MuTE Settings, GUI

. . . and several examples [1]

I am going to explain how to set MuTE in order to perform several analyses. I will provide screenshoots and a description of the code to modify as detailed as possible.

1 Toolbox Structure

MuTE has a very simple structure as we can notice from 1. The Main function allows the user to define the preliminary settings and to call the parametersAndMethods function. parametersAndMethods simply wraps the parameters defined in Main, calls the callingMethods function and calls the post precessing functions responsible to store the most informative quantities. In 1 the schematic representation of the main toolbox's parts and their relationship are shown.

In the following, the toolbox structure is illustrated in detail, step by step, in order to allow every user to use MuTE. The choice to show the **function names in bold** is aimed to let the user keep track of the most important functions.

NB:

- From now on the user is kindly asked to look at MuTE/exampleToolbox/exampleMain.m function
- Parts of code will be showed either in blue boxes according to what is the current set up or in green providing some examples of possible set up
- The % symbol is used for commented lines

2 Prerequisites

1. **Data** should be stored as files containing the field data. data should be a matrix of (number of series × time points) dimensions. If the user wants to perform an analysis across many trials it is possible to store data as 3D matrices of (number of variables × number of points

[1] Alessandro Montalto; E-mail: alessandro.montalto@ugent.be, Web-site: `http://mutetoolbox.guru/mutesettings/`

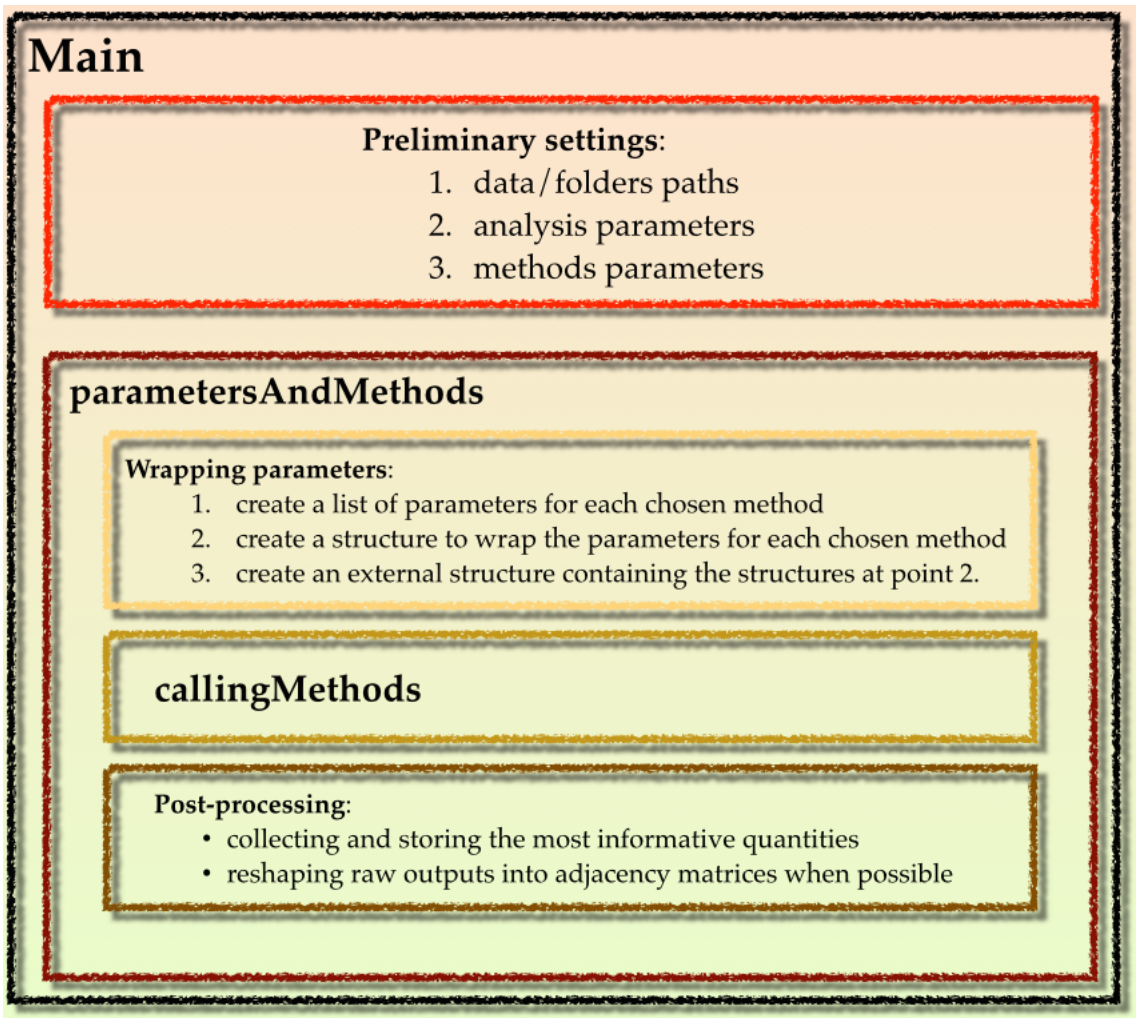

Figure 1: Toolbox Structure.

