

2018

SEMESTER 2

Lab 3: Non Linear Equations

Connor McDowall
530913386
cmcd398

September 4, 2018

Contents

1	Questions	2
1.1	Task 2	2
1.1.1	Question 1	2
1.1.2	Question 2	2
1.2	Task 4	2
1.2.1	Question 1	2
1.2.2	Question 2	2
2	Plots	2
3	Code: Non Linear Equations	5
3.1	NLE Functions	5
3.2	NLE Plotting	10
4	Systems of Non Linear Equations	12
4.1	Newton Two Variable	12
4.2	Newton Two Variable Plotting	15
4.3	Newton Multiple Variable	18

Listings

1	Bisection	5
2	Secant	6
3	Regula Falsi	6
4	Newton	7
5	Combined	8
6	Task 1	9
7	Task 2	10
8	Newton Two Variables	12
9	Task 3	14
10	Task 4	15
11	Newton Multiple Variables	18

List of Figures

1	NLE Function Comparison	3
2	NLE Function Comparison	4

1 Questions

1.1 Task 2

1.1.1 Question 1

The newton method finds the roots for $f(x) = x^2 - 1$ and $f(x) = \cos(x) + \sin(x^2) - 0.5$ in four and seven iterations respectively. The newton method fails to find the root before the maximum number of iterations for $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. The newton method uses the derivative to calculate the new root value. If the derivative is too small, the new root estimated is significantly greater than the current iteration. This was the case for $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ as jumped from a negative function value to a large, positive function value. The combined method uses a combination of the bisection and newton method. The newton method is used first to find a new root estimate. If this estimate falls outside a root range, the bisection method is used instead to find the new root estimate, avoiding extreme leaps. Thereafter, the root bracket is updated. The method continues to iterate until a suitable root is found. The combined method found a root in 4 iterations for $f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$. (I have excluded the initial root estimates from my iterations).

1.1.2 Question 2

Use parallelization. Set up array of root estimates on a plausible range of values. Next, apply the desired root finding method on each element of the array. The desired root finding method is based on the method's properties. Perform the method for each element simultaneously and select the best of the root estimates found. The best root estimates will be the global minima/maxima.

1.2 Task 4

1.2.1 Question 1

The initial root estimate is either too far away from the actual root or the derivative of the function is really small or zero.

1.2.2 Question 2

Use parallelization. Set up matrix of root estimates on a plausible range of values (two to n dimensions) where each column is a different combination of starting points and each row is a different variable in the function set. Next, apply the desired root finding method on each column of the matrix. The desired root finding method is based on the properties you wish to have. Perform the method for each column simultaneously and select the best of the root estimates found. The best root estimates will be the global minima/maxima. This will work for non linear functions with two to n dimensions.

2 Plots

The plots for both Tasks 2 and 4 are on the following pages.

