Quantitative Finance

Release 9.6

The Sage Development Team

Jun 30, 2024

CONTENTS

CHAPTER

ONE

STOCK MARKET PRICE SERIES

This module's main class is *Stock*. It defines the following methods:

market_value()	Return the current market value of this stock.
current_price_data()	Get Yahoo current price data for this stock.
history()	Return an immutable sequence of historical price data for this stock
open()	Return a time series containing historical opening prices for this stock.
close()	Return the time series of all historical closing prices for this stock.
load_from_file()	Load historical data from a local csv formatted data file.

Warning: The *Stock* class is currently broken due to the change in the Yahoo interface. See trac ticket #25473.

AUTHORS:

- William Stein, 2008
- Brett Nakayama, 2008
- Chris Swierczewski, 2008

1.1 Classes and methods

class sage.finance.stock.**OHLC**(*timestamp*, *open*, *high*, *low*, *close*, *volume*)

Bases: `object`

Open, high, low, and close information for a stock. Also stores a timestamp for that data along with the volume.

INPUT:

- `timestamp` – string
- `open`, `high`, `low`, `close` – float
- `volume` – int

EXAMPLES:

```
sage: from sage.finance.stock import OHLC
sage: OHLC('18-Aug-04', 100.01, 104.06, 95.96, 100.34, 22353092)
 18-Aug-04 100.01 104.06 95.96 100.34   22353092
```

class sage.finance.stock.Stock(*symbol*, *cid=''*)

Bases: object

Class for retrieval of stock market information.

close(**args*, ***kwds*)

Return the time series of all historical closing prices for this stock. If no arguments are given, will return last acquired historical data. Otherwise, data will be gotten from Google Finance.

INPUT:

- startdate – string, (default: 'Jan+1,+1900')
- enddate – string, (default: current date)
- histperiod – string, ('daily' or 'weekly')

OUTPUT:

A time series – close price data.

EXAMPLES:

You can directly obtain close data as so:

```
sage: finance.Stock('vmw').close(startdate='Jan+1,+2008', enddate='Feb+1,+2008
↪')                 # optional -- internet # known bug
[84.6000, 83.9500, 80.4900, 72.9900, ... 83.0000, 54.8700, 56.4200, 56.6700, 57.
↪8500]
```

Or, you can initialize stock data first and then extract the Close data:

```
sage: c = finance.Stock('vmw')  # optional -- internet # known bug
sage: c.history(startdate='Feb+1,+2008', enddate='Mar+1,+2008')[:5]    #␣
↪optional -- internet # known bug
[
  1-Feb-08 56.98 58.14 55.06 57.85    2490481,
  4-Feb-08 58.00 60.47 56.91 58.05    1840709,
  5-Feb-08 57.60 59.30 57.17 59.30    1712179,
  6-Feb-08 60.32 62.00 59.50 61.52    2211775,
  7-Feb-08 60.50 62.75 59.56 60.80    1521651
]
sage: c.close()    # optional -- internet # known bug
[57.8500, 58.0500, 59.3000, 61.5200, ... 58.2900, 60.1800, 59.8600, 59.9500, 58.
↪6700]
```

Otherwise, *history()* will be called with the default arguments returning a year's worth of data:

```
sage: finance.Stock('vmw').close()   # random; optional -- internet # known bug
[57.7100, 56.9900, 55.5500, 57.3300, 65.9900 ... 84.9900, 84.6000, 83.9500, 80.
↪4900, 72.9900]
```

current_price_data()

Get Yahoo current price data for this stock.

This method returns a dictionary with the following keys:

'price'	'change'	'volume'	'avg_daily_volume'
'stock_exchange'	'market_cap'	'book_value'	'ebitda'
'dividend_per_share'	'dividend_yield'	'earnings_per_share'	'52_week_high'
'52_week_low'	'50day_moving_avg'	'200day_moving_avg'	'price_earnings_ratio'
'price_earnings_growth_ratio'	'price_sales_ratio'	'price_book_ratio'	'short_ratio'.

