %f\n", xreal[j]);
125   fprintf(salida,"El valor de la primera raiz es aproximadamente: %f\n", xreal[j]);
126
127   //Se calcula la segunda raiz
128   printf("Calculos para la segunda raiz:\n");
129   fprintf(salida,"Calculos para la segunda raiz:\n");
130   printf("#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
131   fprintf(salida, "#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
132
133   fscanf(entrada,"%f %f\n",&xizq[0], &xder[0]);
134
135   nitera=0; i=0;
136
137   do {
138       fxizq[i]= funcionx(xizq[i]);
139       fxder[i]= funcionx(xder[i]);
140       xreal[i]= operacionizqder(xizq[i],xder[i],fxizq[i],fxder[i]);
141       fxreal[i]= funcionx(xreal[i]);
142       vafxreal[i]= vafxreal2(fxreal[i]);
143
144       nitera=nitera+1;
145       if (xreal[i]<0){
146           xizq[i+1]=xreal[i];
147           xder[i+1]=xder[i];
148           errhz[i]= errorhz(xizq[i],xreal[i]);}
149       if (xreal[i]>0) {
150           xder[i+1]=xreal[i];
151           xizq[i+1]=xizq[i];
152           errhz[i]= errorhz(xder[i],xreal[i]);}
153
154       printf("%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
      fxreal[i], errhz[i], vafxreal[i]);
155       fprintf(salida,"%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
      fxreal[i], errhz[i], vafxreal[i]);
156       j=i;
157       i=i+1;
158       }while ((errhz[j]>tolhor||vafxreal[j]>tolver)&&(itera>nitera));
159
160   printf("El valor de la segunda raiz es aproximadamente: %f\n", xreal[j]);
161   fprintf(salida,"El valor de la segunda raiz es aproximadamente: %f\n", xreal[j]);
162
163   //Se calcula la tercer raiz
164   printf("Calculos para la tercera raiz:\n");
165   fprintf(salida,"Calculos para la tercera raiz:\n");
166   printf("#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
167   fprintf(salida, "#itera\txzurda\t\txreal\t\txdiestra\tfxreal\t\tEhz\t\t|fxreal|\n");
168
169   fscanf(entrada,"%f %f\n",&xizq[0], &xder[0]);
170   nitera=0; i=0;
171
172   do {
173       fxizq[i]= funcionx(xizq[i]);
174       fxder[i]= funcionx(xder[i]);
175       xreal[i]= operacionizqder(xizq[i],xder[i],fxizq[i],fxder[i]);
176       fxreal[i]= funcionx(xreal[i]);
177       vafxreal[i]= vafxreal2(fxreal[i]);
178
179       nitera=nitera+1;
180       if (xreal[i]<0){
181           xizq[i+1]=xreal[i];
182           xder[i+1]=xder[i];
183           errhz[i]= errorhz(xizq[i],xreal[i]);}
184       if (xreal[i]>0) {
185           xder[i+1]=xreal[i];
186           xizq[i+1]=xizq[i];
187           errhz[i]= errorhz(xder[i],xreal[i]);}
188
189       printf("%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
      fxreal[i], errhz[i], vafxreal[i]);
190       fprintf(salida,"%d\t%f\t%f\t%f\t%f\t%f\t%f\n", nitera, xizq[i], xreal[i], xder[i],
      fxreal[i], errhz[i], vafxreal[i]);
```

AE2.CD1.I1

AE2.CD1.I1

```
191         j=i;
192         i=i+1;
193         }while ((errhz[j]>tolhor||vafxreal[j]>tolver)&&(itera>nitera));
194
195     printf("El valor de la tercera raiz es aproximadamente: %f\n", xreal[j]);
196     fprintf(salida,"El valor de la tercera raiz es aproximadamente: %f\n", xreal[j]);
197
198
199     printf("\n\nMetodo de Newton Rampson de segundo orden\n");
200     fprintf(salida,"\n\nMetodo de Newton Rampson de segundo orden\n");
201     fgets(letrero,120,entrada);
202     fgets(letrero,120,entrada);
203     fscanf(entrada,"%f %f\n", &tolhor, &tolver);
204     fgets(letrero,120,entrada);
205
206     //Variables flotantes: fpx[50]: guarda el valor de cada x[i] evaluada en la primera
        derivada funcion f, fp2x[50]: guarda el valor de cada x[i] evaluada en la segunda
        derivada de la funcion f; x[50]: guarda el valor de cada x desde la incial hasta cada
