hfhd Documentation

Jan Philipp Woeltjen

Aug 04, 2020

CONTENTS

hfhd is an accelerated Python library for estimating (integrated) covariance matrices with high frequency data in high dimensions. Its main focus lies on the estimation of covariance matrices of financial returns. This is a challenging task since high frequency data are observed irregularly (and non-synchronously across assets) and are contaminated with microstructure noise. The `hf` module provides a collection of tools for synchronization and noise reduction.

When many assets are considered relative to the sample size, we say that the covariance matrix has high dimension. Then, sample eigenvalues are overdispersed relative to the population eigenvalues due to the curse of dimensionality. This overdispersion leads to an ill-conditioned covariance matrix estimate. The `hd` module provides tools to improve the condition of the matrix.

Loss functions in the `loss` module help to judge the performance of covariance estimators. The `sim` module provides a Universe class with which heavy tailed, noisily and irregularly observed, high dimensional asset returns with an underlying factor model can be simulated.

To install hfhd run

```
$ pip install hfhd
```

If you experience problems with the parallelism, you may need to install/upgrade the Threading Building Blocks (TBB). The easiest way is to

```
$ conda install tbb
```

CHAPTER

ONE

THE PRICE PROCESS

The library assumes the following standard model. Let $\left(\Omega, \mathcal{F},\left\{\mathcal{F}_t\right\}_{0 \leq t \leq T}, \mathbb{P}\right)$ be a filtered probability space on which the log-price process of the p assets under study, $\left\{\mathbf{X}_t\right\}_{0 \leq t \leq T}$, is adapted, where $\mathbf{X}_t=\left(X_t^{(1)}, \ldots, X_t^{(p)}\right)^{\prime}$. It is assumed that $\mathbf{X}_t$ follows a diffusion process satisfying $$ d \mathbf{X}_{t}=\boldsymbol{\mu}_{t} d t+\boldsymbol{\sigma}_{t} d \mathbf{W}_{t}, \quad t \in[0,T]. $$

The process $\{\mathbf{W}_t\}$ is a p-dimensional standard Brownian motion. The drift $\boldsymbol{\mu}_t \in \mathbb{R}^p$ and the volatility $\boldsymbol{\sigma}_t \in \mathbb{R}^{p \times p}$ are càdlàg. Analogously to the covariance matrix in the low frequency setting, the integrated covariance matrix over the interval $[a, b] \subset [0, T]$, is defined as $$ \boldsymbol{\Sigma}(a, b)=\int_{a}^{b} \boldsymbol{\sigma}_{t} \boldsymbol{\sigma}_{t}^{\prime} dt $$ The log-price of each asset is observed at discrete times and with market microstructure noise. The set of observation times of asset j is denoted as $t^{(j)}=\{t_0^{(j)} \leq t_1^{(j)} \leq \ldots \leq t_{n^{(j)}}^{(j)}\}$. Importantly, observation times are non-synchronous across assets, i.e., $t^{(j)} \neq t^{(k)}$ for $j \neq k$ in general. Furthermore, the concentration ratio $c_n = p/n$ is not a small number. The observed log-price process of asset j is \begin{equation} \label{eqn:obs} {Y_t^{(j)}}={X_t^{(j)}}+{\epsilon_t^{(j)}}, \quad t \in t^{(j)}, \end{equation} where $\epsilon_t^{(j)}$ is a noise process independent of $\mathbf{X}_t$ with mean 0. $\{\epsilon_t\}_{0 \leq t \leq T}$ is assumed to be adapted to $\{\mathcal{F}_t\}_{0 \leq t \leq T}$. Hence, the observed price process $\left\{Y_t^{(j)}\right\}_{0 \leq t \leq T}$ is also adapted. The microstructure noise is the resulting process of an interplay of many effects. Among them are for example price discreteness and the bid–ask bounce. When the interest lies on the integrated covariance matrix of the underlying return process, it is important to account for the variance due to the noise process. When the observation frequency is high, the variance due to the the noise process dominates the variance of the underlying process and estimators that do not cancel the noise are severely biased.

Note: Accelerated functions are compiled Just In Time (JIT). This may cause the first call to be several orders of magnitude slower than the following calls.

CHAPTER

TWO

DESIGN PHILOSOPHY

hfhd is written with the following priciples in mind:

- Thorough and notationally consistent documentation, explaining not only the code but also the theory behind it.
- The code needs to run sufficiently fast to be usable. Since many functions are inherently iterative, JIT compilation with Numba is used throughout to speed things up.
- The API should be user friendly with consistent inputs and outputs.

2.1 hfhd

2.1.1 hf Module

The hf module provides functions for synchronization of asynchronously observed multivariate time series and observation noise cancelling. When possible, functions are parallelized and accelerated via JIT compilation with Numba. By default all cores of your machine are used. If your pipeline allows for parallelization on a higher level, it is preferable to do so. You may manually set the number of cores used by setting `numba.set_num_threads(n)`. Every estimator takes `tick_series_list` as the first argument. This is a list of pd.Series (one for each asset) containing tick log-prices with pandas.DatetimeIndex. If you want to comute the covariance of residuals after predictions are subtracted from log-returnsjust cumsum the residuals. The output is the integrated covariance matrix estimate as a 2d numpy.ndarray.

Functions

ensemble(estimates, var_weights, cov_weights)	Ensemble multiple covariance matrix estimates with weights given by `var_weights` and `cov_weights` for the diagonal and off-diagonal elements, respectively.
gamma(data, h)	The h-th realized autocovariance.
get_bandwidth(n, var_ret, var_noise, kernel)	Compute the optimal bandwidth parameter $H$ for *krvm()* according to Barndorff-Nielsen et al.
get_cumu_demeaned_resid(price[, y_hat])	From a pd.Series of tick prices and predictions get a pd.Series of tick log-prices with zero-mean returns, i.e.
hayashi_yoshida(tick_series_list[, theta, k])	The (pairwise) Hayashi-Yoshida estimator of Hayashi and Yoshida (2005).
krvm(tick_series_list, H[, pairwise, kernel])	The kernel realized volatility matrix estimator (KRVM) of Barndorff-Nielsen et al.
mrc(tick_series_list[, theta, g, ...])	The modulated realised covariance (MRC) estimator of Christensen et al.

continues on next page

Table 1 – continued from previous page

msrc(tick_series_list[, M, N, pairwise])	The multi-scale realized volatility (MSRV) estimator of Zhang (2006).
parzen_kernel(x)	The Parzen weighting function used in the kernel realized volatility matrix estimator (`krvm()`) of Barndorff-Nielsen et al.
preaverage(data[, K, g, return_K])	The preaveraging scheme of Podolskij and Vetter (2009).
quadratic_spectral_kernel(x)	The Quadratic Spectral weighting function used in the kernel realized volatility matrix estimator (`krvm()`) of Barndorff-Nielsen et.
refresh_time(tick_series_list)	The all-refresh time scheme of Barndorff-Nielsen et al.
tsrc(tick_series_list[, J, K])	The two-scales realized volatility (TSRV) of Zhang et al.

ensemble

`hf.`**`ensemble`**(*estimates*, *var_weights*, *cov_weights*)

Ensemble multiple covariance matrix estimates with weights given by `var_weights` and `cov_weights` for the diagonal and off-diagonal elements, respectively. This function is used in the ensembled pairwise integrated covariance (EPIC) estimator of Woeltjen (2020). The `msrc()` estimator , the `mrc()` estimator, the `krvm()` estimator and the preaveraged `hayashi_yoshida()` estimator are ensembled to compute an improved finite sample estimate of the pairwise integrated covariance matrix. The EPIC estimator uses every available tick, and compares favorable in finite samples to its constituents on their own. The preaveraged HY estimates of the off-diagonals have better finite sample properties than the other estimators so it might be preferable to overweight them by setting the corresponding `cov_weights` element to a number >1/4.

Parameters

estimates [list of numpy.ndarrays with shape = (p, p)] The covariance matrix estimates. var_weights : numpy.ndarray The weights with which the diagonal elements of the MSRC, MRC, and the preaveraged HY covariance estimates are weighted, respectively. The weights must sum to one.

cov_weights [numpy.ndarray] The weights with which the off-diagonal elements of the MSRC, MRC, and the preaveraged HY covariance estimates are weighted, respectively. The HY estimator uses the data more efficiently and thus may deserve a higher weight. The weights must sum to one.

Returns

cov [numpy.ndarray] The ensemble estimate of the integrated covariance matrix.

gamma

`hf.`**`gamma`**(*data*, *h*)

The h-th realized autocovariance.

Parameters

data [numpy.ndarray, shape = (p, n)] An array of synchronized and demeaned log_returns. (e.g. with `refresh_time()`).

h [int] The order of the autocovariance.

Returns

gamma_h [numpy.ndarray, shape = (p, p)] The h-th realized autocovariance matrix.

Notes

The h-th realized autocovariance is given by \begin{equation} \boldsymbol{\gamma}^{(h)}\left(\mathbf{Y}\right)= \sum_{s=h+2}^{n+1}\left(\mathbf{Y}(s)-\mathbf{Y}(s-1)\right) \left(\mathbf{Y}(s-h)-\mathbf{Y}(s-h-1)\right)^{\prime}, \quad h \geq 0 \end{equation} and \begin{equation} \boldsymbol{\gamma}^{(h)}\left(\mathbf{Y}\right)= \boldsymbol{\gamma}^{(-h)}\left(\mathbf{Y}\right)^{\prime}, \quad h < 0, \end{equation} where $\mathbf{Y}$ denotes the synchronized zero-return log-price.

get_bandwidth

hf.**get_bandwidth**(*n*, *var_ret*, *var_noise*, *kernel*)

Compute the optimal bandwidth parameter H for `krvm()` according to Barndorff-Nielsen et al. (2011).

Parameters

n [int >0] The sample size.

var_ret [float > 0] The variance of the efficient return process.

var_noise :float >=0 The variance of the noise process.

Returns

H [int] The bandwidth parameter.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011). Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, Journal of Econometrics 162(2): 149– 169.

get_cumu_demeaned_resid

hf.**get_cumu_demeaned_resid**(*price*, *y_hat=None*)

From a pd.Series of tick prices and predictions get a pd.Series of tick log-prices with zero-mean returns, i.e. the reconstructed log-prices from de-meaned log-return residuals. These log-prices are inputs to the integrated covariance matrix estimators.

Parameters

series [pd.Series] Tick prices of one asset with datetime index.

y_hat [pd.Series] The predictions.

Returns

out [pd.Series] Log-prices corresponding to zero-mean returns.

hayashi_yoshida

`hf.hayashi_yoshida(tick_series_list, theta=None, k=None)`

The (pairwise) Hayashi-Yoshida estimator of Hayashi and Yoshida (2005). This estimtor sums up all products of time-overlapping returns between two assets. This makes it possible to compute unbiased estimates of the integrated covariance between two assets that are sampled non-synchronously. The standard realized covariance estimator is biased toward zero in this case. This is known as the Epps effect. The function is accelerated via JIT compilation with Numba. The preaveraged version handles microstructure noise as shown in Christensen et al. (2010).

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains tick-log-prices of one asset with datetime index.

theta [float, theta>=0, default=None] If `theta=None` and `k` is not specified explicitly, theta will be set to 0. If theta>0, the log-returns are preaveraged with theta and $g(x) = min(x, 1-x)$. Hautsch and Podolskij (2013) suggest values between 0.4 (for liquid stocks) and 0.6 (for less liquid stocks). If `theta=0`, this is the standard HY estimator.

k [int, >=1, default=None] The bandwidth parameter with which to preaverage. Alternative to `theta`. Useful for non-parametric eigenvalue regularization based on sample splitting. When `k=None` and `theta=None`, `k` will be set to 1. If `k=1`, this is the standard HY estimator.

Returns

cov [numpy.ndarray] The pairwise HY estimate of the integrated covariance matrix.

