The *segment* function to fit a piecewise constant curve

Wolfgang Huber

October 1, 2012

Contents

1 A simple example

The problem of segmenting a series of numbers into piecewise constant segments occurs in multiple application areas. Two examples are

- arrayCGH data, where the segments correspond to regions of copy number gain, loss, or no change.
- tiling microarray data for transcription profiling, where the segments correspond to transcripts. Here we assume that the probe effects (which lead to different fluorescence intensities even for the same mRNA abundance) have been normalized away, so that all probes for the same unique target sequence have approximately, and in expectation, the same fluorescence.

To demonstrate the algorithm, let us generate simulated data:

```
> genData = function(lenx, nrSeg, nrRep=1, stddev=0.1) {
+   x  = matrix(as.numeric(NA), nrow=lenx, ncol=nrRep)
+   cp = sort(sample(1:floor(lenx/15), nrSeg-1) * 15)
```

```
+   cpb = c(1, cp, lenx+1)
+   s  = 0
+   for (j in 2:length(cpb)) {
+     sel = cpb[j-1]:(cpb[j]-1)
+     s = (.5+runif(1))*sign(rnorm(1))+s
+     x[sel, ] = rnorm(length(sel)*nrRep, mean=s, sd=stddev)
+   }
+   return(list(x=x, cp=cp))
+ }

> set.seed(4711)
> lenx = 1000
> nrSeg = 10
> gd = genData(lenx, nrSeg)
> plot(gd$x, pch=".")
> abline(v=gd$cp, col="blue")
```

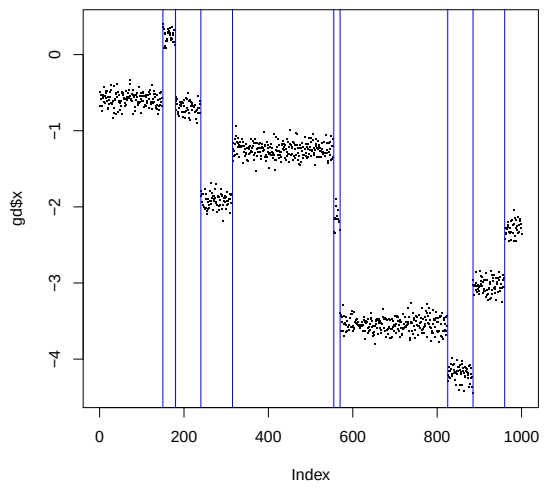

Figure 1: A simulated data example with 10 segments. Their estimated locations are shown with blue vertical lines

The result is shown in Figure 1. We can use the function *segment* to reconstruct the change-points from the data in `gd$x` alone.

```
> library("tilingArray")

> maxseg = 12
> maxk = 500
> seg  = segment(gd$x, maxk=maxk, maxseg=maxseg)
> seg

Object of class 'segmentation':
Data matrix: 1000 x 1
Change point estimates for number of segments S = 1:12
Selected S = NA

> seg@breakpoints[nrSeg+(-1:1)]

[[1]]
     estimate
[1,]      150
[2,]      180
[3,]      240
[4,]      315
[5,]      570
[6,]      825
[7,]      885
[8,]      960

[[2]]
      estimate
 [1,]      150
 [2,]      180
 [3,]      240
 [4,]      315
 [5,]      555
 [6,]      570
 [7,]      825
 [8,]      885
 [9,]      960

[[3]]
      estimate
 [1,]      150
 [2,]      180
```

```
 [3,]      240
 [4,]      315
 [5,]      555
 [6,]      570
 [7,]      825
 [8,]      882
 [9,]      885
[10,]      960
```

```
> gd$cp
```

```
[1] 150 180 240 315 555 570 825 885 960
```

We see that the 10-th element of the list `segbreakpoints` exactly reconstructs the change-points `gd$cp` that were used in the simulation.

