# An Object-Oriented Framework for Robust Factor Analysis *

**Ying-Ying Zhang**
Chongqing University

**Abstract**

Taking advantage of the S4 class system of the programming environment R, which facilitates the creation and maintenance of reusable and modular components, an object-oriented framework for robust factor analysis was developed. The framework resides in the packages **robustfa** and **rrcov**. The design of the framework follows common statistical design patterns. The application of the framework to multivariate data analysis is demonstrated on a stocks example which itself is part of the package **robustfa**.

*Keywords*: robustness, factor analysis, object-oriented framework, R, statistical design patterns.

## 1. Introduction

Outliers exist in virtually every data set in any application domain. In order to avoid the masking effect, robust estimators are needed. The classical estimators of multivariate location and scatter are the sample mean $\bar{\mathbf{x}}$ and the sample covariance matrix $\mathbf{S}$. These estimates are optimal if the data come from a multivariate normal distribution but are extremely sensitive to the presence of even a few outliers. If outliers are present in the input data they will influence the estimates $\bar{\mathbf{x}}$ and $\mathbf{S}$ and subsequently worsen the performance of the classical factor analysis (Pison *et al.* 2003). Therefore it is important to consider robust alternatives to these estimators. There are several robust estimators in the literature: MCD (Rousseeuw 1985; Rousseeuw and Driessen 1999), OGK (Maronna and Zamar 2002), MVE (Rousseeuw 1985), M (Rocke 1996), S (Davies 1987; Ruppert 1992; Woodruff and Rocke 1994; Rocke 1996; Salibian-Barrera and Yohai 2006) and Stahel-Donoho (Stahel 1981a,b; Donoho 1982; Maronna and Yohai 1995). Substituting the classical location and scatter estimates by their robust analogues is the most straightforward method for robustifying many multivariate procedures (Maronna *et al.* 2006; Todorov and Filzmoser 2009), which is our choice for robustifying the factor analysis procedure.

Taking advantage of the new S4 class system (Chambers 1998) of R (R Development Core Team 2009) which facilitate the creation of reusable and modular components an object-oriented framework for robust factor analysis was implemented. The goal of the framework is manyfold (Todorov and Filzmoser 2009):

1. to provide the end-user with a flexible and easy access to newly developed robust factor

*The research was supported by Natural Science Foundation Project of CQ CSTC CSTC2011BB0058.

analysis methods for multivariate data analysis;

2. to allow the programming statisticians an extension by developing, implementing and testing new methods with minimum effort, and

3. to guarantee the original developers and maintainer of the packages a high level of maintainability.

The application of the framework to multivariate data analysis is demonstrated on a stocks example which itself is part of the package **robustfa**. We follow the object-oriented paradigm as applied to the R object model (naming conventions, access methods, coexistence of S3 and S4 classes, usage of UML, etc.) as described in Todorov and Filzmoser (2009). The framework is implemented in the R package **robustfa** which is available by sending an email to the author.

The rest of the paper is organized as follows. In the next Section 2 the design approach and structure of the framework are given. Section 3 describes the robust factor analysis method, its computation and implementation. The Sections 3.1, 3.2, and 3.3 are dedicated to the object model, method of robust factor analysis, and a stocks example, respectively. Section 4 concludes.

## 2. Design approach and structure of the framework

We follow the route of Todorov and Filzmoser (2009). The main part of the framework is implemented in the package **robustfa** but it relies on codes in the packages **rrcov** (Todorov 2009), **robustbase** (Rousseeuw *et al.* 2009), and **pcaPP** (Filzmoser *et al.* 2009). The structure of the framework and its relation to other R packages are shown in Figure 1. Like **robust** (Wang *et al.* 2008), **robustfa** extends **rrcov** with options for dealing with robust factor analysis problems.

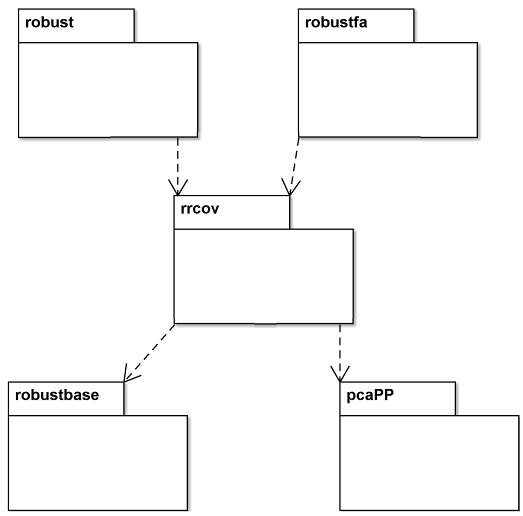


Figure 1: Class diagram: structure of the framework and relation to other R packages.

# 3. Robust factor analysis

The following example in Figure 2 illustrates the effect of outliers on the classical factor analysis. We use the Hawkins, Bradu and Kass data set `hbk` from the package **robustbase** consists of 75 observations in 4 dimensions (one response and three explanatory variables). The first 10 observations are bad leverage points, and the next four points are good leverage points (i.e., their **x** are outlying, but the corresponding **y** fit the model quite well). We will consider only the X-part of the data. The left panel shows the plot of the scores on the first two classical factors (the first two factors account for 99.4% of the total variation). The outliers are identified as separate groups, but the regular points are far from the origin (where the mean of the scores should be located). Furthermore, the 97.5% tolerance ellipse does not cover the regular points, which shows that the tolerance ellipse is highly influenced by the outliers. The right panel shows the same plot based on robust estimates. We see that the estimate of the center is not shifted by the outliers and these outliers are clearly separated by the 97.5% tolerance ellipse.

```
R> ##
R> ## Load the 'robustfa' package and two data sets
R> ##
R> library("robustfa")
R> data("hbk")
R> hbk.x <- hbk[,1:3]              # take only the X part
R> data("stock611")
R> stock608 = stock611[-c(92,2,337),]
R> stock604 = stock611[-c(92,2,337,338,379,539,79),]
R> R611 = cor(stock611[,3:12])
```

