# sequenza usage example

Francesco Favero[*], Tejal Joshi, Andrea M. Marquard, Aron C. Eklund

April 4, 2014

## Contents

## 1 Abstract

Deep sequence of tumor DNA along with corresponding normal DNA can provide a valuable perspective on the mutations and aberrations that characterize the tumor. However, analysis of this data can be impeded by tumor cellularity and heterogeneity and by unwieldy data. Here we describe *Sequenza*, which comprises a fast python-based pre-processor and an R-based analysis

[*]favero@cbs.dtu.dk

package. *Sequenza* enables the efficient estimation of tumor cellularity and ploidy, and generation of copy number, loss-of-heterozygosity, and mutation frequency profiles.

This document details a typical analysis of matched tumor-normal exome sequence data using *sequenza*.

## 2 Getting started

### 2.1 Minimum requirements

- Software: R, Python
- Operating system: Linux, OS X, Windows
- Memory: Minimum 4 GB of RAM. Recommended >8 GB.
- Disk space: 1.5 GB for sample
- R version: 2.15.1
- Python version: 2.7 (or Pypy 2.*)

### 2.2 Installation

In order to install sequenza, you can download the package from the nearest CRAN mirror doing:

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

### 2.3 Note on the helper program `sequenza-utils.py`.

For convenience and efficiency we have implemented pre-processing algorithms in a standalone (not called from R) Python program. This program is provided with the R package; its exact location can be found like this:

```
> system.file("exec", "sequenza-utils.py", package="sequenza")
```

You may wish to copy this program to a location on your path.

### 2.4 Workflow overview

A typical workflow developed with Sequenza on pre-aligned sequencing files (BAM format) is structured as follows:

1. Convert pileup to *seqz* format
2. GC normalization
3. Allele-specific segmentation using the depth ratio and the B allele frequencies (BAFs)
4. Infer cellularity and ploidy by model fitting
5. Call CNV and variant alleles

## 3 Preparing inputs for Sequenza

In order to obtain precise mutational and aberration patterns in a tumor sample, Sequenza requires a matched normal sample from the same patient. Typically, the following files are needed to get started with Sequenza.

1. A BAM file (or a derived pileup file) from the tumor specimen.
2. A BAM file (or a derived pileup file) from the normal specimen.
3. A FASTA reference genomic sequence file (to extract GC-content information, and to transform BAM to pileup if needed.)

Alternatively, it is possible to use the output of VarScan2[1] (`http://varscan.sourceforge.net`), which would require a similar approach and the generation of pileups as well. In this case, you can skip ahead to the "Converting VarScan2 output to seqz" section below.

The genome sequence file can be obtained from e.g. `http://hgdownload.cse.ucsc.edu/downloads.html` or `http://www.ensembl.org/info/data/ftp/index.html`.

### 3.1 Generating pileup files from BAM files

We recommend using pre-processed and quality-filtered BAM files to obtain pileup calls for both samples.

Pileup files can be generated using `samtools`[2]. For example:

```
samtools mpileup -f hg19.fasta -Q 20 normal.bam | gzip > normal.pileup.gz
samtools mpileup -f hg19.fasta -Q 20 tumor.bam | gzip > tumor.pileup.gz
```

### 3.2 Generating a genome-wide GC content file

The genome-wide GC content is used to normalize the depth ratio. To obtain the GC content file, you can use a function from `sequenza-utils.py` to extract the average GC content from a genome FASTA file using fixed genomic windows.

The following example calculates GC content in 50-base windows:

```
sequenza-utils.py GC-windows -w 50 hg19.fa | gzip > hg19.gc50Base.txt.gz
```

Alternatively, you can download the gc5Base file from golden path (`http://hgdownload-test.cse.ucsc.edu/goldenPath/hg19/gc5Base/`).

### 3.3 Generate a seqz file

A *seqz* file contains genotype information, alleles and mutation frequency, and other features. This file is used as input for the R-based part of Sequenza.