        nueva que se obtenga por el metodo de newton rampson de segundo orden; mtkhz[50]: guarda
        el error horizontal calculado en cada iteracion; fx[50]: guarda cada valor de x[i]
        evaluado en la funcion f; vafx[50]: guarda el error vertical calculado en cada iteracion
207     float fpx[50], fp2x[50], x[50], mtkhz[50], fx[50], vafx[50];
208
209     printf("Calculo de la primera raiz:\n");
210     fprintf(salida,"Calculo de la primera raiz:\n");
211     printf("#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
212     fprintf(salida,"#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
213     i=0; nitera=0;
214     fscanf(entrada,"%f\n",&x[0]);
215
216     /*Actualizacion y calculo de cada tipo de variable y parametro requerido para la
        realizacion del metodo de newton rampson de segundo orden mediante el uso de funciones y
        un ciclo do/while*/
217     do {
218         fx[i]= funcionx(x[i]);
219         fpx[i]= funcionpx(x[i]);
220         fp2x[i]= funcionp2x(x[i]);
221         x[i+1]= xnueva(x[i],fx[i],fpx[i],fp2x[i]);
222         mtkhz[i]=errorhz(x[i],x[i+1]);
223         vafx[i]=vafxreal2(fx[i]);
224
225     //Esta parte imprime en el programa y archivo de salida los resultados obtenidos para
        cada parametro por cada iteracion usando el metodo de newton rampson de segundo orden
226         printf("%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
227         fprintf(salida,"%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
228         j=i;
229         i=i+1;
230         nitera=j+1;
231     //Las condiciones para que se cumpla el while respectivamente son: el error horizontal
        debe ser menor a la tolerancia horizontal o el error vertical debe ser menor a la
        tolerancia vertical y las iteraciones realizadas por el programa deben ser menor al
        numero de estas asignadas en el archivo de entrada
232     }while ((mtkhz[j]>tolhor||vafx[j]>tolver)&&(itera>nitera));
233     printf("El valor de la primera raiz es aproximadamente: %f\n", x[j]);
234     fprintf(salida,"El valor de la primera raiz es aproximadamente: %f\n", x[j]);
235
236     printf("Calculo de la segunda raiz:\n");
237     fprintf(salida,"Calculo de la segunda raiz:\n");
238     printf("#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
239     fprintf(salida,"#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
240     i=0; nitera=0;
241     fscanf(entrada,"%f\n",&x[0]);
242
243     //Se calcula la segunda raiz
244     do {
245         fx[i]= funcionx(x[i]);
246         fpx[i]= funcionpx(x[i]);
247         fp2x[i]= funcionp2x(x[i]);
248         x[i+1]= xnueva(x[i],fx[i],fpx[i],fp2x[i]);
249         mtkhz[i]=errorhz(x[i],x[i+1]);
250         vafx[i]=vafxreal2(fx[i]);
251
252         printf("%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
253         fprintf(salida,"%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
254         j=i;
255         i=i+1;
256         nitera=j+1;
```

AE2.CD1.I1

AE2.CD1.I1

AE2.CD1.I1

```
262   fprintf(salida,"Calculo de la tercera raiz:\n");
263   printf("#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
264   fprintf(salida,"#itera\tx\t\tfx\t\tEhz\t\t|fxreal|\n");
265   i=0; nitera=0;
266   fscanf(entrada,"%f\n",&x[0]);
267
268   //Se calcula la tercer raiz
269   do {
270       fx[i]= funcionx(x[i]);
271       fpx[i]= funcionpx(x[i]);
272       fp2x[i]= funcionp2x(x[i]);
273       x[i+1]= xnueva(x[i],fx[i],fpx[i],fp2x[i]);
274       mtkhz[i]=errorhz(x[i],x[i+1]);
275       vafx[i]=vafxreal2(fx[i]);
276
277       printf("%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