The parameters `maxseg` and `maxk` are the maximum number of segments and the maximum length per segment. The algorithm finds for each value of k from `1` to `maxseg` the best segmentation under the restriction that no individual segment be longer than `maxk`. In the paper of Picard et al. [1] and in their software, `maxk` is implicitely set to the number of data points `length(x)`. I have introduced this parameter to reduce the algorithm's complexity. The complexity of Picard's software is `length(x)*length(x)` in memory and `length(x)*length(x)*maxcp` in time, the complexity of the *segment* function is `length(x)*maxk` in memory and `length(x)*maxk*maxcp` in time. As I am applying it to data with `length(x)` $\approx 10^5$ and `maxk` ≈ 250, the difference can be substantial.

2 More testing of the change-point estimates on simulated data

Here is a little for-loop that generates data using random parameters and checks whether *segment* can reconstruct them.

```
> for(i in 1:20){
+   gd  = genData(lenx, nrSeg)
+   seg = segment(gd$x, maxk=maxk, maxseg=maxseg)
+   stopifnot(seg@breakpoints[[nrSeg]][, "estimate"] == gd$cp)
+ }
```

3 Model selection on simulated data

In this section we show that the BIC works pretty well for finding the correct number of segments (parameter S in the paper) *if* the data are generated by the model.

```
> nrSeg = 22
> gd = genData(lenx, nrSeg, nrRep=2, stddev=1/3)
> s = segment(gd$x, maxk=lenx, maxseg=as.integer(nrSeg * 2.5))
```

Plot the segmented data (Figure 2a)

```
> par(mai=c(1,1,0.1,0.01))
> plot(row(gd$x), gd$x, pch=".")
```

and the log likelihoods and the penalized log likelihoods. This is similar to what is done in the segmentation.Rnw vignette for *real* data. and call it it:

```
> par(mai=c(1,1,0.1,0.01))
> plotPenLL(s)
```

The result is shown in Figure 2b.

```
> which.max(logLik(s, penalty="AIC"))

[1] 27

> which.max(logLik(s, penalty="BIC"))

[1] 22
```

References

[1] A statistical approach for CGH microarray data analysis. Franck Picard, Stephane Robin, Marc Lavielle, Christian Vaisse, Gilles Celeux, Jean-Jacques Daudin. Rapport de recherche No. 5139, Mars 2004, *Institut National de Recherche en Informatique et en Automatique (INRIA)*, ISSN 0249-6399. http://www.inapg.fr/ens_rech/mathinfo/recherche/mathematique/outil.html

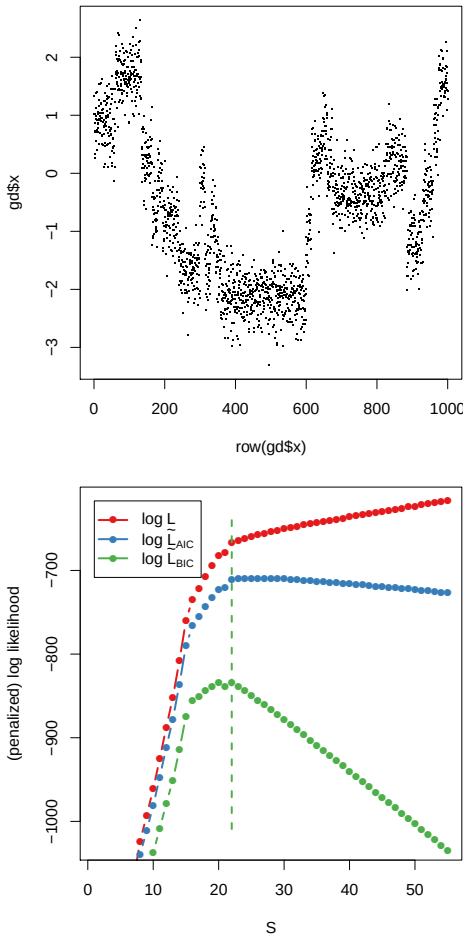

Figure 2: a) simulated data example with `nrSeg=22` segments and vertical lines representing the fitted model with S =22, selected by maximum $\log L_{\mathrm{BIC}}$. b) log-likelihood $\log L$, penalized likelihoods $\log \tilde{L}_{\mathrm{AIC}}$ and $\log \tilde{L}_{\mathrm{BIC}}$.