× number of trials) dimensions. It is possible to find an example of data files in MuTE/exampleToolbox/.

2. **Create a folder** in which the data are stored. Let us call this folder as dataFolder from now on.

3 Main

The Main script is the only one that should be modified by the user. The analyses can be set outside the specific functions. The examples will show how Main can be customized according to the user needs. The Main gives the opportunity to set the methods parameters, where to read the data, where to store the results. It is also possible to concatenate experiments in order to run Main only once. As soon as the set up is made, MuTE is ready to run and the user should not give any further contribution until the end of the experiments.

4 Preliminary Settings

4.1 Some Useful Comments

At the very beginning of exampleMain there are what I hope are useful comments. First of all we can find the method order.

1: line 5: % Method order: please take the order into account because afterwards
2: line 6: % you should set autoPairwiseTarDriv or handPairwiseTarDriv that need
3: line 7: % the precise order of the methods
4: line 8:
5: line 9: binue
6: line 10: binnue
7: line 11: linue
8: line 12: linnue
9: line 13: nnue
10: line 14: nnnue
11: line 15: neunetue
12: line 16: neunetnue

Afterwards the parameters for each method are listed. I explain them in detail in parametersAndMethods.

4.2 Folders Set Up

The folder set up is the first step to take into account. It is necessary to add MuTE to MATLAB path as follows:

1: line 185: % Set MuTE folder path including also all the subfolders, for instance
2: line 186: mutePath = ('/Users/alessandromontalto/Dropbox/MuTE/');
3: line 187: cd(mutePath);
4: line 188: addpath(genpath(pwd));

Change according to where you decide to store MuTE

1: % Set MuTE folder path including also all the subfolders, for instance
2: mutePath = '/Users/.../Desktop/MuTE/';
3: % Adjust according to your path -> just an example: mutePath = '/home/alessandro/Desktop/MuTE/';
4: cd(mutePath);
5: addpath(genpath(pwd));

Then, the user should set up the folder where the data are stored.

1: line 190: nameDataDir = 'exampleToolbox/'
2: % Set the directory in which the data files are stored. In this directory the outcome of the experiments will be stored too.
3: dataDir = ['/Users/alessandromontalto/Dropbox/MuTE/' nameDataDir];

Change according to where your data are stored

1: nameDataDir = 'folder containing your data/'
2: % Set the directory in which the data files are stored. In this directory the outcome of the experiments will be stored too.
3: dataDir = ['/Users/.../folder containing your data/'];
4: % Adjust according to your path -> just as example: dataDir = ['/home/alessandro/Dropbox/ nameDaraDir];
5: cd(mutePath);
6: addpath(genpath(pwd));

4.3 Number of Processors

Set the number of processors you can use to run your experiments changing value to the following variable

1: line 197: numProcessors = 1;

4.4 File's Variable Set Up

At this point two parameters should be set to load the correct data files:

- **dataFileName** takes the part of the name file common to all the files involved in the analysis.
- **dataLabel** is useful to distinguish files. If no label is required set dataLabel as an empty string.
- **dataExtension** defines the file extension

Variable needed to load the data

1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'realization5000p';
3: line 209: dataLabel = '';
4: line 210: dataExtension = '.mat';

EXAMPLE 1

I have 100 trials named, for instance, realizations5000p(...).mat where (...) can be a counter ranging from 1 to 100. In this case I do not to set dataLabel.

1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'realization5000p';
3: line 209: dataLabel = '';
4: line 210: dataExtension = '.mat';

EXAMPLE 2

I have 100 trials named, for instance, henonMaps_circular(...).mat and other 100 trials named henonMaps_sparse(...).mat where (...) can be a counter ranging from 1 to 100. In this case I would set the parameters as follows to perform the analysis taking into account henonMaps_sparse(...).mat.

1: line 207: % Defining the strings to load the data files
2: line 208: dataFileName = 'henonMaps';
3: line 209: dataLabel = 'sparse';
4: line 210: dataExtension = '.mat';

4.5 General Parameters Set Up

Here I am going to describe the more general parameters useful to handle the methods:

- **channels** is useful to select a subset of variables for a certain analysis. It is highly recommended to enter the series id from sorted in ascending order from left to right.

