A Short Tutorial on Experimental Analysis of Optimization Heuristics with R

Marco Chiarandini

October 2, 2008

Abstract

This is a short tutorial on R for the specific task of analyzing the results of computational experiments on heuristic algorithms for optimization. In the first two sections a short introduction to statistical programming with R is provided. In Section 3, the least median regression problem and optimization heuristics for its solution are presented. This section serves to create a running example and it can be skipped if results are already available on another problem. Sections 4 and 5 show how to carry out a comparison among few different algorithms. Both the single instance and the multi-instance scenarios are considered.

This is a very preliminary version.

1 R Download and Installation

- R web page (`http://www.r-project.org/`), select Download - CRAN → Select a mirror → Select in the first frame, named 'Download and Install R', your operating system.
- Windows: → select 'base' → click on 'R-?.?.?-win32.exe'.
- Under Linux Ubuntu:

  ```
  sudo apt-get install r-base
  ```

- Alternatively, go to `http://cran.r-project.org/doc/manuals/R-admin.html` and follow description there.
- A package is a collection of functions and programs that can be used within R. To install new packages type from R command line:

  ```
  > install.packages("lattice")
  ```

- Package `Rcmdr` provides a graphical interface to R.
- R documentation: see `http://www.sbtc.ltd.uk/freenotes.html`: 'Getting Started in R' by Saghir Bashir, and the references under the R link `http://cran.r-project.org/manuals.html`.
- Emacs users can find a mode for R at `http://ess.r-project.org/`.

2 Basic commands

2.1 System commands

- `R` starts R from command line.
- `library(Rcmdr)` loads the `Rcmdr` package and starts graphical interface.
- `q()` quits your R session.
- `options(width=120)` determines the position of the line break in R output.
- `?plot` a question mark followed by the name of the function opens the help page relative to that function.
- `help.start()` opens a browser with documentation.
- `example(plot)` calls one or more examples implemented for the function.
- `demo(package.name)` calls a demonstration for the functionality of the defined package.
- `ls()` provides a list of objects in the current R workspace.

2.2 Data and operations

- `read.table()` and `write.table()` read and write from file. See help page for details.
- `^`, `%%`, `%/%` operators for power, modulus and integer part of the division, respectively.
- `vector()`, `matrix()`, `array()`, `data.frame()`, `list()` data structures. Tests or coercions can be done with `is.data.frame()`, `as.data.frame()`, respectively
- `c()` function used to collect things together into a vector, example: `x <- c(1,2,3)`
- `str()` compactly displays the structure of an arbitrary R object
- `integer()` `double()` data types. Can be queried or coerced with `is.integer()` and `as.integer()`
- `mean()`. `median()`, `sum()`, `var()`, `summary()`, `interquartile()` `range()` compute sample statistics.
- `factor()` offers an alternative way of storing character data. For example a *factor* can have four elements and two *levels*:

```
> algorithms <- c("greedy", "grasp", "greedy", "grasp")
> algorithms

[1] "greedy" "grasp"  "greedy" "grasp"

> algorithms <- factor(algorithms)
> algorithms
```

```
[1] greedy grasp  greedy grasp
Levels: grasp greedy
```

- `1:12`, `seq(1, 21, by=2)`, `rep(3,12)` generate sequences of integers.
- `expand.grid()` creates a data frame from all combinations of factors

```
> expand.grid(algorithm = algorithms, instance = c("A", "B"))

  algorithm instance
1    greedy        A
2     grasp        A
3    greedy        A
4     grasp        A
5    greedy        B
6     grasp        B
7    greedy        B
8     grasp        B
```

- `paste()`, `substr()` work with strings. The first concatenates the second returns substrings within two positions. Example:

```
> colors <- c("red", "yellow", "green")
> paste(colors, "flowers")

[1] "red flowers"    "yellow flowers" "green flowers"

> paste("several ", colors, "s", sep = "")

[1] "several reds"    "several yellows" "several greens"

> paste("I like", colors, collapse = ", ")

[1] "I like red, I like yellow, I like green"

> substr(colors, 1, 2)

[1] "re" "ye" "gr"
```

2.3 Graphics

- `plot()`, `lines()`, `points()`, `curve()`, `hist()`, `barplot()`, `boxplot()`, graphics functions from the base installation
- `par()` lists and changes graphic setting
- `colors()` the colors available in R
- Graphics are device independent. Type `?device` to see on which device they can be printed and how.

- `dev.copy(dev=pdf,file='Rplot.pdf')` copies the graphic in a pdf file. Remember to close the pipeline with `dev.off()`
- `lattice` and `ggplot` two packages for multivariate conditional plots. Try `demo()` on them for a demonstration of their facilities. The package `lattice` is thoroughly explained in reference [3].

