Analysis Pipeline of Oyster Developmental Methylome Dynamics by MeDIP-seq

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

General Comments

All the scripts in this repository should be usable without significant modification. The rationale and logic behind the scripts and for most parameters definition in analyses are given below and/or in in-script annotations. Some common parts between script were maintained in the code on purpose for more convenience when running the scripts - and changing parameters and gene subsets- such as the ‘loading data’ section or other subparts like aov() or lm() analyses. Some annotations in French within scripts may remain but should not compromise their understanding. Scripts didn’t have their code cleaned or optimized, and minor errors may remain. Not all the data files related to – or generated by - the scripts are extensively described but are explicitly labelled. This repository is related to the methods section in the corresponding manuscript, but it was not formally formatted nor referenced, and it should not be considered publishable as it.

\*\*\*\* PLEASE CITE RIVIERE ET AL. WHEN USING SPECIFIC CODE PARTS YOU FIND HERE \*\*\*

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, France) . 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\_90 (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 *troc1.bam* for more convenient use in the scripts. Of the 24 initial samples, 20 sequenced libraries were retained for analyses, with n =2 or 3 per development stages (biological replicates). Indeed some original replicates were not processed based on lower DNA fragments length cocnsistency for MeDIP (2 samples), library QC (1 sample) or sequencing output (1 sample). 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 (with some modifications, see below) and are based on v.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% is), 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 inconsistent 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 genome assembly (cf the FEBS and BMC bioinfo paper studies), we have no other choice than to use the 1 kb upstream to gene starts as promoters (otherwise we have to 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>

**OTH feature** -> "Other" 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*.

Read counts. The HTseq counts parameters were set as follows:

-f format option set as bam.

-r order option set as default (name).

-s stranded option as no. That means that for paired-end reads, the first read has to be on the same strand as the feature and the second read on the opposite strand.

-a minimum quality left set as default (10).

-t feature type set to the value in the third column of the GFF file.

-i The feature id (default gene\_id). (ex pro\_id for promoters).

-m Union mode: sums counts at positions where multiple reads overlap (is the most appropriate and the default value).

-o Output file name. This can also be set using redirect (>).

-q Suppress warning and progress reports.

-h Show user manual (like man in Unix).

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, far<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/. Each of these files contains five rows at the end of the file with the following information:

\_\_no\_feature: reads (or read pairs) which could not be assigned to any feature in the GFF file. These correspond to reads in other features than what is in the GFF file.

\_\_ambiguous: reads (or read pairs) which could have been assigned to more than one feature and hence were not counted for any of these.

\_\_too\_low\_aQual: reads (or read pairs) which were skipped due to the -a option. Skip all reads with alignment quality lower than the given minimum value (default: 10 — Note: the default used to be 0 until version 0.5.4.). This default value of 10 is the most stringent and reflects problems with mapping, not sequencing quality.

\_\_not\_aligned: reads (or read pairs) in the SAM file without alignment. Not aligned to the reference genome.

\_\_alignment\_not\_unique: reads (or read pairs) with more than one reported alignment. These reads are recognized from the NH optional SAM field tag. (If the aligner does not set this field, multiply aligned reads will be counted multiple times, unless they getv filtered out by due to the -a option.) This is always zero in our case indicating removal of ambiguous reads.

**Processing count results with EdgeR and generating data tables for further analyses**

At this step, the distribution of the data was checked with *counts\_dev\_\*feature\*.r* (ex. *counts\_dev\_cds.r*)*,* which removes the last five rows of the *counts\_dev* files (containing not relevant information for edger see above). Differences in the distibiution of counts within features between samples were tested using Kolmogorov-Smirnoff tests. Results were plotted with xminp <- 0 because of log transformation retuning –Inf causing an error. The distributions of reads within features appear dynamic and specific for all features between development stages, which nicely cluster together.

Next scripts were written following the protocol in Anders et al 2013 Nature Protocols to use edger: *edgeR\_dev\_\*feature\*.R* (such as *edgeR\_dev\_cds.R)*.Data are loaded using

listfiles <- list.files()

data = readDGE(listfiles)

This reads all individual count files and merges them together into a single R object, with rows listing all the gene IDs and columns listing all the samples and the count number.

This is called a DGEList object, and it is composed of:

- A data.frame called samples, containing library size information etc.

- A matrix called counts, containing the counts for each gene ID for each sample.