- **samplingRate** should be greater than 1 if data should be downsampled
- **pointsToDiscard** allows to discard a certain amount of points, starting from the last one. To better clarify **samplingRate** and **pointsToDiscard**, the meaningful part of line 315 in parametersAndMethods function is shown: data(channels,1:samplingRate:(end-pointsToDiscard))
- **realization** may be left unchanged because it stores the files defined in dataFileName, dataLabel and dataExtension. It is worth taking a look at line 244 in exampleMain to get how realization is set: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
- **autoPairwiseTarDriv** takes into account the opportunity to investigate all the pair wise combinations of the variables chosen by means of channels. autoPairwiseTarDriv is then a vector with either 0 or 1 entries. The methods order shown at the beginning of this section has to be preserved.
- **handPairwiseTarDriv** is useful to when the user already know how many targets is going to choose for the analysis. If the the number of targets can be reshaped in a square matrix then it is worth setting as 1 the entriy of handPairwiseTarDriv corresponding to the method choosen. handPairwiseTarDriv is then a vector with either 0 or 1 entries. The methods order shown at the beginning of this section has to be preserved.

Variables useful to handle data

1: line 240: % Defining the string to get data files
2: line 241: channels = 1:5;
3: line 242: samplingRate = 1;
4: line 243: pointsToDiscard = 4500;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 1 0 1 0 1];
7: line 246: handPairwiseTarDriv = [0 0 0 0 0 0];

EXAMPLE 1

Let's say that we want to perform an analysis on henon*.mat files in MuTE/exampleToolbox/. Those files contain a matrix called data with 6 time series of 2500 points. For instance, we want to analyse all the pair wise combinations taking into account only 3 variable out of the 6 available: variables 2, 5, 6. We also want to consider the first 2000 points out of

the 2500 available. Finally, we only want to investigate binnue and nnnue performances. In this case we should set the parameters as follows:

1: line 240: % Defining the experiment parameters
2: line 241: channels = [2 5 6];
3: line 242: samplingRate = 1;
4: line 243: pointsToDiscard = 500;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 1 0 0 0 1];
7: line 246: handPairwiseTarDriv = [0 0 0 0 0 0];

EXAMPLE 2

Taking into account the same data as "Example 1", we do not want to analyse all the pairwise combinations, but we already know that we are going to choose a number of targets that can be reshaped in a square and that will involve all the variables. We want to downsample the time series at the half of the sampling rate and we want to take into account all the time points. This time we want to investigate binue, linue and linnue performances.

1: line 240: % Defining the experiment parameters
2: line 241: channels = [1:6];
3: line 242: samplingRate = 2;
4: line 243: pointsToDiscard = 0;
5: line 244: realization = dir([dataDir [dataFileName '*' dataLabel '*' dataExtension]]);
6: line 245: autoPairwiseTarDriv = [0 0 0 0 0 0];
7: line 246: handPairwiseTarDriv = [1 0 1 1 0 0];

5 parametersAndMethods

parametersAndMethods is the only funcion in the main script. It takes in input the parameters seen so far and the list of the methods with their own parameters. Inside parametersAndMethods the methods parameters are rearranged in only one structure used by other functions handle the proper parameters and perform the correct analyses.

It is worth seeing how parametersAndMethods is called in the exampleMain.

In the following the screenshot of the call to parametersAndMethods.

It is useful, then, to take a look in detail at the parameters that the user should set for each method. The parameters can be devided into two big groups: the common parameters that have to be set up for each of the 6 methods and the particular parameters that are useful to a particulr method

```
%% STATISTICAL METHODS

fprintf('\n*******************************\n\n');
disp('Computing statistical methods...');
fprintf('\n\n');

tic
[output1,params1]               = parametersAndMethods(listRealization,samplingRate,pointsToDiscard,channels,autoPairwiseTarDriv,...
                                  handPairwiseTarDriv,resultDir,dataDir,copyDir,numProcessors,...
                                  'linue',[],[],[],5,'multiv',5,5,'bayesian',@linearEntropy,[1 0],[1 1],@generateConditionalTerm,0,...
                                  'linnue',[],[],[],5,'multiv',@evaluateLinearNonUniformEntropy,[1 1],100,0.05,@generateConditionalTerm,0,...
                                  'binue',[],[],[],5,'multiv',6,@conditionalEntropy,@quantization,[1 0],[1 1],100,0.05,20,...
                                  @generateConditionalTerm,0,...
                                  'binnue',[],[],[],5,'multiv',6,@evaluateNonUniformEntropy,@quantization,[1 1],100,0.05,@generateConditionalTerm,0,0,...
                                  'neunetue',[],[],[],5,[1 1],'multiv',[],[],{@sigmoid @identity},30,0,4000,2/3,15,...
                                  valThreshold,@resilientBackPropagation,1.1,0.9,1,numHiddenNodes,100,20,0.05,@generateConditionalTerm,0,...
                                  'neunetnue',[],[],[],[],5,[1 0],[1 1],'multiv',[],[],{@sigmoid @identity},30,0,4000,threshold,2/3,15,...
                                  valThreshold,@resilientBackPropagation,1.1,0.9,1,numHiddenNodes,@generateConditionalTerm,1,...
                                  'nnue',[],[],[],5,'multiv',[1 1],100,'maximum',10,nnMexa64Path,mutePath,0.05,10,@generateConditionalTerm,0,...
                                  'nnnue',[],[],[],5,'multiv',[1 1],100,'maximum',10,@nearNeiConditionalMutualInformation,...
                                  @evalNearNeiTestSurrogates2rand,nnMexa64Path,mutePath,0.05,@generateConditionalTerm,0);
```

only. I will provide some examples in order to make the meaning of the parameters as clear as possible.

