# Using the charm package to estimate DNA methylation levels and find differentially methylated regions

Peter Murakami*, Martin Aryee, Rafael Irizarry

October 10, 2012

Johns Hopkins School of Medicine / Johns Hopkins School of Public Health
Baltimore, MD, USA

## 1 Introduction

The Bioconductor package charm can be used to analyze DNA methylation data generated using McrBC fractionation and two-color Nimblegen microarrays. It is customized for use with data the from the custom CHARM microarray [2], but can also be applied to many other Nimblegen designs. The preprocessing and normalization methods are described in detail in [1].

Functions include:

- Quality control
- Finding suitable control probes for normalization
- Percentage methylation estimates
- Identification of differentially methylated regions
- Plotting of differentially methylated regions

As input we will need raw Nimblegen data (.xys) files and a corresponding annotation package built with pdInfoBuilder. This vignette uses the following packages:

- charm: contains the analysis functions
- charmData: an example dataset
- pd.charm.hg18.example: the annotation package for the example dataset

*pmurakam@jhsph.edu

- BSgenome.Hsapiens.UCSC.hg18: A BSgenome object containing genomic sequence used for finding non-CpG control probes

Each sample is represented by two xys files corresponding to the untreated (green) and methyl-depleted (red) channels. The 532.xys and 635.xys suffixes indicate the green and red channels respectively.

# 2 Analyzing data from the custom CHARM microarray

Load the charm package:

```
R> library(charm)
R> library(charmData)
```

# 3 Read in raw data

Get the name of your data directory (in this case, the example data):

```
R> dataDir <- system.file("data", package="charmData")
R> dataDir
```

```
[1] "/home/biocbuild/bbs-2.13-bioc/R/library/charmData/data"
```

First we read in the sample description file:

```
R> phenodataDir <- system.file("extdata", package="charmData")
R> pd <- read.delim(file.path(phenodataDir, "phenodata.txt"))
R> phenodataDir
```

```
[1] "/home/biocbuild/bbs-2.13-bioc/R/library/charmData/extdata"
```

```
R> pd
```

```
          filename   sampleID tissue
1   136421_532.xys  441_liver  liver
2   136421_635.xys  441_liver  liver
3   136600_532.xys 449_spleen spleen
4   136600_635.xys 449_spleen spleen
5  3788602_532.xys  449_liver  liver
6  3788602_635.xys  449_liver  liver
7  3822402_532.xys 441_spleen spleen
8  3822402_635.xys 441_spleen spleen
9  5739902_532.xys  624_colon  colon
10 5739902_635.xys  624_colon  colon
11 5875602_532.xys  441_colon  colon
12 5875602_635.xys  441_colon  colon
```

A valid sample description file should contain at least the following (arbitrarily named) columns:

- a filename column
- a sample ID column
- a group label column (optional)

The sample ID column is used to pair the methyl-depleted and untreated data files for each sample. The group label column is used when identifying differentially methylated regions between experimental groups.

The `validatePd` function can be used to validate the sample description file. When called with only a sample description data frame and no further options `validatePd` will try to guess the contents of the columns.

```
R> res <- validatePd(pd)
```

Now we read in the raw data. The `readCharm` command makes the assumption (unless told otherwise) that the two xys files for a sample have the same file name up to the suffixes 532.xys (untreated) and 635.xys (methyl-depleted). The sampleNames argument is optional. Note that if the ff package has been loaded previously in your R session, the output of readCharm will contain `ff` rather than `matrix` objects, and all subsequent charm functions acting on it (except those in pipeline 2 described below) will recognize this and use `ff` objects also. Using the ff package is recommended when the data set is otherwise too large for the amount of memory available.

```
R> rawData <- readCharm(files=pd$filename, path=dataDir, sampleKey=pd,
                        sampleNames=pd$sampleID)

Checking designs for each XYS file... Done.
Allocating memory... Done.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/136421_532.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/136600_532.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/3788602_532.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/3822402_532.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/5739902_532.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/5875602_532.xys.
Checking designs for each XYS file... Done.
Allocating memory... Done.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/136421_635.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/136600_635.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/3788602_635.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/3822402_635.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/5739902_635.xys.
Reading /home/biocbuild/bbs-2.13-bioc/R/library/charmData/data/5875602_635.xys.

R> rawData
```

```
TilingFeatureSet (storageMode: lockedEnvironment)
assayData: 243129 features, 6 samples
  element names: channel1, channel2
protocolData
  rowNames: 441_liver 449_spleen ... 441_colon (6
    total)
  varLabels: filenamesChannel1 filenamesChannel2
    dates1 dates2
  varMetadata: labelDescription channel
phenoData
  rowNames: 441_liver 449_spleen ... 441_colon (6
    total)
  varLabels: sampleID tissue arrayUT arrayMD
  varMetadata: labelDescription channel
featureData: none
experimentData: use 'experimentData(object)'
Annotation: pd.charm.hg18.example
```

# 4 Array quality assessment

We can calculate array quality scores and generate a pdf report with the `qcReport` command.

