MANOR: Micro-Array NORmalization of array-CGH data

Pierre Neuvial [1,2,3], Philippe Hupé [1,2,3,4], Isabel Brito [1,2,3], Emmanuel Barillot [1,2,3]

October 17, 2016

1. Institut Curie, 26 rue d'Ulm, Paris cedex 05, F-75248 France
2. INSERM, U900, Paris, F-75248 France
3. École des Mines de Paris, ParisTech, Fontainebleau, F-77300 France
4. UMR 144 CNRS, Paris, F-75248 France
`manor@curie.fr`

Contents

1 Overview

This document gives an overview of the *MANOR* package, which is devoted to the normalization of Array Comparative Genomic Hybridization (array-CGH) data(**?????**). Normalization is a crucial step of microarray analysis which aims at separating biologically relevant signal from experimental artifacts. Typical input data is a file generated by an image analysis software such as Genepix or SPOT (**?**), containing several measurements for each biological variable of interest, i.e. several replicated *spots* for each *clone*; this spot-level data is filtered with various statistical criteria (including a spatial bias detection step which is described in (**?**)), and aggregated into clean clone-level data.

Using the *arrayCGH* framework developped in the package GLAD, which is available under Bioconductor. We propose the formalism of `flags` to handle clone and spot filtering: the core of the normalization process consists in applying to an *arrayCGH* object a list of flags that successively exclude from the data all irrelevant spots or clones.

We also define quality scores (`qscores`) that quantify the quality of an array after normalization: these scores can be used directly to compare

the quality of different arrays after the same normalization process, or to compare the efficiency of different normalization processes on a given array or on a given batch of arrays.

This document is organized as follows: after a short description of optional items we add to *arrayCGH* objects (section **??**, we introduce the classes *flag* (section **??**) and *qscore* (section **??**) with their attributes and dedicated methods; then we describe two useful graphical representation functions (section **??**), namely `genome.plot` and `report.plot`; Afterwards we give a short description of the array-CGH datasets we provide (section **??**); finally we illustrate the usage of *MANOR* by a sample R script (section **??**).

2 *arrayCGH* class

For the purpose of normalization we have added several optional items to the *arrayCGH* objects defined in the R package *GLAD*, including:

cloneValues a data frame with aggregated (clone-level) information, quite similar to *profileCGH* objects of *GLAD*

id.rep the name of a variable common to `cloneValues` and `arrayValues`, that can be used as an identifier for the replicates.

3 *flag* class

We view the process of filtering microarray data, and especially array-CGH data, as a succession of steps consisting in *excluding* from the data unreliable spots or clones (according to criteria such as signal to noise ratio or replicate consistency), and *correcting* signal values from various non-biologically relevant sources of variations (such as spotting effects, spatial effects, or intensity effects).

We introduce the formalism of *flags* to deal with this filtering issue: in the two following subsections, we describe the attributes and methods devoted to *flag* objects.

3.1 Attributes

A *flag* object `f` is a list whose most important items are a function (`f$FUN`) which has to be applied to an object of class *arrayCGH*, and a character value (`f$char`) which identifies flagged spots. Optionally further arguments

can be passed to `f$FUN` via `f$args`, and a label can be added via `f$label`. The examples of this subsection use the function `to.flag`, which is explained in subsection **??**.

3.1.1 Exclusion and correction flags

As stated above, we make the distinction between flags that *exclude* spots from further analysis and flags that *correct* signal values:

exclusion flags If `f` is an exclusion flag, `f$FUN` returns a list of spots to exclude and `f$char` is a non `NULL` value that quickly identifies the flag. In the following example, we define `SNR.flag`, a *flag* objects that excludes spots whose signal to noise ratio lower than the threshold `snr.thr`.

```
> SNR.FUN <- function(arrayCGH, var.FG, var.BG, snr.thr) {
+   which(arrayCGH$arrayValues[[var.FG]] < arrayCGH$arrayValues[[var.BG]]*snr.thr)
+ }
> SNR.char <- "B"
> SNR.label <- "Low signal to noise ratio"
> SNR.flag <- to.flag(SNR.FUN, SNR.char, args=alist(var.FG="REF_F_MEAN", var.BG="REF_
```

