Lab 3

Table of Contents

Problem 1

Plot the graph of f(x) = x2|sin(x)| for x 2 [0; 4]. Use the Secant method to find the smallest positive zero accurate to within 10^-6 using p0 = 3.6 and p1 = 3.7. [What happens if p0 = 2:8 and p1 = 2:9 are used?]

```
fprintf('Problem 1\n')

% Define the function f
f = @(x) (x.^2) .* abs( sin(x) ) - 4;

% Graph f(x)
xVals = 0:0.01:4;                % set xVals so x in [0,4]
yVals = f(xVals);                % find corresponding yVals
plot( xVals, yVals);             % plot f(x)
grid on                          % turn on gridlines on plot

% p0 = 3.6, p1 = 3.7 run
p0 = 3.6;
p1 = 3.7;
fprintf('p0 = %.1f, p1 = %.1f\n', p0, p1)
pVec = [p0, p1];                 % package p0,p1 as vector to pass to
 method
tolerance = 10^(-6);             % set tolerance
stoppingCriteria = 2;            % use reletive error
method = 2;                      % secant method
maxIterations = 150;             % set maxIterations so we don't run
 forever

[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(pVec, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
printResults(numIterations, approxSol, finalError);

% p0 = 2.8, p1 = 2.9 run
```

```
p0 = 2.8;
p1 = 2.9;
fprintf('p0 = %.1f, p1 = %.1f\n', p0, p1)
pVec = [p0, p1];                % package p0,p1 as vector to pass to
 method
tolerance = 10^(-6);            % set tolerance
stoppingCriteria = 2;           % use reletive error
method = 2;                     % secant method
maxIterations = 150;            % set maxIterations so we don't run
 forever

[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(pVec, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
printResults(numIterations, approxSol, finalError);
```

Problem 1
p0 = 3.6, p1 = 3.7

Problem 2

Use Newton's method to approximate the zero of the function f(x) = x^2 - 2exp(-x) + exp(-2x) accurate to within 10^-8 using p0 = 1. [What do you notice about the convergence of Newton's method?] Repeat the problem using the modified Newton's method and compare the number of iterations required for both methods.

```
fprintf('Problem 2\n')
f = @(x) x.^2 - 2*exp(-x) + exp(-2*x);

% Set parameters for run
p0 = 1;                         % set p0
tolerance = 10^(-8);            % set tolerance
stoppingCriteria = 2;           % use reletive error
method = 3;                     % Netwon's method
maxIterations = 150;            % set maxIterations so we don't run
 forever
fprintf('Netwon''s Method\n');
% Run FP using Newton's Method
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
% Print Results
printResults(numIterations, approxSol, finalError);

% Change method to Modified Netwon's method
fprintf('Modified Netwon''s Method\n');
method = 4;                      % Modified Newton's Method
% Run FP using Modified Newton's Method
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
% Print Results
printResults(numIterations, approxSol, finalError);
```

```
% The convergence for Netwon's method was pretty fast; it only
 required 4
% iterations to find the zero. Modified Newton's method also found the
 zero
% in 4 iterations.
```

Problem 2
Netwon's Method

Problem 3

Plot the graph of f(x) = x^3 - 12.42x^2 + 50.444x - 66.552 for x in [4, 6] and use Newton's method to find all zeros accurate to within 10^-8 on this interval. [How are the convergence of the method, the multiplicity of the zeros, and the values of the derivative at the zeros related?]

```
fprintf('Problem 3\n')

% Define the function f
f = @(x) x.^3 - 12.42*(x.^2) + 50.444*x - 66.552;
syms x
dfdx = matlabFunction(diff(f,x));
% Graph f(x)
xVals = 4:0.01:6;                % set xVals so x in [0,4]
yVals = f(xVals);                % find corresponding yVals
plot( xVals, yVals);             % plot f(x)
grid on                          % turn on gridlines on plot

% Set parameters for run. Starting run from the lower bound
p0 = 4;                          % set p0
tolerance = 10^(-8);             % set tolerance
stoppingCriteria = 2;            % use reletive error
method = 3;                      % Netwon's method
maxIterations = 150;             % set maxIterations so we don't run
 forever

% Run FP using Newton's Method
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
% Print Results
printResults(numIterations, approxSol, finalError);
% find value of derivative at zero
fprintf('The value of dfdx(%.8f) = %.8f)\n\n',approxSol,
 dfdx(approxSol))

% Change p0 to 6 to start run from the upper bound
p0 = 6;                          % set p0

% Run FP using Newton's Method
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, method, maxIterations, f);
```

```
% Print Results
printResults(numIterations, approxSol, finalError);
% find value of derivative at zero
fprintf('The value of dfdx(%.8f) = %.8f)\n\n',approxSol,
 dfdx(approxSol))


% I graphed this function in both MATLAB an on Desmos.com. It was
 difficult
% to determine the zero's with the MATLAB graph but the Desmos one
% highlights them and it was easy to see that there would be two on
 this
% interval. According to the Desmos graph, there are 3 zeros for this
% function. Since this is a degree 3 polynomial, that means it can
 have at
% most 3 zeros. Since it has 3 real zeros, that means all zeros have
% muliplicity 1. Since both of the zeros were of multiplicity 1, that
 means
% that neither of them were local min or max, which in turn means that
 dfdx
% evaluted at them was not equal to zero.
```

