
FACULTAD DE CIENCIAS

Tarea 4

ANÁLISIS NÚMÉRICO

Oscar Andrés Rosas Hernandez

Alarcón Alvarez Aylin

Laurrabaquio Rodríguez Miguel Salvador

Pahua Castro Jesús Miguel Ángel

Noviembre 2018

Índice

1. Problemas de Computadora	2
1.1. Notas	3
1.2. 1	4
1.3. 2	5
1.4. 4	6
1.5. 9	6
1.6. 13, 14, 15	7
1.7. 16	11
1.8. 18	12
1.8.1. Usar Newton para calcular la $\sqrt[n]{A}$	12
1.8.2. ¿Como calcular raíces de números negativos?	12
1.9. 19	14
1.10. 20	15
1.11. 21	16
1.12. 22	18
2. Anexo	19
2.1. Bisection	19
2.2. FixedPoint	20
2.3. Helpers	20
2.4. NewtonRaphson	21
2.5. NewtonSecant	22
2.6. RegulaFalsi	23

1. Problemas de Computadora

Una nota importante es que al inicio de CADA script se incluyen los algoritmos, porfavor cambia el valor de la variable `Directory` para que sea un string que apunte desde donde estas a donde estan la carpeta de los algoritmos

Esta linea:

```
1 getd(pwd() + Directory);
```

Para ejecutar cada uno basta con hacer algo como:

```
1 exec("/Users/mac/Documents/Projects/Learning/UNAM/NumericalAnalysis/Homework4/Code/1.sce", -1)
```

1.1. Notas

Sobre la tolerancia:

Hay que definir que es la tolerancia porque puede significar varias cosas:

- Que $|f(x)| < tolerance$
- Que $|x_{k+1} - x_k| < tolerance$

Así que cuando tu me mandas una tolerancia en cualquiera de los métodos la solución que recibes de regrese cumple con **alguna** de las condiciones que he dicho antes.

1.2. 1

Ejecuta los scripts que esta dentro de Code llamado: 1.sce

En este código muestra justo lo que se nos pide, por ejemplo una entrada válida sería:

1. 1
2. " $x * 3 - 10$ "
3. $[0, 3]$
4. 0.001
5. 30

Después de esto tienes acceso a una variable llamada *estimation* por si quieres probar algo.

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 disp("=====");
10 disp("===== ROOTS =====");
11 disp("=====");
12
13 disp("Select a method:");
14 disp("1) Bisection");
15 disp("2) Secant");
16 disp("3) Newton Rapshon");
17 disp("4) Regula Falsi");
18
19 number = input("Method?: ");
20 someFunction = input("Function as string f(x) = : ");
21 initialPoints = input("Initial point(s) (as vector): ");
22 tolerance = input("Tolerance: ");
23 MaxIterations = input("Max Iterations: ");
24
25 a = initialPoints(1)
26 try
27     b = initialPoints(2)
28 catch
29     if (number ~= 3) disp("Not enough initial points") end
30 end
31
32 deff('y = f(x)', ['y = evstr(someFunction)']);
33
34 select number
35     case 1 then
36         if (f(a) * f(b) >= 0) then
37             disp("No valid point :(");
38             break;
39         end
40         [estimation, iterations] = Bisection(a, b, f, tolerance, MaxIterations)
41
42     case 2 then
43         [estimation, iterations] = Secant(a, b, f, tolerance, MaxIterations)
44
45     case 3 then
46         [estimation, iterations] = NewtonRaphson(a, f, tolerance, MaxIterations)
47
48     case 4 then
49         [estimation, iterations] = RegulaFalsi(a, b, f, tolerance, MaxIterations)
50 end
51
52
53
54 disp("estimation: " + string(estimation))
55 disp("f(estimation): " + string(f(estimation)))
56 disp("Iterations required: " + string(iterations))

```

1.3. 2

Ejecuta los scripts que esta dentro de Code llamado: 2.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurabaquio Rodríguez Miguel Salvador
4 // @Author: Pahuá Castro Jesús Miguel Ángel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-5
10 MaxIterations = 100
11
12 function [x] = fa(x)
13     x = x - 2^(-x)
14 endfunction
15
16 function [x] = fb(x)
17     x = exp(x) - x^2 + 3*x - 2
18 endfunction
19
20 function [x] = fc(x)
21     x = 2*x * cos(2*x) - (x+1)^2
22 endfunction
23
24 function [x] = fd(x)
25     x = x * cos(x) - 2*x*x + 3*x - 1
26 endfunction
27
28
29 estimation_fa = Bisection(0, 1, fa, tolerance, MaxIterations)
30 disp("a) f(x) = x - 2^-x")
31 disp("estimation_fa = " + string(estimation_fa))
32 disp("fa(estimation_fa) = " + string(fa(estimation_fa)))
33
34 estimation_fb = Bisection(0, 1, fb, tolerance, MaxIterations)
35 disp("b) f(x) = exp(x) - x^2 + 3*x - 2")
36 disp("estimation_fb = " + string(estimation_fb))
37 disp("fb(estimation_fb) = " + string(fb(estimation_fb)))
38
39 estimation_fc = Bisection(-3, -2, fc, tolerance, MaxIterations)
40 disp("c) f(x) = 2*x * cos(2*x) - (x+1)^2")
41 disp("estimation_fc = " + string(estimation_fc))
42 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
43
44 estimation_fc = Bisection(-1, 0, fc, tolerance, MaxIterations)
45 disp("estimation_fc = " + string(estimation_fc))
46 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
47
48 estimation_fd = Bisection(0.2, 0.3, fd, tolerance, MaxIterations)
49 disp("estimation_fd = " + string(estimation_fd))
50 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
51
52 estimation_fd = Bisection(1.2, 1.3, fd, tolerance, MaxIterations)
53 disp("estimation_fd = " + string(estimation_fd))
54 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

