

Analysis of data from aCGH experiments using parallel computing and ff objects: long list of examples

Ramón Díaz-Uriarte¹

June 21, 2013

1. Department of Biochemistry, Universidad Autonoma de Madrid Instituto de Investigaciones Biomedicas “Alberto Sols” (UAM-CSIC), Madrid (SPAIN).
rdiaz02@gmail.com

Contents

1	This vignette	1
2	Creating objects	1
3	The examples	4
3.1	RAM objects and forking	4
3.2	ff objects and cluster	5
3.3	ff objects and forking	6
3.4	Comparing output	8
4	Clean up actions	10

1 This vignette

We provide here example calls of all segmentation methods, with different options for methods, as well as different options for type of input object and clustering. This is provided here as both extended help and as a simple way of checking that all the functions can be run and yield the same results regardless of type of input and clustering.

2 Creating objects

We must ensure that we can run this vignette as stand alone. Thus, we load the package and create all necessary objects. This repeats work done in the main vignette.

We first try to move to the “/tmp” directory, if it exists. If it does not, the code will be executed in your current directory.

```
> try(setwd("~/tmp"))  
  
> library(ADaCGH2)  
> ## loading in-RAM objects  
> data(inputEx)  
> summary(inputEx)
```

ID	chromosome	position	L.1
Hs.101850: 1	Min. :1.00	Min. :1.18e+06	Min. : -1.078
Hs.1019 : 1	1st Qu.:1.00	1st Qu.:3.60e+07	1st Qu.: -0.226
Hs.105460: 1	Median :2.00	Median :7.08e+07	Median : -0.016
Hs.105656: 1	Mean :2.28	Mean :9.26e+07	Mean : -0.035
Hs.105941: 1	3rd Qu.:3.00	3rd Qu.:1.50e+08	3rd Qu.: 0.160
Hs.106674: 1	Max. :5.00	Max. :2.44e+08	Max. : 0.883
(Other) :494			NA's :5

L.2	m4	m5	L3
Min. : -0.795	Min. : -0.187	Min. : -4.67	Min. : -13.27
1st Qu.: -0.139	1st Qu.: 1.979	1st Qu.: -0.02	1st Qu.: 3.63
Median : -0.006	Median : 2.281	Median : 0.44	Median : 3.92
Mean : 0.008	Mean : 3.450	Mean : 1.60	Mean : 1.98
3rd Qu.: 0.134	3rd Qu.: 5.824	3rd Qu.: 3.04	3rd Qu.: 4.11
Max. : 1.076	Max. : 6.604	Max. : 9.60	Max. : 6.37
NA's :15		NA's :41	NA's :9

m6
Min. : -0.77
1st Qu.: -0.23
Median : -0.04
Mean : -0.04
3rd Qu.: 0.16
Max. : 0.78
NA's :203

```
> head(inputEx)
```

	ID	chromosome	position	L.1	L.2	m4	m5
1*1180411*Hs.212680	Hs.212680	1	1180411	NA	0.038	6.226	3.226
1*1188041.5*Hs.129780	Hs.129780	1	1188042	NA	0.028	6.174	3.174
1*1194444*Hs.42806	Hs.42806	1	1194444	NA	0.042	6.174	3.174
1*1332537*Hs.76239	Hs.76239	1	1332537	NA	0.285	5.624	2.624
1*2362211*Hs.40500	Hs.40500	1	2362211	NA	0.058	5.851	2.851
1*2372287*Hs.449936	Hs.449936	1	2372287	0.294	-0.006	5.685	2.685

	L3	m6
1*1180411*Hs.212680	6.038	NA
1*1188041.5*Hs.129780	6.028	NA
1*1194444*Hs.42806	6.042	NA
1*1332537*Hs.76239	NA	NA
1*2362211*Hs.40500	NA	NA
1*2372287*Hs.449936	NA	NA

```
> cgh.dat <- inputEx[, -c(1, 2, 3)]
> chrom.dat <- as.integer(inputEx[, 2])
> pos.dat <- inputEx[, 3]
> ## choosing working dir for cluster
> originalDir <- getwd()
> if(!file.exists("ADaCGH2_vignette_tmp_dir"))
+   dir.create("ADaCGH2_vignette_tmp_dir")
> setwd("ADaCGH2_vignette_tmp_dir")
> ## creating ff objects
> fnameRdata <- list.files(path = system.file("data", package = "ADaCGH2"),
+                           full.names = TRUE, pattern = "inputEx.RData")
```

```

> inputToADaCGH(ff.or.RAM = "ff",
+               RDatafilename = fnameRdata)

... done reading; starting checks

... checking identical MidPos

... checking need to reorder inputData, data.frame version

... done with checks; starting writing

... done writing/saving probeNames

... done writing/saving chromData

... done writing/saving posData

... done writing/saving cghData

```

Calling gc at end

```

          used (Mb) gc trigger   (Mb) max used   (Mb)
Ncells 1565209 83.6    2403845 128.4   1967602 105.1
Vcells 1528146 11.7    2481603  19.0   2153106  16.5

```

Files saved in current directory
/home/ramon/tmp/ADaCGH2_vignette_tmp_dir
with names :
chromData.RData, posData.RData, cghData.RData, probeNames.RData.

```

> ## initializing cluster
> number.of.nodes <- detectCores()
> cl2 <- parallel::makeCluster(number.of.nodes,"PSOCK")
> parallel::clusterSetRNGStream(cl2)
> parallel::setDefaultCluster(cl2)
> parallel::clusterEvalQ(NULL, library("ADaCGH2"))

```

```

[[1]]
[1] "ADaCGH2"      "ffbase"      "ff"          "bit"         "tools"
[6] "snapCGH"     "DNACopy"     "limma"       "waveslim"    "aCGH"
[11] "multtest"    "survival"    "splines"     "cluster"     "tilingArray"
[16] "pixmap"      "Biobase"     "BiocGenerics" "parallel"    "methods"
[21] "stats"       "graphics"    "grDevices"   "utils"       "datasets"
[26] "base"

```

```

[[2]]
[1] "ADaCGH2"      "ffbase"      "ff"          "bit"         "tools"
[6] "snapCGH"     "DNACopy"     "limma"       "waveslim"    "aCGH"
[11] "multtest"    "survival"    "splines"     "cluster"     "tilingArray"
[16] "pixmap"      "Biobase"     "BiocGenerics" "parallel"    "methods"
[21] "stats"       "graphics"    "grDevices"   "utils"       "datasets"
[26] "base"

```

```
[[3]]
[1] "ADaCGH2"      "ffbase"      "ff"          "bit"         "tools"
[6] "snapCGH"      "DNACopy"     "limma"       "waveslim"    "aCGH"
[11] "multtest"     "survival"    "splines"     "cluster"     "tilingArray"
[16] "pixmap"       "Biobase"     "BiocGenerics" "parallel"    "methods"
[21] "stats"        "graphics"    "grDevices"   "utils"       "datasets"
[26] "base"
```

```
[[4]]
[1] "ADaCGH2"      "ffbase"      "ff"          "bit"         "tools"
[6] "snapCGH"      "DNACopy"     "limma"       "waveslim"    "aCGH"
[11] "multtest"     "survival"    "splines"     "cluster"     "tilingArray"
[16] "pixmap"       "Biobase"     "BiocGenerics" "parallel"    "methods"
[21] "stats"        "graphics"    "grDevices"   "utils"       "datasets"
[26] "base"
```

```
> wdir <- getwd()
> parallel::clusterExport(NULL, "wdir")
> parallel::clusterEvalQ(NULL, setwd(wdir))
```

```
[[1]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"
```

```
[[2]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"
```

```
[[3]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"
```

```
[[4]]
[1] "/home/ramon/tmp/ADaCGH2_vignette_tmp_dir"
```

3 The examples

3.1 RAM objects and forking

```
> cbs.mergel.RAM.fork <- pSegmentDNACopy(cgh.dat, chrom.dat,
+                                       merging = "mergeLevels")
> cbs.mad.RAM.fork <- pSegmentDNACopy(cgh.dat, chrom.dat, merging = "MAD")
> cbs.none.RAM.fork <- pSegmentDNACopy(cgh.dat, chrom.dat, merging = "none")
> hmm.mergel.RAM.fork <- pSegmentHMM(cgh.dat, chrom.dat, merging = "mergeLevels")
> hmm.mad.RAM.fork <- pSegmentHMM(cgh.dat, chrom.dat, merging = "MAD")
> hs.mergel.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+                                       merging = "mergeLevels")
> hs.mad.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+                                       merging = "MAD")
> hs.none.RAM.fork <- pSegmentHaarSeg(cgh.dat, chrom.dat,
+                                       merging = "none")
> glad.RAM.fork <- pSegmentGLAD(cgh.dat, chrom.dat)
> biohmm.mergel.RAM.fork <- pSegmentBioHMM(cgh.dat,
+                                           chrom.dat,
```

```

+             pos.dat,
+             merging = "mergeLevels")
> biohmm.mad.RAM.fork <- pSegmentBioHMM(cgh.dat,
+             chrom.dat,
+             pos.dat,
+             merging = "MAD")
> biohmm.mad.bic.RAM.fork <- pSegmentBioHMM(cgh.dat,
+             chrom.dat,
+             pos.dat,
+             merging = "MAD",
+             aic.or.bic = "BIC")
> cghseg.mergel.RAM.fork <- pSegmentCGHseg(cgh.dat,
+             chrom.dat,
+             merging = "mergeLevels")
> cghseg.mad.RAM.fork <- pSegmentCGHseg(cgh.dat,
+             chrom.dat,
+             merging = "MAD")
> cghseg.none.RAM.fork <- pSegmentCGHseg(cgh.dat,
+             chrom.dat,
+             merging = "none")
> waves.mergel.RAM.fork <- pSegmentWavelets(cgh.dat,
+             chrom.dat, merging = "mergeLevels")
> waves.mad.RAM.fork <- pSegmentWavelets(cgh.dat,
+             chrom.dat, merging = "MAD")
> waves.none.RAM.fork <- pSegmentWavelets(cgh.dat,
+             chrom.dat, merging = "none")
>

```

3.2 *ff* objects and cluster

Compared to the section 3.1, the main differences are that we explicitly set the `typeParall` argument to "cluster" (the default is "fork") and the change in the names of the input data (which now refer to the names of the RData objects that contain the *ff* objects).

```

> cbs.mergel.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+             merging = "mergeLevels",
+             typeParall = "cluster")
> cbs.mad.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+             merging = "MAD",
+             typeParall = "cluster")
> cbs.none.ff.cluster <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+             merging = "none",
+             typeParall = "cluster")
> hmm.mergel.ff.cluster <- pSegmentHMM("cghData.RData", "chromData.RData",
+             merging = "mergeLevels",
+             typeParall = "cluster")
> hmm.mad.ff.cluster <- pSegmentHMM("cghData.RData", "chromData.RData",
+             merging = "MAD",
+             typeParall = "cluster")
> hs.mergel.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+             merging = "mergeLevels",
+             typeParall = "cluster")

```

```

> hs.mad.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+                                     merging = "MAD", typeParall = "cluster")
> hs.none.ff.cluster <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+                                     merging = "none", typeParall = "cluster")
> glad.ff.cluster <- pSegmentGLAD("cghData.RData", "chromData.RData",
+                                 typeParall = "cluster")
> biohmm.mergel.ff.cluster <- pSegmentBioHMM("cghData.RData",
+                                             "chromData.RData",
+                                             "posData.RData",
+                                             merging = "mergeLevels",
+                                             typeParall = "cluster")
> biohmm.mad.ff.cluster <- pSegmentBioHMM("cghData.RData",
+                                          "chromData.RData",
+                                          "posData.RData",
+                                          merging = "MAD",
+                                          typeParall = "cluster")
> biohmm.mad.bic.ff.cluster <- pSegmentBioHMM("cghData.RData",
+                                              "chromData.RData",
+                                              "posData.RData",
+                                              merging = "MAD",
+                                              aic.or.bic = "BIC",
+                                              typeParall = "cluster")
> cghseg.mergel.ff.cluster <- pSegmentCGHseg("cghData.RData",
+                                           "chromData.RData",
+                                           merging = "mergeLevels",
+                                           typeParall = "cluster")
> cghseg.mad.ff.cluster <- pSegmentCGHseg("cghData.RData",
+                                         "chromData.RData",
+                                         merging = "MAD",
+                                         typeParall = "cluster")
> cghseg.none.ff.cluster <- pSegmentCGHseg("cghData.RData",
+                                          "chromData.RData",
+                                          merging = "none",
+                                          typeParall = "cluster")
> waves.mergel.ff.cluster <- pSegmentWavelets("cghData.RData",
+                                              "chromData.RData",
+                                              merging = "mergeLevels",
+                                              typeParall = "cluster")
> waves.mad.ff.cluster <- pSegmentWavelets("cghData.RData",
+                                          "chromData.RData",
+                                          merging = "MAD",
+                                          typeParall = "cluster")
> waves.none.ff.cluster <- pSegmentWavelets("cghData.RData",
+                                           "chromData.RData",
+                                           merging = "none",
+                                           typeParall = "cluster")
>

```

3.3 *ff* objects and forking

The main difference with section 3.2 is the argument `typeParall`; we did not need to pass it explicitly (since the default is `fork`), but we will do for clarity.

```

> cbs.mergel.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+                                     merging = "mergeLevels",
+                                     typeParall = "fork")
> cbs.mad.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+                                   merging = "MAD",
+                                   typeParall = "fork")
> cbs.none.ff.fork <- pSegmentDNAcopy("cghData.RData", "chromData.RData",
+                                    merging = "none", typeParall = "fork")
> hmm.mergel.ff.fork <- pSegmentHMM("cghData.RData", "chromData.RData",
+                                  merging = "mergeLevels", typeParall = "fork")
> hmm.mad.ff.fork <- pSegmentHMM("cghData.RData", "chromData.RData",
+                                merging = "MAD", typeParall = "fork")
> hs.mergel.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+                                     merging = "mergeLevels", typeParall = "fork")
> hs.mad.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+                                   merging = "MAD", typeParall = "fork")
> hs.none.ff.fork <- pSegmentHaarSeg("cghData.RData", "chromData.RData",
+                                    merging = "none", typeParall = "fork")
> glad.ff.fork <- pSegmentGLAD("cghData.RData", "chromData.RData",
+                              typeParall = "fork")
> biohmm.mergel.ff.fork <- pSegmentBioHMM("cghData.RData",
+                                         "chromData.RData",
+                                         "posData.RData",
+                                         merging = "mergeLevels",
+                                         typeParall = "fork")
> biohmm.mad.ff.fork <- pSegmentBioHMM("cghData.RData",
+                                       "chromData.RData",
+                                       "posData.RData",
+                                       merging = "MAD",
+                                       typeParall = "fork")
> biohmm.mad.bic.ff.fork <- pSegmentBioHMM("cghData.RData",
+                                           "chromData.RData",
+                                           "posData.RData",
+                                           merging = "MAD",
+                                           aic.or.bic = "BIC",
+                                           typeParall = "fork")
> cghseg.mergel.ff.fork <- pSegmentCGHseg("cghData.RData",
+                                         "chromData.RData",
+                                         merging = "mergeLevels",
+                                         typeParall = "fork")
> cghseg.mad.ff.fork <- pSegmentCGHseg("cghData.RData",
+                                       "chromData.RData",
+                                       merging = "MAD", typeParall = "fork")
> cghseg.none.ff.fork <- pSegmentCGHseg("cghData.RData",
+                                       "chromData.RData",
+                                       merging = "none", typeParall = "fork")
> waves.merge.ff.fork <- pSegmentWavelets("cghData.RData",
+                                          "chromData.RData",
+                                          merging = "mergeLevels",
+                                          typeParall = "fork")
> waves.mad.ff.fork <- pSegmentWavelets("cghData.RData",
+                                       "chromData.RData",

```

```

+                               merging = "MAD",
+                               typeParall = "fork")
> waves.none.ff.fork <- pSegmentWavelets("cghData.RData",
+                                       "chromData.RData",
+                                       merging = "none",
+                                       typeParall = "fork")
>
>

```

3.4 Comparing output

Here we verify that using different input and clustering methods does not change the results. Before carrying out the comparisons, however, we open the *ff* objects gently.

First, we will open the objects created above (same objects as were also created in the main vignette, in section "Carrying out segmentation and calling"). Instead of inserting many calls to each individual object, we open all available objects that match *ff.cluster*. To do that quickly we store the names of the objects

```
> ff.cluster.obj <- ls(pattern = "*.ff.cluster")
```

pages with the string "TRUE")

```

> tmpout <-
+   capture.output(
+     lapply(ff.cluster.obj, function(x) lapply(get(x), open))
+   )

```

We repeat that operation with the output from section 3.3:

```

> ff.fork.obj <- ls(pattern = "*.ff.fork")
> tmpout <-
+   capture.output(
+     lapply(ff.fork.obj, function(x) lapply(get(x), open))
+   )
>

```

And we create the list of results from the RAM and forking runs (no need for special opening here, since these are not *ff* objects)

```
> RAM.fork.obj <- ls(pattern = "*.RAM.fork")
```

We can now compare the output. We want to compare the output from three different methods, so we need to run three comparisons (this is what we did explicitly in the help for *pSegment*). Since this is a very repetitive operation, we define a small utility function that will return TRUE if both components (*outSmoothed* and *outState*) of all three objects are identical. (Since the function will take as input not an actual object, but a name, we use *get* inside the function.)

We use *all.equal* to compare the output from the smoothing, to allow for possible numerical fuzz (that could result from differences in storage). When comparing the assigned state, however, we check for exact identity.

```

> identical3 <- function(x, y, z) {
+   comp1 <- all.equal(get(x)$outSmoothed[ , ], get(y)$outSmoothed[ , ])
+   comp2 <- all.equal(get(y)$outSmoothed[ , ], get(z)$outSmoothed[ , ])

```



```

+   comp3 <- identical(get(x)$outState[ , ], get(y)$outState[ , ])
+   comp4 <- identical(get(y)$outState[ , ], get(z)$outState[ , ])
+   if (!all(isTRUE(comp1), isTRUE(comp2), comp3, comp4)) {
+     cat(paste("Comparing ", x, y, z, "\n",
+               "not equal: some info from comparisons.\n",
+               "\n comp1 = ", paste(comp1, sep = " ", collapse = "\n    "),
+               "\n comp2 = ", paste(comp2, sep = " ", collapse = "\n    "),
+               "\n comp3 = ", paste(comp3, sep = " ", collapse = "\n    "),
+               "\n comp4 = ", paste(comp4, sep = " ", collapse = "\n    "),
+               "\n\n"))
+     return(FALSE)
+   } else {
+     TRUE
+   }
+ }

```

You should expect most (though not necessarily all) the comparisons to yield a TRUE. In some cases, however, different runs of the same method might not yield the same results (e.g., CBS, HMM, etc). If you get non-identical results, you can try running those methods a few times, to check for differences.

```

> mapply(identical3, RAM.fork.obj,
+         ff.fork.obj, ff.cluster.obj)

```

```

Comparing  hmm.mad.RAM.fork hmm.mad.ff.fork hmm.mad.ff.cluster
not equal: some info from comparisons.

```

```

comp1 = Component 6: Mean relative difference: 0.1417
comp2 = Component 3: Mean relative difference: 0.02184
      Component 6: Mean relative difference: 0.1534
comp3 = TRUE
comp4 = TRUE

```

```

Comparing  hmm.mergel.RAM.fork hmm.mergel.ff.fork hmm.mergel.ff.cluster
not equal: some info from comparisons.

```

```

comp1 = Component 3: Mean relative difference: 0.007541
comp2 = Component 3: Mean relative difference: 0.007512
      Component 4: Mean relative difference: 0.02446
comp3 = TRUE
comp4 = FALSE

```

biohmm.mad.bic.RAM.fork	biohmm.mad.RAM.fork	biohmm.mergel.RAM.fork
TRUE	TRUE	TRUE
cbs.mad.RAM.fork	cbs.mergel.RAM.fork	cbs.none.RAM.fork
TRUE	TRUE	TRUE
cghseg.mad.RAM.fork	cghseg.mergel.RAM.fork	cghseg.none.RAM.fork
TRUE	TRUE	TRUE
glad.RAM.fork	hmm.mad.RAM.fork	hmm.mergel.RAM.fork
TRUE	FALSE	FALSE
hs.mad.RAM.fork	hs.mergel.RAM.fork	hs.none.RAM.fork
TRUE	TRUE	TRUE

<code>waves.mad.RAM.fork</code>	<code>waves.mergel.RAM.fork</code>	<code>waves.none.RAM.fork</code>
<code>TRUE</code>	<code>TRUE</code>	<code>TRUE</code>

>

(Of course, we depend on the lists of names of objects having the output from the same method and option in the same position, which is the case in these examples).

4 Clean up actions

These are not strictly necessary, but we will explicitly stop the cluster. In this vignette, we will not execute the code below to remove the directory we created or the objects, in case you want to check them out or play around with them, but the code is below.

To make sure there are no file permission problems, we add code below to explicitly delete some of the "ff" files and objects (and we wait a few seconds to allow pending I/O operations to happen before we delete the directory).

```
> parallel::stopCluster(cl2)

> ## This is the code to remove all the files we created
> ## and the temporary directory.
> ## We are not executing it!
>
> load("chromData.RData")
> load("posData.RData")
> load("cghData.RData")
> delete(cghData); rm(cghData)
> delete(posData); rm(posData)
> delete(chromData); rm(chromData)
> tmpout <-
+   capture.output(
+     lapply(ff.fork.obj, function(x) {
+       lapply(get(x), delete)}))
> rm(list = ff.fork.obj)
> tmpout <-
+   capture.output(
+     lapply(ff.cluster.obj, function(x) {
+       lapply(get(x), delete)}))
> rm(list = ff.cluster.obj)
> setwd(originalDir)
> print(getwd())
> Sys.sleep(3)
> unlink("ADaCGH2_vignette_tmp_dir", recursive = TRUE)
> Sys.sleep(3)
```