ME 5390 - Homework 6

James Grisham

October 31, 2013

Problem 1

Problem Statement

For a 3 degree of freedom spring-mass system, the stiffness and mass matrices are

$$\mathbf{M} = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix} \quad \text{and} \quad \mathbf{K} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \tag{1}$$

For $m_1 = 2$ kg, $m_2 = 1$ kg, and $m_3 = 4$ kg, $k_1 = 1000$ N/m, $k_2 = 400$ N/m, and $k_3 = 500$ N/m,

(a). Compute the derivatives $d\lambda_1/dk_1$ and $d\lambda_2/dk_3$ symbolically.

(b). Compute the above derivatives numerically by

- Forward difference
- Central difference
- Complex value method

using the following step sizes: 0.1, 0.01, 1×10^{-4}, 1×10^{-8}, 1×10^{-16}, and 1×10^{-100}.

(c). Calculate $\lambda_2(k_3)$ for 400 N/m $\leq k_3 \leq$ 600 N/m using the following 2 methods:

- Exact solution (solve the eigenvalue problem for each value of k_3).
- Using first order approximation:

$$\lambda_2 \approx \lambda_2(500) + \frac{d\lambda_2}{dk_3}(k_3 - 500)$$

Plot exact and approximate solutions vs k_3 on the same figure.

Solution

Parts A and B

The symbolic operations were accomplished using MATLAB. The code can be seen in the appendix.

The eigenvalues, eigenvectors, exact and approximate derivatives as computed by MATLAB are given below:

```
================================================================
Eigenvectors and eigenvalues
================================================================

Eigenvalues =
   1.0e+03 *

   0.040353961833781
   0.542339335518695
   1.142306702647524

Eigenvectors =
   0.096327617310803  -0.592930284826365  -0.373034405221487
   0.317710655625315  -0.467408913484071   0.824978088737981
   0.469175318934346   0.139996662698697  -0.101367916699923

================================================================
Symbolic Solution
================================================================

dLam1/dk1 = 0.009279009856776
dLam2/dk3 = 0.368941533977921

================================================================
Forward Difference Solution
================================================================

step = 1.0e-01      dL1/dk1 = 0.0092782429
step = 1.0e-02      dL1/dk1 = 0.0092789332
step = 1.0e-04      dL1/dk1 = 0.0092790092
step = 1.0e-08      dL1/dk1 = 0.0092775565
step = 1.0e-16      dL1/dk1 = 0.0000000000
step = 1.0e-100     dL1/dk1 = 0.0000000000

step = 1.0e-01      dL2/dk3 = 0.3688904551
step = 1.0e-02      dL2/dk3 = 0.3689364258
step = 1.0e-04      dL2/dk3 = 0.3689414837
step = 1.0e-08      dL2/dk3 = 0.3689592631
step = 1.0e-16      dL2/dk3 = 0.0000000000
step = 1.0e-100     dL2/dk3 = 0.0000000000

================================================================
Central Difference Solution
================================================================

step = 1.0e-01      dL1/dk1 = 0.0092790099
step = 1.0e-02      dL1/dk1 = 0.0092790099
step = 1.0e-04      dL1/dk1 = 0.0092790099
step = 1.0e-08      dL1/dk1 = 0.0092796881
step = 1.0e-16      dL1/dk1 = 0.0000000000
step = 1.0e-100     dL1/dk1 = 0.0000000000

step = 1.0e-01      dL2/dk3 = 0.3689415377
step = 1.0e-02      dL2/dk3 = 0.3689415340
step = 1.0e-04      dL2/dk3 = 0.3689415331
step = 1.0e-08      dL2/dk3 = 0.3689535788
step = 1.0e-16      dL2/dk3 = 0.0000000000
step = 1.0e-100     dL2/dk3 = 0.0000000000
```

```
==================================================================
Complex Value Method
==================================================================

step = 1.0e-01      dL1/dk1 = 0.0092790098
step = 1.0e-02      dL1/dk1 = 0.0092790099
step = 1.0e-04      dL1/dk1 = 0.0092790099
step = 1.0e-08      dL1/dk1 = 0.0092790099
step = 1.0e-16      dL1/dk1 = 0.0092790099
step = 1.0e-100     dL1/dk1 = 0.0092790099

step = 1.0e-01      dL2/dk3 = 0.3689415303
step = 1.0e-02      dL2/dk3 = 0.3689415339
step = 1.0e-04      dL2/dk3 = 0.3689415340
step = 1.0e-08      dL2/dk3 = 0.3689415340
step = 1.0e-16      dL2/dk3 = 0.3689415340
step = 1.0e-100     dL2/dk3 = 0.3689415340
```

With a step size smaller than 1×10^{-16}, the results were questionable. I believe this step size is too small (i.e., it is smaller than the machine precision). Typing eps into the MATLAB command prompt displays a machine precision of 2.2204×10^{-16}.

Part C

A comparison between the exact solution and the first-order approximation is provided in Figure 1. The first-order approximation seems adequate.

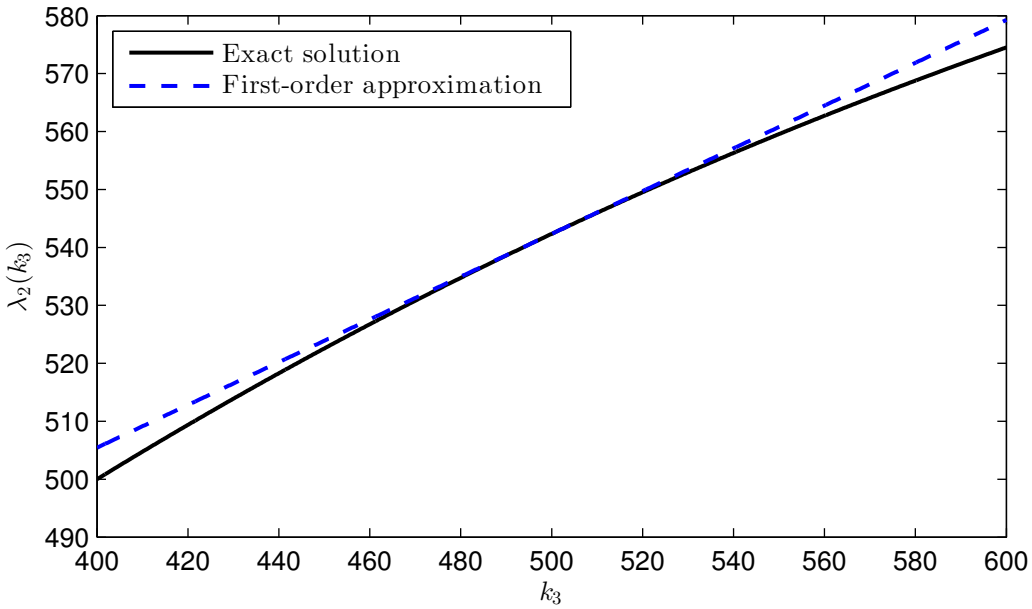

Figure 1. Comparison between exact solution of $\lambda_2(k_3)$ and the first-order approximation.