3 An optimization example

Let's consider a basic linear regression model:

$$Y_i = \beta_0 + \beta_1 X_i + \epsilon \qquad i = 1, \ldots, n$$

where X_i is the value of the predictor variable in the ith trial, Y_i is the value of the response variable in the ith trial, β_0 and β_1 are parameters and ϵ_i are mutually independent random errors with $E[\epsilon_i] = 0$ and $\sigma[\epsilon_i] = 0$. We can rewrite in matrix notation:

$$\boldsymbol{Y} = \begin{bmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{bmatrix} \qquad \boldsymbol{X} = \begin{bmatrix} 1 & X_1 \\ 1 & X_2 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \qquad \boldsymbol{\beta} = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} \qquad \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{bmatrix}$$

The $\mathbf{X}$ is often referred to as the *design matrix*. Hence,

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}$$

We simulate these data in R. The function `rexp(N,l)` generates `N` following an exponential distribution with mean `l`.

```
> set.seed(1)
> N <- 100
> X <- array(dim = c(N, 2))
> X[, 1] <- 1
> X[, 2] <- seq(0.01, 1, by = 0.1)
> Y <- X %*% matrix(c(0, 1)) + matrix(rexp(N, 1))
```

The usual way to estimate the values of the parameters $\boldsymbol{\beta}$ is by means of the least square method that minimizes:

$$\min \sum_{i=1}^{n} (Y_i - \beta_0 - \beta_1 X_i)^2$$

The estimators β_0 and β_1 can be found by analytical procedure and in R using the method `lm`.

```
> l <- lm(Y ~ X[, 2])
> plot(X[, 2], Y)
> abline(l)
```

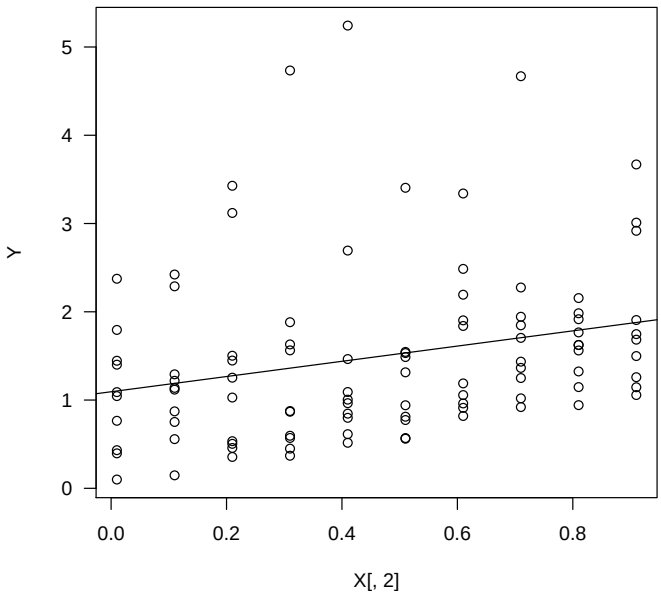

Figure 1: Scatter plot and linear regression line superimposed.

Here, we want instead to use the *least median of squares method*, that is,

$$z(\boldsymbol{\beta}) = \min\left\{ \text{median}[(Y_i - \beta_0 - \beta_1 X_i)^2]\right\}$$

The least median method should be more robust against outliers in the model, however the estimation of the parameters β_0 and β_1 requires minimizing a non-differentiable, non-linear, multi-modal function for which an analytical procedure is not known. We can inspect this visually using R methods for 3D plots.

First, we declare a function that implements the function we want to minimize, i.e., median$[(\boldsymbol{Y} - \boldsymbol{\beta X})^2]$. R allows to declare functions in the following way:

```
lmedian <- function(beta, Response, Design) {
  counter <<- counter + 1
  median((Response - Design %*% beta)^2)
}
```

The operator `<-` is used for assignments. Assignments within functions are local. The operator `<<-` is a global assignment, the value of counter will remain modified also outside the function.

Then, we use the method `wireframe` from the package `lattice` to produce a 3D plot. In order to do this we must first determine plot points and then evaluate the function on these points

```
  gr <- expand.grid(beta.0=seq(-1,1,0.08),
                    beta.1=seq(-1,5,0.1))
```

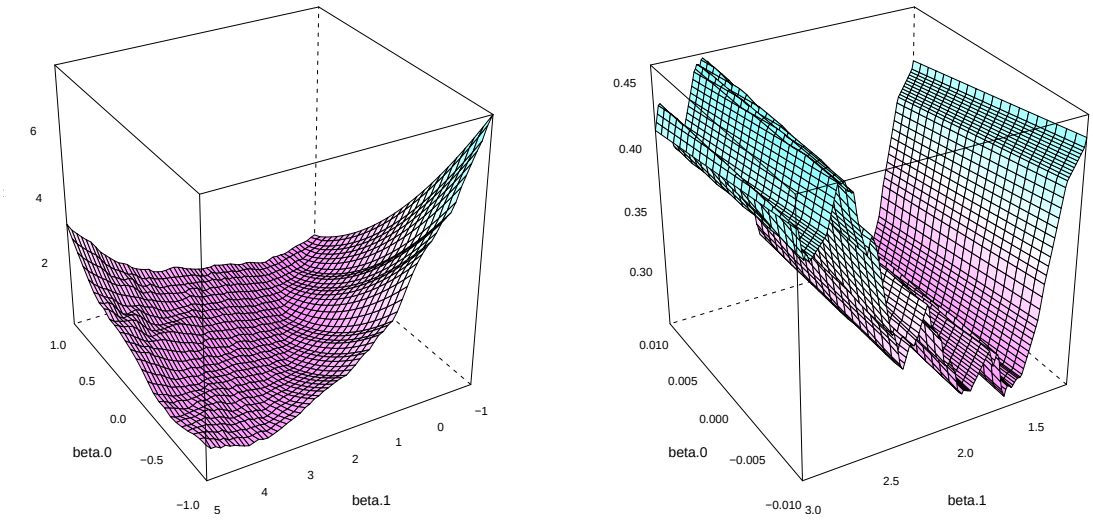

Figure 2: Surface of the function to minimize on the two parameters `beta.0` and `beta.1`.

```
gr$z <- apply(gr,1,function(x) lmedian(x,Y,X))
trellis.par.set("axis.line",list(col=NA,lty=1,lwd=1))
print(
      wireframe(z ~ beta.0 * beta.1, data = gr,
                scales = list(arrows = FALSE),
                drape = TRUE, colorkey = FALSE,
                aspect=c(1,1),
                screen = list(z = 120, x = -60),zoom=1)
      )
```

