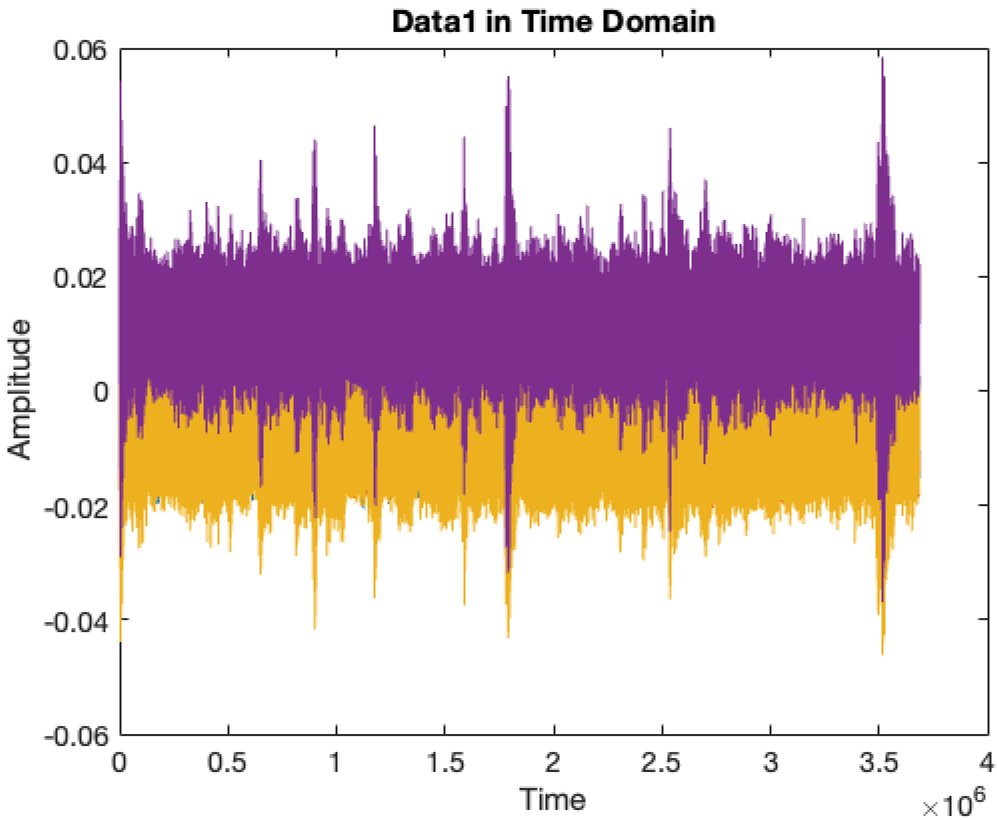
Table of Contents


```
clearvars;
close all;
clc;

set(0,'defaultAxesFontSize',14)

currentFile = matlab.desktop.editor.getActiveFilename;
currentFileDir = fileparts(currentFile);

% Get a list of all .mat files in the folder
fileList = dir(fullfile(currentFileDir, '*.mat'));
numFiles = length(fileList);

disp(['Data1:' , fileList(1).name])
filePath = fullfile(currentFileDir, fileList(1).name);
Data1 = load(filePath).y;
Data1 = Data1./0.01;
fsamp1=load(filePath).fsamp;
```

Data1:Data.mat

```
figure;
plot(Data1);
xlabel('Time');
ylabel('Amplitude');
title('Data1 in Time Domain');
```