Problem 2

Problem Statement

For the three degree of freedom system defined in Problem 1, for $m_1 = 2$ kg, $m_2 = 1$ kg, $m_3 = 4$ kg, and $k_2 = 400$ N/m, find k_1 and k_3 to minimize $k_1 + k_3$ under the constraint that $\lambda_1 = 16$ (rad/s)^2. Solve the problem using a numerical method (e.g., fmincon, CVM_fmincon, etc). Validate your solution by finding the exact solution using the method of Lagrange multipliers and the formulas of eigenvalue derivatives in terms of the eigenvectors. Do not use the explicit form of the solution.

Solution

In this case, the objective function is given by

$$f(\mathbf{X}) = k_1 + k_3 \tag{2}$$

and subject to the equality constraint

$$\lambda_1 = 16 \text{ (rad/s)}^2 \tag{3}$$

where an expression for $\lambda_1(k_1, k_3)$ is determined using symbolic computations.

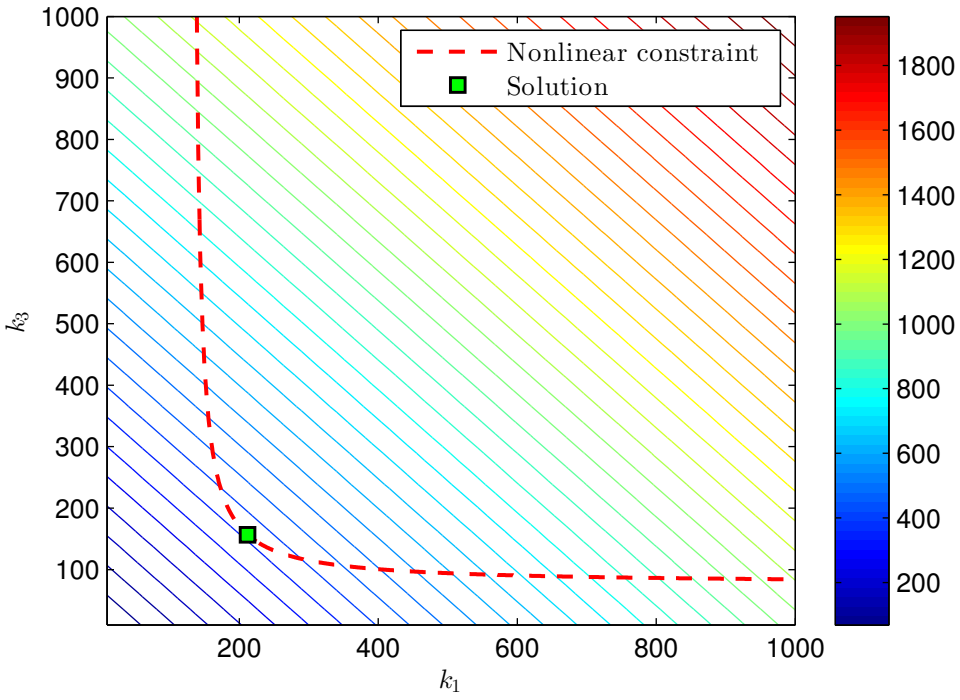

Figure 2. Contours of the objective function with nonlinear constraint and solution.

fmincon solution

```
With k_1 = 212.00 and k_3 = 156.80, the first eigenvalue is 16.00.
```

Now, the fmincon solution will be verified by deriving the analytical solution using Lagrange multipliers and the derivatives of the eigenvalues in terms of the eigenvectors. Given the objective function and nonlinear equality constraints given in (2) and (3), the Lagrangian is given by

$$\mathcal{L}(k_1, k_3, \mu) = f(k_1, k_3) + \mu \left[\lambda_1(k_1, k_3) - 16\right] \tag{4}$$

which can be simplified to

$$\mathcal{L}(k_1, k_3, \mu) = k_1 + k_3 + \mu \left[\lambda_1(k_1, k_3) - 16\right] \tag{5}$$

Now, the necessary conditions for the solution to be an optimum are given by

$$\frac{\partial \mathcal{L}}{\partial k_1} = 0 \tag{6a}$$
$$\frac{\partial \mathcal{L}}{\partial k_3} = 0 \tag{6b}$$
$$\frac{\partial \mathcal{L}}{\partial \mu} = 0 \tag{6c}$$

Applying (6) to (5),

$$\frac{\partial \mathcal{L}}{\partial k_1} = 1 + \mu \frac{\partial}{\partial k_1} \left(\lambda_1(k_1, k_3)\right) = 0 \tag{7a}$$
$$\frac{\partial \mathcal{L}}{\partial k_3} = 1 + \mu \frac{\partial}{\partial k_3} \left(\lambda_1(k_1, k_3)\right) = 0 \tag{7b}$$
$$\frac{\partial \mathcal{L}}{\partial \mu} = \lambda_1(k_1, k_3) - 16 = 0 \tag{7c}$$

where the derivatives $\partial\lambda_1/\partial k_1$ and $\partial\lambda_1/\partial k_3$ will be determined using the eigenvectors as follows.