EXAMPLES:

```
sage: finance.Stock('GOOG').current_price_data()  #  random; optional -␣
↪internet # known bug
{'200day_moving_avg': '536.57',
 '50day_moving_avg': '546.01',
 '52_week_high': '599.65',
 '52_week_low': '487.56',
 'avg_daily_volume': '1826450',
 'book_value': '153.64',
 'change': '+0.56',
 'dividend_per_share': 'N/A',
 'dividend_yield': 'N/A',
 'earnings_per_share': '20.99',
 'ebitda': '21.48B',
 'market_cap': '366.11B',
 'price': '537.90',
 'price_book_ratio': '3.50',
 'price_earnings_growth_ratio': '0.00',
 'price_earnings_ratio': '25.62',
 'price_sales_ratio': '5.54',
 'short_ratio': '1.50',
 'stock_exchange': '"NMS"',
 'volume': '1768181'}
```

history(*startdate='Jan+1,+1900'*, *enddate=None*, *histperiod='daily'*)

Return an immutable sequence of historical price data for this stock, obtained from Google. OHLC data is stored internally as well. By default, returns the past year's daily OHLC data.

Dates `startdate` and `enddate` should be formatted `'Mon+d,+yyyy'`, where `'Mon'` is a three character abbreviation of the month's name.

Note: Google Finance returns the past year's financial data by default when `startdate` is set too low from the equity's date of going public. By default, this function only looks at the NASDAQ and NYSE markets. However, if you specified the market during initialization of the stock (i.e. `finance.Stock("OTC:NTDOY")`), this method will give correct results.

INPUT:

- `startdate` – string, (default: `'Jan+1,+1900'`)
- `enddate` – string, (default: current date)
- `histperiod` – string, (`'daily'` or `'weekly'`)

OUTPUT:

A sequence.

EXAMPLES:

We get the first five days of VMware's stock history:

```
sage: finance.Stock('vmw').history('Aug+13,+2007')[:5] # optional -- internet #␣
↪known bug
[
 14-Aug-07 50.00 55.50 48.00 51.00   38262850,
 15-Aug-07 52.11 59.87 51.50 57.71   10689100,
 16-Aug-07 60.99 61.49 52.71 56.99    6919500,
 17-Aug-07 59.00 59.00 54.45 55.55    3087000,
 20-Aug-07 56.05 57.50 55.61 57.33    2141900
]
sage: finance.Stock('F').history('Aug+20,+1992', 'Jul+7,+2008')[:5] # optional -
↪- internet # known bug
[
 20-Aug-92 0.00 7.90 7.73 7.83    5492698,
 21-Aug-92 0.00 7.92 7.66 7.68    5345999,
 24-Aug-92 0.00 7.59 7.33 7.35   11056299,
 25-Aug-92 0.00 7.66 7.38 7.61    8875299,
 26-Aug-92 0.00 7.73 7.64 7.68    6447201
]
```

Note that when `startdate` is too far prior to a stock's actual start date, Google Finance defaults to a year's worth of stock history leading up to the specified end date. For example, Apple's (AAPL) stock history only dates back to September 7, 1984:

```
sage: finance.Stock('AAPL').history('Sep+1,+1900', 'Jan+1,+2000')[0:5] #␣
↪optional -- internet # known bug
[
  4-Jan-99 0.00 1.51 1.43 1.47  238221200,
  5-Jan-99 0.00 1.57 1.48 1.55  352522800,
  6-Jan-99 0.00 1.58 1.46 1.49  337125600,
  7-Jan-99 0.00 1.61 1.50 1.61  357254800,
  8-Jan-99 0.00 1.67 1.57 1.61  169680000
]
```