The outcome is shown in Figure 2, left. At first sight this might seem a well-shaped convex function. However a closer look, shown in Figure 2, right, unveils its real nature:

```
gr <- expand.grid(beta.0=seq(-0.01,0.01,0.001),
                  beta.1=seq(1.2,3,0.01))

gr$z <- apply(gr[,1:2],1,function(x) lmedian(x,Y,X))
trellis.par.set("axis.line",list(col=NA,lty=1,lwd=1))
print(
      wireframe(z ~ beta.0 * beta.1, data = gr,
                scales = list(arrows = FALSE),
                drape = TRUE, colorkey = FALSE,
                aspect=c(1,1),
                screen = list(z = 120, x = -60),zoom=1)
      )
```

The presence of local optima becomes evident and together with it the difficulty of solving this optimization problem.

We must therefore resort to numerical methods and heuristics. The R method `optim` offers an implementation of the Nelder-Mead [2] method and Simulated Annealing for continuous optimization. The Nelder-Mead is the default in the function `optim` provided by R. It requires a starting value for the parameters, the function to optimize and its parameters.

```
> beta.init <- matrix(c(0,0))
> O <- optim(c(beta=beta.init), lmedian, Response=Y, Design=X, control=list(trace=6))

Nelder-Mead direct search function minimizer
function value for initial parameters = 1.730979
  Scaled convergence tolerance is 2.57936e-08
Stepsize computed as 0.100000
BUILD              3 1.730979 1.477877
EXTENSION          5 1.599763 1.184003
EXTENSION          7 1.477877 0.856270
EXTENSION          9 1.184003 0.381746
REFLECTION        11 0.856270 0.306620
.....
REFLECTION       115 0.247260 0.247260
HI-REDUCTION     117 0.247260 0.247260
Exiting from Nelder Mead minimizer
    119 function evaluations used
> O
$par
  beta1   beta2
0.85100 0.45652

$value
[1] 0.24726

$counts
function gradient
     119       NA

$convergence
[1] 0

$message
NULL

> plot(X[,2],Y)
> lines( X[,2], O$par[1]+O$par[2]*X[,2], lty=2, col="blue")
```

Repeating the optimization from different starting points we obtain a value of 0.24581 after 107 function evaluations starting from $\boldsymbol{\beta}_0 = [0, 2]^T$ and a value of 0.39371 after 81 function evaluations starting from $\boldsymbol{\beta}_0 = [0, 3]^T$.

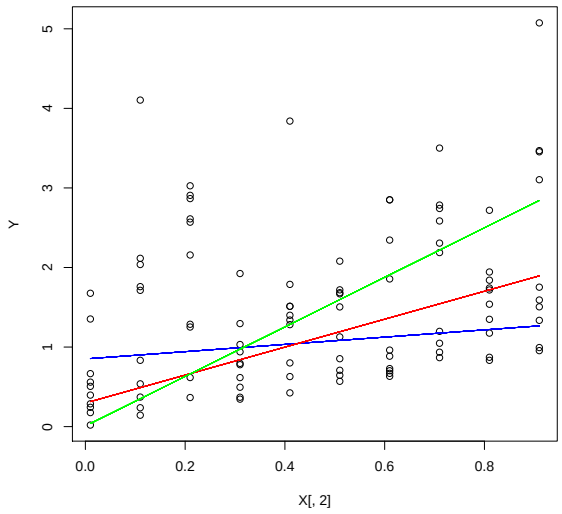

Figure 3: Least median of squares regression. Colors refer to different starting points: blue line refers to $\boldsymbol{\beta}_0 = [0, 0]^T$ with $z(\boldsymbol{\beta}) = 0.24726$, red to $\boldsymbol{\beta}_0 = [0, 2]^T$ with $z(\boldsymbol{\beta}) = 0.24581$ and green to $\boldsymbol{\beta}_0 = [0, 3]^T$ with $z(\boldsymbol{\beta}) = 0.39371$. Finally the black line is the supposed optimum found by one of the random restart strategies illustrated in the next section.

3.1 Optimization methods and algorithms

In the previous section we saw that the initial solution has a strong impact on the final result of the Nelder-Mead algorithm. In this section we want to study the effect of random restart on this algorithm. Each descent starts from a solution randomly chosen in the square $[-2, 3] \times [0, 100]$. We will compare the algorithm without restart and two versions with 100 random restarts. In the first version start points are chosen uniformly at random. In the second version a quasi Monte Carlo method is used. Quasi-Monte Carlo methods are based on low-discrepancy sequences. We use the algorithm and the Fortran implementation by Niederreiter [1]. The difference between the two sampling methods is illustrated in Figure 4.

```
> no.restart <- function(seed = 1) {
+     set.seed(seed)
+     pc <- matrix(c(runif(1, -2, 3), runif(1, 0, 100)))
+     O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
+         control = list(trace = 0))
+     return(O$value)
+ }
> restart.uniform <- function(N, seed = 1) {
+     set.seed(seed)
+     values <- array(dim = c(N))
+     for (i in 1:N) {
+         pc <- matrix(c(runif(1, -2, 3), runif(1, 0, 100)))
+         O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
```

