1 Write three matlab functions that solve the general spring-mass IVP

We will consider the generalized linear system without damping which has an invertible mass matrix:

$$M\ddot{\vec{x}} + K\vec{x} = 0 \tag{1}$$

a [tarray xarray] = SpringmassNUM(tspan,x0,v0,K,M)

This can use ODE45 or your own ODE integrator, your choice. It should work with arbitrary positive definite symmetric M and K matrices of any size.

Equation 1 can be rearranged to form Equation 2, which can be numerically integrated. To do this, the equation is first converted into two first order equations, as seen below.

$$\ddot{\vec{x}} = -M^{-1}K\vec{x} \tag{2}$$

$$\vec{z} = \dot{\vec{x}} \tag{3}$$

$$\dot{\vec{z}} = \ddot{\vec{x}} = -M^{-1}K\vec{x} \tag{4}$$

It is worth noting that this is a linear system, and as such can be simulated in Matlab using `lsim()` both faster and more accurately than using any numerical integration method. However, the code below uses `ode45()`.

```
function [tarray xarray] = SpringmassNUM(tspan,x0,v0,K,M)
% Numerically solves for the motion of the system defined by:
%        M(xddot) + K x = 0
%
% Inputs:
%   M,K - Matrices as defined in the equation above
%   x0 - vector of initial positions
%   v0 - vector of initial velocities
%   tspan - times at which to output the state of the system. tspan must be
%       a vector of length ≥ 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray - array of times corresponding to xarray
%   xarray - array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);
z0 = [x0; v0];

n = length(x0);
```

```
% Define rhs function — returns the derivative of the state
% Because anonymous functions
zdot = @(¬,z) [z(n+1:end); −M^−1*K*z(1:n)];

% Use ode45 to integrate
[tarray,xarray] = ode45(zdot,tspan,z0);
xarray = xarray(:,1:n); % only keep positions
end
```

b [tarray xarray] = SpringmassMinv(tspan, x0,v0,K,M)

This should use a superposition of normal mode solutions based in either (your choice) eig(K,M) or eig(M^-1*K).

First, we rearrange Equation 1 slightly, forming Equation 5.

$$\ddot{\vec{x}} + M^{-1}K\vec{x} = 0 \tag{5}$$

This is the form of the harmonic oscillator equation. Thus, we expect to see oscillatory solutions composed of sines and cosines. Our assumed solution will be written:

$$\vec{x}(t) = x(\omega t)\vec{v} = (Acos(\omega t) + Bsin(\omega t))\vec{v} \tag{6}$$

Substituting in:

$$(-\omega^2 I + M^{-1}K)x(\omega t)\vec{v} = 0 \tag{7}$$

Since $x(\omega t)$ is just a scalar, it can be divided out. The result is exactly the eigenvalue problem for the matrix $M^{-1}K$, producing n eigenpairs $(\lambda_i = \omega_i^2, \vec{v}_i)$. Each of the n pairs represents a valid form for our assumed solution in Equation 6. Since the sum of solutions to a homogeneous differential equation is also a solution to that differential equation, we can write the general form of the solution to Equation 5 as:

$$\vec{x}(t) = \sum_{i=1}^{n} \vec{v}_i(A_i cos(\omega_i t) + B_i sin(\omega_i t)) \tag{8}$$

To find the constants A_i and B_i, we will make use of the initial conditions: $\vec{x}(0) = \vec{x}_0$ and $\vec{v}(0) = \vec{v}_0$. Applying these conditions, we get the equations:

$$\vec{x}_0 = \sum_{i=1}^{n} \vec{v}_i A_i = V\vec{A} \tag{9}$$

$$\vec{v}_0 = \sum_{i=1}^{n} \vec{v}_i \omega_i B_i = V\Lambda^{1/2}\vec{B} \tag{10}$$

where $V = [\vec{v}_1|\vec{v}_2|...|\vec{v}_n]$ and Λ, having the eigenvalues λ_i on its diagonal, are the matrices output by the `eig()` function and we have simply defined vectors $\vec{A}$ and $\vec{B}$ to contain the

coefficients A_i and B_i, respectively. Noting the fact that $V^{-1} = V^T$ and solving for these coefficients, we obtain:

$$\vec{A} = V^T \vec{x}_0 \tag{11}$$

$$\vec{B} = \Lambda^{-1/2} V^T \vec{v}_0 \tag{12}$$

Using these results with Equation 8, we have the analytical solution to Equation 1.

```
function [tarray xarray] = SpringmassMinv(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using eigenvectors of M^-1*K
%
% Inputs:
%   M,K - Matrices as defined in the equation above
%   x0 - vector of initial positions
%   v0 - vector of initial velocities
%   tspan - times at which to output the state of the system. tspan must be
%       a vector of lenght >= 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray - array of times corresponding to xarray
%   xarray - array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);

% Form timespan
if length(tspan) == 2
    tarray = linspace(tspan(1),tspan(2));
else
    tarray = tspan(:)';
end

[V,w2] = eig(M^-1*K);
w = sqrt(diag(w2));

% Solve for coefficients
A = V'*x0;
B = w2^(-1/2)*V'*v0;

% Vectorized linear combination of normal modes
cospart = repmat(A,1,length(tarray)).*cos(w*tarray);
sinpart = repmat(B,1,length(tarray)).*sin(w*tarray);
x = V*(cospart + sinpart);
xarray = x';
end
```

c [tarray xarray] = SpringmassSqrtM(tspan, x0,v0,K,M)

This should use a superposition of normal mode solutions based on the methods of lecture on 10/18 (using two changes of coordinates)

We can select a set of coordinates $\vec{q} = M^{1/2}\vec{x}$ and multiply Equation 1 by $M^{-1/2}$ to obtain:

$$\ddot{\vec{q}} + M^{-1/2}KM^{-1/2}q = \ddot{\vec{q}} + \tilde{K}q = 0 \tag{13}$$

$\tilde{K}$ has the convenient property that it has orthogonal eigenvectors. As such, $P^T\tilde{K}P = \Lambda$, where P has the eigenvectors of $\tilde{K}$ as its columns (in the same way V was defined with respect to $M^{-1}K$ in Section b) and Λ has the eigenvalues of $\tilde{K}$ on its diagonal (the are the same as the eigenvalues of $M^{-1}K$). We will exploit this fact by defining a new set of coordinates, $\vec{r} = P^T\vec{q}$. These are called "modal coordinates". We substitute this into Equation 13 and multiply by P^T to obtain:

$$\ddot{\vec{r}} + P^T\tilde{K}P\vec{r} = \ddot{\vec{r}} + \Lambda\vec{r} = 0 \tag{14}$$

Here we could proceed with the same analysis as in Section b to arrive at solutions for $\vec{r}(t)$, but instead, we will recognize that, due to the diagonal nature of Λ, Equation 14 represents a system of n decoupled differential equations with only a single degree of freedom. Each of these equations can be written as in Equation 15.

