RobStatTM Package Vignette

January 26, 2020

Ricardo A. Maronna, R. Douglas Martin, Victor J. Yohai, Matias Salibian-Barrera

Abstract

The RobStatTM R package is a companion to the book *Robust Statistics:Theory and Methods* (*with R*) by the authors of this document. The purpose of this vignette is to guide you on how to use the R scripts, and related data sets, in the package that replicate examples in the book. The vignette also includes instructions on how to use the `fit.models` function in RobStatTM to compare different regression model fits such as least squares versus robust fits, and compare different covariance matrix estimates such as classical and robust covariance matrix estimates. A short section on convergence of robust estimation algorithms is also included

Contents

1 The RobStatTM Scripts and Data Sets

The first column of Table 1 below lists the numbering of the examples in the book, and the second column contains the names in the book of the corresponding R scripts. The third and fourth columns provide the names of data sets used by the scripts. Some of the R scripts in the table make use of packages other than RobStatTM, for either accessing data, or for using certain R functions, and such packages are listed in the fifth column of the table. In order to run those scripts, you need to have already installed those package(s) from CRAN (https://cran.r-project.org/).

EXAMPLE	NAME	RobStatTM DATA	OTHER DATA	OTHER PACKAGES REQUIRED
4.1	shock.R	shock		quantreg
4.2	oats.R	oats		
5.1	mineral.R	mineral		quantreg
5.2	wood.R		wood	robustbase
5.3	step.R		made up data	
5.4	algae.R	algae		
5.5	ExactFit.R		synthetic data	
6.1	biochem.R	biochem		
6.2	wine.R	wine		
6.3	vehicle.R	vehicle		rrcov
6.4	bus.R	bus		
6.5-6.6	wine1.R	wine		GSE
6.7	autism.R		autism	WWGbook, robustvarComp, nlme
7.1	leukemia.R		leuk.dat	robust
7.2	skin.R	skin		
7.3	epilepsy.R		breslow.dat	robust
8.1	ar1.R		synthetic data	robustarima
8.2	ar3.R		synthetic data	robustarima
8.3	identAR2.R		synthetic data	robustarima
8.4	identMA1.R		synthetic data	robustarima
8.5	MA1-AO.R		synthetic data	robustarima
8.6	resex.R	resex		robustarima

Table 1: R Scripts and Data in the RobStatTM R Package

We note that the script flour.R, and the data set flour used by the script, contained in RobStatTM but not listed in the above table are for Example 1.1, Figure 2.1, and Table 2.4 of the book. Furthermore, the data set neuralgia contained in RobStatTM but not listed in the table above, is used only in Problem 7.1 of the book.

1.1 Accessing the Example R Scripts in Your Installed RobStatTM

In case you have not already installed RobStatTM, install it with the command:

```
install.packages("RobStatTM")
```

With RobStatTM installed, load it in your current R session with:

```
library("RobStatTM")
```

In order to access the example R scripts, you need to find your installed RobStatTM "scripts" folder, which is one of several RobStatTM package folders. You can find the location of the scripts folder on your computer by using the function `system.file()` as follows:

```
system.file("scripts", package = "RobStatTM")
```

NOTE: Copy/paste of the above line does not typically work, so you should type it in. The result of using this command will depend upon your computer. For example, in the case of a particular computer running Windows 10, the result is:

```
[1] "C:/Users/Doug/Documents/R/win-library/3.4/RobStatTM/scripts"
```

Then you just need to navigate to the `scripts` folder, where you will see all of the example R scripts, with the same name they are given in the book, i.e., the name in the NAME column of Table 1. You should then copy/paste any script, or all the scripts, to some other location on your computer where you want to run them.

1.2 Loading Data Sets Provided in RobStatTM

To load any data set that is in RobStatTM, just use the `data()` function with the data set name. For example, in the case of the `shock` data set:

```
data(shock)
head(shock, 2)

##   n.shocks time
## 1        0 11.4
## 2        1 11.9
```

1.3 Installing Other Packages Needed for Data or Functions Access

You will notice in the table on the previous page that there is a column named OTHER PACKAGES REQUIRED. The other package is required for accessing a data set in the package, or for using an R function in the package, or both. In the case of any example R script that requires another package for one of those reasons, you need to have installed the package before running the script. You could do this on a case by case basis, but it may be easier to just be sure and install all the packages in the OTHER PACKAGES REQUIRED once and for all. You can do this quite easily from RStudio.