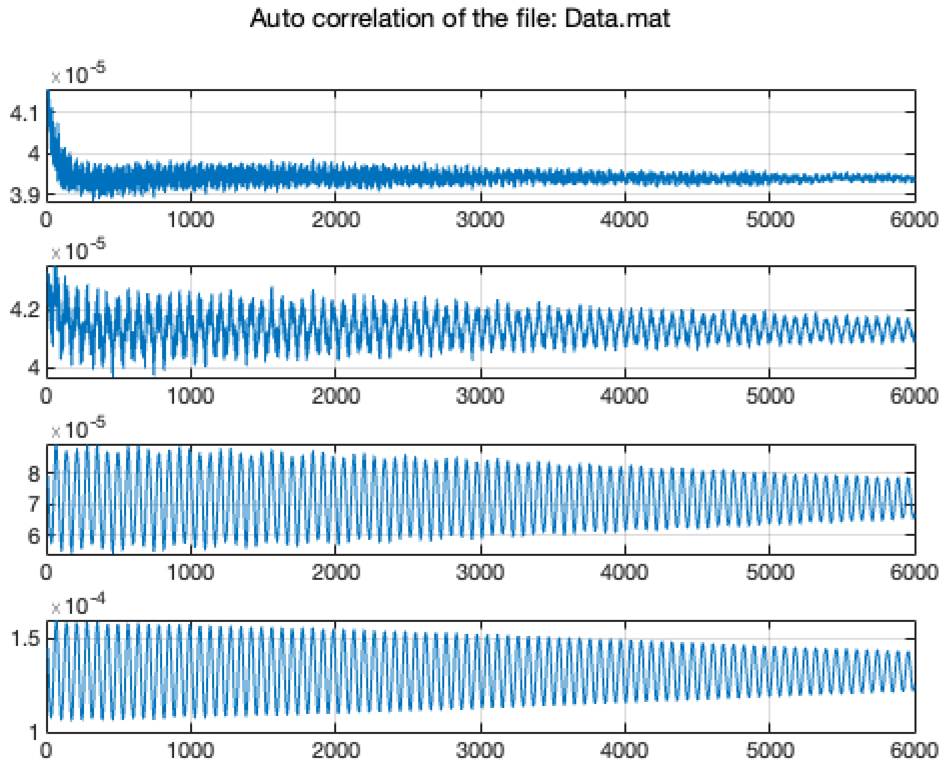


Auto and cross correlation matrix

```
[G1,lag1] = Autocorr(Data1);
```

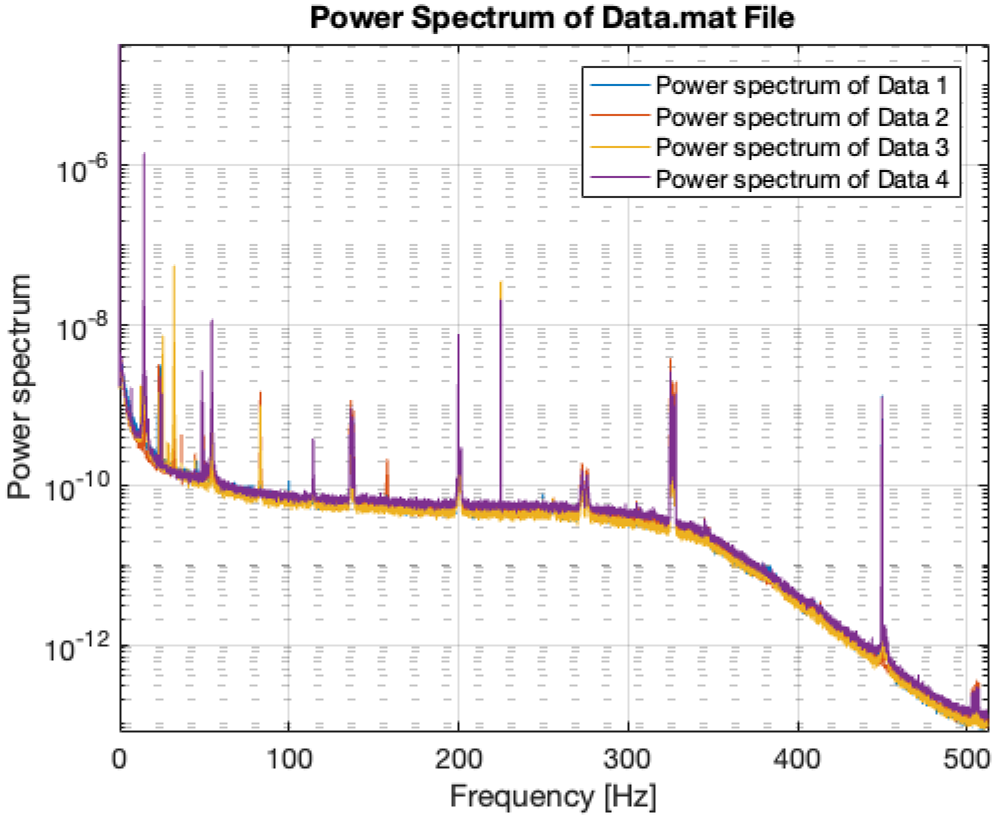
Plotting the Auto_correlation

```
limit_x=6000;
AutocorrVisual(G1,lag1,limit_x,fileList(1).name);
```



Plotting the PSD

```
overlap=0.6;
splitsec = 30;
[Gyy1,freq_of_frf1] =
PowerSpecPlot(Data1,fsamp1,splitsec,overlap,fileList(1).name);
```



Main ITDM

```
order=100;
p=2000;
matrix1 = ITDM2svd(G1,fsamp1,order,p,fileList(1).name,Gyy1,freq_of_frf1);

poles_num = 6;
params_matrix1 =ParamFilter(matrix1,fsamp1,poles_num);
selected_ind = selectColumnsWithCheckboxes(params_matrix1,1:6);
params_matrix1 = params_matrix1(:,selected_ind);
```

Starting ITDM:
Doing it for the order of the sys:

```
%visualization of the modes

ModeVisual(params_matrix1,'data1')
```

Functions

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Auto and cross correlation

```
function [G,lag] = Autocorr(Data)
n = size(Data, 2); % Number of DOFs
N = size(Data, 1); % Number of instances
G = zeros(n,n, N);

for i = 1:n
    for j = 1:n

        [correlation,lag] = xcorr(Data(:, i), Data(:, j), 'unbiased');
        positive_indices = lag >= 0;

        % Extract positive parts of the correlation
        positive_correlation = correlation(positive_indices);
        G(i,j, :) = positive_correlation;
    end
end

lag = lag(positive_indices);
end

function AutocorrVisual(G,lag,limit_x,name)

starting = 14;
figure;
for j= 1:4
    subplot(4,1,j);
    plot(lag(starting:end), squeeze(G(j,j,starting:end)));
    grid on
    xlim([0,limit_x]);
end
linkaxes([subplot(4,1,1), subplot(4,1,2), subplot(4,1,3), subplot(4,1,4)],
'x');
sgtitle(['Auto correlation of the file: ',name]);
end

function [autocross_mean,freqs]= mycross(signal1,signal2,
splitsec,fsamp,overlap,Win)
dt = 1/fsamp;
windows1 = mysplit(signal1,splitsec/dt,overlap);
windows2 = mysplit(signal2,splitsec/dt,overlap);
% Pre-allocate for average cross-spectrum (assuming same size windows)
num_windows = length(windows1);

df = 1/splitsec;

if (splitsec*fsamp/2)==(floor(splitsec*fsamp/2))
    freqs=0:df:(fsamp-df)/2;

else
    freqs=0:df:(fsamp)/2;
end
```

