

# Package ‘tawny’

February 7, 2012

**Type** Package

**Title** Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators

**Version** 2.0.2

**Depends** R (>= 2.10.0), tawny.types (>= 1.0.0), futile.matrix (>= 1.1.0), futile.logger (>= 1.2.0), PerformanceAnalytics, zoo,xts, quantmod, RUnit

**Date** 2012-02-07

**Author** Brian Lee Yung Rowe

**Maintainer** Brian Lee Yung Rowe <r@nurometic.com>

**Description** Portfolio optimization typically requires an estimate of a covariance matrix of asset returns. There are many approaches for constructing such a covariance matrix, some using the sample covariance matrix as a starting point. This package provides implementations for two such methods: random matrix theory and shrinkage estimation. Each method attempts to clean or remove noise related to the sampling process from the sample covariance matrix.

**License** GPL-2

**Repository** CRAN

**Date/Publication** 2012-02-07 16:38:49

## R topics documented:

tawny-package . . . . .	2
cov_shrink . . . . .	4
denoise . . . . .	6
divergence . . . . .	7
getPortfolioReturns . . . . .	8
optimizePortfolio . . . . .	10
sp500 . . . . .	12
sp500.subset . . . . .	12

---

tawny-package	<i>Provides various portfolio optimization strategies including random matrix theory and shrinkage estimators</i>
---------------	---

---

## Description

Portfolio optimization typically requires an estimate of a covariance matrix of asset returns. There are many approaches for constructing such a covariance matrix, some using the sample covariance matrix as a starting point. This package provides implementations for two such methods: random matrix theory and shrinkage estimation. Each method attempts to clean or remove noise related to the sampling process from the sample covariance matrix. Random matrix theory does this by using the known eigenvalue distribution of a random matrix as the null hypothesis, scaling any eigenvalues below a threshold to a lower bound, thus eliminating the noise related to the idiosyncratic noise of the matrix. Shrinkage estimation shrinks the sample covariance matrix towards a so-called global average that theoretically represents a truer estimate of the covariance matrix. A single API is provided for generating asset weights based on the different approaches.

## Details

Package: tawny  
 Type: Package  
 Version: 2.0.2  
 Date: 2012-02-07  
 License: GPL-2

There are a number of ways to use this package. At a high level, the estimation techniques can be applied to a portfolio and optimized portfolio weights are returned. This is followed by calculation of basic portfolio statistics and comparison functions to provide a quick, visual check to the results. It is possible to embark on further study using other packages (e.g. PerformanceAnalytics). If a zoo object already exists, then this is as simple as calling `optimizePortfolio` and specifying an appropriate (and built-in) function for generating a correlation matrix.

In addition to these functions there are a number of convenience methods for constructing simple portfolios for a given date range via `quantmod`. This includes `getPortfolioReturns` and `ensure`.

To get started using the package, the only requirement is to have a history of returns for the assets in the portfolio. The length of the portfolio is the sum of the window selected and the time frame to optimize against,

For people interested in studying the core behavior of Random Matrix Theory, the underlying `mp.*` functions are available. These functions provide direct control over eigenvalue density histogram plotting, theoretical distributions as specified by Marcenko and Pastur, and optimization functions for fitting the two. In most cases the functions are designed to be pluggable as they climb the tree of abstraction, meaning that an arbitrary optimization function can be plugged into the fitting function, and so on.

For people interested in studying shrinkage estimation techniques, these functions are primarily exposed as shrinkage.\*.

NOTE: This is an alpha release and the high-level portfolio functions have not been fully ported nor tested. . Use PerformanceAnalytics for performance analysis . Clean up optimization workflow . Clean up back testing vs single day workflows

### Author(s)

Brian Lee Yung Rowe

Maintainer: Brian Lee Yung Rowe <r@nurometic.com>

### References

Gatheral, Jim. "Random Matrix Theory and Covariance Estimation." 3 Oct. 2008. New York. 7 Oct. 2008 <[http://www.math.nyu.edu/fellows/\\_fin/\\_math/gatheral/RandomMatrixCovariance2008.pdf](http://www.math.nyu.edu/fellows/_fin/_math/gatheral/RandomMatrixCovariance2008.pdf)>.

