Fitting Fundamental Factor Models: factorAnalytics vignette

Sangeetha Srinivasan

March 20, 2018

Abstract

The purpose of this vignette is to demonstrate the use of `fitFfm` and related control, analysis and plot functions in the `factorAnalytics` package.

Contents

1 Overview

1.1 Load Package

The latest version of the `factorAnalytics` package used in this vignette is hosted in the publicly available GitHub repository https://github.com/sangeeuw/factorAnalytics. Note that there are plans for further updates to the package before its moved back to R-Forge and gets a CRAN release later this year.

The package can be installed from GitHub using `devtools` as follows.

```
library(devtools)
install_github("sangeeuw/factorAnalytics")
```

```
# load the package and its dependencies
library(factorAnalytics)
options(digits=3)
```

The focus of this vignette is on the `fitFfm` function and related methods. The original function was designed by Doug Martin and initially implemented in S-PLUS by a number of University of Washington Ph.D. students: Christopher Green, Eric Aldrich, and Yindeng Jiang. Guy Yollin ported the function to R and Yi-An Chen modified that code. Sangeetha Srinivasan tested and expanded the functionalities and S3 methods. Doug Martin, Avinash Acharya, Lingjie Yi and Chindhanai Uthaisaad further added options to model EWMA and GARCH errors, allow for a market + industry and/or sector and/or country model specification, etc. Refer to the previous Fundamental factor model vignette for more examples elaborating on these recent functionalities and reporting functions.

1.2 Summary of related functions

Here's a list of the functions and methods demonstrated in this vignette:

- `fitFfm(data, asset.var, ret.var, date.var, exposure.vars, weight.var, fit.method, rob.stats, full.resid.cov, z.score, add.intercept, lag.exposures, resid.scale.type, lambda, GARCH.params, GARCH.MLE, std.return, analysis, target.vol, ...)`: Fits a fundamental factor model for one or more asset returns or excess returns using T cross-sectional regressions a.k.a. the "BARRA" approach (detailed in Grinold and Kahn (2000)), where T is the number of time periods. Least squares (LS), weighted least squares (WLS), robust (rob) and weighted-robust regression (W-Rob) fitting

are possible. Options for computing residual variances include sample variance, EWMA, Robust EWMA and GARCH(1,1). An object of class "ffm" containing the fitted objects, factor exposures, factor returns, R-squared, residual volatility, etc. is returned.

- `coef(object, ...)`: Returns a data.frame containing the coefficients (intercept and factor exposures) for the last time period for all assets.
- `fitted(object, ...)`: Returns an "xts" data object of fitted asset returns from the factor model for all assets.
- `residuals(object, ...)`: Returns an "xts" data object of residuals from the fitted factor model for all assets.
- `fmCov(object, use, ...)`: Returns the `N x N` symmetric covariance matrix for asset returns based on the fitted factor model using exposures from the last time period.
- `fmSdDecomp(object, use, ...)`: Returns a list containing the standard deviation of asset returns based on the fitted factor model and the marginal, component and percentage component factor contributions estimated from the given sample. `"use"` specifies how missing values are to be handled.
- `fmVaRDecomp(object, p, ...)`: Returns a list containing the value-at-risk for asset returns based on the fitted factor model and the marginal, component and percentage component factor contributions estimated from the given sample. VaR computation can be non-parametric (sample quantile) or based on a Normal distribution. And, `"p"` specifies the confidence level.
- `fmEsDecomp(object, p, ...)`: Returns a list containing the expected shortfall for asset returns based on the fitted factor model and the marginal, component and percentage component factor contributions estimated from the given sample. Expected shortfall computation can be non-parametric (sample quantile) or based on a Normal distribution.
- `plot(x)`: The `plot` method for class "ffm" can be used for plotting factor model characteristics of a group of assets (default) or an individual asset. The user can select the type of plot either from the menu prompt or directly via argument `which`. In case multiple plots are needed, the menu is repeated after each plot (enter 0 to exit). User can also input a numeric vector of plot options via `which`.
- `predict(object, newdata, pred.date, ...)`: The `predict` method for class "ffm" returns a vector or matrix of predicted values for a new data sample or simulated values. `pred.date` allows user to choose a date, and hence the estimated factor exposures from that date to be used in the prediction.

- `summary(object, ...)`: The `summary` method for class "ffm" returns an object of class `"summary.ffm"` containing the summaries of the fitted objects. Printing the factor model summary object outputs the call, estimated factor returns, r-squared and residual volatility for each time period.

