Risk and Performance Estimators Standard Errors (RPESE)

Anthony Christidis, Doug Martin, Xin Chen

March 9, 2021

Abstract

The Risk and Performance Estimators Standard Errors package (`RPESE`) implements a new method for computing accurate standard errors of risk and performance estimators when returns are serially correlated as well as when they are uncorrelated. The new method makes use of the representation of a risk or performance estimator as a summation of a time series of influence-function (IF) transformed returns, and computes estimator standard errors using a sophisticated method of estimating the spectral density at frequency zero of the time series of IF transformed returns. The `RPESE` package allows users to compute accurate standard errors for six risk estimators, including the standard deviation, semi-standard deviation, value-at risk and expected shortfall, and eight performance estimators, including the Sharpe ratio, Sortino ratio, and expected shortfall ratio. This vignette provides basic instruction on how to use the `RPESE` 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.

This motivated us to develop a Risk and Performance Estimator Standard Errors (`RPESE`) package for computing risk and performance estimator SE's that are accurate: (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. `RPESE` uses a new method for computing risk and performance estimators standard errors due to Chen and Martin [2018], henceforth (CM).

`RPESE` 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 `RPESE`, 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.

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
robLoc	Robust sample mean
SR	Sharpe ratio
DSR	Downside Sharpe ratio
SorR	Sortino ratio with threshold a constant $c$or mean μ
ESratio	Mean excess return to ES ratio with tail probability α
VaRratio	Mean excess return to VaR ratio with tail probability α
RachRatio	Rachev ratio with lower upper tail probabilities α and β
OmegaRatio	Omega ratio with threshold c

Table 2: Performance Estimator Names and Descriptions

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.

The New Method

The basic elements of the new method are as follows. For a given risk or performance estimator, 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 Zhang et al. [2019]. It turns out that risk and performance estimators can be represented as the sample mean of the time series of influence-function transformed returns. It is well-known 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. Thus, 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 influence-function transformed returns, and then using a regularized general 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) method that encourages sparsity of coefficients and is well-known in the machine learning literature. The intercept of such GLM fitting provides the an estimate of the spectral density at zero frequency, and hence a risk or performance estimator standard error. The interested reader can find the details in the CM paper.

2 RPESE Component Packages

The overall structure of `RPESE`, depicted in Figure 1, shows that `RPESE` makes use of the following two new packages:

- `RPEIF` (Influence Functions of risk and performance estimators)
- `RPEGLMEN` (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.

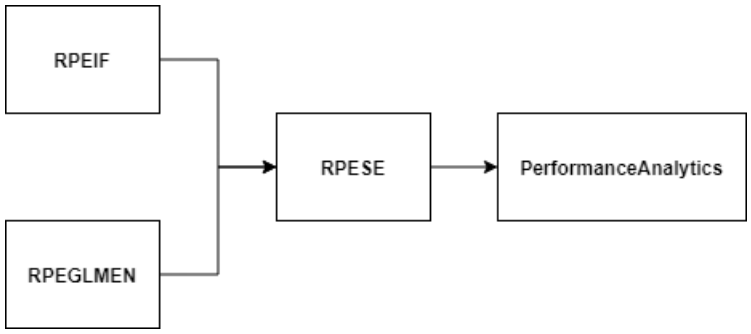

Figure 1: Packages Relations between `RPEIF`, `RPGLMEN`, `RPESE` and `PerformanceAnalytics`

3 How to Use RPESE

In the following sections, we show how to use the functions in `RPESE` 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

In order to use RPESE, you don't need to manually install any dependent packages as they will be installed automatically when RPESE is installed. You can install RPESE form CRAN as follows:

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

To load RPESE, use the code line:

```
library(RPESE)
```

We will use the xts data set edhec of hedge fund returns, contained in RPESE, to demonstrate the functionality of RPESE. 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.

```
data(edhec)
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 Functions in RPESE

To see what functions are contained in the `RPESE` package, use the code line:

```
ls("package:RPESE")
```

```
##  [1] "DSR.SE"          "edhec"           "ES.SE"           "ESratio.SE"
##  [5] "EstimatorSE"     "LPM.SE"          "Mean.SE"         "OmegaRatio.SE"
##  [9] "printSE"         "RachevRatio.SE" "robLoc.SE"       "SD.SE"
## [13] "SemiSD.SE"       "SoR.SE"          "SR.SE"           "VaR.SE"
## [17] "VaRratio.SE"
```

The twelve functions whose function name ends in `.SE` compute the standard error of the estimator corresponding to the first part of the function name. Note that there are only five risk estimators instead of the six in Table 1, which is because `LPM.SE` uses an optional argument to choose computing $LPM1$ or $LPM2$, and similarly there are only seven performance estimator instead of the eight in Table 2 because `SortinorRatio.SE` uses an optional argument to choose computing $SoRc$ or $SoR\mu$.

The function `printSE` is a utility function whose use is demonstrated below, and the function `estimatorSE` is another utility function whose use is illustrated in an example in the function's help file.

3.3 Basic Functionality

The arguments of the twelve standard error computation functions are all similar, and are illustrated below for the cases of the `SD.SE` and `SR.SE` functions using the `args` function:

```
args(SD.SE)

## function (data, se.method = c("IFiid", "IFcor", "IFcorAdapt",
##     "IFcorPW", "BOOTiid", "BOOTcor")[1:2], cleanOutliers = FALSE,
##     fitting.method = c("Exponential", "Gamma")[1], d.GLM.EN = 5,
##     freq.include = c("All", "Decimate", "Truncate")[1], freq.par = 0.5,
##     corOut = c("none", "retCor", "retIFCor", "retIFCorPW")[1],
##     return.coef = FALSE, ...)
## NULL
```

```
args(SR.SE)

## function (data, rf = 0, se.method = c("IFiid", "IFcor", "IFcorAdapt",
##     "IFcorPW", "BOOTiid", "BOOTcor")[c(1, 3)], cleanOutliers = FALSE,
##     fitting.method = c("Exponential", "Gamma")[1], d.GLM.EN = 5,
##     freq.include = c("All", "Decimate", "Truncate")[1], freq.par = 0.5,
##     corOut = c("none", "retCor", "retIFCor", "retIFCorPW")[1],
##     return.coef = FALSE, ...)
## NULL
```

The only argument that is required for the standard error computing functions is the `data` argument, and if only the data argument is supplied then the function uses the defaults of the other arguments. However the `se.method =` argument is particularly important, and the standard error computation methods for the various choices for this argument are as follows:

- "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.

Our 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

Note how these choices are made in the `method =` arguments for the `SD.SE` and `SR.SE` functions above.

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`.

As a simple example of using an SE function, the following code computes the standard error (SE) of the standard deviation estimator for the convertible arbitrage (CA) hedge fund, using the IFiid and IFcor methods:

```
# Standard deviation SE computation for a single hedge fund
SD.SE(edhec[, "CA"])


## $SD
## [1] 0.01629694
##
## $IFiid
## $IFiid$se
```

```
## [1] 0.002336667
##
## $IFiid$coef
## NULL
##
##
## $IFcor
## $IFcor$se
## [1] 0.004044631
##
## $IFcor$coef
## NULL
```

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

We note that the result returned by an SE function is a list, and so the result is printed using the default print method for a list. A more compact display of the results, with rounding to three digits by default, can be obtained using the `printSE` function, whose arguments are:

```
args(printSE)


## function (SE.data, round.digit = 3, round.out = TRUE)
## NULL
```

For example:

```
sdSE <- SD.SE(edhec[, "CA"])
printSE(sdSE)


##         SD IFiid IFcor
## [1,] 0.016 0.002 0.004
```

Of course if you want to compute any subset of the four IF based SE's and the two bootstrap SE's, you can do. For example, you can obtain all six of those standard error estimates as follows:

```
# Sharpe Ratio SE computation for single hedge fund
Sharpe.out <- SR.SE(edhec[,"CA"],
                    se.method = c("IFiid","IFcor",
                                  "IFcorAdapt","IFcorPW",
                                  "BOOTiid","BOOTcor"))
# Print output
printSE(Sharpe.out)

##         SR IFiid IFcor IFcorAdapt IFcorPW BOOTiid BOOTcor
## [1,] 0.338 0.096 0.117      0.173   0.203   0.096   0.146
```

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, using the first three method in the set of six methods.