Notes

The estimator is defined as

$$\left\langle X^{(k)}, X^{(l)}\right\rangle_{HY} = \sum_{i=1}^{n^{(k)}} \sum_{i'=1}^{n^{(l)}} \Delta X_{t_i^{(k)}}^{(k)} \Delta X_{t_{i'}^{(l)}}^{(l)} \mathbf{1}_{\left\{\left(t_{i-1}^{(k)}, t_i^{(k)}\right] \cap \left(t_{i'-1}^{(l)}, t_{i'}^{(l)}\right] \neq \emptyset\right\}}, \tag{2.1}$$

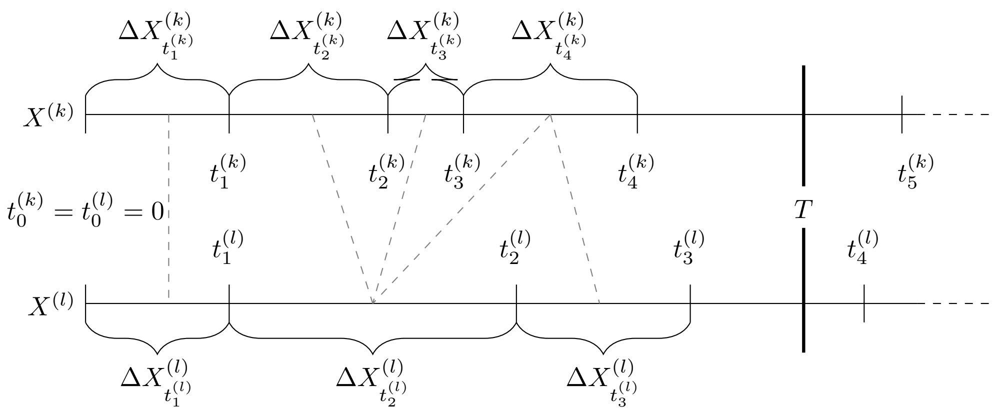
where

$$\Delta X_{t_i^{(j)}}^{(j)} := X_{t_i^{(j)}}^{(j)} - X_{t_{i-1}^{(j)}}^{(j)}$$

denotes the jth asset tick-to-tick log-return over the interval spanned from

$$t_{i-1}^{(j)} \text{ to } t_i^{(j)}, i = 1, \cdots, n^{(j)}.$$

and $n^{(j)} = |t^{(j)}| - 1$ denotes the number of tick-to-tick returns. The following diagram visualizes the products of returns that are part of the sum by the dashed lines.


$\Delta X^{(k)}_{t^{(k)}_1}$
$\Delta X^{(k)}_{t^{(k)}_2}$
$\Delta X^{(k)}_{t^{(k)}_3}$
$\Delta X^{(k)}_{t^{(k)}_4}$
$X^{(k)}$
$t^{(k)}_1$
$t^{(k)}_2$
$t^{(k)}_3$
$t^{(k)}_4$
$t^{(k)}_5$
$t^{(k)}_0 = t^{(l)}_0 = 0$
T
$t^{(l)}_1$
$t^{(l)}_2$
$t^{(l)}_3$
$t^{(l)}_4$
$X^{(l)}$
$\Delta X^{(l)}_{t^{(l)}_1}$
$\Delta X^{(l)}_{t^{(l)}_2}$
$\Delta X^{(l)}_{t^{(l)}_3}$

When returns are preaveraged with `preaverage()`, the HY estimator of can be made robust to microstructure noise as well. It is then of the slightly adjusted form

$$\left\langle X^{(k)}, X^{(l)}\right\rangle^{\theta}_{HY} = \frac{1}{(\psi_{HY}K)^2}\sum_{i=K}^{n^{(k)}}\sum_{i'=K}^{n^{(l)}} \bar{Y}^{(k)}_{t^{(k)}_i}\bar{Y}^{(l)}_{t^{(l)}_{i'}}\mathbf{1}_{\left\{\left(t^{(k)}_{i-K},t^{(k)}_i\right]\cap\left(t^{(l)}_{i'-K},t^{(l)}_{i'}\right]\neq\emptyset\right)} \quad (2.2)$$

where $\psi_{HY} = \frac{1}{K}\sum_{i=1}^{K-1} g\left(\frac{i}{K}\right)$ The preaveraged HY estimator has optimal convergence rate $n^{-1/4}$, where $n = \sum_{j=1}^{p} n^{(j)}$. Christensen et al. (2013) subsequently proof a central limit theorem for this estimator and show that it is robust to some dependence structure of the noise process. Since preaveraging is performed before synchronization, the estimator utilizes more data than other methods that cancel noise after synchronization. In particular, the preaveraged HY estimator even uses the observation $t^{(j)}_2$ in the figure, which does not contribute the the covariance due to the log-summability.

References

Hayashi, T. and Yoshida, N. (2005). On covariance estimation of non-synchronously observed diffusion processes, Bernoulli 11(2): 359–379.

Christensen, K., Kinnebrock, S. and Podolskij, M. (2010). Pre-averaging estimators of the ex-post covariance matrix in noisy diffusion models with non-synchronous data, Journal of Econometrics 159(1): 116–133.

Hautsch, N. and Podolskij, M. (2013). Preaveraging-based estimation of quadratic variation in the presence of noise and jumps: theory, implementation, and empirical evidence, Journal of Business & Economic Statistics 31(2): 165–183.

Christensen, K., Podolskij, M. and Vetter, M. (2013). On covariation estimation for multivariate continuous it^o semimartingales with noise in non-synchronous observation schemes, Journal of Multivariate Analysis 120: 59–84.

Examples

```
>>> np.random.seed(0)
>>> n = 10000
>>> returns = np.random.multivariate_normal([0, 0], [[1,0.5],[0.5,1]], n)/n**0.5
>>> prices = np.exp(returns.cumsum(axis=0))
>>> # sample n/2 (non-synchronous) observations of each tick series
>>> series_a = pd.Series(prices[:, 0]).sample(int(n/2)).sort_index()
>>> series_b = pd.Series(prices[:, 1]).sample(int(n/2)).sort_index()
>>> # take logs
>>> series_a = np.log(series_a)
>>> series_b = np.log(series_b)
>>> icov = hayashi_yoshida([series_a, series_b])
>>> np.round(icov, 3)
array([[0.983, 0.512],
       [0.512, 0.99 ]])
```

krvm

`hf.`**`krvm`**(*tick_series_list*, *H*, *pairwise=True*, *kernel=CPUDispatcher(<function quadratic_spectral_kernel at 0x7f247c865a60>)*)

The kernel realized volatility matrix estimator (KRVM) of Barndorff-Nielsen et al. (2011).

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains tick-log-prices of one asset with datetime index.

H [int, > 0] The bandwidth parameter for the Parzen kernel. Should be on the order of $n^{3/5}$.

pairwise [bool, default=True] If `True` the estimator is applied to each pair individually. This increases the data efficiency but may result in an estimate that is not p.s.d even for the p.s.d version of thiss estimator.

kernel [function, default=quadratic_spectral_kernel] The kernel weighting function.

Returns

cov [numpy.ndarray] The intgrated covariance matrix estimate.

Notes

The multivariate realized kernel estimator smoothes the autocovariance operator and thereby achieves the optimal convergence rate in the multivariate setting with noise and asynchronous observation times. Incidentally, this estimator is similar in form to the HAC, widely used in the statistics and econometrics literature to deal with heteroscedastic and autocorrelated noise. Observations are synchronized with the `refresh-time()` scheme. In addition, m observation are averaged at the beginning and at the end of the trading day to estimate the efficient price at these times. The authors call this 'jittering'. In practice the effect of jittering is negligible but it is needed for proving consistency. (It is ignored in this implementation.) The, with parameter m, jittered log-price vectors are denoted as $\mathbf{Y}^{(m)}(s), s = 1, \ldots, n-2m+1$. The kernel estimator is defined by \begin{equation} \widehat{\mathbf{\Sigma}}^{(KRVM)}=\boldsymbol{\gamma}^{(0)} \left(\mathbf{Y}^{(m)}\right)+\sum_{h=1}^{n-2 m} k\left(\frac{h-1}{H} \right)\left[\boldsymbol{\gamma}^{(h)}\left(\mathbf{ \boldsymbol{\gamma}^{(-h)}\left(\mathbf{Y}^{(m)}\right)\right], \end{equation} where \begin{equation} \boldsymbol{\gamma}^{(h)}\left(\mathbf{Y}\right)= \sum_{s=h+2}^{n+1}\left(\mathbf{Y}(s)-\mathbf{Y}(s-1)\right) \left(\mathbf{Y}(s-h)-\mathbf{Y}(s-h-1)\right)^{\prime}, \quad h \geq 0 \end{equation} and \begin{equation} \boldsymbol{\gamma}^{(h)}\left(\mathbf{Y}\right)=

\boldsymbol{\gamma}^{(-h)}\left(\mathbf{Y}\right)^{\prime}, \quad h < 0, \end{equation} with $\mathbf{Y}$ denoting the synchronized zero-return log-price. $\gamma^{(h)}$ is the hth realized autocovariance (`gamma()`). $k(\cdot)$ is the kernel function with its bandwidth parameter H. It is assumed that (i) $k(0) = 1$ and $k'(0) = 0$, (ii) $k(\cdot)$ is twice differentiable with continuous derivatives, and (iii) $\int_0^\infty k(x)^2 dx, \int_0^\infty k'(x)^2 dx$ and $\int_0^\infty k''(x)^2 dx$ are finite. A slightly adjusted form of this estimator that is positive semidefinite is given by \begin{equation} \widehat{\mathbf{\Sigma}}^{(KRVM_{psd})}=\boldsymbol{\gamma}^{(0)} \left(\mathbf{Y}^{(m)}\right)+\sum_{h=1}^{n-2 m} k\left(\frac{h}{H}\right) \left[\boldsymbol{\gamma}^{(h)}\left(\mathbf{Y}^{(m)}\right)+ \boldsymbol{\gamma}^{(-h)}\left(\mathbf{Y}^{(m)}\right)\right]. \end{equation} This form requires the additional assumption $\int_{-\infty}^\infty k(x)\exp(ix\lambda)dx \geq 0$ for all $\lambda \in \mathbb{R}$.

Choosing the right kernel function is important. The authors show, for example, that the estimator based on the Bartlett weight function is inconsistent. Instead, the Parzen kernel (`parzen_kernel()`) is suggested as a weight function that yields a consistent estimator and can be efficiently implemented. The bandwidth H must be on the order of $n^{3/5}$. The authors choose the scalar H as the average of optimal individual $H^{(j)}$: $$\bar{H}=p^{-1} \sum_{j=1}^{p} H^{(j)},$$ where \begin{equation} H^{(j)}=c^{*} \xi_{j}^{4 / 5} n^{3 / 5}, \end{equation} with \begin{equation} c^{*}=\left\{k^{\prime \prime}(0)^{2} / k_{\bullet}^{0,0}\right\}^{1 / 5}, \end{equation} and \begin{equation} \xi_{j}^{2}={\Sigma}_{\epsilon, j j} / {\Sigma}_{j j}. \end{equation} $\mathbf{\Sigma}_\epsilon$ and $\mathbf{\Sigma}$ denote, as previously defined, the integrated covariance matrix of the noise and the efficient return process, respectively. Here these quantities are understood over the interval under consideration. Hence, ξ_j^2 can be interpreted as the ratio of the noise variance and the return variance. For the Parzen kernel $c^* = 3.51$, as tabulated by the authors. It is a measure of the relative asymptotic efficiency of the kernel. Σ_{jj} may be estimated via a low frequency estimator and $\Sigma_{\epsilon,jj}$ via a high frequency estimator.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011). Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, Journal of Econometrics 162(2): 149– 169.

mrc

`hf.`**`mrc`**(*tick_series_list*, *theta=None*, *g=None*, *bias_correction=True*, *pairwise=True*, *k=None*)

The modulated realised covariance (MRC) estimator of Christensen et al. (2010).

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains tick-log-prices of one asset with datetime index.

theta [float, optional, default=None] Theta is used to determine the preaveraging window `k`. If `bias_correction` is True (see below) then $k = \theta\sqrt{n}$, else $k = \theta n^{1/2+0.1}$. Hautsch & Podolskij (2013) recommend 0.4 for liquid assets and 0.6 for less liquid assets. If `theta=0`, the estimator reduces to the standard realized covariance estimator. If `theta=None` and `k` is not specified explicitly, the suggested theta of 0.4 is used.

g [function, optional, default = `None`] A vectorized weighting function. If `g = None`, $g = min(x, 1-x)$

bias_correction [boolean, optional] If `True` (default) then the estimator is optimized for convergence rate but it might not be p.s.d. Alternatively as described in Christensen et al. (2010) it can be ommited. Then k should be chosen larger than otherwise optimal.

pairwise [bool, default=True] If `True` the estimator is applied to each pair individually. This increases the data efficiency but may result in an estimate that is not p.s.d.

k [int, optional, default=None] The bandwidth parameter with which to preaverage. Alternative to theta. Useful for non-parametric eigenvalue regularization based on sample spliting.

Returns

mrc [numpy.ndarray] The mrc estimate of the integrated covariance.