```

1.4. 4

Ejecuta los scripts que esta dentro de Code llamado: 4.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-4
10 MaxIterations = 100
11
12 function [x] = f(x)
13     x = x**2 - 3
14 endfunction
15
16 estimation = Bisection(0, 10, f, tolerance, MaxIterations)
17 disp("a) f(x) = x^2 - 3")
18 disp("estimation = " + string(estimation))
19 disp("f(estimation) = " + string(f(estimation)))

```

1.5. 9

Ejecuta los scripts que esta dentro de Code llamado: 9.sce

En este código muestra justo lo que se nos pide y pues nos muestra como es que se comporta de función $g(x)$ comparandola con la identidad y el punto en donde se intersectan calculado usando las iteraciones de punto fijo

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-2
10 MaxIterations = 100
11
12 function [x] = g(x)
13     x = %pi + 0.5 * sin(x / 2)
14 endfunction
15
16 estimation = FixedPoint(2, g, tolerance, MaxIterations)
17 disp("a) g(x) = pi + 0.5 * sin(x / 2)")
18 disp("estimation = " + string(estimation))
19 disp("g(estimation) = " + string(g(estimation)))
20
21 x = linspace(2, 2* 3.1416, 100)
22 plot(x, g(x), '-')
23 plot(estimation, g(estimation), 'r*')
24 plot(x, x, 'g-')
25
26 hl=legend(['function'; 'fixed point'; 'Identity']);
27 xtitle("Fixed Point", "x-axis", "y-axis")

```

1.6. 13, 14, 15

Ejecuta los scripts que esta dentro de Code llamado: 13.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurabaquio Rodríguez Miguel Salvador
4 // @Author: Pádua Castro Jesús Miguel Ángel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-5
10 MaxIterations = 100
11
12 function [x] = fa(x)
13     x = exp(x) + 2^(-x) + 2*cos(x) - 6
14 endfunction
15
16 function [x] = fb(x)
17     x = log(x - 1) + cos(x - 1)
18 endfunction
19
20 function [x] = fc(x)
21     x = 2*x*cos(2*x) - (x - 2)^2
22 endfunction
23
24 function [x] = fd(x)
25     x = (x - 2)^2 - log(x)
26 endfunction
27
28 function [x] = fe(x)
29     x = exp(x) - 3 * x^2
30 endfunction
31
32 function [x] = ff(x)
33     x = sin(x) - exp(-x)
34 endfunction
35
36
37 estimation_fa = NewtonRaphson( (1 + 2) / 2, fa, tolerance, MaxIterations)
38 disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
39 disp("estimation_fa = " + string(estimation_fa))
40 disp("fa(estimation_fa) = " + string(fa(estimation_fa)))
41
42 estimation_fb = NewtonRaphson( (1.3 + 2) / 2, fb, tolerance, MaxIterations)
43 disp("b) f(x) = log(x - 1) + cos(x - 1)")
44 disp("estimation_fb = " + string(estimation_fb))
45 disp("fb(estimation_fb) = " + string(fb(estimation_fb)))
46
47 estimation_fc = NewtonRaphson( (2 + 3) / 2, fc, tolerance, MaxIterations)
48 disp("c) f(x) = 2*x*cos(2x) - (x - 2)^2 ")
49 disp("estimation_fc = " + string(estimation_fc))
50 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
51
52 estimation_fc = NewtonRaphson( (3 + 4) / 2, fc, tolerance, MaxIterations)
53 disp("estimation_fc = " + string(estimation_fc))
54 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
55
56 estimation_fd = NewtonRaphson( (1 + 2) / 2, fd, tolerance, MaxIterations)
57 disp("d) f(x) = (x - 2)^2 - log(x)")
58 disp("estimation_fd = " + string(estimation_fd))
59 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
60
61 estimation_fd = NewtonRaphson( (%e + 4) / 2, fd, tolerance, MaxIterations)
62 disp("estimation_fd = " + string(estimation_fd))
63 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
64
65 estimation_fe = NewtonRaphson( (0 + 1) / 2, fe, tolerance, MaxIterations)
66 disp("e) f(x) = exp(x) - 3 * x^2")
67 disp("estimation_fe = " + string(estimation_fe))
68 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
69
70 estimation_fe = NewtonRaphson( (3 + 5) / 2, fe, tolerance, MaxIterations)
71 disp("estimation_fe = " + string(estimation_fe))
72 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
73
74 estimation_ff = NewtonRaphson( ( 0 + 1 ) / 2, ff, tolerance, MaxIterations)
75 disp("f) f(x) = sin(x) - exp(-x)")
76 disp("estimation_ff = " + string(estimation_ff))
77 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
78
79 estimation_ff = NewtonRaphson( ( 3 + 4 ) / 2, ff, tolerance, MaxIterations)

```



