Facultad de Ciencias

Tarea 4

Análisis Número

Oscar Andrés Rosas Hernandez
Alarcón Alvarez Aylin
Laurrabaquio Rodríguez Miguel Salvador
Pahua Castro Jesús Miguel Ángel

Noviembre 2018

Índice

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:

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

Para ejecutar cada uno basta con hacer algo como:

```
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

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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Angel

getd(pwd() + Directory);
clc;

disp("=================================");
disp("=====      ROOTS          =======");
disp("=================================");

disp("Select a method:");
disp("1) Bisection");
disp("2) Secant");
disp("3) Newton Rapshon");
disp("4) Regula Falsi");

number = input("Method?: ");
someFunction = input("Function as string f(x) = : ");
initialPoints = input("Initial point(s) (as vector): ");
tolerance = input("Tolerance: ");
MaxIterations = input("Max Iterations: ");

a = initialPoints(1)
try
    b = initialPoints(2)
catch
    if (number ~= 3) disp("Not enough initial points") end
end

deff('y = f(x)', ['y = evstr(someFunction)']);

select number
    case 1 then
        if (f(a) * f(b) >= 0) then
            disp("No valid point :(");
            break;
        end
        [estimation, iterations] = Bisection(a, b, f, tolerance, MaxIterations)

    case 2 then
        [estimation, iterations] = Secant(a, b, f, tolerance, MaxIterations)

    case 3 then
        [estimation, iterations] = NewtonRaphson(a, f, tolerance, MaxIterations)

    case 4 then
        [estimation, iterations] = RegulaFalsi(a, b, f, tolerance, MaxIterations)
end



disp("estimation: " + string(estimation))
disp("f(estimation): " + string(f(estimation)))
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-5
MaxIterations = 100

function [x] = fa(x)
    x = x - 2^(-x)
endfunction

function [x] = fb(x)
    x = exp(x) - x^2 + 3*x - 2
endfunction

function [x] = fc(x)
    x = 2*x * cos(2*x) - (x+1)^2
endfunction

function [x] = fd(x)
    x = x * cos(x) - 2*x*x + 3*x - 1
endfunction


estimation_fa = Bisection(0, 1, fa, tolerance, MaxIterations)
disp("a) f(x) = x - 2^-x")
disp("estimation_fa = " + string(estimation_fa))
disp("fa(estimation_fa) = " + string(fa(estimation_fa)))

estimation_fb = Bisection(0, 1, fb, tolerance, MaxIterations)
disp("b) f(x) = exp(x) - x^2 + 3*x - 2")
disp("estimation_fb = " + string(estimation_fb))
disp("fb(estimation_fb) = " + string(fb(estimation_fb)))

estimation_fc = Bisection(-3, -2, fc, tolerance, MaxIterations)
disp("c) f(x) = 2*x * cos(2*x) - (x+1)^2")
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fc = Bisection(-1, 0, fc, tolerance, MaxIterations)
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fd = Bisection(0.2, 0.3, fd, tolerance, MaxIterations)
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fd = Bisection(1.2, 1.3, fd, tolerance, MaxIterations)
disp("estimation_fd = " + string(estimation_fd))
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Angel

getd(pwd() + Directory);
clc;

tolerance = 10^-4
MaxIterations = 100

function [x] = f(x)
    x = x**2 - 3
endfunction

estimation = Bisection(0, 10, f, tolerance, MaxIterations)
disp("a) f(x) = x^2 - 3")
disp("estimation = " + string(estimation))
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Angel

getd(pwd() + Directory);
clc;

tolerance = 10^-2
MaxIterations = 100

function [x] = g(x)
    x = %pi + 0.5 * sin(x / 2)
endfunction

