Contents

1 Note 1

2 Introduction 1
  2.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
    2.1.1 Example 1 . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
    2.1.2 Example 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  2.2 Data generating function . . . . . . . . . . . . . . . . . . . . . . 3
    2.2.1 Illustration of generated Data . . . . . . . . . . . . . . . . 3
  2.3 A very simple example . . . . . . . . . . . . . . . . . . . . . . . 4
  2.4 A simple example . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  2.5 Plotting a bit . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  2.6 Memory considerations . . . . . . . . . . . . . . . . . . . . . . . 14

3 Reproducing some results from BKY (2006) 16

1 Note

This SweaveFile is part of µToss package. But because some parts of this document are computational intensiv the extension of the Sweavefile is ".actuallyRnw".

2 Introduction

This document will give an introduction to the use of simTool. We will start with a very simple application then raise the degree of complexity in a few steps and in the end reproduce some of the results from Benjamini, Krieger, Yekutieli (2006). Basically for a simulation we need the following things

1. a function that generates the data (for example p-values)

2. some procedures that evaluates the generated data (for example the bonferroni correction)

3. statistics we want to calculate. For example the the false discovery proportion (FDP).
If we are speak for example of 1000 replications, we mean that these 3 steps were repeated 1000 times. That means, after 1000 replications there have been 1000 data sets generated. Every procedure was applied to these 1000 data sets. If we have specified for example 3 procedures then every of these procedures will give us an output, this means 3000 "output objects". And every specified statistic is applied to these 3000 "output objects".

2.1 Motivation

2.1.1 Example 1

Suppose we want to compare some characteristics of the two procedures BH and holm. Denote by \( V_n(BH) \) and \( V_n(holm) \) the number of true hypotheses rejected. For example we are interested in the distribution of \( V_n(BH) \) and \( V_n(holm) \) and also in \( E[V_n(BH)] \) and \( E[V_n(holm)] \). Lets have a look on the arguments of theses procedures:

\[
\texttt{args(BH)}
\]

function (pValues, alpha, silent = FALSE)

\[
\texttt{NULL}
\]

\[
\texttt{args(holm)}
\]

function (pValues, alpha, silent = FALSE)

\[
\texttt{NULL}
\]

Both have the argument \alpha, which stands for the niveau of the corresponding error measure. BH controls the false discovery rate (FDR) and holm controls the family-wise error rate (FWER). One question may be how must \alpha be set in holm such that for fixed \alpha = 0.1 in BH we have \( E[V_n(BH)] \approx E[V_n(holm)] \)? And perhaps we are also interested in how a dependence structure affects the distribution of \( V_n(BH) \) and \( V_n(holm) \).

2.1.2 Example 2

Suppose you have developed a new procedure controlling some error measure and you want to compare this new procedure with some other already established procedures. Then a simulation study will consist of creating data and gathering statistics for different sample sizes, dependence structures and parameter constellations.

Basically it is always the same story. Data is generated by a specific function, perhaps this depends on some parameters, and we want to apply different procedures, again depending on some paramters, to this generated data. Nearly everyone will say, "don’t bother me with the details of programming, just do the simulation with the above information and give me a nice data.frame which I can analyze." And this is exactly the purpose of the simTool.
2.2 Data generating function

For now we only want to use the procedures BH and holm. Again, let’s have a look on the arguments of these procedures:

\[
\texttt{args(BH)}
\]

\[
\text{function (pValues, alpha, silent = FALSE)}
\]

\[
\text{NULL}
\]

\[
\texttt{args(holm)}
\]

\[
\text{function (pValues, alpha, silent = FALSE)}
\]

\[
\text{NULL}
\]

Not much has to be specified. The parameter \texttt{alpha} is the level at which the corresponding error rate should be controlled and \texttt{pValues} are the \(p\)-values of the hypotheses that should be tested. In general, if we say that we reject a \(p\)-value this means that we reject the corresponding hypotheses. The following function generates data and will be used throughout the whole document. The data generating function must have an entry \$procInput. All in \$procInput will be used as input for the specified procedures. In our situation we will generate a list with 2 entries. \$procInput will only consist of the generated \(p\)-values. And \$groundTruth indicates which \(p\)-value corresponds to a true or false hypothesis.

