Assignment 3 Xavier Rodriguez

function [y] = horner2(c, x)

% [y] = horner2(C,X)

%function that grabs a row coefficients c and a vector of values x of any polynomial

%and outputs a vector of values

y = c(1)\*ones(size(x));

for i = 2: length(c)

y = y .\* x + c(i);

end

end

function [yval] = problem\_1(x)

%[yval] = problem\_1(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

c = [1 0 -2 -5];

[yval] = horner2(c,x);

end

function [yval] = problem\_2(x)

%[yval] = problem\_2(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

yval = x - 0.1\*sin(x) - 24.851090;

end

function [yval] = problem\_3(x)

%[yval] = problem\_3(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

yval = sqrt(1-x) + sqrt(1-2\*x^2) + sqrt(1-3\*x^3)-2;

end

function [yval] = problem\_4(x)

%[yval] = problem\_4(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

yval = cos(x)\*cosh(x)-1;

end

function [yval] = problem\_5(x)

%[yval] = problem\_1(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

c = [6 3];

yval = horner2(c,x);

end

function [yval] = problem\_6(x)

%[yval] = problem\_1(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

yval = (x-1)\*(x-2);

end

function [yval] = problem\_7(x)

%[yval] = problem\_1(x)

% Input: x - point to be evaluated at

% Output: yval - value at x

c = [1 0 0 1];

yval = horner2(c,x);

end

function [sol,flag] = regulafalsix(a, b, relerr, maxit, func)

%[sol,flag] = regulafalsix(a, b, relerr, maxit, func)

% Solves f(x) = 0 using Regula Falsi method

%

% Input: scalar a - starting bound of the bracket

% scalar b - ending bound of the bracket

% small positive scalar relerr - desired relative error

% positive integer maxit - maximum no. of iterations permitted

% func: name of the program that defines f(x);

% Note: when running the problem defined by func, you

% must enter @func in the input line.

% The calling sequence for the program func must have the format

%

% Output: scalar sol - solution found

% scalar flag - flag = 0 indicates solution successfully found

% flag = 1 indicates too many iterations; halt

%compute & store yval of starting bounds

fa = func(a);

fb = func(b);

delta = sqrt(eps); %stores machine eps

k = 0;

while (k < maxit)

k = k + 1;

step = (fa / ((fb-fa)/(b-a))); %regulafalsi formula

x = a - step; %calculate my new step

fx = func(x); % evlaulates my new fb

% checks if signs are the same for fa to fx then swap a else swap b

if (sign(fx) == sign(fa))

approx = a; %stores varibles and swaps

a = x; %swap bound a

fa = fx; %update fa

else

approx = fb;

b = x; %swap bound b

fb = fx; %update fb

end

%termination condition checks if the error < relerr

if abs(x - approx) < max([relerr \* abs(x), delta])

sol = x; flag = 0; return %returns ans

end

sol = x; flag = 1; %halt to many iterations

end

function [sol,flag] = secantx(x0, relerr, maxit, func)

%[sol,flag] = secantx(x0, relerr, maxit, func)

% Solves f(x) = 0 using Secant method

%

%Input: scalar x0 - starting point of the iteration

% positive integer maxit - maximum number of iteraions permitted

% func: name of the program that defines f(x);

% Note: when running the problem defined by func, you

% must enter @func in the input line.

