User's Manual for inveRsion package

Alejandro Cáceres, Suzanne Sindi, Ben Raphael,
Mario Cáceres and Juan Ramon González
acaceres@creal.cat

February 24, 2011

Contents

Abstract

This document illustrates the use of `inveRsion` package. The manual shows how to analyze SNP-array data to detect inversion events and to assign inversion genotypes to each subject in the sample. `inveRsion` software handles both haplotype (phased SNP-array) and genotype data.

1 Introduction

Genetic inversions are a set of structural variants, which like CNVs and mosaics, can be detected with SNP-array data. While current studies collect phenotypic information to assess its association to nucleotide variation, the

ability to detect inversions form already available information allows immediate assessment of the impact of inversion polymorphisms to phenotypic expression.

In this document, we present the full functionality of the `inveRsion` package. The software detects inversions in haplotype (phased) and genotype data. It features an optimized implementation of the inversion model presented in [1] and a novel generalization for genotypes. A quick guide to the tool can be found in its vignette, where only haplotype data of a small set of SNPs is analyzed. Here, we show the analysis of a more comprehensive data set of haplotypes and genotyoes, which can take up to two hours. As a guiding example, we use data simulated with the `invertFregene` software [2]. While we encourage you to produce your own data-set following `invertFregene` instructions, the sample data used here can be downloaded from `www.creal.cat/jrgonzalez/software.htm`. We refer the reader to the article [3], where the methodology is explained in detail. The general work-flow is shown in figure 1, with the functions involved in each step at the bottom of each box. The package accepts text files with the haplotype information (1:variant allele, 0:non variant allele) for each chromosome (row) and SNP (column) labelled by their spatial coordinates (first row). For analysis of genotype data, the subject genotypes should be given at each row encoded as 0:homozygous, 1:heterozygous and 2:variant heterozygous. In the final section, we show how to convert phased data from HapMap samples to the required format and how to treat genotypes encoded in a `SNPmatrix` object.

2 Installation

`inveRsion` is written on R (`www.r-cran.org`). The package can be downloaded from `www.bioconductor.org`. Alternatively, enter in the R prompt the commands

```
source(" http://bioconductor.org/biocLite.R ")
biocList(inveRsion)
```

The library is loaded with the command

```
> library(inveRsion)

Hola!
welcome to inevRsion package.


type: manual() for full manual
      vignette("inveRsion") for a quick start
```