```

80 disp("estimation_ff = " + string(estimation_ff))
81 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
82
83 estimation_ff = NewtonRaphson( ( 6 + 7 ) / 2 , ff , tolerance , MaxIterations)
84 disp("estimation_ff = " + string(estimation_ff))
85 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahlua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-5
10 MaxIterations = 100
11
12 function [x] = fa(x)
13     x = exp(x) + 2^(-x) + 2*cos(x) - 6
14 endfunction
15
16 function [x] = fb(x)
17     x = log(x - 1) + cos(x - 1)
18 endfunction
19
20 function [x] = fc(x)
21     x = 2*x*cos(2*x) - (x - 2)^2
22 endfunction
23
24 function [x] = fd(x)
25     x = (x - 2)^2 - log(x)
26 endfunction
27
28 function [x] = fe(x)
29     x = exp(x) - 3 * x^2
30 endfunction
31
32 function [x] = ff(x)
33     x = sin(x) - exp(-x)
34 endfunction
35
36
37 estimation_fa = Secant( 1, 2, fa, tolerance, MaxIterations)
38 disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
39 disp("estimation_fa = " + string(estimation_fa))
40 disp("fa(estimation_fa) = " + string(fa(estimation_fa)))
41
42 estimation_fb = Secant( 1.3, 2, fb, tolerance, MaxIterations)
43 disp("b) f(x) = log(x - 1) + cos(x - 1)")
44 disp("estimation_fb = " + string(estimation_fb))
45 disp("fb(estimation_fb) = " + string(fb(estimation_fb)))
46
47 estimation_fc = Secant( 2, 3, fc, tolerance, MaxIterations)
48 disp("c) f(x) = 2*x*cos(2x) - (x - 2)^2 ")
49 disp("estimation_fc = " + string(estimation_fc))
50 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
51
52 estimation_fc = Secant( 3, 4, fc, tolerance, MaxIterations)
53 disp("estimation_fc = " + string(estimation_fc))
54 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
55
56 estimation_fd = Secant( 1, 2, fd, tolerance, MaxIterations)
57 disp("d) f(x) = (x - 2)^2 - log(x)")
58 disp("estimation_fd = " + string(estimation_fd))
59 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
60
61 estimation_fd = Secant( %e, 4, fd, tolerance, MaxIterations)
62 disp("estimation_fd = " + string(estimation_fd))
63 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
64
65 estimation_fe = Secant( 0, 1, fe, tolerance, MaxIterations)
66 disp("e) f(x) = exp(x) - 3 * x^2")
67 disp("estimation_fe = " + string(estimation_fe))
68 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
69
70 estimation_fe = Secant( 3, 5, fe, tolerance, MaxIterations)
71 disp("estimation_fe = " + string(estimation_fe))
72 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
73
74 estimation_ff = Secant( 0, 1, ff, tolerance, MaxIterations)
75 disp("f) f(x) = sin(x) - exp(-x)")
76 disp("estimation_ff = " + string(estimation_ff))
77 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
78
79 estimation_ff = Secant( 3, 4, ff, tolerance, MaxIterations)

```