Potters, Marc; Bouchaud, Jean-Philippe; Laloux, Laurent. "Financial Applications of Random Matrix Theory: Old Laces and New Pieces." Jul. 2005. Paris. 10 Dec. 2008 <<http://www.cfm.fr/papers/0507111.pdf>>

Olivier Ledoit and Michael Wolf. "Improved Estimation of the Covariance Matrix of Stock Returns With an Application to Portfolio Selection." Oct. 2001. London. 12 Feb. 2009 <<http://ideas.repec.org/a/eee/empfin/v10y2003/621.html>>

### See Also

[optimizePortfolio](#), [denoise](#), [getPortfolioReturns](#)

### Examples

```
# This is autorun outside of examples
tawny:::init()

## High level use of package
# Select a portfolio using 200 total observations
data(sp500.subset)
h <- sp500.subset
p <- create(TawnyPortfolio, h, 150)
b <- create(BenchmarkPortfolio, '^GSPC', 150, nrow(h), end=end(h))

# Optimize using a window of length 200 (there will be 51 total iterations)
ws <- optimizePortfolio(p, create(RandomMatrixFilter))
rs <- create(PortfolioReturns, p, ws)
o <- zoo(cbind(portfolio=rs, benchmark=b$returns), index(rs))
charts.PerformanceSummary(o)

# Generate weights based on the constant correlation shrinkage estimator
ws <- optimizePortfolio(p, create(ShrinkageFilter))
rs <- create(PortfolioReturns, p, ws)
o <- zoo(cbind(portfolio=rs, benchmark=b$returns), index(rs))
charts.PerformanceSummary(o)
```

---

 cov\_shrink

*Shrink the covariance matrix towards some global mean*


---

### Description

This performs a covariance shrinkage estimation as specified in Ledoit and Wolf. Using within the larger framework only requires using the ShrinkageFilter type, which handles the work of constructing a shrinkage estimate of the covariance matrix of returns (and consequently its corresponding correlation matrix).

### Usage

```

cov_shrink(...)
cov_sample(...)
cov.prior.cc(S)
cor.mean(S)
shrinkage.intensity(returns, prior, sample)
shrinkage.p(returns, sample)
shrinkage.r(returns, sample, pi.est)
shrinkage.c(prior, sample)
cov.shrink(...)
cov.sample(...)

```

### Arguments

returns	A zoo object of returns. This is TxM
sample	The sample covariance matrix (synonomous to S)
prior	The shrinkage target covariance matrix (synonomous to F)
S	The sample covariance matrix
pi.est	The estimate returned from shrinkage.p
...	Additional parameters to pass to prior.fun

### Details

```

cov_shrink(h, prior.fun = cov.prior.cc, ...) cov_shrink(h, T, constant.fun, prior.fun = cov.prior.cc, ...)
cov_shrink(h, ...)
cov_sample... cov_samplereturns cov_samplereturns

```

T - Length of returns series used in scaling of shrinkage coefficient h - A generic tawny object representing either a returns, covariance, or correlation matrix constant.fun -Use this function to calculate the shrinkage constant prior.fun - Generates the prior/model covariance matrix

Most of the code related to the shrinkage estimator is tied to calculating a value for the shrinkage coefficient. The remainder of the code shrinks the sample covariance matrix towards the target. In addition, there is a function generator used in conjunction with the optimizePortfolio process to produce a correlation matrix based on the shrinkage.

## Value

Scalars are produced by all of the shrinkage.\* functions, resulting in the final shrinkage coefficient, calculated by shrinkage.intensity.

The cov.sample function calculates the sample covariance matrix and is MxM.

The cov.shrink function produces the shrunk version of the covariance matrix and has the same dimensions as the sample covariance matrix.

The cor.mean function calculates the constant correlation used in estimating the global mean (aka the shrinkage target) produced by cov.prior.cc.