\[
\texttt{pValFromEquiCorrData <- function(sampleSize, rho, mu, pi0) {}
+ \ nmbOfFalseHyp <- round(sampleSize * (1 - pi0))
+ \ nmbOfTrueHyp <- sampleSize - nmbOfFalseHyp
+ \ muVec <- c(rep(mu, nmbOfFalseHyp), rep(0, nmbOfTrueHyp))
+ \ Y <- sqrt(rho) * rnorm(1) + sqrt(1 - rho) * rnorm(sampleSize) +
+ \ muVec
+ \ return(list(procInput = list(pValues = 1 - pnorm(Y)), groundTruth = muVec ==
+ \ 0))
+ }}
\]

2.2.1 Illustration of generated Data

We will now generate 1000 \(p\)-Values independently (\(\rho = 0\)). \texttt{sampleSize * (1-pi0)} = 700 of them correspond to false hypotheses and 300 to true hypotheses.

\[
\texttt{set.seed(123)}
\]

\[
\texttt{data <- pValFromEquiCorrData(sampleSize = 1000, rho = 0, mu = 2,
+ \ pi0 = 0.3)}
\]

Let’s visualize the different \(p\)-values.

\[
\texttt{local({}
+ \ pValues = data$procInput$pValues
+ \ groundTruth = data$groundTruth
+ \ plot(ecdf(pValues), do.points = FALSE, verticals = TRUE,
+ \ main = "ecdf’s of generated \(p\)-values")
+ })
\]
2.3 A very simple example

Let’s directly start with the function call and then analysing what happened. Just for now we implement a simple version of the data generating function in section 2.2 and a simple version of the BH.

```r
> myGen <- function() {
+   pValFromEquiCorrData(sampleSize = 200, rho = 0, mu = 2, pi0 = 0.5)
+ }
> BH.05 = function(pValues) {
+   BH(pValues = pValues, alpha = 0.05, silent = TRUE)
+ }
> set.seed(123)
> sim <- simulation(replications = 10, list(funName = "myGen",
+   fun = myGen), list(list(funName = "BH.simple", fun = BH.05)))
```

The following happened:
1. Call \texttt{myGen}

2. Append generated data set to \texttt{sim$data}

3. Call \texttt{BH.05} with the generated \textit{p}-values

4. Add to the results of \texttt{BH.05} the parameter constellation used by \texttt{myGen} and \texttt{BH.05} and also the position of the used data in \texttt{sim$data}

5. Append this extended results of \texttt{BH.05} to \texttt{sim$results}

6. repeat this 9 more times

Since \texttt{replications = 10} in simulation the object \texttt{sim} consists of:

\begin{verbatim}
> names(sim)
[1] "data"   "results"
> length(sim$data)
[1] 10
> length(sim$results)
[1] 10
\end{verbatim}

The structure of one object in \texttt{sim$data} coincides with the structure of the return of \texttt{myGen}:

\begin{verbatim}
> names(myGen())
[1] "procInput" "groundTruth"
> names(sim$data[[1]])
[1] "procInput" "groundTruth"
\end{verbatim}

Let's have a look at the additional information added by simulation to the object returned by \texttt{BH.05}. First the entries of the pure return value of \texttt{BH.05} and then of \texttt{sim$results}

\begin{verbatim}
> names(BH.05(runif(100)))
[1] "adjPValues" "criticalValues" "rejected" "errorControl"
> names(sim$results[[1]])
[1] "data.set.number" "parameters" "adjPValues" "criticalValues"
[5] "rejected" "errorControl"
\end{verbatim}

data.set.number is the number of the used data set in \texttt{sim$data}. Since every generated data set is used only once we have

\begin{verbatim}
> sapply(sim$results, function(x) x$data.set.number)
[1] 1 2 3 4 5 6 7 8 9 10
\end{verbatim}
Thus it is possible to reproduce any result directly. Let us reproduce the results of the 6th replication

```r
> idx <- sim$results[[6]]$data.set.number
> pValues <- sim$data[[idx]]$procInput$pValues
> all(BH.05(pValues)$adjPValues == sim$results[[6]]$adjPValues)

[1] TRUE
```

Since our data generating function and procedure did not really have any parameter, there is not much information in:

```r
> sim$results[[1]]$parameters
```

```r
$funName
[1] "myGen"

$method
[1] "BH.simple"
```

Note, BH.05 technically has the parameter `pValues` but since this parameter is the "input channel" for the output data of the data generating function it is not really regarded as parameter that affect the behaviour of the procedure like for example the level $\alpha$ for the error measure.

