Analysis Pipeline of Oyster Developmental Methylome Dynamics by MeDIP-seq

Contributors: S Tecchio, E Crowell, G Riviere, Y He.

General Comments

\*\* All the scripts in the repository and their definition should be usable without any modification by the user. The rationale and logic behind the scripts and for parameters definition in analyses are given below as well as in in-script annotations. Some common parts between script are useless but were maintained, such as the ‘loading data’ section or other subparts that remain identical from copy/paste between related scripts. These parts were kept on purpose for more convenience (…) when running the scripts in R. Some annotations within scripts may remain in French without compromising their understanding.\*\*

Source files

**Primary and first part secondary analyses**

The raw data were produced by MeDIPseq on an Illumina GAIIx platform (SESAME – SF ICORE – University of Caen Normandy) . The primary analysis including QC were realized using RTA (Illumina proprietary) with default parameters (QC>30). The first part of the secondary analysis was done using CASAVA v1.8 (Illumina prop.) with configurebcltofastq.pl. The parameter UseBaseMask (Y76,I6,Y76) was used for demultiplexing. The script is in *Run\_GigaMedev\_support-SeSAME.txt*.

**Mapping to the oyster genome**

The genome assembly used is v.9 (Zhang et al. 2012). Reads were mapped using BWA with default parameters (bwa aln) using the *bwa.sh* bash script. From this point it appeared that for some reason one ovocyte library index gave less than half a million reads (problem of representation when combining indexes?). This sample was not kept in the final versions of downstream analyses. The output are BAM files (available at the NCBI under the project number PRJNA324546) such as *Trocophore\_1\_mapped\_converted\_BAM.bam* that were renamed into say like *troc1.bam* for more convenient use in the scripts. 22 Libraries were sequenced, with n =2 or 3 per development stages based on DNA fragments quality for MeDIP and sequencing output. See below for details

**Counting Reads in features**

This work was done using Python *HTseq* and R *HTseq-counts* and *EdgeR* packages (The whooole list of required parent packages and dependencies and libraries for Python and R has to be installed first). The feature GFF files are from Steven Robert’s lab and are based on the version 9\_90 of the oyster genome (<http://nbviewer.ipython.org/github/sr320/ipython_nb/blob/master/TJGR_OysterGenome_IGV.ipynb>).

.Additional details can be found here (<https://genefish.wikispaces.com/Methylation+Patterns+in+the+Pacific+Oyster+Genome>). Not all the genome is covered (90%), which was deemed largely sufficient for this study. Some feature files present problems and had to be modified to be manageable by *HTseq-counts*. Problems include issues with different idattr between files (parent or Target, TEs and repeats) or with start and/or end columns (ex.: exclusion of promoter values <100 pb). Although highly inaccurate because of unknown consensus sequence and somehow crappy annotation due to poor information on 5’UTRs in the v9\_90 geneome (cf the FEBS and BMC bioinfo paper studies), we have no other choice than to use the 1 kb upstream to gene starts unless characterize each and every promoter in silico and in vitro… The summary of final feature files used in the study is below:

**GEN feature** -> "Gene" type=mRNA idattr=ID Exploitable from <http://eagle.fish.washington.edu/trilobite/Crassostrea_gigas_v9_tracks/Cgigas_v9_gene.gff>

**CDS feature** -> "Exons" type=CDS idattr=Parent Exploitable from

<http://eagle.fish.washington.edu/trilobite/Crassostrea_gigas_v9_tracks/Cgigas_v9_exon.gff>

**INT feature** -> "Intron" type=intrn idattr=Parent Exploitable from

<http://eagle.fish.washington.edu/trilobite/Crassostrea_gigas_v9_tracks/Cgigas_v9_intron.gff>

**PRO feature** -> "Promoter (= 1kbp 5' of genes)" type=promoter idattr=ID Some very small interval lengths. Was modified (see below).