```
# Sharpe Ratio SE computation for all hedge funds in data set
Sharpe.out <- SR.SE(edhec,
                    se.method = c("IFiid","IFcor","IFcorAdapt"))
printSE(Sharpe.out)

##         SR IFiid IFcor IFcorAdapt
## CA   0.338 0.096 0.117      0.173
## CTA  0.180 0.060 0.057      0.057
## DIS  0.392 0.080 0.108      0.121
## EM   0.194 0.069 0.084      0.084
## EMN  0.543 0.110 0.112      0.116
## ED   0.372 0.079 0.100      0.101
## FIA  0.377 0.113 0.134      0.159
## GM   0.371 0.054 0.057      0.057
## LS   0.315 0.066 0.079      0.079
## MA   0.560 0.092 0.098      0.098
## RV   0.504 0.097 0.117      0.120
## SS  -0.041 0.061 0.072      0.062
## FOF  0.275 0.066 0.083      0.084
```

Help files for the functions in the `RPESE` are available in the usual ways. For example, you can get the help file for Sharpe Ratio in both of the following ways.

```
`?`(SR.SE)
help(SR.SE)
```

3.4 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))
```

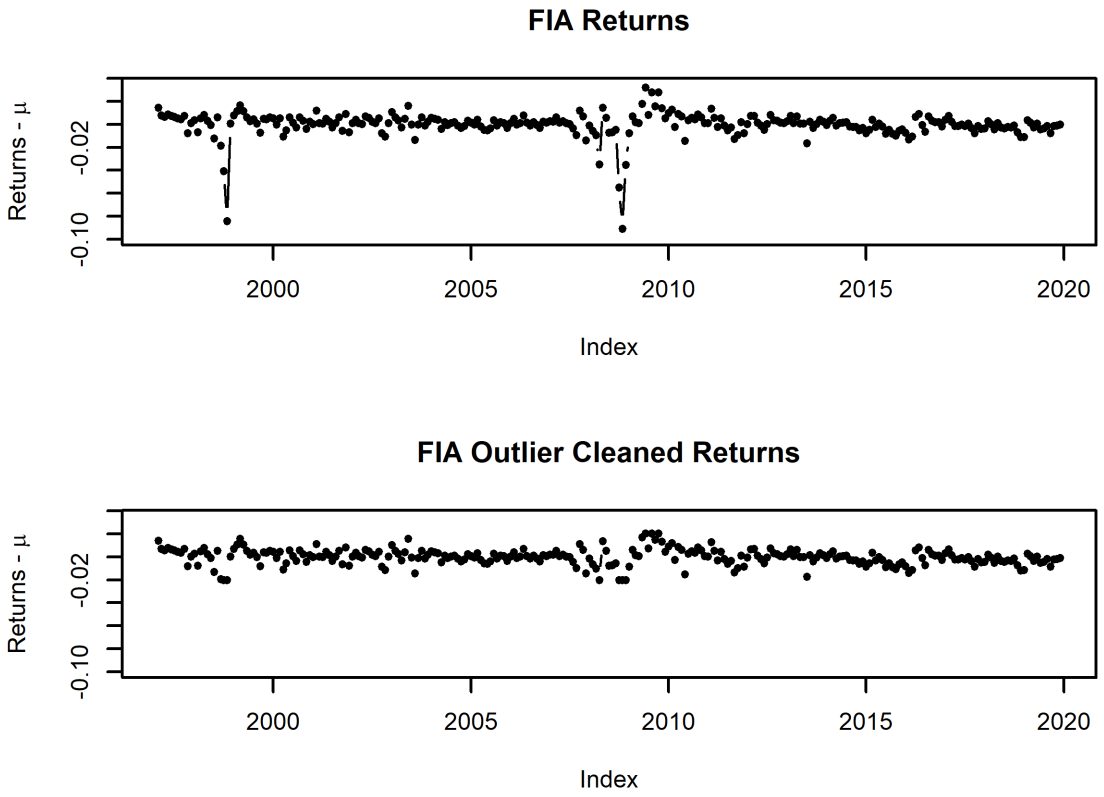

Figure 2: FIA Returns Versus Outlier Cleaned FIA Returns (with sample mean subtracted)