## 3.1. Object model

The object model for the `S4` classes and methods implementing the robust factor analysis follows the Unified Modeling Language (UML) (OMG 2009a,b) class diagram and is presented in Figure 3. A class is denoted by a box with three compartments which contain the name, the attributes (slots) and operations (methods) of the class, respectively. The class names, `Fa` and `FaRobust`, in italics indicate that the classes are abstract. Each attribute is followed by its type and each operation–by the type of its return value. We use the `R` types like `numeric`, `vector`, `matrix`, etc. but the type can be also a name of an `S4` class (`Fa`, `FaClassic`, or `FaCov`). The classes `Ulogical`, `Unumeric` etc. are class unions for optional slots, e.g., for definition of slots which will be computed on demand. Relationships between classes are denoted by arrows with different form. The inheritance relationship is depicted by a large empty triangular arrowhead pointing to the base class. We see that both `FaClassic` and `FaRobust` inherit from `Fa`, and `FaCov` inherits from `FaRobust`. Composition means that one class contains another one as a slot. This relation is represented by an arrow with a solid diamond on the side of the composed class. We see that `SummaryFa` is a composed class of `Fa`.

As in Todorov and Filzmoser (2009), all UML diagrams of the framework were created with the open source UML tool **ArgoUML** (Robbins 1999; Robbins and Redmiles 2000) which

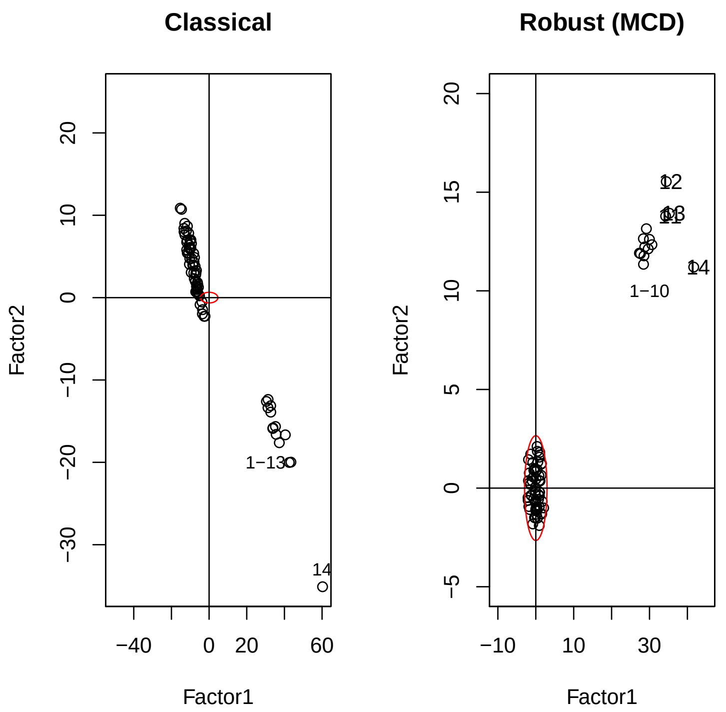


Figure 2: Plot of the first two factors of `hbk`: classical and robust.

is available for download from `http://argouml.tigris.org/`. The naming conventions of the framework follow the recommended Sun's Java coding style. See `http://java.sun.com/docs/codeconv/`.

## 3.2. FA based on a robust covariance matrix (MCD, OGK, MVE, etc.)

As in Todorov and Filzmoser (2009), the most straightforward and intuitive method to obtain robust factor analysis is to replace the classical estimates of location and covariance by their robust analogues. The package **stats** in base R contains the function `factanal()` which performs a factor analysis on a given numeric data matrix and returns the results as an object of S3 class `factanal`. This function has an argument `covmat` which can take a covariance matrix, or a covariance list as returned by `cov.wt`, and if supplied, it is used rather than the covariance matrix of the input data. This allows to obtain robust factor analysis by supplying the covariance matrix computed by `cov.mve` or `cov.mcd` from the package **MASS**. The reason to include such type of function in the framework is the unification of the interfaces by leveraging the object orientation provided by the S4 classes and methods. The function `FaCov()` computes robust factor analysis by replacing the classical covariance matrix with one of the robust covariance estimators available in the framework—MCD, OGK, MVE, M, S or Stahel-Donoho, i.e., the parameter `cov.control` can be any object of a class derived from the base class `CovControl`. This control class will be used to compute a robust estimate of the covariance matrix. If this parameter is omitted, MCD will be used by default. Of course any newly developed estimator following the concepts of the framework can be used as input

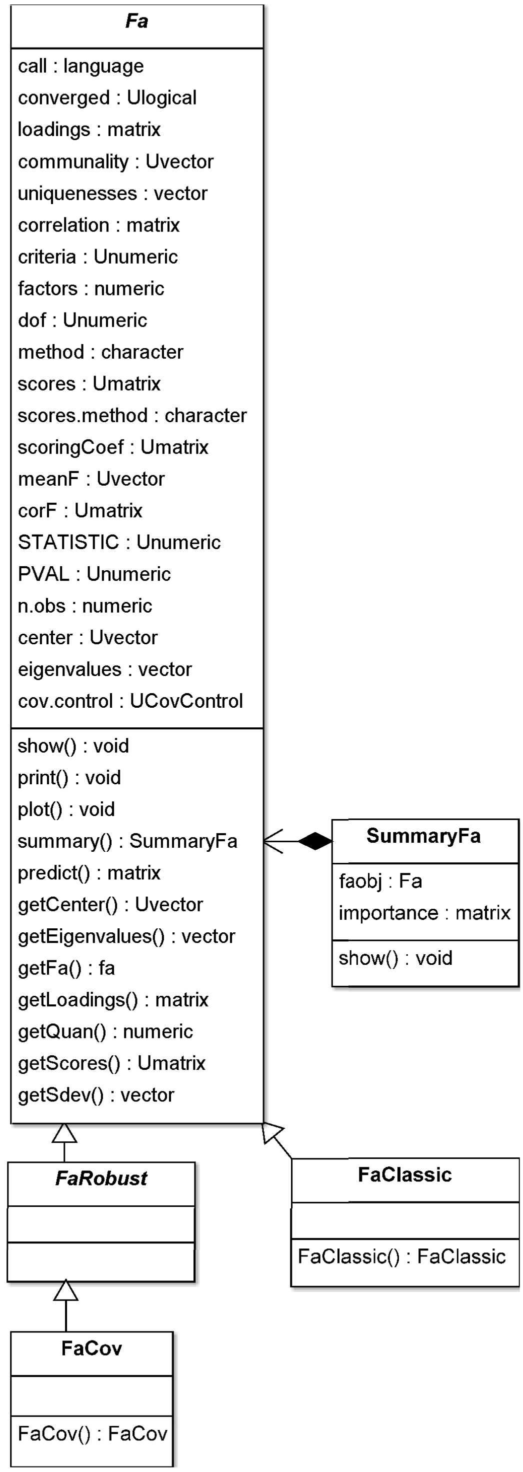


Figure 3: Object model for robust factor analysis.

to the function FaCov().

