
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
2. Anexo	5
2.1. Bisection	5
2.2. FixedPoint	6
2.3. Helpers	6
2.4. NewtonRaphson	7
2.5. NewtonSecant	8
2.6. RegulaFalsi	8

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))

```

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: Pahuá Castro Jesús Miguel Ángel
16 // */
17
18 function [estimation] = FixedPoint(initialPoint, f, tolerance, MaximumIterations)
19     iterations = 0;
20     estimation = f(initialPoint);
21
22     while ((abs(norm(f(estimation)))) > tolerance) && (iterations < MaximumIterations))
23         oldEstimation = estimation;
24         estimation = f(estimation);
25
26         if (iterations > 1 && RelativeDifference(oldEstimation, estimation) < tolerance)
27             break;
28         end
29
30         iterations = iterations + 1;
31     end
32 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(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^-7;
36     derivative = (f(estimation + dx) - f(estimation)) / dx;
37     estimation = estimation - f(estimation) / derivative;
38 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: Pádua 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         disp(estimation)
25         disp(oldEstimation)
26
27         if (abs(f(estimation)) < tolerance)
28             break;
29         elseif (RelativeDifference(oldEstimation, estimation) < tolerance)
30             break;
31         end
32
33         iterations = iterations + 1;
34     end
35 endfunction
36
37 function [newStep] = SecantStep(step, oldStep, f)
38     derivative = (f(step) - f(oldStep)) / (step - oldStep);
39     newStep = step - f(step) / derivative;
40 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
42 endfunction
```