```

80 disp("estimation_ff = " + string(estimation_ff))
81 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
82
83 estimation_ff = Secant( 6, 7, ff, tolerance, MaxIterations)
84 disp("estimation_ff = " + string(estimation_ff))
85 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahlua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-5
10 MaxIterations = 100
11
12 function [x] = fa(x)
13     x = exp(x) + 2^(-x) + 2*cos(x) - 6
14 endfunction
15
16 function [x] = fb(x)
17     x = log(x - 1) + cos(x - 1)
18 endfunction
19
20 function [x] = fc(x)
21     x = 2*x*cos(2*x) - (x - 2)^2
22 endfunction
23
24 function [x] = fd(x)
25     x = (x - 2)^2 - log(x)
26 endfunction
27
28 function [x] = fe(x)
29     x = exp(x) - 3 * x^2
30 endfunction
31
32 function [x] = ff(x)
33     x = sin(x) - exp(-x)
34 endfunction
35
36
37 estimation_fa = Secant( 1, 2, fa, tolerance, MaxIterations)
38 disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
39 disp("estimation_fa = " + string(estimation_fa))
40 disp("fa(estimation_fa) = " + string(fa(estimation_fa)))
41
42 estimation_fb = Secant( 1.3, 2, fb, tolerance, MaxIterations)
43 disp("b) f(x) = log(x - 1) + cos(x - 1)")
44 disp("estimation_fb = " + string(estimation_fb))
45 disp("fb(estimation_fb) = " + string(fb(estimation_fb)))
46
47 estimation_fc = Secant( 2, 3, fc, tolerance, MaxIterations)
48 disp("c) f(x) = 2*x*cos(2*x) - (x - 2)^2 ")
49 disp("estimation_fc = " + string(estimation_fc))
50 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
51
52 estimation_fc = Secant( 3, 4, fc, tolerance, MaxIterations)
53 disp("estimation_fc = " + string(estimation_fc))
54 disp("fc(estimation_fc) = " + string(fc(estimation_fc)))
55
56 estimation_fd = Secant( 1, 2, fd, tolerance, MaxIterations)
57 disp("d) f(x) = (x - 2)^2 - log(x)")
58 disp("estimation_fd = " + string(estimation_fd))
59 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
60
61 estimation_fd = Secant( %e, 4, fd, tolerance, MaxIterations)
62 disp("estimation_fd = " + string(estimation_fd))
63 disp("fd(estimation_fd) = " + string(fd(estimation_fd)))
64
65 estimation_fe = Secant( 0, 1, fe, tolerance, MaxIterations)
66 disp("e) f(x) = exp(x) - 3 * x^2")
67 disp("estimation_fe = " + string(estimation_fe))
68 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
69
70 estimation_fe = Secant( 3, 5, fe, tolerance, MaxIterations)
71 disp("estimation_fe = " + string(estimation_fe))
72 disp("fe(estimation_fe) = " + string(fe(estimation_fe)))
73
74 estimation_ff = Secant( 0, 1, ff, tolerance, MaxIterations)
75 disp("f) f(x) = sin(x) - exp(-x)")
76 disp("estimation_ff = " + string(estimation_ff))
77 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
78
79 estimation_ff = Secant( 3, 4, ff, tolerance, MaxIterations)

```

```
80 disp("estimation_ff = " + string(estimation_ff))
81 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
82
83 estimation_ff = Secant( 6, 7, ff, tolerance, MaxIterations)
84 disp("estimation_ff = " + string(estimation_ff))
85 disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
```

1.7. 16

Ejecuta los scripts que esta dentro de Code llamado: 9.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```
1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-6
10 MaxIterations = 100
11
12 function [x] = f(x)
13     x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
14 endfunction
15
16 disp('e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9')
17
18 estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
19 disp("estimation Regula Falsi = " + string(estimation))
20 disp("f(estimation) = " + string(f(estimation)))
21
22 estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
23 disp("estimation Regula Falsi = " + string(estimation))
24 disp("f(estimation) = " + string(f(estimation)))
25
26
27 estimation = Secant(-1, 0, f, tolerance, MaxIterations)
28 disp("estimation Secant = " + string(estimation))
29 disp("f(estimation) = " + string(f(estimation)))
30
31 estimation = Secant(0, 1, f, tolerance, MaxIterations)
32 disp("estimation Secant = " + string(estimation))
33 disp("f(estimation) = " + string(f(estimation)))
34
35 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
36 disp("estimation NewtonRaphson = " + string(estimation))
37 disp("f(estimation) = " + string(f(estimation)))
38
39 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
40 disp("estimation NewtonRaphson = " + string(estimation))
41 disp("f(estimation) = " + string(f(estimation)))
```

1.8. 18

Ejecuta los scripts que esta dentro de Code llamado: 18.sce

En este código nos da dos métodos, y ahhhhh, este si me costo un rato, así que añadiré un fragmento del texto que explica como funciona esto:

1.8.1. Usar Newton para calcular la $\sqrt[n]{A}$

Si quieres calcular $\sqrt[n]{A}$ entonces puedes solucionar la ecuación $f(x) = x^n - A = 0$.

