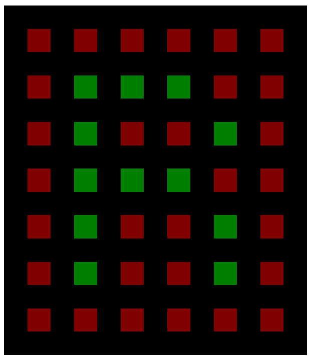
How-to guide to cghRA

Last edition February 23rd 2017

Sylvain Mareschal

http://bioinformatics.ovsa.fr/cghRA



Contents

1 Introduction

This document describes how to use the cghRA package, showing the step-by-step analysis of a few arrays available in ArrayExpress. Questions and feedback may be sent to mareschal@ovsa.fr, news and updates will be made available on the package web page.

The analysis of a batch of CGH arrays that were published in the ArrayExpress archive, under the E-MTAB-4497 accession number, will be developped in this document. While the whole dataset could be analyzed in a similar fashion, we will focus on 5 representative arrays in order to limit downloads and computation: CHB-02048, CHB-04234, CHB-05212, CHB-05967 and CHB-06016. The raw data files can be downloaded from the archive, as they were produced by the array scanner software FeatureExtraction.

For each step on the analysis, we will introduce both the command line (CLI) and the graphical user (GUI) interfaces, to highlight cghRA duality. Command line users should refer to Rgb's "How-to" vignette first, in order to get familiar with the `track.table` reference class from which most cghRA classes inherit.

1.1 Installation

cghRA has two mandatory dependencies : Rgb and DNAcopy.

Rgb is distributed through the CRAN, and thus can be installed by classical R means. More recent stable version may however be available on its dedicated webpage, and here again installed as usual (`install.packages` function or any GUI you may be used to). Finaly the latest development version can be downloaded from its GitHub page as a source package.

DNAcopy is a stand-alone part of Bioconductor. It can be installed using Bioconductor's `biocLite` helper function as described on its webpage, or downloaded and installed directly using `install.packages` or any other GUI substitute.

Finally **cghRA** can be downloaded from its webpage as a .zip (Windows) or .tar.gz (MacOS / Linux) package to install with R `install.packages`. Latest development version and sources can be found on its decicated GitHub page. cghRA may join the CRAN or Bioconductor repositories in a near future.

1.2 Quick-start

Users comfortable with CGH array analysis and computers in general may consider the following workflow to get a quick grasp of cghRA capabilities :

1. Download and uncompress the original Agilent design file for E-MTAB-4497 on the cghRA website.

2. Download original array files as produced by Agilent's FeatureExtraction software, for the 5 aforementioned samples of the E-MTAB-4497 accession, from the corresponding ArrayExpress archive.

3. Download human annotation tracks for the GRCh37 assembly (NCBI genes, UCSC cytobands and DGV supporting variants) from the Rgb website.

4. Launch R, load the cghRA package and execute the command `tk.cghRA()`.

5. In the **Design processing** panel, select the file downloaded at step 1 as "Input file", uncheck the "Remap probes" and "Compute WACA biases" boxes and hit the "Process design" blue button. More details about this panel and other formats of design files are provided in chapter 2. A log text file should open, and the computation end in a few seconds.

6. In the **Array processing** panel, select the 5 files downloaded at step 2 as "Probe files", the .design.rdt file produced in current R working directory by step 5 as "Design file", uncheck the "WACA correction" box and hit the "Process arrays" button. Here again a log file should pop up, and the computation end in less than a minute. More details about this panel and other formats of array files are provided in chapter 3.

7. In the **Copy number modelization** panel, click "Select files" to select the .regions.rdt files produced in current R working directory at step 6 and use the "Next" and "Previous" buttons to visualize their copy-number models. These models should be well fit, however feel free to move the cursors to observe how changing parameters influence the copy number modelization. Details about these parameters and the whole modeling process are provided in section 3.5 and chapter 4.

8. In the **Annotate regions** panel, click "Track(s) to process" to select the .copies.rdt files produced in current R working directory at step 6 and hit the "Export to CSV" blue button. For each selected file, there should now be a .csv file located in the same folder that can be browsed with Excel or any other spread-sheet editor. More details about this step can be found in section 5.4.

9. In the same **Annotate regions** panel, select the "GRCh37.UCSC_bands.rdt" file downloaded at step 3 as "Annotation track", select "cytoband" as "Crossing type" and hit the "Compute annotation" blue button. Perform again step 8 to observe the new "UCSC bands.cytoband" column. More details about this step can be found in section 5.1.

10. In the same **Annotate regions** panel, select the "GRCh37.NCBI_genes.rdt" file downloaded at step 3 as "Annotation track", select "name" as "Crossing type" and hit the "Compute annotation" blue button. Perform again step 8 to observe the new "NCBI genes.name" column. More details about this step can be found in section 5.2.

11. In the same **Annotate regions** panel, select the .design.rdt file produced at step 5 as "Design file", the "GRCh37.DGV_supp.rdt" file downloaded at step 3 as "Polymorphism track", uncheck the "Filter" box and hit the "Compute CNV score" blue button. Perform again step 8 to observe the new "cnvScore" column. More details about polymorphism filtering can be found in chapter 5.3. Feel free to check the "Filter" box and perform this step again to see the difference.

12. In the **Series processing** panel, enter "cghRA vignette" as "Series name", select the .copies.rdt files produced at step 6 as "Region files" and hit the first blue "Compute" button. Feel free to observe the PNG image files that were produced, the rest of the files will be visualized later. Further details about all these files can be found in section 6.1.

13. In the same **Series processing** panel, hit the first "Compute" blue button on "STEPS" row. The resulting files will be visualized later, further details about them can be found in section 6.2.

14. Close cghRA window and execute the command `tk.browse()` to use **Rgb** in interactive mode.

15. Click the "Track" button, then "Add from file" and select the following files : .design.rdt produced at step 5, the set of .probes.rdt, .regions.rdt and .copies.rdt produced at step 6 for sample "CHB-05212", "GRCh37.NCBI_genes.rdt" and "GRCh37.UCSC_bands.rdt" files downloaded at step 3. Then hit "Done".

16. Type "CDKN2A" in the text field right to the "Find" button, select "NCBI genes" in the nearby menu and hit the "Find" button. The selected tracks should appear in the main window, around the locus of the CDKN2A gene which is frequently deleted in lymphomas. You can use the arrow keys to move around this locus, or refer to chapter 7 for further details about Rgb usage.

17. Click the "Track" button again and add penetrance and pool files produced at step 13. You may want to hide some of the previous tracks checking the "Hide" boxes, to keep them readable on small screens. Once back to the main window, hit the "Jump" button to refresh the plot.

18. Repeat the previous operation with the "STEPS" file produced at step 14.

1.3 Reference

Mareschal S, Ruminy P, Alcantara M, Villenet C, Figeac M, Dubois S, Bertrand P, Bouzelfen A, Viailly PJ, Penther D, Tilly H, Bastard C, Jardin F. **Application of the cghRA framework to the genomic characterization of Diffuse Large B-Cell Lymphoma.**
Article in submission

2 Design processing

There are many types of commercial CGH arrays, and all differ in their **design**, i.e. the amount of probes spotted on the slide and the genomic regions they monitor. Each manufacturer provides information on these designs under various formats, that need to be converted by cghRA into a format suitable for computation. cghRA offers several tools to enhance this information, as we will see.

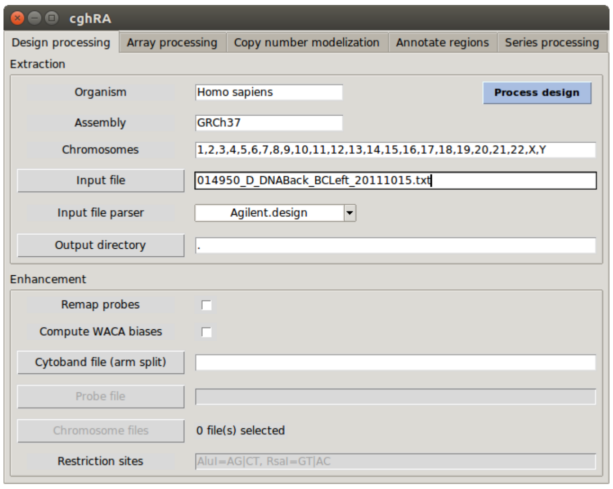
To obtain such a RDT design file, one can use the graphical interface or R command lines. Using the graphical interface consists mainly in filling the form in the "Design processing" panel and press the blue "Process design" button. Once cghRA is launched, a text log file is filled continuously until the end or a blocking error is met, and should be open automatically. If any error is encountered, one can correct the form and press the button again to restart the analysis from the beginning.

2.1 Importing an Agilent design

As cghRA was developed using Agilent CGH arrays, several features facilitate their use with this software. The manufacturer provides TDT text files describing the array design, that are handled in cghRA by the `Agilent.design()` function. For the example we are developing, the GRCh37 TDT file can be downloaded (and unzipped) from cghRA website and selected using the "Input file" button. Organism and assembly names should also be provided, to track assembly inconsistencies between files during the analysis. The chromosome list is also optional, but providing it allows to enforce the order of their display (the default alphabetical order is a bit messy for human chromosomes).

```
>   # Parse an Agilent TDT file
>   design <- Agilent.design(
+     file = "014950_D_DNABack_BCLeft_20111015.txt",
+     organism = "Human",
+     assembly = "GRCh37",
+     chromosomes = c(1:22, "X", "Y")
+   )

>   # Export as a cghRA-compliant file
>   saveRDT(design, file="design.rdt")
```

The same features are proposed in the "Extraction" frame on the "Design processing" tab :



Extraction

Organism	Homo sapiens
Assembly	GRCh37
Chromosomes	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y
Input file	014950_D_DNABack_BCLeft_20111015.txt
Input file parser	Agilent.design
Output directory	.

Enhancement

Remap probes	☐
Compute WACA biases	☐
Cytoband file (arm split)	
Probe file	
Chromosome files	0 file(s) selected
Restriction sites	AluI=AG|CT, RsaI=GT|AC

2.2 Importing a custom design

As a TDT file may not always be available, a few manual steps may be required to convert a random annotation file to something cghRA can handle. This can be achieved in R directly, taking advantages of its powerful table-handling capabilities, or using a spread-sheet editor. To illustrate custom design import, we will use the E-MTAB-4497 design as it is provided by ArrayExpress (hg18 ADF file) rather than from the Agilent TDT file.

2.2.1 Graphical interface

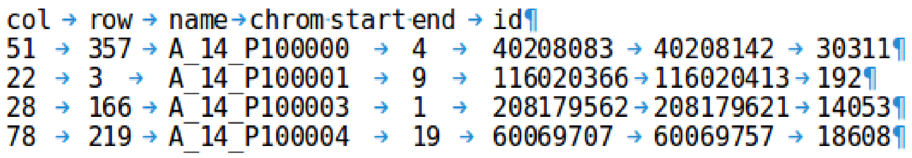
Aside Agilent TDT files, the graphical interface is able to import any text file, as long as it provides the requested columns (at least 'chrom', 'start' and 'end', preferably 'strand', 'name' and 'id' as well) and respect the expected format (columns separated by tabulations, periods as decimal separators, first row as column headers). Thus it is possible to edit the ADF file we use as an example using a spread-sheet editor, to enforce these few constraints.

Notice that the 'id' column must be filled with caution, as it is used to match rows between design and probe files. Agilent files provide it in both files as "FeatureNum" columns, but it may be required to build them with other array types. In our case, we will rebuild it for the design and let cghRA reads the "FeatureNum" column from probe files to match with. "FeatureNum" IDs are attributed to probes from the left to the right, and then from the top to the bottom.

For this example, we will have to :

- Remove unnecessary header lines at the beginning of the file.
- Split 'Reporter Database Entry[chromosome_coordinate:unknown]' into 3 columns named 'chrom', 'start' and 'end'.
- Remove 'chr' from the chromosome names (optional).
- Rename 'Column', 'Row' and 'Reporter Name' into 'col', 'row' and 'name' respectively.
- Create a new 'id' column, filled with the formula `(row - 1L) * max(col) + col`. This is a quite generic function that can be used on any array providing row and column numbers for each probe.
- Remove or rename additional columns ('Block Column', 'Block Row', 'Reporter Database Entry[hugo]', 'Reporter Database Entry[refseq]', 'Reporter Database Entry[embl]', 'Reporter Group[role]', 'Control Type').
- Export the table using the 'CSV' format, with tabulations as column separators.

