The *ChAMP* Package

Morris TJ, Butcher L, Feber A, Teschendorff A, Chakravarthy A, Beck S

July 18, 2014

The *ChAMP* package is a pipeline that integrates currently available 450k analysis methods and also offers its own novel functionality. It utilises the data import, quality control and SWAN (Maksimovic, 2012) normalization functions offered by the *minfi* (Hansen and Ayree, 2011) package. In addition, the *ChAMP* package includes the Peak Based Correction (PBC) method (Dedeurwaerder, 2011) and the BMIQ normalization (Teschendorff, 2013) is set as the default method.

A number of other pipelines and packages are available for 450k analysis including *IMA* (Wang, 2012), *minfi* (Hansen and Ayree, 2011), *methylumi* (Davis *et al.*, 2013), *R n Beads* (Assenov *et al.*, 2012) and *watermelon* (Pidsley *et al.*, 2013). However, *ChaMP* includes several analysis options that are not available in other packages. The singular value decomposition (SVD) method (Teschendorff, 2009) allows an in-depth look at batch effects and for correction of batch related to slide number the ComBat method (Johnson, 2007) has been implemented. For the identification of differentially methylated regions (DMRs) *ChAMP* offers the new Probe Lasso method. Finally, *ChAMP* has an additional function to analyse 450k for copy number alterations (Feber *et al.*, 2014).

1 Installation

It is essential that you have R already installed on your computer. *ChAMP* is a pipeline that utilises many Bioconductor packages that are currently available from CRAN and Bioconductor. For all of the steps of the pipeline to work make sure that you have installed *minfi*, *DNAcopy*, *impute*, *marray*, *limma*, *preprocessCore*, *RPMM*, *sva*, *IlluminaHumanMethylation450kmanifest* and *wateRmelon*.

This can be done in one go using the following commands:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite(c('minfi', 'DNAcopy', 'impute', 'marray', 'limma', 'preprocessCore',
+ 'RPMM', 'sva', 'IlluminaHumanMethylation450kmanifest','wateRmelon'))
```

Load the *ChAMP* package.

```
> library(ChAMP)
```

It is then easier to set your working directory to the folder containing your .idat files and sample sheet. There can only be one sample sheet in this folder.

2 Test Dataset

The package contains a test dataset of HumanMethylation450 data which can be used to test functions available in *ChAMP*.
This can be loaded by pointing the directory to the testDataSet:

```
> testDir=system.file("extdata",package="ChAMPdata")
```

Also a pre-filtered saved version can be loaded using

```
> data(testDataSet)
> myLoad=testDataSet
```

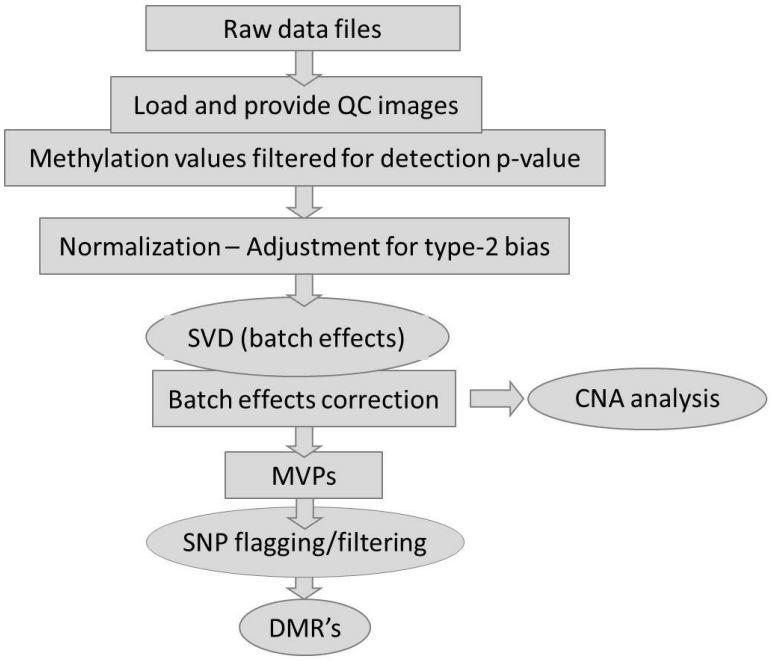

Figure 1: An overview of the *ChAMP* pipeline showing all major steps available in the pipeline. This steps can be run as a full pipeline or individually as functions combined with other packages. This vignette will explain in detail how to proceed.

3 Full Pipeline

Figure 1 outlines the steps in the *ChAMP* pipeline. Each step can be run individually as a separate function. This allows integration with other analysis pipelines. It also enables the user to save the results of each step for future reference or further analysis. Alternatively the full pipeline can be run at once with one command:

```
> champ.process(directory = testDir)
```

When running the full pipeline through the champ.process() function a number of parameters can be adjusted.

4 A note on computational requirements

The ability to run the pipeline on a large number of samples depends somewhat on the memory available. The *ChAMP* pipeline runs 200 samples successfully on a computer with 8GB of memory. Beyond this it may be necessary to find a server/cluster to run the analysis on.

The champ.load() function uses the most memory. If you plan to run the analysis more than once it is recommended to to run myLoad=champ.load() and save this list for future analyses. In this case, when the list of load objects is saved to the variable 'myLoad' you can simply run champ.process(fromIDAT=F).

```
> save(myLoad,file="currentStudyloadedData.RData")
> load("currentStudyloadedData.RData")
> champ.process(fromIDAT=FALSE)
```

Another option if you have time or space constraints or if you are combining *ChAMP* with other analysis pipelines is to run the analysis step by step and not use the champ.process() function. Each separate function is described in detail below, but the full pipeline in a step-wise process would go:

```
> myLoad=champ.load(directory = testDir)
> myNorm=champ.norm()
> champ.SVD()
> batchNorm=champ.runCombat()
> limma=champ.MVP()
> lasso=champ.lasso(fromFile =TRUE, limma=limma)
> champ.CNA()
```

5 Description of *ChAMP* functions

As previously mentioned, the user has the option to run each of the *ChAMP* functions individually to allow integration with other pipelines or to save the results of each step. Here each function is described in detail.

5.1 Load

5.1.1 Load Data from idat files

The champ.load() function utilises *minfi* to load data from .idat files. By default this loads data from the current working directory, in this directory

you should have all .idat files and a sample sheet. The sample sheet currently needs to be a .csv file as came with your results following hybridization. You can choose whether you want M-values or beta-values. For small datasets M-values are recommended (Zhuang, 2012).

The *minfi* function used to load the data from the idat files automatically filters out the 65 SNP probes that were included on the chip as internal controls which can useful for identifying sample mixups. As such, before filtering for low quality probes the dataset will include 485,512 probes.

```
> myLoad$pd
```

```
   Sample_Name Sample_Plate Sample_Group Pool_ID Project Sample_Well  Array
C1          C1           NA            C      NA      NA         E09 R03C02
C2          C2           NA            C      NA      NA         G09 R05C02
C3          C3           NA            C      NA      NA         E02 R01C01
C4          C4           NA            C      NA      NA         F02 R02C01
T1          T1           NA            T      NA      NA         B09 R06C01
T2          T2           NA            T      NA      NA         C09 R01C02
T3          T3           NA            T      NA      NA         E08 R01C01
T4          T4           NA            T      NA      NA         C09 R01C02
        Slide
C1 7990895118
C2 7990895118
C3 9247377086
C4 9247377086
T1 7766130112
T2 7766130112
T3 7990895118
T4 7990895118

C1 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
C2 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
C3 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/92473770
C4 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/92473770
T1 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/77661301
T2 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/77661301
T3 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
T4 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951

