Assignment 2

Colin Turner: 250956100

MME 9621: Computational Methods in Mechanical Engineering

Dr. Mohammad Z. Hossain

March 1, 2024

Q1)

The Figure below describes how the matrix was arranged to ensure the diagonal remains non-zero thus allowing for all methods to yield a solution. This process involved simple row operations.

1	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21	b
2	-1	0	-0.7071	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	-1	-0.7071	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.7071	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	-1	0	0	0.7071	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	-1	0	-0.7071	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	-1	-0.7071	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.7071	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	-1	0	0	0.7071	0	1	0.1961	0.7071	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	-1	0	-0.7071	0	0	0.9806	0.7071	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	-1	-0.7071	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.7071	1	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.9806	0	0	0	0.9806	0.7433	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.7071	0	-1	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.7071	-1	0	0	0	1	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.1961	0	0	1	0.1961	0.669	0	0	8000
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.9806	0	0	0	5000
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.669	0	-1	0
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.7433	-1	0	-5000
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-0.1961	0	0	1	8660.25404

Assignment_2.m

/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```matlab
%% Q1  Comment on Accuracy, comp cost, Plot Struct of Coefficient Matrix
clc;clear all;format long e;
[Aug1, Text]=xlsread('DataFile_Assn2.xlsx');
A=Aug1(:,1:21);
b=Aug1(:,22);

tic;
max_iter=1000;TOL=1e-5;
[xcJ, iterJ]=Jacobi(A,b,max_iter,TOL);
tJ=toc;

tic;
max_iter=1000;TOL=1e-5;
[xcG, iterG]=GaussSeidel(A,b,max_iter,TOL);
tG =toc;

tic;
max_iter=1000;TOL=1e-5;
[xcS0, iterS0]=SOR(A,b,0.5,max_iter,TOL);
tS0=toc;

tic;
max_iter=1000;TOL=1e-5;
[xcS1, iterS1]=SOR(A,b,1.5,max_iter,TOL);
tS1=toc;

tic;
xc=A\b;
tB=toc;

% Residuals
residual_J = norm(b-A*xcJ);
residual_G = norm(b-A*xcG);
residual_S0 = norm(b-A*xcS0);
residual_S1 = norm(b-A*xcS1);
residual_backslash = norm(b-A*xc);

fprintf("Jacobi Method\t\t Gauss Seidel Method\t SOR w < 1 Method\t SOR w > 1 Method\t BackSlash Method\n");
for i = 1:1:21
    fomatSpec='F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\n';
    fprintf(fomatSpec,i,xcJ(i),i,xcG(i),i,xcS0(i),i,xcS1(i),i,xc(i));
end
```

Assignment_2.m

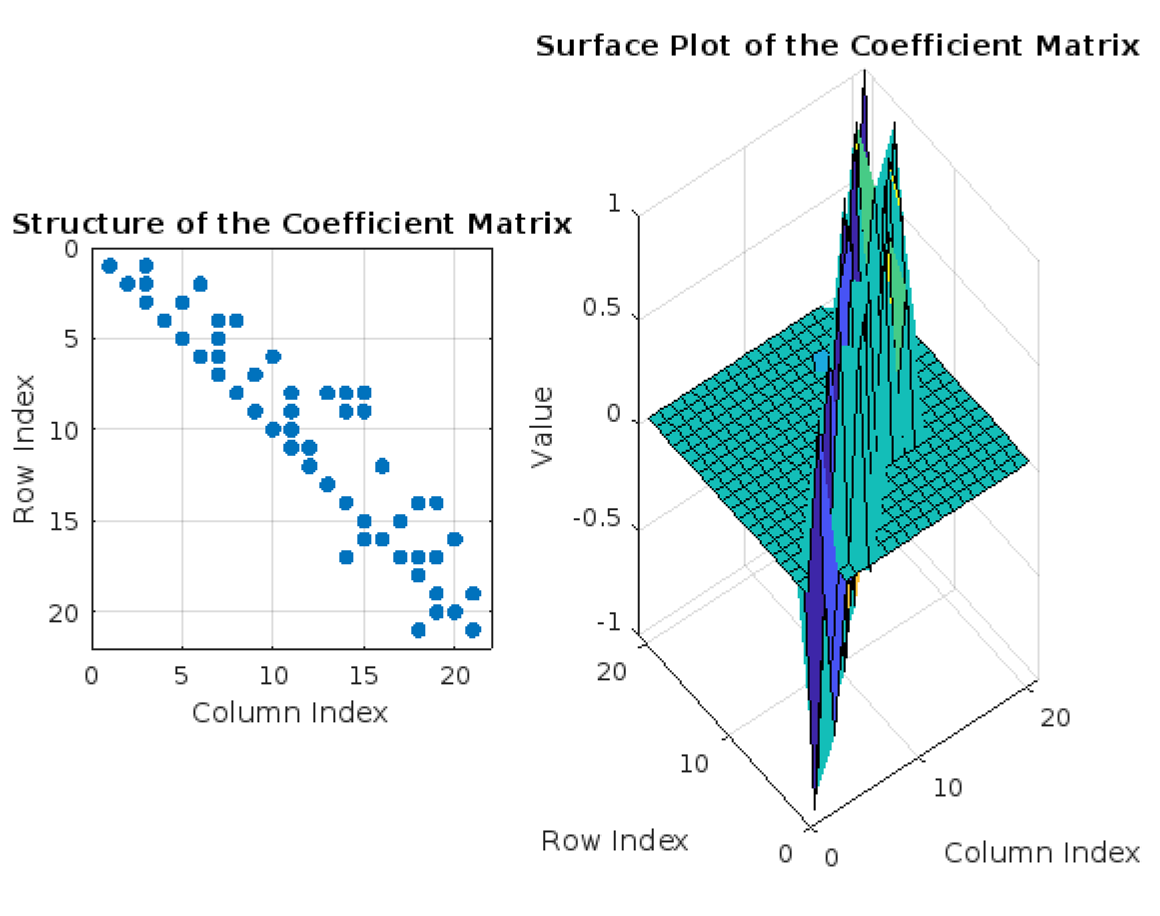
/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```
formatSpec1='time:\nt = %fs\t\t t = %fs\t\t t = %fs\t\t t = %fs\t\t t = %fs\n';
fprintf(formatSpec1,tJ,tG,tS0,tS1,tB);
formatSpec2='Iterations:\niter = %d\t\t iter = %d\t\t iter = %d\t\t iter = %d\n';
fprintf(formatSpec2,iterJ,iterG,iterS0,iterS1);
formatSpec3='Residuals:\nRes = %e\t Res = %e\t Res = %e\t Res = %e\t Res = %e\n';
fprintf(formatSpec3,residual_J,residual_G,residual_S0,residual_S1,residual_backslash);

figure;

% Sparsity patern of matrix representing non zero elements
subplot(1, 2, 1);
spy(A);
title('Structure of the Coefficient Matrix');
xlabel('Column Index');
ylabel('Row Index');
grid on;

% 3d view of plot showing magnitude difference in coefficients
subplot(1, 2, 2);
surf(A);
title('Surface Plot of the Coefficient Matrix');
xlabel('Column Index');
ylabel('Row Index');
zlabel('Value');
```