En ese caso $f'(x) = nx^{n-1}$ y nuestra iteración es:

$$x_{k+1} = x_k - \frac{(x_k)^n - A}{n(x_k)^{n-1}} = \frac{(n-1)(x_k)^n + A}{n(x_k)^{n-1}}$$

1.8.2. ¿Como calcular raíces de números negativos?

Ok, si quieres usar Newton para calcular raices de números positivos no hay problema.

Ahora, si es negativo pero la n-ava raíz es impar entonces tampoco hay problema basta con calcular la raíz absoluta y luego negar la respuesta, pero, pero que pasa si tratamos con un número negativo y una raíz n-ava par.

En ese caso lo que podemos calcular es con este método es r del número complejo $re^{i\theta}$

Ahora, solo usamos nuestro formulazo para sacar θ y pasarlo a forma rectangular y listo :v

USAR EL METODO ROOT

```

1 // Function to aproximate a \sqrt[n]{A} using the Newton Raphson method
2 // @param: x_0 a initial guess
3 // @param: someFunction a string that represent the expression to get x
4 // @param: tolerance a number to set how exact you want a root
5 // @param: MaximumIterations a number of maximum iterations
6 // @return: estimation
7
8 // @Author: Rosas Hernandez Oscar Andres
9 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
10 // @Author: Laurrabaquio Rodríguez Miguel Salvador
11 // @Author: Pádua Castro Jesús Miguel Angel
12
13 function [estimation] = NRootNewtonRaphson(A, n, tolerance, MaximumIterations)
14     iterations = 0;
15     estimation = A;
16
17     while ((abs(estimation**n - A) > tolerance) && (iterations < MaximumIterations))
18         oldEstimation = estimation;
19         estimation = ((n-1) * estimation**n + A) / (n * estimation**(n-1));
20
21         if (RelativeDifference(oldEstimation, estimation) < tolerance)
22             break;
23         end
24
25         iterations = iterations + 1;
26     end
27 endfunction
28
29
30 function [estimation] = Roots(A, n)
31     if (A == 0) then estimation = 0
32     elseif (A > 0)
33         estimation = NRootNewtonRaphson(A, n, 10e-7, 50)
34     else
35         estimation = -NRootNewtonRaphson(A * -1, n, 10e-7, 50)
36         if (modulo(n, 2) == 0) then
37             theta = %pi / n
38             real = estimation * cos(theta)
39             imaginary = estimation * sin(theta) * %i
40             estimation = real + imaginary
41         end
42     end
43 endfunction

```

1.9. 19

Ejecuta los scripts que esta dentro de Code llamado: 19.sce

En este código muestra justo lo que se nos pide y pues...ya :v

- La primera opción queda como anillo al dedo porque cumple con todas las restricciones así que no hay mucho que comentar
- Pero la segunda, se toma casi el doble de iteraciones y no llega a un punto válido según las restricciones, pero eso se debe basicamente a la periodicidad de las trigonometricas