Here is an example where we create and get the history of a stock that is not in NASDAQ or NYSE:

```
sage: finance.Stock("OTC:NTDOY").history(startdate="Jan+1,+2007", enddate=
↪"Jan+1,+2008")[:5]  # optional -- internet # known bug
[
  3-Jan-07 32.44 32.75 32.30 32.44     156283,
  4-Jan-07 31.70 32.40 31.20 31.70     222643,
  5-Jan-07 30.15 30.50 30.15 30.15      65670,
  8-Jan-07 30.10 30.50 30.00 30.10     130765,
  9-Jan-07 29.90 30.05 29.60 29.90     103338
]
```

Here, we create a stock by cid, and get historical data. Note that when using historical, if a cid is specified, it will take precedence over the stock's symbol. So, if the symbol and cid do not match, the history based on the contract id will be returned.

```
sage: sage.finance.stock.Stock("AAPL", 22144).history(startdate='Jan+1,+1990
↪')[:5] #optional -- internet # known bug
[
```

(continues on next page)

(continued from previous page)

```
  8-Jun-99 0.00 1.74 1.70 1.70   78414000,
  9-Jun-99 0.00 1.73 1.69 1.73   88446400,
 10-Jun-99 0.00 1.72 1.69 1.72   79262400,
 11-Jun-99 0.00 1.73 1.65 1.66   46261600,
 14-Jun-99 0.00 1.67 1.61 1.62   39270000
]
```

load_from_file(*file*)

Load historical data from a local csv formatted data file. Note that no symbol data is included in Google Finance's csv data. The csv file must be formatted in the following way, just as on Google Finance:

```
Timestamp,Open,High,Low,Close,Volume
```

INPUT:

- `file` – local file with Google Finance formatted OHLC data.

OUTPUT:

A sequence – OHLC data.

EXAMPLES:

Suppose you have a file in your home directory containing Apple stock OHLC data, such as that from Google Finance, called `AAPL-minutely.csv`. One can load this information into a Stock object like so. Note that the path must be explicit:

```
sage: filename = tmp_filename(ext='.csv')
sage: with open(filename, 'w') as fobj:
....:     _ = fobj.write("Date,Open,High,Low,Close,Volume\n1212405780,187.80,
↪187.80,187.80,187.80,100\n1212407640,187.75,188.00,187.75,188.00,2000\
↪n1212407700,188.00,188.00,188.00,188.00,1000\n1212408000,188.00,188.11,188.00,
↪188.00,2877\n1212408060,188.00,188.00,188.00,188.00,687")
sage: finance.Stock('aapl').load_from_file(filename)[:5]
...
1212408060 188.00 188.00 188.00 188.00        687,
1212408000 188.00 188.11 188.00 188.00       2877,
1212407700 188.00 188.00 188.00 188.00       1000,
1212407640 187.75 188.00 187.75 188.00       2000,
1212405780 187.80 187.80 187.80 187.80        100
]
```

Note that since the source file doesn't contain information on which equity the information comes from, the symbol designated at initialization of Stock need not match the source of the data. For example, we can initialize a Stock object with the symbol `'goog'`, but load data from `'aapl'` stock prices:

```
sage: finance.Stock('goog').load_from_file(filename)[:5]
[
1212408060 188.00 188.00 188.00 188.00        687,
1212408000 188.00 188.11 188.00 188.00       2877,
1212407700 188.00 188.00 188.00 188.00       1000,
1212407640 187.75 188.00 187.75 188.00       2000,
1212405780 187.80 187.80 187.80 187.80        100
]
```

market_value()
Return the current market value of this stock.

OUTPUT:

A Python float.

EXAMPLES:

```
sage: finance.Stock('goog').market_value()   # random; optional - internet #␣
↪known bug
575.83000000000004
```

open(**args*, ***kwds*)
Return a time series containing historical opening prices for this stock. If no arguments are given, will return last acquired historical data. Otherwise, data will be gotten from Google Finance.

INPUT:

- `startdate` – string, (default: `'Jan+1,+1900'`)
- `enddate` – string, (default: current date)
- `histperiod` – string, (`'daily'` or `'weekly'`)

OUTPUT:

A time series – close price data.

