MSc

Computational Linear Algebra

COURSEWORK 2

FANG ZIJUNCID: 01811420

Contents

Question 1 ..... ***2***

Question 2 ..... ***3***

a. Rayleigh Quotient Iteration along with Householder deflation- write *subroutines RQI and HHD* for this purpose; **3**

a-1 Theoretical function: 3

a-2 The Matlab codes: 4

b. QR iteration technique – write QRITER for this purpose; **7**

b-1. Theoretical function: 7

b-2. The Matlab code: 7

* QRITER Method: 7

c. Subspace Iteration- write *subroutines SSI and RITZ* for this purpose. **9**

c-1. Theoretical function: 9

c-2. The Matlab codes: 9

d. In the development of codes, advantage must be taken of the banded structures of matrices involved. Application of methods such as Skyline storage method is encouraged. **11**

d-1. The Matlab code: 11

e. Comment on the sensitivity of each method to numerical errors in matrices. **11**

e-1. Theoretical function： 11

Main scripts of Q1-3 and Q4: **13**

* Q1-3: 13

* Q4 16

Question 3 ..... ***18***

3-a-c: Comparing the Computation Time, Accuracy and Storage Capacity **18**

* the accuracy calculation function/method: 18

* the outcome table: 18

* the description between these methods: 18

Question 4 ..... ***19***

4-e. Subroutine SENS – to calculate the sensitivity of eigenvalues of the system. **19**

* the outcome table: S1: by changing the Young's Modulus (E_0 = 4) of the system; 19

* the description of sensitivity through the eigenvalues: 19

Question 1

Using a method of your choice, write a code using a development environment of your choice (Matlab is Recommended) to Calculate natural frequencies and the relevant eigenvectors of the system.

The function of calculating the **natural frequencies** and the **relevant eigenvectors**, for the undamped system without resultant force:

$$M\ddot{x} + Kx = 0$$

With setting the natural frequencies, using the **Euler function of mode displacements**:

$$(-\omega^2 M + K)x = 0$$
$$(M^{-1}K)x = Ax = \lambda x = (\omega^2)x$$
$$\omega_i = (\lambda)^{0.5}$$

So, the iteration basic matrix **A**, the natural frequencies by the square root of the eigenvalues and the **M, K** matrices resulted from FE model defined.

The Natural Frequencies and Relevant Eigenvectors of the system:

Natural Frequencies $\omega(rad/s)$	Relevant Eigenvectors X			
3.1101 2.3147 1.4355 0.5794	-0.23147	-0.29679	-0.39250	-0.22056
	0.49478	0.31480	-0.20530	-0.43921
	-0.60603	0.17030	0.41536	-0.58983
	0.57821	-0.88533	0.79452	-0.64073

Question 2

Write a code using the same development environment as used in 1 which can solve a general/standard N*N eigenvalue problem using the following techniques:

a. Rayleigh Quotient Iteration along with Householder deflation- write ***subroutines RQI and HHD*** for this purpose;

a-1 Theoretical function:
* the Rayleigh Quotient (RQ) method to calculate the eigenvalues:

$$\lambda = \frac{x^T A x}{x^T x}$$

* for the k times iteration of RQ method to calculate the eigenvectors:

$$\lambda_k = \frac{x_k^T A x_k}{x_k^T x_k}$$
$$(A - \lambda_k I) y_{k+1} = x_k$$

the next step calculation for the continuous eigenvectors:

$$x_{k+1} = \frac{y_{k+1}}{\|y_{k+1}\|_{inf}}$$

* the Householder Deflation (HHD), using the transformation function to calculation the continuous eigenvalues and corresponding eigenvectors:

$$HAH^{-1} = \begin{bmatrix} \lambda_1 & b^T \\ 0 & B \end{bmatrix}$$

$$B \epsilon \mathbb{R}^{(n-1)*(n-1)}$$

the HHD transformation matrix **H**:

$$H = [1 - 2\{v\}\{v\}^H]$$

the HHD eigenvalues and corresponding eigenvectors function:
by knowing the initial condition of eigenvector to calculate the continue values

$$Hx_1 = \alpha_1 e_1 = \|x_1\| e_1$$
$$\lambda_1 = u_1^T x_1$$
$$a_1 = \|e_1\|$$

$$c = 2(a_1)^2 - 2a_1 e_1$$
$$v_1 = \frac{1}{2}(1 - \frac{e_1}{c})$$

* the Shift Iteration (SHIFI)-the judgement method for continue calculation loops:

$$\frac{|\lambda_2 - \lambda_k|}{|\lambda_1 - \lambda_k|} > \frac{|\lambda_2|}{|\lambda_1|}$$

a-2 The Matlab codes:

* RQI Method

```
function [xk , lam] = RQI (A , n , itera)
%% Function explaination
% the function is calculating the largest eigenvalues MODULUS (lam) and
% eigenvectors (x) of the matrix A
% the first iteration with the Power Iteration (PwI) method Rayleigh
% Quotient Iteration (RQI), with the x0 as the initial vector


% Function Variables Inputs:
% A     %Input matrix
% n     %Matrix Dimension
% itera %Maximum Iteration Timing

% Function Variables Outputs:
% x    %Eigenvector
% lam  %Eigenvalue
%% Function settings
% Initial conditions
xk = ones (n , 1);                          %Trail vector
tr = 1e-7;                                 %Error Tolerance

% Power Iteration method first 2 rounds
for k = 1 : 2
    x1 = A * xk;                            %Initial x1 product
    lam = norm(x1);
    xk = x1 / lam;                          %Repeat calculation 2nd round
end
% PwI method for RQI
for k = 3 : itera

    lam = (xk' * A * xk) / (xk' * xk);          %Eigenvalues from 3rd to iter
    x1 = (A - lam * eye(n)) \ (x1);            %Continuous eigenvecotrs calculations
    er = norm( (A - lam * eye(n)) * xk );  %Define err and Convergence of PwI

    % If-loop: the error limitation
    if er < tr
        break;
    end
end
% Finish function running
end
```

* HHD Method:

```
function B = HHD(A , n , e)
%% Function explaination
% this function is returning the results of Householder deflation (HHD),
% with providing the first eigenvalues and corresponding eigenvecotrs by
% using the initial condition.

% Function Variables Inputs:
% A           %Input Matrix
% n           %Matrix Dimension
% e           %First eigenvector of A

% Function Variables Output
% B           %HHD function output matrix
%% Function settings
% Initial condition settings
a = norm(e);                  %the constant norm value-alipha of eigenvector matrix
v = zeros(n , 1);             %the initial unit vector perpendicular to hyperplane
% v-vector generation
c = 2 * (a^2) - (2 * a * e(1));                    %initial constant ratio of eigenvector 1
v(1) = sqrt(.5*(1-e(1) / c));          %first unit vector
% for contiue unit vector-v generate from i = 2:n
for i = 2:n
    v(i)= -sign(e(i)) * sqrt(e(i)^2 / c);     %v(i=2:n) continue generate
end

% B and H matrix generation through HHD
H = eye(n) - 2 * v * v';                  %H = [I - 2{v}({v}^H)]
B = H * A * H;                            %orthogonal method matrix
B = B(2:n , 2:n);                         %B matrix update without b11

% Function finish
end
```

* SHIFI Method:

```
function ve = SHIFI(A , n , p , itera , va)
%% Function explaination
% this function is running the program for finding all the eigenvectors of
% matrix A through the knwon eigenvalues

% Function Variable Inputs:
% A     %Input matrix
% n     %Matrix dimension
% p     %Eigenvector dimension
% itera %Maximum iteration timing
% valu  %Eigenvalue generating matrix

% Function Variable Output:
% vetr  %Eigenvector
%% Function settings
% Shift Inverse Iteration calculation loop

ve = ones (n , p);                          %initial eigenvector unit matrix

tr = 1e-7;                                    %loop judgement tolerance

for i = 1 : p
    %i-th time calculating range
    A1 = (A - (va(i , i) + 2e-4)*eye(n,n))^-1;          %first matrix of generating segma =
2e-4

    for k = 1 : itera
```

```
        yk = A1 * ve(: , i);                                %the output eigevector value through
shift
        yk = yk / norm(yk);                                     %next step y-value define
        er = norm(yk - ve(:,i));                            %eigevector error caluclation
eige(:,i)-perdict
        ve(: , i) = yk;

        if er < tr
            %error smaller than tolerance
            break
            %print out the if-loop stop
        end
    end
end

% Finish function running
end
```