C1 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
```

```
C2 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
C3 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/92473770
C4 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/92473770
T1 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/77661301
T2 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/77661301
T3 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
T4 /Users/regmtmo/Desktop/Sync/ACTIVE work/ChrisP_450k_Lung_7Aug2012/testSet/79908951
```

5.1.2 Filtering for failed probes

By default *ChAMP* filters the data for detection p-value (< 0.01). This utilises *minfi* method for calculating the detection p-value which differs from the method used in Genome Studio. A file failedSamples.txt is saved (see Figure 2) and also printed to the screen showing the fraction of failed probes per sample. If any of these values is high (>0.05) you may want to consider removing that sample from the analysis and rerunning.

By default *ChAMP* will filter out probes with <3 beads in at least 5% of samples per probe. This default can changed with the filterBeads parameter or the frequency can be adjusted with the beadCutoff parameter.

1	Sample_Name	Fraction_Failed_Probes
2	S1	0.000976289
3	S2	0.001157541
4	S3	0.000486085
5	S4	0.000922737
6	S5	0.001157541
7	S6	0.001011304

Figure 2: An example of the output showing the portion of probes with a detection p-value above the specified cutoff (default is 0.01) for each sample. Users may want to consider removing samples above 0.05.

```
> myLoad=champ.load(directory = testDir, filterBeads=TRUE)
```

5.1.3 Output

The load function saves 3 quality control images (see Figures 3, 5 and 6. The clustering image will not be saved if there are more than 65 samples in

the dataset.

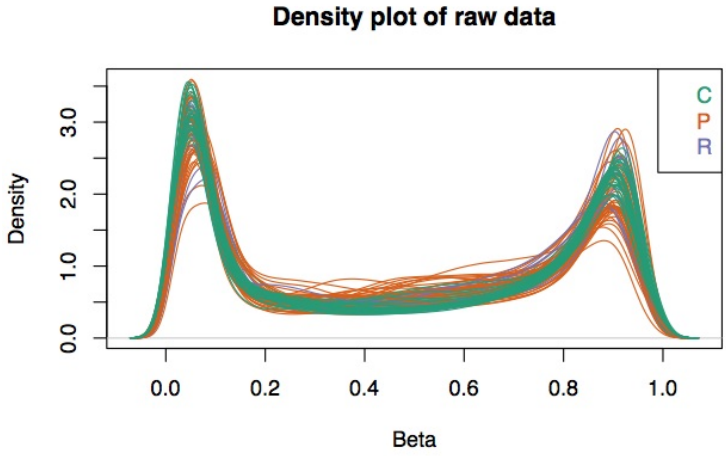

Figure 3: An example of a density plot showing the density of unnormalised beta values for all the samples that have been uploaded. They are coloured based on the Sample Group as defined in the sample sheet (Figure 4). This plot may identify potential outliers that have significantly different profiles.

	A	B	C	D	E	F	G	H	I	J
1	[Header]									
2	Investigator Name									
3	Project Name									
4	Experiment Name									
5	Date	3/18/2012								
6										
7	[Data]									
8	Sample_Nam	Sample_Plat	Sample_Gro	Pool_ID	Project	Sample_Wel	Sentrix_ID	Sentrix_Position		
9	24_blC	1	C	blood		A01	7766130072	R01C01		
10	22_blC	1	C	blood		B01	7766130072	R02C01		
11	24_blT	1	T	blood		C01	7766130072	R03C01		
12	22_blT	1	T	blood		D01	7766130072	R04C01		
13	24_brC	1	C	brain		E01	7766130072	R05C01		
14	22_brC	1	C	brain		F01	7766130072	R06C01		
15	24_brT	1	T	brain		G01	7766130072	R01C02		
16	22_brT	1	T	brain		H01	7766130072	R02C02		
17	24_pC	1	C	pancreas		A02	7766130072	R03C02		
18	22_pC	1	C	pancreas		B02	7766130072	R04C02		
19	24_pT	1	T	pancreas		C02	7766130072	R05C02		
20	22_pT	1	T	pancreas		D02	7766130072	R06C02		
21										

Figure 4: An example of the sample sheet.

5.1.4 Usage