<http://eagle.fish.washington.edu/trilobite/Crassostrea_gigas_v9_tracks/Cgigas_v9_1k5p_gene_promoter.gff>

**REP feature** -> "Repeats" type=similarity idattr=Target. Exploitable from

<http://eagle.fish.washington.edu/cnidarian/rm_020713/oysterv9_90.fa.out.gff>

**TE feature** -> "Transposable Elements" type=Tandem\_Repeat idattr=. No attributes. Was modified (see below)

<http://eagle.fish.washington.edu/trilobite/Crassostrea_gigas_v9_tracks/Cgigas_v9_TE.gff>

Feature files were modified as follows:

The TICL script *subsetprom.tcl* was used to remove promoter sequences under 100bp (see above about 5’UTRs):

tclsh ./subsetprom.tcl ./features/promoter.gff 0 100 > ./features/promoter100bp.gff

The TICL script *addattr.tcl* was used to add attributes to the te.gff file.

tclsh ./addattr.tcl ./features/te.gff 0 ID= > ./features/teattr.gff

After preparing the feature files, the BAM files from bwa mapping were then sorted so that paired reads are on adjacent lines for HTseq to run properly. An example command is like

BAM\_sorted$ samtools sort -n -o morula1s.bam -T sorted morula1.bam

for the morula1 sample to be paired-sorted and stored in the BAM\_sorted directory as *morula1s.bam*. The script processing all the files is *sortbam.sh*.

Ready for the counts. An example command line for read counts looks like that

/usr/bin/python -m HTSeq.scripts.count --format=bam --stranded=no --type=CDS --idattr=Parent --mode=union ./BAM\_sorted/troc1.bam ./features/exon.gff > troc1\_1cds.counts

for a trochophore sample file in the BAM\_sorted directory to be counted the number of reads falling in the CDS feature in the union mode. The script processing all the files for all the features is *RunHTseq-countsDev.sh*.

After running HTseq count, warnings indicated that some reads could not be matched to their pair counterpart:

Warning: Read PC-LABO-NGS:13:FC:3:1:7260:11033 claims to have an aligned mate which could not be found in an adjacent line.

Warning: 1312 reads with missing mate encountered.

3851735 SAM alignment pairs processed.

Such warnings appear for all the files with few ‘single’ reads. How sad for the poor lonely reads that couldn’t find their paired mate partner. The proportion across samples is consistently very low (0.0003% here, <0.1% anyways) and it was decided to proceed further neglecting those reads.

The counts data are into /counts\_dev/ and then each type of files sorted into subfolders according to the feature type: /cds/, /gen/, /int/, /pro/, /rep/, and /te/.

Analysis pipelines and specific points

**DMR analyses using diffReps (15-18/09/2015 ->)**

This is a PERL script that uses a sliding window technique to scan the genome and identify regions that show read count differences. A sliding window is defined as a fix-sized genomic region where the reads falling into this region can be counted. The sliding window moves along the genome in a fixed step size that is typically a fraction of the window size. Once the differential sites are found, diffReps automatically classifies each of them into one of the following categories: Proximal Promoter, Promoter1k, Promoter3k and Genebody. Of course, in the case of the oyster genome this classification is not yet possible.

The script accepts BED files as input. We only have BAM files, so to convert them I used *bedtools* on the Linux computer. I created a bash script to do it in sequence:

for x in \*.bam; do  
 bedtools bamtobed –i “$x” > “${x%.bam}.bed”  
done

diffReps was installed by first installing cpanminus (apt-get install cpanminus), then selecting the correct http proxy (export http\_proxy=http://proxy.unicaen.fr:3128), then entering the directory where the tar.gz file was downloaded and typing *cpanm filename*. The installation worked flawlessly from there.

