PSSeg: Parent-Specifc copy number segmentation

M. Pierre-Jean, P. Neuvial

April 3, 2013

Abstract

This vignette describes how to use the `PSSeg` function from the `jointSeg` package to partition bivariate DNA copy number signals into segments of constant parent-specific copy number. The first proposed method [4] starts by identifying a list of candidate change points through a fast (greedy) recursive binary segmentation (RBS). This list is then pruned using dynamic programming [1]. The second proposed method is the implementation in `R` of GFLARS initially programming in `Matlab` by [5]. The resulting approach is able to quickly and accurately identify true changes in DNA copy numbers.

keywords: segmentation, change point model, binary segmentation, dynamic programming, DNA copy number, parent-specific copy number.

Contents

1 Preparing data to be segmented

`PSSeg` requires normalized copy number signals, in the form of total copy number estimates and allele B fractions for tumor, the (germline) genotype of SNP. Loci are assumed to come from a single chromosome and to be ordered by genomic position.

For illustration, we show of such a data set may be created from real data. We use data from a public microarray data set, which is distributed in the `acnr` package (from which the `jointSeg` package depends).

```
> library(jointSeg)
> ## load known real copy number regions
> data <- loadCnRegionData(platform="Affymetrix", tumorFraction=1)
> str(data)


'data.frame':       192667 obs. of  4 variables:
 $ c       : num  0.909 0.859 1.304 0.647 0.947 ...
 $ b       : num  NaN NaN NaN NaN NaN NaN NaN -0.035 NaN NaN ...
 $ genotype: num  NA NA NA NA NA NA NA 0 NA NA ...
 $ region  : chr  "(0,1)" "(0,1)" "(0,1)" "(0,1)" ...
```
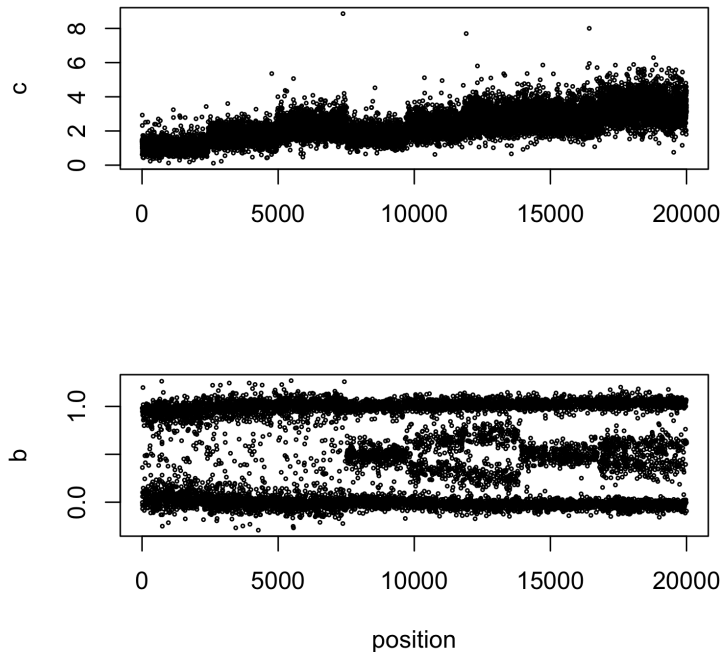
This data set consists of copy number signals from 8 types of genomic regions:

These regions are coded as (C_1, C_2), where C_1 denotes the minor copy number and C_2 denotes the major copy number, i.e. the smallest and the largest of the two parental copy numbers (see e.g. [3] for more detailed definitions). For example, $(1, 1)$ corresponds to a normal state, $(0, 1)$ to an hemizygous deletion, $(1, 2)$ to a single copy gain and $(0, 2$ to a copy-neutral LOH (loss of heterowygosity).

```
> idxs <- sort(sample(1:nrow(data), 2e4))
> plotSeg(data[idxs, ])
```