Problem 3

Problem 4

Use fixed-point iteration to approximate the solution of x = 5^(-x) accurate to within 10^-8 using p0 = 0.5. Repeat the problem using Steffenson's method and compare the number of iterations required for both methods.

```
fprintf('Problem 4\n')

f = @(x) 5^(-x);                % define function
p0 = 0.5;                       % set p0
tolerance = 10^(-8);            % set tolerance
stoppingCriteria = 2;           % use reletive error
method = 1;                     % use fixed-point iteration
maxIterations = 150;            % set maxIterations so we don't run
 forever

% Run FP using Newton's Method
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(...
    p0,tolerance, stoppingCriteria, method, maxIterations, f);
% Print Results
fprintf('Results using fixed-point iteration:\n');
printResults(numIterations, approxSol, finalError);

% Run FP using Steffenson's Method
[foundSol, numIterations, approxSol, finalError] =
 SteffensonsMethod(p0, ...
    tolerance, stoppingCriteria, maxIterations, f);
% Print Results
fprintf('Results using Steffenson''s Method:\n');
```

```
printResults(numIterations, approxSol, finalError);

% Steffon's method found the solution in 3 iterations while fixed
 point
% iteration took 59. This was a huge improvement.
```

Problem 4

local helper functions

```
%+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++
```

absolute error

```
function absError = AbsoluteError(pn, pnMinus1)
% This function finds the absolute error between pn and pnMinus1
%   INPUTS: pn, pnMinus1
%   OUTPUTS: e = |pn - pnMinus1|

absError = abs(pn - pnMinus1);
end
```

relative error

```
function relError = RelativeError(pn, pnMinus1)
% This function finds the relative error between pn and pnMinus1
%   INPUTS: pn, pnMinus1
%   OUTPUTS: e = |pn - pnMinus1| / pn

relError = abs(pn - pnMinus1)/pn;
end

function output = printResults(numIterations, approxSol, finalError)
    output = fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n\n',
 numIterations,...
           numIterations, approxSol, finalError);
end
```