3

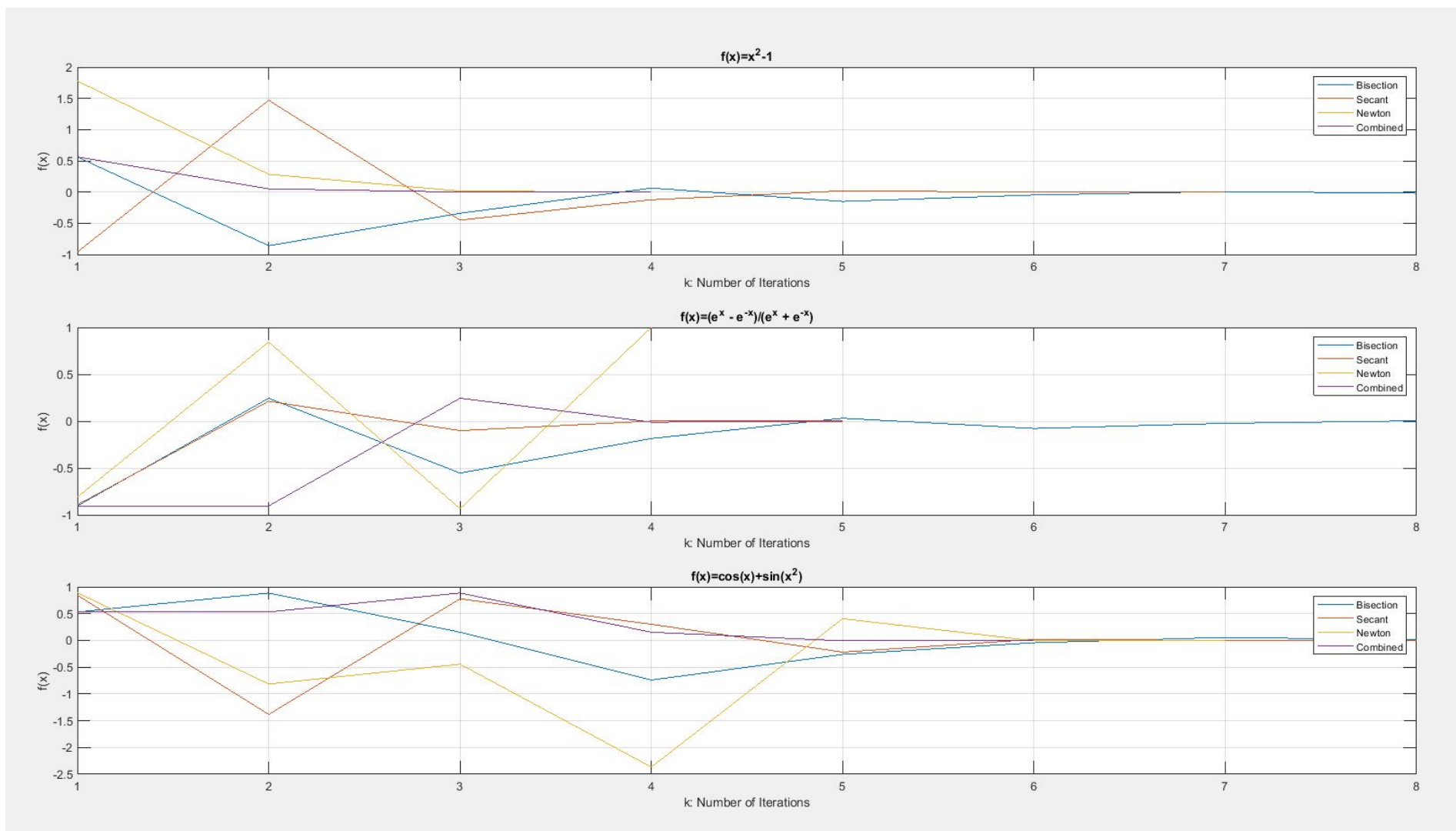


Figure 1: NLE Function Comparison

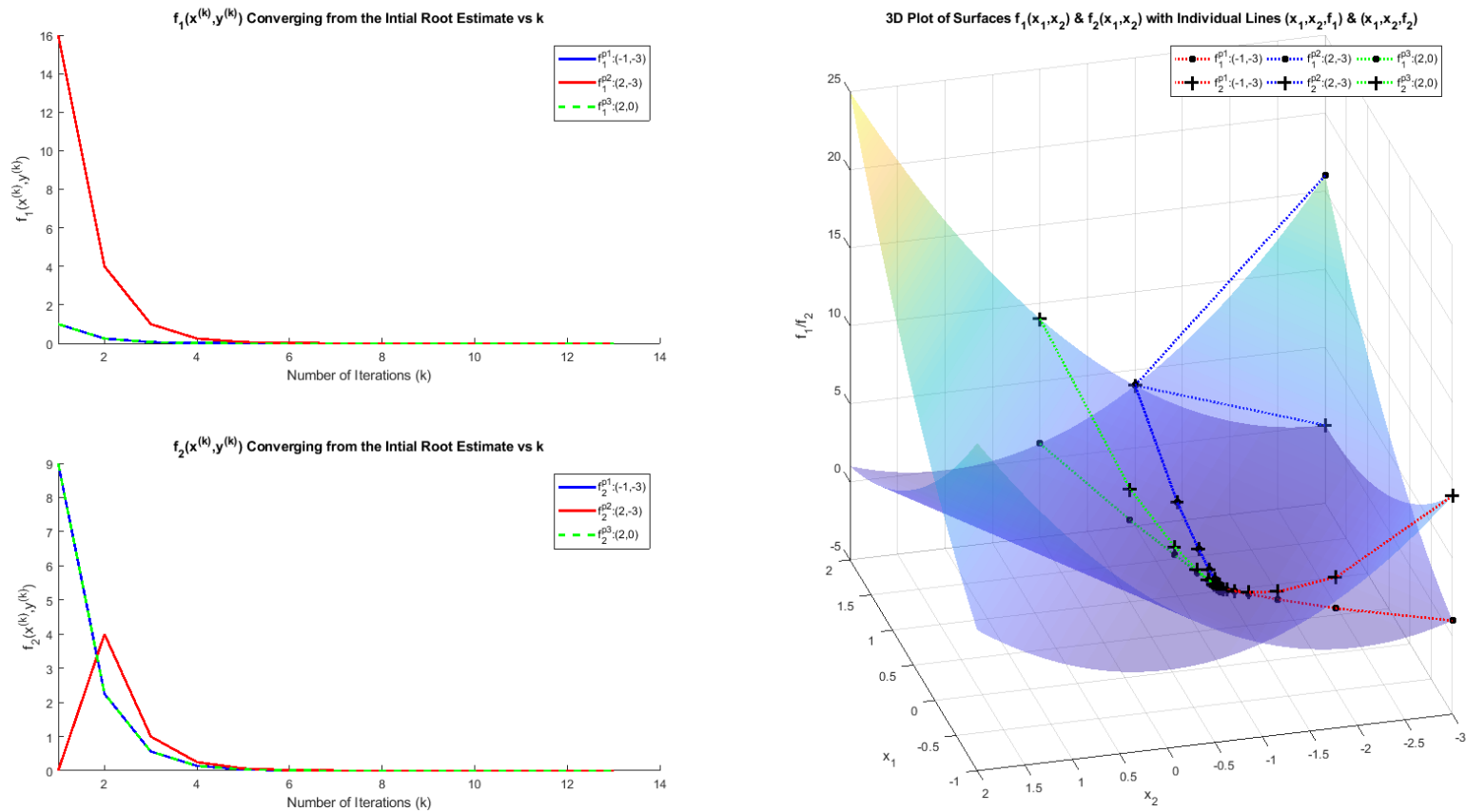


Figure 2: NLE Function Comparison

3 Code: Non Linear Equations

3.1 NLE Functions

Listing 1: Bisection

```

1 % Nonlinear equation root finding by the bisection method.
2 % Inputs
3 % f      : nonlinear function
4 % xl, xr : initial root bracket
5 % nmax   : maximum number of iterations performed
6 % tol    : numerical tolerance used to check for root
7 % Outputs
8 % x      : one-dimensional array containing estimates of root
9
10 % Hint 1:
11 % Iterate until either a root has been found or maximum number of
    iterations has been reached
12
13 % Hint 2:
14 % Check for root each iteration, making use of tol
15
16 % Hint 3:
17 % Update the bracket each iteration
18
19 function x = Bisection(f, xl, xr, nmax, tol)
20     % Initial array of root estimates, iteration and variable
        stores.
21     x = [xl,xr];
22     xbrac = x;
23     n = 1;
24     % Set iterative loop for the function
25     while n < nmax + 1
26         % Calculate the new root
27         xnew = xbrac(1) + ((xbrac(2) - xbrac(1))/2);
28         % Append the root estimate to the array
29         x = [x,xnew];
30         % Terminate function if at derivative point
31         if abs(f(xnew)) <= tol
32             return
33         % Calculate sign on the function with the new root
34         % and find the new root bracket.
35         elseif (f(xnew)*f(xbrac(1)))> 0
36             % Reset the bracket with new LHS
37             xbrac = [xnew,xbrac(2)];
38         elseif (f(xnew)*f(xbrac(2)))> 0
39             % Reset the bracket with new RHS
40             xbrac = [xbrac(1),xnew];
41         end
42         % Increase iteration counter
43         n = n + 1;
44     end

```

```

45     % Warn the user the maximum number of iterations have been
        performed
46     disp('The maximum number of iterations have been performed
        without satisfying the required root finding condition');
47
48 end

```

Listing 2: Secant

```

1  % Nonlinear equation root finding by the secant method.
2  % Inputs
3  % f      : nonlinear function
4  % x0, x1 : initial root bracket
5  % nmax   : maximum number of iterations performed
6  % tol    : numerical tolerance used to check for root
7  % Outputs
8  % x      : one-dimensional array containing estimates of root
9
10 function x = Secant(f, x0, x1, nmax, tol)