The seqz file is generated like this:

```
sequenza-utils.py pileup2seqz -gc hg19.gc50Base.txt.gz \
                              -r normal.pileup.gz \
                              -s tumor.pileup.gz | gzip > out.seqz.gz
```

Alternatively, if you haven't already generated the pileup files, and you are not interested in storing the pileup for further use, you may want to create two named pipes (*FIFOs*) directing `samtools` output directly into `sequenza-utils.py`:

```
mkfifo normal.fifo tumor.fifo
samtools mpileup -f hg19.fasta -Q 20 normal.bam > normal.fifo &
samtools mpileup -f hg19.fasta -Q 20 tumor.bam > tumor.fifo &
sequenza-utils.py pileup2seqz -gc hg19.gc50Base.txt.gz \
                              -r normal.fifo \
                              -s tumor.fifo | gzip > out.seqz.gz
rm normal.fifo tumor.fifo
```

To further compress the seqz file, it is possible to use a binning function provided in `sequenza-utils.py`. This binning decreases the memory requirement to load the data into R, and it also speeds up the processing of the sample:

```
sequenza−utils.py seqz−binning −w 50 \
                                        out.seqz.gz | gzip > out_small.seqz.gz
```

Where the parameter *-w* indicate a window size (in bases), to be used for the binning. The heterozygous positions and the positions carrying variant calls remain untouched.

## 3.4 Converting VarScan2 output to seqz

Since many projects might already have been processed with VarScan2, it can be convenient to be able to import such results. For this purpose a simple function is provided within the package, to convert the output of the *somatic* and *copynumber* programs of the VarScan2 suite into the *seqz* format.

```
> snp <- read.table("varscan.snp", header = TRUE, sep = "\t")
> cnv <- read.table("varscan.copynumber", header = TRUE, sep = "\t")
> seqz.data <- VarScan2seqz(varscan.somatic = snp, varscan.copynumber = cnv)
> write.table(seqz.data, "my.sample.seqz", col.names = TRUE, row.names = FALSE, sep = "\t")
```

For *whole genome* sequencing, the information in the *varscan.snp* could be enough to estimate the ploidy and cellularity, and define the copy number and mutations; hence the `varscan.copynumber` argument is optional. For *exome* sequencing, it is strongly suggested to supply `varscan.copynumber`.

# 4 Exploring the *seqz* file and GC-correction details

After the aligned sequence data have been pre-processed, the `sequenza` R package handles all the normalization and analysis steps. Thus, the remainder of this vignette will take place in R.

```
> library("sequenza")
```

## 4.1 Read the *seqz* file

In the package we provide an example file. To find the complete path of the example data file:

```
> data.file <-  system.file("data", "example.seqz.txt.gz", package = "sequenza")
> data.file
```

The seqz file can be read all at once, but processing one chromosome at a time is less demanding on computational resources, especially while processing whole genome data, and might be preferable in case of limited computational resources.

Read only the data corresponding to chromosome 1:

```
> seqz.data <- read.seqz(data.file, chr.name = "1")
```

Alternatively, read all data at once:

```
> seqz.data <- read.seqz(data.file)
```

```
> str(seqz.data, vec.len = 2)
```

```
'data.frame':        53937 obs. of  14 variables:
 $ chromosome     : chr  "1" "1" ...
 $ position       : int  866168 878255 880150 881992 884173 ...
 $ base.ref       : chr  "C" "T" ...
 $ depth.normal   : int  7 21 9 53 80 ...
 $ depth.tumor    : int  10 10 16 37 61 ...
 $ depth.ratio    : num  1.429 0.476 ...
 $ Af             : num  0.9 0.9 0.9 0.514 0.586 ...
 $ Bf             : num  0 0 0 0.486 0.397 ...
 $ zygosity.normal: chr  "hom" "hom" ...
 $ GC.percent     : num  66 72 54 63 64 ...
 $ good.reads     : num  10 10 10 35 58 ...
 $ AB.normal      : chr  "C" "T" ...
 $ AB.tumor       : chr  "A0.1" "G0.1" ...
 $ tumor.strand   : chr  "A1.0" "G1.0" ...
```

The files can be read even faster; after mapping the chromosomes location in the file, it is possible to select the specific lines of the file to read. See the man page of `read.seqz` for an example.

## 4.2 Quality control

Each aligned base, in the next generation sequencing, is associated with a quality score. The `sequenza-utils.py` software is capable of filtering out bases with a quality score lower then a specified value (default, 20). The number of reads that have passed the filter is returned in the column *good.reads*, while the *depth.tumor* column contains the raw depth indicated in the pileup.

## 4.3 GC-normalization

The GC content bias affects most of the samples; however, some samples are more biased than others. We attempt to remove this bias by normalizing with the mean depth ratio value of a corresponding GC content value.