1.4 Loading Data Sets Provided in Another Package

To load a data set that is in another package that you have already installed in your R (but have not loaded, and in fact should not load) you use the `data()` function with an optional argument that specifies the package the data is in. For example, in the case of the wood data set that is in the robustbase package:

```
data(wood, package = "robustbase")
head(wood, 1)

##      x1     x2    x3    x4    x5     y
## 1 0.573 0.1059 0.465 0.538 0.841 0.534
```

Note that the above use of `data()` does not result in loading the named package, and this is the recommended way to load data in another package. The reason for not first loading another package with `library()`, and then using the `data()` function, is that the result of loading a package can result in problematic masking of functions in RobStatTM.

1.5 Using Functions from Another Installed Package without Loading the Package

When a script requires a function from a package other than RobStatTM, the other package needs to have been already installed. However, in order to avoid masking problems mentioned above, the script does not load the library, and instead uses that standard package referencing mechanism "`::`". For example, you can use the function `rq` in the `quantreg` package to do an L1 (least absolute deviation regression) fit of `zinc` to `copper` in the `mineral` data set with the code below:

```
mineral11 <- quantreg::rq(zinc ~ copper,
    data = mineral)
```

1.6 Differences Between the Example R Scripts Results and the Book Examples

There are a number of example R scripts where part of the output, either pure numerical values or numerical values in plots, is not exactly the same as in the book examples. Scripts where this is the case include oats.R (different p-values, Figure 4.4), mineral.R (Figures 5.3, 5.5, 5.6), wood.R (Figures 5.10, 5.11), algae.R (Figure 5.14, 5.15), wine.R (Figure 6.11). This is because the final code for some of the scripts is different from the code used for the book examples, and the scripts code is usually improved in some way. Typically, the conclusions drawn from the figures, concerning which data points are outliers, and same for the book figures and the figures produced by the example scripts.

1.7 Running the Example Scripts

We recommend running all the example scripts, and especially the Table 1 Examples 5.1, 5.2, 5.3, 5.4, 5.5, which reflect our recent recommendation to use `family = "mopt"`, with `efficiency = 0.95` for the robust regression function `lmrobdetMM`.

2 Using the `fit.models` Function

The `fit.models` function in RobStatTM allows you to compare two different model estimates, such as model estimates with different variables, and especially classical versus robust model estimates. As such `fit.models` is a very useful function, and we illustrate its use in the next two subsections for: (1) classical versus robust linear regression model fits, and (2) classical versus robust covariance matrix estimates.

2.1 Robust Versus Least Squares Fits of Linear Regression Models

The code below use the functions `lm` and `lmrobdetMM` to compute and compare least squares (LS) and robust fits of the `zinc` data to the `copper` data in the `minerals` data set. You will note that the object created by `fit.models` has the class `lmfm`, where the "lm" stands for linear model and the "fm" stands for fitted models.

```
LSfit <- lm(zinc ~ copper, data = mineral)
control <- lmrobdet.control(family = "mopt",
    eff = 0.95)
robfit <- lmrobdetMM(zinc ~ copper, control = control,
    data = mineral)
fmLSrob <- fit.models(LSfit, robfit)
class(fmLSrob)
```

```
## [1] "lmfm"

summary(fmLSrob)

##
## Calls:
##  LSfit: lm(formula = zinc ~ copper, data = mineral)
## robfit: lmrobdetMM(formula = zinc ~ copper, data = mineral, control = control)
##
## Residual Statistics:
##            Min     1Q  Median    3Q    Max
##  LSfit: -41.62 -6.457 -0.3829 5.158  46.86
## robfit: -16.54 -5.421  1.0491 7.919 116.83
##
## Coefficients:
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept):  LSfit:  7.96063    2.70191   2.946  0.00484 **
##              robfit: 15.19455    2.25842   6.728 1.46e-08 ***
##
##      copper:  LSfit:  0.13457    0.01983   6.787 1.18e-08 ***
##               robfit:  0.01261    0.02232   0.565  0.57467
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual Scale Estimates:
##  LSfit: 14.66 on 51 degrees of freedom
## robfit: 9.994 on 51 degrees of freedom
##
## Multiple R-squared:
##  LSfit: 0.4746
## robfit: 0.008241
```