$$\ddot{r}_i + \lambda_i r_i = 0 \tag{15}$$

We can now solve each of these n equations using familiar techniques for 1-DOF oscillators. We can guess the form of the solutions:

$$r_i(t) = C_i cos(\sqrt{\lambda_i}t) + D_i sin(\sqrt{\lambda_i}t) \tag{16}$$

In order to apply the initial conditions for our equations, we must convert them to modal coordinates. By the definitions of $\vec{r}$ and $\vec{q}$, we have the transformations between coordinates:

$$\vec{r} = P^T M^{1/2}\vec{x} \tag{17}$$

$$\vec{x} = M^{-1/2}P\vec{r} \tag{18}$$

As such, we can transform the initial conditions with Equation 17 to obtain $\vec{r}_0 = P^T M^{-1/2}\vec{x}_0$ and $\dot{\vec{r}}_0 = P^T M^{-1/2}\vec{v}_0$. We apply these initial conditions to Equation 15 to obtain the coefficients as

$$C_i = r_{0,i} \tag{19}$$

$$D_i = \frac{\dot{r}_{0,i}}{\sqrt{\lambda_i}} \tag{20}$$

We have solved for $\vec{r}(t)$ by finding its individual components. To obtain the solution $\vec{x}(t)$, we simply apply the transformation in Equation 18.

```
function [tarray xarray] = SpringmassSqrtM(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using modal coordinates
%
% Inputs:
%   M,K - Matrices as defined in the equation above
%   x0 - vector of initial positions
%   v0 - vector of initial velocities
%   tspan - times at which to output the state of the system. tspan must be
%       a vector of lenght >= 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray - array of times corresponding to xarray
%   xarray - array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);

% For time span
if length(tspan) == 2
    tarray = linspace(tspan(1),tspan(2));
else
    tarray = tspan(:)';
end

Msqrt = M^(-1/2);
Ktwid = Msqrt*K*Msqrt;
[P,w2] = eig(Ktwid);
w = sqrt(diag(w2));

% Transform ICs
r0 = P'*Msqrt^-1*x0;
rd0 = P'*Msqrt^-1*v0;

% Solve for coefficients
C = r0;
D = rd0./w;

% Solution for each 1DOF oscillator
r = repmat(C,1,length(tarray)).*cos(w*tarray) +...
    repmat(D,1,length(tarray)).*sin(w*tarray);

% Transform back to x-coordinates
x = M*P*r;
xarray = x';
end
```

d For some fairly complex problem show that your three methods agree as well as they should.

This comparison includes the functions described below in Section 2.

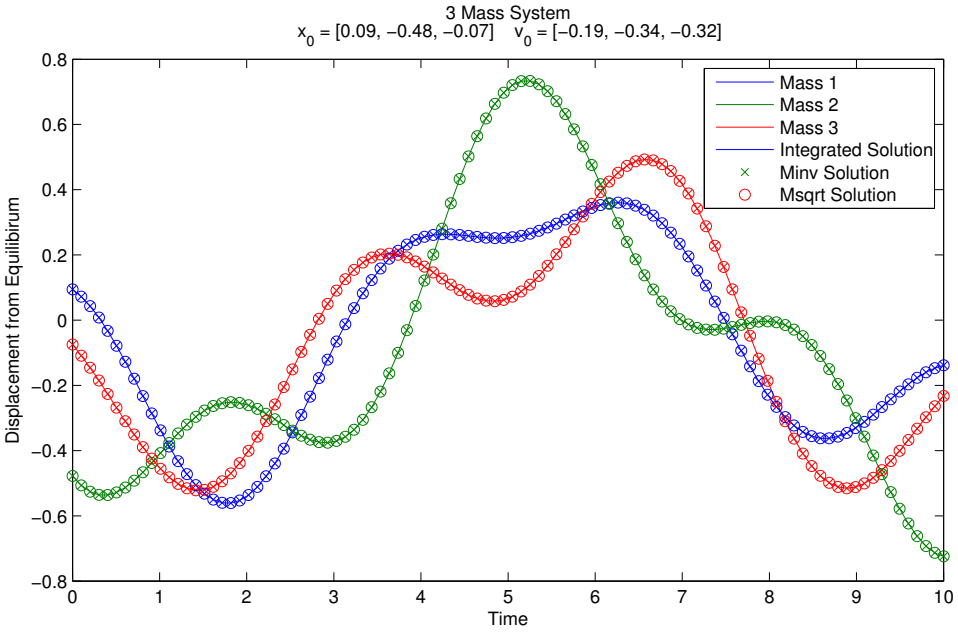

Figure 1: Solutions for a three mass system with a nonsingular K matrix using the methods described above that do not handle singular matrices.

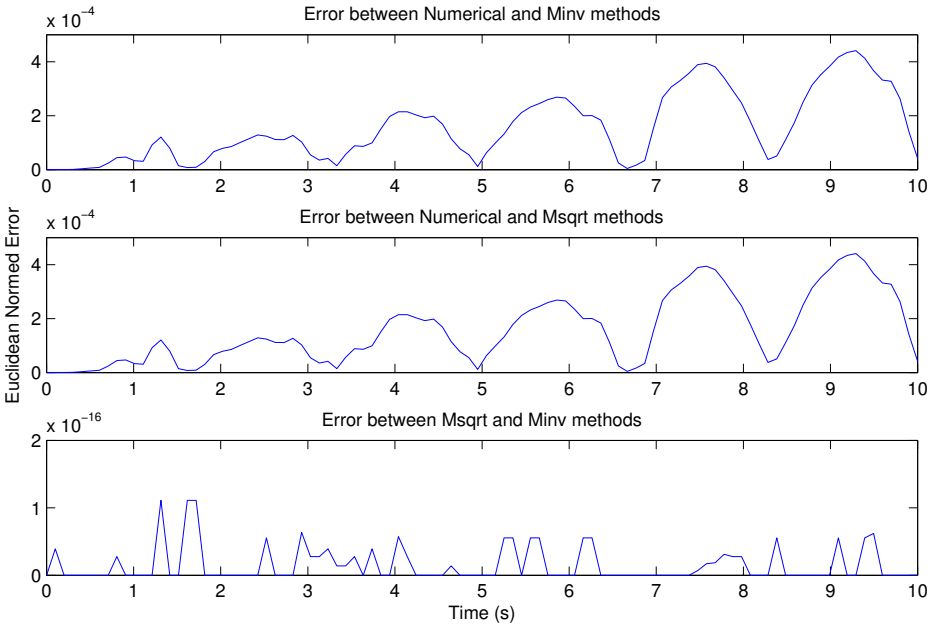

Figure 2: Associated error for the system in Figure 1

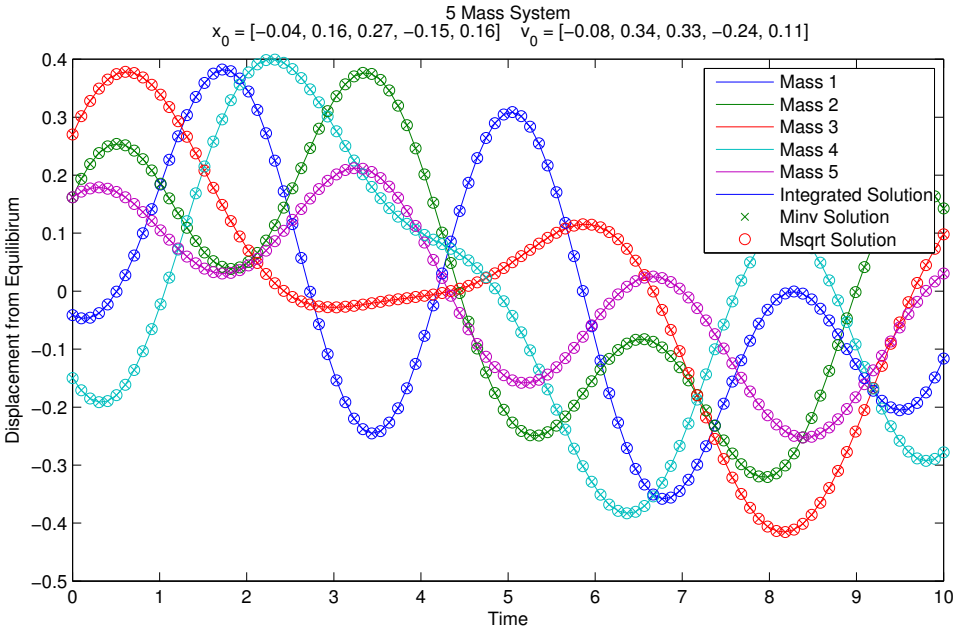

Figure 3: Solutions for a five mass system with a nonsingular K matrix using the methods described above that do not handle singular matrices. Note that this system is not simply five masses connected with springs in series, but instead has some springs that connect non-adjacent masses.