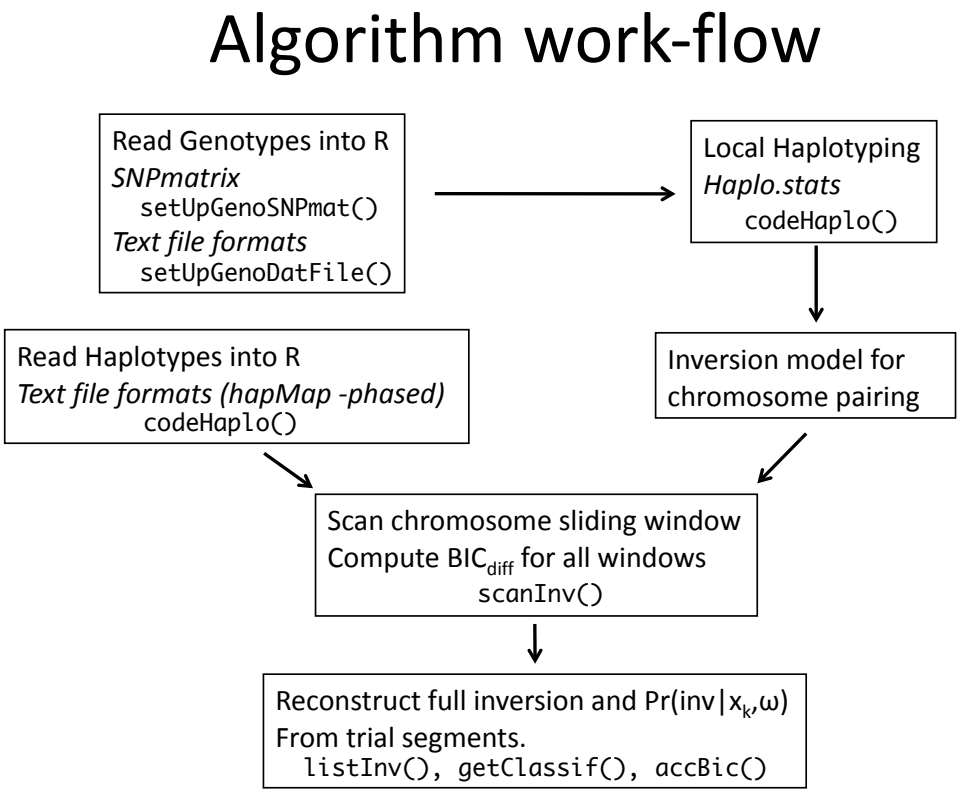


Figure 1: work-flow of `inveRsion` package

3 invertFregene

We use `invertFregene` to simulate inversion events, which allow us to assess the performance of `inveRsion`. The software can be accessed in `www.ebi.ac.uk/projects/BARGEN`.

The sample data we use was obtained by running the commands in the quickstart guide. We run the simulation of population equilibrium and an inversion in such population without any changes. We thus simulated an inversion between 0.75Mb and 1.25Mb with population frequency of 0.4 (final value: 0.4135). We finally sampled the haplotypes and genotypes for N=1000 individuals (set `-control 1000` in SAMPLE). This produces the files `haplotypes_0.dat`, `genotype_0.dat` and `InversionSummary.txt` where you can find the haplotype and genotype data and the indexing of the subjects with the inversion. Note that the chromosomes are zero indexed, e.g. they start with 0 for the first chromosome, 1 for the second and so on.

To start the demo, move those files into your working directory after running the simulations or downloading them from

www.creal.cat/jrgonzalez/software.htm. Extract the numbering from InversionSummary.txt, and correct its zero indexing, by adding one to all labels, then write it into a text file, e.g. mem.txt. The last step is required since chromosomes in inveRsion are naturally indexed from 1.

4 Haplotype data

The file haplotypes_0.dat contains on the first row the coordinates of the SNPs, and on subsequent rows the presence (1) or absence (0) of the variant allele for each chromosome in the population. Chromosomes in $2 * i$ and $2 * i - 1$ (i=1...N) rows correspond to the i-th individual.

Data in such format is simultaneously loaded on the R session and coded in an object of class haploCode

```
> hap <- "haplotypes_0.dat"
> hapCode <- codeHaplo(blockSize = 5, file = "haplotypes_0.dat",
+     minAllele = 0.3, saveRes = FALSE)
```

.....

Each candidate break-point, which is a pair of contiguous SNPs, is flaked by haplotype blocks of n-SNPs. The blocks, two for each break-point, are encoded for the efficient estimation of the inversion model. In this example, haplotype blocks are built such that each candidate break-point is flanked by a set of blockSize (=5) SNPs. The coding of the blocks is then the labelling of haplotypes of size 2*blockSize, present in the population. In this implementation, the binary strings corresponding to the haplotypes are taken as binary numbers with 2*blockSize digits and transformed into decimal integers. Data is read from file path file (=hap, i.e ="haplotypes_0.dat") and filtered such that the brake-points have at least a Minor allele frequency of minAllele (=0.3). The inversion model can be fitted to any segment delimited by two break-points (left and right). However we use only trial segments of fixed window size to scan a whole chromosome for the presence of an inverted sequence. Trial segments that significantly fit the inversion model then cover the inverted sequence. You can scan your data for inversion events with trial segments of fixed length (window) with