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

```
# IFcorAdapt SE results with outliers present and with outliers cleaned
SR <- SR.SE(edhec,se.method = "IFcorAdapt",cleanOutliers = F)
SRcl.out <- SR.SE(edhec,se.method = "IFcorAdapt",cleanOutliers = T)
clean.compare <- data.frame(SR$IFcorAdapt$se, SRcl.out$IFcorAdapt$se)
names(clean.compare) <- c("With Outliers", "Outliers Cleaned")
row.names(clean.compare) <- names(edhec)
round(clean.compare,3)

##     With Outliers Outliers Cleaned
```

```
## CA          0.163           0.108
## CTA         0.057           0.057
## DIS         0.121           0.112
## EM          0.084           0.084
## EMN         0.110           0.090
## ED          0.101           0.091
## FIA         0.159           0.107
## GM          0.057           0.059
## LS          0.077           0.078
## MA          0.098           0.088
## RV          0.133           0.099
## SS          0.072           0.072
## FOF         0.084           0.082
```

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

3.5 M-Estimators of Location

As an alternative to the outlier cleaning method described in Section 3.4, a standard error of the robust M-estimator of location based on the influence function approach is available in `RPESE`. A rigorous treatment of the influence function of location M-estimators can be found in [Maronna et al., 2019].

We illustrate the effect of using a robust estimator for the mean of serially correlated returns. The following code computes, and plots in Figure 3, the IF.mean transformed FIA returns, and the IF.robLoc transformed FIA returns.

```
retFIA <- edhec$FIA
iftrFIA <- IF.mean(returns=retFIA,IFprint=T)
iftrFIArob <- IF.robLoc(returns=retFIA,
                        family=c("mopt", "opt", "bisquare")[1], eff=0.95,
                        IFprint=T)
par(mfrow=c(2,1))
plot(iftrFIA, main="IF.mean Transformed FIA Returns",lwd=.8)
plot(iftrFIArob, main="IF.robLoc Tranformed FIA Returns", lwd=.8)
par(mfrow=c(1,1))
```