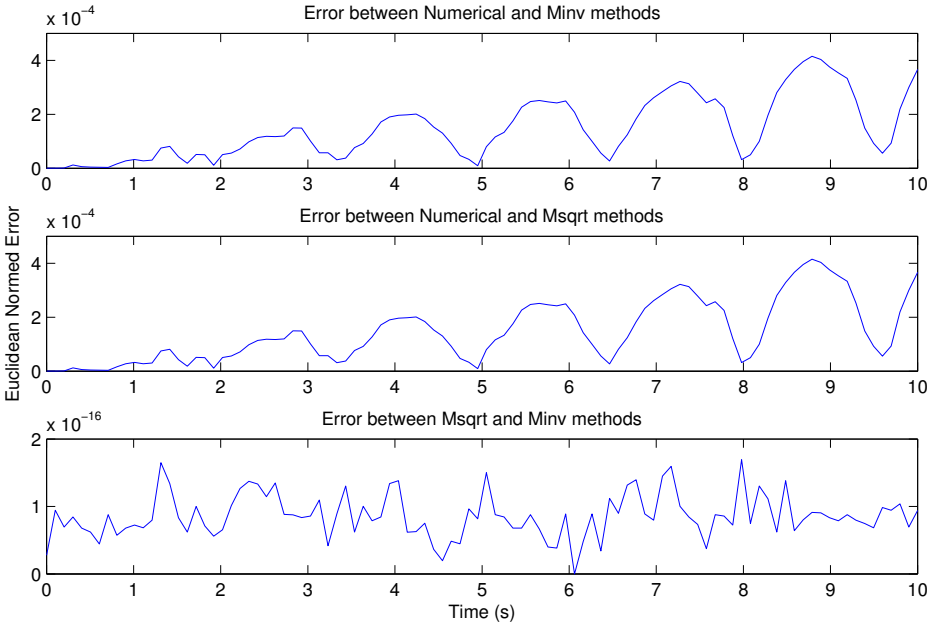

Figure 4: Associated error for the system in Figure 3

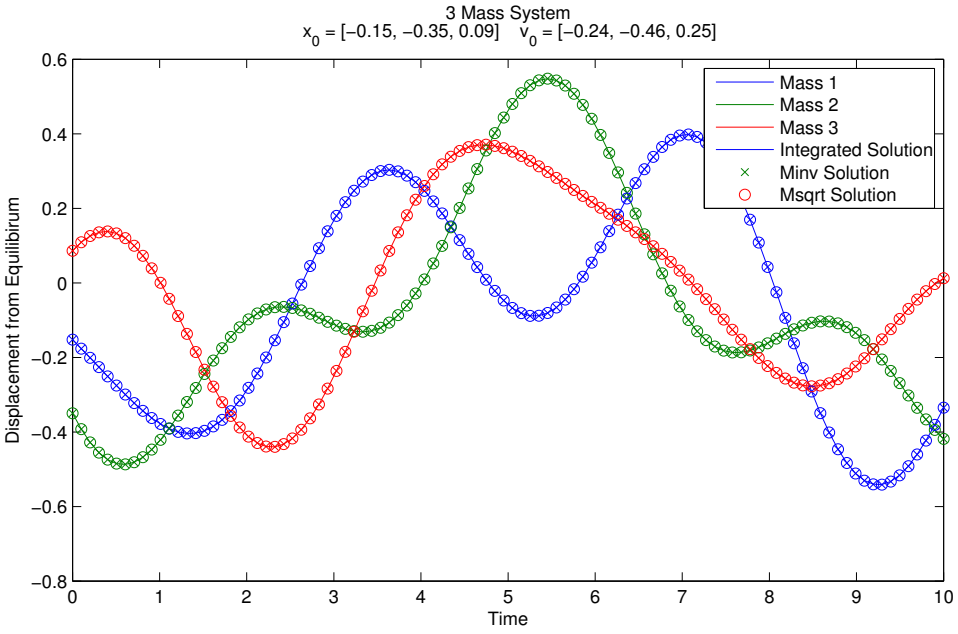

Figure 5: Solutions for a three mass system with a nonsingular K matrix using the methods described below that employ sine and cosine functions to find the motion when K may be singular.

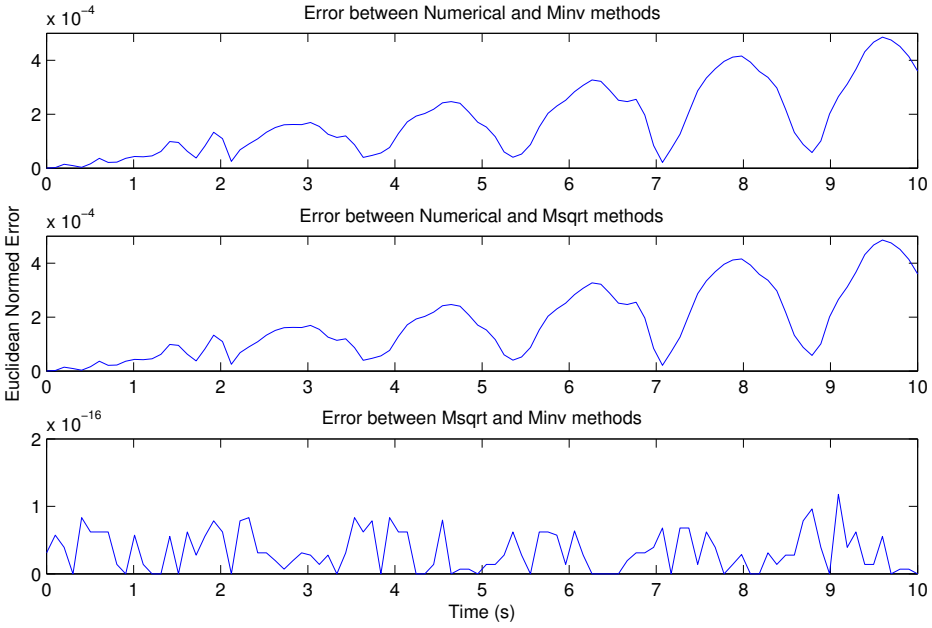

Figure 6: Associated error for the system in Figure 5

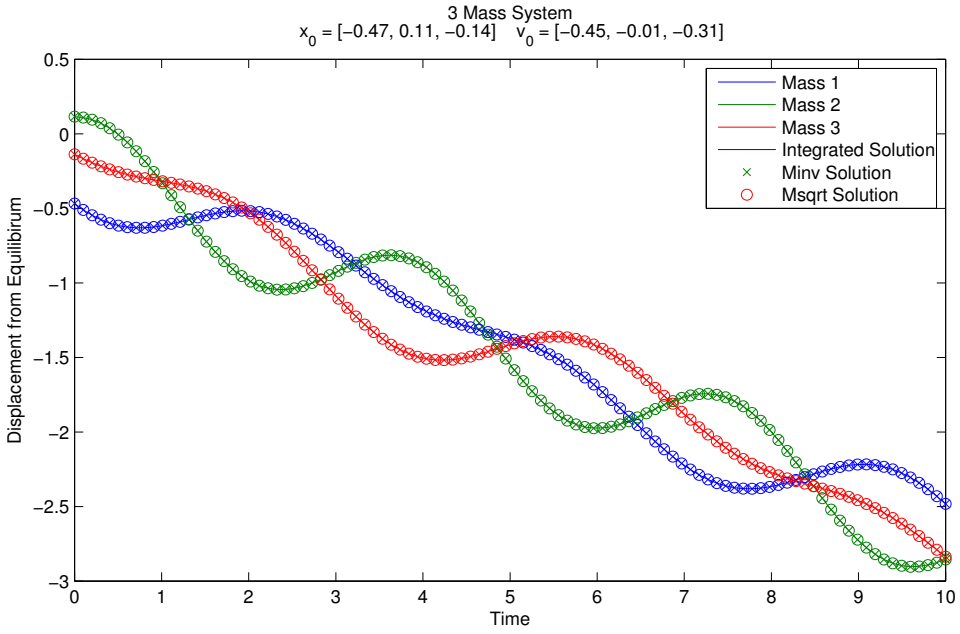

Figure 7: Solutions for a three mass system with a singular K matrix using the methods described below that employ sine and cosine functions to find the motion when K may be singular.

