Standard Errors for Risk and Performance Estimators

Anthony Christidis, Doug Martin

December 29, 2019

Abstract

The `PerformanceAnalytics` package allows users to perform risk analysis of financial instruments or portfolios. In general, the package requires returns data and allows users to compute risk estimators (including the standard deviation, semi-standard deviation, value-at risk and expected shortfall), and performance estimators (such as the Sharpe ratio, Sortino ratio, and expected shortfall ratio). However, users have no way of knowing the statistical accuracy of the estimates. A new method for computing accurate standard errors of risk and performance measures for serially correlated or uncorrelated returns has been developed using a sophisticated method based on the spectral density of the influence-function (IF) returns, and has been implemented in the `RPESE` package. This capability has been integrated in the `PerformanceAnalytics` package, and this vignette provides basic instruction on how to use the standard errors capability of the `PerformanceAnalytics` package.

1 Introduction

The current finance industry practice in reporting risk and performance estimates for individual assets and portfolios seldom includes reporting estimate standard errors (SEs). For this reason, consumers of such reports have no way of knowing the statistical accuracy of the estimates. As a leading example, one seldom sees SE's reported for Sharpe ratios, and consequently cannot tell whether or not two Sharpe ratios for two different portfolio products are significantly different. Chen and Martin [2018], henceforth (CM), have developed a method to compute accurate standard errors for these risk and performance estimators for different cases: (1) when returns are serially correlated as well as when they are uncorrelated, and (2) account for fat-tailed and skewed non-normality of returns distributions. The package

`RPESE` was developed to implement this new powerful method by (CM) and this capability was integrated in the `PerformanceAnalytics` package due to its high popularity and usage in the finance community.

`PerformanceAnalytics` supports computing SEs for the six risk estimators shown in Table 1, and the eight performance estimators shown in Table 2. For each of the names in the Name column of the two tables, there is a corresponding `R` function with a similar name in `PerformanceAnalytics`, except for $LMP1$ and $LPM2$ in Table 1 there is a single function with an optional argument to choose between these two risk estimators, and for $SorR.\mu$ and $SorR.c$ in Table 2 there is a single function with an optional argument to choose between these two performance estimators.

The `PerformanceAnalytics` package uses the `RPESE` package to compute the standard errors of the estimators in Tables 1 and 2, and computes the point estimates internally. The main function of `RPESE` is to compute the standard errors of the point estimates listed in Tables 1 and 2, using the new method that we now briefly describe.

Name	**Estimator Description**
SD	Sample standard deviation
SemiSD	Semi-standard deviation
LPM1	Lower partial moment of order 1
LPM2	Lower partial moment of order 2
ES	Expected shortfall with tail probability α
VaR	Value-at-risk with tail probability α

Table 1: Risk Estimator Names and Descriptions

Name	Estimator Description
Mean	Sample mean
SR	Sharpe ratio
SorR.μ	Sortino ratio with threshold the mean
SorR.c	Sortino ratio with threshold a constant c
ESratio	Mean excess return to ES ratio with tail probability α
VaRratio	Mean excess return to VaR ratio with tail probability α
RachevRatio	Rachev ratio with lower upper tail probabilities α and β
OmegaRatio	Omega ratio with threshold c

Table 2: Performance Estimator Names and Descriptions

New Method: Computing Standard Errors of Risk and Performance Estimators

The essence of the new method is as follows. Given a risk or performance estimators, such as the Sharpe ratio, the time series of returns used to compute the estimate are transformed using the *influence function* (IF) of the estimator. For an introduction to influence functions for risk and performance estimators, and derivations of the influence functions of the estimators in Tables 1 and 2, see the paper Zhang et al. [2019]. In CM, it is shown that risk and performance estimators can be represented as the sample mean of the time series of IF transformed returns.

It is a well-known result that an appropriately standardized (with respect to sample size) sum of a stationary time series has a variance that is approximated by the spectral density of the time series at zero frequency, with the approximation becoming exact as the sample size tends to infinity. Using this result, computing the standard error of a risk or performance estimator reduces to estimating the spectral density at zero frequency of a standardized sum of the influence-function transformed returns.

CM developed an effective method of doing so based on first computing the periodogram of the IF transformed returns, and then using a regularized generalized linear model (GLM) method for Exponential and Gamma distributions, fitting a polynomial to the periodogram values. The regularization method used is an elastic net (EN) penalty, well-known in the machine learning community, that encourages sparsity of coefficients. The intercept of such

GLM fitting provides an estimate of the spectral density at zero frequency, and hence a risk or performance estimator standard error.