* RQIHHD Method:

```
function [eiga , eige , memo] = RQIHHD (A , n , itera)
%% Function explaination
% the function is calculating the eigenvalues and eigenvectors, relating to
% the matrix A through the Rayleigh Quotient Iteration (RQI), including
% the power method and Householder Deflection (HHD).

% Function Variables Inputs:
% A     %Input Matrix
% n     %Matrix Dimension
% itera %Maximum Iteration Timing

% Function Variables Outputs:
% eige    %Eigenvector
% eiga    %Eigenvalue
% memo    %Memory generate

%% Function Settings
% Initial conditions
B = A;
eige = zeros (n , n);                   %initial nth degree eigvector
eiga = zeros (n , n);                   %initial nth degree eigevalue

% Flapping steps of calculating
for m = 0 : n-2
    [vctr , eiga(m+1 , m+1)] = RQI(B , n-m , itera);
    B = HHD(B , n-m , vctr);
end

eiga(n , n) = B(1 , 1);

% Shifted inverse iteration
eige = SHIFI(A , n , n , 5 , eiga);

% Memory calculation
memo = memoryfunc;

end
```

b. QR iteration technique – write **QRITER** for this purpose;

b-1. Theoretical function:

* QR Iteration Method:

$$A = QR$$
$$Q = \left[\frac{A_{1i}}{||A_{1i}||} \quad \frac{A_{2i}}{||A_{2i}||} \quad \frac{A_{2i}}{||A_{2i}||}\right]$$
$$R = Q^H QR = Q^H A$$

* Orthogonal Iteration Method to calculate the eigenvalues and corresponding eigenvectors:

$$x_k = Ax_{k-1}$$
$$Q_k R_k = x_{k-1}$$
$${Q_k}^H x_k = {Q_k}^H A Q_k R_k x_{k-1}$$

b-2. The Matlab code:

* QRITER Method:

```
function [eiga , eige , memo] = QRITER(A , n , itera)
%% Function explaination
% using the orthogonal method to calculate the eigenvalues and eigenvectors
% of matrix A, the continuous calculating through QR method by Ak = Qk * Rk

% QR fractorization and decomposition
% for analysis the QR calculation to the matrix A, with Orthogonal matrix
% (Q) and upper triangular matrix (R)

% Function Variables Inputs:
% A     %Input matrix
% n     %Matrix dimension
% itera %Maximum iteration timing


% Function Variables Outputs:
% eiga        %Eigevalue
% eige        %Eigevector
% memo        %Memory generate
% Function for QR Iteration
%%initial settings of A (Ak)
Ak = A;
eige = eye(n);                  %initial eigevector
tr = 1e-7;                      %looping continuous tolerance

%%QR iteration
for i = 1 : itera
    [Q , R] = QRSPI(Ak , n);

    Ak = R * Q;
    yk = eige * Q;
    er = norm(yk - eige);
    eige = yk;                %continuous y-vector
    if er < tr
        break;
    end
```

```
    % if-judge the tolerance limit
end

eiga = diag(Ak)' .* eye(n , n);                    %calculating eigevalues through known
eigevectors


%%Using SHIFI-function for finding eigevectors
eige = SHIFI (A , n , n , 5 , eiga);


%%Memory function collection
memo = memoryfunc;
end
```

* QRSPI Method:

```
function [Q , R] = QRSPI(Ak , n)
%% Function explaination
% Function Varibale Inputs:
% Ak       %Tested matrix of M^-1*K
% n        %Dimension of matix

% Function Variable Outputs:
% Q        %Othorgonal matrix
% R        %Upper triangular matrix


%% Function settings
% Function for Q and R generate
%%function with Q and U intial settings
Q = zeros(n , n);
U = zeros(n , n);
U = Ak;             %given value of matrix A

%%Q matrix generate (i = 1 and i = 2 : n)
Q(: , 1) = U(: , 1) / norm(U(: , 1));

for k = 2 : n
    for j = 1 : k-1

        U(: , k) = U(: , k) - Ak(: , k)' * Q(: , j) * Q(: , j);
        %U-matrix Q'A = Q'QR = R; U = A = A-A'QQ

    end

    Q(: , k) = U(: , k) / norm(U(: , k));
    %Qi = (Ui / ||Ui||)-norm number div

end


%%R matrix generate through R = Q'A
R = Q' * Ak;                      %Upper tri-matrix QR decomposition
```

c. Subspace Iteration- write ***subroutines SSI and RITZ*** for this purpose.