EXAMPLES:

You can directly obtain Open data as so:

```
sage: finance.Stock('vmw').open(startdate='Jan+1,+2008', enddate='Feb+1,+2008')␣
↪                # optional -- internet # known bug
[85.4900, 84.9000, 82.0000, 81.2500, ... 82.0000, 58.2700, 54.4900, 55.6000, 56.
↪9800]
```

Or, you can initialize stock data first and then extract the Open data:

```
sage: c = finance.Stock('vmw') # optional -- internet # known bug
sage: c.history(startdate='Feb+1,+2008', enddate='Mar+1,+2008')[:5]    #␣
↪optional -- internet # known bug
[
  1-Feb-08 56.98 58.14 55.06 57.85    2490481,
  4-Feb-08 58.00 60.47 56.91 58.05    1840709,
  5-Feb-08 57.60 59.30 57.17 59.30    1712179,
  6-Feb-08 60.32 62.00 59.50 61.52    2211775,
  7-Feb-08 60.50 62.75 59.56 60.80    1521651
]
sage: c.open()    # optional -- internet # known bug
[56.9800, 58.0000, 57.6000, 60.3200, ... 56.5500, 59.3000, 60.0000, 59.7900, 59.
↪2600]
```

Otherwise, *history()* will be called with the default arguments returning a year's worth of data:

```
sage: finance.Stock('vmw').open()   # random; optional -- internet # known bug
[52.1100, 60.9900, 59.0000, 56.0500, 57.2500, ... 83.0500, 85.4900, 84.9000, 82.
↪0000, 81.2500]
```

CHAPTER

TWO

TOOLS FOR WORKING WITH FINANCIAL OPTIONS

AUTHORS: - Brian Manion, 2013: initial version

sage.finance.option.**black_scholes**(*spot_price*, *strike_price*, *time_to_maturity*, *risk_free_rate*, *vol*, *opt_type*)
Calculates call/put price of European style options using Black-Scholes formula. See [Shr2004] for one of many standard references for this formula.

INPUT:

- `spot_price` – The current underlying asset price
- `strike_price` – The strike of the option
- `time_to_maturity` – The # of years until expiration
- `risk_free_rate` – The risk-free interest-rate
- `vol` – The volatility
- `opt_type` – string; The type of option, either `'put'` for put option or `'call'` for call option

OUTPUT:

The price of an option with the given parameters.

EXAMPLES:

```
sage: finance.black_scholes
doctest:warning...
DeprecationWarning: the package sage.finance is deprecated...
<built-in function black_scholes>

sage: finance.black_scholes(42, 40, 0.5, 0.1, 0.2, 'call')       # abs tol 1e-10
4.759422392871532
sage: finance.black_scholes(42, 40, 0.5, 0.1, 0.2, 'put')        # abs tol 1e-10
0.8085993729000958
sage: finance.black_scholes(100, 95, 0.25, 0.1, 0.5, 'call')     # abs tol 1e-10
13.695272738608132
sage: finance.black_scholes(100, 95, 0.25, 0.1, 0.5, 'put')      # abs tol 1e-10
6.349714381299734
sage: finance.black_scholes(527.07, 520, 0.424563772, 0.0236734,0.15297,'whichever␣
↪makes me more money')
Traceback (most recent call last):
...
ValueError: 'whichever makes me more money' is not a valid string
```

CHAPTER

THREE

MULTIFRACTAL RANDOM WALK

This module implements the fractal approach to understanding financial markets that was pioneered by Mandelbrot. In particular, it implements the multifractal random walk model of asset returns as developed by Bacry, Kozhemyak, and Muzy, 2006, *Continuous cascade models for asset returns* and many other papers by Bacry et al. See http://www.cmap.polytechnique.fr/~bacry/ftpPapers.html

See also Mandelbrot's *The Misbehavior of Markets* for a motivated introduction to the general idea of using a self-similar approach to modeling asset returns.

One of the main goals of this implementation is that everything is highly optimized and ready for real world high performance simulation work.