11 % Initial array of root estimates, iteration and variable stores.
12     x = [x0,x1];
13     n = 1;
14     k = 2;
15     % Set iterative loop for the function
16     while n <= nmax
17         % Calculate the new root
18         xnew = x(k) - (f(x(k))*(x(k)-x(k-1))/(f(x(k))- f(x(k-1))))
            );
19         % Append the root estimate to the array
20         x = [x,xnew];
21         % Terminate function if at derivative point
22         if abs(f(xnew)) < tol
23             return
24         end
25         % Increase iteration counter and k value
26         n = n + 1;
27         k = k + 1;
28     end
29     % Warn the user the maximum number of iterations have been
        performed
30     disp('The maximum number of iterations have been performed
        without satisfying the required root finding condition');

```

Listing 3: Regula Falsi

```

1  % Nonlinear equation root finding by the Regula falsi method.
2  % Inputs
3  % f      : nonlinear function
4  % x1, xr : initial root bracket
5  % nmax   : maximum number of iterations performed
6  % tol    : numerical tolerance used to check for root
7  % Outputs

```

```

8 % x      : one-dimensional array containing estimates of root
9
10 function x = Regulafalsi(f, xl, xr, nmax, tol)
11 % Initial array of root estimates, iteration and variable stores
12     .
13     x = [xl,xr];
14     xbrac = x;
15     n = 1;
16     % Set iterative loop for the function
17     while n <= nmax
18         % Calculate the new root
19         xnew = xbrac(2) - (f(xbrac(2))*(xbrac(2)-xbrac(1))/(f(
20             xbrac(2))- f(xbrac(1)))));
21         % Append the root estimate to the array
22         x = [x,xnew];
23         % Terminate function if at derivitive point
24         if abs(f(xnew)) < tol
25             return
26         % Calculate sign on the function with the new root
27         % and find the new root bracket.
28         elseif f(xnew)*f(xbrac(1))>0
29             % Reset the bracket with new LHS
30             xbrac = [xnew,xbrac(2)];
31         elseif f(xnew)*f(xbrac(2))>0
32             % Reset the bracket with new RHS
33             xbrac = [xbrac(1),xnew];
34         end
35         % Increase iteration counter
36         n = n + 1;
37     end
38     % Warn the user the maximum number of iterations have been
39     % performed
40     disp('The maximum number of iterations have been performed
41         without satisfying the required root finding condition');