```
> myLoad = champ.load(directory=testDir)
```

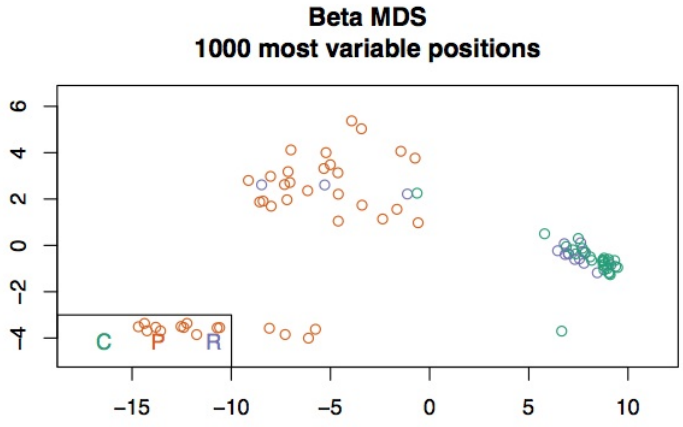

Figure 5: An example of a MDS (multidimensional scaling) plot. This plot allows a visualisation of the similarity of samples based on the top 1000 most variable probes amongst all samples. The samples are coloured by Sample_Group (as defined in the sample sheet Figure 4.

```
[read.450k.sheet] Found the following CSV files:
[1] "/Library/Frameworks/R.framework/Versions/3.1/Resources/library/ChAMPdata/extdata
          C1           C2           C3           C4           T1           T2
0.0013429122 0.0022162171 0.0003563249 0.0002842360 0.0003831007 0.0011946152
          T3           T4
0.0014953286 0.0015447610
```

5.2 Normalization for adjustment of type-2 bias

There are several normalization methods available: BMIQ (Teschendorff, 2013), SWAN (Maksimovic, 2012), PBC (Dedeurwaerder, 2011) or NONE.

The default method is BMIQ. It will save three quality control images to the folder '/Normalization' for each sample (see Figures 7, 8 and 9). These images show the fit that the normalization is applying to each probe type.

After normalization a second set of quality control images will be saved similar to pre-normalization (see Figures 3, 5 and 6). The clustering image will not be saved if there are more than 65 samples in the dataset.

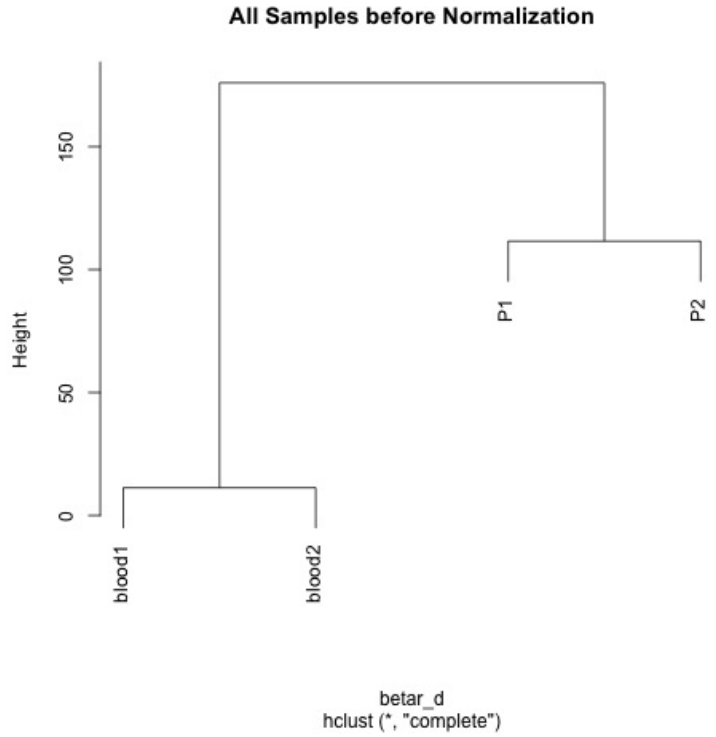

Figure 6: An example of a cluster plot. This offers a second way to visualise the similarity of samples based on all probes using hierarchical clustering.

5.2.1 Usage

```
> myNorm=champ.norm()