It is possible to gather GC-content information from the entire file and in the meantime map the chromosome position in the file (to fast access chromosome by chromosome later, see `?read.seqz`):

```
> gc.stats <- gc.sample.stats(data.file)
```

```
> str(gc.stats)
```

```
List of 6
 $ raw         : num [1:70, 1:3] 0.556 0.834 0.894 0.445 0.714 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:70] "15" "16" "17" "18" ...
  .. ..$ : chr [1:3] "25%" "50%" "75%"
 $ adj         : num [1:70, 1:3] 1 0.75 0.921 0.772 0.857 ...
  ..- attr(*, "dimnames")=List of 2
  .. ..$ : chr [1:70] "15" "16" "17" "18" ...
  .. ..$ : chr [1:3] "25%" "50%" "75%"
 $ gc.values   : num [1:70] 15 16 17 18 19 20 21 22 23 24 ...
 $ raw.mean    : Named num [1:70] 0.556 0.926 0.93 0.523 0.96 ...
  ..- attr(*, "names")= chr [1:70] "15" "16" "17" "18" ...
 $ raw.median  : Named num [1:70] 0.556 1.111 0.971 0.576 0.833 ...
  ..- attr(*, "names")= chr [1:70] "15" "16" "17" "18" ...
 $ file.metrics:'data.frame':        23 obs. of  4 variables:
  ..$ chr    : Factor w/ 23 levels "1","10","11",..: 1 12 16 17 18 19 20 21 22 2 ...
  ..$ n.lines: int [1:23] 5817 3377 2957 2179 1976 2756 2765 1837 2488 2673 ...
  ..$ start  : num [1:23] 1 5818 9195 12152 14331 ...
  ..$ end    : num [1:23] 5817 9194 12151 14330 16306 ...
```

Or alternatively, it is possible to collect the GC-contents information from an object already loaded in the environment.

```
> gc.stats <- gc.norm(x = seqz.data$depth.ratio,
+                     gc = seqz.data$GC.percent)
```

In either case the the normalization to the depth.ratio is performed in the following way:

```
> gc.vect  <- setNames(gc.stats$raw.mean, gc.stats$gc.values)
> seqz.data$adjusted.ratio <- seqz.data$depth.ratio /
+                             gc.vect[as.character(seqz.data$GC.percent)]
```

```
> par(mfrow = c(1,2), cex = 1, las = 1, bty = 'l')
> matplot(gc.stats$gc.values, gc.stats$raw,
+         type = 'b', col = 1, pch = c(1, 19, 1), lty = c(2, 1, 2),
+         xlab = 'GC content (%)', ylab = 'Uncorrected depth ratio')
> legend('topright', legend = colnames(gc.stats$raw), pch = c(1, 19, 1))
> hist2(seqz.data$depth.ratio, seqz.data$adjusted.ratio,
+       breaks = prettyLog, key = vkey, panel.first = abline(0, 1, lty = 2),
+       xlab = 'Uncorrected depth ratio', ylab = 'GC-adjusted depth ratio')
```

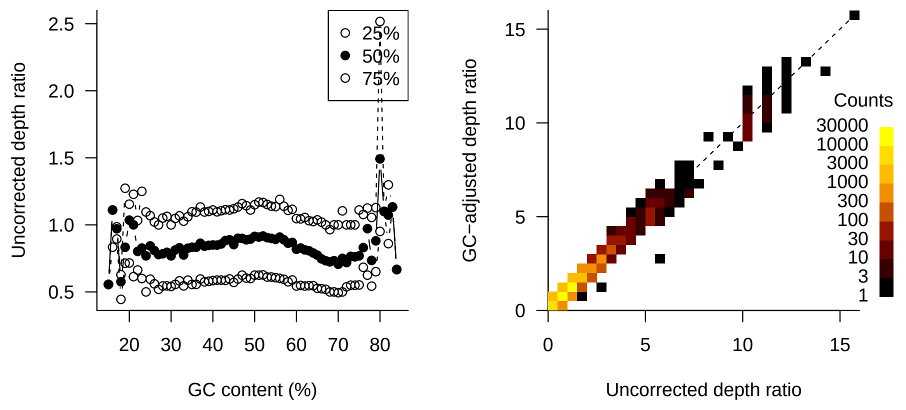

Figure 1: Visualization of depth.ratio bias in relation of GC content (left), and resulting normalization effect (right).

## 5 Analyzing sequencing data with sequenza

The R package `sequenza` offers an ensemble of functions and models that can be used to design customized workflows and analyses. Here we provide generic and most commonly used analysis steps.

- Extract the relevant information from the raw *seqz* file.
- Fit the *sequenza* model to infer cellularity and ploidy.
- Apply the estimated parameters to detect CNV variant alleles

### 5.1 Extract the information from the *seqz* file.

The function `sequenza.extract` is designed to efficiently access the *seqz* file and take care of normalization steps. The arguments enable customization of a set of actions listed below:

- binning depth ratio and B allele frequency in a desired window size (allowing a desired number of overlapping windows);
- performing a fast, allele specific segmentation using the `copynumber` package[3];
- filter mutations by frequency and noise.

```
> test <- sequenza.extract(data.file)
> names(test)
```

After the raw data is processed, the size of the data is considerably reduced. Typically, the R object resulting from `sequenza.extract` can be stored as a file of a few megabytes, even for whole genome sequencing data.