A useful quick way of assessing data quality is to examine the untreated channel where we expect every probe to have signal. Very low signal intensities on all or part of an array can indicate problems with hybridization or scanning. The CHARM array and many other designs include background probes that do not match any genomic sequence. Any signal at these background probes can be assumed to be the result of optical noise or cross-hybridization. Since the untreated channel contains total DNA a successful hybridization would have strong signal for all untreated channel genomic probes. The array signal quality score (pmSignal) is calculated as the average percentile rank of the signal robes among these background probes. A score of 100 means all signal probes rank above all background probes (the ideal scenario).

```
R> qual <- qcReport(rawData, file="qcReport.pdf")
R> qual
```

```
           pmSignal       sd1       sd2
441_liver  78.56437 0.1950274 0.1932112
449_spleen 81.46541 0.1755225 0.1227921
449_liver  83.95419 0.1249030 0.2409803
441_spleen 81.43751 0.1180708 0.1824810
624_colon  82.55727 0.1490854 0.2035761
441_colon  79.38069 0.3130266 0.3962373
```

The PDF quality report is shown in Appendix A. Three quality metrics are calculated for each array:

1. Average signal strength: the average percentile rank of untreated channel signal probes among the background (anti-genomic) probes.
2. Untreated channel signal standard deviation. The array is divided into a series of rectangular blocks and the average signal level calculated for each. Since probes are arranged randomly on the array there should be no large differences between blocks. Arrays with spatial artifacts have a larg standard deviation between blocks.
3. Methyl-depleted channel signal standard deviation.

To remove samples with a quality score less than 78, we could do this:

```
R> qc.min = 78
R> ##Remove arrays with quality scores below qc.min:
R> rawData=rawData[,qual$pmSignal>=qc.min]
R> qual=qual[qual$pmSignal>=qc.min,]
R> pd=pd[pd$sampleID%in%rownames(qual),]
R> pData(rawData)$qual=qual$pmSignal
```

and to identify which probes have a mean quality score above 75 we could do this:

```
R> pmq = pmQuality(rawData)
R> rmpmq = rowMeans(pmq)
R> okqc = which(rmpmq>75)
```

We now want to calculate probe-level percentage methylation estimates for each sample. As a first step we need to identify a suitable set of unmethylated control probes from CpG-free regions to be used in normalization.

```
R> library(BSgenome.Hsapiens.UCSC.hg18)
R> ctrlIdx <- getControlIndex(rawData, subject=Hsapiens, noCpGWindow=600)
```

We can check the success of the control probes by comparing their intensity distribution with the non-control probes (before any normalization in which the control probes are used).

```
R> cqc = controlQC(rawData=rawData, controlIndex=ctrlIdx, IDcol="sampleID",
           expcol="tissue", ylimits=c(-6,8),
           outfile="boxplots_check.pdf", height=7, width=9)
R> cqc
```

```
             non_control   control     diff
624_colon   3.539289e-01 -2.026216 2.380145
441_colon  -5.328170e-02 -1.723868 1.670586
441_liver  -1.475199e-01 -1.650623 1.503104
449_liver   3.455137e-01 -1.747331 2.092844
449_spleen -4.097402e-02 -2.404196 2.363222
441_spleen -1.070891e-05 -1.481341 1.481330
```

We can also access the probe annotation using standard functions from the oligo package.

```
R> chr = pmChr(rawData)
R> pns = probeNames(rawData)
R> pos = pmPosition(rawData)
R> seq = pmSequence(rawData)
R> pd  = pData(rawData)
```

# 5 Percentage methylation estimates and differentially methylated regions (DMRs)

The minimal code required to estimate methylation would be `p <- methp(rawData, controlIndex=ctrlIdx)`. However, it is often useful to get `methp` to produce a series of diagnostic density plots to help identify non-hybridization quality issues. The `plotDensity` option specifies the name of the output pdf file, and the optional `plotDensityGroups` can be used to give groups different colors. Remember that if the ff package was loaded before producing `rawData` with `readCharm`, the output of `methp` will be an `ff` rather than a `matrix` object. charm functions (except those in pipeline 2 described below) will handle `ff p` objects automatically.

```
R> p <- methp(rawData, controlIndex=ctrlIdx,
            plotDensity="density.pdf", plotDensityGroups=pd$tissue)
R> head(p)
```

```
     441_liver 449_spleen 449_liver 441_spleen 624_colon
[1,] 0.2233236  0.3849991 0.3835754  0.5535150 0.3252377
[2,] 0.7942475  0.6897546 0.3517542  0.8643681 0.5209210
[3,] 0.1413068  0.1242645 0.2403568  0.2061614 0.3158434
[4,] 0.5753208  0.4747665 0.4786111  0.4864661 0.3536892
[5,] 0.5856725  0.5202928 0.4142163  0.4310780 0.3622609
[6,] 0.6466182  0.7479228 0.7484972  0.7122268 0.8621122
     441_colon
[1,] 0.2945773
[2,] 0.8785779
[3,] 0.6959181
[4,] 0.4580922
[5,] 0.3668198
[6,] 0.8154390
```

For a simple unsupervised clustering of the samples, we can plot the results of a classical multi-dimensional scaling analysis.

```
R> cmdsplot(labcols=c("red","black","blue"), expcol="tissue",
          rawData=rawData, p=p, okqc=okqc, noXorY=TRUE,
          outfile="cmds_topN.pdf", topN=c(100000,1000))
```

```
null device
          1
```

The density plots are shown in Appendix B and the MDS plot is shown in Appendix C.