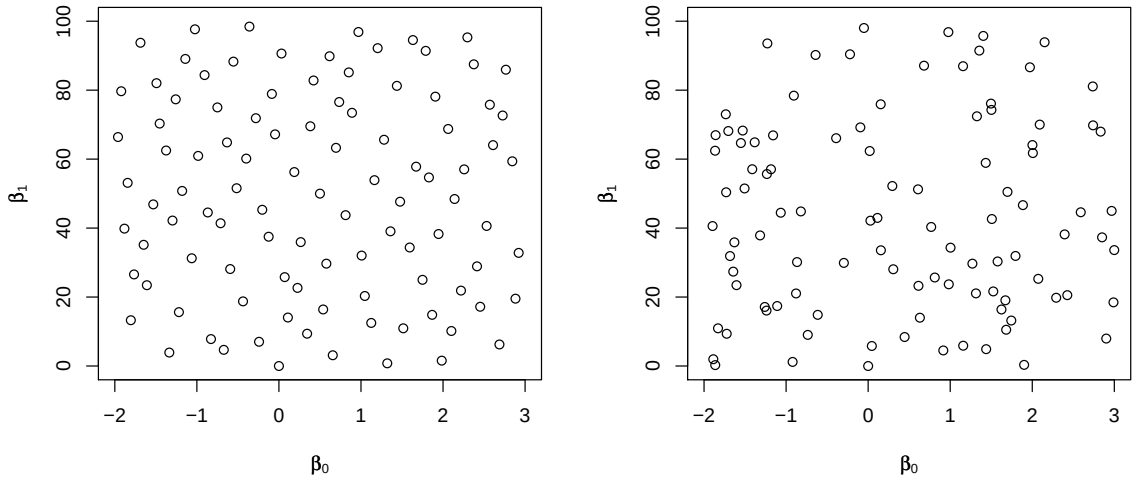

Figure 4: Comparison of a sample of 100 points in the square $[-2, 3] \times [0, 100]$ by quasi-Monte Carlo method (left) and uniform distribution (right). (For the use of mathematical symbols in R plots see `?plotmath`.)

```
+             control = list(trace = 0))
+         values[i] <- O$value
+     }
+     return(min(values))
+ }
> restart.qmc <- function(N, seed = 1) {
+     mylib <- file.path(".", "LowDiscrepancy", paste("niederreiter_prb",
+         .Platform$dynlib.ext, sep = ""))
+     dyn.load(mylib)
+     values <- array(dim = c(N))
+     s <- seed
+     f <- vector(mode = "double", length = 2)
+     for (i in 1:N) {
+         qmc <- .Fortran("generate", as.integer(2), as.integer(2),
+             s = as.integer(s), f = as.double(f))
+         s <- qmc$s
+         pc <- matrix(c(-2 + qmc$f[1] * (3 - (-2)), 0 + qmc$f[2] *
+             (100 - 0)))
+         O <- optim(c(beta = pc), lmedian, Response = Y, Design = X,
+             control = list(trace = 0))
+         values[i] <- O$value
+     }
+     return(min(values))
+ }
```

4 Comparison on one instance

4.1 Univariate analysis

We compare these three algorithms on the basis of the solution cost. We first collect the data:

```
D <- data.frame(no.restart=sapply(1:30,function(x) no.restart(x)),
                uniform.restart=sapply(1:30,function(x) uniform.restart(10,x)),
                qmc.restart=sapply(1:30,function(x) qmc.restart(10,x)))
> D
   no.restart uniform.restart qmc.restart
1      1.7857          1.7857      1.7852
2      1.7857          1.7855      1.7852
3      1.8247          1.7852      1.7852
.....
29     1.7846          1.7846      1.7852
30     1.8069          1.7857      1.7852
> str(D)
'data.frame':        30 obs. of  3 variables:
 $ no.restart     : num  1.79 1.79 1.82 1.98 1.80 ...
 $ uniform.restart: num  1.79 1.79 1.79 1.78 1.79 ...
 $ qmc.restart    : num  1.79 1.79 1.79 1.79 1.79 ...
```

It is good practice to inspect the data and to check that there are no strange results. We can do this drawing some plots. The following lines show how to produce histograms, empirical cumulative distribution functions and boxplots (see Figure 5 and Figure 6).

```
> hist(D$no.restart,
       breaks=seq(-0.01+min(D$no.restart),max(D$no.restart+0.01),0.01),
       probability=TRUE,panel.first=grid())
> lines(density(D$no.restart),col="blue")
> lines(density(D$no.restart,adjust=3),lty=2,col="blue")
> plot.ecdf(D$no.restart,panel.first=grid(),do.points=FALSE,verticals=TRUE)
> boxplot(D$no.restart,horizontal=TRUE,names=c("no.restart"),las=1)
```

Repeating the inspection for the other two algorithms we see that the `qmc.restart` procedure returns always the same value. Indeed the algorithm of Niederreiter [1] is deterministic if the same number of samples is required.