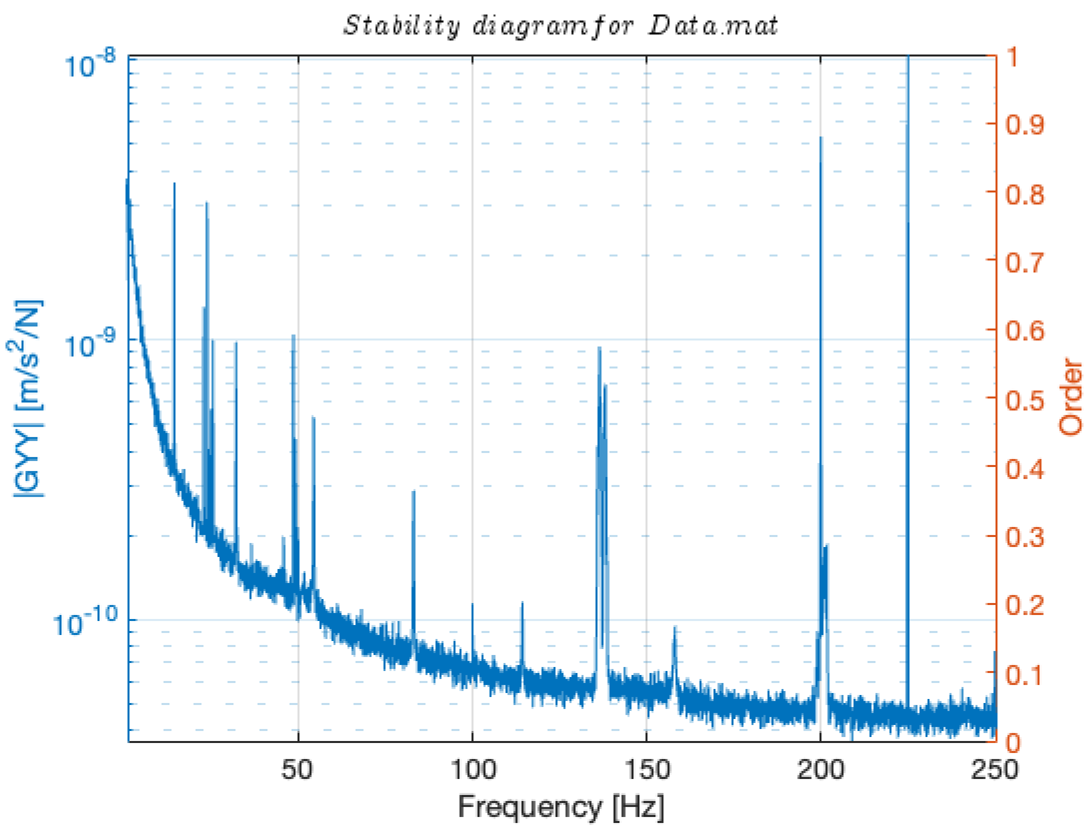
```
freqs = freqs';
NF =length(freqs);

autocross=zeros(length(freqs)*2,num_windows);
for i = 1:num_windows
    window1 = windows1{i};
    window2 = windows2{i};
    if isequal(length(window1),length(freqs)*2)
        fft_window1 = fft(window1.*Win);
        fft_window1 = fft_window1./(fsamp*splitsec);
        fft_window2 = fft(window2.*Win);
        fft_window2 = fft_window2./(fsamp*splitsec);
        cross_spectrum = conj(fft_window2) .* fft_window1;

        autocross(:,i) = cross_spectrum;
    end

end

autocross_mean=mean(autocross(1:NF,:),2);

end
```

Plotting the power spectrum

```
function [Gyy,freq_of_frf]=PowerSpecPlot(Data, fsamp, splitsec, overlap, name)

Win = hanning(splitsec * fsamp);

figure;
plotHandles = [];
legendEntries = {};
colors = lines(size(Data,2)); % Define a color scheme

for i = 1:size(Data,2)
    [Gyy(:,i), freq_of_frf] = mycross(Data(:,i), Data(:,i), splitsec, fsamp,
overlap, Win);

    % Store the handles for the plot
    h = semilogy(freq_of_frf, Gyy(:,i), 'Color', colors(i,:)); % Use
different colors
    hold on
    plotHandles = [plotHandles; h]; % Collect the plot handle
    legendEntries{end+1} = ['Power spectrum of Data ', num2str(i)]; % Collect
legend entry

    title(['Power Spectrum of ', name, ' File'])
    grid on
    axis tight
end

% Adding axis labels
xlabel('Frequency [Hz]')
```

```
ylabel('Power spectrum')

% Adding the legend
legend(plotHandles, legendEntries, 'Location', 'best')
hold off;

end

function Gxy=CrossSpecPlot(Data,fsamp,splitsec,overlap,name)

Win=hanning(splitsec*fsamp);

figure;
plotHandles = [];
legendEntries = {};
cisi = [5, 90, 25, 50]; % Define the cisi percentages
for i = 2:size(Data,2)
    [Gxy(:,i), freq_of_frf] = mycross(Data(:,1), Data(:,i), splitsec, fsamp,
overlap, Win);

    % Store the handles for the first subplot
    subplot(2,1,1)
    h = semilogy(freq_of_frf, abs(Gxy(:,i)));
    hold on
    plotHandles = [plotHandles; h]; % Collect the plot handle
    legendEntries{end + 1} = ['cross spectrum with $csi = ',
num2str(cisi(i)), '\%$']; % Collect legend entry in LaTeX format

    title(['Cross spectrum of the sensor on 5% with others in', name, '
file'])
    grid on
    axis tight
    xlabel('Frequency [Hz]')
    modulus_symbol = '|';
    ylabel([modulus_symbol 'Cross spectrum' modulus_symbol])
    subplot(2,1,2)
    plot(freq_of_frf, angle(Gxy(:,i)))
    xlabel('Frequency [Hz]')
    ylabel('\angle Cross spectrum','Interpreter','latex')
    grid on
    hold on
end

% Adding the legend to the first subplot
subplot(2,1,1)
legend(plotHandles, legendEntries, 'Location', 'best','Interpreter', 'Latex')

subplot(2, 1, 1)
xline(24.7, 'k--', 'LineWidth', 1.5);
xline(25.067, 'k--', 'LineWidth', 1.5);

subplot(2, 1, 2)
xline(24.7, 'k--', 'LineWidth', 1.5);
xline(25.067, 'k--', 'LineWidth', 1.5);
```

```
hold off;

linkaxes([subplot(2,1,1), subplot(2,1,2)], 'x');


end
```