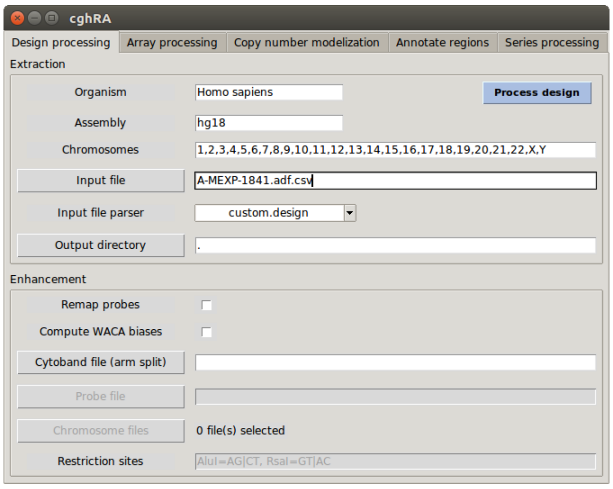
Here are the first rows of the resulting file, to illustrate the expected output. Blues arrows represent tabulations and paragraph marks a line break (any of usual Windows, Linux or Mac styles).



```
col → row → name→chrom·start·end → id¶
51  → 357 → A_14_P100000  → 4  →  40208083 → 40208142 → 30311¶
22  → 3   → A_14_P100001  → 9  →  116020366→116020413→192¶
28  → 166 → A_14_P100003  → 1  →  208179562→208179621→14053¶
78  → 219 → A_14_P100004  → 19 →  60069707 → 60069757 → 18608¶
```

Once the file ready it can be handled by cghRA, the only difference with Agilent TDT files being that the file will be processed by the `custom.design()` function rather than `Agilent.design()`. This implies that it can be imported using R command lines as follows, or that the "Input file parser" parameter in the graphical interface must be switched accordingly.

```
>   # Parse a manually edited ADF file
>   design <- custom.design(
+     file = "A-MEXP-1841.adf.csv",
+     organism = "Human",
+     assembly = "hg18",
+     chromosomes = c(1:22, "X", "Y")
+   )

>   # Export as a cghRA-compliant file
>   saveRDT(design, file="design.rdt")
```



Extraction

Organism	Homo sapiens
Assembly	hg18
Chromosomes	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y
Input file	A-MEXP-1841.adf.csv
Input file parser	custom.design
Output directory	.

Enhancement

Remap probes	☐
Compute WACA biases	☐
Cytoband file (arm split)	
Probe file	
Chromosome files	0 file(s) selected
Restriction sites	AluI=AG|CT, RsaI=GT|AC

2.2.2 R command lines

Within R, the custom design file should be imported as a classic R `data.frame`, and appropriate columns must be passed to the `cghRA.design` class constructor (see this function's help page for further details). All the steps manually performed in the previous section can easily be automatized in R :

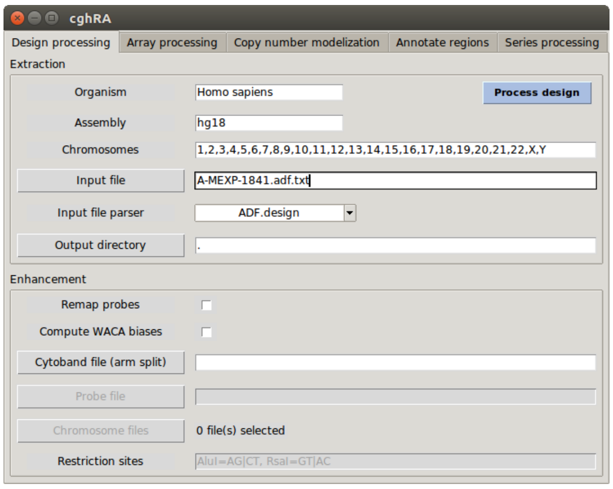
```
>   # Import file
>   rawDesign <- read.table(
+     file="A-MEXP-1841.adf.txt", sep="\t", skip=18,
+     header=TRUE, quote=NULL, stringsAsFactors=FALSE
+   )

>   # Split coordinates
>   coords <- rawDesign$"Reporter.Database.Entry.chromosome_coordinate.unknown."
>   rawDesign$chrom <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\1", coords)
>   rawDesign$start <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\2", coords)
>   rawDesign$end <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\3", coords)

>   # Rebuild FeatureNum
>   row <- rawDesign$Row
>   col <- rawDesign$Column
>   rawDesign$FeatureNum <- (row - 1L) * max(col) + col

>   # Build design
>   design <- cghRA.design(
+     name = rawDesign$Reporter.Name,
+     chrom = rawDesign$chrom,
+     strand = NA,
+     start = as.integer(rawDesign$start),
+     end = as.integer(rawDesign$end),
+     id = rawDesign$FeatureNum,
+     row = rawDesign$Row,
+     col = rawDesign$Column,
+     .name = "Agilent 014950",
+     .organism = "Human",
+     .assembly = "hg18",
+     .chromosomes = c(1:22, "X", "Y")
+   )

>   # Export as a cghRA-compliant file
>   saveRDT(design, file="design.rdt")
```

2.2.3 Extend cghRA capabilities

Additionally, the user can define its own parsing functions to make cghRA able to directly read files from other manufacturers using the graphical interface. The `Agilent.design()` and `custom.design()` functions defined in cghRA are good examples of the function to define in the R global environment before cghRA is launched. Once this done, the user can simply write in the "Input file parser" field of the graphical interface the name of the R function to use for reading the provided design file. Note however that the function will have to be redefined in each session, as it will be lost at R closure.

As an example, here is a custom parsing function embedding the R commands we used previously to parse the ArrayExpress ADF file. Notice the fixed arguments, that will be filled by the graphical interface. Additional arguments can also be set for command line usage only, but default values will be required for the graphical interface to work.

```
>   ADF.design <- function(
+     file,
+     name = "Custom ADF file",
+     organism = as.character(NA),
+     assembly = as.character(NA),
+     chromosomes = NULL,
+     ...
+   ) {
+     # Import file
+     rawDesign <- read.table(
+       file=file, sep="\t", skip=18,
+       header=TRUE, quote=NULL, stringsAsFactors=FALSE
+     )
+
+     # Split coordinates
+     coords <- rawDesign$"Reporter.Database.Entry.chromosome_coordinate.unknown."
+     rawDesign$chrom <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\1", coords)
+     rawDesign$start <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\2", coords)
+     rawDesign$end <- sub("^chr(.+):([0-9]+)-([0-9]+)$", "\\3", coords)
+
+     # Rebuild FeatureNum
+     row <- rawDesign$Row
+     col <- rawDesign$Column
+     rawDesign$FeatureNum <- (row - 1L) * max(col) + col
+
+     # Build design
+     design <- cghRA.design(
+       name = rawDesign$Reporter.Name,
+       chrom = rawDesign$chrom,
+       strand = NA,
+       start = as.integer(rawDesign$start),
+       end = as.integer(rawDesign$end),
+       id = rawDesign$FeatureNum,
+       row = rawDesign$Row,
+       col = rawDesign$Column,
+       .name = name,
+       .organism = organism,
+       .assembly = assembly,
+       .chromosomes = chromosomes
+     )
+
+     return(design)
+   }
```



Extraction

Organism: Homo sapiens
Assembly: hg18
Chromosomes: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y
Input file: A-MEXP-1841.adf.txt
Input file parser: ADF.design
Output directory: .

Enhancement

Remap probes: ☐
Compute WACA biases: ☐
Cytoband file (arm split):
Probe file:
Chromosome files: 0 file(s) selected
Restriction sites: AluI=AG|CT, RsaI=GT|AC

Of course, R command line users should use this custom function directly to obtain design objects :

```
>   design <- ADF.design(
+     file = "A-MEXP-1841.adf.txt",
+     name = "Agilent 014950",
+     organism = "Human",
+     assembly = "hg18",
+     chromosomes = c(1:22, "X", "Y")
+   )
```

2.3 Remapping a design

As the assembly of well-studied genomes such as the human one evolves on a regular basis, you may want to work with a specific assembly for which design files are not provided. cghRA can be used to remap the probes to the assembly of your choice, using fast alignment software, through the GUI or directly using R commands.

2.3.1 Both methods : BLAT setup

The first step is to download the BLAT executable from its author website, in a version that matches your operating system (Windows, Mac OS, Linux...). It is usually packaged as a "blatSuite" archive, the only file we will use in it this archive is the "blat" / "blat.exe" executable file.

On Windows, you will also require Cygwin's "cygwin1.dll" file, that you can obtain by freely installing the whole software. Alternatively you may extract it from a compressed archive provided by mirror sites like this one (64 bits) or this one (32 bits), usually in the /usr/bin directory.

Once you have collected the required files, execute the following command (update the file names and paths according to your downloads) :

```
>   # On Windows
>   blatInstall(blat="blat.exe", cygwin="cygwin1.dll")

>   # On Linux or MacOS
>   blatInstall(blat="blat")
```

2.3.2 Both methods : required files

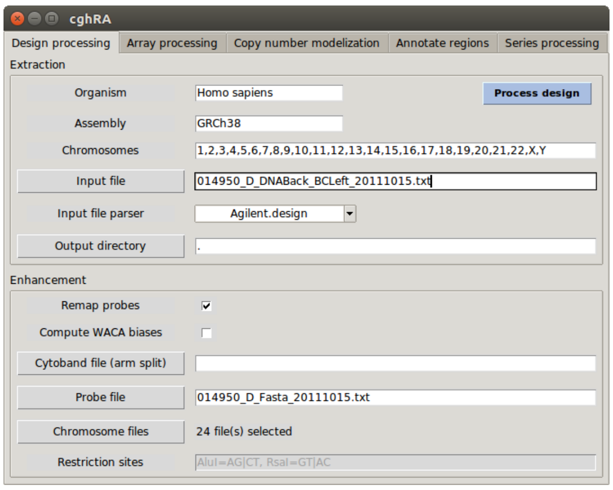
You will also require the sequence of the studied organism chromosomes, as separated (uncompressed) FASTA files. For human and numerous model organism genomes, they can be downloaded from the University of California Santa Cruz (UCSC) repository, e.g. here for the hg38 / GRCh38 assembly of the human genome. Files should be renamed to discard the 'chr' prefix, and alternative versions or unmapped chromosome files should be discarded as well.

You will finally require the sequences of the probes, as a single multi-FASTA file with probe names as comments. While Agilent provides this file, you may be required to rebuild it in other cases (ArrayExpress' ADF file may or may not contain a sequence column). For the example we are developing, the multi-FASTA probe file can be downloaded (and unzipped) from cghRA website.

2.3.3 Graphical interface

To apply the remapping through cghRA GUI, just tick the "Remap probes" checkbox during the design processing, and provide the files discussed above. Select the multi-FASTA probe file using the "Probe file" button, and select all the chromosome FASTA files using the "Chromosome Files" button. Finally remeber to update the "Assembly" field with the remapped assembly name, not the assembly of the original file.

The processing can take several minutes, even a few hours for a high-resolution array. A "design.log" file is regularly populated as the processing goes on, you can monitor progress and errors in it.



Extraction

Organism: Homo sapiens
Assembly: GRCh38
Chromosomes: 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y
Input file: 014950_D_DNABack_BCLeft_20111015.txt
Input file parser: Agilent.design
Output directory: .

Enhancement

Remap probes: ✓
Compute WACA biases: ☐
Cytoband file (arm split):
Probe file: 014950_D_Fasta_20111015.txt
Chromosome files: 24 file(s) selected
Restriction sites: AluI=AG|CT, RsaI=GT|AC

2.3.4 R command lines

For designs built using command lines, apply the `remap()` method to the previously built `design` file. This method proposes several options to handle probes that map to distinct locations or overlap, as well as atypical chromosome file names. The example below is the simplest case in which the FASTA chromosome files were stored in the "hg38" directory and were named "chr1.fa", "chr2.fa"... See the help page of the underlying `localize()` function for further details.

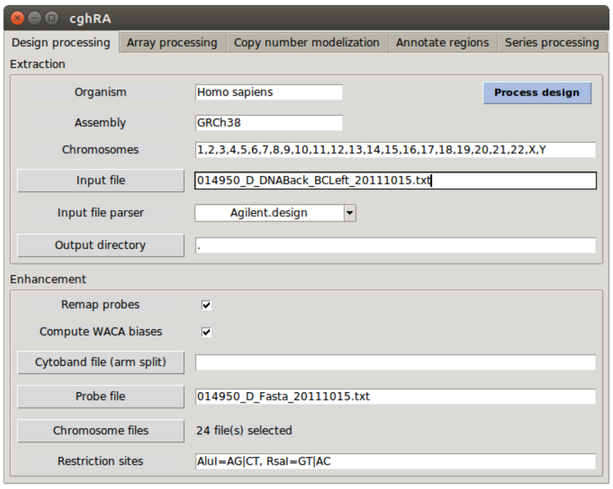
```
>   # Import the design we previously built
>   design <- readRDT(file="design.rdt")

>   # Apply remapping
>   design$remap(
+     probeFile = "014950_D_Fasta_20111015.txt",
+     chromFiles = dir("hg38", full.names=TRUE)
+   )

>   # Update assembly name
>   design$assembly <- "GRCh38"

>   # Export the remapped design
>   saveRDT(design, file="design_hg38.rdt")
```

2.4 WACA pre-computation

In order to use the WACA (Waved Array-cgh Correction Algorithm, *Leprêtre et al, Nucleic Acids Res. 2010 Apr;38(7):e94*), some precomputations must be performed at the time of the design processing. As for the remapping described above, this step will require the full chromosome sequences in individual FASTA files. You will also be required to describe the restriction enzymes used for the genomic DNA fragmentation, and more precisely the restriction sites recognized by each of them.

Graphical interface

With the GUI, simply provide the chromosome files as described previously, and the restriction sites as shown in the example.



Extraction

Organism	Homo sapiens
Assembly	GRCh38
Chromosomes	1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,X,Y
Input file	014950_D_DNABack_BCLeft_20111015.txt
Input file parser	Agilent.design
Output directory	.

