fathon: A Python package for a fast computation of detrendend fluctuation analysis and related algorithms

Stefano Bianchi

October 13, 2019

1 Introduction

Detrended fluctuation analysis (DFA) was first developed by Peng to study memory effects in sequences of DNA [1]. Since then, it found various applications in other fields, like geophysics or economy [2, 3, 4, 5, 6, 7, 8, 9, 10, 11]. DFA is useful in recognising long memory processes by means of the scaling properties of the fluctuation function $F(n) \sim n^H$. H is the Hurst exponents, and its value can tell if a process (most likely a time series) is persistent or anti-persistent:

- $H \in [0.0, 0.5) \rightarrow$ antipersistency;
- $H = 0.5 \rightarrow$ uncorrelated process;
- $H \in (0.5, 1.0] \rightarrow$ persistency;
- $H > 1.0$ nonstationary process, stronger long-range correlations are present.

The algorithm has later evolved to multifractal detrended fluctuation analysis (MFDFA) [12] to take into account the variability of time series and the possible multiple scaling properties of the fluctuation function. Other algorithms have been further developed, such as the time dependent Hurst exponent [13] that examines changes of persistency within time intervals, and the detrended cross-correlation analysis (DCCA) [14] that studies cross-correlations in terms of persistency between non-linear time series. Recently, a cross-correlation index ρ has been introduced in order to make the interpretation of DCCA clearer [15].

fathon is a Python package for DFA (Detrended Fluctuation Analysis) and related algorithms. It aims at providing a simple and interactive interface to perform multiple analyses on multiple time series. Its purpose is to gather all the algorithms together, in order to have a single package including different methodologies. They are optimised, easy to use in their basic or advanced versions, and the user can easily switch between them. The Python interface is class-based and easy to use, while the underlying code is written in Cython and C allowing a fast computation of the algorithms. *fathon* implements common operations for these kind of algorithms, such as the fluctuation function and its fit in different ranges, the multifractal spectrum, and the detrended cross-correlation index and its confidence intervals.

2 Software description

The *fathon* package provides codes for DFA (Detrended Fluctuation Analysis) and related algorithms. They can be computationally expensive, therefore the package is written mostly in Cython and C. Anyway, as the goal of *fathon* is to be used from Python, all the scripts are compiled as shared objects during installation and can be imported as usual as with Python modules. C scripts make use of GSL (GNU Scientific Library), that needs to be already installed before *fathon* installation. Cython allows to parallelise the code using OpenMP, and Linux users can exploit parallelisation by having a C compiler that supports OpenMP. Instead, default C compiler on macOS does not support OpenMP, the user can either disregard parallelisation or install a C compiler with OpenMP support (for enabling OpenMP parallelisation on macOS see the suggested link in the README.md file). After the installation, *fathon* can be used from terminal by writing `import fathon`, and then running the methods chosen for the specific analysis. Nevertheless, it is probably better to use *fathon* in a jupyter notebook, that allows an immediate visualisation of the analysis' results.

2.1 Software Architecture and Functionalities

fathon code is object-oriented and is organised into four main modules, namely **dfa**, **mfdfa**, **dcca**, and **ht**. Every module is written in Cython and calls external C functions for computationally expensive operations. Another pure Python module is present, **tsHelper**, whose purpose is to provide useful methods to preprocess time series. The object-oriented programming along with the Python programming language allows to easily run different analyses and to quickly modify parameters if different options have to be tested. The structure of the package is shown in Figure 1.

2.1.1 fathon.tsHelper

The **tsHelper** module provides the user with two methods, *subtractMean* and *toAggregated*. The former subtracts the mean of the input time series `tsVec`, a mandatory step in order to apply the algorithms. The latter instead computes the cumulative sum of `tsVec` after its mean has been subtracted. This could be not a mandatory step if the time series is already in this form.