[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.006191036
[1] 0.004188465
[1] 0.005634045
[1] 0.006276877
[1] 0.006500057
[1] 0.006500568
[1] 0.006378492
[1] 0.006186263
[1] 0.005952443
```

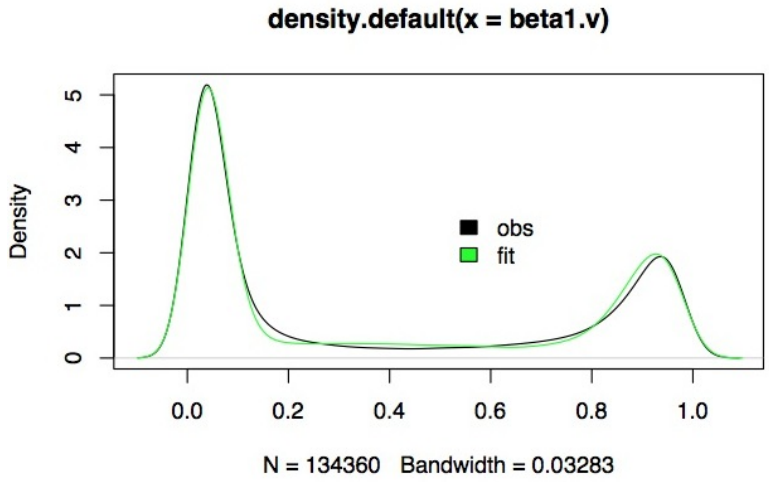

Figure 7: An example of the Type 1 fit with BMIQ.

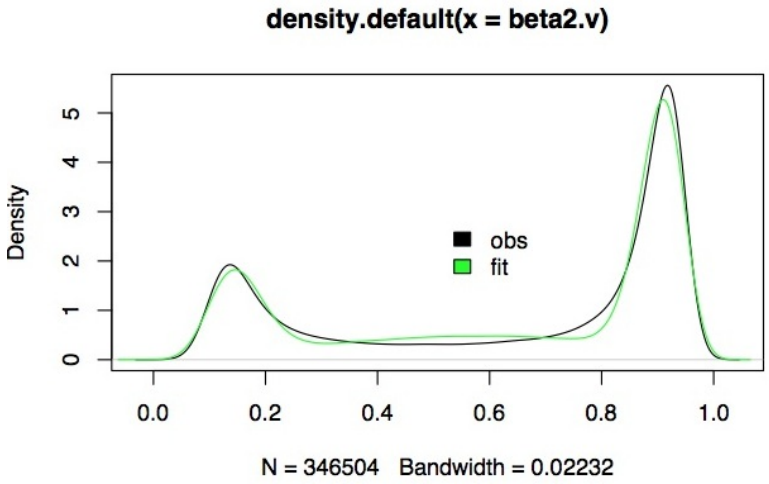

Figure 8: An example of the Type 2 fit with BMIQ.

```
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.007033563
[1] 0.003338089
```

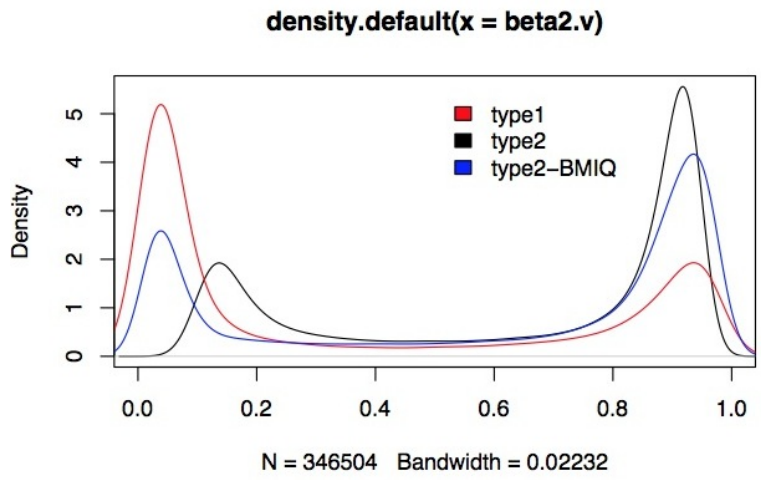

Figure 9: An overview of both type 1 and 2 fit achieved with BMIQ.