The result of this first step consists of a list of lists. All the sub-lists have a different information subdivided by chromosome. Every list share the same chromosome order.

### 5.1.1 Plot chromosome view with mutations, BAF, depth ratio and segments

Each chromosome can be visualized using the function `chromosome.view` as in Figure 2. The same function can be used to visualize the data after the estimation of *cellularity* and *ploidy* parameters as in Figure 5.

```
> chromosome.view(mut.tab = test$mutations[[1]], baf.windows = test$BAF[[1]],
+                 ratio.windows = test$ratio[[1]], min.N.ratio = 1,
+                 segments = test$segments[[1]], main = test$chromosomes[1])
```

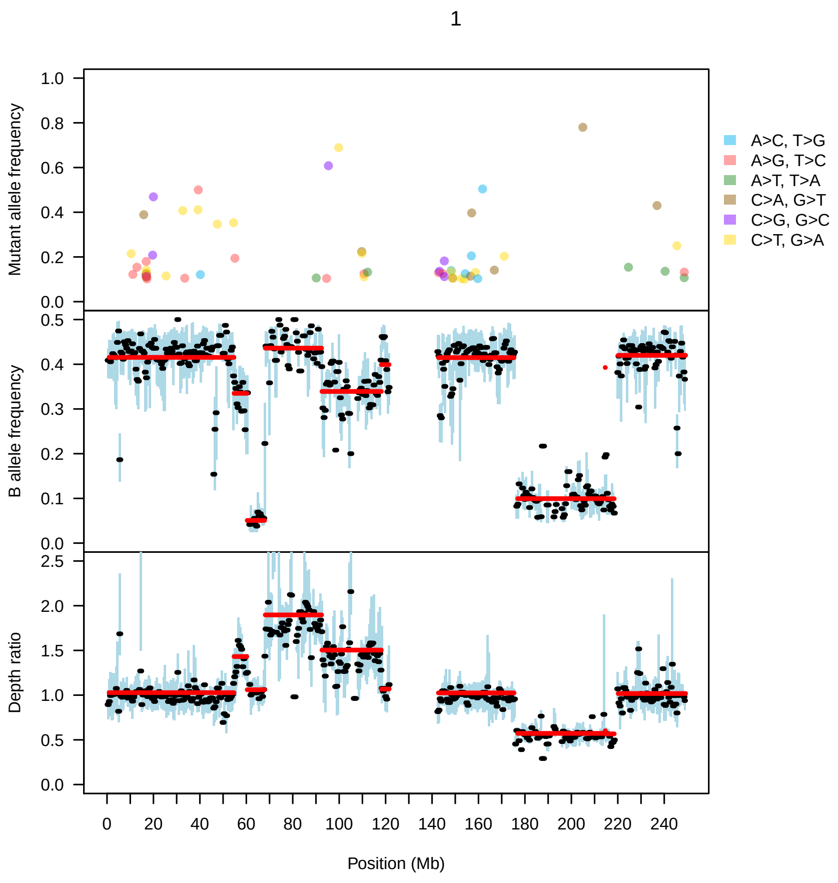

Figure 2: Plots of mutant allele frequency (top), B-allele frequency (middle) and depth ratio (bottom) for chromosome position.

## 5.2 Inference of cellularity and ploidy

After the raw data is conveniently processed, we can apply the Bayesian inference implemented in the package. The function `sequenza.fit` performs the inference using the calculated B allele frequency and depth ratio of the obtained segments. The method can be explored in more detail by reading the manual pages for the function `baf.model.fit`.

```
> CP.example <- sequenza.fit(test)
```

The result is a list in the format *list(x, y, z)*, which is directly usable by standard graphical functions, such as `image`. However we provide functions to explore and better display the results, and to extract the point estimate and confidence intervals.

## 5.3 Results of model fitting

The last part of the workflow is to apply the estimated parameters. There is an all-in-one function that plots and saves the results, giving control on file names and output directory:

```
> sequenza.results(sequenza.extract = test, sequenza.fit = CP.example,
+                  sample.id = "Test", out.dir="TEST")
```

Although this standard way of presenting the result would be appropriate for most situations, it is possible to create an alternative wrapper by using functions in the following sub-sections.