The stiffness matrix is given by

$$\mathbf{K} = \begin{bmatrix} k_1 + k_2 & -k_2 & 0 \\ -k_2 & k_2 + k_3 & -k_3 \\ 0 & -k_3 & k_3 \end{bmatrix} \tag{8}$$

The eigenvalue problem is

$$\mathbf{K}\boldsymbol{\Phi} = \mathbf{M}\boldsymbol{\Phi}\boldsymbol{\Lambda} \tag{9}$$

Now, the derivatives can be approximated by

$$\frac{\partial \lambda_1}{\partial k_1} = \frac{\boldsymbol{\phi}_1^\top \frac{d\mathbf{K}}{dk_1} \boldsymbol{\phi}_1}{\boldsymbol{\phi}_1^\top \mathbf{M} \boldsymbol{\phi}_1} \tag{10a}$$
$$\frac{\partial \lambda_1}{\partial k_3} = \frac{\boldsymbol{\phi}_1^\top \frac{d\mathbf{K}}{dk_3} \boldsymbol{\phi}_1}{\boldsymbol{\phi}_1^\top \mathbf{M} \boldsymbol{\phi}_1} \tag{10b}$$

Manipulating (7a) and (7b) yields

$$\boxed{\frac{\partial}{\partial k_1} \left(\lambda_1(k_1, k_3)\right) = \frac{\partial}{\partial k_3} \left(\lambda_1(k_1, k_3)\right)} \tag{11}$$

Similarly,

$$\boxed{\lambda_1(k_1, k_3) = 16} \tag{12}$$

The equations given in (11) and (12) must be satisfied for the Kuhn-Tucker conditions to be satisfied. The first eigenvalue can be normalized as follows

$$\boldsymbol{\phi}_1 = \begin{Bmatrix} 1 \\ \phi_2 \\ \phi_3 \end{Bmatrix} \tag{13}$$

Inserting (13) into (11) and solving yields

$$\phi_2 = \phi_3 - 1 \tag{14}$$

Next, the eigenvalue problem is solved to determine the other three unknowns (ϕ_3, k_1, and k_3).

$$(\mathbf{K} - \lambda_1 \mathbf{M})\, \boldsymbol{\phi}_1 = \{0\} \tag{15}$$

The solution was accomplished using Mathematica (code in appendix).

Results:

$$\boxed{k_1 = 212 \text{ N/m} \quad \text{and} \quad k_3 = 156.8 \text{ N/m}} \tag{16}$$

The result in (16) agrees perfectly with the numerical result determined by fmincon.

Problem 3

Problem Statement

For the damped vibration absorber shown below, the mass, stiffness, and damping matrices are given by

$$\mathbf{M} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{K} = \begin{bmatrix} k_a + 100 & -k_a \\ -k_a & k_a \end{bmatrix} \quad \text{and} \quad \mathbf{C} = \begin{bmatrix} c_a & -c_a \\ -c_a & c_a \end{bmatrix}$$

The equation of motion is given by

$$\mathbf{M}\ddot{\mathbf{u}} + \mathbf{C}\dot{\mathbf{u}} + \mathbf{K}\mathbf{u} = 0$$

subject to the initial conditions in the time domain

$$\mathbf{u}(0) = \begin{Bmatrix} 0 \\ 0 \end{Bmatrix} \quad \text{and} \quad \dot{\mathbf{u}}(0) = \begin{Bmatrix} 0 \\ 10 \end{Bmatrix}$$

and in the frequency domain

$$\mathbf{p} = \begin{Bmatrix} 1 \\ 0 \end{Bmatrix}$$

The steady-state responses are given by

$$\mathbf{U} = \mathbf{Z}^{-1}\mathbf{p}$$

where

$$\mathbf{Z} = \mathbf{K} - \omega^2\,\mathbf{M} + i\,\omega\,\mathbf{C}$$

Find k_a and c_a to minimize

- The time objective function.
- The frequency domain objective function.

Plot the responses for the initial and optimal designs for each case. You need to define the bounds of the design variables.

The time domain objective function is given by

$$f_1 = \int_0^{10} [u_1(t)]^2\,dt$$

and the frequency domain objective function is

$$f_2 = \int_0^{10} [U_1(\Omega)]^2\,d\Omega$$

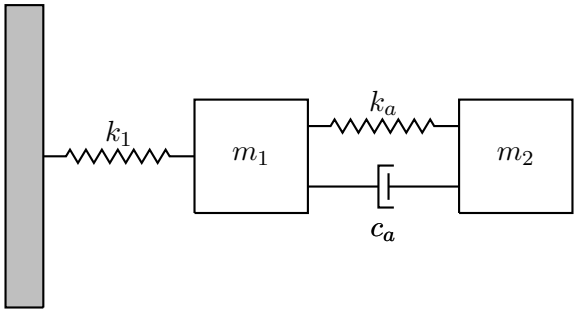

Figure 3. Damped vibration absorber.

Solution

Time-domain solution

To get an idea of the bounds on the design variables, a contour plot was created (see Figure 4). Each point on the contour plot represents a numerical solution to the differential equation that describes the dynamics for a time interval from 0 to 10 seconds.

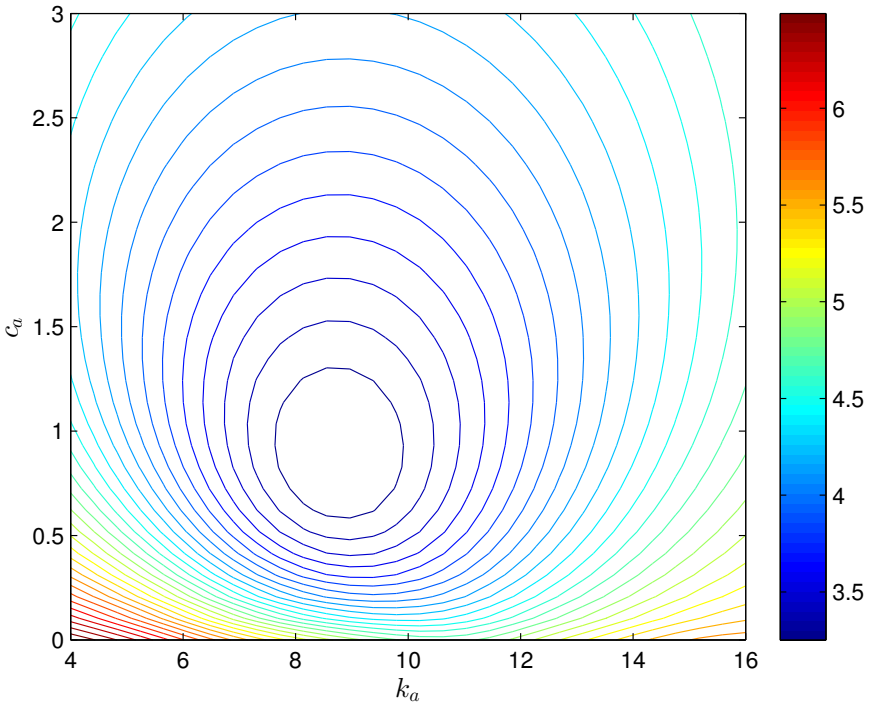

Figure 4. Contours of the objective function with $0 \leq t \leq 10$ sec.