correction flags If `f` is a correction flag, `f$FUN` returns an object of type *arrayCGH* and `f$char` is `NULL`. In the following example, `global.spatial.flag` computes a spatial trend on the array, and corrects the signal log-ratios from this spatial trend:

```
> global.spatial.FUN <- function(arrayCGH, var)
+   {
+     if (!is.null(arrayCGH$arrayValues$Flag))
+         arrayCGH$arrayValues$LogRatio[which(arrayCGH$arrayValues$Flag!="")] <- NA
+ ##     Trend <- arrayTrend(arrayCGH, var, span=0.03, degree=1, iterations=3, family
+     Trend <- arrayTrend(arrayCGH, var, span=0.03, degree=1, iterations=3)
+     arrayCGH$arrayValues[[var]] <- Trend$arrayValues[[var]]-Trend$arrayValues$Trend
+     arrayCGH
+   }
> global.spatial.flag <- to.flag(global.spatial.FUN, args=alist(var="LogRatio"))
```

3.1.2 Permanent and temporary flags

We introduce an additional distinction between *permanent* and *temporary* flags in order to deal with the case of spots or clone that are known to be

biologically relevant, but that have not to be taken into account for the computation of a scaling normalization coefficient. For example in breast cancer, when the reference DNA comes from a male, we expect a gain of the X chromosome and a loss of the Y chromosome in the tumoral sample, and we do not want log-ratio values for X and Y chromosome to bias the estimation of a scaling normalization coefficient.

Any *flag* object therefore contains an argument called `type`, which defaults to `"perm"` (*permanent*) but can be set to `"temp"` in the case of a temporary flag. In the following example, `chromosome.flag` is a *temporary* flag that identifies clones correcponding to X and Y chromosome:

```
> chromosome.FUN <- function(arrayCGH, var) {
+   var.rep <- arrayCGH$id.rep
+   w <- which(!is.na(match(as.character(arrayCGH$cloneValues[[var]]), c("X", "Y"))))
+   l <- arrayCGH$cloneValues[w, var.rep]
+   which(!is.na(match(arrayCGH$arrayValues[[var.rep]], as.character(l))))
+ }
> chromosome.char <- "X"
> chromosome.label <- "Sexual chromosome"
> chromosome.flag <- to.flag(chromosome.FUN, chromosome.char, type="temp.flag", args=
```

3.2 Methods

3.2.1 to.flag

The function `to.flag` is used of the creation of *flag* objects, with the specificities described in subsection **??**.

```
> args(to.flag)

function (FUN, char = NULL, args = NULL, type = "perm.flag",
    label = NULL)
NULL
```

3.2.2 flag.arrayCGH

Function `flag.arrayCGH` simply applies function `flag$FUN` to a *flag* object for filtering, and returns:

- a filtered array with field `arrayCGH$arrayValues$Flag` filled with the value of `flag$char` for each spot to be excluded from further analysis in the case of an exclusion flag;

- an array with corrected signal value in the case of a correction flag.

```
> args(flag.arrayCGH)

function (flag, arrayCGH)
NULL
```

3.2.3 flag.summary

Function `flag.summary` computes spot-level information about normalization (including the number of flagged spots and numeric normalization parameters), and displays it in a convenient way. This function can either be applied to an object of type *arrayCGH*:

```
> args(flag.summary.arrayCGH)

function (arrayCGH, flag.list, flag.var = "Flag", nflab = "not flagged",
    ...)
NULL
```

or to plain spot-level information, by using the default method:

```
> args(flag.summary.default)

function (spot.flags, flag.list, nflab = "not flagged", ...)
NULL
```

4 *qscore* class

As we point out in the introduction of this document, evaluating the quality of an array-CGH after normalization is of major importance, since it helps answering the following questions:

- which is the best normalization process ?
- which array is of best quality ?
- what is the quality of a given array ?

To this purpose we define quality scores (*qscores*), which attributes and methods are explianed in the two following subsections.