### 5.3.1 Confidence intervals, confidence region and point estimate

The object resulting from `sequenza.fit` has two vectors, x and y, indicating respectively the tested values of ploidy and cellularity, and a matrix z with x columns and y rows, containing the estimated log-likelihood. Confidence intervals for these two parameters can be calculated using the function `get.ci`.

```
> cint <- get.ci(CP.example)
```

It is also possible to plot the likelihood over the combinations of the two parameters, highlighting the point estimate and the confidence region.

```
> cp.plot(CP.example)
> cp.plot.contours(CP.example, add = TRUE, likThresh = c(0.95))
```

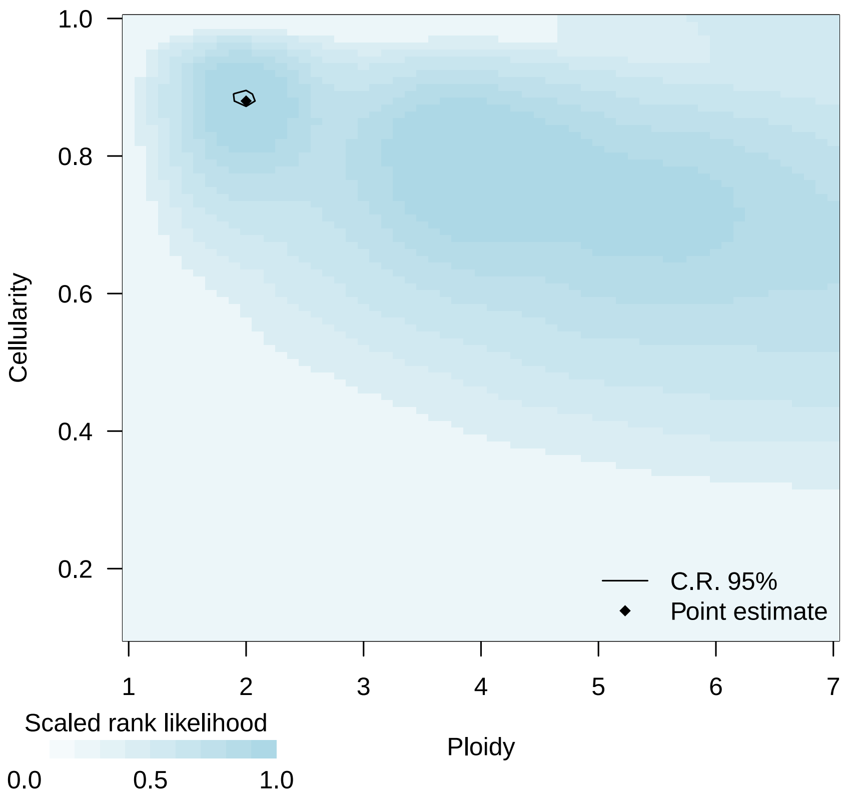

Figure 3: Result from the Bayesian inference over the defined range of cellularity and ploidy. Color intensity indicates the log-likelihood of corresponding cellularity/ploidy values.

By exploring the results for *cellularity* and *ploidy* separately, it is possible to draw the likelihood distribution for each parameter. The information is returned by the `get.ci` function.

```
> par(mfrow = c(2,2))
> cp.plot(CP.example)
> cp.plot.contours(CP.example, add = TRUE)
> plot(cint$values.cellularity, ylab = "Cellularity",
+      xlab = "likelihood", type = "n")
> select <- cint$confint.cellularity[1] <= cint$values.cellularity[,2] &
+           cint$values.cellularity[,2] <= cint$confint.cellularity[2]
> polygon(y = c(cint$confint.cellularity[1], cint$values.cellularity[select, 2], cint$confint.cel
+         x = c(0, cint$values.cellularity[select, 1], 0), col='red', border=NA)
> lines(cint$values.cellularity)
> abline(h = cint$max.cellularity, lty = 2, lwd = 0.5)
> plot(cint$values.ploidy, xlab = "Ploidy",
+      ylab = "likelihood", type = "n")
> select <- cint$confint.ploidy[1] <= cint$values.ploidy[,1] &
+           cint$values.ploidy[,1] <= cint$confint.ploidy[2]
> polygon(x = c(cint$confint.ploidy[1], cint$values.ploidy[select, 1], cint$confint.ploidy[2]),
+         y = c(0, cint$values.ploidy[select, 2], 0), col='red', border=NA)
> lines(cint$values.ploidy)
> abline(v = cint$max.ploidy, lty = 2, lwd = 0.5)
>
```

