PREMER

USER'S GUIDE

Alejandro F. Villaverde (ale.fer.vi@gmail.com)
Julio R. Banga (julio@iim.csic.es)
Kolja Becker (kolja.becker@gmx.de)

With the collaboration of:

David Rodríguez Penas (CSIC)
David Henriques (CSIC)
John Ross (Stanford)
Federico Morán (UCM)
Abel Folch Fortuny (UPV)
Alberto Ferrer (UPV)

March 20, 2017

Contents

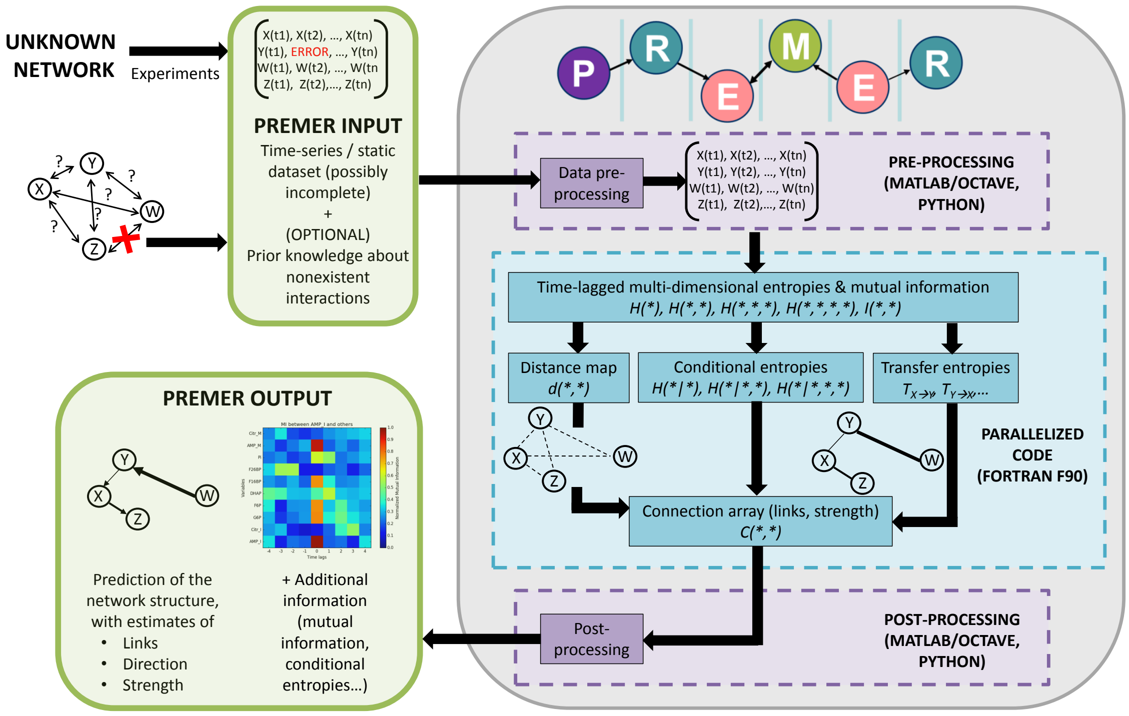

UNKNOWN NETWORK
Experiments
X(t1), X(t2), ..., X(tn)
Y(t1), ERROR, ..., Y(tn)
W(t1), W(t2), ..., W(tn
Z(t1), Z(t2),..., Z(tn)
PREMER INPUT
Time-series / static dataset (possibly incomplete)
+
(OPTIONAL)
Prior knowledge about nonexistent interactions
P R E M E R
Data pre-processing
PRE-PROCESSING (MATLAB/OCTAVE, PYTHON)
Time-lagged multi-dimensional entropies & mutual information
H(*), H(*,*), H(*,*,*), H(*,*,*,*), I(*,*)
Distance map d(*,*)
Conditional entropies H(*|*), H(*|*,*), H(*|*,*,*)
Transfer entropies T_X→Y, T_Y→X,...
PARALLELIZED CODE (FORTRAN F90)
Connection array (links, strength) C(*,*)
Post-processing
POST-PROCESSING (MATLAB/OCTAVE, PYTHON)
PREMER OUTPUT
Prediction of the network structure, with estimates of
Links
Direction
Strength
+ Additional information (mutual information, conditional entropies...)