Ibrahim method function

```
% Hankel matrix (R) creation
function [R1,R2] = myHankel(g,p,m)

% myHankel constructs two Hankel-like matrices R1 and R2 from a 3D matrix g.
%
% INPUTS:
% g - a 3D matrix where each slice g(:,:,k) represents Auto and cross corr
matrix at time k
% p - an integer representing the number of columns (h_p) in hannle matrix
% m - an integer representing the number of rows (h_m) in hannle matrix
%
% OUTPUTS:
% R1 - a matrix constructed from g with dimensions (4*m, (p-1)*4)
% R2 - another matrix constructed from g with dimensions (4*m, (p-1)*4)

% starting time
```

```
dt0=2;

% Determine the number of rows in each slice of g
m=m+dt0;
% Creating the G(t) and G(t+dt) matrix
o=size(g,1);
A1=zeros(o*m,o);

for k = dt0:m
    A1(o*(k-1)+1:o*k,:)=g(:,:,k);
end

R1 = zeros(o*m,(p-1)*o);
R2 = zeros(o*m,(p-1)*o);
R1(:,1:o)=A1;

for l = 1:p
    % Shift A1 up by o rows and append the next slice of g
    A1 = [A1(o+1:end,:);g(:,:,m+l)];

    if l~=p
        R1(:,o*l+1:o*(l+1)) = A1;
    end

    R2(:,o*(l-1)+1:o*(l))=A1;
end

end

% Ibrahim method without svd reduction
function [matrix,stable_freqs] = ITDM2(G, fsamp, order, p, name, Gyy,
freq_of_frf)
    % inputs:
    % G: Correlation matrix or response data in the form of (n*n*number of
instances)
    % fsamp: Sampling frequency.
    % order: Maximum order of the system to be considered.
    % p: the number of time lags used.
    % name: Name of the plot
    % Gyy: Response data for the plot
    % freq_of_frf: Frequencies for the FRF plot

    % outputs:
    % matrix: Filtered matrix of parameters based on the stable poles

    dt = 1/fsamp;
    eigen_freqs = cell(order, 1);  % Store frequencies by order
    eigen_damps = cell(order, 1);  % Store damping ratios by order
    eigen_freqs2 = [];
    eigen_damps2 = [];
    sai=[];
    disp('Starting ITDM:')
    disp('Doing it for the order of the sys:')
```

```
    % Initialize the figure and plot Gyy
    figure(10);
    yyaxis left
    semilogy(freq_of_frf, Gyy(:,1))
    title(['$Stability\ diagram for \ ',name,'$'],'Interpreter','latex')
    axis tight
    grid on
    xlabel('Frequency [Hz]')
    ylabel('|GYY| [m/s^2]^2')
    xlim([1,250])
    yyaxis right
    ylabel('Order')
    hold on

    % Ibrahim method for system identification
    for i = 1:order
        m = 2*i;
        [R1, R2] = myHankel(G, p, m);

        if mod(i, 2) == 0
            disp(['N = ', num2str(i)]);
        end

        % Parameter extraction
        [saihat, exp_sr] = eig(R2 * pinv(R1));
        sai = [sai, saihat(1:4,:)];  % Unrefined Mode shapes

        D = diag(exp_sr);  % Diagonals of the exp matrix

        pole_sr = log(D) / dt;

        cisi = sqrt(1 ./ ((imag(pole_sr) ./ real(pole_sr)).^2 + 1));
        eigen_damps2=[eigen_damps2,cisi'];

        freque = -real(pole_sr) ./ cisi / 2 / pi;
        eigen_freqs2 = [eigen_freqs2, freque']; % Concatenate with existing
eigen_freqs2    % Unrefined eigenfrequencies


        sai2{i} = saihat(1:4,:);
        eigen_damps{i} = cisi';
        eigen_freqs{i} = freque';

    end
    matrix = [eigen_freqs2;eigen_damps2;sai];
    % Identify stable poles
    stable_freqs = identify_stable_poles(eigen_freqs,eigen_damps,sai2);

    % Plot stabilization diagram
    for i = 1:order
        scatter( eigen_freqs{i},repmat(i, length(eigen_freqs{i}), 1), 'k+');
% Unrefined frequencies
    end
    scatter(stable_freqs(:, 2),stable_freqs(:, 1), 'filled', 'DisplayName',
```