4.1 Attributes

A *qscore* object `qs` is a list which contains a function (`qs$FUN`), a name (`qs$name`), and optionnally a label (`qs$label`) and arguments to be passed to `qs$FUN` (`qs$args`). In the following example, the quality score `pct.spot.qscore` evaluates the percentage of spots that have passed the filtering steps of normalization; it provides an evaluation of the array quality for a given normalization process. The function `to.qscore` is explained in subsection **??**.

```
> pct.spot.FUN <- function(arrayCGH, var) {
+   100*sum(!is.na(arrayCGH$arrayValues[[var]]))/dim(arrayCGH$arrayValues)[1]
+ }
> pct.spot.name <- "SPOT_PCT"
> pct.spot.label <- "Proportion of spots after normalization"
> pct.spot.qscore <- to.qscore(pct.spot.FUN, name=pct.spot.name, args=alist(var="LogR
```

4.2 Methods

4.2.1 to.qscore

The function `to.qscore` is used of the creation of *qscore* objects, with the specificities described in subsection **??**.

```
> args(to.qscore)

function (FUN, name = NULL, args = NULL, label = NULL, dec = 3)
NULL
```

4.2.2 qscore.arrayCGH

Function `qscore.arrayCGH` simply computes and returns the value of *qscore* for *arrayCGH*:

```
> args(qscore.arrayCGH)

function (qscore, arrayCGH)
NULL
```

4.2.3 qscore.summary.arrayCGH

Function `qscore.summary.arrayCGH` computes all quality scores of a list (using function `qscore.arrayCGH`), and displays the results in a convenient way.

```
> args(qscore.summary.arrayCGH)

function (arrayCGH, qscore.list)
NULL
```

5 Data

We provide examples of array-CGH data coming from two different platforms. These data illustrate the need for appropriate within-array normalization methods, and especially the need for methods that handle spatial effects.

```
> data(spatial)
```

For each array we provide raw data (generated by Genepix or SPOT (**?**)), as well as the corresponding *arrayCGH* object before and after normalization.

These arrays illustrate the main source of non biological variability of these data sets, namely spatial effects. We classify these effects into two non exclusive types: local bias and global gradients. In the case of *local bias*, entire areas of the array show lower or higher signal values than the rest of the array, with no biological explanation (array `edge`); to our experience, this particular type of artifact roughly affects an array out of two. In the case of *global gradients*, the array shows an obvious signal gradient from one side of the slide to the other (array `gradient`).

5.1 `edge`

Bladder cancer tumors were collected at Henri Mondor Hospital (Créteil, France) (**?**) and hybridized on arrays CGH composed of 2464 Bacterian Artificial Chromosomes (F. Radvanyi, D. Pinkel et al., unpublished results); each of these BAC is spotted three times on the array, and the three replicates are neighbors on the array. We give the example of an `arrayCGH` with local spatial effects (figure **??**): high log-ratios cluster in the upper-right corner of the array.

5.2 `gradient`

We give the example of two arrays from a breast cancer data set from Institut Curie (O. Delattre, A. Aurias et al., unpublished results). These arrays consist of 3342 clones, organized as a 4×4 superblock that is replicated

```
> data(spatial)
> ## edge: example of array with local spatial effects
> arrayPlot(edge, "LogRatio", main="Local spatial effects", zlim=c(-1,1), mediancente
```