Enhancement

Remap probes	✓
Compute WACA biases	✓
Cytoband file (arm split)	
Probe file	014950_D_Fasta_20111015.txt
Chromosome files	24 file(s) selected
Restriction sites	AluI=AG|CT, RsaI=GT|AC

R command lines

In the R console, apply the `bias()` method to a previously built design file, with the same arguments. Note that the chromosome files must correspond to the design assembly at the time of the `bias()` method call, so this step should be performed after the remapping if any has to occur. Here again, additional details on this method can be found in the help page of the underlying `bias()` function.

```
>   # Import the previously remapped design
>   design <- readRDT(file="design_hg38.rdt")

>   # Apply WACA bias computation
>   design$bias(
+     chromFiles = dir("hg38", full.names=TRUE),
+     fragSites = c(AluI="AG|CT", RsaI="GT|AC")
+   )

>   # Export the remapped / WACA compliant design
>   saveRDT(design, file="design_hg38_WACA.rdt")
```

2.5 Chromosome arm split

According to the user preference, cghRA may work with chromosomes separated in two arms (e.g. "1p" for the short arm of chromosome 1 and "1q" for the long arm), and can build designs that distinguish them or not. To incorporate chromosome arm information in your design, simply provide a cytoband annotation file matching the organism and assembly of your (eventually remapped) design. For the GRCh38 assembly of the human genome, this file can be downloaded directly from the Rgb website. In other cases, please refer to the help page of the `track.bands.UCSC()` function for further details.

Working with arm-split designs require arm-split annotation files as well, at most steps of the analysis. Annotation files that can be downloaded from Rgb website are not, and require to be split as show below. As most of this vignette is written with non-split designs, new users are discouraged to use chromosome arm split. Otherwise, users should be particularly attentive to error messages indicating mismatches in chromosome lists, as chromosome arms are considered as individual chromosomes after the split.

```
>   # Import a cytoband file to use for split
>   bands <- readRDT(file="GRCh37.UCSC_bands.rdt")

>   # Import an annotation file to arm-split
>   genes <- readRDT(file="GRCh37.NCBI_genes.rdt")

>   # Add arms
>   genes$addArms(bands)

>   # Save as RDT with a distinct name
>   saveRDT(genes, file="GRCh37.arm.NCBI_genes.rdt")
```

Graphical interface

With the GUI, provide the corresponding RDT file using the "Cytoband file (arm split)" button to obtain an arm-aware design, or leave it blank to obtain a design that will not differentiate them.

R command lines

In the R console, import the content of this RDT file and apply the `addArms()` method to a previously built design file. An `eraseArms()` method is also provided to perform the opposite operation.

```
>   # Import the previously remapped / WACA compliant design
>   design <- readRDT(file="design_hg38_WACA.rdt")

>   # Import the cytoband annotation
>   bands <- readRDT(file="GRCh38.UCSC_bands.rdt")

>   # Add arm information
>   design$addArms(bands)

>   # Export the remapped / WACA compliant design
>   saveRDT(design, file="design_hg38_WACA_arm.rdt")
```

3 Array processing

Once the design ready, multiple files corresponding to multiple arrays can be processed. cghRA performs this processing for each array independently from the others as a straight pipeline, each step taking as input the output of the previous one. The currently implemented steps are shortly described here-after and more extensively described in sections 3.3, 3.4 and 3.5; custom steps can be added as well (see the `process()` help page for further details). Currently the graphical interface only handles the full default pipeline, but custom pipelines can easily be built using the `process()` function.

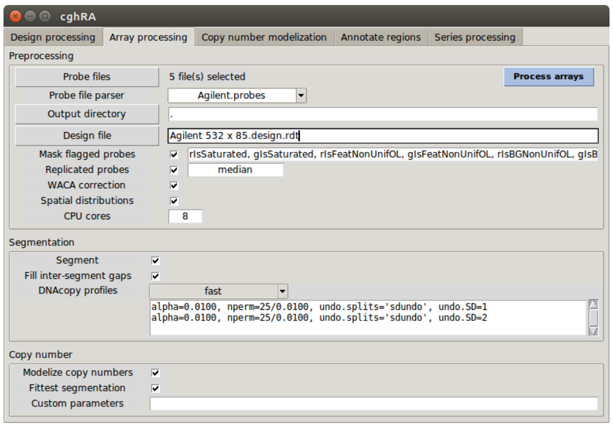
- **parse** will read data from provided files, using a parsing function. A parser for Agilent's FeatureExtraction format is provided, but an example of custom parsing will be developed here-after.
- **mask** will turn to NA the log-ratios of probes showing specified flags (saturated or control probe...).
- **replicates** will replace the log-ratio of replicated probes with a single representative value (mean, median...), and set other replicates to NA.
- **waca** will correct the log-ratios using the WACA algorithm (Waved Array-cgh Correction Algorithm, *Leprêtre et al, Nucleic Acids Res. 2010 Apr;38(7):e94*).
- **spatial** will produce a physical representation of the array, to visually identify spatial biases.
- **segment** will produce one or many genomic segmentation of the log-ratios, using CBS (see 3.4).
- **fill** will fill gaps between segments to make their coordinates perfectly jointed (see 3.4.1).
- **modelize** will apply the cghRA copy-calling model (see 3.5).
- **fittest** will select the fittest copy-calling model (see 3.5).
- **applyModel** will transform log-ratios into copy-numbers, applying the selected model.
- **export** will produce a RDT file from its input. This step may be used many times in a pipeline to save results from each other steps.

Using the graphical interface consists mainly in filling the form in the "Array processing" panel and press the blue "Process arrays" button. Arrays will be processed independently (possibly in parallel, see 3.3.5, and a text log file is filled continuously until the end or a blocking error is met. If any error is encountered, one can correct the form (possibly unselecting arrays successfully analyzed) and press the button again to restart the analysis from the beginning.

3.1 Importing Agilent arrays

To stick to the example developed in the current document, raw data files can be downloaded from the ArrayExpress archive, under the E-MTAB-4497 accession number. The CHB-02048, CHB-04234, CHB-05212, CHB-05967 and CHB-06016 samples are recommended for the alterations they present.

To analyze Agilent arrays scanned using Feature Extraction, the procedure is straight-forward: click "Probe files" to select one or many text files produced by Feature Extraction, select the "Output directory" and the design file (in RDT format, as produced using the previous panel) and click "Process arrays". Selected files must relate to a single design, but several analyses corresponding to distinct designs can be ran separately. Results from these distinct analyses can be merged at later steps, keeping in mind that distinct designs imply distinct resolutions in the discovery of chromosomal imbalances (see 6).



```
Preprocessing
  Probe files               5 file(s) selected
  Probe file parser         Agilent.probes
  Output directory          .
  Design file               Agilent 532 x 85.design.rdt
  Mask flagged probes       ✓ rIsSaturated, gIsSaturated, rIsFeatNonUnifOL, gIsFeatNonUnifOL, rIsBGNonUnifOL, gIsB
  Replicated probes         ✓ median
  WACA correction           ✓
  Spatial distributions     ✓
  CPU cores                 8
Segmentation
  Segment                   ✓
  Fill inter-segment gaps   ✓
  DNAcopy profiles          fast
                            alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=1
                            alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=2
Copy number
  Modelize copy numbers     ✓
  Fittest segmentation      ✓
  Custom parameters
```

The mouse cursor should change to a sand timer (or anything related according to the operating system) while the analysis runs, and a "process.log" file should be populated in real time with diagnostic messages. Notice that due to the multi-threading architecture, messages will only appear in the log file when the analysis of an array is complete, which can take from a few seconds (44k probe arrays) to half an hour (1M probe arrays) depending on the array size and computer speed.

3.2 Importing custom arrays

As for design files, data files from other manufacturers are not handled by cghRA in a "out of the box" fashion but can be dealt with in several ways. Array files can be reshaped manually with a spread-sheet editor or with R directly, or a new parsing function can be integrated to cghRA. To illustrate this feature, we will try to import the Agilent raw data files of the E-MTAB-4497 ArrayExpress accession without using cghRA built-in support for this format.

3.2.1 Graphical interface

Only few columns are really useful for us in the raw data files, essentially the log-ratio of fluorescences and a numeric probe ID matching the 'id' column provided during the design import (please refer to 2.2.1 for further details on the 'id' column). Any additional columns can be kept as well and won't interfere with cghRA processing.

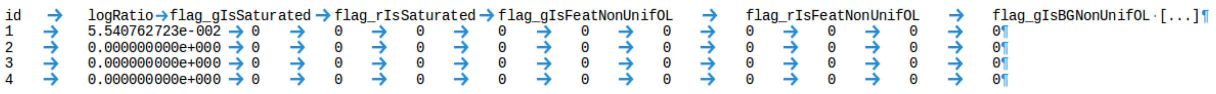
The log-ratio can be already present in the data file or computed manually from the red and green fluorescence signals using the following formula : `logRatio = log2(G/R)`, where G and R refer to the green (sample) and red (reference) fluorescence respectively. If dyes were swapped, always use the color corresponding to the (supposed normal) reference sample as the denominator of the ratio. The column name must be 'logRatio' to be recognized by cghRA.

cghRA also handles 'flags', which are usually quality-check outputs formatted as a single logical value for each probe. These columns can be included in the custom file, and referred to later for probe filtering (probes with a flag set to TRUE will be discarded during this step). Please refer to 3.3.1 for further details on this optional step of the cghRA pipeline.

In the current example, the following steps will be required to convert the Agilent TXT file into the custom file format handled by cghRA (CSV-like file using tabulations as column separators, periods as decimal separators and the first row as column headers) :

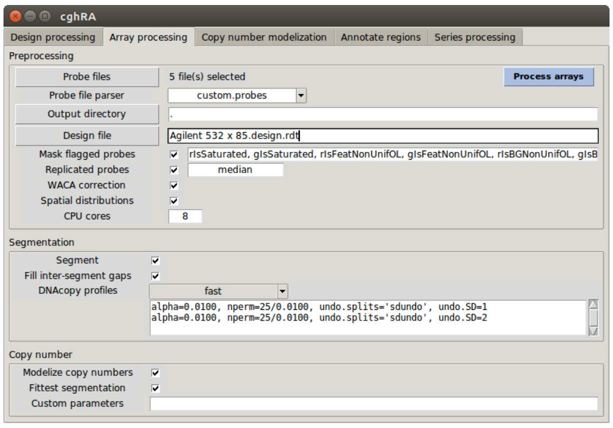
- Remove the first lines containing two small tables on top of the main table.
- Remove the first header line with column types.
- Keep only columns B ('FeatureNum'), P ('LogRatio'), BA ('gIsSaturated'), BB ('rIsSaturated'), and BE ('gIsFeatNonUnifOL') to BL ('rIsBGPopnOL').
- Rename 'FeatureNum' into 'id' and 'LogRatio' into 'logRatio' (beware of the case !).
- Add 'flag_' in front of the names of all the flag columns to use for probe filtering.
- Export the table using the 'CSV' format, with tabulations as column separators.

Here are the first rows of the resulting file, to illustrate the expected output. Blues arrows represent tabulations and paragraph marks represent line breaks (any of usual Windows, Linux or Mac styles). Notice the first line is truncated on the following image (truncation is represented by the '[...]' sign), but column names are similar until the end of the line.



```
id → logRatio→flag_gIsSaturated→flag_rIsSaturated→flag_gIsFeatNonUnifOL → flag_rIsFeatNonUnifOL → flag_gIsBGNonUnifOL·[...]¶
1  → 5.540762723e-002 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0¶
2  → 0.000000000e+000 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0¶
3  → 0.000000000e+000 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0¶
4  → 0.000000000e+000 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0 → 0¶
```

Once the file ready, it can be handled by cghRA as Agilent data files, the only difference being that the file will be processed by the `custom.probes()` function rather than `Agilent.probes()`. This implies that the "Probe file parser" parameter in the graphical interface must be switched accordingly.



Preprocessing

5 file(s) selected

Probe file parser: custom.probes

Output directory: .

Design file: Agilent 532 x 85.design.rdt

Mask flagged probes: ✓ rIsSaturated, gIsSaturated, rIsFeatNonUnifOL, gIsFeatNonUnifOL, rIsBGNonUnifOL, gIsB

Replicated probes: ✓ median

WACA correction: ✓

Spatial distributions: ✓

CPU cores: 8

Segmentation

Segment: ✓

Fill inter-segment gaps: ✓

DNAcopy profiles: fast

```
alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=1
alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=2
```

Copy number

Modelize copy numbers: ✓

Fittest segmentation: ✓

Custom parameters:

3.2.2 R command lines

Within R, the custom data file should be imported as a classic R `data.frame`, and appropriate columns must be passed to the `cghRA.probes` class constructor (see this function's help page for further details). All the steps manually performed in the previous section can easily be automatized in R :

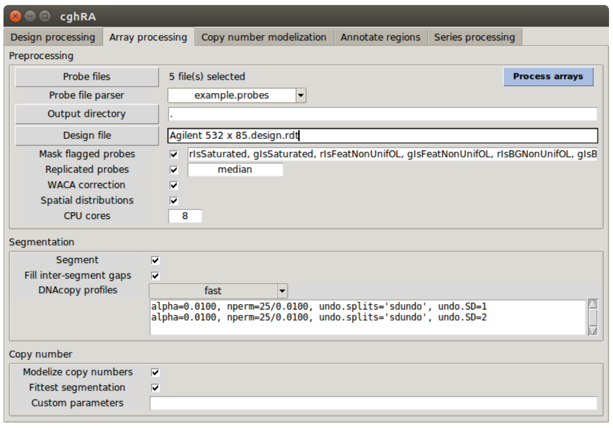
```
>   # Import file
>   rawFile <- read.table(
+     file="DLBCL_Feb2016_CHB02048.txt", sep="\t", skip=9,
+     header=TRUE, quote=NULL, stringsAsFactors=FALSE
+   )

>   # Select columns
>   rawFile <- rawFile[, c(2, 16, 53, 54, 57:64) ]

>   # Rename columns
>   colnames(rawFile)[ colnames(rawFile) == "FeatureNum" ] <- "id"
>   colnames(rawFile)[ colnames(rawFile) == "LogRatio" ] <- "logRatio"
>   colnames(rawFile)[3:12] <- sprintf("flag_%s", colnames(rawFile)[3:12])

>   # Build probes
>   probes <- cghRA.probes(
+     rawFile,
+     .name = "CHB02048"
+   )

>   # Export as a cghRA-compliant file
>   saveRDT(probes, file="DLBCL_Feb2016_CHB02048.probes.rdt")
```

3.2.3 Extend cghRA capabilities

Additionally, the user can define its own parsing functions to make cghRA able to directly read files from other manufacturers using the graphical interface. The `Agilent.probes()` and `custom.probes()` functions defined in cghRA are good examples of the function to define in the R global environment before cghRA is launched. Once this done, the user can simply write in the "Probe file parser" parameter the name of the R function to use for reading the provided design file. Note however that the function will have to be redefined in each session, as it will be lost at R closure.

