Table of Contents


```
% Ruipu Ji
% SE 265
% Homework #4

clc; clear; close all;

set(0, 'DefaultTextInterpreter', 'latex');
set(0, 'DefaultLegendInterpreter', 'latex');
set(0, 'DefaultAxesTickLabelInterpreter', 'latex');

set(0, 'DefaultAxesFontSize', 15);
set(0, 'DefaultTextFontSize', 15);
```

Task A. Form the mass matrix [M].

```
nDOF = 8; % Define the number of DOFs for the system.
m = [0.4194 0.4194 0.4194 0.4194 0.4194 0.4194 0.4194 0.4194]; % Mass of
each component (unit = kg).
M = zeros(nDOF,nDOF); % Initialization for the mass matrix.

% Assign the non-zero diagonal elements in the mass matrix.
for i = 1:nDOF
    M(i,i) = m(i);
end

% Display the mass matrix [M] in the command window.
disp('Mass matrix [M]:');
disp(M);
```

```
Mass matrix [M]:
  Columns 1 through 7

    0.4194         0         0         0         0         0         0
         0    0.4194         0         0         0         0         0
         0         0    0.4194         0         0         0         0
         0         0         0    0.4194         0         0         0
         0         0         0         0    0.4194         0         0
         0         0         0         0         0    0.4194         0
```

```
         0         0         0         0         0         0    0.4194
         0         0         0         0         0         0         0

  Column 8

         0
         0
         0
         0
         0
         0
         0
    0.4194
```

Task B. Form the stiffness matrix [Ku] for the undamaged structure.

```
ku = [56700 56700 56700 56700 56700 56700 56700 56700]; % Stiffness of each
spring in the undamaged structure (unit = N/m).
Ku = zeros(nDOF,nDOF); % Initialization for the stiffness matrix.

% Assign the non-zero elements in the stiffness matrix.
for i = 1:nDOF-1
    Ku(i,i) = ku(i) + ku(i+1);
    Ku(i,i+1) = -ku(i+1);
    Ku(i+1,i) = -ku(i+1);
end

Ku(nDOF, nDOF) = ku(nDOF);

% Display the stiffness matrix [K] in the command window.
disp('Stiffness matrix [Ku] for the undamaged structure:');
disp(Ku);

Stiffness matrix [Ku] for the undamaged structure:
  Columns 1 through 6

      113400      -56700           0           0           0           0
      -56700      113400      -56700           0           0           0
           0      -56700      113400      -56700           0           0
           0           0      -56700      113400      -56700           0
           0           0           0      -56700      113400      -56700
           0           0           0           0      -56700      113400
           0           0           0           0           0      -56700
           0           0           0           0           0           0

  Columns 7 through 8

           0           0
           0           0
           0           0
```

```
     0          0
     0          0
-56700          0
113400     -56700
-56700      56700
```

Task C. Form the stiffness matrix [Kd] for the damaged structure.

Consider the damage case that there is 10% reduction of stiffness in spring 6.

```
kd = ku;
kd(6) = 0.9 * kd(6);

% Initialization for the stiffness matrix.
Kd = zeros(nDOF,nDOF);

% Assign the non-zero elements in the stiffness matrix.
for i = 1:nDOF-1
    Kd(i,i) = kd(i) + kd(i+1);
    Kd(i,i+1) = -kd(i+1);
    Kd(i+1,i) = -kd(i+1);
end

Kd(nDOF, nDOF) = kd(nDOF);

% Display the stiffness matrix [K] in the command window.
disp('Stiffness matrix [Kd] for the damaged structure:');
disp(Kd);
```

```
Stiffness matrix [Kd] for the damaged structure:
  Columns 1 through 6

     113400     -56700          0          0          0          0
     -56700     113400     -56700          0          0          0
          0     -56700     113400     -56700          0          0
          0          0     -56700     113400     -56700          0
          0          0          0     -56700     107730     -51030
          0          0          0          0     -51030     107730
          0          0          0          0          0     -56700
          0          0          0          0          0          0

  Columns 7 through 8

          0          0
          0          0
          0          0
          0          0
          0          0
     -56700          0
     113400     -56700
```

```
    -56700        56700
```

Task D. Solve the system eigenvalues (natural frequencies) and eigenvectors (mode shapes) for the damaged structure.