```

Listing 4: Newton

```

1 % Nonlinear equation root finding by Newton's method
2 % Inputs
3 % f      : nonlinear function
4 % x0     : initial root estimate
5 % h      : step size for central difference formula
6 % nmax   : maximum number of iterations performed
7 % tol    : numerical tolerance used to check for root
8 % Outputs
9 % x      : one-dimensional array containing estimates of root
10
11 function x = Newton(f, x0, h, nmax, tol)
12 % Initial array of root estimates, iteration and variable stores.
13     x = x0;
14     n = 1;
15     k = 1;

```



```

16 % Set iterative loop for the function
17 while n < nmax
18     % Calculate the new root
19     xnew = x(k) - (f(x(k)))/ ((f(x(k)+h) - f(x(k)-h))/(2*h));
20     % Append the root estimate to the array
21     x = [x,xnew];
22     % Terminate function if at derivative point
23     if abs(f(xnew)) < tol
24         return
25     end
26     % Increase iteration counter and k value
27     n = n + 1;
28     k = k + 1;
29 end
30 % Warn the user the maximum number of iterations have been
    performed
31 disp('The maximum number of iterations have been performed
    without satisfying the required root finding condition');

```

Listing 5: Combined

```

1 % Nonlinear equation root finding by the combined binsection/
    Newton's method
2 % Inputs
3 % f      : nonlinear function
4 % xl, xr : initial root bracket
5 % h      : step size for central difference formula
6 % nmax   : maximum number of iterations performed
7 % tol    : numerical tolerance used to check for root
8 % Outputs
9 % x      : one-dimensional array containing estimates of root
10
11 function x = Combined(f, xl, xr, h, nmax, tol)
12 % Initial array of root estimates, iteration and variable stores.
13     x = [xl,xr];
14     xbrac = x;
15     n = 1;
16     % Calculate the starting estimate
17     xstart = xbrac(1) + (xbrac(2) - xbrac(1))/2;
18     % Append the starting value
19     x = [x,xstart];
20     k = 3;
21     % Set iterative loop for the function
22     while n < nmax
23         % Use newton method to calculate the new root estimate
24         xnew = x(k) - (f(x(k)))/ ((f(x(k)+h) - f(x(k)-h))/(2*h));
25         % Use if condition to check inside the bracket
26         if (xnew < xbrac(1)) || (xnew > xbrac(2))
27             % Use the bisection method to get a better estimate
28             xnew = xbrac(1) + (xbrac(2) - xbrac(1))/2;
29         end
30         % Append the root estimate to the array

```

```

31     x = [x,xnew];
32     % Terminate function if at derivative point
33     if abs(f(xnew)) < tol
34         return
35     % Calculate sign on the function with the new root
36     % and find the new root bracket.
37     elseif f(xnew)*f(xbrac(1))>0
38         % Reset the bracket with new LHS
39         xbrac = [xnew,xbrac(2)];
40     elseif f(xnew)*f(xbrac(2))>0
41         % Reset the bracket with new RHS
42         xbrac = [xbrac(1),xnew];
43     end
44     % Increase iteration counter and k count by one.
45     n = n + 1;
46     k = k + 1;
47 end
48 % Warn the user the maximum number of iterations have been
    performed
49 disp('The maximum number of iterations have been performed
    without satisfying the required root finding condition');

```

Listing 6: Task 1

```

1  %% Task 1 - Bisection, Secant, Regula Falsi and Newton's Methods
2  % You do NOT need to modify this script
3
4  % clear workspace
5  clear
6  clc
7
8  % Initialisation
9  f = @(x) 2*x.^2-8*x+4; % function to evaluate
10 tol = 1.0e-4;          % tolerance for asserts
11 h = 1.0e-4;            % step size for numerical estimate of
    gradient
12 x0 = 0.0;              % initial interval left
13 x1 = 2.0;              % initial interval right
14 nmax = 50;             % maximum number of iterations
15
16 % Bisection method
17 xb = Bisection(f, x0, x1, nmax, tol);
18 assert(abs(f(xb(end))) < tol)
19 disp(['Bisection converged to root at x = ' num2str(xb(end))]);
20
21 % Secant method
22 xs = Secant(f, x0, x1, nmax, tol);
23 assert(abs(f(xs(end))) < tol)
24 disp(['Secant converged to root at x = ' num2str(xs(end))]);
25
26 % Regula Falsi method and verification
27 xrf = Regulafalsi(f, x0, x1, nmax, tol);

```

```

28 assert(abs(f(xrf(end))) < tol)
29 disp(['Regula Falsi converged to root at x = ' num2str(xrf(end))
30      ']);
31 % Newton's method and verification
32 xn = Newton(f, x0, h, nmax, tol);
33 assert(abs(f(xn(end))) < tol)
34 disp(['Newton converged to root at x = ' num2str(xn(end))]);
35
36 % Combined Bisection/Newton's method and verification
37 xc = Combined(f, x0, x1, h, nmax, tol);
38 assert(abs(f(xc(end))) < tol)
39 disp(['Combined Bisection/Newton converged to root at x = '
      num2str(xc(end))]);