5.1 Common Parameters

1. **idTargets** is a row vector with entries the series id chosen as targets of the analysis. It may contain repeated indeces.

2. **idDrivers** is a matrix with columns the series id whose influence on the corresponding target column wise in idTarget is going to be evaluated.

3. **idOtherLagZero** is a matrix with columns the series id chosen among the conditioning variables for that analysis that should be taken into account with the instantaneous effect.

4. **modelOrder** can be a vector indicating how many past states should be taken into account for each series chosen for the analisys by means of channels. If modelOrder is an integer the same amount of past states is considered for all the series in channels.

5. **multi_bivAnalysis** is a string containing either "biv" or "multiv" according to whether perform a bivariate or multivariate analysis respectively.

6. **genCondTermFun** is a pointer to function assuming either "@generateCondTermLagZero" or "@generateConditionalTerm" values. The first function, generateCondTermLagZero, takes into account the instantaneous effect for the drivers and for the variables indicated in idOtherLagZero. The second function, generateConditionalTerm, will not take into account the instantaneous effect, even if are explicitly mentioned in idOtherLagZero. This parameters should be set in accordance with usePresent.

7. **usePresent** is an integer assuming 1 or 0 values according to whether genCondTermFun is set to "@generateCondTermLagZero" or "@generateConditionalTerm" respectively.

EXAMPLE 1

Let's assume that we have a matrix of (10×3000) dimensions as data. We are interested in evaluating the adjacency matrix of the 10 variables: this means that we want to investigate all the pair wise combinations of the 10 time series taking into account all the time points. The implication is that we set idTargets and idDrivers as empy matrices.

Important: if we set an entry of autoPairWiseTarDriv as 1 idTargets and idDrivers of the correponding method will not be taken into account because autoPairWiseTarDriv has priority over idTargets and idDrivers.

Furthermore, we want a conditioned analysis taking 5 past states for all the variables and we do not want to consider the instananeous effects. We select binnue, linnue and nnnue methods. The parameters should be set as follows:

- channels = [1:10];
- samplingRate = 1;
- pointsToDiscard = 0;

...

- autoPairwiseTarDriv = [0 1 0 1 0 1];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = []
- idDrivers = []
- idOtherLagZero = []
- modelOrder = 5
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateConditionalTerm
- usePresent = 0

EXAMPLE 2

Let's assume that we have a matrix of (10×3000) dimensions as data. We are interested, for instance, in evaluating the adjacency matrix of 4 variables out of 10.

Important: assume that we are choosing series 2, 5, 8, 9. the time series chosen for the analysis will be extracted from the original matrix. In this way the user should keep into account that idTargets and idDrivers will assume values from 1 to channels length. In this case they will assume values ranging from 1 to 4.

Furthermore, let's assume that we do not want to investigate all the pair wise combinations. Instead, we only want to check whether there are information flows between certain driver-target couples and we need to have an idea about the directed dynamical links as soon as possible. Then we can use 500 points only. We also want a conditioned analysis taking 8 past states for all the variables and we want to consider the instantaneous effects of certain conditioning variables. We select binnue, linnue and nnnue methods. The parameters should be set as follows:

- channels = [2 5 8 9];
- samplingRate = 1;
- pointsToDiscard = 2500;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 1 0 1 0 1];
- idTargets = [1 1 2 2 3 3 4 4 4]
- idDrivers = [2 4 1 3 1 4 1 2 3]
- idOtherLagZero = [3 0 0 0 2 0 0 1 2; 4 0 0 0 0 0 0 0 1]
- modelOrder = 8
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

As we can notice, I chose 9 targets so I could set a 1 in handPairwiseTarDriv vector corresponding to the right method. To better explain the set up I can say that we are going to take into account the instantaneous effects of the variables 3 and 4 (corresponding to the 5th and the 8th time series of the original matrix) when investigating how 2 is influencing 1, conditioned to the other two variables. **At this point the instantaneous effects for the drivers are not taken into account**.

EXAMPLE 3

In this example we still want to deal with a matrix of (10×3000) dimensions as data. This time, we want a bivariate analysis including the instantaneous effects for the drivers. The procedure is slightly different: the id of the driver should be repeated. The explanation of the difference between the set up of the instantaneous effects for the drivers and for the conditioning variables is that the conditioning variables are taken into account setting multi_bivAnalysis = 'multiv' so it is easy to arrange the specific instantaneous effects in another vector. We only want to use the first 5 time series without discarding any point. We want to use a vector to set a specific model order for each time series. Binue, linue and linnue will be the methods involved in the analysis.