## 5.1 Pipeline 1 (recommended): Regression-based DMR-finding after correcting for batch effects

Optionally, we may wish to restrict our search for DMRs to non-control probes exceeding some quality threshold. We may do that simply by subsetting:

```
R> Index = setdiff(which(rmpmq>75),ctrlIdx)
R> Index = Index[order(chr[Index], pos[Index])]
R> p0 = p #save for pipeline 2 example
R> p = p[Index,]
R> seq = seq[Index]
R> chr = chr[Index]
R> pos = pos[Index]
R> pns = pns[Index]
R> pns = clusterMaker(chr,pos)
```

You might also wish to consider excluding some probes from the between-array normalization step in `methp` earlier using the `excludeIndex` argument, e.g., `excludeIndex=which(rmpmq<=50)`, however, note that it is probably inadvisable to remove probes from between-array normalization in `methp` that you will end up using in the analysis (note that the probes with mean qc $<$ x1 are a subset of the probes with mean qc $<$ x2 when x1=50 and x2=75 as in this example. Setting x1>x2 is not recommended as it would result in un-normalized probes being used in the analysis). Using the `clusterMaker` function was necessary in order to redefine the array regions since removing probes may result in too few probes per region or unacceptably large gaps between probes within the same region. At this point it may also be helpful to remove arrays whose average correlation with all other arrays is below some threshold, since it is often reasonable to assume that most probes are not differentially methylated between arrays. Unsupervised clustering would also probably tend to show such arrays as clustering separately. Another reason for removing arrays at this point is if they have missing data on any of the covariates to be used in the following analysis. Remember that any arrays excluded from this point forward should be removed from both `p` and `pd`.

To identify DMR candidates, we use the `dmrFind` function. As it requires the same `mod` and `mod0` arguments as the `sva()` function from the `sva` package, we must first create these. Data with paired samples may be accommodated by including the pair ID column as a factor in `mod` and `mod0`.

```
R> mod0 = matrix(1,nrow=nrow(pd),ncol=1)
R> mod  = model.matrix(~1 +factor(pd$tissue,levels=c("liver","colon","spleen")))
```

We may now call `dmrFind`. Setting the `coeff` argument to 2 means that we are interested in the colon-liver comparison, since it is the second column of `mod` that defines that comparison in the linear model.

```
R> library(corpcor)
R> thedmrs = dmrFind(p=p, mod=mod, mod0=mod0, coeff=2, pns=pns, chr=chr, pos=pos)

Running SVA
Number of significant surrogate variables is:  1
Iteration (out of 5 ):1  2  3  4  5
Regression
Obtaining estimates for  factor(pd$tissue, levels = c("liver", "colon", "spleen"))colon
Smoothing
============================================================
....

Found 188 potential DMRs
```

To compare liver and spleen, set `coeff` to 3. To compare colon and spleen, you must redefine `mod` such that the first level is either colon or spleen, and then set `coeff` appropriately, e.g., `mod = model.matrix( 1 + factor(pd$tissue, levels=c("colon","spleen","liver")))` and `coeff=2`. To avoid repeating the SVA analysis within `dmrFind`, you may provide the surrogate variables already identified above as an argument to subsequent `dmrFind` calls through the `svs` argument. The surrogate variables are located in `thedmrs$args$svs`, so adding `svs=thedmrs$args$svs` to the call to `dmrFind` would prevent SVA from being called again. As long as `p` (or `logitp`, if you provided `logitp` to `dmrFind`), `mod`, and `mod0` are the same, the surrogate variables will be the same regardless of which comparison you explore.

Also note that if you adjust for covariates, their effects will be controlled for when finding DMRs, however their effects are not removed from the matrix of "cleaned" percent methylation estimates (i.e., `cleanp`) returned by `dmrFind`, which by default removes only batch effects (i.e., the surrogate variables identified by SVA). Consequently the adjustment covariate effects will still show up in the clustering results (and will probably be enhanced) and in the DMR plots (since they do not get removed from the `cleanp` matrix). Only the surrogate variables identified by SVA will be removed from the clustering results and the DMR plots, regardless of whether or not you adjust for covariates. Setting `rob=FALSE` in dmrFinder will cause the covariates' effects to be removed from `cleanp` as well (all except the covariate of interest). `rob=TRUE` by default because covariates explicitly adjusted for should typically be real biological rather than technical confounders, and removing the effects of real biological confounders from the percent methylation estimates would change them from being our best estimate of what the true percent methylation is for each probe in our sample to an adjusted version of this.

If you want to obtain FDR q-values for the DMR candidates returned by `dmrFind`, you may use the `qval` function as follows:

```
R> withq = qval(p=p, dmr=thedmrs, numiter=3, verbose=FALSE, mc=1)

....
```

The `numiter` argument is set to 3 here only for convenience of demonstration. In reality it should be much higher (hundreds, if not thousands). The `p` argument provided must be the same as the one used in `dmrFind`. The `qval` function utilizes the parallel package that comes with R as of version 2.14. By default, `mc=1` (no parallelization), however on multiple-core machines you can set `qval` to use more cores to parallelize the process. If you are working in a shared computing environment, take care not to request more cores than are available to you.