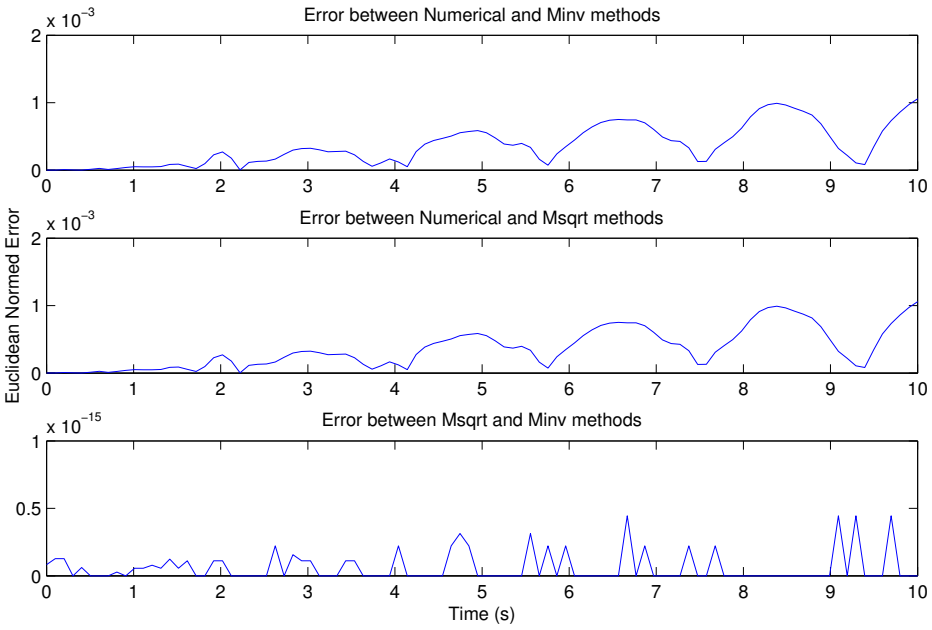

Figure 8: Associated error for the system in Figure 7

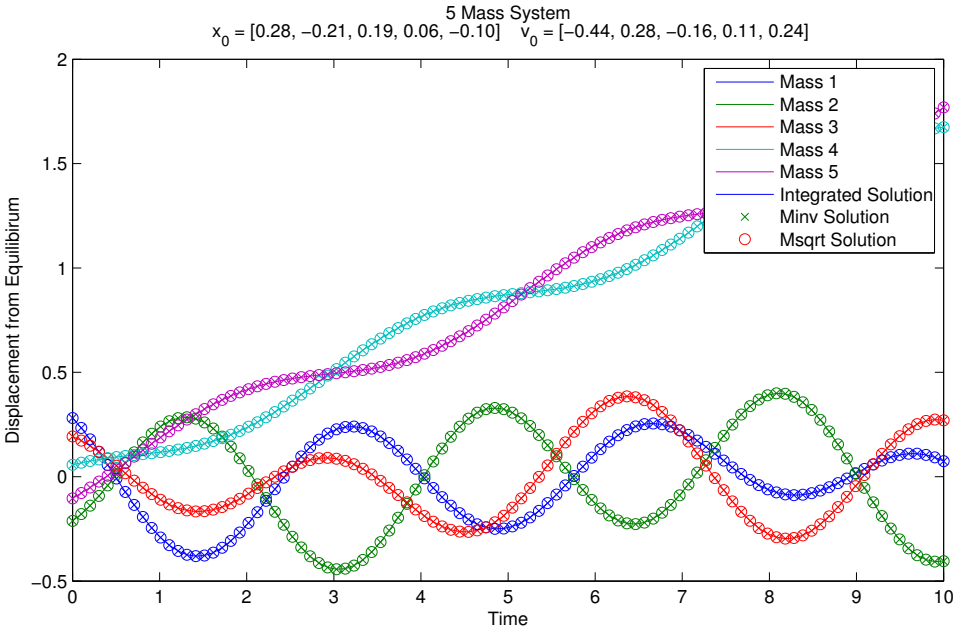

Figure 9: Solutions for a five mass system with a singular K matrix using the methods described below that employ sine and cosine functions to find the motion when K may be singular.

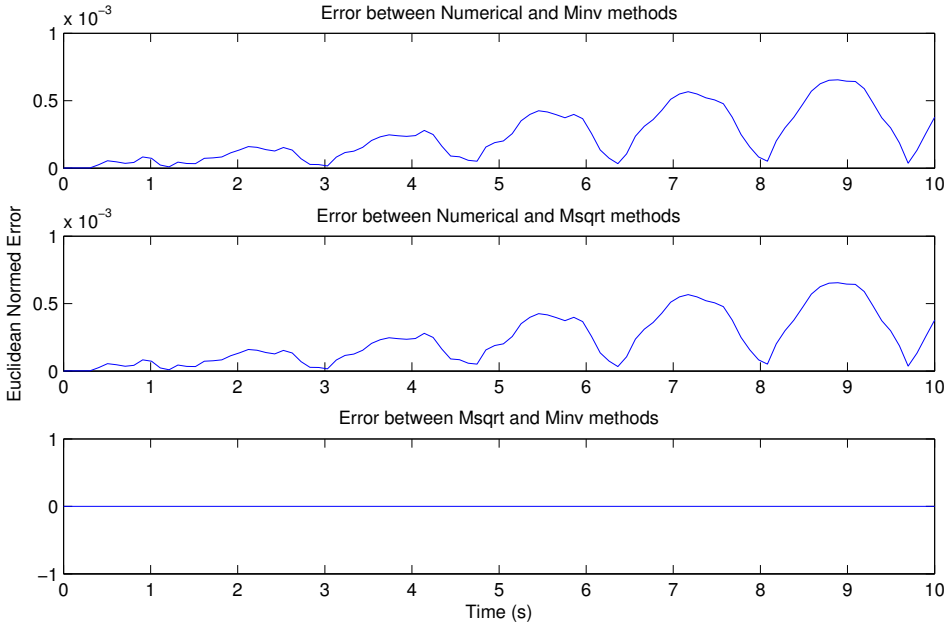

Figure 10: Associated error for the system in Figure 9

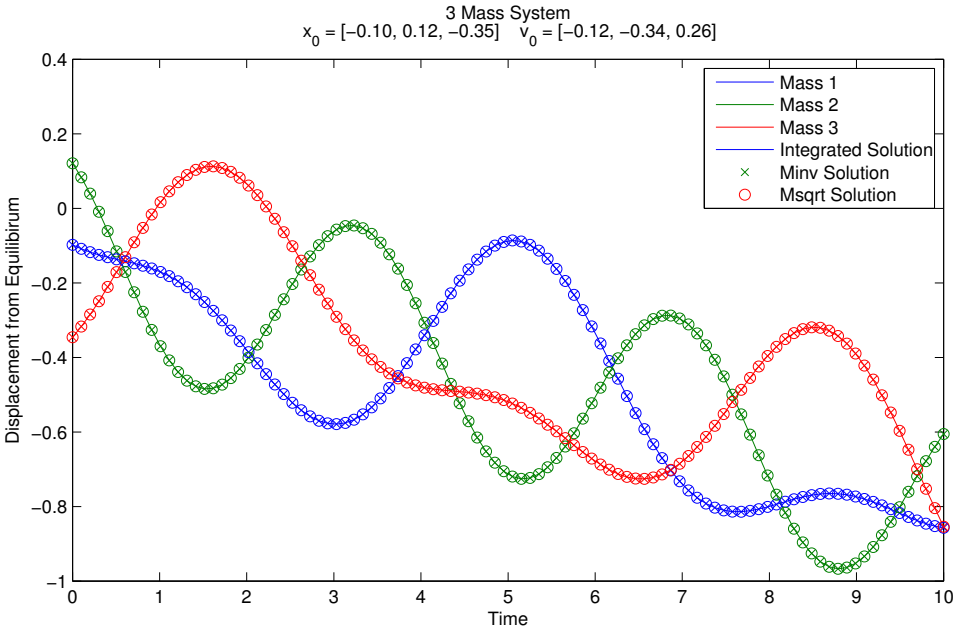

Figure 11: Solutions for a three mass system with a singular K matrix using the methods described below that employ exponential functions to find the motion when K may be singular.