2.1.2 fathon.dfa

The **dfa** module computes the detrended fluctuation analysis algorithm [1]. An object of the **DFA** class can be instantiated setting the time series to be analysed, `tsVec`, as a parameter. Every method of that object will then refer to `tsVec` during computations. Fluctuations can be computed via the **computeFlucVec** method. It offers flexibility allowing to choose different parameters. The smaller window's size `nMin` used during computation is the only required parameter. The other following parameters are optional: the bigger window's size `nMax` used during computation, that by default is one-fourth of the length of the time series [16]; the polynomial order `polOrd` to be fitted to the time series in every window, 1 by default; the step between two different window's sizes `nStep`, that is 1 by default and can be set to a different value to reduce computation time in case of very long time series; a boolean variable `revSeg`, false by default, used to choose how to compute the fluctuations, i.e. going only from the beginning to the end of the time series (false) or going from the beginning to the end and then backwards from the end to the beginning (true). If the fluctuations have been computed, two methods can be used to fit them and find the Hurst exponent. The former, **fitFlucVec**, fits the fluctuations once between window's sizes *fathon*

fathon.tsHelper

```
subtractMean(tsVec)
toAggregated(tsVec)
```

fathon.dfa

```
class DFA:

  __init__(tsVec)

  computeFlucVec(nMin,
          nMax=len(tsVec)/4,
          polOrd=1,
          nStep=1,
          revSeg=False)

  fitFlucVec(n_start=nMin,
          n_end=nMax)

  multiFitFlucVec(limits_list)
```

fathon.mfdfa

```
class MFDFA:

  __init__(tsVec)

  computeFlucVec(nMin,
          q_list,
          nMax=len(tsVec)/4,
          polOrd=1,
          nStep=1,
          revSeg=False)

  fitFlucVec(n_start=nMin,
          n_end=nMax)

  computeMassExponents()

  computeMultifractalSpectrum()
```

fathon.dcca

```
class DCCA:

  __init__(tsVec1=[],
          tsVec2=[])

  computeFlucVec(nMin,
          nMax=len(tsVec1)/4,
          polOrd=1,
          nStep=1,
          absVals=True)

  fitFlucVec(n_start=nMin,
          n_end=nMax)

  multiFitFlucVec(limits_list)

  computeRho(nMin,
          nMax=len(tsVec1)/4,
          polOrd=1,
          nStep=1)

  rhoThresholds(L,
          nMin,
          nMax,
          nSim,
          confLvl,
          polOrd=1,
          nStep=1)
```

fathon.ht

```
class HT:

  __init__(tsVec)

  computeHt(scales,
          polOrd=1,
          mfdfaPolOrd=1)
```

Figure 1: *fathon* package and its modules.

`n_start` and `n_end`. These parameters are optional and by default are equal to `nMin` and `nMax`, respectively. The latter, **multiFitFlucVec**, allows to fit the fluctuations separately in the different intervals given by `limits_list`, a list or 2-dimensional array in the form `[[n_start1, n_end1], [n_start2, n_end2], [n_start3, n_end3], ...]`.

2.1.3 fathon.mfdfa

The **mfdfa** module computes the multifractal detrended fluctuation analysis algorithm [12]. As for the **DFA** class, an object of the **MFDFA** class can be instantiated providing as a parameter the time series to be analysed `tsVec`. Every method of that object will then refer to `tsVec` during the computations. Fluctuations can be computed via the **computeFlucVec** method. It is essentially the same of the **dfa** module and requires only two parameters, the smaller window's size `nMin` used during computation and the list of q-orders (or a single q-order). If $q = 2$ is used, the algorithm is equivalent to detrended fluctuation analysis. The optional parameters are: the bigger window's size `nMax` used during computation, that by default is one-fourth of the length of the time series [16]; the polynomial order `polOrd` to be fitted to the time series in every window, 1 by default; the step between two different window's sizes `nStep`, that is 1 by default and can be set to a different value to reduce computation time in case of very long time series; a boolean variable `revSeg`, false by default, used to choose how to compute the fluctuations, i.e. going only from the beginning to the end of the time series (false) or going from the beginning to the end and then backwards from the end to the beginning (true). When the fluctuations have been computed, they can be fitted in order to find a list of generalised Hurst exponents h_q. The method **fitFlucVec** returns that list, that can be plotted against q to visually inspect the presence of multifractality in the time series. To further study multifractality, methods **computeMassExponents** and **computeMultifractalSpectrum** returns the mass exponents and the multifractal spectrum respectively.