> dim(data)

[1] 28031 21

> head(data$samples)

files group lib.size norm.factors

28cell1cds 28cell1cds.counts 1 4678399 1

28cell2cds 28cell2cds.counts 1 3667137 1

28cell3cds 28cell3cds.counts 1 5854773 1

blastula2cds blastula2cds.counts 1 2274132 1

blastula3cds blastula3cds.counts 1 2336354 1

dlarv1cds dlarv1cds.counts 1 3426396 1

> head(data$counts)

28cell1cds 28cell2cds 28cell3cds blastula2cds blastula3cds

CGI\_10000002 1 1 0 0 0

CGI\_10000003 0 0 0 0 0

CGI\_10000004 0 0 0 2 0

CGI\_10000005 0 0 1 0 0

CGI\_10000009 2 4 4 0 1

CGI\_10000010 0 0 0 0 0

dlarv1cds dlarv2cds dlarv3cds gastrula1cds gastrula2cds

CGI\_10000002 0 0 0 0 0

CGI\_10000003 0 0 0 0 0

CGI\_10000004 0 1 0 0 1

CGI\_10000005 0 0 0 0 1

CGI\_10000009 0 0 0 0 0

CGI\_10000010 0 0 0 0 0

gastrula3cds morula1cds morula2cds ovo1cds ovo2cds ovo3cds

CGI\_10000002 0 0 0 0 0 0

CGI\_10000003 0 0 0 0 1 0

CGI\_10000004 0 0 0 2 1 0

CGI\_10000005 0 0 0 0 0 1

CGI\_10000009 0 0 0 1 2 0

CGI\_10000010 0 0 0 0 0 0

spat1cds spat2cds spat3cds troc1cds troc2cds

CGI\_10000002 0 0 0 0 0

CGI\_10000003 0 0 0 0 0

CGI\_10000004 0 0 0 0 1

CGI\_10000005 1 1 0 0 0

CGI\_10000009 2 1 0 1 0

CGI\_10000010 0 0 0 0 0

The \_\_no\_feature, \_\_ambiguous etc data are tagged on the end in the DGEList object and they ought to be removed. Feature elements were filtered out (i.e. considered unmethylated) if having very low counts across all samples. There are at least two datasets (1 replicate) representing each condition (development stage). If replicates are comparable, and the counts for a feature are not high in one sample just because of technical artifacts, counts for each feature of interest should be found in at least two datasets. Therefore, features displaying at least one count per million read in at least two replicates of the same development stage (whatever the stage) were considered methylated. The following code was used to filter out those noninformative features, and at the same time, remove the "\_\_no\_feature" etc lines. This command gives a logical vector containing FALSE or TRUE for each position in rownames. If the value of rownames is "no\_feature" or "ambiguous" or one of the other tags, the value of noint (‘not interesting’) is set to TRUE to be removed afterwards. If it is a regular geneID, the value of noint is set to FALSE.

noint = rownames(data) %in% c("\_\_no\_feature","\_\_ambiguous","\_\_too\_low\_aQual","\_\_not\_aligned","\_\_alignment\_not\_unique")

> table(noint)

noint

FALSE TRUE

28026 5

> tail(noint)

[1] FALSE TRUE TRUE TRUE TRUE TRUE

Next, counts per million were calculated and stored in cpmd using the *cpm()* function included in edgeR:

cpmd = cpm(data). Finally, the values retained after filtering were kept if counts per million is greater than one for at least two of the samples of the same development stage and the value in noint is not TRUE (i.e. is FALSE). This stores in keep all the filtered rows except those containing the \_\_no\_feature data etc.

keep = rowSums(cpmd > 1) >=2 & !noint

data = data[keep,]

The samples were then classified in groups for normalisation between libraries (samples) using the EdgeR built-in *calcNormFactors ()* function. To do this, a targets file called *counts\_dev/targets.txt*. was used to list the of counts files in column 1, their group in column 2, and their label in column 3.

$samples

group lib.size norm.factors

28cell1cds 28cell 934086 1

28cell2cds 28cell 686912 1

28cell3cds 28cell 1288520 1

blastula2cds blastula 590792 1

blastula3cds blastula 644476 1

16 more rows ...

After running the script, the scaling factors for the library sizes from *calcNormFactors()* were not very close to 1 (ie. libraries similar in composition): the lowest is 0.6 for 28cell3, but others are around 0.7 and 1.1. Therefore the data were visualized using plotMDS(data)(*edgeR\_dev\_cds\_MDS.pdf)*. When libraries are not similar in composition, it is best to use the BCV (‘biological coefficient of variation ‘) method in plotMDS because of the large discrepancies between library sizes, and the lack of an abundance trend in the dispersions." BCV means. This command is rather slow to execute.

