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))
```

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

    while ((abs(norm(f(estimation))) > tolerance) && (iterations < MaximumIterations))
        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(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^-7;
    derivative = (f(estimation + dx) - f(estimation)) / dx;
    estimation = estimation - f(estimation) / derivative;
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;

        disp(estimation)
        disp(oldEstimation)

        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;
```

```
36              break;
37          end
38
39          iterations = iterations + 1;
40      end
41
42  endfunction
```