

**EMAT20920: Numerical Methods in MATLAB**

**COURSEWORK ASSESSMENT**

**Jake Bowhay (UP19056)**

All figures in this report have been saved using `saveFigPDF` function as it automatically resizes the paper to the correct size.

```
function saveFigPDF(fileName)
    %saveFigPDF saves open figure as a PDF file
    %
    %Inputs:
    %  fileName = File name to save figure as
    %Usage:
    %  saveFigPDF("polynomial") -> Saves current figure as polynomial.pdf

    % Get current figure handle
    figureHandle = gcf;
    % Resize paper
    set(figureHandle, 'PaperPosition', 3*[0 0 6 4]);
    set(figureHandle, 'PaperSize', 3*[6 4]);
    set(figureHandle, 'PaperUnits', 'centimeters');

    print(fileName, '-dpdf');
end
```

**Question 1: Root-finding**

- (a) To find how many solutions each equation has in the given domain I will rearrange all the equations to be equal to zero and then looks for the zeros of the rearranged equations. As a corollary to the intermediate value theorem, if a function is continuous and changes sign in a bracket then that bracket must contain a zero. So I will plot each of the rearranged equation and I look for appropriate brackets. I will use the `pltFunc` function to plot the functions as it removes values outside a defined limit which prevents MATLAB plotting discontinuous functions as continuous. The limits can then be changed using the property explorer to give a more useful plot.

```
function pltFunc(f, domain, discountLim)
    %pltFunc plots function f between values of xLim removing any values
    % that are greater than discountLim to prevent MATLAB plotting
    % discontinuous functions as continuous and plots a line of x = 0 to
    % help make any zeros clear
    %
    % Input:
    %  f = function handle to plot
    %  domain = 1x2 vector containing the lower and upper bound of the
    %  domain of f
    %  discountLim = absolute values of the function greater than this are
    %  changed to NaN. Setting to inf will plot all values of the function
    %
    % Usage:
    %  pltFunc(@(x) 1./x, [-10 10], 5) -> Plots 1/x between -10 and 10
    %  changing the values where |1/x|>5 to NaN
```

```

% Check xlim is the correct dimensions
assert(isequal(size(domain), [1 2]), "domain must be a 1x2 vector")

%% Generate values to plot
x = linspace(domain(1), domain(2));
y = f(x);

% Remove large values of y to prevent MATLAB plotting discontinuous
% functions as continuous
y(abs(y)>discontLim) = NaN;

%% Plot function and line x = 0
plot(x, y, [min(x) max(x)], [0 0], "g-", "LineWidth", 2);
xlabel("x");
ylabel("f(x)");
xlim(domain);
title("Plot of f(x)");
grid on;
end

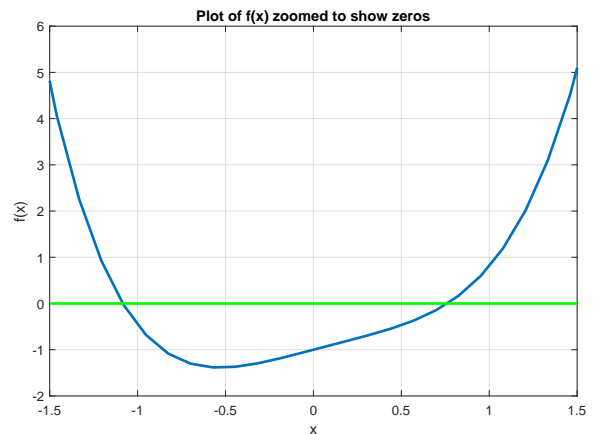
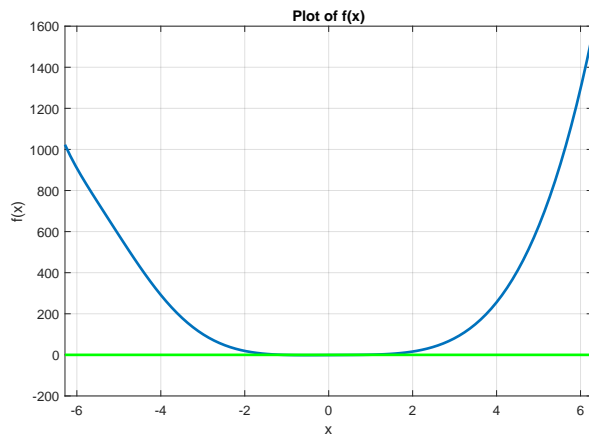
```