The result is extremely different using the BCV method :

> plotMDS(data, col=c("tomato","tomato","tomato","green","green","darkorchid","darkorchid","darkorchid","cyan","cyan","cyan","orange","orange","red","red","red","violet","violet","violet",

"blue","blue"), method="bcv")

the replicates of the same developmental stage cluster together very nicely.

This script was duplicated to process all the features in *edgeR\_dev\_cds.R, edgeR\_dev\_gen.R*, etc.

The developmental stages cluster together pretty nicely no matter what genome feature is examined.

The filtered and normalized *cpmd* (counts per million for each feature ID) were added for identical feature ID (eg, gene), pooled and stored as tables with the feature ID as rows and sample as columns in the *counts\*feature\*filtered.txt* files (ex. *countsCDSfiltered.txt*). Those files were used as a starting point source files for further analyses to avoid time-consuming repeated re-processing of raw data during analysis steps.

Analysis pipelines

The oyster developmental methylomes and their dynamics were analyzed from two distinct, but parallel, points of view:

1. large scale , physical methylation to understand wide methylome landscapes -> DMRs and maps
2. feature-related patterns for a more functional approach -> transcript variants …

Both were examined for relation with gene expression and gene ontology and gave consistent results.

**LARGE SCALE LANDSCAPES**

**DMR analyses using diffReps**

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 they were converted using *bedtools* with this 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, then entering the directory where the tar.gz file was downloaded and typing *cpanm filename*.

To launch diffReps, you want 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 a quick R script was written to treat it into the necessary format (*genomeTreatment.r* ) 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. A bash script was chmodded to executable, and launched. 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

\*\*\*\* WARNING \*\*\*\* There is an issue with the limit to the number of open files in Linux, that is too low, as diffReps opens a new file for each scaffold. To try and compute the ~ 7000 scaffolds in the oyster genome, the file /etc/security/limits.conf has to be modified adding the following lines at the end:

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

This is 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, a second part was added 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, so the reduction from 7658 to 2000 allows 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 takes quite some time to run (2 days approximately with a quad core processor and 32 GB RAM).

First, all stages were compared against ovocyte. Second, comparisons were run between 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. Finally, comparisons between each one of these stages with the previous one in the temporal order were retained:

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

**Physical methylation maps**

\*\* WARNING: Processing requires ~4.2 Gb diskspace per million bases. The maps of the three longest scaffolds were compared and found extremely consistent across development stages therefore only the Scaffold 22 (longest) map is presented in the manuscript. Below is the script *methylation maps.r* exampled for Scaffold 22. \*\*

A reduced BED file for each sample was generated containing only the selected scaffold\*number\* counts *(\*sample\*\_Scaffold\*number\*.bed*). Then, files containing count information for each base were generated using BEDTools Genome coverage. Files are renamed *\*sample\*\_covsca22.txt* and stored in a *covsca22files* folder. Then the count values were normalized by million reads per sample, and average by stage (Ovo, 2-8 cells, Interm. Spat). Then the scaffold was sliced in 10 kb intervals before cpm counts within intervals were added. Values below 2 cpm (methylation) per interval were considered background. Maps were drawn with each interval given a plotpos value on the x-axis corresponding to its position on the scaffold and using par(…, new=T) to have the maps of all the stages on the same plot. The rest of the *methylation maps.r* script corresponds to a rectangle plotted under the methylation map plot. The rectangle represents the gene and CpG content map of the considered scaffold (from *sca\*number\*\_genemap.txt* and *CpG\_Scaffold\*number\*.txt* generated using *methylation map.r* and *CGxymap.r*, respectively). Not all the genome was processed for a number of reasons including the very short length of many scaffolds (~90% of the scaffolds are shorter than 200kb). ‘Final’ maps were produced for the two longest scaffolds (Scaffold 22 and Scaffold 1009, 2Mb each), and the feature counts were examined for *de novo* methylation during development. There is almost no *de novo* methylation; maps are extremely consistent and ca. 99% of the genes that are methylated in spat were already methylated in oocytes.

