

Package ‘tawny’

June 22, 2009

Type Package

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

Version 1.1.0

Depends R (>= 2.8.0), futile, zoo, xts, quantmod

Suggests PerformanceAnalytics

Date 2009-06-20

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 2009-06-22 07:29:02

R topics documented:

tawny-package	2
cov.shrink	4
divergence	6
filter.RMT	7
getPortfolioReturns	9
optimizePortfolio	11
plotPerformance	13
sp500	15
sp500.subset	15

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: 1.1.0
Date: 2009-06-20
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.*`.

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>.

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](#), [filter.RMT](#), [getPortfolioReturns](#)

Examples

```
## High level use of package
# Select a portfolio using 200 total observations
data(sp500.subset)
h <- sp500.subset
# Optimize using a window of length 200 (there will be 51 total iterations)
ws <- optimizePortfolio(h, 150, getCorFilter.RMT() )
# Plot the performance of the resulting optimized portfolio
pf <- plotPerformance(h, ws, 150, y.min=-0.4)
# Compare the performance with a naive equal-weighted portfolio
ef <- compare.EqualWeighted(h, 150, y.min=-0.4)
# Also compare against the S&P 500
mf <- compare.Market('^GSPC',200,150, y.min=-0.4)

# Generate weights based on the constant correlation shrinkage estimator
ws <- optimizePortfolio(h, 150, getCorFilter.Shrinkage())
pf <- plotPerformance(h, ws, 150, y.min=-0.4)

## Low level use of the package
# View the theoretical Marcenko-Pastur distribution of eigenvalues of a
# random matrix
mp.theory(Q=4.2, sigma=1.1)

# Calculate the density of eigenvalues for a matrix of portfolio returns
h <- getRandomMatrix(20, 100)
hist <- mp.density.kernel(h)

# Fit an empirical distribution to a theoretical curve
hint <- c(6,2)
o <- optim(hint, mp.fit.kernel(hist))
```

 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 `getCorFilter.Shrinkage` function, which handles the work of constructing a shrinkage estimate of the covariance matrix of returns (and consequently its corresponding correlation matrix).

Usage

```

cov.shrink(h, ...)
## Default S3 method:
cov.shrink(h, ...)
## S3 method for class 'returns':
cov.shrink(h, prior.fun = cov.prior.cc, ...)
## S3 method for class 'covariance':
cov.shrink(h, T, constant.fun, prior.fun = cov.prior.cc, ...)
## S3 method for class 'correlation':
cov.shrink(h, ...)

cov.sample(returns)

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)

```

Arguments

<code>returns</code>	A zoo object of returns. This is TxM
<code>sample</code>	The sample covariance matrix (synonomous to S)
<code>prior</code>	The shrinkage target covariance matrix (synonomous to F)
<code>prior.fun</code>	Generates the prior/model covariance matrix
<code>constant.fun</code>	Use this function to calculate the shrinkage constant
<code>S</code>	The sample covariance matrix
<code>T</code>	Length of returns series used in scaling of shrinkage coefficient

<code>pi.est</code>	The estimate returned from <code>shrinkage.p</code>
<code>h</code>	A generic tawny object representing either a returns, covariance, or correlation matrix
<code>...</code>	Additional parameters to pass to <code>prior.fun</code>

Details

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 $M \times M$.

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

```
# Estimate the covariance matrix based on the given asset returns
data(sp500.subset)
ys <- sp500.subset
S.hat <- cov.shrink(ys)

# Optimize the portfolio weights using the shrinkage estimator
ws <- optimizePortfolio(ys, 150, getCorFilter.Shrinkage())
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)
```