The following figure shows the output terminal of the code. The force values for each, computation time, iteration count and residuals of each method can be observed. Due to the row operations performed on the matrix, all methods yielded results. From the results shown below we can rank the speed of each method from fastest to slowest: Backslash, SOR w>1, Jacobi, Gauss Seidel and finally Sor w<1. For this specific situation the highly optimized MATLAB backslash method was the fastest which should be the case for simple problems such as this. The efficiency of SOR method can be further increase with optimized selection of w. analysis of the residuals determine that Jacobi method yielded the lowest error with Gauss Seidel following a close second. The SOR methods yield the highest residuals showing poor accuracy, however accuracy of these methods can be improved through an optimized selection of w. Additional lowering the tolerance can increase the accuracy of all methods except for Backslash. Finally it can be observed that iteration count does not necessarily determine computation cost as Jacobi method has a higher iteration count than Gauss Seidel however it has a loser computation time. For this specific problem backslash method is the best option as it provides good speed and accuracy.

```
Command Window
Jacobi Method           Gauss Seidel Method     SOR w < 1 Method        SOR w > 1 Method        BackSlash Method
F1 =  10000.00000       F1 =  10000.00000       F1 =   9999.99999       F1 =   9999.99999       F1 =  10000.00000
F2 =  51246.78774       F2 =  51246.78774       F2 =  51246.78773       F2 =  51246.78773       F2 =  51246.78774
F3 = -14142.27125       F3 = -14142.27125       F3 = -14142.27125       F3 = -14142.27125       F3 = -14142.27125
F4 = -57907.04178       F4 = -57907.04178       F4 = -57907.04178       F4 = -57907.04178       F4 = -57907.04178
F5 =  10000.00000       F5 =  10000.00000       F5 =  10000.00000       F5 =  10000.00000       F5 =  10000.00000
F6 =  41246.78774       F6 =  41246.78774       F6 =  41246.78774       F6 =  41246.78774       F6 =  41246.78774
F7 = -14142.27125       F7 = -14142.27125       F7 = -14142.27125       F7 = -14142.27125       F7 = -14142.27125
F8 = -47907.04178       F8 = -47907.04178       F8 = -47907.04178       F8 = -47907.04178       F8 = -47907.04178
F9 =  10000.00000       F9 =  10000.00000       F9 =  10000.00000       F9 =  10000.00000       F9 =  10000.00000
F10 =  31246.78774      F10 =  31246.78774      F10 =  31246.78774      F10 =  31246.78774      F10 =  31246.78774
F11 = -44190.05479      F11 = -44190.05479      F11 = -44190.05479      F11 = -44190.05479      F11 = -44190.05479
F12 =  31246.78774      F12 =  31246.78774      F12 =  31246.78774      F12 =  31246.78774      F12 =  31246.78774
F13 =       0.00000     F13 =       0.00000     F13 =       0.00000     F13 =       0.00000     F13 =       0.00000
F14 =  -5846.44194      F14 =  -5846.44194      F14 =  -5846.44194      F14 =  -5846.44194      F14 =  -5846.44194
F15 = -21939.98978      F15 = -21939.98978      F15 = -21939.98978      F15 = -21939.98978      F15 = -21939.98978
F16 =  31246.78774      F16 =  31246.78774      F16 =  31246.78774      F16 =  31246.78774      F16 =  31246.78774
F17 =  15513.76677      F17 =  15513.76677      F17 =  15513.76677      F17 =  15513.76677      F17 =  15513.76677
F18 =   5098.91903      F18 =   5098.91903      F18 =   5098.91903      F18 =   5098.91903      F18 =   5098.91903
F19 = -14439.68918      F19 = -14439.68918      F19 = -14439.68918      F19 = -14439.68918      F19 = -14439.68918
F20 =  15733.02097      F20 =  15733.02097      F20 =  15733.02097      F20 =  15733.02097      F20 =  15733.02097
F21 =   9660.15206      F21 =   9660.15206      F21 =   9660.15206      F21 =   9660.15206      F21 =   9660.15206
time:
t = 0.003457s           t = 0.003779s           t = 0.004400s           t = 0.001811s           t = 0.000108s
Iterations:
iter = 14               iter = 11               iter = 70               iter = 95
Residuals:
Res = 6.003457e-12      Res = 7.368257e-12      Res = 1.190731e-05      Res = 9.428088e-06      Res = 9.229625e-12
>>
```

The following figure depicts the structure of the coefficient matrix on the left. With blue dots representing non-zero elements. The right side of the figure depicts the surface plot of the matrix, giving a representation of the magnitude of the non-zero elements.



Q2)

The Figure below describes how the formula was derived and how the variables were assigned.

2) $b = h_p dx^2 \quad a = 2+b \quad c = \sigma dx^2$

$d = bT_i + cT_i^4 \quad e = T_a + d \quad f = T_b + d$

$$\boxed{\begin{array}{ll} 1: & aT_1 + cT_1^4 - T_2 - e = 0 \\ 2: & -T_1 + aT_2 + cT_2^4 - T_3 - d = 0 \\ 3: & -T_2 + aT_3 + cT_3^4 - T_4 - d = 0 \\ 4: & -T_3 + aT_4 + cT_4^4 - f = 0 \end{array}}$$

Assignment_2.m

/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```
%% Q2
clc; clear all; format long e;
% Given parameters
L=10;Hp=0.05;sig=2.7*10^-9;Ti=200;Ta=300;Tb=400;dx=2;
b=Hp*dx^2;a=2+b;c=sig*dx^2;d=b*Ti+c*Ti^4;e=Ta+d;f=Tb+d;

% Initial guesses
Guess1=[2; 4; 6; 8]; Guess2=[2.1; 4.1; 6.1; 8.1];

% Tolerance and maximum iterations for the solvers
TOL = 0.5e-3; max_iter = 100;

% System of non linear eqn
Fn=@(x,a,b,c,d,f,e) [...
    a.*x(1)+c.*x(1).^4-x(2)-e;...
    -x(1)+a.*x(2)+c.*x(2).^4-x(3)-d;...
    -x(2)+a.*x(3)+c.*x(3).^4-x(4)-d;...
    -x(3)+a.*x(4)+c.*x(4).^4-f];
options=optimset('display','iter');

% Compute the symbolic Jacobian matrix
syms x1 x2 x3 x4;
Fs = [a*x1 + c*x1^4 - x2 - e;
      -x1 + a*x2 + c*x2^4 - x3 - d;
      -x2 + a*x3 + c*x3^4 - x4 - d;
      -x3 + a*x4 + c*x4^4 - f];
Xs = [x1; x2; x3; x4];
DFs = jacobian(Fs, Xs);

