Standard Errors for Risk and Performance Estimators in PerformanceAnalytics

Anthony Christidis and Doug Martin

October 30, 2020

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 assessing 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) transformed 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 assessing the statistical accuracy of the estimates. As a leading example, SE's are seldom reported for Sharpe ratios, and consequently one 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 risk and performance estimators that properly reflects the impacts of: (1) returns that are serially correlated as well as when

they are uncorrelated, and (2) fat-tailed and skewed non-normality of returns distributions. The risk and performance standard errors computing package `RPESE` was developed to implement this new CM methodology, and this capability has been integrated in the highly popular `PerformanceAnalytics` package that is frequently used in the quantitative finance community.

The current release of `PerformanceAnalytics` supports computing SEs for the six risk estimators described in Table 1, and the eight performance estimators described in Table 2. The Name column in each of those two tables contains the name of the R function in the RPESE package, For each of the names in the Name column of the two tables, there is a corresponding `R` function in `PerformanceAnalytics`, except for $LMP1$ and $LPM2$ in Table 1 for which 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 for which there is a single function with an optional argument to choose between these two performance estimators. The names of the PerformanceAnalytics functions that correspond to the names in Tables 1 and 2 are provided in Tables 3 and 4 of Section 3.2.

The `PerformanceAnalytics` package internally computes the point estimates in Tables 1 and 2, and uses the `RPESE` package to compute the standard errors of the point estimates.

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
DSR	Downside Sharpe ratio
SoR	Sortino ratio with threshold a constant c or the mean
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

The New Method of 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 a risk or performance estimator can be represented as a 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, to fit a polynomial to the periodogram values. The regularization method used is an elastic net (EN) penalty that is well-known in the machine learning community, and 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 available at CRAN and in 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 many capabilities other than standard error computation. 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. As the figure indicates, `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.

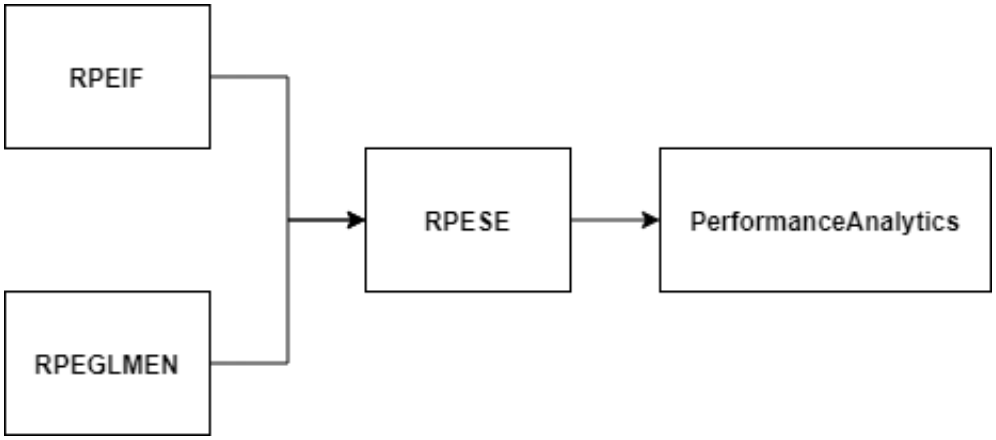

Figure 1: Packages Relations between `RPEIF`, `RPGLMEN`, `RPESE` and `PerformanceAnalytics`

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 to 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 the `edhec` 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" "2019-11-30"
```

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 PerformanceAnalytics Functions for Computing Standard Errors

The names of the PerformanceAnalytics for computing standard errors are provide in the second column, with the names of the corresponding RPESE functions in the first column.

Name	PerformanceAnalytics Function
SD	StdDev
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"
DSR	DownsideSharpeRatio or SharpeRatio with argument FUN="SemiSD"
SoR	SortinoRatio
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 of risk and performance estimator functions in PerformanceAnalytics are different for different functions, and sometimes quite extensive. For example, below we show 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 computing standard errors of the risk and performance estimators of Tables 3 and 4 in `PerformanceAnalytics`, only two arguments are two arguments besides the data set name are needed, namely the arguments `SE` and `SE.control`, and often one only uses the `SE` argument by settting SE=TRUE. This works for example when you want to compute the standard error of an expected shortfall estimate for the convertible arbitrage (CA) hedge fund, as follows.

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


##         ES      IFiid      IFcor
## CA 0.03655 0.01041347 0.01569438
```

The first column above contains the ES estimate for CA hedge fund, and the second and third column contain the standard errors for two different methods, "`IFiid`" and "`IFcor`". These are just two of the following possible choices, which can be set as demonstrated in Section 3.4:

- "`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, and can sometimes result in better accounting 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. For example, in the above case of computing SEs for the CA hedge fund, the IFcor standard error is a considerable 76% larger than that of the IFiid standard error.

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.