divergence

*Measure the divergence and stability between two correlation matrices***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(h, count, window = NULL, filter = getCorFilter.RMT(), measure = 'information')
divergence.information(h, count, window, filter)
divergence.stability(h, count, window, filter)
divergence.kl(sigma.1, sigma.2)
divergenceLimit.kl(m, t = NULL)
stabilityLimit.kl(m, t = NULL)
plotDivergenceLimit.kl(m, t.range, ..., overlay = FALSE)
```

Arguments

<code>h</code>	A zoo object representing a portfolio with dimensions T x M
<code>count</code>	The number of bootstrap observations to create
<code>window</code>	The number of samples to include in each observation. Defaults to the anylength of <code>h</code> .
<code>filter</code>	The correlation filter to measure
<code>measure</code>	The type of divergence to calculate. Possible choices are information (default) or stability.
<code>sigma.1</code>	The sample correlation matrix
<code>sigma.2</code>	The model correlation matrix (aka the filtered matrix)
<code>m</code>	The number of assets
<code>t</code>	The number of samples (dates) in an observation
<code>t.range</code>	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.
<code>overlay</code>	Overlay the divergence limit plot on an existing plot. Default is FALSE.
<code>...</code>	Additional parameters to pass to plot or lines

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

```
data(sp500.subset)

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

divergence(sp500.subset, 25, filter=getCorFilter.RMT())
divergence(sp500.subset, 25, filter=getCorFilter.Shrinkage())
```

 filter.RMT

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

```
filter.RMT(h, hint, ..., type = 'kernel')

getRandomMatrix(m, t)

mp.theory(Q, sigma, e.values = NULL, steps = 200)

mp.density.hist(h, breaks = NULL, cutoff = 0.01)

mp.density.kernel(h, ...)
## Default S3 method:
mp.density.kernel(h, ...)
## S3 method for class 'returns':
mp.density.kernel(h, ...)
## S3 method for class 'covariance':
mp.density.kernel(h, ...)
## S3 method for class 'correlation':
mp.density.kernel(h, adjust = 0.2, kernel = 'e', ...)

# Fit the appropriate MP curve to the data. This will estimate Q and sigma.
mp.fit.hist(hist)
mp.fit.kernel(hist)

# Marcenko-Pastur theoretical minimum and maximum eigenvalues
mp.eigen.max(Q, sigma)
```

```

mp.eigen.min(Q, sigma)

# Generate eigenvalues for theoretical MP distribution
mp.lambdas(Q, sigma, steps)

# This provides the density of the eigenvalues
mp.rho(Q, sigma, e.values)

r.normalize(h)

cor.empirical(h)

denoise(hist, lambda.plus = 1.6, h = NULL)

```

Arguments

<code>h</code>	A generic tawny object that represents either a returns stream, covariance matrix or correlation matrix.
<code>hint</code>	A hint to the optimizer for fitting a theoretical curve to the empirical distribution
<code>type</code>	The type of density calculation to use, either kernel or hist
<code>m</code>	Scalar representing the number of assets in a portfolio
<code>t</code>	Scalar representing the number of observations
<code>breaks</code>	The breaks specification for building a histogram
<code>cutoff</code>	A threshold to eliminate eigenvalues below this value. Used to improve the performance when correlations are too high and negative eigenvalues exist.
<code>adjust</code>	Option for density to scale bandwidth
<code>kernel</code>	Option for density to define what kernel estimator to use
<code>hist</code>	Histogram-like object that contains eigenvalues, eigenvectors, and density information
<code>Q</code>	Ratio of data points per time series
<code>sigma</code>	Standard deviation of entries in <code>h</code>
<code>e.values</code>	Eigenvalues
<code>steps</code>	Number of steps to use in plotting theoretical MP distribution
<code>lambda.plus</code>	The upper eigenvalue bound used to cutoff noisy eigenvalues. The default of 1.6 is based on empirical data and is not optimal.
<code>...</code>	Further arguments to pass to <code>mp.density.*</code> function

Details

The functions described here are the individual components for filtering a correlation matrix using Random Matrix Theory. This is only of interest to those looking to delve deeper into the mechanics of RMT and/or creating custom behavior from these building blocks.

When using the kernel density estimate (the default), it is important to set `n` to a reasonable number. In the implementation, the kernel estimator's default is used which may not be optimum. For shorter windows it is suitable but larger windows (e.g. greater than 500 days) requires manually setting `n`. A value that seems to work well is 4096. See `density` for more information. In the future, this may be changed to include a heuristic to determine a good value for `n`.

Value

A cleaned version of the eigenvalues is returned by `clean.bouchaud`. The functions `mp.eigen.min/max` return scalars while `mp.lambdas` returns a vector and `mp.rho` returns either a scalar or a vector.

Author(s)

Brian Lee Yung Rowe

See Also

`tawny`, `optimizePortfolio`

Examples