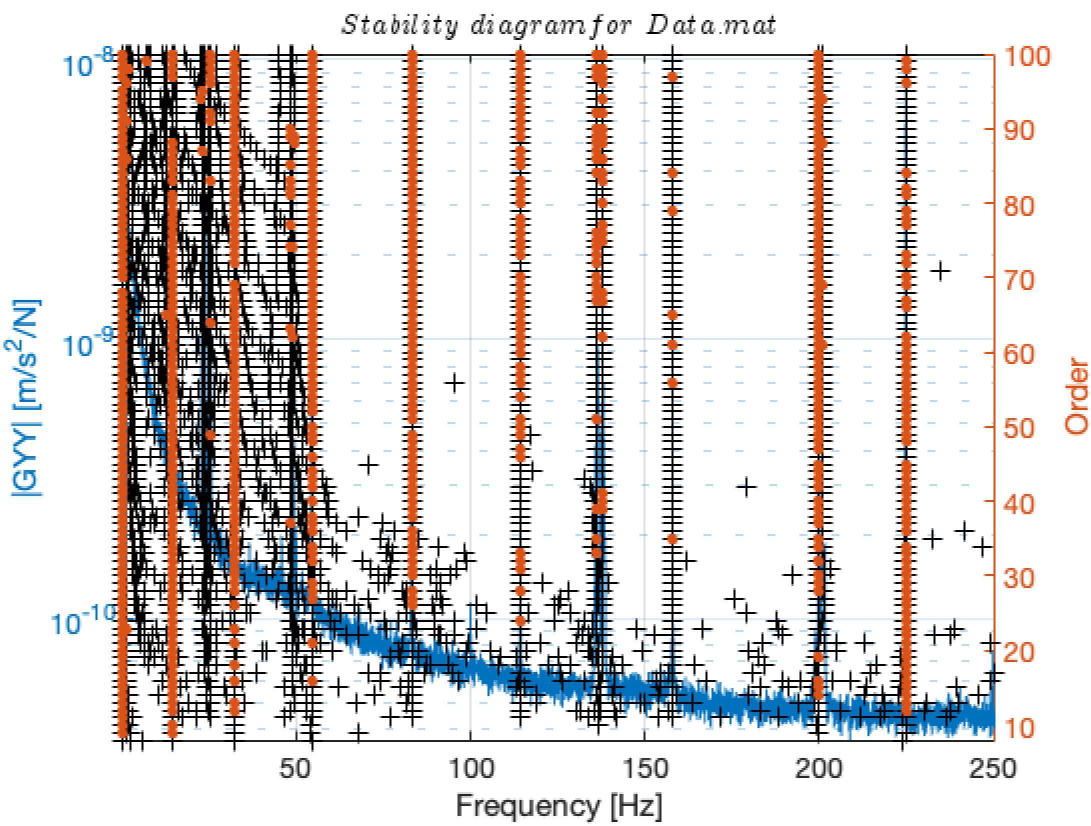
```
'Stable Frequencies');
    hold off

    % Assign the matrix output
end

% Ibrahim method incorporating svd
function [matrix,stable_freqs] = ITDM2svd(G, fsamp, order, p, name, Gyy,
freq_of_frf)
    % inputs:
    % G: Correlation matrix or response data in the form of (n*n*number of
instances)
    % fsamp: Sampling frequency.
    % order: Maximum order of the system to be considered.
    % p: the number of time lags used.
    % name: Name of the plot
    % Gyy: Response data for the plot
    % freq_of_frf: Frequencies for the FRF plot

    % outputs:
    % matrix: Filtered matrix of parameters based on the stable poles

    dt = 1/fsamp;
    eigen_freqs = cell(order, 1);  % Store frequencies by order
    eigen_damps = cell(order, 1);  % Store damping ratios by order
    eigen_freqs2 = [];
    eigen_damps2 = [];
    sai=[];
    disp('Starting ITDM:')
    disp('Doing it for the order of the sys:')

    % Initialize the figure and plot Gyy
    figure(10);
    yyaxis left
    semilogy(freq_of_frf, Gyy(:,1))
    title(['$Stability\ diagram for \ ',name,'$'],'Interpreter','latex')
    axis tight
    grid on
    xlabel('Frequency [Hz]')
    ylabel('|GYY| [m/s^2/N]')
    xlim([1,250])
    yyaxis right
    ylabel('Order')
    hold on

    % Ibrahim method for system identification
    for i = 8:order
        m = 2*i;
        [R1, R2] = myHankel(G, p, m);

        if mod(i, 2) == 0
            disp(['N = ', num2str(i)]);
        end
        [U, S, V] = svd(R1,"econ");
```