```

3.2 NLE Plotting

Listing 7: Task 2

```

1 %% Task 2 - Iterative Algorithm Comparison
2
3 % clear workspace
4 clear
5 clc
6
7 % Initialisation
8 tol = 1.0e-4; % tolerance for asserts
9 h = 1.0e-4; % step size for numerical estimate of gradient
10 nmax = 20; % maximum number of iterations
11
12 % functions to test algorithms on
13 f1 = @(x) x.^2 - 1; % function 1
14 f2 = @(x) (exp(x)-exp(-x))./(exp(x)+exp(-x)); % function 2
15 f3 = @(x) cos(x)+sin(x.*x)-0.5; % function 3
16
17 % initial root estimates for each function
18 % column 1: x0 for bisection, secant, regula falsi and combined
   methods
19 % column 2: x1 for bisection, secant, regula falsi and combined
   methods
20 % column 3: x0 for Newton's method
21 xint1 = ([-3.0,0.5,-3.0]);
22 xint2 = ([-5.,2.,1.1]);
23 xint3 = ([-2.0,1.5,-0.40]);
24
25 % function titles for plots
26 title1 = 'f(x)=x^2-1';
27 title2 = 'f(x)=(e^x - e^{-x})/(e^x + e^{-x})';
28 title3 = 'f(x)=cos(x)+sin(x^2)';
29

```

```

30 % set disp_func = false when you don't need to produce plot of
    functions
31 disp_func = false;
32 if disp_func
33     x = linspace(-5.,5.,1000);
34     figure(1), clf
35     subplot(3,1,1)
36     plot(x,f1(x))
37     grid on, xlabel('x'), ylabel('f(x)'), title(title1)
38     subplot(3,1,2)
39     plot(x,f2(x))
40     grid on, xlabel('x'), ylabel('f(x)'), title(title2)
41     subplot(3,1,3)
42     plot(x,f3(x))
43     grid on, xlabel('x'), ylabel('f(x)'), title(title3)
44 end
45
46
47 %% find one root for each function using bisection, secant,
    newton's and combined methods
48 % Function 1
49 xB1 = Bisection(f1, xint1(1), xint1(2), nmax, tol);
50 xS1 = Secant(f1, xint1(1), xint1(2), nmax, tol);
51 xR1 = Regulafalsi(f1, xint1(1), xint1(2), nmax, tol);
52 xN1 = Newton(f1, xint1(3), h, nmax, tol);
53 xC1 = Combined(f1, xint1(1), xint1(2), h, nmax, tol);
54
55 % Function 2
56 xB2 = Bisection(f2, xint2(1), xint2(2), nmax, tol);
57 xS2 = Secant(f2, xint2(1), xint2(2), nmax, tol);
58 xR2 = Regulafalsi(f2, xint2(1), xint2(2), nmax, tol);
59 xN2 = Newton(f2, xint2(3), h, nmax, tol);
60 xC2 = Combined(f2, xint2(1), xint2(2), h, nmax, tol);
61
62 % Function 3
63 xB3 = Bisection(f3, xint3(1), xint3(2), nmax, tol);
64 xS3 = Secant(f3, xint3(1), xint3(2), nmax, tol);
65 xR3 = Regulafalsi(f3, xint3(1), xint3(2), nmax, tol);
66 xN3 = Newton(f3, xint3(3), h, nmax, tol);
67 xC3 = Combined(f3, xint3(1), xint3(2), h, nmax, tol);
68
69 %% individual plot for each function of f(x^k) vs k for each
    method
70 % i.e. each of the three plots (one per function) will have four
    lines, one for each method called.
71 figure(1), clf
72
73 % create top plot for function 1
74 % The intial root estimates have been excluded from the
    iterations.
75 subplot(3,1,1)

```

```

76 plot(1:length(xB1)-2,f1(xB1(3:length(xB1))))
77 hold on
78 plot(1:length(xS1)-2,f1(xS1(3:length(xS1))))
79 hold on
80 plot(1:length(xN1)-1,f1(xN1(2:length(xN1))))
81 hold on
82 plot(1:length(xC1)-2,f1(xC1(3:length(xC1))))
83 legend('Bisection','Secant','Newton','Combined')
84 xlim([1,8]);
85 grid on, xlabel('k: Number of Iterations'), ylabel('f(x)'), title
    (title1)
86
87 % create middle plot for function 2
88 subplot(3,1,2)
89 plot(1:length(xB2)-2,f2(xB2(3:length(xB2))))
90 hold on
91 plot(1:length(xS2)-2,f2(xS2(3:length(xS2))))
92 hold on
93 plot(1:length(xN2)-1,f2(xN2(2:length(xN2))))
94 hold on
95 plot(1:length(xC2)-2,f2(xC2(3:length(xC2))))
96 legend('Bisection','Secant','Newton','Combined')
97 xlim([1,8]);
98 grid on, xlabel('k: Number of Iterations'), ylabel('f(x)'), title
    (title2)
99
100 % create bottom plot for function 3
101 subplot(3,1,3)
102 plot(1:length(xB3)-2,f3(xB3(3:length(xB3))))
103 hold on
104 plot(1:length(xS3)-2,f3(xS3(3:length(xS3))))
105 hold on
106 plot(1:length(xN3)-1,f3(xN3(2:length(xN3))))
107 hold on
108 plot(1:length(xC3)-2,f3(xC3(3:length(xC3))))
109 legend('Bisection','Secant','Newton','Combined')
110 grid on, xlabel('k: Number of Iterations'), ylabel('f(x)'), title
    (title3)
111 xlim([1,8]);
112
113 % Save the plot
114 savefig('Task2Plot')