Boxplots can be used for the comparison of the three algorithms. The result of the following line is reported in Figure 7.

```
> boxplot(D)
```

It clearly arises that `qmc.restart` and `uniform.restart` outperform `no.restart` while in the comparison between `qmc.restart` and `uniform.restart` it is more difficult to distinguish a winner. It is also relevant to have a closer insight at some statistics of the numerical data:

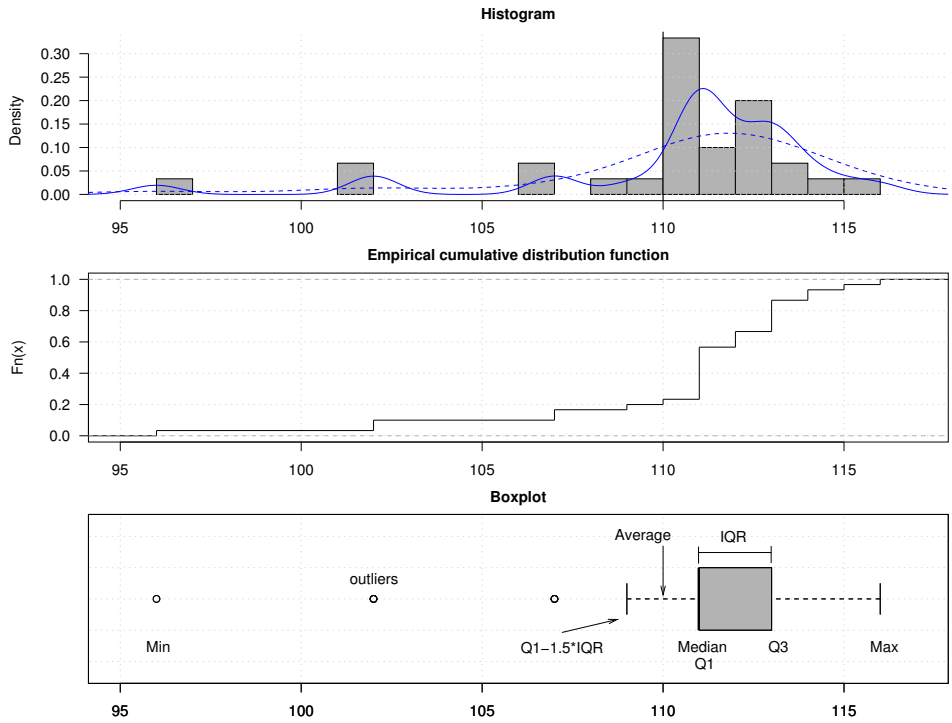

Figure 5: Histogram, density function, empirical distribution function and boxplots are different ways to look at the distribution of empirical data.

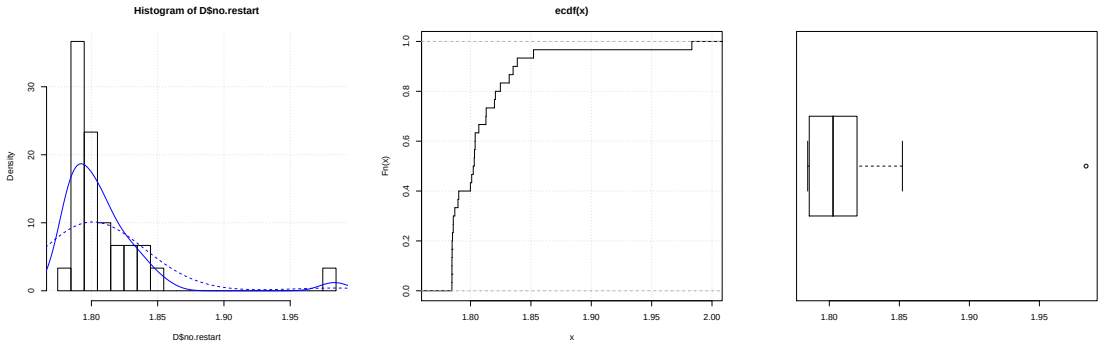

Figure 6: Three different ways to look at the distribution of sampled data.

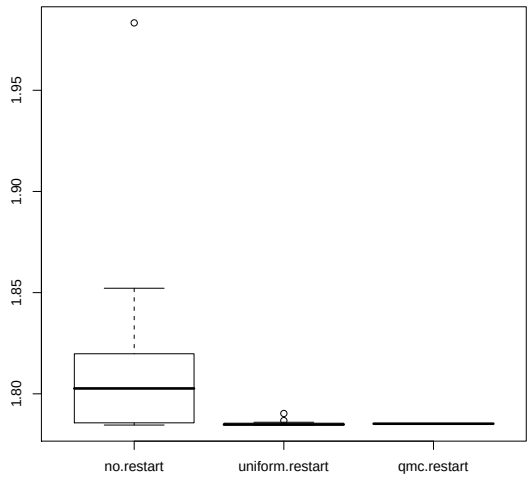

Figure 7: Boxplot comparing the three algorithms.

```
> summary(D)
   no.restart   uniform.restart  qmc.restart
 Min.   :1.78   Min.   :1.78    Min.   :1.79
 1st Qu.:1.79   1st Qu.:1.78    1st Qu.:1.79
 Median :1.80   Median :1.78    Median :1.79
 Mean   :1.81   Mean   :1.79    Mean   :1.79
 3rd Qu.:1.82   3rd Qu.:1.79    3rd Qu.:1.79
 Max.   :1.98   Max.   :1.79    Max.   :1.79
```

The conclusion is that `qmc.restart` and `uniform.restart` perform about the same on this specific instance.