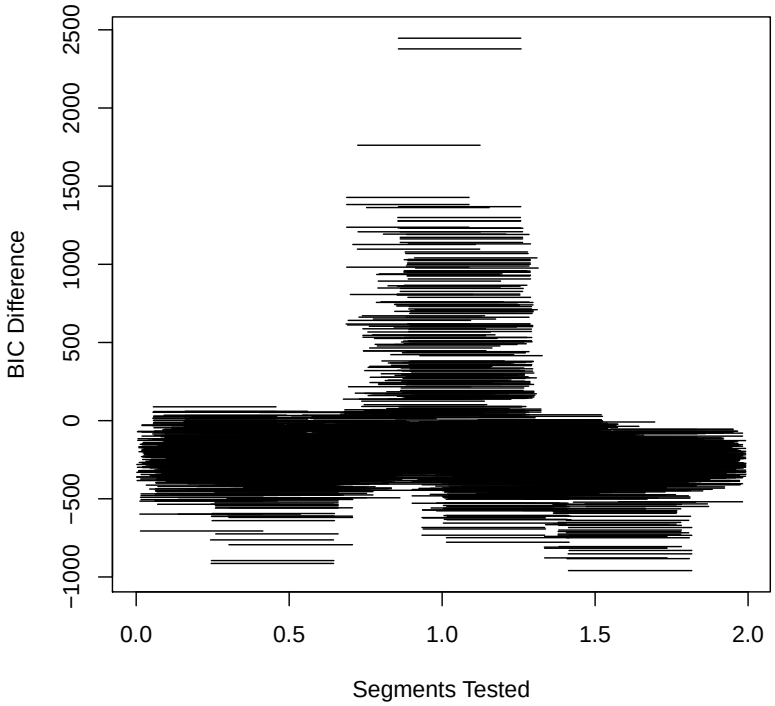
```
> window<-0.4
> scanRes<-scanInv(hapCode,window=window,saveRes=FALSE)
```

.....

```
> scanRes

-Showing object of class: scan-

Top 10 brake-points with highest Likelihood ratio:

             LeftBP         RightBP LogLike  Prob  BicDiff
1  0.85706-0.85741 1.25707-1.25714 2712.43 0.586 2446.397
2  0.85741-0.85769 1.25803-1.25821 2712.43 0.586 2377.989
3  0.72393-0.72405 1.12425-1.12509 2012.49 0.545 1761.661
4  0.75348-0.75369 1.15361-1.15370 1750.67 0.600 1363.024
5  0.68811-0.68834 1.08837-1.08841 1732.01 0.577 1382.366
6  0.68834-0.68982 1.08837-1.08841 1731.98 0.577 1427.940
7  0.85669-0.85706 1.25680-1.25703 1650.06 0.564 1368.828
8  0.68811-0.68811 1.08837-1.08841 1632.50 0.597 1237.251
9  0.85617-0.85639 1.25680-1.25703 1620.35 0.587 1278.307
10 0.85639-0.85657 1.25680-1.25703 1620.35 0.586 1232.701

others:
window: length window ( 0.4 ) for searching inversion segments
```

The algorithm applies an inversion model to each trial segment and assesses the likelihood that it constitutes an inverted segment. The process returns an object of class `scan` that displays the segments with their significance measures (log-likelihood ratio and Bayes information criterion -BIC). The `scan` object can be readily plotted to show the distribution of BIC values for each of the trial segments.

```
> plot(scanRes)
```



At this stage, the identified segments might not estimate the whole inversion, but they roughly cover the left or right sections of the true inverted segment. Numerical values for selected trial segments can be retrieved into the R session with

```
> a <- getInv(scanRes, thBic = 0)
> head(a)
```

	LeftBP	RightBP	LogLike	Prob	Ent	BIC	NumHap
1	0.857	1.258	2712.429	0.586	2.377	2377.989	44
2	0.857	1.257	2712.429	0.586	2.468	2446.397	35
3	0.724	1.124	2012.491	0.545	2.698	1761.661	33
4	0.753	1.154	1750.670	0.600	3.189	1363.024	51
5	0.688	1.088	1732.008	0.577	2.671	1382.366	46
6	0.688	1.088	1731.977	0.577	2.675	1427.940	40