- channels = [1:5];
- samplingRate = 1;
- pointsToDiscard = 0;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = [1 1 2 3 3 4 5]
- idDrivers = [2 4 3 1 4 2 3; 5 0 0 2 1 5 0]
- idOtherLagZero = []
- modelOrder = [4;3;5;3;3]
- multi_bivAnalysis = 'biv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

There are 7 targets so it is better to have handPairwiseTarDriv = [0 0 0 0 0 0] otherwise there will be an error when the results will be reshaped in a square matrix. idOtherLagZero will never be taken into account when multi_bivAnalysis = 'biv'.

EXAMPLE 4

Another example trying to combine the configurations that we have previously seen in Examples 1-3. I am assuming that I have a matrix of (10×3000) dimensions as data. I will first show the set up and then I will provide some comments about it.

- channels = [1 3 4 7 8];
- samplingRate = 3;
- pointsToDiscard = 100;

...

- autoPairwiseTarDriv = [0 0 0 0 0 0];
- handPairwiseTarDriv = [0 0 0 0 0 0];
- idTargets = [1 2 3 4 5]
- idDrivers = [2 3 4 1 2; 3 0 5 0 4; 4 0 2 0 0]
- idOtherLagZero = []
- modelOrder = 8
- multi_bivAnalysis = 'multiv'
- genCondTermFun = @generateCondTermLagZero
- usePresent = 1

I am performing an analysis taking into account series 1, 3, 4, 7 and 8 downsampled at one third of the original sampling rate. I am discarding the last 100 time points. I am investigating how:

1. variables 2, 3 and 4 are influencing 1 conditioned to 5. This is equivalent to say that I am interested in detecting the information flow from variables 3, 4 and 7 towards variable 1 conditioned to variable 8 of the original matrix. The same reasonining can be applied in the following;
2. variable 3 is influencing 2 conditioned to variables 1, 4 and 5;

3. variables 4, 5 and 2 are influencing 3 conditioned to 1;
4. variable 1 is influencing 4 conditioned to the variables 2, 3 and 5;
5. variables 2 and 4 are influencing 5 conditioned to 1 and 3.

5.2 Particular Parameters LIN UE

We embedded "arfit" in linue in order to allow the user to look for the best model order. For further references the user is kindly asked to read arfit documentation.

- **minOrder** → integer: lower bound of the interval in which arfit can look for the best model order able to fit the data
- **maxOrder** → integer: upper bound of the interval in which arfit can look for the best model order able to fit the data
- **orderCriterion** → string: order selection criterion for arfit. If orderCriterion is not set to 'bayesian', the Akaike's Final Prediction Error will be chosen as selection criterion
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account

5.3 Particular Parameters LIN NUE

- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **alphaPercentile** → integer: significance level

5.4 Particular Parameters BIN UE

- **numQuantLevels** → integer: number of quantum levels
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **preProcessingFun** → pointer: pointer to the function needed to preprocess of the data
- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **alphaPercentile** → integer: significance level
- **tauMin** → integer: number of shifts to produce a surrogate

5.5 Particular Parameters BIN NUE

- **numQuantLevels** → integer: number of quantum levels
- **entropyFun** → pointer: pointer to the function that will evaluate the entropy
- **preProcessingFun** → pointer: pointer to the function needed to preprocess of the data
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **alphaPercentile** → integer: significance level

5.6 Particular Parameters NN UE

- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **metric** → string: it is possible to set the metric (either 'euclidian' or 'maximum') used to evaluate the distance in the phase space
- **numNearNei** → integer: number of nearest neighbors to compute
- **funcDir** → string: mex files path
- **homeDir** → string: MuTE path
- **alphaPercentile** → integer: significance level
- **tauMin** → integer: number of shifts to produce a surrogate

5.7 Particular Parameters NN NUE

- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **metric** → string: it is possible to set the metric (either 'euclidian' or 'maximum') used to evaluate the distance in the phase space
- **numNearNei** → integer: number of nearest neighbors to compute
- **informationTransCriterionFun** → pointer: pointer to the function needed to evaluate the conditional mutual information;
- **surrogatesTestFun** → pointer: pointer to the function that performs the surrogates test;
- **funcDir** → string: mex files path
- **homeDir** → string: MuTE path
- **alphaPercentile** → integer: significance level

5.8 Particular Parameters NeuNet UE

- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **eta** [2] → real number: learning rate to update the weights with gradient descent and gradient descent with momentum
- **alpha** [3] → real number: parameter required in gradient descent with momentum [4]
- **actFunc** → cell array of pointers: it contains pointers to functions that are used as activation functions. There must be (number hidden layers + output layer) number of entries specifying the activation function for each layer
- **numEpochs** → integer: number of training epochs
- **bias** → integer: it allows to take into account the bias if it is set as 1. If bias nodes are not required, "bias" has to be set as 0
- **candidateEpochs** → integer: number of maximum iterations needed to train the network with the current candidate. It is used in the outer "while" of the non-uniform wrapper
- **dividingPoint** → real number: amount of points used to train the networks. It is expressed as percentage of the data set number of points
- **valStep** → integer: number of iterations after which the validation phase takes place
- **valThreshold** → real number: threshold needed during the validation phase
- **learnAlg** → pointer: points to the function used as learning algorithm
- **rbpIncrease** → real number: η^-, [**?**]
- **rbpDescrease** → real number: η^+, [**?**]