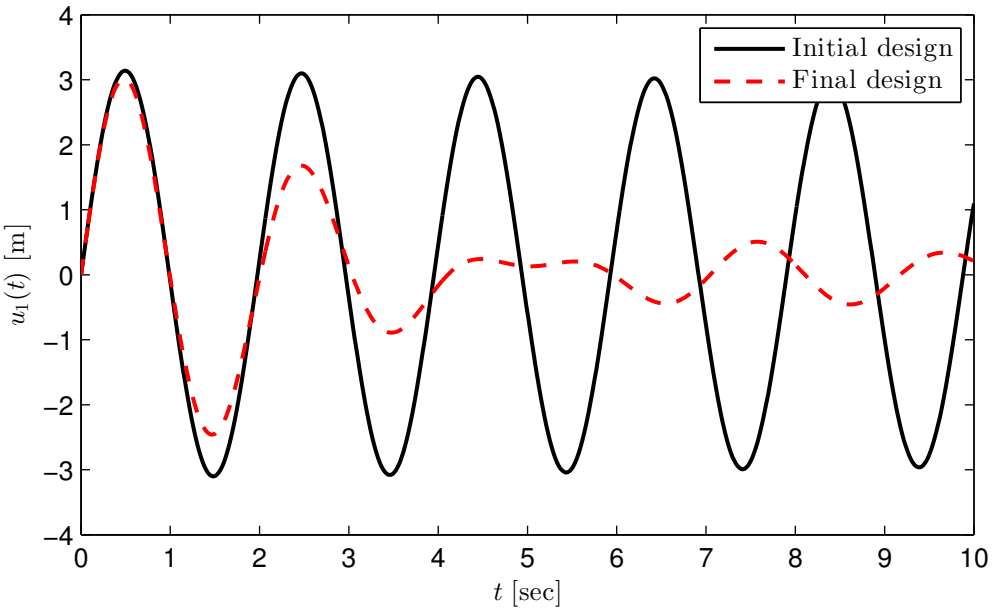

Figure 5. Plot of transient data for initial and final design.

The optimum solution as determined by fmincon is

$$\boxed{k_a = 8.7489 \quad \text{and} \quad c_a = 0.88295} \tag{17}$$

Frequency-domain solution

For the frequency-domain solution, the frequency response function (FRF) was computed for each set of design variables. Because this process did not require integrating an ODE, the code ran faster. Again, a contour plot of the objective function was created to get an idea of the bounds on the design variables.

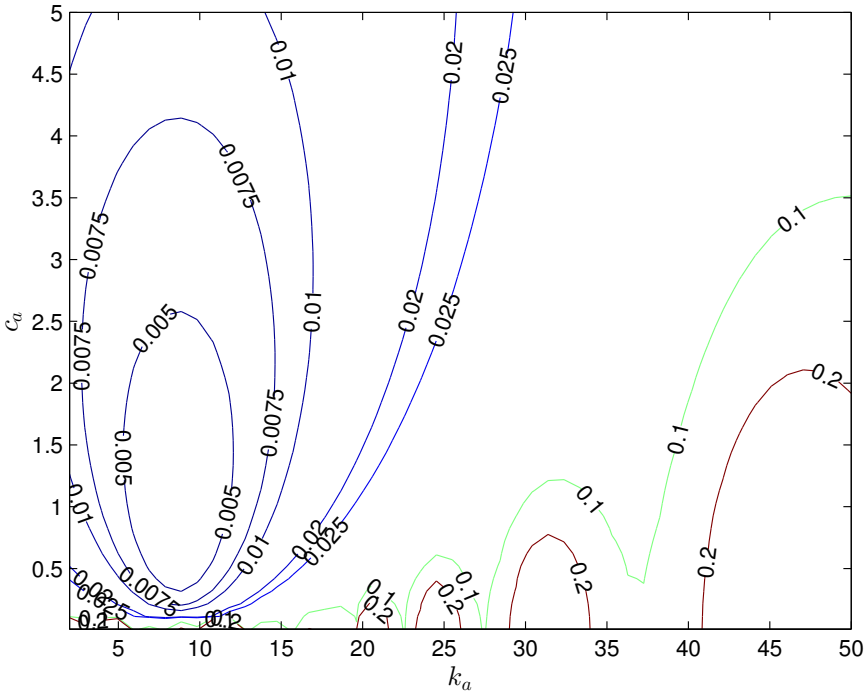

Figure 6. Contours of the objective function with $0 \leq t \leq 20$ sec.

The optimum solution as determined by fmincon is

$$\boxed{k_a = 8.6854 \quad \text{and} \quad c_a = 0.89661} \tag{18}$$

which agrees well with the result from the time-domain solution.

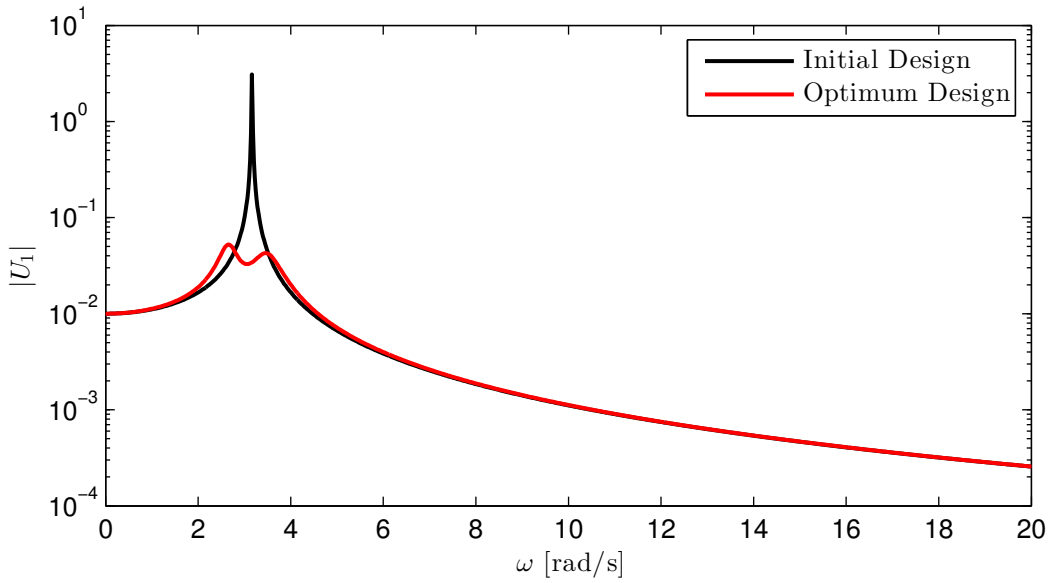

Figure 7. Comparison between initial and final designs.

