Lab 2

Table of Contents

Peter Mowen Due: 24/09/18

Problem 1

Computer approximations for root(1/3, 21) using fixed-point iteration

```
% parameters for all problems in problem 1
p0 = 1;                  % set initial guess
tolerance = 10^(-10);    % set tolerance
stoppingCriteria = 1;    % absolute error
maxIterations = 150;     % set max iterations so computer doesn't run
 forever

% 1.a
fprintf('Problem 1a\n')
f = @(n) (20*n + (21/(n^2)))/21;
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, f);
% print results
fprintf('n: %d \tp%d: %.10f \t |error|: %-0.10f\n', numIterations, ...
    numIterations, approxSol, finalError);

%1.b
fprintf('Problem 1b\n')
f = @(n) n - (n^3 - 21)/(3*(n^2));
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, f);
% print results
fprintf('n: %d \tp%d: %.10f \t |error|: %-0.10f\n', numIterations, ...
    numIterations, approxSol, finalError);

%1.c
fprintf('Problem 1c\n')
maxIterations = 150;
f = @(n) n - (n^4 - 21*n)/((n^2) - 21); % define function
% run fixed-point iteration
```

```
[foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, f);
% print results
fprintf('n: %d \tp%d: %.10f \t |error|: %-0.10f\n', numIterations, ...
    numIterations, approxSol, finalError);
%1.d
fprintf('Problem 1d\n')
maxIterations = 150;
f = @(n) (21/n)^(1/2); % define function
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, f);
% print results
fprintf('n: %d \tp%d: %.10f \t |error|: %-0.10f\n', numIterations, ...
    numIterations, approxSol, finalError);

fprintf('The different methods can be ranked from fastest to slowest
 as follows:\n');
fprintf('1b, 1a, 1d\n')
fprintf( '1c did not converge to the expected value so is not a good
 method.\n');
```

Problem 1a

Problem 2

Use Netwon's method to solve find root(1/3, 25).

```
fprintf('\nProblem 2\n')
% set parameters
p0 = 1;                   % set initial guess
tolerance = 10^(-10);     % set tolerance
stoppingCriteria = 2;     % relative error
maxIterations = 150;      % set max iterations so computer doesn't run
 forever

% Create function g(x) for f(x) = x^3 - 25
g = @(x) x - ((x)^(3) - 25 ) / ( (3) * (x^(2)) );
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, g);
% print results
fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n', numIterations, ...
    numIterations, approxSol, finalError);

fprintf('Using the bisection method, it took 26 iterations to find the
 approximate solution. \n')
fprintf('while using Newton''s Method, it only took 9 iterations.\n')
```

Problem 2

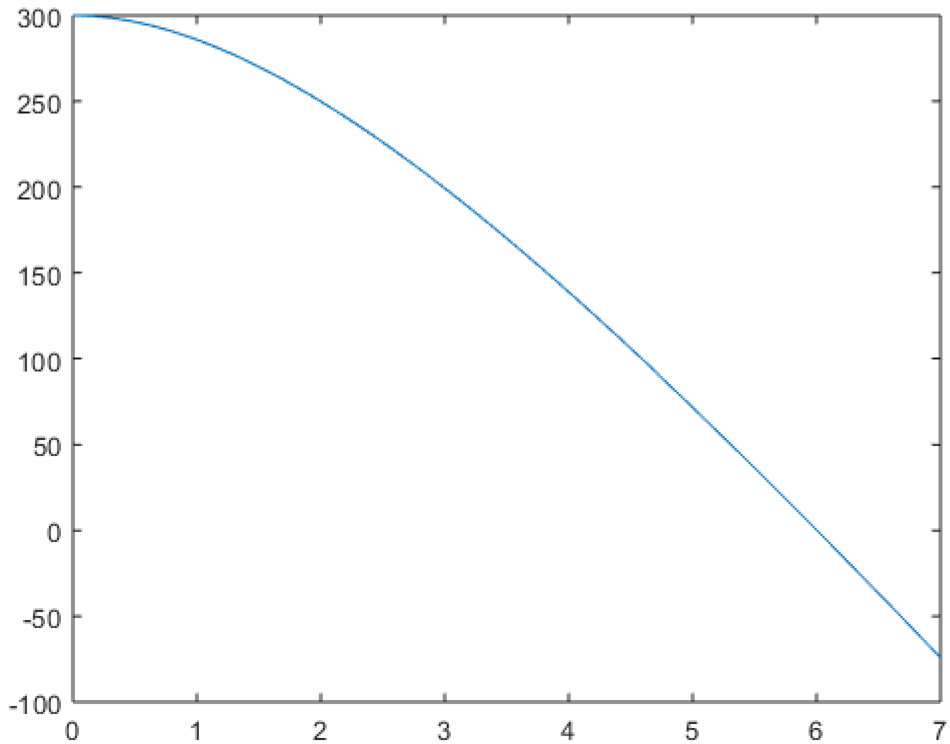
```
n: 9  p9: 2.9240177382   |error|: 0.0000000000
Using the bisection method, it took 26 iterations to find the
 approximate solution.
while using Newton's Method, it only took 9 iterations.
```

