Table of Contents

Problem 1

```
%{
Use the Bisection method to find a solution accurate to within epsilon
 = 10^(-8)
    for x - 2^(-x) = 0 on the interval [0, 1].
    Set the maximum number of iterations to be 30, and use the
    stopping criteria (b_n - a_n)/2 < epsilon
%}

fprintf('Problem 1\n')
% Set parameters
lowerBound = 0;         % set lower bound
upperBound = 1;         % set upper bound
tolerance = 10^(-8);    % set tolerance
maxIterations = 30;     % set maximum iterations
f = @(x) x - 2^(-x);    % define function f

% Define Stop Criteria function
stopCriteriaF = @(lowerBound,upperBound) (upperBound - lowerBound)/2;

% Run bisection method
n = 0;                  % initialize counter
found = 0;              % initialize found to false

while n < maxIterations
    % find midpoint
    pn = (lowerBound + upperBound)/2;
    %test f(p) and stopping criteria.
    if f(pn) == 0 || stopCriteriaF(lowerBound,upperBound) < tolerance
        found = 1;
        approxSolution = pn;
        numIterations = n;
        break
    end
    % increment n
    n = n + 1;
    % assign new bounderies and save previous p
    if sign(f(lowerBound))*sign(f(pn)) < 0
        upperBound = pn; % update upperBound
    else
        lowerBound = pn; % update lowerBound
    end
end
```

```
% Print Results
if found == 0
    fprintf('Method could not find a zero within %d iterations for:
\n', maxIterations)
    fprintf('\tepsilon = %s\n', tolerance)
else
    fprintf('\tapproximation: %1.8f \n\titerations: %d \n\ttolerance:
 %s\n',approxSolution, numIterations, tolerance)
end
```

```
Problem 1
 approximation: 0.64118574
 iterations: 26
 tolerance: 1.000000e-08
```

Problem 2

```
%{
Repeat Problem 1 using epsilon = 10^-12.
%}

fprintf('\nProblem 2\n')

% Set parameters
lowerBound = 0;          % set lower bound
upperBound = 1;          % set upper bound
tolerance = 10^(-12);    % set tolerance
maxIterations = 30;      % set maximum iterations
f = @(x) x - 2^(-x);     % define function f

% Define Stop Criteria function
stopCriteriaF = @(lowerBound,upperBound) (upperBound - lowerBound)/2;

% Run bisection method
n = 0;                   % initialize counter
found = 0;               % initialize solution found to false

while n < maxIterations
    % find midpoint
    pn = (lowerBound + upperBound)/2;
    %test f(p) and stopping criteria.
    if f(pn) == 0 || stopCriteriaF(lowerBound,upperBound) < tolerance
        found = 1;
        approxSolution = pn;
        numIterations = n;
        break
    end
    % increment n
    n = n + 1;
    % assign new bounderies and save previous p
    if sign(f(lowerBound))*sign(f(pn)) < 0
        upperBound = pn; % update upperBound
```

```
    else
        lowerBound = pn; % update lowerBound
    end
end

% Print Results
if found == 0
    fprintf('Method could not find a zero within %d iterations for:
\n', maxIterations)
    fprintf('\tepsilon = %s\n', tolerance)
else
    fprintf('\tapproximation: %1.8f \n\titerations: %d \n\ttolerance:
 %s\n',approxSolution, numIterations, tolerance)
end
```