As an example, here is a custom parsing function embedding the R commands we used previously to parse the Agilent data file in a custom way. Notice the fixed arguments, that will be filled by the graphical interface. Additional arguments can also be set for command line usage only, but default values will be required for the graphical interface to work.

```
>   example.probes <- function(
+     file,
+     columns = NULL,
+     ...
+   ) {
+     # Import file
+     rawFile <- read.table(
+       file=file, sep="\t", skip=9,
+       header=TRUE, quote=NULL, stringsAsFactors=FALSE
+     )
+
+     # Select columns
+     rawFile <- rawFile[, c(2, 16, 53, 54, 57:64) ]
+
+     # Rename columns
+     colnames(rawFile)[ colnames(rawFile) == "FeatureNum" ] <- "id"
+     colnames(rawFile)[ colnames(rawFile) == "LogRatio" ] <- "logRatio"
+
+     # Rename only flag columns mentioned in the interface
+     if(length(columns) > 0L) {
+       for(i in 1:length(columns)) {
+         colnames(dat)[ colnames(dat) == columns[i] ] <- names(columns)[i]
+       }
+     }
+
+     # Build probes
+     probes <- cghRA.probes(
+       rawFile,
+       .name = basename(file)
+     )
+
+     return(probes)
+   }
```



Preprocessing

Probe files	5 file(s) selected
Probe file parser	example.probes
Output directory	.
Design file	Agilent 532 x 85.design.rdt
Mask flagged probes	✓ rIsSaturated, gIsSaturated, rIsFeatNonUnifOL, gIsFeatNonUnifOL, rIsBGNonUnifOL, gIsB
Replicated probes	✓ median
WACA correction	✓
Spatial distributions	✓
CPU cores	8

Segmentation

Segment	✓
Fill inter-segment gaps	✓
DNAcopy profiles	fast

```
alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=1
alpha=0.0100, nperm=25/0.0100, undo.splits='sdundo', undo.SD=2
```

Copy number

Modelize copy numbers	✓
Fittest segmentation	✓
Custom parameters	

Command line users can consider to pass this function to `process` via the `probeParser` argument, which will be used at the "parse" step.

3.3 Preprocessing steps

3.3.1 Mask flagged probes

As many array scanning software, Feature Extraction populates several "flags" for each probe, in order to identify probes of poor quality. This ranges from a saturated channel to heterogeneous intensity of the marking, and the full description of these flags can be found in the Feature Extraction Reference Guide provided by the manufacturer. The standard behavior of cghRA is to import these flags as logical values appending "flag_" at the beginning of their names, and to filter out (turn log-ratio to `NA`) any probe that shows any of these flags. The list of the flags to consider is present in the interface, as a comma-separated list of column names.

3.3.2 Replicated probes

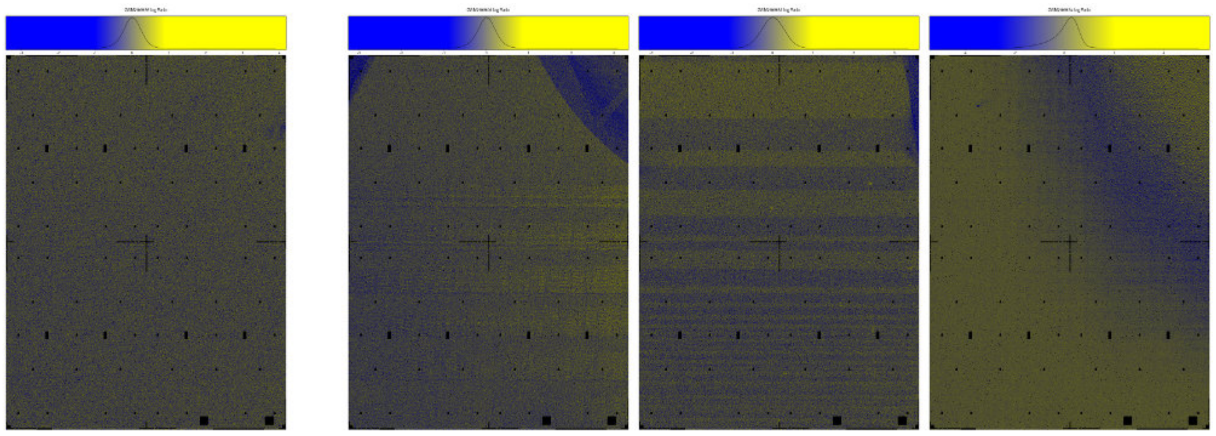
Several array designs include replicated probes, i.e. probes with identical nucleotidic sequence spotted at distinct locations on the array. Such replicated signals are useful to compute the technical noise or identify spatial artifacts, but may introduce biases during the segmentation step. cghRA proposes to replace these multiple replicated values (as identified by their identical probe name) by a single representative one, turning log-ratios of the other group members to `NA`. This representative value is computed from the values of all the replicates, using the function named here in the interface. This must be the name of a R function (typically "mean" or "median", but custom functions available in the R global environment may also be used) that accepts a single numeric vector as argument and return a numeric scalar.

3.3.3 WACA correction

Checking this box will introduce a step of wave correction, using a new implementation of the WACA algorithm (Waved Array-cgh Correction Algorithm, *Leprêtre et al, Nucleic Acids Res. 2010 Apr;38(7):e94*). Notice that this requires specific cares during the design file production, as mentionned earlier (see 2.4).

3.3.4 Spatial distributions

In most CGH array designs, there is no correlation at all between the physical location of the probe on the array and its genomic location. This fact proves particularly useful to detect hybridization problems, which lead to an unexpected correlation between signal intensity and physical position. Checking this box triggers the production of "spatial plots", on which each probe of the array is represented by a distinct pixel at its physical location, with a color depending on the probe log-ratio. Examples of a clean sample (on the left) and various artifacts that can be expected are illustrated below.



As genomically consecutive probes are far from each other, such artifacts should not lead to the observance of false segments, but rather contribute to the experiment noise. This can lead to poor copy-calling models, and the simplest way to deal with such arrays is to discard them.

Users willing to get more information and tools to deal with this phenomenon can refer to *Neuvial et al, BMC Bioinformatics. 2006 May 22;7:264* and the corresponding R package MANOR, available from Bioconductor. cghRA currently does not handle this correction itself, but data can easily be exported from and to the `cghRA.probes` and `cghRA.array` classes by users with sufficient R CLI skills to operate MANOR.

3.3.5 CPU cores

Most computers have several processors nowadays, and cghRA can take benefit from this to run faster. As each array is analyzed independently from the others, many arrays can be analyzed simultaneously by distinct processors of the same machine, using the "parallel" R package. This is what is proposed here, the user being required to choose how many arrays should be analyzed in the same time. The default value is populated by the "parallel" R package as the amount of CPUs detected on the machine, but can freely be increased or decreased. Notice that increasing it beyond the CPU count is very likely to slow the computation rather than speeding it.

3.4 Segmentation

As usual in CGH-array analysis, the second step once the probe signals are filtered and normalized is to segment the genome into regions of homogeneous log-ratios. This is performed in cghRA by the CBS algorithm, as implemented in the DNAcopy package (Venkatraman and Olshen, Bioinformatics 2007). The resulting segments will be exported in ".regions.rdt" files, that will be used for down-stream analysis.

3.4.1 Fill gaps

As CBS starts and ends segments on a probe, there is usually a short gap of undetermined copy number between segments. These gaps can disrupt the down-stream analysis based on penetrance, and it is generally preferable to ask cghRA to fill these gaps. This filling is performed by extending each segment on its right until it reaches the start of the following segment. In the graphical interface, this feature can be disabled by unchecking the corresponding box. Using R command lines, one can simply remove the "fill" step from the "steps" argument of `process()`.

3.4.2 Segmentation parameters

CBS has numerous arguments with strong influence on the results for which optimal values can be hard to predict. To minimize this problem, cghRA is able to perform several segmentations with distinct parameters and select the copy number model which seems to fit best to the data. For further details on the model and the selection criterion, one can refer to the publication associated with cghRA (reference at the beginning of this file). Of course multiple segmentations imply more computation, and a good balance needs to be found between the precision of the model and the time spent in computation.

In the graphical interface, the user can switch between the various segmentation profiles proposed by cghRA ('accurate', 'fast' and 'fastest'). A 'CBS default' profile is also provided, which consists in a single segmentation using the default parameters in the DNAcopy package. Finally a 'custom' profile is available for the user to define how many segmentations should be tried and with which arguments. Each line in the text area beneath the profile selection will define the arguments to be passed to the `segment()` function of the DNAcopy package, as an R expression (see the existing profiles for examples).

Using R command lines, segmentation profiles are passed via the "segmentArgs" argument of `process()`. The profiles proposed in the interface can be obtained using the `process.default` function, as shown below. For further details on the usage of `process()`, one can refer to its R manual page.

```
>   process.default("segmentArgs", "fastest")
```

3.5 Copy number modelization

Once the segmentation done, cghRA will apply its original model to infer copy numbers from the average log-ratio of each segment. Here again, one can refer to the cghRA publication mentioned at the beginning of this tutorial for theoretical aspects of this model.

When multiple segmentations are performed (see 3.4), each of them is modelized independently from the others. Checking the "Fittest segmentation" box in the graphical interface or using the "fittest" step in the `process()` function will ask cghRA to select the best one from these. Each segmentation attempt produces a ".regions.rdt" file numbered according to its order in the segmentation profile, which will be updated during modelization. When cghRA is asked for a selection, the best one will simply be copied without numbering.

As an example, if a sample named "Sample1" is processed with the "accurate" segmentation profile, 10 files named "Sample1#1.regions.rdt", "Sample1#2.regions.rdt" etc will be produced. If cghRA is asked to select the fittest segmentation and the third one proves to be the best according to its criterion, "Sample1#3.regions.rdt" will be copied as "Sample1.regions.rdt", which is the only one that should be considered for down-stream analysis.

Finally cghRA proposes default values for all the modelization parameters, but the user is free to update some of them (see 4 for an interactive way to change these parameters and observe how the model is affected). In the graphical interface, the user can define arguments for the `model.auto()` function in the "Custom parameters" text area. Name / value pairs are simply to be written here separated by commas, as in any R function call. Parameters not defined there will keep their default values, so only parameters to change need to be mentioned. Using R command lines, the user can provide the same string as in the graphical interface via the "modelizeArgs" argument of the `process()` function.

4 Model visualization and tuning

The cghRA copy number model assumes that genomic segments have actually very few distinct average log-ratio values, and that these values follow a simple distribution depending essentially on the tumoral burden of the sample. An appropriate plot of a sample segments make this assumption very easy to check, and cghRA proposes an interactive way to use this representation to adjust the model parameters in real time.

4.1 R command lines

In R, this representation is produced by the `model.test()` function. While the function is generic enough to be used with any data structure, the cghRA classes makes it easier to use via a method. Here is an example, assuming that the array file was preprocessed, segmented and modelized as explained in 3.

```
>   regions <- readRDT(file="DLBCL_Feb2016_CHB02048.regions.rdt")
>   regions$model.test()
```

The `model.test()` function will plot the model stored in the file, but can also use an alternative model defined via its arguments (notably "width" and "center"). To automatically fit a new model, one can use the `model.auto()` function and method in the same way, eventually providing distinct constraints to the fitting via its arguments.

```
>   regions$model.auto()
>   regions$model.auto(minWidth=0.5, maxWidth=0.7)
```

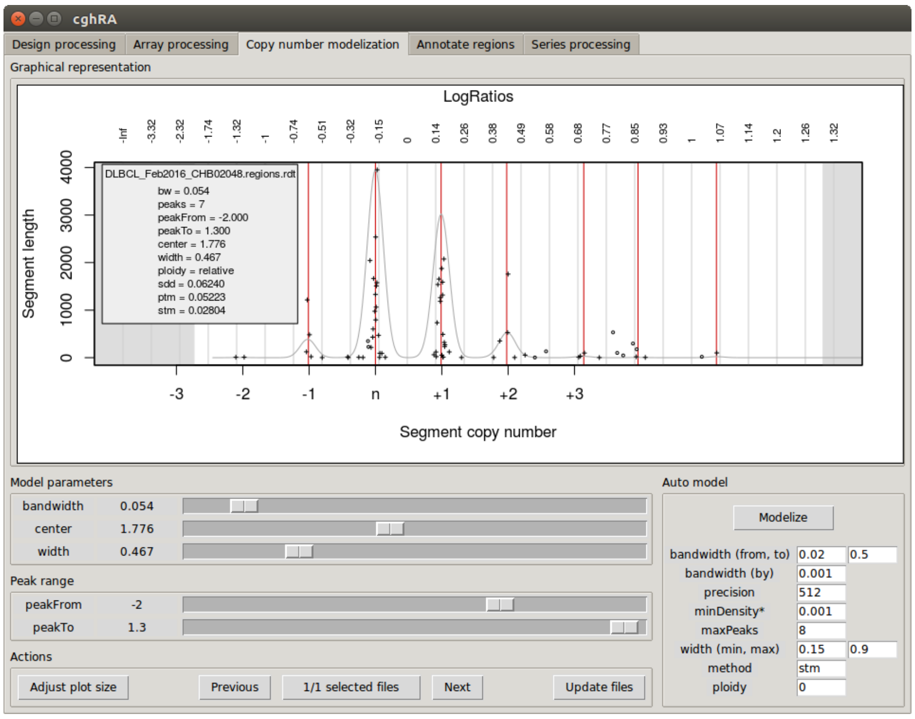
Notice that the interactive interface can be summoned at any time in R command lines, just calling the `tk.modelize()` function.

