1 Introduction

ERA, the *Eigensystem Realization Algorithm*, is widely used as a modal analysis technique, generating a system realization using time-domain response input and output data. It was proposed by Juang and Pappa in 1985.\(^1\)

2 General Processes

2.1 Impulse Response \(h[k]\)

Several equations based on (Caicedo, 2011).\(^2\) Given the Impulse Response \(h[k]\), which induced by equation (1)

\[ y[k] = h[k]p[0] \]  

\(y[k]\) is system response (the output) and \(p[0]\) is impulse input. \(h[k]\) is a \(q \times p\) vector, for \(q\) outputs and \(p\) inputs.

2.2 Hankel Matrix

Then we begin to constructing *Hankel Matrix*. *Hankel Matrix* is shown in equation (2)

\[
H[k-1] = \begin{bmatrix}
h[k] & h[k+1] & \cdots & h[k+s-1] \\
h[k+1] & h[k+2] & \cdots & h[k+s] \\
\vdots & \vdots & \ddots & \vdots \\
h[k+r-1] & h[k+r] & \cdots & h[k+r+s-2]
\end{bmatrix}
\]  

*Hankel Matrix* is a \((ny \times r) \times (np \times s)\) matrix. The parameter \(r\) and \(s\) are chosen by programmer. \(ny\) is the number of outputs and \(np\) is the number of inputs. In other words, \(ny = q\) and \(np = p\).

2.3 Execute SVD

Thin SVD to \(H[0]\), obtained equation (3)

\[
H[0] = U \Sigma V^T
\]  

We extract the first \(n \times n\) part of matrix(Thin SVD), so Matrix \(U\) is \((ny \times r) \times n\), \(\Sigma\) is \(n \times n\), and \(V\) is \((np \times s) \times n\). \(n\) is the number of state variables.

2.4 Inducing Matrix A B and G

Constructing auxiliary matrix \(E_y\) and \(E_p\) as equation (4)

\[
E_y = \begin{bmatrix}
I_{ny \times ny} \\
0_{ny \times ny} \\
\vdots \\
0_{ny \times ny}
\end{bmatrix}_{(ny \times r) \times ny} \\
E_p = \begin{bmatrix}
I_{np \times np} \\
0_{np \times np} \\
\vdots \\
0_{np \times np}
\end{bmatrix}_{(np \times s) \times np}
\]
Then we got matrix used in state-space representation in equation (5)

\[
A = \Sigma^{-1/2} U^T H[1] V \Sigma^{-1/2} (n \times n) \tag{5}
\]

Input Matrix: \( B = \Sigma^{1/2} V^T E_p (n \times p) \) \tag{6}

Output Matrix: \( G = E_y^T U \Sigma^{1/2} (q \times n) \) \tag{7}

3 Useful Techniques

3.1 How to construct Hankel Matrix in MATLAB

As shown in equation (2), the Hankel Matrix is a block matrix, so we use cell structure to input the block matrix via loop.

```matlab
function [A,B,G]=Era(h,n,s ,r)

% input
% h: Impulse response
% n: Degree of freedoms(Only if structural system)
% r: number of rows
% s: number of columns
% output
% A: State Matrix, 2n*2n. n is DOFs, 2n is the number of state variables
% B: Input Matrix, 2n*p. p is the number of input
% G: Output Matrix, q*2n. q is the number of output

[MM,NN]=size(h);

% Generate Hankel Matrix H0 & H1
H0=cell(r,s);
H1=cell(r,s);

k=1; % Hankel Matrix H0
for i=1:r
    for j=1:s
        H0{i , j}=h(:,k+i+j-2);
    end
end

k=2; % Hankel Matrix H1
for i=1:r
    for i=1:s
        H1{i , j}=h(:,k+i+j-2);
    end
end

% Transform cell to matrix
H0=cell2mat(H0);
H1=cell2mat(H1);
```
### 3.2 The number of state variables for structure

Attention! The number of state variables of structure is $2n$, and $n$ is DOFs. In modal analysis, we use displacement and velocity to describe the state of a structure, so the state vector $X$ is

$$X = \begin{bmatrix} x(1), x(2), \cdots, x(n), \dot{x}(1), \dot{x}(2), \cdots, \dot{x}(n) \end{bmatrix}^T_{2n \times 1} \quad (8)$$