We may plot DMR candidates from `dmrFind` using the `plotDMRs` function. In order to mark the location of CpG islands in the second panel of each plot, we must first obtain a table identifying CpG islands. CpG island definitions according to the method of Wu et al (2010) [3] are available for a large number of genomes at http://rafalab.jhsph.edu/CGI. Alternatively, CpG island definitions may be obtained from the UCSC Genome Browser.

```
R> cpg.cur = read.delim("http://rafalab.jhsph.edu/CGI/model-based-cpg-islands-hg18.txt",
                        as.is=TRUE)
R> plotDMRs(dmrs=thedmrs, Genome=Hsapiens, cpg.islands=cpg.cur, exposure=pd$tissue,
          outfile="./colon-liver.pdf", which_plot=c(1),
          which_points=c("colon","liver"), smoo="loess", ADD=3000,
          cols=c("black","red","blue"))

Making 1 figures
Plotting DMR candidate 1
```

Instead of plotting the probe p-values in the 3rd panel, you may also wish to inspect the behavior of the green channel (total) across the DMR regions. To do this, you must first have obtained the green channel intensity matrix, which we do here after spatial adjustment and background correction. In addition to specifying `panel3="G"`, we must also provide `G` and the sequences corresponding its rows (because the intensities are further corrected for gc-content).

```
R> dat0 = spatialAdjust(rawData, copy=FALSE)
R> dat0 = bgAdjust(dat0, copy=FALSE)
R> G = pm(dat0)[,,1] #from oligo
R> G = G[Index,]
R> plotDMRs(dmrs=thedmrs, Genome=Hsapiens, cpg.islands=cpg.cur, exposure=pd$tissue,
          outfile="./colon-liver2.pdf", which_plot=c(1),
          which_points=c("colon","liver"), smoo="loess", ADD=3000,
          cols=c("black","red","blue"), panel3="G", G=G, seq=seq)

......Making 1 figures
Plotting DMR candidate 1
```

### 5.1.1 Continuous covariate of interest

The `dmrFind` function also handles a continuous covariate of interest. Here we generate an artificial continuous covariate called `x` and perform the analysis using that.

```
R> pd$x = c(1,2,3,4,5,6)
R> mod0 = matrix(1,nrow=nrow(pd),ncol=1)
R> mod  = model.matrix(~1 +pd$x)
R> coeff = 2
R> thedmrs2 = dmrFind(p=p, mod=mod, mod0=mod0, coeff=coeff, pns=pns, chr=chr, pos=pos)
```

```
Running SVA
Number of significant surrogate variables is:  2
Iteration (out of 5 ):1  2  3  4  5
Regression
Obtaining estimates for  pd$x
Smoothing
============================================================
....

Found 214 potential DMRs
```

To plot the DMR results, you may either categorize the continuous covariate as for example as follows

```
R> groups = as.numeric(cut(mod[,coeff],c(-Inf,2,4,Inf))) #You can change these cutpoints.
R> pd$groups = c("low","medium","high")[groups]
R> plotDMRs(dmrs=thedmrs2, Genome=Hsapiens, cpg.islands=cpg.cur, exposure=pd$groups,
          outfile="./test.pdf", which_plot=c(1), smoo="loess", ADD=3000,
          cols=c("black","red","blue"))
```

```
Making 1 figures
Plotting DMR candidate 1
```

or you may plot the correlation of each probe with the covariate as follows:

```
R> plotDMRs(dmrs=thedmrs2, Genome=Hsapiens, cpg.islands=cpg.cur, exposure=pd$x,
          outfile="./x.pdf", which_plot=c(1), smoo="loess", ADD=3000,
          cols=c("black","red","blue"))
```

```
Making 1 figures
Plotting DMR candidate 1
```

An additional function that can be helpful for working with tables with columns "chr", "start", and "end" as many of the objects required or returned by these functions are is the `regionMatch` function, which finds for each region in one table the nearest region in another table (using the `nearest()` function in the IRanges package) and provides information on how near they are to each other.

```
R> ov = regionMatch(thedmrs$dmrs,thedmrs2$dmrs)

chr1  chr10  chr11  chr12  chr13  chr14  chr15  chr16  chr17  chr18  chr19  chr2  chr20  chr

R> head(ov)

  dist matchIndex   type amountOverlap insideDist size1
1    0          2  cover            NA         NA  1467
2    0          7  cover            NA         NA  1053
3    0          9  cover            NA         NA  1117
4    0        125  cover            NA         NA   877
5    0          1 inside            NA          0  1001
6    0         34  cover            NA         NA   873
  size2
1  1467
2   948
3   877
4   142
5  1136
6   561
```

One may also plot regions other than DMR candidates returned by `dmrFind`, using the `plotRegions` function.

```
R> mytable = thedmrs$dmrs[,c("chr","start","end")]
R> mytable[2,] = c("chr1",1,1000) #not on array
R> mytable$start = as.numeric(mytable$start)
R> mytable$end = as.numeric(mytable$end)
R> plotRegions(thetable=mytable[c(1),], cleanp=thedmrs$cleanp, chr=chr,
             pos=pos, Genome=Hsapiens, cpg.islands=cpg.cur, outfile="myregions.pdf",
             exposure=pd$tissue, exposure.continuous=FALSE)

Making 1 figures
null device
          1
```