The function `summary` used above is a generic function in R, which calls a *method* function appropriate for the object class of its argument, and in this case the method is called `summary.lmfm`. You can view the help file for `summary.lmfm` with the code line:

```
help(summary.lmfm)
```

Likewise, when applied to an object of class `lmfm`, the generic `plot` function in R calls a `plot.lmfm` `method` function, whose behavior is described the help file obtained with the following code line:

```
help(plot.lmfm)
```

If you try the above code line, you will find out that the `plot.lmfm` method allows you to create the following 10 different types of plots:

1. (not used)
2. Normal QQ Plot of Residuals,
3. Kernel Density Estimate of Residuals,
4. Residuals vs. Mahalanobis Distance,
5. Residuals vs. Fitted Values,
6. Scale-Location,
7. Response vs. Fitted Values,
8. Residuals vs. Index (Time),
9. Overlaid Normal QQ Plot of Residuals,
10. Overlaid Kernel Density Estimate of Residuals,
11. Scatter Plot with Overlaid Fits (for simple linear regression models).

You can create any single one of the plots, or any subgroup of the 10 plots, through appropriate use of the generic function `plot` with an lmfm object argument. To see how this may be done, first look at the following code line and its results:

```
args(plot.lmfm)

## function (x, which.plots = c(5, 2, 6, 4), ...)
## NULL
```

It follows that if you use `plot(fmLSrob)` in RStudio, you will get plot types 5, 2, 6, 4 in that order in the Plots window by pressing the Enter key four times in RStudio Console right after the Console text line `Hit <Return> to see next plot`.

The following code will create the scatter plots with overlaid LS and Robust line fits shown in Figure 1.

```
plot(fmLSrob, which.plots = 11)
```