## Author(s)

Brian Lee Yung Rowe

## See Also

[tawny](#), [optimizePortfolio](#)

## Examples

```
# This is autorun outside of examples
tawny:::.init()

# Estimate the covariance matrix based on the given asset returns
data(sp500.subset)
ys <- create(TawnyPortfolio, sp500.subset, 150)
S.hat <- cov_shrink(ys)

# Optimize the portfolio weights using the shrinkage estimator
ws <- optimizePortfolio(ys, create(ShrinkageFilter))
#plotPerformance(ys,ws, bg='white', name='Shrinkage')

# Calculate the sample covariance matrix
#S <- cov.sample(ys)

# Calculate the shrinkage coefficient
#F <- cov.prior.cc(S)
#k <- shrinkage.intensity(ys, F, S)
```

---

denoise *Filter noise from a correlation matrix using RMT to identify the noise*

---

### Description

Used to filter out all eigenvalues below  $k^*$ . At a later date this will become pluggable so other people can use their own functions and/or provide their own parameters to this function.

### Usage

```
denoise(...)  
cor.empirical(h)  
cor.clean(es, lambda.plus=1.6, h=NULL)  
normalize(...)
```

### Arguments

h	A returns object
es	Eigenvalues and vectors
lambda.plus	Eigenvalue cutoff for cleaning
...	Arguments

### Details

```
denoise(p, estimator) normalize(h)
```

These are different implementations of the denoise function based on the estimator provided.

### Value

A cleaned correlation matrix.

### Author(s)

Brian Lee Yung Rowe

### See Also

[tawny](#), [optimizePortfolio](#)

**Examples**

```
# This is autorun outside of examples
tawny:::init()
data(sp500.subset)

h <- sp500.subset
p <- create(TawnyPortfolio, h, 150)
r1 <- denoise(p, create(SampleFilter))

r2 <- denoise(p, create(EmpiricalFilter))

r3 <- denoise(p, create(ShrinkageFilter))

r4 <- denoise(p, create(RandomMatrixFilter))
```

---

divergence	<i>Measure the divergence and stability between two correlation matrices</i>
------------	--

---

**Description**

The Kullback-Leibler distance function can be used to measure the divergence between two correlation matrices. Although originally designed for probability density functions, the literature shows how this can be extended to correlation matrices. By using this function, one can determine objectively the effectiveness of a particular filtering strategy for correlation matrices.

**Usage**

```
divergence(...)
divergence.kl(sigma.1, sigma.2)
divergence_lim(...)
stability_lim(...)
divergence.stability(h, count, window, filter)
plotDivergenceLimit.kl(m, t.range, ..., overlay = FALSE)
```

**Arguments**

h	A zoo object representing a portfolio with dimensions T x M
count	The number of bootstrap observations to create
window	The number of samples to include in each observation. Defaults to the anylength of h.
filter	The correlation filter to measure
sigma.1	The sample correlation matrix
sigma.2	The model correlation matrix (aka the filtered matrix)
m	The number of assets

t	The number of samples (dates) in an observation
t.range	A range of date samples. This can be a simple interval so long as it matches the number of samples per asset in the measured correlation matrix.
overlay	Overlay the divergence limit plot on an existing plot. Default is FALSE.
...	Additional parameters to pass to plot or lines

### Details

divergence(h, count, window = NULL, filter, measure = 'information') divergence\_lim(m, t = NULL) stability\_lim(m, t = NULL)

measure - The type of divergence to calculate. Possible choices are information (default) or stability.

### Value

A summary of the results of the divergence calculation including the mean divergence and an effective limit based on a random matrix.

### Author(s)

Brian Lee Yung Rowe

### Examples

```
# This is autorun outside of examples
tawny:::.init()

data(sp500.subset)
h <- sp500.subset

plotDivergenceLimit.kl(100, 80:499, col='green', ylim=c(0,55))

divergence(h, 25, filter=function(x) denoise(x, create(RandomMatrixFilter)))
divergence(h, 25, filter=function(x) denoise(x, create(ShrinkageFilter)))
```

---

getPortfolioReturns      *Utility functions for creating portfolios of returns and other functions*

---

### Description

Gets portfolio returns from closing prices (configurable). This uses quantmod under the hood to retrieve prices and construct returns based on a configurable transform.

Also included is a function that returns the composition of select indexes that can be used in conjunction with getPortfolioReturns() to get the underlying returns of the given index.

Additionally, there is a utility function that ensures that symbols have been properly loaded.