Figure 1: Workflow of the PREMER algorithm. First, a data curation module imputes missing data and detects and corrects outliers, thus allowing the use of faulty datasets. Then PREMER calculates the distance between every possible pair of variables $d(X, Y)$ for several time delays. To this end it estimates the entropies of all variables $H(*)$, as well as the joint entropies $H(*, *)$ and the mutual information $I(*, *)$ of all pairs of variables. The user can choose to estimate also the multi-dimensional joint entropies of 3 and 4 variables ($H(*, *, *)$, $H(*, *, *, *)$), in order to use them in the subsequent entropy reduction step. The aim of this step is to determine whether all the variation in a variable Y can be explained by the variation in another variable X or, more generally, in a set of variables $\mathbf{X}$. By iterating through cycles of adding a variable X that reduces $H(Y|X, \mathbf{X})$ until no further reductions are obtained, the entropy reduction step yields the complete set of variables that control the variation in Y. Finally, directions are assigned to the links using transfer entropy, $T_{X\rightarrow Y}$, a non-symmetric measure of causality calculated from time-lagged conditional entropies.

1 Introduction

PREMER (**P**arallel **R**everse **E**ngineering with **M**utual information & **E**ntropy **R**eduction) is a general purpose software tool for inferring network structures. It calculates distances among variables using an entropic measure based on mutual information, which takes into account time delays. For this purpose the user can choose between several definitions and normalizations of mutual information. After obtaining the distance map, conditional entropies calculated from joint entropies of multiple variables are used to distinguish between direct and indirect interactions and to assign directionality. The whole workflow is summarized in Fig. 1. A first version of PREMER was originally presented at the 14th International Conference in Computational Methods in Systems Biology, CMSB 2016; the associated publication is Villaverde et al. (2016).

As a key component, PREMER includes an advanced F90 implementation of the MIDER method (Mutual Information Distance and Entropy Reduction). MIDER is a reverse-engineering algorithm (Villaverde et al., 2014) which was presented as a MATLAB implementation. Due to the fact that F90 (Fortran) is computationally more efficient than MATLAB, PREMER performs much faster than MIDER. Even more importantly, PREMER uses OpenMP directives that enable it to run seamlessly in parallel environments, thus allowing for additional speedups in performance.

Additionally, PREMER has user interfaces in MATLAB (which is fully compatible with Octave) and Python.

Furthermore, PREMER offers two additional capabilities that were not provided in the original MIDER toolbox. One is the ability to take prior knowledge into account, allowing to specify if a particular interaction is known to be nonexistent. This is of particular importance in applications such as gene regulatory network (GRN) inference, where only a subset of the genes — the transcription factors, TFs — can regulate other genes. The second one is the ability to handle datasets with missing values and/or outliers, using statistical techniques to impute new values which are coherent with the latent structure of the data.

2 License

PREMER is licensed under the GNU General Public License version 3 (GPLv3), a free, copyleft license for software.

3 Software contents

The PREMER code consists of a number of scripts (in MATLAB/Octave or Python, depending on the version) and an executable file. Additionally, source files written in F90 are also provided, which can be used for recompiling the executable.

- **Main script:** the master file is named `runPREMER.m` (in MATLAB/Octave) or `pyPREMER.py` (in Python). Running it will execute PREMER. The MATLAB/Octave version can be edited by the user in order to tune the algorithm's settings, as explained in section 6. In the Python version of PREMER these options are given as command line options.

- **Core calculations:** `PREMER.*` is the executable file that carries out most of the computations; it is called by `runPREMER.m` and `pyPREMER.py`. `PREMER.*` performs adaptive estimation of mutual information and multi-dimensional joint entropies using the algorithm presented in Cellucci et al. (2005), which was kindly provided by Dr. Alfonso Albano (aalbano@brynmawr.edu).

- **Auxiliary preprocessing:** Functions `OUTLIERS` and `TSR` preprocess the data, allowing to handle missing data and outliers. They were originally developed in MATLAB by Abel Folch Fortuny (abfolfor@upv.es) and Alberto Ferrer (aferrer@eio.upv.es), from the Universitat Politècnica de València (UPV), Spain. Python versions were written by Kolja Becker (IMB, Mainz, Germany). If the input dataset is incomplete, that is, if some data points are missing for a certain variable or set of variables, `TSR` (Trimmed Scores Regression) generates artificial data points and fills the gaps in the data, in a way that is coherent with the latent structure of the dataset. Additionally, `OUTLIERS` detects if the dataset contains abnormal (faulty) values. If this is the case, the user can choose to use the original dataset, or alternatively to remove the outliers and impute new data with the TSR procedure. These additions are described in Folch-Fortuny et al. (2015).