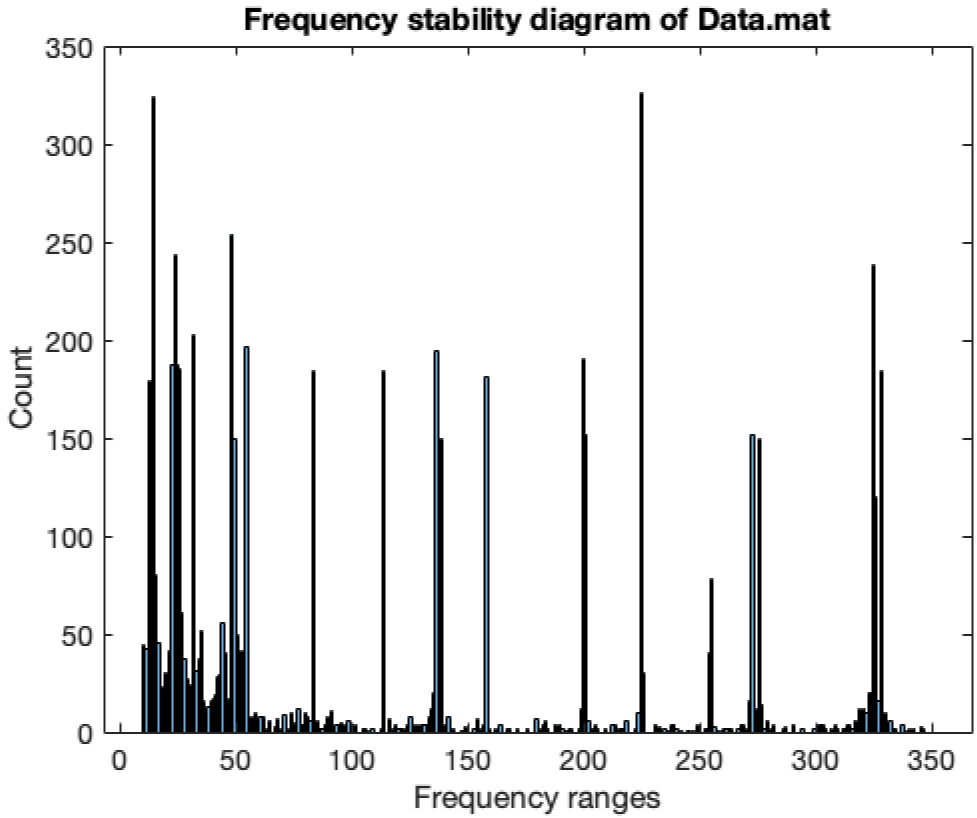
```
        k = m;
        if k > 20
            % If the iteration number is odd, add 1
            k = 90;
        end

        % Step 2: Decide on the number of singular values to keep (k)
         % Example: keeping the first 5 singular values

        % Step 3: Truncate the matrices
        U_prime = U(:, 1:k);

        [saihat_prime,exp_sr]=eig(U_prime.'*R2*pinv(U_prime.'*R1));
        saihat = U_prime*saihat_prime;

        sai = [sai, saihat(1:4,:)];  % Unrefined Mode shapes

        D = diag(exp_sr);  % Diagonals of the exp matrix
        pole_sr = log(D) / dt;

        cisi = sqrt(1 ./ ((imag(pole_sr) ./ real(pole_sr)).^2 + 1));
        eigen_damps2=[eigen_damps2,cisi'];

        freque = -real(pole_sr) ./ cisi / 2 / pi;
        eigen_freqs2 = [eigen_freqs2, freque']; % Concatenate with existing
eigen_freqs2    % Unrefined eigenfrequencies

        sai2{i} = saihat(1:4,:);
        eigen_damps{i} = cisi';
        eigen_freqs{i} = freque';

    end
    matrix = [eigen_freqs2;eigen_damps2;sai];
    % Identify stable poles
    stable_freqs = identify_stable_poles(eigen_freqs,eigen_damps,sai2);

    % Plot stabilization diagram
    for i = 1:order
        scatter( eigen_freqs{i},repmat(i, length(eigen_freqs{i}), 1), 'k+');
% Unrefined frequencies
    end
    scatter(stable_freqs(:, 2),stable_freqs(:, 1), 'filled', 'DisplayName',
'Stable Frequencies');
    hold off

    freq_interval = 0.002*fsamp/2;                    % based on 0.1% change
    freq_bins = 10:freq_interval:350;

    figure;
    histogram(eigen_freqs2, freq_bins, 'Edgecolor', 'k');
    xlabel('Frequency ranges');
    ylabel('Count');
    title(['Frequency stability diagram of ',name]);
```

```
    % Assign the matrix output
end

% rough extraction of the stable poles
function stable_freqs = identify_stable_poles(eigen_freqs, eigen_damps,sai)
    % Identify stable poles by comparing the frequencies and damping ratios
    % of poles across different orders.

    stable_freqs = [];

    for i = 2:length(eigen_freqs)
        for j = 1:length(eigen_freqs{i})
            if any(abs(eigen_freqs{i}(j) - cell2mat(eigen_freqs(i-1))) <
1e-3) && ...
               any(abs(eigen_damps{i}(j) - cell2mat(eigen_damps(i-1))) < 1e-3)
               stable_freqs = [stable_freqs; i, eigen_freqs{i}
(j),eigen_damps{i}(j),sai{i}(:,j).'];
            end
        end
    end
end
```

```
N = 8
N = 10
N = 12
N = 14
N = 16
N = 18
N = 20
N = 22
N = 24
N = 26
N = 28
N = 30
N = 32
N = 34
N = 36
N = 38
N = 40
N = 42
N = 44
N = 46
N = 48
N = 50
N = 52
N = 54
N = 56
N = 58
N = 60
N = 62
N = 64
N = 66
N = 68
N = 70
```

```
N = 72
N = 74
N = 76
N = 78
N = 80
N = 82
N = 84
N = 86
N = 88
N = 90
N = 92
N = 94
N = 96
N = 98
N = 100
```





Stability analysis

```
% detailed stability analysis
function cond_matrix3 = ParamFilter(matrix,fsamp,pole_nums)

freq_interval = 0.0002*fsamp/2;                              % based on 0.1% change
freq_bins = 10:freq_interval:180;
[num1,edge1] = histcounts(matrix(1,:), freq_bins);
[~, indices] = sort(num1, 'descend');


edge1 = sort(edge1(indices(1:pole_nums)),'ascend');

% cond_matrix2=[];
cond_matrix3 = [];
disp('The ranges of stable frequencies are:')

for i=1:pole_nums
    freq_Lower_bound = edge1(i);
    freq_Uper_bound = edge1(i)+freq_interval;


    disp([num2str(freq_Lower_bound), ...
        ' & ', ...
        num2str(freq_Uper_bound)] ...
```