estimation = FixedPoint(2, g, tolerance, MaxIterations)
disp("a) g(x) = pi + 0.5 * sin(x / 2)")
disp("estimation = " + string(estimation))
disp("g(estimation) = " + string(g(estimation)))

x = linspace(2, 2* 3.1416, 100)
plot(x, g(x), '-')
plot(estimation, g(estimation), 'r*')
plot(x, x, 'g-')

hl=legend(['function'; 'fixed point'; 'Identity']);
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-5
MaxIterations = 100

function [x] = fa(x)
    x = exp(x) + 2^(-x) + 2*cos(x) - 6
endfunction

function [x] = fb(x)
    x = log(x - 1) + cos(x - 1)
endfunction

function [x] = fc(x)
    x = 2*x*cos(2*x) - (x - 2)^2
endfunction

function [x] = fd(x)
    x = (x - 2)^2 - log(x)
endfunction

function [x] = fe(x)
    x = exp(x) - 3 * x^2
endfunction

function [x] = ff(x)
    x = sin(x) - exp(-x)
endfunction


estimation_fa = NewtonRaphson( (1 + 2) / 2, fa, tolerance, MaxIterations)
disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
disp("estimation_fa = " + string(estimation_fa))
disp("fa(estimation_fa) = " + string(fa(estimation_fa)))

estimation_fb = NewtonRaphson( (1.3 + 2) / 2, fb, tolerance, MaxIterations)
disp("b) f(x) = log(x - 1) + cos(x - 1)")
disp("estimation_fb = " + string(estimation_fb))
disp("fb(estimation_fb) = " + string(fb(estimation_fb)))

estimation_fc = NewtonRaphson( (2 + 3) / 2, fc, tolerance, MaxIterations)
disp("c) f(x) = 2*x*cos(2x) - (x - 2)^2 ")
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fc = NewtonRaphson( (3 + 4) / 2, fc, tolerance, MaxIterations)
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fd = NewtonRaphson( (1 + 2) / 2 , fd, tolerance, MaxIterations)
disp("d) f(x) = (x - 2)^2 - log(x)")
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fd = NewtonRaphson( (%e + 4) / 2, fd, tolerance, MaxIterations)
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fe = NewtonRaphson( (0 + 1) / 2, fe, tolerance, MaxIterations)
disp("e) f(x) = exp(x) - 3 * x^2")
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_fe = NewtonRaphson( (3 + 5) / 2, fe, tolerance, MaxIterations)
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_ff = NewtonRaphson( ( 0 + 1 ) / 2, ff, tolerance, MaxIterations)
disp("f) f(x) = sin(x) - exp(-x)")
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

estimation_ff = NewtonRaphson( ( 3 + 4 ) / 2 , ff, tolerance, MaxIterations)
```

```
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

estimation_ff = NewtonRaphson( ( 6 + 7 ) / 2 , ff, tolerance, MaxIterations)
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-5
MaxIterations = 100

function [x] = fa(x)
    x = exp(x) + 2^(-x) + 2*cos(x) - 6
endfunction

function [x] = fb(x)
    x = log(x - 1) + cos(x - 1)
endfunction

function [x] = fc(x)
    x = 2*x*cos(2*x) - (x - 2)^2
endfunction

function [x] = fd(x)
    x = (x - 2)^2 - log(x)
endfunction

function [x] = fe(x)
    x = exp(x) - 3 * x^2
endfunction

function [x] = ff(x)
    x = sin(x) - exp(-x)
endfunction


estimation_fa = Secant( 1, 2, fa, tolerance, MaxIterations)
disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
disp("estimation_fa = " + string(estimation_fa))
disp("fa(estimation_fa) = " + string(fa(estimation_fa)))

estimation_fb = Secant( 1.3, 2, fb, tolerance, MaxIterations)
disp("b) f(x) = log(x - 1) + cos(x - 1)")
disp("estimation_fb = " + string(estimation_fb))
disp("fb(estimation_fb) = " + string(fb(estimation_fb)))

estimation_fc = Secant( 2, 3, fc, tolerance, MaxIterations)
disp("c) f(x) = 2*x*cos(2x) - (x - 2)^2 ")
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fc = Secant( 3, 4, fc, tolerance, MaxIterations)