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurabaquio Rodriguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-6
10 MaxIterations = 100
11
12 function [x] = f(x)
13     x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
14 endfunction
15
16 disp('e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9')
17
18 estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
19 disp('estimation Regula Falsi = ' + string(estimation))
20 disp('f(estimation) = ' + string(f(estimation)))
21
22 estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
23 disp('estimation Regula Falsi = ' + string(estimation))
24 disp('f(estimation) = ' + string(f(estimation)))
25
26
27 estimation = Secant(-1, 0, f, tolerance, MaxIterations)
28 disp('estimation Secant = ' + string(estimation))
29 disp('f(estimation) = ' + string(f(estimation)))
30
31 estimation = Secant(0, 1, f, tolerance, MaxIterations)
32 disp('estimation Secant = ' + string(estimation))
33 disp('f(estimation) = ' + string(f(estimation)))
34
35 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
36 disp('estimation NewtonRaphson = ' + string(estimation))
37 disp('f(estimation) = ' + string(f(estimation)))
38
39 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
40 disp('estimation NewtonRaphson = ' + string(estimation))
41 disp('f(estimation) = ' + string(f(estimation)))

```

1.10. 20

Ejecuta los scripts que esta dentro de Code llamado: 20.sce

En este código muestra justo lo que se nos pide y pues...ya :v

- La primera opción nos muestra una respuesta, una buena respuesta
- La segunda casi casi niega el primer término de la primera pero la segunda componente se mueve π
- Finalmente la tercera se comporta igual se mueve y salta 2π causado por la periodicidad de las trigonometricas

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahuá Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-6
10 MaxIterations = 100
11
12 function [x] = f(x)
13     x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
14 endfunction
15
16 disp("e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9")
17
18 estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
19 disp("estimation Regula Falsi = " + string(estimation))
20 disp("f(estimation) = " + string(f(estimation)))
21
22 estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
23 disp("estimation Regula Falsi = " + string(estimation))
24 disp("f(estimation) = " + string(f(estimation)))
25
26
27 estimation = Secant(-1, 0, f, tolerance, MaxIterations)
28 disp("estimation Secant = " + string(estimation))
29 disp("f(estimation) = " + string(f(estimation)))
30
31 estimation = Secant(0, 1, f, tolerance, MaxIterations)
32 disp("estimation Secant = " + string(estimation))
33 disp("f(estimation) = " + string(f(estimation)))
34
35 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
36 disp("estimation NewtonRaphson = " + string(estimation))
37 disp("f(estimation) = " + string(f(estimation)))
38
39 estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
40 disp("estimation NewtonRaphson = " + string(estimation))
41 disp("f(estimation) = " + string(f(estimation)))

```


1.11. 21

Ejecuta los scripts que esta dentro de Code llamado: 21a / b / c.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-7
10 MaxIterations = 40
11
12 function [x] = f(x)
13     x = [
14         (x(1) + x(2)*(x(2)*(5 - x(2)) - 2) - 13);
15         (x(1) + x(2)*(x(2)*(1 + x(2)) + 14) - 29);
16     ]
17 endfunction
18
19 estimation = [
20     15;
21     -2;
22 ]
23
24 [estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)
25 disp("estimation = " + string(estimation))
26 disp("f(estimation) = " + string(f(estimation)))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-7
10 MaxIterations = 40
11
12 function [x] = f(x)
13     x1 = x(1)
14     x2 = x(2)
15     x3 = x(3)
16     x = [
17         (x1**2 + x2**2 + x3**2 - 5);
18         (x1 + x2 - 1);
19         (x1 + x3 - 3);
20     ]
21 endfunction
22
23 estimation = [
24     (1 + sqrt(3)) / 2;
25     (1 - sqrt(3)) / 2;
26     (sqrt(3));
27 ]
28
29 [estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)
30 disp("estimation = " + string(estimation))
31 disp("f(estimation) = " + string(f(estimation)))

```

```

1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Angel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 10^-5
10 MaxIterations = 100
11
12 function [x] = f(x)
13     x1 = x(1)
14     x2 = x(2)
15     x3 = x(3)
16     x4 = x(4)

```

```
17     x = [  
18         (x1 + 10*x2);  
19         sqrt(5) * (x3 - x4);  
20         (x2 - x3)**2;  
21         sqrt(10) * (x1 - x4)**2;  
22     ]  
23 endfunction  
24  
25 estimation = [1;2;1;1]  
26  
27 [estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)  
28 disp("estimation = " + string(estimation))  
29 disp("f(estimation) = " + string(f(estimation)))  
30  
31 disp("Error, because the LU found a zero in a pivot :v")
```

1.12. 22

Ejecuta los scripts que esta dentro de Code llamado: 22.sce

En este código muestra justo lo que se nos pide y pues...ya :v

```
1 // @Author: Rosas Hernandez Oscar Andres
2 // @Author: Alarcón Alvarez Aylin Yadira Guadalupe
3 // @Author: Laurrabaquio Rodríguez Miguel Salvador
4 // @Author: Pahua Castro Jesús Miguel Ángel
5
6 getd(pwd() + Directory);
7 clc;
8
9 tolerance = 0.001;
10 MaxIterations = 40
11
12 function [x] = f(x)
13     x1 = x(1)
14     x2 = x(2)
15     x3 = x(3)
16     x = [
17         (5 - x2**2 - x3**2) / x1;
18         (1 - (1 - x2));
19         (3 - x1);
20     ]
21 endfunction
22
23 function [x] = fReal(x)
24     x1 = x(1)
25     x2 = x(2)
26     x3 = x(3)
27     x = [
28         (x1**2 + x2**2 + x3**2 - 5);
29         (x1 + x2 - 1);
30         (x1 + x3 - 3);
31     ]
32 endfunction
33
34
35 estimation = [
36     2; 1212; 2.1
37 ]
38
39 [estimation, iterations] = FixedPoint(estimation, f, tolerance, MaxIterations)
40 disp("estimation = " + string(estimation))
41 disp("f(estimation) = " + string(fReal(estimation)))
```

2. Anexo

2.1. Bisection

