Comprehensive Pipeline for Analyzing and Visualizing Array-Based CGH Data

Frederic Commo*

*fredcommo@gmail.com

October 27, 2023

1 Introduction

Genomic profiling using array-based comparative genomic hybridization (aCGH) is widely used within precision medicine programs, in combination with DNA sequencing, to match specific molecular alterations (amplifications or deletions) with therapeutic orientations.
We present *rCGH*, a comprehensive array-based CGH analysis workflow, integrating functionalities specifically designed for precision medicine. *rCGH* ensures a full traceability by saving all the process parameters , and facilitates genomic profiles interpretation and decision-making through interactive visualizations.
rCGH supports commercial arrays : Agilent (from 44K to 400K arrays), and Affymetrix SNP6.0 and cytoScanHD. Custom arrays can also be supported, provided a suitable data format is passed. See subsection 4.1 for details, and [1].

2 Quick start

A typical workflow is of the form:

```
> cgh <- readAffyCytoScan("path/to/cytoScan.CNCHP.txt")
> cgh <- adjustSignal(cgh)
> cgh <- segmentCGH(cgh)
> cgh <- EMnormalize(cgh)
```

Then, the genomic profile can be visualized or stored as any R object. The segmentation table can be extracted, then transformed into a by-gene table, or used for any further analysis. All these functions and features are detailed in the next sections.

3 *rCGH* object structure

In order to store (or update) data, sample information, and the workflow parameters all along a genomic profile analysis process, `rCGH` objects are structured as follow:

- info: the sample information.
- cnSet: the full by-probe dataset.
- param: the workflow parameters, for traceability.
- segTable: the segmentation data.

All these slots are accessible through specific functions, as described in the next sections.
Notice that *rCGH* is a superclass designed for calling common methods. Depending on the type of array and the *read* functions used, the resulting objects will be assigned to classes *rCGH-Agilent*, *rCGH-SNP6*, *rCGH-cytoScan*, or *rCGH-generic*. These classes inherit from the superclass, and allow array-specific pre-parametrizations.
rCGH-generic is a particular class, not dedicated to a specific platform. The associated `readGeneric` read function allows the creation of a *rCGH* object from custom arrays, provided the data contains mandatory columns, as described in the next section.

4 *rCGH* functions

rCGH provides functions for each of the analysis steps, from reading files to visualizing genomic profiles. Several *get* functions allow the user to get access to specific results and workflow parameters, saved and stored at each step.

4.1 Reading files

4.1.1 Commercial arrays

Agilent Feature Exraction files (from 44K to 400K arrays), and Affymetrix SNP6.0 and cytoScanHD data are supported.
To keep more flexibility, Affymetrix CEL files have to be first read using ChAS or Affymetrix Power Tools (APT) [2], and then exported as cychp.txt or cnchp.txt files. Notice that cnchp.txt files contain Allelic differences, that allow the loss of heterozygosity (LOH) to be estimated, while cychp.txt files do not.

Due to specific files structures, and since preambles may be missing (depending on ChAS and APT versions), *rCGH* provides specific read/build-object functions:

- `readAgilent()`: 44K to 400K FE (.txt) files.
- `readAffySNP6()`: cychp, cnchp and probeset (.txt) files, exported from SNP6.0 CEL, through ChAS or APT.
- `readAffyCytoScan()`: cychp, cnchp and probeset (.txt) files, exported from CytoScanHD CEL, through ChAS or APT.

Notice that these `read` functions have a *genome*, which allow the user to specify what genome build to use with the current array. The supported genome builds are hg18, hg19 (default) and hg38. This value is stored, then used in the plot functions.

4.1.2 Custom arrays

Custom arrays can be read using `readGeneric()`, which leads to construct an object of class *rCGH-generic*. Data as to be provided as a text file, with the following mandatory information.

Mandatory columns for custom arrays:

- ProbeName: Character strings. Typicaly the probe ids.
- ChrNum: numeric. The chromosome numbers. In case Chr X and Y are used and named as *"X"* and *"Y"*, these notations will be converted into *"23"* and *"24"*, respectively.
- ChrStart: numeric. The chromosomal probe locations.
- Log2Ratio: numeric. The corresponding Log2Ratios.