Therefore, after SVD, we need to extract the first $2n \times 2n$ part of matrix if the input of ERA is DOFs not the number of state variables. The MATLAB code is shown below.

```matlab
% Singular Value Decomposition
[U,S,V] = svd(H0,0);
% Extract first n*n part of matrix
U = U(:,1:2*n);
S = S(1:2*n,1:2*n);
V = V(:,1:2*n);
```

### 3.3 The whole MATLAB code

```matlab
function [A,B,G]=Era(h,n,s,r)

%% input
%% h: Impulse response
%% n: Degree of freedoms(Only if structural system)
%% r: number of rows
%% s: number of columns

%% output
%% A: State Matrix, 2n*2n. n is DOFs, 2n is the number of state variables
%% B: Input Matrix, 2n*p. p is the number of input
%% G: Output Matrix, q*2n. q is the number of output

[MM,NN]=size(h);

%% Generate Hankel Matrix H0 & H1
H0=cell(r,s);
H1=cell(r,s);

k=1; % Hankel Matrix H0
for i=1:r
    for j=1:s
        H0{i,j}=h(:,k+i+j-2);
    end
end

k=2; % Hankel Matrix H1
for i=1:r
    for j=1:s
        H1{i,j}=h(:,k+i+j-2);
    end
end

%% Transform cell to matrix
H0=cell2mat(H0);
```
Study notes

<table>
<thead>
<tr>
<th>Hi=cell2mat(H1);</th>
</tr>
</thead>
</table>
| \ [% Singular Value Decomposition  
\ [U,S,V] = svd(H0,0);  
\ [% Extract first n*n part of matrix  
\ U = U(:,1:2*n);  
\ S = S(1:2*n,1:2*n);  
\ V = V(:,1:2*n);  |

\ [% Generate Auxiliary Matrix Ey and Ep  
ny=M; np=1; % ny=M and np=1 because ny should be equal to the number of outputs and np should be equal to the number of inputs  
Ey = [eye(ny); zeros(ny*(r-1),ny)]; % Generate Ey(nyr*ny)  
Ep = [eye(np); zeros(np*(s-1),np)]; % Generate Ep(nps*np)  |

\ [% Induce the A B G matrix  
A = S^(-1/2) * U' * H1 * V * S^(-1/2);  
B = S^(1/2) * V' * Ep;  
G = Ey' * U * S^(1/2);  |

References


Appendices

A Matlab code to generate vibration mode

```matlab
function [Omega, Damping, VibrationMode]=GenerateMode(A, B, G, T)

% input
% A: State Matrix, 2n*2n, n is DOFs, 2n is the number of state variables
% B: Input Matrix, 2n*p, p is the number of input
% G: Output Matrix, q*2n, q is the number of output
% T: Sampling time, real number

% output
% Omega: Frequency Vector, n*1
% Damping: Damping Ratio Vector, n*1
% VibrationMode: Vibration Mode, n*n matrix

[VM_notSorted, Omega_notSorted]=eig(A);
Omega_notSorted=diag(logm(Omega_notSorted)./T);

% Sorting frequency according to Imaginary part of Omega_notSorted
PositiveImagNumber=find(imag(Omega_notSorted)>0);
% PositiveImagNumber is the serial number of elements, no order.
[Dummy, SortNumber]=sort(imag(Omega_notSorted(PositiveImagNumber))); % Sorting frequency descending order
% "SortNumber" is a relative sequence of Omega_notSorted’s element which has a positive imaginary, in descending order considering imaginary

% Inducing frequency in ascending order
SortResult=PositiveImagNumber(SortNumber); % SortResult is the descending sequence of all elements with positive imaginary.

for i=1:length(SortResult)
    lambda(i)=Omega_notSorted(SortResult(i)); % Sorting frequency in descending order of imaginary
    VibrationMode(:,i)=VM_notSorted(:,SortResult(i)); % Sorting eigenvector in descending order of imaginary
end

Omega=abs(lambda); % Frequency is module of lambda
Damping=-real(lambda)./Omega; % Damping ratio is real part of lambda divided by frequency
```

Omega=abs(lambda); % Frequency is module of lambda
Damping=-real(lambda)./Omega; % Damping ratio is real part of lambda divided by frequency