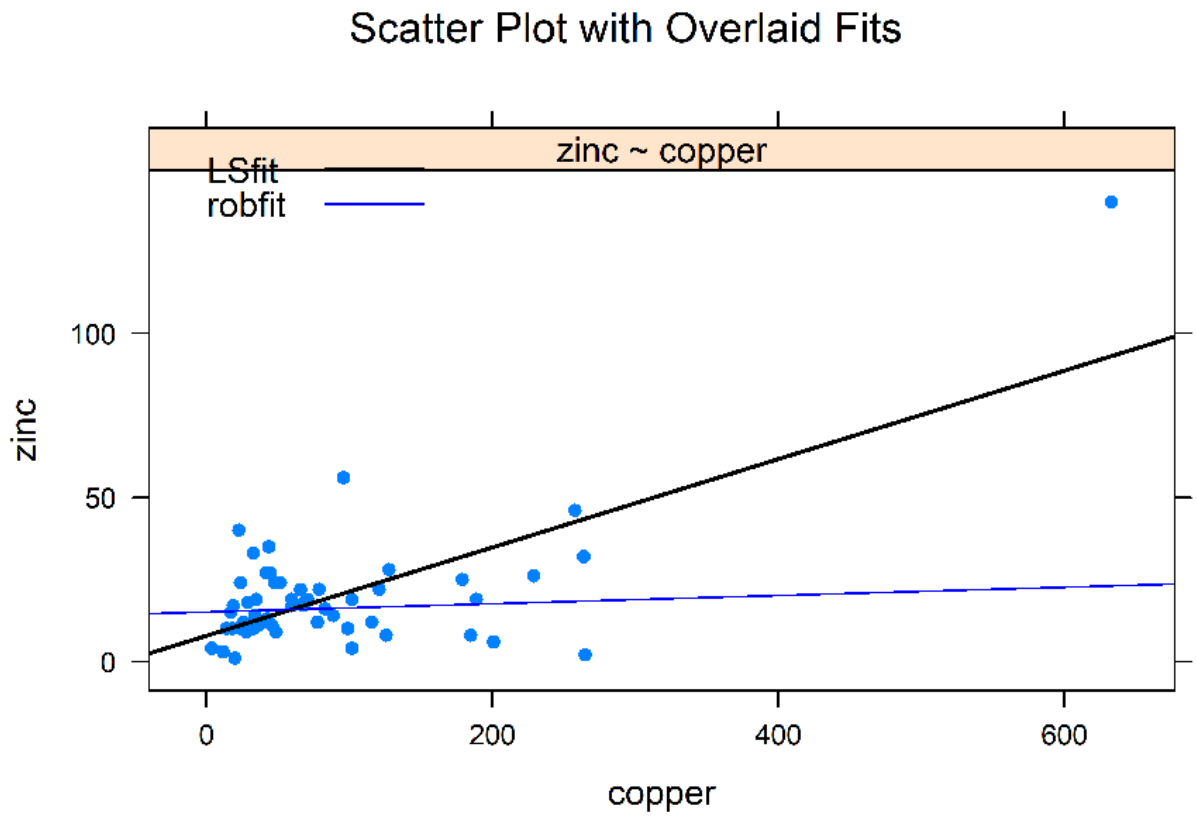

Figure 1: Scatter Plot of Zinc versus Copper with LS and Robust Line Fits

The following code will create the LS and Robust fits Normal QQ Plots of residuals shown in Figure 2.

```
plot(fmLSrob, which.plots = 2)
```