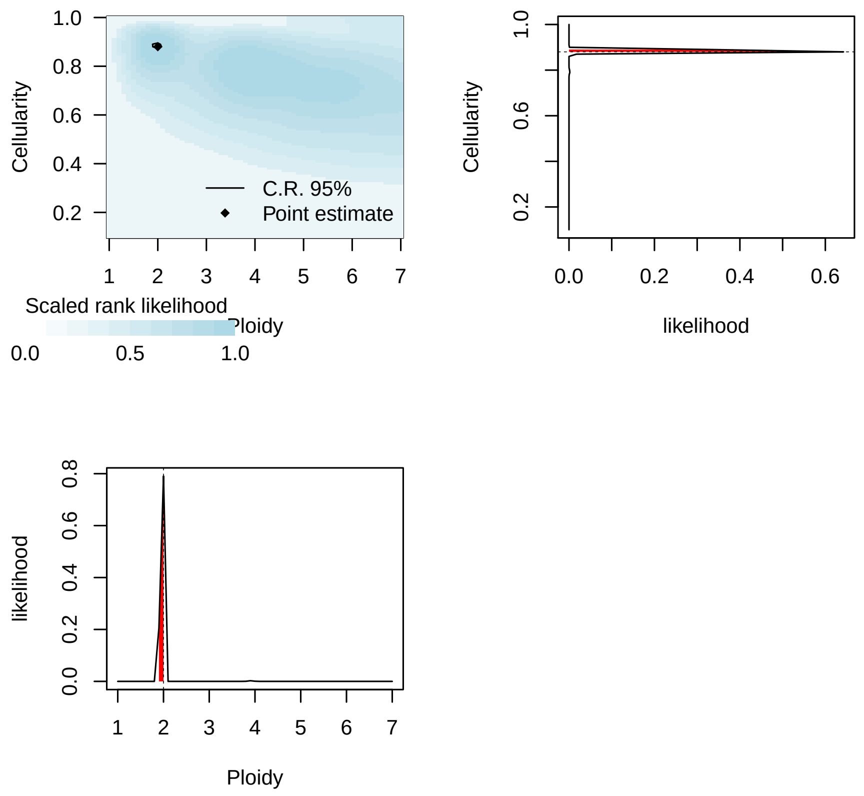

Figure 4: Plot of the log-likelihood with respective cellularity and ploidy probability distribution and confidence intervals.

## 5.4 Call CNVs and mutations using the estimated parameters

The point estimate value corresponds to the point of maximum likelihood, detected after the confidence interval computation:

```
> cellularity <- cint$max.cellularity
> cellularity
```

```
[1] 0.88
```

```
> ploidy <- cint$max.ploidy
> ploidy
```

```
[1] 2
```

In addition we need to calculate the average normalized depth ratio, used to set a value for the baseline copy number.

```
> avg.depth.ratio <- mean(test$gc$adj[, 2])
> avg.depth.ratio
```

```
[1] 1
```

### 5.4.1 Detect variant alleles (mutations)

To detect variant alleles, we use a mutation frequency model that is implemented as the `mufreq.bayes` function:

```
> mut.tab     <- na.exclude(do.call(rbind, test$mutations))
> mut.alleles <- mufreq.bayes(mufreq = mut.tab$F,
+                             depth.ratio = mut.tab$adjusted.ratio,
+                             cellularity = cellularity, ploidy = ploidy,
+                             avg.depth.ratio = avg.depth.ratio)
> head(mut.alleles)
```

```
   CNn CNt Mt          L
4    2   2  1 -19.046897
3    2   2  0 -27.664531
41   2   2  1 -27.631805
42   2   2  1  -9.124629
31   2   2  0 -27.664531
32   2   2  0 -25.073141
```

```
> head(cbind(mut.tab[,c("chromosome","position","F","adjusted.ratio", "mutation")],
+            mut.alleles))
```

```
      chromosome position     F adjusted.ratio mutation CNn
1.364          1 10436585 0.215       1.027689      C>T   2
1.386          1 11140488 0.122       1.027689      A>G   2
1.510          1 12888791 0.154       1.027689      A>G   2
1.652          1 15821826 0.389       1.027689      G>T   2
1.795          1 16890737 0.115       1.027689      C>A   2
1.797          1 16890771 0.109       1.027689      G>A   2
      CNt Mt          L
1.364   2  1 -19.046897
1.386   2  0 -27.664531
1.510   2  1 -27.631805
1.652   2  1  -9.124629
1.795   2  0 -27.664531
1.797   2  0 -25.073141
```

The result consists of four values for every imputed mutation: *CNn* is the provided copy number of the normal sample at the given position (default = 2); *CNt* is the estimated copy number of the tumor at the given position; *Mt* is the estimated numbers of alleles carrying the mutation; *L* is the log-likelihood of the model fit.

