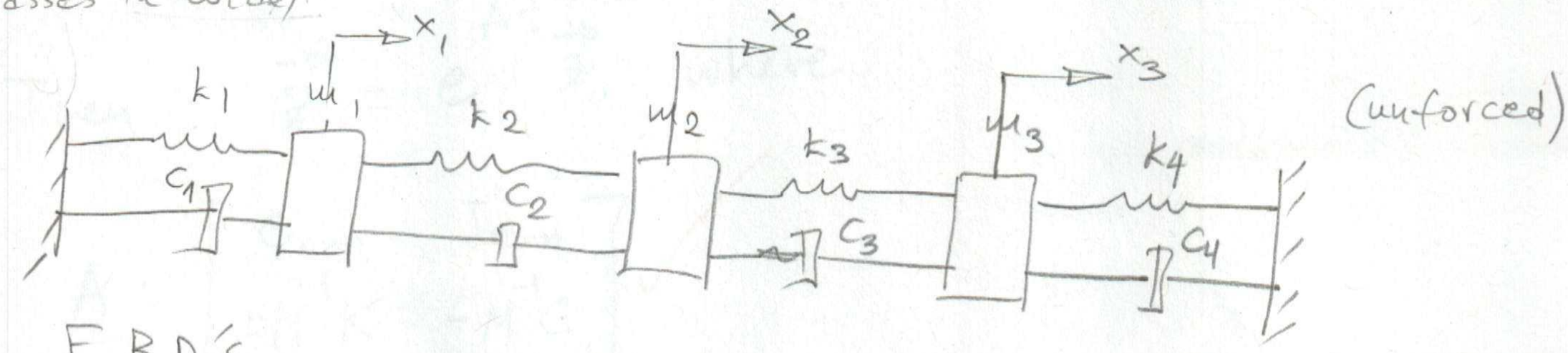
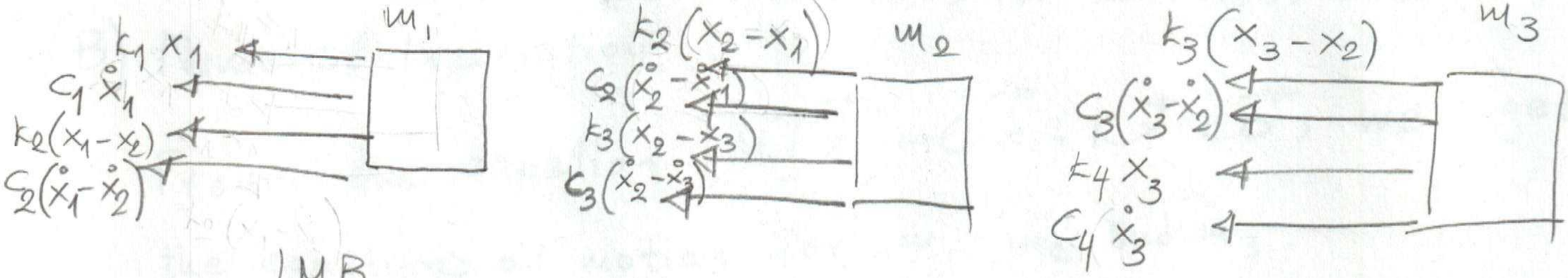
General MDOF damped oscillator

A general case with 2 springs & 2 dampers around each mass. (3 masses in total)



(unforced)

F.B.D's



LMB

$$
\left.
\begin{aligned}
&m_1 \ddot{x}_1 + (k_1+k_2)x_1 - k_2 x_2 + (c_1+c_2)\dot{x}_1 - c_2 \dot{x}_2 = 0 \\
&m_2 \ddot{x}_2 + (k_2+k_3)x_2 - k_2 x_1 - k_3 x_3 + (c_2+c_3)\dot{x}_2 - c_2 \dot{x}_1 - c_3 \dot{x}_3 = 0 \\
&m_3 \ddot{x}_3 + (k_3+k_4)x_3 - k_3 x_2 + (c_3+c_4)\dot{x}_3 - c_3 \dot{x}_2 = 0
\end{aligned}
\right\} \Rightarrow
$$

$$
\begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \ddot{\vec{x}} + \begin{bmatrix} c_1+c_2 & -c_2 & 0 \\ -c_2 & c_2+c_3 & -c_3 \\ 0 & -c_3 & c_3+c_4 \end{bmatrix} \dot{\vec{x}} + \begin{bmatrix} k_1+k_2 & -k_2 & 0 \\ -k_2 & k_2+k_3 & -k_3 \\ 0 & -k_3 & k_3+k_4 \end{bmatrix} \vec{x} = \vec{0}
$$

For given initial conditions:

(A) Matrix exponential Solution

$$\vec{z} = \begin{bmatrix} \vec{x} \\ \vec{v} \end{bmatrix}, \text{ with } \vec{z}_0 = \begin{bmatrix} \vec{x}_0 \\ \vec{v}_0 \end{bmatrix}.$$

Then $\vec{z} = e^{At}\vec{z}_0$, where:

$$A = \begin{bmatrix} O_{n\times n} & I_{n\times n} \\ -M^{-1}K & -M^{-1}C \end{bmatrix}$$ ✓

We can then find the displacements for each mass over time.

(B) Numerial Integration.

From the equation $[M]\ddot{\vec{x}} + C\dot{\vec{x}} + K\vec{x} = \vec{0}$, we get the equations of motion for m_1, m_2 and m_3.

Starting from the given initial conditions, using ODE45 we can integrate and derive the displacements over time.

(C) The "Classical" Method

$$M^{-1/2} \cdot \left\{ M\ddot{\vec{x}} + C\dot{\vec{x}} + K\vec{x} = \vec{0} \right\} \Rightarrow \quad \vec{x} = M^{-1/2}\vec{q}$$

$$M^{1/2}\ddot{\vec{x}} + M^{-1/2}C\dot{\vec{x}} + M^{-1/2}K\vec{x} = \vec{0} \Rightarrow \quad \vec{q} = P\vec{r}$$

$$\ddot{\vec{q}} + M^{-1/2}CM^{-1/2}\dot{\vec{q}} + M^{-1/2}KM^{-1/2}\vec{q} = \vec{0} \quad \because P^{-1} = P^T$$

$$P\ddot{\vec{r}} + M^{-1/2}CM^{-1/2}P\dot{\vec{r}} + M^{-1/2}KM^{-1/2}P\vec{r} = \vec{0}$$

$$\ddot{\vec{r}} + \underbrace{P^T M^{-1/2}CM^{-1/2}P}_{\tilde{C}}\dot{\vec{r}} + \underbrace{P^T M^{-1/2}KM^{-1/2}P}_{\Lambda}\vec{r} = \vec{0}$$

$$\ddot{\vec{r}} + \tilde{C}\dot{\vec{r}} + \Lambda\vec{r} = \vec{0}$$

P eigenvectors of $\tilde{K} = M^{-1/2}KM^{-1/2}$

Λ diagonal matrix with $\tilde{K}$'s eigenvalues

general form: $\ddot{x} + 2\zeta\omega_n\dot{x} + \omega_n^2 x = 0$ for each DoF.

If $\tilde{C}$ = diagonal, for example $\tilde{C} = \alpha M + \beta K$, then, we get n decoupled equations, each corresponding to a different DOF.

Then if we identify the constants J, ω_n, given the initial conditions $\vec{x}_0$, $\vec{v}_0$, we can use the general analytical solution for each mass(DOF):

$$x(t) = e^{-J\omega_n t}\left(b_1 \cos\omega_d t + b_2 \sin\omega_d t\right), \quad \omega_d = \omega_n\sqrt{1-J^2}$$

b_1: we can easily see that $\vec{x}(t=0) = \vec{x}_0 \Rightarrow \vec{b}_1 = \vec{x}_0$. ✓

b_2: Differentiating the analytical sol wrt time we get:

$$v(t) = -J\omega_n e^{-J\omega_n t}\left(b_1 \cos\omega_d t + b_2 \sin\omega_d t\right) + e^{-J\omega_n t}\left(-\omega_d b_1 \sin\omega_d t + \omega_d b_2 \cos\omega_d t\right)$$

Setting $t=0 \rightsquigarrow v_0 = -J\omega_n \cdot b_1 + \omega_d b_2 \Rightarrow$

$$\boxed{b_2 = \frac{v_0 + J\omega_n b_1}{\omega_d}}$$ ✓

Having determined all constants, we can compute $\vec{x}(t)$ for each point in time.

1), 2) I first considered a general case, as described in the front page, by picking random numbers for $[K]$, $[M]$ and $[C]$, $\vec{x}_0, \vec{v}_0$.

I solved for a randomly picked time span using A) the matrix exponential and B) a numerical solver (ODE45).

The solutions agree to an impressive extent. The computation time though is much* less when using the matrix exponential solution. This is due to the comparatively "slow" ODE solver, which among others runs an algorithm to pick the step size at each loop. As expected, the analytical soln involving the matrix exponential is much faster.

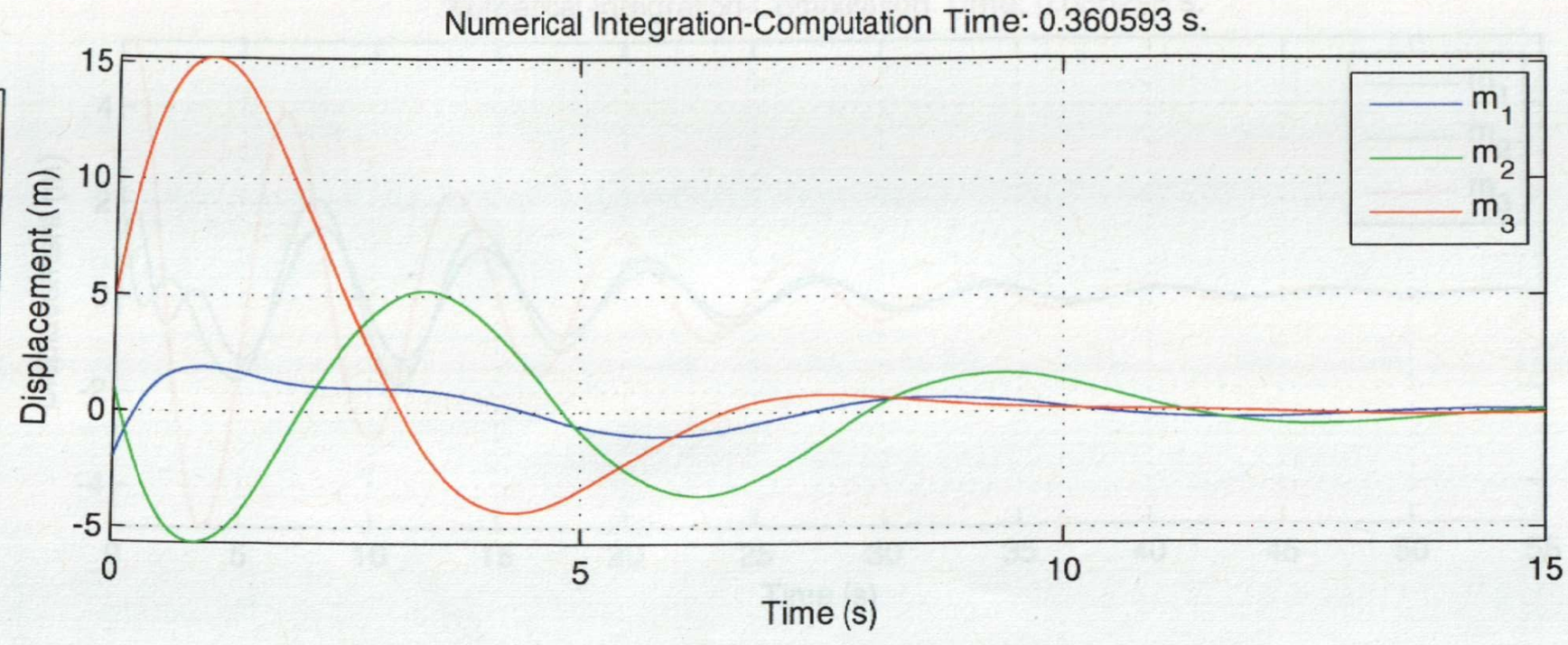
3) Then, assuming $C = \alpha M + bK$, I solved using the "classical" method as well and compared computation times. It appears that this method is the fastest one among all three. This makes sense because it only involves simple matrix algebra, which is much cheaper for a computer to implement. In particular, the time it needs for the same computation is much* less than the ODE45 slver.

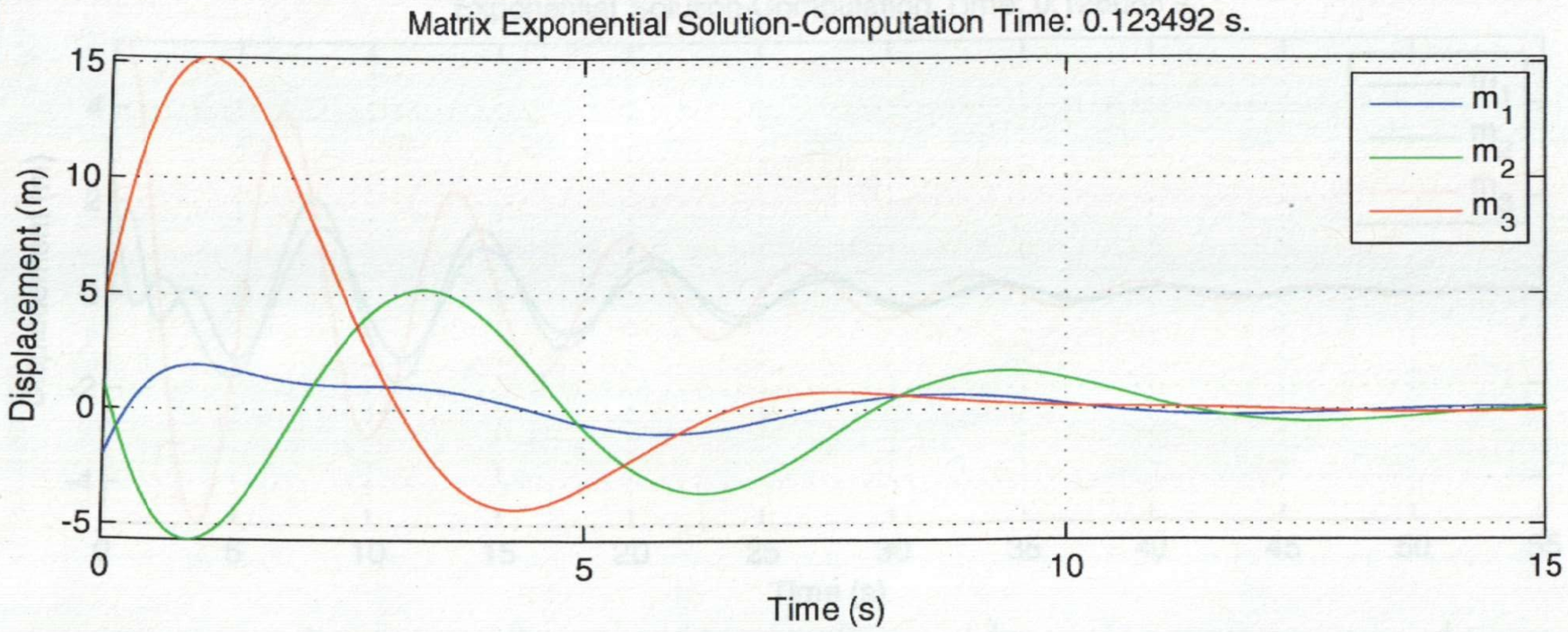
Plot Results for each case, along with the corresponding code, follow, for different initial conditions.

* see detailed time count. on figures, it depends on time span as well.

Problem 1,2

m_1=1kg
m_2=2kg
m_3=5kg
k_1=2N/m
k_2=0.1N/m
k_3=1N/m
k_4=4N/m
c_1=1Ns/m
c_2=0.4Ns/m
c_3=0.2Ns/m
c_4=3Ns/m
$x0=[-2\ 1\ 5]^T$
$v0=[10\ -15\ 20]^T$

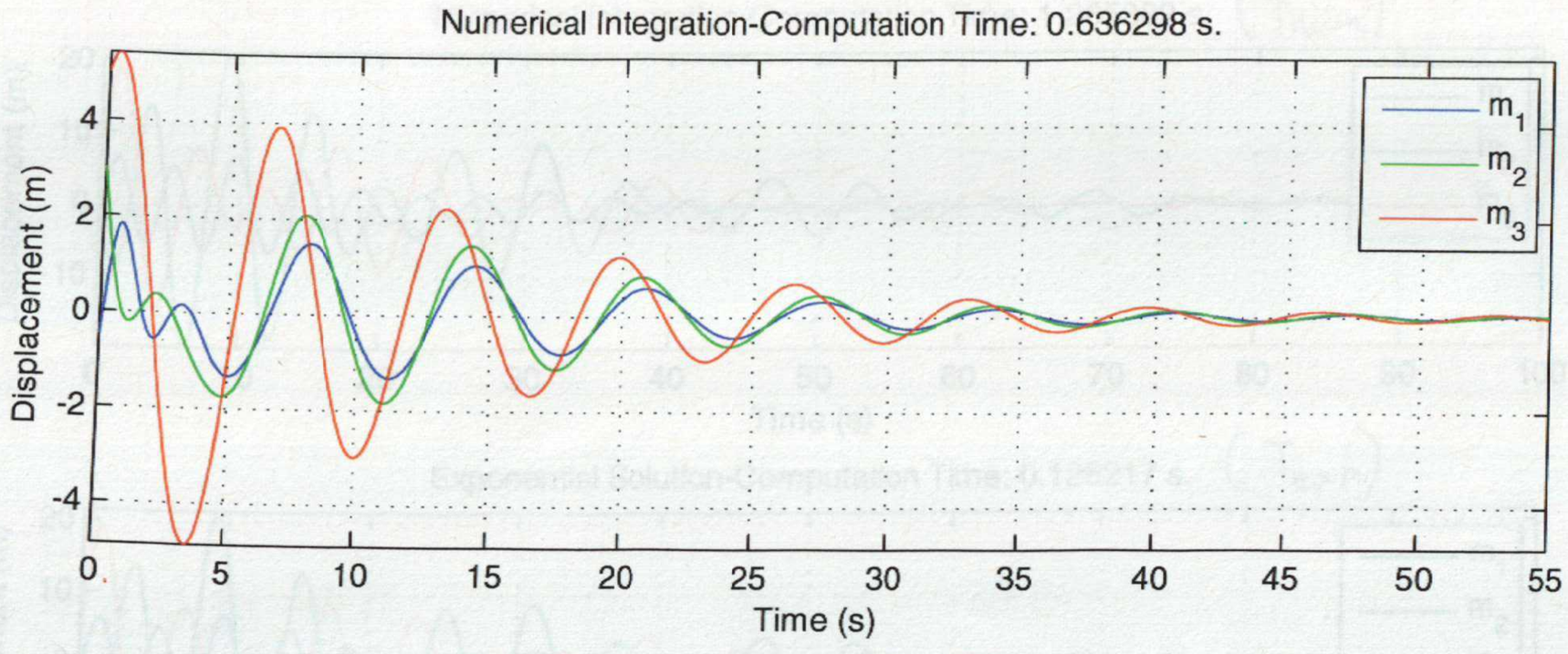


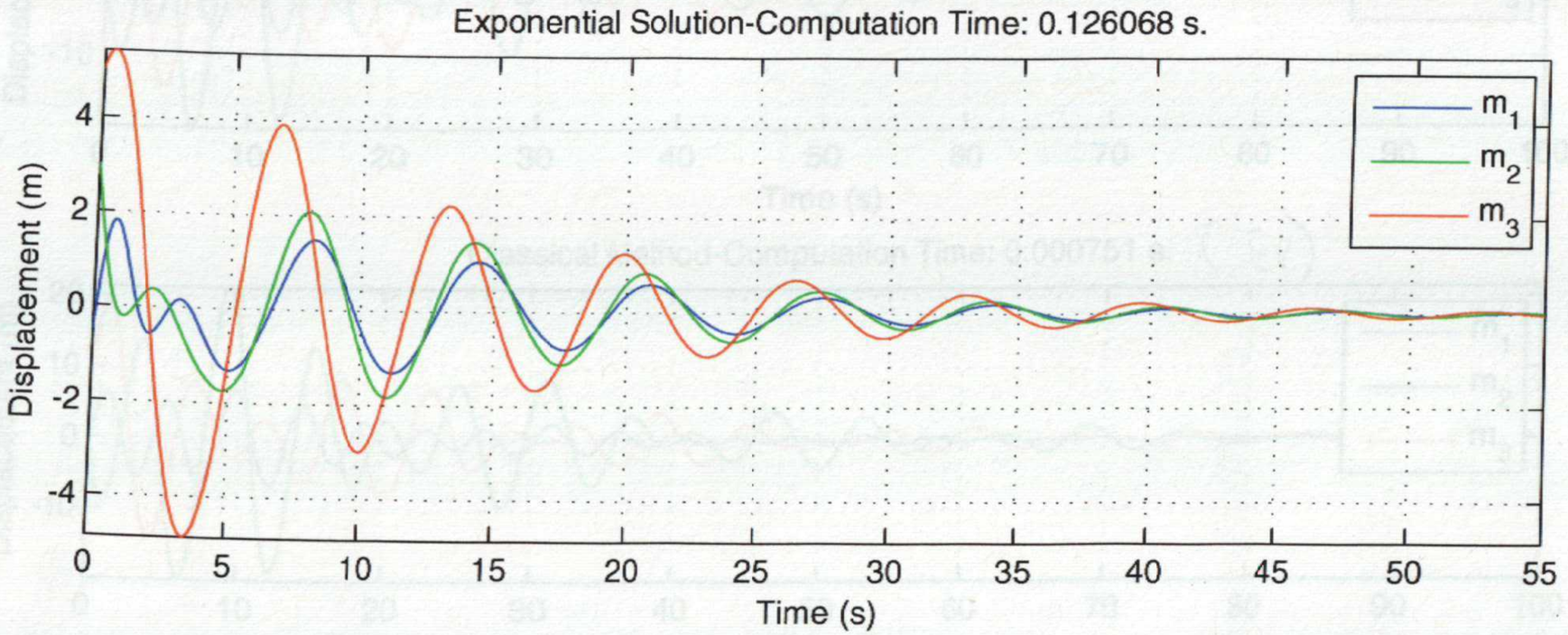


Almost 3 times as much computation time for the numerical Integration. Time span sufficient for convergence.

Problem 1,2

m_1=1kg
m_2=2kg
m_3=5kg
k_1=2N/m
k_2=0.3N/m
k_3=1N/m
k_4=4N/m
c_1=1Ns/m
c_2=0.4Ns/m
c_3=0.02Ns/m
c_4=0.3Ns/m
$x_0=[-0.5\ 3\ 5]^T$
$v0=[5\ -5\ 2]^T$

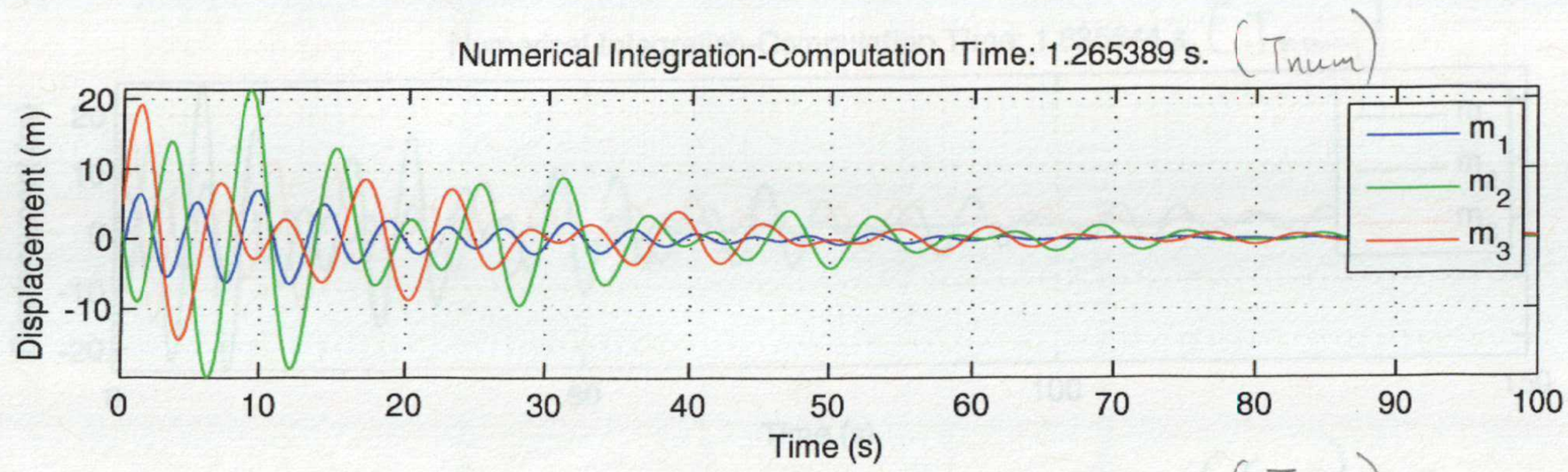




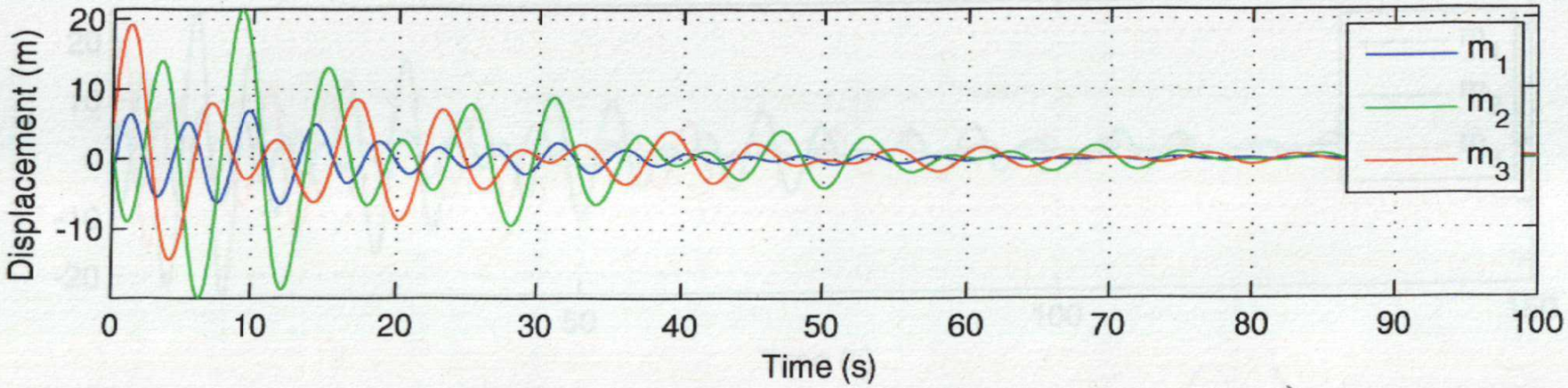
Different initial conditions, less damping ⇒ more time needed for convergence. Numerical → 5 times slower.
It seems that as timespan ↑, the difference in comp. time becomes more acute.

Problem 3

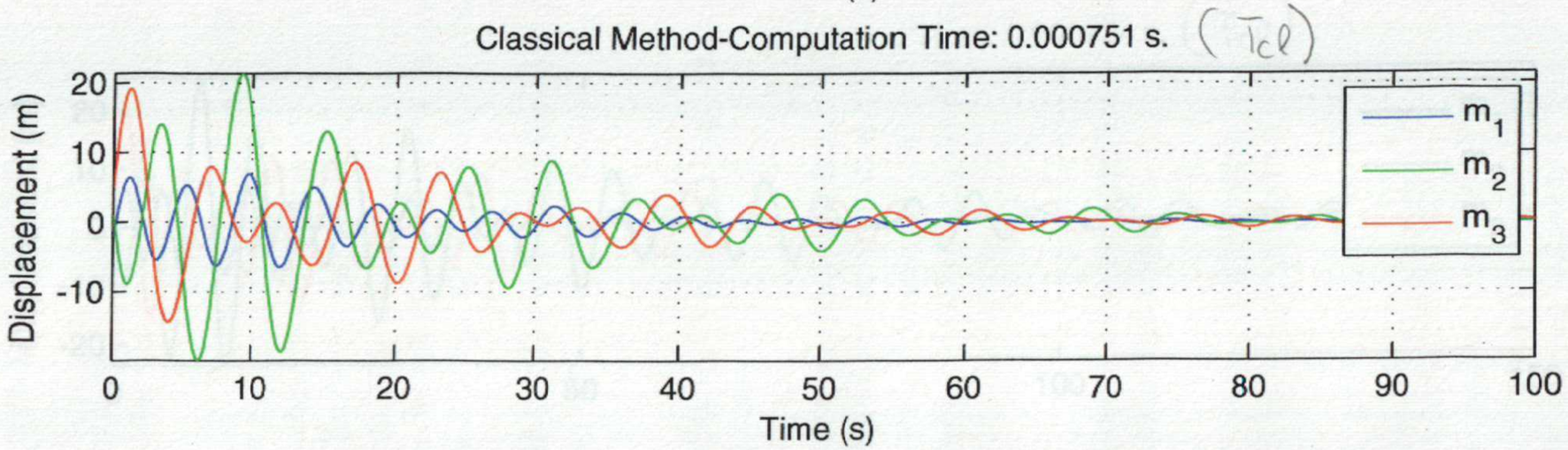
m_1=1kg
m_2=2kg
m_3=5kg
k_1=2N/m
k_2=0.1N/m
k_3=1N/m
k_4=4N/m
C=0.05M+0.03K
$x0=[-2\ 1\ 5]^T$
$v0=[10\ -15\ 20]^T$



(T_{num})



(T_{exp})

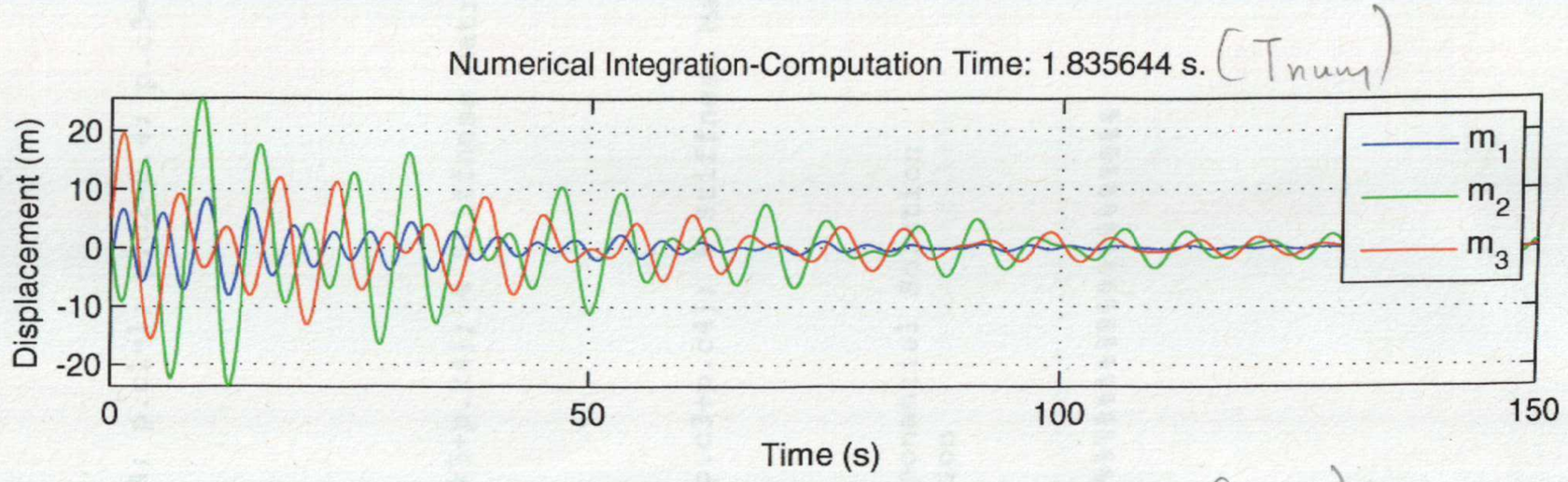


(T_{cl})

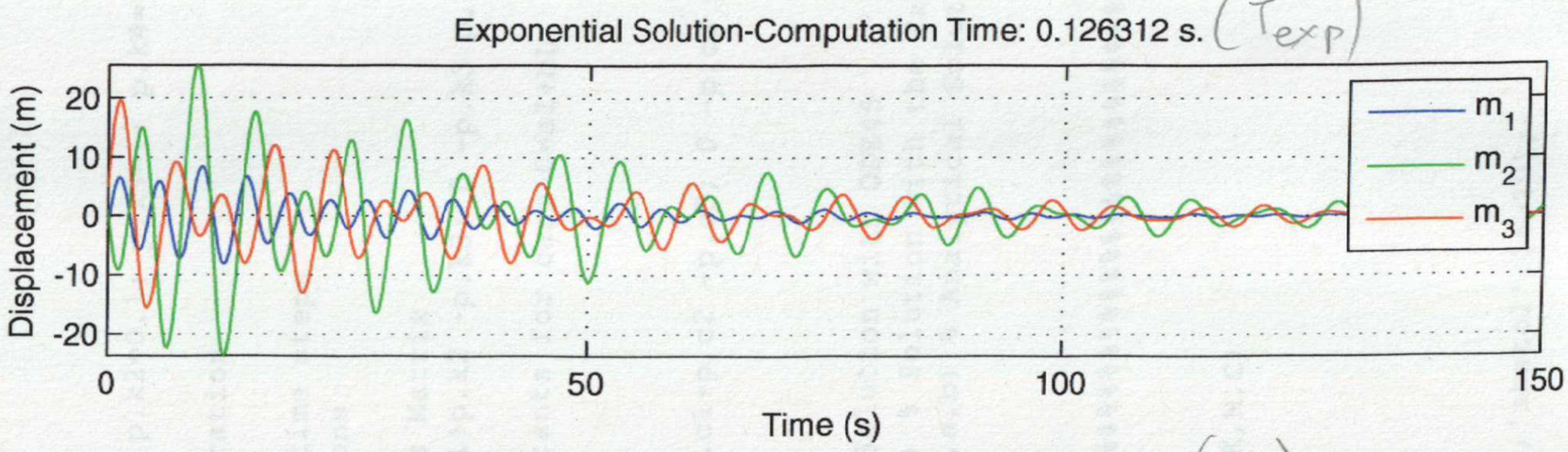
$T_{num} \simeq 10 \times T_{exp} \simeq 1800 \times T_{cl}$!

Problem 3

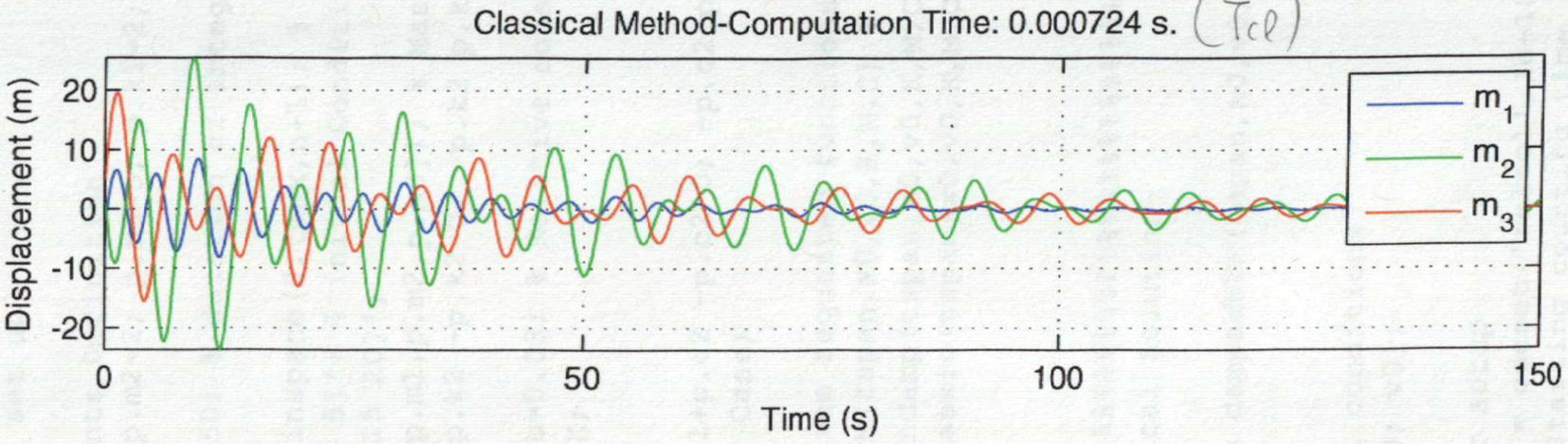
m_1=1kg
m_2=2kg
m_3=5kg
k_1=2N/m
k_2=0.1N/m
k_3=1N/m
k_4=4N/m
C=0.01M+0.03K
$x_0=[-2\ 1\ 5]^T$
$v_0=[10\ -15\ 20]^T$



(T_{num})



(T_{exp})



(T_{cl})

Less damping ⇒ slower convergence.

$T_{num} \simeq 15 \times T_{exp} \simeq 2500\, T_{cl}$.

(The time cost of the "classical" method remained the same every for 50% more time! - See explanation P.4)

```
function hw9()
clear all
close all
clc

%Problem set up

% Constants Definition
p.m1=1; p.m2=2; p.m3=5; p.k1=2; p.k2=0.1; p.k3=1; p.k4=4; p.c1=1; p.c2=0.4; p.c3=0.2; p.
c4=3;
tmax = 150; % duration of integration
n=1000;
tspan= linspace(0,tmax,n+1); % time steps
x0=[-2 1 5]'; % initial Conditions
v0=[10 -15 20]';
M=diag([p.m1,p.m2,p.m3]); % Mass Matrix
K=[p.k1+p.k2 -p.k2 0; -p.k2 p.k1+p.k2 -p.k3; 0 -p.k3 p.k3+p.k4]; % Stiffness Matrix

a=0.01; b=0.03;% Relative constants for case C=aI+bL
C=a*M+b*K;

% C=[p.c1+p.c2 -p.c2 0; -p.c2 p.c1+p.c2 -p.c3; 0 -p.c3 p.c3+p.c4]; % Stiffness Matrix
(General Case)

% The three requested functions
dampedNUM(tspan,x0,v0,K,M,C) % Solution with ODE45
exponSolndamp(tspan,x0,v0,K,M,C) % Solution with the Exponential Solution
dampedclassic(tspan,x0,v0,K,M,C,a,b) % Analytical Solution

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Numerical Solution
function dampedNUM(tspan,x0,v0,K,M,C)

%Initial conditions
z0 = [x0; v0];

% solver setup
options = odeset('reltol',1e-10,'abstol',1e-10);
tic; % Starting counting time
[t,zarray,tcost] = ode45(@ODEs,tspan, z0,options,K,M,C);
tcost=single(toc); % Stopping counting time
```

```
x1= zarray(:,1);
x2= zarray(:,2);
x3= zarray(:,3);

% Plot Setup
subplot(3,1,1); plot(t,x1,'b',t,x2,'g',t,x3,'r')
axis('tight');
titlestring = sprintf('Numerical Integration-Computation Time: %1f s.', tcost);
title(titlestring)
ylabel('Displacement (m)')
xlabel('Time (s)')
legend('m_{1}','m_{2}','m_{3}')
hold on
grid

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function zdot = ODEs(t,z,K,M,C)
% ODEs of motion
n=length(z);
r=z(1:0.5*n);
v=z(0.5*n+1:end);
rdot=v;
mall=diag(M,0);
kx=K*r;
cv=C*v;
a=-kx./mall-cv./mall;
vdot=a;

zdot = [rdot; vdot];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Exponential Solution

function exponSolndamp(t,x0,v0,K,M,C)

n=length(x0);
z0=[x0; v0];
tic
A=zeros(2*n,2*n);
A(1:n,n+1:end)=eye(n);
A(n+1:end,1:n)=-inv(M)*K;
A(n+1:end,n+1:end)=-inv(M)*C;

for i=1:length(t)
```

```
    z(:,i)=expm(A*t(i))*z0;
end
tcost=toc;

% Plot Setup
subplot(3,1,2); plot(t,z(1,:),'b',t,z(2,:),'g',t,z(3,:),'r')
axis('tight'); titlestring = sprintf('Exponential Solution-Computation Time: %1f s.',
tcost);
title(titlestring)
ylabel('Displacement (m)')
xlabel('Time (s)')
legend('m_{1}','m_{2}','m_{3}')
hold on
grid

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Classical Method

function [tarray,xarray] = dampedclassic(t,x0,v0,K,M,C,a,b)

n=length(x0);
tic;
Knew=M^(-.5)*K*M^(-.5);
[P,L]=eig(Knew);
U=M^(-.5)*P;
w2=diag(L,0);
wn=sqrt(w2);


Mid=a*ones(n,1)+b*w2;
zd=Mid./(2*wn);
wd=wn.*sqrt(1-zd.^2);

%Solve for initial conditions

x0new=U\x0;
v0new=U\v0;
b1=x0new;
b2=(v0new+zd.*b1)./wd;

for i=1:n
    r(i,:)=exp(-zd(i)*wn(i)*t').*(b1(i)*cos(wd(i)*t')+b2(i)*sin(wd(i)*t'));
end

X=U*r;
tcost=toc;

% Plot Setup
```

```
subplot(3,1,3); plot(t,X(1,:),'b',t,X(2,:),'g',t,X(3,:),'r')
axis('tight'); titlestring = sprintf('Classical Method-Computation Time: %1f s.', tcost);
title(titlestring)
ylabel('Displacement (m)')
xlabel('Time (s)')
legend('m_{1}','m_{2}','m_{3}')
grid

end
```