To launch diffReps, we also need a file containing all scaffolds in one column, and their length in the second column. The full genome was downloaded from GenBank (*Crassostrea gigas* v.9) and I wrote a quick R script to treat it into the necessary format. The script genomeTreatment.r is described in the scripts section.

diffReps works with a single command line in bash. The idea is to run it using always ovocyte data as control, and comparing it with all the other stages, which will be defined as treatments. Thus, I prepared a bash script while I was sitting comfortably in the cocoon of my office, then brought it to the Linux computer, chmodded it to executable, and launched it. Each command looked like this:

**diffReps.pl -–control** ovo1s.bed ovo2s.bed ovo3s.bed **-–treatment** 28cell1s.bed 28cell2s.bed 28cell3s.bed **-–chrlen** genomeLength.txt **–-report** out28.txt **–-nproc** 4

An issue that I encountered was the limit to the number of open files in Linux, that was too low, as diffReps opens a new file for each scaffold. We have approximately 7000 scaffolds in the oyster genome, so I tried modifying the file /etc/security/limits.conf to add the following lines at the end:

\* hard nofile 50000  
\* soft nofile 50000  
root hard nofile 50000  
root soft nofile 50000

This was still not enough because of the hard limit on inodes, which is not modifiable. The solution was to select a subset of scaffolds that represent a considerate amount of the total genome. So, I added a second part to the genomeTreatment script to calculate the cumulative sum of length of all scaffolds, starting with the longest one. We chose to keep the first 2000 scaffolds, representing 93.4% of the total genome, and hoping that a reduction from 7658 to 2000 would be enough for diffReps to work: this selection was saved to genomeLength2000.txt.

This selection had to be performed on all BED files, too. An R script was written for the job, scaffolda.r, which took quite some time to run (2 days approximately). After it, diffReps was launched and worked correctly.

Which comparisons should be considered? The first idea was to compare all stages against ovocyte. The second idea was to compare the stages “2-8 cells”, “intermediate”, and “spat”, all against ovocyte. The “intermediate” grouping was chosen because five stages showed only small differences in methylation levels between them in the smear plot. The third idea, which I think will be the final one, is to compare each one of these stages with the previous one in the temporal order:

C:\Users\Samuele\Documents\Dropbox\Huitres\Dev experiment\Paper figures\DMR conceptual scheme.tif

**Treatment of diffReps results (5-9/10/2015 ->)**

The whole annotated genome was built into a big table, starting from the GFF files downloaded from the internet, and saved as an R-data file, *wholeGenome.data*.

A list object was created merging all interesting scaffolds and their relative DMRs identified by diffReps. The script for this purpose was *prepareBigList.r*. Only scaffolds previously selected in the *genomeLength2000.txt* file were appearing. The structure of the object is as follows (will be modified in the future -> was modified : two extra colums were added at the $map object : Distance to the nearest DMR LogFC of the nearest DMR at each development step and (“FoldA28c” (Cleavage), “Fold, and Stage . The new object is the *matrixReloaded.data* R object):

$scaffold1714: one list object for each scaffold  
..$map  
.. ..$Feature: factor, “CDS”, “mRNA”, “intron”, “promoter”, or a series of repetition or transposon variants  
.. ..$Start and $Stop: integer positions in the scaffold  
.. ..$Strand: factor, + or – depending on the direction of transcription  
.. ..$Annot: gene identifier  
.. ..$Expr: expression levels of the gene  
.. ..$TranStart: reference point of transcription, chosen by looking at the strand direction  
.. ..$DistToDMRcentre: minimum value among distances between transcription point and the centre of DMRs  
.. ..$DistToDMRboundary: min. value among distances between transcription point and the first DMR position

.. ..$DistA28c: distance between feature Transstart and nearest DMR centre at the C step

.. ..$FoldA28c: DMR methylation fold change (from DiffReps) at the C step

.. ..$DistBInt: distance between feature Transstart and nearest DMR centre at the I step