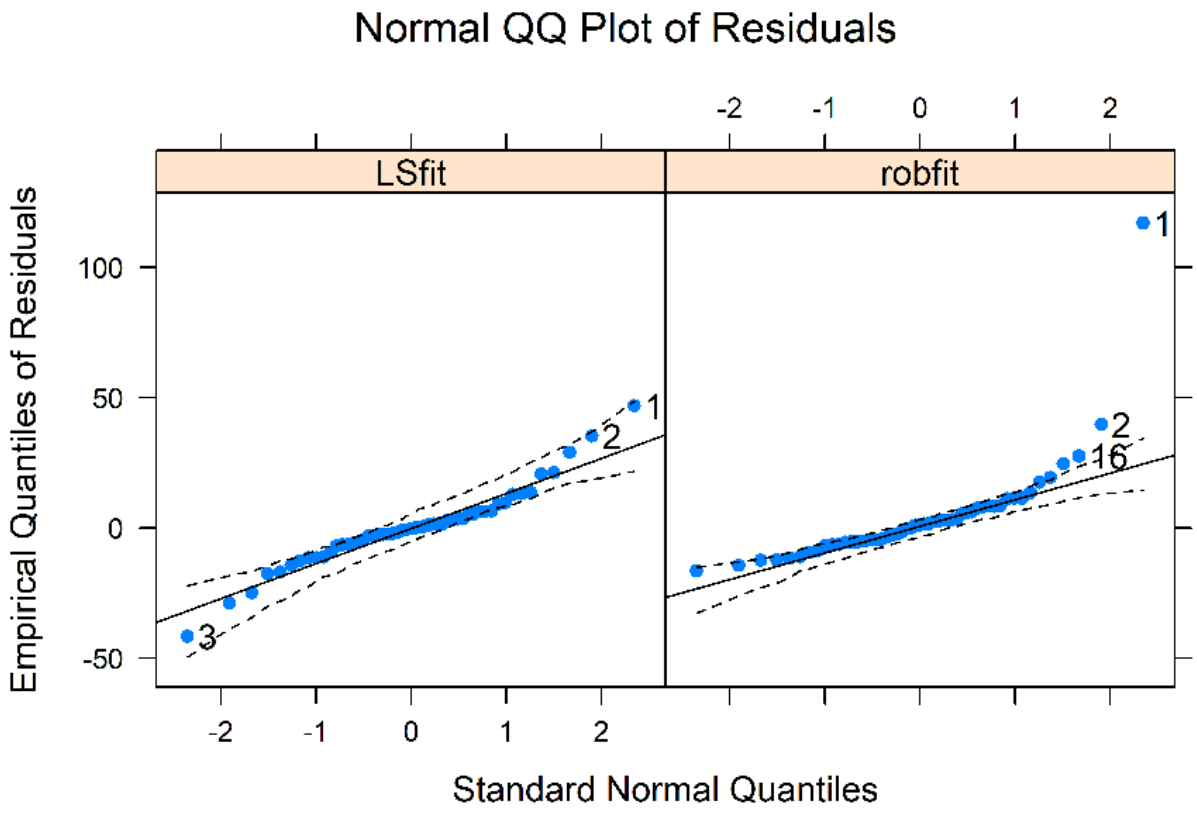

Figure 2: Normal QQ Plots of LS and Robust Fits Residuals

2.2 Robust versus Classical Covariance Matrices

You can also use `fit.models` function for comparing two different covariance matrix estimates, and we show below by example one way to do so for classical and robust covariance matrix estimates, where the classical covariance matrix is computed with the function `covClassic` in RobStatTM and the robust covariance matrix is computed with the function `covRob` in RobStatTM. We note that `covRob` by default automatically choose an "MM" type estimate if the the number of variables is less than 10, and computes a "Rocke" type estimate if the number of variables is 10 or greater. See `covRob` help file for more details, including how to set `type = "MM"` or `type = "Rocke"` in case you want to over-ride the automatic choice.[1]

The code below illustrate how to computed the separate classic and robust fits inside the `fit.models` function, rather than prior to use of `fit.models`. And since the number of variables is 5, the choice `"MM"` is automatically made. You see below that the class of `cov.fm` is `covfm`, and the summary method (`summary.covfm`) prints groups together the elements of the classic and robust covariance matrix elements, the location estimates, and the eigenvalues.

[1]It is also possible to use the functions `covRobMM` and `covRobRocke` directly by name. However, in the current release of `RobStatTM` the ellipses plot below does not work in that case.

```
library(robust)  # This is only needed until the package fit.models is updated in CRAN
```

```
data(wine)
wine5 <- wine[, 1:5]
cov.fm <- fit.models(Classic = covClassic(wine5),
    Robust = covRob(wine5, type = "auto"))
class(cov.fm)

## [1] "covfm"

summary(cov.fm)

##
## Calls:
## Classic : covClassic(data = wine5)
## Robust : covRob(data = wine5, type = "auto")
##
## Comparison of Covariance/Correlation Estimates:
##  (unique correlation terms)
##          [1,1]     [2,1]    [3,1]   [4,1]  [5,1]   [2,2]    [3,2]
## Classic 0.2136 -0.012891 -0.01560 -0.3746 0.7732 0.47410 0.004101
## Robust  0.1765  0.009822 -0.01338 -0.3102 1.9052 0.03802 0.014975
##           [4,2]  [5,2]   [3,3]  [4,3]  [5,3] [4,4] [5,4] [5,5]
## Classic 0.10525 0.5734 0.05160 0.3178 0.9124 6.484 6.372 110.2
## Robust  0.03642 0.3274 0.04217 0.1529 0.3141 4.467 1.593 144.7
##
## Comparison of Location Estimates:
##            V1    V2    V3    V4    V5
## Classic 13.74 2.011 2.456 17.04 106.3
## Robust  13.80 1.748 2.459 17.13 106.6
##
## Comparison of Eigenvalues:
##         Eval. 1 Eval. 2 Eval. 3 Eval. 4 Eval. 5
## Classic   110.6   6.136  0.4707 0.17810 0.03237
## Robust    144.8   4.480  0.1270 0.04986 0.02260
```

You can see the names of cov.fm object, and the names of the Classic and Robust components of the cov.fm object with the following code lines:

```
names(cov.fm)

## [1] "Classic" "Robust"

names(cov.fm$Classic)

## [1] "call"   "cov"    "center" "dist"   "corr"

names(cov.fm$Robust)

##  [1] "call"       "cov"        "center"     "dist"       "raw.cov"
##  [6] "raw.center" "raw.dist"   "corr"       "estim"      "control"
```

In order to see what plot methods are available for a cov.fm object, take a look at the help file with:

```
help(plot.covfm)
```