- **Auxiliary postprocessing:** The MATLAB/Octave version uses functions `plotResults.m`, `projectVars.m` and `arrow.m` are used for visualization purposes. The latter was developed by Dr. Erik A. Johnson (johnsone@usc.edu), who kindly agreed to its distribution with the PREMER package. Another auxiliary function, `formatFortranResults.m`, takes the output of the Fortran implementation and returns it as a MATLAB/Octave structure.

  In the Python version, the `ResultsClass` (within the `GetResults.py` module) parses the saved results file and creates a results object containing all relevant PREMER results as well as run information. `ResultsClass` can easily be imported into a Python shell (`from source.GetResults import ResultsClass`), and takes as input argument the (relative) path of the results folder. Plotting functions of pyPREMER are summarized within the `PlotMI.py` module.

4 Requirements and installation

4.1 Download and install

Download the PREMER package from `https://sites.google.com/site/premertoolbox/`. To install it, simply unzip it and save it somewhere in your computer.

4.2 Permissions

In some operating systems it may be necessary to give execution permissions to the PREMER executable before using it. This step is typically needed in Linux and OSX, but not in Windows. For example, on a Linux machine this can be done with the following command:

```
$ sudo chmod +x PREMERlin64.out
```

4.3 Requirements

PREMER can run on any operating system compatible with MATLAB, Octave, or Python. Some version of MATLAB, Octave, or Python must be installed. The only additional requisite is the Statistics package (if using Octave[1]) or the Statistics toolbox (if using MATLAB). The Python version uses various packages (NumPy, SciPy, Pandas) which are usually installed together with Python, and for which no licenses are needed. However, note that even the use of the Statistics toolbox/package is optional, since the core methodology does not require it; it is only used for visualization purposes.

Known issues: In some cases the shared multi-processing library (libiomp5.so) is not loaded correctly, and the user will have to do this manually. For this, please open a terminal an type:

```
export  LD_LIBRARY_PATH=$LD_LIBRARY_PATH:<premer_path>
```

where `<premer_path>` is the path where PREMER is located.

4.4 Re-compiling the code

In principle, no compilation is needed for using PREMER, as long as the operating system (Windows, Linux, or OSX) is 64-bit. The core computations are carried out by `PREMER.*`, which is already provided in the download as a compiled program.

However, source files are also provided (in the folder `fortran_sources`) to let the user re-compile the code. This may be useful if one wants to use the method in a different operating system, or for modifying parts of the method. Compilation instructions may differ depending on the choice of compiler and/or operating system, and it is not our intention to provide here detailed instructions to do so. Instead, we show some examples below.

A typical command for compiling the code using the Intel Fortran Compiler (IFORT) on a Linux 64-bit machine would be as follows:

```
$ ifort -openmp PREMER.f90 estimaH2.f90 estimaH3.f90 estimaH4.f90...
... ssort.f90 -o PREMERlin64
```

If we use instead GFortran, which is part of the GNU Compiler Collection, GCC, the compilation flags change slightly, and we need to build .o files for each of the auxiliary functions before compiling the main function as follows:

```
$ gfortran -c estimaH2.f90
$ gfortran -c estimaH3.f90
$ gfortran -c estimaH4.f90
$ gfortran -c ssort.f90
$ gfortran -static -fopenmp -ffree-line-length-none PREMER.f90 estimaH2.o...
...  estimaH3.o estimaH4.o ssort.o -o PREMERlin64
```

4.4.1 Remarks on compiling in OSX

The -static flag is not supported in GCC for OSX, which may be problematic in some cases. To avoid problems with library paths in MATLAB, we recommend to download GCC from

[1]To load the package, run "pkg load statistics" from the Octave prompt before using PREMER

`http://hpc.sourceforge.net/`and compile with the following command:

```
$ /usr/local/bin/gcc *.f90 -o ../PREMERosx.out -fopenmp -ffree-line-length-none...
... -L/usr/local/lib -lgfortran
```

We have tested this option with GCC 4.9 (Mavericks & Yosemite).

4.4.2 Remarks on compilers and performance

If you intend to re-compile the code, please be aware that the choice of the compiler and its options may influence software performance greatly. The executable files provided with this toolbox have been compiled with IFORT, Intel Fortran Compiler 14.0. For comparison we have also compiled PREMER using a free compiler, GFortran, version GCC 4.8.2. Table 1 shows differences in the performance of PREMER between these two versions. It can be noticed that the program compiled with IFORT is consistently faster than the one compiled with GFortran.