- (i) Rearranging  $x^4 = e^{-x} \cos(x)$  gives  $f(x) = x^4 - e^{-x} \cos(x)$ .

```

f = @(x) x.^4 - exp(-x).*cos(x);
pltFunc(f, [-2*pi 2*pi], inf);

```



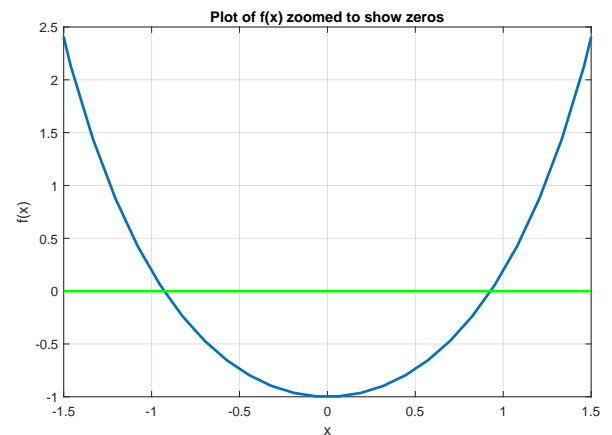
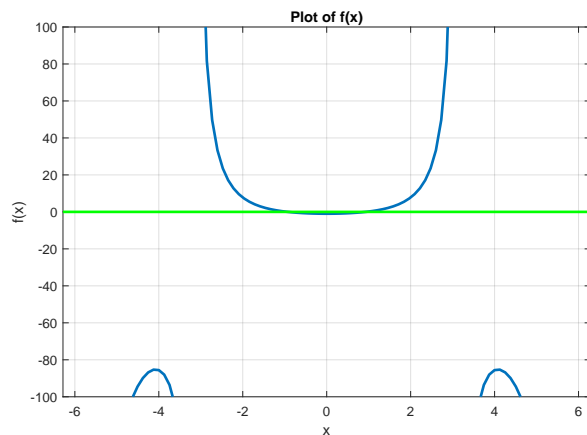
The second zoomed in plot shows there are two zeros in the given domain. The first zero can be bracketed by the interval  $[-1.5, -1]$  as  $f(-1.5) = 4.7455$  and  $f(-1) = -0.4687$  so since the function is continuous and there is a change of sign this bracket must contain a zero. Like wise the second root can be bracketed by the interval  $[0.5, 1]$  as  $f(0.5) = -0.4698$  and  $f(1) = 0.8012$ .

- (ii) Setting  $f(x) = \frac{x^3}{\sin(x)} - 1$ .

```

f = @(x) (x.^3)./sin(x) - 1;
pltFunc(f, [-2*pi 2*pi], 500);

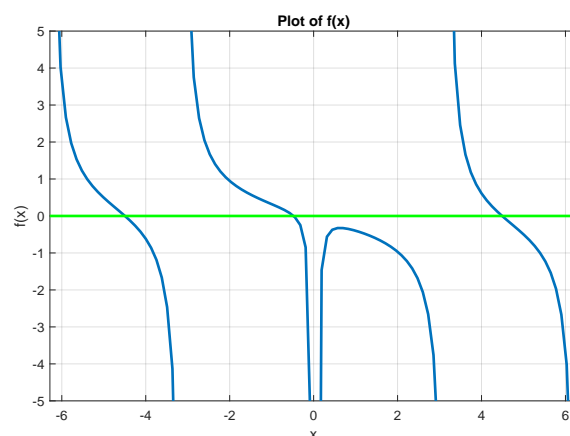
```



The second plot shows there are two roots. The first root can be bracketed by the interval  $[-1, -0.5]$  as  $f(-1) = 0.1884$  and  $f(-0.5) = -0.7393$  and  $f(x)$  is continuous in this bracket. Likewise, the second root can be bracketed by the interval  $[0.5, 1]$  as  $f(0.5) = -0.7393$  and  $f(1) = 0.1884$ .

