## Contents

1 Installation 3

2 An exemplary one-dimensional fractional Ornstein–Uhlenbeck process 5
   2.1 Generating a fractional Ornstein–Uhlenbeck process 5
   2.2 Using the MFDFA 6

3 Multifractality in one dimensional distributions 9
   3.1 Univariate random numbers from a Lévy stable distribution 9

4 Extensions 11
   4.1 Employing Empirical Mode Decompositions for detrending 11
      4.1.1 Understanding MFDFA’s EMD detrender 11
      4.1.2 Using MFDFA with EMD 12
   4.2 Extended Detrended Fluctuation Analysis 12
      4.2.1 Using MFDFA’s eDFA extension 12
   4.3 Moving window for segmentation 12
      4.3.1 Using MFDFA’s window extension 13

5 Literature 15

6 Funding 17

7 Table of Content 19
   7.1 Installation 19
   7.2 An exemplary one-dimensional fractional Ornstein–Uhlenbeck process 19
      7.2.1 Generating a fractional Ornstein–Uhlenbeck process 19
      7.2.2 Using the MFDFA 20
   7.3 Multifractality in one dimensional distributions 21
      7.3.1 Univariate random numbers from a Lévy stable distribution 21
   7.4 Extensions 23
      7.4.1 Employing Empirical Mode Decompositions for detrending 23
      7.4.2 Extended Detrended Fluctuation Analysis 24
      7.4.3 Moving window for segmentation 24
   7.5 Functions 25
      7.5.1 MFDFA 25
      7.5.2 Empirical Mode Decomposition for detrending timeseries 26
      7.5.3 Fractional Gaussian noise 27
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.6 License</td>
<td>27</td>
</tr>
<tr>
<td>7.7 Contact</td>
<td>28</td>
</tr>
<tr>
<td>Bibliography</td>
<td>29</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>31</td>
</tr>
<tr>
<td>Index</td>
<td>33</td>
</tr>
</tbody>
</table>
MFDFA is a python implementation of Multifractal Detrended Fluctuation Analysis, first developed by by Peng et al.\textsuperscript{1} and later extended to study multifractality MFDFA by Kandelhardt et al.\textsuperscript{2}.
MFDFA is available from PyPI, so you can use

```bash
pip install MFDFA
```

Then on your favourite editor just use

```python
from MFDFA import MFDFA
```

**Warning:** To use the extension to include Empirical Mode Decomposition detrending you will also need

```bash
pip install EMD-signal
```
An exemplary one-dimensional fractional Ornstein–Uhlenbeck process

For a more detailed explanation on how to integrate an Ornstein–Uhlenbeck process, see the kramersmoyal’s package.
You can also follow the fOU.ipynb

2.1 Generating a fractional Ornstein–Uhlenbeck process

This is one method of generating a (fractional) Ornstein–Uhlenbeck process with $H = 0.7$, employing a simple Euler–Maruyama integration method.

```python
# Imports
from MFDFA import MFDFA
from MFDFA import fgn
# where this second library is to generate fractional Gaussian noises

# integration time and time sampling
t_final = 500
delta_t = 0.001

# Some drift theta and diffusion sigma parameters
theta = 0.3
sigma = 0.1

# The time array of the trajectory
time = np.arange(0, t_final, delta_t)

# The fractional Gaussian noise
H = 0.7
dB = (t_final ** H) * fgn(N = time.size, H = H)

# Initialise the array y
y = np.zeros([time.size])

# Integrate the process
```

(continues on next page)
for i in range(1, time.size):
y[i] = y[i-1] - theta * y[i-1] * delta_t + sigma * dB[i]

And now you have a fractional process with a self-similarity exponent $H = 0.7$

### 2.2 Using the MFDFA

To now utilise the MFDFA, we take this exemplary process and run the (multifractal) detrended fluctuation analysis. For now let's consider only the monofractal case, so we need only $q = 2$.