2.1.4 fathon.dcca

The **dcca** module computes the detrended cross-correlation analysis algorithm [14]. Even though the algorithm is different, the structure of the module is similar to the **dfa** module. A **DCCA** object requires zero or two parameters in order to be instantiated. If the following two parameters, namely `tsVec1` and `tsVec2`, are provided, they represent the two time series to be correlated and they should have the same length. If lengths are different, the bigger time series is reduced to the length of the smaller one. All methods except the last one, `rhoThresholds`, require `tsVec1` and `tsVec2` to work. The method **computeFlucVec** computes the fluctuations and has the same parameters described for the **dfa** module in section 2.1.2. The smaller window's size `nMin` is the only required parameter, and the optional parameters have the same default values of the ones previously described in the other modules. Only the boolean value has a different denomination (`absVals`) and a different purpose. It allows to compute the fluctuations using the absolute or signed values of the difference between the time series and the fit in each window. The default value is true, meaning that absolute values are considered. As for the **dfa** module, if the fluctuations have been computed they can be fitted in order to find H. The **fitFlucVec** method fits the fluctuations once between window's sizes `n_start` and `n_end`, while the **multiFitFlucVec** method allows to fit the fluctuations separately in the different intervals given by `limits_list`, a list or 2-dimensional array in the form `[[n_start1, n_end1], [n_start2, n_end2], [n_start3, n_end3], ...]`. A recent algorithm [15] uses detrended cross-correlation analysis to compute an index, ρ, that ranges between -1 and 1 as for the standard cross-correlation, and that can be therefore easily interpreted. ρ can be computed using the method **computeRho**. This method computes detrended cross-correlation three times, between `tsVec1` and `tsVec2`, between `tsVec1` and `tsVec1`, and between `tsVec2` and `tsVec2`. Finally ρ is computed. The only required parameter is again the smaller window's size `nMin`. The optional parameters are the same of the method **computeFlucVec** with the same default values, with the exception of `absVals` that is not present since it has to be set to false in order to compute ρ. Confidence intervals of ρ can be computed using the method **rhoThresholds**. This is the only method that can be called with a zero-parameters inizialisation of the **DCCA** object, since it does not require information on the time series and can be therefore called separately. It computes ρ between two gaussian white noise series as much times as the value of the parameter `nSim` is, and selects the values of the confidence intervals based on the quantile value given by the `confLvl` parameter. The other required parameters are the smaller and bigger window's sizes `nMin` and `nMax`, and the length of the random time series `L`. `L` must be equal to the length of the time series that the confidence levels are evaluated for. The optional parameters are the polynomial order `polOrd` to be fitted to the time series in every window and the step between two different window's sizes `nStep`, both 1 by default. Depending on the values of `L` and `nSim`, confidence levels can be really expensive to evaluate, and a value of `nStep` greater than 1 is recommended.

2.1.5 fathon.ht

The **ht** module computes the time-dependent Hurst exponent H_t as explained in reference [13]. It has only one method, **computeHt**. It first computes the generalised Hurst exponent at $q = 0$, h_0, via multifractal detrended fluctuation analysis. It consequently computes H_t from that value. It takes a list of window's sizes that the algorithm is evaluated at as the only required parameter. The other two parameters are optional and are both values of the order of the polynomial to be fitted in each window. The former is used during the H_t computation while the latter is used during the h_0 computation. Both orders have a default values of 1.