Each of the `read` functions take the file's path as the unique mandatory argument. Other optional arguments allow the user to save supplementary information: *sampleName, labName*:

```
> library(rCGH)
> filePath <- system.file("extdata",
+     "Affy_cytoScan.cyhd.CN5.CNCHP.txt.bz2",
+     package = "rCGH")
> cgh <- readAffyCytoScan(filePath, sampleName = "CSc-Example",
+                         labName = "myLab")
```

```
> cgh

                                                    info
fileName         Affy_cytoScan.cyhd.CN5.CNCHP.txt.bz2
sampleName                                  CSc-Example
labName                                           myLab
analysisDate                                 2023-10-27
usedProbes                                          snp
genome                                            hg19
ploidy                                               2
platform                                CytoScanHD_Array
barCode             @520825009581671130164248036027l5
gridName               CytoScanHD_Array.na33.annot.db
scanDate                                     2015-01-22
programVersion                                    5.0.0
gridGenomicBuild                            hg19/GRCh37
reference          CytoScanHD_Array.na33.r1.REF_MODEL
rCGH_version                                      1.32.0
```

In complement, any kind of useful annotation (logical, string or numeric) can be added, with `setInfo()`:

```
> setInfo(cgh, "item1") <- 35
> setInfo(cgh, "item2") <- TRUE
> setInfo(cgh, "item3") <- "someComment"
```

At any time, the full (or specific) annotations stored can be accessed:

```
> getInfo(cgh)

                                                    info
fileName         Affy_cytoScan.cyhd.CN5.CNCHP.txt.bz2
sampleName                                  CSc-Example
labName                                           myLab
analysisDate                                 2023-10-27
usedProbes                                          snp
genome                                            hg19
ploidy                                               2
platform                                CytoScanHD_Array
barCode             @520825009581671130164248036027l5
gridName               CytoScanHD_Array.na33.annot.db
```

```
scanDate                                      2015-01-22
programVersion                                     5.0.0
gridGenomicBuild                              hg19/GRCh37
reference          CytoScanHD_Array.na33.r1.REF_MODEL
rCGH_version                                     1.32.0
item1                                                35
item2                                              TRUE
item3                                       someComment
```

```
> getInfo(cgh, c("item1", "item3"))
```

```
       item1         item3
        "35" "someComment"
```

4.2 Adjusting signals

When Agilent dual-color hybridization are used, GC content and the cy3/cy5 bias are necessary adjustments. `adjustSignal()` handle these steps before computing the $\log_2(RelativeRatios)$ (LRR). In both cases, a local regression (`loessFit`, R package *limma*) is used [3].
Note that by default, the cyanine3 signal is used as the reference. Use `Ref=cy5` if cyanine5 signal has to be used as the reference.
In case of Affymetrix cychp or cnchp files, these steps have already been processed, and `adjustSignal()` simply rescale the LRR, when `Scale=TRUE` (default). As for Agilent data, some useful quality scores: the derivative Log Ratio Spread (dLRs) and the LRR Median Absolute Deviation (MAD), are stored in the object.

```
> cgh <- adjustSignal(cgh, nCores=1)
```

```
Log2Ratios QCs:
 dLRs:  0.162
 MAD: 0.128


Scaling...
Signal filtering...
Modeling allelic Difference...
```

4.3 Segmenting

One possible strategy for segmenting the genome profile consists in identifying breakpoints all along the genome, when exist. These breakpoints define the DNA segments start and end positions. To do so, *rCGH* uses the Circular Binary Segmentation algorithm (*CBS*) [4] from the *DNAcopy* package [5].
All the steps are wrapped into one unique easy-to-use function, `segmentCGH()`. In order to faclitate its use, all the parameters but one are predefined: `UndoSD` is kept free. When this parameter is set to `NULL` (default), its optimal value is estimated directly from the values. However, the user can specify its own value, generaly from 0.5 to 1.5.
The resulting segmentation table is of the form of a standard *DNAcopy* output, plus additional columns:

- ID : sample Id.
- chrom : chromosome number.
- loc.start : segment start position.
- loc.end : segment end position.
- num.mark : number of markers within each segment.
- seg.mean : the mean LRR along each segment.
- seg.med : the median LRR along each segment.
- probes.Sd : the LRR probes' standard deviation along each segment.
- estimCopy : a copy number estimation, given the expected values for copy = 0,...,n.

```
> cgh <- segmentCGH(cgh, nCores=1)

Computing LRR segmentation using UndoSD: 0.179

Merging segments shorter than 10Kb.

Number of segments:  25

> segTable <- getSegTable(cgh)
```

```
> head(segTable)

           ID chrom loc.start   loc.end num.mark seg.mean seg.med probes.Sd
1 CSc.Example     1    882803 249116709     1209   0.0087 -0.0504 0.9799602
2 CSc.Example     2     15703 242497851     1317   0.8874  0.8791 0.9901649
3 CSc.Example     3     62614 197683938     1100   0.8791  0.8791 0.9786349
4 CSc.Example     4     46691 190921709     1042  -0.0075 -0.0504 0.9883702
5 CSc.Example     5    113577 180579439      986   0.8502  0.8791 0.9907562
6 CSc.Example     6    184719 170849100     1103  -0.0105 -0.0504 1.0052332
  estimCopy
1         2
2         4
3         4
4         2
5         4
6         2
```

Note that such data format allows GISTIC-compatible inputs to be exported [6].

4.4 Centering LRR

Centering LRR is a key step in the genomic analysis process since it defines the base line (the expected 2-copies level) from where gains ad losses are estimated. To do so, LRRs are considered as a mixture of several gaussian populations , and an expectation-maximization (EM) algorithm is used to estimate their parameters.
In order to increase the EM efficacy, we use the segmentation results, and model the LRR distributions, given each segment mean and sd (estimated from probes assigned to each given segment).
The centralization value is chosen according to the user specification: the mean of the sub-population with a density peak higher than a given proportion of the highest density peak [7]. The default value is 0.5. Setting `peakThresh = 1` leads to choose the highest density peak.

The `plotDensity()` function gives access to a graphical check on how the centralization step worked, and what LRR population has been chosen for centering the profile:

```
> cgh <- EMnormalize(cgh)

Merging peaks closer than 0.1 ...

Gaussian mixture estimation:

n.peaks =  3


Group parameters:

Grp 1:
prop:  0.504, mean:  -0.061, Sd:  0.149, peak height:  1.344

Grp 2:
prop:  0.481, mean:  0.861, Sd:  0.149, peak height:  1.284

Grp 3:
prop:  0.015, mean:  2.04, Sd:  0.149, peak height:  0.041


Correction value:   -0.061

Use plotDensity() to visualize the LRR densities.
```

```
> plotDensity(cgh)
```

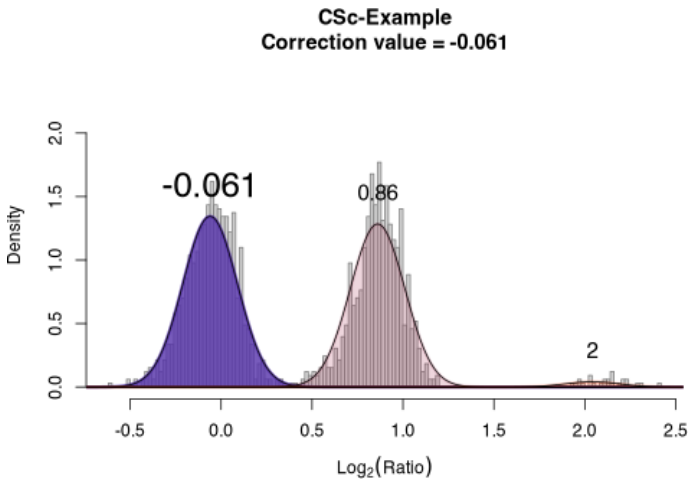

Figure 1: plotDensity. `plotDensity()` shows how *EM* models the *LRR* distribution, and what peak is chosen for centralizing the profile (in bold).