Problem 3

An object falling vertically through the air is subjected to viscous resistance as well as to the force of gravity. Assume that an object with mass m is dropped from a height s0 and that the height of the object after t seconds is given by: s(t) = s0 - (mg/k)t + (((m^2)g)/(k^2))(1 - exp(-kt/m)) where s0, m, g, and k are defined below. Use Newtons method to find the time, accurate to 10^(-5)s it takes to hit the groun

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

% constants for function s
s0 = 300;   % intial height in ft
m = 0.25;   % mass in lb
g = 32.17;  % ft/(s^2)
k = 0.1;    % lb/s

% define function s
s = @(t) s0 - (((m * g) / k) * t) + ( ((m^2)*g) / (k^2) )*( 1 -
 exp( (-k / m) * t ));

% define dsdt as derivative of s
dsdt = @(t) (-(m * g) / k) + ( ((m * g) / k) * exp( (-k / m) * t ));

% define g(t) = t - s(t)/dsdt(t)
g = @(t) t - s(t)/dsdt(t);

% plot s(t) to estimate zero
tVals = [0:0.01:7];     % define a range of inputs to test
yVals = s(tVals);       % find outputs using y = g(t)
figure(1);
plot(tVals, yVals)      % view graph to figure out a good p0

% set parameters
p0 = 6;                 % set initial guess based on plot from above
tolerance = 10^(-5);    % set tolerance
stoppingCriteria = 2;   % relative error
maxIterations = 150;    % set max iterations so computer doesn't run
 forever
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, maxIterations, g);
% print results
fprintf('n: %d \tp%d: %.5f \t |error|: %.5f\n', numIterations,...
    numIterations, approxSol, finalError);