The interested reader can find further details in the vignette of the `RPESE` package or the CM paper.

2 Packages Involved in the Computation of Standard Errors

In order to compute the standard errors of the point estimates in `PerformanceAnalytics`, the `RPESE` package must be installed by the user. However, note that if a user does not have the `RPESE` package downloaded, then the `PerformanceAnalytics` package will still work with all its other capabilities. In the source code of `PerformanceAnalytics`, the `RPESE` package is only suggested. If a user attempts to compute standard errors without the required package, an error will be returned with the required instructions to download `RPESE`.

On a side note, the overall structure of the packages involved in the computation of standard errors for risk and performance estimators in `PerformanceAnalytics` are depicted in Figure 1. First, `RPESE` makes use of the following two new packages:

- `RPEIF` (Risk and Performance Estimators Influence Functions)
- `RPEGLMEN` (Risk and Performance Estimators Generalized Linear Model fitting with Elastic Net, for Exponential and Gamma distributions)

The purpose of `RPEIF` is to provide the analytic formulas of influence functions in support of computing the IF transformed returns for the risk and performance estimators. For each risk and performance estimator in Tables 1 and 2, the `RPEGLMEN` package fits an EN regularized GLM polynomial fit to the periodogram of the time series of IF-transformed returns, using a GLM for Exponential distributions or Gamma distributions. Figure 1 shows the relationship between the above two packages and the overall `RPESE` package.

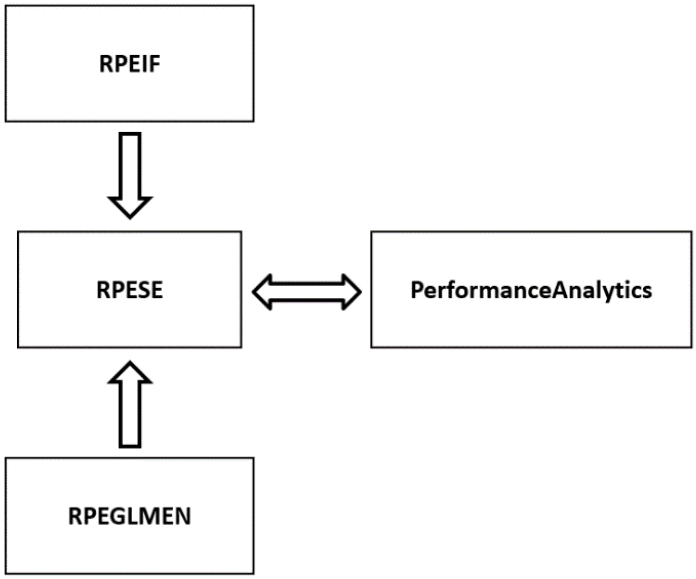

Figure 1: Packages Relations between `RPEIF`, `RPGLMEN`, `RPESE` and `PerformanceAnalytics`

If a user of `PerformanceAnalytics` wishes to compute standard errors of risk and performance estimators, only the `RPESE` packages is needed. The other two packages, `RPEIF` and `RPEGLMEN` will be installed automatically if `RPESE` is installed (they are imported packages by `RPESE`).

3 How Compute Standard Errors in `PerformanceAnalytics`

In the following sections, we show how to use the functions in `PerformanceAnalytics` to compute standard errors of risk and performance estimators using time series of monthly hedge fund returns contained in the `PerformanceAnalytics` package.

3.1 Installing and Loading RPESE and Loading an Examples Data Set

As previously mentioned, to use the standard errors computation capability in `PerformanceAnalytics`, the `RPESE` package must be installed. The `RPESE` package can be installed form CRAN as follows:

```
install.packages("RPESE")
```

We will use the `xts` data set `edhec` of hedge fund returns, contained in `PerformanceAnalytics`, in demonstrating the functionality of `PerformanceAnalytics`. The following code loads the `edhec` data, confirms the object's class, lists the names of the hedge funds, and displays the range of dates of the data.

```
library(PerformanceAnalytics)
data(edhec, package='PerformanceAnalytics')
class(edhec)

## [1] "xts" "zoo"

names(edhec)

##  [1] "Convertible Arbitrage"  "CTA Global"
##  [3] "Distressed Securities"  "Emerging Markets"
##  [5] "Equity Market Neutral"  "Event Driven"
##  [7] "Fixed Income Arbitrage" "Global Macro"
##  [9] "Long/Short Equity"      "Merger Arbitrage"
## [11] "Relative Value"         "Short Selling"
## [13] "Funds of Funds"

library(xts) # Need this for the next line and later use of plot.zoo
range(index(edhec))

## [1] "1997-01-31" "2019-07-31"
```

Since the hedge fund names are too long for convenient display, the following code is used to create shorter two or three letter names:

```
names(edhec) <- c("CA", "CTA", "DIS", "EM","EMN", "ED", "FIA",
                  "GM", "LS","MA", "RV", "SS", "FOF")
```

3.2 Functions with Standard Errors Computation Availability in `PerformanceAnalytics`

The functions with standard errors availability in `PerformanceAnalytics` are given below.