- (iii) Rearranging  $\cot(x) = \frac{25}{25x-1}$  gives  $f(x) = \cot(x) - \frac{25}{25x-1}$ .

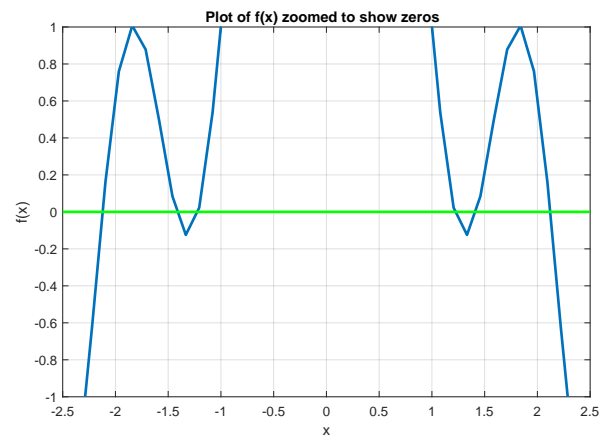
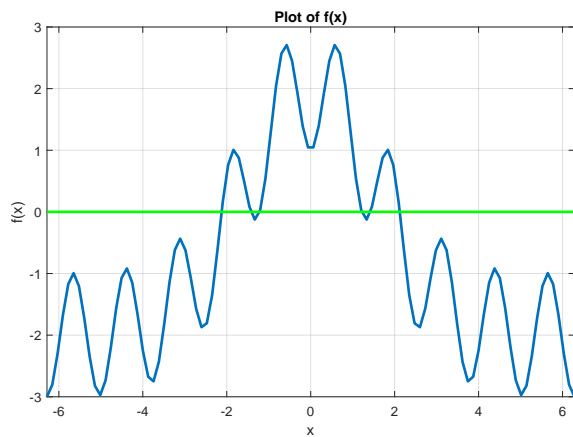
```
f = @(x) cot(x) - 25./(25*x - 1);
pltFunc(f, [-2*pi 2*pi], 30);
```



The plot shows that the equation has three solutions. The first can be bracketed by the interval  $[-5, -4]$  as  $f(-5) = 0.4942$  and  $f(-4) = -0.6162$ . The second solution can be bracketed by the interval  $[-1, -0.1]$  as  $f(-1) = 0.3194$  and  $f(-0.1) = -2.8238$ . The third solution can be bracketed by the interval  $[4, 5]$  as  $f(4) = 0.6112$  and  $f(5) = -0.4974$ .  $f(x)$  is continuous in each of the bracketing intervals.

- (iv) Rearranging  $4e^{-x^2/5} = \cos(5x) + 2$  gives  $f(x) = 4e^{-x^2/5} - \cos(5x) - 2$ .

```
f = @(x) 4*exp(-x.^2/5) - cos(5*x) - 2;
pltFunc(f, [-2*pi 2*pi], inf);
```



The second plot shows that the equation has 6 solutions. The bracketing intervals are shown in the table below.