Notes

The MRC estimator is the equivalent to the realized integrated covariance estimator using preaveraged returns. It is of thus of the form

$$[\mathbf{Y}]^{(\text{MRC})}=\frac{n}{n-K+2}\frac{1}{\psi_2 K}\sum_{i=K-1}^{n}\bar{\mathbf{Y}}_i\bar{\mathbf{Y}}_i', (2.3)$$

where $\frac{n}{n-K+2}$ is a finite sample correction, and

$$\begin{gathered}\psi_1^k=k\sum_{i=1}^{k}\left(g\left(\frac{i}{k}\right)-g\left(\frac{i-1}{k}\right)\right)^2\\ \psi_2^k=\frac{1}{k}\sum_{i=1}^{k-1}g^2\left(\frac{i}{k}\right).\end{gathered} \quad (2.4)$$

$$\psi_2^k=\frac{1}{k}\sum_{i=1}^{k-1}g^2\left(\frac{i}{k}\right).$$

In this form, however, the estimator is biased. The bias corrected estimator is given by

$$[\mathbf{Y}]^{(\text{MRC})}=\frac{n}{n-K+2}\frac{1}{\psi_2 k}\sum_{i=K-1}^{n}\bar{\mathbf{Y}}_i\left(\bar{\mathbf{Y}}_i-\frac{\psi_1}{\theta^2\psi_2}\hat{\mathbf{\Psi}}\right)',$$

where

$$\hat{\mathbf{\Psi}}=\frac{1}{2n}\sum_{i=1}^{n}\Delta_i\mathbf{Y}\left(\Delta_i\mathbf{Y}\right)'.$$

The rate of convergence of this estimator is determined by the window-length K. Choosing $K = \mathcal{O}(\sqrt{n})$, delivers the best rate of convergence of $n^{-1/4}$. It is thus suggested to choose $K = \theta\sqrt{n}$, where θ can be calibrated from the data. Hautsch and Podolskij (2013) suggest values between 0.4 (for liquid stocks) and 0.6 (for less liquid stocks).

Note: The bias correction may result in an estimate that is not positive semi-definite.

If positive semi-definiteness is essential, the bias-correction can be omitted. In this case, K should be chosen larger than otherwise optimal with respect to the convergence rate. Of course, the convergence rate is slower then. The optimal rate of convergence without the bias correction is $n^{-1/5}$, which is attained when $K = \theta n^{1/2+\delta}$ with $\delta = 0.1$.

`theta` should be chosen between 0.3 and 0.6. It should be chosen higher if (i) the sampling frequency declines, (ii) the trading intensity of the underlying stock is low, (iii) transaction time sampling (TTS) is used as opposed to calendar time sampling (CTS). A high `theta` value can lead to oversmoothing when CTS is used. Generally the higher the sampling frequency the better. Since *mrc()* and *msrc()* are based on different approaches it might make sense to ensemble them. Monte Carlo results show that the variance estimate of the ensemble is better than each component individually. For covariance estimation the preaveraged *hayashi_yoshida()* estimator has the advantage that even ticks that don't contribute to the covariance (due to log-summability) are used for smoothing. It thus uses the data more efficiently.

References

Christensen, K., Kinnebrock, S. and Podolskij, M. (2010). Pre-averaging estimators of the ex-post covariance matrix in noisy diffusion models with non-synchronous data, Journal of Econometrics 159(1): 116–133.

Hautsch, N. and Podolskij, M. (2013). Preaveraging-based estimation of quadratic variation in the presence of noise and jumps: theory, implementation, and empirical evidence, Journal of Business & Economic Statistics 31(2): 165–183.

Examples

```
>>> np.random.seed(0)
>>> n = 2000
>>> returns = np.random.multivariate_normal([0, 0], [[1, 0.5],[0.5, 1]], n)
>>> returns /=  n**0.5
>>> prices = 100 * np.exp(returns.cumsum(axis=0))
>>> # add Gaussian microstructure noise
>>> noise = 10 * np.random.normal(0, 1, n * 2).reshape(-1, 2)
>>> noise *= np.sqrt(1 / n ** 0.5)
>>> prices += noise
>>> # sample n/2 (non-synchronous) observations of each tick series
>>> series_a = pd.Series(prices[:, 0]).sample(int(n/2)).sort_index()
>>> series_b = pd.Series(prices[:, 1]).sample(int(n/2)).sort_index()
>>> # take logs
>>> series_a = np.log(series_a)
>>> series_b = np.log(series_b)
>>> icov_c = mrc([series_a, series_b], pairwise=False)
>>> # This is the unbiased, corrected integrated covariance matrix estimate.
>>> np.round(icov_c, 3)
array([[0.882, 0.453],
       [0.453, 0.934]])
>>> # This is the unbiased, corrected realized variance estimate.
```

(continues on next page)

(continued from previous page)

```
>>> ivar_c = mrc([series_a], pairwise=False)
>>> np.round(ivar_c, 3)
array([[0.894]])
>>> # Use ticks more efficiently by pairwise estimation
>>> icov_c = mrc([series_a, series_b], pairwise=True)
>>> np.round(icov_c, 3)
array([[0.894, 0.453],
       [0.453, 0.916]])
```

msrc

hf.**msrc**(*tick_series_list*, *M=None*, *N=None*, *pairwise=True*)

The multi-scale realized volatility (MSRV) estimator of Zhang (2006). It is extended to multiple dimensions following Zhang (2011). If `pairwise=True` estimate correlations with pairwise-refresh time previous ticks and variances with all available ticks for each asset.

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains tick-log-prices of one asset with datetime index. Must not contain nans.

M [int, >=1, default=None] The number of scales If `M=None` all scales $i = 1, ..., M$ are used, where M is chosen $M = n^{1/2}$ acccording to Eqn (34) of Zhang (2006).

N [int, >=0, default=None] The constant N of Tao et al. (2013) If `N=None` $N = n^{1/2}$. Lam and Qian (2019) need $N = n^{2/3}$ for non-sparse integrated covariance matrices, in which case the rate of convergence reduces to $n^{1/6}$.

pairwise [bool, default=True] If `True` the estimator is applied to each pair individually. This increases the data efficiency but may result in an estimate that is not p.s.d.

Returns

out [numpy.ndarray] The mrc estimate of the integrated covariance matrix.

Notes

Realized variance estimators based on multiple scales exploit the fact that the proportion of the observed realized variance over a specified interval due to microstructure noise increases with the sampling frequency, while the realized variance of the true underlying process stays constant. The bias can thus be corrected by subtracting a high frequency estimate, scaled by an optimal weight, from a medium frequency estimate. The weight is chosen such that the large bias in the high frequency estimate, when scaled by the weight, is exactly equal to the medium bias, and they cancel each other out as a result.

By considering M time scales, instead of just two as in `tsrc()`, Zhang2006 improves the rate of convergence to $n^{-1/4}$. This is the best attainable rate of convergence in this setting. The proposed multi-scale realized volatility (MSRV) estimator is defined as \begin{equation} \langle\widehat{X^{(j)}, X^{(j)}}\rangle^{(MSRV)}_T=\sum_{i=1}^{M} \alpha_{i}[Y^{(j)}, Y^{(j)}]^{\left(K_{i}\right)}_T \end{equation} where α_i are weights satisfying \begin{equation} \begin{aligned} &\sum \alpha_{i}=1\ &\sum_{i=1}^{M}\left(\alpha_{i} / K_{i}\right)=0 \end{aligned} \end{equation} The optimal weights for the chosen number of scales M, i.e., the weights that minimize the noise variance contribution, are given by \begin{equation} a_{i}=\frac{K_{i}\left(K_{i}-\bar{K}\right)} {M \operatorname{var}\left(K\right)}, \end{equation} where $\%\bar{K}$ denotes the mean of K. $$\bar{K}=\frac{1}{M} \sum_{i=1}^{M} K_{i} \quad \text { and } \quad \operatorname{var}\left(K\right)=\frac{1}{M} \sum_{i=1}^{M} K_{i}^{2}-\bar{K}^{2}.

$$ If all scales are chosen, i.e., $K_i = i$, for $i = 1, \ldots, M$, then $\bar{K} = (M+1)/2$ and $\operatorname{var}(K) = \left(M^2 - 1\right)/12$, and hence

\begin{equation} a_{i}=12 \frac{i}{M^{2}} \frac{i / M-1 / 2-1 / \left(2 M\right)}{1-1 / M^{2}}. \end{equation} In this case, as shown by the author in Theorem 4, when M is chosen optimally on the order of $M = \mathcal{O}(n^{1/2})$, the estimator is consistent at rate $n^{-1/4}$.

References

Zhang, L. (2006). Efficient estimation of stochastic volatility using noisy observations: A multi-scale approach, Bernoulli 12(6): 1019–1043.

Zhang, L. (2011). Estimating covariation: Epps effect, microstructure noise, Journal of Econometrics 160.

Examples

```
>>> np.random.seed(0)
>>> n = 200000
>>> returns = np.random.multivariate_normal([0, 0], [[1,0.5],[0.5,1]], n)/n**0.5
>>> prices = 100*np.exp(returns.cumsum(axis=0))
>>> # add Gaussian microstructure noise
>>> noise = 10*np.random.normal(0, 1, n*2).reshape(-1, 2)*np.sqrt(1/n**0.5)
>>> prices +=noise
>>> # sample n/2 (non-synchronous) observations of each tick series
>>> series_a = pd.Series(prices[:, 0]).sample(int(n/2)).sort_index()
>>> series_b = pd.Series(prices[:, 1]).sample(int(n/2)).sort_index()
>>> # get log prices
>>> series_a = np.log(series_a)
>>> series_b = np.log(series_b)
>>> icov = msrc([series_a, series_b], M=1, pairwise=False)
>>> icov_c = msrc([series_a, series_b])
>>> # This is the biased, uncorrected integrated covariance matrix estimate.
>>> np.round(icov, 3)
array([[11.553,  0.453],
       [ 0.453,  2.173]])
>>> # This is the unbiased,  corrected integrated covariance matrix estimate.
>>> np.round(icov_c, 3)
array([[0.985, 0.392],
       [0.392, 1.112]])
```

parzen_kernel

`hf.`**`parzen_kernel`**`(`*x*`)`

The Parzen weighting function used in the kernel realized volatility matrix estimator (*`krvm()`*) of Barndorff-Nielsen et al. (2011).

Parameters

x [float]

Returns

y [float] The weight.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011). Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, Journal of Econometrics 162(2): 149– 169.

preaverage

`hf.`**`preaverage`**(*data*, *K=None*, *g=None*, *return_K=False*)

The preaveraging scheme of Podolskij and Vetter (2009). It uses the fact that if the noise is i.i.d with zero mean, then averaging a rolling window of (weighted) returns diminishes the effect of microstructure noise on the variance estimate.

Parameters

data [pd.Series or pd.DataFrame] A time series of log-returns. If multivariate, the time series has to be synchronized (e.g. with `refresh_time()`).

K [int, default = `None`] The preaveraging window length. `None` implies $K = 0.4n^{1/2}$ is chosen as recommended in Hautsch & Podolskij (2013).

g [function, default = `None`] A weighting function. `None` implies $g(x) = min(x, 1-x)$ is chosen.

Returns

data_pa [pd.Series] The preaveraged log-returns.

Notes

The preaveraged log-returns using the window-length K are given by

$$\bar{\mathbf{Y}}_i = \sum_{j=1}^{K-1} g\left(\frac{j}{K}\right) \Delta_{i-j+1}\mathbf{Y}, \quad \text{for } i = K, \ldots, n,$$

where $\mathbf{Y}_i$ have been synchronized beforehand, for example with `refresh_time()`. Note that the direction of the moving window has been reversed compared to the definition in Podolskij and Vetter (2009) to stay consistent within the package. g is a weighting function. A popular choice is

$$g(x) = \min(x, 1-x).$$

References

Podolskij, M., Vetter, M., 2009. Estimation of volatility functionals in the simultaneous presence of microstructure noise and jumps. Bernoulli 15 (3), 634–658.

quadratic_spectral_kernel

`hf.`**`quadratic_spectral_kernel`**(*x*)

The Quadratic Spectral weighting function used in the kernel realized volatility matrix estimator (*krvm()*) of Barndorff-Nielsen et. al (2011).

Parameters

x [float]

Returns

y [float] The weight.

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011). Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, Journal of Econometrics 162(2): 149– 169.

refresh_time

`hf.`**`refresh_time`**(*tick_series_list*)

The all-refresh time scheme of Barndorff-Nielsen et al. (2011). If this function is applied to two assets at a time, it becomes the pairwise-refresh time. The function is accelerated via JIT compilation with Numba.

Parameters

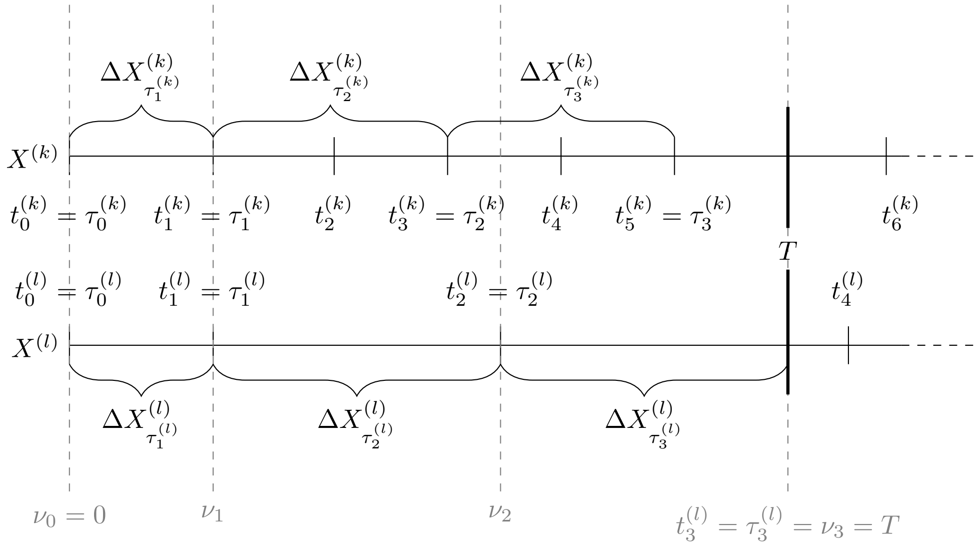
tick_series_list [list of pd.Series] Each pd.Series contains tick prices of one asset with datetime index.