```
[1] 0.001729439
[1] 0.0009104943
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample C1"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.006573775
[1] 0.007992349
[1] 0.008885233
[1] 0.008956266
[1] 0.008667631
[1] 0.008222169
[1] 0.007709996
[1] 0.007178562
[1] 0.006636301
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.003113658
[1] 0.001810929
```

```
[1] 0.001627103
[1] 0.001317367
[1] 0.001025845
[1] 0.0007695485
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample C2"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.01018754
[1] 0.01085656
[1] 0.01108518
[1] 0.01085507
[1] 0.01037245
[1] 0.009766443
[1] 0.009111337
[1] 0.008450835
[1] 0.007810543
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.001331181
[1] 0.001004833
[1] 0.001087493
[1] 0.001219968
[1] 0.001280608
[1] 0.001290297
[1] 0.001277337
[1] 0.0013372
[1] 0.001615906
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample C3"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.007404439
[1] 0.009520642
```

```
[1] 0.009859673
[1] 0.009508654
[1] 0.008913462
[1] 0.008236997
[1] 0.007536016
[1] 0.006836855
[1] 0.006138831
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.001192635
[1] 0.0007472413
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample C4"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.007180818
[1] 0.01055302
[1] 0.01148289
[1] 0.01125918
[1] 0.01051869
[1] 0.009562452
[1] 0.008491017
[1] 0.007405749
[1] 0.006358774
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.005048938
[1] 0.005134682
[1] 0.005432618
[1] 0.00548507
[1] 0.005346675
[1] 0.005085212
[1] 0.004751035
[1] 0.004379809
```

```
[1] 0.00399619
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample T1"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.005385036
[1] 0.00266804
[1] 0.001490984
[1] 0.001554641
[1] 0.001811534
[1] 0.001846905
[1] 0.001771417
[1] 0.001642411
[1] 0.001490677
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.00919769
[1] 0.004458889
[1] 0.002482428
[1] 0.001549068
[1] 0.001086805
[1] 0.0008587614
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample T2"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.004899276
[1] 0.003942516
[1] 0.00537106
[1] 0.005797481
[1] 0.005804544
[1] 0.005632317
[1] 0.00538287
[1] 0.005101132
```

```
[1] 0.004806927
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.003487716
[1] 0.001880718
[1] 0.001105817
[1] 0.0006594542
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
[1] "Finished for sample T3"
[1] "Fitting EM beta mixture to type1 probes"
[1] Inf
[1] 0.005411847
[1] 0.006029011
[1] 0.006865831
[1] 0.00702813
[1] 0.006863271
[1] 0.006550351
[1] 0.006175359
[1] 0.005778846
[1] 0.005379059
[1] "Done"
[1] "Check"
[1] "Fitting EM beta mixture to type2 probes"
[1] Inf
[1] 0.009150745
[1] 0.005213355
[1] 0.003543846
[1] 0.002718733
[1] 0.002282639
[1] 0.002043122
[1] 0.001907019
[1] 0.001825575
[1] 0.001772188
[1] "Done"
[1] "Start normalising type 2 probes"
[1] "Generating final plot"
```

```
[1] "Finished for sample T4"
```

5.3 SVD for identification of components of variation (batch effects)

The singular value decomposition method (SVD) implemented by Teschendorff (Teschendorff, 2009) for 27k data is used to identify the most significant components of variation. These components of variation would ideally be biological factors of interest, but it may be technical variation (batch effects). To get the most from this analysis it is useful to include as much information as possible. If samples have been loaded from .idat files then the 18 internal controls on the bead chip (including bisulphite conversion efficiency) are included as well as any study details defined in the sample sheet (well position, sentrix id, batch etc) (Figure 4). Additional phenotype/study information (age, gender, batch) can be included in a separate file (set studyInfo=TRUE) (Figure 10). This must be a .txt file saved as studyInfo.txt and the first column must be labelled "Sample_Name" with the sample names as used in the sample sheet. These new covariates will be considered categorical by default. If any are continuous it is necessary to include a vector in the parameter infoFactor=c() that defines them. TRUE = categorical, FALSE=continuous. These settings will print to the screen after the SVD analysis has been run to confirm they were correctly assigned.

It is important to realise SVD does not manipulate the data, but analyses it to compute p-values based on the significance of each component. The result is a heatmap (saved as SVDsummary.pdf) of the top 6 principle components correlated to the information provided (Figure 11). The darker colours represent a lower p-value indicating a larger component of variation. If it becomes clear from this SVD analysis that the largest components of variation are technical factors (batch effects) then it is worth considering the experimental design and implementing other normalization methods that may help remove technical variation. ComBat is included in this pipeline to remove variation related to chip number but it or other methods may be implemented independently to remove other sources of technical variation revealed in the SVD analysis.

5.3.1 Usage