### 2.4 A simple example

Indeed, for the simple example above the simTool is not very helpful. Let us increase the complexity of our simulation. As a first step, we want to increase the parameter $\rho$ of the data generating function gradually and investigate $EV_n(BH)$ and $EV_n^2(BH)$.

```r
> set.seed(123)
> sim <- simulation(replications = 10, list(funName = "pVGen",
+ fun = pValFromEquiCorrData, sampleSize = 100, rho = seq(0,
+ 1, by = 0.2), mu = 2, pi0 = 0.5), list(list(funName = "BH",
+ fun = BH, alpha = c(0.5, 0.25), silent = TRUE)))
```

The following happened:

1. Call `myGen` with $\rho = 0$
2. Append generated data set to `sim$data`
3. Call BH with $\alpha = 0.5$ and the generated $p$-values
4. Add to the results the parameter constellation used by `myGen` and BH and also the position of the used data in `sim$data`
5. Append this extended results to `sim$results`
6. Call BH with $\alpha = 0.25$ and the same $p$-values as in 3.
7. Add to the results the parameter constellation used by `myGen` and BH and also the position of the used data in `sim$data`
8. Append this extended results of to `sim$results`
9. repeat this 9 more times
10. Call `myGen` with `rho = 0.2`
11. and so on

So we have now a bunch of data sets and results.

```r
> length(sim$data)
[1] 60
> length(sim$results)
[1] 120
```

60 data sets have been generated because we have `rho = 0, 0.2, 0.4, 0.6, 0.8, 1` and for each `rho` we generated 10 data sets. We have applied BH with `alpha = 0.5` to all 60 data sets and BH with `alpha = 0.25`, yielding 120 objects in `results`. We see from the `data.set.number` that after generating a data set it is used two times in series

```r
> sapply(sim$results, function(x) x$data.set.number)
[1] 1 1 2 2 3 3 4 4 5 5 6 6 7 7 8 8 9 9 10 10 11 11 12 12 13 13 14 14 15 15 16 16
[51] 30 30 31 31 32 32 33 33 34 34 35 35 36 36 37 37 38 38 39 39 40 40 41 41 42 42
[84] 43 43 44 44 45 45 46 46 47 47 48 48 49 49 50 50
```

In order to gather how many true hypotheses have been rejected corresponding to the different parameter constellations we need a function that is able to calculate it.

```r
> NumberOfType1Error <- function(data, result) sum(data$groundTruth * + result$rejected)
> V2 <- function(data, result) NumberOfType1Error(data, result)^2

Let us calculate for the first result object the number of rejected true hypotheses explicitly.

```r
> idx = sim$results[[1]]$data.set.number
> data = sim$data[[idx]]
> NumberOfType1Error(data, sim$results[[1]])
[1] 15