Appendices

Code listing

Problem 1 Code

```
%=========================================================================
%% Clearing workspace
%=========================================================================

clc,clear,close all

%=========================================================================
%% Inputs
%=========================================================================

m1n = 2;                    % kg
m2n = 1;                    % kg
m3n = 4;                    % kg
k1n = 1000;                 % N/m
k2n = 400;                  % N/m
k3n = 500;                  % N/m

% Mass and stiffness matrices
syms k1 k3 lam
M = [m1n 0 0; 0 m2n 0; 0 0 m3n];
K = [k1+k2n -k2n 0;
    -k2n k2n+k3 -k3;
    0 -k3 k3];
```

```

% Numerical
symVec = [k1 k3];
numVec = [k1n k3n];
Kn = subs(K,symVec,numVec);
Mn = subs(M,symVec,numVec);
[eigVec,eigVal] = eig(Kn,Mn);

%=========================================================================
%% Symbolic Calculations
%=========================================================================

% Finding eigenvalues
Lambda = det(K - lam*M);
Lambda = solve(Lambda==0,lam);

Lambda = sort(Lambda);
subs(Lambda,symVec,numVec)

% Determining eigenvalues
syms ka kb
% subs(Lambda,symVec,numVec)
solve(Lambda(3)==16,k3)

Lam1 = subs(Lambda(3),[k1 k3],[ka k3n]);
Lam2 = subs(Lambda(2),[k1 k3],[k1n kb]);
Lam3 = subs(Lambda(1),[k1 k3],[k1n kb]);

% Finding eigenvectors
% V1 = null(K - Lam1*M);

% % Computing derivatives
dLam1qdk1 = real(subs(diff(Lam1,ka),ka,k1n))
dLam2qdk3 = real(subs(diff(Lam2,kb),kb,k3n))
```

```

%=========================================================================
% Clearing workspace
%=========================================================================

clc,clear,close all

%=========================================================================
% Inputs
%=========================================================================

m1 = 2;                     % kg
m2 = 1;                     % kg
m3 = 4;                     % kg
k1 = 1000;                  % N/m
k2 = 400;                   % N/m
k3 = 500;                   % N/m

% Mass and stiffness matrices
M = [m1 0 0; 0 m2 0; 0 0 m3];
K = [k1+k2 -k2 0; -k2 k2+k3 -k3; 0 -k3 k3];

% Path for images
```

```
ImgPath = '../Images/';

%=========================================================================
% Exact values from symbolic calculations
%=========================================================================

dLam1qdk1 = 0.009279009856776;
dLam2qdk3 = 0.368941533977921;

%=========================================================================
% Calculations
%=========================================================================

[Phi,Lambda] = eig(K,M);
Lambda = sort(diag(Lambda));

disp('=====================================================================')
disp('Eigenvectors and eigenvalues')
disp('=====================================================================')
disp(' ')
disp('Eigenvalues = ')
disp(Lambda)
disp('Eigenvectors = ')
disp(Phi)

%=========================================================================
% Using finite difference to compute derivatives
%=========================================================================

% Setting up structure for solution
soln = struct('step',[],'fd_dL1dk1',[],'fd_dL2dk3',[],'cd_dL1dk1',[], ...
    'cd_dL2dk3',[],'cvm_dL1dk1',[],'cvm_dL2dk3',[]);

% Step size
soln(1).step = 0.1;
soln(2).step = 0.01;
soln(3).step = 1e-4;
soln(4).step = 1e-8;
soln(5).step = 1e-16;
soln(6).step = 1e-100;

% Calling functions to compute forward difference solutions
for N = 1:numel(soln)

    %-------------------------------------------------------------------------
    % Forward difference solutions
    %-------------------------------------------------------------------------

    % Computing the eigenvalues
    K_d = K + [soln(N).step 0 0; 0 0 0; 0 0 0];
    [~,Lambda1_d] = eig(K_d,M);
    K_d = K + [0 0 0;
        0 soln(N).step -soln(N).step;
        0 -soln(N).step soln(N).step];
    [~,Lambda3_d] = eig(K_d,M);
    Lambda3_d = sort(diag(Lambda3_d));

    % Approximating the derivatives
    soln(N).fd_dL1dk1 = (Lambda1_d(1) - Lambda(1))/soln(N).step;
```

```
    soln(N).fd_dL2dk3 = (Lambda3_d(2) - Lambda(2))/soln(N).step;

    %-------------------------------------------------------------------------
    % Central difference solution
    %-------------------------------------------------------------------------

    % Computing the eigenvalues
    K_d = [k1 + soln(N).step + k2 -k2 0;
        -k2 k2+k3 -k3;
        0 -k3 k3];
    [~,Lambda_d] = eig(K_d,M);
    lambda1_ip1 = Lambda_d(1,1);
    K_d = [k1 - soln(N).step + k2 -k2 0;
        -k2 k2+k3 -k3;
        0 -k3 k3];
    [~,Lambda_d] = eig(K_d,M);
    lambda1_im1 = Lambda_d(1,1);
    K_d = [k1+k2 -k2 0;
        -k2 k2+k3+soln(N).step -k3-soln(N).step;
        0 -k3-soln(N).step k3+soln(N).step];
    [~,Lambda_d] = eig(K_d,M);
    lambda2_ip1 = Lambda_d(2,2);
    K_d = [k1+k2 -k2 0;
        -k2 k2+k3-soln(N).step -k3+soln(N).step;
        0 -k3+soln(N).step k3-soln(N).step];
    [~,Lambda_d] = eig(K_d,M);
    lambda2_im1 = Lambda_d(2,2);

    % Approximating the derivatives
    soln(N).cd_dL1dk1 = (lambda1_ip1 - lambda1_im1)/(2*soln(N).step);