4.5 Parallelization

rCGH allows parallelization within `EMnormalise()` and `segmentCGH()`, through `mclapply()` from R package *parallel*.
By default, `nCores` will be set to half of the available cores, but any value, from 1 to `detectCores()`, is allowed. However, this feature is currently only available on `Linux` and `OSX`: `nCores` will be automatically set to 1 when a `Windows` system is detected.

4.6 Getting the by-gene table

This step converts a segmentation table into a by-genes table. `byGeneTable()` extracts the list of genes included in each segment, and constructs a dataset, easy to export and to manipulate outside R. The final genes' list reports the corresponding segmentation values (expressed in Log2Ratio), and the official positions and annotations, with respect to the genome build specified by the user. As for the `read` functions, the supported genome builds are hg18, hg19 (default) and hg38. For hg19, locations and annotations are exported from *TxDb.Hsapiens.UCSC.hg19.knownGene* and *org.Hs.eg.db*. The corresponding TxDb is used in case another genome build is specified with the *genome* argument.

```
> geneTable <- byGeneTable(segTable)

  403 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.

Creating byGene table...

> head(geneTable, n=3)

  entrezid   symbol                       fullName cytoband chr chrStart
1        1     A1BG         alpha-1-B glycoprotein 19q13.43  19 58858172
2   503538 A1BG-AS1           A1BG antisense RNA 1 19q13.43  19 58859117
3    29974     A1CF APOBEC1 complementation factor 10q11.23  10 52559169
    chrEnd width strand Log2Ratio num.mark segNum segLength(kb) estimCopy
1 58874214 16043      -   0.80185      231     21      58810.89         4
2 58866549  7433      +   0.80185      231     21      58810.89         4
3 52645435 86267      -   0.94135      751     10     135239.66         4
  relativeLog genomeStart
```

```
1           0  2718302494
2           0  2718303439
3           0  1732932312
```

Notice that the `byGeneTable()` function takes a segmentation table as its first argument, and not a `rCGH` object. This means that this function can be used to extract genes from any other segmentation table, provided this table is of the same format, and the genome build to use is specified (default setting is *"hg19"*).

```
> byGeneTable(segTable, "erbb2", genome = "hg19")[,1:6]

  403 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.

  symbol entrezid                         fullName cytoband chr chrStart
1  ERBB2     2064 erb-b2 receptor tyrosine kinase 2    17q12  17 37844393

> byGeneTable(segTable, "erbb2", genome = "hg18")[,1:6]

  379 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.

  symbol entrezid                         fullName cytoband chr chrStart
1  ERBB2     2064 erb-b2 receptor tyrosine kinase 2    17q12  17 35097919
```

4.7 Accessing the analysis parameters

For traceability and reproducibility, it may be useful to keep track to a profile analysis parameters. At each step, the workflow parameters, defined by default or specified by the user, are stored in a `params` slot. They are accessible at any time using `getParam()`.

```
> getParam(cgh)[1:3]

$ksmooth
[1] 39
```

```
$Kmax
[1] 20

$Nmin
[1] 160
```

4.8 Visualizing the genomic profile

In a context of Precision Medicine, visualizing and manipulating a genomic profile is crucial to interpret imbalances, to identify targetable genes, and to make decisions regarding a potential therapeutic orientation. In many situations, considering LOH can also help to better interpret imbalances.
rCGH provides 2 ways for visualizing a genomic profile: `plotProfile()`, `plotLOH()` and `multiplot()` are simple static ways to visualize a profile, possibly with some tagged gene, while `view()` is a more sophisticated and interactive visualization method, build on top of shiny. A control panel allows the user to interact with the profile, and to export the results.
Notice that `plotLOH()` and `multiplot()` are relevant only in case the allelic difference is available, namely when Affymetrix cnchp.txt files are used.