% Multivariate Newton
tic;
[xcN, iterN] = multivariateNewton_fs(Fn, a, b, c, d, f, e, @Jac_fs, DFs, Xs, Guess1, TOL, max_iter);
tN=toc;

% Jacobian is assumed to be Identity matrix
A= eye(4);

% Broyden Method
tic
[xcB,iterB]=BroydenMethod1(Fn,a, b, c, d, f, e,Guess1,Guess2,A,TOL,max_iter);
tB=toc;
```

Assignment_2.m × +

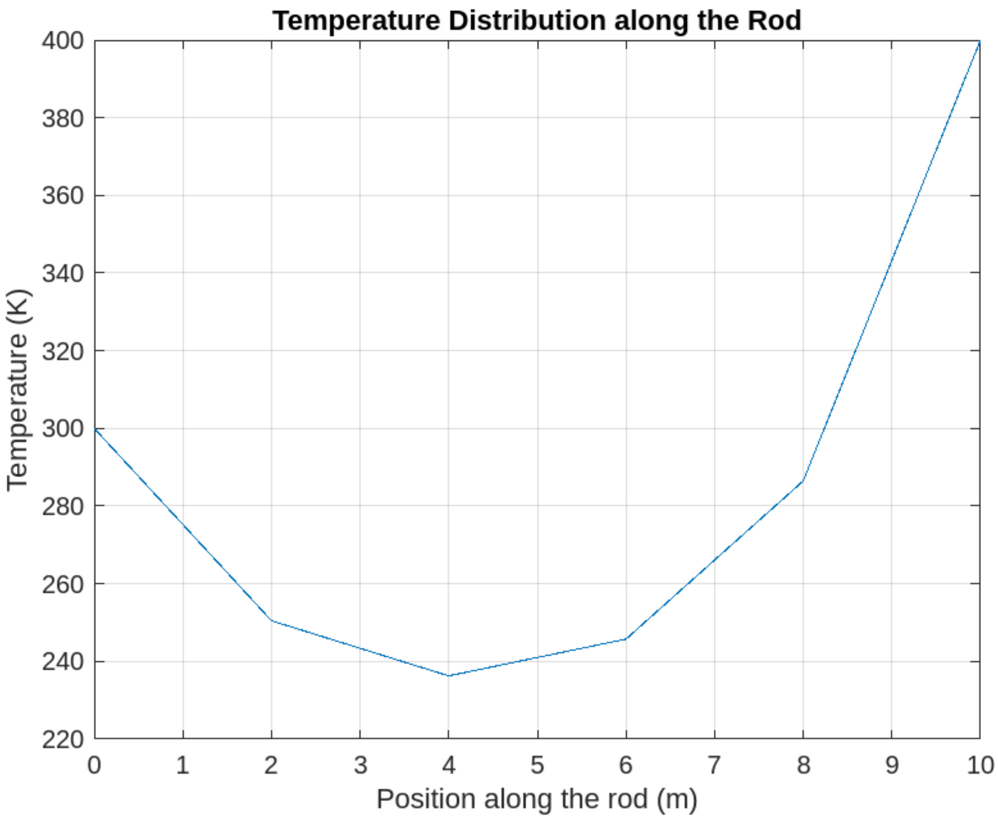
/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```matlab

% fSolve Method
tic
[xcF]=fsolve(Fn,Guess1,options,a,b,c,d,f,e);
tF=toc;

% Residuals
residual_N = norm(Fn(xcN, a, b, c, d, f, e));
residual_B = norm(Fn(xcB, a, b, c, d, f, e));
residual_F = norm(Fn(xcF, a, b, c, d, f, e));

%Display the final solutions, comp times, iterations & Residuals
fprintf("Multi Newton Method\t Broyden Method\t\t fSolve\n");
for i = 1:1:4
    fomatSpec='F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\n';
    fprintf(fomatSpec,i,xcN(i),i,xcB(i),i,xcF(i));
end

formatSpec1='time:\nt = %fs\t\t t = %fs\t\t t = %fs\n';
fprintf(formatSpec1,tN,tB,tF);
formatSpec2='Iterations:\niter = %d\t\t iter = %d\t\t iter = %d\n';
fprintf(formatSpec2,iterN,iterB,12);
formatSpec3='Residuals:\nRes = %e\t Res = %e\t Res = %e\n';
fprintf(formatSpec3,residual_N,residual_B,residual_F);

% Temperature distribution plot
x = 0:2:10;
Ttot = [Ta; xcF; Tb];
plot(x, Ttot);
xlabel('Position along the rod (m)');
ylabel('Temperature (K)');
title('Temperature Distribution along the Rod');
grid on;
```

The following figure depicts the output attained from the provided code. All methods yielded very similar results. Through analysis of the results it can be determined that for this problem Broyden method produced the quickest results with fSolve in second and finally Multivariate Newtons method. Again it can be seen that iteration count does not independently dictate computation cost as broyden method has the highest iteration count with the lost computation time. Through analysis of the residuals it can be seen that Multivariate Newtons method produced the most accurate results with fSolve following close behind. Broyden method produced extremely inaccurate results when compared with the other two options. For this specific problem fSolve is the best method as it falls effectively in the middle ground with good accuracy and speed.

```
Command Window

                                                Norm of      First-order   Trust-region
 Iteration  Func-count     ||f(x)||^2           step         optimality    radius
     0          5           332329                                  924             1
     1         10           329797                 1                919             1
     2         15           323536               2.5                906           2.5
     3         20           308306              6.25                873          6.25
     4         25           272873            15.625                790          15.6
     5         30           200621           39.0625                587          39.1
     6         35           118104           97.6562                163          97.7
     7         40          60970.8           244.141                465           244
     8         45          6103.44           272.944                115           610
     9         50          46.9104           50.8069               9.63           610
    10         55       0.00406647           5.26715             0.0896           610
    11         60      3.14437e-11         0.0496314           7.79e-06           610
    12         65      4.60442e-26        4.3542e-06           5.58e-13           610

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>
Multi Newton Method      Broyden Method          fSolve
F1 =    250.48271        F1 =    250.48271       F1 =    250.48271
F2 =    236.29623        F2 =    236.29623       F2 =    236.29623
F3 =    245.75960        F3 =    245.75959       F3 =    245.75960
F4 =    286.49211        F4 =    286.49211       F4 =    286.49211
time:
t = 0.159728s            t = 0.008422s           t = 0.086585s
Iterations:
iter = 6                 iter = 14               iter = 12
Residuals:
Res = 1.302446e-13       Res = 6.512410e-06      Res = 2.145792e-13
>>
```

The following figure depicts the temperature distribution in the rod.



Q3)

The Figure below describes how the formula was derived and how the variables were assigned.