```
# Fit the appropriate MP curve to the data. This will estimate Q and sigma.
# hist is a histogram object
h <- getRandomMatrix(1000, 6000)
hist <- mp.density.hist(h)
o <- optim(c(6,1), mp.fit.hist(hist))

# Calculate and plot the theoretical density distribution
rho <- mp.theory(6,1)

clean <- filter.RMT(h, c(6,1))
```

```
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 = func)
ensure(serie, src = 'FRED', reload = FALSE, ...)
```

Arguments

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

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

```
# 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: `getCorFilter.RMT` for using random matrix theory, `getCorFilter.Shrinkage` 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 `getCorFilter.RMT`, one can choose whether a histogram or a kernel density estimator is used to calculate the probability density function. With `getCorFilter.Shrinkage` one can select a constant correlation model or the identity as the model.

Usage

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

getCorFilter.Shrinkage(prior.fun = cov.prior.cc, ...)

getCorFilter.ShrinkageM(market, prior.fun = cov.prior.cc, ...)

getCorFilter.RMT(hint = c(4,1), ...)

getCorFilter.Raw()

getCorFilter.Sample()

p.optimize(h, c.denoised)

classify(x)
```

Arguments

<code>h</code>	A zoo object representing a portfolio with dimensions $T \times M$
<code>window</code>	The size of the window to use when generating a correlation matrix
<code>cor.gen</code>	A function that takes a portfolio with window observations and generates a correlation matrix
<code>prior.fun</code>	The function that generates the prior/model covariance matrix for shrinkage estimation
<code>hint</code>	A hint for the correct value of Q and σ for random matrix theory
<code>market</code>	The market to remove from the returns stream
<code>c.denoised</code>	A cleaned correlation matrix
<code>...</code>	Additional parameters to pass to the <code>cor.gen</code> function
<code>x</code>	An object representing either a returns covariance or correlation matrix

Details

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 `getCorFilter.RMT` function or `getCorFilter.Shrinkage` 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

```

data(sp500.subset)
h <- sp500.subset
ws <- optimizePortfolio(h, 190, getCorFilter.Sample() )
ws <- optimizePortfolio(h, 190, getCorFilter.Raw() )
ws <- optimizePortfolio(h, 190, getCorFilter.RMT() )
ws <- optimizePortfolio(h, 190, getCorFilter.Shrinkage() )

# This is computationally faster although the convenient approach is to pass
# in the character symbol directly: getCorFilter.Shrinkage('^GSPC')
m <- getPortfolioReturns('^GSPC', obs=1000, end='2009-02-27')
ws <- optimizePortfolio(h, 190, getCorFilter.ShrinkageM(m) )

```

plotPerformance	<i>Calculate some portfolio statistics and compare with other portfolios or benchmarks</i>
-----------------	--

Description

The group of compare.* functions plot the performance of additional benchmarks such as an equal weighted portfolio over the same time period or a benchmark market (e.g. DJIA or S&P500).

Usage

```

portfolioReturns(h, weights)

portfolioPerformance(p, rf.rate = 0.01)

plotPerformance(h, weights, window = NULL, rf.rate = 0.01, new.plot = TRUE, y.min =

compare.EqualWeighted(h, window, color = "#342a31", ...)