### 3.3. Example: Stocks data

In this Section, a data set stock611 is used to show the base functionalities of the robust factor analysis framework. This data set consists of 611 observations with 12 variables. The data set is from Chinese stock market in the year 2001. It is used in Wang (2009) to illustrate factor analysis methods.

Robust factor analysis is represented by the class FaCov which inherits from class Fa by class FaRobust of distance 2, and uses all slots and methods defined from Fa. The function FaCov() serves as a constructor (generating function) of the class. It can be called either by providing a data frame or matrix or a formula with no response variable, referring only to numeric variables. The usage of the default method of FaCov is:

```
FaCov(x, factors = 2, cov.control = CovControlMcd(),
method = c("mle", "pca", "pfa"),
scoresMethod = c("none", "regression", "Bartlett"), ...)
```

where x is a numeric matrix or an object that can be coerced to a numeric matrix. factors is the number of factors to be fitted. cov.control specifies which covariance estimator to use by providing a CovControl-class object. The default is CovControlMcd-class which will indirectly call CovMcd. If cov.control = NULL is specified, the classical estimates will be used by calling CovClassic. method is the method of factor analysis, one of "mle" (the default), "pca", and "pfa". scoresMethod specifies which type of scores to produce. The default is "none", "regression" gives Thompson's scores, and "Bartlett" gives Bartlett's weighted least-squares scores (Xue and Chen 2007). The usage of the formula interface of FaCov is:

```
FaCov(formula, data = NULL, factors = 2, method = "mle",
scoresMethod = "none", ...)
```

where formula is a formula with no response variable, referring only to numeric variables. data is an optional data frame containing the variables in the formula. Classical factor analysis is represented by the class FaClassic which inherits directly from Fa, and uses all slots and methods defined there. The function FaClassic() serves as a constructor (generating function) of the class. It also has its default method and formula interface as FaCov().

The code line

```
R> ##
R> ## facovRegOgk is obtained from FaCov.default,
R> ##                uses the default method = "mle"
R> ##
R> facovRegOgk = FaCov(x = scale(stock611[,3:12]), factors = 3,
+  cov.control = CovControlOgk(), scoresMethod = "regression")
```

generates an object facovRegOgk of class FaCov, where x is a scaled (standardized) numeric matrix. In fact, it is equivalent to use the formula interface

```
R> ## In fact, it is equivalent to use FaCov.formula
R> ## facovForRegOgkMle = facovRegOgk
R> facovForRegOgkMle = FaCov(~., data = as.data.frame(scale(stock611[,3:12])),
+  factors = 3, cov.control = CovControlOgk(), method = "mle",
+  scoresMethod = "regression")
```

That is `facovRegOgk = facovForRegOgkMle`. Type

```
R> ## obj = show(obj) = print(obj) = myFaPrint(obj)
R> class(facovRegOgk)

[1] "FaCov"
attr(,"package")
[1] "robustfa"
```

We see that the class `FaCov` is defined in the package **robustfa**. For an object `obj` of class `Fa`, we have `obj = show(obj) = print(obj) = myFaPrint(obj)`. Here `show()` and `print()` are `S4` generic functions, while `myFaPrint()` is an `S3` function acting as a function definition for `setMethod` of the generic functions `show` and `print`.

```
R> show(facovRegOgk)

Call:
FaCov(x = scale(stock611[, 3:12]), factors = 3, cov.control = CovControlOgk(),
    scoresMethod = "regression")

Standard deviations:
 [1] 2.2712126 1.5508138 1.0753673 0.8269330 0.4632001 0.4302182
 [7] 0.3448025 0.1828619 0.1768218 0.1144628

Loadings:
        Factor1      Factor2     Factor3
x1   0.03497840  0.621425042  0.21467039
x2   0.26612308  0.712587019  0.21976615
x3   0.73343068  0.655298981  0.11696274
x4   0.74258997  0.651317351  0.10243603
x5   0.81139278  0.110600355  0.56962048
x6   0.19019678  0.172383512  0.73236654
x7   0.86410352 -0.001556731  0.13069929
x8   0.96132464 -0.005838747  0.05779281
x9  -0.04622534  0.932727757  0.15414752
x10  0.03771512  0.807563540 -0.40194790
```

From Figure 3 we see that `summary()` generates an object of class `SummaryFa` which has its own `show()` method.

```
R> summaryFacovRegOgk = summary(facovRegOgk); summaryFacovRegOgk
```

```
Call:
FaCov(x = scale(stock611[, 3:12]), factors = 3, cov.control = CovControlOgk(),
    scoresMethod = "regression")
Importance of components:
               Factor1 Factor2 Factor3
SS loadings      3.530   3.312   1.185
Proportion Var   0.353   0.331   0.119
Cumulative Var   0.353   0.684   0.803
```

Next we calculate prediction/scores using `predict()`. The usage is `predict(object, ...)`, where `object` is an object of class `Fa`. If missing `...`(`newdata`), the `scores` slot of `object` is used. To save space, only the first five and last five rows of the scores are displayed.

```
R> ## If missing newdata, the scores are used
R> predict(facovRegOgk)

      Factor1    Factor2    Factor3
1   -3.262637  2.8221686 -1.7092313
2   -2.419031  2.1811941 -1.3034082
3   -2.933711  2.4284628 -1.8735330
4   -2.691886  2.4220225 -2.0482075
5   -2.902526  2.3891286 -2.1738301
607  1.088122 -0.9009931 -0.4127510
608  1.330205 -1.2581093  1.3890134
609  1.253710 -1.0038307 -0.2261910
610  1.443916 -1.4849762  1.3876016
611  1.533917 -1.4868511  0.7715401
```