Solve the square of eigenvalues [LambdaMatrix] and eigenvectors [Phi], only select select the real part.

```
[Phi, LambdaMatrix] = eig(-Kd, M);
Phi = real(Phi);
LambdaMatrix = real(LambdaMatrix);

% Save the square of eigenvalues in a new vector {LambdaVector}.
LambdaVector = zeros(nDOF, 1);
for i = 1:nDOF
    LambdaVector(i) = LambdaMatrix(i,i);
end

% Display the square of eigenvalues {LambdaVector} and eigenvectors [Phi].
disp('Square of eigenvalues:');
disp(LambdaVector);
disp('Eigenvectors (mode shapes):');
disp(Phi);
```

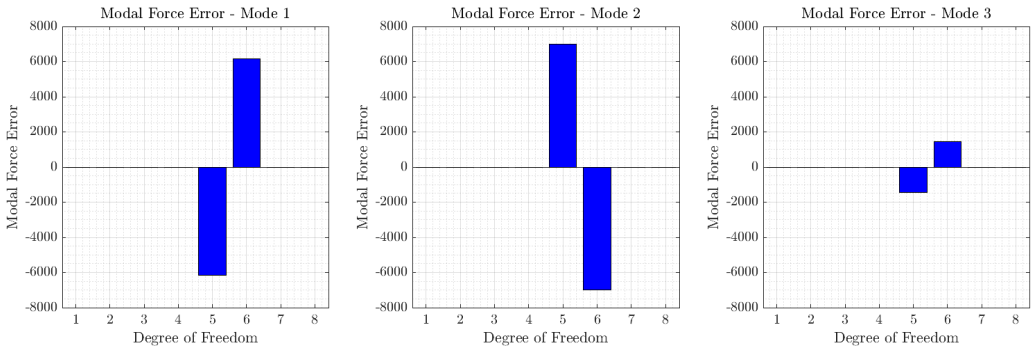
```
Square of eigenvalues:
   1.0e+05 *

   -5.1431
   -4.6229
   -3.9057
   -2.8853
   -1.9403
   -1.0708
   -0.3947
   -0.0457

Eigenvectors (mode shapes):
  Columns 1 through 7

   -0.3397   -0.4625    0.6852    0.7556    0.6934    0.6040    0.3878
    0.6129    0.6565   -0.6091   -0.1014    0.3916    0.7296    0.6625
   -0.7661   -0.4694   -0.1437   -0.7420   -0.4722    0.2773    0.7436
    0.7694    0.0098    0.7368    0.2010   -0.6583   -0.3946    0.6077
   -0.6221    0.4555   -0.5113    0.7151    0.1004   -0.7540    0.2944
    0.4614   -0.7799   -0.2568   -0.4094    0.7833   -0.4897   -0.1493
   -0.3187    0.7751    0.7142   -0.5477    0.2737    0.1360   -0.5050
    0.1137   -0.3204   -0.3781    0.4829   -0.6288    0.6540   -0.7133

  Column 8
```

```
   -0.1364
   -0.2683
   -0.3910
   -0.5006
   -0.5932
   -0.6738
   -0.7236
   -0.7489
```

Task E. Calculate modal force error [E] for the damaged structure.

Initialization of the matrix [E] for the modal force error. The column number represents the mode shape number and the row number represents the number of DOF.