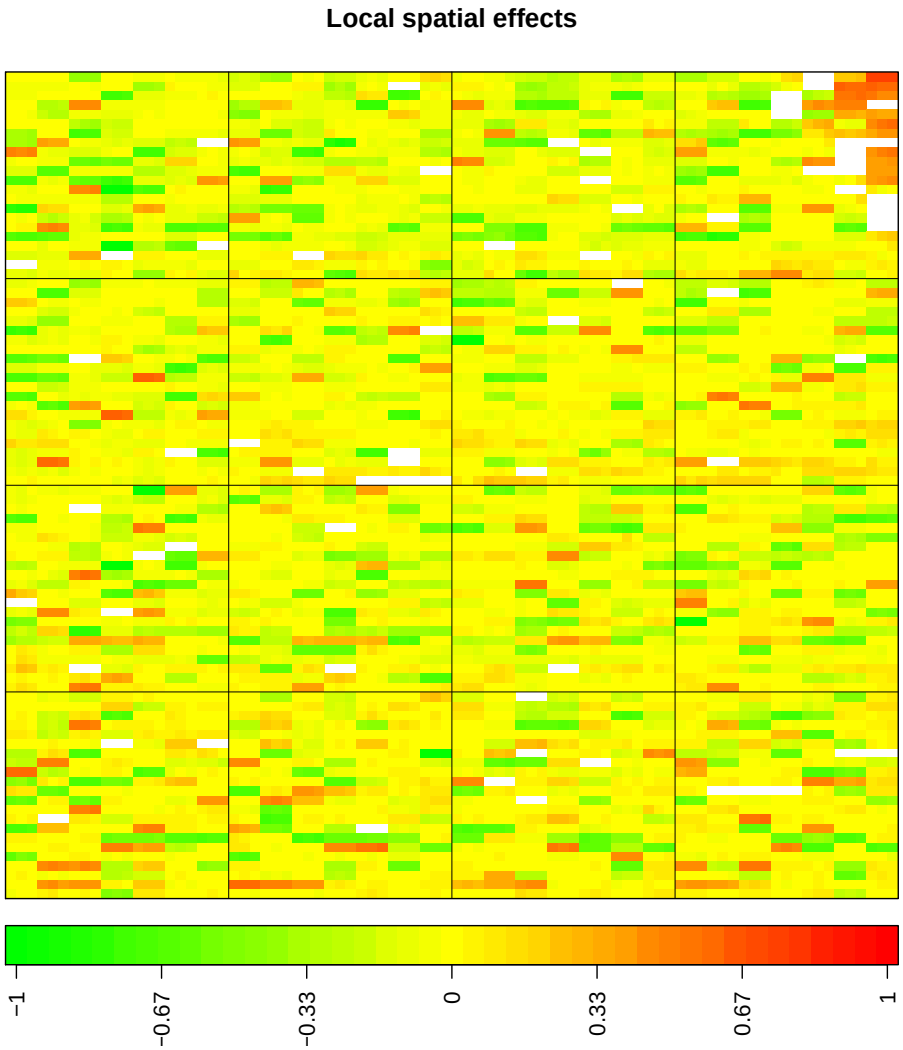

Figure 1: *array with local spatial effects.*

three times. This data set is affected by the two types of spatial effects: local bias areas (as for the previous data set), and spatial gradients from one side of the array to the other. The array `gradient` illustrates this second type of spatial effect.

```
> data(spatial)
> arrayPlot(gradient, "LogRatio", main="Spatial gradient" , zlim=c(-2,2), mediancente
```