4.8.1 Static profile visualizations

`plotProfile()` allows the genomic profile visualization. Any gene(s) of interest can be added to the plot by passing a valid HUGO symbol. Other arguments can be used to color the segments according to specified gain/loss thresholds, or to change the plot title.
Two other static functions can be useful for reporting alterations: `plotLOH()` to visualize LOH, and `multiplot()` to build a full report, including both the genomic profile and LOH plot.

Comment: Notice that genes will be located with respect to the genome build version stored in the `rCGH` object. See subsection 4.1 for details.

Comment: By default, multiplot() will combine all the visualizations available: profile by LRR, profile by copy numbers, and B-allele differences. The `p` argument, which specifies the proportion of each plot within the layout, can be used to remove the 2nd and/or the 3rd plot from the output, e.g. `p = c(1/2, 0, 1/2)` would remove the profile by copy numbers.

```
> multiplot(cgh, symbol = c("egfr", "erbb2"))
```

```
  403 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.
403 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.
```

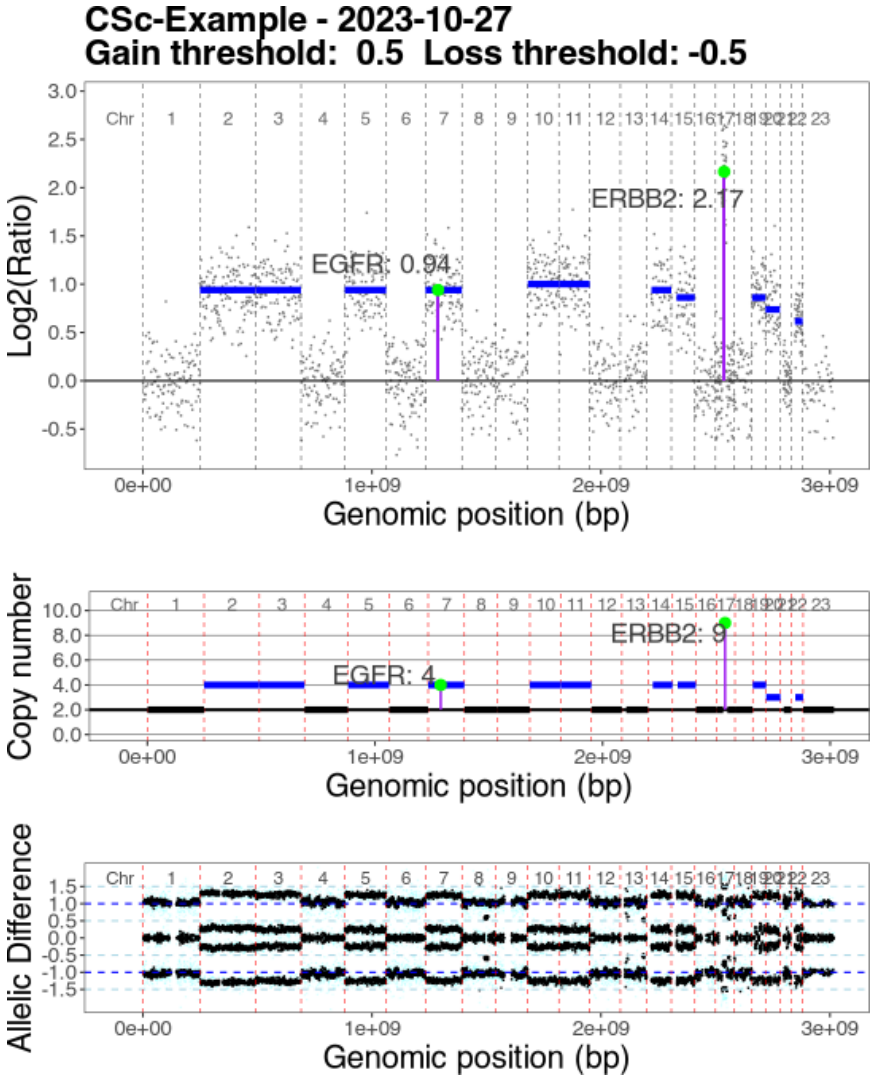

Figure 2: Static views. `multiplot()` provides static visualisations combining the genomic profile and the LOH.

