ESM NAH - Documentation

Badiane David Giuseppe, Lampis Alessio, Losi Andrea Eugenio

24 June 2021

1 Introduction

This projects adopts equivalent source method (ESM) based NAH to a violin back-plate. Since ESM based NAH avoids the discretization of the KH integral and can be applied to arbitrary geometries, it is particularly suited to our case study.

The basic idea of ESM is that the acoustic field of the vibrating object can be approximated to the field produced by a set of virtual point sources, called equivalent sources, located on a surface right behind the vibrating object. Finding the complex weights of the equivalent sources from the pressure measurements means solving an undetermined problem. ESM is a two steps method: first, we obtain the weights through regularization techniques; then, the weights are back-propagated to the reconstruction surface by the means of the proper driving functions (based on 3D Green's functions) to estimate the vibration velocity of the object.

Here there is the link of the repository containing the project.

2 Modules

Listing 1: `Main_synthetic.m`

```
%% Applies ESM on synthetic data.
```

In order we have the following agents:

- function **importData** - Extracts the violin model geometry, velocity fields and hologram fields and points for each eigenfrequency.
- Compute the **normal vectors** on the reconstruction surface, i.e. violin Mesh
- For each eigenfrequency, the user can choose to:
  - set retreat distance (RD) between virtual points and violin mesh and compute ESM estimation. Save their metrics;
  - optimize the estimation on the free parameters of the virtual points (RD - scaleX - scaleY), using the ground-truth data. Save the optimization results;
  - see the optimization results, save their metrics;

ESM is carried on by the Matlab function **applyESM.m**, which will be explained in depth later.

Listing 2: `Main_experimental.m`

```
%% Applies ESM on experimental data.
```

Apply ESM on experimental data. The structure is the same as *Main_synthetic.m* but we have to treat different types of data (experimental measurement).

The violin mesh is chosen from a set of grids located in the directory **violinMeshes**. The hologram is filled with our experimental data. The velocity groundtruth is obtained from 27 measured points on the violin.

Listing 3: `Preprocessing_pressure.m`

```
%% Obtain the hologram entries for main_Experimental.m
```

The hologram geometry is modeled on the experimental setup and filled with experimental data. We recorded 9 channels (8 measurement mics + 1 reference mic) for 8 different heights. For each height, we have 7 independent takes of the plate excitation. In order, for each channel, the script:

- Imports the audio and force signals;
- Isolates each take synchronizing the signal with the reference mic acquisition;
- Apply exponential filters;
- Compute the H1 estimator;
- Perform singular values decomposition (SVD) on the H1 estimator to reduce noise;
- Peak analysis and pressure fields retrieval;
- save results in .csv and .mat files;

Listing 4: `velocityGroundtruth.m`

```
%% Obtain velocity ground-truth for main_Experimental.m
```

The experimental velocity ground-truth is the H1 estimator of the mobility on 27 points. The data are obtained thourgh an hammer - accelerometer measurement system. In order, for each point, the script:

- read acceleration and force files;
- applies exponential filter;
- computes the mobility and the H1 estimator of the mobility;
- applies SVD on the H1 estimator to reduce noise;
- peak analysis and velocity fields retrieval;
- save results in .csv and .mat files;

Listing 5: `violinMeshGenerator.m`

```
%% Create and handle violin mesh
```

The violin meshes have been generated from a 4 million points .stl file obtained by laser scanning the violin back plate. The code is divided in three sections that are meant to be opportunistically used:

- **first section** - read the .stl mesh and downsample it to a 128x128 grid with the function **downsampling_regular.m**. Save it back to a .csv file;
- **second section** read a target grid and symmetrize it manually (user input and figures) with respect to the y axis of the violin;
- **third section** - downsample target grid and save it;

Listing 6: `virtualPointsGenerator.m`

```
%% create and handle virtual sources grid points
```

This script generates grids of virtual points choosing between different various geometry, sparsity levels and spatial sampling parameters. The meshes are generated by user input, more in particular its possible to decide between the following geometries:

- rectangular;
- ellipsoidal;
- circular + violin border;
- ellipsoidal + violin border;
- inner points + violin border;
- inner only
- rectangular + violin border + inner points;

For each chosen geometry, the user can decide the level of sparsity of the grid, adjust its size and choose the downsampling parameters. The functions used for the virtual points generation are in the folder **\functions\virtualPointsGenerators**.