4.2 Graphical interface

The "Copy number modelization" panel of the graphical interface allows to visualize and assess the modelization of a sample, and update its parameters if the automatic one is not satisfactory. To begin, click the "Select files" button and select one or many ".regions.rdt" files produced during the segmentation (see chapter 3).

You will then be able to switch between the selected files using "Previous" and "Next". Don't forget however to save your changes to the model using the "Update files" button before switching, else they will be lost.

A click on a point of the plot will moreover summon a small popup window, in which you will find various information about the segment it represents (size, genomic location, log-ratio...).



DLBCL_Feb2016_CHB02048.regions.rdt
bw = 0.054
peaks = 7
peakFrom = -2.000
peakTo = 1.300
center = 1.776
width = 0.467
ploidy = relative
sdd = 0.06240
ptm = 0.05223
stm = 0.02804

bandwidth	0.054
center	1.776
width	0.467
peakFrom	-2
peakTo	1.3

bandwidth (from, to)	0.02	0.5
bandwidth (by)	0.001	
precision	512	
minDensity*	0.001	
maxPeaks	8	
width (min, max)	0.15	0.9
method	stm	
ploidy	0	

4.2.1 Adjust plot size

cghRA will try its best to adjust automatically the size of the plot (delimited by a white background) to the size of the bordered frame containing it, each time the plot is refreshed. This is particularly important for the mouse clicks to be correctly mapped to the points which were clicked at.

However on Windows it seems that the magnifying factor that can be set on high resolution screens to keep readable texts and icons is not handled properly by the `tkrplot` package on which cghRA relies. If you experience such a case where the plot is constantly smaller than its enclosing frame, click the "Adjust plot size" button. You will be asked for an expansion factor that will be applied to the plot size, and that should correspond to the magnifying factor set in Windows settings (150% in Windows = 1.5 in cghRA). Once the correct value found, notice that you can launch `tk.modelize` and `tk.cghRA` directly with the correct setting, filling the `tkrplot.scale` argument.

This bug should not be present on Linux and MacOS, and should disappear from Windows starting with R version 3.4.0. In these cases, the rendering is no longer performed by `tkrplot` but using PNG images instead. Here the "Adjust plot size" button will only refresh the plot, which can still be useful to make it fill the window again after manually resizing it.

4.2.2 Keys to plot interpretation

The plot corresponding to the first selected file should appear in the main frame. In this plot, each genomic segment of the sample is represented by a cross, with the X axis describing its average log-ratio (top axis) or modelized copy number (bottom axis) and the Y axis describing its size (in consecutive probe count). Notice that segments located on sexual chromosomes are plotted with circles rather than crosses, highlighting the fact that they are not considered in the analysis.

cghRA will estimate a density function from this segment distribution, drawn with a solid grey line. In a sample with at least a few genomic alterations, this distribution should display multiple peaks regularly spaced, each peak grouping segments with similar log-ratios and probably similar copy numbers.

If this distribution shows peaks, they will be detected by cghRA and marked with vertical red bars. The median distance between two consecutive peaks define the "width" parameter of the model, while the highest peak is used to define the "center" of the model (assuming that the most frequent copy number is of known ploidy, usually 2n).

cghRA computes several scores able to quantify the model fitness to the data. Currently only STM is actively used during modelization (and fully described in the publication associated with cghRA), other scores should be disregarded. The smaller STM is for a sample, the more precise is the model.

To conclude, a successful CGH experiment should show multiple regularly spaced groups of points, each materialized by a peak in the distribution and a vertical red bar. Overlap between point groups will hamper the modelization, and can occur for the following main reasons :

- **high technical noise / poor DNA quality** : peaks are very diffuse and wide, meaning that regions with same copy numbers show poorly reproducible log-ratios.
- **high normal cell contamination of the sample** : peaks are very close to each others, meaning that log-ratios show little variation on the array.
- **poor clonality of the sample** : some peaks correspond to the copy number states in one clone and some other peaks to the copy number states in another. cghRA assumes that the sample is (mostly) clonal and will focus on the most represented one, but can fail if secondary clones are significantly represented.

4.2.3 Model parameters

Beside the plot, the graphical interface offers several slide buttons that can be used to modify the parameters of the model, i.e. the bottom axis linking log-ratios to copy-numbers. Modifying one of these parameters will instantly update the plot, so the cursors can be slid around to find the best value.

Bandwidth controls the width of the peaks in the density function estimator (grey line). As only the maximum of the peak is used by cghRA this parameter has limited impact, except from the cases where some peaks are very close to each others and the user thinks they should be merged. A large bandwidth will merge peaks more easily, and only suit samples with high tumoral cell content.

Center defines the shift between the theoretical normal state (log-ratio = 0) and what is observed in the data, assuming that the highest peak (most frequent copy number in the genome) corresponds to this normal state. Larger values shift this state toward higher log-ratios, while smaller values shift it toward lower and possibly negative log-ratios.

Width defines the distance between consecutive peaks, assuming that most copy number states are represented in the data (consecutive peaks should differ of one copy number exactly). This parameter is the hardest to define, especially when subclones are present in the sample or if the sample shows very few genomic alterations. In the later case (essentially one narrow peak is visible), cghRA will probably fail at proposing a model, and a large width value (typically 1) should be used to consider most segments as unaltered.

PeakFrom and **PeakTo** controls the window in which the model is computed (displayed by a gray shading of the excluded intervals). Peaks with extreme log-ratios are usually describing very high or very low copy numbers, rarely found in the sample genome and thus associated with higher noise. Limiting the model to most recurrent peaks (in most cases -1, n and +1 are enough) will render the model more stable, and actually lead to better estimates of the copy number of these extreme regions.

4.2.4 Automatized modeling

The graphical interface also offers a simple way to fit automatically a model to the data, as can be performed during the array processing step (3). Here you will however be able to change the constraints to apply to the fitting and visualize immediately how it improves or not the modelization of the data. All the parameters proposed in the graphical interface are directly linked to arguments for the `model.auto()` R function, however you will find here after a quick description of each of them.

Bandwidth from, **to** and **by** describe the values allowed for the **bandwidth** parameter of the model (see 4.2.3). cghRA will try all possible values between 'from' and 'to', using a step of 'by', in order to find the best value for this parameter. In some rare cases cghRA will prefer bandwidths resulting in peaks too wide or narrow according to what is suspected of the tumoral burden of the sample, and changing the extremes allows to force it to find an optimal solution closer to your expectations. The bandwidth of the model can also be forced to a specific value just defining 'from' and 'to' to the expected value.

Precision sets the amount of computed points in the density estimation (grey line). Higher values would imply higher resolution in the drawing of this curve and the detection of the peak maxima, however this has usually little consequence on the resulting model.

MinDensity sets the minimal height for a density peak (grey line) to be detected (red vertical bar), as a proportion of the highest peak in the sample. The more copy number states will be detected, the more accurate the model can be. It is thus important that small peaks which seems to be regularly spaced from reliable copy number states are detected, and very small spurious peaks resulting from a single segment with unusual log-ratios are not.

MaxPeaks sets the maximum amount of peaks (copy number states) allowed in a model to be retained. Models resulting in more than this number of peaks will be silently discarded, however considering that extreme copy number states are usually poorly represented in the genome and thus not detected as peaks, the default value is rarely reached, except in subclonal or poor quality samples. Notice that the minimal amount of peaks is 2, considering that the "width" parameter of the model is computed as the difference between consecutive peaks.

Width min and **max** sets the boundaries of allowed values fr the **width** parameter of the model (see 4.2.3). Models in which the median shift between consecutive detected peaks fall outside of these boundaries will be silently discarded. Please remind that this parameter can be interpreted as an estimate of the tumoral burden, thus pure tumoral samples like cell-line derived samples (expected width of 1) or samples containing very high amounts of normal cells may require to tune these parameters. The default values were optimized to what was encountered during cghRA development on Diffuse Large B-Cell Lymphoma biopsies in our institution.

Method names the score to minimize for model selection. The "STM" score should always be used, other scores are provided for historical reasons. More details about these scores can be found in the R manual page of the `model.auto()` R function.

Ploidy defines the copy number to attribute to the highest peak in the distribution, i.e. the most common copy number state in the genome. Other copy number states are then assigned according to their distance to this supposedly known copy number. This value can be inferred using "wet" lab methods or assumed from what is known of the pathology (tumors with few genomic alterations will have a ploidy of 2 in most cases). If one prefer to not assume anything about the tumor ploidy, a value of 0 can be used to obtain relative rather than absolute copy numbers. This value is simply a fixed value added to copy number estimates, and won't impact the fitting of the model itself.

5 Annotate regions

Once amplified and deleted regions have been discovered, analyzing them usually require to look at how they intersect with genomic annotation, such as genes, polymorphisms or cytogenetic banding. This can be achieved by adding columns to regions files, as described in this chapter, or visually using the R genome browser packed with cghRA (see 7). The "Annotate regions" panel of the graphical interface provides an interactive way to produce such additional columns, which can also be obtained using R commands and methods. While both are generic enough to work with any pair of Rgb-compliant track files, we will illustrate this annotating "regions" files produced by cghRA in chapter 3.

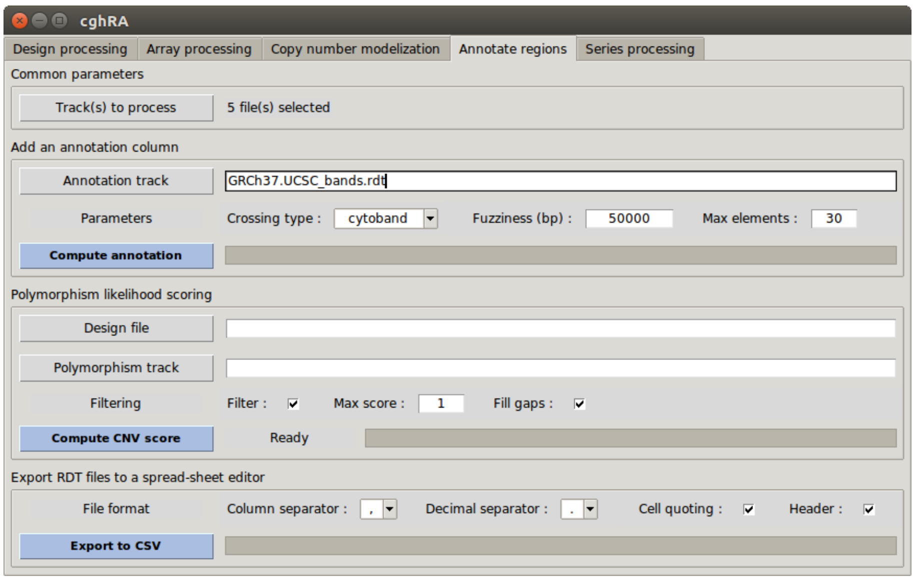
5.1 Cytogenetic coordinates

A first annotation of interest would be to obtain cytogenetic coordinates of the amplified and deleted regions, which may be easier to picture than raw numeric coordinates. For this purpose we will require a CNV banding annotation track, as produced using Rgb `track.bands()` or `track.bands.UCSC()` functions (please refer to Rgb documentation for further details). Such a track for the human genome can be downloaded from Rgb website, for GRCh37 and GRCh38 assemblies.

Graphical interface

To perform the annotation, select the RDT files to annotate using the "Track(s) to process" button and the cytoband file using the "Annotation track" button. Then change the "Crossing type" to "cytoband", which only applies in this specific case, and press "Compute annotation". In this particular case, the two other parameters won't have any effect on the result.

Files will be processed one by one, the new column being added in the same RDT file. The name of the new column will be generated combining the name of the annotation track and the name of the crossing type, which should be in our case "UCSC bands.cytoband".



5 file(s) selected

GRCh37.UCSC_bands.rdt

Crossing type : cytoband Fuzziness (bp) : 50000 Max elements : 30

Filter : ✓ Max score : 1 Fill gaps : ✓

Ready

Column separator : , Decimal separator : . Cell quoting : ✓ Header : ✓

To observe the results, the RDT files can be converted to CSV and read with a spread-sheet editor, as described in section 5.4.