1.3 Data

The following examples primarily use the `Stock.df` dataset. It contains fundamental and monthly return data for 447 stocks listed on the NYSE over a 8-year period. The dataset is balanced, i.e., every asset has a complete set of observations for all variables in each time period. Here are some properties of the dataset:

```
# load the dataset into the environment
data(Stock.df)
# get a list of the variable names
colnames(stock)

##  [1] "DATE"                "RETURN"              "TICKER"
##  [4] "PRICE"               "VOLUME"              "SHARES.OUT"
##  [7] "MARKET.EQUITY"       "LTDEBT"              "NET.SALES"
## [10] "COMMON.EQUITY"       "NET.INCOME"          "STOCKHOLDERS.EQUITY"
## [13] "LOG.MARKETCAP"       "LOG.PRICE"           "BOOK2MARKET"
## [16] "GICS"                "GICS.INDUSTRY"       "GICS.SECTOR"

# time period covered in the data
range(stock[,"DATE"])

## [1] "1996-02-29" "2003-12-31"

# number of stocks
length(unique(stock[,"TICKER"]))

## [1] 447

# count stocks by GICS sector as of the last time period
stocklist<-subset(stock,DATE=="2003-12-31")
table(stocklist$GICS.SECTOR)

##
```

```
##     Consumer Discretionary           Consumer Staples
##                         86                         30
##                     Energy                Financials
##                         17                         55
##                Health Care                Industrials
##                         35                         89
##     Information Technology                  Materials
##                         57                         32
## Telecommunication Services                  Utilities
##                          6                         40
```

2 Fit a fundamental factor model

A fundamental factor model uses observed cross-sectional asset characteristics such as dividend yield, earnings yield, book-to-market ratio, market capitalization, sector or industry classification, price volatility, price momemtum, leverage, etc. to determine common risk factors that contribute to asset returns. There are 2 main approaches to estimating the fundamental factor model - the "BARRA" approach (detailed in Grinold and Kahn (2000)) and the "Fama-French" approach (introduced in Fama and French (1992)). In the "BARRA" approach, the obsered fundamental attributes are the factor betas and the unknown factor returns are estimated via cross-sectional regressions for each time period. In the "Fama-French" approach, the factor returns are the observed returns of a hypothetical hedge portfolio that's long/short the top/bottom quintile of stocks for a given attribute (ex: market cap for the size factor). After the factor returns are computed for each characterictic, each asset's factor exposures are estimated via a time series regression. `fitFfm` described in this vignette uses the "BARRA" approach.

Let's take a look at the arguments for `fitFfm`.

```
args(fitFfm)

## function (data, asset.var, ret.var, date.var, exposure.vars,
##     weight.var = NULL, fit.method = c("LS", "WLS", "Rob", "W-Rob"),
##     rob.stats = FALSE, full.resid.cov = FALSE, z.score = c("none",
##         "crossSection", "timeSeries"), add.intercept = FALSE,
##     lag.exposures = TRUE, resid.scale.type = c("stdDev", "EWMA",
##         "robEWMA", "GARCH"), GARCH.params = list(omega = 0.09,
##         alpha = 0.1, beta = 0.81), lambda = 0.9, GARCH.MLE = FALSE,
##     std.return = FALSE, analysis = c("none", "ISM", "NEW"), target.vol = 0.06,
##     ...)
## NULL
```

The default model fitting method is ordinary least squares (LS) regression, with the option to choose robust regression (Rob), weighted least squares (WLS) or weighted robust regression (W-Rob). The different model fitting options are demonstrated in the following sections. If weighted regression (WLS or W-Rob) is chosen, inverse of the residual variances are used as weights. `resid.scale.type` allows the user to choose the method for computing residual variances - sample variance, EWMA, Robust EWMA and GARCH(1,1).

`z.score` provides the option to standardize factor exposures cross-sectionally across assets or across time periods. User can also choose to weight the factor exposures by some characteristic

(ex: market cap) via `weight.var`. `add.intercept` gives the option to add an intercept term for fitting a Market + Sector or a Market + Sector + Country model. Note that these models can also include other style factors. `lag.exposures` gives the option to use lag the factor exposures by one time period. `full.resid.cov` provides the option to choose between a diagonal vs. full residual covariance matrix. And, `rob.stats` allows for robust estimates of covariance, correlation, location and univariate scale.

These and other control parameters are demonstrated in the following sections.

2.1 Single Factor Model

Here's an example of a single factor model using the book-to-market ratio for the 447 stocks in our dataset.

```
# Single Factor Model
fit.single <- fitFfm(data=stock, asset.var="TICKER", ret.var="RETURN",
                     date.var="DATE", exposure.vars="BOOK2MARKET")
```

The resulting object, `fit.single`, has the following attributes.

```
class(fit.single)

## [1] "ffm"

names(fit.single)

##  [1] "factor.fit"      "beta"            "factor.returns"
##  [4] "residuals"       "r2"              "factor.cov"
##  [7] "g.cov"           "resid.cov"       "return.cov"
## [10] "restriction.mat" "resid.var"       "call"
## [13] "data"            "date.var"        "ret.var"
## [16] "asset.var"       "exposure.vars"   "weight.var"
## [19] "fit.method"      "asset.names"     "factor.names"
## [22] "time.periods"    "activeWeights"   "activeReturns"
## [25] "IR"
```

The component `factor.fit` contains a list of "lm" or "lmRob" objects, one for each time period. The fitted objects is of class "lm" if `fit.method="LS" or "WLS"`, or class "lmRob", if `fit.method="Rob" or "W-Rob"`. The estimated factor returns are in `factor.returns` and factor exposures from the last time period are in `beta`. R-squared and residual variance are in `r2` and

`resid.var` respectively. The estimated factor, residual and asset return covariance matrices are in `factor.cov`, `resid.cov` and `return.cov` respectively. The remaining components contain the input choices and the data.