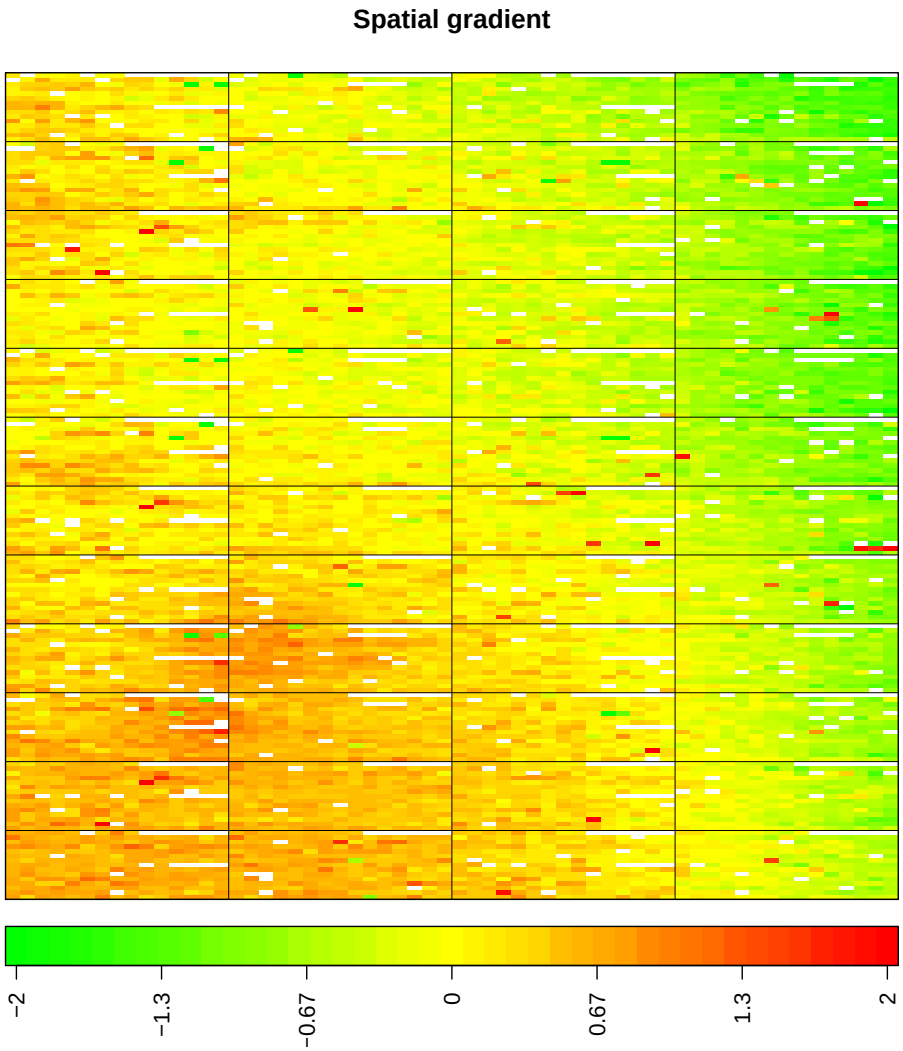

Figure 2: *Example of array with spatial gradient.*

6 Graphical representations

As for any type of data analysis, appropriate graphical representations are of major importance for data understanding. Array-CGH data are typically ratios or log-ratios, that correspond to locations on the array (spots) and to locations on the genome (clones). Therefore in the case of array-CGH data normalization, two complementary types of representations are necessary:

- a dotplot of the array, that takes into account the array design. This is a crucial tool in the case of array-CGH data normalization for two reasons: first it provides an easy way to *identify* spatial artifacts such as row, column, print-tip group effects, as well as spatial bias and spatial gradients on the array; then it performs a post-normalization *control*, to ensure that the normalization procedure reached its goals, i.e. significantly reduced the observed effects.

- a plot of the signal values along the genome, which gives a visual impression of the array quality on the edge of biological relevance; comparing the signal shape before and after normalization provides a qualitative idea of the imrpovement in data quality provided by the normalization method.

The `arrayPlot` method provided by the *GLAD* package and based on `maImage` (**?**) addresses the first point; we add two methods to this toolbox:

- the `genome.plot` method displays a plot of any signal value (e.g. log-ratios) along the genome;

- the `report.plot` method successively calls `arrayPlot` and `genome.plot` in order to provide a simultaneous vision of the data using the two relevant metrics (array and genome), with approproate color scales.

6.1 genome.plot

This method provides a convenient way to plot a given signal along the genome; the signal values can be colored according to their level (which is the default comportment of the function) or to the level of any other variable, in the following way:

- if the variable is numeric (e.g. signal to noise ratio), the function assumes that it is a quantitative variable and adapts a color palette to its values (figure **??**)

```
> data(spatial)
> par(mfrow=c(7,5), mar=par("mar")/2)
> genome.plot(edge.norm, chrLim="LimitChr", cex=1)
```

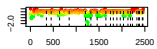

Figure 3: *Pan-genomic profile of the array. Colors are proportional to log-ratio values.*

- if the variable is not numeric (e.g. the copy number variation as estimated by *GLAD*, or a character variable making the disitnction between flagged and un-flagged clones), the function counts the number of modalities of the variable and defines an appropriate color scale using the `rainbow` function (figure **??**).

6.2 report.plot

This method successively calls `arrayPlot` and `genome.plot`; it checks for color scale consistency between plots, and can automatically set the plot layout (figure **??**).