**DiffReps output treatments**

The whole annotated genome was built into a (big) table, starting from the source GFF feature files, 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 : the *matrixReloaded.data* R object):

$scaffold”…”: one list object for each scaffold  
..$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

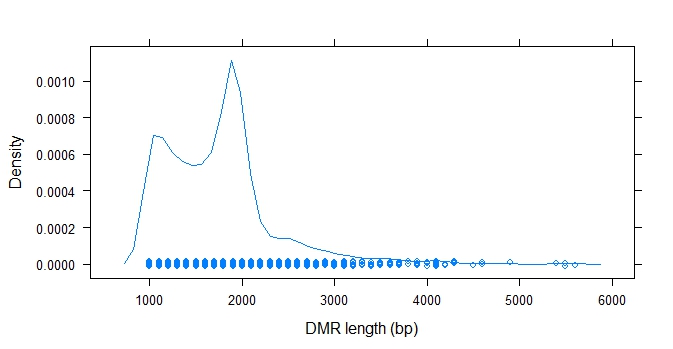
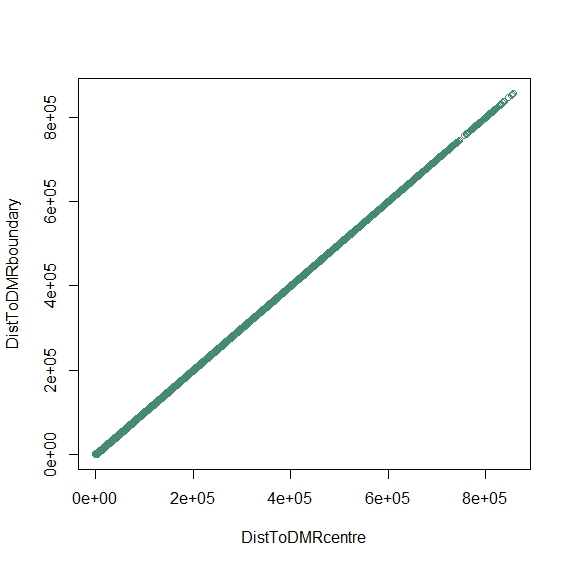
.. ..$DistCSpa: distance between feature Transstart and nearest DMR centre at the M step

.. ..$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 centre and boundary, a quick comparison was made. 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 decided to filter out a list of ‘maternal’ genes from the expression table: the maternal mRNA that are present in the ovocyte and decreased progressively its expression levels during development. An R script was used 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. Similar filtering were performed using the GigaTON database (BMC Bioinfo 2015 paper) with the same outcome, so these genes were kept for consistency between the analyses.



Analyses that were conducted after this cleaning:

1. Distance to DMR for 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.r* script was modified to extract distance data by diffReps comparison, and plot a figure.
3. Comparison of logFC of DMR by stage / logFC of gene expression: this analysis needs to create data pairs associating a DMR to a gene. Therefore, each gene was related to its with its closest DMR. The big genome object was integrated by adding, next to each gene, the logFC of its nearest DMR (see above).  
   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, the points in each quadrant were counted and a contingency table prepared to test using a classical χ2 test. These results were not included in the manuscript for simplification of the results. *Script is DMRlogVSEXPRlog.r*.
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, corresponding to C, I and M steps respectively. A statistical comparison of variances using an arbitrary 100 kb limit for DMR distance. Results are not included either, but are to be taken with caution due to the bias consecutive of the skewed distribution of DMRs within features. Scripts used: *DMRdistances.r, DMRdistancesWithTE.r, DMRdistVSexpr.r.*
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 consistent with the dramatic reduction of distances between DMR and features at metamorphosis. Script used: *DMRlength.r.*
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 led to focusing only on the 2-8 cells and spat for this analysis. Scripts used: *DMRannotation.r*, *DMRannotation-calc.r* and *DMRannotation-plot.r.*

**FEATURE-RELATED LANDSCAPES**

**Transcript variants**

A simple table with gene IDs and their respective number of transcript variants was generated using the GigaTON database. 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.r*, 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. This is included in the supplementary figures of the paper.

**Feature-related landscapes between stages**

The distribution of methyl counts in features between stages was examined using *revFig1a.r.* Pairwise comparisons of individual feature elements (ie genes and TEs in the final plot) were generated using *smearplot.r* and *newSmearPlot.r* (see below). MDS-BCV ‘pca-like’ analyses were realized using edgeR with the BCV method (see above). 3D heatmaps were generated from the log normalized values of significantly methylated features using *3dplots-plot.r* and parameters adjusted with *3dplots-launch.r*. \*\*WARNING these plots are quite greedy and resource-consuming and each feature plot need approx. 3 to 4 hours to generate at production acceptable resolution a dual core/16 GB ram machine\*\*

**Correlation between methylation and expression**

Differences in the expression of methylated vs. non methylated genes

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 and these two lists were extracted, and the mean expression level for each gene, across all development stages was calculated. R *ggplot* was used to 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). All gene body   
This comparison has been included in the figure regarding the functional differences in methylation.