Name	`PerformanceAnalytics` **Function**
SD	`StedDev`
SemiSD	`SemiSD`
LPM1	`lpm` with argument `n=1`
LPM2	`lpm` with argument `n=2`
ES	`ES`
VaR	`VaR`

Table 3: Risk Estimator Names and Descriptions

Name	`PerformanceAnalytics` **Function**
Mean	`mean.arithmetic`
SR	`SharpeRatio` with argument `FUN="StdDev"`
SorR.μ	`Sortino` with argument `threshold="mean"`
SorR.c	`Sortino` with argument `threshold="const"`
ESratio	`SharpeRatio` with argument `FUN="ES"`
VaRratio	`SharpeRatio` with argument `FUN="VaR"`
RachevRatio	`RachevRatio`
OmegaRatio	`Omega`

Table 4: Performance Estimator Names and Descriptions

3.3 Basic Functionality

The arguments risk and performance measures in `PerformanceAnalytics` are all quite different. For example, below are the arguments of the `StdDev` and `ES` (expected shortfall) functions using the `args` function:

```
args(StdDev)
```

```
## function (R, ..., clean = c("none", "boudt", "geltner", "locScaleRob"),
##     portfolio_method = c("single", "component"), weights = NULL,
##     mu = NULL, sigma = NULL, use = "everything", method = c("pearson",
##         "kendall", "spearman"), SE = FALSE, SE.control = NULL)
## NULL
```

```
args(ES)
```

```
## function (R = NULL, p = 0.95, ..., method = c("modified", "gaussian",
##     "historical"), clean = c("none", "boudt", "geltner", "locScaleRob"),
##     portfolio_method = c("single", "component"), weights = NULL,
##     mu = NULL, sigma = NULL, m3 = NULL, m4 = NULL, invert = TRUE,
##     operational = TRUE, SE = FALSE, SE.control = NULL)
## NULL
```

However, for the standard errors integration in `PerformanceAnalytics`, only two arguments are of interest: `SE` and `SE.control`. The other arguments are described in further details in the corresponding documentation and vignettes of `PerformanceAnalytics`. We first demonstrate the most basic usage for returning standard errors using the expected shortfall for the convertible arbitrage (CA) hedge fund as an example. To return the standard errors of the points estimates, we set the argument `SE=TRUE`.

```
# Expected shortfall SE computation for a single hedge fund
ESout.CA <- t(ES(edhec$CA, SE=TRUE))
ESout.CA
```

```
##             ES       IFiid      IFcor
## CA -0.08990506 0.008973782 0.01580664
```

The first column contains the point estimates for the managers data, and the second and third column contain the standard errors for two different methods, ‘‘IFiid’’ and ’’IFcor’’ as seen in the output above. All possible choices, which can be set as demonstrated in Section 3.4, are the following:

- "IFiid": This results in an influence function (IF) method based computation of a standard error assuming i.i.d. returns
- "IFcor": This is the basic IF method computation of a standard error that takes into account serial correlation in the returns
- "IFcorAdapt": This IF based method adaptively interpolates between IFcor and IFcorPW to better account for serial correlation in the returns than with either IFcor or IFcorPW alone
- "IFcorPW": This IF based method uses pre-whitening of the IF transformed returns and is useful when serial correlation is large
- "BOOTiid": This choice results in computing a bootstrap standard error assuming i.i.d. returns
- "BOOTcor": This choice uses a block bootstrap method to compute a standard error that takes into account serial correlation of returns.

The two default choices of methods are:

- ‘‘IFiid’’ and ’’IFcor’’ for risk estimators, and for performance estimators when returns serial correlation are known to be small
- ‘‘IFiid’’ and ’’IFcorAdapt’’ for performance estimators when returns correlations are unknown and may be large

The value of including IFiid, along with IFcor and IFcorAdapt is that it allows the user to see whether or not serial correlation results in a difference in the standard error that assumes i.i.d. returns and the standard error that takes into account serial correlation. If there is no serial correlation there will not be much difference, but if there is serial correlation the difference can be considerable.