## 5.2 Pipeline 2: DMR-finding without adjusting for batch or other covariates

We can identify differentially methylated regions using the original `dmrFinder`:

```
R> dmr <- dmrFinder(rawData, p=p0, groups=pd$tissue,
         compare=c("colon", "liver","colon", "spleen"),
         removeIf=expression(nprobes<4 | abs(diff)<.05 | abs(maxdiff)<.05))

R> names(dmr)
```

```
 [1] "tabs"    "p"       "l"       "chr"     "pos"
 [6] "pns"     "index"   "gm"      "groups"  "args"
[11] "comps"   "package"
```

```
R> names(dmr$tabs)
```

```
[1] "colon-liver"  "colon-spleen"
```

```
R> head(dmr$tabs[[1]])
```

```
       chr    start      end        p1        p2
319  chr12 88272817 88273844 0.8430144 0.1999983
358  chr13 27090247 27091263 0.7749159 0.1837313
1752  chr6 52637747 52638747 0.7076174 0.1871022
1118 chr20 60187423 60188227 0.8208041 0.2027668
473  chr15 58673117 58673819 0.8252451 0.3093024
133  chr11 14620645 14621065 0.8446964 0.3519393
                 regionName indexStart indexEnd nprobes
319  chr12:88266873-88274292      40465    40489      25
358  chr13:27090144-27095500      45272    45291      20
1752  chr6:52635302-52638967     160819   160843      25
1118 chr20:60143957-60188418     122600   122623      24
473  chr15:58669815-58674073      57658    57677      20
133  chr11:14620645-14623686      28438    28450      13
         area   ttarea      diff   maxdiff
319  16.075403 790.8966 0.6430161 0.7596896
358  11.823692 734.3303 0.5911846 0.7333419
1752 13.012878 653.7700 0.5205151 0.6574953
1118 14.832894 533.3540 0.6180372 0.8313704
473  10.318855 527.0546 0.5159428 0.6581105
133   6.405842 466.0673 0.4927571 0.6413595
```

When called without the `compare` option, `dmrFinder` performs all pairwise comparisons between the groups.

We can also plot DMR candidates with the dmrPlot function. Here we plot just the top DMR candidate from the first DMR table.

```
R> dmrPlot(dmr=dmr, which.table=1, which.plot=c(1), legend.size=1,
         all.lines=TRUE, all.points=FALSE, colors.l=c("blue","black","red"),
         colors.p=c("blue","black"), outpath=".", cpg.islands=cpg.cur, Genome=Hsapiens)
```

```
Smoothing:
============================================================
Done.


Plotting finished.
```

We can also plot any given genomic regions using this data by using the `regionPlot` function, supplying the regions in a data frame that must have columns with names "chr", "start", and "end". Naturally, regions that are not on the array will not appear in the resulting file.

```
R> mytab = data.frame(chr=as.character(dmr$tabs[[1]]$chr[1]),
                      start=as.numeric(c(dmr$tabs[[1]]$start[1])),
                      end=as.numeric(c(dmr$tabs[[1]]$end[1])), stringsAsFactors=FALSE)
R> regionPlot(tab=mytab, dmr=dmr, cpg.islands=cpg.cur, Genome=Hsapiens,
            outfile="myregions.pdf", which.plot=1:5, plot.these=c("liver","colon"),
            cl=c("blue","black"), legend.size=1, buffer=3000)

Smoothing:
============================================================
Done.
1
Plotting finished.

R>
```

The DMR plot is shown in Appendix D, and the plot of the user-provided region is shown in Appendix E.

### 5.2.1 Analysis of paired samples

If the samples are paired, we can also analyze them as such. To show this, let's pretend that the samples in our test data set are paired, and then use the `dmrFinder` function with the `"paired"` argument set to TRUE and the `"pairs"` argument specifying which samples are pairs. (In this example we also have to lower the cutoff since there are not enough samples to find any regions with the default cutoff of 0.995.)

```
R> pData(rawData)$pair = c(1,1,2,2,1,2)
R> dmr2 <- dmrFinder(rawData, p=p0, groups=pd$tissue,
         compare=c("colon", "liver","colon", "spleen"),
         removeIf=expression(nprobes<4 | abs(diff)<.05 | abs(maxdiff)<.05),
         paired=TRUE, pairs=pData(rawData)$pair, cutoff=0.95)

============================================================
```

We plot the, say, third DMR with the `dmrPlot` function (shown in Appendix F)

```
R> dmrPlot(dmr=dmr2, which.table=1, which.plot=c(3), legend.size=1, all.lines=TRUE,
         all.points=FALSE, colors.l=c("blue","black"), colors.p=c("blue","black"),
         outpath=".", cpg.islands=cpg.cur, Genome=Hsapiens)
```

```
============================================================


Plotting finished.
```

Plotting user-provided regions using the results of paired analysis is done using the `regionPlot` function as before (shown in Appendix G).

```
R> regionPlot(tab=mytab, dmr=dmr2, cpg.islands=cpg.cur, Genome=Hsapiens,
            outfile="myregions_paired.pdf", which.plot=1:5,
            plot.these=c("colon-liver"), cl=c("black"), legend.size=1, buffer=3000)

============================================================
1
 Plotting finished.
```