3 Functions

Here the functions are described one by one, sorting them by their role.

3.1 ESM related functions

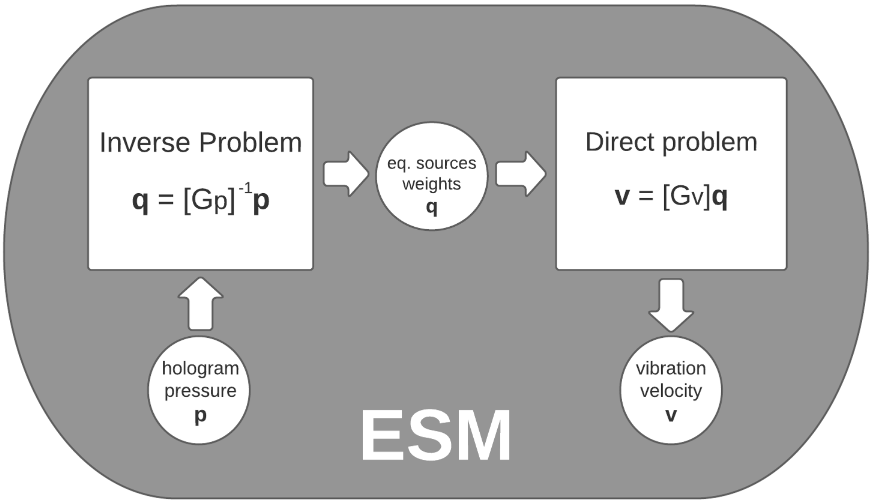

Figure 1: Block diagram of the method, G_p is the Green's functions matrix (hologram to virtual points), G_v is the normal gradient of the Green's functions matrix (virtual points to reconstruction surface)

Green's Matrices

As we see from Fig1, we need to compute the Green's functions matrix (hologram to virtual points) and the normal gradient of the Green's functions matrix (virtual points to reconstruction surface). The analytic expression of the 3D Green's function is the following:

$$g_w(\mathbf{r}, \mathbf{a}) = \frac{1}{4\pi} \frac{e^{-jk||\mathbf{r}-\mathbf{a}||}}{||\mathbf{r}-\mathbf{a}||}, \tag{1}$$

Listing 7: `Green_matrix.m`

```
function [G, deleteIndexesVirt] = Green_matrix(hologramPoints , ...
    virtualPoints, omega)
% ¬ computes the Green's matrix hologram 2 virtual points
% INPUTS
% hologramPoints = x,y,z coordinates of the measurement points ...
    (2DArray)
% virtualPoints  = x,y,z coordinates of virtual sources (2DArray)
% omega          = eigenfrequencies in radians per seconds (1DArray)
% OUTPUTS
% G = Green's matrices (cell array)  for each omega
% deleteIndexesVirt = nan points of the virtual sources grid
```

Listing 8: `normalGradient`

```
function [G_v] = normalGradient(virtualPoints , platePoints , ...
    omega, normalPoints)
% ¬ computes the normal gradient of the Green's matrix, obtained ...
    analytically by us
% double checked the result was correct with the symbolic library
% INPUTS
% virtualPoints = x,y,z coordinates of the virtual point (2DArray)
% platePoints  = x,y,z coordinates of surface points (2DArray)
% omega = eigenfrequencies in radians per seconds (1DArray)
% normalPoints = x,y,z components of the normal vector (2DArray)
% OUTPUTS
% G_v = normal gradient of Green's matrix for each omega (cell ...
    array)
```

Regularization

The library *Regtools* of Per Christian Hansen (2021), MATLAB Central File Exchange. The following functions are employed:

Listing 9: `l_curve.m`

```
function [reg_corner,rho,eta,reg_param] = l_curve(U,sm,b,method,L,V)
% ¬ L Curve
% Employed to find the Tikhonov regularization parameter lambda
% or the truncated SVD regularidation parameter k
```

Listing 10: `tikhonov.m`

```
function [x_lambda,rho,eta] = tikhonov(U,s,V,b,lambda,x_0)
% ¬ Tikhonov Regularization
% Employed to find virtual sources weights
```

Listing 11: `tsvd.m`

```
function [x_k,rho,eta] = tsvd(U,s,V,b,k)
%TSVD Truncated SVD regularization.
% Employed to find virtual sources weights
```

Listing 12: `csvd.m`

```
function [U,s,V] = csvd(A,tst)
% CSVD Compact singular value decomposition.
% applied on G_p to feed it to l_curve.m
```

For further readings we provide here the manual of the library.

perform ESM

The heart of ESM is carried on by the following function.