AUTHOR:

- William Stein (2008)

sage.finance.fractal.**fractional_brownian_motion_simulation**(*H*, *sigma2*, *N*, *n=1*)

Return the partial sums of a fractional Gaussian noise simulation with the same input parameters.

INPUT:

- `H` – float; `0 < H < 1`; the Hurst parameter.
- `sigma2` - float; innovation variance (should be close to 0).
- `N` – positive integer.
- `n` – positive integer (default: 1).

OUTPUT:

List of `n` time series.

EXAMPLES:

```
sage: set_random_seed(0)
sage: finance.fractional_brownian_motion_simulation(0.8,0.1,8,1)
[[-0.0754, 0.1874, 0.2735, 0.5059, 0.6824, 0.6267, 0.6465, 0.6289]]
sage: set_random_seed(0)
sage: finance.fractional_brownian_motion_simulation(0.8,0.01,8,1)
[[-0.0239, 0.0593, 0.0865, 0.1600, 0.2158, 0.1982, 0.2044, 0.1989]]
sage: finance.fractional_brownian_motion_simulation(0.8,0.01,8,2)
[[-0.0167, 0.0342, 0.0261, 0.0856, 0.1735, 0.2541, 0.1409, 0.1692],
 [0.0244, -0.0153, 0.0125, -0.0363, 0.0764, 0.1009, 0.1598, 0.2133]]
```

sage.finance.fractal.**fractional_gaussian_noise_simulation**(*H*, *sigma2*, *N*, *n=1*)

Return `n` simulations with `N` steps each of fractional Gaussian noise with Hurst parameter `H` and innovations variance `sigma2`.

INPUT:

- H – float; 0 < H < 1; the Hurst parameter.
- sigma2 - positive float; innovation variance.
- N – positive integer; number of steps in simulation.
- n – positive integer (default: 1); number of simulations.

OUTPUT:

List of n time series.

EXAMPLES:

We simulate a fractional Gaussian noise:

```
sage: set_random_seed(0)
sage: finance.fractional_gaussian_noise_simulation(0.8,1,10,2)
[[-0.1157, 0.7025, 0.4949, 0.3324, 0.7110, 0.7248, -0.4048, 0.3103, -0.3465, 0.
↪2964],
 [-0.5981, -0.6932, 0.5947, -0.9995, -0.7726, -0.9070, -1.3538, -1.2221, -0.0290, 1.
↪0077]]
```

The sums define a fractional Brownian motion process:

```
sage: set_random_seed(0)
sage: finance.fractional_gaussian_noise_simulation(0.8,1,10,1)[0].sums()
[-0.1157, 0.5868, 1.0818, 1.4142, 2.1252, 2.8500, 2.4452, 2.7555, 2.4090, 2.7054]
```

ALGORITHM:

See *Simulating a Class of Stationary Gaussian Processes using the Davies-Harte Algorithm, with Application to Long Meoryy Processes*, 2000, Peter F. Craigmile for a discussion and references for why the algorithm we give – which uses the `stationary_gaussian_simulation()` function.

sage.finance.fractal.**multifractal_cascade_random_walk_simulation**(*T*, *lambda2*, *ell*, *sigma2*, *N*, *n=1*)

Return a list of n simulations of a multifractal random walk using the log-normal cascade model of Bacry-Kozhemyak-Muzy 2008. This walk can be interpreted as the sequence of logarithms of a price series.

INPUT:

- T – positive real; the integral scale.
- lambda2 – positive real; the intermittency coefficient.
- ell – a small number – time step size.
- sigma2 – variance of the Gaussian white noise eps[n].
- N – number of steps in each simulation.
- n – the number of separate simulations to run.

OUTPUT:

List of time series.