[2] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[3] This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[4] For further references – [**?**]

- **rangeW** → real number: it represents the range of values assumed by the weights when initialized. If rangeW is set as 1, the weights will be initialized between -1 and 1
- **coeffHidNodes** → real number: percentage of hidden nodes with respect to the amount of available candidates
- **numSurrogates** → integer: number of surrogates necessary to asses the statistical threshold
- **tauMin** → integer: number of shifts to produce a surrogate
- **alphaPercentile** → integer: significance level

5.9 Particular Parameters NeuNet NUE

- **data** → matrix: data arranged differently with respect to data used by the other methods. It might be useful to arrange data in order to line up the realizations that the other methods would analyze separately
- **firstTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 0]). Set 1 according to wich candidates you want to take into account
- **secondTermCaseVect** → vector: the first position refers to idTargets, the second one refers to idDrivers ([1 1]). Set 1 according to wich candidates you want to take into account
- **eta** [5] → real number: learning rate to update the weights with gradient descent and gradient descent with momentum
- **alpha** [6] → real number: parameter required in gradient descent with momentum [7]
- **actFunc** → cell array of pointers: it contains pointers to functions that are used as activation functions. There must be (number hidden layers + output layer) number of entries specifying the activation function for each layer

[5]This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[6]This parameter is not useful to set the method appeared in literature. Nonetheless, it might be useful for further analyses considering a variation of the published method.

[7]For further references – [?]

- **numEpochs** → integer: number of training epochs
- **bias** → integer: it allows to take into account the bias if it is set as 1. If bias nodes are not required, "bias" has to be set as 0
- **candidateEpochs** → integer: number of maximum iterations needed to train the network with the current candidate. It is used in the outer "while" of the non-uniform wrapper
- **dividingPoint** → real number: amount of points used to train the networks. It is expressed as percentage of the data set number of points
- **valStep** → integer: number of iterations after which the validation phase takes place
- **valThreshold** → real number: threshold needed during the validation phase
- **learnAlg** → pointer: points to the function used as learning algorithm
- **rbpIncrease** → real number: η^-, [**?**]
- **rbpDescrease** → real number: η^+,[**?**]
- **rangeW** → real number: it represents the range of values assumed by the weights when initialized. If rangeW is set as 1, the weights will be initialized between -1 and 1
- **coeffHidNodes** → real number: percentage of hidden nodes with respect to the amount of available candidates

NB: I would strongly recommend to leave firstTermCaseVect and secondTermCaseVect as they are. For a more exhaustive documentation of the nearest neighbor parameters please take a look at the openTSTOOL documentation.

6 Wrapping Parameters

In parametersAndMethods lines 12-74 are devoted to convert the list of parameters given in input to a list of variables that will belong a particular method. Lines 180-288 are devoted to encapsulate the parameters of the chosen methods in a unique structure. This step is the needed brigde between the interface of MuTE with the user and the methods. To complete

this step the funcion createNameMethodParams that locks the input parameters in the method structure is needed. After that the callingMethods is called.

7 callingMethods

callingMethods function calls the chosen methods. The function only requires the more general data structure built during the wrapping of the parameters and the data. The output is a structure that groups the results of the methods.

8 Storing and Visualization of the Results

Three functions are involved in this step, called in parametersAndMethods:

- **storingOutput** → rearrange and store callingMethods output creating a different file for each realization and method. In this way each method will have its performances stored according to the analyzed file. Each file contains a structure named outputToStore. This is the most important post-processing function: starting from its output (the fields of outputToStore) the user can manage all the most useful informations in order to apply his own post-processing.
- **generateExpReport** → adds other fields to outputToStore structure such as pValue, modelOrder, testThreshold and bestOrder if those fields belong to the chosen method. The function, furthermore, generates the following files in entropyMatrices folder:
  - methodName_matrixPValues
  - methodName_testThresholdMtx
  - methodName_transferEntropyMtx
- **reshapeResults** → If either autoPairwiseTarDriv = 1 or handPairwiseTarDriv = 1 and the number of targets can be reshaped in a square, the function reshapes and stores the significance matrix and evaluates the mean values of the transfer entropy with respect to the number of trials obtaining a vector the same length as the number of targets. Then the mean transfer entropy values are reshaped and stored as well.

9 Output Returned

Let's take a closer look at the output files stored during the post-processing phase. We can find two folders where the outputs are stored, entropyMatrices and results folders.

In entropyMatrices the following files can be found:

- methodName_transferEntropyMtx contains a first column with the id of the data files. The other columns representing the entropy values: if the user sets allPairWiseCombinatins = 1 and he is dealing with 5 time series, he is going to evaluate $5 \times 4 = 20$ pairwise combinations not considering each series as target of itself. This means that the number of targets is 20. transferEntropyMtx will have 21 columns and as much rows as the number of trials.