Listing 13: `reguResults.m`

```
function [Qs, v_TSVD, v_TIK] = reguResults( k, lambda, pressure, ...
    omega, rho, G_p, G_v, virtualPoints )
% reguResults - Carries on ESM calculations -
% (regularization inverse problem) + (solution direct problem)
%   INPUTS
%   k             (double)   = regularization parameter TSVD;
%   lambda        (double)   = regularization parameter TIK;
%   pressure      (1Darray)  = hologram vector;
%   omega         (double)   = radians frequency;
%   rho           (double)   = medium density;
%   G_p           (2Darray)  = Green matrix - pressure;
%   G_v           (2Darray)  = normal gradient Green matrix - ...
    velocity;
%   virtualPoints (2Darray)  = virtual points matrix (nPts x 3);
%   OUTPUTS
%   Qs            (struct) =  eq sources weight struct - members ...
    = qTIK  qTSVD;
%   v_TSVD        (1Darray)  = estimated velocity TSVD;
%   v_TIK         (1Darray)  = estimated velocity TIK;
```

reguResults.m is used in *errorVelocity.m*, which computes the metrics with respect to the velocity groundtruth.

Listing 14: `reguResults.m`

```
function [velocityErrors, ESM_Results] = ...
    errorVelocity(v_GT_vector, violinMesh, xData, yData, ...
    measuredPressure, G_p, G_v, lambda_L, k_L, omega, rho)
% ERRORVELOCITY - ESM + metrics,
% interpolates over the surface of the estimated velocity on the ...
    measured points( xDAta, yData),
% calculates the metrics [nmseTIK, nccTIK, normcTIK, reTIK]
% and saves it back into a struct
%   INPUT
%   v_GT_vector      (1DArray)  = vector of the velocity groundtruth;
%   violinMesh       (2DArray)  = mesh of the geometry;
%   xData            (2DArray)  = x matrix for the interpolation ...
    over the measured points;
%   yData            (2DArray)  = y matrix for the interpolation ...
    over the measured points;
%   measuredPressure (1DArray)    = hologram pressure for the ...
    given radians frequency
%   G_p              (2DArray)  = Green's matrix for the pressure;
%   G_v              (2DArray)  = Green's matrix for the velocity;
%   lambda_L         (double)   = Thikonov regularization parameter;
%   k_L              (double)   = TSVD regularization parameter;
%   omega            (double)   = radians frequency where ...
    evaluate the function;
%   rho              (double)   = medium density;
%   OUTPUT
%   velocityErrors   (struct) = struct where the metric values ...
    are stored;
%   ESM_Results      (struct) = struct containing ew sources ...
    weights and                      estimated velocities;
```

The metrics are Normalized Mean Square Error (NMSE), Normalized Cross-Correlation (NCC), normalCorrelation (NORMC), Reconstruction Error (RE).

The whole computation, including Green's Matrix, L-curve, regularization and metrics is executed by the function *applyESM*, which contains all the functions we have described up until now. The function sets the retreat distance (RD), i.e. the z distance between virtual points plane and lowest point of the reconstruction surface, and two scale factors (on x and y) for the virtual points. Moreover it computes a loss function based on the metrics, so that it is suitably optimizable.

Listing 15: `applyESM.m`

```
 [lossFx, ESM_metrics_table , ESM_Results] = applyESM( ...
     controlParams, boundZ, pressure, hologramPoints, ...
     normalPoints, violinMesh , omega, xData, yData , ...
     v_GT_vector, virtualPtsFilename, gridTablesNames, plotData, ...
     experimentalData)

% applyESM - this function applies the whole ESM method, saving ...
    metrics and results
% INPUTS
% controlParams  ([3x1] Array) = control Params of the virtual ...
    points grids
%                                to be driven for minimization
%                                [RD  scaleX   scaleY]
% boundZ              (double)      = lower bound of z for ...
    minimization
% pressure            (1DArray)     = pressure vector
% hologramPoints      (2DArray)     = hologram points coordinates
% normalPoints        (2DArray)     = normal points coordinates
% violinMesh          (2DArray)     = violin points coordinates
% omega               (double)      = radians frequency
% xData               (2DArray)     = X query measured points to ...
    interpolate
% yData               (2DArray)     = Y query measured points to ...
    interpolate
% v_GT_vector         (1DArray)     = velocity groundtruth vector
% virtualPtsFilename (string)   = string of the filename of ...
    virtual points
% gridTablesNames     (cell)     = names for the table
% plotData            (boolean) = to plot figures
% experimentalData

% OUTPUTS
% lossFx              (double) = loss function over which we ...
    minimize value
% ESM_metrics_table   (table)   = table containing the metrics
% ESM_Results         (struct) = struct containing the results : ...
    virtual points weights, estimated veloicties, velocities ...
    surfaces
```