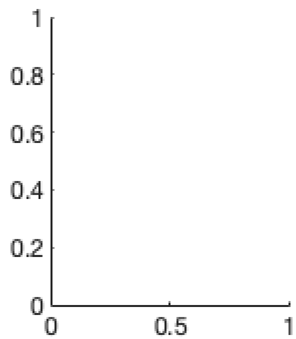
```
    );


% Specify the conditions for each row
row1_condition = matrix(1, :) >= freq_Lower_bound & ...
    matrix(1, :) <= freq_Uper_bound;


% Find the columns that satisfy all conditions
columns_satisfying_conditions = all(row1_condition, 1);
cond_matrix = matrix(:, columns_satisfying_conditions);



max_damp_bin =  0.01;
damp_interval = max_damp_bin*0.01;                 % based on 5% change
damp_bins = 0.001:damp_interval:max_damp_bin;


[num1,edge2] = histcounts(cond_matrix(2,:), damp_bins);
[~, indices2] = sort(num1, 'descend');

damping_Lower_bound = edge2(indices2(1));
damping_Upper_bound = edge2(indices2(1))+damp_interval;

row2_condition = cond_matrix(2, :) >= damping_Lower_bound & ...
    cond_matrix(2, :) <= damping_Upper_bound;

cond_matrix2 =cond_matrix(:,all(row2_condition,1));
mac_stabel =[];
if ~isempty(cond_matrix2)

    threshold = 0.9; % Set the desired threshold value

    % Loop through each pair of mode shapes
    for i = 1:size(cond_matrix2, 2)
        for j = i+1:size(cond_matrix2, 2)
            Phi1 = cond_matrix2(3:end, i);
            Phi2 = cond_matrix2(3:end, j);

            % Calculate the Mac value
            mAc = Mac(Phi1, Phi2);

            % Check if the Mac value is above the threshold
            if mAc >= threshold
                mac_stabel = [mac_stabel,cond_matrix2(:,j)];
            end
        end
    end
   if ~isempty(mac_stabel)
   mac_stabel =  mac_stabel(:,1);
   end
end
cond_matrix3 = [cond_matrix3,mac_stabel];
```

```
end


end

function mAc=Mac(Phi1,Phi2)
mAc= (abs(Phi1'*Phi2))^2./((Phi1'*Phi1).*(Phi2'*Phi2));
end
```

```
The ranges of stable frequencies are:
14.4032 & 14.5056
32.1184 & 32.2208
54.4416 & 54.544
83.0112 & 83.1136
114.2432 & 114.3456
157.8656 & 157.968
```

splitting the time domain signal with windowing and overlapping

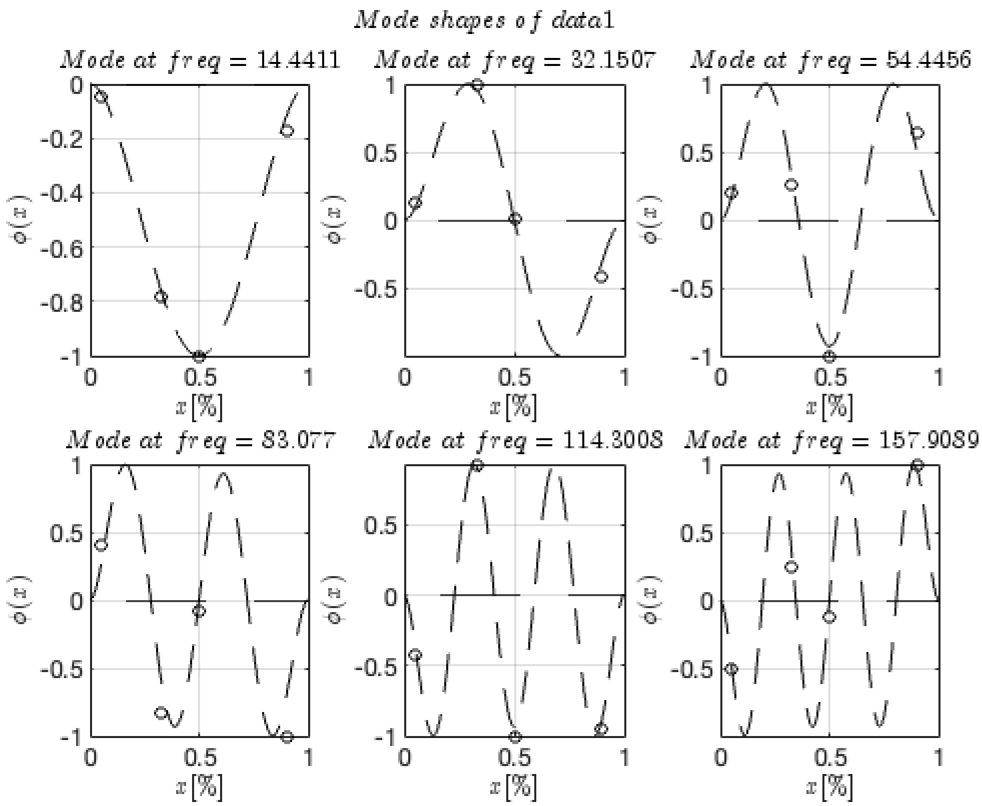
```
function windows = mysplit(signal, window_size, overlap)


% Error handling for overlap value
if overlap < 0 || overlap > 1
    error('Overlap must be between 0 and 1');
end

% Calculate hop size (distance between window starts)
hop_size = window_size * (1 - overlap);

% Calculate number of windows (rounded up)
num_windows = ceil(length(signal) / hop_size);

% Pre-allocate windows cell array
windows = cell(1, num_windows);

% Loop through windows and extract portions of the signal
for i = 1:num_windows
    % Calculate start and end index for current window
    start_index = (i - 1) * hop_size + 1;
    end_index = min(start_index + window_size - 1, length(signal));

    % Extract window from signal
    windows{i} = signal(start_index:end_index);
end
end
```