4.8.2 Recentering

When the profile centering doesn't seem appropriate, `recenter()` allows the user to choose another centralization value. The new choice has to be specified as the peak index to use: peaks are indexed, from 1 to k (from left to right) as they appear on the density plot.

```
> # Recentering on peak #2
> recenter(cgh) <- 2

Profile recentered on:  0.861
```

```
> plotProfile(cgh, symbol = c("egfr", "erbb2"))

  403 genes were dropped because they have exons located on both strands
of the same reference sequence or on more than one reference sequence,
so cannot be represented by a single genomic range.
Use 'single.strand.genes.only=FALSE' to get all the genes in a
GRangesList object, or use suppressMessages() to suppress this message.
```

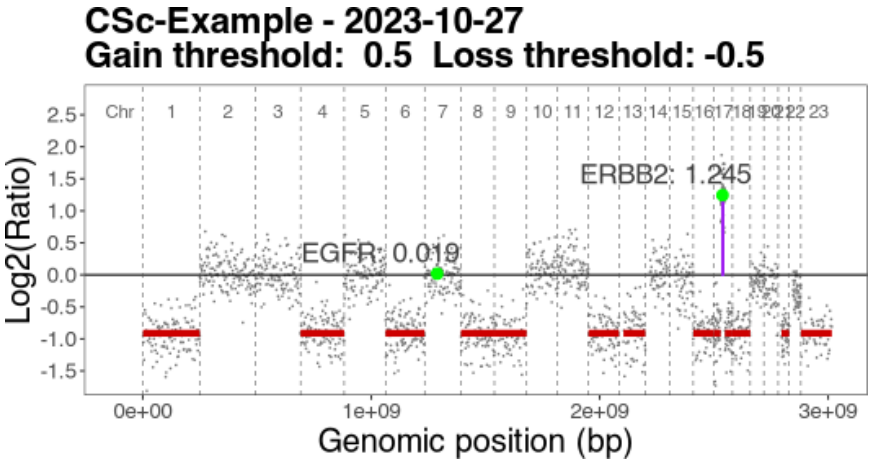

Figure 3: Recentering. By default, the EM-based normalization choose a possibly optimal peak to center the profile, but any other peak can be chosen, using `recenter()`.

4.8.3 Interactive visualization

The *view()* function provides a more flexible way for interpreting genomic profile, individually. This application allows interactive manipulations through a command panel: defining the gain/loss thresholds, displaying a gene, resizing the y-axis, selecting one unique chromosome, and recentering the entire profile. Note that the *Genes table* is updated whenever changes are made through that command panel, e.g. selecting one unique chromosome on the graph filters the *Genes table* on that chromosome, simultaneously.
The Download buttons, *Plot*, *LOH* and *Table*, allow plots and gene table to be exported, as they have been modified.

Comment: Notice that genes will be located according to the genome build value stored in the `rCGH` *object. This value has to be specified when a file is read. See subsection 4.1 for details.*

The `view()` control panel:

- Gene Symbol : display any existing gene, providing its official HUGO symbol.
- Show chromosome : display the entire profile (default is 'All'), or one specific chromosome.
- Gain/Loss colors : choose blue/red or red/blue.
- Recenter profile : recenter the profile on-the-fly. Gene values are updated in the 'Genes table'.
- Merge segments... : merge segments shorter than the specified value, in Kb. Gene values are updated in the 'Genes table'.
- Recenter profile : recenter the profile on-the-fly. Gene values are updated in the 'Genes table'.
- Rescale max(y) : adjust the top y-axis ($0<y$) using a proportion of the maximum value.
- Rescale min(y) : adjust the bottom y-axis ($y<0$) using a proportion of the minimum value.
- Gain threshold (Log2ratio) : define the gain threshold. Segments higher than this value are colored according to the chosen color code, and the 'Genes table' is filtered, consequently.
- Loss threshold (Log2ratio) : same as 'Gain threshold' but for losses.
- Download - Profile : download the profile as it is displayed on the screen, including modifications.