3)

$a = \frac{k_1}{m_1} \quad b = \frac{k_2}{m_1} \quad c = \frac{k_2}{m_2} \quad d = \frac{k_3}{m_2} \quad e = \frac{k_3}{m_3}$

$$x_1'' = a x_1 + b x_2 - b x_1$$
$$x_2'' = c x_1 - c x_2 + d x_3 - d x_2$$
$$x_3'' = e x_2 - e x_3$$

Let
$x_1 = y_1$
$x_1'' = y_2$
$x_2 = y_3$
$x_2'' = y_4$
$x_3 = y_5$
$x_3'' = y_6$

1: $y_1' = y_2$
2: $y_2' = a y_1 + b y_3 - b y_1$
3: $y_3' = y_4$
4: $y_4' = c y_1 - c y_3 + d y_5 - d y_3$
5: $y_5' = y_6$
6: $y_6' = e y_3 - e y_5$

ABS Error = |RK4 - ODE45|

Assignment_2.m × +

/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```matlab
%% Q3
clc; clear all; close all; format long e;
% Given parameters
m=[12000 10000 8000];k=[3000 2400 1800];
a = k(1)/m(1); b = k(2)/m(1); c = k(2)/m(2);
d = k(3)/m(2); e = k(3)/m(3);

% RK4 parameters
h = 0.1; t0 = 0; tn = 20;
NStep = abs(tn - t0) / h;

% Initial conditions
Y0 = [0; 1; 0; 0; 0; 0];

% RK4 method
tic;
[t_rk4, Y_rk4] = VectorRK4(@ode_function, t0, Y0, NStep, h, a, b, c, d, e);
trk4 = toc;
% ode45 solver
tic;
[t_ode45, Y_ode45] = ode45(@(t, Y) ode_function(t, Y, a, b, c, d, e), [t0, tn], Y0);
tode45 = toc;

% Extract positions and velocities
x1_rk4 = Y_rk4(:, 1);   x1_ode45 = Y_ode45(:, 1);
v1_rk4 = Y_rk4(:, 2);   v1_ode45 = Y_ode45(:, 2);
x2_rk4 = Y_rk4(:, 3);   x2_ode45 = Y_ode45(:, 3);
v2_rk4 = Y_rk4(:, 4);   v2_ode45 = Y_ode45(:, 4);
x3_rk4 = Y_rk4(:, 5);   x3_ode45 = Y_ode45(:, 5);
v3_rk4 = Y_rk4(:, 6);   v3_ode45 = Y_ode45(:, 6);

% Plotting Displacement
figure;
subplot(2, 1, 1);
plot(t_rk4, [x1_rk4, x2_rk4, x3_rk4], 'LineWidth', 1.5);
hold on;
plot(t_ode45, [x1_ode45, x2_ode45, x3_ode45], '--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Displacement');
title('Displacement versus Time');
legend('x1 (RK4)', 'x2 (RK4)', 'x3 (RK4)', 'x1 (ode45)', 'x2 (ode45)', 'x3 (ode45)');
grid on;
```

Assignment_2.m × +

/MATLAB Drive/MME 9621/Assignment_2/Assignment_2.m

```matlab

    % Plotting Velocity
    subplot(2, 1, 2);
    plot(t_rk4, [v1_rk4, v2_rk4, v3_rk4], 'LineWidth', 1.5);
    hold on;
    plot(t_ode45, [v1_ode45, v2_ode45, v3_ode45], '--', 'LineWidth', 1.5);
    xlabel('Time (s)');
    ylabel('Velocity');
    title('Velocity versus Time');
    legend('v1 (RK4)', 'v2 (RK4)', 'v3 (RK4)', 'v1 (ode45)', 'v2 (ode45)', 'v3 (ode45)');
    grid on;
    hold off;

    % Plot the 3D phase plane of displacements
    figure;
    plot3(x1_rk4, x2_rk4, x3_rk4, 'LineWidth', 1.5);
    hold on;
    plot3(x1_ode45, x2_ode45, x3_ode45, '--', 'LineWidth', 1.5);
    legend('RK4','ode45')
    xlabel('x1');
    ylabel('x2');
    zlabel('x3');
    title('3D Phase Plane of Displacements');

    %Comp time for each process
    fprintf("Computation time:\n");
    fomatSpec='Vectorized RK4:%fs\tODE45 solver:%fs\n';
    fprintf(fomatSpec,trk4,tode45);

    % Calculate the absolute error for displacement components
    common_length = min(length(x1_rk4), length(x1_ode45));
    abs_error_x1 = abs(x1_rk4(1:common_length) - x1_ode45(1:common_length));
    abs_error_x2 = abs(x2_rk4(1:common_length) - x2_ode45(1:common_length));
    abs_error_x3 = abs(x3_rk4(1:common_length) - x3_ode45(1:common_length));
    % Calculate the absolute error for velocity components
    common_length_v = min(length(v1_rk4), length(v1_ode45));
    abs_error_v1 = abs(v1_rk4(1:common_length_v) - v1_ode45(1:common_length_v));
    abs_error_v2 = abs(v2_rk4(1:common_length_v) - v2_ode45(1:common_length_v));
    abs_error_v3 = abs(v3_rk4(1:common_length_v) - v3_ode45(1:common_length_v));

```

```
% Display the maximum absolute errors for each displacement component
fprintf('Maximum Absolute Error for x1: %e\n', max(abs_error_x1));
fprintf('Maximum Absolute Error for x2: %e\n', max(abs_error_x2));
fprintf('Maximum Absolute Error for x3: %e\n', max(abs_error_x3));
% Display the maximum absolute errors for each velocity component
fprintf('Maximum Absolute Error for v1: %e\n', max(abs_error_v1));
fprintf('Maximum Absolute Error for v2: %e\n', max(abs_error_v2));
fprintf('Maximum Absolute Error for v3: %e\n', max(abs_error_v3));
```

Assignment_2.m × ode_function.m ×

/MATLAB Drive/MME 9621/Assignment_2F/ode_function.m

```
function dydt = ode_function(t, Y, a, b, c, d, e)
    % Extract variables
    x1 = Y(1);
    v1 = Y(2);
    x2 = Y(3);
    v2 = Y(4);
    x3 = Y(5);
    v3 = Y(6);

    % Compute derivatives
    dx1dt = v1;
    dv1dt = -a .* x1 + b .* (x2 - x1);
    dx2dt = v2;
    dv2dt = c .* (x1 - x2) + d .* (x3 - x2);
    dx3dt = v3;
    dv3dt = e .* (x2 - x3);

    % Return the derivatives
    dydt = [dx1dt; dv1dt; dx2dt; dv2dt; dx3dt; dv3dt];
end
```

Assignment_2.m × VectorRK4.m ×

/MATLAB Drive/MME 9621/Assignment_2F/VectorRK4.m