.. ..$FoldBInt: DMR methylation fold change (from DiffReps) at the I step

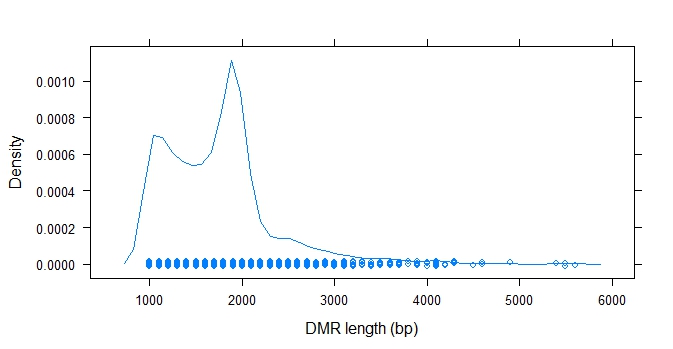
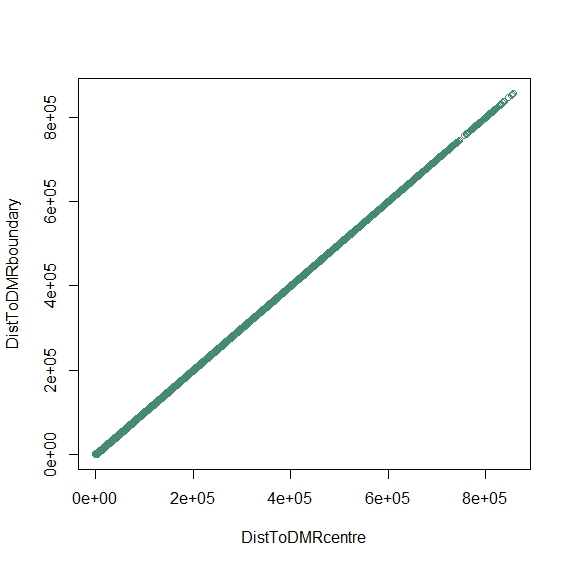
.. ..$DistCSpa: distance between feature Transstart and nearest DMR centre at the M step

.. ..$FoldCSpa: DMR methylation fold change (from DiffReps) at the M step

.. $dmr  
.. ..$Start and $Stop: integer positions in the scaffold  
.. ..$Length: region length  
.. ..$logFC: log (base 2) fold change  
.. ..$Stage: factor, indicating on which diffReps comparison the DMR was identified

After calculating the two distances from DMR, I have performed a quick comparison of the two. The result is that they are exactly the same (figure below, right), probably due to the different orders of magnitude between the distances and the size of the DMR themselves (figure below, left). Thus, we chose to keep only the centre measure to reduce the size of the object. In the first table, the two distances columns were substituted by three columns, representing the minimum distance to DMR centres, for the three diffReps comparisons.

We also needed to filter out a list of genes from the expression table: the maternal mRNA that is present in the ovocyte and decreased progressively its expression levels during development. I wrote a quick script to select those genes that only decrease during development from ovocyte to spat, and eventually clean them out from the table (*filter.mother.genes.r*). Only 94 genes over 27900 were found that satisfied the check, thus we decided to keep them.



Analyses that were conducted after this cleaning:

1. Distance to DMR by mRNA, Repetitions, and TEs: for all features, the general pattern is a slight increase in distance to DMR up to the intermediate development stages, and a strong decrease at the spat stage. This can be also explained by the fact that at spat stage, the number of DMR was by far the highest.
2. Distance to DMR boxplots by comparison, using features as factor: the DMRdistances script was modified to extract distance data by diffReps comparison, and plot a publishable figure.
3. Comparison of logFC of DMR by stage / logFC of gene expression: this analysis was first deemed without sense, as there was no way of creating data pairs by associating a DMR to a gene. Then, a possible solution was found: relate each gene with its closest DMR. The big genome object was integrated by adding, next to each gene, the logFC of its nearest DMR.  
   Once plotted, it became clear that there were interesting differences of the distribution of points relative to the position above/below and left/right their relative zero reference. For this reason, I counted the points in each quadrant and prepared a contingency table, then applied a classical χ2 test.
4. Distance to DMR / logFC of gene expression: three scripts were made to extract the required data from the genome object, for each of the three comparisons: DMRtreatmentA, B, C. After plotting, we chose an arbitrary limit of 100,000 bp as a discriminating factor to perform a statistical comparison of variances.
5. Length of DMR by comparison: the distributions of DMR length for each of the three comparisons were plotted with histograms (500 bp length each) and contrasted with Kolmogorov-Smirnov tests, which identified only a significant difference between 2-8 cells and spat stages. Can this difference explain the concept by which DMR get wider and more distributed during development?
6. Annotations of DMR: an iterative script went through each DMR and extracted all features in the same scaffold whose start and stop positions were included between the start and stop positions of the DMR. This means that we considered only those features entirely included within a DMR. Again, the reduced number of DMR at the intermediate stage did not permit a good statistic validity of the results, so we had the idea of focusing only on the 2-8 cells and spat for this analysis.

**Further selection of interesting genes (23/10/2015 ->)**

To perform an additional selection of genes, in the hope of cleaning the diffReps results, I will prepare vectors of those genes that change their methylation and expression in a significant way. The concept is to use the test already applied by the smear plot function, i.e. those points marked in red in the plots. The vector sigtags was already prepared, I just had to transfer it on an object for the three comparisons. At the 2-8 cells vs. ovocyte comparison, 3794 genes were selected; at the intermediate comparison, only 15 genes; at the spat stage, 8907 genes. Joining all these lists together, and keeping only the unique values, we obtained a vector of 9873 genes that significantly change their methylation and expression.  
This vector (*significantGenes.data*) was used to re-select the data to plot the MDS-BCV, the boxplots, and the gene variants. In any case, all results were somewhat “washed up” by this selection, so we decided to not continue on this road.

**Gene variants (1/10/2015 -> )**

A simple table with gene IDs and their respective number of transcript variants was provided. This dataset was filtered excluding all genes with only one variant, and all genes with more than 40 variants (which excessively skewed the distributions). The methylation dataset for each feature was then loaded, using a script similar to the first part of enrich-methyl, which returns a vector with the coefficients of variation of methylation considering all development stages. The names of this vector are the gene IDs. This vector is then filtered keeping only the gene IDs that are present in the filtered variants table. Each feature is then plotted with a Gaussian-family smoothing function.  
It seems that there are no evident correlations between methylation levels and number of transcript variants, so this part was scrapped.

**Correlation between methylation and expression (12/11/2015 ->)**

We have a list of methylated genes (all genes present in the countsCDSfiltered table), and a list of all genes (retrievable from the expression table). The difference is logically the list of genes for which no methylation was detected. I extracted these two lists, and I calculated the mean expression level for each gene, across all development stages. I used ggplot to quickly plot a comparison boxplot between the two conditions. The result was that non-methylated genes had an extremely lower expression than methylated genes (Cliff’s delta = 0.565, effect estimated large *sensu* Romano, 2006).  
This comparison has been included in the figure regarding the functional differences in methylation.

**Clustering of genes according to their methylation profile (17-20/11/2015 -> )**

First, Guillaume gave me a table of methylation values, normalized with the “log-centred-reduced” law, which is the equivalent of a logarithm transformation followed by a normalisation of the values by row (gene). Then, I wrote a script called “clusters.r” to run the usual ANOVA by gene, and run a k-means partitioning. The lines plots that describe the temporal pattern of methylation are then plotted using only one point per stage. I then extracted the list of genes divided by cluster. This list has been saved as an R object in *GenesWithClusters.data*. Finally, I also saved the results of the K-means, to be able to use always the same order of clusters.