The `print` method displays a summary of the T cross-sectional regressions, where T is the number of time periods.

```
# print the fitted "ffm" object
fit.single

##
## Call:
## fitFfm(data = stock, asset.var = "TICKER", ret.var = "RETURN",
##     date.var = "DATE", exposure.vars = "BOOK2MARKET")
##
## Model dimensions:
## Factors  Assets Periods
##       1     447      94
##
## Factor returns across periods:
##   BOOK2MARKET
##  Min.   :-0.0332
##  1st Qu.:-0.0053
##  Median : 0.0045
##  Mean   : 0.0048
##  3rd Qu.: 0.0139
##  Max.   : 0.0446
##
## R-squared values across periods:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0000  0.0010  0.0043  0.0076  0.0122  0.0475
##
## Residual Variances across assets:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.0036  0.0141  0.0209  0.0294  0.0347  0.1590
```

Figure 1 shows a scatter plot of residuals, with histograms, density overlays, correlations and

significance stars. (Plot options are explained later in section 4.) Note the high residual correlation between MSFT and ORCL might be due to a sector/industry factor

```
# plot residual correlations for the single factor model
# default is to plot the 1st 6 assets
plot(fit.single, which=6, f.sub=1)
```

```
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
```

```
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
```

```
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
## Warning in axis(side = side, at = at, labels = labels, ...):  "method" is not a
graphical parameter
## Warning in plot.xy(xy.coords(x, y), type = type, ...):  "method" is not a graphical
parameter
## Warning in plot.window(...):  "method" is not a graphical parameter
## Warning in plot.xy(xy, type, ...):  "method" is not a graphical parameter
## Warning in title(...):  "method" is not a graphical parameter
```