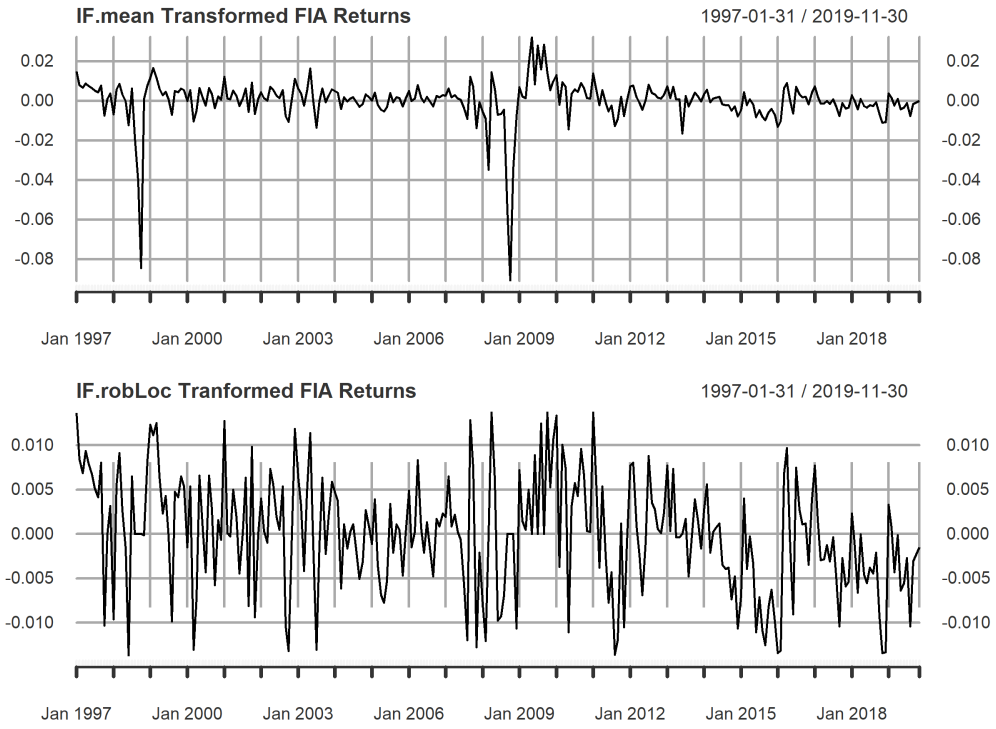

Figure 3: IF.mean and IF.robLoc FIA Transformed FIA Returns

In `RPESE`, the families of ψ functions for M-estimators of location and their influence functions are the same ones available in the package `RobStatTM`, namely `mOpt`, `opt` and `bisquare`. The default is the `mOpt` family, where

$$\psi_{mOpt}(x) = \begin{cases} x & |x| \leq 1 \\ \frac{\phi(1)}{\phi(1)-a}\left(x - SGN(x)\frac{a}{\phi(x)}\right)U(c-|x|) & |x| > 1 \end{cases} \tag{1}$$

where $\phi(x)$ is the standard normal density function, $U(x)$ is the unit step function, and the constants a and c depend on the desired normal distribution efficiency. These constants are computed internally in `RPESE` for a desired efficiency. The following code shows an example on how to compute the standard error of robust M-estimators of location, specifying the `family` and `eff` function arguments.

```
# Robust Location
robLoc.out <- robLoc.SE(edhec, se.method=c("IFiid", "IFcorAdapt"),
                        family = c("mopt", "opt", "bisquare")[1], eff = 0.95)
printSE(robLoc.out, round.digit = 3)

##     robLoc IFiid IFcorAdapt
## CA   0.006 0.001      0.001
## CTA  0.004 0.001      0.001
## DIS  0.008 0.001      0.001
## EM   0.008 0.002      0.002
## EMN  0.005 0.000      0.000
## ED   0.008 0.001      0.001
## FIA  0.005 0.000      0.001
## GM   0.004 0.001      0.001
## LS   0.007 0.001      0.001
## MA   0.006 0.001      0.001
## RV   0.007 0.001      0.001
## SS  -0.004 0.002      0.003
## FOF  0.004 0.001      0.001
```

3.6 Correlations of Returns and Influence Function Transformed Returns

Note that you can also return the (lag-1) correlations of the returns time series, as well as the influence function transformed returns as part of the output using the `corOut` argument. The available options are `retCor, retIFCor and retIFCorPW`, as shown in the example below.

```
# Sharpe Ratio SE computation for all hedge funds in data set
# with output of correlations of returns and IF transformed returns
Sharpe.retCor <- SR.SE(edhec,
                       se.method=c("IFiid","IFcor","IFcorAdapt"),
                       corOut=c("retCor", "retIFCor"))
printSE(Sharpe.retCor)

##         SR IFiid IFcor IFcorAdapt retCor retIFCor
## CA   0.338 0.096 0.117      0.174  0.565    0.554
```

```
## CTA   0.180 0.060 0.057       0.057 -0.008   -0.039
## DIS   0.392 0.080 0.108       0.121  0.492    0.486
## EM    0.194 0.069 0.084       0.084  0.296    0.280
## EMN   0.543 0.110 0.110       0.115  0.284    0.153
## ED    0.372 0.079 0.099       0.101  0.341    0.334
## FIA   0.377 0.113 0.134       0.159  0.500    0.440
## GM    0.371 0.054 0.057       0.057  0.057    0.060
## LS    0.315 0.066 0.079       0.079  0.216    0.252
## MA    0.560 0.092 0.098       0.097  0.277    0.130
## RV    0.504 0.097 0.117       0.133  0.424    0.480
## SS   -0.041 0.061 0.072       0.072  0.154    0.154
## FOF   0.275 0.066 0.067       0.084  0.311    0.334
```