### 5.4.2 Detect copy number variations

To detect copy number variations we use a B allele frequency model, implemented in the function `baf.bayes`, with the estimated parameters of *cellularity* and *ploidy*:

```
> seg.tab     <- na.exclude(do.call(rbind, test$segments))
> cn.alleles <- baf.bayes(Bf = seg.tab$Bf, depth.ratio = seg.tab$depth.ratio,
+                         cellularity = cellularity, ploidy = ploidy,
+                         avg.depth.ratio = avg.depth.ratio)
> head(cn.alleles)

     CNt A B         L
[1,]   2 1 1 -8.111186
[2,]   3 2 1 -8.046104
[3,]   2 2 0 -7.354803
[4,]   4 2 2 -8.009951
[5,]   3 2 1 -8.015173
[6,]   2 1 1 -8.369453

> seg.tab <- cbind(seg.tab, cn.alleles)
> head(seg.tab)

    chromosome start.pos   end.pos         Bf N.BAF
1.1          1    881992  54694219 0.41541244  1636
1.2          1  54700724  60223464 0.33497406    73
1.3          1  60381518  67890614 0.05069544    94
1.4          1  68151686  92445264 0.43610321   265
1.5          1  92568300 118165373 0.33914750   314
1.6          1 118165645 121485317 0.39920977    62
    depth.ratio N.ratio CNt A B         L
1.1    1.027689    2172   2 1 1 -8.111186
1.2    1.431476     101   3 2 1 -8.046104
1.3    1.059963     114   2 2 0 -7.354803
1.4    1.896806     340   4 2 2 -8.009951
1.5    1.503767     431   3 2 1 -8.015173
1.6    1.067938      85   2 1 1 -8.369453
```

The result consists of four values for every imputed segment: *CNt* is the estimated copy number of the tumor of the given segment; *A* is the estimated number of A alleles; *B* is the estimated number of B alleles; *L* is the log-likelihood of the model fit.

## 5.5 Visualize detected copy number changes and variant alleles

To visualize the data after detection of CNV and variant alleles, it is possible to use the `chromosome.view`. In order to draw the relative model points (and to evaluate how the estimated model fits the real data) more information is needed compared to Figure 2:

- Each segment must have the columns relative to the copy number variation calling.
- *Cellularity* and *ploidy* estimates.
- Average normalized depth ratio.

```
> chromosome.view(mut.tab = test$mutations[[3]], baf.windows = test$BAF[[3]],
+                  ratio.windows = test$ratio[[3]],  min.N.ratio = 1,
+                  segments = seg.tab[seg.tab$chromosome == test$chromosomes[3],],
+                  main = test$chromosomes[3],
+                  cellularity = cellularity, ploidy = ploidy,
+                  avg.depth.ratio = avg.depth.ratio)
```