These real data can then be used to create a realistic DNA copy number profile of user-defined length, and harboring a user-defined number of breakpoints. This is done using the `getCopyNumberDataByResampling` function. Breakpoint positions are drawn uniformly) among all possible loci. Between two breakpoints, the copy number state corresponds to one of the types of regions in
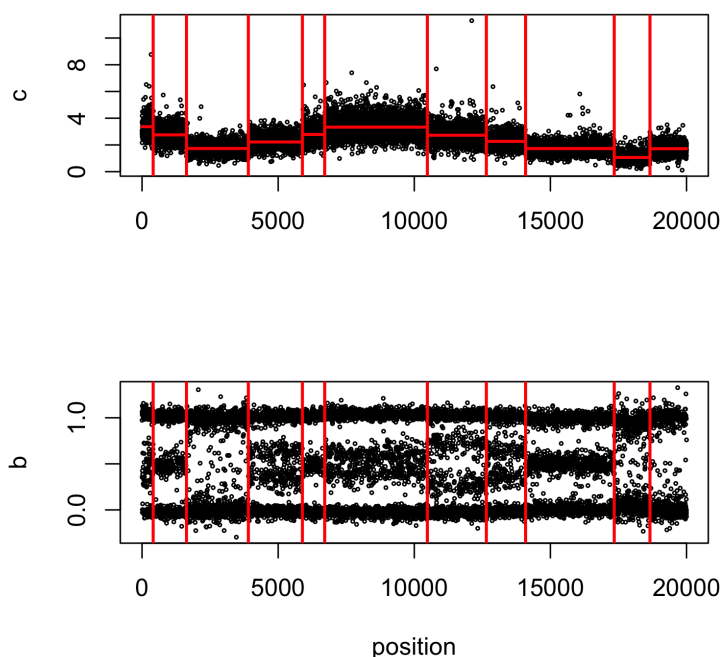
2

data, and each data point is drawn with replacement from the corresponding true copy number signal from the region. More options are available from the documentation of getCopyNumberDataByResampling.

```
> K <- 10
> bkp <- c(408,1632,3905, 5890,6709, 10481, 12647,14089,17345,18657)
> len <- 2e4
> sim <- getCopyNumberDataByResampling(len, bkp=bkp, minLength=500, regData=data)
> datS <- sim$profile
> str(datS)


'data.frame':        20000 obs. of  4 variables:
 $ c       : num  3.01 2.83 3.68 3.12 2.16 ...
 $ b       : num  0.973 1.064 0.355 NaN NaN ...
 $ genotype: num  1 1 0.5 NA NA 1 NA NA 1 NA ...
 $ region  : chr  "(2,3)" "(2,3)" "(2,3)" "(2,3)" ...
```

The resulting copy-number profile is plotted below.

```
> plotSeg(datS, sim$bkp)
```


2   PSSeg segmentation using RBS

We can now use the PSSeg function to segment signals. The method consists in three steps:

1. run a fast (yet approximate) segmentation on these signals in order to obtain a set of (at most hundreds of) candidate change points. This is done using Recursive Binary Segmentation (RBS) [4];

2. prune the obtained set of change points usign dynamic programming [1]

3. select the best number of change points using a model selection criterion proposed by [2]