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fd = Secant( 1, 2, fd, tolerance, MaxIterations)
disp("d) f(x) = (x - 2)^2 - log(x)")
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fd = Secant( %e, 4, fd, tolerance, MaxIterations)
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fe = Secant( 0, 1, fe, tolerance, MaxIterations)
disp("e) f(x) = exp(x) - 3 * x^2")
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_fe = Secant( 3, 5, fe, tolerance, MaxIterations)
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_ff = Secant( 0, 1, ff, tolerance, MaxIterations)
disp("f) f(x) = sin(x) - exp(-x)")
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

estimation_ff = Secant( 3, 4, ff, tolerance, MaxIterations)
```

```
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

estimation_ff = Secant( 6, 7, ff, tolerance, MaxIterations)
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-5
MaxIterations = 100

function [x] = fa(x)
    x = exp(x) + 2^(-x) + 2*cos(x) - 6
endfunction

function [x] = fb(x)
    x = log(x - 1) + cos(x - 1)
endfunction

function [x] = fc(x)
    x = 2*x*cos(2*x) - (x - 2)^2
endfunction

function [x] = fd(x)
    x = (x - 2)^2 - log(x)
endfunction

function [x] = fe(x)
    x = exp(x) - 3 * x^2
endfunction

function [x] = ff(x)
    x = sin(x) - exp(-x)
endfunction


estimation_fa = Secant( 1, 2, fa, tolerance, MaxIterations)
disp("a) f(x) = exp(x) + 2^(-x) + 2*cos(x) - 6")
disp("estimation_fa = " + string(estimation_fa))
disp("fa(estimation_fa) = " + string(fa(estimation_fa)))

estimation_fb = Secant( 1.3, 2, fb, tolerance, MaxIterations)
disp("b) f(x) = log(x - 1) + cos(x - 1)")
disp("estimation_fb = " + string(estimation_fb))
disp("fb(estimation_fb) = " + string(fb(estimation_fb)))

estimation_fc = Secant( 2, 3, fc, tolerance, MaxIterations)
disp("c) f(x) = 2*x*cos(2x) - (x - 2)^2 ")
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fc = Secant( 3, 4, fc, tolerance, MaxIterations)
disp("estimation_fc = " + string(estimation_fc))
disp("fc(estimation_fc) = " + string(fc(estimation_fc)))

estimation_fd = Secant( 1, 2, fd, tolerance, MaxIterations)
disp("d) f(x) = (x - 2)^2 - log(x)")
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fd = Secant( %e, 4, fd, tolerance, MaxIterations)
disp("estimation_fd = " + string(estimation_fd))
disp("fd(estimation_fd) = " + string(fd(estimation_fd)))

estimation_fe = Secant( 0, 1, fe, tolerance, MaxIterations)
disp("e) f(x) = exp(x) - 3 * x^2")
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_fe = Secant( 3, 5, fe, tolerance, MaxIterations)
disp("estimation_fe = " + string(estimation_fe))
disp("fe(estimation_fe) = " + string(fe(estimation_fe)))

estimation_ff = Secant( 0, 1, ff, tolerance, MaxIterations)
disp("f) f(x) = sin(x) - exp(-x)")
disp("estimation_ff = " + string(estimation_ff))
disp("ff(estimation_ff) = " + string(ff(estimation_ff)))

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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-6
MaxIterations = 100

function [x] = f(x)
    x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
endfunction

disp("e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9")

estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))


estimation = Secant(-1, 0, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = Secant(0, 1, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
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-abla 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-abla 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