where selection is given by a BIC threshold `thBic`

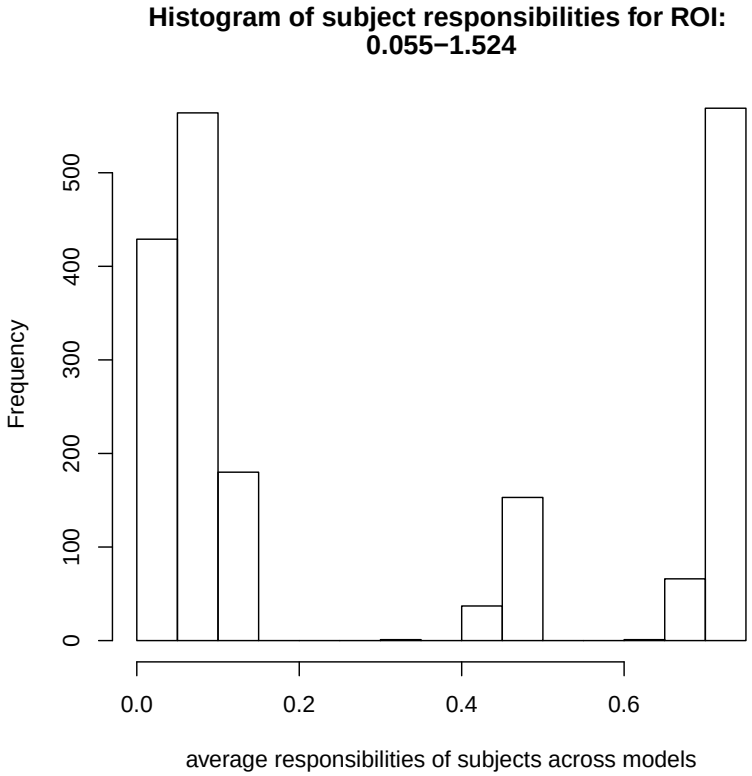
The overlapping segments define a region of interest where the inverted sequence is present;

```
> ROI <- getROIs(scanRes, thBic = 0)
> ROI

  LeftBinf LeftBsup RightBinf RightBsup
1 0.055405 1.124254  0.455881  1.524297
```

`ROI` gives a list of possible regions of interest and the limits of the left and right brake points, from the collection of all the probes with $BIC > thBic$. In this example, it identifies only one region.

The full characterization of the inversion sequence within the ROIs is given by

```
> invList<-listInv(scanRes,hapCode=hapCode,geno=FALSE,all=FALSE,ROI=ROI)

.....................................................................................

> invList

-Showing object of class: inversionList-

    LBPmin   LBPmax   RBPmin   RBPmax   MaxBic invFreq ModelNum
1 0.055405 1.124254 0.455881 1.524297 2705.998   0.318      361

> plot(invList, wROI = 1)
```



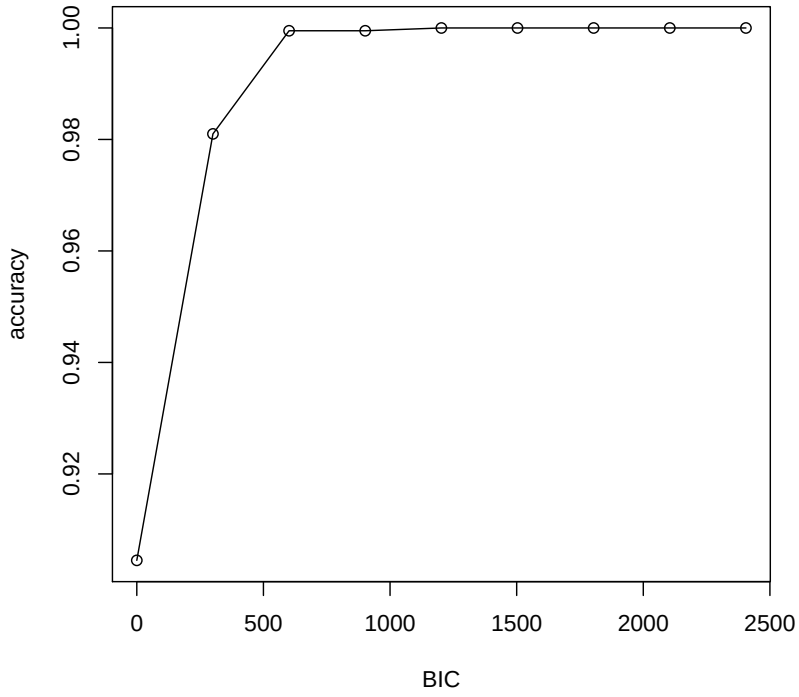
where a set of ROIs can be explicitly given or a BIC-threshold (`thBic`) used as argument (default value is zero). For each ROI an interval for the left and right brake points is computed according to the overlapping of the segment probes, if you want to obtain the previous scan results then set `all =FALSE`. A more intense computation of the inversion model for *all* possible segments within the intervals is launched with `all =TRUE`. This can be an interesting option if the initial scan has not produced enough probes with high BIC. The plot of `invList` above gives the histogram of classification probabilities within the population, for the first ROI (`wROI=1`). This is obtained as a majority vote of all the classifications obtained for each trial segment.