If not missing `...`, then `...` must have the name `newdata`. Moreover, `newdata` should have the same `center` and `scale` attributes as the original data. For example, the original data is `x = scale(stock611[,3:12])`, and `newdata` is a one row `data.frame`.

```
R> ## If not missing newdata, newdata must have the same center and
R> ## scale attributes as the original data.
R> newdata = stock611[1,3:12]
R> x = scale(stock611[,3:12])
R> newdata = scale(newdata, center = attr(x,"scaled:center"),
+                           scale = attr(x,"scaled:scale"))
```

Then, to compute the prediction, `newdata` should be scaled using the robust center of the previously computed `Fa` object. This is actually done in "`FaCov.default`" when computing the scores:

```
scores <- computeScores(out,
newdata = scale(data, center = covmat$center, scale = F),
scoresMethod = scoresMethod)
```

Finally we get

```
R> ## To compute prediction, newdata should be scaled using the robust center,
R> ## this is done in "FaCov.default" when computing the scores:
R> ## scores <- computeScores(out,
R> ##           newdata = scale(data, center = covmat$center, scale = F),
R> ##           scoresMethod = scoresMethod)
R> ## Now, prediction = predict(facovRegOgk)[1,] = facovRegOgk@scores[1,]
R> newdata = scale(newdata, center = facovRegOgk@center, scale = F)
R> prediction = predict(facovRegOgk, newdata = newdata)
R> prediction
```

```
    Factor1  Factor2   Factor3
1 -3.262637 2.822169 -1.709231
```

One can easily check that

```
prediction = predict(facovRegOgk)[1,] = facovRegOgk@scores[1,]
```

To visualize the factor analysis results, the `plot` method can be used. We have `setMethod` `plot` with signature `x = "Fa", y = "missing"`. The usage is

```
plot(x, which = c("factorScore", "screeplot"), choices = 1:2)
```

Where `x` is an object of class `Fa`. The argument `which` indicates what kind of plot. If `which = "factorScore"`, then a scatterplot of the factor scores is produced; if `which = "screeplot"`, then the eigenvalues plot is created which is helpful to select the number of factors. The argument `choices` is an integer vector of length two indicating which columns of the factor scores to plot. To see how `plot` is functioning, we first generate the `Fa` objects. The following code lines use the default `method = "mle"`, and result in errors:

```
R> faclassicReg611 = FaClassic(x = scale(stock611[,3:12]), factors = 3,
+       scoresMethod = "regression"); faclassicReg611
```

```
Error in factanal(factors = factors, covmat = covmat) :
  unable to optimize from these starting value(s)
```

So we change to use `method = "pca"`, and it works.

```
R> ## method = "mle" error
R> ##  faclassicReg611 = FaClassic(x = scale(stock611[,3:12]), factors = 3,
R> ##        scoresMethod = "regression"); faclassicReg611
R> ## Error in factanal(factors = factors, covmat = covmat) :
R> ##   unable to optimize from these starting value(s)
R>
R> ## method = "pca" OK
R> ## x = scale(stock611[,3:12])
R>
R> ## FaClassic
R> faclassicRegPca611 = FaClassic(x = scale(stock611[,3:12]), factors = 2,
+  method = "pca", scoresMethod = "regression"); faclassicRegPca611
```

```
Call:
FaClassic(x = scale(stock611[, 3:12]), factors = 2, method = "pca",
    scoresMethod = "regression")

Standard deviations:
 [1] 2.40625224 1.52281163 1.00433419 0.75771973 0.37846924 0.31490830
 [7] 0.22719326 0.09697058 0.06300416 0.02779799

Loadings:
        Factor1     Factor2
X1   0.98781949 -0.02611587
X2   0.99154548 -0.00633362
X3   0.99222167  0.05353998
X4   0.98486045  0.07865320
X5   0.05473434  0.95570764
X6  -0.01321817  0.62527581
X7   0.01924642  0.48022545
X8   0.04578987  0.88138600
X9   0.94053884 -0.01792872
X10  0.99068004 -0.04474676

R> summary(faclassicRegPca611)

Call:
FaClassic(x = scale(stock611[, 3:12]), factors = 2, method = "pca",
    scoresMethod = "regression")
Importance of components:
               Factor1 Factor2
SS loadings      5.785   2.324
Proportion Var   0.579   0.232
Cumulative Var   0.579   0.811
```

faclassicRegPca611 is an object of class FaClassic. From the show result of faclassicRegPca611, we see that its Factor1 explains variables x1-x4, x9, x10; its Factor2 explains variables x5-x8 (with loadings larger than 0.48). From the summary result of faclassicRegPca611, we see that the first two factors accout for about 81.1% of its total variance. Next we generate an object facovRegOgkPca of class FaCov using the same data set.

```
R> ## FaCov
R> facovRegOgkPca = FaCov(x = scale(stock611[,3:12]), factors = 2, method = "pca",
+  cov.control = CovControlOgk(), scoresMethod = "regression"); facovRegOgkPca

Call:
FaCov(x = scale(stock611[, 3:12]), factors = 2, cov.control = CovControlOgk(),
    method = "pca", scoresMethod = "regression")

Standard deviations:
```

```
 [1] 2.2712126 1.5508138 1.0753673 0.8269330 0.4632001 0.4302182
 [7] 0.3448025 0.1828619 0.1768218 0.1144628

Loadings:
       Factor1      Factor2
X1   0.1192720  0.771765320
X2   0.3585258  0.808017502
X3   0.7582825  0.596037255
X4   0.7604049  0.586322722
X5   0.9546105  0.072062014
X6   0.4300313  0.111234443
X7   0.8787653 -0.002537877
X8   0.9349237 -0.028219013
X9   0.0457114  0.931164519
X10 -0.0712708  0.827516094
```

```
R> summary(facovRegOgkPca)
```

```
Call:
FaCov(x = scale(stock611[, 3:12]), factors = 2, cov.control = CovControlOgk(),
    method = "pca", scoresMethod = "regression")
Importance of components:
               Factor1 Factor2
SS loadings      4.046   3.518
Proportion Var   0.405   0.352
Cumulative Var   0.405   0.756
```

From the `show` result of `facovRegOgkPca`, we see that its `Factor1` explains variables `x3-x8` (with loadings larger than 0.43); its `Factor2` explains variables `x1-x4`, `x9`, `x10`. Thus `Factor1` (`Factor2`) of `faclassicRegPca611` and `Factor2` (`Factor1`) of `facovRegOgkPca` are similar. From the `summary` result of `facovRegOgkPca`, we see that the first two factors accout for about 75.6% of its total variance.