The BOOTiid and BOOTcor methods are provided for users who want to see how these bootstrap methods of computing standard errors compare with the IF based methods. Our

experience to date indicates that `BOOTiid` generally agrees quite well with `IFiid`, but that `BOOTcor` is not as consistent in giving values similar to those of `IFcor`.

In this case the `IFcor` standard error is considerable 76% larger than that of the `IFiid` standard error.

The risk and performance estimator functions allow you to return the standard errors for more than one asset or portfolio, e.g. a portfolio of assets, at the same time. The following code results in standard errors for all thirteen of the `edhec` hedge funds.

```
# Expected shortfall SE computation for all hedge funds in data set
ESout <- t(ES(edhec, SE=TRUE))
ESout


##              ES       IFiid       IFcor
## CA  -0.08990506 0.008973782 0.015806639
## CTA -0.04113351 0.003459795 0.003433112
## DIS -0.05187065 0.006728856 0.010162329
## EM  -0.11608981 0.013914916 0.016351096
## EMN -0.03799789 0.004262266 0.004570448
## ED  -0.05138432 0.006000260 0.007780399
## FIA -0.05045308 0.007971074 0.013084620
## GM  -0.01643822 0.002098174 0.002105147
## LS  -0.04393666 0.005082402 0.007254537
## MA  -0.02993162 0.003892194 0.004335635
## RV  -0.04419635 0.004975177 0.009392001
## SS  -0.07192478 0.009600362 0.010968308
## FOF -0.03998667 0.005092672 0.005353737
```

Help files for the functions in the `PerformanceAnalytics` are available in the usual ways. For example, you can get the help file for the expected shortfall function using the following code.

```
?ES
help(ES)
```

3.4 Controlling Standard Errors Computation Using the `RPESE.control` Function

The other argument used in the computation of standard errors for risk and performance estimators is `SE.control`. This argument is used to control the parameters in the computation of standard errors, namely: the standard error methods used (as listed in Section 3.3), the outlier cleaning option, the model fitting method, the frequency decimation or truncation option, and the adaptive correlation standard errors tuning parameters.

The `SE.control` argument should be set using the `RPESE.control` function. The arguments for this function are given below.

```
args(RPESE.control)
```

Once can also look at the `RPESE.control` documentation for further details.

```
help(RPESE.control)
```

The arguments for this function are:

- `"measure"`: This argument takes one of the optinos `"Mean"`, `"SD"`, `"VaR"`, `"ES"`, `"SR"`, `"SoR"`, `"ESratio"`, `"VaRratio"`, `"LPM"`, `"OmegaRatio"`, `"SemiSD"`, `"RachevRatio"` and sets the other arguments to the function with the default for the measure used. If this argument is ignored, then the default used is `"Mean"` as indicated in the documentation. If this argument is used and the other arguments listed below are also used, then the default from the measure is overwritten by the user set option. See example code for demonstration.
- `"se.method"`: This argument controls which of the standard errors method as listed in Section 3.3 to use.
- `"cleanOutliers"`: Argument `TRUE` or `FALSE` (default) to determine if the returns data should be cleaned using a robust filter.
- `"fitting.method"`: Distribution used in the standard errors computation. Should be one of `"Exponential"` (default) or `"Gamma"`.
- `"freq.include"`: Frequency domain inclusion criteria. Must be one of `"All"` (default), `"Decimate"` or `"Truncate"` choices.

- "freq.par": Percentage of the frequency used if "freq.include" is "Decimate" or "Truncate." Default is 0.5.
- "a" and "b": Adaptive parameters for the standard errors computation if one of the methods is "IFcorAdapt" choice.

For example, for the control parameters for the expected shortfall, we can use the default.

```
ES.control <- RPESE.control(measure="ES")
ES.control
```

Note that the return value is a list with the control parameters. We can change them manually as follows.

```
ES.control$cleanOutliers <- TRUE
```

If we want to enforce some parameters directly by the function call, we can use the relevant arguments of RPESE.control directly.

```
ES.control <- RPESE.control(measure="ES", cleanOutliers=TRUE,
                            freq.include="Decimate")
ES.control
```