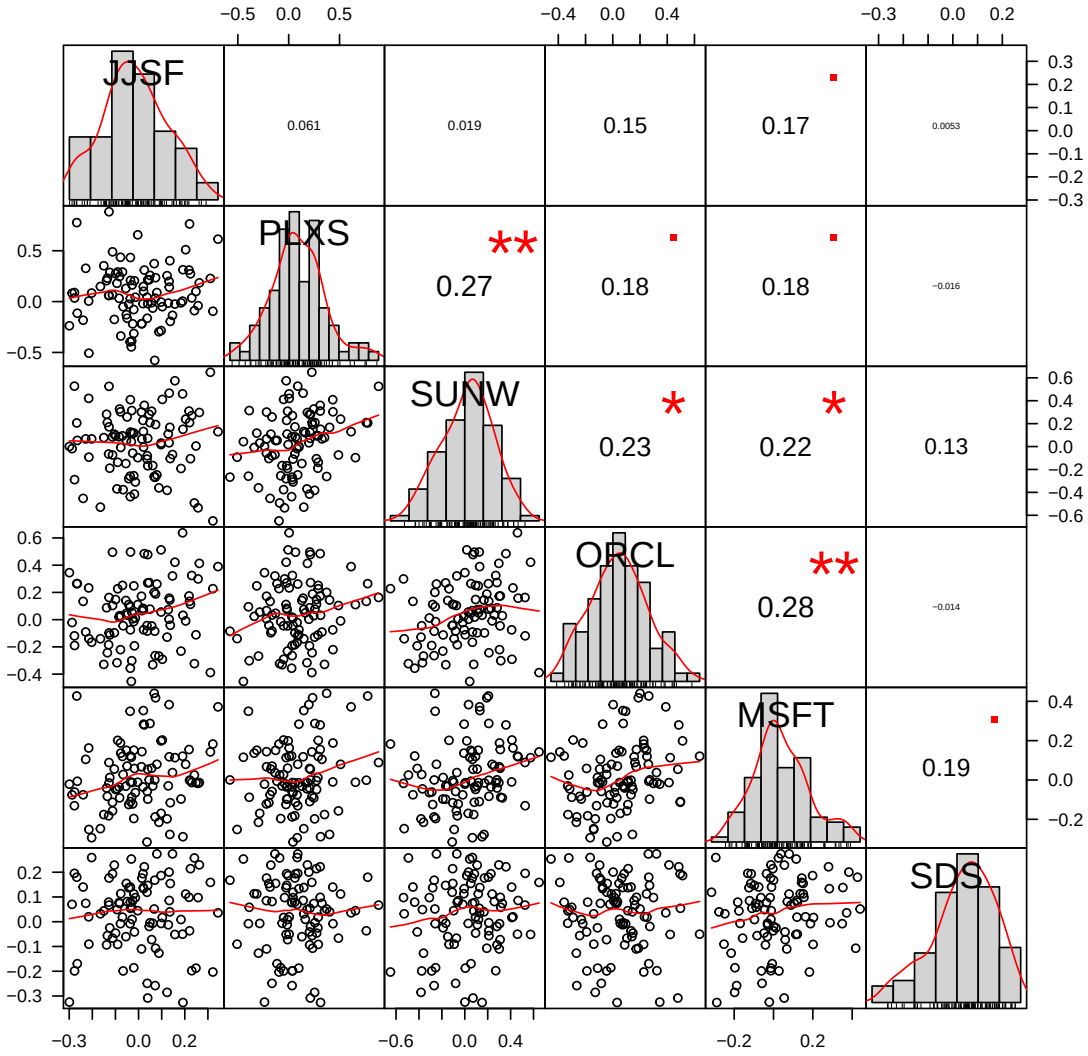

Figure 1: Single factor model: Residual correlations

2.2 BARRA-type Industry Factor Model

A BARRA-type industry (sector) factor model is a fundamental factor model with multiple factors. It is implemented here for the 447 NYSE stocks in our chosen dataset, using the 10 mutually exclusive GICS sector classifications as the 10 factors. The factor exposures will be dummy variables that indicate if a given stock belongs to a particular sector or not. Mutually exclusive sectors means that each stock belongs to a unique sector in any given time period.

```
# Sector Factor Model
fit.sector <- fitFfm(data=stock, asset.var="TICKER", ret.var="RETURN",
                     date.var="DATE", exposure.vars="GICS.SECTOR")
```

Let's take a look at the fitted factor model from the last time period (Dec 2003). Materials, Telecomm and the Energy sector had particularly strong returns, with their estimated factor returns over 10% for that month. Energy stocks rebounded in 2003 from the beating they took in 2002 following the Enron scandal. Telecomm stocks benefited from the increased spending by companies investing in internet-based phone systems etc.

```
# print the summary from the last period's fit
num.periods <- length(fit.sector$time.periods)
summary(fit.sector$factor.fit[[num.periods]])

##
## Call:
## FUN(formula = ..1, data = data[x, , drop = FALSE], na.action = ..3,
##     contrasts = ..2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.3984 -0.0806 -0.0067  0.0780  0.5362
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## Consumer Discretionary       0.0124     0.0154    0.80   0.4236
## Consumer Staples             0.0480     0.0261    1.84   0.0666 .
## Energy                       0.1131     0.0347    3.26   0.0012 **
## Financials                   0.0466     0.0193    2.41   0.0162 *
```

```
## Health Care                   0.0358     0.0242    1.48   0.1398
## Industrials                   0.0415     0.0152    2.74   0.0064 **
## Information Technology        0.0339     0.0190    1.79   0.0744 .
## Materials                     0.1146     0.0253    4.53  7.6e-06 ***
## Telecommunication Services    0.1025     0.0584    1.76   0.0799 .
## Utilities                     0.0684     0.0226    3.02   0.0027 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.143 on 437 degrees of freedom
## Multiple R-squared:  0.131,Adjusted R-squared:  0.112
## F-statistic: 6.61 on 10 and 437 DF,  p-value: 1.44e-09
```

Figure 2 shows the distribution of estimated factor returns, sorted by descending order of their mean. We observe that the "Information Technology" sector had the highest average return (not suprising, given this dataset covers a significant part of the dot-com bubble).

```
# plot distribution of factor returns by sector sorted by means
plot(fit.sector, which=1, colorset="black", f.sub=1:10, lwd=1,
     sort.by="mean")
```