```matlab
function [t_rk4, Y_rk4] = VectorRK4(ode_function, t0, Y0, NStep, h, a, b, c, d, e)
    % Pre-allocation
    t_rk4 = zeros(NStep, 1);
    Y_rk4 = zeros(NStep, length(Y0));

    % to store results of y1 and y2 in a singe array
    Y_rk4(1, :) = Y0;

    % initial condition
    t_rk4(1) = t0;

    for k = 1:NStep
        s1 = ode_function(t_rk4(k), Y_rk4(k, :), a, b, c, d, e);
        s2 = ode_function(t_rk4(k) + h/2, Y_rk4(k, :) + h/2 .* s1', a, b, c, d, e);
        s3 = ode_function(t_rk4(k) + h/2, Y_rk4(k, :) + h/2 .* s2', a, b, c, d, e);
        s4 = ode_function(t_rk4(k) + h, Y_rk4(k, :) + h .* s3', a, b, c, d, e);
        Y_rk4(k + 1, :) = Y_rk4(k, :) + h * (s1/6 + s2/3 + s3/3 + s4/6)';
        t_rk4(k + 1) = t_rk4(k) + h;
    end
end
```

The Following images depict the output terminal with a step size of 0.1 and 2 respectively.

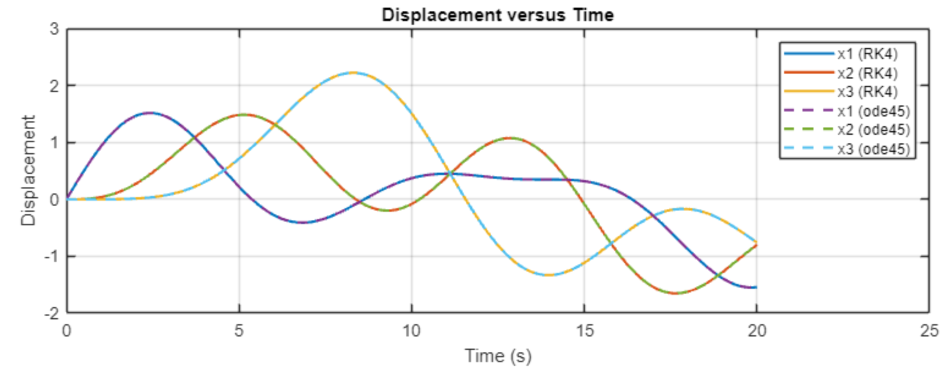
Command Window

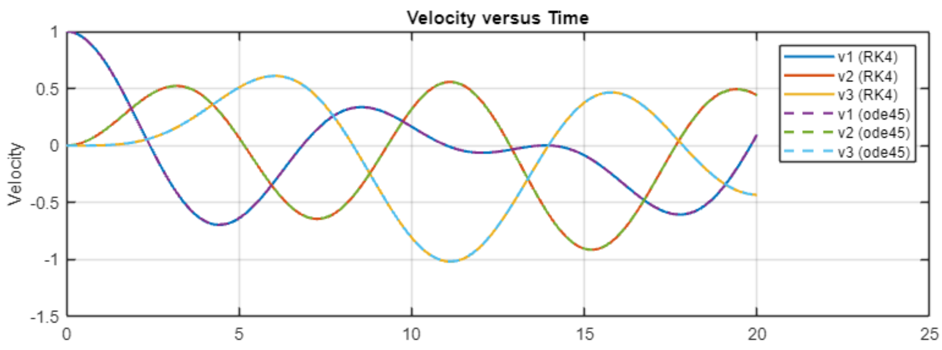
```
Computation time:
Vectorized RK4:0.013048s        ODE45 solver:0.052169s
Maximum Absolute Error for x1: 1.937102e+00
Maximum Absolute Error for x2: 2.319822e+00
Maximum Absolute Error for x3: 3.328863e+00
Maximum Absolute Error for v1: 1.115761e+00
Maximum Absolute Error for v2: 1.385842e+00
Maximum Absolute Error for v3: 1.452191e+00
>>
```

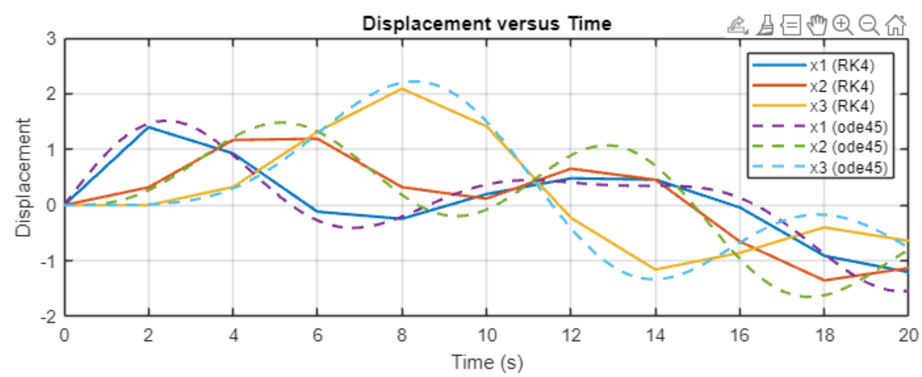
Command Window

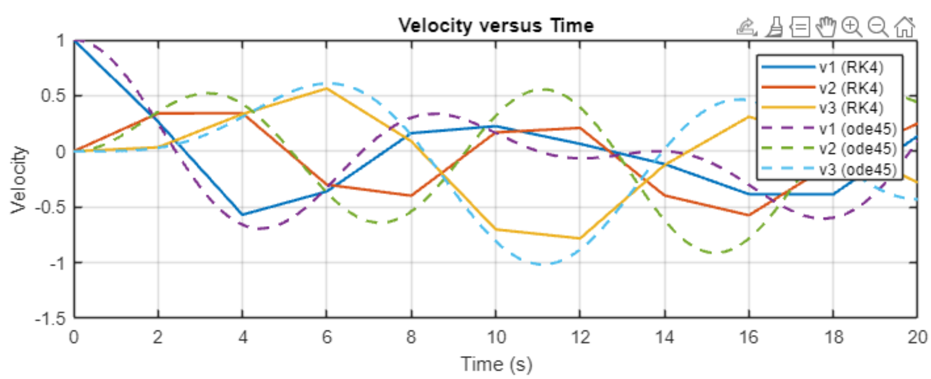
```
Computation time:
Vectorized RK4:0.015048s        ODE45 solver:0.052336s
Maximum Absolute Error for x1: 1.399950e+00
Maximum Absolute Error for x2: 1.352237e+00
Maximum Absolute Error for x3: 2.092473e+00
Maximum Absolute Error for v1: 1.569700e+00
Maximum Absolute Error for v2: 5.745463e-01
Maximum Absolute Error for v3: 7.821917e-01
>>
```

The Following images depict displacement & velocity VS Time with a step size of 0.1 and 2 respectively.

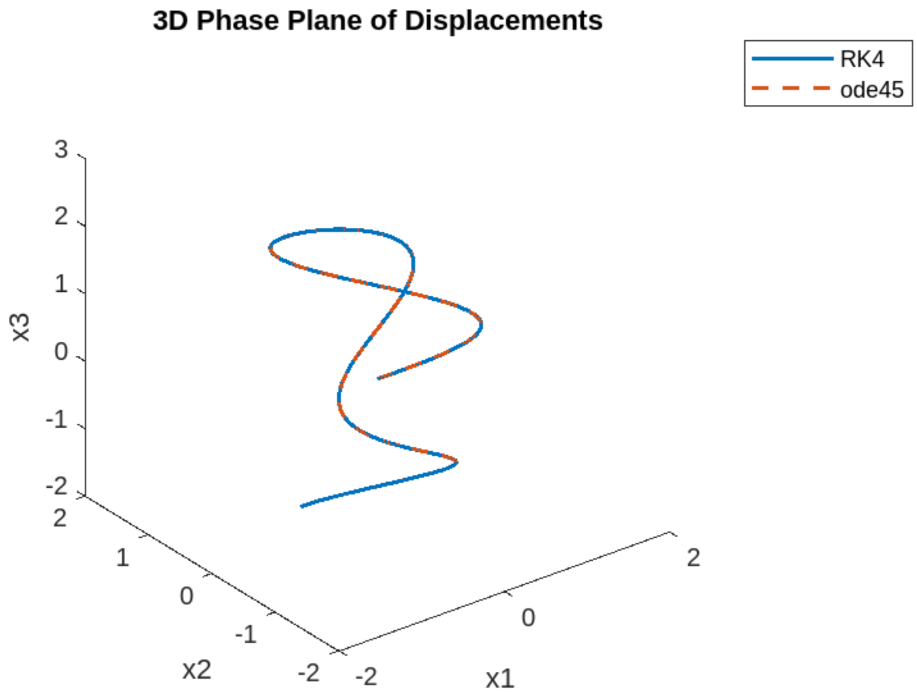


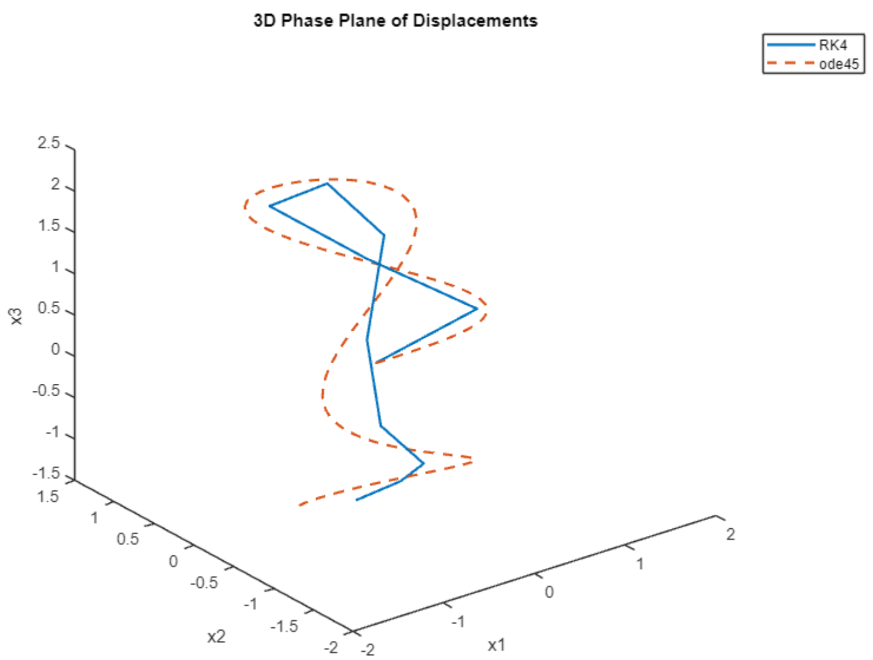






The Following images depict the 3D phase plane of displacements with a step size of 0.1 and 2 respectively.





Through analysis of the 4 figures above and the 2 output terminals, it can be observed that increasing the step size decreases the accuracy of the results and decreases the computational cost for vectorized RK4 Method. Due to the nature of ODEs it can be difficult to obtain correct error readings, this paper utilizes absolute error. Changing the step size substantially increases the absolute error in the velocities however it deceases the absolute error in the displacements. I am unable to explain the reasoning behind this. The change in step size has no effect on the ode45 method as MATLAB utilizes a fluid step size process. The vectorized RK4 method yielded a substantially faster solution in regards to this specific problem, which can be further increased by increasing the step size, however this would hinder the accuracy of the results and it is recommended that a step size of 0.1 is utilized as it strikes a good balance between speed and accuracy.

Appendix

Main File