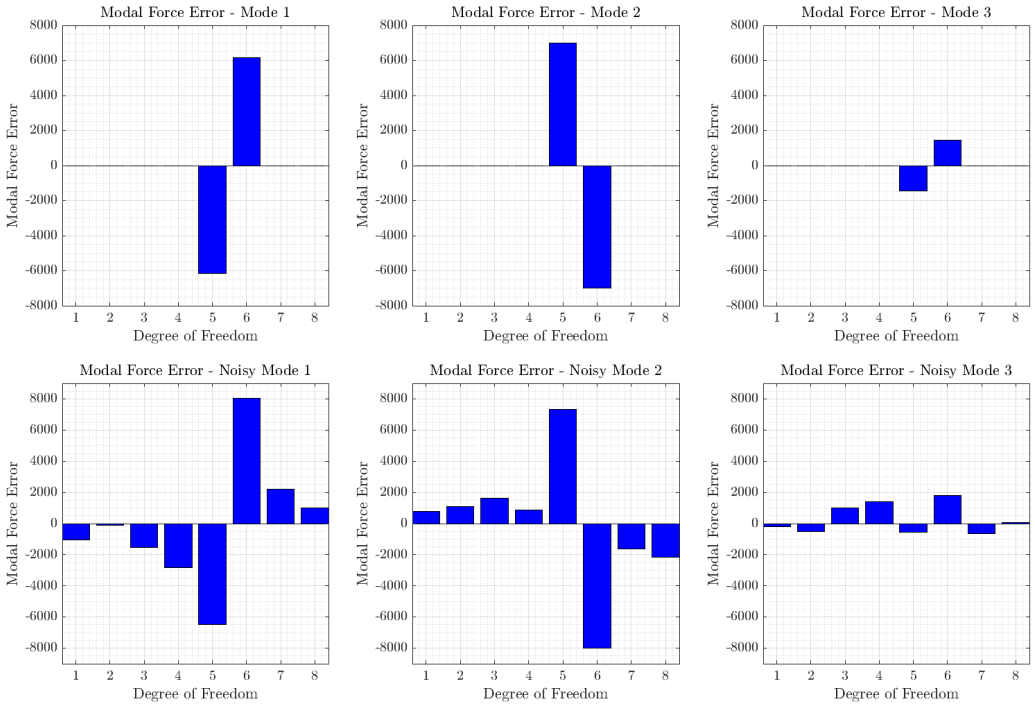
```
E = zeros(nDOF, nDOF);

% Loop over all the mode shapes, calculate the corresponding modal force
error vector and store it in the i-th column.
for i = 1:nDOF
    E(:,i) = (LambdaVector(i)*M + Ku) * Phi(:,i);
end

% Create a bar plot of the modal force error vectors for the first 3 modes.
figure('Renderer', 'painters', 'Position', [10 10 1800 500]);

for i = 1:3
    subplot(1,3,i);
    bar(E(:,i), 0.8, 'blue');
    grid on;
    grid minor;
    box on;
    xlim([0.6 8.4]);
    ylim([-8000 8000]);
    xticks(1:1:8);
    yticks(-8000:2000:8000);
    xlabel('Degree of Freedom');
    ylabel('Modal Force Error');
    title(sprintf(['Modal Force Error - Mode ', num2str(i)]));
end
```