% The calling sequence for the program func must have the format

% Output: scalar sol - solution found

% scalar flag - flag = 0 inditcates solution succesfully found

% flag = 1 indicates too many iterations; halt

%compute & store yval of starting point and calculating

delta = sqrt(eps); %stores machine eps

x1 = x0 + sqrt(eps)\*abs(x0); % x + h = x1

fa = func(x0);

for i = 0: maxit

fb = func(x1); %x1 evalulation per iteration

step = (fb \* (x1 -x0)/(fb - fa)); %secant formula

xnew = x1 - step;

if abs(step) < max([relerr \* abs(xnew), delta])%termination condition checks if error < relerr

sol = xnew; flag = 0; return

end

%swaps and stores the varibles

x0 = x1;

x1 = xnew;

fa = fb;

%solution or flag if to many itertations

sol = xnew ; flag = 1;

end

All relevant graphs

1) x3 - 2x - 5

A picture containing chart

Description automatically generated

2)x -0.1sin(x) -24.851090

Chart, line chart

Description automatically generated

3)

Chart

Description automatically generated with medium confidence

Algebraic proof that it is our zero

4)cos(x)cosh(x) - 1

Chart

Description automatically generated

5)6x +3

Chart, line chart

Description automatically generated

6)(x-1)\*(x-2)

Chart, line chart

Description automatically generated

7)x3 + 1

A picture containing line chart

Description automatically generated

Testing Regula-Falsi

1)

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 10, @problem\_1)

solRegula1 =

2.0344

flagRegula1 =

1

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 100, @problem\_1)

solRegula1 =

2.0946

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 1000, @problem\_1)

solRegula1 =

2.0946

flagRegula1 =

0

2)

[solRegula1, flagRegula1] = regulafalsix(20, 30, 0.000000001, 10, @problem\_2)

solRegula1 =

24.8204

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(20, 30, 0.000000001, 100, @problem\_2)

solRegula1 =

24.8204

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 1000, @problem\_2)

solRegula1 =

24.8204

flagRegula1 =

0

3)

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 10, @problem\_3)

solRegula1 =

0.5592 - 0.0154i

flagRegula1 =

1

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 100, @problem\_3)

solRegula1 =

0.5516 - 0.0000i

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(1, 5, 0.000000001, 1000, @problem\_3)

solRegula1 =

0.5516 - 0.0000i

flagRegula1 =

0

4)

[solRegula1, flagRegula1] = regulafalsix(4.5, 10, 0.000000001, 10, @problem\_4)

solRegula1 =

4.7300

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(4.5, 10, 0.000000001, 100, @problem\_4)

solRegula1 =

4.7300

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(4.5, 10, 0.000000001, 1000, @problem\_4)

solRegula1 =

4.7300

flagRegula1 =

0

Independent testing Regula-Falsi

5)

[solRegula1, flagRegula1] = regulafalsix(-1, 1, 0.000000001, 1000, @problem\_5)

solRegula1 =

-0.5000

flagRegula1 =

1

6)

[solRegula1, flagRegula1] = regulafalsix(0, 4, 0.000000001, 1000, @problem\_6)

solRegula1 =

2.0000

flagRegula1 =

0

[solRegula1, flagRegula1] = regulafalsix(1, 3, 0.000000001, 5, @problem\_6)

solRegula1 =

1.0000

flagRegula1 =

0

7)

[solRegula1, flagRegula1] = regulafalsix(-2, 0, 0.000000001, 1000, @problem\_7)

solRegula1 =

-1.0000

flagRegula1 =

0

Testing Secant

1)

[sol,flag] = secantx(1, 0.000001, 10, @problem\_1)

sol =

2.0946

flag =

1

[sol,flag] = secantx(1, 0.000001, 100, @problem\_1)

sol =

2.0946

flag =

0

[sol,flag] = secantx(1, 0.000001, 1000, @problem\_1)

sol =

2.0946

flag =

0

2)

[sol,flag] = secantx(20, 0.000001, 10, @problem\_2)

sol =

24.8204

flag =

0

[sol,flag] = secantx(20, 0.000001, 100, @problem\_2)

sol =

24.8204

flag =

0

[sol,flag] = secantx(20, 0.000001, 1000, @problem\_2)

sol =

24.8204

flag =

0

3)