2.1 Initial segmentation and pruning

```
> resRBS <- PSSeg(data=datS, K=2*K, flavor="RBS", profile=TRUE)
```
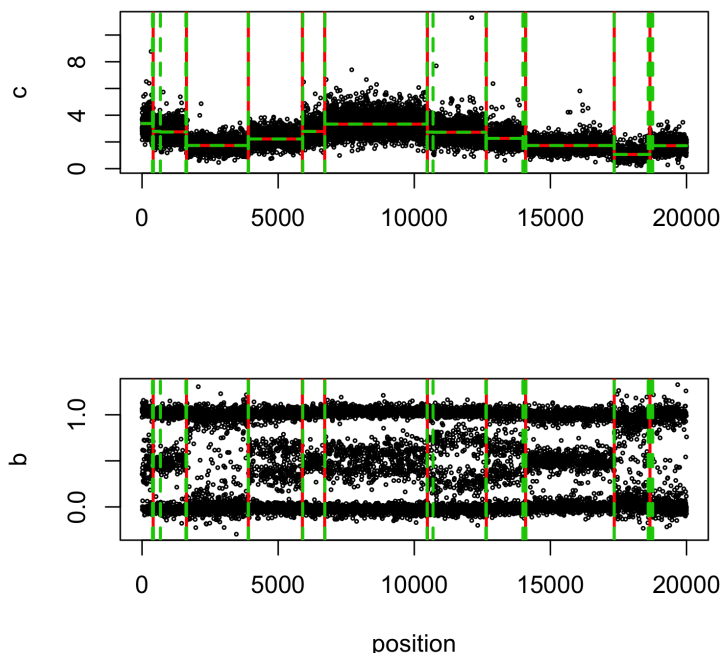
Note that this is fast:

```
> resRBS$prof[, "time"]
```

```
segmentation          dpseg
       0.26           0.00
```

2.2 Plot segmented profile

To plot the PSSeg segmentation results together with the true breakpoints, do :

```
> plotSeg(datS, list(true=sim$bkp, est=resRBS$bestBkp))
```

2.3 Results evaluation

The `PSSeg` function returns the original segmentation (by `RBS`), the result of the pruning step, and the best model (among those selected by dynamic programming) according to the criterion proposed by [2].

The quality of the best segmentation can be assessed by calculating the true positive rate and true negative rate. The true positive rate (TPR) is defined as the proportion of true change points for which there exists a candidate change point closer than a given tolerance `tol`. The false positive rate (FPR) is defined as the proportion of true negative (all those which are not change points) for which the candidate change points are out of tolerance area and those in tolerance area where there already exists a candidate change point. The true negative rate (TNR) is defined as 1-FPR. By construction, $TPR \in \{0, 1/K, \cdots, 1 - 1/K, 1\}$ where K is the number of true change points and TNR is very close to 1.

```
> print(getTprTnr(resRBS$bestBkp, sim$bkp, nrow(datS), tol=5, relax = -1))


      TPR       TNR
0.8000000 0.9998999
```

Obviously, this performance measure depends on the chosen tolerance:

```
> perf <- sapply(0:10, FUN=function(tol) {
+    getTprTnr(resRBS$bestBkp, sim$bkp, nrow(datS), tol=tol,relax = -1)
+ })
> print(perf)


         [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
TPR 0.2000000 0.7000000 0.7000000 0.7000000 0.8000000 0.8000000 0.8000000
TNR 0.9995998 0.9998499 0.9998499 0.9998499 0.9998999 0.9998999 0.9998999
         [,8]      [,9]     [,10]     [,11]
TPR 0.8000000 0.8000000 0.8000000 0.90000
TNR 0.9998999 0.9998999 0.9998999 0.99995
```

3 PSSeg segmentation using GFLARS

We can now use the `PSSeg` function to segment signals with GFLARS method only on heterozygous SNP. The method consists in three steps:

1. run a fast (yet approximate) segmentation on these signals in order to obtain a set of (at most hundreds of) candidate change points. This is done using Group Fused Lars [5];

2. prune the obtained set of change points usign dynamic programming [1]

3. select the best number of change points using a model selection criterion proposed by [2]

3.1 Initial segmentation and pruning

```
> resGFL <- PSSeg(data=datS, K=5*K, flavor="GFLars", profile=TRUE, stat = "(c,d)|het")
```