```
// Function to aproximate a \sqrt[n]{A} using the Newton Raphson method
// @param: x_0 a initial guess
// @param: someFunction a string that represent the expression to get x
// @param: tolerance a number to set how exact you want a root
// @param: MaximumIterations a number of maximum iterations
// @return: estimation

// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Angel

function [estimation] = NRootNewtonRaphson(A, n, tolerance, MaximumIterations)
    iterations = 0;
    estimation = A;

    while ((abs(estimation**n - A) > tolerance) && (iterations < MaximumIterations))
        oldEstimation = estimation;
        estimation = ((n-1) * estimation**n + A) / (n * estimation**(n-1));

        if (RelativeDifference(oldEstimation, estimation) < tolerance)
            break;
        end

        iterations = iterations + 1;
    end
endfunction


function [estimation] = Roots(A, n)
    if (A == 0) then estimation = 0
    elseif (A > 0)
        estimation = NRootNewtonRaphson(A, n, 10e-7, 50)
    else
        estimation = -NRootNewtonRaphson(A * -1, n, 10e-7, 50)
        if (modulo(n, 2) == 0) then
            theta = %pi / n
            real = estimation * cos(theta)
            imaginary = estimation * sin(theta) * %i
            estimation = real + imaginary
        end
    end
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-6
MaxIterations = 100

function [x] = f(x)
    x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
endfunction

disp("e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9")

estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))


estimation = Secant(-1, 0, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = Secant(0, 1, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
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 *pi*
- Finalmente la tercera se comporta igual se mueve y salta 2π causado por la periodicidad de las trigonometricas

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-6
MaxIterations = 100

function [x] = f(x)
    x = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9
endfunction

disp("e) f(x) = 230*x^4 + 18*x^3 + 9*x^2 - 221*x - 9")

estimation = RegulaFalsi(-1, 0, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = RegulaFalsi(0, 1, f, tolerance, MaxIterations)
disp("estimation Regula Falsi = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))


estimation = Secant(-1, 0, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = Secant(0, 1, f, tolerance, MaxIterations)
disp("estimation Secant = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

estimation = NewtonRaphson(-0.5, f, tolerance, MaxIterations)
disp("estimation NewtonRaphson = " + string(estimation))
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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-7
MaxIterations = 40

function [x] = f(x)
    x = [
    (x(1) + x(2)*(x(2)*(5 - x(2)) - 2)  - 13);
    (x(1) + x(2)*(x(2)*(1 + x(2)) + 14) - 29);
    ]
endfunction

estimation = [
    15;
    -2;
]

[estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)
disp("estimation = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-7
MaxIterations = 40

function [x] = f(x)
    x1 = x(1)
    x2 = x(2)
    x3 = x(3)
    x = [
        (x1**2 + x2**2 + x3**2 - 5);
        (x1 + x2 - 1);
        (x1 + x3 - 3);
    ]
endfunction

estimation = [
    (1 + sqrt(3)) / 2;
    (1 - sqrt(3)) / 2;
    (sqrt(3));
]

[estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)
disp("estimation = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))
```

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 10^-5
MaxIterations = 100

function [x] = f(x)
    x1 = x(1)
    x2 = x(2)
    x3 = x(3)
    x4 = x(4)
```

```
    x = [
        (x1 + 10*x2);
        sqrt(5) * (x3 - x4);
        (x2 - x3)**2;
        sqrt(10) * (x1 - x4)**2;
    ]
endfunction

estimation = [1;2;1;1]

[estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance, MaxIterations)
disp("estimation = " + string(estimation))
disp("f(estimation) = " + string(f(estimation)))

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

```
// @Author: Rosas Hernandez Oscar Andres
// @Author: Alarcón Alvarez Aylin Yadira Guadalupe
// @Author: Laurrabaquio Rodríguez Miguel Salvador
// @Author: Pahua Castro Jesús Miguel Ángel

getd(pwd() + Directory);
clc;

tolerance = 0.001;
MaxIterations = 40

function [x] = f(x)
    x1 = x(1)
    x2 = x(2)
    x3 = x(3)
    x = [
        (5 - x2**2 - x3**2) / x1;
        (1 - (1 -x2 ));
        (3 - x1);
    ]
endfunction

function [x] = fReal(x)
    x1 = x(1)
    x2 = x(2)
    x3 = x(3)
    x = [
        (x1**2 + x2**2 + x3**2 - 5);
        (x1 + x2 - 1);
        (x1 + x3 - 3);
    ]
endfunction


estimation = [
    2; 1212; 2.1
]

[estimation, iterations] = FixedPoint(estimation, f, tolerance, MaxIterations)
disp("estimation = " + string(estimation))
disp("f(estimation) = " + string(fReal(estimation)))
```

2. Anexo

2.1. Bisection