```

4 Systems of Non Linear Equations

4.1 Newton Two Variable

Listing 8: Newton Two Variables

```

1  % Nonlinear equation root finding in two dimensions using Newton's
   % Method.
2  % Inputs
3  % func      : array of function handles for system of nonlinear
   % equations
4  % x0        : vector of initial root estimates for each independent
   % variable
5  % h         : step size for numerical estimate of partial
   % derivatives
6  % nmax      : maximum number of iterations performed
7  % tol       : numerical tolerance used to check for root
8  % Outputs
9  % x         : two-dimensional array (two-row matrix) containing
   % estimates of root
10
11 % Hint 1:
12 % Include the initial root estimate as the first column of x
13
14 % Hint 2:
15 % Use MATLAB in-built functionality for solving the matrix
   % equation for vector of updates, delta
16
17 % Hint 3:
18 % Check for root each iteration, continuing until the maximum
   % number of iterations has been reached
19
20 function x = Newton2Var(func, x0, h, nmax, tol)
21 % Initial array of root estimates, iteration and variable stores.
22 % Initialise a storage array
23 xstore = transpose(x0);
24 x = xstore; % Vector of the most recent root variables
25 % Set iterative loop for the function
26 for i = 1:nmax
27     % Calculate the function variables from the most recent
       % iteration.
28     f1 = func{1}(x);
29     f2 = func{2}(x);
30
31     %Calculate the derivatives for each of the points for the
       % jacobian
32     f1x1 = (func{1}(x + [h;0]) - func{1}(x - [h;0]))/(2*h);
33     f1x2 = (func{1}(x + [0;h]) - func{1}(x - [0;h]))/(2*h);
34     f2x1 = (func{2}(x + [h;0]) - func{2}(x - [h;0]))/(2*h);
35     f2x2 = (func{2}(x + [0;h]) - func{2}(x - [0;h]))/(2*h);
36
37     % Set Jacobian and f
38     f = [f1;f2];
39     J = [f1x1,f1x2;f2x1,f2x2];
40
41     % Calculate the Delta
42     del = -1*linsolve(J,f);

```

```

43
44     % Find the new xvalues
45     xnew = del + x;
46
47     % Calculate the new f values
48     fnew = [func{1}(xnew);func{2}(xnew)];
49     %Use condition criteria to cancel out of the list
50     if (abs(fnew(1))< tol) && (abs(fnew(2))< tol)
51         % Add to the storage arrays
52         xstore = [xstore,xnew];
53         x = xstore;
54         return
55     end
56     xstore = [xstore,xnew];
57     x = xnew;
58 end
59 disp('The maximum number of iterations have been performed
60     without satisfying the required root finding condition');
end

```

Listing 9: Task 3

```

1  %% Task 3 - System of Nonlinear Equations
2
3  % clear workspace
4  clear all
5  clc
6
7  % initialisation
8  tol = 1.0e-6;          % numerical tolerance
9  h = 1.0e-4;           % step size for central difference
10 nmax = 50;             % maximum number of iterations
11 x0 = [2,0];            % initial root estimate
12 func = {@f1, @f2};    % array of function handles
13
14 % set func_usage to false once you know how vector/array func
   works
15 func_usage = true;
16 if func_usage
17     f1_initial = func{1}(x0);
18     f2_initial = func{2}(x0);
19 end
20
21 % 2D Newton's method and verification
22 disp(['Newton2Var starting at point (x0,y0) = (' num2str(x0(1)), '
   ',num2str(x0(2)),')']);
23 xn = Newton2Var(func, x0, h, nmax, tol);
24 disp([xn(1,end),xn(2,end)]);
25 assert(abs(func{1}([xn(1,end),xn(2,end)])) <= tol)
26 assert(abs(func{2}([xn(1,end),xn(2,end)])) <= tol)
27 disp(['Newton2Var converged to root at (x,y) = (' num2str(xn(1,
   end)), ', ',num2str(xn(2,end)),') in ',num2str(length(xn)), '

```

```

    iterations']]);
28
29
30 % Functions to be used for testing out Newton2Var
31 function f = f1(x)
32     f = x(1)*x(1)-2*x(1)+x(2)*x(2)+2*x(2)-2*x(1)*x(2)+1;
33 end
34 function f = f2(x)
35     f = x(1)*x(1)+2*x(1)+x(2)*x(2)+2*x(2)+2*x(1)*x(2)+1;
36 end