- The functions ***reguFiguresSynthetic.m*** and ***reguFiguresExperimental.m*** plot the figures relative to the estimation.

3.2 Grid related functions

Virtual Points Generation

We wrote four functions that realize meshes of four different geometries.

Listing 16: `rectVirtPoints.m`

```
function [rectPts] = rectVirtPoints(pts, xRect, yRect, xCenter, ...
    yCenter, zVal)
%RECTVIRTPOINTS creates a rectangular grid
% INPUTS
% pts   [nPts x 3]  (2DArray)  = points of the mesh
% xRect             (double)   = edgeX value
% yRect             (double)   = edgeY value
% xCenter           (double)   = x coordinate of the center
% y Center          (double)   = y coordinate of the center
% zVal              (double)   = value of the z coordinate of the grid
% OUTPUTS
% rectPts [nPts x 3](2DArray) = rectangluar grid points
```

Listing 17: `ellipseVirtualPoints.m`

```
function [ellipse] = ellipseVirtualPoints(pts, maxRx, ...
    maxRy,minRx, minRy,zVal)
%ellipseVirtualPoints - creates an ellipsoidal grid
% INPUTS
% pts   [nPts x 3]  (2DArray)  = points of the mesh
% maxRx           (double)   = max radius along x of the ...
    ellipsoidal grid
% maxRy           (double)   = max radius along y of the ...
    ellipsoidal grid
% minRx           (double)   = min radius along x of the ...
    ellipsoidal grid
% minRy           (double)   = min radius along y of the ...
    ellipsoidal grid
% zVal            (double)   = value of the z coordinate of the grid
% OUTPUTS
% ellipse [nPts x 3](2DArray) = ellipsoidal grid of points
```

Listing 18: `borderVirtualPoints.m`

```
function [border] = borderVirtualPoints(pts, xBorder, yBorder, zVal)
% borderVirtualPoints this function creates a grid with the ...
    border of the
% geometry
% INPUTS
% pts   [nPts x 3]  (2DArray)  = points of the mesh
% xBorder           (double)   = how many points take from the ...
    border along X
% yBorder           (double)   = how many points take from the ...
    border along y
% zVal              (double)   = value of the z coordinate of the grid
% OUTPUTS
% border [nPts x 3](2DArray) = grid with the border of the geometry
```

Listing 19: `innerVirtualPoints.m`

```
function [intPts] = innerVirtualPoints(pts, xBorder, yBorder, zVal)
%innerVirtualPoints form a grid with the inner points of the ...
    geometry
% INPUTS
% pts  [nPts x 3] (2DArray) = points of the mesh
% xBorder          (double)  = how many points interior to the ...
    border along X
% yBorder          (double)  = how many points interior to the ...
    border along y
% zVal             (double)  = value of the z coordinate of the grid
% OUTPUTS
% intPts [nPts x 3](2DArray) = grid with the inner points of the ...
    geometry
```

Then a function that makes the given grid sparser:

Listing 20: `sparserVirtualPoints.m`

```
function [outPts] = sparserVirtualPoints(pts,controller, xCut, yCut)
%SPARSERVIRTUALPOINTS Summary of this function goes here
% INPUTS
% pts         (2DArray)= points of the mesh
% controller (double) = 0, modular sparsing
% controller          = 1, random sparsing
% controller          = 2, modular random sparsing - modular ...
    weighted randomically
% xCut        (double) = for modular sparsing along x, takes 1 ...
    point out of xCut
% yCut        (double) = for modular sparsing along x, takes 1 ...
    point out of xCut
% OUTPUTS
% outPs       (2DArray)= out matrix with sparser points
```

Those blocks are linked together by the function *genVirtualPoints.m*, which guides the user into the creation of the virtual sources grid.