$[a, b]$	$f(a)$	$f(b)$
$[-2.5, -2]$	-1.8518	0.6364
$[-1.5, -1.25]$	0.2039	-0.0730
$[-1.25, -1]$	-0.0730	0.9913
$[1, 1.25]$	0.9913	-0.0730
$[1.25, 1.5]$	-0.0730	0.2039
$[2, 2.5]$	0.6364	-1.8518

(b) The bisection method is used by calling the the `bisectRoot` function.

```
function [sol, i, err] = bisectRoot(f, a, b, tol)
    %bisectRoot Use the bisection method to find roots of the function f
    % bracketed within the intervals [a, b].
    %
    %Inputs:
    % f = function handle to function whose root is to be found
    % a = 1*n array containing all the lower ends of the brackets
    % where n is the number of roots
    % b = 1*n array containing all the lower ends of the brackets
    % where n is the number of roots
    % tol = absolute error tolerance with which to find the root;
    % Iteration terminates when the root is known to within +/- tol
    %
    %Outputs:
    % sol = 1*n array of of roots
    % i = 1*n array of the number of iterations required to find the nth
    % root
    % err =
    %
    %Usage:
    % [r, i, err] = bisect(@(x) x.^2 - 4, 1, 3, 5e-9) -> returns the
    % approximation to root of  $x^2 - 4 = 0$  within  $[1, 3]$ , the number of
    % iterations required to find the root and the final absolute error
    %
    % check if all intervals are correctly defined
    assert(isequal(size(a), size(b)), ...
        "Must be an equal number of upper and lower bounds");
```

```

% check whether f changes sign
assert(all(sign(f(a)) ~= sign(f(b))),...
    'f(a) and f(b) should have opposite sign');

% initialise variables
% iteration counter
i = zeros(size(a));
% current solution estimate
sol = (a + b)/2;
% previous solution estimate
sol_old = Inf;
% absolute error
err = Inf;
withinTol = zeros(size(a));

% bisection algorithm:
% at each iteration, find the half-interval that contains a sign change
% and relabel the endpoints appropriately
while any(~withinTol)
    i(~withinTol) = i(~withinTol) + 1;
    sol_old = sol;
    mid = (a + b)/2;

    % mid point is a root
    exactRoot = f(mid) == 0;
    sol(exactRoot) = mid(exactRoot);
    err(exactRoot) = 0;
    withinTol(exactRoot) = true;

    % solution is in first half of interval and mid point not a root
    firstHalf = (sign(f(a)) ~= sign(f(mid))) & ~exactRoot;
    b(firstHalf) = mid(firstHalf);

    % solution is in second half of interval and mid point not a root
    secondHalf = (sign(f(a)) == sign(f(mid))) & ~exactRoot;
    a(secondHalf) = mid(secondHalf);

    % update solutions and errors values that aren't within tolerance
    sol(~withinTol) = (a(~withinTol) + b(~withinTol))/2;
    err(~withinTol) = abs(sol(~withinTol) - sol_old(~withinTol));
    withinTol(err < tol) = true;
end
end

```

- (i) Solutions to  $f(x) = x^4 - e^{-x} \cos(x) = 0$   $x \in [-2\pi, 2\pi]$ .

```

f = @(x) x.^4 - exp(-x).*cos(x);
a = [-1.5 0.5];
b = [-1 1];
[r, i, err] = bisectRoot(f, a, b, [5e-8 5e-9])

```

Note the two different tolerances since one root is an order of magnitude larger so requires one less decimal place of accuracy to be accurate to 8 significant figures.

$[a, b]$	Root	# Iterations
$[-1.5, -1]$	-1.0843597	23
$[0.5, 1]$	0.76221107	26

- (ii) Solutions to  $f(x) = \frac{x^3}{\sin(x)} - 1 = 0 \quad x \in [-2\pi, 2\pi]$ .

```
f = @(x) (x.^3)./sin(x) - 1;
a = [-1 0.5];
b = [-0.5 1];
[r, i, err] = bisectRoot(f, a, b, 5e-9)
```

$[a, b]$	Root	# Iterations
$[-1, -0.5]$	-0.92862631	26
$[0.5, 1]$	0.92862631	26

- (iii) Solutions to  $f(x) = \cot(x) - \frac{25}{25x-1} = 0 \quad x \in [-2\pi, 2\pi]$ .

```
f = @(x) cot(x) - 25./(25*x - 1);
a = [-5 -1 4];
b = [-4 -0.1 5];
[r, i, err] = bisectRoot(f, a, b, [5e-8 5e-9 5e-8])
```