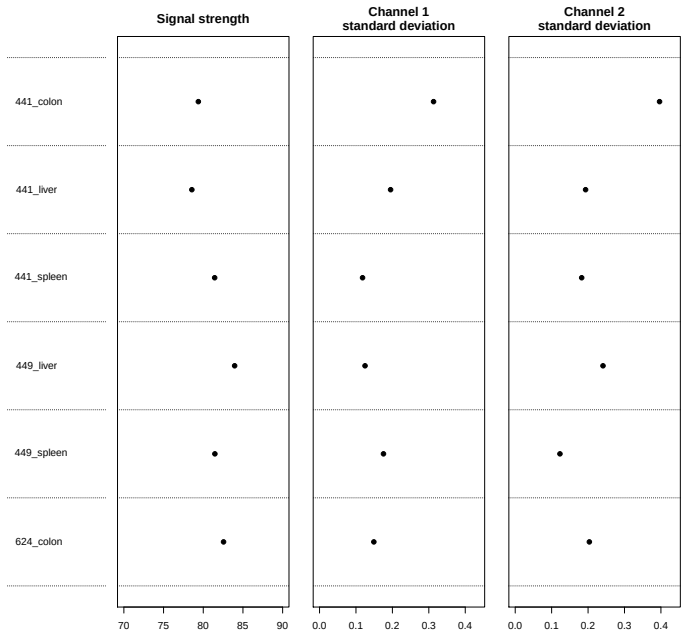
# References

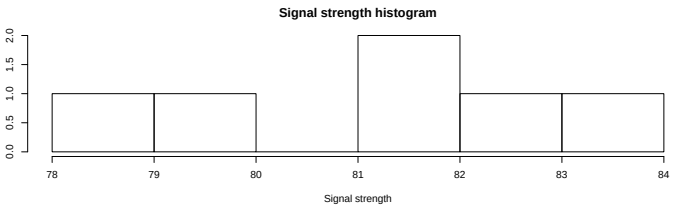

[1] Martin J. Aryee, Zhijin Wu, Christine Ladd-Acosta, Brian Herb, Andrew P. Feinberg, Srinivasan Yegnasubramanian, and Rafael A. Irizarry. Accurate genome-scale percentage dna methylation estimates from microarray data. *Biostatistics*, 12(2):197–210, 2011.

[2] Irizarry et al. Comprehensive high-throughput arrays for relative methylation (charm). *Genome Research*, 18(5):780–790, 2008.

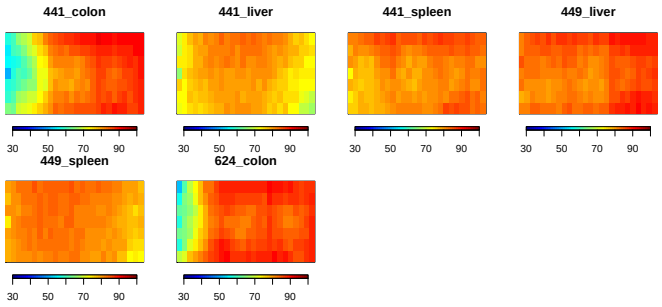
[3] Wu et al. Redefining cpg islands using a hierarchical hidden markov model. *Biostatistics*, 11(3):499–514, 2010.


# 6 Appendix A: Quality report

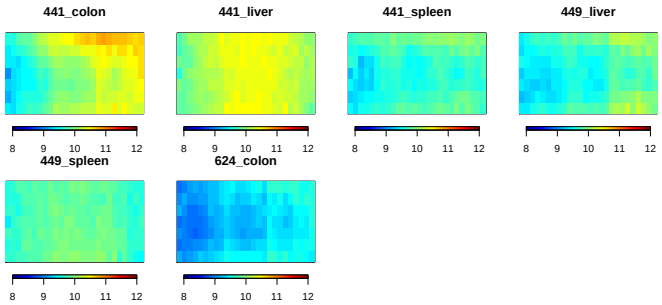

Signal strength
Channel 1
standard deviation
Channel 2
standard deviation
441_colon
441_liver
441_spleen
449_liver
449_spleen
624_colon
70
75
80
85
90
0.0
0.1
0.2
0.3
0.4
0.0
0.1
0.2
0.3
0.4


Signal strength histogram
0.0
0.5
1.0
1.5
2.0
78
79
80
81
82
83
84
Signal strength