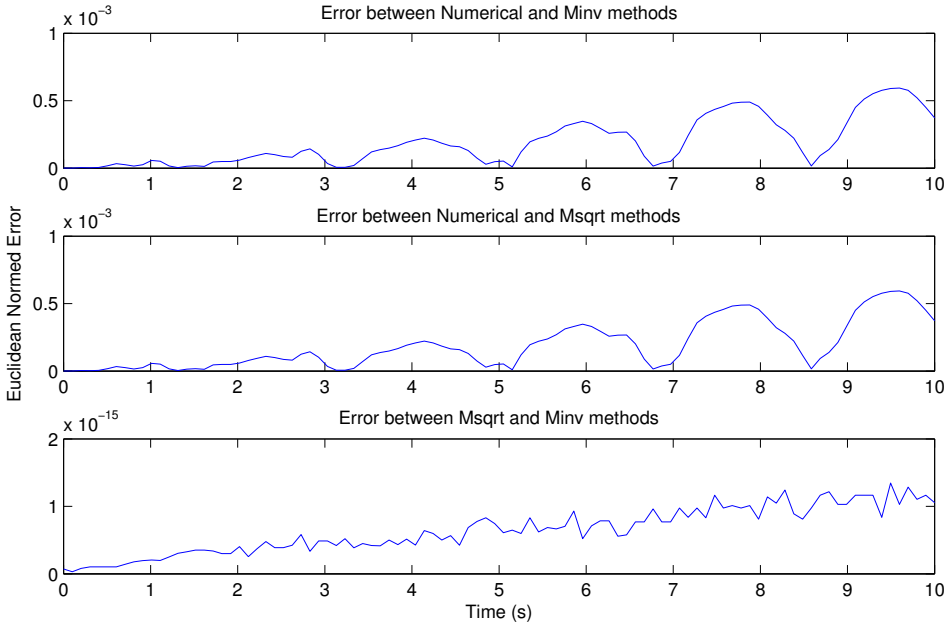

Figure 12: Associated error for the system in Figure 11

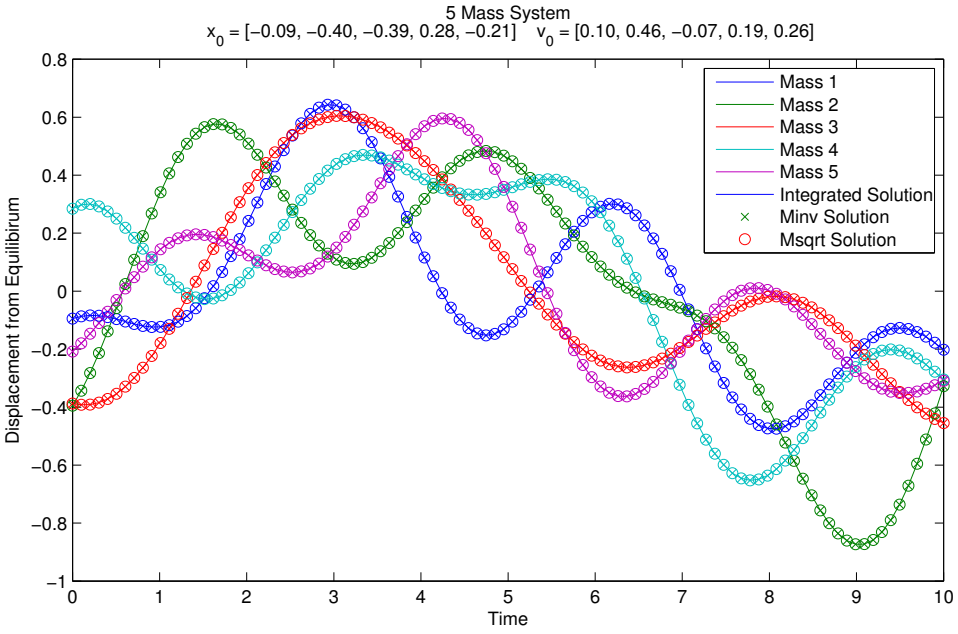

Figure 13: Solutions for a five mass system with a nonsingular K matrix using the methods described below that employ exponential functions to find the motion when K may be singular.

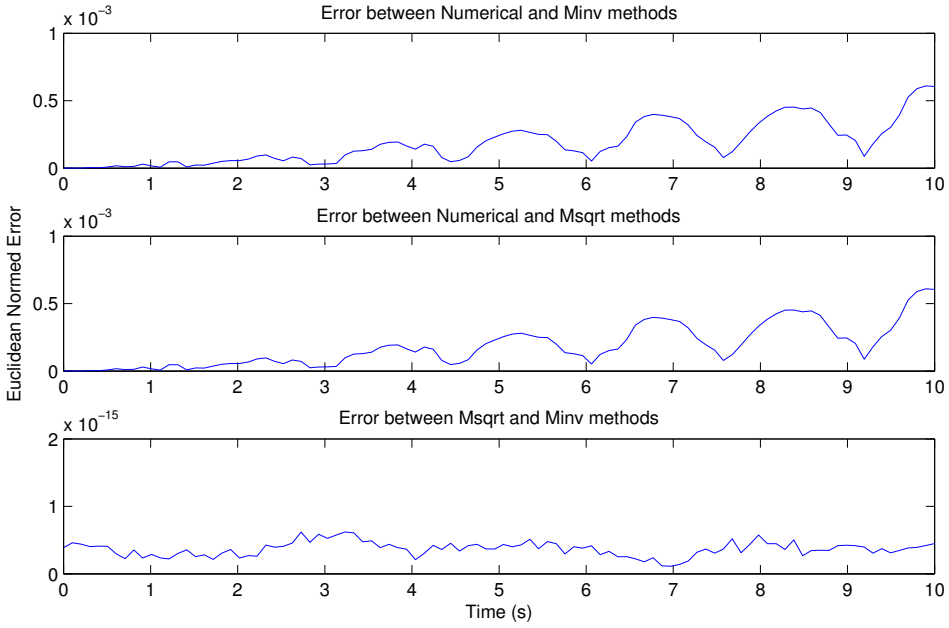

Figure 14: Associated error for the system in Figure 13

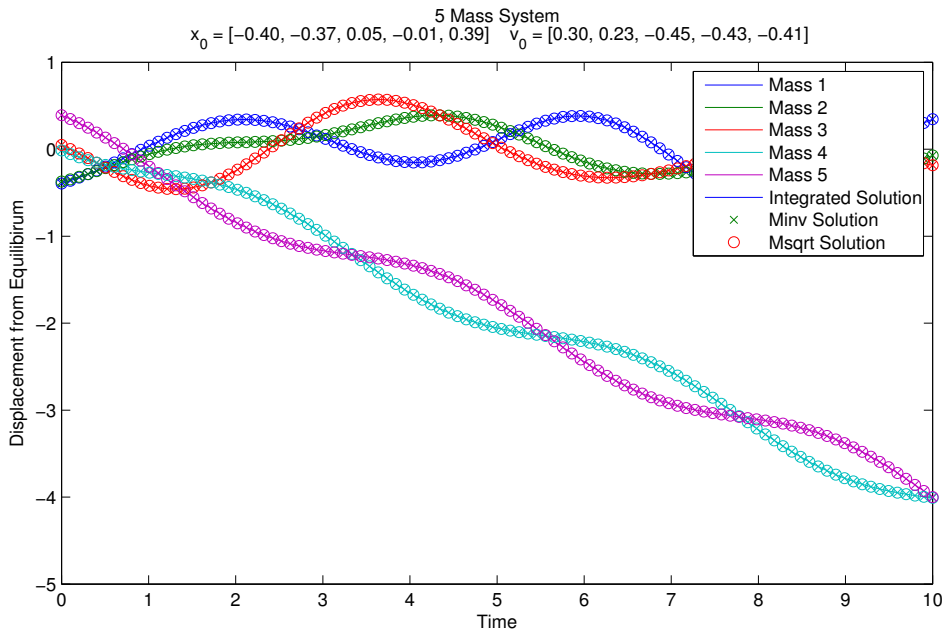

Figure 15: Solutions for a five mass system with a singular K matrix using the methods described below that employ exponential functions to find the motion when K may be singular.