Returns

out [pd.DataFrame] Synchronized previous ticks according to the refresh-time scheme.

Notes

Multivariate estimators require synchronization of the time series. This can be achieved via a grid. A grid is a subset of $[0, T]$ and it is defined as \begin{equation} \mathcal{V}=\left\{v_{0}, v_{1}, \ldots, v_{\tilde{n}}\right\}\subset[0, T] \end{equation} with $v_0 = 0$ and $v_{\tilde{n}} = T$, where $\tilde{n}$ is the sampling frequency, i.e., the number of grid intervals. Two prominent ways to specify the grid are (i) a regular grid, where $v_m - v_{m-1} = \Delta v,$ for $m = 1, \ldots, \tilde{n}$, and (ii) a grid based on 'refresh times' of Barndorff et al. (2011), where the grid spacing is dependent on the observation times. If more than two assets are considered, refresh times can be further classified into 'all-refresh-times' and 'pairwise-refresh times'. Estimators based on pairwise-refresh times use the data more efficiently but the integrated covariance matrix estimate might not be positive definite. The pairwise-refresh time $\mathcal{V}_p = \{v_0, v_1, \ldots, v_{\tilde{n}}\}$ can be obtained by setting $v_0 = 0$, and \begin{equation} v_{m}=\max \left\{\min \left\{t^{(k)}_i \in t^{(k)}: t^{(k)}_i > v_{m-1}\right\},\min \left\{t^{(l)}_i \in t^{(l)}: t^{(l)}_i > v_{m-1}\right\}\right\} \end{equation} where $\tilde{n}$ is the total number of refresh times in the interval $(0, 1]$. This scheme is illustrated in the figure. The procedure has to be repeated for every asset pair. In contrast, the all-refresh time scheme uses a single grid for all assets, which is determined based on the trade time of the slowest asset of each grid interval. Hence, the spacing of grid elements can be much wider. This implies that estimators based on the latter scheme may discard a large proportion of the data, especially if there is a very slowly trading asset. In any case, there has to be at least one observation time of each asset between any two grid elements. With that condition in mind, the 'previous tick time' of asset j is defined as \begin{equation} \tau^{(j)}_m=\max \left\{ t^{(j)}_i \in t^{(j)}: t^{(j)}_i \leq v_{m}\right\} \end{equation} The following diagram illustrates the scheme for two assets, k and l.


$\Delta X^{(k)}_{\tau^{(k)}_1}$
$\Delta X^{(k)}_{\tau^{(k)}_2}$
$\Delta X^{(k)}_{\tau^{(k)}_3}$
$X^{(k)}$
$t^{(k)}_0 = \tau^{(k)}_0$
$t^{(k)}_1 = \tau^{(k)}_1$
$t^{(k)}_2$
$t^{(k)}_3 = \tau^{(k)}_2$
$t^{(k)}_4$
$t^{(k)}_5 = \tau^{(k)}_3$
$t^{(k)}_6$
T
$t^{(l)}_0 = \tau^{(l)}_0$
$t^{(l)}_1 = \tau^{(l)}_1$
$t^{(l)}_2 = \tau^{(l)}_2$
$t^{(l)}_4$
$X^{(l)}$
$\Delta X^{(l)}_{\tau^{(l)}_1}$
$\Delta X^{(l)}_{\tau^{(l)}_2}$
$\Delta X^{(l)}_{\tau^{(l)}_3}$
$\nu_0 = 0$
ν_1
ν_2
$t^{(l)}_3 = \tau^{(l)}_3 = \nu_3 = T$

References

Barndorff-Nielsen, O. E., Hansen, P. R., Lunde, A. and Shephard, N. (2011). Multivariate realised kernels: consistent positive semi-definite estimators of the covariation of equity prices with noise and non-synchronous trading, Journal of Econometrics 162(2): 149–169.

Examples

```
>>> np.random.seed(0)
>>> n = 20
>>> returns = np.random.multivariate_normal([0, 0], [[1,0.5],[0.5,1]], n)/n**0.5
>>> prices = np.exp(returns.cumsum(axis=0))
>>> # sample n/2 (non-synchronous) observations of each tick series
>>> series_a = pd.Series(prices[:, 0]).sample(int(n/2)).sort_index().rename('a')
>>> series_b = pd.Series(prices[:, 1]).sample(int(n/2)).sort_index().rename('b')
>>> previous_ticks = refresh_time([series_a, series_b])
>>> np.round(previous_ticks.values,4)
array([[0.34  , 0.4309],
       [0.2317, 0.4313],
       [0.1744, 0.4109],
       [0.1336, 0.3007],
       [0.1383, 0.4537],
       [0.1292, 0.1665],
       [0.0936, 0.162 ]])
```

tsrc

`hf.tsrc(tick_series_list, J=1, K=None)`

The two-scales realized volatility (TSRV) of Zhang et al. (2005). It is extentended to handle multiple dimension according to Zhang (2011). `msrc()` has better convergence rate and is thus prefrerred.

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains tick-log-prices of one asset with datetime index.

K [int, default = `int(n**(2/3))`] long scale, default = `int(n**(2/3))` as per Zhang (2005)

J [int, default = 1] short scale

Returns

out [numpy.ndarray] The TSRV estimate.

Notes

The two-scales realized volatility (TSRV) estimator is defined as

\begin{equation} \widehat{\langle X^{(j)}, X^{(j)}\rangle}^{(\mathrm{TSRV})}_{T}= \left[Y^{(j)}, Y^{(j)}\right]_{T}^{(K)}-\frac{\bar{n}_{K}}{\bar{n}_{J}} \left[Y^{(j)}, Y^{(j)}\right]_{T}^{(J)}, \end{equation} where \begin{equation} \left[Y^{(j)}, Y^{(j)}\right]_{T}^{(K)}=\frac{1}{K} \sum_{i=K}^{n}\left(Y_{\tau_{i}^{(j)}}^{(j)}- Y_{\tau_{i-K}^{(j)}}^{(j)}\right)^2, \end{equation} with K being a positive integer usually chosen much larger than 1 and $\bar{n}_K = (n-K+1)/K$ and $\bar{n}_J = (n-J+1)/J$. If K is chosen on the order of$K = \mathcal{O}\left(n^{2/3}\right)$ this estimator is asymptotically unbiased, consistent, asymptotically normal distributed and converges at rate $n^{-1/6}$.

Zhang (2011) proposes the (multivariate) two scales realized covariance (TSCV) estimator based on previous-tick times of asset k and l, which simultaneously corrects for the bias due to asynchronicity and the bias due to microstructure noise. Previous-tick times may be computed via `refresh_time()`.

The TSCV estimator is defined as \begin{equation} \widehat{\langle X^{(k)},X^{(l)}\rangle}_{T}^{(TSCV)}=c\left(\left[Y^{(k)}, Y^{(l)}\right]_{T}^{(K)}-\frac{\bar{n}_{K}}{\bar{n}_{J}}\left[Y^{(k)}, Y^{(l)}\right]_{T}^{(J)}\right), \end{equation} where \begin{equation} \left[Y^{(k)}, Y^{(l)}\right]_{T}^{(K)}=\frac{1}{K} \sum_{i=K}^{\tilde{n}}\left(Y^{(k)}_{\tau^{(k)}_{i}}-Y^{(k)}_{\tau^{(k)}_{i-K}} \right)\left(Y^{(l)}_{\tau^{(l)}_{i}}-Y^{(l)}_{\tau^{(l)}_{i-K}}\right) \end{equation} $c = 1 + o_p\left(\tilde{n}^{-1/6}\right)$ is a small sample correction. K is again a positive integer usually chosen much larger than 1 and $\bar{n}_K = (\tilde{n}-K+1)/K$ and $\bar{n}_J = (\tilde{n}-J+1)/J$. The author shows that if $K = \mathcal{O}\left((n^{(k)} + n^{(l)})^{2/3}\right)$ this estimator is asymptotically unbiased, consistent, asymptotically normal distributed and converges at rate $\tilde{n}^{-1/6}$.

Note: Use `msrc()` since it has better converges rate.

References

Zhang, L., Mykland, P. A. and Ait-Sahalia, Y. (2005). A tale of two time scales: Determining integrated volatility with noisy high-frequency data, Journal of the American Statistical Association 100(472): 1394–1411.

Zhang, L. (2011). Estimating covariation: Epps effect, microstructure noise, Journal of Econometrics 160.

Examples

```
>>> np.random.seed(0)
>>> n = 200000
>>> returns = np.random.multivariate_normal([0, 0], [[1, 0.5],[0.5, 1]], n)/n**0.5
>>> prices = 100*np.exp(returns.cumsum(axis=0))
>>> # add Gaussian microstructure noise
>>> noise = 10*np.random.normal(0, 1, n*2).reshape(-1, 2)*np.sqrt(1/n**0.5)
>>> prices += noise
>>> # sample n/2 (non-synchronous) observations of each tick series
>>> series_a = pd.Series(prices[:, 0]).sample(int(n/2)).sort_index()
>>> series_b = pd.Series(prices[:, 1]).sample(int(n/2)).sort_index()
>>> # take logs
>>> series_a = np.log(series_a)
>>> series_b = np.log(series_b)
>>> icov_c = tsrc([series_a, series_b])
>>> # This is the unbiased,  corrected integrated covariance matrix estimate.
>>> np.round(icov_c, 3)
array([[0.995, 0.361],
       [0.361, 0.977]])
```

2.1.2 hd Module

This module provides functions to estimate covariance matrices in high dimension, i.e., when the concentration ratio is not small or even greater than one.

Functions

fsopt(S, sigma)	The infeasible finite sample optimal rotation equivariant covariance matrix estimator of Ledoit and Wolf (2018).
linear_shrink_target(cov, target[, step, . . .])	Linearly shrink a covariance matrix until a condition number target is reached.
linear_shrinkage(X)	The linear shrinkage estimator of Ledoit and Wolf (2004).
nercome(X[, m, M])	The nonparametric eigenvalue-regularized covariance matrix estimator (NERCOME) of Lam (2016).
nerive(tick_series_list[, stp, estimator])	The nonparametric eigenvalue-regularized integrated covariance matrix estimator (NERIVE) of Lam and Feng (2018).
nonlinear_shrinkage(X)	Compute the shrunk sample covariance matrix with the analytic nonlinear shrinkage formula of Ledoit and Wolf (2018).
to_corr(cov)	Convert a covariance matrix to a correlation matrix.

fsopt

`hd.`**`fsopt`**(*S*, *sigma*)

The infeasible finite sample optimal rotation equivariant covariance matrix estimator of Ledoit and Wolf (2018).

Parameters

S [numpy.ndarray] The sample covariance matrix.

sigma [numpy.ndarray] The (true) population covariance matrix.

Returns

out [numpy.ndarray] The finite sample optimal rotation equivariant covariance matrix estimate.

Notes

This estimator is given by

$$S_n^* := \sum_{i=1}^{p} d_{n,i}^* \cdot u_{n,i} u_{n,i}' = \sum_{i=1}^{p} \left(u_{n,i}' \Sigma_n u_{n,i} \right) \cdot u_{n,i} u_{n,i}',$$

where $[u_{n,1} \dots u_{n,p}]$ are the sample eigenvectors.

References

Ledoit, O. and Wolf, M. (2018). Analytical nonlinear shrinkage of large-dimensional covariance matrices, University of Zurich, Department of Economics, Working Paper (264).

linear_shrink_target

`hd.`**`linear_shrink_target`**(*cov*, *target*, *step=0.05*, *max_iter=100*)

Linearly shrink a covariance matrix until a condition number target is reached. Useful for reducing the impact of outliers in `nerive()`.

Parameters

cov [numpy.ndarray, shape = (p, p)] The covariance matrix.

target: float > 1 The highest acceptable condition number.

step [float > 0] The linear shrinkage parameter for each step.

max_iter [int > 1] The maximum number of iterations until giving up.

Returns

cov [numpy.ndarray, shape = (p, p)] The linearly shrunk covariance matrix estimate.

linear_shrinkage

`hd.linear_shrinkage(X)`

The linear shrinkage estimator of Ledoit and Wolf (2004). The observations need to be synchronized and i.i.d.