```

4.2 Newton Two Variable Plotting

Listing 10: Task 4

```

1 %% Task 4
2
3 % clear workspace
4 clear all
5 clc
6
7 % initialisation
8 tol = 1.0e-6;          % numerical tolerance
9 h = 1.0e-4;           % step size for central difference
10 nmax = 50;            % maximum number of iterations
11 x0_p1 = [-1,-3];      % initial root estimate - point 1
12 x0_p2 = [2,-3];       % initial root estimate - point 2
13 x0_p3 = [2,0];        % initial root estimate - point 3
14 func = {@f1, @f2};    % array of function handles
15
16 % Two function two variable Newton's method for each starting
    location
17 xn_p1 = Newton2Var(func, x0_p1, h, nmax, tol);
18 xn_p2 = Newton2Var(func, x0_p2, h, nmax, tol);
19 xn_p3 = Newton2Var(func, x0_p3, h, nmax, tol);
20
21 %% start figure for algorithm visualisation
22 figure(1), clf
23 % Calculate all the values needed
24
25 % Iteration count 1
26 [~,c1] = size(xn_p1);
27 k1 = 1:c1;
28
29 % Call the first function for each iteration from the first
    starting point.
30 for i = 1:c1
31     fp11(i) = func{1}([xn_p1(1,i),xn_p1(2,i)]);
32 end
33
34 %Iteration Count 2

```



```

35 [~,c2] =size(xn_p2);
36 k2 = 1:c2;
37
38 % Call the first function for each iteration from the second
   starting point.
39 for i = 1:c2
40     fp21(i) = func{1}([xn_p2(1,i),xn_p2(2,i)]);
41 end
42
43 % Iteration count 3
44 [~,c3] =size(xn_p3);
45 k3 = 1:c3;
46
47 % Call the first function for each iteration from the third
   starting point.
48 for i = 1:c3
49     fp31(i) = func{1}([xn_p3(1,i),xn_p3(2,i)]);
50 end
51
52 % Call the second function for each iteration from the first
   starting point.
53 for i = 1:c1
54     fp12(i) = func{2}([xn_p1(1,i),xn_p1(2,i)]);
55 end
56
57 % Call the first function for each iteration from the second
   starting point.
58 for i = 1:c2
59     fp22(i) = func{2}([xn_p2(1,i),xn_p2(2,i)]);
60 end
61
62 % Call the first function for each iteration from the third
   starting point.
63 for i = 1:c3
64     fp32(i) = func{2}([xn_p3(1,i),xn_p3(2,i)]);
65 end
66
67
68 % Create all the labels to plot with
69 f1title = 'f_{1}(x^{(k)},y^{(k)}) Converging from the Intial Root
   Estimate vs k';
70 f2title = 'f_{2}(x^{(k)},y^{(k)}) Converging from the Intial Root
   Estimate vs k';
71 surftitle = '3D Plot of Surfaces f_{1}(x_1,x_2) & f_{2}(x_{1},x_
   {2}) with Individual Lines (x_1,x_2,f_1) & (x_1,x_2,f_2)';
72 f1xlabel = 'Number of Iterations (k)';
73 f1ylabel = 'f_{1}(x^{(k)},y^{(k)})';
74 f2xlabel = 'Number of Iterations (k)';
75 f2ylabel = 'f_{2}(x^{(k)},y^{(k)})';
76 surfxlabel = 'x_1';
77 surfylabel = 'x_2';

```

```

78 surfzlabel = 'f_1/f_2';
79
80 % Create axis labels
81 f1p1 = 'f_1^{p1}:(-1,-3)';
82 f1p2 = 'f_1^{p2}:(2,-3)';
83 f1p3 = 'f_1^{p3}:(2,0)';
84 f2p1 = 'f_2^{p1}:(-1,-3)';
85 f2p2 = 'f_2^{p2}:(2,-3)';
86 f2p3 = 'f_2^{p3}:(2,0)';
87 f1legend = {f1p1,f1p2,f1p3};
88 f2legend = {f2p1,f2p2,f2p3};
89 surflegend = {f1p1,f2p1,f1p2,f2p2,f1p3,f2p3};
90
91 % create top left plot for function 1
92 subplot(2,2,1)
93 hold on
94 plot(k1,fp11,'b','Linewidth',2)
95 hold on
96 plot(k2,fp21,'r','Linewidth',2)
97 hold on
98 plot(k3,fp31,'g--','Linewidth',2)
99 ylabel('Function 1 values')
100 xlabel('Number of Iterations (k)')
101 title(f1title)
102 legend(f1legend)
103 xlabel(f1xlabel)
104 ylabel(f1ylabel)
105 xlim([1,14]);
106
107 % create bottom left plot for function 2
108 subplot(2,2,3)
109 hold on
110 plot(k1,fp12,'b','Linewidth',2)
111 hold on
112 plot(k2,fp22,'r','Linewidth',2)
113 hold on
114 plot(k3,fp32,'g--','Linewidth',2)
115 ylabel('Function 2 values')
116 xlabel('Number of Iterations (k)')
117 title(f2title)
118 legend(f2legend)
119 xlabel(f2xlabel)
120 ylabel(f2ylabel)
121 xlim([1,14]);
122
123 % create right plot for 3d visualisation of 2d newton's method
124 subplot(2,2,[2 4])
125 % Plot both the functions
126 [X,Y] = meshgrid(-1:0.1:2,-3:0.1:2);
127 Z1 = X.*X-2.*X+Y.*Y+2.*Y-2.*X.*Y+1;
128 Z2 = X.*X+2.*X+Y.*Y+2.*Y+2.*X.*Y+1;