This shows that there are four types of plots, numbered 2, 3, 4, 5, that compare classic and robust co variance matrix properties as follows:

1. (not used)
2. Eigenvalues of Covariance Estimate
3. Sqrt of Mahalanobis Distances
4. Ellipses Matrix
5. Distance - Distance Plot

Figures 3, 4, 5, 6 below are produced by the code lines preceding them.

```
plot(cov.fm, which.plot = 2)
```

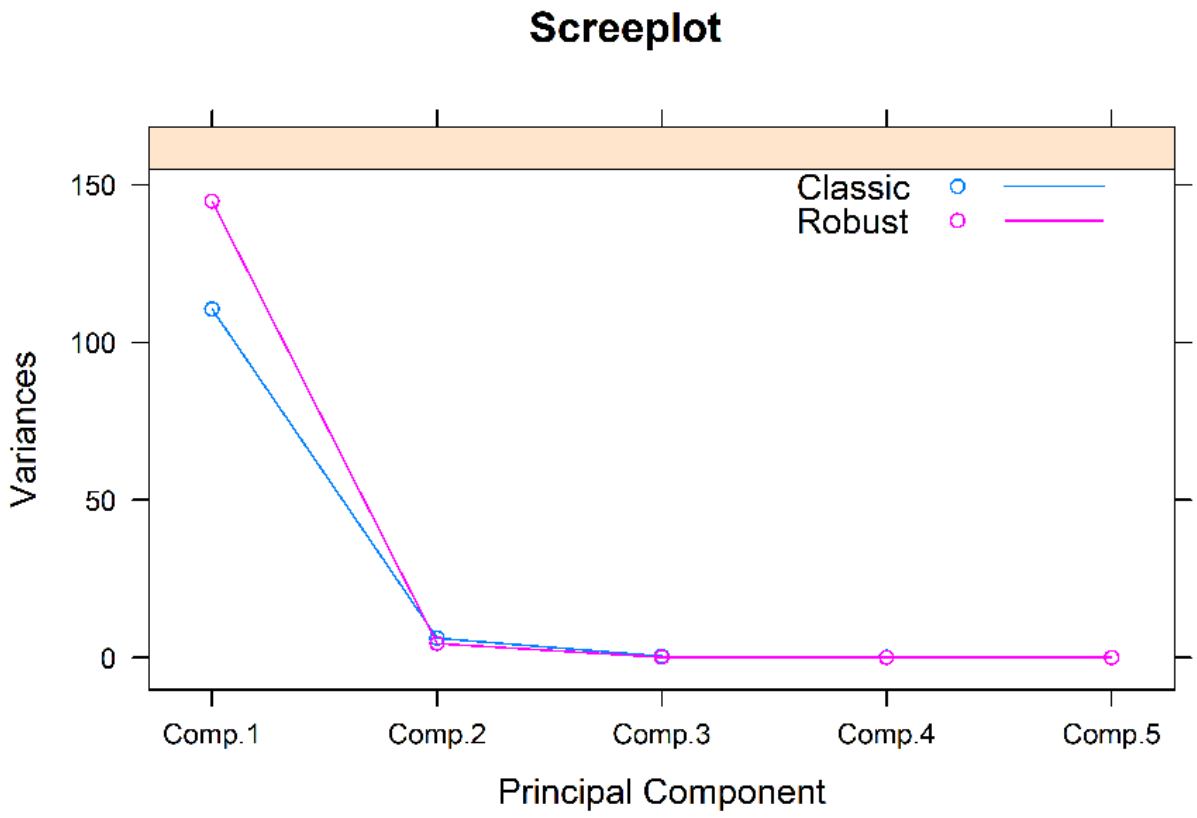

Figure 3: Classic Versus Robust Eigenvalues

```
plot(cov.fm, which.plot = 3)
```