278       fprintf(salida,"%d\t%f\t%f\t%f\t%f\n", nitera, x[i], fx[i], mtkhz[i], vafx[i]);
279       j=i;
280       i=i+1;
281       nitera=j+1;
282   }while ((mtkhz[j]>tolhor||vafx[j]>tolver)&&(itera>nitera));
283   printf("El valor de la tercera raiz es aproximadamente: %f\n", x[j]);
284   fprintf(salida,"El valor de la tercera raiz es aproximadamente: %f\n", x[j]);
285       return 0; }
286
287
```

AE2.CD1.I1

salida

Metodo de Regula falsi
Primera raiz:

#itera	xzurda	xreal	xdiestra	fxreal	Ehz	|fxreal|
1	0.100000	0.258441	0.400000	0.008059	54.774364	0.008059
2	0.100000	0.262405	0.258441	-0.001455	1.510792	0.001455
3	0.100000	0.261692	0.262405	0.000255	0.272363	0.000255
4	0.100000	0.261818	0.261692	-0.000045	0.047808	0.000045
5	0.100000	0.261795	0.261818	0.000008	0.008435	0.000008
6	0.100000	0.261799	0.261795	-0.000001	0.001491	0.000001
7	0.100000	0.261799	0.261799	0.000000	0.000262	0.000000
8	0.100000	0.261799	0.261799	-0.000000	0.000046	0.000000
9	0.100000	0.261799	0.261799	0.000000	0.000011	0.000000
10	0.100000	0.261799	0.261799	0.000000	0.000000	0.000000

El valor de la primera raiz es aproximadamente: 0.261799
Calculos para la segunda raiz:

#itera	xzurda	xreal	xdiestra	fxreal	Ehz	|fxreal|
1	0.600000	0.775668	0.900000	-0.023338	16.029034	0.023338
2	0.600000	0.787893	0.775668	0.005988	1.551566	0.005988
3	0.600000	0.784808	0.787893	-0.001416	0.393092	0.001416
4	0.600000	0.785540	0.784808	0.000342	0.093238	0.000342
5	0.600000	0.785363	0.785540	-0.000082	0.022495	0.000082
6	0.600000	0.785406	0.785363	0.000020	0.005411	0.000020
7	0.600000	0.785396	0.785406	-0.000005	0.001305	0.000005
8	0.600000	0.785398	0.785396	0.000001	0.000311	0.000001
9	0.600000	0.785397	0.785398	-0.000000	0.000068	0.000000
10	0.600000	0.785398	0.785397	0.000000	0.000015	0.000000
11	0.600000	0.785398	0.785398	-0.000000	0.000008	0.000000

El valor de la segunda raiz es aproximadamente: 0.785398
Calculos para la tercera raiz:

#itera	xzurda	xreal	xdiestra	fxreal	Ehz	|fxreal|
1	1.100000	1.304201	1.500000	0.011507	15.012925	0.011507
2	1.100000	1.310576	1.304201	-0.003791	0.486415	0.003791
3	1.100000	1.308496	1.310576	0.001200	0.158940	0.001200
4	1.100000	1.309157	1.308496	-0.000385	0.050437	0.000385
5	1.100000	1.308945	1.309157	0.000123	0.016156	0.000123
6	1.100000	1.309013	1.308945	-0.000039	0.005154	0.000039
7	1.100000	1.308991	1.309013	0.000013	0.001648	0.000013
8	1.100000	1.308998	1.308991	-0.000004	0.000528	0.000004
9	1.100000	1.308996	1.308998	0.000001	0.000164	0.000001
10	1.100000	1.308996	1.308996	-0.000000	0.000046	0.000000
11	1.100000	1.308996	1.308996	0.000000	0.000009	0.000000

El valor de la tercera raiz es aproximadamente: 1.308996

Metodo de Newton Rampson de segundo orden
Calculo de la primera raiz:

#itera	x	fx	Ehz	|fxreal|
0	0.100000	0.330133	45.131050	0.330133
1	0.182252	0.183745	22.729900	0.183745
2	0.235864	0.061992	9.394031	0.061992
3	0.260319	0.003553	0.565303	0.003553
4	0.261798	0.000001	0.000114	0.000001
5	0.261799	0.000000	0.000000	0.000000

El valor de la primera raiz es aproximadamente: 0.261799
Calculo de la segunda raiz:

#itera	x	fx	Ehz	|fxreal|
0	0.600000	-0.358703	14.511898	0.358703
1	0.701852	-0.192217	7.248411	0.192217
2	0.756701	-0.068532	3.411225	0.068532
3	0.783425	-0.004733	0.251033	0.004733
4	0.785397	-0.000002	0.000083	0.000002
5	0.785398	-0.000000	0.000000	0.000000

El valor de la segunda raiz es aproximadamente: 0.785398
Calculo de la tercera raiz:

#itera	x	fx	Ehz	|fxreal|
0	1.100000	0.380093	11.375250	0.380093
1	1.241188	0.158287	3.834709	0.158287
2	1.290682	0.043866	1.358295	0.043866
3	1.308455	0.001299	0.041364	0.001299
4	1.308996	0.000000	0.000000	0.000000

El valor de la tercera raiz es aproximadamente: 1.308996