```
> champ.SVD()

[1] 4
[1] 0
```

	Sample_Name	Sex	Cell Type
1			
2	B1	M	bad
3	B2	M	bad
4	B3	M	bad
5	BH1	M	good
6	BH2	M	good
7	BH3	M	good
8	BH4	F	good
9	BH5	F	good
10	BH6	F	good

Figure 10: An example of the study info file. This file must include the first column "Sample_Name" and be identical to the first column of the sample sheet (Figure 4). It should be saved as studyInfo.txt and can include any addition study information including gender, age, batch, cell type, patientID etc.

```
[1] 4
     [,1]   [,2]
[1,] "TRUE" "Sample_Well"
[2,] "TRUE" "Sample_Group"
[3,] "TRUE" "Slide"
[4,] "TRUE" "Array"
null device
          1
```

5.4 Correction for batch effects related to bead chip using ComBat

This function implements the ComBat normalization method (Johnson, 2007) that was developed for microarrays. The *sva* R package is used to implement this function. ComBat is specifically defined in the *ChAMP* package to correct for batch effects related to the slide (Sentrix_ID). More advanced users can implement ComBat using *sva* documentation to adjust for other batch effects.

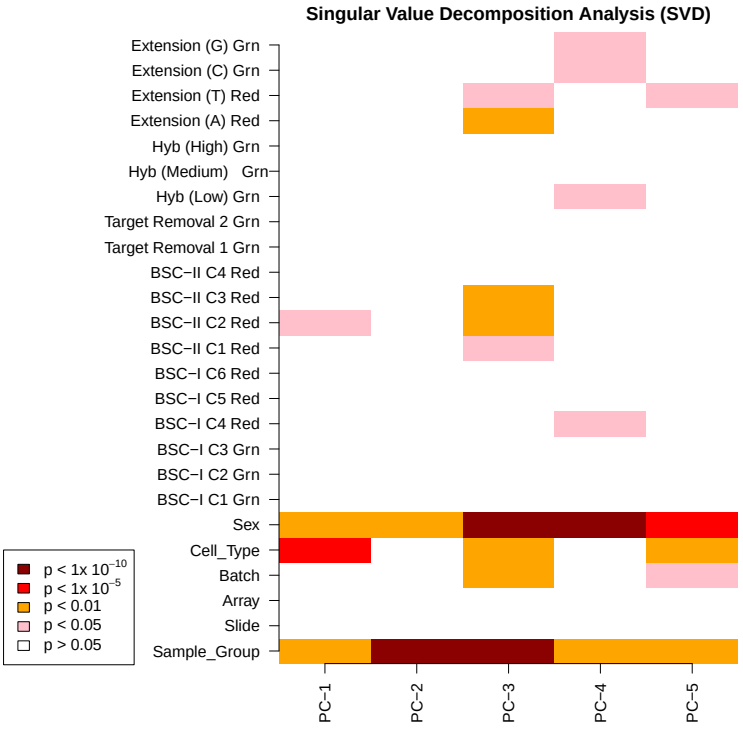

Figure 11: An example of the SVD output. This heatmap shows all the components of variation that have been included in the analysis. The first 18 are internal controls on the bead chip representing potential technical variation, The remaining covariates are obtained from data in the sample sheet (Figure 4) and the study info file (Figure 10).

5.4.1 Procedure

ComBat uses an empirical Bayes method to correct for technical variation. If ComBat were applied directly to beta values zeros could be introduced as beta values have a range between 0 and 1. For this reason *ChAMP* logit transforms beta values before ComBat adjustment and then computes the reverse logit transformation following the ComBat adjustment. If the user has chosen to use M-values no logit transformation will be done. If the dataset only includes data from one slide the adjustment will abort. As the CNA method utilises intensity values rather than beta or M -values the ComBat adjustment will be repeated for that function. The ComBat

function can be a time consuming step in the pipeline if you have a large number of samples.

5.4.2 Usage