Parameters

X [numpy.ndarray, shape = (p, n)] A sample of synchronized log-returns.

Returns

shrunk_cov [numpy.ndarray] The linearly shrunk covariance matrix.

Notes

The most ubiquitous estimator of the rotation equivariant type is perhaps the linear shrinkage estimator of Ledoit and Wolf (2004). These authors propose a weighted average of the sample covariance matrix and the identity (or some other highly structured) matrix that is scaled such that the trace remains the same. The weight, or shrinkage intensity, ρ is chosen such that the squared Frobenius loss is minimized by inducing bias but reducing variance. In Theorem 1, the authors show that the optimal ρ is given by \begin{equation} \rho=\beta^{2} /\left(\alpha^{2}+\beta^{2}\right)=\beta^{2} / \delta^{2}, \end{equation} where $\mu = \mathrm{tr}(\mathbf{\Sigma})/p$, $\alpha^2 = \|\mathbf{\Sigma} - \mu\mathbf{I}_p\|_F^2$, $\beta^2 = E\left[\|\mathbf{S} - \mathbf{\Sigma}\|_F^2\right]$, and $\delta^2 = E\left[\|\mathbf{S} - \mu\mathbf{I}_p\|_F^2\right]$. β^2/δ^2 can be interpreted as a normalized measure of the error of the sample covariance matrix. Shrinking the covariance matrix towards the μ-scaled identity matrix has the effect of pulling the eigenvalues towards μ. This reduces eigenvalue dispersion. In other words, the elements of the diagonal matrix in the rotation equivariant form are chosen as \begin{equation} \widehat{\mathbf{\delta}}^{(l, o)}:=\left(\widehat{d}_{1}^{(l, o)}, \ldots, \widehat{d}_{p}^{(l, o)}\right)=\left(\rho \mu+(1-\rho) \lambda_{1}, \ldots, \rho \mu+(1-\rho) \lambda_{p}\right). \end{equation} In the current form it is still an oracle estimator since it depends on] unobservable quantities. To make it a feasible estimator, these quantities have to be estimated. To this end, define the grand mean as \begin{equation} \label{eqn: grand mean} \bar{\lambda}:=\frac{1}{p} \sum_{i=1}^{p} \lambda_{i}, \end{equation} The estimator for ρ is given by \begin{equation} \widehat{\rho} : = \frac{b^{2}}{d^{2}}, \end{equation} where $d^2 = \left\|\mathbf{S} - \bar{\lambda}\mathbf{I}_p\right\|_F^2$ and $b^2 = \min\left(\bar{b}^2, d^2\right)$ with $\bar{b}^2 = \frac{1}{n^2}\sum_{k=1}^{n}\left\|\mathbf{x}_k\left(\mathbf{x}_k\right)' - \mathbf{S}\right\|_F^2$, where $\mathbf{x}_k$ denotes the kth column of the observation matrix $\mathbf{X}$ for $k = 1, \ldots, n$. In order for this estimator to be consistent, the assumption that $\mathbf{X}$ is i.i.d with finite fourth moments must be satisfied. The feasible linear shrinkage estimator is then of form rotation equivatiant form with the elements of the diagonal chosen as \begin{equation} \widehat{\mathbf{\delta}}^{(l)}:=\left(\widehat{d}_{{1}}^{(l)}, \ldots, \widehat{d}_{p}^{(l)}\right)=\left(\widehat{\rho} \bar{\lambda}+ (1- \widehat{\rho}) \lambda_{1}, \ldots, \widehat{\rho} \bar{\lambda}+ (1- \widehat{\rho}) \lambda_{p}\right). \end{equation} Hence, the linear shrinkage estimator is given by \begin{equation} \widehat{\mathbf{S}}:=\sum_{i=1}^{p} \widehat{d}_{i}^{(l)} \mathbf{u}_{i} \mathbf{u}_{i}^{\prime} \end{equation} The result is a biased but well-conditioned covariance matrix estimate.

References

Ledoit, O. and Wolf, M. (2004). A well-conditioned estimator for large-dimensional covariance matrices, Journal of Multivariate Analysis 88(2): 365–411.

Examples

```
>>> np.random.seed(0)
>>> n = 5
>>> p = 5
>>> X = np.random.multivariate_normal(np.zeros(p), np.eye(p), n)
>>> cov = linear_shrinkage(X.T)
>>> cov
array([[1.1996234, 0.       , 0.       , 0.       , 0.       ],
       [0.       , 1.1996234, 0.       , 0.       , 0.       ],
       [0.       , 0.       , 1.1996234, 0.       , 0.       ],
       [0.       , 0.       , 0.       , 1.1996234, 0.       ],
       [0.       , 0.       , 0.       , 0.       , 1.1996234]])
```

nercome

hd.**nercome**(*X*, *m=None*, *M=50*)

The nonparametric eigenvalue-regularized covariance matrix estimator (NERCOME) of Lam (2016).

Parameters

X [numpy.ndarray, shape = (p, n)] A 2d array of log-returns (n observations of p assets).

m [int, default = `None`] The size of the random split with which the eigenvalues are computed. If `None` the estimator searches over values suggested by Lam (2016) in Equation (4.8) and selects the best according to Equation (4.7).

M [int, default = 50] The number of permutations. Lam (2016) suggests 50.

Returns

opt_Sigma_hat [numpy.ndarray, shape = (p, p)] The NERCOME covariance matrix estimate.

Notes

A very different approach to the analytic formulas of nonlinear shrinkage is pursued by Abadir et al 2014. These authors propose a nonparametric estimator based on a sample splitting scheme. They split the data into two pieces and exploit the independence of observations across the splits to regularize the eigenvalues. Lam (2016) builds on their results and proposes a nonparametric estimator that is asymptotically optimal even if the population covariance matrix $\mathbf{\Sigma}$ has a factor structure. In the case of low-rank strong factor models, the assumption that each observation can be written as $\mathbf{x}_t = \mathbf{\Sigma}^{1/2}\mathbf{z}_t$ for $t = 1, \ldots, n$, where each $\mathbf{z}_t$ is a $p \times 1$ vector of independent and identically distributed random variables z_{it}, for $i = 1, \ldots, p$, with zero-mean and unit variance, is violated since the covariance matrix is singular and its Cholesky decomposition does not exist. Both *`linear_shrinkage()`* and *`nonlinear_shrinkage()`* are build on this assumption are no longer optimal if it is not fulfilled. The proposed nonparametric eigenvalue-regularized covariance matrix estimator (NERCOME) starts by splitting the data into two pieces of size n_1 and $n_2 = n - n_1$. It is assumed that the observations are i.i.d with finite fourth moments such that the statistics computed in the different splits are likewise independent of each other. The sample covariance matrix of the first partition is defined as $\mathbf{S}_{n_1} := \mathbf{X}_{n_1}\mathbf{X}_{n_1}'/n_1$. Its spectral decomposition is given by $\mathbf{S}_{n_1} = \mathbf{U}_{n_1}\mathbf{\Lambda}_{n_1}\mathbf{U}_{n_1}'$, where $\mathbf{\Lambda}_{n_1}$ is a diagonal matrix, whose elements are the eigenvalues $\lambda_{n_1} = (\lambda_{n_1,1}, \ldots, \lambda_{n_1,p})$ and an orthogonal matrix $\mathbf{U}_{n_1}$, whose columns $[\mathbf{u}_{n_1,1} \ldots \mathbf{u}_{n_1,p}]$ are the corresponding eigenvectors. Analogously, the sample covariance matrix of the second partition is defined by $\mathbf{S}_{n_2} := \mathbf{X}_{n_2}\mathbf{X}_{n_2}'/n_2$. Theorem 1 of Lam (2016) shows that \begin{equation} \widetilde{\mathbf{d}}_{n_1}^{(\text{NERCOME})}= \operatorname{diag}(\mathbf{U}_{n_1}^{\prime} \mathbf{S}_{n_2} \mathbf{U}_{n_1}) \end{equation} is asymptotically the same as the finite-sample optimal rotation equivariant $\mathbf{d}_{n_1}^{*} = \mathbf{U}_{n_1}'\mathbf{\Sigma}\mathbf{U}_{n_1}$ based on the section n_1. The proposed estimator is thus of the rotation equivariant form, where the elements of the diagonal matrix are chosen according to $\widetilde{\mathbf{d}}_{n_1}^{(\text{NERCOME})}$. In other words the estimator is given by \begin{equation} \widetilde{\mathbf{S}}_{ n_1}^{(\text{NERCOME})}:= \sum_{i=1}^{p}\widetilde{\mathbf{d}}_{n_1}^{(\text{NERCOME})} \cdot \mathbf{u}_{n_1, i} \mathbf{u}_{n_1, i}^{\prime} \end{equation} The author shows that this estimator is asymptotically optimal with respect to the Frobenius loss even under factor structure. However, it uses the sample data inefficiently since only one section is utilized for the calculation of each component. The natural extension is to permute the data and bisect it anew. With these sections an estimate is computed according to $\widetilde{\mathbf{S}}_{n_1}^{(\text{NERCOME})}$. This is done M times and the covariance matrix estimates are averaged: \begin{equation} \widetilde{\mathbf{S}}_{n_1, M}^{(\text{NERCOME})}:=\frac{1}{M} \sum_{j=1}^{M}\widetilde{\mathbf{S}}_{n_1, j}^{(\text{NERCOME})}. \end{equation} The estimator depends on two tuning parameters, M and n_1. Higher M give more accurate results but the computational cost grows as well. The author suggests that more than 50 iterations are generally not needed for satisfactory results. n_1 is subject to regularity conditions. The author proposes to search over the contenders \begin{equation} n_1=\left[2 n^{1 / 2}, 0.2 n, 0.4 n, 0.6 n, 0.8 n, n-2.5 n^{1 / 2}, n-1.5 n^{1 / 2}\right] \end{equation} and select the one that minimizes the following criterion inspired by Bickel (2008) \begin{equation} g(n_1)=\left\|\frac{1}{M} \sum_{j=1}^{M} \left(\widetilde{\mathbf{S}}_{n_1, j}^{(\text{NERCOME})} - \mathbf{S}_{n_2, j}\right)\right\|_{F}^{2}. \end{equation}

References

Abadir, K. M., Distaso, W. and Zikeˇs, F. (2014). Design-free estimation of variance matrices, Journal of Econometrics 181(2): 165–180.

Bickel, P. J. and Levina, E. (2008). Regularized estimation of large covariance matrices, The Annals of Statistics 36(1): 199–227.

Lam, C. (2016). Nonparametric eigenvalue-regularized precision or covariance matrix estimator, The Annals of Statistics 44(3): 928–953.

Examples

```
>>> np.random.seed(0)
>>> n = 13
>>> p = 5
>>> X = np.random.multivariate_normal(np.zeros(p), np.eye(p), n)
>>> cov = nercome(X.T, m=5, M=10)
>>> cov
array([[ 1.34226722,  0.09693429,  0.00233125,  0.17717658, -0.01643898],
       [ 0.09693429,  1.0508423 ,  0.10112215, -0.22908987, -0.04914651],
       [ 0.00233125,  0.10112215,  1.0731665 , -0.02959628,  0.38652859],
       [ 0.17717658, -0.22908987, -0.02959628,  1.10753766,  0.1807373 ],
       [-0.01643898, -0.04914651,  0.38652859,  0.1807373 ,  0.88832791]])
```

nerive

`hd.nerive`(*tick_series_list*, *stp=None*, *estimator=None*, *\\*\\*kwargs*)

The nonparametric eigenvalue-regularized integrated covariance matrix estimator (NERIVE) of Lam and Feng (2018). This estimator is similar to the `nercome()` estimator extended into the hight frequency setting.

Parameters

tick_series_list [list of pd.Series] Each pd.Series contains ticks of one asset with datetime index.

K [numpy.ndarray, default= `None`] An array of sclales. If `None` all scales $i = 1, ..., M$ are used, where M is chosen $M = n^{1/2}$ acccording to Eqn (34) of Zhang (2006).

stp [array-like of datetime.time() objects, default = [9:30, 12:45, 16:00]] The split time points.

estimator [function, default = `None`] An integrated covariance estimator taking `tick_series_lists` as the first argument. If `None` the `msrc_pairwise()` is used.

****kwargs** [miscellaneous] Keyword arguments of the `estimator`.

Returns

out [numpy.ndarray, 2d] The NERIVE estimate of the integrated covariance matrix.