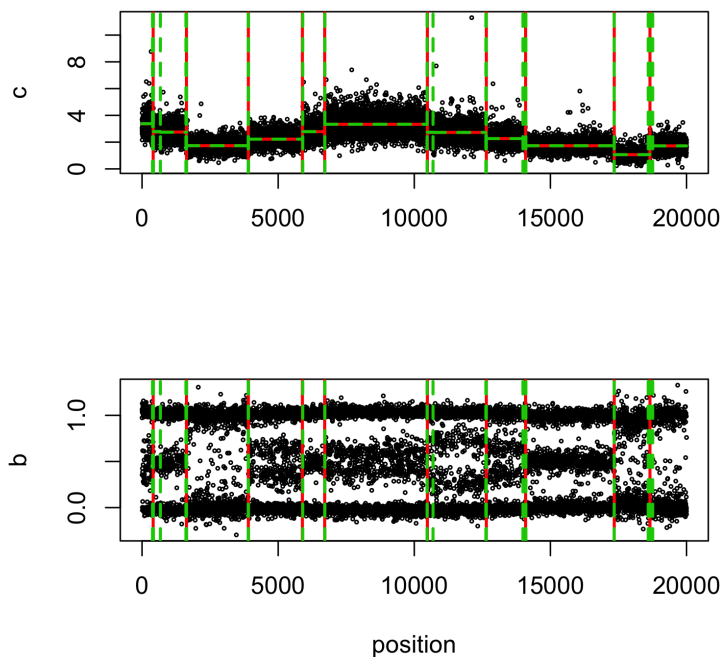Note that this is fast due to the low number in the data.

```
> resGFL$prof[, "time"]

  smoothing segmentation        dpseg
       0.00         0.18         0.06
```

3.2 Plot segmented profile

To plot the PSSeg segmentation results together with the true breakpoints, do
:

```
> plotSeg(datS, list(true=sim$bkp, est=resGFL$bestBkp))
```


3.3 Results evaluation

The PSSeg function returns the original segmentation (by GFLARS), the result
of the pruning step, and the best model (among those selected by dynamic
programming) according to the criterion proposed by [2].

```
> print(getTprTnr(resGFL$bestBkp, sim$bkp, nrow(datS), tol=15, relax = -1))
```

```
       TPR       TNR
1.0000000 0.9995498
```

Obviously, this performance measure depends on the chosen tolerance:

```
> perf <- sapply(0:20, FUN=function(tol) {
+   getTprTnr(resGFL$bestBkp, sim$bkp, nrow(datS), tol=tol, relax = -1)
+ })
> print(perf)


          [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
TPR 0.0000000 0.2000000 0.3000000 0.5000000 0.5000000 0.6000000 0.6000000
TNR 0.9990495 0.9991495 0.9991996 0.9992996 0.9992996 0.9993496 0.9993496
          [,8]      [,9]     [,10]     [,11]     [,12]     [,13]     [,14]
TPR 0.6000000 0.7000000 0.7000000 0.8000000 0.9000000 0.9000000 1.0000000
TNR 0.9993496 0.9993997 0.9993997 0.9994497 0.9994997 0.9994997 0.9995498
         [,15]     [,16]     [,17]     [,18]     [,19]     [,20]     [,21]
TPR 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000 1.0000000
TNR 0.9995498 0.9995498 0.9995498 0.9995498 0.9995498 0.9995498 0.9995498
```

A   Session information

```
> sessionInfo()


R version 2.15.2 (2012-10-26)
Platform: i386-apple-darwin9.8.0/i386 (32-bit)

locale:
[1] C

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

other attached packages:
[1] jointSeg_0.3.8    acnr_0.1.2        matrixStats_0.6.2 R.utils_1.19.3
[5] R.oo_1.11.4       R.methodsS3_1.4.2

loaded via a namespace (and not attached):
[1] tools_2.15.2
```

References

[1] R. Bellman. Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767, 1956.

[2] E. Lebarbier. Detecting multiple change-points in the mean of gaussian process by model selection. *Signal processing*, 85(4):717–736, 2005.

[3] Pierre Neuvial, Henrik Bengtsson, and Terence P Speed. Statistical analysis of single nucleotide polymorphism microarrays in cancer studies. In *Handbook of Statistical Bioinformatics*, Springer Handbooks of Computational Statistics. Springer, 1st edition, March 2011.

[4] Morgane Pierre-Jean and Pierre Neuvial. In preparation.

[5] J.-P. Vert and K. Bleakley. Fast detection of multiple change-points shared by many signals using group LARS. *Advances in Neural Information Processing Systems*, 23:2343–2351, 2010.