7 Sample `MANOR` sessions

In this section we illustrate the use of *MANOR* on two CGH arrays. Our examples contain several steps, including data preparation, flag definition, array normalization, quality criteria definition, and quality assessment of the array, and highlights of the normalization process.

```
> data(spatial)
> edge.norm$cloneValues$ZoneGNL <- as.factor(edge.norm$cloneValues$ZoneGNL)
> par(mfrow=c(7,5), mar=par("mar")/2)
> genome.plot(edge.norm, col.var="ZoneGNL", chrLim="LimitChr", cex=1)
```

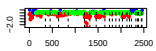

Figure 4: *Pan-genomic profile of the array. Colors correspond to the values of the variable "ZoneGNL".*

```
> data(spatial)
> report.plot(edge.norm, chrLim="LimitChr", zlim=c(-1,1), cex=1)
```

Figure 5: ***report.plot**: array image and pan-genomic profile after normalization.*

7.1 array edge

7.1.1 Data preparation: import

```
> dir.in <- system.file("extdata", package="MANOR")
> ## import from 'spot' files
> spot.names <- c("LogRatio", "RefFore", "RefBack", "DapiFore", "DapiBack", "SpotFlag
> clone.names <- c("PosOrder", "Chromosome")
> edge <- import(paste(dir.in, "/edge.txt", sep=""), type="spot",
+ spot.names=spot.names, clone.names=clone.names, add.lines=TRUE)

[1] "number of lines does not match array design: adding empty lines..."
```

7.1.2 Normalization: norm

Figure **??** shows the results of the normalization process.

```
> data(flags)
> data(spatial)
> ## local.spatial.flag$args <- alist(var="ScaledLogRatio", by.var=NULL, nk=5, prop=0
> local.spatial.flag$args <- alist(var="ScaledLogRatio", by.var=NULL, nk=5, prop=0.25
> flag.list <- list(spatial=local.spatial.flag, spot=spot.corr.flag, ref.snr=ref.snr.
> edge.norm <- norm(edge, flag.list=flag.list, FUN=median, na.rm=TRUE)

[1] "spatial"

**********************************************
*** Spatial Classification with EM algorithm ***
**********************************************


Data :   nb points   =       7392
  grid size =    88 rows,   84 columns

Neighborhood system :
  max neighb =           4
  Default 1st-order neighbors (horizontal and vertical)


NEM parameters :
  beta       =        1.00   |   nk                    =   5
```

```
Computing initial partition (sort variable 1) ...

  criterion NEM = 19782.898 / Ps-Like = 5035.969 / Lmix = 9250.229
  NEM converged after 173 iterations

[1] "mean of unbiased zone :  -0.0233232743043007"
[1] "Spatial bias has been detected"
  zone.number           mu effectif effectif.cumul frequency.cumul biased.zone
4           5  0.467833333       66             66     0.009189641           1
3           4  0.046085967     1582           1648     0.229462545           0
5           3  0.005084592     2648           4296     0.598162072           0
1           2 -0.032626474     1866           6162     0.857978279           0
2           1 -0.080052941     1020           7182     1.000000000           0
[1] "spot"
[1] "ref.snr"
[1] "dapi.snr"
[1] "rep"
[1] "unique"
```

```
> edge.norm <- sort(edge.norm, position.var="PosOrder")
```

```
> report.plot(edge.norm, chrLim="LimitChr", zlim=c(-1,1), cex=1)
```