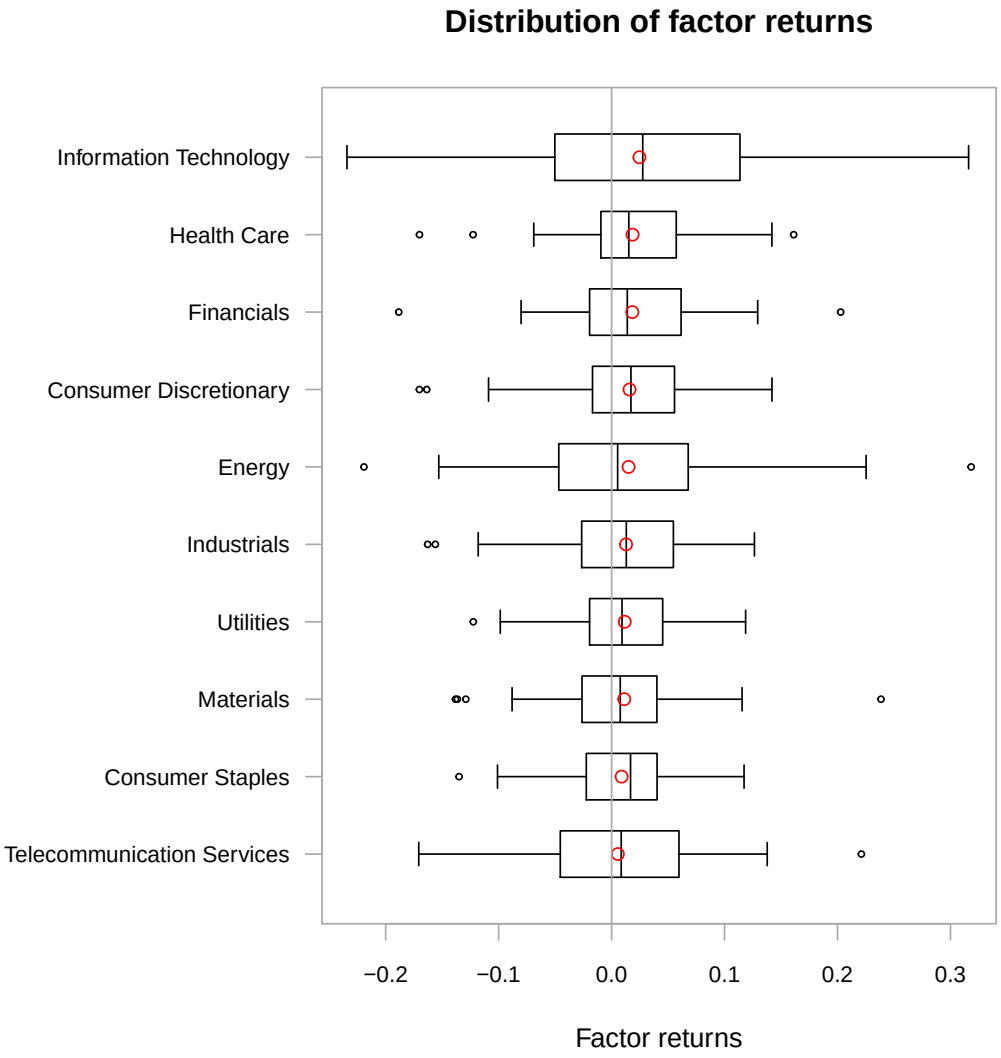

Figure 2: Sector model: Distribution of Factor Returns

2.3 S3 generic methods

```
methods(class="ffm")

##  [1] coef          fitted        fmCov         fmEsDecomp    fmRsq
##  [6] fmSdDecomp    fmTstats      fmVaRDecomp   plot          portEsDecomp
## [11] portSdDecomp  portVaRDecomp portVolDecomp predict       print
## [16] repRisk       residuals     riskDecomp    summary
## see '?methods' for accessing help and source code
```

Many useful generic accessor functions are available for "ffm" fit objects. `coef()` returns a matrix of estimated model coefficients including the intercept. `fitted()` returns an xts data object of the component of asset returns explained by the factor model. `residuals()` returns an xts data object with the component of asset returns not explained by the factor model. `predict()` uses the fitted factor model to estimate asset returns given a set of new or simulated factor return data.

`summary()` prints standard errors and t-statistics for all estimated coefficients in addition to R-squared values and residual volatilities. Argument `se.type`, one of "Default", "HC" or "HAC", allows for heteroskedasticity and auto-correlation consistent estimates and standard errors whenever possible. A "summary.ffm" object is returned which contains a list of summary objects returned by "lm", "lm.Rob" or "lars" for each asset fit.

Note: Standard errors are currently not available for the "lars" variable selection method, as there seems to be no consensus on a statistically valid method of calculating standard errors for the lasso predictions.

Factor model covariance and risk decomposition functions are explained in section 3 and the `plot` method is discussed separately in Section 4.

Here are some examples using the time series factor models fitted earlier.

```
# all estimated coefficients from the LS fit using all 3 factors
coef(fit.ols)

## Error in coef(fit.ols):  object 'fit.ols' not found

# compare returns data with fitted and residual values for HAM1 from fit.lars
HAM1.ts <- merge(fit.lars$data[,1], fitted(fit.lars)[,1], residuals(fit.lars)[,1])

## Error in merge(fit.lars$data[, 1], fitted(fit.lars)[, 1], residuals(fit.lars)[,
:  object 'fit.lars' not found
```

```r
colnames(HAM1.ts) <- c("HAM1.return","HAM1.fitted","HAM1.residual")
```

```
## Error in colnames(HAM1.ts) <- c("HAM1.return", "HAM1.fitted", "HAM1.residual"):
object 'HAM1.ts' not found
```

```r
tail(HAM1.ts)
```

```
## Error in tail(HAM1.ts):  object 'HAM1.ts' not found
```

```r
# summary for fit.sub computing HAC standard erros
summary(fit.sub, se.type="HAC")
```

```
## Error in summary(fit.sub, se.type = "HAC"):  object 'fit.sub' not found
```