The following code lines generate classical and robust scatterplot of the first two factor scores. See Figure 4.

```
R> usr <- par(mfrow=c(1,2))
R> plot(faclassicRegPca611, which = "factorScore", choices = 2:1)
R> plot(facovRegOgkPca, which = "factorScore", choices = 1:2)
R> par(usr)
```

The following code lines generate classical and robust scree plot. See Figure 5.

```
R> usr <- par(mfrow=c(1,2))
R> plot(faclassicRegPca611, which = "screeplot")
R> plot(facovRegOgkPca, which = "screeplot")
R> par(usr)
```

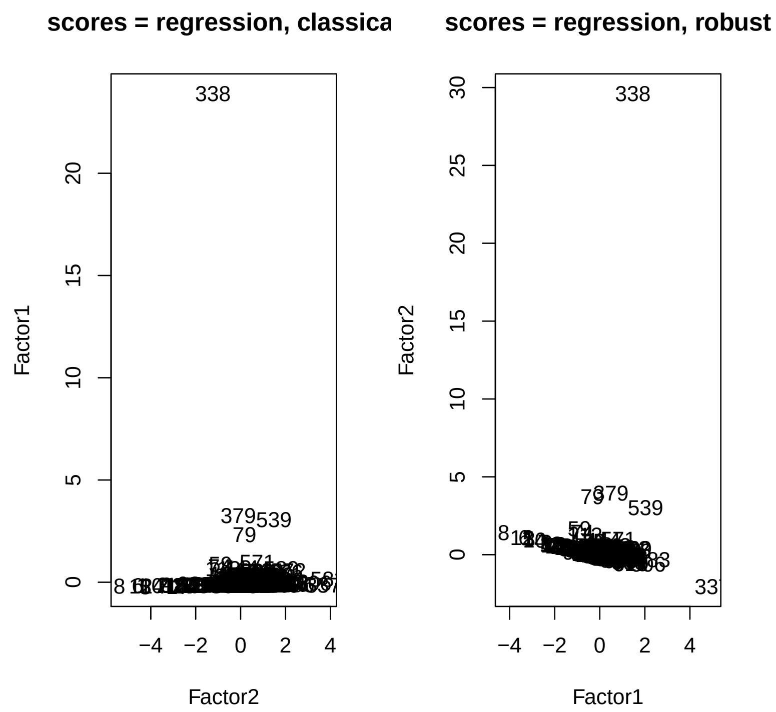


Figure 4: Classical and robust scatterplot of the first two factor scores of the stocks data.

Next we impose a 97.5% tolerance ellipse on the scatterplot of the first two factors of the stocks data by using a function `rrcov:::.myellipse`. See Figure 6.

We see that the two scatterplots are similar. However, the robust 97.5% tolerance ellipse is tighter than that of classical. By inspecting the classical and robust ordered scores, we find that they are quite different. In the following, `orderedFsC[[1]]` and `orderedFsOgk[[1]]` are decreasing on their first column; `orderedFsC[[2]]` and `orderedFsOgk[[2]]` are decreasing on their second column. To save space, only the first 10 rows of the scores are displayed.

```
R> orderedFsC = fsOrder(faclassicRegPca611@scores[,2:1]); orderedFsC

[[1]]
     Factor2       Factor1
583 3.898163  1.442520e-01
337 3.681513 -1.520371e-01
606 3.263503 -5.357695e-02
486 2.503726 -1.257069e-01
563 2.252890 -8.903074e-03
598 2.247478 -1.065788e-01
454 2.129092 -1.118824e-01
572 2.123974  5.649819e-01
588 2.006510  1.021676e-01
526 1.995515  1.440485e-05
```

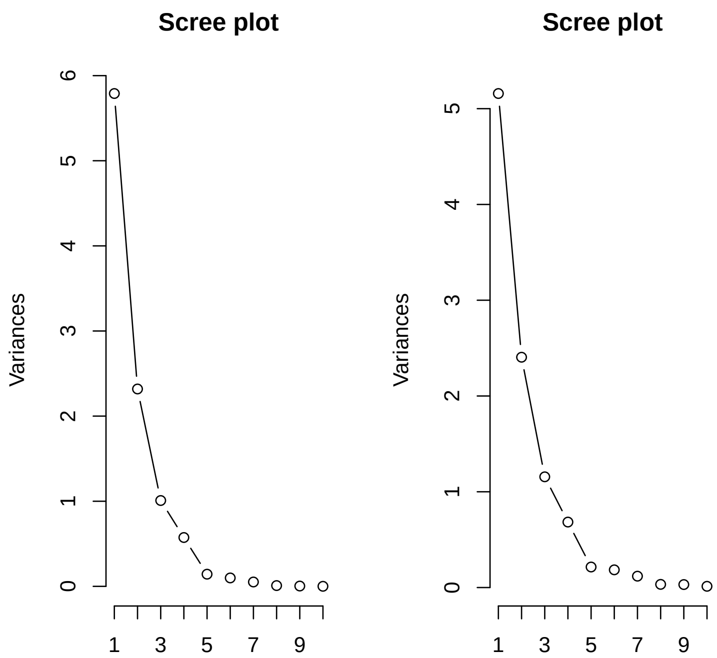


Figure 5: Classical and robust scree plot of the stocks data.

```
[[2]]
        Factor2    Factor1
338 -1.22930368 23.8967397
379 -0.10141001  3.2591570
539  1.48465281  3.0500510
79   0.17308239  2.3347627
571  0.74880375  0.9807258
59  -0.89391045  0.8830250
74  -0.92530280  0.7621726
444  0.01917557  0.7219354
589  1.81077486  0.6714384
113 -0.78396701  0.6627497
```

```
R> orderedFsOgk = fsOrder(facovRegOgkPca@scores[,1:2]); orderedFsOgk
```

```
[[1]]
     Factor1    Factor2
337 4.999632 -2.0584790
583 2.350900 -0.3137101
606 2.139466 -0.6254701
539 2.033750  3.0066474
```