Untreated Channel: PM probe quality

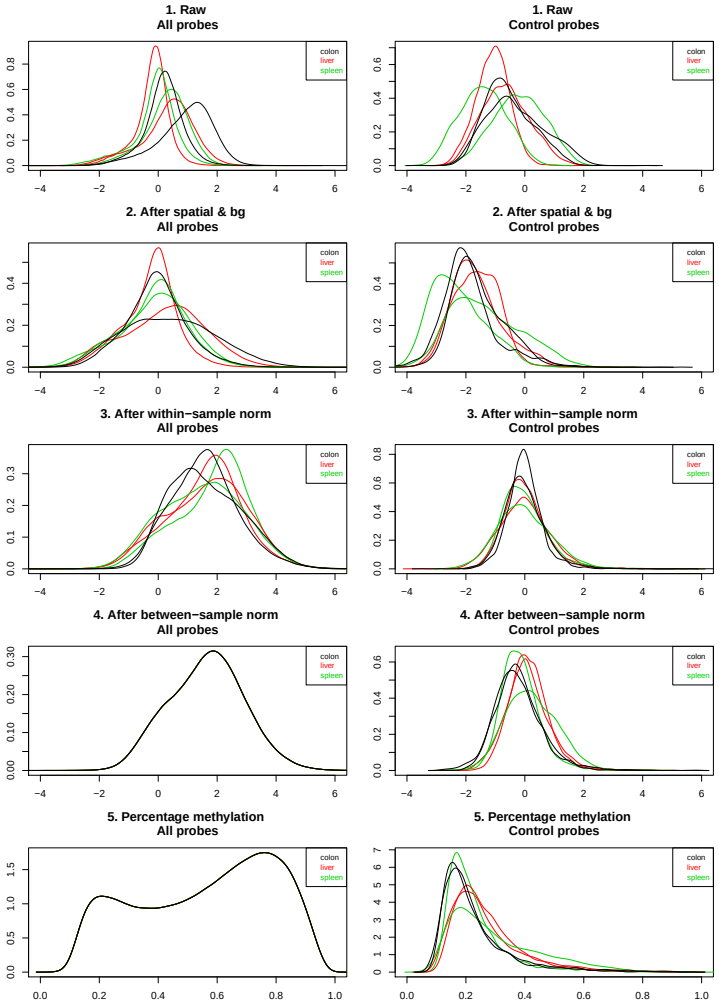


Enriched Channel: PM signal intensity

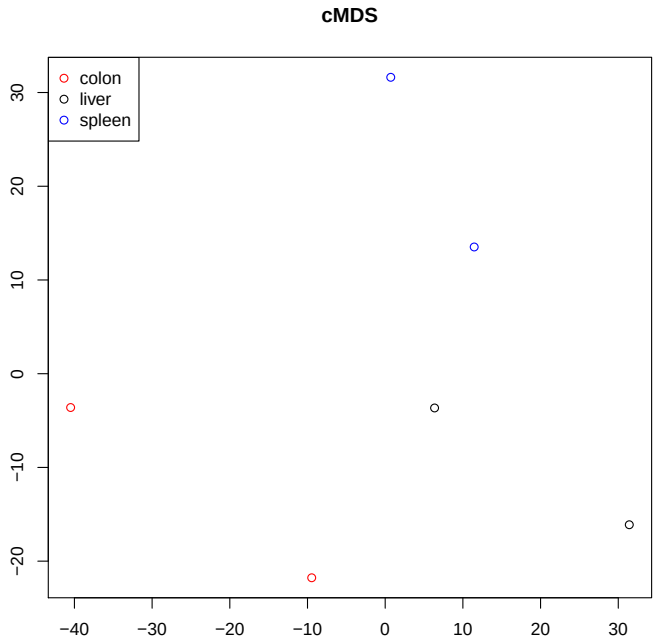


# 7 Appendix B: Density plots

Each row corresponds to one stage of the normalization process (Raw data, After spatial and background correction, after within-sample normalization, after between-sample normalization, percentage methylation estimates). The left column shows all probes, while the right column shows control probes.

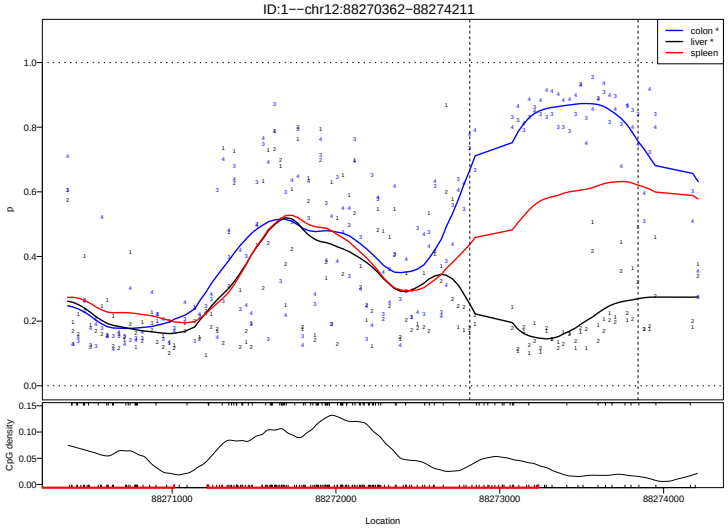

1. Raw
All probes
1. Raw
Control probes
2. After spatial & bg
All probes
2. After spatial & bg
Control probes
3. After within−sample norm
All probes
3. After within−sample norm
Control probes
4. After between−sample norm
All probes
4. After between−sample norm
Control probes
5. Percentage methylation
All probes
5. Percentage methylation
Control probes
colon
liver
spleen

# 8 Appendix C: MDS plot


cMDS
colon
liver
spleen
30
20
10
0
−10
−20
−40
−30
−20
−10
0
10
20
30
Using the 1e+05 most variable probes, out of 145217 total.
Does not use probes in sex chromosomes.

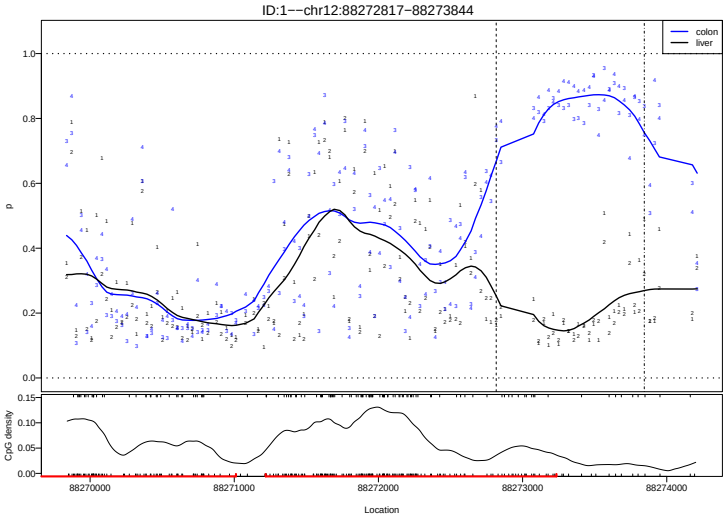
# 9 Appendix D: DMR plot

DMR plot for the first DMR in the list.