```
1 // /**
2 // * Function to aproximate a root of f(x) using the bisection method
3 // *
4 // * @param a - a number such f(a)f(b) < 0
5 // * @param b - a number such f(a)f(b) < 0
6 // * @param f - a function :v
7 // * @param tolerance - a number to set how exact you want a root
8 // * @param MaxIterations - a number of maximum iterations
9 // * @return estimation - a number such someFunction(estimation) = 0
10 // *
11 // * @author: Rosas Hernandez Oscar Andres
12 // * @author: Alarcón Alvarez Aylin Yadira Guadalupe
13 // * @author: Laurrabaquio Rodríguez Miguel Salvador
14 // * @author: Pádua Castro Jesús Miguel Ángel
15 // */
16 function [estimation, iterations] = Bisection(a, b, f, tolerance, MaxIterations)
17     iterations = 0;
18     estimation = a + (b - a) / 2;
19
20     while (iterations < MaxIterations)
21         [a, b] = BisectionStep(a, b, f);
22
23         if (RelativeDifference(a, b) < tolerance)
24             estimation = a + (b - a) / 2;
25             break;
26         end
27
28         if (abs(f(a)) < tolerance)
29             estimation = a;
30             break;
31         elseif (abs(f(b)) < tolerance)
32             estimation = b;
33             break;
34         end
35
36         iterations = iterations + 1;
37     end
38 endfunction
39
40 function [begin, finish] = BisectionStep(begin, finish, f)
41     middle = begin + (finish - begin) / 2;
42
43     if (SameSign(f(begin), f(middle)))
44         begin = middle;
45     else
46         finish = middle;
47     end
48 endfunction
```

2.2. FixedPoint

```

1  // /**
2  // * Function to aproximate a root of f(x) using the fixed point method , so f(x) can be written as
3  // * f(x) = g(x) - x = 0
4  // *
5  // * @param a - a number such f(a)f(b) < 0
6  // * @param b - a number such f(a)f(b) < 0
7  // * @param f - a function :v
8  // * @param tolerance - a number to set how exact you want a root
9  // * @param MaxIterations - a number of maximum iterations
10 // * @return estimation - a number such someFunction(estimation) = 0
11 // *
12 // * @author: Rosas Hernandez Oscar Andres
13 // * @author: Alarcón Alvarez Aylin Yadira Guadalupe
14 // * @author: Laurrabaquio Rodríguez Miguel Salvador
15 // * @author: Pádua Castro Jesús Miguel Ángel
16 // */
17
18 function [estimation, iterations] = FixedPoint(initialPoint, f, tolerance, MaximumIterations)
19     iterations = 0;
20     estimation = f(initialPoint);
21
22     while ((abs(norm(f(estimation)))) > tolerance) && (iterations < MaximumIterations)
23         disp(estimation)
24         oldEstimation = estimation;
25         estimation = f(estimation);
26
27         if (iterations > 1 && RelativeDifference(oldEstimation, estimation) < tolerance)
28             break;
29         end
30
31         iterations = iterations + 1;
32     end
33 endfunction

```

2.3. Helpers

```

1 function [result] = SameSign(a, b)
2     result = sign(a) == sign(b)
3 endfunction
4
5 function [result] = AbsoluteDifference(a, b)
6     a = norm(a)
7     b = norm(b)
8     result = abs(abs(b) - abs(a))
9 endfunction
10
11 function [result] = RelativeDifference(old, new)
12     old = norm(old)
13     new = norm(new)
14     result = abs(abs(new) - abs(old)) / abs(new)
15 endfunction

```

2.4. NewtonRaphson