fprintf('The object will hit the ground in %.5f seconds\n',
 approxSol);
```

```
Problem 3
n: 2  p2: 6.00373   |error|: 0.00000
The object will hit the ground in 6.00373 seconds
```



Problem 4

A drug administered to a patient produces a concentration in the bloodstream given by c(t) = Ate^(-t/3) milligrams per milliliter, t hours after A units have been injected. The maximum safe concentration is 1 mg/mL. What amount should be injected to reach this maximum safe concentration, and when does this maximum occur? An additional amount of this drug is to be administered to the patient after the concentration falls to 0.25 mg/mL. Determine, to the nearest minute, when this second injection should be given.

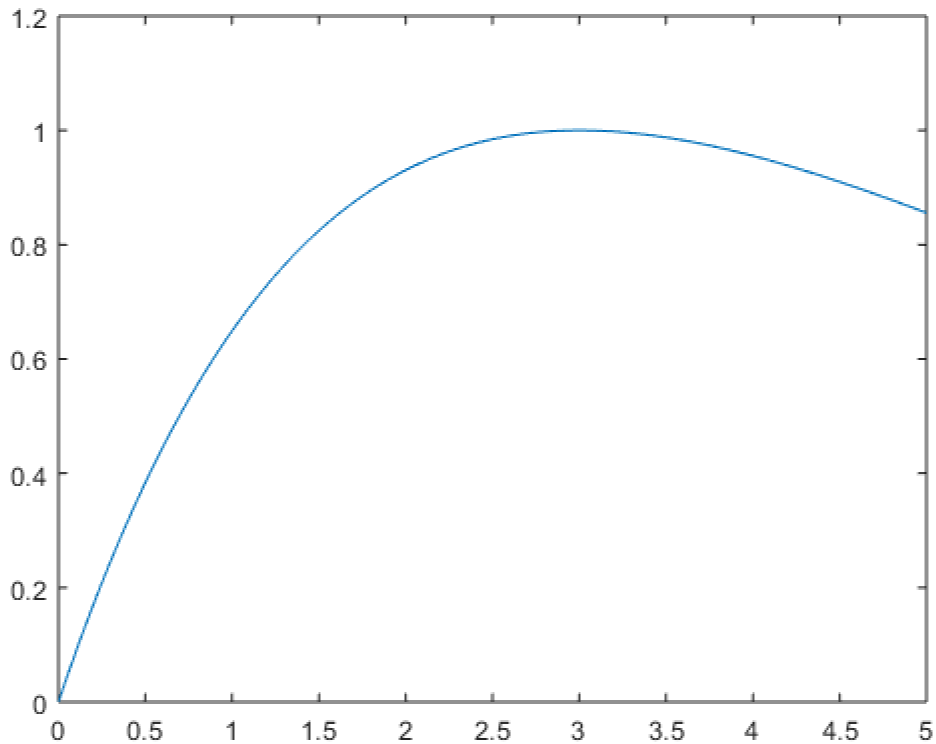
```
% Differentiating c(t) gives us dcdt(t) = A(e^(-t/3))*(1-t/3). Setting
 this
% equal to zero and solving gives us the critical value of 3. Looking
 at
% the graph of (e^(-t/3))*(1-t/3) shows us that this is where the
 maximum
% will occur. We wantthis maximum to be 1, so we can setup the
 following to
% find a value for A:
%       A ( 3 ) (e ^(-3/3)) = 1
%   =>  A = e/3

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

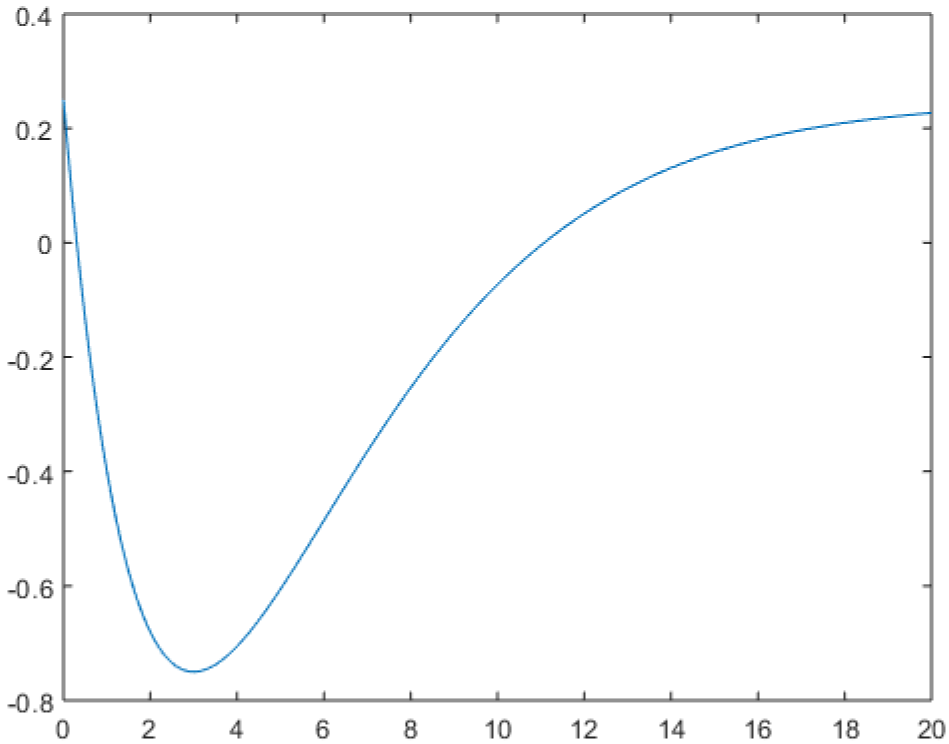
```
A = exp(1) /3                     % Amount to be administered
c = @(t) A* t .* exp( -t / 3);   % concentration over time

% plot c(t)
tVals = 0:0.01:5;
yVals = c(tVals);
figure(3);
plot(tVals, yVals)

maxConcentration = c(3)          % display to show that c(3) = 1

% We can use Newton's method to find when the concentration will be
 0.25 mg
% using the following:
%       c(t) = 0.25
%   =>  0 = 0.25 - c(t) =: f(t)
%   => g(t) = t - f(t)/f'(t)
f = @(t) 0.25 - c(t);
dfdt = @(t) - A * (exp( -t / 3)) .* ( 1 - t/3 );
g = @(t) t - f(t) / dfdt(t);

% plot f(t) to find p0 near where f(t) = 0 and where t > 3
tVals = [0:0.1:20];
yVals = f(tVals);
figure(4);
plot(tVals,yVals)

% set parameters
p0 = 11;                 % set initial guess based on plot from above
tolerance = 10^(-2);     % set tolerance based on min value of drug in
 system
stoppingCriteria = 2;    % relative error
maxIterations = 150;     % set max iterations so computer doesn't run
 forever
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, maxIterations, g);
% print results
fprintf('n: %d \tp%d: %.2f \t |error|: %.2f\n', numIterations,...
    numIterations, approxSol, finalError);

concentationAtTimetoReadminister = c(approxSol)
timeForReadministrationToMinute = approxSol * 60;
fprintf('The second injection should be given %.0f minutes after the
 initial injection\n\n',...
    timeForReadministrationToMinute)
```