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. For example, the following code results in computing 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,
              method="historical", invert=FALSE))
ESout
```

```
##             ES       IFiid       IFcor
## CA  0.03655000 0.010413470 0.015504982
## CTA 0.04126429 0.003415791 0.003386796
## DIS 0.03669286 0.007122003 0.009564148
## EM  0.07236429 0.013827809 0.015966142
## EMN 0.01687857 0.004226422 0.004529794
## ED  0.03856429 0.006076204 0.007513395
## FIA 0.02825714 0.008998122 0.012997036
## GM  0.02062857 0.002072157 0.002078879
## LS  0.04220714 0.005286489 0.007383943
## MA  0.01914286 0.003859357 0.003877278
## RV  0.02465000 0.005643819 0.009296035
## SS  0.09682143 0.009579051 0.010838203
## FOF 0.03320714 0.005267224 0.007143345
```

Help files for the functions in the `PerformanceAnalytics` are available as usual. For functions in Tables 3 and 4 that were already in `PerformanceAnalytics (PA)` prior to the current release, the help file will be almost the same as before, with the only difference being that you will see the addtional arguments `SE = FALSE` and `SE.control = NULL`. You can confirm this claim, for example, for the ES function with the following code.

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

PA functions for computing SEs that are new with this release, will of course also have the optional arguments `SE = FALSE` and `SE.control = NULL`.

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

The argument `SE.control` in the `PA` risk and performance estimator function in Tables 3 and 4 is used to control the parameters in the computation of standard errors, namely: the

standard error methods used, as listed in Section 3.3, along with the outlier cleaning option, the model fitting method, the frequency decimation or truncation option, and the adaptive correlation standard errors tuning parameters described in this Section.

The `SE.control` argument should be set using the `RPESE.control` function, whose arguments are:

```
args(RPESE.control)


## function (estimator = c("Mean", "SD", "VaR", "ES", "SR", "DSR",
##     "SoR", "ESratio", "VaRratio", "SoR", "LPM", "OmegaRatio",
##     "SemiSD", "RachevRatio")[1], se.method = NULL, cleanOutliers = NULL,
##     fitting.method = NULL, freq.include = NULL, freq.par = NULL,
##     a = NULL, b = NULL)
## NULL
```

Here is what the `RPESE.control` arguments do:

- `"estimator"`: 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 methods listed in Section 3.3 are used.
- `"cleanOutliers"`: Argument `TRUE` or `FALSE` (default) to determine if the returns data outlier should be shrunk.
- `"fitting.method"`: Distribution used in `RPEGLMEN` fit method. Should be one of `"Exponential"` (default) or `"Gamma"`.
- `"freq.include"`: DFT frequencies inclusion criteria. Must be one of `"All"` (default), `"Decimate"` or `"Truncate"` choices. If the argument `"freq.include"` is set to `"Decimate"` or `"Truncate"`, a value of 0.5 is used for the `"freq.par"` argument: every second frequency is used in the decimation case, and only the first half of the frequencies are used in the truncation case.

- "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 when "IFcorAdapt" is chosen.

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

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

Note that the return value is a list with the control parameters, and we can change individual components of the list, as for example:

```
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(estimator="ES", cleanOutliers=TRUE,
                            freq.include="Decimate")
ES.control
```

This the above ES.control is to be used for the SE.control argument if SE=TRUE.

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

##              ES       IFiid       IFcor
## CA  0.03655000 0.010413470 0.003552308
## CTA 0.04126429 0.003415791 0.003507299
## DIS 0.03669286 0.007122003 0.006107129
## EM  0.07236429 0.013827809 0.007443200
## EMN 0.01687857 0.004226422 0.001770876
```

```
## ED  0.03856429 0.006076204 0.003345595
## FIA 0.02825714 0.008998122 0.002287958
## GM  0.02062857 0.002072157 0.002215368
## LS  0.04220714 0.005286489 0.005916010
## MA  0.01914286 0.003859357 0.002220002
## RV  0.02465000 0.005643819 0.003034097
## SS  0.09682143 0.009579051 0.009596103
## FOF 0.03320714 0.005267224 0.002974921
```

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

```
##             ES       IFiid       IFcor     BOOTiid     BOOTcor
## CA  0.03655000 0.010413470 0.015691636 0.009993180 0.013497573
## CTA 0.04126429 0.003415791 0.003386447 0.003086297 0.003436462
## DIS 0.03669286 0.007122003 0.009564148 0.007351608 0.007578429
## EM  0.07236429 0.013827809 0.015966104 0.012951732 0.014391666
## EMN 0.01687857 0.004226422 0.004529794 0.003882246 0.005228409
## ED  0.03856429 0.006076204 0.007513395 0.005533530 0.005693686
## FIA 0.02825714 0.008998122 0.012997036 0.008306053 0.010491074
## GM  0.02062857 0.002072157 0.002079201 0.001755466 0.001878030
## LS  0.04220714 0.005286489 0.007384051 0.004566721 0.005590183
## MA  0.01914286 0.003859357 0.004285057 0.003427259 0.003202854
## RV  0.02465000 0.005643819 0.009296035 0.004735891 0.006632636
## SS  0.09682143 0.009579051 0.010850244 0.008312536 0.015908306
## FOF 0.03320714 0.005267224 0.007147555 0.005404278 0.005867643
```