- Download - LOH : download the LOH plot as it is displayed on the screen, including modifications.
- Download - Table : download the 'Genes table', including modifications.

```
> view(cgh)
```

Figure 4: Interactive profile. The genomic profile is displayed in the first *CGH profile* tab (left). Several changes can be applied using the control panel (in blue). The list of genes is accessible through the *Genes table* tab (right). Both are updated simultaneously and can be exported, after modifications are applied.

5 Notes regarding the example files

In order to reduce the computation time, we provide subsets of real data for the 3 supported platforms:

```
> list.files(system.file("extdata", package = "rCGH"))

[1] "Affy_cytoScan.cyhd.CN5.CNCHP.txt.bz2"
[2] "Affy_snp6_cnchp.txt.bz2"
[3] "Agilent4x180K.txt.bz2"
[4] "generic.txt.bz2"
[5] "oncoscan.tsv.bz2"
```

Comment:
In order to speed up demos, the provided example files contain only a subset of the original probes.
Affymetrix example files (cytoScan and SNP6) only contain SNP probes. Setting `useProbes = "cn"` *in* `readAffy` *functions should return an error.*

6 Server version

A web browser version of the interactive visualization is available at
https://fredcommo.shinyapps.io/aCGH_viewer
As inputs, this application support the *rCGH* segmentation tables, or any segmentation table in the same format as the *CBS* outputs.
For more details about this application, or to install it on your own server, please visit
https://github.com/fredcommo/aCGH_viewer.

7 Session information

```
> sessionInfo()

R version 4.3.1 (2023-06-16)
Platform: aarch64-apple-darwin20 (64-bit)
Running under: macOS Ventura 13.6.1

Matrix products: default
BLAS:   /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRblas.0.dylib
LAPACK: /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/lib/libRlapack.dylib;
```

rCGH package

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

time zone: America/New_York
tzcode source: internal

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

other attached packages:
[1] GenomicRanges_1.54.0 S4Vectors_0.40.1     IRanges_2.36.0
[4] rCGH_1.32.0          knitr_1.43

loaded via a namespace (and not attached):
  [1] DBI_1.1.3
  [2] bitops_1.0-7
  [3] biomaRt_2.58.0
  [4] rlang_1.1.1
  [5] magrittr_2.0.3
  [6] matrixStats_1.0.0
  [7] compiler_4.3.1
  [8] RSQLite_2.3.1
  [9] GenomicFeatures_1.54.0
 [10] png_0.1-8
 [11] vctrs_0.6.3
 [12] stringr_1.5.0
 [13] pkgconfig_2.0.3
 [14] crayon_1.5.2
 [15] fastmap_1.1.1
 [16] magick_2.7.4
 [17] dbplyr_2.3.3
 [18] XVector_0.42.0
 [19] ellipsis_0.3.2
 [20] labeling_0.4.2
 [21] utf8_1.2.3
 [22] Rsamtools_2.18.0
 [23] promises_1.2.0.1
 [24] rmarkdown_2.23
 [25] preprocessCore_1.64.0
 [26] bit_4.0.5
 [27] xfun_0.39
 [28] zlibbioc_1.48.0
```

```
       [29] cachem_1.0.8
       [30] GenomeInfoDb_1.38.0
       [31] progress_1.2.2
       [32] blob_1.2.4
       [33] highr_0.10
       [34] later_1.3.1
       [35] DelayedArray_0.28.0
       [36] BiocParallel_1.36.0
       [37] cluster_2.1.4
       [38] parallel_4.3.1
       [39] prettyunits_1.1.1
       [40] R6_2.5.1
       [41] stringi_1.7.12
       [42] limma_3.58.0
       [43] rtracklayer_1.62.0
       [44] DNAcopy_1.76.0
       [45] Rcpp_1.0.11
       [46] SummarizedExperiment_1.32.0
       [47] splines_4.3.1
       [48] httpuv_1.6.11
       [49] Matrix_1.6-0
       [50] tidyselect_1.2.0
       [51] abind_1.4-5
       [52] yaml_2.3.7
       [53] codetools_0.2-19
       [54] affy_1.80.0
       [55] curl_5.0.1
       [56] lattice_0.21-8
       [57] tibble_3.2.1
       [58] plyr_1.8.8
       [59] withr_2.5.0
       [60] Biobase_2.62.0
       [61] shiny_1.7.4.1
       [62] KEGGREST_1.42.0
       [63] evaluate_0.21
       [64] survival_3.5-5
       [65] BiocFileCache_2.10.1
       [66] xml2_1.3.5
       [67] mclust_6.0.0
       [68] Biostrings_2.70.1
       [69] pillar_1.9.0
       [70] affyio_1.72.0
```