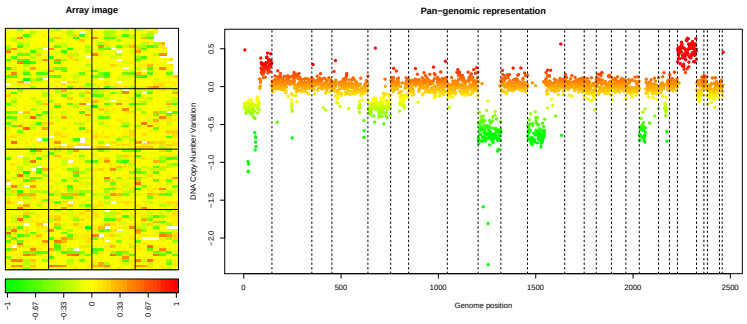

Figure 6: *array 'edge' after normalization.*

7.1.3 Quality assessment: `qscore.summary.arrayCGH`

```
> ##DNA copy number assessment: GLAD
> profileCGH <- as.profileCGH(edge.norm$cloneValues)
> profileCGH <- daglad(profileCGH, smoothfunc="lawsglad", lkern="Exponential", model=
```

```
[1] "Smoothing for each Chromosome"
[1] "Optimization of the Breakpoints and DNA copy number calling"
[1] "Check Breakpoints Position"
[1] "Results Preparation"
```

```
> edge.norm$cloneValues <- as.data.frame(profileCGH)
> edge.norm$cloneValues$ZoneGNL <- as.factor(edge.norm$cloneValues$ZoneGNL)
> data(qscores)
> ## list of relevant quality scores
> qscore.list <- list(smoothness=smoothness.qscore,
+                     var.replicate=var.replicate.qscore,
+                     dynamics=dynamics.qscore)
> edge.norm$quality <- qscore.summary.arrayCGH(edge.norm, qscore.list)
> edge.norm$quality
```

```
             name                                        label score
1 LOCAL_SMOOTHNESS Local signal variability along the genome 0.021
2    VAR_REPLICATE      Average variability among replicates 0.011
3  SIGNAL_DYNAMICS Dynamics of the DNA copy number variation 0.398
```

7.1.4 Highlights of the normalization process: `html.report`

Function `html.report` generates an HTML file with key features of the normalization process: array image and genomic profile before and after normalization, spot-level flag report, and value of the quality criteria.

```
> html.report(edge.norm, dir.out=".", array.name="an array with local bias", chrLim="
```

The results of the previous command can be viewed in the file edge.html.

7.2 array `gradient`

Here we give the example of the normalization of an array with spatial gradient.

7.2.1 Data preparation: `import`

```
> ## import from 'gpr' files
> spot.names <- c("Clone", "FLAG", "TEST_B_MEAN", "REF_B_MEAN", "TEST_F_MEAN", "REF_F
> clone.names <- c("Clone", "Chromosome", "Position", "Validation")
> ac <- import(paste(dir.in, "/gradient.gpr", sep=""), type="gpr", spot.names=spot.na
```

```
[1] "number of lines does not match array design: adding empty lines..."
[1] "calculating array design..."
```

```
> ## compute log-ratio
> ac$arrayValues$F1 <- log(ac$arrayValues[["TEST_F_MEAN"]], 2)
> ac$arrayValues$F2 <- log(ac$arrayValues[["REF_F_MEAN"]], 2)
> ac$arrayValues$B1 <- log(ac$arrayValues[["TEST_B_MEAN"]], 2)
> ac$arrayValues$B2 <- log(ac$arrayValues[["REF_B_MEAN"]], 2)
> Ratio <- (ac$arrayValues[["TEST_F_MEAN"]]-ac$arrayValues[["TEST_B_MEAN"]])/
+     (ac$arrayValues[["REF_F_MEAN"]]-ac$arrayValues[["REF_B_MEAN"]])
> Ratio[(Ratio<=0)|(abs(Ratio)==Inf)] <- NA
> ac$arrayValues$LogRatio <- log(Ratio, 2)
> gradient <- ac
```

7.2.2 Normalization: `norm`

Figure **??** shows the results of the normalization process.

```
> data(spatial)
> data(flags)
> flag.list <- list(local.spatial=local.spatial.flag, spot=spot.flag, SNR=SNR.flag, g
> gradient.norm <- norm(gradient, flag.list=flag.list, FUN=median, na.rm=TRUE)
```

```
[1] "local.spatial"

**********************************************
*** Spatial Classification with EM algorithm ***
**********************************************


Data :   nb points   =      10800
  grid size =   180 rows,   60 columns

Neighborhood system :
  max neighb =           4
  Default 1st-order neighbors (horizontal and vertical)


NEM parameters :
  beta       =        1.00   |   nk                    =   7
```

```
Computing initial partition (sort variable 1) ...
Warning : pt 0 density = 0

  criterion NEM = 12882.131 / Ps-Like = -11189.528 / Lmix = 9832.894
  NEM converged after 1555 iterations

[1] "mean of unbiased zone :  8.43300291647224"
[1] "There is no spatial bias"
  zone.number       mu effectif effectif.cumul frequency.cumul biased.zone
6           2 8.442732     1460           1460       0.1455343           0
4           1 8.439766     1419           2879       0.2869817           0
7           3 8.435628     1408           4287       0.4273325           0
2           5 8.432529     1474           5761       0.5742624           0
3           6 8.431828     1370           7131       0.7108254           0
1           4 8.431820     1414           8545       0.8517743           0
5           7 8.426740     1487          10032       1.0000000           0
[1] "spot"
[1] "SNR"
[1] "global.spatial"
[1] "val.mark"
[1] "position"
[1] "unique"
[1] "amplicon"
[1] "chromosome"
[1] "replicate"
```

```
> gradient.norm <- sort(gradient.norm)
```