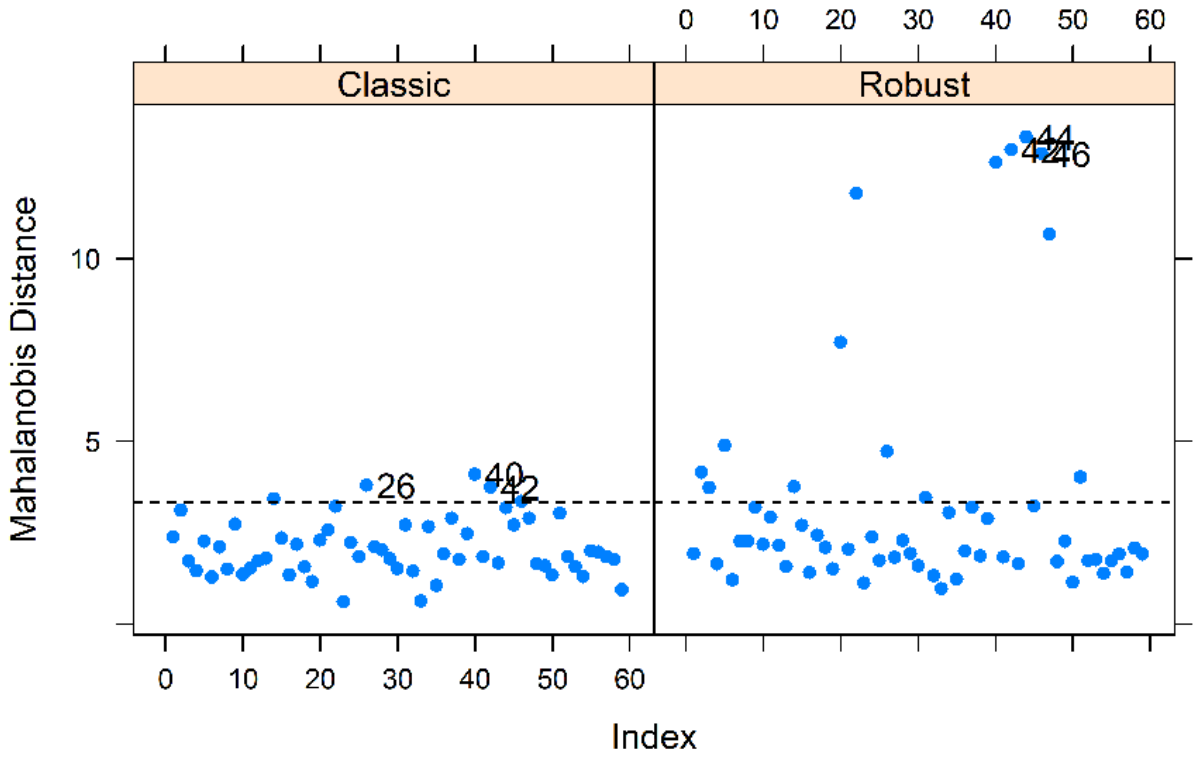

Figure 4: Classic versus Robust Mahalanobis Distances

```
plot(cov.fm, which.plot = 4)
```