```
// /**
//  * Function to aproximate a root of f(x) using the bisection method
//  *
//  * @param a - a number such f(a)f(b) < 0
//  * @param b - a number such f(a)f(b) < 0
//  * @param f - a function :v
//  * @param tolerance - a number to set how exact you want a root
//  * @param MaxIterations - a number of maximum iterations
//  * @return estimation - a number such someFunction(estimation) = 0
//  *
//  * @author: Rosas Hernandez Oscar Andres
//  * @author: Alarcón Alvarez Aylin Yadira Guadalupe
//  * @author: Laurrabaquio Rodríguez Miguel Salvador
//  * @author: Pahua Castro Jesús Miguel Angel
//  */
function [estimation, iterations] = Bisection(a, b, f, tolerance, MaxIterations)
    iterations = 0;
    estimation = a + (b - a) / 2;

    while (iterations < MaxIterations)
        [a, b] = BisectionStep(a, b, f);

        if (RelativeDifference(a, b) < tolerance)
            estimation = a + (b - a) / 2;
            break;
        end

        if (abs(f(a)) < tolerance)
            estimation = a;
            break;
        elseif (abs(f(b)) < tolerance)
            estimation = b;
            break;
        end

        iterations = iterations + 1;
    end
endfunction

function [begin, finish] = BisectionStep(begin, finish, f)
    middle = begin + (finish - begin) / 2;

    if (SameSign(f(begin), f(middle)))
        begin = middle;
    else
        finish = middle;
    end
endfunction
```

2.2. FixedPoint

```
//  /**
//   * Function to aproximate a root of f(x) using the fixed point method , so f(x) can be written as
//   * f(x) = g(x) - x = 0
//   *
//   * @param a - a number such f(a)f(b) < 0
//   * @param b - a number such f(a)f(b) < 0
//   * @param f - a function :v
//   * @param tolerance - a number to set how exact you want a root
//   * @param MaxIterations - a number of maximum iterations
//   * @return estimation - a number such someFunction(estimation) = 0
//   *
//   * @author: Rosas Hernandez Oscar Andres
//   * @author: Alarcón Alvarez Aylin Yadira Guadalupe
//   * @author: Laurrabaquio Rodríguez Miguel Salvador
//   * @author: Pahua Castro Jesús Miguel Ángel
//   */

function [estimation, iterations] = FixedPoint(initialPoint, f, tolerance, MaximumIterations)
    iterations = 0;
    estimation = f(initialPoint);

    while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaximumIterations))
        disp(estimation)
        oldEstimation = estimation;
        estimation = f(estimation);

        if (iterations > 1 && RelativeDifference(oldEstimation, estimation) < tolerance)
            break;
        end

        iterations = iterations + 1;
    end
endfunction
```

2.3. Helpers

```
function [result] = SameSign(a, b)
    result = sign(a) == sign(b)
endfunction

function [result] = AbsoluteDiffence(a, b)
    a = norm(a)
    b = norm(b)
    result = abs(abs(b) - abs(a))
endfunction

function [result] = RelativeDifference(old, new)
    old = norm(old)
    new = norm(new)
    result = abs(abs(new) - abs(old)) / abs(new)
endfunction
```

2.4. NewtonRaphson

```
// /**
//  * Function to aproximate a root of f(x) using the Newton Raphson method
//  *
//  * @param: estimation a initial guess to the root
//  * @param f - a function :v
//  * @param tolerance - a number to set how exact you want a root
//  * @param MaxIterations - a number of maximum iterations
//  * @return estimation - a number such someFunction(estimation) = 0
//  *
//  * @author: Rosas Hernandez Oscar Andres
//  * @author: Alarcón Alvarez Aylin Yadira Guadalupe
//  * @author: Laurrabaquio Rodríguez Miguel Salvador
//  * @author: Pahua Castro Jesús Miguel Angel
//  */

function [estimation, iterations] = NewtonRaphson(estimation, f, tolerance, MaxIterations)
    iterations = 0;

    while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaxIterations))
        oldEstimation = estimation;
        estimation = NewtonRaphsonStep(estimation, f);

        if (isnan(estimation)) then
            disp("Wrong initial point")
            break;
        elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
            break
        end

        iterations = iterations + 1;
    end
endfunction

function [estimation] = NewtonRaphsonStep(estimation, f)
    dx = 10^-6;
    derivative = (f(estimation + dx) - f(estimation)) / dx;
    estimation = estimation - f(estimation) / derivative;
endfunction


function [estimation, iterations] = NewtonRaphsonGeneralized(estimation, f, tolerance,
    MaxIterations)
    iterations = 0;
    [n, _] = size(estimation)
    [m, _] = size(f(estimation))

    while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaxIterations))
        oldEstimation = estimation;
        estimation = NewtonRaphsonGeneralizedStep(estimation, f, m, n);

        if (isnan(estimation)) then
            disp("Wrong initial point")
            break;
        elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
            break;
        end

        iterations = iterations + 1;
    end
endfunction

function [estimation] = NewtonRaphsonGeneralizedStep(estimation, f, m, n)
    jacobian = Jacobian(f, estimation);
    if (m == n)
        [L, U] = LUDecomposition(jacobian);
        y = FowardSubstitution(L, -f(estimation));
        S = BackwardSubstitution(U, y);
    else
        S = LeastSquares(jacobian, -f(estimation));
    end
    estimation = estimation + S;
endfunction
```

2.5. NewtonSecant

```
// /**
//  * Function to aproximate a root of f(x) using the secant method
//  *
//  * @param a - a number
//  * @param b - a number
//  * @param f - a function :v
//  * @param tolerance - a number to set how exact you want a root
//  * @param MaxIterations - a number of maximum iterations
//  * @return estimation - a number such someFunction(estimation) = 0
//  *
//  * @author: Rosas Hernandez Oscar Andres
//  * @author: Alarcón Alvarez Aylin Yadira Guadalupe
//  * @author: Laurrabaquio Rodríguez Miguel Salvador
//  * @author: Pahua Castro Jesús Miguel Angel
//  */
function [estimation, iterations] = Secant(a, b, f, tolerance, MaximumIterations)
    iterations = 0;
    estimation = a, oldEstimation = b;

    while (iterations < MaximumIterations)
        newEstimation = SecantStep(estimation, oldEstimation, f);
        oldEstimation = estimation, estimation = newEstimation;

        if (abs(f(estimation)) < tolerance)
            break;
        elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
            break;
        end

        iterations = iterations + 1;
    end
endfunction

function [newStep] = SecantStep(step, oldStep, f)
    derivative = (f(step) - f(oldStep)) / (step - oldStep);
    newStep = step - f(step) / derivative;
endfunction
```

2.6. RegulaFalsi

```
// /**
//  * Function to aproximate a root of f(x) using the Regular Falsi method
//  *
//  * @param a - a number
//  * @param b - a number
//  * @param f - a function :v
//  * @param tolerance - a number to set how exact you want a root
//  * @param MaxIterations - a number of maximum iterations
//  * @return estimation - a number such someFunction(estimation) = 0
//  *
//  * @author: Rosas Hernandez Oscar Andres
//  * @author: Alarcón Alvarez Aylin Yadira Guadalupe
//  * @author: Laurrabaquio Rodríguez Miguel Salvador
//  * @author: Pahua Castro Jesús Miguel Angel
//  */
function [estimation, iterations] = RegulaFalsi(a, b, f, tolerance, MaximumIterations)
    iterations = 0;
    estimation = a + (b - a) / 2;

    while (iterations < MaximumIterations)
        c = SecantStep(a, b, f);
        if (SameSign(f(c), f(a)))
            a = c;
        else
            b = c;
        end

        if (abs(f(a)) < tolerance)
            estimation = a;
            break;
        elseif (abs(f(b)) < tolerance)
            estimation = b;
            break;
        elseif (RelativeDifference(a, b) < tolerance)
            estimation = a + (b - a) / 2;
            break;
        end

        iterations = iterations + 1;
    end

endfunction
```