compare.Market(market, obs, window, end = Sys.Date(), color = "#44bc43", ...)

```

Arguments

h	Asset returns as used elsewhere in this package
p	Portfolio returns (h %*% weights)
weights	Portfolio weights over the given time period
window	Size of window used in other calculations
obs	Number of observations to retrieve
end	End date of data to retrieve
rf.rate	A risk free rate to use when calculating the sharpe ratio
new.plot	Pass-through to new in plot
y.min	Minimum y value for plot

y.max	Maximum y value for plot
bg	
name	The name of the series to use in the legend
color	Color of the plot
colors	Current colors (and names) of lines plotted on the chart
market	A string or zoo/xts object representing the market to use
...	Additional parameters to plot/lines

Details

These functions are provided to assist in the analysis of the resultant portfolio weights generated by the package. The function `plotPerformance` plots the performance of a portfolio and calculates basic statistics. For more detailed analysis, use a specialized package such as `PerformanceAnalytics`.

The equal weighted portfolio is simply the same portfolio weight for each asset in the specified portfolio. For this to be meaningful, the same portfolio and window are passed into the function as with the `optimizePortfolio` call thus ensuring that the equal weighted portfolio and corresponding returns are constructed correctly.

When comparing to the market portfolio, the number of observations need to be specified again so the market portfolio and corresponding returns are constructed properly.

Value

A list of portfolio statistics.

returns	The daily portfolio returns
perf	Cumulative return over time period
stdev	Standard deviation of portfolio over time period
avg.return	Average return of portfolio
sharpe.ratio	Sharpe ratio of portfolio

Author(s)

Brian Lee Yung Rowe

Examples

```
data(sp500.subset)
ys <- sp500.subset
ws <- optimizePortfolio(ys, 150, getCorFilter.RMT())
plotPerformance(ys, ws, 150)
compare.EqualWeighted(ys, 150)
compare.Market('^GSPC', 200, 150)
```

`sp500`*A (mostly complete) subset of the SP500 with 250 data points*

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`*A subset of the SP500 with 200 data points*

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, 15
sp500.subset, 15

*Topic **package**

tawny-package, 2

*Topic **ts**

cov.shrink, 4
divergence, 6
filter.RMT, 7
getPortfolioReturns, 9
optimizePortfolio, 11
plotPerformance, 13
tawny-package, 2

classify (*optimizePortfolio*), 11

clean.bouchaud (*filter.RMT*), 7

compare.EqualWeighted
(*plotPerformance*), 13

compare.Market (*plotPerformance*),
13

cor.empirical (*filter.RMT*), 7

cor.mean (*cov.shrink*), 4

cov.prior.cc (*cov.shrink*), 4

cov.prior.identity (*cov.shrink*), 4

cov.sample (*cov.shrink*), 4

cov.shrink, 4

denoise (*filter.RMT*), 7

divergence, 6

divergenceLimit.kl (*divergence*), 6

ensure (*getPortfolioReturns*), 9

filter.RMT, 3, 7

getCorFilter.Raw
(*optimizePortfolio*), 11

getCorFilter.RMT
(*optimizePortfolio*), 11

getCorFilter.Sample
(*optimizePortfolio*), 11

getCorFilter.Shrinkage
(*optimizePortfolio*), 11

getCorFilter.ShrinkageM
(*optimizePortfolio*), 11

getIndexComposition
(*getPortfolioReturns*), 9

getPortfolioReturns, 3, 9

getRandomMatrix (*filter.RMT*), 7

mp.density.hist (*filter.RMT*), 7

mp.density.kernel (*filter.RMT*), 7

mp.eigen.max (*filter.RMT*), 7

mp.eigen.min (*filter.RMT*), 7

mp.fit.hist (*filter.RMT*), 7

mp.fit.kernel (*filter.RMT*), 7

mp.lambdas (*filter.RMT*), 7

mp.rho (*filter.RMT*), 7

mp.theory (*filter.RMT*), 7

optimizePortfolio, 3, 5, 9, 11

p.optimize (*optimizePortfolio*), 11

plotDivergenceLimit.kl
(*divergence*), 6

plotPerformance, 13

portfolioPerformance
(*plotPerformance*), 13

portfolioReturns
(*plotPerformance*), 13

r.normalize (*filter.RMT*), 7

shrinkage.c (*cov.shrink*), 4

shrinkage.intensity (*cov.shrink*),
4

shrinkage.p (*cov.shrink*), 4

shrinkage.r (*cov.shrink*), 4

sp500, 15

sp500.subset, 15

stabilityLimit.kl (*divergence*), 6

tawny, [5](#), [9](#), [10](#)

tawny ([tawny-package](#)), [2](#)

tawny-package, [2](#)