R command lines

The same result can be obtained using the `cross()` method of the `track.table` class :

```
>   # Import files to process
>   bands <- readRDT("GRCh37.UCSC_bands.rdt")
>   regions <- readRDT("DLBCL_Feb2016_CHB02048.copies.rdt")

>   # Before annotation
>   print(regions$extract(1:3))
>   # Annotate
>   regions$cross(
+     annotation = bands,
+     colname = "UCSC banding",
+     type = "cytoband"
+   )

>   # After annotation
>   print(regions$extract(1:3))
```

Notice the `colname` argument allows to set the name of the newly produced column to any custom value. Alternatively, `colname` can be set to `NULL` to produce a vector that will be returned by the method rather than stored in the object.

5.2 Gene list

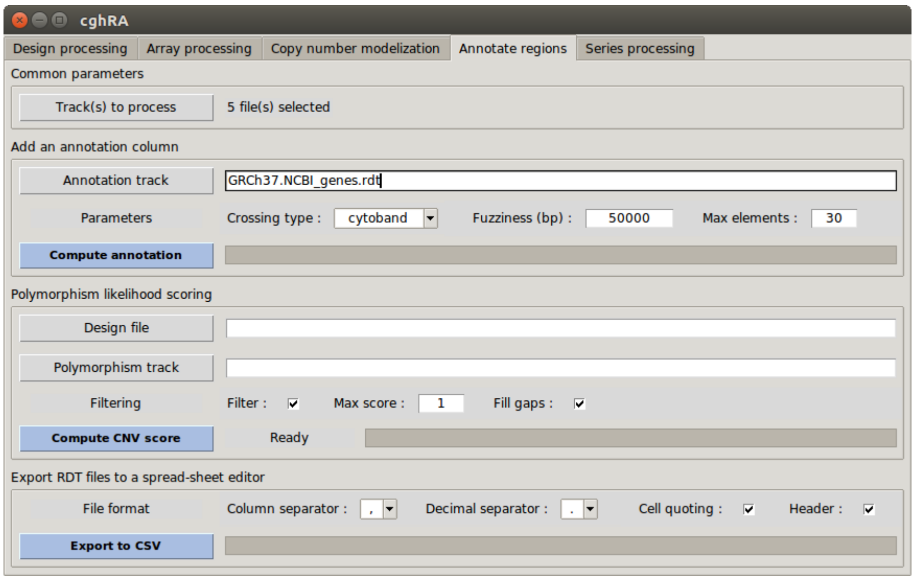
Listing the genes located in a region could also prove to be very usefull when assessing CGH array results. Data to perform such an annotation can be produced using `track.genes()` or `track.genes.NCBI()` Rgb functions (please refer to Rgb documentation for further details). The human genes with HGNC symbols for the human genome can be found on Rgb website as well, for GRCh37 and GRCh38 assemblies.

Graphical interface

Select the RDT files to annotate using the "Track(s) to process" button and the gene file using the "Annotation track" button. This time the "Crossing type" will be different : actually you can name here a column whose value should be listed for each overlap found. In this case, you can type "name" in the field to list HGNC symbols or "GeneID" to list numeric NCBI Gene IDs instead. This behavior can be generalized to any column available in the annotation track.

Considering that regions can widely vary in size, the two other parameters can be helping. The "Fuzziness" describes how regions to annotate will be extended on both sides before looking for overlaps, in order to list genes nearby but not strictly overlapping the annotated region. The "Max elements" value avoids to produce huge files listing hundreds or thousands of genes for large regions. When this value is exceeded for a region, only the amount of overlaps will be reported rather than the complete list of symbols or IDs.

Once the parameters set to your convenience, hit the "Compute annotation" button. The selected RDT files should be updated with a new column, whose name corresponds to the name of the annotation track and the crossing type. To visualize this result, please refer to section 5.4.



5 file(s) selected

GRCh37.NCBI_genes.rdt

Crossing type : cytoband Fuzziness (bp) : 50000 Max elements : 30

Filter : ✓ Max score : 1 Fill gaps : ✓

Ready

Column separator : , Decimal separator : . Cell quoting : ✓ Header : ✓

R command lines

The same result can be obtained using the `cross()` method of the `track.table` class :

```
>   # Import files to process
>   genes <- readRDT("GRCh37.NCBI_genes.rdt")
>   regions <- readRDT("DLBCL_Feb2016_CHB02048.copies.rdt")

>   # Before annotation
>   print(regions$extract(1:3))

>   # Annotate using HGNC symbols
>   regions$cross(
+     annotation = genes,
+     colname = "NCBI gene symbols",
+     type = "name"
+   )

>   # Annotate using GeneID
>   regions$cross(
+     annotation = genes,
+     colname = "NCBI gene IDs",
+     type = "GeneID"
+   )

>   # After annotation
>   print(regions$extract(1:3))
```

5.3 Polymorphism likelihood (cnvScore)

When CGH is performed using a pool of normal DNA as control rather than germline DNA from matching patients, amplified and deleted regions can be specific to the tumor (somatic) or only a reflect of the patient's genome uniqueness (polymorphism). As only somatic events are usually targeted when performing CGH on tumoral DNA, cghRA provides an original algorithm to classify regions as somatic or polymorphic, using the Database of Genomic Variants. More details about this algorithm can be found in the article related to cghRA, cited in the beginning of this document.

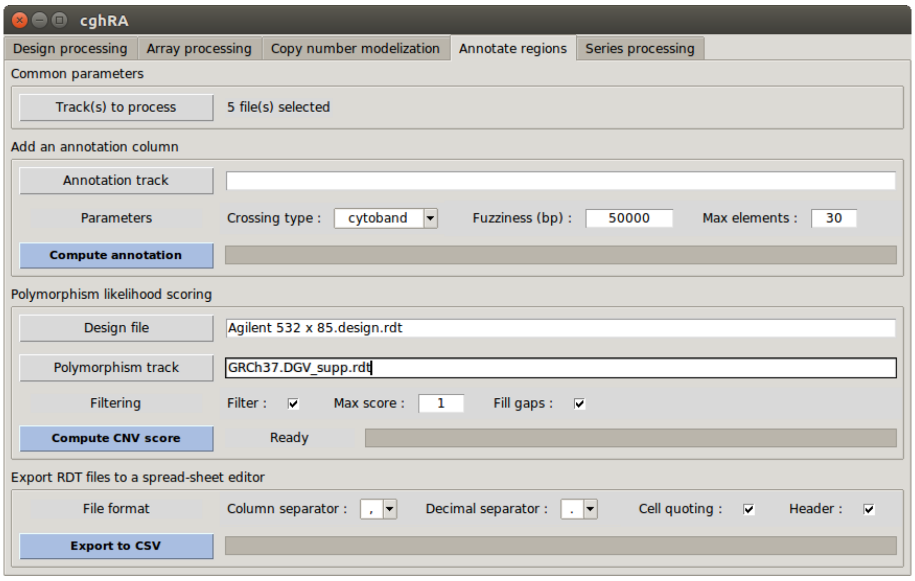
To compute such a score for segments obtained in a CGH experiment, one first needs to prepare a bank of polymorphisms to compare segments to. Any list of amplified or deleted regions found in a cohort of healthy individuals can be used as a polymorphism bank, however cghRA proposes the `track.CNV.DGVsupp()` function to parse files downloaded from the Database of Genomic Variants. The resulting track file can be downloaded from Rgb website, for the GRCh37 and GRCh38 assemblies.

Graphical interface

The first step will consist in remapping the polymorphism bank to the CGH array design used in our experiment. Indeed depending on the resolution of the array and the location of the probes (which tend to avoid highly polymorphic regions), the set of polymorphisms that can be detected and their apparent locations vary from a design to another. **This implies that if multiple designs were used in a data set, each sample will have to be scored with a polymorphism bank remapped to the corresponding design**.

Using the graphical interface, the bank remapping will be performed for the first sample, and the resulting map stored in the same directory as the polymorphism bank. Next samples will use the same map as long as it matches their design, else you will have to remove the map file to allow another one to be produced. This implies that the computation can be quite long for the first sample (more than ten minutes for high-resolution arrays), but should be faster for the next ones (usually a few seconds).

To compute a cnvScore for each genomic region in a RDT file (whether they were produced by cghRA as described in chapter 3 or not), first select these target files using the "Track(s) to process" button. Then click the "Design file" button to select the matching RDT design file, as produced in chapter 2, and a RDT file containing polymorphisms using the "Polymorphism track" button. You may then hit the "Compute CNV score" button to launch the processing. A new "cnvScore" column will be added to all selected files, which can be visualized following instructions in section 5.4.



Common parameters

Track(s) to process: 5 file(s) selected

Add an annotation column

Crossing type : cytoband   Fuzziness (bp) : 50000   Max elements : 30

Polymorphism likelihood scoring

Design file: Agilent 532 x 85.design.rdt

Polymorphism track: GRCh37.DGV_supp.rdt

Filter : ✓   Max score : 1   Fill gaps : ✓

Ready

Export RDT files to a spread-sheet editor

Column separator : ,   Decimal separator : .   Cell quoting : ✓   Header : ✓

cghRA also proposes to filter regions according to their cnvScore, in order to remove polymorphisms before down-stream analyses. To do so, just check the "Filter" box and provide a threshold above which a region will be considered as a polymorphism. The recommended threshold of 1 is a good balance between sensitivity and specificity, increasing it will enhance specificity (removed regions are certain polymorphisms but some may be missed) while decreasing it will enhance sensitivity (more regions are removed, including more misclassified somatic events). Please refer to cghRA publication for further details on the cnvScore thresholds.

R command lines

The same steps as described for the graphical interface can be performed using R command lines :

```
>   # Import files to process
>   dgv <- readRDT("GRCh37.DGV_supp.rdt")
>   design <- readRDT(file="design.rdt")
>   target <- readRDT(file="DLBCL_Feb2016_CHB02048.copies.rdt")

>   # Remap the polymorphism bank
>   dgvMap <- map2design(dgv, design)

>   # Compute cnvScore for a sample
>   sampleMap <- map2design(target, design)
>   score <- cnvScore(sampleMap, dgvMap)

>   # Add cnvScore in sample table
>   target$addColumn(content=score, name="cnvScore")
>   print(target)

>   # Filter using cnvScore
>   target$rowOrder(which(target$extract(,"cnvScore") < 1))
>   print(target)
```

5.4 Export to a spread-sheet editor

While the RDT format is convenient to store tables of genomic regions in an efficient way, only Rgb and software relying on it support this format. Thus users not familiar with Rgb would find more convenient to extract the content of RDT files to flat text files that can be processed with Excel or any other spread-sheet editor. Keep in mind however that the RDT format is an efficiently compressed format, meaning that some datasets stored in such files (e.g. exons from an entire genome or probes from a high-resolution array) can become huge when exported to CSV.

Notice finally that the resulting files will only be an export at time t of the RDT files, and that cghRA and Rgb are unable to directly work with this format. This implies that a new export may be required each time the source RDT file is updated, cghRA will **not** keep the two file content synchronized.

Graphical interface

The last section of the "Annotate regions" panel offers to convert selected tracks to CSV. Simply select the RDT files to export using the "Track(s) to process" button, define the format of the CSV file(s) to produce and hit the "Export to CSV" button. Files will be produced in the same directory as source files, replacing the ".rdt" file extension by ".csv".

As the CSV (originally standing for "Comma-Separated Values") file format can vary from a platform to another, several parameters can be set to obtain CSV files compatible to your spread-sheet editor. In Europe, where the decimal separator is usually a comma (","), CSV files use semi-colons (";") to differentiate columns and commas for decimal numbers. In other countries using the period (".") as decimal separator, CSV files generally use commas to separate columns. An alternative is to use tabulations (represented by "\t") to separate columns.

The user is proposed to enable or disable the **cell quoting** mechanism, which consists in putting the content of cells between double-quotes to protect special characters they may contain. This is only useful if a cell is susceptible to contain line breaks or characters used for column separation, and some spread-sheet editor may need specific configuration to handle this.

Finally the user is proposed to enable or disable the addition of a **header** to the CSV file. This header consists of a few lines beginning with a "#" symbol before the table, providing meta-data that were stored in the RDT file. Keeping this header is recommended, as it usually helps to identify and interpret the content of the table.

R command lines

Exporting the content of a RDT using R is straight-forward. Please refer to R documentation on data export and on the `write.table()` function in particular for more control of the exported file format.

```
>   object <- readRDT(file="DLBCL_Feb2016_CHB02048.copies.rdt")
>   content <- object$extract()
>   write.csv(content, file="DLBCL_Feb2016_CHB02048.copies.csv")
```

6 Series processing

While all other steps of the analysis described in here focused on one sample at a time, tools provided in the "Series processing" panel allow to widen one's point of view to the level of a series of samples, in order to identify recurring events. To do so, cghRA proposes to summarize a series of samples into various tables, that can be exported to Excel or visualized interactively in Rgb. This comprises condensed packings of the data, and output from several algorithms aiming at identifying "Minimal Common Regions" (MCR), i.e. the most recurrent overlaps between altered segments across a series.

6.1 Produce Rgb track files

Output types

A **copy number matrix** is an exhaustive representation of modelized copy-numbers in a two-dimension matrix, with samples in columns and genomic regions in rows. The genome is split in as many rows as necessary to describe all breakpoints found in the series of samples considered, which can be quite high when considering large series of highly rearranged genomes. While not directly visualizable in Rgb, it can be interesting to export such a matrix to CSV and open it with a spread-sheet editor.