This means for the following parameter constellation

```r
> sim$results[[1]]$parameters
```
we one time observed
[1] 15
rejected true null hypotheses. In order to estimate $E V_n(BH)$ and $E V^2_n(BH)$ we have to search in sim$results for the other 9 results using the same parameter constellation. In order to facilitate this task we provide a function.

```r
> result <- gatherStatistics(sim, list(V = NumberOfType1Error, +   V2 = V2), mean)
```

```r
> result
```

```r
$statisticDF
funName sampleSize rho mu pi0 method alpha silent V.mean V2.mean
1 pVGen 100 0 2 0.5 BH 0.5 TRUE 15.1 245.3
2 pVGen 100 0 2 0.5 BH 0.25 TRUE 5.9 40.9
3 pVGen 100 0.2 2 0.5 BH 0.5 TRUE 21.5 591.9
4 pVGen 100 0.2 2 0.5 BH 0.25 TRUE 9.2 163.2
5 pVGen 100 0.4 2 0.5 BH 0.5 TRUE 14.6 401.6
6 pVGen 100 0.4 2 0.5 BH 0.25 TRUE 6.2 126.6
7 pVGen 100 0.6 2 0.5 BH 0.5 TRUE 22.3 860.7
8 pVGen 100 0.6 2 0.5 BH 0.25 TRUE 9.3 265.7
9 pVGen 100 0.8 2 0.5 BH 0.5 TRUE 18.1 711.1
10 pVGen 100 0.8 2 0.5 BH 0.25 TRUE 8.0 328.6
11 pVGen 100 1 2 0.5 BH 0.5 TRUE 20.0 1000.0
12 pVGen 100 1 2 0.5 BH 0.25 TRUE 15.0 750.0
```
As you can see every parameter constellation has its own row in `$statisticDF`. By the way, again, we see that *p*-values is a parameter of BH but since it is contained in `$procInput` it is not considered as "real parameter". If we are interested in more than one statistics, then we simply provide a list with "average functions":

```r
> result <- gatherStatistics(sim, list(V = NumberOfType1Error, + V2 = V2), list(mean = mean, sd = function(vec) round(sd(vec), + 1)))
> result$statisticDF

<table>
<thead>
<tr>
<th>funName</th>
<th>sampleSize</th>
<th>rho</th>
<th>mu</th>
<th>pi0</th>
<th>method</th>
<th>alpha</th>
<th>silent</th>
<th>V.mean</th>
<th>V2.mean</th>
<th>V.sd</th>
<th>V2.sd</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>15.1</td>
<td>245.3</td>
<td>4.4</td>
<td>131.7</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.2</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>21.5</td>
<td>591.9</td>
<td>12.0</td>
<td>597.3</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>9.2</td>
<td>163.2</td>
<td>9.3</td>
<td>329.0</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>14.6</td>
<td>401.6</td>
<td>14.5</td>
<td>684.8</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>6.2</td>
<td>126.6</td>
<td>9.9</td>
<td>318.5</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>22.3</td>
<td>860.7</td>
<td>20.1</td>
<td>981.2</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.6</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>9.3</td>
<td>265.7</td>
<td>9.9</td>
<td>318.5</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.6</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>18.1</td>
<td>711.1</td>
<td>20.6</td>
<td>1051.5</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.8</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>8.0</td>
<td>328.6</td>
<td>17.1</td>
<td>801.7</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>20.0</td>
<td>1000.0</td>
<td>25.8</td>
<td>1291.0</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>15.0</td>
<td>750.0</td>
<td>24.2</td>
<td>1207.6</td>
</tr>
</tbody>
</table>

Another possibility is to call `gatherStatistics` without and "average function". Then every result get his own row in `$statisticDF`

```r
> result <- gatherStatistics(sim, list(V = NumberOfType1Error, + V2 = V2))
> head(result$statisticDF)

<table>
<thead>
<tr>
<th>funName</th>
<th>sampleSize</th>
<th>rho</th>
<th>mu</th>
<th>pi0</th>
<th>method</th>
<th>alpha</th>
<th>silent</th>
<th>V</th>
<th>V2</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>15</td>
<td>225</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>20</td>
<td>400</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>11</td>
<td>121</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>19</td>
<td>361</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>16</td>
<td>256</td>
</tr>
</tbody>
</table>
```
funName sampleSize rho mu pi0 method alpha silent V V2
115   pVGen 100 1 2 0.5 BH 0.25 TRUE 0 0
116   pVGen 100 1 2 0.5 BH 0.25 TRUE 50 2500
117   pVGen 100 1 2 0.5 BH 0.25 TRUE 50 2500
118   pVGen 100 1 2 0.5 BH 0.25 TRUE 50 2500
119   pVGen 100 1 2 0.5 BH 0.25 TRUE 0 0
120   pVGen 100 1 2 0.5 BH 0.25 TRUE 0 0

2.5 Plotting a bit
> if (!is.element("sim.plot", ls())) {
  + sim.plot <- simulation(replications = 1000, list(funName = "pVGen",
  +   fun = pValFromEquiCorrData, sampleSize = 100, rho = seq(0,
  +   1, by = 0.2), mu = 0, pi0 = 1), list(list(funName = "BH",
  +   fun = function(pValues, alpha) BH(pValues, alpha, silent = TRUE),
  +   alpha = c(0.5, 0.75))))
  + }
> length(sim.plot$data)
[1] 6000

> length(sim.plot$results)
[1] 12000

The are already different kind of R functions that take data.frames and generate
histograms, boxplots and so on from them. For some plot example we will need
the lattice package.

> require(lattice)

First we calculate V for every MutossSim object and then plot a histogram and
a boxplot of V, see Figure 1 (p. 11), Figure 2 (p. 12).

> result.all <- gatherStatistics(sim.plot, list(V = NumberOfType1Error))

Also after the "average process" we again get an data.frame which can be used
to generate plots.

> result <- gatherStatistics(sim.plot, list(V = NumberOfType1Error),
+   list(Ml = function(x) mean(x) - 2 * sd(x)/sqrt(length(x)),
+     Mu = function(x) mean(x) + 2 * sd(x)/sqrt(length(x)),
+     M = mean, SD = sd))
> subset(result$statisticDF, alpha == "0.5")[1:3, -1]
sampleSize rho mu pi0 method alpha V.Ml V.Mu V.M V.SD
 1 100  0  0  1 BH 0.5 1.610993 1.973007 1.792 2.861980
 3 100 0.2  0  1 BH 0.5 8.833764 11.544236 10.189 21.428169
 5 100 0.4  0  1 BH 0.5 12.131991 15.860009 13.996 29.472574
> print(histogram(~V | rho, data = subset(result.all$statisticDF, + alpha == "0.5")))

Figure 1: Histogram of the number of rejected true hypotheses for alpha equals 0.5
> print(bwplot(V ~ rho | alpha, data = subset(result.all$statisticDF)))

Figure 2: Boxplot of the number of rejected true hypotheses
> print(xyplot(V.MI + V.M + V.Mu ~ rho | alpha, data = result$statisticDF,  
+     type = "a", auto.key = list(space = "right", points = FALSE,  
+     lines = TRUE)))

Figure 3: XYPlot of the mean number of rejected true hypotheses with asymptotic 95% confidence intervals (pointwise)
2.6 Memory considerations

Up to now we have kept any information in memory. Sometimes, it is useful not to discard any generated information. For example, if some objects in results appear odd, they can directly be reproduced and of course with all information available we can investigate anything like distribution or any statistic we did not consider before the simulation. The price for this liberty is in general an extensive memory usage which of course restrict the size of our simulation. On the other hand if we exactly know that we are only interested in \( E_V \) the mean number of true rejected hypotheses, why should we keep the generated p-values or the index of the rejected p-values? Thus it would be enough that an object in results contains \( V_n \) the number of rejected true hypotheses. We will now show an example were only the information needed is kept in memory. The following is basically the "simple example" from section 2.4 The main issue is to save memory thus we will calculate the \( V_n \) right after applying the procedure to the generated data set. If done so, we can also discard the generated data!