Table 1: Performance differences between IFORT and GFortran compilers

Number of threads	1	2	3	4	5	6	7	8
IFORT	69.14	44.18	34.90	34.51	29.04	27.31	26.71	25.13
GFortran	139.8	80.29	63.96	56.67	47.27	45.35	44.43	41.09

Comparison of CPU times (in seconds) taken by running versions of PREMER compiled with IFORT and GFortran for solving benchmark problem B7 of Villaverde et al. (2014) with 2 entropy reduction rounds. Results obtained on a computer with Intel Xeon Quadcore processor, 2.50 GHz, using MATLAB 7.9.0.529 (R2009b) 64-bit.

5 Quick start: How to infer a network with PREMER

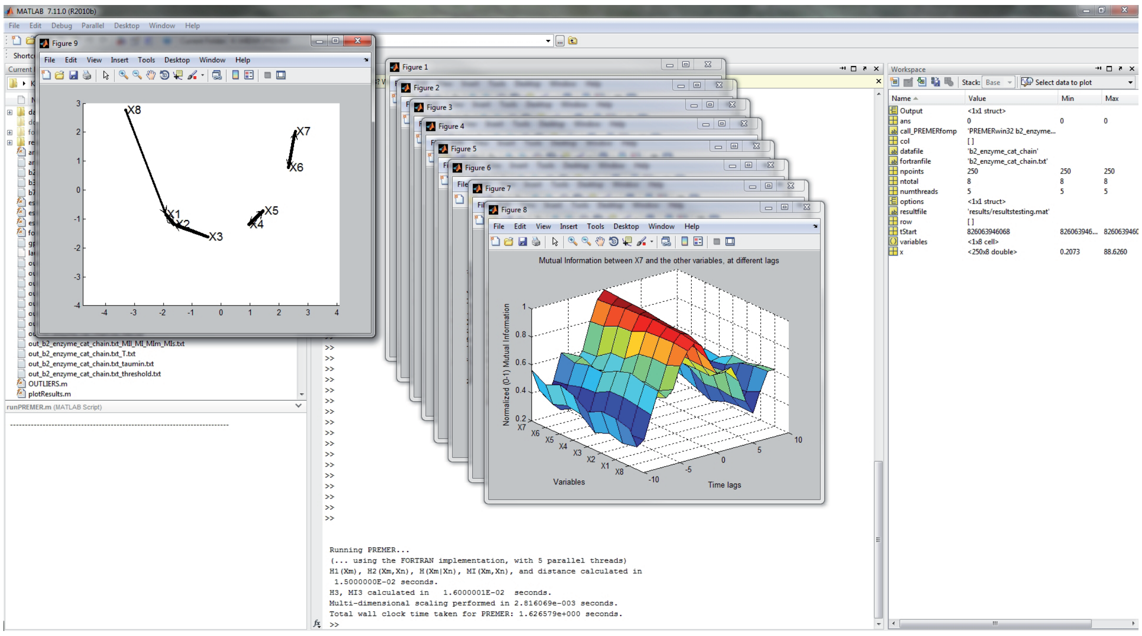
5.1 Basic Usage (MATLAB/Octave)

To start using PREMER in MATLAB or Octave you only need to follow these four steps:

1. Open a MATLAB or Octave session, and go to the PREMER root directory ("PREMER").
2. If you are using MATLAB, you must have installed the Statistics toolbox[2]. If you are using Octave, type "pkg load statistics" at the Octave prompt.
3. Define the problem and options by editing the script `runPREMER.m` (for details, see section 6).
   EXAMPLE (DEMO): If you are running PREMER for the first time and/or just want to see how it works, you can skip this step and leave `runPREMER.m` unedited. This will solve the benchmark problem B2 with default options.
4. Run `runPREMER.m` (to do this you can either type "runPREMER" in the command window, or right-click `runPREMER.m` in the "Current Directory" tab and select "run").

Done! Results should be obtained in a few seconds. A screenshot is shown in Figure 2. PREMER outputs two types of figures: (1) a 2D map of the distances among variables and the predicted links ("Figure 9" in the screenshot), and (2) for every variable, a plot of the mutual information between that variable and the rest, for all the time lags considered ("Figure 1"–"Figure 8" in the screenshot). Additionally, the results of the calculations are stored in the workspace and saved in a MAT-file. PREMER outputs are described in more detail in section 6.3. Further details about the use of PREMER are given in the next section.

[2]If your MATLAB doesn't have the Statistics toolbox, you can still use PREMER, since most of its features will work. In this case you must change the default options, setting "options.useStatistics = 1", as explained in section 6.2.


