`MPAgenomics` : An `R` package for Multi-Patients Analysis of Genomic Markers.

Quentin Grimonprez, Samuel Blanck, Alain Celisse, Guillemette Marot

March, 2014

Contents

1 Introduction

This package provides functions to preprocess and analyze genomic data. Markers refer to SNPs or copy number variations which are designed on the arrays. `MPAgenomics` is devoted to: (*i*) efficient segmentation and (*ii*) genomic marker selection from multi-patient copy number and SNP data profiles.

For both types of analyses, a pre-processing step (section 4) is proposed as a wrapper of `aroma.*` packages [1][4]. This enables to keep maximum information from the original signals and improve the multi-patients analysis. In particular, this is useful to keep quantitative data for SNPs rather than usual genotype calls (AA, AB or BB) when these states are not relevant (eg in cancer studies where the number of copies differs from two copies). Note that the use of `aroma.*` packages [1] implies that the pre-processing step is only available for

Affymetrix DNA chips. The other functionalities of the package `MPAgenomics` can however be used independently and such procedures are explained in the sections which detail each step of the proposed analysis.

Efficient segmentation is proposed by refining the parameter choice of PELT segmentation [10] originally implemented in the `changepoint` package [9]. `MPAgenomics` offers a pipeline to perform normalization, segmentation and calling at the same time.

Selection of genomic markers relies on the use of penalized regularization techniques implemented in `glmnet` [7] and wrappers are offered to relate this package with the normalisation packages. For the users who would like to go further in the use of penalized regularization techniques, we also provide some guidelines to quickly build inputs for the `HDPenReg` package and interpret some outputs of this latter one.

2 Preliminaries

2.1 Citing MPAgenomics

Please always cite the following paper when using `MPAgenomics`:

- Q. Grimonprez, A. Celisse, M. Cheok, M. Figeac, and G. Marot. Mpagenomics : An r package for multi-patients analysis of genomic markers, 2014. Preprint http://hal.inria.fr/hal-00933614

Moreover, `MPAgenomics` wraps and extends fonctionnalities of several packages. Please try to cite the appropriate methodological papers when you use results from the `MPAgenomics` software in a publication, as such citations are the main means by which the authors receive professional credit for their work.

If you use CRMAv2 normalization in `MPAgenomics`, please cite:

- H. Bengtsson, P. Wirapati, and T. P. Speed. A single-array preprocessing method for estimating full-resolution raw copys from all affymetrix genotyping arrays including genomewidesnp 5 & 6. Bioinformatics, 25(17):2149–2156, 2009

If you use TumorBoost normalization in `MPAgenomics`, please cite:

- H. Bengtsson, P. Neuvial, and T. P. Speed. Tumorboost: Normalization of allele-specific tumor copy numbers from a single pair of tumor-normal genotyping microarrays. BMC Bioinformatics, 11, 2010

If you use `MPAgenomics` to segment copy number or allele B fraction signals with the PELT segmentation method, please cite:

- R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association, 107(500):1590–1598, 2012

If you use `MPAgenomics` for calling with `CGHcall`, please cite:

- M. A. van de Wiel, K. I. Kim, S. J. Vosse, W. N. van Wieringen, S. M. Wilting, and B. Ylstra. CGHcall: calling aberrations for array CGH tumor profiles. Bioinformatics, 23(7):892–894, 2007

2.2 Installation

The package can be installed either from a GUI (selecting the R-forge repository) or with the command line (to be typed in the R console)

```
> install.packages("MPAgenomics", repos="http://R-Forge.R-project.org")
```

This vignette can be produced with the command

```
> vignette("MPAgenomics")
```

Since the package provides simple wrappers of many existing packages and all users are not interested by the same wrappers, we did not force `MPAgenomics` to depend on all the packages. We however recommend the user who would like to preprocess data to install `aroma` packages [1] with the following commands:

```
> source("http://www.braju.com/R/hbLite.R")
> hbLite("sfit")
> source("http://bioconductor.org/biocLite.R")
> biocLite("affxparser")
> source("http://aroma-project.org/hbLite.R")
> hbInstall("aroma.affymetrix")
> hbInstall("aroma.cn")
> install.packages("aroma.core")
```

For users who want to perform the single analysis (segmentation and calling), we recommand to install the packages `changepoint`[9] and `CGHcall` [13].

```
> #for segmentation
> install.packages("changepoint")
> #for calling
> source("http://bioconductor.org/biocLite.R")
> biocLite("CGHcall")
```

The installation of all these packages at the very beginning of the use of `MPAgenomics` is only for convenience, not mandatory. Indeed, the user who would have forgotten to install some packages would be reminded to install them when applying the corresponding wrapper.

3 Starting example

This introductory example aims at helping the user understand the main functions of the package. For more details on each step, we invite the user to read the appropriate section.

3.1 Download a toy data-set

The example is based on a free data-set containing 8 .CEL files (251Mo) which can be downloaded from the Broad Institute website, following this link: `http://www.broadinstitute.org/mpg/birdsuite/downloads/birdsuite_inputs_1.5.3.tgz` If you are used to decompress .tgz files, extract this file and copy the following command in the R console, replacing the path to the directory where the raw data (.CEL files) are stored with the appropriate one. Note that we deliberately use ”/” and not ”\ ” in `celPATH`. If you copy paste paths from Windows (e.g. looking at properties of the directory), you must change ”\ ” by ”/” or ”\\ ”.

```
> celPATH="/home/user/Documents/workdir/CELdata/cel"
```

In this example, the directory pointed by celPATH must contain the files shown in Figure 1.

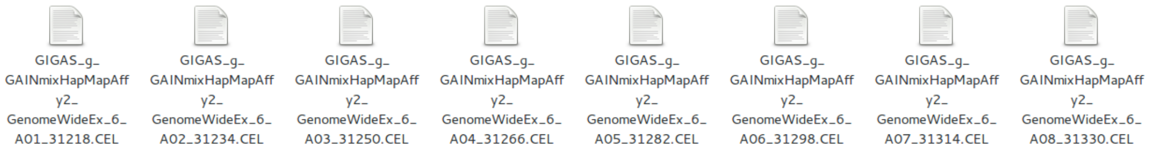

GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A04_31266.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A06_31298.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A08_31330.CEL

Figure 1: CEL files in `celPATH` (variable containing the path to the downloaded CEL files).

If it is not the case, e.g. if you do not know how to download and extract .tgz files (e.g. via a right click) , you can use the following commands in the R console:

```
> #set your working directory (replace with the appropriate path)
> workdir="/home/user/Documents/workdir"
> setwd(workdir)
> #download file
> download.file("http://www.broadinstitute.org/mpg/birdsuite/downloads/birdsuite_inputs_1.5.3.tgz",
+ destfile="./CELdata.tgz")
> #untar the file
> untar("./CELdata.tgz",files="cel",exdir=".")
> #indicate the path containing .cel files
> celPATH="./cel"
```

Once .CEL files are extracted, you must download all the GenomeWideSNP_6 annotation files which are needed to pre-process the data. These files have extensions .cdf, .ufl, .ugp, .acs. You can download the cdf files (258Mo) from the Affymetrix website (an user account is required), following this link `http://www.affymetrix.com/Auth/support/downloads/library_files/genomewidesnp6_libraryfile.zip`

Extract these files and replace the following path with the appropriate one pointing to the folder containing the extracted .cdf files (only GenomeWideSNP_6.Full.cdf is needed but you can leave the other files in the same folder):

```
> chipPATH="/home/user/Documents/workdir/CD_GenomeWideSNP_6_rev3/Full/GenomeWideSNP_6/LibFiles/"
```

If you do not know how to decompress .zip files, you can use the following commands in the R console for convenience:

```
> #unzip required files
> unzip("./genomewidesnp6_libraryfile.zip",
+  files=c("CD_GenomeWideSNP_6_rev3/Full/GenomeWideSNP_6/LibFiles/GenomeWideSNP_6.Full.cdf"),exdir=".")
> #indicate the path containing .cdf files
> chipPATH="./home/user/Documents/workdir/CD_GenomeWideSNP_6_rev3/Full/GenomeWideSNP_6/LibFiles/"
```

The other annotation files can be downloaded from `http://www.aroma-project.org/data/annotationData/chipTypes/GenomeWideSNP_6/`. For this example, we download GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl.gz (6Mo), GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp.gz (7Mo), GenomeWideSNP_6,HB20080710.acs.gz (37Mo). Extract these files in the folder where your cdf files are in order to have all the annotation files in the same folder (see Figure 2).

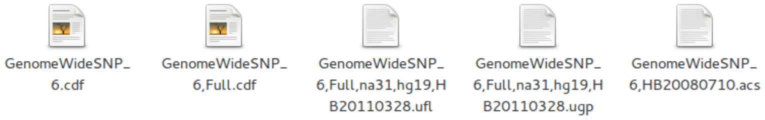

GenomeWideSNP_6.cdf
GenomeWideSNP_6,Full.cdf
GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl
GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp
GenomeWideSNP_6,HB20080710.acs

Figure 2: Chip GenomeWideSNP_6 files in `chipPATH` (variable containing the path to the downloaded chip files).

If you do not know how to download or decompress .gz files, you can use the following commands in the R console for convenience:

```
> #set the directory where the .cdf files are as your working directory
> setwd(chipPATH)
> ##download the 3 files .ufl, .ugp, .acs
> download.file("http://www.aroma-project.org/data/annotationData/chipTypes/GenomeWideSNP_6/GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl.gz",
+ destfile="GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl.gz")
> download.file("http://www.aroma-project.org/data/annotationData/chipTypes/GenomeWideSNP_6/GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp.gz",
+ destfile="GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp.gz")
> download.file("http://www.aroma-project.org/data/annotationData/chipTypes/GenomeWideSNP_6/GenomeWideSNP_6,HB20080710.acs.gz",
+ destfile="GenomeWideSNP_6,HB20080710.acs.gz")
> #unzip the gz files
> #install R.utils package containing the gunzip function
> install.packages("R.utils")
> library("R.utils")
> gunzip("GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl.gz",
+ destname="GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl")
> gunzip("GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp.gz",
+ destname="GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp")
> gunzip("GenomeWideSNP_6,HB20080710.acs.gz",destname="GenomeWideSNP_6,HB20080710.acs")
```

3.2 Use of the main functions of the package

In the following example, we suggest to perform segmentation and calling before selection of markers even if only the pre-processing step, which estimates the copy-number and the allele B fraction signal, is required for the last one.

We assume that all the required chip files are in the `chipPATH` folder and that the .CEL files are in the `celPATH` folder.

First, we normalize .CEL files using the CRMAv2 method [5] from aroma packages. These packages require a specific architecture containing .CEL and annotation files. This normalization step is required before any statistical analysis. The following command performs the creation of architecture and the normalization at the same time :

```
> #list files to check that your path to annotation files (.cdf, .ugp, .ufl, .acs) is correctly set
> dir(chipPATH)
> # list files to check that your path to cel files (.cel) is correctly set
> dir(celPATH)
> #set your working directory (where you have rights to write)
> setwd(workdir)
> #normalize data (might take several hours)
> signalPreProcess(dataSetName="datatest1", chipType="GenomeWideSNP_6",
+ dataSetPath=celPATH,chipFilesPath=chipPATH, path=".",
+ createArchitecture=TRUE, savePlot=TRUE, tags="Full")
```

The first pipeline offered in `MPAgenomics` consists in the segmentation of every copy-number signal of the data. Then, a calling method can be applied on all the segments to label them (e.g. ”loss”, ”normal”, ”gain”). To perform segmentation and calling at the same time, type the following lines in your R console:

```
> segcall=cnSegCallingProcess("datatest1",chromosome=c(1,5))
> #summary of segmentation and calling process
> segcall
```

In this example, chromosomes 1 and 5 for all patients are studied. Figures can be found at `./figures/datatest1/segmentation/`. To better understand the way the results have been produced, we refer to sections 4, 5 and 6.

Results of the calling can be filtered to keep only segments of a minimal size or with a specific label. For example, if you want to only keep segments which represent losses or gains with a minimal size of 10pb and containing at least 2 probes, run the following command:

```
> callfiltered=filterSeg(segcall,minLength=10,minProbes=2,keptLabel=c("gain","loss"))
> head(callfiltered)
```

To perform selection of markers, a response is associated with each profile and the goal is to find the most relevant markers in the profile according to the response. You can run the following commands:

```
> dataResponse=data.frame(files=getListOfFiles("datatest1"),
+ response=c(2.105092,1.442868,1.952103,1.857819,2.047897,1.654766,2.385327,2.113406))
> res=markerSelection("datatest1",dataResponse,chromosome=21:22,signal="CN",
+ onlySNP=TRUE,loss="linear")
```

For interpretation of all the results, we refer to section 7.

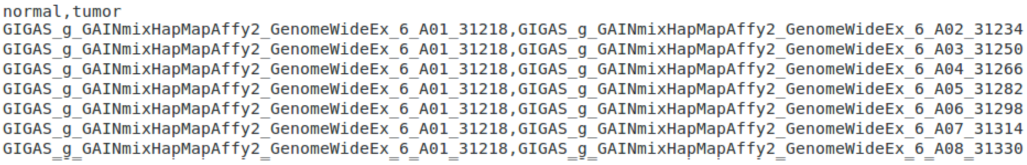
3.3 To go further...

Use of control samples In the previous example, there was no control sample. By default, the median of all the samples is used as a reference to estimate the copy-number profile. It is generally better to provide control samples. When this is the case, it is necessary to provide a file or a data.frame giving the correspondence between each control and each sample. For this example, we assume that the control is arbitrarily the first file:

```
> #get the file names of our data-set
> files=getListOfFiles("datatest1")
> #create the data.frame linking normal and tumor files
> normalTumorArray=data.frame(normal=rep(files[1],7),tumor=files[2:8])
```

Note that the extension .CEL is not provided in the `normalTumorArray` dataframe. In the following of the vignette, a study which provides both control and tumoral samples will be refered as a normal-tumor study.

A new normalization can be run by taking into account information from the control sample. To separate analyses with or without control samples, we will store results from this normal-tumor study under the name ”datatest2”. As the architecture has already been created for ”datatest1”, it is not necessary to run `signalPreProcess` with `createArchitecture=TRUE`. It is faster to add the CEL files in ”datatest2” while leaving the chip definition files as they are by using the function `addData` and then `signalPreProcess` with `createArchitecture=FALSE`.



```
normal,tumor
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A04_31266
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A06_31298
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218,GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A08_31330
```

Figure 3: CSV file linking normal sample with tumoral sample.

```
> addData(dataSetName="datatest2",dataPath=celPATH,chipType="GenomeWideSNP_6")
> signalPreProcess(dataSetName="datatest2", chipType="GenomeWideSNP_6",
+ normalTumorArray=normalTumorArray, createArchitecture=FALSE, savePlot=TRUE, tags="Full")
```

Single-Patient Analysis of allele B fraction The segmentation example (3.2) was run on the copy number signal, the allele B fraction can also be segmented after some transformation. First, we omit the homozygous SNPs of the signal and then symmetrize around 0.5 the signal and multiply by 2 to have values between 0 and 1 (a point x become $2 * |x - 0.5|$)(see Figure 4).

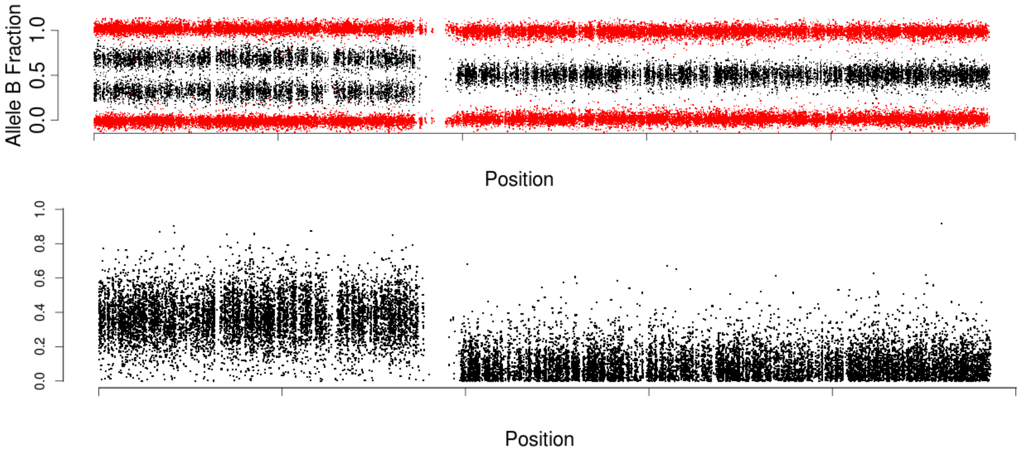


Figure 4: Top : Allele B fraction signal on a chromosome (in red : homozygous SNPs. Bottom : symmetrized signal without homozygous SNPs.

To perform the segmentation on the allele B fraction, run the following command

```
> #run the segmentation
> segfracB=segFracBSignal("datatest1",chromosome=c(1,5))
> #print summary of segmentation
> segfracB
```

Selecting steps to be executed to avoid a complete procedure Each step can be performed with or without the aroma packages if arguments of functions are given in the right format (excepted the normalization procedure described in section 4). Almost each wrapper can be used from matrices independently from the other steps and more details are given in the following of this vignette. The paper is organized as follows : the section 4 presents the preprocessing step and how to access normalized copy-number and allele B fraction. If you already have these data, you can skip this section. Then, an efficient single patient analysis is introduced in the next two sections. The segmentation process is described in section 5. In section 6, we present the calling process of segmented data. The selection of markers is described in section 7.

4 Data Normalization

This preprocessing step consists in a correction of biological and technical biases due to the experiment.

Raw data from Affymetrix DNA chip are provided in different .CEL files. These data need to have some corrections and normalizations before being usable.

This normalization process can be performed via functions from `aroma.*` packages [1][4]. Here, we provide some user-friendly wrappers to these functions. We assume that all the required chip files are in the `chipPATH` folder and the CEL files are in the `celPATH` folder.

4.1 Folder architecture

All the functions from aroma packages have to be used with a particular architecture in the working directory. In `MPAgenomics`, we use the same architecture and provide some functions to create it.

A function is provided to create the architecture and copy the specified files in the right directory : `createArchitecture`. It also checks that the names of the chip files begin by the specified `chipType` argument.

```
createArchitecture(dataSetName,chipType,dataSetPath,chipFilesPath,path,verbose,tags)
```

Parameters	Description
`dataSetName`	The name of the data-set folder to create.
`chipType`	The name of the used chip.
`dataSetPath`	Path to the folder containing the data CEL files.
`chipFilesPath`	Path to the folder containing the chip files.
`path`	Path where the architecture should be created (default=".").
`verbose`	Print information during the process (default=FALSE).
`tags`	Common tag which appears in the different file names (cdf, ugp, ufl) of the chip. For no tag, use tags=NULL (default = NULL).

For example, assume that we have the downloaded files of the download section (Fig 5 & 6).

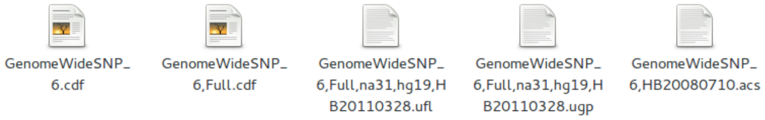

GenomeWideSNP_6.cdf
GenomeWideSNP_6,Full.cdf
GenomeWideSNP_6,Full,na31,hg19,HB20110328.ufl
GenomeWideSNP_6,Full,na31,hg19,HB20110328.ugp
GenomeWideSNP_6,HB20080710.acs

Figure 5: Chip GenomeWideSNP_6 files in `chipPATH` (variable containing the path to the downloaded chip files).

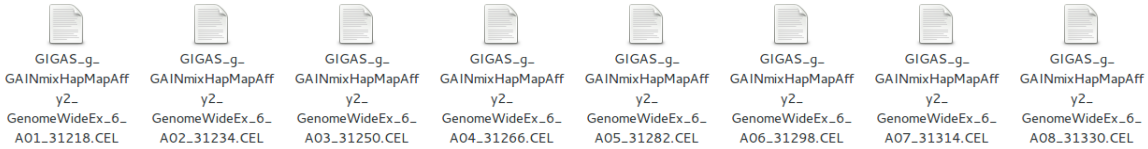

GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A04_31266.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A06_31298.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314.CEL
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A08_31330.CEL

Figure 6: CEL files in `celPATH` (variable containing the path to the downloaded CEL files).

The `celPATH` and `chipPATH` variables contain the different paths to the files downloaded. The following command enables to both create the architecture and add the previous dataset under the name ”datatest1”.

```
> createArchitecture("datatest1","GenomeWideSNP_6",celPATH,chipPATH,".",TRUE,"Full")
```

In your working directory, you must find the following architecture :

```
<current working directory>
  +- annotationData/
  |  +- chipTypes/
  |     +- GenomeWideSNP_6/
  |        +- CDF file(s) and other annotation
  |
  +- rawData/
     +- datatest1/
        +- GenomeWideSNP_6/
           +- CEL files
```

You can see that files are only copied and not moved.
This architecture can also be created with the normalization function (see section 4.2).

The minimal architecture contains 2 folders : `annotationData` and `rawData`. Some supplementary folders will be created during the execution of some functions.

In the `annotationData` folder, each type of chip used for the experiment has to have his own folder containing its definition files (.cdf) and other files (.ugp[1], .acs[2], .ufl[3]). In this folder, every file name must comply with the following format <chipType>,<tags>.<extension>(see Fig. 5).

The `rawData` folder contains the different data-sets with the CEL files. Each data-set folder contains one or more folders (one per different chip type used) containing the CEL files, these files will not be modified during the process. Note that the `chipType` folder in the data-set folder must match exactly a chip type folder under annotationData.

In case of new data-sets or new chip types, it is not useful to create a new architecture but just add new folders in the existing architecture as follows :

```
<current working directory>
  +- annotationData/
  |  +- chipTypes/
  |     +- <chipTypeA>/ <-- must match exactly the name of the CDF file (fullname minus tags)
  |     |  +- CDF file(s) and other annotation (possibly subdirectories)
  |     |
  |     +- <chipTypeB>/ <-- must match exactly the name of the CDF file (fullname minus tags)
  |        +- CDF file(s) and other annotation (possibly subdirectories)
  |        ...
  |
  +- rawData/
  |  +- <dataSet1>/
  |  |  +- <chipTypeA>/ <-- must match exactly a chip type folder under annotationData/
  |  |     +- CEL files
  |  |
  |  +- <dataSet2>/
  |  |  +- <chipTypeB>/ <-- must match exactly a chip type folder under annotationData/
  |  |     +- CEL files
  |  |
  |  +- <dataSet3>/
  |  |  +- <chipTypeA>/ <-- must match exactly a chip type folder under annotationData/
  |  |     +- CEL files
  |  |  +- <chipTypeB>/ <-- must match exactly a chip type folder under annotationData/
  |  |     +- CEL files
  |  ...
```

In order to easily update your existing architecture you can use the following `addData` function.
It enables to add to your existing architecture a new data-set in the `rawData` folder.

```
addData(dataSetName,dataPath,chipType)
```

Parameters	Description
`dataSetName`	The name of the data-set folder to create.
`dataPath`	Path of the folder containing the data CEL files.
`chipType`	The name of the used chip.

---

[1] See `http://www.aroma-project.org/node/43` for more details.

[2] See `http://www.aroma-project.org/node/100` for more details.

[3] See `http://www.aroma-project.org/node/47` for more details.

The `addChipType` function enables to add to your existing architecture a new chip type in the `annotationData` folder.

```
addChipType(chipType,chipPath)
```

Parameters	Description
`chipType`	Name of the new chip type to add.
`chipPath`	Path to the files to add.

With the function `getListOfFiles`, you can obtain the list of the files contained in the specified data-set.

```
getListOfFiles(dataSetName,chipType)
```

Parameters	Description
`dataSetName`	The name of a data-set folder.
`chipType`	The name of the used chip.

If you do not specify a `chipType`, it returns the files for the first chip in the alphabetic order in the `dataSetName` folder.

4.2 Normalization process

The main function for the normalization of CEL files is `signalPreProcess`. This function executes 3 different methods on your data:

- **CRMAv2** [5]: Normalize signals to obtain the copy-number and the allele B fraction. This step consists in the correction of biological and technical biases due to the experiment.
- **Genotype calls**: Assign label *AA*, *AB* or *BB* to each SNP.
- **TumorBoost** [3]: Only in a case of normal-tumor study, normalization of the allele B fraction tumor signal using the control signal.

The function `signalPreProcess` executes all these steps and saves the results in different folders in the aroma architecture (`totalAndFracBData`,. . .), you can also obtain some graphics (Fig 7) in the `figures/signal` folder.

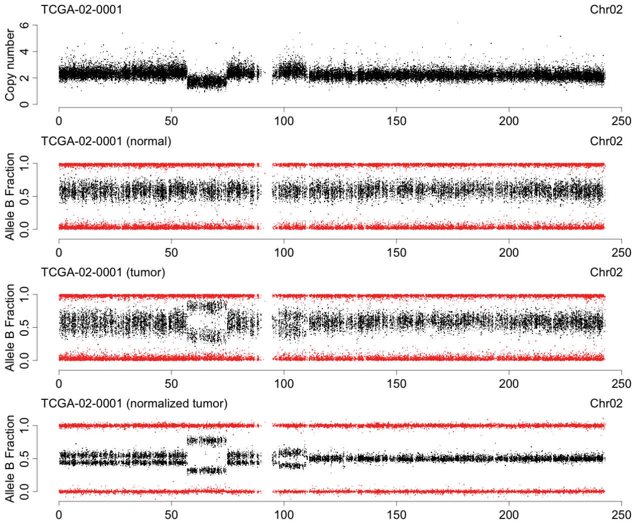
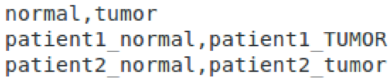


Figure 7: From top to bottom : the normalized copy-number signal, the allele B fraction for the normal profile, the allele B fraction for the tumor profile, the allele B fraction for the tumor profile after Tumorboost normalization. Graphics from `http://aroma-project.org`.

The `signalPreProcess` is defined as follows:

```
signalPreProcess(dataSetName, chipType, normalTumorArray, dataSetPath, chipFilesPath, path,
createArchitecture, savePlot)
```

Parameters	Description
`dataSetName`	The name of the data-set.
`chipType`	The type of the used chip (e.g. ”GenomeWideSNP_6”).
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`dataSetPath`	Only if `createArchitecture=TRUE`. Path to the folder containing the CEL files of the data-set.
`chipFilesPath`	Only if `createArchitecture=TRUE`. Path to the folder containing all the annotations files for the specified chip type.
`path`	Only if `createArchitecture=TRUE`. Path where the architecture should be created (default=”.”).
`createArchitecture`	if TRUE, the aroma architecture will be automatically created (default=TRUE).
`savePlot`	if TRUE, graphics of the CN signal and allele B fraction signal will be saved in the `figures/signal` folder. (default=TRUE).
`tags`	Common tag which appears in the different file names (cdf, ugp, ufl) of the chip. For no tag, use tags=NULL (default = NULL).



```
normal,tumor
patient1_normal,patient1_TUMOR
patient2_normal,patient2_tumor
```

Figure 8: Example of the content of a csv file for `normalTumorArray` parameter. The first column contains the name of the different normal files (without the .cel extension) in the data-set folder in rawData. The second column contains the name of the tumor files. The name of the two columns are respectively normal and tumor. The extensions of the files (.CEL for example) should be removed.

If you have specified `createArchitecture=TRUE` and have given the required parameters, the function will create the architecture and copy your files in the right folder. After the creation of the aroma architecture, it runs the normalization process and saves in the aroma architecture all the files necessary to obtain the allele B fraction and the copy-number signal.

In the case of a study without reference, only parameters `dataSetName`, `chipType`, `createArchitecture` and `savePlot` have to be specified. In the case of a normal-tumor study, TumorBoost process is executed. If you want a CEL file to be used as reference for all the tumor data, then fill the normal column of the `normalTumorArray` with the filename of your reference for all files.

4.3 Accessing the copy-number, allele B fraction and genotype

After running the `signalPreProcess` function, you can not directly access the copy-number and the allele B fraction signals with the files in the architecture, a treatment is required to read them. Note that the copy-number signal saved is the raw signal, it needs a reference to normalize it. With the `getCopyNumberSignal` and `getFracBSignal` functions, you can access the copy-number and the allele B fraction profiles.

The `getCopyNumberSignal` allows you to extract the copy-number signal a chromosome at a time. In the case of a study without reference, the median of all the signals of the data-set is used to normalize every profile. In the normal-tumor study, the normal signal is used to normalize the tumor signal.

The getter for the copy-number profile is defined as follows:

```
getCopyNumberSignal(dataSetName,chromosome,normalTumorArray,onlySNP,listOfFiles,verbose)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`chromosome`	A vector containing the chromosomes for which the signal will be extracted.
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`onlySNP`	If TRUE, only the copy-number for SNPs positions will be returned (default=FALSE).
`listOfFiles`	A vector containing the names of the files in dataSetName folder for which the copy-number profiles will be extracted (default is all the files).
`verbose`	If TRUE print some information (default=TRUE).

This function returns a list of size `length(chromosome)` (each element is named `chrX` with `X` the number of the chromosome) containing a data.frame with columns:

- **chromosome** Chromosome of the signal.
- **position** Positions associated with the copy-number.
- **copynumber** Copy number profiles of selected files; the name of each column is the name of the associated data file name.
- **featureNames** Names of the probes.

If you want the allele B fraction, you can use the `getFracBSignal` function. In a normal-tumor study, allele B fraction for both normal and tumoral samples is returned. For the tumoral sample, it is the allele B fraction after the TumorBoost normalization.

```
getFracBSignal(dataSetName,chromosome,normalTumorArray,listOfFiles,verbose)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`chromosome`	A vector containing the chromosomes for which the allele B fraction signal must be extract.
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`listOfFiles`	A vector containing the names of the files in dataSetName folder for which the allele B fraction profiles will be extracted (default is all the files).
`verbose`	If TRUE print some information (default=TRUE).

This function returns a list of size `length(chromosome)` (each element is named `chrX` with `X` the number of the chromosome) containing a data.frame with columns:

- **chromosome** Chromosome of the signal.
- **position** Positions associated with the allele B fraction.
- **fracB** Allele B fraction profiles of selected files; the name of each column is the name of the associated data file name.
- **featureNames** Names of the probes.

If you want the genotype call, you can use the `getGenotypeCalls` function.

```
getGenotypeCalls(dataSetName,chromosome,listOfFiles=NULL,verbose=TRUE)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`chromosome`	A vector containing the chromosomes for which the genotype call will be extracted.
`listOfFiles`	A vector containing the names of the files in dataSetName folder for which the genotype signal will be extracted (default is all the files).
`verbose`	If TRUE print some information (default=TRUE).

This function returns a list of size `length(chromosome)` (each element is named `chrX` with `X` the number of the chromosome) containing a data.frame with columns:

- **chromosome** Chromosome of the signal.
- **position** Positions associated with the genotype.
- **genotype** Genotype calls corresponding to selected files; the name of each column is the name of the associated data file name.
- **featureNames** Names of the probes.

The next getter, `getSymFracBSignal`, returns the symmetrized allele B fraction only for heterozygous positions. To symmetrize the allele B fraction the transformation $x \mapsto 2 * |x - 0.5|$ is applied. It centers the data in 0.5 corresponding to heterozygous allele B fraction then symmetrize and multiply by 2 to have a signal between 0 and 1

`getSymFracBSignal(dataSetName,chromosome,normalTumorArray,file,verbose=TRUE)`

Parameters	Description
`dataSetName`	The name of the data-set folder.
`chromosome`	A vector with the chromosomes for which the symetrized signal will be extracted .
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`file`	The name of the file in dataSetName folder for which the symetrized signal will be extracted .
`verbose`	If TRUE print some information (default=TRUE).

This function returns a list of size `length(chromosome)` (each element is named `chrX` with `X` the number of the chromosome) containing a data.frame with columns:

- **chromosome** Chromosome of the signal.
- **position** Positions associated with the genotype.
- **fracB** One column named by the data file name. It contains the symmetrized allele B fraction signal for the specified profile.
- **featureNames** Names of the probes.

4.4 Usage

We refer to sections 3.2 and 3.3 to normalize input data without or with control samples in the study, respectively.

Get a signal

The following commands enable to extract the copy-number or the allele B fraction profiles of all the files for the chromosome 5:

```
> #normal-tumor study
> CNdata2=getCopyNumberSignal("datatest2",5,normalTumorArray=normalTumorArray,TRUE)
> fracBdata2=getFracBSignal("datatest2",5,normalTumorArray=normalTumorArray)
> symFracB2=getSymFracBSignal("datatest2",5,
+ file="GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218",
+ normalTumorArray=normalTumorArray)
> #study without reference
> CNdata1=getCopyNumberSignal("datatest1",5,onlySNP=TRUE)
> fracBdata1=getFracBSignal("datatest1",5)
> symFracB1=getSymFracBSignal("datatest1",5,
+ file="GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218")
```

To access the copy-number or allele B fraction profiles, run:

```
> CNdata2$chr5
> fracBdata2$chr5$tumor
> fracBdata2$chr5$normal
> fracBdata1$chr5$tumor
> symFracB1$chr5
> symFracB2$chr5
```

5 Segmentation

In the package `MPAgenomics`, we provide a segmentation method, which uses the PELT method of the `cpt.mean` function from `changepoint` package [9].

In the initial PELT method, the penalty is $\log(n)$ with n the length of the signal. Here we will run the PELT method with a penalty $\lambda \times \log(n)$ for a range of λ. Then, we plot the evolution of the number of segments with regards to the λ parameter (Fig 9).

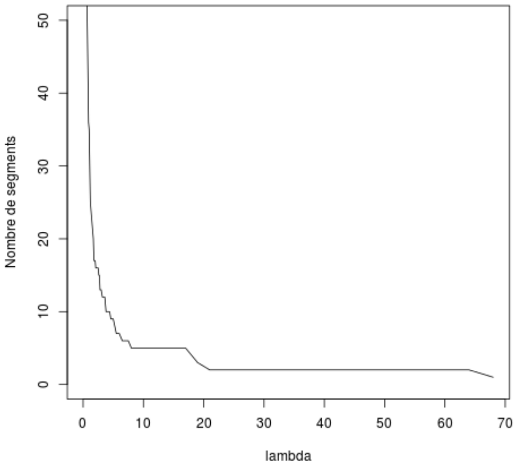


Figure 9: Variation of the number of segment with regards to lambda.

We want to adjust the λ parameter to have a reasonable number of segments. So, we look for the greatest stabilization of the number of segments. Indeed, our intuition is that we can be confident in the found breakpoints because the penalty has to increase significantly to remove some breakpoints.

5.1 From data normalized by aroma

The procedure described above can be run by the `PELTaroma` function if the working directory respects the aroma architecture.

```
PELTaroma(dataSetName,normalTumorArray,chromosome,Lambda,listOfFiles,onlySNP=TRUE,savePlot=TRUE)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`chromosome`	A vector with the chromosomes to be segmented.
`Lambda`	A vector containing all the penalization values to test for the segmentation. If no values are provided, default values will be used.
`onlySNP`	If TRUE, only the copy-number for SNPs positions will be returned (default=TRUE).
`listOfFiles`	A vector containing the names of the files in dataSetName folder for which the copy number profiles will be segmented (default is all the files).
`savePlot`	if TRUE, graphics of the segmented CN signal will be saved in the `figures/dataSetName/segmentation/CN` folder. (default=TRUE).

If `savePlot=TRUE`, some graphics will be plotted (see Fig 10).

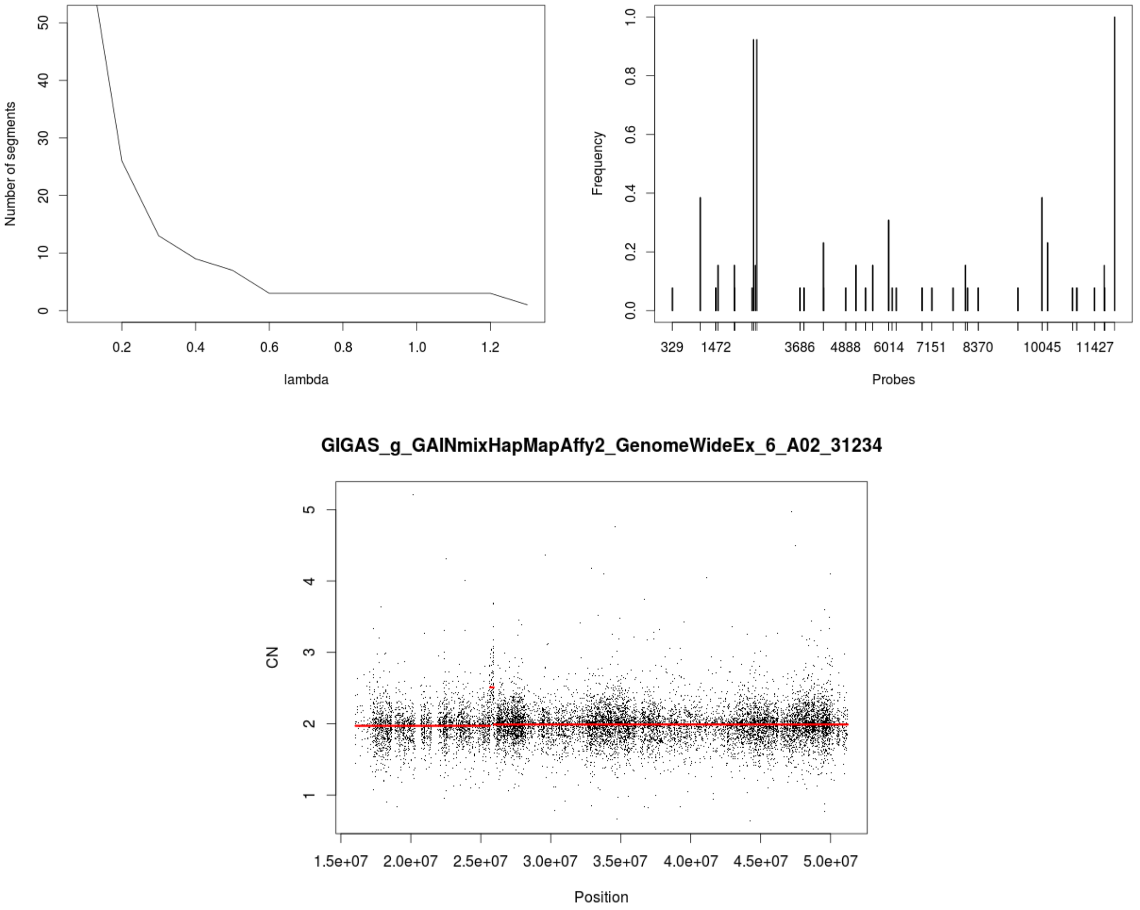
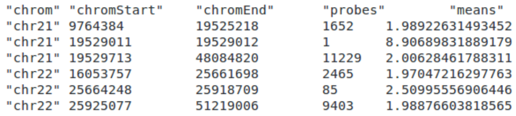


Figure 10: Top left: variation of the number of segment with regards to λ. Top right: the frequency of occurrence of breakpoints in all segmentation. Bottom: Segmentation with the optimal λ found.

The graphics of the segmented signal are saved in the `figures/dataSetName/segmentation` folder. A summary of the segmented profile is saved in a text file (Fig. 11) in the `segmentation/dataSetName/` folder, it contains 5 columns : chrom, chromStart, chromEnd, probes and means containing the number of the chromosome, the starting position of the segment, the ending position of the segment, the number of probes in the segment and the mean of the segment.



```
"chrom" "chromStart"    "chromEnd"      "probes"        "means"
"chr21" 9764384         19525218        1652    1.98922631493452
"chr21" 19529011        19529012        1       8.90689831889179
"chr21" 19529713        48084820        11229   2.00628461788311
"chr22" 16053757        25661698        2465    1.97047216297763
"chr22" 25664248        25918709        85      2.50995556906446
"chr22" 25925077        51219006        9403    1.98876603818565
```

Figure 11: Summary of a segmentation for a profile.

The output of the function is a list where every element of the list contains the segmentation results for a different signal formatted in a list containing:

- **copynumber** A vector containing the copy-number signal.
- **segmented** A vector of the same size as copynumber containing the segmented values.
- **startPos** The position of every probe.
- **chromosome** A vector of the same size as copynumber containing the chromosome number.
- **featureNames** Names of the probes.
- **sampleNames** The name of the signal.
- **segment** A data.frame that sums up the results of the segmentation. Each row is a different segment with the chromosome, start position, end position, number of probes in the segment and the value of the segment.

The output for a profile is formatted to be used in input of the `callingProcess` function (see section 6.2) but if you use the aroma architecture and want to label your segments, you can use directly the `cnSegCallingProcess` function (see section 6.1). This function runs the segmentation and the calling processing.

For example, run the segmentation processing on one file of the downloaded data.

```
> file="GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234"
> seg1=PELTaroma("datatest1",chromosome=21:22,onlySNP=TRUE,plot=TRUE,
+ listOfFiles=file)
```

You will obtain the results shown in figures 10 & 11 for one profile.

You can also segment the symmetrized allele B fraction with the function `segFracBSignal`:

```
segFracBSignal(dataSetName,normalTumorArray,chromosome=1:22,Lambda=NULL,listOfFiles=NULL,
savePlot=TRUE,verbose=TRUE)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`chromosome`	A vector with the chromosomes to be segmented.
`Lambda`	A vector containing all the penalization values to test for the segmentation. If no values are provided, default values will be used.
`listOfFiles`	A vector containing the names of the files in dataSetName folder for which the allele B profile is segmented (default is all the files).
`savePlot`	if TRUE, graphics of the segmented allele B profile will be saved in the `figures/dataSetName/segmentation/fracB` folder. (default=TRUE).
`verbose`	If TRUE, print some information (default=TRUE).

The output of the function is a data.frame containing the segmentation results, each row corresponding to a different segment and with columns:

- **sampleName** Name of the file the segment belongs to.
- **chromosome** Chromosome the segment belongs to.
- **chromStart** Starting position of the segment.
- **chromEnd** Ending position of the segment.
- **probes** Number of probes in the segment.
- **means** Mean of the segment.

You can run the following code to test the function

```
> #run the segmentation
> segfracB=segFracBSignal("datatest1",chromosome=c(1,5))
> #print summary of segmentation
> segfracB
```

5.2 For any data provided in matrix

If you have your own normalized data stored in a matrix, you can use the `PELT` function to perform the PELT segmentation with the parameter choice proposed in `MPAgenomics`.

```
PELT(signal,Lambda,position=NULL,plot=TRUE,verbose=TRUE)
```

Parameters	Description
`signal`	A vector containing the signal.
`Lambda`	A vector containing all the penalization values to test for the segmentation. If no values are provided, default values will be used.
`position`	A vector containing the position of all elements of the signal (not necessary).
`plot`	If TRUE, plot some graphics (default=TRUE).
`verbose`	If TRUE, print some information (default=TRUE).

If `plot=TRUE`, some graphics will be plotted (see Fig 10).
The output of the function is a list containing:

- **signal** A vector containing the signal.
- **segmented** A vector of the same size as the signal containing the segmented values.
- **startPos** The position of each probe.
- **segment** A data.frame that sums up the results of the segmentation. Each row is a different segment with the start position, end position, number of points in the segment and the value of the segment.

For example, assume you already have done the preprocessing of your data (see section 4.2). You want to do the segmentation of the copy-number signal of one chromosome for one patient.
First, get the signal :

```
> file="GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314"
> CNdata1=getCopyNumberSignal("datatest1",20,onlySNP=TRUE,listOfFiles=file)
> copyNumber=CNdata1$chr20$GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314
> position=CNdata$chr20$position
```

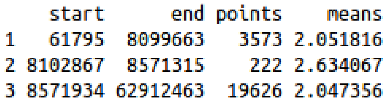
Now, run the segmentation process :

```
> seg=PELT(copyNumber,position=position,plot=TRUE,verbose=TRUE)
```

You obtain the plots seen previously (Fig 10) and the results in the right format for the calling function (see section 6.2).

You can access to the list of found segments with :

```
> seg$segment
```



```
    start      end points    means
1   61795  8099663   3573 2.051816
2 8102867  8571315    222 2.634067
3 8571934 62912463  19626 2.047356
```

6 Calling aberrations in copy-number profiles

Once you have a segmentation of your copy-number profile, you may want to assign a label (*loss*, *normal* or *gain*) to each segment. This can be done with the `CGHcall` method [14]. A wrapper of the CGHcall calling process is available in `MPAgenomics`.

Starting with your signal and the associated segmented signal, the method assumes that the segmented values follow a mixture of normals. The variables of the model are estimated with an EM algorithm with a specific initialization. Then, every segment is called by the most likely label according to the model.

6.1 From data normalized by aroma

The `cnSegCallingProcess` function executes the segmentation (see section 5) and the calling process. The specified data will be automatically imported by the function.

```
cnSegCallingProcess(dataSetName,normalTumorArray,chromosome,Lambda,listOfFile,onlySNP,savePlot,
nclass,cellularity,...)
```

Parameters	Details
`dataSetName`	name of the data-set folder in the rawData folder containing the signals to use.
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`chromosome`	A vector containing the chromosome to segment.
`Lambda`	A vector containing all the penalization values to test for the segmentation. If no values are provided, default values will be used.
`listOfFiles`	A vector containing the file names from the dataSetName to use.
`onlySNP`	If TRUE, only the SNP probes will be used.
`savePlot`	If TRUE, print some graphics (default=TRUE).
`nclass`	The number of levels to be used for calling. Either 3 (loss, normal, gain), 4 (including amplifications), 5 (including double deletions) (default=3).
`cellularity`	Percentage of tumor cells in the sample (default=1).
`...`	Others parameters for CGHcall function [13].

This function will save every segment in a text file in the `segmentation` folder in the working directory and return a data.frame with columns:

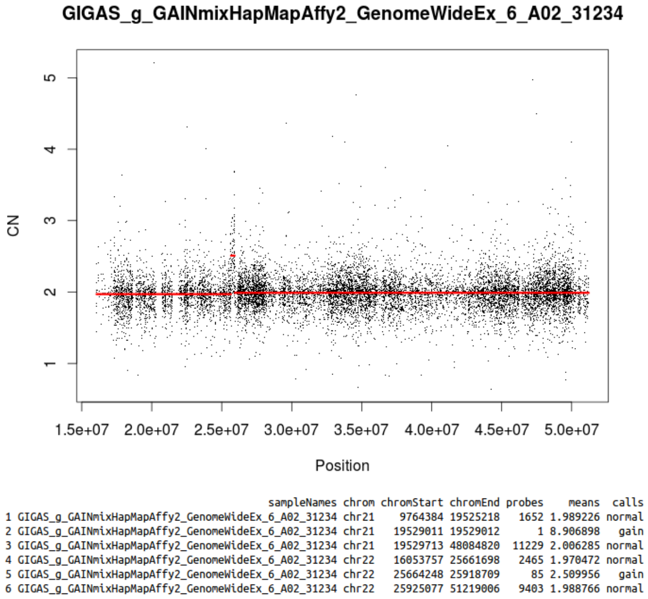
- **sampleNames** Name of the file.
- **chrom** The chromosome of the segment.
- **chromStart** The starting position (in bp) of the segment. This position is not included in the segment.
- **chromEnd** The ending position (in bp) of the segment. This position is included in the segment.
- **probes** Number of probes in the segment.

- **means** Mean of the segment.
- **calls** The calling of the segment (”double loss”, ”loss”, ”normal”, ”gain” or ”amplification”).

The graphic of the segmented signal is saved in the `figures` folder of the architecture.

```
> seg2=cnSegCallingProcess("datatest1",chromosome=21:22)
```

The output of the function is shown in the figure 12.



```
                                   sampleNames chrom chromStart chromEnd probes    means  calls
1 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr21    9764384 19525218   1652 1.989226 normal
2 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr21   19529011 19529012      1 8.906898   gain
3 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr21   19529713 48084820  11229 2.006285 normal
4 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr22   16053757 25661698   2465 1.970472 normal
5 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr22   25664248 25918709     85 2.509956   gain
6 GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 chr22   25925077 51219006   9403 1.988766 normal
```

Figure 12: Top: graphics saved in the `figures` folder. Bottom: output of the function (saved in a text file too).

6.2 For any data provided in matrix

The function `callingProcess` executes all the calling process and can be used without the aroma architecture. You can use this function with the segmentation method provided in the package or any other methods as long as you give the results of the segmentation in the right format (see `segmentData` argument).

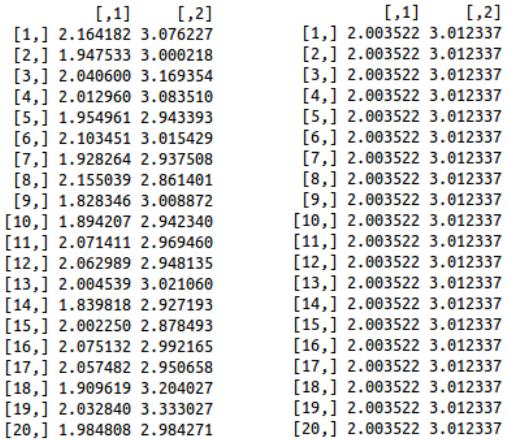
```
callingProcess(segmentData,nclass=5,cellularity=1,...)
```

Parameters	Description
`segmentData`	A list (see details below).
`nclass`	Number of labels. 3 corresponds to ”loss”, ”normal” and ”gain”, 4 adds ”amplification” and 5 adds ”double loss”.
`cellularity`	Percentage of tumor cells in the sample (default=1).
`verbose`	If TRUE, print some information.
`...`	Others parameters for CGHcall function [13].

`segmentData` is a list containing :

- **copynumber** A matrix. Every column represents a copy-number signal for a different sample.
- **segmented** A matrix of the same size as `copynumber` containing the segmented values of the copy-number signal.
- **chromosome** A vector, of length the number of rows of `copynumber` matrix, containing the number of the chromosome for each position.

- **startPos** A vector, of length the number of rows of `copynumber` matrix, containing the genomic position of each probe.
- **featureNames** A vector, of length the number of rows of `copynumber` matrix, containing the name of each probe.
- **sampleNames** A vector, of length the number of columns of `copynumber` matrix, containing the name of each file.



```
           [,1]     [,2]                  [,1]     [,2]
 [1,] 2.164182 3.076227         [1,] 2.003522 3.012337
 [2,] 1.947533 3.000218         [2,] 2.003522 3.012337
 [3,] 2.040600 3.169354         [3,] 2.003522 3.012337
 [4,] 2.012960 3.083510         [4,] 2.003522 3.012337
 [5,] 1.954961 2.943393         [5,] 2.003522 3.012337
 [6,] 2.103451 3.015429         [6,] 2.003522 3.012337
 [7,] 1.928264 2.937508         [7,] 2.003522 3.012337
 [8,] 2.155039 2.861401         [8,] 2.003522 3.012337
 [9,] 1.828346 3.008872         [9,] 2.003522 3.012337
[10,] 1.894207 2.942340        [10,] 2.003522 3.012337
[11,] 2.071411 2.969460        [11,] 2.003522 3.012337
[12,] 2.062989 2.948135        [12,] 2.003522 3.012337
[13,] 2.004539 3.021060        [13,] 2.003522 3.012337
[14,] 1.839818 2.927193        [14,] 2.003522 3.012337
[15,] 2.002250 2.878493        [15,] 2.003522 3.012337
[16,] 2.075132 2.992165        [16,] 2.003522 3.012337
[17,] 2.057482 2.950658        [17,] 2.003522 3.012337
[18,] 1.909619 3.204027        [18,] 2.003522 3.012337
[19,] 2.032840 3.333027        [19,] 2.003522 3.012337
[20,] 1.984808 2.984271        [20,] 2.003522 3.012337
```

Figure 13: Left : example of `copynumber` matrix for 2 samples. Right : example of the associated `segmented` matrix.

This function returns a list with the same elements as `segmentData` and some supplementary elements :

- **calls** A matrix, of the same size as `copynumber` matrix, containing the label of each point.
- **segment** A data.frame that sums up all the segments found.
- **probdloss** (if you ran `callingProcess` with `nclass`=5) A matrix of the same size as `copynumber` matrix. It contains the probability for each segmented copy-number to be a double loss.
- **probloss** A matrix of the same size as `copynumber` matrix. It contains the probability for each segmented copy-number to be a loss.
- **probdnorm** A matrix of the same size as `copynumber` matrix. It contains the probability for each segmented copy-number to be normal.
- **probdgain** A matrix of the same size as `copynumber` matrix. It contains the probability for each segmented copy-number to be a gain.
- **probdamp** (if you have run `callingProcess` with `nclass`=4 or 5) A matrix of the same size as `copynumber` matrix. It contains the probability for each segmented copy-number to be an amplification.

A wrapper is provided for creating the `segmentData` argument :

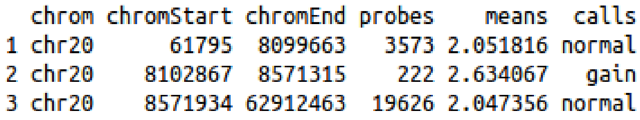
```
callingObject(copynumber, segmented, chromosome, position, featureNames, sampleNames)
```

The parameters `featureNames` and `sampleNames` can be omitted.

The following lines continue the example of the section 5. They create the `segmentData` list and run the calling process:

```
> #create the segmentData object
> callobj= callingObject(copynumber=seg$signal, segmented=seg$segmented,
+  chromosome=rep(20,length(seg$signal)), position=seg$startPos,
+  sampleNames="sample1")
> #run the calling
> call=callingProcess(callobj,nclass=3,cellularity=1,verbose=TRUE)
> call$segment
```

The result of the calling can be seen in figure 14.



```
  chrom chromStart chromEnd probes    means  calls
1 chr20      61795  8099663   3573 2.051816 normal
2 chr20    8102867  8571315    222 2.634067   gain
3 chr20    8571934 62912463  19626 2.047356 normal
```

Figure 14: The summation of all the segments after running the function `callingProcess`.

A wrapper `CNAobjectToCGHcallObject` to convert CNA object from DNAcopy package [11] (CBS segmentation) to the desired format (see `segmentData`) is also provided.

```
CNAobjectToCGHcallObject(CNAobject)
```

Parameters	Description
`CNAobject`	Output of the `segment` function from `DNAcopy` package.

6.3 `filterSeg` function

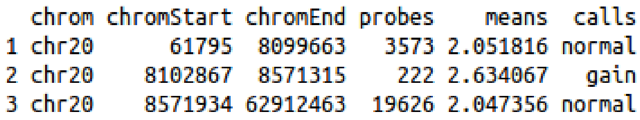
At the end of the calling process, if you have a lot of segments, you might want to filter some uninteresting segments. The `filterSeg` function allows to filter segments from 3 criteria : the length in bp, the size in probes and the calling.

```
filterSeg(segmentList,minLength=1,minProbes=1,normalFilter=TRUE)
```

Arguments	Description
`segmentList`	A data.frame containing a description of segments, it must have at least columns : ”chromStart”, ”chromEnd”, ”probes” and ”calls” (see the output of `cnSegLabelProcess` and `callingProcess` functions).
`minLength`	The minimum length (in bp) for a segment. All the shorter segments are removed.
`minProbes`	The minimum number of probes for a segment. All the segments with less probes are removed.
`keptLabel`	Vector of label to keep. Only segment with one of the specified label will be kept.

It returns a data.frame of the same format as `segmentList` without the filtered segments.

We assume to have a data.frame called `call$segment` containing the columns plot in Fig. 15



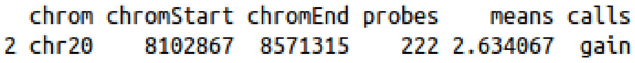
```
  chrom chromStart chromEnd probes    means  calls
1 chr20      61795  8099663   3573 2.051816 normal
2 chr20    8102867  8571315    222 2.634067   gain
3 chr20    8571934 62912463  19626 2.047356 normal
```

Figure 15: `call$segment` before filtering.

The following command executes the code to keep only segments presenting a gain.

```
> segmentfilter=filterSeg(call$segment,keptLabel="gain")
> segmentfilter
```

Obtained results are shown in Fig. 16



```
  chrom chromStart chromEnd probes    means calls
2 chr20    8102867  8571315    222 2.634067  gain
```

Figure 16: Result of the filtering.

7 Selection of genomic markers

In this section, the goal is to select some relevant markers according to a response.

It consists in minimizing $\beta \in \mathbb{R}^P \mapsto g(\beta)$, where

$$g_\rho(\beta) = \sum_{i=1}^{I} (y_i - (X\beta)_i)^2 + \rho \sum_{p=1}^{P} |\beta_p| \ ,$$

with $(X\beta)_i = \sum_p x_{i,p}\beta_p$ and $\rho > 0$ controlling the number of non-zero coordinates of β. After minimization, non-zero coefficients β_p correspond to influential positions to predict the response.

To solve this problem, we use the LARS algorithm [6] that solves the lasso problem for all the values of t. In order to choose the best solution, we use a cross validation to select the best values of t and return the associated solution.

7.1 From data normalized by aroma

The function `markerSelection` will extract the data from the aroma architecture and run the LARS algorithm and the cross-validation for each chromosome separately.