4.2 Bivariate analysis

In the previous analysis we have not considered the computation time. In the following we take that also into account.

```
D2 <- data.frame()
for ( i in 1:30 )
  {
    T <- system.time(S <- no.restart(i))
    D2 <- rbind(D2,data.frame(algorithm="no.restart",trial=i,quality=S,time=T[[1]]))
    T <- system.time(S <- uniform.restart(10,i))
    D2 <- rbind(D2,data.frame(algorithm="uniform.restart",trial=i,quality=S,time=T[[1]]))
    T <- system.time(S <- qmc.restart(10,i))
    D2 <- rbind(D2,data.frame(algorithm="qmc.restart",trial=i,quality=S,time=T[[1]]))
  }
```

```
> D2
         algorithm trial quality  time
1       no.restart     1  1.7857 0.024
2  uniform.restart     1  1.7857 0.292
3      qmc.restart     1  1.7852 0.328
.....
88      no.restart    30  1.8069 0.028
89 uniform.restart    30  1.7857 0.352
90     qmc.restart    30  1.7852 0.312
> str(D2)
'data.frame':        90 obs. of  4 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ trial    : int  1 1 1 2 2 2 3 3 3 4 ...
 $ quality  : num  1.79 1.79 1.79 1.79 1.79 ...
 $ time     : num  0.024 0.292 0.328 0.028 0.344 ...
```

This time we collected the data in the data frame structure in a long format, contrary to `D` that was in wide format. It is however easy to go from one form to the other by means of the functions `stack`, `unstack` and `reshape`, for example,

```
> unstack(D2[,c(3,1)])

   no.restart uniform.restart qmc.restart
1      1.7857          1.7857      1.7852
2      1.7857          1.7855      1.7852
.....
29     1.7846          1.7846      1.7852
30     1.8069          1.7857      1.7852
```

The second format is however more convenient for multivariate analysis, in which the responses for different variables are reported in different columns. In our case the two variables are time and quality. Moreover, this format allows us to use the methods from the package `lattice` which is specific for multivariate data visualization. For example,

```
> library(lattice)
> print(histogram(~quality | factor(algorithm), data = D2, layout = c(1,
+     3)))
```

produces the output in Figure 8.

Back to our comparison, we can plot the results on a time-quality plot. We may do this by plotting all the data or by summarizing them by means of the median. See Figure 9.

```
> print(xyplot(quality ~ time, groups = algorithm, data = D2))

> A <- aggregate(D2$quality, list(algorithm = D2$algorithm), median)
> B <- aggregate(D2$time, list(algorithm = D2$algorithm), median)
> D2s <- merge(A, B, by = "algorithm")
> names(D2s) <- c("algorithm", "quality", "time")
> D2s
```

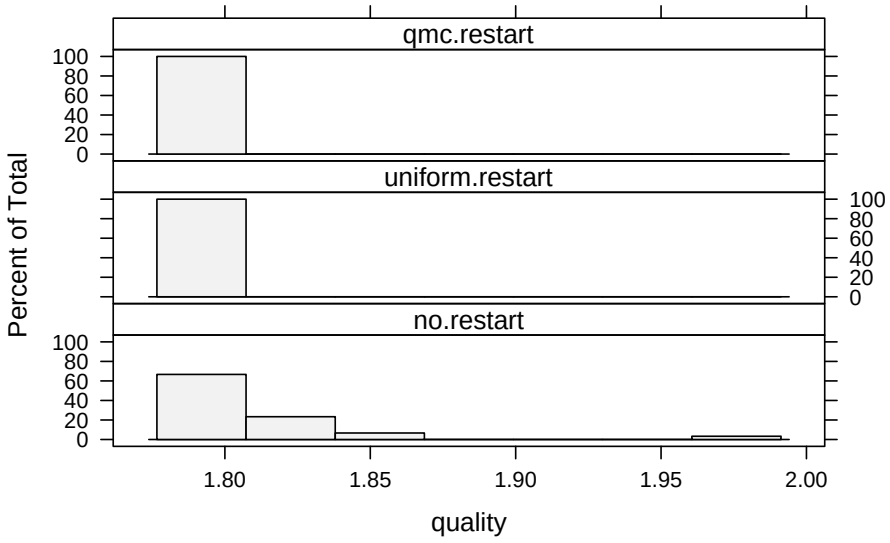

Figure 8: A conditional histogram. The different panels represent differnt algorithms.

```
        algorithm  quality  time
1      no.restart 1.802657 0.028
2     qmc.restart 1.785228 0.304
3 uniform.restart 1.784831 0.312
```

```
> print(xyplot(quality ~ time, data = D2, groups = algorithms,
+     scales = list(relation = "free", y = list(rot = 0, log = FALSE)),
+     panel = function(x, y, subscripts, groups) {
+         panel.grid(h = -1, v = -1, lty = 3)
+         ltext(x = x, y = y, label = groups[subscripts], cex = 0.8,
+             fontfamily = "Helvetica")
+     }, ylab = "Mean time", xlab = "Mean quality"))
```

From Figure 9, right, we may conclude that since the quality of the solutions returned is slightly better for the `uniform.restart` and the computation time is about the same, `uniform.restart` seems a better algorithm in this experiment.