```r
> V.BH <- function(pValues, groundTruth, alpha) {
  + out <- BH(pValues, alpha, silent = TRUE)
  + V <- sum(out$rejected * groundTruth)
  + return(list(V = V))
+ }
```  
As already mentioned anything in $procInput will be used as an input for the procedures. Since our procedure now has a parameter groundTruth our data generated function has to be adapted. We just move groundTruth in $procInput.

```r
> pValFromEquiCorrData2 <- function(sampleSize, rho, mu, pi0) {
+   nmbOfFalseHyp <- round(sampleSize * (1 - pi0))
+   nmbOfTrueHyp <- sampleSize - nmbOfFalseHyp
+   muVec <- c(rep(mu, nmbOfFalseHyp), rep(0, nmbOfTrueHyp))
+   Y <- sqrt(rho) * rnorm(1) + sqrt(1 - rho) * rnorm(sampleSize) +
+     muVec
+   return(list(procInput = list(pValues = 1 - pnorm(Y), groundTruth = muVec ==
+     0)))
+ }
```  
Obviously, it is unnecessary to store this generated data, because the number of Type 1 Errors we are interested in will be directly calculated. Setting discardProcInput = TRUE in simulation removes $procInput from the object generated by the data generating function after $procInput has been used by all specified procedures. In our case these will be the 2 procedures V.BH with alpha = 0.5 and alpha = 0.25.

```r
> set.seed(123)
> sim <- simulation(replications = 10, list(funName = "pVGen2",
+   fun = pValFromEquiCorrData2, sampleSize = 100, rho = seq(0,
+     1, by = 0.2), mu = 2, pi0 = 0.5), list(list(funName = "V.BH",
+     fun = V.BH, alpha = c(0.5, 0.25)), discardProcInput = TRUE)
> result <- gatherStatistics(sim, list(V = function(data, results) results$V),
+   mean)
> result
```
Here the corresponding table from section 2.4.