Listing 21: `genVirtualPoints.m`

```
function genVirtualPoints(pts, fileName, controller, ...
    zVal,virtualPtsFolder, saveData)
% INPUTS
% pts  [nPts x 3]  (2DArray) = points of the mesh
% fileName           (string) = name of the .csv file containing ...
    the points
% controller         (double) = 0, gen rectangular grids
% controller = 1, gen circular grids
% controller = 2, gen ellipsoidal grids
% controller = 3, gen circular + border
% controller = 4, gen ellipsoidal + border
% controller = 5, gen inner + border
% controller = 6, gen border only
% controller = 7, gen inner only
% controller = 8, gen rect + border + inner
% zVal              (double) = value of the z coordinate of the grid
% virtualPtsFolder (string) = filepath to the folder containing ...
    the virtual points
% saveData         (boolean) = if true creates the csv file of ...
    name fileName
% OUTPUT
% ¬
```

Other grid functions

Another set of grid related function are widely used into the code. *interpGrid.m* is used to compute the metrics. It interpolates a given discrete surface over a set of query points.

Listing 22: `interpGrid.m`

```
% INTERPGRID this function interpolates in the surface defined ...
    by meshPoints
%           over the points of coordinate xData and yData
%   INPUTS
%   meshPoints  (2Darray)  = points of the grid on which we ...
    interpolate
%   xData       (2Darray)  = X matrix - target points for the ...
    interpolation;
%   yData       (2Darray)  = Y matrix - target points for the ...
    interpolation;
%   pX          (double)   = number x coordinates of the mesh;
%   pY          (double)   = number y coordinates of the mesh;
%   plotData    (boolean)  = choose if the plot have to be shown;
%   OUPUTS
%   zCordInter  (2Darray)  = Z matrix of the interpolated ...
    coordinates;
```

The function *downsampling_regular.m* was conceived to downsample the not equispaced four million vertices mesh that we acquired from the violin. It individuates the point with the minimum distance from the target point of a rectangular grid and substitutes the corresponding z value.

Listing 23: `downsampling_regular.m`

```
function [outMatrix] = downsampling_regular(inputMatrix, nrows, ...
    ncols, fileName, saveData)
% this fucntion performs donwnsapling over a rectangular grid on ...
    a matrix
% INPUTS
% inputMatrix = matrix to downsample     (2DArray)
% nrows     = target number of rows      (double)
% ncols     = target number of columns   (double)
% fileName = fileName for saving [only the name, not .csv] (double)
% saveData = true if you want to save data on .csv file (boolean)
% OUTPUTS
% outMatrix = resampled matrix or array (2DArray)
```

The function *downsampling.m* is conceived to downsample a small grid of points. It uses the interpolating function of matlab *interp2*.

Listing 24: `downsampling.m`

```
function [outMatrix] = downsampling(inputMatrix, nrows, ncols)
% DOWNSAMPLING this function performs donwnsapling on a matrix ...
    or array
% INPUTS
% inputMatrix = matrix to downsample (2DArray)
% nrows = target number of rows       (double)
% ncols = target number of columns    (double)
% OUTPUTS
% outMatrix = downsampled matrix      (2DArray)
```

The function *addNans.m* is used for the computation of the metrics in *errorVelocity.m*. It assigns an input vector to the vector masked with nans of the z coordinate of a grid.

Listing 25: `addNans.m`

```
function [nanVel] = addNans(points, velocity)
%ADDNANS this function creates an array of size ...
    [length(points(:,3),1]
% where the output array has the same nan indexes of points(:,3)
% and its not nan indexes are filled by the value of velocity
%    INPUTS
%    points        (2Darray) = matrix of points whose nans mask ...
    is taken;
%    velocity      (1Darray) = signal on which the mask is applied;
%    OUTPUT
%    nanVel        (1Darray) = output signal with NaNs inserted;
```

The function *getVelocityGroundtruth* shows the interpolated surface resulting from our measured points

Listing 26: `getVelocityGroundtruth.m`

```
function [X,Y,surfV] = getVelocityGroundtruth(v_ex_vector, ...
    velocityFilename, figureNum)
% GETVELOCITYGROUNDTRUTH this function converts the scattered ...
    points of the
% velocity vector into a surface through the fx. ...
    ScatteredInterpolant
%    INPUTS
%    v_ex_vector      (1Darray) = vector of the velocities groung ...
    truth ;
%    velocityFilename (string)  = name of the velocityData.csv ...
    file to see;
%    figureNum        (double)  = number of the figure to plot;
%    OUTPUTS
%    X                (2Darray)   = x matrix of the mesh;
%    Y                (2Darray)   = y matrix of the mesh;
%    surfV            (2Darray)   = z matrix of the velocity;
```

3.3 Preprocessing related functions

FRF Analysis - signal processing

FFT.m computes the single sided spectrum of the signal.