5 Comparison on a set of instances

So far we only evaluated our three algorithms on a single instance. This might be misleading if we want to take a conclusion on which algorithm is the best for instances of a certain type because the instance used for comparison might not be a good representative of the whole population. In this section we will focus on the comparison over a set of instances sampled from the distribution of instances of a certain type.

Let's first generate the instances and store them in a list

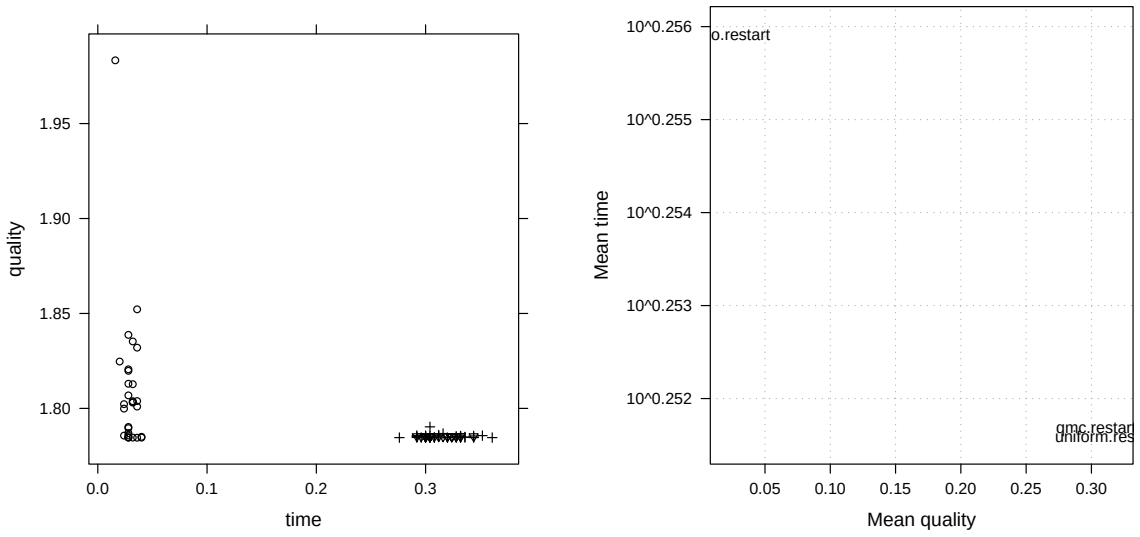

Figure 9: Time-quality plot of the results of 30 runs of the three algorithms. On the left, all data are plotted and algorithms differ by the sign of the points. On the right, the median results are used as coordinate for the algorithm's label.

```
generate.instance <- function(seed)
  {
    set.seed(seed)
    N <- 1000
    X <- array(dim=c(N,2))
    X[,1] <- 1
    X[,2] <- 10*seq(0.01,1,by=0.1)#rnorm(N,5,10)#seq(0.01,1,by=0.1)
    Y <- X %*% matrix(c(0,1)) + matrix(rnorm(N,0,2))
    return(list(X=X,Y=Y))
  }

instances <- list()
for (i in 1:30)
  {
    instances[[i]] <- generate.instance(i)
  }
```

and then let's collect the results

```
D3 <- data.frame()
for ( i in 1:30 )
  {
    X <<- instances[[i]]$X
    Y <<- instances[[i]]$Y
    T <- system.time(S <- no.restart(i))
```

```
    D3 <- rbind(D3,data.frame(algorithm="no.restart",instance=i,quality=S,time=T[[1]]))
    T <- system.time(S <- uniform.restart(10,i))
    D3 <- rbind(D3,data.frame(algorithm="uniform.restart",instance=i,quality=S,time=T[[1]]))
    T <- system.time(S <- qmc.restart(10,i))
    D3 <- rbind(D3,data.frame(algorithm="qmc.restart",instance=i,quality=S,time=T[[1]]))
  }
```

This time we present the analysis of the two variates time and quality using the conditional plots of `lattice`. We first need to reshape the data as follows

```
> str(D3)

'data.frame':        90 obs. of  4 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ instance : int  1 1 1 2 2 2 3 3 3 4 ...
 $ quality  : num  1.79 1.79 1.79 1.90 1.83 ...
 $ time     : num  0.032 0.312 0.332 0.044 0.364 ...

> D3r <- reshape(D3, idvar = "id", timevar = "response", varying = list(c("time",
+     "quality")), direction = "long", times = c("time", "quality"),
+     v.names = "values")
> str(D3r)

'data.frame':        180 obs. of  5 variables:
 $ algorithm: Factor w/ 3 levels "no.restart","uniform.restart",..: 1 2 3 1 2 3 1 2 3 1 ...
 $ instance : int  1 1 1 2 2 2 3 3 3 4 ...
 $ response : chr  "time" "time" "time" "time" ...
 $ values   : num  0.032 0.312 0.332 0.044 0.364 ...
 $ id       : int  1 2 3 4 5 6 7 8 9 10 ...
 - attr(*, "reshapeLong")=List of 4
  ..$ varying:List of 1
  .. ..$ : chr  "time" "quality"
  ..$ v.names: chr "values"
  ..$ idvar  : chr "id"
  ..$ timevar: chr "response"
```

and then produce the plot of Figure 10 with the command

```
> print(bwplot(algorithm ~ values | response, data = D3r, layout = c(1,
+     2), scales = "free"))
```

Looking at the quality response we see that apparently there are no significant differences. This contrasts with the conclusion of Figure 7 and should make us suspicious. Indeed, if no transformation of data is applied the different scale of the instances is likely to hide differences among the performance of the algorithms. We redo the analysis by using rank transformation of the results within each instance (hence results are mapped in the interval $[1, 3]$ because we have three algorithms and one run per instance). The result is illustrated in Figure 11.