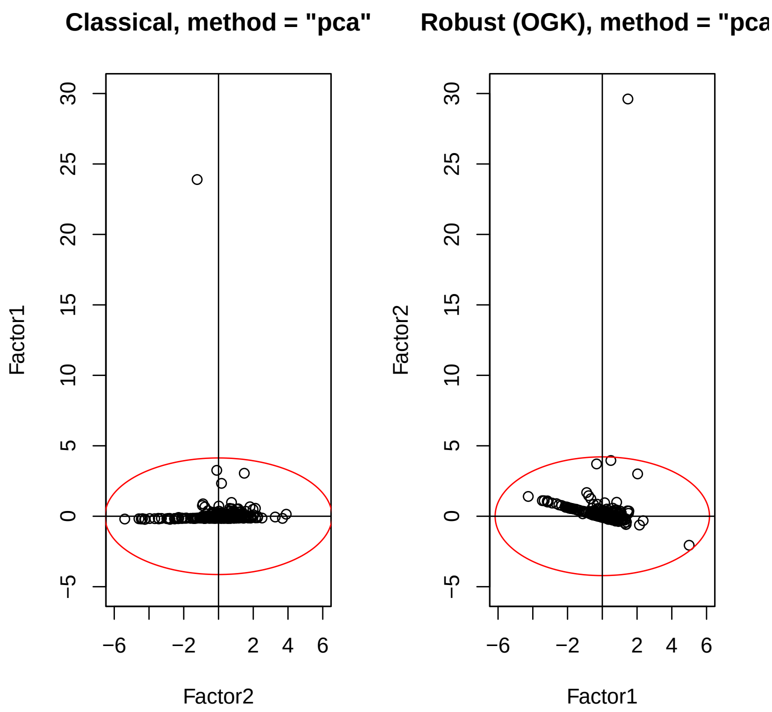


Figure 6: Classical and robust scatterplot of the first two factor scores of the stocks data with a 97.5% tolerance ellipse.

```
589 1.530120  0.3694489
576 1.506893  0.2107224
338 1.472831 29.6137874
572 1.449248  0.3826593
598 1.389924 -0.4146966
610 1.373669 -0.5685485

[[2]]
       Factor1   Factor2
338  1.4728308 29.613787
379  0.4920890  3.956331
79  -0.3287029  3.712642
539  2.0337498  3.006647
59  -0.8996599  1.681898
74  -0.7868141  1.458448
8   -4.2627943  1.404842
113 -0.6601598  1.244956
12  -3.4504160  1.106161
6   -3.3461210  1.101734
```

Next we utilize the `plot-methods` defined in the package `rrcov` which can plot a `Cov-class`.

```
R> ##
R> ## all the following plots are OK for x = covOgk
R> ## myplotDD shows id.n and ind
R> ##
R> covOgk = CovRobust(x = scale(stock611[,3:12]), control = "ogk"); covOgk
R> covOgk68 = CovRobust(x = scale(stock611[,c(8,10)]), control = "ogk")

R> plot(x = covOgk, which = "dd")

R> result = myplotDD(x = covOgk)

R> plot(x = covOgk, which = "distance", classic = T)

R> plot(x = covOgk, which = "qqchi2", classic = T)

R> plot(x = covOgk68, which = "tolEllipsePlot", classic = T)

R> plot(x = covOgk, which = "screeplot", classic = T)
```

See Figure 7. The two plots in the first row show distance-distance plots. We see that the robust (mahalanobis) distances are far larger than the (classical) mahalanobis distances. The outliers have large robust distances. The right plot is generated by `myplotDD(x = covOgk)`. `myplotDD` is a revised version of `.myddplot` in `plot-utils.R` in the package `rrcov`. In `myplotDD`, `id.n` and `ind` are printed out. Here `id.n` is the number of observations to identify by a label. By default, the number of observations with robust distance larger than `cutoff` is used. By default `cutoff <- sqrt(qchisq(0.975, p))`. `ind` is the index of robust distance whose values are larger than `cutoff`.