EXAMPLES:

```
sage: set_random_seed(0)
sage: a = finance.multifractal_cascade_random_walk_simulation(3770,0.02,0.01,0.01,
↪10,3)
sage: a
[[-0.0096, 0.0025, 0.0066, 0.0016, 0.0078, 0.0051, 0.0047, -0.0013, 0.0003, -0.
↪0043],
 [0.0003, 0.0035, 0.0257, 0.0358, 0.0377, 0.0563, 0.0661, 0.0746, 0.0749, 0.0689],
 [-0.0120, -0.0116, 0.0043, 0.0078, 0.0115, 0.0018, 0.0085, 0.0005, 0.0012, 0.0060]]
```

The corresponding price series:

```
sage: a[0].exp()
[0.9905, 1.0025, 1.0067, 1.0016, 1.0078, 1.0051, 1.0047, 0.9987, 1.0003, 0.9957]
```

MORE DETAILS:

The random walk has n-th step $\text{eps}_n e^{\omega_n}$, where eps_n is gaussian white noise of variance σ^2 and ω_n is renormalized gaussian magnitude, which is given by a stationary gaussian simulation associated to a certain autocovariance sequence. See Bacry, Kozhemyak, Muzy, 2006, *Continuous cascade models for asset returns* for details.

sage.finance.fractal.**stationary_gaussian_simulation**(*s*, *N*, *n=1*)

Implementation of the Davies-Harte algorithm which given an autocovariance sequence (ACVS) `s` and an integer `N`, simulates `N` steps of the corresponding stationary Gaussian process with mean 0. We assume that a certain Fourier transform associated to `s` is nonnegative; if it isn't, this algorithm fails with a `NotImplementedError`.

INPUT:

- `s` – a list of real numbers that defines the ACVS. Optimally `s` should have length `N+1`; if not we pad it with extra 0's until it has length `N+1`.
- `N` – a positive integer.

OUTPUT:

A list of `n` time series.

EXAMPLES:

We define an autocovariance sequence:

```
sage: N = 2^15
sage: s = [1/math.sqrt(k+1) for k in [0..N]]
sage: s[:5]
[1.0, 0.7071067811865475, 0.5773502691896258, 0.5, 0.4472135954999579]
```

We run the simulation:

```
sage: set_random_seed(0)
sage: sim = finance.stationary_gaussian_simulation(s, N)[0]
doctest:warning...
DeprecationWarning: the package sage.finance is deprecated...
```

Note that indeed the autocovariance sequence approximates `s` well:

```
sage: [sim.autocovariance(i) for i in [0..4]]
[0.98665816086255..., 0.69201577095377..., 0.56234006792017..., 0.48647965409871...,
↪ 0.43667043322102...]
```

Warning: If you were to do the above computation with a small value of `N`, then the autocovariance sequence would not approximate `s` very well.

REFERENCES:

This is a standard algorithm that is described in several papers. It is summarized nicely with many applications at the beginning of *Simulating a Class of Stationary Gaussian Processes Using the Davies-Harte Algorithm, with Application to Long Memory Processes*, 2000, Peter F. Craigmile, which is easily found as a free PDF via a Google search. This paper also generalizes the algorithm to the case when all elements of `s` are nonpositive.

The book *Wavelet Methods for Time Series Analysis* by Percival and Walden also describes this algorithm, but has a typo in that they put a 2π instead of π a certain sum. That book describes exactly how to use Fourier transform. The description is in Section 7.8. Note that these pages are missing from the Google Books version of the book, but are in the Amazon.com preview of the book.

CHAPTER

FOUR

MARKOV SWITCHING MULTIFRACTAL MODEL

REFERENCE:

How to Forecast Long-Run Volatility: Regime Switching and the Estimation of Multifractal Processes, Calvet and Fisher, 2004.

AUTHOR:

- William Stein, 2008

class sage.finance.markov_multifractal.**MarkovSwitchingMultifractal**(*kbar*, *m0*, *sigma*, *gamma_kbar*, *b*)

Bases: `object`

INPUT:

- `kbar` – positive integer
- `m0` – float with `0 <= m0 <= 2`
- `sigma` – positive float
- `gamma_kbar` – float with `0 <= gamma_kbar < 1`
- `b` – float > 1

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,0.5,0.95,3); msm
Markov switching multifractal model with m0 = 1.4, sigma = 0.5, b = 3.0, and gamma_
↪8 = 0.95
sage: yen_usd = finance.MarkovSwitchingMultifractal(10,1.448,0.461,0.998,3.76)
sage: cad_usd = finance.MarkovSwitchingMultifractal(10,1.278,0.262,0.644,2.11)
sage: dm = finance.MarkovSwitchingMultifractal(10,1.326,0.643,0.959,2.7)
```

b()

Return parameter b of Markov switching multifractal model.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1,0.95,3)
sage: msm.b()
3.0
```

gamma()

Return the vector of the kbar transitional probabilities.

OUTPUT:

- gamma – a tuple of `self.kbar()` floats.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1.0,0.95,3)
sage: msm.gamma()
(0.001368852970712986, 0.004100940201672509, 0.012252436441829..., 0.
↪03630878209190..., 0.10501923017634..., 0.28312883556311..., 0.6315968501359..
↪, 0.950000000000000...)
```

gamma_kbar()

Return parameter gamma_kbar of Markov switching multifractal model.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,0.01,0.95,3)
sage: msm.gamma_kbar()
0.95
```

kbar()

Return parameter kbar of Markov switching multifractal model.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,0.01,0.95,3)
sage: msm.kbar()
8
```

m0()

Return parameter m0 of Markov switching multifractal model.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1,0.95,3)
sage: msm.m0()
1.4
```

sigma()

Return parameter sigma of Markov switching multifractal model.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1,0.95,3)
sage: msm.sigma()
1.0
```

simulation(*n*)

Same as `self.simulations`, but run only 1 time, and returns a time series instead of a list of time series.

INPUT:

- n – a positive integer.

EXAMPLES:

```
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1.0,0.95,3)
sage: m = msm.simulation(5); m  # random
[0.0059, -0.0097, -0.0101, -0.0110, -0.0067]
```

(continues on next page)

(continued from previous page)

```
sage: len(m)
5
sage: m = msm.simulation(3); m  # random
[0.0055, -0.0084, 0.0141]
sage: len(m)
3
```

simulations(*n*, *k=1*)

Return k simulations of length n using this Markov switching multifractal model for n time steps.

INPUT:

- n – positive integer; number of steps.
- k – positive integer (default: 1); number of simulations.

OUTPUT:

list – a list of TimeSeries objects.

EXAMPLES:

```
sage: cad_usd = finance.MarkovSwitchingMultifractal(10,1.278,0.262,0.644,2.11);␣
↪cad_usd
Markov switching multifractal model with m0 = 1.278, sigma = 0.262, b = 2.11,␣
↪and gamma_10 = 0.644
```

CHAPTER

FIVE

MARKOV SWITCHING MULTIFRACTAL MODEL

Cython code

sage.finance.markov_multifractal_cython.**simulations**(*n*, *k*, *m0*, *sigma*, *kbar*, *gamma*)

Return k simulations of length n using the Markov switching multifractal model.

INPUT: n, k – positive integers m0, sigma – floats kbar – integer gamma – list of floats

OUTPUT: list of lists

EXAMPLES:

```
sage: set_random_seed(0)
sage: msm = finance.MarkovSwitchingMultifractal(8,1.4,1.0,0.95,3)
doctest:warning...
DeprecationWarning: the package sage.finance is deprecated...
sage: import sage.finance.markov_multifractal_cython
sage: sage.finance.markov_multifractal_cython.simulations(5,2,1.278,0.262,8,msm.
↪gamma())
[[0.0014, -0.0023, -0.0028, -0.0030, -0.0019], [0.0020, -0.0020, 0.0034, -0.0010, -
↪0.0004]]
```

CHAPTER

SIX

INDICES AND TABLES

- Index
- Module Index
- Search Page

PYTHON MODULE INDEX

f

INDEX