ID:1−−chr12:88270362−88274211
colon *
liver *
spleen
p
1.0
0.8
0.6
0.4
0.2
0.0
CpG density
0.15
0.10
0.05
0.00
88271000
88272000
88273000
88274000
Location

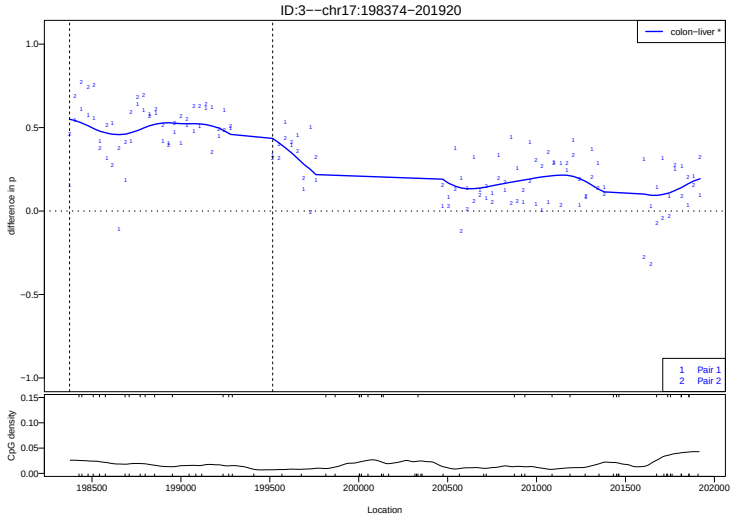
# 10 Appendix E: Plot of an arbitrary genomic region

For the arbtirary region we just chose the first DMR.


ID:1−−chr12:88272817−88273844
colon
liver
p
1.0
0.8
0.6
0.4
0.2
0.0
CpG density
0.15
0.10
0.05
0.00
88270000
88271000
88272000
88273000
88274000
Location

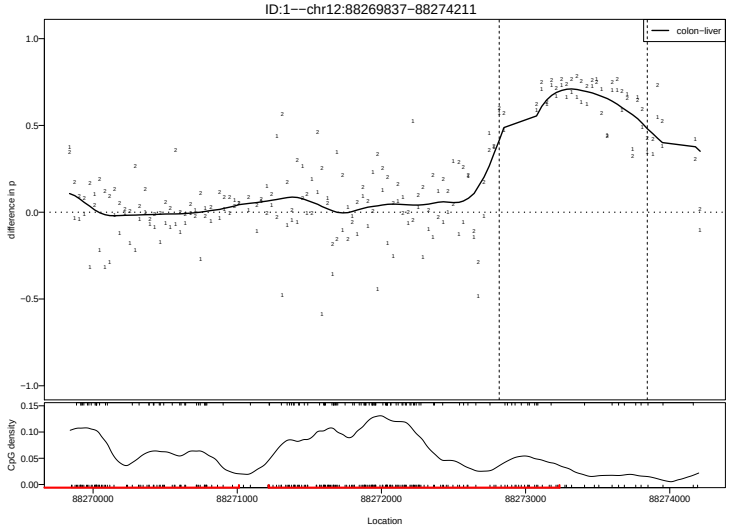
# 11 Appendix F: DMR plot from analysis of paired samples

DMR plot for the third DMR in the list


ID:3−−chr17:198374−201920
colon−liver *
1.0
0.5
0.0
−0.5
−1.0
difference in p
1 Pair 1
2 Pair 2
0.15
0.10
0.05
0.00
CpG density
198500
199000
199500
200000
200500
201000
201500
202000
Location

# 12 Appendix G: Plot of an arbitrary genomic region, shown using paired results

For the arbtirary region we simply chose the same first DMR as in appendix E.


ID:1−−chr12:88269837−88274211
colon−liver
difference in p
1.0
0.5
0.0
−0.5
−1.0
CpG density
0.15
0.10
0.05
0.00
88270000
88271000
88272000
88273000
88274000
Location

# 13 Details

This document was written using:

```
R> sessionInfo()

R version 3.0.2 (2013-09-25)
Platform: x86_64-unknown-linux-gnu (64-bit)

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

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

other attached packages:
 [1] locfit_1.5-9.1
 [2] corpcor_1.6.6
 [3] BSgenome.Hsapiens.UCSC.hg18_1.3.19
 [4] BSgenome_1.30.0
 [5] Biostrings_2.30.0
 [6] GenomicRanges_1.14.0
 [7] XVector_0.2.0
 [8] IRanges_1.20.0
 [9] charmData_0.99.8
[10] pd.charm.hg18.example_0.99.3
[11] oligo_1.26.0
[12] oligoClasses_1.24.0
[13] RSQLite_0.11.4
[14] DBI_0.2-7
[15] charm_2.8.0
[16] genefilter_1.44.0
[17] RColorBrewer_1.0-5
[18] fields_6.8
[19] maps_2.3-6
[20] spam_0.40-0
[21] SQN_1.0.5
[22] nor1mix_1.1-4
[23] mclust_4.2
[24] Biobase_2.22.0
[25] BiocGenerics_0.8.0
```

```
loaded via a namespace (and not attached):
 [1] AnnotationDbi_1.24.0  BiocInstaller_1.12.0
 [3] MASS_7.3-29           XML_3.98-1.1
 [5] affxparser_1.34.0     affyio_1.30.0
 [7] annotate_1.40.0       bit_1.1-10
 [9] codetools_0.2-8       ff_2.2-11
[11] foreach_1.4.1         gtools_3.1.0
[13] iterators_1.0.6       lattice_0.20-24
[15] limma_3.18.0          multtest_2.18.0
[17] preprocessCore_1.24.0 siggenes_1.36.0
[19] splines_3.0.2         stats4_3.0.2
[21] survival_2.37-4       sva_3.8.0
[23] tools_3.0.2           xtable_1.7-1
[25] zlibbioc_1.8.0
```