- methodName_significanceOnDriv only contains as much columns as the number of targets and as much rows as the number of trials. Each entry may assume 1 or 0 values according to whether the link is significant or not.

- methodName_reshapedSignificance contains the significance values reshaped to form the adjacency matrix if the experimental set up allows the reshaping.

In results the following files are stored:

- methodName_typeAnalysis_patient_idTrial contains the outputToStore structure. This is the basic file from which everything can be evaluated during the post-processing phase.

- methodName_meanReshapeMtx is provided in results folder. This file contains the average of the values stored in transferEntropyMtx reshaped (if the experimental set up allows the reshaping) in order to show the adjacency matrix and be able to see the directed dynamical links.

NB: the user is kindly invited to develop his own functions to evaluate what may be the most useful informations according to his particular experiment. The post processing that I provide should be considered as an example of what can be evaluated.

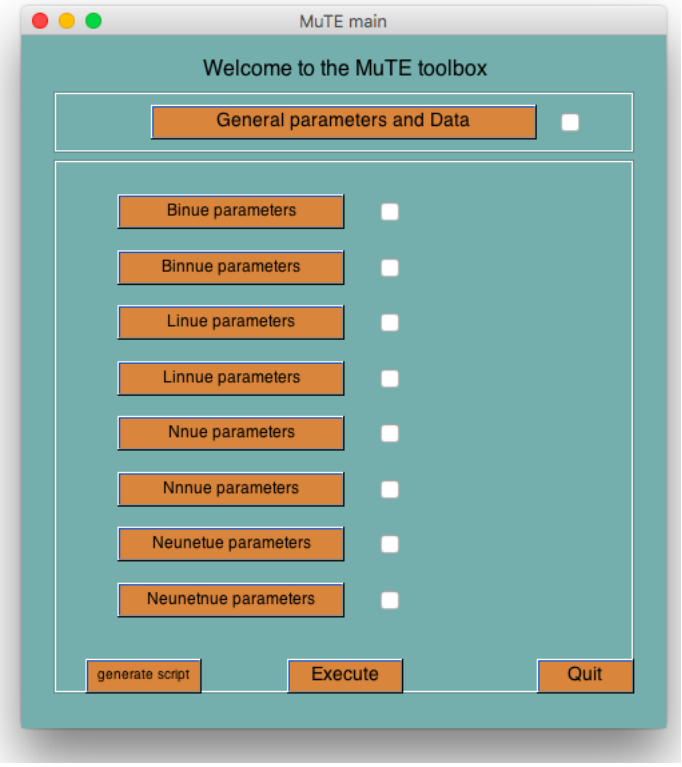

Figure 2: MuTE GUI main window.

10 The GUI

The MuTE GUI has been developed to make the MuTE toolbox more user-friendly. Please be aware that this tutorial only provides information on how to use the GUI. More information regarding the analysis methods can be found at the mute website and references therein `http://mutetoolbox.guru/`. It is strongly recommended to get familiarized with these methods before using the toolbox. In order to use the GUI, the latest version of the MuTE toolbox must be downloaded and added to your MATLAB path (add with subfolders). The most recent version of MuTE can be downloaded at the following address for free: `http://mutetoolbox.guru/downloads/`. To launch the GUI, type "mute" at the MATLAB command prompt. The following window will appear in figure 2.

11 How to use the GUI

The work flow can be separated into 3 parts: 1) Setting the general parameters and data selection, 2) setting method specific parameters and 3) generate script/execute the analysis.

11.1 General Parameters and Data

The general parameters refer to those parameters that apply to all the methods used (i.e. the methods selected in the 2nd step). These can be set by clicking the 'General parameters and Data' button on top of the main window. A new window will appear where the data can be selected and general parameters can be set, figure 3. In the new window, all the relevant parameter that need to be specified are listed. Most of the parameters have default values. In addition, every parameter is accompanied with a info ("?") button. More info regarding specific parameters can be obtained by clicking on the info button. Once all the relevant parameters are set, you can save the settings by clicking the 'Save settings' button. The default values can be restored by clicking the 'Restore default' button.

The GUI does an automatic check with respect to the selected data. The number of data files selected by the MuTE GUI will be displayed at the command prompt. If for some reasons the GUI is unable to find the data files, an error will pop up. If no data is found, it will be impossible to save the settings and thus also to generate a script/execute the analysis. Once the correct general parameters are saved, the general parameters window will be closed and the checkbox next to the 'general parameters and data' button in the main window will be checked.

As soon as the general parameters are saved, if everything is set up correctly the following warning is printed on the workspace in order to remind the user to wait until the computation is done and to not worry about the prompt shown in the workspace: "WARNING: the computation is taking place in another workspace. Please wait until '...COMPUTATION DONE!' is displayed."