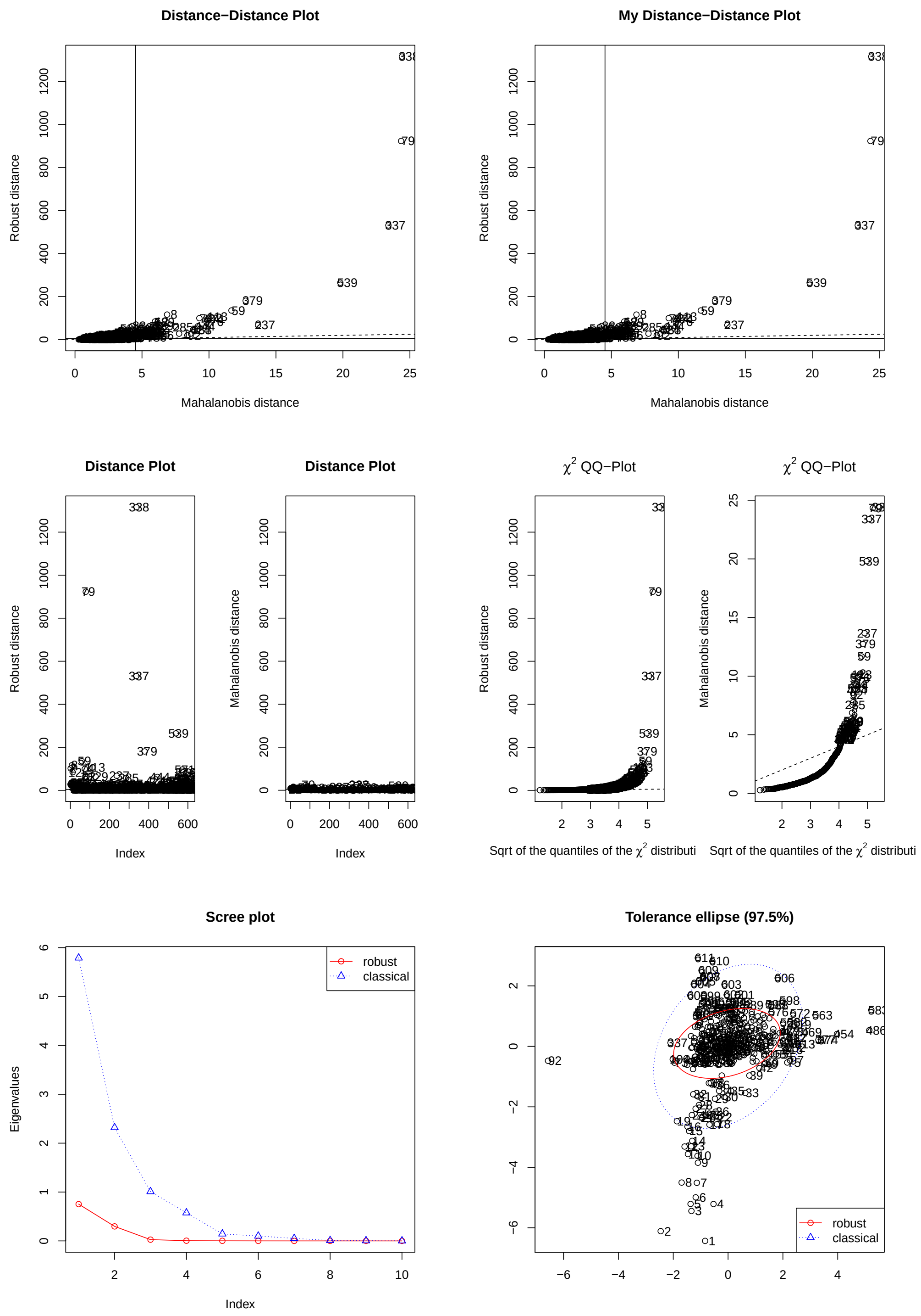


Figure 7: Mahalanobis distances based on robust and classical estimates of the location and the covariance matrix in different plots.

```
cutoff = 4.525834

id.n <- length(which(rd>cutoff))

id.n = 287

Here y is the robust distance (rd).

sort.y = (To save space, only the smallest five and largest five
elements of sort.y$x and sort.y$ix are shown.)

$x
        281          368          239          312          229
  0.6880160    0.8109610    0.9234880    0.9763902    1.0429306
        379          539          337           79          338
180.7741644  264.6799442  531.0887434  923.0346401 1315.6544592

$ix
 [1] 281 368 239 312 229 379 539 337  79 338

ind =

  [1] 601 577  36 202 429 387 466 290 570  46 203 566  29 559 313 397
 [17] 582 122 597 517 192 596 489 261 417 133  30 266 105 549 585 462
 [33] 493 565 591 547 469 513 289 420 222 502 382 163 194 406 252  41
 [49]  69 532  73  34 602 435 372 150 481 276 148  38 278 139 401 350
 [65] 340  70 273 132  72 374 314 366  96 586 394 169  81  49 506 407
 [81] 184 173 603 524 391  39 141 356 561 599 102 357 538 418 568  37
 [97] 220 124 187 293  84 578 590 437 459 144  33 518 523 225 156 198
[113] 419 531  83 600 322 254 519 454 384 495 260  22 554  35  63  23
[129] 351 609 354 399 448 449 213 174 478 557 188 607 166 442 503 190
[145] 558 574 104 371 343 365 598 145 176  31 467 486  40 463 287 103
[161] 567 253 346  61 608 563 117 140 552 409 182 378  26 579 575  28
[177] 332  97 415 610  53 546 320 108 479 295 341 548  21  32 584  94
[193]  20 272 471  42 611 544 160  18  87 377 526 288 339 321 138  44
[209] 606 226  25 153  66  24  14 592 595  92  13 605   7 344  57  16
[225] 175 309   9  11  19 193   4 413 345 303 504 206 604 216 472 553
[241]  17  15   1 230 580   3 307  27  10   6 114 562 594   5  56 588
[257] 541 492 490 540 583 542 424 593 581 569 572 285  75 444 129  82
[273] 237 589 576  12 571  74   2 113   8  59 379 539 337  79 338
```

From the above results we see that the `cutoff` is computed as 4.5258344369638. There are `id.n = 287` observations with robust distance larger than `cutoff`. `sort.y` is a list containing the sorted values of `y` (the robust distance). `sort.y$x` is arranged in increasing order. To save space, only the smallest five and largest five robust distances with their indices are shown. `sort.y$ix` contains the indices. `ind` shows the indices of the largest `id.n = 287` observations whose robust distances are larger than `cutoff`. The two plots in the second row show a index plot and a Chisquare QQ-plot of the robust and mahalanobis distances. We also see that the robust distances are far larger than the mahalanobis distances and the outliers have large robust distances. The left plot in the third row shows a eigenvalues comparison plot. We see that the largest four eigenvalues of the robust method are much smaller than those of the classical method, and the largest several eigenvalues of the classical method decrease very fast. The right plot in the third row shows robust and classical 97.5% tolerance ellipses plot. We see that the robust tolerance ellipse is tighter than the classical one.

The accessor functions `getCenter()`, `getEigenvalues()`, `getFa()`, `getLoadings()`, `getQuan()`, `getScores()`, and `getSdev()` are used to access the corresponding slots. For instance

```
R> ##
R> ## accessor functions
R> ##
R> getEigenvalues(facovRegOgk)

 [1] 5.15840672 2.40502329 1.15641490 0.68381812 0.21455429 0.18508773
 [7] 0.11888878 0.03343847 0.03126596 0.01310173
```

Run the file "testsRobustfa.R" in the "tests" folder for more details. They explain why we choose `x = scale(stock611[,3:12])` as the date input; show other results by different robust estimators, e.g., MCD etc..