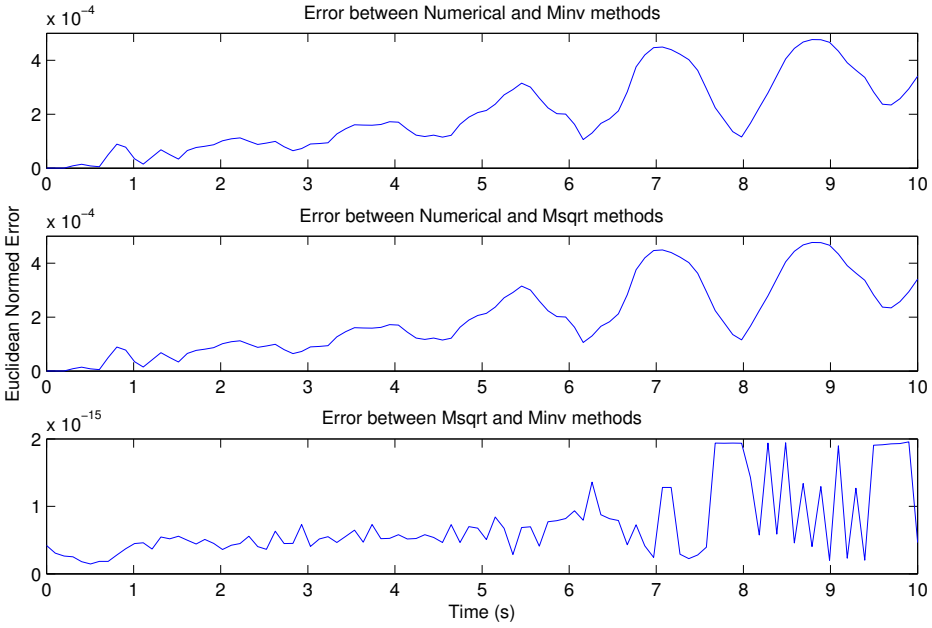

Figure 16: Associated error for the system in Figure 15

Matlab code used to produce the above plots:

```
% Driver file

% n = 3;
% K = [2 -1 0; -1 2 -1; 0 -1 2]; % For K-nonsingular cases
% K = [1 -1 0; -1 2 -1; 0 -1 1]; % For K-singular cases

% Alternatives
n = 5;
% K = [3 -1 -1 0 0; -1 2 0 -1 0; -1 0 2 0 -1; 0 -1 0 2 -1; 0 0 -1 -1 3];
K = [2 -1 0 0 0; -1 2 -1 0 0; 0 -1 2 0 0; 0 0 0 1 -1; 0 0 0 -1 1];

M = eye(n);
x0 = rand(n,1)-.5; % Random initial conditions from -.5 to .5
v0 = rand(n,1)-.5;

tspan = linspace(0,10,1000);

% Get solutions
[~,xNum] = SpringmassNUM(tspan,x0,v0,K,M);

% K-Nonsingular methods
% [~,xMinv] = SpringmassMinv(tspan,x0,v0,K,M);
% [~,xMsqrt] = SpringmassSqrtM(tspan,x0,v0,K,M);

% K-Singular methods
% [~,xMinv] = SpringmassMinvKSingular(tspan,x0,v0,K,M);
% [~,xMsqrt] = SpringmassSqrtMKSingular(tspan,x0,v0,K,M);

% Exponential methods (handles K-singular)
[~,xMinv] = SpringmassMinvExponential(tspan,x0,v0,K,M);
[~,xMsqrt] = SpringmassSqrtMExponential(tspan,x0,v0,K,M);

% Error Analysis
errNumMinv = xNum-xMinv;
errNumMinv = sqrt(sum(errNumMinv.^2,2));

errNumMsqrt = xNum-xMsqrt;
errNumMsqrt = sqrt(sum(errNumMsqrt.^2,2));

errMsqrtMinv = xMsqrt-xMinv;
errMsqrtMinv = sqrt(sum(errMsqrtMinv.^2,2));

% Plot
figure(1)
clf
hNUM = plot(tspan',xNum);
hold on
hMinv = plot(tspan',xMinv,'x');
hMsqrt = plot(tspan',xMsqrt,'o');

xlabel('Time')
ylabel('Displacement from Equilibirum')

vecPrint = '';
Names = cell(1,n);
for i = 1:n
    vecPrint = [vecPrint ', %3.2f']; %#ok<AGROW>
    Names{i} = sprintf('Mass %d',i);
end
vecPrint = vecPrint(3:end);

```

```
62 title({sprintf('%d Mass System',n),...
63     sprintf(['x_0 = [' vecPrint ']    v_0 = [' vecPrint ']'],x0,v0)})
64 legend([hNUM; hNUM(1); hMinv(2); hMsqrt(3)],Names{:},...
65     'Integrated Solution','Minv Solution','Msqrt Solution')
66
67 figure(2)
68 clf
69 subplot(3,1,1)
70 plot(tspan,errNumMinv)
71 title('Error between Numerical and Minv methods')
72 subplot(3,1,2)
73 plot(tspan,errNumMsqrt)
74 title('Error between Numerical and Msqrt methods')
75 ylabel('Euclidean Normed Error')
76 subplot(3,1,3)
77 plot(tspan,errMsqrtMinv)
78 title('Error between Msqrt and Minv methods')
79 xlabel('Time (s)')
80
81 % Animate
82 figure(3)
83 AnimateSolution(1:n,xNum,tspan)
```

e Animate the solution (using moving dots, circles or squares, your choice).

The following Matlab code will animate the solved spring mass system. It uses circles.

```
1  function AnimateSolution(x0,xarray,tspan)
2  % Animates an n-mass spring system's 1 dimensional motion in the current
3  % figure
4  %
5  % Inputs:
6  %   x0 - Array of equilibrium positions
7  %   xarray - Array of displacements for each mass in the system relative to
8  %            equilibrium. The i'th row corresponds to the positions at the
9  %            i'th time in tspan
10 %   tspan - array of times. tspan must include ALL time steps to be
11 %           plotted, not just the start and end times. The animation
12 %           assumes that all time steps in tspan are equal, and cannot
13 %           accurately animate a timestep shorter than .01s
14
15 nt = length(tspan);
16 n = size(xarray,2);
17
18 % Determine time step
19 dt = tspan(2)-tspan(1);
20
21 % Find absolute positions of the masses
22 xarray = xarray + repmat(x0(:)',nt,1);
23
24 xMax = max(max(xarray));
25 xMin = min(min(xarray));
26
27 % Set up plot
28 hArray = zeros(n,1);
29 hold on
30 for i = 1:n
31     hArray(i) = plot(xarray(1,i),0,'ko');
```

```
end
axis([xMin xMax -1 1]);

for i = 1:nt
    for j = 1:n
        set(hArray(j),'XData',xarray(i,j));
    end
    pause(dt)
end
```