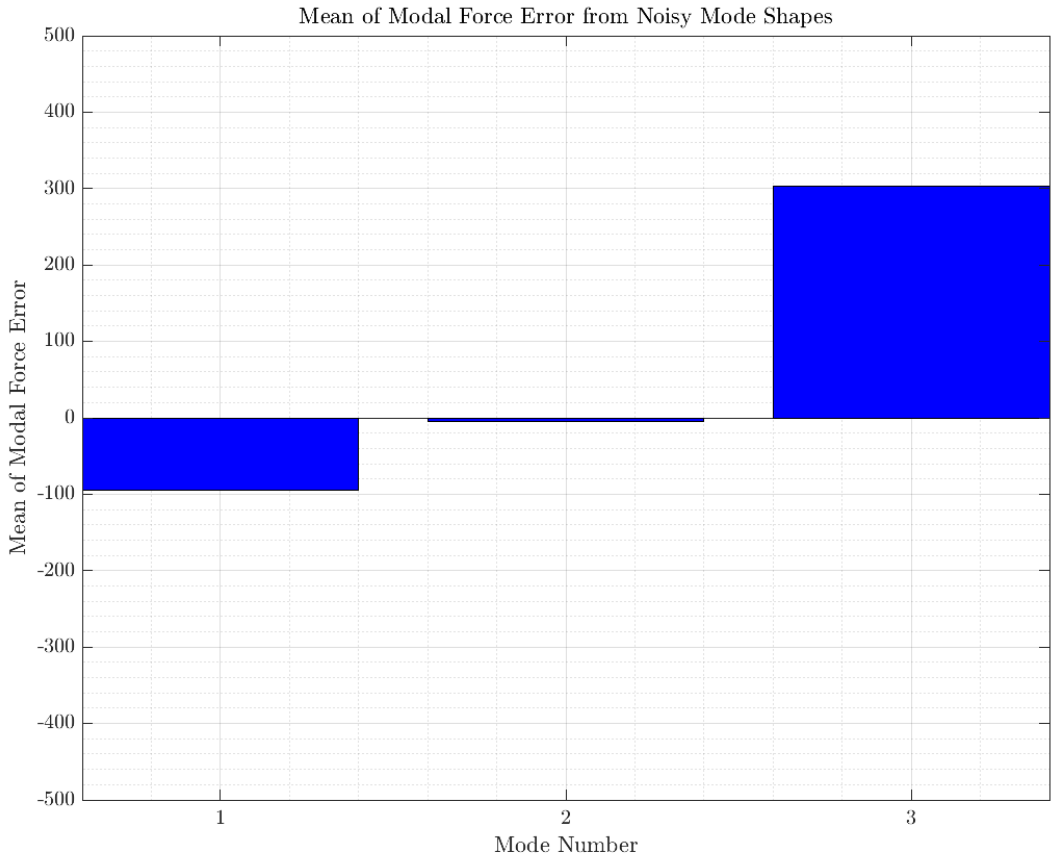


Task F. Calculate modal force error [E_noisy] for the damage structure considering the noise in the data.

```
load('noise_matrix.mat'); % Load the noise data file.
Phi_noisy = Phi + noise_matrix; % Add noise to the eigenvectors (mode
shapes).

% Initialization of the matrix [E_noisy] for the modal force error
considering the noise in the data.
% The column number represents the mode shape number and the row number
represents the number of DOF.
E_noisy = zeros(nDOF, nDOF);

% Loop over all the mode shapes, calculate the corresponding modal force
error vector and store it in the i-th column.
for i = 1:nDOF
    E_noisy(:,i) = (LambdaVector(i)*M + Ku) * Phi_noisy(:,i);
end

% Create a bar plot of the modal force error vectors for the first 3 modes
considering the noise in the data.
figure('Renderer', 'painters', 'Position', [10 10 1800 500]);

for i = 1:3
    subplot(1,3,i);
    bar(E_noisy(:,i), 0.8, 'blue');
    grid on;
    grid minor;
    box on;
    xlim([0.6 8.4]);
    ylim([-9000 9000]);
    xticks(1:1:8);
    yticks(-8000:2000:8000);
    xlabel('Degree of Freedom');
    ylabel('Modal Force Error');
    title(sprintf(['Modal Force Error - Noisy Mode ', num2str(i)]));
end
```

```
% Create a bar plot for the mean of the error vectors obtained with the
noisy data.
E_noisy_mean = mean(E_noisy);

figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

bar(E_noisy_mean(1:3), 0.8, 'blue');
grid on;
grid minor;
box on;
xlim([0.6 3.4]);
ylim([-500 500]);
xticks(1:1:3);
yticks(-500:100:500);
xlabel('Mode Number');
ylabel('Mean of Modal Force Error');
title(sprintf('Mean of Modal Force Error from Noisy Mode Shapes'));
```