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.unique(np.logspace(0.5, 3, 100).astype(int))
# Notice these must be ints, since these will segment
# the data into chunks of lag size

# Select the power q
q = 2

# The order of the polynomial fitting
order = 1

# Obtain the (MF)DFA as
lag, dfa = MFDFA(y, lag = lag, q = q, order = order)
```

Now we need to visualise the results, which can be understood in a log-log scale. To find $H$ we need to fit a line to the results in the log-log plot.

```python
# To uncover the Hurst index, let's get some log-log plots
plt.loglog(lag, dfa, 'o', label='fOU: MFDFA q=2')

# And now we need to fit the line to find the slope. We will
# fit the first points, since the results are more accurate
# there. Don't forget that if you are seeing in log-log
# scales, you need to fit the logs of the results
np.polyfit(np.log(lag[:15]), np.log(dfa[:15]),1)[0]

# Now what you should obtain is: slope = H + 1
```
2.2. Using the MFDFA

\[ F_q(s) \]

- \[ fOU: H = 0.7, q = 2 \]
- \[ \alpha = H + 1 \]
CHAPTER 3

Multifractality in one dimensional distributions

To show how multifractality can be studied, let us take a sample of random numbers of a symmetric Lévy distribution.

3.1 Univariate random numbers from a Lévy stable distribution

To obtain a sample of random numbers of Lévy stable distributions, use *scipy’s* `levy_stable`. In particular, take an \( \alpha \)-stable distribution, with \( \alpha = 1.5 \).

```python
# Imports
from MFDFA import MFDFA
from scipy.stats import levy_stable

# Generate 100000 points
alpha = 1.5
X = levy_stable.rvs(alpha=alpha, beta = 0, size=10000)
```

For MFDFA to detect the multifractal spectrum of the data, we need to vary the parameter \( q \in [-10, 10] \) and exclude 0. Let us also use a quadratic polynomial fitting by setting \( \text{order}=2 \).

```python
# Select a band of lags, which are ints
lag = np.unique(np.logspace(0.5, 3, 100).astype(int))

# Select a list of powers q
q_list = np.linspace(-10,10,41)
q_list = q_list[q_list!=0.0]

# The order of the polynomial fitting
order = 2

# Obtain the (MF)DFA as
lag, dfa = MFDFA(y, lag = lag, q = q_list, order = order)
```
Again, we plot this in a double logarithmic scale, but now we include 6 curves, from 6 selected $q = -10, -5, -2, 2, 5, 10$. Include as well are the theoretical curves for $q = -10$, with a slope of $1/\alpha = 1/1.5$ and $q = 10$, with a slope of $1/q = 1/10$. 

$F_q(s)$

$\log$ lag $s$
MF DFA as seen since its development a set of enhancements. In particular the usage of Empirical Mode Decomposition as a source of detrending, instead of polynomial fittings, which allows for a more precise removal of known trends in the timeseries.

4.1 Employing Empirical Mode Decompositions for detrending

Empirical Mode Decomposition (EMD), or maybe more correctly described, the Hilbert–Huang transform is a transformation analogous to a Fourier or Hilbert transform that decomposes a one-dimensional timeseries or signal into its Intrinsic Mode Functions (IMFs). For our purposes, we simply want to employ EMD to detrend a timeseries.

**Warning:** To use this feature, you need to first install PyEMD (EMD-signal) with

```
pip install EMD-signal
```

4.1.1 Understanding MF DFA’s EMD detrender

Take a timeseries $y$ and extract the Intrinsic Mode Functions (IMFs)

```python
# Import
from MFDFA import IMFs

# Extract the IMFs simply by employing
IMF = IMFs(y)
```

From here one obtains a $(..., \ y.size)$. Best now to study the different IMFs is to plot them and the timeseries $y$

```python
# Import
import matplotlib.pyplot as plt
```

(continues on next page)
# Plot the timeseries and the IMFs 6, 7, and 8

```python
plt.plot(X, color='black')
plt.plot(np.sum(IMF[:, [6, 7, 8]], axis=0).T)
```

## 4.1.2 Using MFDFA with EMD

To now perform the multifractal detrended fluctuation analysis, simply insert the IMFs desired to be subtracted from the timeseries. This will also for order = 0, not to do any polynomial detrending.

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa = MFDFA(y, lag = lag, extensions = {'EMD': [6, 7, 8]})
```