```
 [71] BiocManager_1.30.22
 [72] filelock_1.0.2
 [73] MatrixGenerics_1.14.0
 [74] TxDb.Hsapiens.UCSC.hg19.knownGene_3.2.2
 [75] stats4_4.3.1
 [76] generics_0.1.3
 [77] RCurl_1.98-1.12
 [78] hms_1.1.3
 [79] ggplot2_3.4.2
 [80] munsell_0.5.0
 [81] scales_1.2.1
 [82] BiocStyle_2.30.0
 [83] xtable_1.8-4
 [84] glue_1.6.2
 [85] tools_4.3.1
 [86] BiocIO_1.12.0
 [87] TxDb.Hsapiens.UCSC.hg38.knownGene_3.17.0
 [88] GenomicAlignments_1.38.0
 [89] XML_3.99-0.14
 [90] grid_4.3.1
 [91] aCGH_1.80.0
 [92] AnnotationDbi_1.64.0
 [93] colorspace_2.1-0
 [94] GenomeInfoDbData_1.2.10
 [95] restfulr_0.0.15
 [96] cli_3.6.1
 [97] rappdirs_0.3.3
 [98] fansi_1.0.4
 [99] S4Arrays_1.2.0
[100] dplyr_1.1.2
[101] gtable_0.3.3
[102] digest_0.6.33
[103] BiocGenerics_0.48.0
[104] SparseArray_1.2.0
[105] farver_2.1.1
[106] org.Hs.eg.db_3.17.0
[107] rjson_0.2.21
[108] multtest_2.58.0
[109] memoise_2.0.1
[110] htmltools_0.5.5
[111] lifecycle_1.0.3
[112] httr_1.4.6
```

```
      [113] TxDb.Hsapiens.UCSC.hg18.knownGene_3.2.2
      [114] statmod_1.5.0
      [115] mime_0.12
      [116] MASS_7.3-60
      [117] bit64_4.0.5
```

References

[1] Commo F, Guinney J, Ferte C, Bot B, Lefebvre C, Soria JC, and Andre F. rcgh : a comprehensive array-based genomic profile platform for precision medicine. *Bioinformatics*, 2015.

[2] URL: http://www.affymetrix.com/estore/partners_programs/programs/developer/tools/powertools.affx.

[3] Smyth GK and Speed TP. Normalization of cdna microarray data. *Methods*, 31:265–273, 2003. URL: http://www.statsci.org/smyth/pubs/normalize.pdf.

[4] Venkatraman ES and Olshen AB. A faster circular binary segmentation algorithm for the analysis of array cgh data. *Bioinformatics*, 15(23):657–663, 2007.

[5] Venkatraman E. Seshan and Adam Olshen. *DNAcopy: DNA copy number data analysis*. R package version 1.40.0.

[6] Mermel CH, Schumacher SE, Hill B, Meyerson ML, Beroukhim R, and Getz G. Gistic2.0 facilitates sensitive and confident localization of the targets of focal somatic copy-number alteration in human cancers. *Genome Biology*, 12(4):R41, 2011.

[7] Commo F, Ferte C, Soria JC, Friend SH, Andre F, and Guinney J. Impact of centralization on acgh-based genomic profiles for precision medicine in oncology. *Ann Oncol.*, 2014.