Notes

The nonparametric eigenvalue-regularized integrated covariance matrix estimator (NERIVE) proposed by Lam and Feng (2018) splits the sample into L partitions. The split points are denoted by $$ 0=\widetilde{\tau}_{0}<\widetilde{\tau}_{1}<\cdots<\widetilde{\tau}_{L}=T $$ and the lth partition is given by $(\widetilde{\tau}_{l-1}, \widetilde{\tau}_l]$. The integrated covariance estimator for the lth partition is \begin{equation} \widehat{\mathbf{\Sigma}}_l=\mathbf{U}_{-l} \operatorname{diag}\left(\mathbf{U}_{-l}’ \widetilde{\mathbf{\Sigma}}_l \mathbf{U}_{-l}\right) \mathbf{U}_{-l}’ \end{equation} where $\mathbf{U}_{-l}$ is an orthogonal matrix depending on all observations over the full interval $[0, T]$ except the lth partition. The NERIVE estimator over the full interval $[0, T]$ is given by \begin{equation} \widehat{\mathbf{\Sigma}}=\sum_{l=1}^{L} \widehat{\mathbf{\Sigma}}_l= \sum_{l=1}^{L} \mathbf{U}_{-l} \operatorname{diag}\left(\mathbf{U}_{-l}’ \widetilde{\mathbf{\Sigma}}_l \mathbf{U}_{-l}\right) \mathbf{U}_{-l}’. \end{equation} $\widetilde{\mathbf{\Sigma}}$ is an integrated covariance estimator that corrects for asynchronicity and microstructure noise, e.g., one of `hf`. Lam and Feng (2018) choose the TSRC for the sake of tractablility in the proofs. Importantly, NERIVE does not assume i.i.d. observations but weak dependence between the log-price process and the microstructure noise process within partition l, and weak serial dependence of microstructure noise vectors, given $\mathcal{F}_{-l}$. Similar to NERCOME, NERIVE allows for the presence of pervasive factors as long as they persist between refresh times.

Warning: NERIVE splits the data into smaller subsamples. Estimator parameters that depend on the sample size must be adjusted. Further, the price process must be preprocessed to have zero mean return over the **full** sample.

References

Lam, C. and Feng, P. (2018). A nonparametric eigenvalue-regularized integrated covariance matrix estimator for asset return data, Journal of Econometrics 206(1): 226–257.

nonlinear_shrinkage

`hd.`**`nonlinear_shrinkage`**`(`*X*`)`

Compute the shrunk sample covariance matrix with the analytic nonlinear shrinkage formula of Ledoit and Wolf (2018). This estimator shrinks the sample sample covariance matrix with `_nonlinear_shrinkage_cov()`. The code has been adapted from the Matlab implementation provided by the authors in Appendix D.

Parameters

x [numpy.ndarray, shape = (p, n)] A sample of synchronized log-returns.

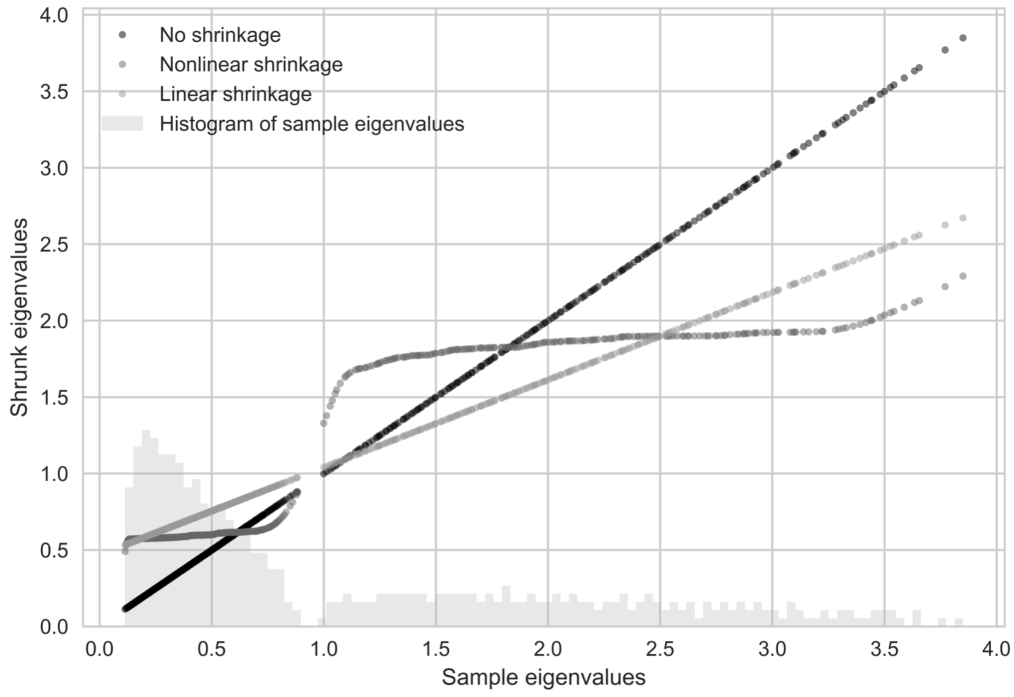
Returns

sigmatilde [numpy.ndarray, shape = (p, p)] The nonlinearly shrunk covariance matrix estimate.

Notes

The idea of shrinkage covariance estimators is further developed by Ledoit and Wolf (2012) who argue that given a sample of size $\mathcal{O}(p^2)$, estimating $\mathcal{O}(1)$ parameters, as in linear shrinkage, is precise but too restrictive, while estimating $\mathcal{O}(p^2)$ parameters, as in the the sample covariance matrix, is impossible. They argue that the optimal number of parameters to estimate is $\mathcal{O}(p)$. Their proposed non-linear shrinkage estimator uses exactly p parameters, one for each eigenvalue, to regularize each eigenvalue with a specific shrinkage intensity individually. Linear shrinkage, in contrast, is restricted by a single shrinkage intensity with which all eigenvalues are shrunk uniformly. Nonlinear shrinkage enables a nonlinear fit of the shrunk eigenvalues, which is appropriate when there are clusters of eigenvalues. In this case, it may be optimal to pull a small eigenvalue (i.e., an eigenvalue that is below the grand mean) further downwards and hence further away from the grand mean. Linear shrinkage, in contrast, always pulls a small eigenvalue upwards. Ledoit and Wolf (2018) find an analytic formula based on the Hilbert transform that nonlinearly shrinks eigenvalues asymptotically optimally with respect to the MV-loss function (as well as the quadratic Frobenius loss). The shrinkage function via the Hilbert transform can be interpreted as a local attractor. Much like the gravitational field extended into space by massive objects, eigenvalue clusters exert an attraction force that increases with the mass (i.e. the number of eigenvalues in the cluster) and decreases with the distance. If an eigenvalue λ_i has many neighboring eigenvalues slightly smaller than itself, the exerted force on λ_i will have large magnitude and downward direction. If λ_i has many neighboring eigenvalues slightly larger than itself, the exerted force on λ_i will also have large magnitude but upward direction. If the neighboring eigenvalues are much larger or much smaller than λ_i the magnitude of the force on λ_i will be small. The nonlinear effect this has on the shrunk eigenvalues can be seen in the figure below. The linearly shrunk eigenvalues, on the other hand, follow a line. Both approaches reduce the dispersion of eigenvalues and hence deserve the name shrinkage.

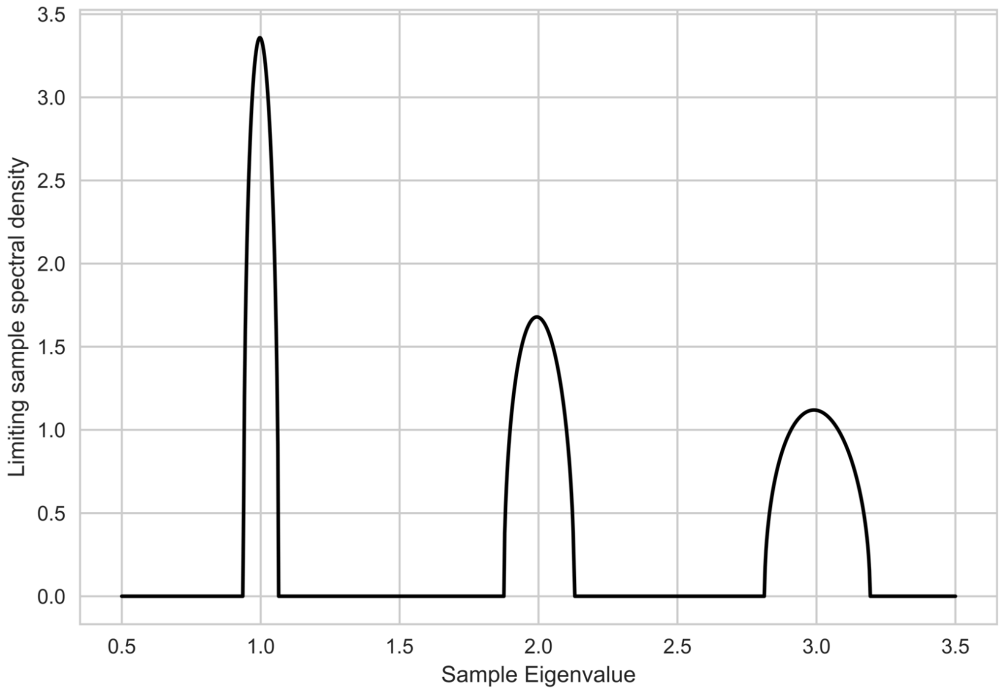
Figure 1: 1500 variates are drawn from a zero-mean 500-dimensional multivariate-normal distribution with a diagonal covariance matrix, which has 300 eigenvalues equal to 0.5 and 200 eigenvalues equal to 2. The sample eigenvalues are shrunk by the linear and nonlinear shrinkage methods and plotted against the original sample eigenvalues. At the bottom is a histogram of sample eigenvalues.



The authors assume that there exists a $n \times p$ matrix $\mathbf{Z}$ of i.i.d. random variables with mean zero, variance one, and finite 16th moment such that the matrix of observations $\mathbf{X} := \mathbf{Z}\mathbf{\Sigma}^{1/2}$. Neither $\mathbf{\Sigma}^{1/2}$ nor $\mathbf{Z}$ are observed on their own. This assumption might not be satisfied if the data generating process is a factor model. Use `nercome()` or `nerive()` if you believe the assumption is in your dataset violated. Theorem 3.1 of Ledoit and Wolf (2018) states that under their assumptions and general asymptotics the MV-loss is minimized by a rotation equivariant covariance estimator, where the elements of the diagonal matrix are \begin{equation} \label{eqn: oracle diag} \widehat{\mathbf{\delta}}^{(o, nl)}:=\left(\widehat{d}_{1}^{(o, nl)}, \ldots, \widetilde{d}_{p}^{(o, nl)}\right):=\left(d^{(o, nl)}\left(\lambda_{1}\right), \ldots, d^{(o, nl)}\left(\lambda_{p}\right)\right). \end{equation} $d^{(o,nl)}(x)$ denotes the oracle nonlinear shrinkage function and it is defined as \begin{equation} \forall x \in \operatorname{Supp}(F) \quad d^{(o, nl)}(x):=\frac{x}{ [\pi c x f(x)]^{2}+\left[1-c-\pi c x \mathcal{H}_{f}(x)\right]^{2}}, \end{equation} where $\mathcal{H}_g(x)$ is the Hilbert transform. As per Definition 2 of Ledoit and Wolf (2018) it is defined as \begin{equation} \forall x \in \mathbb{R} \quad \mathcal{H}_{g}(x):=\frac{1}{\pi} P V \int_{-\infty}^{+\infty} g(t) \frac{d t}{t-x}, \end{equation} which uses the Cauchy Principal Value, denoted as PV to evaluate the singular integral in the following way \begin{equation} P V \int_{-\infty}^{+\infty} g(t) \frac{d t}{t-x}:=\lim _{\varepsilon \rightarrow 0^{+}}\left[\int_{-\infty}^{x-\varepsilon} g(t) \frac{d t} {t-x}+\int_{x+\varepsilon}^{+\infty} g(t) \frac{d t}{t-x}\right]. \end{equation} It is an oracle estimator due to the dependence on the limiting sample spectral density f, its Hilbert transform $\mathcal{H}_f$, and the limiting concentration ratio c, which are all unobservable. Nevertheless, it represents progress compared to `hd.fsopt()`, since it no longer depends on the full population covariance matrix, $\mathbf{\Sigma}$, but only on its eigenvalues. This reduces the number of parameters to be estimated from the impossible $\mathcal{O}(p^2)$ to the manageable $\mathcal{O}(p)$. To make the estimator feasible, unobserved quantities have to be replaced by statistics. A consistent estimator for the limiting concentration c is the sample concentration $c_n = p/n$. For the limiting sample spectral density f, the authors propose a kernel estimator. This is necessary, even though $F_n \overset{\text{a.s}}{\to} F$, since F_n is discontinuous at every λ_i and thus its derivative f_n, which would've been the natural estimator for f, does not exist there. A kernel density estimator is a non-parametric estimator of the probability density function of a random variable. In kernel density estimation data from a finite sample is smoothed with a non-negative function K, called the kernel, and a smoothing parameter h, called the bandwidth, to make inferences about the population density. A kernel estimator is of the form \begin{equation} \widehat{f}_{h}(x)=\frac{1}{N} \sum_{i=1}^{N} K_{h}\left(x-x_{i}\right)= \frac{1}{N h} \sum_{i=1}^{N} K\left(\frac{x-x_{i}}{h}\right). \end{equation}

The chosen kernel estimator for the limiting sample spectral density is based on the Epanechnikov kernel with a variable bandwidth, proportional to the eigenvalues, $h_i := \lambda_i h$, for $i = 1, \ldots, p$, where the global bandwidth is set to $h := n^{-1/3}$. The reasoning behind the variable bandwidth choice can be intuited from the figure below, which shows that the support of the limiting sample spectral distribution is approximately proportional to the eigenvalue.