### 4.2 Extended Detrended Fluctuation Analysis

In the publication Detrended fluctuation analysis of cerebrovascular responses to abrupt changes in peripheral arterial pressure in rats, the authors introduce a new metric similar to the conventional Detrended Fluctuation Analysis (DFA) which they denote Extended Detrended Fluctuation Analysis (eDFA), which relies on extracting the difference of the minima and maxima for each segmentation of the data, granting a new power-law exponent to study, i.e., as in eq. (5) in the paper

\[ dF(n) = \max[F(n)] - \min[F(n)], \]

which in turn results in

\[ dF(n) \sim n^\beta. \]

#### 4.2.1 Using MFDFA’s eDFA extension

To obtain the eDFA, simply set the extension to True and add a new output function, here denoted edfa

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa, edfa = MFDFA(y, lag = lag, extensions = {'eDFA': True})
```

### 4.3 Moving window for segmentation

For short timeseries the segmentation of the data—especially for large lags—results in bad statistics, e.g. if a timeseries has 2048 datapoints and one wishes to study the fluctuation analysis up to a lag of 512, only 4 segmentations of the data are possible for the lag 512. Instead one can use an moving window over the timeseries to obtain better statistics at large lags.
4.3.1 Using MFDFA's window extension

To utilise a moving window one has to declare the moving windows step-size, i.e., the number of data points the window will move over the data. Say we wish to increase the statistics of the aforementioned example to include a moving window moving $32$ steps (so one has $64$ segments at a lag of $512$)

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa, edfa = MFDFA(y, lag = lag, extensions = {'window': 32})
```
CHAPTER 5

Literature


CHAPTER 6

Funding

7.1 Installation

MFDFA is available from PyPI, so you can use

```
pip install MFDFA
```

Then on your favourite editor just use

```
from MFDFA import MFDFA
```

**Warning:** To use the extension to include Empirical Mode Decomposition detrending you will also need

```
pip install EMD-signal
```

7.2 An exemplary one-dimensional fractional Ornstein–Uhlenbeck process

For a more detailed explanation on how to integrate an Ornstein–Uhlenbeck process, see the kramersmoyal’s package

You can also follow the fOU.ipynb

7.2.1 Generating a fractional Ornstein–Uhlenbeck process

This is one method of generating a (fractional) Ornstein–Uhlenbeck process with $H = 0.7$, employing a simple Euler–Maruyama integration method
# Imports
from MFDFA import MFDFA
from MFDFA import fgn
# where this second library is to generate fractional Gaussian noises

# integration time and time sampling
t_final = 500
delta_t = 0.001

# Some drift theta and diffusion sigma parameters
theta = 0.3
sigma = 0.1

# The time array of the trajectory
time = np.arange(0, t_final, delta_t)

# The fractional Gaussian noise
H = 0.7
dB = (t_final ** H) * fgn(N = time.size, H = H)

# Initialise the array y
y = np.zeros([time.size])

# Integrate the process
for i in range(1, time.size):
y[i] = y[i-1] - theta * y[i-1] * delta_t + sigma * dB[i]

And now you have a fractional process with a self-similarity exponent $H = 0.7$

## 7.2.2 Using the MFDFA

To now utilise the MFDFA, we take this exemplary process and run the (multifractal) detrended fluctuation analysis. For now let's consider only the monofractal case, so we need only $q = 2$.

# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.unique(np.logspace(0.5, 3, 100).astype(int))
# Notice these must be ints, since these will segment
# the data into chunks of lag size

# Select the power q
q = 2

# The order of the polynomial fitting
order = 1

# Obtain the (MF)DFA as
lag, dfa = MFDFA(y, lag = lag, q = q, order = order)

Now we need to visualise the results, which can be understood in a log-log scale. To find $H$ we need to fit a line to the results in the log-log plot

# To uncover the Hurst index, let's get some log-log plots
plt.loglog(lag, dfa, 'o', label='fOU: MFDFA q=2')

# And now we need to fit the line to find the slope. We will
7.3 Multifractality in one dimensional distributions

To show how multifractality can be studied, let us take a sample of random numbers of a symmetric Lévy distribution.