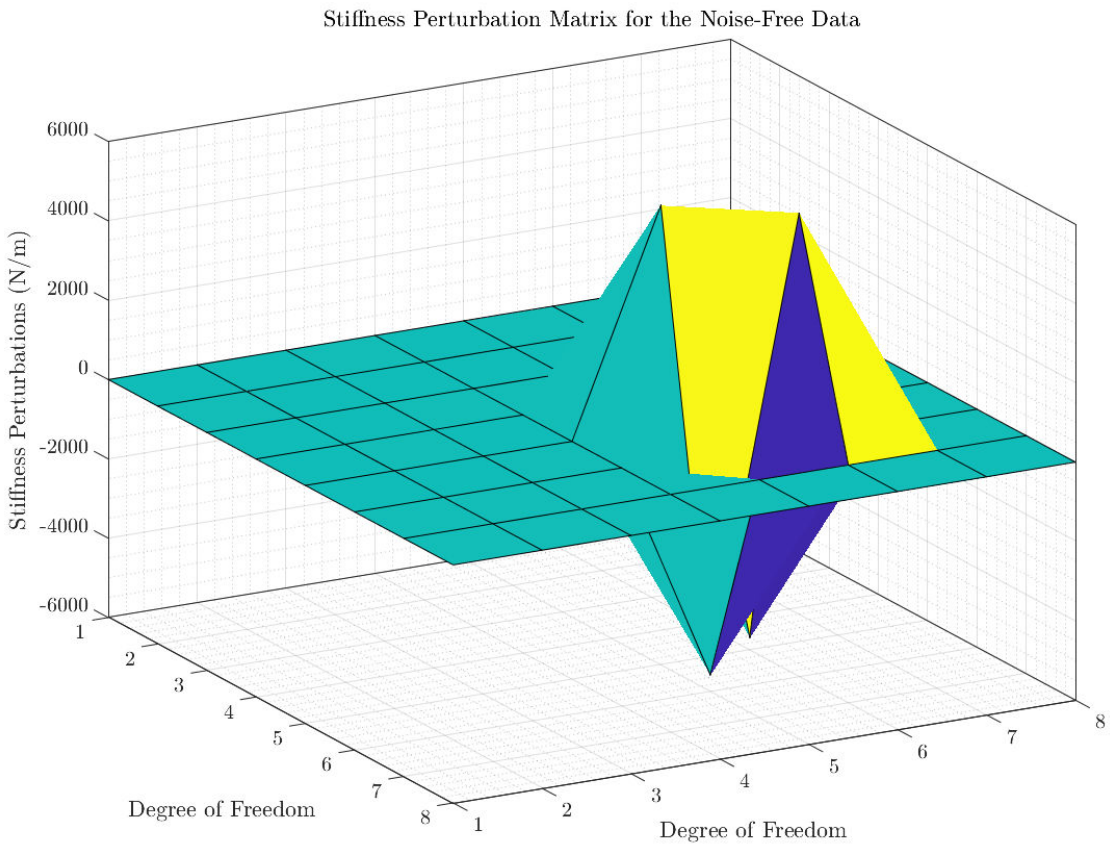
Task G. Minimum rank update.