MATLAB 7.11.0 (R2010b)
Figure 9
Figure 8
Mutual Information between X7 and the other variables, at different lags
Normalized (0-1) Mutual Information
Variables
Time lags
Workspace

Name	Value	Min	Max
Output	<1x1 struct>		
ans	0	0	0
call_PREMERfomp	'PREMERwin32 b2_enzyme...		
col	[ ]		
datafile	'b2_enzyme_cat_chain'		
fortranfile	'b2_enzyme_cat_chain.txt'		
npoints	250	250	250
ntotal	8	8	8
numthreads	5	5	5
options	<1x1 struct>		
resultfile	'results/resultstesting.mat'		
row	[ ]		
tStart	826063946068	826063946...	82606394б0
variables	<1x8 cell>		
x	<250x8 double>	0.2073	88.6260

```
Running PREMER...
(... using the FORTRAN implementation, with 5 parallel threads)
H1(Xm), H2(Xm,Xn), H(Xm|Xn), MI(Xm,Xn), and distance calculated in
 1.5000000E-02 seconds.
H3, MI3 calculated in   1.6000001E-02  seconds.
Multi-dimensional scaling performed in 2.816069e-003 seconds.
Total wall clock time taken for PREMER: 1.626579e+000 seconds.
>>
```

Figure 2: Screenshot of an execution of PREMER

5.2 Basic Usage (Python)

To use PREMER in Python you only need to follow these steps:

1. Start a session with your Python environment, e.g. using the IPython console (best installed via anaconda: `https://docs.continuum.io/anaconda/install`).

2. In the IPython console, go to the pyPREMER folder.

3. EXAMPLE (DEMO): To run pyPREMER with the B2 example provided with the toolbox, type:
   `run pyPREMER.py ./data/b2_enzyme_cat_chain.csv`

These steps should yield similar results as those obtained with the MATLAB/Octave version discussed in the previous subsection.

5.3 Usage of PREMER including prior knowledge

It is possible to include prior knowledge about the structure of the network inferred by PREMER by excluding certain interactions from the inferred network *a priori*. For a quick example please follow these simple steps (instructions given for MATLAB/Octave):

1. Open the `runPREMER.m` script and change the name of the data file to `data_file = 'b10_Ecoli_18_timecourse-data_priorknow.mat'`. Upon running PREMER this will load all files necessary to run example b10 into the workspace. Here the variable `conn_matrix` represents a binary matrix in which each element of the matrix states whether an interaction between a regulator (columns) and a target (rows) should be included during the network inference steps.

2. Run `runPREMER` and inspect the output.

6 Usage

6.1 Input data

In principle, any dataset whose entries measure some feature of the network nodes may be used as input for PREMER. For example, if the nodes are chemical species, a dataset which contains the concentrations of the species at different time points would be appropriate.

The **input dataset** must be a $p \times n$ array, where each of the p rows corresponds to a measurement, and each of the n columns corresponds to a variable. The only requirement is that all the data contained in a row must have been measured at the same time instant. This means that, if you want to use a dataset which does not fulfill this condition (i.e. a dataset where different variables have been measured at different time points) you need to transform it first, typically by interpolating the values.

Datasets with **missing values** (coded as "NaN") are acceptable. To enable this, a preprocessing module analyzes the data and, if missing values are detected, new data points are imputed to fill the gaps. This is done using the TSR procedure described in Folch-Fortuny et al. (2015).

Input data can be provided in some of the most widely used **formats**: In MATLAB input files can be provided as a MATLAB data file (.mat), as an Excel file (.xlsx), or as a text file with columns delimited by tabulations (.txt). In Python input files should be provided as comma separated text file (csv). For each of these formats the data must be organized as follows:

- MATLAB (.mat) files must contain two variables, named 'x' and 'variables':

  x must be a $p \times n$ array of input data (p data points; n variables)

  variables must be a vector of n strings with the names of the variables

- Text (.txt) and Excel (.xlsx) files must have the following content:

  First row: names of the variables (one per column)

  Second and subsequent rows: data of each variable

- CSV-files (.csv) are similar to the above mentioned text file (.txt) with the difference that a comma (,) character is used as separator. Column headers should contain variable names. Row names (first column) can refer to experimental conditions or time-points. In any case this first column will need to be provided even if fields are empty.

Additionally, it is possible to specify **prior knowledge** about nonexistent interactions, in case it is available. In order to do this in MATLAB, include along with 'x' and 'variables' a third variable named 'conn_matrix' of size $n \times n$. The 'conn_matrix' represents the matrix of allowed connections. The entry (i,j) in this matrix corresponds to the link from variable x(j) to x(i). If it is known that this interaction is impossible (e.g. because x(j) is a gene that does not code for a transcription factor) the entry should be zero. Otherwise (if the interaction is in principle possible) the entry should be equal to one. In Python again a CSV-file with the information on interactions can be passed to pyPREMER using the -c argument. Column headers and row names of this CSV-file should contain variable names in the same order as in the data file. Columns indicate regulators while rows indicate targets (e.g. An interaction where variable i regulates variable j is found in column i, row j). If 'conn_matrix' is not included as a variable in the data file, PREMER assumes that all interactions are possible and creates a connection matrix with all entries equal to one.