7.3.1 Univariate random numbers from a Lévy stable distribution

To obtain a sample of random numbers of Lévy stable distributions, use scipy’s `levy_stable`. In particular, take an \(\alpha\)-stable distribution, with \(\alpha = 1.5\)

```python
# Imports
from MFDFA import MFDFA
from scipy.stats import levy_stable

# Generate 100000 points
```
alpha = 1.5
X = levy_stable.rvs(alpha=alpha, beta = 0, size=10000)

For MFDFA to detect the multifractal spectrum of the data, we need to vary the parameter $q \in [-10, 10]$ and exclude 0. Let us also use a quadratic polynomial fitting by setting order=2

```python
# Select a band of lags, which are ints
lag = np.unique(np.logspace(0.5, 3, 100).astype(int))

# Select a list of powers q
q_list = np.linspace(-10, 10, 41)
q_list = q_list[q_list!=0.0]

# The order of the polynomial fitting
order = 2

# Obtain the (MF)DFA as
lag, dfa = MFDFA(y, lag = lag, q = q_list, order = order)
```

Again, we plot this in a double logarithmic scale, but now we include 6 curves, from 6 selected $q = -10, -5, -2, 2, 5, 10$. Include as well are the theoretical curves for $q = -10$, with a slope of $1/\alpha = 1/1.5$ and $q = 10$, with a slope of $1/q = 1/10$.
7.4 Extensions

MFDFA as seen since its development a set of enhancements. In particular the usage of Empirical Mode Decomposition as a source of detrending, instead of polynomial fittings, which allows for a more precise removal of known trends in the timeseries.

7.4.1 Employing Empirical Mode Decompositions for detrending

Empirical Mode Decomposition (EMD), or maybe more correctly described, the Hilbert–Huang transform is a transformation analogous to a Fourier or Hilbert transform that decomposes a one-dimensional timeseries or signal into its Intrinsic Mode Functions (IMFs). For our purposes, we simply want to employ EMD to detrend a timeseries.

**Warning:** To use this feature, you need to first install PyEMD (EMD-signal) with

```python
pip install EMD-signal
```

**Understanding MFDFA's EMD detrender**

Take a timeseries \( y \) and extract the Intrinsic Mode Functions (IMFs)

```python
# Import
from MFDFA import IMFs

# Extract the IMFs simply by employing
IMF = IMFs(y)
```

From here one obtains a \((..., y.size)\). Best now to study the different IMFs is to plot them and the timeseries \( y \)

```python
# Import
import matplotlib.pyplot as plt

# Plot the timeseries and the IMFs 6, 7, and 8
plt.plot(X, color='black')
plt.plot(np.sum(IMF[[6, 7, 8], :], axis=0).T)
```

**Using MFDFA with EMD**

To now perform the multifractal detrended fluctuation analysis, simply insert the IMFs desired to be subtracted from the timeseries. This will also for \( \text{order} = 0 \), not to do any polynomial detrending.

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa = MFDFA(y, lag = lag, extensions = {"EMD": [6, 7, 8]})
```
7.4.2 Extended Detrended Fluctuation Analysis

In the publication Detrended fluctuation analysis of cerebrovascular responses to abrupt changes in peripheral arterial pressure in rats, the authors introduce a new metric similar to the conventional Detrended Fluctuation Analysis (DFA) which they denote Extended Detrended Fluctuation Analysis (eDFA), which relies on extracting the difference of the minima and maxima for each segmentation of the data, granting a new power-law exponent to study, i.e., as in eq. (5) in the paper

\[ dF(n) = \max[F(n)] - \min[F(n)], \]

which in turn results in

\[ dF(n) \sim n^\beta. \]

Using MFDFA’s eDFA extension

To obtain the eDFA, simply set the extension to True and add a new output function, here denoted edfa

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa, edfa = MFDFA(y, lag = lag, extensions = {'eDFA': True})
```

7.4.3 Moving window for segmentation

For short timeseries the segmentation of the data—especially for large lags—results in bad statistics, e.g. if a timeseries has 2048 datapoints and one wishes to study the fluctuation analysis up to a lag of 512, only 4 segmentations of the data are possible for the lag 512. Instead one can use an moving window over the timeseries to obtain better statistics at large lags.