This return value is to be used in the SE.control argument if SE=TRUE.

```
# Expected shortfall SE computation with control parameters
ESout <- t(ES(edhec, SE=TRUE, SE.control=ES.control))
ESout
```

```
##              ES       IFiid       IFcor
## CA  -0.08990506 0.008973782 0.003723893
## CTA -0.04113351 0.003459795 0.003549186
## DIS -0.05187065 0.006728856 0.006104730
## EM  -0.11608981 0.013914916 0.007730818
## EMN -0.03799789 0.004262266 0.001670694
## ED  -0.05138432 0.006000260 0.003387294
## FIA -0.05045308 0.007971074 0.001991072
```

```
## GM  -0.01643822 0.002098174 0.002073852
## LS  -0.04393666 0.005082402 0.006085146
## MA  -0.02993162 0.003892194 0.002277884
## RV  -0.04419635 0.004975177 0.003087658
## SS  -0.07192478 0.009600362 0.011187394
## FOF -0.03998667 0.005092672 0.003038685

# We can use RPESE.control directly in the function call
ESout <- t(ES(edhec,
              SE=TRUE,
              SE.control=RPESE.control(measure="ES",
                                       se.method=c("IFiid",
                                                   "IFcor",
                                                   "BOOTiid",
                                                   "BOOTcor"))))
ESout

##               ES       IFiid       IFcor
## CA  -0.08990506 0.008973782 0.015804837
## CTA -0.04113351 0.003459795 0.003433112
## DIS -0.05187065 0.006728856 0.010248921
## EM  -0.11608981 0.013914916 0.016426163
## EMN -0.03799789 0.004262266 0.004570448
## ED  -0.05138432 0.006000260 0.007778517
## FIA -0.05045308 0.007971074 0.013084620
## GM  -0.01643822 0.002098174 0.002105139
## LS  -0.04393666 0.005082402 0.007471442
## MA  -0.02993162 0.003892194 0.004335495
## RV  -0.04419635 0.004975177 0.009393838
## SS  -0.07192478 0.009600362 0.010968308
## FOF -0.03998667 0.005092672 0.007237515
```

3.5 Outlier Cleaning

There is also the outlier cleaning functionality in the `RPEIF` package that is fully described in Section 7 of CM, and is available in `RPESE`. Here we illustrate the use of the outlier cleaning

facility in terms of the influence function transformed returns for the sample mean estimator. It is shown in Section 2 of CM that the influence function for the sample mean estimator is $IF(r;\mu) = r - \mu$. Thus the IF transformed returns time series, computed with the function `IF.mean` is just $r_t - \mu$, where μ would be replaced by the sample mean in applications. The function `IF.mean` is made accessible by loading the package RPEIF. The following code produces Figure 2, which illustrate the effect of outlier cleaning relative to no outlier cleaning for the `FIA` hedge fund returns.

```
library(RPEIF)
IFout <- IF.mean(returns = edhec[,"FIA"], cleanOutliers = F, IFprint = T)
IFout.clean <- IF.mean(returns = edhec[,"FIA"], cleanOutliers = T,
                       IFprint = T)
par(mfrow = c(2,1))
ylim = c(-.1,.035)
plot.zoo(IFout,type = "b",
         ylab = expression(paste("Returns - ",mu)),
         main = "FIA Returns", pch = 20, lwd = .8, cex = .9,
         ylim = ylim)
plot.zoo(IFout.clean,type = "b", ylab = expression(paste("Returns - ",mu)),
         main = "FIA Outlier Cleaned Returns", pch = 20, lwd = .8, cex = .9,
         ylim = ylim)
par(mfrow = c(1,1))
```