7.2.3 Quality assessment: qscore.summary.arrayCGH

```
> ##DNA copy number assessment: GLAD
> profileCGH <- as.profileCGH(gradient.norm$cloneValues)
> profileCGH <- daglad(profileCGH, smoothfunc="lawsglad", lkern="Exponential", model=
```

```
[1] "Smoothing for each Chromosome"
[1] "Optimization of the Breakpoints and DNA copy number calling"
[1] "Check Breakpoints Position"
[1] "Results Preparation"
```

```
> gradient.norm$cloneValues <- as.data.frame(profileCGH)
> gradient.norm$cloneValues$ZoneGNL <- as.factor(gradient.norm$cloneValues$ZoneGNL)
```

```
> genome.plot(gradient.norm, chrLim="LimitChr", cex=1)
```

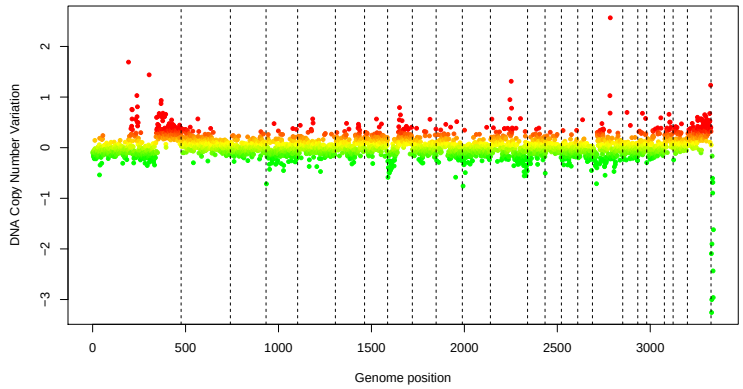

Figure 7: *array* ***gradient*** *after normalization.*

```
> data(qscores)
> ## list of relevant quality scores
> qscore.list <- list(smoothness=smoothness.qscore, var.replicate=var.replicate.qscor
> gradient.norm$quality <- qscore.summary.arrayCGH(gradient.norm, qscore.list)
> gradient.norm$quality
```

```
             name                                          label score
1 LOCAL_SMOOTHNESS Local signal variability along the genome 0.033
2    VAR_REPLICATE      Average variability among replicates 0.050
3  SIGNAL_DYNAMICS Dynamics of the DNA copy number variation 0.294
```

7.2.4 Highlights of the normalization process: `html.report`

Function `html.report` generates an HTML file with key features of the normalization process: array image and genomic profile before and after normalization, spot-level flag report, and value of the quality criteria.

```
> html.report(gradient.norm, dir.out=".", array.name="an array with spatial gradient"
```

The results of the previous command can be viewed in the file gradient.html.

8 Session information

The version number of R and packages loaded for generating this document are:

```
> sessionInfo()

R version 3.3.1 (2016-06-21)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: OS X 10.9.5 (Mavericks)

locale:
[1] C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] MANOR_1.46.0 GLAD_2.38.0

loaded via a namespace (and not attached):
[1] tools_3.3.1
```

9 Supplementary data

The package *MANOR* provides sample gpr and spot files, as examples to the `import` funciton. However, due to space limitations, only the first 100 lines these file are provided in the current distribution of *MANOR*. The full files can be downloaded from here:

- 'gpr' file: gradient.gpr
- 'spot' file: edge.txt