```
%% Q1
clc;clear all;format long e;
[Aug1, Text]=xlsread('DataFile_Assn2.xlsx');
A=Aug1(:,1:21);
b=Aug1(:,22);

tic;
max_iter=1000;TOL=1e-5;
[xcJ, iterJ]=Jacobi(A,b,max_iter,TOL);
tJ=toc;

tic;
max_iter=1000;TOL=1e-5;
[xcG, iterG]=GaussSeidel(A,b,max_iter,TOL);
tG =toc;

tic;
max_iter=1000;TOL=1e-5;
[xcS0, iterS0]=SOR(A,b,0.5,max_iter,TOL);
tS0=toc;

tic;
max_iter=1000;TOL=1e-5;
[xcS1, iterS1]=SOR(A,b,1.5,max_iter,TOL);
tS1=toc;

tic;
xc=A\b;
tB=toc;

% Residuals
residual_J = norm(b-A*xcJ);
residual_G = norm(b-A*xcG);
residual_S0 = norm(b-A*xcS0);
residual_S1 = norm(b-A*xcS1);
residual_backslash = norm(b-A*xc);

fprintf("Jacobi Method\t\t Gauss Seidel Method\t SOR w < 1 Method\t SOR w > 1
Method\t BackSlash Method\n");
for i = 1:1:21
    fomatSpec='F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\t F%d =
%12.5f\n';
    fprintf(fomatSpec,i,xcJ(i),i,xcG(i),i,xcS0(i),i,xcS1(i),i,xc(i));
end

formatSpec1='time:\nt = %fs\t\t t = %fs\t\t t = %fs\t\t t = %fs\t\t t = %fs\n';
fprintf(formatSpec1,tJ,tG,tS0,tS1,tB);
formatSpec2='Iterations:\niter = %d\t\t iter = %d\t\t iter = %d\t\t iter = %d\n';
fprintf(formatSpec2,iterJ,iterG,iterS0,iterS1);
formatSpec3='Residuals:\nRes = %e\t Res = %e\t Res = %e\t Res = %e\t Res = %e\n';
```

```
fprintf(formatSpec3,residual_J,residual_G,residual_S0,residual_S1,residual_backslash)
;

figure;