    soln(N).cd_dL2dk3 = (lambda2_ip1 - lambda2_im1)/(2*soln(N).step);

    %-------------------------------------------------------------------------
    % Complex value method
    %-------------------------------------------------------------------------

    % Computing the eigenvalues
    K_d = K + [1i*soln(N).step 0 0; 0 0 0; 0 0 0];
    [~,Lambda1_d] = eig(K_d,M);
    Lambda1_d = sort(diag(Lambda1_d));
    K_d = K + [0 0 0; 0 1i*soln(N).step -1i*soln(N).step;
        0 -1i*soln(N).step 1i*soln(N).step];
    [~,Lambda3_d] = eig(K_d,M);
    Lambda3_d = sort(diag(Lambda3_d));

    % Approximating the derivatives
    soln(N).cvm_dL1dk1 = imag(Lambda1_d(1))/soln(N).step;
    soln(N).cvm_dL2dk3 = imag(Lambda3_d(2))/soln(N).step;

end

%==========================================================================
% Printing results
%==========================================================================

disp('======================================================================')
disp('Symbolic Solution')
disp('======================================================================')
disp(' ')
```

```
disp('dLam1/dk1 = 0.009279009856776')
disp('dLam2/dk3 = 0.368941533977921')
disp(' ')
disp('==========================================================================')
disp('Forward Difference Solution')
disp('==========================================================================')
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL1/dk1 = %1.10f\n',soln(N).step, ...
        soln(N).fd_dL1dk1)
end
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL2/dk3 = %1.10f\n',soln(N).step, ...
        soln(N).fd_dL2dk3)
end
disp(' ')
disp('==========================================================================')
disp('Central Difference Solution')
disp('==========================================================================')
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL1/dk1 = %1.10f\n',soln(N).step, ...
        soln(N).cd_dL1dk1)
end
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL2/dk3 = %1.10f\n',soln(N).step, ...
        soln(N).cd_dL2dk3)
end
disp(' ')
disp('==========================================================================')
disp('Complex Value Method')
disp('==========================================================================')
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL1/dk1 = %1.10f\n',soln(N).step, ...
        soln(N).cvm_dL1dk1)
end
disp(' ')
for N = 1:numel(soln)
    fprintf('step = %1.1e\t\tdL2/dk3 = %1.10f\n',soln(N).step, ...
        soln(N).cvm_dL2dk3)
end
disp(' ')
```

```
%==========================================================================
% Clearing workspace
%==========================================================================

clc,clear,close all

%==========================================================================
% Inputs
%==========================================================================

m1 = 2;                     % kg
```

```
m2 = 1;                     % kg
m3 = 4;                     % kg
k1 = 1000;                  % N/m
k2 = 400;                   % N/m
k3lin = 500;                % N/m

% Setting up k3 vector
k3 = linspace(400,600,1e3);

% Path for images
ImgPath = '../Images/';

%=========================================================================
% Exact values from symbolic calculations
%=========================================================================

dLam2qdk3 = 0.368941533977921;

%=========================================================================
% Exact solution
%=========================================================================

% Preallocating
lambda2 = zeros(size(k3));

for n = 1:numel(k3)

    % Constructing new mass and stiffness matrices
    M = [m1 0 0; 0 m2 0; 0 0 m3];
    K = [k1+k2 -k2 0; -k2 k2+k3(n) -k3(n); 0 -k3(n) k3(n)];

    % Computing the eigenvalues
    [~,Lambda] = eig(K,M);

    % Sorting eigenvalues
    Lambda = sort(diag(Lambda));
    lambda2(n) = Lambda(2);

end

%=========================================================================
% First order approximation
%=========================================================================

% Calculating lambda_2(500)
M = [m1 0 0; 0 m2 0; 0 0 m3];
K = [k1+k2 -k2 0; -k2 k2+k3lin -k3lin; 0 -k3lin k3lin];

% Computing the eigenvalues
[~,Lambda] = eig(K,M);

% Sorting eigenvalues
Lambda = sort(diag(Lambda));
lambda2_500 = Lambda(2);

% Calculating first order approximation
lambda2lin = lambda2_500 + dLam2qdk3*(k3 - k3lin);

%=========================================================================
```

```
% Plotting
%==========================================================================

figure('Position',[200 40 550 300])
plot(k3,lambda2,'-k',k3,lambda2lin,'--b','LineWidth',1.5)

% Labels and legend
xlabel('$k_3$','Interpreter','LaTeX')
ylabel('$\lambda_2(k_3)$','Interpreter','LaTeX')
[lh1,lh2] = legend('Exact solution','First-order approximation');
txtobj = findall(lh2,'type','text');
set(txtobj,'Interpreter','LaTeX')
set(lh1,'Location','NorthWest')
lPos = get(lh1,'Position');
set(lh1,'Position',[lPos(1) lPos(2) lPos(3)*1.05 lPos(4)])

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw6Prob1c.eps'])
```