6.2 Problem definition and options

Using the Python version of PREMER, problem definition and options detailed below are passed as user arguments (in the IPython console type run pyPREMER -h for details). In the MATLAB/Octave version these will be set by the user insider the `runPREMER.m` main file. This file starts with a section called "INITIALIZATION", followed by "PROBLEM DEFINITION", where the user must define two items:

- **datafile**: the name of the input file.
  Default: `datafile = 'b2_enzyme_cat_chain.mat'` .
- **resultfile**: the name of the output file (.mat) where the results will be stored.
  Default: `resultfile = 'results/resultstesting.mat'`.

The problem definition is followed by a "READ DATA" section. After it, the "OPTIONS" section defines a MATLAB structure called `options` containing the following fields:

- **options.useStatistics**: in PREMER, some calculations require the use of the Statistics toolbox (in MATLAB) or the Statistics package (in Octave[3]). Set it to 1 if this toolbox/package is available in your system, set it to 0 otherwise. Note that, if it is set to 0, the visual output will be disabled, and it won't be possible to correct outliers or missing data.
  Default: `options.useStatistics = 1`.
- **options.correctOutliers**: chooses whether to replace the outliers in the dataset or not. Set it to 1 if you want to replace any existing outliers with new data, or to 0 if you want to use the original dataset even if it has outliers.
  Default: `options.correctOutliers = 1`.
- **options.q**: value of the entropic parameter. Choose $q = 1$ for the classic Shannon entropy (also known as Boltzmann-Gibbs), or $q > 1$ for the generalized Tsallis entropy (Tsallis, 1988). Hint: in case you want to try Tsallis entropy, typical values are $1.5 < q < 3.5$.
  Default: `options.q = 1`.
- **options.MItype**: selects the type of normalization of mutual information used to create the distance map. Choose 'MI' for the classic, not normalized value; 'MImichaels' for the normalization presented in Michaels et al. (1998); 'MIlinfoot' for the one in Linfoot (1957); or 'MIstudholme' for the one in Studholme et al. (1999). Note that PREMER always calculates (and outputs) all the normalizations; this option selects the one used in the distance map.
  Default: `options.MItype = 'MI'`.
- **options.taumax**: the maximum time lag between two variables X and Y considered in the calculation of mutual information.
  Default: `options.taumax = 10`.
- **options.ert_crit**: number of entropy reduction rounds to carry out (0, 1, 2, or 3).
  Default: `options.ert_crit = 2`.
- **options.threshold**: entropy reduction threshold. Enter a number between 0.0 and 0.2 to fix it manually, or 1 to use a value obtained from the data.
  Default: `options.threshold = 1`.
- **options.plotMI**: plot mutual information arrays (=1) or not (=0).
  Default: `options.plotMI = 1`.
- **options.numthreads**: number of parallel threads. It should be an integer.
  Default: `options.numthreads = 1`.

Additionally, a file containing the input data must exist in the "data" folder. The PREMER distribution includes data files of 10 different case studies: one for each of the seven benchmark problems studied in Villaverde et al. (2014), plus three additional problems consisting of Gene Regulatory Networks of different sizes. The input file is specified in the first lines of the `runPREMER.m` script; by default the B2 benchmark data is chosen (`datafile = 'b2_enzyme_cat_chain.mat';`). For more information about input files see subsection 6.1.

Finally, it is recommended to save the results in another file. To do this, specify a name of the results file (e.g.: `resultfile = 'results/results_PREMER_b2_test1.mat';`). For more information about the outputs see subsection 6.3.

[3]To load it, run "pkg load statistics" from the Octave prompt.