**Usage**

```
getIndexComposition(ticker = '^GSPC', hint = NA, src = 'yahoo')
getPortfolioReturns(symbols, obs = NULL, start = NULL, end = Sys.Date(), fun = function(x) Delt(Cl(x)),
ensure(serie, src = 'FRED', reload = FALSE, ...)
```

**Arguments**

ticker	The ticker of the index. For best mileage, use Yahoo! compatible tickers (including the caret prefix).
hint	A hint that specifies the number of assets in the index. If omitted, a default will be used based on pre-configured data for common indexes.
src	The data vendor to use. Defaults to yahoo but could work with google
symbols	A vector (or scalar) of tickers to download
obs	The number of observations to get
start	Alternatively, a start date can be used to specify the beginning of a range to download.
end	The end date of the range. Defaults to current date.
fun	A transform applied to the downloaded data. The default is to calculate returns on the close.
reload	Whether to reload data or just download missing data
na.value	What value to use if the resulting portfolio has NAs. The default is to omit any assets containing NA values.
serie	A vector (or scalar) of tickers to ensure exist in the current environment
...	Additional parameters to pass to getSymbols

**Details**

Typically only `getPortfolioReturns` and `getIndexComposition` will be used on a regular basis.

`Ensure` isn't as useful as initially conceived given that naming collisions have caused numerous issues. The code now uses `auto.assign=FALSE` in the underlying `getSymbols` call.

**Value**

`getPortfolioReturns` returns a TxM xts object of asset returns.

`getIndexComposition` returns a vector of asset symbols (i.e. tickers).

`ensure` returns nothing.

**Author(s)**

Brian Lee Yung Rowe

**See Also**

[tawny](#)

**Examples**

```

# This is autorun outside of examples
tawny:::init()

# Get a portfolio
h <- getPortfolioReturns(c('A', 'AA', 'AAPL'), obs=150)

# Get an index portfolio
h <- getPortfolioReturns(getIndexComposition('^DJI'), obs=100, reload=TRUE)

# Doesn't work because of numerical symbols - need to fix
#h <- getPortfolioReturns(getIndexComposition('^HSI'), obs=100, reload=TRUE)

# Ensure that some assets exist
ensure(c('K', 'JNPR'), src='yahoo')

```

---

optimizePortfolio

*Optimize a portfolio using the specified correlation filter*


---

**Description**

Performs basic minimum variance optimization on the given portfolio over the given window returning a zoo object of portfolio weights. The window must be less than or equal to the number of observations in `h` (s.t. if they are equal then only one resultant weight vector will be returned).

A number of preconfigured correlation matrix filters are available: `RandomMatrixFilter` for using random matrix theory, `ShrinkageFilter` returns a function for filtering the correlation matrix using a shrinkage estimator. A raw version is provided for comparison. These functions provide reasonable configurability, for example with `RandomMatrixFilter`, one can choose whether a histogram or a kernel density estimator is used to calculate the probability density function. With `ShrinkageFilter` one can select a constant correlation model or the identity as the model.

**Usage**

```

optimizePortfolio(...)

p.optimize(h, c.denoised)

```

**Arguments**

<code>h</code>	A zoo object representing a portfolio with dimensions $T \times M$
<code>c.denoised</code>	A cleaned correlation matrix
<code>...</code>	Additional parameters

**Details**

```
optimizePortfolio(h, window, cor.gen, ...)
```

This is the primary entry point to using the tawny package. This function calculates the portfolio weights over the portfolio based on a rolling window. Given  $M$  assets in the portfolio,  $T$  total observations, and a window of length  $t$ , the resulting weights object will have dimensions  $T - t + 1 \times M$ .

The weights matrix can then be analyzed to calculate standard portfolio performance metrics. A simple analytics function is provided so that cumulative returns can be easily viewed, although for more sophisticated analysis other packages should be used.

In theory any compatible correlation matrix generator can be used (and has in practice to test against proprietary risk models) and the function will generate portfolio weights accordingly. To leverage the remainder of the package, the `RandomMatrixFilter` function or `ShrinkageFilter` should be called. These wrappers are somewhat superfluous but do provide some utility by ensuring compatibility with the underlying RMT code that uses transposed matrices (pre-zoo integration). Additionally, by way of closures these functions are used to store hints to the optimizer and any final data massaging, potentially cleaning up code but admittedly can be serviced via the normal dots mechanism.