Listing 27: FFT.m

```
function frequencySignal = FFT(timeSignal)
% FFT computes the single sided spectrum of the time signal
%   INPUT
%   timeSignal       (1DArray) = input time signal to transform
%   OUTPUT
%   frequencySignal  (1DArray) = signal transformed
```

The function *peaks.m* finds the resonances of the body by analysing the cumulative sum of the frequency response functions (FRFs). Two independent peak finders are employed, one for low frequencies and another for high frequencies. Used to find the peaks both for pressure and acceleration measurements.

Listing 28: peaks.m

```
function [peaksLoc, fpeakPositions] = peaks(matrix, f, ...
    fThreshold, ignorePeaksLow, ignorePeaksHigh, ...
    highPeaksParams, lowPeaksParams)
%PEAKS finds peaks of the FRF by analysing their cumulative sum
%   INPUTS
%   matrix            (2Darray) = matrix of the H1 estimator;
%   f                 (1Darray)  = frequency axis;
%   fThreshold        (double)  = frequency threshold for peak ...
    fining (we use two find peaks);
%   ignorePeaksLow    (double)   = low threshold - ignore peaks at ...
    lower frequency than it;
%   ignorePeaksHigh   (double)   = high threshold - ignore peaks ...
    at higher frequency than it;
%   highPeaksParams   (1Darray) = [2x1] = minPeakProminence, ...
    minPeak width for highFreq findpeaks;
%   lowPeaksParams    (1Darray) = [2x1] = minPeakProminence, ...
    minPeak width for lowFreq findpeaks;
%   OUTPUTS
%   peaksLoc          (array)    = peaks location values;
%   fpeakPositions    (array)    = peaks location indices;
```

The function *findSubBands.m* individuates the bands of the peaks of the FRF. The subBands are first coarsely estimated and then the estimation is refined by evaluating the gradient of the FRF.

Listing 29: `finSubBands.m`

```
function [freqIndexes, coarseIndexes] = findSubBands(Hv, fAxis, ...
    fAmps, fLocs, ΔfLocs, plotData)
%findSubBands find the bands of the peaks of an FRF
%   INPUTS
%   Hv          (1DArray) = FRF to analyse;
%   fAxis       (1DArray) = frequency axis of the FRF;
%   fAmps       (1DArray) = amplitudes of the peaks;
%   fLocs       (1DArray) = indexes of the peaks;
%   ΔfLocs   (1DArray)  = (fLocs - circshift(fLocs,1)) to coarsely get
%                           the subbands boundaries
%   plotData    (boolean) = see images
%   OUTPUTS
%   freqIndexes   (1DArray) = frequency indexes of the bands ...
    after gradient
%       computation - the limit bwLeft is independent of bwRight ...
    - subbands
%       may be asymmetric;
%   coarseIndexes (1DArray) = coarse indexes obtained by ...
    evaluating ΔfLocs -
%                             subbands are symmetric;
```

The function *EMASimple* performs modal analysis on the FRF. The function employs a peak finder, uses *findSubBands.m* and interpolates a parabola for each peak. The parabola is then utilized to compute the adimensional damping ratios with the half power point formula

$$\xi = \frac{\omega_2^2 - \omega_1^2}{4\omega_0^2}$$

where ω_1 and ω_2 are the half power frequencies of the band, i.e. frequencies at which the magnitude of the FRF is 0.707 times the peak value.