3 Illustrative Examples

In this section, three examples of the use of the *fathon* package are given. Fig-

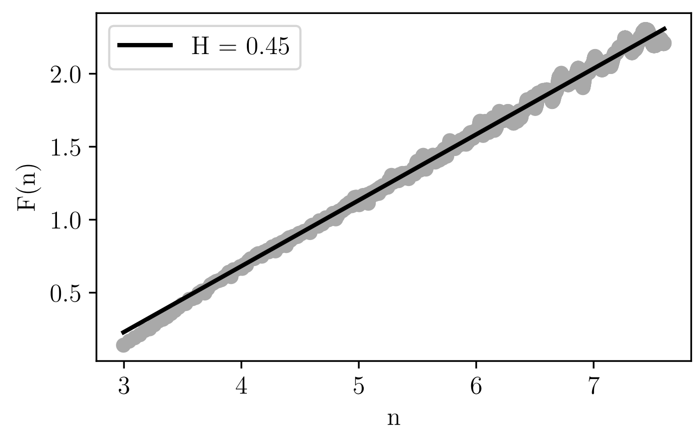

Figure 2: Detrended fluctuation analysis applied to a white noise time series. The solid black line is the linear fit of the logarithm of the fluctuations $F(n)$ versus the logarithm of the window's sizes n. The slope of the line corresponds to the value of the Hurst exponent H.

ure 2 shows the detrended fluctuation analysis applied to a white noise time series. Fluctuations have been computed via the **computeFlucVec** method using `revSeg=True`, and have been fitted via the **fitFlucVec** method. The result for the Hurst exponent is closer to the expected one ($H = 0.5$).

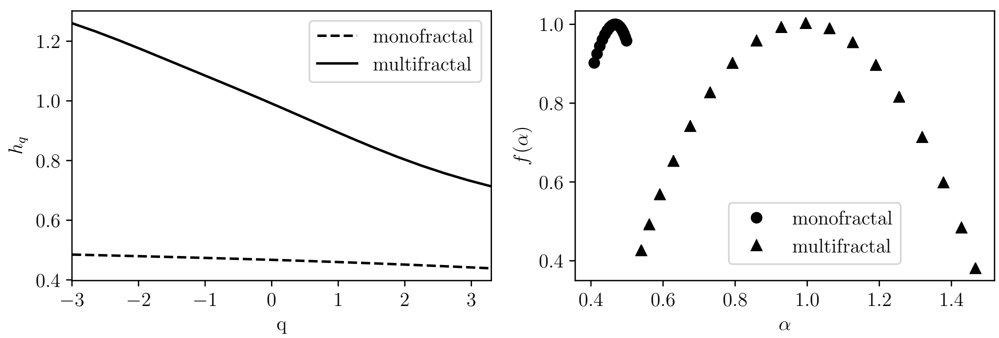

Figure 3: Left panel: h_q for a monofractal and a multifractal time series. Right panel: multifractal spectra $f(\alpha)$ for the same monofractal and multifractal time series.

In Figure 3, a comparison between a monofractal and a multifractal time series is presented as an example of multifractal detrended fluctuation analysis. The left panel shows h_q for both the time series, computed via the **computeFlucVec** and **fitFlucVec** methods. The multifractal time series' h_q exhibits a donward trend, meaning the Hurst exponent is not the same at all time scales, while the monofractal time series h_q is approximately constant. These behaviours are confirmed by the right panel of Figure 3, where the multifractal spectrum is shown. It has been computed via the **computeMultifractalSpectrum** method and is wider for the multifractal time series, as expected.

The last example shows the cross-correlation coefficient ρ between the residuals of two CO_2 time series after the yearly cycle removal, along with 95% confidence intervals (dashed lines in Figure 4). ρ has been computed via the **computeRho** method of the **DCCA** class, while the confidence levels have been computed via the **rhoThresholds** method with `nSim=200`. The two time series exhibit significant (even though not high) correlations at all the different time scales.