```
markerSelection(dataSetName,dataResponse,chromosome,signal,normalTumorArray,onlySNP,nbFolds,
loss,plot,...)
```

Parameters	Description
`dataSetName`	The name of the data-set folder.
`dataResponse`	A csv files or a data.frame with 2 columns : ”files” and ”response”. The column ”files” contains the filename to extract and the second column the response associated with the file.
`chromosome`	A vector containing the number of the chromosomes for the SNPs selection.
`signal`	”CN” or ”fracB”, corresponding to which signal will be analyzed (default=”CN”).
`normalTumorArray`	Only in the case of normal-tumor study. A csv file or a data.frame containing the mapping between normal and tumor files(Fig. 8).
`onlySNP`	Only if `signal="CN"`. If TRUE, only the SNPs probes are used (default=FALSE).
`nbFolds`	Number of folds for the cross-validation (default=10).
`loss`	either ”logistic” (binary response) or ”linear” (quantitative response), default is ”logistic”.
`plot`	If TRUE, cross-validation mean squared error is plotted (default=TRUE).
`...`	supplementary arguments for `cv.glmnet` function [7] in case of logistic loss or for `HDlars` function for linear loss.

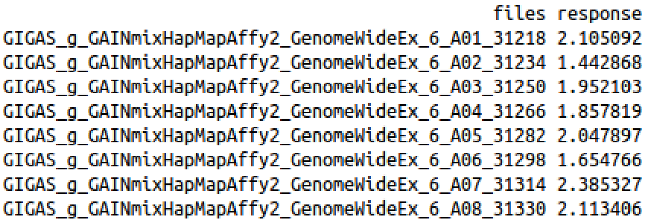
For `signal="fracB"`, the selection of markers is done for the tumor allele B fraction.

The function returns a list containing as many elements as the number of chromosomes specified in parameters. The name `chrX`, where X is the number of a chromosome. Each element of this list is a list containing:

- **chr** The chromosome corresponding to the signal.
- **markers.index** A vector containing the index of all selected markers.
- **markers.position** A vector containing the position of all selected markers.
- **markers.names** A vector containing the name of all selected markers.

- **coefficient** A vector containing the coefficients ($\hat{\beta}$) of all selected markers.
- **intercept** Intercept of the model.

For the following example, we assume that the architecture is already created and the data-set is preprocessed as in the previous examples (see section 4.1 and 4.2). The response is stored in a data.frame named `dataResponse` (see Fig. 17).



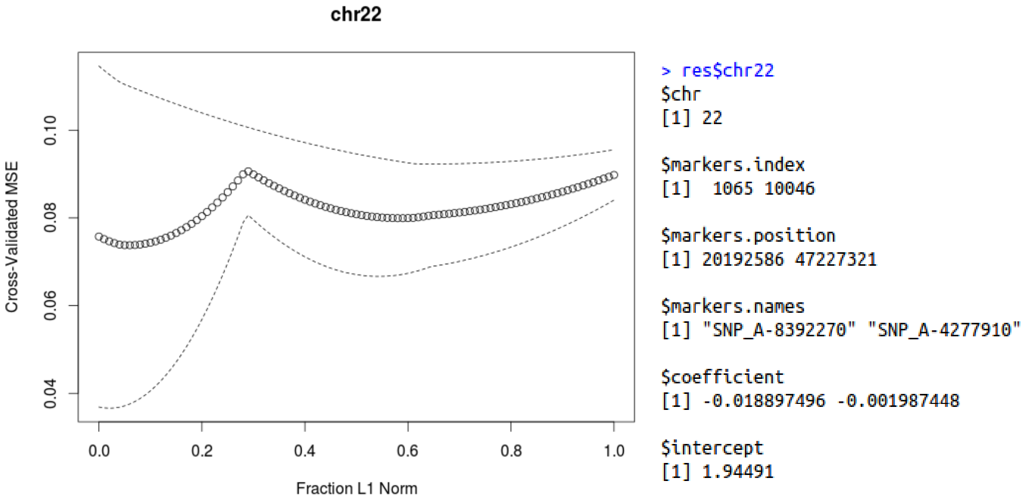
```
                                                      files response
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A01_31218 2.105092
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A02_31234 1.442868
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A03_31250 1.952103
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A04_31266 1.857819
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A05_31282 2.047897
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A06_31298 1.654766
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A07_31314 2.385327
GIGAS_g_GAINmixHapMapAffy2_GenomeWideEx_6_A08_31330 2.113406
```

Figure 17: Example of `dataResponse` parameters for the `markerSelection` function.

The relevant markers can be selected by executing the following code:

```
> dataResponse=data.frame(files=getListOfFiles("datatest1"),
+ response=c(2.105092,1.442868,1.952103,1.857819,2.047897,1.654766,2.385327,2.113406))
> res=markerSelection(dataSetName="datatest1",dataResponse,chromosome=21:22,signal="CN",
+ onlySNP=TRUE,loss="linear")
```

Figure 18 shows the output of the function.



```
> res$chr22
$chr
[1] 22

$markers.index
[1]  1065 10046

$markers.position
[1] 20192586 47227321

$markers.names
[1] "SNP_A-8392270" "SNP_A-4277910"

$coefficient
[1] -0.018897496 -0.001987448

$intercept
[1] 1.94491
```

Figure 18: Left : cross-validation mean squared error. Right: results for chromosome 22.

In this example, two markers are selected : the 1065th and 10046th markers of the signal (markers.index) at positions 20192586 pb and 47227321 pb on the chromosome 22. The coefficient associated to each selected marker are given in coefficient vector.

7.2 For any data provided in matrix

A version of the selection of markers without the aroma architecture is available:

```
variableSelection(dataMatrix,dataResponse,nbFolds,plot)
```

Parameters	Description
`dataMatrix`	A matrix containing the data, each row is a different signal.
`dataResponse`	A vector containing the response associated with each signal.
`nbFolds`	Number of folds for the cross-validation (default=10).
`loss`	either ”logistic” (binary response) or ”linear” (quantitative response), default is ”logistic”.
`plot`	If TRUE plot cross-validation mean squared error (default=TRUE).
`...`	supplementary arguments for `cv.glmnet` function [7] in case of logistic loss or for `HDlars` function for linear loss.

The output of the functions is:

- **variable** A vector containing the index of all selected markers.
- **coefficient** A vector containing the coefficients ($\hat{\beta}$) of all selected variables.
- **intercept** Intercept of the model.

For example, simulate a data-set and the associated response

```
> dataMatrix=matrix(rnorm(5000,0,0.5),nrow=50)
> dataResponse=drop(dataMatrix%*%sample(c(rep(0,90),rep(1,10))))
> res=variableSelection(dataMatrix,dataResponse,nbFolds=5,loss="linear",plot=TRUE)
```

References

[1] H. Bengtsson. aroma - An R Object-oriented Microarray Analysis environment. Preprint in Mathematical Sciences 2004:18, Mathematical Statistics, Centre for Mathematical Sciences, Lund University, Sweden, 2004.

[2] H. Bengtsson, R. Irizarry, B. Carvalho, and T. Speed. Estimation and assessment of raw copy numbers at the single locus level. Bioinformatics, 24:759–767, 2008.

[3] H. Bengtsson, P. Neuvial, and T. P. Speed. Tumorboost: Normalization of allele-specific tumor copy numbers from a single pair of tumor-normal genotyping microarrays. BMC Bioinformatics, 11, 2010.

[4] H. Bengtsson, K. Simpson, J. Bullard, and K. Hansen. aroma.affymetrix: A generic framework in R for analyzing small to very large Affymetrix data sets in bounded memory. Technical Report 745, Department of Statistics, University of California, Berkeley, February 2008.

[5] H. Bengtsson, P. Wirapati, and T. P. Speed. A single-array preprocessing method for estimating full-resolution raw copys from all affymetrix genotyping arrays including genomewidesnp 5 & 6. Bioinformatics, 25(17):2149–2156, 2009.

[6] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani. Least angle regression. Annals of Statistics, 32:407–499, 2004.

[7] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent, 2009.

[8] Q. Grimonprez, A. Celisse, M. Cheok, M. Figeac, and G. Marot. Mpagenomics : An r package for multi-patients analysis of genomic markers, 2014. Preprint http://hal.inria.fr/hal-00933614.

[9] R. Killick and I. Eckley. changepoint: An R package for changepoint analysis, 2013. R package version 1.1.

[10] R. Killick, P. Fearnhead, and I. A. Eckley. Optimal detection of changepoints with a linear computational cost. Journal of the American Statistical Association, 107(500):1590–1598, 2012.

[11] V. E. Seshan and A. Olshen. DNAcopy: DNA copy number data analysis. R package version 1.34.0.

[12] R. Tibshirani. Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society, Series B, 58:267–288, 1994.

[13] M. van de Wiel and S. Vosse. CGHcall: Calling aberrations for array CGH tumor profiles., 2012. R package version 2.20.0.

[14] M. A. van de Wiel, K. I. Kim, S. J. Vosse, W. N. van Wieringen, S. M. Wilting, and B. Ylstra. CGHcall: calling aberrations for array CGH tumor profiles. Bioinformatics, 23(7):892–894, 2007.