Problem 2 Code

Lagrange Multiplier Method

Defining matrices

In[1]:=
```
k2 = 400;
```

In[17]:=
```
K = {{k1 + k2, -k2, 0}, {-k2, k2 + k3, -k3}, {0, -k3, k3}};
```

In[9]:=
```
m1 = 2;
m2 = 1;
m3 = 4;
M = {{m1, 0, 0}, {0, m2, 0}, {0, 0, m3}};
```

In[19]:=
```
Φ1 = {{1}, {ϕ2}, {ϕ3}};
```

Solving the equations

In[26]:=
```
Solve[Transpose[Φ1].D[K, k1].Φ1 == Transpose[Φ1].D[K, k3].Φ1, ϕ2]
```

Out[26]=
$$\{\{\phi_2 \to -1 + \phi_3\}, \{\phi_2 \to 1 + \phi_3\}\}$$

In[31]:=
```
ϕ2 = ϕ3 - 1;
```

In[33]:=
```
Solve[(K - 16.0 * M).Φ1 == 0 && Transpose[Φ1].D[K, k1].Φ1 == Transpose[Φ1].D[K, k3].Φ1]
```

Out[33]=
$$\{\{k1 \to 212.,\ \phi_3 \to 2.45,\ k3 \to 156.8\}\}$$

```m
function Hw6Prob2

%=========================================================================
% Clearing workspace
%=========================================================================

clc,clear,close all

%=========================================================================
% Inputs
%=========================================================================

% Path for images
ImgPath = '../Images/';

%=========================================================================
% Plotting contours
%=========================================================================

% Defining levels
nLevels = 40;

% Starting and stopping points
k1start = 10;
k1stop = 1000;
k3start = 10;
k3stop = 1000;

% Assembling vectors
K1 = k1start:((k1stop-k1start)/(nLevels-1)):k1stop;
K3 = k3start:((k3stop-k3start)/(nLevels-1)):k3stop;
[K1,K3] = meshgrid(K1,K3);
F = zeros(size(K1));

% Determining the value of the objective function at each point
[nR,nC] = size(K1);
for I = 1:nR
    for J = 1:nC
        F(I,J) = objfun([K1(I,J) K3(I,J)]);
    end
end

figure('Position',[90 20 500 350])
contour(K1,K3,F,nLevels);
colorbar

% Plotting nonlinear constraint
hold on
k1con = linspace(135,k1stop,1e3);
k3con = (384*k1con - 18688)./(5*(k1con - 132));
ph1 = plot(k1con,k3con,'--r','LineWidth',1.5);

%=========================================================================
% Calling optimization procedure
%=========================================================================

% Lower and upper bounds
lb = [145 100];
```

```
ub = [400 400];

% Starting point
X0 = lb;

% Setting options
opts = optimset('MaxIter',1e3,'MaxFunEvals',1e4,'Algorithm','Active-set');

Xdes = fmincon(@objfun,X0,[],[],[],[],lb,ub,@nonlincon,opts);

%=========================================================================
% Using design variables to check that constraint is met (lambda_1 = 16)
%=========================================================================

k_1 = Xdes(1);
k_2 = 400;
k_3 = Xdes(2);
m_1 = 2;
m_2 = 1;
m_3 = 4;

% Constructing mass and stiffness matrices
M = [m_1 0 0; 0 m_2 0; 0 0 m_3];
K = [k_1 + k_2 -k_2 0;
    -k_2 k_2 + k_3 -k_3;
    0 -k_3 k_3];

% Calculating the eigenvalues with the final design k_1 and k_3
[~,eigVals] = eig(K,M);
eigVals = sort(diag(eigVals));
fprintf(['\nWith k_1 = %1.2f and k_3 = %1.2f, the first eigenvalue', ...
    ' is %1.2f.\n\n'],k_1,k_3,eigVals(1))

%=========================================================================
% Using the method of Lagrange multipliers to ensure that the Kuhn-Tucker
% conditions are satisfied
%=========================================================================

% Finding eigenvectors using optimal design variables determined by fmincon
[Phi,Lambda] = eig(K,M);
Lambda = diag(Lambda);

% Separating the eigenvectors
phi_1 = Phi(:,1);

% Approximating the sensitivities of the eigenvalues wrt the design
% variables
Kprime = [1 0 0; 0 0 0; 0 0 0];
dlam1qdk1 = phi_1'*Kprime*phi_1/(phi_1'*M*phi_1);
Kprime = [0 0 0; 0 1 -1; 0 -1 1];
dlam1qdk3 = phi_1'*Kprime*phi_1/(phi_1'*M*phi_1);

% Outputting results
disp(['dlam1/dk1 = ',num2str(dlam1qdk1)])
disp(['dlam1/dk3 = ',num2str(dlam1qdk3)])
disp(['lambda_1 = ',num2str(Lambda(1))])
disp(' ')

% Tolerance for checking equality
```

```
tol = 1e-6;

% Logic to make sure that the Kuhn-Tucker conditions are satisfied within
% the given tolerance
epsCon1 = abs(dlam1qdk1 - dlam1qdk3);
epsCon2 = abs(Lambda(1) - 16);

if epsCon1 < tol && epsCon2 < tol
    disp('Kuhn-Tucker conditions are satisfied by numerical solution.')
else
    disp('Kuhn-Tucker conditions are not satisfied by the numerical solution.')
end


%=========================================================================
% Plotting solution
%=========================================================================

ph2 = plot(Xdes(1),Xdes(2),'sk','LineWidth',1.1,'MarkerFaceColor','g', ...
    'MarkerSize',7);

% Adding labels and legend
xlabel('$k_1$','Interpreter','LaTeX')
ylabel('$k_3$','Interpreter','LaTeX')
[lh1,lh2] = legend([ph1,ph2],'Nonlinear constraint','Solution');
txtobj = findall(lh2,'type','text');
set(txtobj,'Interpreter','LaTeX')
lPos = get(lh1,'Position');
set(lh1,'Position',[0.97*lPos(1) lPos(2) 1.02*lPos(3) lPos(4)])

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw6Prob2_numerical.eps'])

%=========================================================================
% Objective function
%=========================================================================

    function f = objfun(X)

        f = X(1) + X(2);

    end

%=========================================================================
% Nonlinear constraint function
%=========================================================================

    function [c,ceq] =  nonlincon(X)

        % Separating design variables
        k1 = X(1);
        k3 = X(2);

        % Inequality constraints
        c = [];

        % Equality constraints
        ceq = (384*k1 - 18688)./(5*(k1 - 132)) - k3;
```

```
177
178     end
179
180 end
```