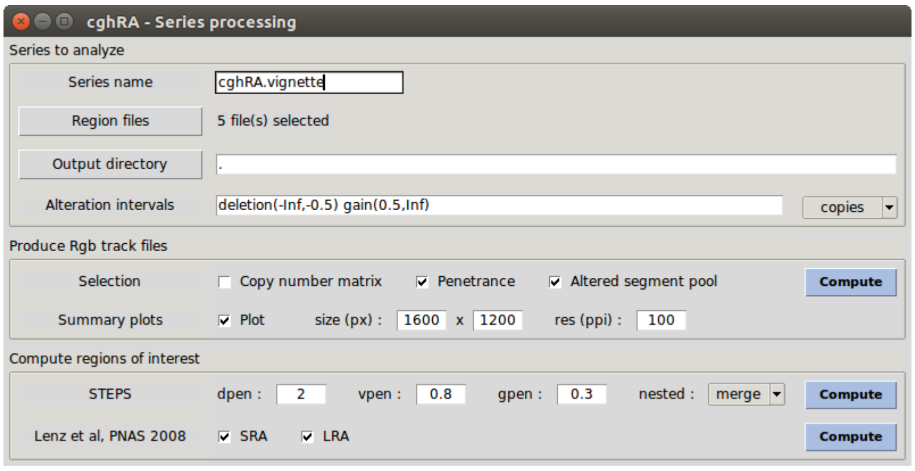
A **penetrance track** is a table describing what proportion of the samples show one specific type of alteration (gain, deletion, loss...) for any region of the genome. As for the copy number matrix, the table contains as many rows as necessary to exhaustively describe all the breakpoints in the sample. This is typically what is used by the STEPS algorithm to highlight regions of interest (see 6.2).

An **altered segment pool** consists in the gathering of all segments found as altered (amplified, deleted...) across a series in a single table, with a column describing which sample it originates from. This is probably the best track to browse with Rgb, in order to visually identify regions in which many distinct samples show similar alterations (see 7).

Summary plots can also be produced from penetrance and pool tracks. These are image files representing the content of the corresponding track(s) allong the 22 autosomes, as produced by Rgb function `singlePlot`. They allow to get an overview of the whole series in the whole genome.

Graphical interface

To produce one or many of these files using the graphical interface, fill the first "Series to analyze" block of the form. You will be required to provide a name for the series (mainly used to name output files and to be displayed in Rgb), the ".copies.rdt" files of the samples to include in the series and a directory in which the files produced will be stored.


cghRA - Series processing

Series to analyze

Series name	cghRA.vignette
Region files	5 file(s) selected
Output directory	.
Alteration intervals	deletion(-Inf,-0.5) gain(0.5,Inf) — copies

Produce Rgb track files

Selection	☐ Copy number matrix ☑ Penetrance ☑ Altered segment pool — Compute
Summary plots	☑ Plot size (px) : 1600 x 1200 res (ppi) : 100

Compute regions of interest

STEPS	dpen : 2 vpen : 0.8 gpen : 0.3 nested : merge — Compute
Lenz et al, PNAS 2008	☑ SRA ☑ LRA — Compute

You will also be required to describe the copy-number states you are interested in. To do so, fill the "Alteration intervals" field with as many alteration states you want, separated by white spaces. For each of them provide a name (e.g. "deletion" or "gain") and between parenthesis describe which copy numbers are part of this state, providing the lower boundary (which is included in the interval) and the upper boundary (which is **not** included in the interval). As can be seen in default values, `Inf` and `-Inf` (positive and negative infinite) can be used ensure that any value is classified.

While the recommended workflow consists in modelizing copy-numbers with cghRA and use these copy-numbers as states in this panel, one can also define alteration using log-ratios thresholds. To do so, switch the spin box at the end of the "Alteration intervals" to indicate that numeric values in your alteration state definitions are log-ratios and not modelized copy numbers. When working with log-ratio thresholds, ".regions.rdt" files can be used alternatively to the ".copies.rdt" files produced by the model at sections 3.5 and 4.

For **copy number matrix**, only the "log-ratio" vs "copies" choice will be used, to define the type of values to export in the matrix. However one distinct **penetrance track** will be produced for each alteration state you define, and the **altered segment pool** will only contain segments matching one these states.

If the "Plot" box is checked, a PNG file will also be produced for the penetrance (if penetrance tracks are produced) and the altered segment pool (if checked here again). These images represent the content of the track along the 22 chromosomes, providing a convenient overview of all the samples in the whole genome. You can define the size of the image file to produce in pixels (widths and height), as well as its resolution (in pixels per inches), in order to produce figures of the quality of your choice. Such representations can be further refined using Rgb function `singlePlot`, please refer to section 7.2 for further details and an example of output.

R command lines

These files can be produced using methods from the `cghRA.series` class. Notice that these methods rely on more generic R functions defined in the cghRA package, that can also be used on `data.frames` without using `cghRA.series` objects (please refer to cghRA package R manual for further details). Moreover these methods can provide additional parameters absent from the graphical interface, see the `cghRA.series` manual for more information.

```
>   # Aggregate sample files as a series object
>   files <- dir(pattern=".copies.rdt$")
>   series <- cghRA.series(files, .name="cghRA vignette")

>   # Produce and export a copy-number matrix
>   mtx <- series$parallelize(value="copies")
>   write.csv(mtx, file="CN_matrix.csv")

>   # Produce and export a penetrance track (deletion)
>   pen <- series$penetrance(
+     value = "copies",
+     states = list(deletion=c(-Inf, -0.5))
+   )
>   saveRDT(pen$del, file="Penetrance_del.rdt")

>   # Produce and export an altered segment pool
>   pool <- series$pool(
+     value = "copies",
+     states = list(deletion=c(-Inf, -0.5), gain=c(0.5, Inf))
+   )
>   saveRDT(pool, file="Segment_pool.rdt")
```

6.2 Compute regions of interest

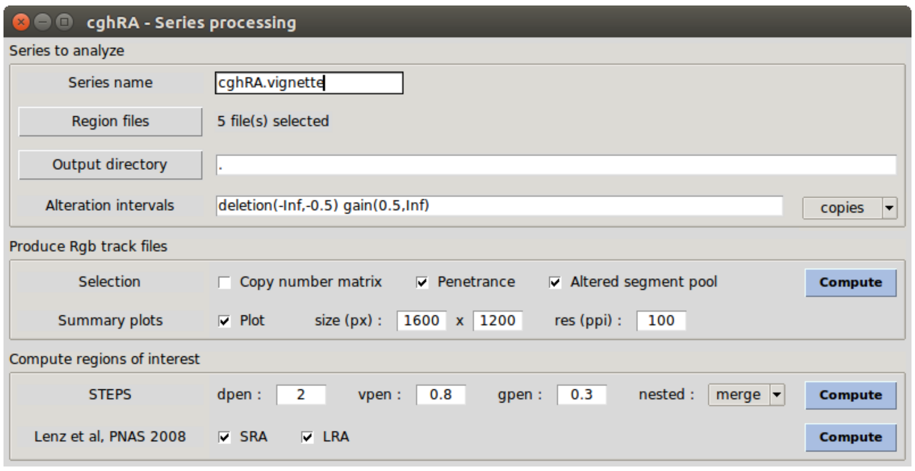
When analyzing multiple samples from the same tumor type, it is common to focus on Minimal Common Regions (MCR), i.e. the most recurrently altered regions, as they are the most likely to harbor genes of interest in this tumor type. Several algorithms were developed to infer such regions, and cghRA proposes three algorithms developed in Diffuse Large B-Cell Lymphomas : STEPS, SRA and LRA.

The first proposed algorithm is **STEPS**, standing for "Selective Trends Evidenced by Penetrance Surges", which was developed specifically for cghRA and described thoroughly in the associated publication referenced at the beginning of the current document. This algorithm will prioritize regions of interest attributing them a score, assessing several criterion found in confirmed regions of interest. Briefly, this algorithm identifies local maxima in the penetrance, and expand these regions on both sides to locate where in the genome the piling of altered segments starts and ends. It will then attribute a score to these regions, favoring regions presenting symmetric patterns (expanded starts and stops are at the same penetrance levels and the same distance from the MCR) and high penetrance in the MCR area. This controlled expansion step allows to discard chromosome-wide alterations from the analysis, and focus on local alterations only, which are the only ones supporting the involvement of a specific locus and its genes.

The two other ones, called "Short / Long Recurrent Abnormalities" (**SRA** / **LRA**), were described in the supplemental methods of *Lenz et al, PNAS 2008 Sep 9;105(36):13520-5*. For each region of interest they report several sets of coordinates of various width, corresponding to arbitrary proportions of the maximal penetrance met (1/3 and 2/3). SRA is more suitable to study narrow recurring events (less than 1 Mb) while LRA is intended to work with large segments. None of them provide scoring of the MCR, that can only be prioritized by decreasing penetrance, despite the fact that chromosome-wide alterations independent from the MCR will somehow blur this ordering.

Graphical interface

To apply one of these three algorithms, you will be required to fill the same form as described for Rgb track files (see section 6.1). Here again the analysis will be run independently for each alteration state you define, using the thresholds you provide.



cghRA - Series processing

Series to analyze

Series name	cghRA.vignette	
Region files	5 file(s) selected	
Output directory	.	
Alteration intervals	deletion(-Inf,-0.5) gain(0.5,Inf)	copies

Produce Rgb track files

Selection	[ ] Copy number matrix	[✓] Penetrance	[✓] Altered segment pool	Compute
Summary plots	[✓] Plot	size (px) : 1600 x 1200	res (ppi) : 100	

Compute regions of interest

STEPS	dpen : 2	vpen : 0.8	gpen : 0.3	nested : merge	Compute
Lenz et al, PNAS 2008	[✓] SRA	[✓] LRA			Compute

For **STEPS**, the user can change the 3 main weights of the computation, despite it is not recommended. **dpen** is the penalty applied when the penetrance increases rather than decreases when moving away from the MCR. Higher values will expand further and allow more nesting, as small penetrance peaks will be passed through rather than stop the region expansion. **vpen** is the vertical symmetry weight, larger values will put a strongest focus on regions with boundaries at the same penetrance level. **gpen** is the genomic symmetry weight, larger values will emphasize on regions with boundaries at the same genomic distance from the MCR. Finally **nested** controls how to deal with regions that overlap when expanded and thus share some altered segments. With the "merge" behavior, only the highest MCR will be retained in the results, considering other MCR as bystanders. With "flag", all MCRs will be retained in the results and overlaping MCRs will be attributed a common nest ID, which will allow to identify them easily. With "none", no action is performed regarding nested MCRs, which will all be reported.

For **SRA** and **LRA**, one can solely enable or disable the computation of one of them, which are usually computed together.

Once the corresponding "Compute" button is hit, a new RDT file will be produced for the corresponding algorithm(s) and for each alteration state, in the output directory you choosed.

R command lines

As for Rgb tracks, regions of interest can be computed using methods from the `cghRA.series` class :

```
>   # Aggregate sample files as a series object
>   files <- dir(pattern=".copies.rdt$")
>   series <- cghRA.series(files, .name="cghRA vignette")

>   # Produce and export STEPS (deletion)
>   steps <- series$STEPS(
+     value = "copies",
+     states = list(deletion=c(-Inf, 0))
+   )
>   saveRDT(steps$del, file="STEPS_del.rdt")

>   # Produce and export SRA (deletion)
>   sra <- series$SRA(
+     value = "copies",
+     states = list(deletion=c(-Inf, 0))
+   )
>   saveRDT(sra$del, file="SRA_del.rdt")

>   # Produce and export LRA (deletion)
>   lra <- series$LRA(
+     value = "copies",
+     states = list(deletion=c(-Inf, 0))
+   )
>   saveRDT(lra$del, file="LRA_del.rdt")
```

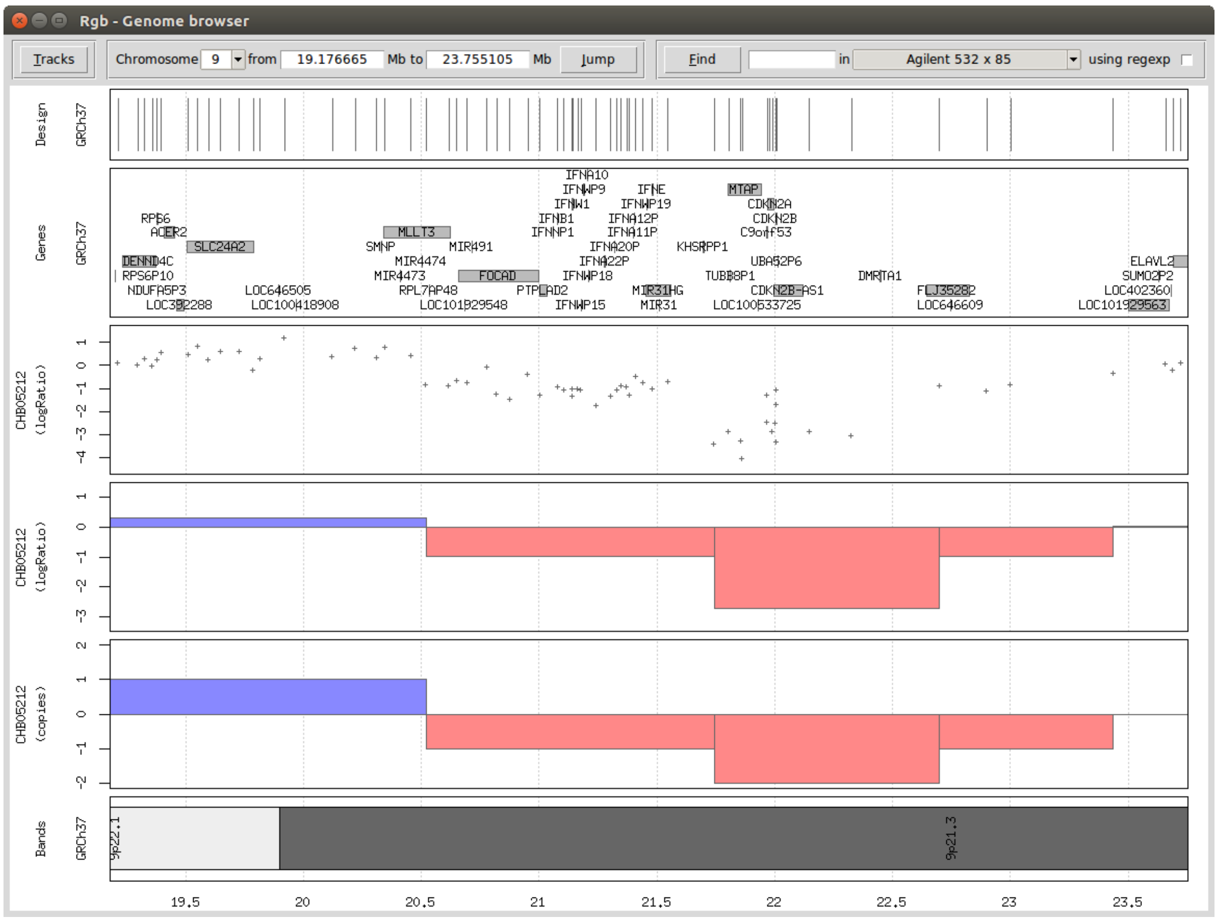
7 Visualize results with Rgb

This section only gives a glimpse of Rgb visualization capabilities. For more details on Rgb and its usage, please refer to its own vignette that can be found in the Rgb package. Most of the files produced or used in this vignette can be directly imported in Rgb for visualization, and notably :