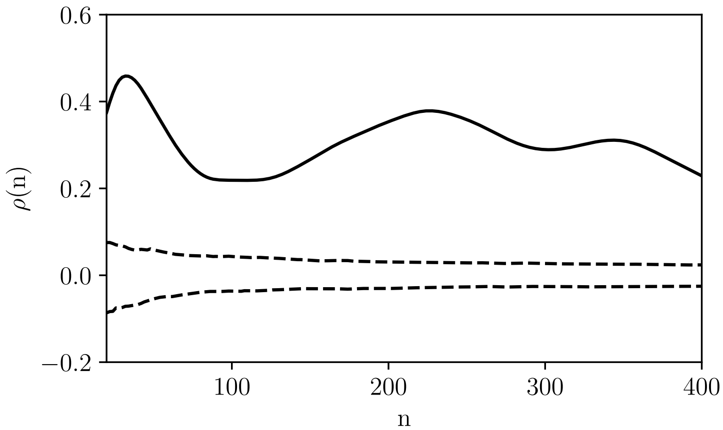

Figure 4: Cross-correlation coefficient ρ of the residuals of two CO_2 time series after the yearly cycle removal (solid line). The dashed lines represent the 95% confidence intervals.

References

[1] Peng CK, Buldyrev SV, Goldberger AL, Havlin S, Simons M, Stanley HE. Finite-size effects on long-range correlations: Implications for analyzing DNA sequences. Physical Review E 1993;47(5):3730.

[2] Varotsos C, Assimakopoulos MN, Efstathiou M. Long-term memory effect in the atmospheric CO_2 concentration at Mauna Loa. Atmospheric Chemistry and Physics 2007;7(3):629-634.

[3] Koscielny-Bunde E, Bunde A, Havlin S, Roman HE, Goldreich Y, Schellnhuber HJ. Indication of a universal persistence law governing atmospheric variability. Physical Review Letters 1998;81(3):729.

[4] Talkner P, Weber RO. Power spectrum and detrended fluctuation analysis: Application to daily temperatures. Physical Review E 2000;62(1):150.

[5] Fraedrich K, Blender R. Scaling of atmosphere and ocean temperature correlations in observations and climate models. Physical Review Letters 2003;90(10):108501.

[6] Matsoukas C, Islam S, Rodriguez-Iturbe I. Detrended fluctuation analysis of rainfall and streamflow time series. Journal of Geophysical Research: Atmospheres 2000;105(D23):29165-29172.

[7] Kavasseri RG, Nagarajan R. Evidence of crossover phenomena in wind-speed data. IEEE Transactions on Circuits and Systems I: Regular Papers 2004;51(11): 2255-2262.

[8] Liu Y, Gopikrishnan P, Stanley HE. Statistical properties of the volatility of price fluctuations. Physical Review E 1999;60(2):1390.

[9] Jànosi IM, Janecskò B, Kondor I. Statistical analysis of 5s index data of the Budapest Stock Exchange. Physica A: Statistical Mechanics and its Applications 1999;269(1):111-124.

[10] Grau-Carles P. Empirical evidence of long-range correlations in stock returns. Physica A: Statistical Mechanics and its Applications 2000;287(3):396-404.

[11] Ivanov PC, Yuen A, Podobnik B, Lee Y. Common scaling patterns in intertrade times of US stocks. Physical Review E 2004;69(5):056107.

[12] Kantelhardt JW, Zschiegner SA, Koscielny-Bunde E, Bunde A, Havlin S, Stanley HE. Multifractal detrended fluctuation analysis of nonstationary time series. Physica A: Statistical Mechanics and its Applications 2002;316(1):87-114.

[13] Ihlen EA. Introduction to multifractal detrended fluctuation analysis in Matlab. Fractal analysis 2012;3:97.

[14] Podobnik B, Stanley HE. Detrended cross-correlation analysis: a new method for analyzing two nonstationary time series. Physical Review Letters 2008;100.8:084102.

[15] Zebende GF. DCCA cross-correlation coefficient: quantifying level of cross-correlation. Physica A: Statistical Mechanics and its Applications 2011;390.4:614-618.

[16] Kantz H, Schreiber T. Nonlinear time series analysis. Cambridge University Press; 2004.