Figure 2: The limiting sample spectral density is an equally weighted mixture of Marchenko-Pastur laws with population eigenvalues 1, 2, 3 and c = 0.001



The Epanechnikov kernel is defined as \begin{equation} \forall x \in \mathbb{R} \quad \kappa^{(E)}(x):= \frac{3}{4 \sqrt{5}}\left[1-\frac{x^{2}}{5}\right]^{+}. \end{equation} The kernel estimators of f and $\mathcal{H}$ are thus \begin{equation} \forall x \in \mathbb{R} \quad \widetilde{f}_{n}(x):= \frac{1}{p} \sum_{i=1}^{p} \frac{1}{h_{i}} \kappa^{(E)} \left(\frac{x-\lambda_{i}}{h_{i}}\right)=\frac{1}{p} \sum_{i=1}^{p} \frac{1}{\lambda_{i} h} \kappa^{(E)}\left(\frac{x-\lambda_{i}}{\lambda_{i} h}\right) \end{equation} and \begin{equation} \mathcal{H}_{\tilde{f}_{n}}(x):=\frac{1}{p} \sum_{i=1}^{p} \frac{1}{h_{i}} \mathcal{H}_{k}\left(\frac{x-\lambda_{i}}{h_{i}}\right)=\frac{1}{p} \sum_{i=1}^{p} \frac{1}{\lambda_{i} h} \mathcal{H}_{k}\left(\frac{x-\lambda_{i}} {\lambda_{i} h}\right)=\frac{1}{\pi} PV \int \frac{\widetilde{f}_{n}(t)}{x-t} d t, \end{equation} respectively. The feasible nonlinear shrinkage estimator is of rotation equivariant form, where the elements of the diagonal matrix are \begin{equation} \forall i=1, \ldots, p \quad \widetilde{d}_{i}:=\frac{\lambda_{i}} {\left[\pi \frac{p}{n} \lambda_{i} \widetilde{f}_{n} \left(\lambda_{i}\right)\right]^{2}+\left[1-\frac{p}{n}-\pi \frac{p}{n} \lambda_{i} \mathcal{H}_{\tilde{f}_{n}}\left(\lambda_{i} \right)\right]^{2}} \end{equation} In other words, the feasible nonlinear shrinkage estimator is \begin{equation} \widetilde{\mathbf{S}}:=\sum_{i=1}^{p} \widetilde{d}_{i}

\mathbf{u}_{i} \mathbf{u}_{i}^{\prime}. \end{equation}

References

Ledoit, O. and Wolf, M. (2018). Analytical nonlinear shrinkage of large-dimensional covariance matrices, University of Zurich, Department of Economics, Working Paper (264).

Examples

```
>>> np.random.seed(0)
>>> n = 13
>>> p = 6
>>> X = np.random.multivariate_normal(np.zeros(p), np.eye(p), n)
>>> nonlinear_shrinkage(X.T)
array([[ 1.50231589e+00, -2.49140874e-01,  2.68050353e-01,
         2.69052962e-01,  3.42958216e-01, -1.51487901e-02],
       [-2.49140874e-01,  1.05011440e+00, -1.20681859e-03,
        -1.25414579e-01, -1.81604754e-01,  4.38535891e-02],
       [ 2.68050353e-01, -1.20681859e-03,  1.02797073e+00,
         1.19235516e-01,  1.03335603e-01,  8.58533018e-02],
       [ 2.69052962e-01, -1.25414579e-01,  1.19235516e-01,
         1.03290514e+00,  2.18096913e-01,  5.63011351e-02],
       [ 3.42958216e-01, -1.81604754e-01,  1.03335603e-01,
         2.18096913e-01,  1.22086494e+00,  1.07255380e-01],
       [-1.51487901e-02,  4.38535891e-02,  8.58533018e-02,
         5.63011351e-02,  1.07255380e-01,  1.07710975e+00]])
```

to_corr

hd.**to_corr**(*cov*)

Convert a covariance matrix to a correlation matrix.

Parameters

cov [numpy.ndarray, 2d, float] A covariance matrix.

Returns

out [numpy.ndarray, 2d] A correlation matrix

Examples

```
>>> cov = np.array([[2., 1.],[1., 2.]])
>>> to_corr(cov)
array([[1. , 0.5],
       [0.5, 1. ]])
```

The need for regularization

The sample covariance matrix and especially its inverse, the precision matrix, have bad properties in high dimensions, i.e., when the concentration ratio $c_n = \frac{p}{n}$ is not a small number. Then (1) the sample covariance matrix is estimated with a lot of noise since $\mathcal{O}(p^2)$ parameters have to be estimated with $pn = \mathcal{O}(p^2)$ observations, if n is of the same order of magnitude as p . And (2), if the first principal component of returns explains a large part of their variance, the condition number of the population covariance matrix, $\mathbf{\Sigma}$, is already high. A high concentration ratio increases the dispersion of sample eigenvalues above and beyond the dispersion of population eigenvalues, increasing the condition number further and leading to a very ill-conditioned sample covariance matrix.

Mathematically, as illustrated by Engle et al. 2019, the last point can be seen as follows. Define the population and sample spectral distribution, i.e., the cross section cumulative distribution function that returns the proportion of population and sample eigenvalues smaller than x, respectively, as

$$\begin{aligned} &H_{n}(x):=\frac{1}{p} \sum_{i=1}^{p} \mathbf{1}_{\left\{x \leq \tau_{i, T}\right\} } , \forall x \in \mathbb{R}, \ &F_{n}(x):=\frac{1}{p} \sum_{i=1}^{p} \mathbf{1}_{\left\{x \leq \lambda_{i, T}\right\}}, \forall x \in \mathbb{R}. \end{aligned}$$ Under general asymptotics and its standard assumptions, p is a function of n and both p and n – not just n – go to infinity. Then, according to Silverstein (1995), $$F_{n}(x) \stackrel{\text { a.s }}{\longrightarrow} F(x), \quad$$ where F denotes the nonrandom limiting spectral distribution. From the equality \begin{equation} \int_{-\infty}^{+\infty} x^{2} d F(x)=\int_{-\infty}^{+\infty} x^{2} d H(x)+c\left[\int_{-\infty}^{+\infty} x d H(x)\right]^{2} \end{equation} it can be seen that if the limiting concentration ratio c is greater than zero, the sample eigenvalue dispersion is inflated. The mean of the sample eigenvalues, however, is unbiased \begin{equation} \label{eqn: eigenvalue_mean} \int_{-\infty}^{+\infty} x d F(x)=\int_{-\infty}^{+\infty} x d H(x). \end{equation}

The distortion of extreme eigenvalues is very large for high concentrations. But even for relatively small c, the eigenvalue dispersion can be high enough such that regularization is necessary to ameliorate instability. The mathematical intuition behind this can be seen from the Marchenko-Pastur, which states that the limiting spectrum of the sample covariance matrix $\mathbf{S} = \mathbf{X}\mathbf{X}'/n$ of independent and identically distributed p-dimensional random vectors $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_p)'$ with mean $\mathbf{0}$ and covariance matrix $\mathbf{\Sigma} = \sigma^2\mathbf{I}_p$, has density \begin{equation} f_{c}(x)=\left\{\begin{array}{ll} \frac{1}{2 \pi x c \sigma^{2}} \sqrt{(b-x)(x-a)}, & a \leq x \leq b \ 0, & \text { otherwise, } \end{array}\right. \end{equation} where the smallest and the largest eigenvalues are given by $a = \sigma^2(1-\sqrt{c})^2$ and $b = \sigma^2(1+\sqrt{c})^2$, respectively, as $p, n \to \infty$ with $p/n \to c > 0$. To illustrate, say the interest lies on estimating a covariance matrix for 1000 stocks on daily data and suppose finite sample behavior of eigenvalues is reasonably well approximated by the Marchenko-Pastur law. Using a rolling window of anything less than approximately 4 years of daily returns results in a singular covariance matrix. Widening the window to include 8 years of data, the concentration ratio would be approximately $c_n = 1/2$. Plugging this number into the equations for the smallest and largest eigenvalues, it can be seen that they are, respectively, 91% smaller and 191% larger than the population eigenvalues, which are equal to σ^2. Even for an extremely wide window of 40 years, the eigenvalues would still be underestimated by 53% for the smallest, and overestimated by 73% for the largest. To reduce the overdispersion enough such that the covariance matrix becomes well-conditioned, such a wide window would be necessary that inaccuracies due to non-stationarity would dominate, if the data are even available. Empirically, the cross-section of stock returns exhibits correlation, hence $\mathbf{\Sigma} \neq \sigma^2\mathbf{I}_p$. Ait-Sahalia and Xiu (2017), for example, find a low rank factor structure plus a sparse industry-clustered covariance matrix of residuals, which may exacerbate the instability of the precision matrix since the eigenvalues of the population covariance matrix are already highly dispersed.

The solution to (1) is to reduce the number of parameters by identifying a small number of factors that explain a large proportion of the variance and then estimating the factor loading of each stock on those factors, which reduces the number of parameters considerably if p is large. The solution to (2) is to shrink the covariance matrix towards a shrinkage target which has a stable structure such as the identity matrix. This has the effect of pulling eigenvalues of the sample covariance matrix towards their grand mean, which is unbiased, while keeping the sample eigenvectors unaltered. Estimators that retain the original eigenvectors while seeking better properties through modification of the eigenvalues are called rotation equivariant. The result is a reduction of dispersion of sample eigenvalues and thus the condition number. Since overfitting is known to increase the dispersion of sample eigenvalues, shrinking them has the effect of regularization.

Rotation equivariance

The literature of eigenstructure regularized covariance estimation for portfolio allocation is predominantly focused on rotation equivariant estimators. To define the estimator consider the sample covariance matrix $\mathbf{S}_n := \mathbf{X}_n\mathbf{X}_n'/n$ based on a sample of n i.i.d. observations $\mathbf{X}_n$ with zero-mean. For the sake of readability, the sample suffix is suppressed in the following text unless it is not clear from the context that the quantity depends on the sample. According to the spectral theorem the sample covariance matrix can be decomposed into $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}'$, where $\mathbf{\Lambda}$ is a diagonal matrix, whose elements are the eigenvalues $\lambda = (\lambda_1, \ldots, \lambda_p)$ and an orthogonal matrix $\mathbf{U}$, whose columns $[\mathbf{u}_1 \ldots \mathbf{u}_p]$ are the corresponding eigenvectors. A rotation equivariant estimator of the population covariance matrix $\mathbf{\Sigma}$ is of the form \begin{equation} \label{eqn: re} \widehat{\mathbf{\Sigma}}:=\mathbf{U}\widehat{\mathbf{\Delta}}\mathbf{U}^{\prime}=\sum_{i=1}^{p} \widehat{\delta}_{i} \cdot \mathbf{u}_{ i} \mathbf{u}_{i}^{\prime}, \end{equation} where $\widehat{\mathbf{\Delta}}$ is a diagonal matrix with elements $\widehat{\delta}$. The infeasible finite-sample optimal rotation equivariant estimator, `hd.fsopt()`, choses the elements of the diagonal matrix as \begin{equation} \label{eqn: optimal re} \mathbf{d}^{*}:=\left(d_{1}^{*}, \ldots, d_{p}^{*}\right):=\left(\mathbf{u}_{1}^{\prime} \mathbf{\Sigma} \mathbf{u}_{1}, \ldots, \mathbf{u}_{p}^{\prime} \mathbf{\Sigma} \mathbf{u}_{p}\right). \end{equation} This estimator is an oracle since it depends on the unobservable population covariance matrix. In order to be feasible, $\widehat{\mathbf{\Delta}}$ has to be estimated from data. The `hd.linear_shrinkage()`, and the `hd.nonlinear_shrinkage()` estimator, estimate the elements of $\widehat{\mathbf{\Delta}}$, i.e., $\widehat{\delta} = \left(\widehat{\delta}_1, \ldots, \widehat{\delta}_p\right) \in (0, +\infty)^p$, as a function of λ_n. These two approaches differ in how many parameters this function takes. They both are elements of the rotation equivariant class, though. Within this framework, originally due to Stein (1986), lecture 4, rotations of the original data are deemed irrelevant for covariance estimation. Hence, the rotation equivariant covariance estimate based on the rotated data equals the rotation of the covariance estimate based on the original data, i.e., $\widehat{\mathbf{\Sigma}}(\mathbf{XW}) = \mathbf{W}'\widehat{\mathbf{\Sigma}}(\mathbf{X})\mathbf{W}$, where $\mathbf{W}$ is some p-dimensional orthogonal matrix. This characteristic distinguishes the class of rotation equivariant estimators from regularization schemes based on sparsity, since a change of basis does generally not preserve the sparse structure of the matrix.