3.7 Exponential and Gamma Distributions

The `RPEGLMEN` package used in the fitting of the elastic net penalized GLM model to the periodogram of the IF transformed return series was developed and tested for exponential distribution GLM models. However some initial work was done for that type of GLM model fitting for the family of Gamma distribution. And initial results for the Gamma distribution, reported in CM, indicates that the Gamma distribution GLM often results in a more parsimonious polynomial model fit the the periodogram. By way of example, the following code, which is very slow, computes standard errors of Sharpe ratio estimators for exponential and Gamma distributions.

```
# Sharpe Ratio SE computation for edhec hedge funds using Gamma distribution
Clean.out <- SR.SE(edhec,
                   se.method=c("IFiid","IFcor","IFcorAdapt"),
                   cleanOutliers = T)
GammaClean.out <- SR.SE(edhec,
                        se.method=c("IFiid","IFcor","IFcorAdapt"),
                        cleanOutliers = T, fitting.method = "Gamma")
GammaExp.comparison <- cbind(printSE(Clean.out)[,4],
                             printSE(GammaClean.out)[,4])
colnames(GammaExp.comparison) <- c("IFcorAdapt", "IFcorAdapt-Gamma")
rownames(GammaExp.comparison) <- names(edhec)
GammaExp.comparison
```

We regard the Gamma family code implemented in `RPEGLMEN`, used for the above computation, to be quite experimental at this stage, and anticipate further development of the code in C++ at a future date.

3.8 Decimation and Truncation of Frequencies of Discrete Fourier Transform

There is an option in the SE functions to use a decimated or truncated percentage of the frequencies of the discrete Fourier transforms for the periodogram in the fitting of the Exponential or Gamma distributions. Decimation implies that only certain frequencies are used, and they will be equally spaced selections from the frequencies. Truncation implies that only a certain percentage of the frequencies (i.e. only the first frequencies until a certain point) will be used.

By default, the SE functions use all the frequencies. 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. Below is some sample code demostration for this usage.

```
# Sharpe Ratio SE using all, decimating, or truncating the frequencies
SE.all <- SR.SE(edhec,
                se.method=c("IFiid","IFcor","IFcorAdapt"),
                cleanOutliers = T,
                freq.include = "All")
SE.decimate <- SR.SE(edhec,
                     se.method=c("IFiid","IFcor","IFcorAdapt"),
                     cleanOutliers = T,
                     freq.include = "Decimate", freq.par = 0.5)
SE.truncate <- SR.SE(edhec,
                     se.method=c("IFiid","IFcor","IFcorAdapt"),
                     cleanOutliers = T,
                     freq.include = "Truncate", freq.par = 0.5)
frequency.comparison <- cbind(printSE(SE.all)[,4],
                              printSE(SE.decimate)[,4],
                              printSE(SE.truncate)[,4])
colnames(frequency.comparison) <- c("IFcorAdapt-All",
                                    "IFcorAdapt-Decimate",
                                    "IFcorAdapt-Truncate")
```

```r
rownames(frequency.comparison) <- names(edhec)
frequency.comparison

##     IFcorAdapt-All IFcorAdapt-Decimate IFcorAdapt-Truncate
## CA           0.108                0.113                0.107
## CTA          0.057                0.057                0.057
## DIS          0.112                0.117                0.111
## EM           0.084                0.087                0.078
## EMN          0.090                0.104                0.092
## ED           0.081                0.096                0.092
## FIA          0.108                0.114                0.107
## GM           0.059                0.060                0.060
## LS           0.078                0.081                0.078
## MA           0.088                0.092                0.087
## RV           0.099                0.109                0.100
## SS           0.072                0.075                0.068
## FOF          0.082                0.081                0.078
```

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.

RA Maronna, RD Martin, VJ Yohai, and M Salibian-Barrera. Robust statistics: Theory and methods (with r) wiley. *Hoboken, NJ, USA*, 2019.

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