****Definition of scripts

The three colour scales used in the presentation of results were kept consistent throughout the figures. When features and stages were used at the same time, a combination of colours was obtained with an HSV transition.

To obtain a matrix of the colours for features, nuanced into eight brightness levels, the following code was used. It produces a matrix of 5 x 8, with an hexadecimal colour at each cell.

oricols <- c("aquamarine3","cadetblue3","chartreuse3","darkorchid3","deeppink3")

rgbs <- col2rgb(oricols)

hsvs <- rgb2hsv(rgbs)

valueseq <- seq(0.1, 0.9, length.out = 8)

HSV <- data.frame(h = rep(hsvs[1,], each = 8),

s = rep(hsvs[2,], each = 8),

v = rep(valueseq, 5))

HSV <- t(HSV)

colvec <- character(40)

for (i in 1:40) colvec[i] <- hsv(HSV[1,i], HSV[2,i], HSV[3,i])

colmat <- matrix(nrow = 8, ncol = 5, byrow = FALSE, data = colvec)

rownames(colmat) <- stages

colnames(colmat) <- c("CDS","INT","PRO","REP","TE")

**enrich-methyl**

*Note: this script is now obsolete due to its error in selecting the correct gene universe, and also because the GO analysis is now performed by clusters and not by stages.*Reads count data for a defined feature and GO mappings given by Guillaume. Runs ANOVA by row (factor as “stage”) to exclude all genes without differences between stages. Calculates the mean gene counts by development stage. Then, finds the 5 most enriched GO for each stage in each of the three ontologies (BP, MF, CC) and calculates the mean score of the related genes. The resulting matrix “enrich-methyl.CDS.data” is like this:

stage BP BP\_Score MF MF\_Score CC CC\_Score

1 ovo GO:0007155 3.000000 GO:0003824 26.97959 GO:0019012 2.000000

2 ovo GO:0022610 3.000000 GO:0005509 19.08696 GO:0019028 2.000000

3 ovo GO:0006950 14.800000 GO:0016787 27.43860 GO:0044423 2.000000

4 ovo GO:0006720 17.500000 GO:0043167 34.39908 GO:0005737 18.062500

5 ovo GO:0008299 17.500000 GO:0043169 33.68072 GO:0012505 19.900000

6 X28cell GO:0007155 9.444444 GO:0003824 46.39909 GO:0019012 5.666667

The second part of the same script is a loop for each ontology (j in 1:3). It creates another matrix with the number of genes in common for each GO that has at least one gene in common between every possible combination of stages:

s1 s2 GO common

1 blastula dlarv GO:0019012 4

2 blastula dlarv GO:0019028 4

3 blastula dlarv GO:0044423 4

4 blastula dlarv GO:0005759 1

5 blastula gastrula GO:0019012 9

Nice parts here include the use of *combn()* to generate a list of all possible combination of stages, and a loop to generate two vectors named *firstname* and *secondname*, that will be used to search the correct column in the gene universe for each iteration.

**enrich-methyl-clusters**

This script reads the full gene universe, the GO mappings, and the K-means clustering saved into K-means.data: this way we will always have the same five clusters without reordering. It runs the enrichment test using Fisher’s method Then, instead of selecting the five most enriched terms, it extracts *all* terms and keeps everything with a significant p-value of the test. The calculation of the genes in common was much simplified by this method.