6.3 Output

PREMER's MATLAB/Octave version yields a structure called **Output** with the following fields:

- **Output.H1** = n-vector of entropies.
- **Output.MI** = n*n*(nlags+1) array, mutual information (several lags).
- **Output.MIl** = mutual information normalized as in Linfoot (1957).
- **Output.MIm** = mutual information normalized as in Michaels et al. (1998).
- **Output.MIs** = mutual information normalized as in Studholme et al. (1999).
- **Output.H2** = n*n*(nlags+1) array, joint entropy of 2 variables.
- **Output.H3** = n*n*n array of joint entropy of 3 variables (calculated only if ert_crit >= 2).
- **Output.MI3** = n*n*n array of three-way mutual information Luo et al. (2008) (calculated only if if ert_crit >= 2).
- **Output.H4** = n*n*n*n array of joint entropy of 4 variables (calculated only if if ert_crit >= 3).
- **Output.dist** = n*n array of distance between variables.
- **Output.taumin** = n*n array of the time lags that minimize the entropic distance.
- **Output.cond_entr2** = n*n array of conditional entropies of 2 variables.
- **Output.cond_entr3** = n*n*n array of conditional entropies of 3 variables (calculated only if if ert_crit >= 2).
- **Output.cond_entr4** = n*n*n*n array of conditional entropies of 4 variables (calculated only if if ert_crit >= 3).
- **Output.con_array** = n*n array of connections between variables.
- **Output.adaptThres** = adaptive threshold value.
- **Output.T** = n*n array of transfer entropies.
- **Output.Y** = coordinates of the points from multidimensional scaling.

The **Output** structure is stored in a .mat file in the results folder. The name of this file is specified in the `resultsfile` variable mentioned above. The file contains not only the output structure, but also information about the problem input: the name of the data file, the names of the variables, their values, and the options used. Thus the results file contains all that is required to perform the calculations.

Additionally, a number of .txt files with partial results are also created and stored in the root directory. These files are redundant and are given for the user's convenience. They can be deleted without causing loss of information.

PREMER's Python implementation provides the same output information as text files.

6.4 Parallelization: OpenMP

A key feature of PREMER is its parallel implementation. The most computationally expensive parts of the network inference algorithm have been parallelized using **OpenMP directives**. OpenMP (Open Multi-Processing) is a specification for a set of compiler directives, library routines, and environment variables for high-level parallelism (see `http://openmp.org/wp/` for details). The OpenMP API is a **shared memory** model for parallel programming in Fortran and C/C++.

OpenMP directives exploit the **multi-core capabilities** of the computer in a **highly transparent** way: in PREMER, the user simply needs to specify the desired number of parallel threads in the variable `options.numthreads`. This is set in the "OPTIONS" section of the `runPREMER.m` script, as explained in subsection 6.2. If `options.numthreads` is given a value larger than one, the algorithm is executed in parallel.

PREMER is specially suited for **multi-core desktop computers**, where parallelization proceeds automatically after setting `options.numthreads = n` (with $n > 1$). Computation times can be reduced by increasing the number of parallel threads n. The natural upper limit for n is the number of available cores; using more parallel threads than the existing cores may degrade performance.

It is also possible to run PREMER on **computer clusters**. This usage may require additional actions depending on the cluster architecture and configuration. Here we illustrate this with an example: on a cluster with a Sun Grid Engine queuing system (SGE) the command used for launching a number of parallel threads would typically be similar to:

```
$ qsub -pe pe_fu n launch_matlab.sh runPREMER
```

where:

- `qsub` is the command used for job submission to the cluster.
- `-pe` is the flag for parallel execution.
- `pe_fu` (or similar) is the name of the parallel environment configuration, which defines how the grid engine distributes the processes among the nodes (e.g. a "fill-up" configuration).
- `n` is the number of threads.
- `launch_matlab.sh` is a shell script that starts MATLAB and runs the application defined by the subsequent text (i.e. `runPREMER`).

In this example it would be necessary to define the parallel environment configuration `pe_fu` (a task typically done by the cluster administrator) and to write the `launch_matlab.sh` shell script. Note that when using OpenMP the best performances are achieved if all the cores that work in parallel belong to the same node. This should be taken into account when configuring the parallel environment.

7 Methodological Details

PREMER starts with a data preprocessing module, which enables handling missing data Folch-Fortuny et al. (2015) and detect and correct outliers, thus allowing the use of incomplete or faulty datasets. This step uses the trimmed scores regression technique Arteaga and Ferrer (2002) based on principal component analysis to impute the missing values coherently with the latent structure of the dataset. Then the network inference algorithm is executed. It first calculates the distance between every possible pair of variables (X, Y) for several time delays (from $\tau = 0$ to $\tau = \tau_{max}$), as shown in Algorithm 1. To this end it estimates the entropies of all variables $H(*)$, as well as the joint entropies $H(*, *)$ and the mutual information $I(*, *)$ of all pairs of variables, using adaptive partitioning Cellucci et al. (2005). Partitioning involves sorting the data; to this end Singleton's algorithm 347 is used Singleton (1969). For visualization, variables are projected to two dimensions using multidimensional scaling Seber (2009).