$[a, b]$	Root	# Iterations
$[-5, -4]$	-4.4953722	24
$[-1, -0.1]$	-0.47773376	27
$[4, 5]$	4.4914097	24

- (iv) Solutions to  $f(x) = 4e^{-x^2/5} - \cos(5x) - 2 = 0 \quad x \in [-2\pi, 2\pi]$ .

```
f = @(x) 4*exp(-x.^2/5) - cos(5*x) - 2;
a = [-2.5 -1.5 -1.25 1 1.25 2];
b = [-2 -1.25 -1 1.25 1.5 2.5];
[r, i, err] = bisectRoot(f, a, b, 5e-8)
```

$[a, b]$	Root	# Iterations
$[-2.5, -2]$	-2.1222382	23
$[-1.5, -1.25]$	-1.4255432	22
$[-1.25, -1]$	-1.2145933	22
$[1, 1.25]$	1.2145933	22
$[1.25, 1.5]$	1.4255432	22
$[2, 2.5]$	2.1222382	23

- (c) The iterative scheme we asked to implement is called Steffensen's method. This is implemented in the `steffensenRoot` function.

```
function [r, n, err] = steffensenRoot(f, x0, tol, nMax)
    %steffensenRoot uses Steffensen's method to find roots of f(x)
    % based on an initial guess x0
    %
    %Inputs:
    % f = function handle to function whose root is to be found
    % x0 = initial guess of the root to begin iteration at
    % tol = absolute error tolerance with which to find the root
    % iteration terminates when the root is known to within +/- tol
    % nMax = the maximum number of iteration to quit after. Prevents an
    % infinite loop if the iterations do not converge
    %
```

```

%Outputs:
% r = the approximate root of f(x)=0
% n = the number of iterations
% err = 1*n vector of the absolute error after each iteration
%
%Usage:
% [r, n, err] = steffensenRoot(@(x) exp(-x) -x, 0, 5e-9, 50) ->
% returns the approximate roo of  $x^x - x = 0$  after n iterations and
% err the absolute error after each iteration

% set initial guess as first root
xn = x0;
%iteration counter
n = 0;
% preallocate error array
err = Inf(1, nMax);

while all(err > tol) && n < nMax
    n = n + 1;
    xOld = xn;
    % Calculate f(xn) to avoid repeat computation
    fn = f(xn);
    % Calculate next iteration
    xn = xn - fn*(f(xn + fn)/fn - 1)^-1;
    err(n) = abs(xn - xOld);
end

% remove any unused preallocated element in error array
err(isinf(err)) = [];

% check if solution converged
assert(err(end) < tol, "No convergence")

r = xn;
end

```

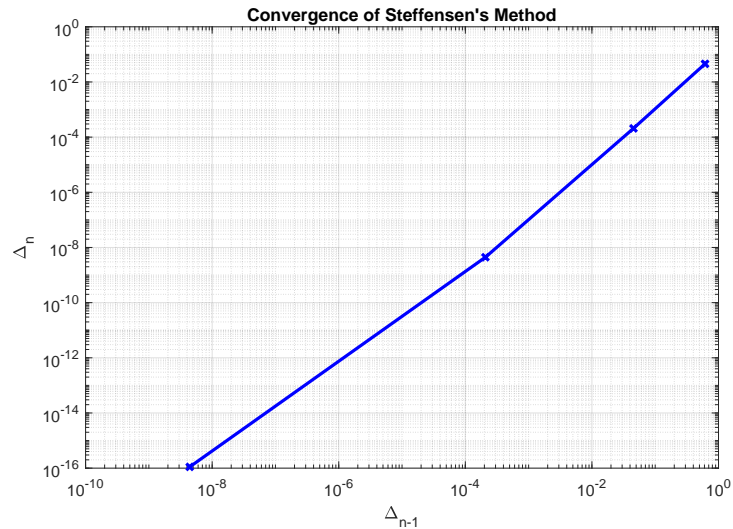
The following uses this function to find the root of  $e^{-x} - x = 0$  and calculate the convergence.

```

f = @(x) exp(-x) - x;
[r, n, e] = steffensenRoot(f, 0, 5e-13, 50)
%% Generate plot of convergence
loglog(e(1:end-1),e(2:end), "bx-", "LineWidth", 2);
title("Convergence of Steffensen's Method")
xlabel('\Delta_{n-1}');
ylabel('\Delta_n');
grid on;
%% Find order of congerence
polyfit(log(e(1:end-1)), log(e(2:end)), 1)

```

After 5 iterations the root  $x = 0.567143290410$  is accurate to 12 decimal places.