Using MFDFA’s window extension

To utilise a moving window one has to declare the moving windows step-size, i.e., the number of data points the window will move over the data. Say we wish to increase the statistics of the aforementioned example to include a moving window moving 32 steps (so one has 64 segments at a lag of 512)

```python
# Select a band of lags, which usually ranges from
# very small segments of data, to very long ones, as
lag = np.logspace(0.7, 4, 30).astype(int)

# Obtain the (MF)DFA by declaring the IMFs to subtract
# in a list in the dictionary of the extensions
lag, dfa, edfa = MFDFA(y, lag = lag, extensions = {'window': 32})
```
7.5 Functions

7.5.1 MFDFA


Multifractal Detrended Fluctuation Analysis of timeseries. MFDFA generates a fluctuation function $F_2(q,s)$, with $s$ the segment size and $q$ the q-powers, Take a timeseries $X$, find the integral $Y = \text{cumsum}(X)$, and segment the timeseries into $N$ segments of size $s$.

$$ F_2(v,s) = \frac{1}{s} \sum_{i=1}^{s} [Y((v-1)s+1) - y_{v,i}]^2, \text{for } v = 1, 2, \ldots, N_s, $$

with $y_{v,i}$ the polynomial fittings of order $m$. Having obtained the variances of each (detrended) segment, average over $s$ and increase $s$, to obtain the fluctuation function $F_2^q(s)$ depending on the segment length.

$$ F_2^q(s) = \left\{ \frac{1}{N_s} \sum_{v=1}^{N_s} [F_2(v,s)]^{q/2} \right\}^{1/q} $$

The fluctuation function $F_2^q(s)$ can now be plotted in a log-log scale, the slope of the fluctuation function $F_2^q(s)$ vs the s-segment size is the self-similarity scaling $h(q)$

$$ F_2^q(s) \sim s^{h(q)}. $$

If $H0$ in a monofractal series, use a second integration step by setting modified = True.

Parameters

- **timeseries** *(np.ndarray)* – A 1-dimensional timeseries ($N$, 1). The timeseries of length $N$.
- **lag** *(np.ndarray of ints)* – An array with the window sizes to calculate (ints). Notice $\min(\text{lag}) > \text{order} + 1$ given a polynomial fit of order $m$ needs at least $m$ points. The results are meaningless for ‘order = m’ and for lag > size of data / 4 since there is low statistics with < 4 windows to divide the timeseries.
- **order** *(int (default = 1))* – The order of the polynomials to approximate. order = 1 is the DFA1, which is a least-square fit of the data with a first order polynomial (a line), order = 2 is a second-order polynomial, etc. order = 0 skips the detrending process and hence gives the nondetrended fluctuation functions, i.e., simply Fluctuation Analysis.
- **q** *(np.ndarray (default = 2))* – Fractal exponent to calculate. Array in $[-10, 10]$. The values $q = 0$ will be removed, since the code does not converge there. $q = 2$ is the standard Detrended Fluctuation Analysis as is set a default.
- **stat** *(bool (default = False))* – Calculates the standard deviation associated with each segment’s averaging.
- **modified** *(bool (default = False))* – For data with the Hurst index 0, i.e., strongly anticorrelated, a standard MFDFA will result in inaccurate results, thus a further integration of the timeseries yields a modified scaling coefficient.
- **extensions** *(dict)* –
  - **EMD** *(list (default False))* If not None, requires a list of indices of the user-chosen IMFs obtained from an (externally performed) EMD analysis. The indexing starts from 0. Will enforce order = 0 since there is no need for a polynomial detrending.
- **eDFA**: bool (default False)  A method to evaluate the strength of multifractality. Calls function eDFA().