[sol,flag] = secantx(5, 0.000001, 10, @problem\_3)

sol =

0.5516 - 0.0000i

flag =

0

[sol,flag] = secantx(5, 0.000001, 100, @problem\_3)

sol =

0.5516 - 0.0000i

flag =

0

[sol,flag] = secantx(5, 0.000001, 1000, @problem\_3)

sol =

0.5516 - 0.0000i

flag =

0

4)

[sol,flag] = secantx(5, 0.000001, 10, @problem\_4)

sol =

4.7300

flag =

0

[sol,flag] = secantx(5, 0.000001, 100, @problem\_4)

sol =

4.7300

flag =

0

[sol,flag] = secantx(5, 0.000001, 1000, @problem\_4)

sol =

4.7300

flag =

0

Independent testing Secant

[solSecant1, flagSecant1] = secantx(1, 0.000000001, 10000, @problem\_5)

solSecant1 =

-0.5000

flagSecant1 =

0

[solSecant1, flagSecant1] = secantx(1, 0.000000001, 10000, @problem\_6)

solSecant1 =

1

flagSecant1 =

0

[solSecant1, flagSecant1] = secantx(2, 0.000000001, 10000, @problem\_6)

solSecant1 =

2

flagSecant1 =

0

[solSecant1, flagSecant1] = secantx(1, 0.000000001, 10000, @problem\_7)

solSecant1 =

-1

flagSecant1 =

0

Xavier Rodriguez

Assignment 3 report

Taking the Regula-Falsi method and computing problem 1 which is a polynomial and a cubic function as x3-2x-5. How this polynomial affects Regula-Falsi is it grab the first two points being [a,b] if the two points are close to the root it takes less iterations to find the root. This is proven in my testing where I change the iterations from 10, 100, 1000. The first check of my bounds being [1,5] and my iterations being 10 you can see that it is 2.0344 which not accurate enough but is close to the root. Then if we have the same bounds but have different iterations like 100 or 1000 iterations it finds the exact root. For my problem 2 my function is x-0.1sin(x)-24.851090 this equation is a trigonometric function, but it is very similar to a linear function. This function follows the same idea with the bounds I mentioned earlier where if I am closer to the root then it takes less iterations. Hence why in my testing my bounds were [20,30] but also another note since this function is very similar to a linear function, I can find the root in 1 iteration. For my problem 3 my function was this function was an algebraic function that has complex roots. For my testing you can see that it outputs my imaginary numbers as well as the actual root. Setting the bounds [1,5] and doing 10 iterations it gives me 0.5592-0.0154i which is close but not the exact root we are looking for which is 0.5516. So, the same holds true that with more iterations the more accurate we get if the bounds are not close to the root. For our problem 4 the function is cos(x)cosh(x)-1 this function is hyperbolic. My bounds are [-4.5,-10] and [4.5,10] and within my testing I switched between 10,100,1000. For all the three-testing mentioned I get the correct root which is -4.7300 and 4.7300.

Taking the Secant method and computing problem 1 we only take in our starting point, iterations, and relative error. Starting with a starting point [a] and computing if they are close to the root then it would take less iterations to get the correct answer. Picking my bound to be [1] and within my testing I switched the iterations to 10, 100, and 1000. When my iterations are complete, I get the correct root of 2.0946 for all my testing no matter the iterations. For problem 2 the case seems to be the same, I chose my starting point to be [20] and within my testing I switched the iterations to 10, 100, and 1000. When my iterations are complete, I get the correct root of 24.8204 for all my testing. For problem 3 it is a complex number, my starting point to be [5] and within my testing I switched between 10, 100, and 1000. It gives us one point and one imaginary number of 0.5516 - 0.000i for 10,100,1000 which is the correct root. Lastly for my problem 4 I chose my bounds to be [5] and within my testing I switched between 10, 100 and 1000. For all the three-testing mentioned I get the correct root which is 4.7300. In conclusion, we can see that Regula-Falsi computes the root faster but is less accurate, and secant takes more iterations but is more accurate.