3 Factor Model Covariance & Risk Decomposition

3.1 Factor model covariance

Following Zivot and Jia-hui (2006), $R_{i,t}$, the return on asset i $(i = 1, ..., N)$ at time t $(t = 1, ..., T)$, is fitted with a factor model of the form,

$$R_{i,t} = \alpha_i + \boldsymbol{\beta}_i' \, \mathbf{f_t} + \epsilon_{i,t} \tag{1}$$

where, α_i is the intercept, $\mathbf{f_t}$ is a $K \times 1$ vector of factor returns at time t, $\boldsymbol{\beta}_i$ is a $K \times 1$ vector of factor exposures for asset i and the error terms $\epsilon_{i,t}$ are serially uncorrelated across time and contemporaneously uncorrelated across assets so that $\epsilon_{i,t} \sim iid(0, \sigma_i^2)$. Thus, the variance of asset i's return is given by

$$var(R_{i,t}) = \boldsymbol{\beta}_i' \, var(\mathbf{f_t}) \, \boldsymbol{\beta}_i + \sigma_i^2 \tag{2}$$

And, the $N \times N$ covariance matrix of asset returns is

$$var(\mathbf{R}) = \boldsymbol{\Omega} = \mathbf{B} \, var(\mathbf{F}) \, \mathbf{B}' + \mathbf{D} \tag{3}$$

where, R is the $N \times T$ matrix of asset returns, B is the $N \times K$ matrix of factor betas, $\mathbf{F}$ is a $K \times T$ matrix of factor returns and D is a diagonal matrix with σ_i^2 along the diagonal.

`fmCov()` computes the factor model covariance from a fitted factor model. The covariance of factor returns is the sample covariance matrix by default, but the option exists for the user to specify their own. Options for handling missing observations include "pairwise.complete.obs" (default), "everything", "all.obs", "complete.obs" and "na.or.complete".

```
fmCov(fit.sub)

## Error in inherits(object, c("tsfm", "sfm", "ffm")):  object 'fit.sub' not found

# factor model return correlation plot
plot(fit.sub, which=8)

## Error in plot(fit.sub, which = 8):  object 'fit.sub' not found
```

3.2 Standard deviation decomposition

Given the factor model in equation 1, the standard deviation of the asset i's return can be decomposed as follows (based on Meucci (2007)):

$$R_{i,t} = \alpha_i + \boldsymbol{\beta}_i' \, \mathbf{f_t} + \epsilon_{i,t} \tag{4}$$

$$= \boldsymbol{\beta}_i^{*'} \, \mathbf{f_t^*} \tag{5}$$

where, $\boldsymbol{\beta}_i^{*'} = (\boldsymbol{\beta}_i' \; \sigma_i)$ and $\mathbf{f}_\mathbf{t}^{*'} = (\mathbf{f}_\mathbf{t}' \; z_t)$, with $z_t \sim iid(0,1)$ and σ_i is asset i's residual standard deviation.

By Euler's theorem, the standard deviation of asset i's return is:

$$Sd.fm_i = \sum_{k=1}^{K+1} cSd_{i,k} = \sum_{k=1}^{K+1} \beta_{i,k}^* \, mSd_{i,k} \tag{6}$$

where, summation is across the K factors and the residual, $\mathbf{cSd_i}$ and $\mathbf{mSd_i}$ are the component and marginal contributions to $Sd.fm_i$ respectively. Computing $Sd.fm_i$ and $\mathbf{mSd_i}$ is very straight forward. The formulas are given below and details are in Meucci (2007). The covariance term is approximated by the sample covariance and $\odot$ represents element-wise multiplication.

$$Sd.fm_i = \sqrt{\boldsymbol{\beta}_i^{*'} \, cov(\mathbf{F}^*) \, \boldsymbol{\beta}_i^*} \tag{7}$$

$$\mathbf{mSd_i} = \frac{cov(\mathbf{F}^*) \, \boldsymbol{\beta}_i^*}{Sd.fm_i} \tag{8}$$