**circular-plot**

This script is driven by two parameters, feature = “CDS” and GOn = “BP”, for example. It reads the corresponding files “enrich-methyl” and “genes-in-common” and creates the circular plot. In the beginning it was independent and the user had to manually change the parameters for each run; now it is better to use the following script to launch it.  
Pro tip: use *switch()* to quickly improve the labels (e.g. "X28cell" = "2-8 cells") without having to modify the original data matrix, and still be able to perform analyses using the original stage labels.  
*The logic behind this plot:*Each circular plot refers to a feature and a gene ontology (BP, MF, or CC). The related enrich-methyl table is read, only the interesting GO columns are selected, and a new column is created with a repeating sequence 1:5. The variables *initmat* and *fa* are created to more easily initialise the circular plot, which requires a factor to separate the space into sectors, and minimum and maximum values to use as x-limits. Every time *trackPlotRegion* is called, a new track is created inside the other. The first call is only used to print the names of each development stage, using *get.cell.meta.data* to retrieve the name of each stage.  
Next, a boxed rectangle is plotted inside, with vertical limits as 0 and the maximum methylation score for the dataset. Inside it, a barplot is added using *circos.rect* and the related GO names are added with *circos.text*. The colours of the bars are retrieved by rescaling the methylation values from 0 to 1, and feeding them to a colour palette function.  
For the links, remember that each bar on the plot was positioned in 1:5 sequence on the horizontal axis. The links must therefore start from each bar’s midpoint (e.g. 1.5 for the first bar) and have a width between 0 and 0.5. For this reason, the table with the genes in common is rescaled between 0 and 0.5. A vector for colour transparency is also added with the same method, i.e. rescaling the values to a 30-200 scale. The starting position of a link is found by looking at which number in the sequence 1:5 of the enrich-methyl table has the same stage and GO code of the corresponding line in the genes-in-common table. The same is done for the end position. The links are then plotted using *circos.link*.  
The final version:  
The script circular-plot-unique will not require any change by the user, apart from having the two input files available: enrich-methyl-CLUSTERS.data and genes\_common\_clusters.data. These two files are generated by the enrich-methyl-clusters script. This new version plots one single circular diagram with five sectors corresponding to the five different clusters identified by K-means. The links indicate the number of GO terms in common, with their width proportional to the maximum number of GO terms in each cluster (thus width can change between the start and the end of each link).

**circular-launch**

This script is a wrapper to easily launch three circular plots in sequence, usually for the three ontologies in a feature. It also saves each plot as a square image in a PDF for later use. The user has to change the *feature* parameter before launching the entire script.

**heatmap (and all variations)**

This script first read the “enrich-methyl” data for one feature, containing the most enriched GOs for each development stage. The data is linearized into a longer matrix and this is then fed into *xtabs()* to create a contingency table, like this:

stage

GO Ovocyte 2-8 cells Morula Blastula Gastrula Trocophore D-larva Spat

GO:0000166 0.00000 50.538462 0.000000 0.000000 0.000000 0.00000 0.000000 0.000000

GO:0003824 26.97959 46.399093 63.598639 37.581633 76.204082 95.77891 56.009070 102.603175

GO:0005509 19.08696 42.028986 49.021739 30.673913 58.014493 73.02174 41.695652 70.681159

GO:0005635 0.00000 0.000000 0.000000 0.000000 47.333333 0.00000 0.000000 0.000000

GO:0005737 18.06250 29.208333 47.187500 28.187500 0.000000 0.00000 0.000000 57.166667

The GOs are translated into their complete names, using the function *Term()* on the row names. The heat map is plotted using the more customizable function *heatmap.2()*  in the package *gplots*. An important parameter in the heatmap.2 function is *margins*, which is more or less equivalent to a par(mar).

**smear-plot and newSmearPlot**

This script first launches (with *source*) the related edgeR\_dev\_*feature*\_loop.r to build the extest list from the pair-wise comparisons. Then, using a complex positioning system that I can only explain when full of rosé wine, each item of the extest list is plotted with plotSmear in its correct position, considering a matrix of comparisons and iteratively reading the extest[[i]]$comparison variable. Instead of using *layout()* like any normal person, I decided to use *split.screen()* to be able then to use *screen()* to navigate through the plotting window. Imagine a 9x9 matrix with numerical positions assigned by row: the variable *seqrow* is therefore a vector of length 8, containing the numbers positioned on the first column, i.e. 1-10-19-28… Then, I look up the names of the two stages compared in each iteration, and position their plot accordingly.  
This method also allowed me to build a nice “legend” in one of the squares left blank, and to insert stage names on the marginal squares.