# 4. Conclusions

As in Todorov and Filzmoser (2009), in this paper we presented an object-oriented framework for robust factor analysis developed in the S4 class system of the programming environment R. The main goal of the framework is to support the usage, experimentation, development and testing of robust factor analysis method as well as simplifying comparisons with related methods. It minimizes the effort for performing any of the following tasks:

- application of the already existing robust factor analysis methods for practical data analysis;
- implementation of new robust factor analysis methods or variants of the existing ones;
- evaluation of existing and new methods by carrying out comparison studies.

A major design goal was the openness to extensions by the development of new robust factor analysis methods in the package **robustfa** or in other packages depending on **robustfa**.

# Acknowledgments

# References

Chambers JM (1998). *Programming with Data: A Guide to the S Language.* Springer-Verlag, New York.

Davies P (1987). "Asymptotic Behavior of S-Estimators of Multivariate Location Parameters and Dispersion Matrices." *The Annals of Statistics*, **15**, 1269–1292.

Donoho DL (1982). "Breakdown Properties of Multivariate Location Estimators." *Technical report*, Harvard University, Boston. URL http://www-stat.stanford.edu/~donoho/Reports/Oldies/BPMLE.pdf.

Filzmoser P, Fritz H, Kalcher K (2009). ***pcaPP**: Robust PCA by Projection Pursuit.* R package version 1.7, URL http://CRAN.R-project.org/package=pcaPP.

Maronna RA, Martin D, Yohai V (2006). *Robust Statistics: Theory and Methods.* John Wiley & Sons, New York.

Maronna RA, Yohai VJ (1995). "The Behavoiur of the Stahel-Donoho Robust Multivariate Estimator." *Journal of the American Statistical Association*, **90**, 330–341.

Maronna RA, Zamar RH (2002). "Robust Estimation of Location and Dispersion for High-Dimensional Datasets." *Technometrics*, **44**, 307–317.

OMG (2009a). "OMG Unified Modeling Language (OMG UML), Infrastructure, V2.2." *Current formally adopted specification*, Object Management Group. URL http://www.omg.org/spec/UML/2.2/Infrastructure/PDF.

OMG (2009b). "OMG Unified Modeling Language (OMG UML), Superstructure, V2.2." *Current formally adopted specification*, Object Management Group. URL http://www.omg.org/spec/UML/2.2/Superstructure/PDF.

Pison G, Rousseeuw PJ, Filzmoser P, Croux C (2003). "Robust Factor Analysis." *Journal of Multivariate Analysis*, **84**, 145–172.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Robbins JE (1999). "Cognitive Support Features for Software Development Tools." *Ph.d. thesis*, University of California, Irvine. URL http://argouml.tigris.org/docs/robbins-dissertation/.

Robbins JE, Redmiles DF (2000). "Cognitive Support, UML Adherence, and XMI Interchange in Argo/UML." *Information and Software Technology*, **42**, 79–89.

Rocke DM (1996). "Robustness Properties of S-Estimators of Multivariate Location and Shape in High Dimension." *The Annals of Statistics*, **24**, 1327–1345.

Rousseeuw PJ (1985). "Multivariate Estimation with High Breakdown Point." In W Grossmann, G Pflug, I Vincze, W Wertz (eds.), *Mathematical Statistics and Applications Vol. B*, pp. 283–297. Reidel Publishing, Dordrecht.

Rousseeuw PJ, Croux C, Todorov V, Ruckstuhl A, Salibian-Barrera M, Verbeke T, Maechler M (2009). ***robustbase**: Basic Robust Statistics.* R package version 0.4-5, URL http://CRAN.R-project.org/package=robustbase.

Rousseeuw PJ, Driessen KV (1999). "A Fast Algorithm for the Minimum Covariance Determinant Estimator." *Technometrics*, **41**, 212–223.

Ruppert D (1992). "Computing S-Estimators for Regression and Multivariate Location/Dispersion." *Journal of Computational and Graphical Statistics*, **1**, 253–270.

Salibian-Barrera M, Yohai VJ (2006). "A Fast Algorithm for S-regression Estimates." *Journal of Computational and Graphical Statistics*, **15**, 414–427.

Stahel WA (1981a). "Breakdown of Covariance Estimators." *Research Report 31*, ETH Zurich. Fachgruppe für Statistik.

Stahel WA (1981b). "Robuste Schätzungen: Infinitesimale Optimalität und Schätzungen von Kovarianzmatrizen." *Ph.d. thesis no. 6881*, Swiss Federal Institute of Technology (ETH), Zürich. URL http://e-collection.ethbib.ethz.ch/view/eth:21890.

Todorov V (2009). ***rrcov**: Scalable Robust Estimators with High Breakdown Point.* R package version 0.5-03, URL http://CRAN.R-project.org/package=rrcov.

Todorov V, Filzmoser P (2009). "An Object-Oriented Framework for Robust Multivariate Analysis." *Journal of Statistical Software*, **32**(3), 1–47. URL http://www.jstatsoft.org/v32/i03/.

Wang J, Zamar R, Marazzi A, Yohai V, Salibian-Barrera M, Maronna R, Zivot E, Rocke D, Martin D, Konis K (2008). ***robust**: Insightful Robust Library.* R package version 0.3-4, URL http://CRAN.R-project.org/package=robust.

Wang XM (2009). *Applied Multivariate Analysis*. ShangHai University of Finance & Economics Press, Shanghai. 3rd edition (This is a Chinese book).

Woodruff DL, Rocke DM (1994). "Computable Robust Estimation of Multivariate Location and Shape in High Dimension Using Compound Estimators." *Journal of the American Statistical Association*, **89**, 888–896.

Xue Y, Chen LP (2007). *Statistical Modeling and R Software*. Tsinghua University Press, Beijing. (This is a Chinese book).

### Affiliation:

Ying-Ying Zhang
Department of Statistics and Actuarial Science
College of Mathematics and Statistics
Chongqing University
Chongqing, China
E-mail: robertzhang@cqu.edu.cn
URL: http://baike.baidu.com/view/7694173.htm