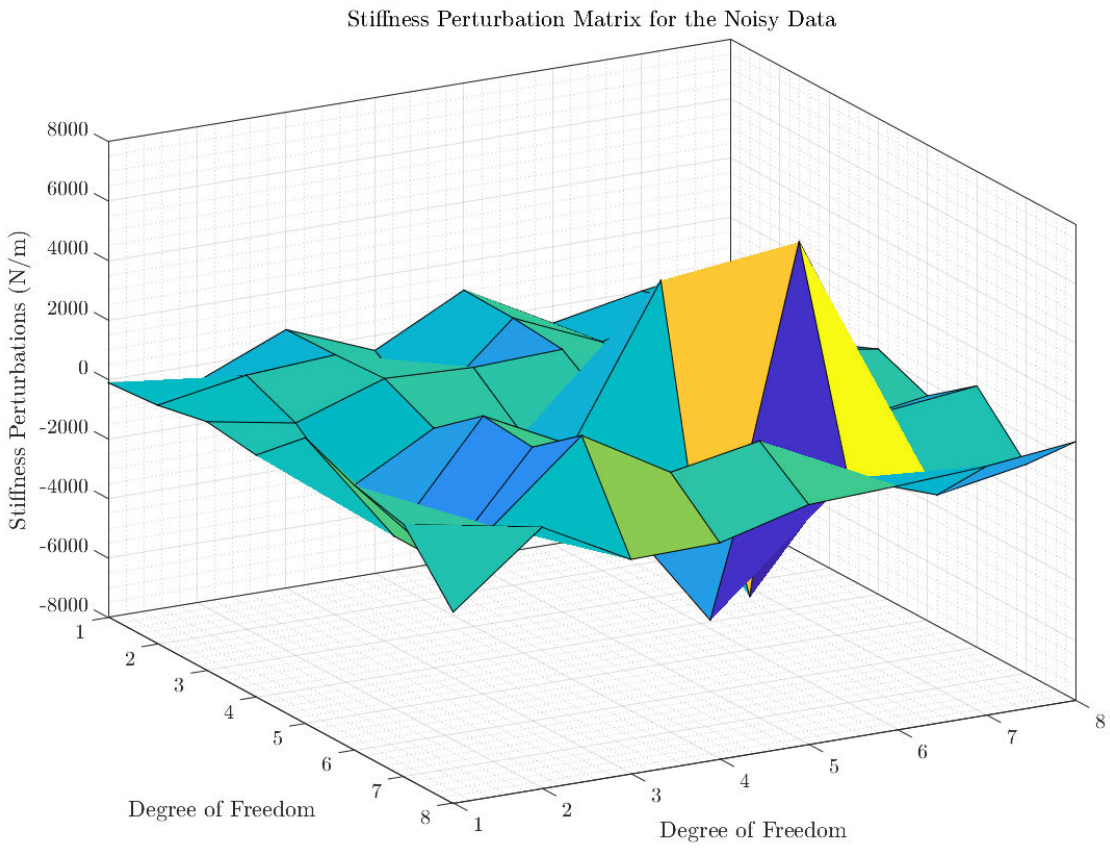
Calculate the perturbation error [DeltaK] for the noise-free data.

```
d = M*Phi*LambdaMatrix + Ku*Phi;
B = inv(d'*Phi);
DeltaK = d*B*d';

% Create the 3-D surface plot of the perturbation error [DeltaK] for the
noise-free data.
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

surf(DeltaK);
view(61,24);
grid on;
grid minor;
box on;
xlim([1 8]);
ylim([1 8]);
zlim([-6000 6000]);
xticks(1:1:8);
yticks(1:1:8);
zticks(-6000:2000:6000);
```

```
xlabel('Degree of Freedom');
ylabel('Degree of Freedom');
zlabel('Stiffness Perturbations (N/m)');
title(sprintf('Stiffness Perturbation Matrix for the Noise-Free Data'));

% Calculate the perturbation error [DeltaK_noisy] for the noisy data.
d_noisy = M*Phi_noisy*LambdaMatrix + Ku*Phi_noisy;
B_noisy = inv(d_noisy'*Phi_noisy);
DeltaK_noisy = d_noisy*B_noisy*d_noisy';

% Create the 3-D surface plot of the perturbation error [DeltaK] for the
noise-free data.
figure('Renderer', 'painters', 'Position', [10 10 1200 900]);

surf(DeltaK_noisy);
view(61,24);
grid on;
grid minor;
box on;
xlim([1 8]);
ylim([1 8]);
zlim([-8000 8000]);
xticks(1:1:8);
yticks(1:1:8);
zticks(-8000:2000:8000);
xlabel('Degree of Freedom');
ylabel('Degree of Freedom');
zlabel('Stiffness Perturbations (N/m)');
title(sprintf('Stiffness Perturbation Matrix for the Noisy Data'));
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND = 2.105132e-16.


Stiffness Perturbation Matrix for the Noise-Free Data
Stiffness Perturbations (N/m)
6000
4000
2000
0
-2000
-4000
-6000
Degree of Freedom
Degree of Freedom
1
2
3
4
5
6
7
8



Published with MATLAB® R2023b