If information is available on which chromosomes have the inversion, you can compute the accuracy of the classification as a function of the BIC threshold. In the case of haplotype data the accuracy is the proportion of correctly classified chromosomes. We have implemented a function to test this relationship

```
> mem <- "mem.txt"
> ac <- accBic(invList, classFile = mem, nsub = 1000, npoints = 10)
```

```
.........
```

```
> plot(ac)
```



where *mem.txt* file stores the chromosome numbers for those classified as inverted. In this example, we can then re-compute `listInv` with `thBic=1500` to get a tighter estimate for the brake point intervals

```
> invList<-listInv(scanRes,hapCode=hapCode,geno=FALSE,all=FALSE,thBic=1500,
+ saveRes=FALSE,saveROI=FALSE)
```

```
.....................................................................................
```

```
> invList
```

```
-Showing object of class: inversionList-

    LBPmin   LBPmax   RBPmin   RBPmax   MaxBic invFreq ModelNum
1 0.723928 0.857414 1.124254 1.258031 2705.998  0.4135       70
```

Finally, note that the models of all trial segments classify the chromosomes into inverted or not inverted populations. The final inversion frequency in the population is given by the average of all chromosome classifications, and can be recovered into the R session;

```
> r <- getClassif(invList, geno = FALSE)
> r[1:10]

 [1] 0 1 0 1 1 0 1 0 1 1
```

A soft classification is obtained with `bin=FALSE` while the classification of the i-th region of interest is retrieved with `wROI=` i.

5 Genotype data

In the same haplotype sampling, `invertFregene` produces the file `genotypes_0.dat` with the genotype data per individual. Analysis of such data in `inveRsion` follows a similar external pathway, with an extra initial set up of the data to accommodate a range of different data formats. In this example, genotype data is provided from a file whose first row contains the SNP coordinates and the subsequent rows encode the genotype information 0,1 or 2. Other genotype formats currently available are those handled by `snp.matrix` (PLINK).

The initial step is to set up the data

```
> gen <- "genotypes_0.dat"
> gDat <-setUpGenoDatFile(file=gen,sortMinor=TRUE,saveRes=FALSE)

-Set up object of class: GenoDat-
```

`sortMinor` calls a procedure to assign 0 to the non-variant homozygote, 1 to the heterozygote and 2 to the variant homozygote. This is an object of class `genoDat` that can be displayed and plotted (`plot(A)`) but more importantly it can be passed directly to `codeHaplo`

```
> hapCode <- codeHaplo(gDat, blockSize = 5, minAllele = 0.3, saveRes = FALSE)

.....................................................................................
```

`codeHaplo` recognizes genotype data and calls `haplo.stats` functions to produce the most probable haplotypes for each subject to flank the candidate brake-points. Haplotypes are of size 2* `blockSize`, made of the concatenation of two flanking blocks, one each side. The result is a `haploCode` object,

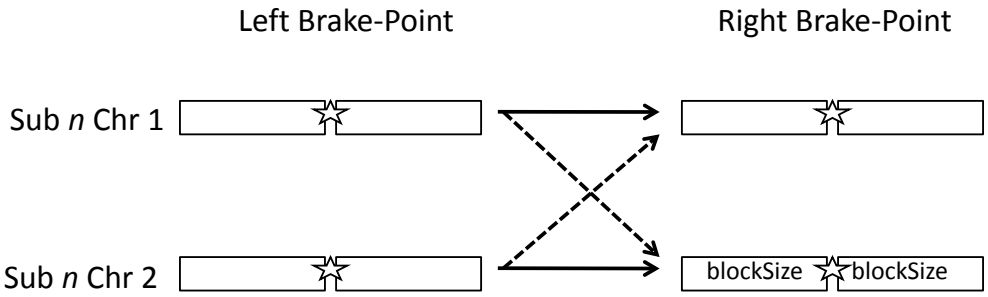


Figure 2: Local haplotyping assigns the most probable haplotypes, of flanking blocks, on each chromosome of subject n. At long distances the coupling between the left brake-point and the right brake-point haplotypes is unsure. `scanInv` computes the most probable chromosome for each of the haplotypes containing the right brake-point, given the haplotypes containing the left brake-point.

where chromosomes in $2*i$ and $2*i-1$ (i=1...N) rows correspond to the i-th individual. As local haplotyping is called, the process increases considerably on computation time.