```

1 // /**
2 // * Function to aproximate a root of f(x) using the Newton Raphson method
3 // *
4 // * @param: estimation a initial guess to the root
5 // * @param f - a function :v
6 // * @param tolerance - a number to set how exact you want a root
7 // * @param MaxIterations - a number of maximum iterations
8 // * @return estimation - a number such someFunction(estimation) = 0
9 // *
10 // * @author: Rosas Hernandez Oscar Andres
11 // * @author: Alarcón Alvarez Aylin Yadira Guadalupe
12 // * @author: Laurrabaquio Rodríguez Miguel Salvador
13 // * @author: Pahua Castro Jesús Miguel Angel
14 // */
15
16 function [estimation, iterations] = NewtonRaphson(estimation, f, tolerance, MaxIterations)
17     iterations = 0;
18
19     while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaxIterations))
20         oldEstimation = estimation;
21         estimation = NewtonRaphsonStep(estimation, f);
22
23         if (isnan(estimation)) then
24             disp("Wrong initial point")
25             break;
26         elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
27             break
28         end
29
30         iterations = iterations + 1;
31     end
32 endfunction
33
34 function [estimation] = NewtonRaphsonStep(estimation, f)
35     dx = 10^-6;
36     derivative = (f(estimation + dx) - f(estimation)) / dx;
37     estimation = estimation - f(estimation) / derivative;
38 endfunction
39
40
41 function [estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance,
42     MaxIterations)
43     iterations = 0;
44     [n, ~] = size(estimation)
45     [m, ~] = size(f(estimation))
46
47     while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaxIterations))
48         oldEstimation = estimation;
49         estimation = NewtonRaphsonGeneralizedStep(estimation, f, m, n);
50
51         if (isnan(estimation)) then
52             disp("Wrong initial point")
53             break;
54         elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
55             break;
56         end
57
58         iterations = iterations + 1;
59     end
60 endfunction
61
62 function [estimation] = NewtonRaphsonGeneralizedStep(estimation, f, m, n)
63     jacobian = Jacobian(f, estimation);
64     if (m == n)
65         [L, U] = LUdecomposition(jacobian);
66         y = FowardSubstitution(L, -f(estimation));
67         S = BackwardSubstitution(U, y);
68     else
69         S = LeastSquares(jacobian, -f(estimation));
70     end
71     estimation = estimation + S;
72 endfunction

```

2.5. NewtonSecant

```

1 // /**
2 // * Function to aproximate a root of f(x) using the secant method
3 // *
4 // * @param a - a number
5 // * @param b - a number
6 // * @param f - a function :v
7 // * @param tolerance - a number to set how exact you want a root
8 // * @param MaxIterations - a number of maximum iterations
9 // * @return estimation - a number such someFunction(estimation) = 0
10 // *
11 // * @author: Rosas Hernandez Oscar Andres
12 // * @author: Alarcón Alvarez Aylin Yadira Guadalupe
13 // * @author: Laurrabaquio Rodríguez Miguel Salvador
14 // * @author: Pahua Castro Jesús Miguel Angel
15 // */
16 function [estimation, iterations] = Secant(a, b, f, tolerance, MaximumIterations)
17     iterations = 0;
18     estimation = a, oldEstimation = b;
19
20     while (iterations < MaximumIterations)
21         newEstimation = SecantStep(estimation, oldEstimation, f);
22         oldEstimation = estimation, estimation = newEstimation;
23
24         if (abs(f(estimation)) < tolerance)
25             break;
26         elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
27             break;
28         end
29
30         iterations = iterations + 1;
31     end
32 endfunction
33
34 function [newStep] = SecantStep(step, oldStep, f)
35     derivative = (f(step) - f(oldStep)) / (step - oldStep);
36     newStep = step - f(step) / derivative;
37 endfunction

```

2.6. RegulaFalsi

```

1  // /**
2  // * Function to aproximate a root of f(x) using the Regular Falsi method
3  // *
4  // * @param a - a number
5  // * @param b - a number
6  // * @param f - a function :v
7  // * @param tolerance - a number to set how exact you want a root
8  // * @param MaxIterations - a number of maximum iterations
9  // * @return estimation - a number such someFunction(estimation) = 0
10 // *
11 // * @author: Rosas Hernandez Oscar Andres
12 // * @author: Alarcón Alvarez Aylin Yadira Guadalupe
13 // * @author: Laurrabaquio Rodríguez Miguel Salvador
14 // * @author: Pádua Castro Jesús Miguel Angel
15 // */
16 function [estimation, iterations] = RegulaFalsi(a, b, f, tolerance, MaximumIterations)
17     iterations = 0;
18     estimation = a + (b - a) / 2;
19
20     while (iterations < MaximumIterations)
21         c = SecantStep(a, b, f);
22         if (SameSign(f(c), f(a)))
23             a = c;
24         else
25             b = c;
26         end
27
28         if (abs(f(a)) < tolerance)
29             estimation = a;
30             break;
31         elseif (abs(f(b)) < tolerance)
32             estimation = b;
33             break;
34         elseif (RelativeDifference(a, b) < tolerance)
35             estimation = a + (b - a) / 2;
36             break;
37         end
38
39         iterations = iterations + 1;
40     end
41 endfunction
42

```