Algorithm 1 Estimating the mutual information distance between two variables (X, Y)

```
1. Estimate joint entropy $H(X(t \pm \tau), Y(t))$ and mutual information $I(X(t \pm \tau), Y(t))$
2. Calculate the minimum distance between variables as $d(X, Y) = min_\tau e^{-I(X(t \pm \tau), Y(t))}$
3. Apply multidimensional scaling (MDS) to the distance matrix $d(X, Y)$ to obtain a 2D map
   of variables, which will be used for visualization
```

The user can choose to estimate also the multi-dimensional joint entropies of 3 and 4 variables ($H(*, *, *)$, $H(*, *, *, *)$), in order to use them in the subsequent entropy reduction step. The aim of this step is to determine whether all the variation in a variable Y can be explained by the variation in another variable X or, more generally, in a set of variables $\mathbf{X}$ Samoilov et al. (2001). The underlying rationale is as follows. Let $H(Y)$ be the entropy of variable Y, and $H(Y|\mathbf{X})$ the entropy of Y conditional on the knowledge of $\mathbf{X}$. If $(Y, \mathbf{X})$ are independent, $H(Y|\mathbf{X}) = H(Y)$; if not, $H(Y|\mathbf{X}) < H(Y)$. The amount by which the entropy of Y is reduced by X is taken as an indication of the strength of the X–Y interaction Villaverde et al. (2014). Hence, by iterating through cycles of adding a variable X that reduces $H(Y|X, \mathbf{X})$ until no further reductions are obtained, the entropy reduction step yields the complete set of variables that control the variation in Y. Thus, after performing the entropy reduction step, it is possible to draw an interaction map where the width of the links is proportional to the interaction strength. For practical reasons, in the current implementation of PREMER the number of entropy reduction cycles is limited to a maximum of three.

Finally, directions are assigned to the links using transfer entropy, $T_{X\to Y}$, a non-symmetric measure of causality Schreiber (2000) calculated from time-lagged conditional entropies as:

$$T_{X\to Y} = H(Y^t|Y^{t-\tau}) - H(Y^t|Y^{t-\tau}, X^{t-\tau}) \tag{1}$$

For every predicted link X–Y the direction is given by $\max(T_{X\to Y}, T_{Y\to X})$.

References

Arteaga, F. and Ferrer, A. (2002). Dealing with missing data in MSPC: several methods, different interpretations, some examples. *J. Chemom.*, 16(8-10):408–418.

Cellucci, C., Albano, A., and Rapp, P. (2005). Statistical validation of mutual information calculations: Comparison of alternative numerical algorithms. *Phys. Rev. E*, 71(6):066208.

Folch-Fortuny, A., Villaverde, A. F., Ferrer, A., and Banga, J. R. (2015). Enabling network inference methods to handle missing data and outliers. *BMC Bioinformatics*, 16(1):283.

Linfoot, E. (1957). An informational measure of correlation. *Inf. Control*, 1:85–89.

Luo, W., Hankenson, K., and Woolf, P. (2008). Learning transcriptional regulatory networks from high throughput gene expression data using continuous three-way mutual information. *BMC Bioinformatics*, 9(1):467.

Michaels, G., Carr, D., Askenazi, M., Fuhrman, S., Wen, X., and Somogyi, R. (1998). Cluster analysis and data visualization of large scale gene expression data. In *Pac. Symp. Biocomp.*, volume 3, pages 42–53.

Samoilov, M., Arkin, A., and Ross, J. (2001). On the deduction of chemical reaction pathways from measurements of time series of concentrations. *Chaos*, 11(1):108–114.

Schreiber, T. (2000). Measuring information transfer. *Phys. Rev. Lett.*, 85(2):461.

Seber, G. A. (2009). *Multivariate observations*, volume 252. John Wiley & Sons.

Singleton, R. C. (1969). Algorithm 347: an efficient algorithm for sorting with minimal storage [m1]. *Commun. ACM*, 12(3):185–186.

Studholme, C., Hill, D., and Hawkes, D. (1999). An overlap invariant entropy measure of 3D medical image alignment. *Pattern Recogn.*, 32:71–86.

Tsallis, C. (1988). Possible generalization of Boltzmann-Gibbs statistics. *J. Stat. Phys.*, 52(1):479–487.

Villaverde, A., Ross, J., Morán, F., and Banga, J. (2014). MIDER: network inference with mutual information distance and entropy reduction. *PLOS ONE*, 9(5):e96732.

Villaverde, A. F., Becker, K., and Banga, J. R. (2016). PREMER: parallel reverse engineering of biological networks with information theory. In *Computational Methods in Systems Biology*, volume 9859 of *Lecture Notes in Computer Science*, pages 323–329. Springer.