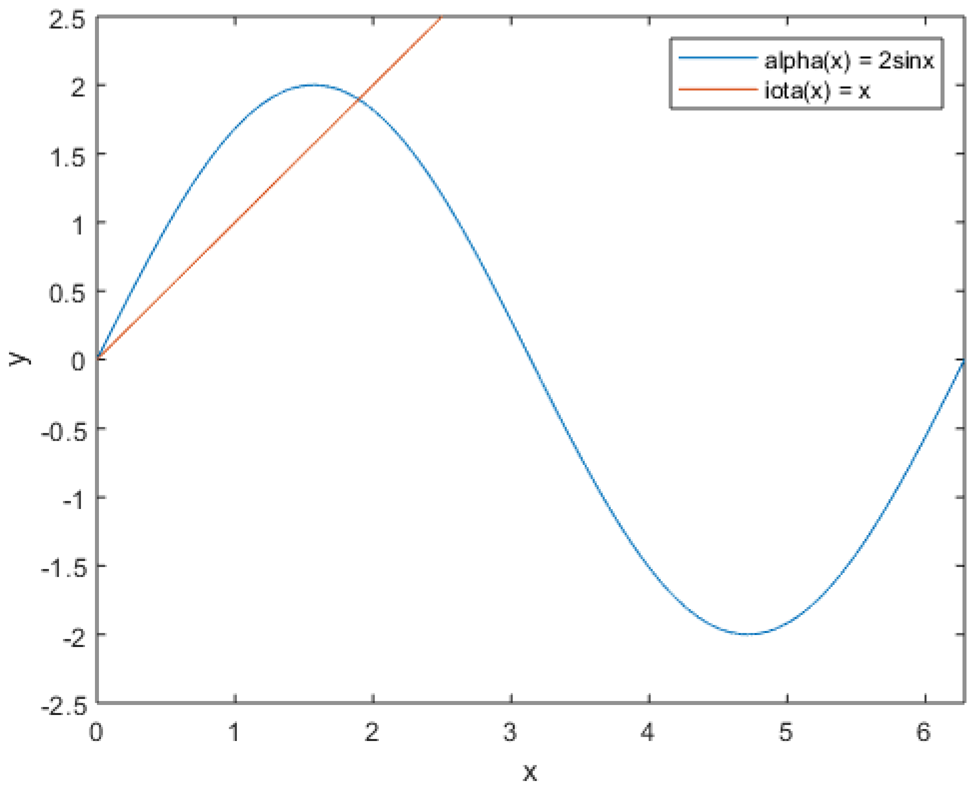
Problem 2
Method could not find a zero within 30 iterations for:
epsilon = 1.000000e-12

Problem 3

```
%{
Plot the graphs of y = x and y = 2*sin(x). Use the Bisection method to
 find an approximation
    to within epsilon = 10^-8 to the first positive value of x with x
 = 2sin x.
    Use the stopping criteria: |(pn - pnMinus1)/pn| < epsilon

Remark: The first positive value of x = 2sin(x) will occur on
 [0,2[ because
        2sin(x) < 2 for every x
%}

fprintf('\nProblem 3\n')

% Define functions as
iota = @(x) x;                  % define function iota as the identity
 function (y=x)
alpha = @(x) 2*sin(x);          % define function alpha

% Define set of inputs and outpoints for functions
xValues = 0:0.001:2*pi;          % define inputs for both functions
 iota and alpha
yValuesIota = xValues;          % because y = x for iota
yValuesAlpha = alpha(xValues);  % find y values for alpha

% Plot functions
plot(xValues,yValuesAlpha)      % plot alpa
axis([0, 2*pi, -2.5, 2.5])      % set axis limits
hold on                         % MATLAB magic to print another
 function on same plots
plot(xValues, yValuesIota)      % plot iota on same graph as alpha
```

```
legend('alpha(x) = 2sinx', 'iota(x) = x'); xlabel('x'); ylabel('y')

% Set parameters
lowerBound = 1.5;                % set lower bound based on looking at
 graph
upperBound = 2;                  % set upper bound based on looking at
 graph
tolerance = 10^(-8);             % define tolerance
maxIterations = 30;              % set max iterations
f = @(x) iota(x) - alpha(x);     % define function with fixed point at
 some p

% Define Stop Criteria function
stopCriteriaF = @(pn, pnMinus1) abs((pn - pnMinus1)/pn);

%Run Bisection Method
n = 0;                           % initialize counter
found = 0;                       % initialize solution found to false

while n < maxIterations
    % find midpoint
    pn = (lowerBound + upperBound)/2;
    %test f(p) and stopping criteria.
    if n > 0
        if f(pn) == 0 || stopCriteriaF(pn,pnMinus1) < tolerance
            found = 1;
            approxSolution = pn;
            numIterations = n;
            break
        end
    end
    % increment n
    n = n + 1;
    % assign new bounderies and save previous p
    if sign(f(lowerBound))*sign(f(pn)) < 0
        pnMinus1 = upperBound; % used in stopping criteria 2
        upperBound = pn; % update upperBound
    else
        pnMinus1 = lowerBound; % used in stopping criteria 2
        lowerBound = pn; % update lowerBound
    end
end

% Print Results
if found == 0
    fprintf('Method could not find a zero within %d iterations for:
\n', maxIterations)
    fprintf('\tepsilon = %s\n', tolerance)
else
    fprintf('\tapproximation: %1.8f \n\titerations: %d \n\ttolerance:
 %s\n',approxSolution, numIterations, tolerance)
end
```

```
Problem 3
 approximation: 1.89549426
 iterations: 26
 tolerance: 1.000000e-08
```