With the paper publication closing in, I updated the script into newSmearPlot, which plots both CDS and TE in the same window, using the colour code for features to indicate the genes that are significant in the test.

**mdsbcv-plots**

This script does not need any parameter change before launch. It sources the edgeR scripts, excluding the lengthy pairwise comparisons, and plots the MDS by development stage, using the “biological coefficient of variation” as distance measure. The “final” version of this script only plots TE and CDS and uses the colour scale combination for each element.

**genomeTreatment**

This script reads the raw genome data downloaded from GenBank. It extracts only the two interesting columns (scaffold name and length), removes the last row (non-numeric), and writes the output into genomeLength.txt, which now has the correct format for diffReps.  
The second part orders the scaffolds by their length, the longest one first. Then adds another column with the cumulative sums of length, then a fourth one calculating the cumulative proportion of the genome that is described by the first n scaffolds. It writes the result on a second text file. This second part was used to select the first 2000 scaffolds for the genomeLength2000.txt file.

**logcentrereduced**

This is a quick script that normalises a matrix by row. It applies a logarithm transformation to the data, in our case log2(x+1), then it calculates mean and standard deviation by row. Finally it normalises each row by its mean and deviation. It has been launched for all methylation tables, and the results have been saved into *LCRcountsCDS.csv*, for example.

Figures for publication

Three big figures were planned. Two of them were completely written in R, to be able to directly save them on PDF and then into a high-resolution TIFF image. Another one had a too complex layout to code into R (due to the use of *screen()* for the smear plots, that impairs the use of layout) and so each component was composed into the final figure using Photoshop.

**Figure 1 – Methylation profiles**

The plots were generated separately and then manually composed in post-production. The scripts to obtain the various components are all located in the “Dev experiment” directory:

* *Figure 1 barplots only.r*, for the vertical line of five barplots;
* *newSmearPlot.r* for the square set of methylation smear plots for CDS and TE;
* *mdsbcv-plots-final.r*, for the two MDS plots for CDS and TE;
* *3dplots-launch.r*, that launches the three 3-dimensional plots in sequence.

**Figure 2 – Functional profiles**

This completely scripted plot has two versions, of which the second one is the final one that was deemed publishable. The script is located in the “Dev experiment” directory. For correct plotting, it is better to first launch the first part up to the hash line, then re-size the width of the plotting window, then launch the rest of the script. Apart from some additional annotations, the scripts that are launched are:

* *expressionVSmethNONmeth.r*, for the first bar plot of expression versus being methylated or not;
* *boxplot launch only2.r*, located in the subfolder “BoxplotData”, for the two boxplots using deciles;
* *clusters.r*, for the five plots of methylation profiles by stage;
* *circular-plot-unique.r*, for the circular plot of gene ontologies by clusters
* *selectedGeneOntologies.r*, for the last three plots at the bottom, about the enrichment and methylation of the selected interesting ontologies.

**Figure 3 – Differentially-Methylated Regions**

This plot is also completely scripted and it also opens a window of the correct size, thus the user only has to launch the full script and watch the magic unfold. The script is located in the “DMRs” subfolder, and the R console needs to be set there as a working directory. In order, it launches:

* A rasterising routine to read the conceptual scheme and plot it as an image;
* *DMRannotation–plot.r* for the features touched by a DMR;
* *DMRdistances.r* for the boxplot showing the distances of features by their nearest DMR;
* *DMRlength.r* for the bar plots of frequencies of DMR length;
* *DMRtreatmentA,B,C.r* for the smoothed scatter plots of distance to DMR versus gene expression;
* *DMRlogvsEXPRlog.r* for the DMR methylation versus gene expression plots. This last script also adds the capital letters for the plots on the previous row, because I am so fucking lazy that the less I have to use Photoshop, the better.