c-1. Theoretical function:
* the Subspace Iteration (SSI) method to obtain the eigenvalues and corresponding eigenvectors:

$$x_{k+1} = K^{-1}Mx_k$$
$$(\frac{1}{\lambda})x_k = K^{-1}Mx_k$$

*the RITZ subroutine function runs the modes of vibration:

$$X_k^T M X_k \ddot{p} + X_k^T K X_k p = 0$$
$$\bar{M}P\Lambda = \bar{K}P$$
$$q = [x_{k+1}]P$$

the sub matrices of mass and stiffness to calculating the eigenvalues and corresponding eigenvectors – **q** and **P**:

$$m_i = x_k{}'Mx_k$$
$$k_i = x_k{}'Kx_k$$

c-2. The Matlab codes:
* SSI Method:

```
function [eiga , eige , memo] = SSI(A , M , K , n , p , itera)
%% Function explaination
% using the Subspace Iteration (SSI) method to obtain all the matrices,
% including the eigenvectors and eigenvalues.

% Function Variable Inputs:
% A       %Input matrix
% M,K     %Matrices of mass and stiffness
% n       %Matrix dimension
% p       %Eigenvector dimension
% itera   %Maximum iteration timing

% Function Variable Outputs:
% eiga    %Eigevalue
% eige    %Eigevector
% memo    %Memory generate
%% Function settings
%initial condition settings
xk = eye(n , p);
tr = 1e-7;                  %tolerance of loop

for k = 1: itera
    xk = K^-1 * M * xk;                     %SSI mrthod: K*xk+1 = M*xk
    [xki , D] = RITZ (xk , M , K);         %RITZ-function output
    er = norm (xki - xk);                  %norm number of error
    xk = xki;

    if er < tr
        break
    end
end
```

```
eiga = D;
% mark the eigevalue as the diagonal matrix-defined previous function

%SHIFI function for all the eigevectors
eige = SHIFI(A , n , p , 5 , eiga);

%Memory function collection
memo = memoryfunc;
%Finish function running
end
```

* RITZ Method:

```
function [qn , Di] = RITZ(xk , M , K)
%% Function explaination
% RITZ Eigenvalues method: qn = xk * Pn

% Function Variable Inputs:
% xk      %Eigevector matrix
% M,K     %Matrices of mass and stiffness


% Function Variable Outputs:
% qn       %Calculating vector with nth-dimension
% Di       %M,K eigenvalues output

%% Function settings
% qn = xk * Pn

Mi = xk' * M * xk;              %intermedia mass matrix through product
Ki = xk' * K * xk;              %intermedia stiffness matrix
[P , Di] = eig(Ki , Mi);        %P for qn-output matrix; Di eigevalue
qn = xk * P;                    %qn-output

return


% Function running finish
end
```

d. In the development of codes, advantage must be taken of the banded structures of matrices involved. Application of methods such as Skyline storage method is encouraged.

d-1. The Matlab code:

* For memory calculating methods: using `whos` command to obtain the space storage magnitude:

```
function memo = memoryfunc
%MONITOR_MEMORY_WHOS uses the WHOS command and evaluates inside the BASE
%workspace and sums up the bytes.  The output is displayed in bytes.

meme = evalin('caller' , 'whos');
memo = sum([meme(:).bytes]);

if nargout == 0

  disp(['Memo ' , num2str(memo, '%.0f') , ' Bs'])
  clear memo


% Function finish running
end
```

e. Comment on the sensitivity of each method to numerical errors in matrices.