Problem 4

A =

```
    0.9061


maxConcentration =

    1.0000

n: 1  p1: 11.08   |error|: 0.01

concentationAtTimetoReadminister =

    0.2500

The second injection should be given 665 minutes after the initial
 injection
```





Problem 5

Approximate the zero of the function f(x) = x^2 - 2e^(-x)+e^(2x) to within 10^(-8) using Newton's method with p0 = 1.

```
fprintf('\nProblem 5\n')

f = @(x) x^2 - (2 * exp(-x)) + exp(2*x);
dfdt = @(x) (2 * x) + (2 * exp(-x)) + ( 2 * exp(2*x));
g = @(x) x - ( f(x) / dfdt(x) );

% set parameters
p0 = 1;                 % set initial guess
tolerance = 10^(-8);    % set tolerance
stoppingCriteria = 2;   % relative error
maxIterations = 150;    % set max iterations so computer doesn't run
 forever
% run fixed-point iteration
[foundSol, numIterations, approxSol, finalError] =
 FixedPointIteration(p0, ...
    tolerance, stoppingCriteria, maxIterations, g);
%print results
fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n', numIterations,...
    numIterations, approxSol, finalError);
```

```
Problem 5
n: 6  p6: 0.2207627398   |error|: 0.0000000002
```

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

% fixed-point iteration
function [foundSol, numIterations, approxSol, finalError] = ...
    FixedPointIteration(p0, tolerance, stoppingCriteria,
 maxIterations, f)
% This function finds an approximate zero to the input function using
%   fixed-point iteration.
%   INPUTS: po - starting guess, tolerance, maximum number of
 iterations,
%           a function f
%   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
pn = p0; %set pn to p0 for first run
while n <= maxIterations
    xn = f(pn);
    if stoppingCriteria == 1
        error = AbsoluteError(xn, pn);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = xn; % appoximate solution
            finalError = error;
            break
        end
    end
    if stoppingCriteria == 2
```

```
        error = RelativeError(xn, pn);
        if error < tolerance
            foundSol = 1; % solution found
            numIterations = n;
            approxSol = xn; % appoximate solution
            finalError = error;
            break
        end
    end
    n = n + 1; % increment iteration counter
    pn = xn; % set input for next iteration to output from this one
    %fprintf('n: %d \tp%d: %.10f \t |error|: %.10f\n', n, n, xn,
 error);
end
end
```

```
n: 135  p135: 2.7589241758   |error|: 0.0000000001
Problem 1b
n: 9  p9: 2.7589241764   |error|: 0.0000000000
Problem 1c
n: 2  p2: 0.0000000000   |error|: 0.0000000000
Problem 1d
n: 37  p37: 2.7589241764   |error|: 0.0000000001
The different methods can be ranked from fastest to slowest as
 follows:
1b, 1a, 1d
1c did not converge to the expected value so is not a good method.
```

Published with MATLAB® R2018b