The graph shows a straight line which shows  $\text{error} \propto \Delta_{n-1}^q$ , where  $q$  is the gradient of the line. Using the MATLAB function `polyfit` the gradient of the above graph as 1.8 which is close to 2 so Steffensen's method is second order.

- (d) The first step in creating a cobweb plot is to implement a fixed point iteration scheme.

```
function xn = fixedPointRoot(g, x0, nMax)
    % fixedPointRoot Iteration to find solutions of x = g(x)
    %
    %Inputs:
    %   g = function handle to find the solutions of x = g(x)
    %   x0 = first term of the iteration
    %   nMax = the maximum number of iteration to quit after
    %
    %Output:
    %   xn = the iterative sequence
    %
    %Usage:
    %   xn = fixedPointRoot(@(x) cos(x), 0.75, 100) -> looks for a
    %   root of the equation x - cos(x) = 0, starting with an initial guess
    %   of 0.75.

    % number of iterations
    n = 0;
    % preallocated sequence array and set initial guess as first term
    xn = NaN(1, nMax);
    xn(1) = x0;
    % set initial error
    err = Inf;

    % iterate x -> g(x)
    while n < nMax
        n = n + 1;
        xn(n + 1) = g(xn(n));
        err = abs(xn(n + 1) - xn(n));
    end

    % remove any unused elements of the preallocated array
    xn(isnan(xn)) = [];
```



```

fprintf('\nAfter %d steps root is %20.14g\n', n, xn(end));
fprintf('Final absolute error is %g\n\n', err);
end
function cobwebDiagram(g, x0, nMax, a, b)
    %cobwebDiagram Creates cobweb diagram for x = g(x) in interval [a,b]
    %
    %Inputs:
    % g = function handle for g(x)
    % x0 = initial guess to start iteration
    % nMax = number of iterations to complete
    % a = lower end of interval [a,b] to plot cobweb diagram over
    % b = upper end of interval [a,b] to plot cobweb diagram over
    %
    %Usage:
    % cobwebDiagram@(x) (x.^5 + 3)/5, 1, 10, 0, 1.5) -> produces a
    % cobweb diagram of x = (x^5 + 3)/5 based on an initial guess of 10
    % and 10 iterations. This is shown over the interval [0,1.5].

    %% get fixed point iteration sequence
    xn = fixedPointRoot(g, x0, nMax);

    %% generate cobweb diagram

    % get values for the line y = x and y = g(x)
    x = linspace(a, b);
    y = g(x);
    y(isinf(y)) = NaN;

    % set up figure
    hold on;
    grid on;
    set(gca, "DefaultLineLineWidth", 2);
    title("Cobweb plot for fixed point iteration");
    xlabel("x");
    ylabel("y")
    xlim([a b])
    ylim([min(x(1), y(1)) max(x(end), y(end))]);

    % plot lines y = x and y = g(x)
    plot(x, x, "r-", "DisplayName", "y = x");
    plot(x, y, "k-", "DisplayName", "y = g(x)");
    legend("AutoUpdate", "off");

    % plot the steps
    plot([xn(1) xn(1)], [0 xn(2)], 'm-');
    for i=1:length(xn) - 2
        plot([xn(i) xn(i + 1)], [xn(i + 1) xn(i + 1)], 'm-');
        plot([xn(i + 1) xn(i + 1)], [xn(i + 1) xn(i + 2)], 'm-');
    end
end
end

```