```r
> set.seed(123)
> sim.simple <- simulation(replications = 10, list(funName = "pVGen",
+   fun = pValFromEquiCorrData, sampleSize = 100, rho = seq(0,
+   1, by = 0.2), mu = 2, pi0 = 0.5), list(list(funName = "BH",
+   fun = BH, alpha = c(0.5, 0.25), silent = TRUE)))
> result.simple <- gatherStatistics(sim.simple, list(V = NumberOfType1Error),
+   mean)
> print(result.simple)
```

<table>
<thead>
<tr>
<th>funName</th>
<th>sampleSize</th>
<th>rho</th>
<th>mu</th>
<th>pi0</th>
<th>method</th>
<th>alpha</th>
<th>silent</th>
<th>V.mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>15.1</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>5.9</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.2</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>21.5</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.2</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>9.2</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>14.6</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.4</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>6.2</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.6</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>22.3</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>0.6</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>9.3</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.5</td>
<td>TRUE</td>
<td>20.0</td>
</tr>
<tr>
<td>pVGen</td>
<td>100</td>
<td>1</td>
<td>2</td>
<td>0.5</td>
<td>V.BH</td>
<td>0.25</td>
<td>TRUE</td>
<td>15.0</td>
</tr>
</tbody>
</table>
$name.parameters
[1] "funName"  "sampleSize"  "rho"     "mu"     "pi0"
[6] "method"   "alpha"     "silent"

$name.statistics
[1] "V.mean"

$name.avgFun
[1] "mean"

The same results but the memory usage differ much:

```r
> print(s1 <- object.size(sim))
127192 bytes
> print(s2 <- object.size(sim.simple))
565912 bytes
> unclass(s2/s1)
[1] 4.449274
```

3 Reproducing some results from BKY (2006)

Now, we will reproduce figure 1 from the publication. For this we need to estimate the FDR. This will be done by calculating the FDP for every object in $results and then calculating the empirical mean. In this simulation where we exactly know what we want and thus retain only the necessary information. To be able to calculate the FDP right after applying the procedure we need to know which pValue belongs to a true or false hypothesis. Thus, like in the foregoing section, $groundTruth will be an element of $procInput.

```r
> pValFromEquiCorrData2 <- function(sampleSize, rho, mu, pi0) {
+   nmbOfFalseHyp <- round(sampleSize * (1 - pi0))
+   nmbOfTrueHyp <- sampleSize - nmbOfFalseHyp
+   muVec <- c(rep(mu, nmbOfFalseHyp), rep(0, nmbOfTrueHyp))
+   Y <- sqrt(rho) * rnorm(1) + sqrt(1 - rho) * rnorm(sampleSize) +
+     muVec
+   return(list(procInput = list(pValues = 1 - pnorm(Y),
+                                groundTruth = muVec ==
+                                0))))
+
```

We also need a function that calculates the FDP.

```r
> FDP <- function(pValues, groundTruth, proc = c("BH", "M-S-HLF")) {
+   if (proc == "BH")
+     out <- BH(pValues, alpha = 0.05, silent = TRUE)
+   else out <- adaptiveSTS(pValues, alpha = 0.05, silent = TRUE)
+   R <- sum(out$rejected)
+   if (R == 0)
```
Now, the (partial) reproduction of the results can start!

```r
> set.seed(123)
> date()

[1] "Thu Dec 01 18:05:29 2011"

> sim <- simulation(replications = 10000, list(funName = "pVGen2",
+       fun = pValFromEquiCorrData2, sampleSize = c(16, 32, 64, 128,
+       256, 512), rho = c(0, 0.1, 0.5), mu = 5, pi0 = c(0.25,
+       0.75)), list(list(funName = "FDP", fun = FDP, proc = c("BH",
+       "M-S-HLF"))), discardProcInput = TRUE)
> date()

[1] "Thu Dec 01 18:39:30 2011"

> result <- gatherStatistics(sim, list(FDP = function(data, results) results$FDP),
+       mean) > result