Complex to real mode shapes

```
function Real=ComplexModeToRealMode(Complex)
% This function converts the complex mode shape to the real valued one
% Reference: Operationa modal analysis of civil engineering structures page
% 182 and 183
% Rotate the complex mode shapes (see the RotationMaxCor function, below)
Complex=RotationMaxCor(Complex);
% 1: find the modulus of the mode shape
Modul=abs(Complex);

% 2: normalize the modulus
if nargin < 2
    Modul=Modul/max(Modul);
end

% 3: Find the phase of each component
Phase=angle(Complex);

% 4: find the sign of each component
for I=1:size(Complex,1)
    if Modul(I,1)~=0
        Sign(I,1)=SignFinder(Phase(I));
    else
        Sign(I,1)=0;
```

```
    end
end

% 5: compute the real valued mode shape
Real=Modul.*Sign;

end

function Sign=SignFinder(In)
if In>=0 && In<pi/2
    % First quarter
    Sign=+1;
elseif In>=pi/2 && In<=pi
    % Second quarter
    Sign=-1;
elseif In>=-pi && In<-pi/2
    % Third quarter
    Sign=-1;
elseif In>=-pi/2 && In<0
    % Forth quarter
    Sign=+1;
end
end

function out=RotationMaxCor(In)
% This function computes the maximum correlation line and then rotate the
% data with respect to this correlation
X=real(In);
Y=imag(In);
p=polyfit(X,Y,1);% Fit a first order line to the data
Teta=-atan(p(1)); % angle of maximum correlation line
Rot=[cos(Teta)  -sin(Teta) ; sin(Teta)  cos(Teta)]; % Rotation matrix
for I=1:size(In,1);
    N=Rot*[X(I);Y(I)];
    out(I,1)=N(1)+N(2)*1i;
end
end
```



Mode shape visualization

```
% Mode shape visualization
function phi = ModeVisual(params_matrix1, name)

figure;
x_domain = [5/100, 33/100, 50/100, 90/100];

% Calcolare la curva numerica
numPoints = 100;
x_numeric = linspace(0, 1, numPoints);
beta_n = [4.730, 7.853, 10.996, 14.137, 17.279, 20.420];
calculate_Yn = @(x, beta, L) cosh(beta * x / L) - cos(beta * x / L) ...
    - (cosh(beta) - cos(beta)) / (sinh(beta) - sin(beta)) * (sinh(beta * x /
L) - sin(beta * x / L));

% Preparazione della figura
for qq = 1:size(params_matrix1,2)
    subplot(2, 3, qq)

    modes=ComplexModeToRealMode(params_matrix1([1, 3, 4, 2] + 2, qq));

    plot(x_domain, modes, 'ko', 'MarkerFaceColor', 'none');
    hold on;
    Y_n = calculate_Yn(x_numeric, beta_n(qq), 1);
```

```
    Y_n = -Y_n / max(abs(Y_n)); % Normalizzazione e inversione

    sgn = sign(Y_n(5))*sign(modes(1));

    plot(x_numeric, Y_n*sgn, 'k--', 'LineWidth', 1);

    xlabel('$$x[\%]$$', 'Interpreter', 'latex')
    ylabel('$$\phi(x)$$', 'Interpreter', 'latex')
    title(['$$Mode\ at\ freq = ', num2str(params_matrix1(1, qq)), '$$'],
'Interpreter', 'latex')
    yline(0, 'k--', 'LineWidth', 1)
    grid on;
    axis tight;
end
sgtitle(sprintf('$$Mode\\ shapes\\ of\\ %s$$', name), 'Interpreter', 'latex');
% exportgraphics(gcf, ['Data', num2str(num), '_Modes.png'], 'Resolution', 600)
end
```

GUI to select the relevant parameters

```
function selectedColumns = selectColumnsWithCheckboxes(inputMatrix,
preSelectedColumns)
% Check if the input is a valid matrix
if nargin < 1 || ~ismatrix(inputMatrix)
    error('Please provide a valid input matrix.');
end

% Initialize the output variable
selectedColumns = [];

% Create the main figure window
fig = figure('Position', [100, 100, 600, 400], 'Name', 'Select Columns',
'NumberTitle', 'off', ...
    'CloseRequestFcn', @closeCallback);

% Display the matrix in a uitable
uitable('Parent', fig, 'Data', inputMatrix, 'Position', [20, 200, 560, 160]);

% Create a panel for checkboxes
checkboxPanel = uipanel('Parent', fig, 'Title', 'Select the columns of the
stable modes:', 'Position', [0.05 0.05 0.9 0.4]);

% Create checkboxes for each column
numCols = size(inputMatrix, 2);
maxColsPerRow = 10; % Maximum number of columns per row

checkboxHandles = gobjects(1, numCols);
for col = 1:numCols
    % Determine row and column position for the checkbox
    colIdx = mod(col-1, maxColsPerRow) + 1;
    rowIdx = floor((col-1) / maxColsPerRow) + 1;

    % X and Y positions
```

```
    xPos = 20 + (colIdx-1) * 50; % Adjust horizontal spacing as needed
    yPos = 100 - (rowIdx-1) * 40; % Adjust vertical spacing and starting Y
position as needed

    % Create the checkbox and pre-select if specified
    checkboxHandles(col) = uicontrol('Parent', checkboxPanel, 'Style',
'checkbox', ...
        'String', sprintf('%d', col), ...
        'Position', [xPos, yPos, 40, 30], ...
        'Value', ismember(col, preSelectedColumns)); % Pre-select if column
is in preSelectedColumns
end

% Create a button to confirm selection
uicontrol('Style', 'pushbutton', 'Position', [250, 20, 100, 30], 'String',
'Confirm', ...
    'Callback', @confirmSelection);

% Wait for the user to confirm the selection
uiwait(fig);

% Nested callback functions
    function confirmSelection(~, ~)
        % Get the selected columns
        selectedColumns = find(cell2mat(get(checkboxHandles, 'Value')));

        % Validate the selection
        if ~isempty(selectedColumns)
            % Resume the GUI to return the output and close the figure
            uiresume(fig);
            delete(fig);
        else
            % Show an error message if no columns are selected
            errordlg('Please select at least one column.', 'Selection Error');
        end
    end

    function closeCallback(~, ~)
        % Handle the case when the user closes the window without confirming
        selectedColumns = [];
        uiresume(fig);
        delete(fig);
    end
end
```

Published with MATLAB® R2023b