Local haplotyping looses the information on how the haplotypes containing the left and right brake-points are coupled within the same subject, as illustrated in figure. Thus, while scanning for inversions, `scanInv` computes the most probable chromosome for each of the haplotypes in the right brake-point (option `geno=TRUE`).

```
> window<-0.4
> scanRes<-scanInv(hapCode,window=window,geno=TRUE,saveRes=FALSE)
```

. . . . . .

```
> a<-getInv(scanRes)
> head(a)
```

```
    LeftBP   RightBP   LogLike   Prob   Ent      BIC   NumHap
1    0.843     1.243  2916.206  0.482 2.347 2642.574       36
```

```
2    0.843     1.243  2807.575  0.479 2.600 2564.346       32
3    0.758     1.158  2793.635  0.553 2.283 2558.007       31
4    0.758     1.158  2771.678  0.557 2.286 2536.050       31
5    0.716     1.117  2769.918  0.561 2.102 2557.093       28
6    0.757     1.158  2760.978  0.559 2.367 2540.552       29
```

You can obtain all the information within the ROI defined by all the overlapping segments with $BIC > 0$

```
> invList<-listInv(scanRes,hapCode=hapCode,geno=TRUE,
+ all=FALSE,saveRes=FALSE, thBic=0,saveROI=FALSE)

.....................................................................

> invList

-Showing object of class: inversionList-

    LBPmin   LBPmax   RBPmin   RBPmax   MaxBic invFreq ModelNum
1 0.055481 0.975834 0.455881 1.376062 2642.574  0.2925      383
```

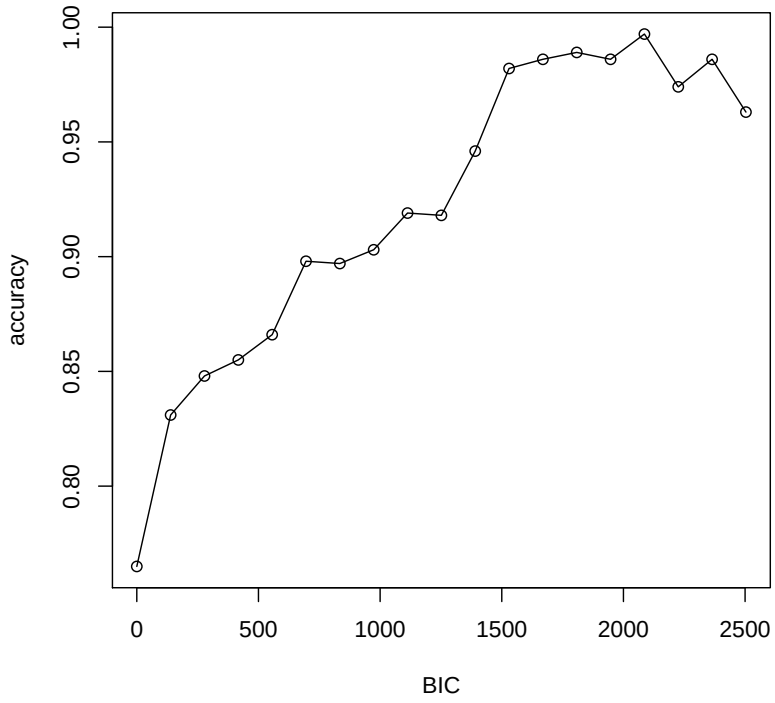
and extract the classification of subjects with `getClasiff`.

```
> r <- getClassif(invList, geno = TRUE)
> r[1:10, ]

      id       hom       het     homInv
1   sub1 0.3716298 0.4969493 0.13142090
2   sub2 0.3844869 0.4712351 0.14427803
3   sub3 0.5657139 0.3855436 0.04874258
4   sub4 0.5051504 0.4126758 0.08217385
5   sub5 0.1922366 0.4928113 0.31495204
6   sub6 0.4891982 0.4654678 0.04533401
7   sub7 0.1014596 0.4341566 0.46438383
8   sub8 0.3491196 0.4845285 0.16635194
9   sub9 0.3451452 0.4950882 0.15976658
10 sub10 0.3620312 0.5631438 0.07482497
```