Problem 3 Code

```
1  function Hw6Prob3
2
3  %=========================================================================
4  % Clearing workspace
5  %=========================================================================
6
7  clc,clear,close all
8
9  %=========================================================================
10 % Inputs
11 %=========================================================================
12
13 M = [10 0;
14     0 1];
15 p = [1; 0];
16 u_0 = [0; 0];
17 v_0 = [10; 0];
18
19 % Path for images
20 ImgPath = '../Images/';
21
22 %=========================================================================
23 % Time domain problem
24 %=========================================================================
25
26 % Upper and lower bounds
27 lb = [1 0.1];
28 ub = [100 10];
29
30 % Starting point
31 X0 = lb;
32
33 % Plotting initial design
34 [~,t_i,u_i] = time_domain(X0);
35 figure('Position',[60 25 540 300])
36 ph1 = plot(t_i,u_i(:,1),'-k','LineWidth',1.5);
37 hold on
38
39 % Calling fmincon
40 Xdes = fmincon(@time_domain,X0,[],[],[],[],lb,ub);
41 disp(Xdes)
42
43 % Plotting final design
44 [~,t_f,u_f] = time_domain(Xdes);
45 ph2 = plot(t_f,u_f(:,1),'--r','LineWidth',1.5);
46
47 % Adding legend and labels
48 xlabel('$t$ [sec]','Interpreter','LaTeX')
```

```
ylabel('$u_1(t)$ [m]','Interpreter','LaTeX')
[~,lh2] = legend([ph1,ph2],'Initial design','Final design');
txtobj = findall(lh2,'type','text');
set(txtobj,'Interpreter','LaTeX')

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw6Prob3_tdcomp.eps'])

%==========================================================================
% Plotting contours of time-domain objective function
%==========================================================================

% dec_str = 'Do you want to plot the contours of the objective function?';
% plot_decision = input([dec_str,' (''y''/''n''): '] );
plot_decision = 'n';

if strcmp(plot_decision,'y')

    % Plotting contours
    nLevels = 30;
    kStart = 4;
    kStop = 16;
    cStart = 0;
    cStop = 3;
    kVec = kStart:((kStop - kStart)/(nLevels - 1)):kStop;
    cVec = cStart:((cStop - cStart)/(nLevels - 1)):cStop;
    [K,C] = meshgrid(kVec,cVec);
    [nR,nC] = size(K);
    F = zeros(nR,nC);

    for I = 1:nR
        for J = 1:nC
            F(I,J) = time_domain([K(I,J) C(I,J)]);
        end
    end

    % Plotting
    figure('Position',[160 25 560 420])
    contour(K,C,F,nLevels)
    colorbar

    % Adding labels to plot
    xlabel('$k_a$','Interpreter','LaTeX','FontSize',12)
    ylabel('$c_a$','Interpreter','LaTeX','FontSize',12)

    % Saving figure
    set(gcf,'PaperPositionMode','Auto')
    print(gcf,'-depsc',[ImgPath,'Hw6Prob3_contours.eps'])

end


% Objective function
    function [L2t,t_n,u_n] = time_domain(X)

        k_a = X(1);
        c_a = X(2);

```

```
        % Solving the ode
        tVec = linspace(0,10,5e2);
        IC = [u_0(1) v_0(1) u_0(2) v_0(2)];
        [t_n,u_n] = ode45(@odefun,tVec,IC);

        % Determine the L^2 norm of the displacement using numerical
        % integration
        L2t = sqrt(trapz(t_n,(u_n(:,1)).^2));

        % Function used by ode45
        function dx = odefun(t,x)

            dx = zeros(4,1);

            dx(1) = x(2);
            dx(2) = 1/10*(-c_a*x(2) + c_a*x(4) - ...
                (k_a + 100)*x(1) + k_a*x(3));
            dx(3) = x(4);
            dx(4) = c_a*x(2) - c_a*x(4) + k_a*x(1) - k_a*x(3);

        end

    end

end
```

```
function Hw6Prob3b

%=========================================================================
% Clearing workspace
%=========================================================================

clc,clear,close all

%=========================================================================
% Inputs
%=========================================================================

M = [10 0;
    0 1];
p = [1; 0];

% Path for images
ImgPath = '../Images/';

%=========================================================================
% Frequency domain problem
%=========================================================================

lb = [0.1 0.1];
ub = [100 5];
X0 = lb;

Xdes = fmincon(@freq_domain,X0,[],[],[],[],lb,ub);
disp(Xdes)

% Plotting initial and final designs
```

```
figure('Position',[110 100 560 280])
[~,w1,u1] = freq_domain(X0);
plot(w1,abs(u1),'-k','LineWidth',1.5)
hold on
[~,w2,u2] = freq_domain(Xdes);
plot(w2,abs(u2),'-r','LineWidth',1.5)


% Adding labels and improving aesthetics
xlabel('$\omega$ [rad/s]','Interpreter','LaTeX')
ylabel('$|U_1|$','Interpreter','LaTeX')
set(gca,'YScale','log')
[lh1,lh2] = legend('Initial Design','Optimum Design');
txtobj = findall(lh2,'type','text');
set(txtobj,'Interpreter','LaTeX')
lPos = get(lh1,'Position');
set(lh1,'Position',[lPos(1)*0.95 lPos(2) lPos(3)*1.11 lPos(4)])

% Saving plot
set(gcf,'PaperPositionMode','Auto')
print(gcf,'-depsc',[ImgPath,'Hw6Prob3_fdcomp.eps'])

%==========================================================================
% Plotting contours of time-domain objective function
%==========================================================================

% dec_str = 'Do you want to plot the contours of the objective function?';
% plot_decision = input([dec_str,' (''y''/''n''): '] );
plot_decision = 'n';

if strcmp(plot_decision,'y')

    % Plotting contours
    nLevels = 50;
    kStart = 2;
    kStop = 50;
    cStart = 0.01;
    cStop = 5;
    kVec = kStart:((kStop - kStart)/(nLevels - 1)):kStop;
    cVec = cStart:((cStop - cStart)/(nLevels - 1)):cStop;
    [Kc,Cc] = meshgrid(kVec,cVec);
    [nR,nC] = size(Kc);
    F = zeros(nR,nC);

    for I = 1:nR
        for J = 1:nC
            F(I,J) = freq_domain([Kc(I,J) Cc(I,J)]);
        end
    end

    % Plotting
    figure('Position',[160 25 560 420])
    cLevels = [0.025 0.005 0.0075 0.01 0.02 0.1 0.2];
    [Ch,h] = contour(Kc,Cc,F,cLevels);
    clabel(Ch,h)

    % Adding labels to plot
    xlabel('$k_a$','Interpreter','LaTeX','FontSize',12)
    ylabel('$c_a$','Interpreter','LaTeX','FontSize',12)
```

```
91
92      % Saving figure
93      set(gcf,'PaperPositionMode','Auto')
94      print(gcf,'-depsc',[ImgPath,'Hw6Prob3b_contours.eps'])
95
96  end
97
98  %=========================================================================
99  % Objective function
100 %=========================================================================
101
102     function [L2f,omega,U] = freq_domain(X)
103
104         % Separating design variables
105         k_a = X(1);
106         c_a = X(2);
107
108         % Constructing K and C matrices
109         K = [k_a + 100 -k_a;
110             -k_a k_a];
111         C = [c_a -c_a;
112             -c_a c_a];
113
114         % Finding the response
115         omega = linspace(0,20,1e3);
116         U = zeros(size(omega));
117         for n = 1:numel(omega)
118
119             Z = K - omega(n)^2*M + 1i*omega(n)*C;
120             Ut = Z\p;
121             U(n) = abs(Ut(1));
122
123         end
124
125         % Calculating the objective function
126         L2f = trapz(omega,U.^2);
127
128     end
129
130 end
```