$$\mathbf{cSd_i} = \boldsymbol{\beta}_i^* \odot \mathbf{mSd_i} \tag{9}$$

`fmSdDecomp` performs this decomposition for all assets in the given factor model fit object as shown below. The total standard deviation and component, marginal and percentage component contributions for each asset are returned.

```
decomp <- fmSdDecomp(fit.sub)

## Error in inherits(object, c("tsfm", "sfm", "ffm")):  object 'fit.sub' not found

names(decomp)

## Error in eval(expr, envir, enclos):  object 'decomp' not found

# get the factor model standard deviation for all assets
decomp$Sd.fm

## Error in eval(expr, envir, enclos):  object 'decomp' not found

# get the component contributions to Sd
decomp$cSd

## Error in eval(expr, envir, enclos):  object 'decomp' not found

# get the marginal factor contributions to Sd
decomp$mSd

## Error in eval(expr, envir, enclos):  object 'decomp' not found
```

```r
# get the percentage component contributions to Sd
decomp$pcSd
```

```
## Error in eval(expr, envir, enclos):  object 'decomp' not found
```

```r
# plot the percentage component contributions to Sd
plot(fit.sub, which=9, f.sub=1:3)
```

```
## Error in plot(fit.sub, which = 9, f.sub = 1:3):  object 'fit.sub' not found
```

3.3 Value-at-Risk decomposition

The VaR version of equation 6 is given below. By Euler's theorem, the value-at-risk of asset i's return is:

$$VaR.fm_i = \sum_{k=1}^{K+1} cVaR_{i,k} = \sum_{k=1}^{K+1} \beta^*_{i,k}\, mVaR_{i,k} \tag{10}$$

The marginal contribution to $VaR.fm$ is defined as the expectation of $F.star$, conditional on the loss being equal to $VaR.fm$. This is approximated as described in Epperlein and Smillie (2006) using a triangular smoothing kernel. $VaR.fm$ is calculated as the sample quantile.

`fmVaRDecomp` performs this decomposition for all assets in the given factor model fit object as shown below.

```
decomp1 <- fmVaRDecomp(fit.sub)

## Error in inherits(object, c("tsfm", "sfm", "ffm")):  object 'fit.sub' not found

names(decomp1)

## Error in eval(expr, envir, enclos):  object 'decomp1' not found

# get the factor model value-at-risk for all assets
decomp1$VaR.fm

## Error in eval(expr, envir, enclos):  object 'decomp1' not found

# get the percentage component contributions to VaR
decomp1$pcVaR

## Error in eval(expr, envir, enclos):  object 'decomp1' not found

# plot the percentage component contributions to VaR
plot(fit.sub, which=11, f.sub=1:3)

## Error in plot(fit.sub, which = 11, f.sub = 1:3):  object 'fit.sub' not found
```

3.4 Expected Shortfall decomposition

The Expected Shortfall (ES) version of equation 6 is given below. By Euler's theorem, the expected shortfall of asset i's return is:

$$ES.fm_i = \sum_{k=1}^{K+1} cES_{i,k} = \sum_{k=1}^{K+1} \beta^*_{i,k}\, mES_{i,k} \tag{11}$$

The marginal contribution to $ES.fm$ is defined as the expectation of $F.star$, conditional on the loss being less than or equal to $VaR.fm$. This is estimated as a sample average of the observations in that data window. Once again, $VaR.fm$ is the sample quantile.

`fmEsDecomp` performs this decomposition for all assets in the given factor model fit object as shown below.

```
decomp2 <- fmEsDecomp(fit.sub, method="historical")

## Error in inherits(object, c("tsfm", "sfm", "ffm")):  object 'fit.sub' not found

names(decomp2)

## Error in eval(expr, envir, enclos):  object 'decomp2' not found

# get the factor model expected shortfall for all assets
decomp2$ES.fm

## Error in eval(expr, envir, enclos):  object 'decomp2' not found

# get the component contributions to Sd
decomp2$cES

## Error in eval(expr, envir, enclos):  object 'decomp2' not found

# get the marginal factor contributions to ES
decomp2$mES

## Error in eval(expr, envir, enclos):  object 'decomp2' not found

# get the percentage component contributions to ES
decomp2$pcES

## Error in eval(expr, envir, enclos):  object 'decomp2' not found

# plot the percentage component contributions to ES
plot(fit.sub, which=10, f.sub=1:3)

## Error in plot(fit.sub, which = 10, f.sub = 1:3):  object 'fit.sub' not found
```

4 Plot

Some types of individual asset (Figure 2) and group plots (Figures 1, 3-9) have already been demonstrated. Let's take a look at all available arguments for plotting a "ffm" object.

```
## S3 method for class "ffm"
plot (x, which=NULL, f.sub=1:2, a.sub=1:6, plot.single=FALSE, asset.name,
      colorset=c("royalblue","dimgray","olivedrab","firebrick",
                 "goldenrod","mediumorchid","deepskyblue","chocolate",
                 "darkslategray"),
      legend.loc="topleft", las=1, lwd=2, maxlag=15, ...)
```