References

Ait-Sahalia, Y. and Xiu, D. (2017). Using principal component analysis to estimate a high dimensional factor model with high-frequency data, Journal of Econometrics 201(2): 384–399.

Stein, C. (1986). Lectures on the theory of estimation of many parameters, Journal of Soviet Mathematics 34(1): 1373–1403.

Marchenko, V. A. and Pastur, L. A. (1967). Distribution of eigenvalues for some sets of random matrices, Matematicheskii Sbornik 114(4): 507–536.

Engle, R. F., Ledoit, O. and Wolf, M. (2019). Large dynamic covariance matrices, Journal of Business & Economic Statistics 37(2): 363–375.

Silverstein, J. W. (1995). Strong convergence of the empirical distribution of eigenvalues of large dimen- sional random matrices, Journal of Multivariate Analysis 55(2): 331–339.

2.1.3 sim Module

Functions

animate_heatmap(data)	Visualize an evolving series of matrices over time.
garch_11(n, sigma_sq_0, mu, alpha, beta, omega)	Generate GARCH(1, 1) log-returns of size n.
simple(size, corr_matrix, spec, liquidity, gamma)	Generate a simple p-dimensional GARCH(1,1) log-price process with microstructure noise and non-synchronous observation times.

animate_heatmap

sim.**animate_heatmap**(*data*)

Visualize an evolving series of matrices over time. This function is useful for visualizing a realization of conditional covariance matricies from a `Universe` simulation.

Parameters

data [list] A list of 2d numpy.ndarrays.

Returns

None

garch_11

sim.**garch_11**(*n*, *sigma_sq_0*, *mu*, *alpha*, *beta*, *omega*)

Generate GARCH(1, 1) log-returns of size n. This function is accelerated via JIT with Numba.

Parameters

n [int] The length of the wished time series.

sigma_sq_0 [float > 0] The variance starting value.

mu [float:] The drift of log-returns.

alpha [float >= 0] The volatility shock parameter. A higher value will lead to larger spikes in volatility. A.k.a short-term persistence.

beta [float >= 0] The volatility persistence parameter. A larger value will result in stronger persistence. A.k.a long-term persistence.

omega [float > 0] The variance constant. A higher value results in a higher mean variance.

Returns

r [numpy.ndarray] The GARCH log-returns time series.

sigma_sq [numpy.ndarray] The resulting variance time series with which each log-return was generated.

Notes

In general, the conditional variance of a GARCH(p,q) model is given by

$$\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{p} \beta_j \sigma_{t-j}^2.$$

The unconditional variance is given by

$$\sigma^2 = \frac{\omega}{1 - \sum_{i=1}^{q} \alpha_i - \sum_{j=1}^{p} \beta_j}.$$

Here, $p = q = 1$, and $\epsilon_t \sim \mathcal{N}(0, 1)$

simple

`sim.`**`simple`**(*size*, *corr_matrix*, *spec*, *liquidity*, *gamma*)

Generate a simple p-dimensional GARCH(1,1) log-price process with microstructure noise and non-synchronous observation times.

Parameters

size [int] The number of 'continous' log-prices.

corr_matrix [numpy.ndarray, shape = (p, p)] The correlation matrix of log-returns.

spec [list]

The garch specification. `[sigma_sq_0, mu, alpha, beta, omega]`

liquidity [float] A value between 0 and 1 that describes liquidity. A value of 1 means that the probability of observation is 100% each minute. 0.5 means that there is a 50% probability of observing a price each minute.

gamma [float >=0] The microstructure noise will be zero-mean Gaussian with variance $\gamma^2 var(r)$, where $var(r)$ is the variance of the underlying true return process. This noise is be added to the price.

Returns

price [numpy.ndarray, shape = (size, p)] The p-dimensional price time series.

Classes

`Universe`(feature_beta, factor_garch_spec, …)	The universe is a specification from which simulated realizations can be sampled.

Universe

class sim.**Universe**(*feature_beta*, *factor_garch_spec*, *industry_garch_spec*, *resid_garch_spec*, *factor_loadings*, *industry_loadings*, *liquidity=0.5*, *gamma=2*, *freq='m'*)

Bases: `object`

The universe is a specification from which simulated realizations can be sampled. Stocks follow a factor model, they belong to industries and have an idiosyncratic component. Stocks are predictable by a single feature.

Attributes

feature_beta [float] The true coefficient.

factor_garch_spec [list] The garch specification for factor returns. `[sigma_sq_0, mu, alpha, beta, omega]`

industry_garch_spec [list] The garch specification for industry returns. `[sigma_sq_0, mu, alpha, beta, omega]`

resid_garch_spec [list] The garch specification for residual returns. `[sigma_sq_0, mu, alpha, beta, omega]`

factor_loadings [numpy.ndarray] An array with factor loadings for each stock and factor. dim = n_stocks x n_factors

industry_loadings [numpy.ndarray] An array with industry loadings for each stock and industry. dim = n_stocks x n_industry This is usually a sparse matrix. One stock loads typically on one or two industries. A good number of industries is 10 to 20.

liquidity [float] A value between 0 and 1 that describes liquidity. A value of 1 means that the probability of observation is 100% each minute. 0.5 means that there is a 50% probability of observing a price each minute.

gamma [float >=0] The microstructure noise will be zero-mean Gaussian with variance $\gamma^2 var(r)$, where $var(r)$ is the variance of the underlying true return process. This noise is be added to the price.

freq [str, `'s'` or `'m'`.] The granularity of the discretized continous price process.

Methods Summary

cond_cov(self)	Compute the daily coditional integrated covariance matrix of stock returns within regular market hours in the universe from a realized universe simulation.
simulate(self, n_days)	The price process of each stock is continous but sampled at random times and with Gaussian microstructure noise.
uncond_cov(self)	Compute the uncoditional covariance of stock returns in the universe from a universe specification.
uncond_var(spec)	Compute the uncoditional variance from a GARCH(1,1) specification.

Methods Documentation

cond_cov (*self*)

Compute the daily coditional integrated covariance matrix of stock returns within regular market hours in the universe from a realized universe simulation.

Returns

list A list containing the conditional integrated covariance matrices of each day.

simulate (*self*, *n_days*)

The price process of each stock is continous but sampled at random times and with Gaussian microstructure noise. Importantly, the observation times are not synchronous across stocks. Observation times are restricted to market hours (9:30, 16:00) but the underlying process continues over night so that there are close-to-open gaps.

Parameters

n_days [int] The number of days of the sample path.

Returns

list of pd.DataFrame A list with two elements, the prices of each stock in a pd.DataFrame and the feature of each stock in a pd.DataFrame with datetime_index.

uncond_cov (*self*)

Compute the uncoditional covariance of stock returns in the universe from a universe specification.

Returns

numpy.ndarray The unconditional covariance matrix.

static uncond_var (*spec*)

Compute the uncoditional variance from a GARCH(1,1) specification.

Parameters

spec [list]

The garch specification. `[sigma_sq_0, mu, alpha, beta, omega]`

Returns

float The unconditional variance.

2.1.4 loss Module

This module provides loss functions frequently encountered in the literature on high dimensional covariance matrix estimation.

Functions

loss_fr(sigma_hat, sigma)	Squared Frobenius norm scaled by 1/p.
loss_mv(sigma_hat, sigma)	The minimum variance loss function of Ledoit and Wolf (2018).
marchenko_pastur(x, c, sigma_sq)	The Marchenko-Pastur distribution.
prial(S_list, sigma_hat_list, sigma[, loss_func])	The percentage relative improvement in average loss (PRIAL) over the sample covariance matrix.

loss_fr

`loss.`**`loss_fr`**`(`*sigma_hat*, *sigma*`)`

Squared Frobenius norm scaled by 1/p. Same as `np.linalg.norm(sigma_hat - sigma, 'fro')**2 *1/p`.

Parameters

sigma_hat [numpy.ndarray] The covariance matrix estimate using the estimator of interest.

sigma [numpy.ndarray] The (true) population covariance matrix.

Returns

out [float] The minimum variance loss.

Notes

The loss function is given by:

$$\mathcal{L}_n^{\mathrm{FR}}\left(\widehat{\Sigma}_n, \Sigma_n\right) := \frac{1}{p}\operatorname{Tr}\left[\left(\widehat{\Sigma}_n - \Sigma_n\right)^2\right]$$

loss_mv

`loss.`**`loss_mv`**`(`*sigma_hat*, *sigma*`)`

The minimum variance loss function of Ledoit and Wolf (2018).

Parameters

sigma_hat [numpy.ndarray] The covariance matrix estimate using the estimator of interest.

sigma [numpy.ndarray] The (true) population covariance matrix.

Returns

out [float] The minimum variance loss.

Notes

The minimum variance (MV)-loss function is proposed by Engle et al. (2019) as a loss function that is appropriate for covariance matrix estimator evaluation for the problem of minimum variance portfolio allocations under a linear constraint and large-dimensional asymptotic theory.

The loss function is given by:

$$\mathcal{L}_n^{\mathrm{MV}}\left(\widehat{\Sigma}_n, \Sigma_n\right) := \frac{\operatorname{Tr}\left(\widehat{\Sigma}_n^{-1}\Sigma_n\widehat{\Sigma}_n^{-1}\right)/p}{\left[\operatorname{Tr}\left(\widehat{\Sigma}_n^{-1}\right)/p\right]^2} - \frac{1}{\operatorname{Tr}\left(\Sigma_n^{-1}\right)/p}.$$

It can be interpreted as the true variance of the minimum variance portfolio constructed from the estimated covariance matrix.

marchenko_pastur

`loss.marchenko_pastur(x, c, sigma_sq)`

The Marchenko-Pastur distribution. This is the pdf of eigenvalues of a sample covariance matrix estimate of the true covariance matrix, which is a``sigma_sq`` scaled identity matrix. It depends on the concentration ratio `c`, which is the ratio of the dimension divided by the number of observations.

Parameters

x [float] The value of the sample eigenvalue.

c [float] The concentration ratio. $c = p/n$.

sigma_sq [float] The value of population eigenvalues.

Returns

p [float] The value of the Marchenko-Pastur distribution at the sample eigenvalue `x`.

Notes

The Marchenko-Pastur law states that the limiting spectrum of the sample covariance matrix $S = X'X/n$ of independent and identically distributed p-dimensional random vectors $\mathbf{X} = (x_1, \ldots, x_n)$ with mean $\mathbf{0}$ and covariance matrix $\mathbf{\Sigma} = \sigma^2 \mathbf{I}_p$, has density \begin{equation} f_{c}(x)=\left\{\begin{array}{ll} \frac{1}{2 \pi x c \sigma^{2}} \sqrt{(b-x)(x-a)}, & a \leq x \leq b \ 0, & \text { otherwise, } \end{array}\right. \end{equation} where the smallest and the largest eigenvalues are given by $a = \sigma^2(1-\sqrt{c})^2$ and $b = \sigma^2(1+\sqrt{c})^2$, respectively, as $p, n \to \infty$ with $p/n \to c > 0$.

References

Marchenko, V. A. and Pastur, L. A. (1967). Distribution of eigenvalues for some sets of random matrices, Matematicheskii Sbornik 114(4): 507–536.

prial

`loss.prial(S_list, sigma_hat_list, sigma, loss_func=None)`

The percentage relative improvement in average loss (PRIAL) over the sample covariance matrix.

Parameters

S_list [list of numpy.ndarray] The sample covariance matrix.

sigma_hat_list [list of numpy.ndarray] The covariance matrix estimate using the estimator of interest.

sigma [numpy.ndarray] The (true) population covariance matrix.

loss_func [function, defualt = None] The loss function. If `None` the minimum variance loss function is used.

Returns

prial [float] The PRIAL.

Notes

The percentage relative improvement in average loss (PRIL) over the sample covariance matrix is given by:

$$\mathrm{PRIAL}_n\left(\widehat{\Sigma}_n\right) := \frac{\mathbb{E}\left[\mathcal{L}_n\left(S_n, \Sigma_n\right)\right] - \mathbb{E}\left[\mathcal{L}_n\left(\widehat{\Sigma}_n, \Sigma_n\right)\right]}{\mathbb{E}\left[\mathcal{L}_n\left(S_n, \Sigma_n\right)\right] - \mathbb{E}\left[\mathcal{L}_n\left(S_n^*, \Sigma_n\right)\right]} \times 100\%$$

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX