TimeSer

November 11, 2015

Contents

Chapter 1

Scripts

1.1 Calculate Mutual Info

The script **"calculate_mutual_info.py"** takes as an input a file containing various time-series replicas: each column will be interpreted as different replica and each row will be a different value as a function of time.

The replicas needs to have the same number of time-measures (i.e. same number of rows).

The output will contain a symmetric matrix of size (N x N) where N = number of replicas, which contains the Mutual Information of each replica against the others (on the diagonal the values of Information Entropy of each replica).

The script starts by loading the needed packages:

```
In [ ]: import ts
        import matplotlib.pyplot as plt
        import numpy as np
        from argparse import ArgumentParser
```

then the argument parser is defined:

```
In [ ]: parser = ArgumentParser( description = 'Calculate Mutual Information')
        #
        # SEE FILE FOR DETAILS.
        #
        options = parser.parse_args()
```

1.1.1 Arguments

The Input file format has been already described. Other options give the possibility to :

- load and analyse the time-series using only one every n-th frame (—**stride**)
- define the number of bins to be used to build the histograms (—**nbins**)
- use a simple (and not so clever) optimization for calculate the optimal bin-width (—**opt**)
- specify the dimensionality and the organization of the data in the input file (—**ndim** and —**interleave**)
  - For more informations concerning this aspect read the next paragraph
- create an image containing a representation of the results (—**plot**)

Data dimensionality and reorganization

By default the program assumes that the data are 1-dimensional time series, so if the input files contains N columns it will generate N replicas. But the data can also be multi dimensional: if the user specify that the data are k-dimensional, if the input files contains N columns it will generate N/k replicas. In the case the user specifies that the data has to be represented in k($>$ 1) dimensions, by default the script assumes that the values of the various dimensions of a given replicas are consecutives columns.

EXAMPLE: If we specify —dim 3 and tha files contains 6 columns, the program will genrate 2 3-dim replicas, and it will assume that the column in the input file are:

X1 Y1 Z1 X2 Y2 Z2

i.e. : the 1-st column is the 1-st dimension of the 1-st replica, the 2-nd column in the 2-nd dimension of the 1-st replica and so on.

Specifing the option —**interleave**, the user can modify this behaviour and the script will instead assume that the input data are organized as the following:

X1 X2 Y1 Y2 Z1 Z2

i.e. : the first N/k colum are the 1-st dimension of replicas, followed by N/K columns containing the 2-nd dimension and son on.

1.1.2 Description

The reorganization of the data in the correct order is made using the following function:

```
In [ ]: def interleave(data,ndim):
            nfr, nrep = data.shape
            out = np.zeros(data.shape)
            for i in range(nrep/ndim):
                or j in range(ndim):
                out[:,ndim*i+j]   = data[:,j*(nrep/ndim)+i]
            return out
```

Following our exploration of the script we now enter in the actual execution.

Firstly the options are stored in more readable variables.

```
In [ ]: f_dat = options.dat
        f_out = options.out
        stride = options.stride
```

and finally the data is read from the file specified from the user and if the —*interleave* option has been selected the data is reorganized as described above and stored in a numpy array named *dat*

```
In [ ]: dat   = np.loadtxt(f_dat)
        dat   = dat[::stride]

        if (options.interleave) & (options.ndim != 1):
                dat = interleave(dat,options.ndim)
```

Then the dat array is used to create an instance of the TimeSer Object defined in the *ts* module.

```
In [ ]: DATA= ts.TimeSer(dat,len(dat),dim=options.ndim,nbins=options.nbins)
        DATA.calc_bins(opt=options.opt)
```

In the TimeSer Object a series of procedre for the calculation of Statistical Entropy, Mutual Information and Information Transfer are available.

The most crucial and time consuming functions multiple are actually wrapper for **FORTRAN90** code that have been compiled with the **f2py** tool. These function variants are identified by the post-fix **"for"**.

Some of the functions have also been parallelized with **OpenMP**. The Module automatically identify the number of processors available on the computer, and automatically gnereates an equal number of threads and equally distributes the calculations among these threads. The parallelized versions are identified by the post-fix **“omp”**.

In the script here presented we use the **“OMP”** version of the **“mutual_info()”** function [called **“mutual_info_omp()”**], wich produces as an output two numpy arrays:

- **M** [size (NxN) N = num. of replicas] : Mutual Information.
- **E** [size (NxN) N = num. of replicas] : Entropies joint distributions of replicas.

```
In [ ]: M, E = DATA.mutual_info_omp()
```

Then finally an image representing the Mutual Information matrix is generated using matplotlib.

```
In [ ]: fig = plt.figure()
        ax  = fig.add_subplot(111)
        mat = ax.matshow(M)
        fig.colorbar(mat)
        plt.show()
```

If asked the image is also saved to a file in the SVG format (Scalable Vector Graphics) that can be easily opened with any vector graphic editor (e.g. Inkscape, Adobe Illustrator)

```
In [ ]: if options.plot:
            fig.savefig(f_out.split('.')[0]+".svg",format='svg')
```

And the Mutual Information Matrix is also saved to disk in text format.

```
In [ ]: np.savetxt(f_out,M)
        quit()
```