For this example we have the correct classification in `mem="mem.txt"`, as above, and can determine the optimal BIC threshold for `listInv`.

```
> ac <- accBic(invList, classFile = mem, nsub = 1000, npoints = 20,
+     geno = TRUE)
```

```
.....................
> plot(ac)
```



Note the option `geno=TRUE` computes the accuracy for inversion genotypes, meaning whether each subject is correctly classified as common inverted homozygote, inverted heterozygote or variant inverted heterozygote. A better estimation of the inverted sequence is then obtained form choosing an optimal BIC threshold from the previous result. Recomputing the estimate for the inversion ($thBic = 2300$) gives tighter break-points intervals

```
> invList<-listInv(scanRes,hapCode=hapCode,geno=TRUE,all=FALSE,
+ saveRes=FALSE,thBic=2300,saveROI=FALSE)
```

```
.....................................................................................
> invList
-Showing object of class: inversionList-

    LBPmin   LBPmax   RBPmin   RBPmax   MaxBic invFreq ModelNum
1 0.707386 0.866699 1.107802 1.266723 2642.574   0.401       56
```

6 Data format

The format of the haplotype and genotype files illustrated here is flexible enough to accommodate the most common data sets. The following pieces of code are not to be run as a demo but are guidelines of how to treat your own data.

Suitable haplotype files can be easily constructed for the HapMap database, with a minimum effort. As a quick example, consider you have downloaded the files

```
genotypes_chr16_CEU_r21_nr_fwd_phased_all
```

and

```
genotypes_chr16_CEU_r21_nr_fwd_legend_all
```

from `www.hapmap.org`. The haplotype file can be constructed with the following set of instructions within a script

```
##begin script##
haploHapMap<-
    read.table(file="genotypes_chr16_CEU_r21_nr_fwd_phased_all",
    header=FALSE)

leg<-
   read.table(file="genotypes_chr16_CEU_r21_nr_fwd_legend_all",
   header=FALSE)

lociPos<-leg[-1,2]

haploHapMap<-rbind(lociPos,haploHapMap)

write.table(haploHapMap, file="haploHapMap.txt",
   col.names=FALSE,row.names=FALSE)
##end script##
```

This phased data can be used to create the genotype data, only for illustration purposes. Remember that genotype data is used in case of not having the full phased data. A reconstruction of the genotypes can be achieved in a handful of steps.

```
##begin script##
pair<-2*(1:(NROW(haploHapMap)/2))
odd<-pair-1

genoDatHapMap<-haploHapMap[pair,]+haploHapMap[odd,]

genoDatHapMap<-rbind(lociPos,genoHapMap)

write.table(genoHapMap, file="genoHapMap.txt",
   col.names=FALSE, row.names=FALSE)
##end script##
```

In case of having your own genotype data see

`http://pngu.mgh.harvard.edu/~purcell/plink/`

on how to use PLINK formats and `SNPmatrix` on how to load your data on an R session. Let us assume you have named your genotype data as a variable `geno` on R, which is an SNPmatrix object, and have saved it on an `geno.Rdata` file. You can then set up the data for chromosome 16 with the help of your annotation file `geno.bim` as following

```
##begin script##
load(geno.RData)

annot<-read.delim("geno.bim",header=FALSE)

chr<-16

A<-setUpGenoDatSNPmat(chr,geno,annot,saveRes=FALSE,saveGeno=FALSE)
##end script##
```

If you select the option `saveGeno=TRUE` a .txt file will be saved in the format discussed in section 4. In case of using `SNPmatrix` objects we recommend you saving them in a `.RData` file, then call the `inveRsion` library on a fresh R session and load them back on R.

The library is unloaded with the command

```
> detach("package:inveRsion")
```

References

[1] Sindi SS, and Raphael BJ, *Identification and frequency estimation of inversion polymorphisms from haplotype data*, J Comput Biol. (2010) 17 (3): 517-31.

[2] O'Reilly PF, Coin LJM and Hoggart CJ, *invertFREGENE: software for simulating inversions in population genetic data*, Bioinformatics (2010) 26 (6): 838-840

[3] Caceres A, Sindi S, Raphael B, Caceres M and Gonzalez JR *Identification of polimorphic inversions from genotypes*, Bioinformatics (2011) submitted.