2 Extra things for 5730 students

a Make the functions above work even if K is singular (has some modes with zero frequency)

a.a [tarray xarray] = SpringmassNUM(tspan, x0,v0,K,M)

No changes are required.

a.b [tarray xarray] = SpringmassMinv(tspan, x0,v0,K,M)

If there exists some i for which $\omega_i = 0$, then Equation 8 is no longer valid. Instead, we have

$$\vec{x}(t) = \sum_{i \in \{i|\lambda_i \neq 0\}} \vec{v}_i(A_i cos(\omega_i t) + B_i sin(\omega_i t)) + \sum_{i \in \{i|\lambda_i = 0\}} \vec{v}_i(A_i + B_i t) \tag{21}$$

Equation 9 is still valid, but Equation 10 must be replaced by considering two cases. Since we see that, in either case, $\vec{v}_0 \propto V\vec{B}$, we will create an intermittent variable $\vec{B}' = V^T\vec{v}_0$. $\vec{B}$ is related to $\vec{B}'$ by

$$B_i = \begin{cases} \frac{B_i'}{\omega_i} & \lambda_i \neq 0 \\ B_i' & \lambda_i = 0 \end{cases} \tag{22}$$

Equations 9, 21, and 22 can be combined for the solution to Equation 1.

```
function [tarray xarray] = SpringmassMinvKSingular(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using eigenvectors of M^-1*K. Handles singular K matrices.
%
% Inputs:
%   M,K - Matrices as defined in the equation above
%   x0 - vector of initial positions
%   v0 - vector of initial velocities
%   tspan - times at which to output the state of the system. tspan must be
%       a vector of lenght >= 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray - array of times corresponding to xarray
%   xarray - array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan
```

```
17
18 % Make IC vectors column
19 x0 = x0(:);
20 v0 = v0(:);
21
22 % Form timespan
23 if length(tspan) == 2
24     tarray = linspace(tspan(1),tspan(2));
25 else
26     tarray = tspan(:)';
27 end
28
29 [V,w2] = eig(M^-1*K);
30 w2 = diag(w2);
31 w2(abs(w2) < 1e-10) = 0;
32 w = sqrt(w2);
33
34 oM = w2~=0; % Oscillitory mode indices
35 rM = w2 == 0; % Rigid mode indices
36
37 % Solve for coefficients
38 A = V'*x0;
39 Bprime = V'*v0;
40 B = Bprime;
41 B(oM) = Bprime(oM)./w(oM); % Logical indexing, also used below
42
43 % Linear combination of the normal modes
44 % Because cos(0) = 1, we can use cospart even for the case of a rigid mode
45 cospart = repmat(A,1,length(tarray)).*cos(w*tarray);
46 % In order sin(0) = 0, but we need the rigid mode to have a t. We will have
47 % to break this out into two cases
48 sinpart(oM,:) = repmat(B(oM),1,length(tarray)).*sin(w(oM)*tarray);
49 sinpart(rM,:) = B(rM)*tarray;
50
51 x = V*(cospart + sinpart);
52 xarray = x';
53 end
```

a.c [tarray xarray] = SpringmassSqrtM(tspan, x0,v0,K,M)

The process of adjusting the modal solution for rigid modes is very similar to what we saw above. Equation 16 is no longer valid for all i. In particular, if we look at Equation 15 with $\lambda_i = 0$, we have $\ddot{r}_i = 0$, which implies a constant velocity solution. Our solution for the decoupled modes then becomes

$$r_i(t) = \begin{cases} C_i cos(\sqrt{\lambda_i}t) + D_i sin(\sqrt{\lambda_i}t) & \lambda_i \neq 0 \\ C_i + D_i t & \lambda_i = 0 \end{cases} \tag{23}$$

We now see that Equation 19 is still valid, while Equation 20 must be reformulated for two cases. Much like Equation 22, we have

$$D_i = \begin{cases} \frac{\dot{r}_{0,i}}{\sqrt{\lambda_i}} & \lambda_i \neq 0 \\ \dot{r}_{0,i} & \lambda_i = 0 \end{cases} \tag{24}$$

and we once again apply the transformation in Equation 18 to solve Equation 1.

```
function [tarray xarray] = SpringmassSqrtMKSingular(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using modal coordinates. Handles singular K matrices
%
% Inputs:
%   M,K — Matrices as defined in the equation above
%   x0 — vector of initial positions
%   v0 — vector of initial velocities
%   tspan — times at which to output the state of the system. tspan must be
%       a vector of lenght ≥ 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray — array of times corresponding to xarray
%   xarray — array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);

% For time span
if length(tspan) == 2
    tarray = linspace(tspan(1),tspan(2));
else
    tarray = tspan(:)';
end

Msqrt = M^(-1/2);
Ktwid = Msqrt*K*Msqrt;
[P,w2] = eig(Ktwid);
w2 = diag(w2);
w2(abs(w2) < 1e-10) = 0;
w = sqrt(w2);

oM = w2≠0; % Oscillitory mode indices
rM = w2 == 0; % Rigid mode indices

% Transform ICs
r0 = P'*Msqrt^-1*x0;
rd0 = P'*Msqrt^-1*v0;

% Solve for coefficients
C = r0;
D = rd0;
D(oM) = rd0(oM)./w(oM); % Logical indexing. Also used below.

% Find solution for each 1DOF oscillator
% Because cos(0) = 1, we can use cospart even for the case of a rigid mode
cospart = repmat(C,1,length(tarray)).*cos(w*tarray);
% In order sin(0) = 0, but we need the rigid mode to have a t. We will have
% to break this out into two cases
sinpart(oM,:) = repmat(D(oM),1,length(tarray)).*sin(w(oM)*tarray);
sinpart(rM,:) = D(rM)*tarray;

r = cospart + sinpart;

% Transform back to x-coordinates
x = M*P*r;
xarray = x';
end
```

b Instead of using cosine and sine for the normal modes, use exponentials and do complex math to solve for the initial conditions.

b.a [tarray xarray] = SpringmassNUM(tspan, x0,v0,K,M)

Not Applicable

b.b [tarray xarray] = SpringmassMinv(tspan, x0,v0,K,M)

We can begin from Equation 5, as we did for the solution using sines and cosines. This time, we assume a solution of the form

$$\vec{x}(t) = x(\omega t)\vec{v} = ae^{\omega t}\vec{v} \tag{25}$$

This results in the eigenvalue problem

$$(\omega^2 I + M^{-1}K)\vec{v} = 0 \tag{26}$$

which differs from what we had in Equation 7 by only a sign. As a result, we now have n eigenpairs $(\lambda_i = \omega_i^2 < 0, \vec{v}_i)$, which implies that the ω_i's are imaginary. Further, while we were previously able to take advantage of the odd/even properties of the sine and cosine functions to ignore the sign on ω_i, we must now take into account both the positive and negative values. We now write our general solution:

$$\vec{x}(t) = \sum_{i=1}^{n} \vec{v}_i(a_i e^{\omega_i t} + b_i e^{-\omega_i t}) \tag{27}$$

If we wish to also factor in the case where K is singular, we note that $\omega_i = -0 = +0$ behaves like a repeated root, and we find:

$$\vec{x}(t) = \sum_{i\in\{i|\lambda_i\neq 0\}} \vec{v}_i(a_i e^{\omega_i t} + b_i e^{-\omega_i t}) + \sum_{i\in\{i|\lambda_i=0\}} \vec{v}_i(a_i + b_i t) \tag{28}$$

We note that the coefficients a_i and b_i are generally complex, while those having $i \in \{i|\lambda_i = 0\}$ being purely real, since $\vec{x}(t)$ is purely real. Applying our initial conditions, we have:

$$\vec{x}_0 = \sum_{i\in\{i|\lambda_i\neq 0\}} \vec{v}_i(a_i + b_i) + \sum_{i\in\{i|\lambda_i=0\}} \vec{v}_i a_i \tag{29}$$

$$\vec{v}_0 = \sum_{i\in\{i|\lambda_i\neq 0\}} \vec{v}_i\omega_i(a_i - b_i) + \sum_{i\in\{i|\lambda_i=0\}} \vec{v}_i b_i \tag{30}$$

Concerning only $i \in \{i|\lambda_i \neq 0\}$, we can see from Equation 29 that, since $\vec{x}_0$ is real, $a_i + b_i$ is real. Equivalently, we can say that the imaginary parts of a_i and b_i are opposite. Equation 30 similarly tells us that $a_i - b_i$ is purely imaginary, since ω_i is imaginary and $\vec{v}_0$ real.