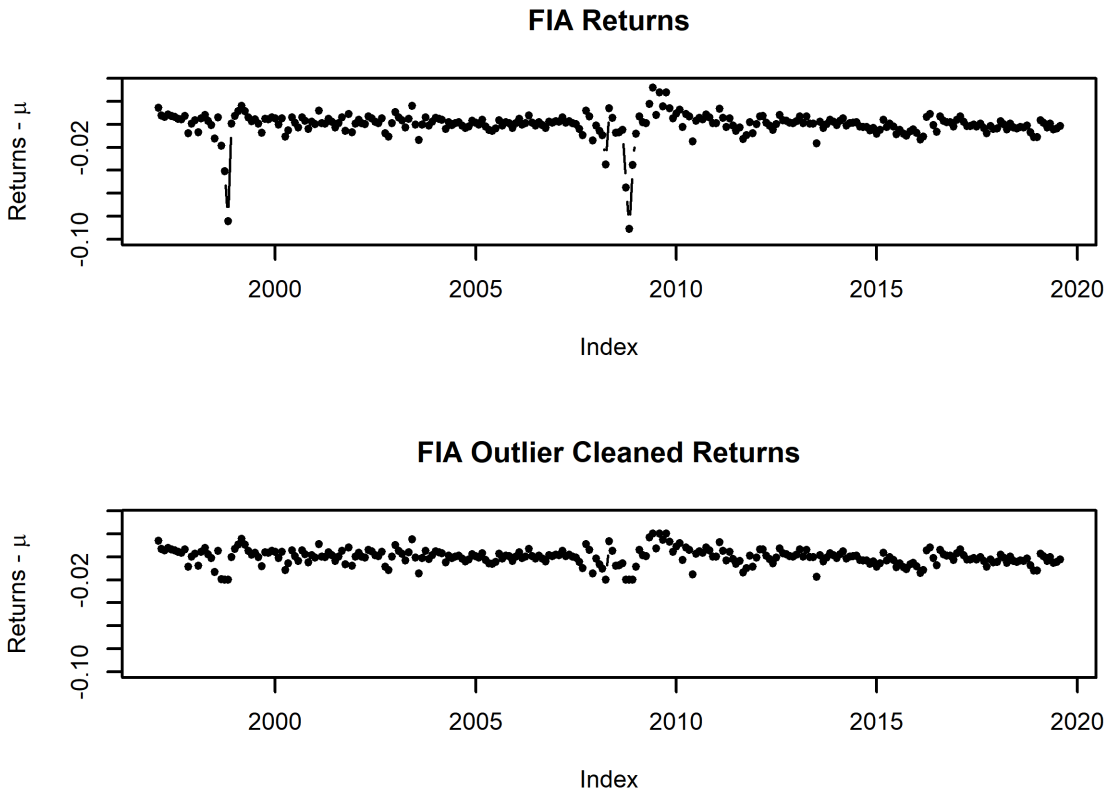

Figure 2: FIA Returns Versus Outlier Cleaned FIA Returns (with sample mean subtracted)

You can use the following code to compare expected shortfall SE's without and with outlier cleaning.

```
# IFcor SE results with outliers present and with outliers cleaned
ESout <- t(ES(edhec, SE=TRUE, SE.control=RPESE.control(measure="ES")))
ESout.clean <- t(ES(edhec, SE=TRUE, SE.control=RPESE.control(measure="ES",
                                                             cleanOutliers=T)))
clean.compare <- data.frame(ESout[,3], ESout.clean[,3])
names(clean.compare) <- c("With Outliers", "Outliers Cleaned")
row.names(clean.compare) <- names(edhec)
round(clean.compare,3)

##     With Outliers Outliers Cleaned
```

```
## CA          0.016           0.004
## CTA         0.003           0.003
## DIS         0.010           0.006
## EM          0.016           0.007
## EMN         0.005           0.002
## ED          0.008           0.003
## FIA         0.013           0.002
## GM          0.002           0.002
## LS          0.007           0.006
## MA          0.004           0.002
## RV          0.009           0.003
## SS          0.011           0.011
## FOF         0.007           0.003
```

It is not surprising that the SE's of the expected shortfall are smaller with outlier cleaning than with the outliers in the returns, as outliers generally inflate estimator variability.

3.6 Remark for Conflicting Arguments for Standard Errors Computation

There are some arguments in some of the `PerformanceAnalytics` functions that will be assumed to hold a certain value if the standard errors are computed. For example, recall the arguments of the `ES` function.

```
args(ES)


## function (R = NULL, p = 0.95, ..., method = c("modified", "gaussian",
##     "historical"), clean = c("none", "boudt", "geltner", "locScaleRob"),
##     portfolio_method = c("single", "component"), weights = NULL,
##     mu = NULL, sigma = NULL, m3 = NULL, m4 = NULL, invert = TRUE,
##     operational = TRUE, SE = FALSE, SE.control = NULL)
## NULL
```

The following arguments will be overwritten if `SE=TRUE` and they do not contain the values below.

- "`method`": This argument will be enforced to be "`historical`".

- "portfolio_method": This argument will be enforced to be "single".
- "invert": This argument will be enforced to be "FALSE".
- "clean": This argument will be enforced to be match the argument in the "SE.control" argument.

References

X. Chen and R. D. Martin. Standard errors of risk and performance measure estimators for serially correlated returns. 2018. URL https://ssrn.com/abstract=3085672.

S Zhang, R D Martin, and A A Christidis. Influence functions for risk and performance estimators. *Working paper*, 2019.