In the future, the default will be a direct handle to the underlying function once the rest of the code is converted to zoo.

The secondary function `optimizePortfolio.RMT` exists to optimize the correlation matrix using RMT exclusively. This is a more direct route to accessing the RMT functionality and might be more convenient to use. The intention is that the base `optimizePortfolio` function becomes a generic function that passes on to specific implementations, but the mechanics haven't been worked out yet. It is also possible to extract the optimizer and pass that in explicitly as a function.

**Value**

A weights zoo object with  $T - t + 1$  dates and  $M$  assets. The dates are aligned to the end date.

**Author(s)**

Brian Lee Yung Rowe

**Examples**

```
# This is autorun outside of examples
tawny:::init()

data(sp500.subset)
p <- create(TawnyPortfolio, sp500.subset, window=190)
ws <- optimizePortfolio(p, create(SampleFilter) )
ws <- optimizePortfolio(p, create(EmpiricalFilter) )
ws <- optimizePortfolio(p, create(RandomMatrixFilter) )
ws <- optimizePortfolio(p, create(ShrinkageFilter) )

# This is computationally faster although the convenient approach is to pass
# in the character symbol directly: ShrinkageFilter(market='^GSPC')
m <- getPortfolioReturns('^GSPC', obs=1000, end='2009-02-27')
ws <- optimizePortfolio(p, create(ShrinkageFilter, market=m) )
```

---

sp500	<i>A (mostly complete) subset of the SP500 with 250 data points</i>
-------	---

---

**Description**

This data set provides all components of the SP500 that are complete within the date range [2008-03-07, 2009-03-04] with returns suitable for running against the portfolio optimization routines. Note that the date range may periodically be updated to reflect current dates.

**Usage**

sp500

**Format**

A 250x491 zoo object of asset returns

**Source**

Yahoo! Finance

---

sp500.subset	<i>A subset of the SP500 with 200 data points</i>
--------------	---

---

**Description**

This data set provides a subset of the SP500 with returns suitable for running against the portfolio optimization routines.

**Usage**

sp500.subset

**Format**

A 200x75 zoo object of asset returns

**Source**

Yahoo! Finance

# Index

- \*Topic **datasets**
  - sp500, [12](#)
  - sp500.subset, [12](#)
- \*Topic **package**
  - tawny-package, [2](#)
- \*Topic **ts**
  - cov\_shrink, [4](#)
  - denoise, [6](#)
  - divergence, [7](#)
  - getPortfolioReturns, [8](#)
  - optimizePortfolio, [10](#)
  - tawny-package, [2](#)
- cor.clean (denoise), [6](#)
- cor.empirical (denoise), [6](#)
- cor.mean (cov\_shrink), [4](#)
- cov.prior.cc (cov\_shrink), [4](#)
- cov.prior.identity (cov\_shrink), [4](#)
- cov.sample (cov\_shrink), [4](#)
- cov.shrink (cov\_shrink), [4](#)
- cov\_sample (cov\_shrink), [4](#)
- cov\_shrink, [4](#)
- create.RandomMatrixFilter (denoise), [6](#)
- create.ShrinkageFilter (denoise), [6](#)
- deform (denoise), [6](#)
- denoise, [3](#), [6](#)
- divergence, [7](#)
- divergence\_lim (divergence), [7](#)
- ensure (getPortfolioReturns), [8](#)
- getIndexComposition
  - (getPortfolioReturns), [8](#)
- getPortfolioReturns, [3](#), [8](#)
- normalize (denoise), [6](#)
- optimizePortfolio, [3](#), [5](#), [6](#), [10](#)
- p.optimize (optimizePortfolio), [10](#)
- plotDivergenceLimit.kl (divergence), [7](#)
- shrinkage.c (cov\_shrink), [4](#)
- shrinkage.intensity (cov\_shrink), [4](#)
- shrinkage.p (cov\_shrink), [4](#)
- shrinkage.r (cov\_shrink), [4](#)
- sp500, [12](#)
- sp500.subset, [12](#)
- stability\_lim (divergence), [7](#)
- tawny, [5](#), [6](#), [9](#)
- tawny (tawny-package), [2](#)
- tawny-package, [2](#)