Equivalently, the real parts of a_i and b_i are equal. As such, we see that a_i and b_i are complex conjugates. The equations are now:

$$\vec{x}(t) = \sum_{i \in \{i|\lambda_i \neq 0\}} \vec{v}_i (a_i e^{\omega_i t} + a_i^* e^{-\omega_i t}) + \sum_{i \in \{i|\lambda_i = 0\}} \vec{v}_i (a_i + b_i t) \tag{31}$$

$$\vec{x}_0 = \sum_{i \in \{i|\lambda_i \neq 0\}} \vec{v}_i 2Re[a_i] + \sum_{i \in \{i|\lambda_i = 0\}} \vec{v}_i a_i = V^* \vec{a} \tag{32}$$

$$\vec{v}_0 = \sum_{i \in \{i|\lambda_i \neq 0\}} \vec{v}_i \omega_i 2Im[a_i] + \sum_{i \in \{i|\lambda_i = 0\}} \vec{v}_i b_i = V^{**} \vec{b} \tag{33}$$

The bookkeeping is somewhat tricky, but V^* and V^{**} both have columns that are multiples of linearly independent eigenvectors, and thus they are invertible. We can solve for $\vec{a}$, which contains the real parts of those a_i corresponding to oscillatory modes and the a_i for rigid modes, and $\vec{b}$, which contains the imaginary parts of those a_i corresponding to oscillatory modes and the b_i for rigid modes, by simple matrix operations. It is worth noting that a similar formulation would allow us to express Equation 22 as a matrix operation.

```
function [tarray xarray] = SpringmassMinvExponential(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using eigenvectors of M^-1*K. Handles singular K matrices. Uses
% exponential solutions.
%
% Inputs:
%   M,K - Matrices as defined in the equation above
%   x0 - vector of initial positions
%   v0 - vector of initial velocities
%   tspan - times at which to output the state of the system. tspan must be
%       a vector of lenght >= 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray - array of times corresponding to xarray
%   xarray - array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);

% Form timespan
if length(tspan) == 2
    tarray = linspace(tspan(1),tspan(2));
else
    tarray = tspan(:)';
end

[V,w2] = eig(M^-1*K);
w2 = diag(w2);
w2(abs(w2) < 1e-10) = 0;
w = sqrt(-w2);

oM = w2~=0; % Oscillitory mode indices
rM = w2 == 0; % Rigid mode indices

```

```
% Solve for coefficients
Vstar(:,oM) = 2*V(:,oM); % Logical indexing. Used throughout.
Vstar(:,rM) = V(:,rM);
a = Vstar^-1*x0;

Vsstar(:,oM) = 2*V(:,oM).*repmat(w(oM)'/1j,length(w),1);
Vsstar(:,rM) = V(:,rM);
b = Vsstar^-1*v0;

% Linear combination of the normal modes
pospart = zeros(length(w),length(tarray));
pospart(oM,:) = repmat(a(oM) + b(oM)*1j ,1,length(tarray)).*exp(w(oM)*tarray);

negpart = zeros(length(w),length(tarray));
negpart(oM,:) = repmat(a(oM) - b(oM)*1j ,1,length(tarray)).*exp(-w(oM)*tarray);

constpart = zeros(length(w),length(tarray));
constpart(rM,:) = repmat(a(rM),1,length(tarray));

linpart = zeros(length(w),length(tarray));
linpart(rM,:) = b(rM)*tarray;

x = V*(pospart + negpart + constpart + linpart);
xarray = x';
end
```

b.c [tarray xarray] = SpringmassSqrtM(tspan, x0,v0,K,M)

We now start with Equation 15, and now assume solutions

$$r_i(t) = c_i e^{\omega_i t} \tag{34}$$

and find that $\omega_i = \pm\sqrt{-\lambda_i}$. If we apply the above process used for the MDOF system to each of our decoupled SDOF modes, we will find the solutions:

$$r_i(t) = \begin{cases} c_i e^{\omega_i t} + c_i^* e^{-\omega_i t} & \lambda_i \neq 0 \\ c_i + d_i t & \lambda_i = 0 \end{cases} \tag{35}$$

and by applying the initial conditions we get:

$$r_{0,i} = \begin{cases} 2Re[c_i] & \lambda_i \neq 0 \\ c_i & \lambda_i = 0 \end{cases} \tag{36}$$

$$\dot{r}_{0,i} = \begin{cases} 2\omega_i Im[c_i] & \lambda_i \neq 0 \\ d_i & \lambda_i = 0 \end{cases} \tag{37}$$

which can be solved in each individual case by simple rearrangement. Once again, we apply the transformation in Equation 18.

```
function [tarray xarray] = SpringmassSqrtMExponential(tspan,x0,v0,K,M)
% Solves for the motion of the system defined by:
%        M(xddot) + K x = 0
% using modal coordinates. Handles singular K matrices. Uses
% exponential solutions.
```

```
%
% Inputs:
%   M,K — Matrices as defined in the equation above
%   x0 — vector of initial positions
%   v0 — vector of initial velocities
%   tspan — times at which to output the state of the system. tspan must be
%       a vector of lenght ≥ 2. The simulation will begin at time in the
%       first element of tspan and end at the time in the last element.
% Outputs:
%   tarray — array of times corresponding to xarray
%   xarray — array of system states. The i'th row corresponds to the state
%            of the system at the i'th element in tspan

% Make IC vectors column
x0 = x0(:);
v0 = v0(:);

% For time span
if length(tspan) == 2
    tarray = linspace(tspan(1),tspan(2));
else
    tarray = tspan(:)';
end

Msqrt = M^(-1/2);
Ktwid = Msqrt*K*Msqrt;
[P,w2] = eig(Ktwid);
w2 = diag(w2);
w2(abs(w2) < 1e-10) = 0;
w = sqrt(-w2);

oM = w2≠0; % Oscillitory mode indices
rM = w2 == 0; % Rigid mode indices

% Transform ICs
r0 = P'*Msqrt^-1*x0;
rd0 = P'*Msqrt^-1*v0;

% Solve for coefficients
c(oM) = r0(oM)/2; % Logical indexing, used throughout
c(rM) = r0(rM);
c = c(:); % logical indexing creates a row vector, make it a column
d(oM) = rd0(oM)./(2*w(oM)*1j);
d(rM) = rd0(rM);
d = d(:); % logical indexing creates a row vector, make it a column

% Solution for each oscillator
pospart = zeros(length(w),length(tarray));
pospart(oM,:) = repmat(c(oM) + d(oM)*1j, 1,length(tarray)).*exp(w(oM)*tarray);

negpart = zeros(length(w),length(tarray));
negpart(oM,:) = repmat(c(oM) - d(oM)*1j, 1,length(tarray)).*exp(-w(oM)*tarray);

constpart = zeros(length(w),length(tarray));
constpart(rM,:) = repmat(c(rM),1,length(tarray));

linpart = zeros(length(w),length(tarray));
linpart(rM,:) = d(rM)*tarray;

r = pospart + negpart + constpart + linpart;

% Transform back to x-coordinates
x = M*P*r;
xarray = x';
end
```