Listing 30: `EMASimple.m`

```
function [Hv,f0, fLocs, csis, Q, modeShapes] = EMASimple(HvSVD, ...
    fAxis,minPeakVal, minPeakWidth, plotData)
%EMASIMPLE Simplified (not using minimization) modal analysis ...
    algorithm
%   INPUTS
%   HvSVD         (1DArray)    = spectrum to analyse;
%   fAxis         (1DArray)    = frequency axis of the spectrum;
%   minPeakVal    (double)   = minimum value of the peaks for peak ...
    analysis;
%   minPeakWidth  (double)   = minimum value of the width of the ...
    maximum;
%   OUTPUTS
%   Hv            (1DArray)    = cutted H1 estimator;
%   f0            (1DArray)    = frequency locations of the peaks;
%   fLocs         (1DArray)    = index locations of the peaks;
%   csis          (1DArray)    = adimensional damping ratios;
%   Q             (1DArray)    = quality factors;
%   modeShapes    (1DArray)    = modeshapes value in the point;
```

EMASimple is used in both pressures and accelerations pre-processing to assign the value at the peak for each FRF. Infact the single FRF peaks may be different with respect to the one found by *peaks.m*, and this may introduce unwanted errors. So, we compute again the peaks for each FRF, if they are reasonably close to the average peaks, then they qualify. If they aren't, we assign the magnitude value at the average peak.

In order to de-noise the H1 estimators (i.e. FRFs), we used the function*SVD.m*, which performs the singular values decomposition and reconstruction of the signal.

Listing 31: SVD.m

```
function [HvSVD, singularVals] = SVD(frf, freq, M, ...
    nUsedSingVals, plotData)
% SVD - Singular Values Decomposition - reduces noise
% computes SVD of an FRF thourgh the hankel matrix, uses ...
    nUsedSingVals to
% reconstruct
%   INPUTS
%   frf           (1Darray)     = FRF on which perform SVD;
%   freq          (1Darray)     = frequency axis relative to the FRF;
%   M             (int)         = max number of singular values ...
    computed;
%   nUsedSingVals (double)      = number of singular values used ...
    to reconstruct
%   plotData      (boolean      = true if you want to generate images;
%   OUTPUTS
%   HvSVD         (2Darray)     = H1 estimator after SVD;
%   singularVals  (array)       = sigualar values;
```

3.4 Others

The function *whiteNoise.m* adds white noise to a signal. It's used for adding noise to the synthetic pressure data.

Listing 32: whiteNoise.m

```
function [out] = whiteNoise(in, SNR)
% WHITENOISE this function adds white gaussian noise to the input
%   INPUTS
%   in   (1Darray)    = input array;
%   SNR  (double)   = signal to noise ratio;
%   OUTPUT
%   out  (1Darray)    = output array;
```

We use the function *writeMat2File.m* to save files.

Listing 33: `writeMat2File.m`

```
function [dataTable] = writeMat2File(data,dstFileName, name, ...
    numVars, singleTitles)
% WRITEMAT2FILE this function saves data into a .csv or .txt file
%   INPUTS
%   data         (2Darray) = data to write ;
%   dstFileName  (string) = filename, can be .txt or .csv;
%   name         (cell array) = containing the variables names;
%   numVars      (2Darray) = num variable names -- length(names);
%   singleTitles (boolean) = if name contains all the names of ...
    the file - true
%                            if you want to numerate name for ...
    all the cols
%                            of the file, false
%   ex1 - name = {'x' 'y'}, numVars = 2, singleTitles = false    ...
    --> variableNames = {'x1' 'y1' ... 'xnCols' 'ynCols'}
%   ex2 - name = {'x' 'y'}, numVars = 2, singleTitles = true --> ...
    variableNames = {'x' 'y'}
%   OUTPUT
%   dataTable    (table) = written table in the file ;
```

4 Data

The file-system of the project is simple and straightforward.

- **CSV** - contains synthetic data, stored in .csv files;
- **Data** - contains experimental data, stored in .csv and .mat files;
- **Estimations** - contains the estimations metrics and parameters for both synthetic and experimental case scenario;
- **violinMeshes** - contains the violin meshes .csv files;
- **VPGrids** - contains the virtual points .csv files;
- **Exp_Measurements** - to be downloaded from the following link - contains the data of the hologram measurement campaign, i.e. audio files of the microphones and the force of the hammer, and the data of the accelerometric measurements.

5 Further Comments

The function *applyESM.m* is suited to be applied over different grids in a cycle in order to decide the best grid.

6 Acknowledgements

We thank our supervisors Raffaele Malvermi and Marco Olivieri and our professors Alberto Bernardini and Augusto Sarti for their support and encouragement.