4.1 Group plots

This is the default option for plotting. Simply running `plot(fit)`, where `fit` is any "ffm" object, will bring up a menu (shown below) for group plots.

```
plot(fit.sub)


# Make a plot selection (or 0 to exit):


#  1: Factor model coefficients: Alpha
#  2: Factor model coefficients: Betas
#  3: Actual and Fitted asset returns
#  4: R-squared
#  5: Residual Volatility
#  6: Scatterplot matrix of residuals, with histograms, density overlays,
#     correlations and significance stars
#  7: Factor Model Residual Correlation
#  8: Factor Model Return Correlation
#  9: Factor Contribution to SD
# 10: Factor Contribution to ES
# 11: Factor Contribution to VaR
# 12: Asset returns vs factor returns (single factor model)
#
# Selection:
```

Remarks:

- Only a subset of assets and factors selected by `a.sub` and `f.sub` are plotted. The first 2 factors and first 6 assets are shown by default.
- The last option for plotting asset returns vs. factor returns (group plot option 12 and individual asset plot option 19) are only applicable for single factor models.

```
# Examples of group plots: looping disabled & no. of assets displayed = 4.
plot(fit.sub, which=3, a.sub=1:4, legend.loc=NULL, lwd=1)

## Error in plot(fit.sub, which = 3, a.sub = 1:4, legend.loc = NULL, lwd = 1):  object
'fit.sub' not found
```

```
plot(fit.sub, which=6) # residual scatter plot matrix with correlations

## Error in plot(fit.sub, which = 6):  object 'fit.sub' not found
```

4.2 Menu and looping

If the plot type argument `which` is not specified, a menu prompts for user input. In case multiple plots are needed, the menu is repeated after each plot (enter 0 to exit). User can also input a numeric vector of plot options via `which`.

4.3 Individual plots

Setting `plot.single=TRUE` enables individual asset plots. If there is more than one asset fit by the fitted object `x`, `asset.name` is also necessary. In case the `ffm` object `x` contains only a single asset's fit, plot.ffm can infer `asset.name` without user input.

Here's the individual plot menu.

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1")


# Make a plot selection (or 0 to exit):
#  1: Actual and fitted asset returns
#  2: Actual vs fitted asset returns
#  3: Residuals vs fitted asset returns
#  4: Sqrt. of modified residuals vs fitted
#  5: Residuals with standard error bands
#  6: Time series of squared residuals
#  7: Time series of absolute residuals
#  8: SACF and PACF of residuals
#  9: SACF and PACF of squared residuals
# 10: SACF and PACF of absolute residuals
# 11: Non-parametric density of residuals with normal overlaid
# 12: Non-parametric density of residuals with skew-t overlaid
# 13: Histogram of residuals with non-parametric density and normal overlaid
# 14: QQ-plot of residuals
# 15: CUSUM test-Recursive residuals
# 16: CUSUM test-LS residuals
# 17: Recursive estimates (RE) test of LS regression coefficients
# 18: Rolling regression over a 24-period observation window
# 19: Asset returns vs factor returns (single factor model)
#
# Selection:
```

Remarks:

- CUSUM plots (individual asset plot options 15, 16 and 17) are applicable only for `fit.method="LS"`.
- Modified residuals, rolling regression and single factor model plots (individual asset plot options 4, 18 and 19) are not applicable for `variable.selection="lars"`.

Here are a few more examples which don't need interactive user input.

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1", which=5, ylim=c(-0.06,0.06))

## Error in plot(fit.sub, plot.single = TRUE, asset.name = "HAM1", which = 5, :  object
'fit.sub' not found
```

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1", which=10)

## Error in plot(fit.sub, plot.single = TRUE, asset.name = "HAM1", which = 10):  object
'fit.sub' not found
```

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1", which=14)

## Error in plot(fit.sub, plot.single = TRUE, asset.name = "HAM1", which = 14):  object
'fit.sub' not found

grid()

## Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...):  plot.new has
not been called yet
```

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1", which=11)

## Error in plot(fit.sub, plot.single = TRUE, asset.name = "HAM1", which = 11):  object
'fit.sub' not found
```

```
plot(fit.sub, plot.single=TRUE, asset.name="HAM1", which=12)

## Error in plot(fit.sub, plot.single = TRUE, asset.name = "HAM1", which = 12):  object
'fit.sub' not found
```

References

E. Epperlein and A. Smillie. Portfolio risk analysis Cracking VAR with kernels. *RISK-LONDON-RISK MAGAZINE LIMITED-*, 19(8):70, 2006.

E. F. Fama and K. R. French. The cross-section of expected stock returns. *the Journal of Finance*, 47(2):427–465, 1992.

R. C. Grinold and R. N. Kahn. Active portfolio management. 2000.

A. Meucci. Risk contributions from generic user-defined factors. *RISK-LONDON-RISK MAGAZINE LIMITED-*, 20(6):84, 2007.

E. Zivot and W. Jia-hui. Modeling Financial Time Series with S-Plus Springer-Verlag. 2006.