e-1. Theoretical function：
the theoretical method of Eigenvalues Sensitivity analysis (SENS):

$$KX_i = \lambda_i M X_i$$

$$\frac{\partial \lambda_i}{\partial a} = X_i^T \left(\frac{\partial K}{\partial a} - \lambda_i \frac{\partial M}{\partial a}\right)$$

$$\begin{bmatrix} \Delta\lambda_1 \\ \vdots \\ \Delta\lambda_n \end{bmatrix} = \{\Delta\lambda_i\} = [S]_i[\Delta a_i] = \begin{bmatrix} \frac{\partial \lambda_1}{\partial a_1} & \cdots & \frac{\partial \lambda_1}{\partial a_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial \lambda_n}{\partial a_1} & \cdots & \frac{\partial \lambda_n}{\partial a_n} \end{bmatrix} \begin{bmatrix} \Delta a_1 \\ \vdots \\ \Delta a_n \end{bmatrix}$$

$$\frac{\partial K}{\partial E} = \begin{bmatrix} k/E & k/E & \cdots & 0 \\ k/E & k/E & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$\frac{\partial M}{\partial \rho} = \begin{bmatrix} m/\rho & m/\rho & \cdots & 0 \\ m/\rho & m/\rho & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

$$[S]_{i1} = \{X_i^T\}(\frac{\partial K}{\partial E})\{X_i\}$$
$$[S]_{i2} = \{X_i^T\}\lambda_i(\frac{\partial M}{\partial \rho})\{X_i\}$$

e-2. The Matlab Code:

```
function Si = SENS(n , ki , mi , eige , eiga , E0 , p0)
%% Function settings
% for initial dKdEi function
dKdEi = zeros(n);           %dimension of dKdEi
dKdEi(1:2 , 1:2) = ki / E0; %right upper corner matrix forming for unit stiffness
dMdp0 = zeros(n);           %dimension of dMdp0
dMdp0(1:2 , 1:2) = mi / p0; %right upper corner matrix forming for unit mass

% for SENS looping calculating the difference of eigenvalues
for k = 1:n

    Si(k , 1) = eige(: , k)' * dKdEi * eige(: , k);          %{eiga_i}={Si}*{del_a}
    Si(k , 2) = eige(: , k)' * (-eiga(k)) * dMdp0 * eige(: , k);

end

% Finish Function Running
End
```

Main scripts of Q1-3 and Q4:

* Q1-3:

```
% Calculating the eigenvalues and corresponding eigenvectors of matrix A
% (metion later); using the known iteration methods to solve the continuous
% eigevalues by known the initial condition.
%-------------------------------------------------------------------------%
% Q1. Matlab Language
% Q2-a. RQI and HHD iteration
% Q2-b. QRITER
% Q2-c. SSI and RITZ
%-------------------------------------------------------------------------%
% Q2-d. Storage using of memory record
% Q2-e. Time running
%-------------------------------------------------------------------------%
%
% Zijun Fang
% CID: 01811420
% 18TH APR 2020
%
%-------------------------------------------------------------------------%
% clear workspace
%
% setup enviroment
%-------------------------------------------------------------------------%
clear;
close all;
clc


%% Initial Settings
n = 4;                                                  % Dimension of Matrix
itera = input('Iteration Number of Calculating:');      % Iteration Number:
itera

%% Matrix Settings
% upper and diagonal matrix forming
M_u = [0.6486, 0.4151, 0.02642];
M_d = [3.2525, 2.0868, 1.332, 0.4709];
K_u = [-4.0024, -2.9024, -2.0932];
K_d = [9.4955, 6.9048, 4.9956, 2.0932];


K_D = sparse(1:n , 1:n , ones(1,n).*K_d , n , n);
M_D = sparse(1:n , 1:n , ones(1,n).*M_d , n , n);
K_U = sparse(2:n , 1:n-1 , ones(1,n-1).*K_u , n , n);
M_U = sparse(2:n , 1:n-1 , ones(1,n-1).*M_u , n , n);


M_L = M_U';
K_L = K_U';
M = M_U + M_D + M_L;
K = K_U + K_D + K_L;              % orthogonal method of matrix combine: A =
U+D+L
```

```
A = full(M^(-1)*K);                % vibration natural freq calculation:
eiga^2*M*x = K*x

A = A + eye(n)*1e-15;
M = M + eye(n)*1e-15;
K = K + eye(n)*1e-15;              % singular matrix spares form into double
mode


%% Question 1-2 Answerings:
% Matlab language calculation

tic

[eigei , eigai , memoi] = eigs(K , M);
for k = 1:n
    eigei(:,k) = -sign(eigei(1,k))*eigei(:,k)/norm(eigei(:,k));    %norm
function for eige and eiga calculation loop
end

memoi = memoryfunc;              %Memory function to record storage by the
Matlab language

TimeStoppingi = toc;


% RQI and HHD calculation
tic

[eigaRH , eigeRH , memo1]=RQIHHD(A , n , itera);

TimeStopping1 = toc;


% QR Iteration calculation
tic

[eigaQR , eigeQR , memo2] = QRITER(A , n , itera);

TimeStopping2 = toc;


% Subspace Iteration calculation
tic

[eigaSI , eigeSI , memo3] = SSI(A , M , K , n , n , itera);

TimeStopping3 = toc;


%% Q1-2 Outcomes:
% Eige and Eiga outputs
fprintf(' \nQ1. Natural Frequencies (omega) and Relevant Eigenvectors of
the system:\n');
omegas = diag(eigai .^(1/2))'
Xs = eigei
```

```
fprintf(' \nQ2a. Rayleigh Quotient Iteration along with Householder
deflation:\n');
omeRH = diag(eigaRH .^(1/2))'
Xs_RH = eigeRH
fprintf(' \nQ2b. QR Iteration technique:\n');
omeQR = diag(eigaQR .^(1/2))'
Xs_QR = eigeQR
fprintf(' \nQ2c. SSI and RITZ:\n');
omeSI = diag(eigaSI .^(1/2))'
Xs_SI = eigeSI


% Accuracy of Eige and Eiga - Using Nth norm number of differences
eigaerRH = norm(eigaRH - eigai)
eigeerRH = norm(eigeRH - eigei)
eigaerQR = norm(eigaQR - eigai)
eigeerQR = norm(eigeQR - eigei)
eigaerSI = norm(eigaSI - eigai)
eigeerSI = norm(eigeSI - eigei)

% Storage using by different method
fprintf(' \nQ2d. Storage using in the initial matlab language
loop: %.0fbites.', memoi);
fprintf(' \nQ2d. Storage using in different function: RQI and
HHD: %.0fbites;QRITER: %.0fbites;SSI and RITZ: %.0fbites.', memo1 , memo2 ,
memo3);

% TimeStopping recording of different software
fprintf(' \nQ2d. Storage using in the initial matlab language loop: %fs.',
TimeStoppingi);
fprintf(' \nQ2d. Storage using in different function: RQI and
HHD: %fs;QRITER: %fs;SSI and RITZ: %fs.', TimeStopping1 , TimeStopping2 ,
TimeStopping3);
```