n: 4 p4: 3.4785085158 |error|: 0.0000003488

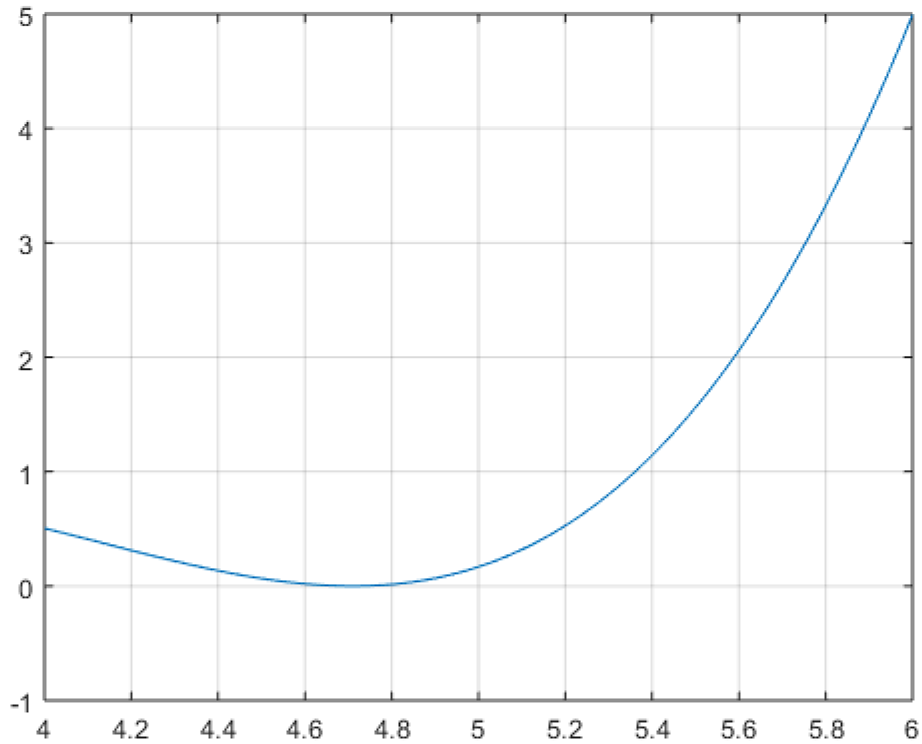
p0 = 2.8, p1 = 2.9

n: 40 p40: 9.4694009967 |error|: 0.0000000349

n: 4 p4: 0.8267700552 |error|: 0.0000000016

Modified Netwon's Method

n: 4 p4: 0.8267700552 |error|: 0.0000000014




```
n: 9   p9: 4.7000000000    |error|: 0.0000000001

The value of dfdx(4.70000000) = -0.03400000)


n: 12   p12: 4.7200000000    |error|: 0.0000000000

The value of dfdx(4.72000000) = 0.03440000)


Results using fixed-point iteration:
n: 59   p59: 0.4696219209    |error|: 0.0000000099
```

Fixed-Point Iteration

```
function [foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(pVec, tolerance, stoppingCriteria, method, ...
    maxIterations, f)
% This function finds an approximate zero to the input function using
%   the secant method.
%   INPUTS: pVec: vecetor containing bases cases
%           tolerance: how close you want to get
%           stoppingCriteria: 1) absolute error, 2) relative error
%           method: 1) Fixed-Point iteration
%                   2) Secant Method
```

```
%                   3) Newton's method [finds f'(x) and uses it to
 create
%                       g(x) = x - f(x)/f'(x) ]
%            max iterations: max number of times method will run before
%                timing out
%            f: function on which you want to run the method
%   OUTPUTS: foundSol - a boolean value of wether or not a solution
 was
%            found within given restrainst, approxSol - approximate
 solution
%            of input function near p0.
foundSol = 0; % we haven't found a solution yet
n = 1; % initialize iteration counter
if method == 1 % Fixed-Point iteration
    pnMinus1 = pVec(1); % set pnMinus to p0
    g = @(x) f(x);
elseif method == 2 % Secant Method
    pnMinus1 = pVec(2);                 % set pnMinus1 to p1
    pnMinus2 = pVec(1);                 % set pnMinus2 to p0
elseif method == 3 || method == 4 % Newton's Method/Modified NM
    pnMinus1 = pVec(1);     % set pnMinus to p0
    syms x
    dfdx = matlabFunction(diff(f,x));
    if method == 3 % Newton's method
        g = @(x) x - (f(x))/(dfdx(x));
    elseif method == 4  % Modified Newton's Methods
        mu = @(x)(f(x))/(dfdx(x));
        dmudx = matlabFunction(diff(mu,x));
        g = @(x) x - (mu(x))/(dmudx(x));
    end
end
while n <= maxIterations
    if method == 1 || method == 3 || method == 4
        pn = g(pnMinus1);
    elseif method == 2
        pn = pnMinus1 - ( (f(pnMinus1) * (pnMinus1 - pnMinus2)) /
 ( f(pnMinus1) - f(pnMinus2)) );
    end
    if stoppingCriteria == 1
        error = AbsoluteError(pn, pnMinus1);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = pn; % appoximate solution
            finalError = error;
            break
        end
    end
    if stoppingCriteria == 2
        error = RelativeError(pn, pnMinus1);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = pn; % appoximate solution
```

```
            finalError = error;
            break
        end
    end
    n = n + 1; % increment iteration counter
    temp = pnMinus1; % set input for next iteration to output from
 this one
    pnMinus1 = pn;
    pnMinus2 = temp;
    %fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n', n, n, xn,
 error);
end
% solution was not found
numIterations = n;
approxSol = pn;
finalError = error;
end
```

Steffenson's Method

```
function [foundSol, numIterations, approxSol, finalError] = ...
    SteffensonsMethod(p0, tolerance, stoppingCriteria, ...
    maxIterations, f)

foundSol = 0;       % set found solution to false
pn = p0;            % set pn
n = 0;              % initialize counter for while loop
while n < maxIterations
    pnPlus1 = f(pn);
    pnPlus2 = f(pnPlus1);
    pHatn = pn - ((pnPlus1 - pn).^2) / (pnPlus2 - 2*pnPlus1 + pn);
    if stoppingCriteria == 1
        error = AbsoluteError(pHatn, pn);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = pHatn; % appoximate solution
            finalError = error;
            break
        end
    elseif stoppingCriteria == 2
        error = RelativeError(pHatn, pn);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = pHatn; % appoximate solution
            finalError = error;
            break
        end
    end
    %fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n', n, n, pHatn,
 error);
    n = n + 1;
```

```
    pn = pHatn;
end
% solution was not found
numIterations = n;
approxSol = pHatn;
finalError = error;
end
```

```
Results using Steffenson's Method:
n: 3  p3: 0.4696219229   |error|: 0.0000000000
```

Published with MATLAB® R2018b