Probelm 4

```
%{
Find an approximation to the root(3, 25) correct to within epsilon =
 10^(-10)
    using the Bisection method. Use the stopping criteria:
        abs((pn - pnMinus1)/pn) < epsilon

Remark: we can define a function with a fixed point at root(3,25) by
        noting that if
            f := x -> root(3,x) : R -> R and   f(25) = root(3,25)
        => f^3:= x ->      x    : R -> R and f^3(25) = 25
           and if g := x -> f^3(x) - 25: R -> R  and if g(x) = 0
        => f^3(x) - 25 = 0, which is a root finding problem whose
 solution
           is the answer for which we are searching and can be found
 with
           the bisection method using this definition of g.

Remark:    8 < 25 < 27
        => root(3,8) < root(3,25) < root(3,27)
```

```
        => 2 < root(3,25) < 3
        => root(3,25) in ]2,3[
%}

fprintf('\nProblem 4\n')

% Set Parameters
lowerBound = 2;                % set lower bound based on looking at
 graph
upperBound = 3;                % set upper bound based on looking at
 graph
tolerance = 10^(-8);           % set tolerance
maxIterations = 30;            % set max iterations
g = @(x) x^3 - 25;             % define function with fixed point at
 some p

% Define Stop Criteria function
stopCriteriaF = @(pn, pnMinus1) abs((pn - pnMinus1)/pn);

%Run Bisection Method
n = 0;                         % initialize counter
found = 0;                     % initialize solution found to false

while n < maxIterations
    % find midpoint
    pn = (lowerBound + upperBound)/2;
    %test f(p) and stopping criteria.
    if n > 0
        if g(pn) == 0 || stopCriteriaF(pn,pnMinus1) < tolerance
            found = 1;
            approxSolution = pn;
            numIterations = n;
            break
        end
    end

    n = n + 1; % increment n

    % assign new bounderies and save previous p
    sgn = sign(g(lowerBound))*sign(g(pn));
    if sgn < 0
        pnMinus1 = upperBound;
        upperBound = pn; % update upperBound
    else
        pnMinus1 = lowerBound;
        lowerBound = pn; % update lowerBound
    end
    %fprintf('% d, %d, %.8f, %.8f, %.8f \n', n, sgn, lowerBound,
 upperBound, pn)
end

% Print Results
if found == 0
```

```
    fprintf('Method could not find a zero within %d iterations for:
\n', maxIterations)
    fprintf('\tepsilon = %s\n', tolerance)
else
    fprintf('\tapproximation: %1.8f \n\titerations: %d \n\ttolerance:
 %s\n',approxSolution, numIterations, tolerance)
end
```

```
Problem 4
 approximation: 2.92401773
 iterations: 26
 tolerance: 1.000000e-08
```

Published with MATLAB® R2018b