Methylation levels vs. expression levels and variability

See annotated scripts *\*feature\*\_vs\_mRNA.r* and *\*feature\*\_vs\_mRNACV.r* (CV is for coefficient of variation). These scripts contain uncleaned modules for the user to change input parameters for the stringency of tests, the selection of gene subsets, etc… and may require minor adjustments.

Methylation pattern and pattern dynamics vs. expression levels and variability

The pattern of methylation within gene bodies was investigated for a possible relationship with gene mRNA expression. This pattern was approximated using the ‘ratio INT/CDS’ parameter, which is the ratio of the counts in the INT feature and in CDS feature for each gene. The variability of this pattern (ratioIN/CDS CV) was also compared to mRNA variability (mRNA CV). When methylation is perfectly consistent or ‘balanced’ between introns and exons the ratio is close to 1, otherwise it is close to 0 (skewed towards exons) or to +Inf (skewed towards introns). **UPDATE:** Preliminary analyses showed similar patterns when this proxy is skewed towards methylation in exons or in introns, and MeDIPseq has a limited resolution . Therefore it was chosen to compute the consistence between CDS and INT methylation (ratio close to 1) and skewed methylation towards introns or exons (ratios close to Inf after values <1 were inversed). The associated scripts are *mRNACV\_vs\_ratioINTCDSCV\_final.r* and *mRNA\_vs\_ratioINT\_CV\_distfrom1\_final2.r* respectively. They also have ‘to play with’ modules for test stringency and/or gene subsets selection. Normalisation of the values, selection of subsets regarding methylation level or variation across development always gave similar outcomes: the consistence in the methylation pattern is associated to expression, and ratioCV is correlated to mRNA CV regardless of the methylation level and/or normalisation (linear model p value<2.10-16).

Clustering of genes according to their methylation profile

First, methylation values were 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). The script *clusters.r* was run for the usual ANOVA by gene, and runs a k-means partitioning. The line plots that describe the temporal pattern of methylation are then plotted using only one point (mean value) per stage. The list of genes divided by cluster was saved as an R object in *GenesWithClusters.data*. Finally, the results of the K-means can be used to maintain always the same order of clusters.

**Gene Ontology**

Functional annotation of gene clusters were realized using the *TopGO* R package based on the methylation of the CDS feature (prevalent methylation of gene bodies). Expression level data and GO mappings are from the table S14 in Zhang et al. 2012 and/or generated using Blast2Go. Enrichment of terms was tested using the Fisher’s exact method using *enrich-methyl.r* as well as *DMRontologies.r* for term enrichment in DMRs. The script related to selected GO terms methylation statistics of variation, dynamics and levels (figure S3 of the manuscript) is *selectedGeneOntologies.r*.

Definition of scripts

\*\* This section provides additional details on script definition that are not obvious from annotations within scripts or in the above description. \*\*

**Figure Colours**

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")

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.

**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 are from the table S14 in Zhang et al. 2012. And/or generated using Blast2Go. 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 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**

*The logic behind this plot:*

The script *circular-plot-unique.r*  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 script plots one circular diagram with five sectors corresponding to the five different clusters identified by K-means on methylation. Each sector is a box plot with three columns, one per ontology (BP, MF, CC). 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).

**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, 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(),* we 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, the names of the two stages compared in each iteration are used to position their plot accordingly.  
This method also allows to build a “legend” in one of the squares left blank, and to insert stage names on the marginal squares.

The script was updated into *newSmearPlot.r*, 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 it 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

Four 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. The fourth figure depicting the ‘spring wire’ model was entirely produced with Illustrator.

**Figure 1 – Differentially-Methylated Regions**

This plot is completely scripted and it opens a window of the correct size, thus the user only has to launch the full script and watch the magic unfold. 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 the less Photoshop or Illustrator, the better.

**Figure 2 – Methylation profiles in features**

The plots were generated separately and then manually composed in post-production. The scripts to obtain the various components are:

* *revFig1a.r*, for the boxplot of counts distribution across stages;
* *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 3 – Expression and Functional profiles**

This is a completely scripted plot . 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 gene expression versus their CDS being methylated or not;
* *boxplot launch only2.r*, 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. This very nice part of the figure has been moved to supplementary data.