- **window**: bool (default False)  A moving window for smaller timeseries. Set window as int > 0 with the number of steps the window should move over the data. window = 1 will move window by 1 step. Since the timeseries is segmented at each lag length, any window choice > lag is only segmented once.

Returns

- **lag** (np.ndarray of ints) – Array of lags, realigned, preserving only different lags and with entries > order + 1

- **f** (np.ndarray) – A array of shape (size(lag),size(q)) of variances over the indicated lag windows and the indicated q-fractal powers.

References

MFDFA.MFDFA.eDFA(F: numpy.ndarray) → numpy.ndarray

In the reference indicated below a measure of nonstationarity was added by including a subsequent calculation of the extrema of the DFA. Denoted \( dF^2_q(s) \) the difference of the extrema at each segment, i.e.,

\[
dF^2_q(s) = \max[F^2_q(s)] - \min[F^2_q(s)]
\]

Parameters **F** (np.ndarray) – Fluctuation function given by the MFDFA().

Returns **res** – Difference of \( \max \) and \( \min \).

Return type np.ndarray

Notes

New in version 0.3.

References

7.5.2 Empirical Mode Decomposition for detrending timeseries

MFDFA.emddetrender.detrendedtimeseries(timeseries: numpy.ndarray, modes: list)

The function calculates the Intrinsic Mode Functions (IMFs) of a given timeseries, subtracts the user-chosen IMFs for detrending, and returns the detrended timeseries. Based on based on Dawid Laszuk’s PyEMD found at https://github.com/laszukdawid/PyEMD

Parameters

- **timeseries** (np.ndarray) – A 1-dimensional timeseries of length \( N \).

- **modes** (list) – List of integers indicating the indices of the IMFs to be subtracted/detrended from the timeseries.

Returns **detrendedTimeseries** – Detrended 1-dimensional timeseries.

Return type np.ndarray

Notes

New in version 0.3.
References

MF DFA.emd d etrender. IMFs (timeseries: numpy.ndarray)

Extract the Intrinsic Mode Functions (IMFs) of a given timeseries.

Parameters timeseries (np.ndarray) – A 1-dimensional timeseries of length N.

Notes

New in version 0.3.

Returns IMFs – The Intrinsic Mode Functions (IMFs) of the Empirical Mode Decomposition. These are of shape (... , timeseries.size), with the first dimension varying depending on the data. Last entry is the residuals.

Return type np.ndarray

7.5.3 Fractional Gaussian Noise

MF DFA.fgn.fgn (N: int, H: float) → numpy.ndarray

Generates fractional Gaussian noise with a Hurst index H in (0,1). If H = 1/2 this is simply Gaussian noise. The current method employed is the Davies–Harte method, which fails for H < 0. A Cholesky decomposition method and the Hosking’s method will be implemented in later versions.

Parameters

• N (int) – Size of fractional Gaussian noise to generate.
• H (float) – Hurst exponent H in (0,1).

Returns f – A array of size N of fractional Gaussian noise with a Hurst index H.

Return type np.ndarray

7.6 License

MIT License

Copyright (c) 2019-2021 Rydin

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
7.7 Contact

If you need help with something, find a bug, issue, or typo on the repository or in the code, you can contact me here: leonardo.rydin@gmail.com or open an issue on the GitHub repository.


Python Module Index

m
MF DFA.emd det render, 26
MF DFA.fgn, 27
MF DFA.MFDFA, 25
Index

D

detrendedtimeseries()  (in module MFDFA.emddetrender), 26

E

eDFA()  (in module MFDFA.MFDFA), 26

F

fgn()  (in module MFDFA.fgn), 27

I

IMFs()  (in module MFDFA.emddetrender), 27

M

MFDFA()  (in module MFDFA.MFDFA), 25
MFDFA.emddetrender (module), 26
MFDFA.fgn (module), 27
MFDFA.MFDFA (module), 25