- **.design.rdt** files resulting from the design processing (chapter 2).
- **.probes.rdt** files resulting from the array import and preprocessing (chapter 3). Notice you will be asked to select the matching ".design.rdt" as well when importing such a file.
- **.regions.rdt** files resulting from the segmentation (section 3.4).
- **.copies.rdt** files resulting from the copy-number modelization (sections 3.5 and 4).
- All **annotation tracks** that can be used in chapter 5, and downloaded from Rgb website.

7.1 Graphical interface

To do so in an interactive mode, one simply needs to launch Rgb using the `tk.browse()` R command or any alternative way that Rgb may provide in the future (refer to Rgb package documentation for further details). Click **Tracks** and "Add from file" to select the RDT files to add to the current view, and "Done" to come back to the main screen.



Rgb - Genome browser

Chromosome 9 from 19.176665 Mb to 23.755105 Mb

in Agilent 532 x 85 using regexp

Then to visualize a region of the genome, you can enter its coordinates in the first part of the form (chromosome name, starting and ending position in Mb) and hit the **Jump** button. Notice that starting and ending positions are optional, leaving these fields blanks will display the whole selected chromosome.

Alternatively, you can set the window to a specific locus searching one of the selected tracks (e.g. zoom on a specific gene providing its symbol). To do so, the track to be searched ("GRCh37.NCBI_genes.rdt" in our example) needs to be part of the selected tracks (but can be hidden though). Then simply type in the **Find** field the symbol of the gene you are looking for, select the track to search in the menu and hit the "Find" button. If multiple hits are found, click again on "Find" to switch from one to another. More experienced users can also check the "using regexp" box to enter a regular expression rather than a fixed string to search. This "Find" feature can work on any track, just keep in mind that the string will be searched for in the "name" column of the track you select, not any other.

Once a region is displayed, you can use the **left** and **right arrow keys** of your keyboard to move along the chromosome, and the **up** and **down** arrows to zoom in or out. Zooming in and out can also be performed using the **mouse wheel**, or by selecting a region to zoom in with the **left mouse button** (press the button over a genomic location, move your mouse to another and release the button).

Visible data tracks can be managed in the panel summoned clicking the "Tracks" button, where they can notably be ordered, hidden or edited. Various graphical parameters can be changed for most tracks, just don't forget to save your changes before closing the window and reload the graphics (hitting the "Jump" button or the "R" key from your keyboard).

7.2 R command lines

Such representations can also be produced using R commands, or a mix of R commands and interactive panels. Notice however that only `tk.browse()` will provide a full interactive experience with chromosome walking using the mouse and the keyboard.

The first step will be to gather the data tracks to visualize into a `drawable.list` object. More details about this class and the methods it offers can be found in R manual pages.

```
>   # Import some data tracks into R memory
>   design <- readRDT(file="design.rdt")
>   sample <- readRDT(file="DLBCL_Feb2016_CHB05212.copies.rdt")

>   # Create an empty drawable list
>   dl <- drawable.list()

>   # Bind tracks from R memory
>   dl$add(file=NA, track=design)
>   dl$add(file=NA, track=sample)

>   # Bind tracks from file
>   dl$add(file="GRCh37.NCBI_genes.rdt")

>   # Or use the interactive interface
>   dl$fix.files()
>   dl <- tk.tracks()
```

Then one can simply use Rgb's `browsePlot()` function to visualize a specific region of the genome.

```
>   # Visualize a region
>   browsePlot(dl, chrom=9, start=19.18e6, end=23.76e6)
```

Such plots can be exported to files, using classical R mechanisms.

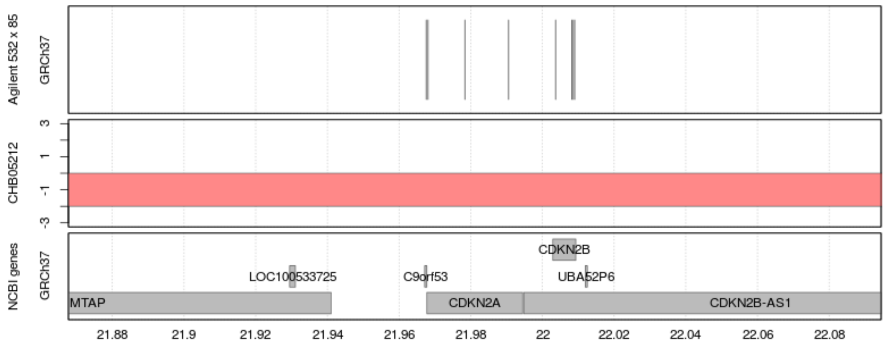
```
>   # Export into a PNG file
>   png("export.png", width=800, height=600, res=100)
>   browsePlot(dl, chrom=9, start=19.18e6, end=23.76e6)
>   dev.off()
```

The whole interactive interface can still be summoned at any time :

```
>   # Visualize the drawable list in Rgb
>   tk.browse(dl)
```

To find regions of interest, one can use tracks for any type of computation or filtering, as extensively described in Rgb vignette. Tracks can be extracted from the `drawable.list` object using the `get`, `getByClasses` or `getByNames` methods or be stored apart in R memory.

```
>   # Select a gene track stored in the list
>   genes <- dl$getByClasses("track.genes")[[1]]
>   # Visualize CDKN2A locus
>   locus <- genes$extract(expression(name == "CDKN2A"))
>   browsePlot(
+           drawables = dl,
+           chrom = locus$chrom,
+           start = locus$start - 1e5,
+           end = locus$end + 1e5
+   )
```



Drawing parameters can be set for any track, using the interface or the `setParam` method of the `drawable` class. Details about drawing parameters are provided by the interface and explained in Rgb documentation.

```
>   # Update parameters of an existing list using the interface
>   dl$fix.param()
```

```
>   # Update parameters of a specific track from the list
>   genes <- dl$getByClasses("track.genes")[[1]]
>   genes$setParam("colorVal", "#FF8888")
```

Keep in mind that classes in Rgb and cghRA are **reference classes**, so copying a variable will not duplicate its content but only the reference to the memory address it is stored in. It implies that modifications to one of the copies of the object will be propagated to all of them, especially for objects added to a `drawable.list`.

```
>   # Import a track and "copy" it
>   genes <- readRDT(file="GRCh37.NCBI_genes.rdt")
>   copy <- genes
```

```
>   # Add it to a drawable list
>   dl <- drawable.list()
>   dl$add(file=NA, track=genes)
```

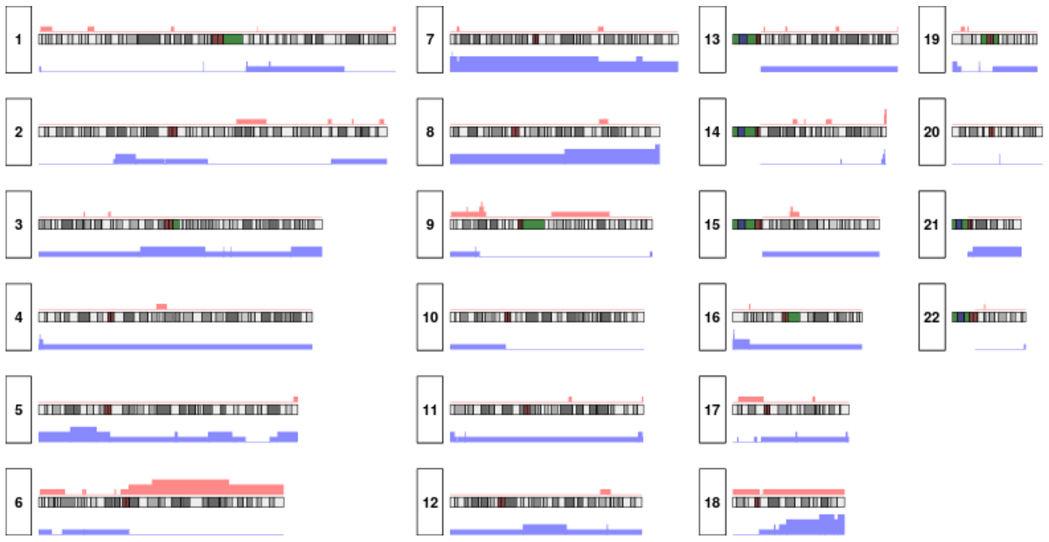
```
>   # Update one of the copies
>   genes$setParam("ylab", "Updated gene list name")
```

```
>   # All copies were updated
>   print(genes$getParam("ylab"))
>   print(copy$getParam("ylab"))
>   print(dl$get(1)$getParam("ylab"))
```

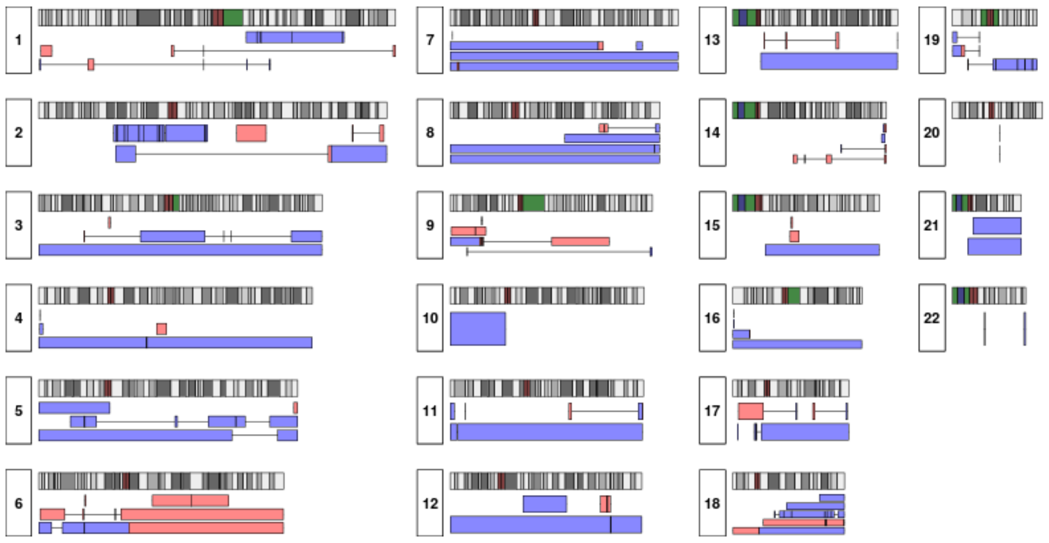
Alternatively to single locus plot, one can use the `singlePlot()` function to plot a representation of the selected tracks in the whole genome.

Keep in mind however that most tracks are not optimized for such a large zoom level, so one or many parameters will probably need to be tuned to obtain neat figures. The "height" drawing parameter is one of them for most annotation tracks, as it is usually set in centimeters rather than relative units. Moreover the default plot window summoned by R may prove too small to hold such a large picture ("figure region too large" or "Plot area too small" errors), so you should consider to plot directly to a file or resize the plot window before calling `singlePlot()`.

```
> # Create a drawable list
> dl <- drawable.list()
> dl$add("cghRA vignette penetrance (deletion).rdt")
> dl$add("GRCh37.UCSC_bands.rdt")
> dl$add("cghRA vignette penetrance (gain).rdt")

> # Tune cytoband track parameters
> bands <- dl$getByClasses("track.bands")[[1]]
> bands$setParam("height", 0.5)
> bands$setParam("label", FALSE)

> # Produce penetrance summary plot
> png("Penetrance summary.png", width=1600, height=800, res=100)
> singlePlot(dl)
> dev.off()
```



```
> # Replace penetrance tracks with the pool track
> dl$remove(c(1, 3))
> dl$add("cghRA vignette pool (copies).rdt")

> # Produce altered pool summary plot
> png("Pool summary.png", width=1600, height=800, res=100)
> singlePlot(dl)
> dev.off()
```


1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22