11.2 Method-specific Parameters

In the middle of the main window, there are 8 buttons listed vertically. Each of these buttons will open a new window where you can specify method specific parameters, figure 4.

The methods listed from top to bottom are the following:

- Binue: Binning uniform embedding

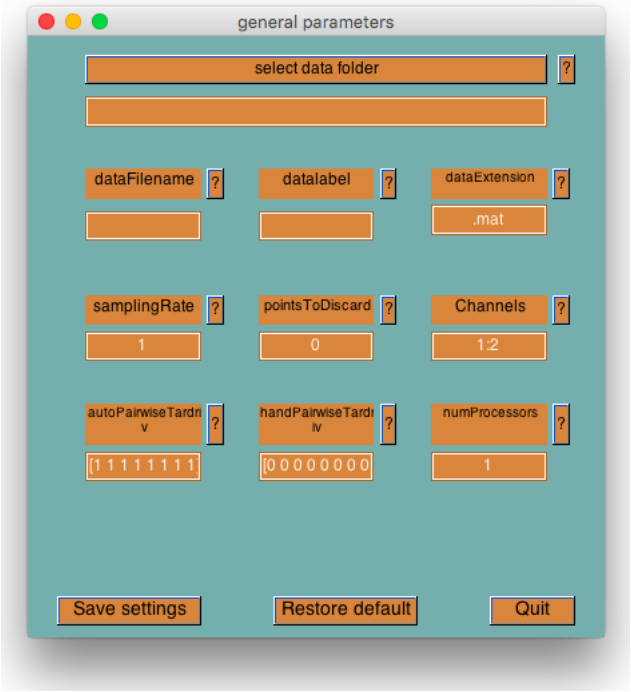

Figure 3: The general parameters and data pop-up window.

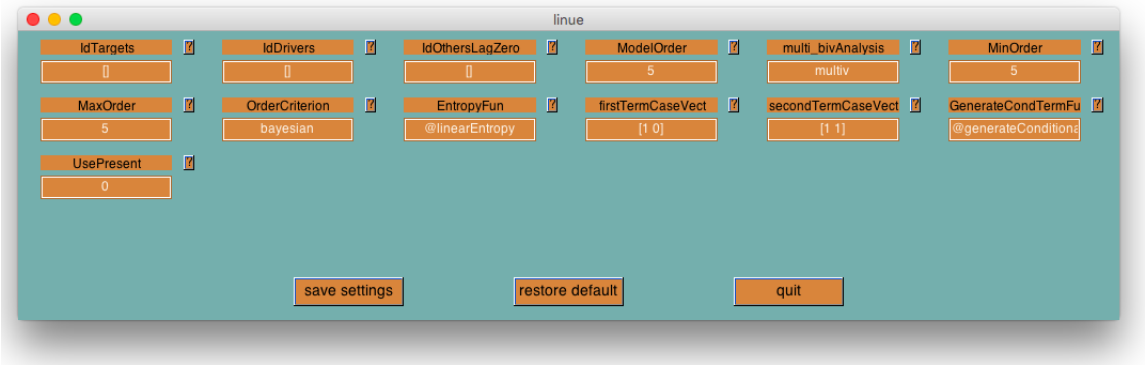

Figure 4: An example of a method specific pop-up window. Here the method specific parameters can be specified. In figure, the linue method settings are displayed.

- Binnue: Binning non-uniform embedding
- Linue: Linear uniform embedding
- Linnue: Linear non-uniform embedding
- Nnue: Nearest neighbor, uniform embedding
- Nnnue: Nearest neighbor, non-uniform embedding
- Neunetue: Neural networks uniform embedding
- Neunetnue: Neural networks non-uniform embedding

The method specific window is similar to the general parameters window. All the parameters have default values. It is recommended to keep the default values for most parameters. Each parameter is again provided with a ("?") info button. The info button opens a window where more information regarding that specific parameter can be obtained. Once the settings are saved, the window will be closed and the checkbox next to the parameter will be checked. If one of the method specific checkboxes is unchecked, then that method will not be performed during the analysis.

11.3 Generate Script/Execute the Analysis

Once all the setting have been set, the user has two options: 1) generate a script or 2) execute the analysis. When clicking the "generate" button, a matlab scrip, a .m file with the date and hour of generation in the name e.g.

$$mute\_analysis\_2016\_1\_28\_9h\_56min.m$$

, will be generated in the data folder. The analysis can than be run by simply running the script. When clicking the "execute" button, the same script will be generated but now it will also be executed automatically. Our goal of generating a script is to make the transition from point-and- click to command line analysis easier. The user must specify at least one of the methods and the general parameters, otherwise an error will pop-up and no script will be generated/executed. The analyses are done once the main window is closed. The results can be found in a folder inside the data directory.

Acknowledgments

The GUI was developed in collaboration with Frederick Van de Steen who arranged the code. For further references about the GUI please do not hesitate to contact him at *frederik.VandeSteen@ugent.be* or send an email by filling in the form available at `http://mutetoolbox.guru/contacts/`.