$statisticDF

<table>
<thead>
<tr>
<th>funName</th>
<th>sampleSize</th>
<th>rho</th>
<th>mu</th>
<th>pi0</th>
<th>method</th>
<th>proc</th>
<th>FDP.mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01243507</td>
</tr>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.04680482</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>32</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01254447</td>
</tr>
<tr>
<td>pVGen2</td>
<td>32</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.04933171</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>64</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01245073</td>
</tr>
<tr>
<td>pVGen2</td>
<td>64</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05027779</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>128</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01250933</td>
</tr>
<tr>
<td>pVGen2</td>
<td>128</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05000333</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>256</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01259937</td>
</tr>
<tr>
<td>pVGen2</td>
<td>256</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05021692</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>512</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01246457</td>
</tr>
<tr>
<td>pVGen2</td>
<td>512</td>
<td>0</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.04993815</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01243852</td>
</tr>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.04929676</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>32</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01290907</td>
</tr>
<tr>
<td>pVGen2</td>
<td>32</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05355816</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>64</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01271804</td>
</tr>
<tr>
<td>pVGen2</td>
<td>64</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05626510</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>128</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01233749</td>
</tr>
<tr>
<td>pVGen2</td>
<td>128</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05637963</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>256</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01231818</td>
</tr>
<tr>
<td>pVGen2</td>
<td>256</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05619898</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>512</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01248149</td>
</tr>
<tr>
<td>pVGen2</td>
<td>512</td>
<td>0.1</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.05719747</td>
<td></td>
</tr>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
<td>FDP</td>
<td>BH</td>
<td>0.01269308</td>
</tr>
<tr>
<td>pVGen2</td>
<td>16</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
<td>FDP M-S-HLF</td>
<td>0.06256181</td>
<td></td>
</tr>
</tbody>
</table>

17
<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>pVGen2</td>
<td>32</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>28</td>
<td>pVGen2</td>
<td>32</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>29</td>
<td>pVGen2</td>
<td>64</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>30</td>
<td>pVGen2</td>
<td>64</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>31</td>
<td>pVGen2</td>
<td>128</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>32</td>
<td>pVGen2</td>
<td>128</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>33</td>
<td>pVGen2</td>
<td>256</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>34</td>
<td>pVGen2</td>
<td>256</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>35</td>
<td>pVGen2</td>
<td>512</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>36</td>
<td>pVGen2</td>
<td>512</td>
<td>0.5</td>
<td>5</td>
<td>0.25</td>
</tr>
<tr>
<td>37</td>
<td>pVGen2</td>
<td>16</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>38</td>
<td>pVGen2</td>
<td>16</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>39</td>
<td>pVGen2</td>
<td>32</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>40</td>
<td>pVGen2</td>
<td>32</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>41</td>
<td>pVGen2</td>
<td>64</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>42</td>
<td>pVGen2</td>
<td>64</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>43</td>
<td>pVGen2</td>
<td>128</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>44</td>
<td>pVGen2</td>
<td>128</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>45</td>
<td>pVGen2</td>
<td>256</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>46</td>
<td>pVGen2</td>
<td>256</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>47</td>
<td>pVGen2</td>
<td>512</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>48</td>
<td>pVGen2</td>
<td>512</td>
<td>0</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>49</td>
<td>pVGen2</td>
<td>16</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>50</td>
<td>pVGen2</td>
<td>16</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>51</td>
<td>pVGen2</td>
<td>32</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>52</td>
<td>pVGen2</td>
<td>32</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>53</td>
<td>pVGen2</td>
<td>64</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>54</td>
<td>pVGen2</td>
<td>64</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>55</td>
<td>pVGen2</td>
<td>128</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>56</td>
<td>pVGen2</td>
<td>128</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>57</td>
<td>pVGen2</td>
<td>256</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>58</td>
<td>pVGen2</td>
<td>256</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>59</td>
<td>pVGen2</td>
<td>512</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>60</td>
<td>pVGen2</td>
<td>512</td>
<td>0.1</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>61</td>
<td>pVGen2</td>
<td>16</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>62</td>
<td>pVGen2</td>
<td>16</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>63</td>
<td>pVGen2</td>
<td>32</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>64</td>
<td>pVGen2</td>
<td>32</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>65</td>
<td>pVGen2</td>
<td>64</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>66</td>
<td>pVGen2</td>
<td>64</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>67</td>
<td>pVGen2</td>
<td>128</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>68</td>
<td>pVGen2</td>
<td>128</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>69</td>
<td>pVGen2</td>
<td>256</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>70</td>
<td>pVGen2</td>
<td>256</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>71</td>
<td>pVGen2</td>
<td>512</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
<tr>
<td>72</td>
<td>pVGen2</td>
<td>512</td>
<td>0.5</td>
<td>5</td>
<td>0.75</td>
</tr>
</tbody>
</table>

$name.parameters
[1] "funName"  "sampleSize"  "rho"  "mu"  "pi0"
[6] "method"  "proc"
$name.statistics
[1] "FDP.mean"

$name.avgFun
[1] "mean"

> print(xyplot(FDP.mean ~ sampleSize | pi0 * rho, data = result$statisticDF,
+     group = proc, type = "a", auto.key = list(space = "right",
+     points = FALSE, lines = TRUE)))