% Sparsity patern of matrix representing non zero elements
subplot(1, 2, 1);
spy(A);
title('Structure of the Coefficient Matrix');
xlabel('Column Index');
ylabel('Row Index');
grid on;

% 3d view of plot showing magnitude difference in coefficients
subplot(1, 2, 2);
surf(A);
title('Surface Plot of the Coefficient Matrix');
xlabel('Column Index');
ylabel('Row Index');
zlabel('Value');
%% Q2
clc; clear all; format long e;
% Given parameters
L=10;Hp=0.05;sig=2.7*10^-9;Ti=200;Ta=300;Tb=400;dx=2;
b=Hp*dx^2;a=2+b;c=sig*dx^2;d=b*Ti+c*Ti^4;e=Ta+d;f=Tb+d;

% Initial guesses
Guess1=[2; 4; 6; 8]; Guess2=[2.1; 4.1; 6.1; 8.1];

% Tolerance and maximum iterations for the solvers
TOL = 0.5e-3; max_iter = 100;

% System of non linear eqn
Fn=@(x,a,b,c,d,f,e) [...
    a.*x(1)+c.*x(1).^4-x(2)-e;...
    -x(1)+a.*x(2)+c.*x(2).^4-x(3)-d;...
    -x(2)+a.*x(3)+c.*x(3).^4-x(4)-d;...
    -x(3)+a.*x(4)+c.*x(4).^4-f];
options=optimset('display','iter');

% Compute the symbolic Jacobian matrix
syms x1 x2 x3 x4;
Fs = [a*x1 + c*x1^4 - x2 - e;
      -x1 + a*x2 + c*x2^4 - x3 - d;
      -x2 + a*x3 + c*x3^4 - x4 - d;
      -x3 + a*x4 + c*x4^4 - f];
Xs = [x1; x2; x3; x4];
DFs = jacobian(Fs, Xs);

% Multivariate Newton
tic;
[xcN, iterN] = multivariateNewton_fs(Fn, a, b, c, d, f, e, @Jac_fs, DFs, Xs, Guess1,
TOL, max_iter);
tN=toc;
```

```matlab
% Jacobian is assumed to be Identity matrix
A= eye(4);

% Broyden Method
tic
[xcB,iterB]=BroydenMethod1(Fn,a, b, c, d, f, e,Guess1,Guess2,A,TOL,max_iter);
tB=toc;

% fSolve Method
tic
[xcF]=fsolve(Fn,Guess1,options,a,b,c,d,f,e);
tF=toc;

% Residuals
residual_N = norm(Fn(xcN, a, b, c, d, f, e));
residual_B = norm(Fn(xcB, a, b, c, d, f, e));
residual_F = norm(Fn(xcF, a, b, c, d, f, e));

%Display the final solutions, comp times, iterations & Residuals
fprintf("Multi Newton Method\t Broyden Method\t\t fSolve\n");
for i = 1:1:4
    fomatSpec='F%d = %12.5f\t F%d = %12.5f\t F%d = %12.5f\n';
    fprintf(fomatSpec,i,xcN(i),i,xcB(i),i,xcF(i));
end

formatSpec1='time:\nt = %fs\t\t t = %fs\t\t t = %fs\n';
fprintf(formatSpec1,tN,tB,tF);
formatSpec2='Iterations:\niter = %d\t\t iter = %d\t\t iter = %d\n';
fprintf(formatSpec2,iterN,iterB,12);
formatSpec3='Residuals:\nRes = %e\t Res = %e\t Res = %e\n';
fprintf(formatSpec3,residual_N,residual_B,residual_F);

% Temperature distribution plot
x = 0:2:10;
Ttot = [Ta; xcF; Tb];
plot(x, Ttot);
xlabel('Position along the rod (m)');
ylabel('Temperature (K)');
title('Temperature Distribution along the Rod');
grid on;
%% Q3
clc; clear all; close all; format long e;
% Given parameters
m=[12000 10000 8000];k=[3000 2400 1800];
a = k(1)/m(1); b = k(2)/m(1); c = k(2)/m(2);
d = k(3)/m(2); e = k(3)/m(3);

% RK4 parameters
h = 0.1; t0 = 0; tn = 20;
NStep = abs(tn - t0) / h;

% Initial conditions
Y0 = [0; 1; 0; 0; 0; 0];

% RK4 method
```

```
tic;
[t_rk4, Y_rk4] = VectorRK4(@ode_function, t0, Y0, NStep, h, a, b, c, d, e);
trk4 = toc;
% ode45 solver
tic;
[t_ode45, Y_ode45] = ode45(@(t, Y) ode_function(t, Y, a, b, c, d, e), [t0, tn], Y0);
tode45 = toc;

% Extract positions and velocities
x1_rk4 = Y_rk4(:, 1);   x1_ode45 = Y_ode45(:, 1);
v1_rk4 = Y_rk4(:, 2);   v1_ode45 = Y_ode45(:, 2);
x2_rk4 = Y_rk4(:, 3);   x2_ode45 = Y_ode45(:, 3);
v2_rk4 = Y_rk4(:, 4);   v2_ode45 = Y_ode45(:, 4);
x3_rk4 = Y_rk4(:, 5);   x3_ode45 = Y_ode45(:, 5);
v3_rk4 = Y_rk4(:, 6);   v3_ode45 = Y_ode45(:, 6);

% Plotting Displacement
figure;
subplot(2, 1, 1);
plot(t_rk4, [x1_rk4, x2_rk4, x3_rk4], 'LineWidth', 1.5);
hold on;
plot(t_ode45, [x1_ode45, x2_ode45, x3_ode45], '--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Displacement');
title('Displacement versus Time');
legend('x1 (RK4)', 'x2 (RK4)', 'x3 (RK4)', 'x1 (ode45)', 'x2 (ode45)', 'x3 (ode45)');
grid on;

% Plotting Velocity
subplot(2, 1, 2);
plot(t_rk4, [v1_rk4, v2_rk4, v3_rk4], 'LineWidth', 1.5);
hold on;
plot(t_ode45, [v1_ode45, v2_ode45, v3_ode45], '--', 'LineWidth', 1.5);
xlabel('Time (s)');
ylabel('Velocity');
title('Velocity versus Time');
legend('v1 (RK4)', 'v2 (RK4)', 'v3 (RK4)', 'v1 (ode45)', 'v2 (ode45)', 'v3 (ode45)');
grid on;
hold off;

% Plot the 3D phase plane of displacements
figure;
plot3(x1_rk4, x2_rk4, x3_rk4, 'LineWidth', 1.5);
hold on;
plot3(x1_ode45, x2_ode45, x3_ode45, '--', 'LineWidth', 1.5);
legend('RK4','ode45')
xlabel('x1');
ylabel('x2');
zlabel('x3');
title('3D Phase Plane of Displacements');

%Comp time for each process
fprintf("Computation time:\n");
fomatSpec='Vectorized RK4:%fs\tODE45 solver:%fs\n';
fprintf(fomatSpec,trk4,tode45);
```

```
% Calculate the absolute error for displacement components
common_length = min(length(x1_rk4), length(x1_ode45));
abs_error_x1 = abs(x1_rk4(1:common_length) - x1_ode45(1:common_length));
abs_error_x2 = abs(x2_rk4(1:common_length) - x2_ode45(1:common_length));
abs_error_x3 = abs(x3_rk4(1:common_length) - x3_ode45(1:common_length));
% Calculate the absolute error for velocity components
common_length_v = min(length(v1_rk4), length(v1_ode45));
abs_error_v1 = abs(v1_rk4(1:common_length_v) - v1_ode45(1:common_length_v));
abs_error_v2 = abs(v2_rk4(1:common_length_v) - v2_ode45(1:common_length_v));
abs_error_v3 = abs(v3_rk4(1:common_length_v) - v3_ode45(1:common_length_v));

% Display the maximum absolute errors for each displacement component
fprintf('Maximum Absolute Error for x1: %e\n', max(abs_error_x1));
fprintf('Maximum Absolute Error for x2: %e\n', max(abs_error_x2));
fprintf('Maximum Absolute Error for x3: %e\n', max(abs_error_x3));
% Display the maximum absolute errors for each velocity component
fprintf('Maximum Absolute Error for v1: %e\n', max(abs_error_v1));
fprintf('Maximum Absolute Error for v2: %e\n', max(abs_error_v2));
fprintf('Maximum Absolute Error for v3: %e\n', max(abs_error_v3));
```

Function Files

```
%% Jacobi Function
function [xout iter_number]=Jacobi(A,b,max_iter,TOL)
    N=length(b);
    d=diag(A); % takes the main diagonal elements only
    x=zeros(N,1); % initial guess vector
        for k=1:max_iter
            x=(b-(A-diag(d))*x)./d; % Note: A=L+D+U, so L+U=A-D
            maxresidual=norm(b-A*x,inf);
            if maxresidual<TOL
                break;
            end
        end
iter_number=k;
xout=x;
%% Gauss Seidel Function
function [xout iter_number]=GaussSeidel(A,b,max_iter,TOL)
    N=length(b);
    d=diag(diag(A)); % takes the main diagonal elements only
    x=zeros(N,1); % initial guess vector
    U=triu(A,1); % above the main diagonal
    L=tril(A,-1); % below the main diagonal
    for k=1:max_iter
        bL=b-U*x;
        for j=1:N
            x(j)=(bL(j)-L(j,:)*x)./d(j,j);
        end
        maxresidual=norm(b-A*x,inf);
        if maxresidual<TOL
            break;
        end
    end
```

```
iter_number=k;
xout=x;
%% SOR Function
function [xout iter_number]=SOR(A,b,relax,max_iter,TOL)
    N=length(b); d=diag(diag(A));
    x=zeros(N,1); % initial guess vector
    U=triu(A,1); % above the main diagonal
    L=tril(A,-1); % below the main diagonal
    for k=1:max_iter
        bL=relax*(b-U*x)+(1-relax)*d*x;
        for j=1:N
            x(j)=(bL(j)-relax*L(j,:)*x)./d(j,j);
        end
        maxresidual=norm(b-A*x,inf);
        if maxresidual<TOL
            break;
        end
    end
iter_number=k;
xout=x;
%% Multivariate Newton Function
function [xout,k]=multivariateNewton_fs(F,a, b, c, d, f, e,J,DFs,Xs,x0,TOL,max_iter)
k=0;
enorm=10; % dummy to start the iteration process
x=x0;xold=x0;
while enorm>TOL
    dx=J(x,DFs,Xs)\F(x,a, b, c, d, f, e); % J-Jacobian (computed symbolically), % F-
function vector
    x=x-dx;
    k=k+1;
    enorm=norm(x-xold,inf);
    xold=x;
    %fprintf(1,'%d %15.10f %15.10f %15.10f\n',k,enorm,x(1),x(2));
    if k>=max_iter
        break;
    end
end
xout=x;
%% Jac_fs Function
function [fJac]=Jac_fs(Xn,DFs,Xs)
fJac=subs(DFs,Xs,Xn);
fJac=double(fJac);
%% Broyden Function
function [xout,k]=BroydenMethod1(F,a, b, c, d, f, e,x0,x1,A,TOL,max_iter)
    for k=1:max_iter
        deltax=x1-x0; deltaF=F(x1,a, b, c, d, f, e)-F(x0,a, b, c, d, f, e);
        A=A+(deltaF-A*deltax)*deltax'/(deltax'*deltax);
        dx=A\F(x1,a, b, c, d, f, e);
        x=x1-dx;
        enorm=norm(x-x1,inf); x0=x1;x1=x;
        %fprintf(1,'%d %15.10f %15.10f %15.10f\n',k,enorm,x(1),x(2));
        if enorm <TOL || k >= max_iter
            break;
        end
    end
```

```
xout=x;
%% Vector RK4 Function
function [t_rk4, Y_rk4] = VectorRK4(ode_function, t0, Y0, NStep, h, a, b, c, d, e)
    % Pre-allocation
    t_rk4 = zeros(NStep, 1);
    Y_rk4 = zeros(NStep, length(Y0));

    % to store results of y1 and y2 in a singe array
    Y_rk4(1, :) = Y0;

    % initial condition
    t_rk4(1) = t0;

    for k = 1:NStep
        s1 = ode_function(t_rk4(k), Y_rk4(k, :), a, b, c, d, e);
        s2 = ode_function(t_rk4(k) + h/2, Y_rk4(k, :) + h/2 .* s1', a, b, c, d, e);
        s3 = ode_function(t_rk4(k) + h/2, Y_rk4(k, :) + h/2 .* s2', a, b, c, d, e);
        s4 = ode_function(t_rk4(k) + h, Y_rk4(k, :) + h .* s3', a, b, c, d, e);
        Y_rk4(k + 1, :) = Y_rk4(k, :) + h * (s1/6 + s2/3 + s3/3 + s4/6)';
        t_rk4(k + 1) = t_rk4(k) + h;
    end
end
%% ODE Function Function
function dydt = ode_function(t, Y, a, b, c, d, e)
    % Extract variables
    x1 = Y(1);
    v1 = Y(2);
    x2 = Y(3);
    v2 = Y(4);
    x3 = Y(5);
    v3 = Y(6);

    % Compute derivatives
    dx1dt = v1;
    dv1dt = -a .* x1 + b .* (x2 - x1);
    dx2dt = v2;
    dv2dt = c .* (x1 - x2) + d .* (x3 - x2);
    dx3dt = v3;
    dv3dt = e .* (x2 - x3);

    % Return the derivatives
    dydt = [dx1dt; dv1dt; dx2dt; dv2dt; dx3dt; dv3dt];
end
```