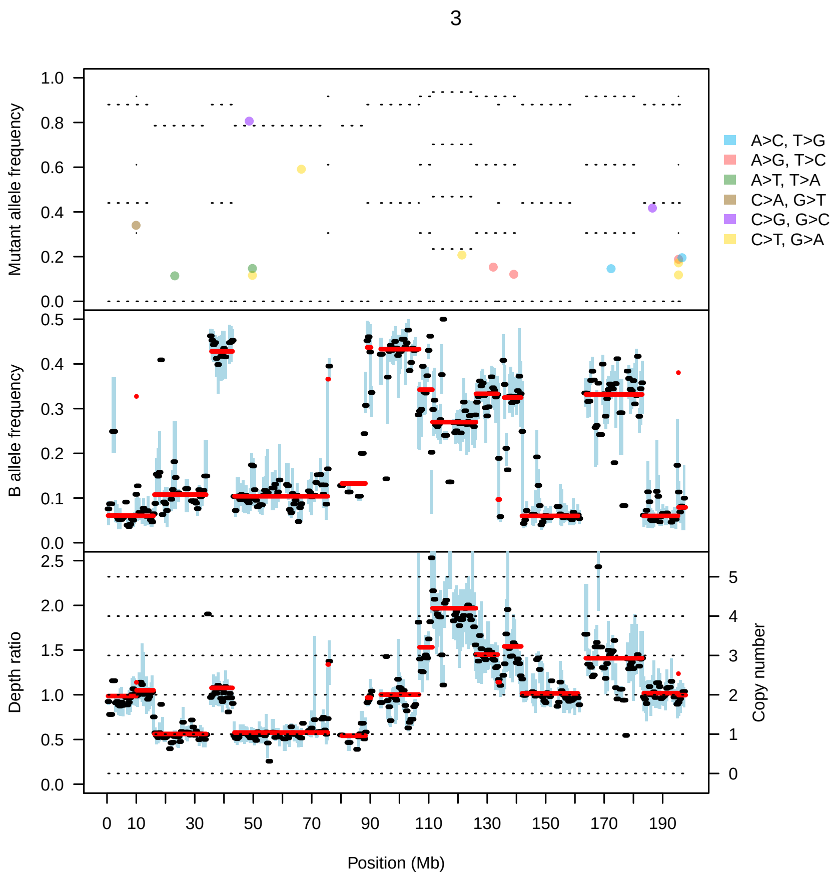

Figure 5: Plots of mutant allele frequency (top), B-allele frequency (middle) and depth ratio (bottom) for chromosome position. Horizontal dotted lines indicate expectation values for various copy number/allele states.

### 5.5.1 Genome-wide view of the allele and copy number state

```
> genome.view(seg.cn = seg.tab, info.type = "CNt")
> legend("bottomright", bty="n", c("Tumor copy number"),col = c("red"),
+        inset = c(0, -0.4), pch=15, xpd = TRUE)
```

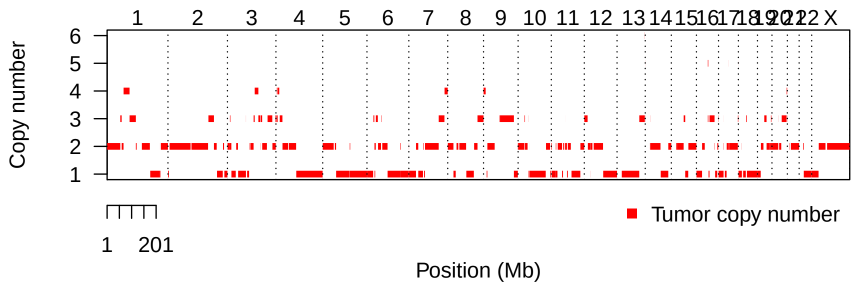

Figure 6: Genome-wide absolute copy number profile obtained from exome sequencing.

```
> genome.view(seg.cn = seg.tab, info.type = "AB")
> legend("bottomright", bty = "n", c("A-allele","B-allele"), col= c("red", "blue"),
+        inset = c(0, -0.45), pch = 15, xpd = TRUE)
```

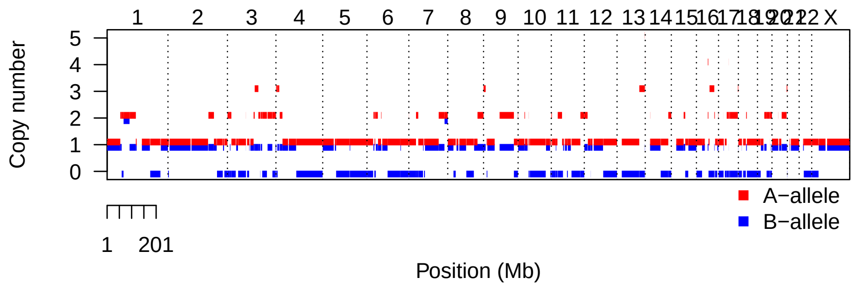

Figure 7: Genome-wide allele-specific copy number profile obtained from exome sequencing.

## References

[1] Daniel C Koboldt, Qunyuan Zhang, David E Larson, Dong Shen, Michael D McLellan, Ling Lin, Christopher A Miller, Elaine R Mardis, Li Ding, and Richard K Wilson. VarScan 2: somatic mutation and copy number alteration discovery in cancer by exome sequencing. *Genome Research*, 22(3):568–76, March 2012.

[2] Heng Li, Bob Handsaker, Alec Wysoker, Tim Fennell, Jue Ruan, Nils Homer, Gabor Marth, Goncalo Abecasis, and Richard Durbin. The Sequence Alignment/Map format and SAMtools. *Bioinformatics (Oxford, England)*, 25(16):2078–9, August 2009.

[3] Gro Nilsen, Knut Liestø l, Peter Van Loo, Hans Kristian Moen Vollan, Marianne B Eide, Oscar M Rueda, Suet-Feung Chin, Roslin Russell, Lars O Baumbusch, Carlos Caldas, Anne-

Lise Bø rresen Dale, and Ole Christian Lingjaerde. Copynumber: Efficient algorithms for single- and multi-track copy number segmentation. *BMC Genomics*, 13:591, January 2012.