```

```

129 surf(X,Y,Z1);
130 hold on;
131 surf(X,Y,Z2);
132
133 % Improve plotting
134 alpha 0.4; % Make more transparent
135 rotate3d on; % Automatic switch on rotate feature
136 shading interp; % Change shading
137 view(255,25); % Specify view found by trial and error.
138
139 % Plot the lines on the surface
140 hold on
141 ob1 = plot3(xn_p1(1,:),xn_p1(2,:),fp11,':r.','Linewidth',2,'
    Markersize',20,'MarkerEdgeColor','k');
142 hold on
143 ob3 = plot3(xn_p2(1,:),xn_p2(2,:),fp21,':b.','Linewidth',2,'
    Markersize',20,'MarkerEdgeColor','k');
144 hold on
145 ob5 = plot3(xn_p3(1,:),xn_p3(2,:),fp31,':g.','Linewidth',2,'
    Markersize',20,'MarkerEdgeColor','k');
146 hold on
147 ob2 = plot3(xn_p1(1,:),xn_p1(2,:),fp12,':r+', 'Linewidth',2,'
    Markersize',10,'MarkerEdgeColor','k');
148 hold on
149 ob4 = plot3(xn_p2(1,:),xn_p2(2,:),fp22,':b+', 'Linewidth',2,'
    Markersize',10,'MarkerEdgeColor','k');
150 hold on
151 ob6 = plot3(xn_p3(1,:),xn_p3(2,:),fp32,':g+', 'Linewidth',2,'
    Markersize',10,'MarkerEdgeColor','k');
152 ob = [ob1,ob2,ob3,ob4,ob5,ob6];
153
154 % Plot labels
155 legend(ob,surflegend,'Location','northeast','NumColumns',3)
156 title(surftitle);
157 xlabel(surfxlabel);
158 ylabel(surfylabel);
159 zlabel(surfzlabel);
160 grid on;
161
162 % Functions to be used for testing out Newton2Var
163 function f = f1(x)
164     f = x(1)*x(1)-2*x(1)+x(2)*x(2)+2*x(2)-2*x(1)*x(2)+1;
165 end
166 function f = f2(x)
167     f = x(1)*x(1)+2*x(1)+x(2)*x(2)+2*x(2)+2*x(1)*x(2)+1;
168 end

```

4.3 Newton Multiple Variable

Listing 11: Newton Multiple Variables

```

1  % Nonlinear equation root finding in n dimensions using Newton's
    Method.
2  % Inputs
3  % n      : number of dimensions for Newton's method
4  % func   : array of function handles for system of nonlinear
    equations
5  % x0     : vector of initial root estimates for each independent
    variable
6  % h      : step size for numerical estimate of partial
    derivatives
7  % nmax   : maximum number of iterations performed
8  % tol    : numerical tolerance used to check for root
9  % Outputs
10 % x      : array (n-row matrix) containing estimates of root
11
12 % Hint 1:
13 % Include the initial root estimate as the first column of x
14
15 % Hint 2:
16 % Use MATLAB in-built functionality for solving the matrix
    equation for vector of updates, delta
17
18 % Hint 3:
19 % Check for root each iteration, continuing until the maximum
    number of iterations has been reached
20
21 function x = NewtonMultiVar(n, func, x0, h, nmax, tol)
22 % Initial array of root estimates, iteration and variable stores.
23 % Initialise a storage array
24 xstore = transpose(x0);
25 x = xstore; % Vector of the most recent root variables
26 n = 1;
27 % Set iterative loop for the function
28 while n < nmax
29     % Calculate the current root estimate, used a nested for
        loop.
30     for i = 1:length(func)
31         f(i,1) = func{i}(x); % Vector of variables passed
            into the function call
32     end
33     % Initialise the size of the jacobian
34     jacob = zeros(length(func):length(x));
35     % Calculate the jacobian
36     for i = 1:length(func)
37         for j = 1: length(x)
38             % Create two small steps for the x values
39             x(j) = x(j) + h;
40             % Find the first part of the derivative
                calculation.
41             func1 = func{i}(x);

```

```

42         % Do the second part of the derivative
           calculation.
43         x(j) = x(j) - 2.*h;
44         % Find the first part of the derivative
           calculation.
45         func2 = func{i}(x);
46         % Calculate the jacobian
47         jacob(i,j) = ((func1 - func2)./(2.*h));
48         % Correct the x value
49         x(j) = x(j) + h;
50     end
51 end
52 % Inverse the jacobian and get del
53 del = -1.*(jacob\f);
54 % Perform the necessary exit conditions
55 % New del
56 xnew = del + x;
57 % Recalculate f with xnew values
58 for i = 1:length(func)
59     fnew(i,1) = func{i}(transpose(xnew)); % Vector of
           variables passed into the function call
60 end
61 % Test for both convergence and function call close to
           zero.
62 if (prod((abs(fnew)<= tol)) == 1) && (prod((abs(del)<=
           tol))== 1)
63     % Add to the store arrays
64     xstore = [xstore,xnew];
65     x = xstore;
66     return
67 end
68 xstore = [xstore,xnew];
69 x = xnew;
70 % Increase iteration counter
71 n = n + 1;
72 end
73 % Warn the user the maximum number of iterations have been
           performed
74 disp('The maximum number of iterations have been performed
           without satisfying the required root finding condition');
75 end

```