* Q4

```
% Making the calculation of sensitivity of eigenvalues of the system with
% the respect to the changes in the elements' parameters (p0 and E0);
% i-the elemental submatrix index
%-------------------------------------------------------------------------%
% Q4. Sensitive test of eigevalues

%
% Zijun Fang
% CID: 01811420
% 18TH APR 2020
%
%-------------------------------------------------------------------------%
% clear workspace
%
% setup enviroment
%-------------------------------------------------------------------------%
clear;
close all;
clc



%% Initial Settings
i = 4;                                              % index of
submatrices
n = i + 1;

%% set parameters
p0 = 3;
E0 = 4;


%% Matrix Settings
% upper and diagonal matrix forming

m1 = [2.2484 1.0087;1.0087 1.8052];
m2 = [1.4473 0.6486;0.6486 1.1594];
m3 = [0.9273 0.4151;0.4151 0.7410];
m4 = [0.5910 0.2642;0.2642 0.4709];             % elementary submatrices of
mass - mi

k1 = [5.4931 -5.4931;-5.4931 5.4931];
k2 = [4.0024 -4.0024;-4.0024 4.0024];
k3 = [2.9024 -2.9024;-2.9024 2.9024];
k4 = [2.0932 -2.0932;-2.0932 2.0932];           % elementary submatrices of
stiffness - ki


K_d = [k1(1,1) , k1(2,2)+k2(1,1) , k2(2,2)+k3(1,1) , k3(2,2)+k4(1,1) ,
k4(2,2)];
M_d = [m1(1,1) , m1(2,2)+m2(1,1) , m2(2,2)+m3(1,1) , m3(2,2)+m4(1,1) ,
m4(2,2)];
K_u = [k1(1,2) , k2(1,2) , k3(1,2) , k4(1,2)];
M_u = [m1(1,2) , m2(1,2) , m3(1,2) , m4(1,2)]; % upper cornor matrix and
diagonal matrix forming by mi and ki
```

```
K_D = sparse(1:n , 1:n , ones(1,n).*K_d , n , n);
M_D = sparse(1:n , 1:n , ones(1,n).*M_d , n , n);
K_U = sparse(2:n , 1:n-1 , ones(1,n-1).*K_u , n , n);
M_U = sparse(2:n , 1:n-1 , ones(1,n-1).*M_u , n , n); % dimension expanded
by the matrix expand


M_L = M_U';
K_L = K_U';
M = M_U + M_D + M_L;
K = K_U + K_D + K_L;              % orthogonal method of matrix combine: A =
U+D+L

A = full(M^(-1)*K);               % vibration natural freq calculation:
eiga^2*M*x = K*x


%% Eigenvalues sensitivity testing - i=4

% Matlab language eiges and eigas calculation
[eiges , eigas] = eigs(K , M);
eigas = diag(eigas);            % for diagonal matrix of eigenvalues

for k = 1:n

    eiges(: , k) = eiges(: , k) / norm(eiges(: , i));
    % norm number calculation by kth iteration of eigenvectors approaching

end


% SENS-func using
Si = SENS(n , k1 , m1 , eiges , eigas , E0 , p0)
```

Question 3

3-a-c: Comparing the Computation Time, Accuracy and Storage Capacity

* the accuracy calculation function/method:

Nth norm number of eigenvectors difference:

$$y_{egieer} = \|X - Xi\|_n$$

Nth norm number of eigenvalues difference:

$$y_{eigaer} = \|\lambda - \lambda_i\|_n$$

Xi, λ_i are the initial comparison group matrices to be run by the Matlab Language.

* the outcome table:

Methods	**Accuracy**		**Time (s)**	**Storage (bites)**
	Eiga	Eige		
Matlab	-	-	0.014325	1344
RQI/HHD	1.0751	2.6865	0.009194	432
QRITER	2.1838e-07	9.9462e-16	0.003652	936
SSI/RITZ	3.2541e-09	1.0478e-15	0.002896	1072

* the description between these methods:

For the aspect of comparing accuracy of eigenvalues and corresponding eigenvectors, which the RQI/HHD method has the largest gap to its calculation done by the Matlab language directly; also, the SSI/RITZ has the smallest. By requiring in different stage of eigenvalues and eigenvectors to obtain the relevant values, which the RQI/HHD method iteration the eigenvalues and eigenvectors together, to reduce the space using, however, increasing the error occurred.

For the timing and storage demonstrate at the upon table, which has been display in negative correlation. In details, the storage takes up in the system to reduce the calculation cost in time. However, comparing to its initial condition of Matlab, directly working out by the internal language which will cost more than the iteration loops methods.

Question 4

4-e. Subroutine SENS – to calculate the sensitivity of eigenvalues of the system.

* the outcome table:
S1: by changing the Young's Modulus (E_0 = 4) of the system;
S2: by changing the density (ρ_0 = 3) of the system;

S1	S2
0.55739	-0.92751
1.59252	-1.69778
0.35765	-0.36160
0.02549	-0.12266
6.163e-33	1.35926e-16

* the description of sensitivity through the eigenvalues:
To influence the different nodes of the system, pivoting the submatrices of mi and ki, where the element eigenvalues of eigenvalues, in fact, changing the elements' parameter dose not show any specific outcome of data difference. In different nodes of element, the sensitivity of each eigenvalue shows that changing the density of system would have larger difference of eigenvalues.
The more sensitivity of eigenvalues has been created by changing the Young's Modulus results.