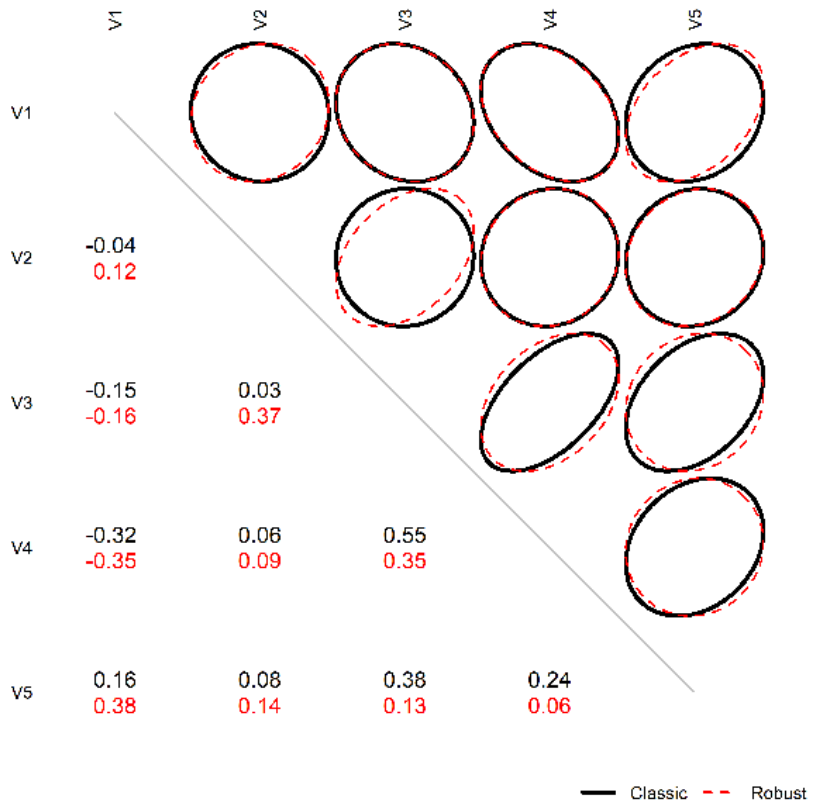

Figure 5: Robust versus Classic Correlations Ellipses

```
plot(cov.fm, which.plot = 5)
```

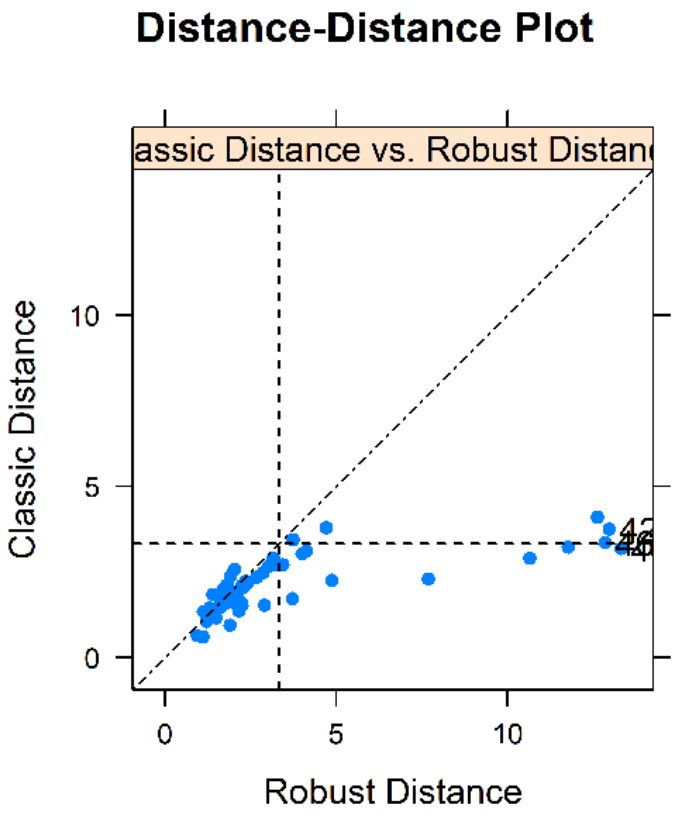

Figure 6: Robust versus Classic Mahalanobis Distances

3 Convergence of Algorithms for Computing Robust Estimates

RobStatTM contains sophisticated optimization algorithms for computing robust estimates, and convergence of the algorithms is determined by certain parameters. For example for the `lmrobM` and `lmrobdetMM` functions, the convergence parameters are set with the functions `lmrobM.control` and `lmrobdet.control`, respectively, which contain a number of parameters that have default settings. For example, `max.it` with default value `max.it = 100`, determines the maximum number of iterations of an iterated weighted least square (IRWLS) algorithm, and `rel.tol` with a default value `rel.tol = 1e-07`, determines a relative change threshold. See the book Section 4.5.2 for the algorithm used by `lmrobM`, including the relative change being measured, where step 4.3 uses for ϵ the value of `rel.tol`.

When a RobStatTM function gives a warning message, for example

```
M-step did NOT converge.  Returning unconverged lm-estimate
```

it does not mean that the estimate is necessarily bad, it is just that the algorithm has not converged given the default values of `rel.tol` and `max.it`, and the estimate is usually (but not always) quite viable. Our recommendation is that when such a warning occurs, one should adjust the parameter `rel.tol` downward,

and/or adjust the parameter `max.it` upward.