3.5 Outlier Cleaning

There is also an 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 is replaced by the sample mean in the SE computation implementation. 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))
```

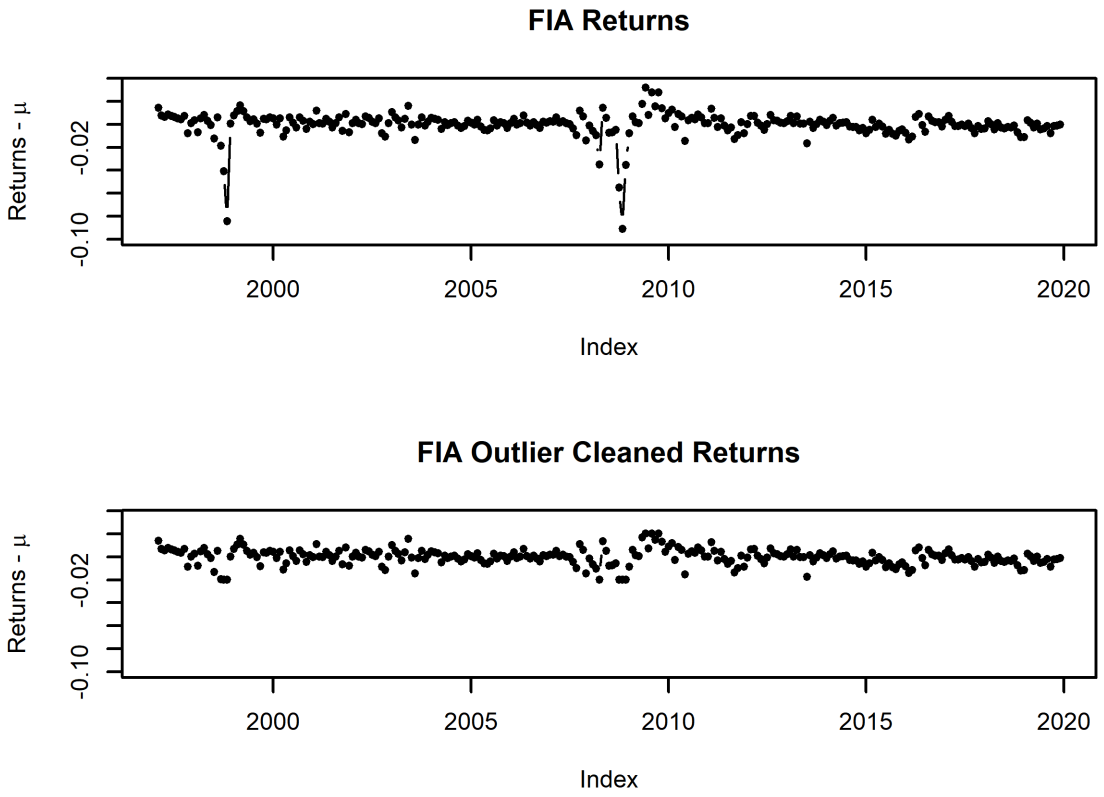

Figure 2: FIA Returns Versus Outlier Cleaned FIA Returns (with sample mean subtracted)

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(estimator="ES"),
              method="historical", invert=FALSE))
ESout.clean <- t(ES(edhec, SE=TRUE, SE.control=RPESE.control(estimator="ES",
                                                             cleanOutliers=T),
                    method="historical", invert=FALSE))
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             0.003
## CTA         0.003             0.003
## DIS         0.010             0.006
## EM          0.016             0.008
## 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 must hold a certain value when 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 `ES` arguments, if `SE=TRUE`, must contain the values below:

- "`method`": This argument must be "`historical`".

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

If the arguments are not as stated above, then NA values will be returned with a warning on the conflicting arguments. For example, see the code below.

```
ESout <- t(ES(edhec, SE=TRUE, method="modified", invert=TRUE))

## Warning:  To return SEs, "method" must be "historical".

## Warning:  To return SEs, "invert" must be FALSE.

ESout

##              ES IFiid IFcor
## CA  -0.08948631    NA    NA
## CTA -0.04116413    NA    NA
## DIS -0.05170691    NA    NA
## EM  -0.11567185    NA    NA
## EMN -0.03786594    NA    NA
## ED  -0.05128337    NA    NA
## FIA -0.05001278    NA    NA
## GM  -0.01638037    NA    NA
## LS  -0.04377715    NA    NA
## MA  -0.02992712    NA    NA
## RV  -0.04417445    NA    NA
## SS  -0.07076187    NA    NA
## FOF -0.03971531    NA    NA
```

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.