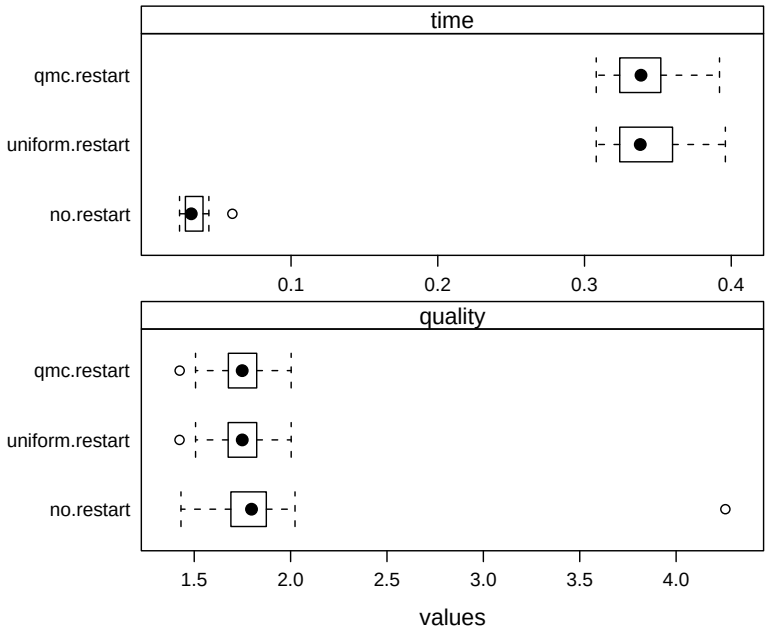

Figure 10: Boxplots of time and quality responses of the three algorithms over 30 instances of the least median of squares regression problem. Times are expressed in seconds and refer to a Intel(R) Core(TM)2 CPU at 1.86GHz.

```
> D3$rank <- unsplit(tapply(D3$quality, D3$instance, rank, ties.method = "average"),
+     D3$instance)
> print(aggregate(D3$rank, list(D3$algorithm), mean))

          Group.1        x
1      no.restart 2.883333
2 uniform.restart 1.650000
3     qmc.restart 1.466667

> print(bwplot(algorithm ~ rank, data = D3))
```

The conclusion we should draw from Figure 11 is opposite to what we had before. When we consider a set of instances the algorithm `qmc.restart` shows best performance.

It might be also worth reporting a table with numerical results. For LaTeX this can be easily achieved with the package `xtable`:

```
> D3t <- unstack(D3[, c(3, 1)])
> library(xtable, lib.loc = "/home/marco/.R/library")
> xtable(D3t[1:5, ])

% latex table generated in R 2.7.0 by xtable 1.4-3 package
% Thu Oct  2 16:16:00 2008
```

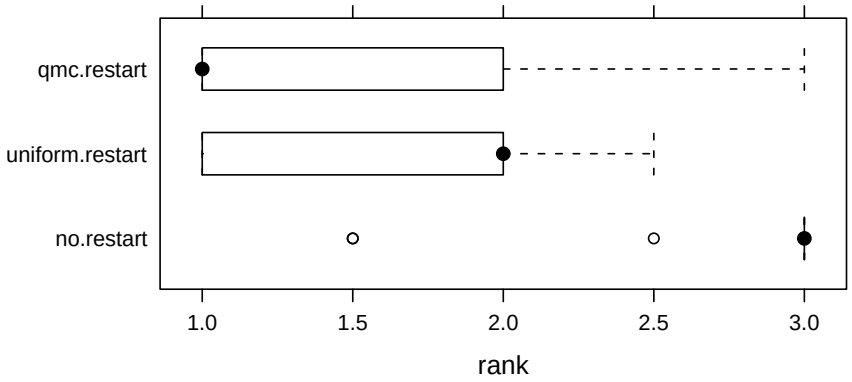

Figure 11: The comparison on a set of instances after rank transformation.

```
\begin{table}[ht]
\begin{center}
\begin{tabular}{rrrr}
  \hline
 & no.restart & uniform.restart & qmc.restart \\
  \hline
1 & 1.79 & 1.79 & 1.79 \\
  2 & 1.90 & 1.83 & 1.83 \\
  3 & 1.81 & 1.78 & 1.78 \\
  4 & 1.67 & 1.67 & 1.67 \\
  5 & 4.26 & 1.72 & 1.72 \\
   \hline
\end{tabular}
\end{center}
\end{table}
```

	no.restart	uniform.restart	qmc.restart
1	1.79	1.79	1.79
2	1.90	1.83	1.83
3	1.81	1.78	1.78
4	1.67	1.67	1.67
5	4.26	1.72	1.72

References

[1] P. Bratley, B. L. Fox, and H. Niederreiter. Algorithm-738 - programs to generate niederreiters low-discrepancy sequences. *ACM Transactions On Mathematical Software*, 20(4):494–495, December 1994.

[2] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4):308–313, 1965. An Errata has been published in The Computer Journal 1965 8(1):27.

[3] Deepayan Sarkar. *Lattice Multivariate Data Visualization with R.* Springer, New York, 2007. ISBN 978-0-387-75968-5.