```
> batchNorm=champ.runCombat()
```

5.5 Calling MVPs - Methylation Variable Positions

This function implements the *limma* package to calculate the p-value for differential methylation using a linear model. At present, the function can only compute differential methylation between two groups. The two groups are determined by the Sample_Group column in the sample sheet. If more than two groups are present the function will calculate the differential methylation between the first two unique groups in the file. As the function is running it will print which two groups the contrast is being computed for and will then printout how many MVPs were found for the given p-value. All probes are saved in a file "MVP_ALL.txt" which can be filtered for a p-value of interest. The filename also includes the two groups compared and the method used to adjust the p-values (for example "MVP_ALL_CvsT_BHadjust.txt"). The file includes the annotation for each probe, the average beta (for the sample group) and the delta beta for the two groups used in the comparison. In addition, a bedfile is saved with only the significant MVPs. This may be useful for downstream pathway analysis. It is planned that a future version of *ChAMP* will include the ability to compare multiple groups. However, a more advanced user can run *limma* with other settings and use the output (including all probes and their p-values) for the DMR hunter.

5.5.1 Usage

```
> limma=champ.MVP()
> head(limma)
```

5.6 DMR Hunter - Probe Lasso

This function computes and returns a data frame of probes binned into discreet differentially methylated regions (DMRs), with accompanying p-value. The data frame is distilled from a *limma* output of probes and their association statistics. It additionally writes three informative images as pdfs: 1) a box plot of probe spacing for the input data as a function of genomic feature/CGI relation (see Figure 12); 2) a cumulative quantile plot

illustrating how the user-specified parameters affect the sizes of all windows employed for DMR-calling (see Figure 13) and; 3) a dot plot with size-scaled dots to further illustrate the resulting window sizes for each genomic feature that follows from the user-specified parameters (see Figure 14).

Differentially methylated regions (DMRs) are extended and discreet segments of the genome that show a quantitative alteration in DNA methylation levels between two groups. A number of different approaches to the identification of DMRs have been implemented, most notably "windows" in which differential methylation is sought over a stretch of DNA of predefined - and often fixed - size. This approach is utilitarian and effective but is confounded by the fluctuating CpG density throughout a genome; it is further limited when using microarrays, in that coverage is often incomplete and non-uniform. Consequently, DMR identification is likely to be biased towards regions of densely tiled probes.

To compensate for this the Probe Lasso DMR Hunter is based on a feature-oriented dynamic window ("lasso"), which aims to capture neighbouring, significant probes and bundle them into DMRs.

5.6.1 Procedure

Because the Probe Lasso DMR Hunter takes into account probe spacing, the first step to identify DMRs is to calculate a dataset-specific record of probe spacing. Although it is tempting to calculate probe spacing based on all 485,512 probes on the Infinium 450k BeadChip, the Probe Lasso DMR Hunter requires input for a number of user-specified parameters (e.g., filtering of X-chromosome probes and putatively polymorphic probes) that can truncate the dataset; additionally, probe failures are inevitable and will differ between experiments. As such, each dataset is bestowed a unique set of probes, which underscores the need for an experiment-specific catalogue of probe-spacing. Once the dataset has been defined, the nearest neighbouring probe is calculated for all available probes; this data is then partitioned into one of 28 different categories comprising information on the genomic feature (1st exon, 3′UTR, 5′UTR, body, intergenic region, TSS200, or TSS1500) and that features relation (if any) to a nearby CpG island (island, shore, shelf, or not associated).

Figure 12 illustrates the wildly variable probe spacing as a function of genomic feature on the Infinium 450k BeadChip. Data were derived from an experiment in which the X-chromosome was omitted, polymorphic probes were included, and >98% of probes were detected across all samples.

The user then specifies a maximum (or minimum) lasso size (bp) and

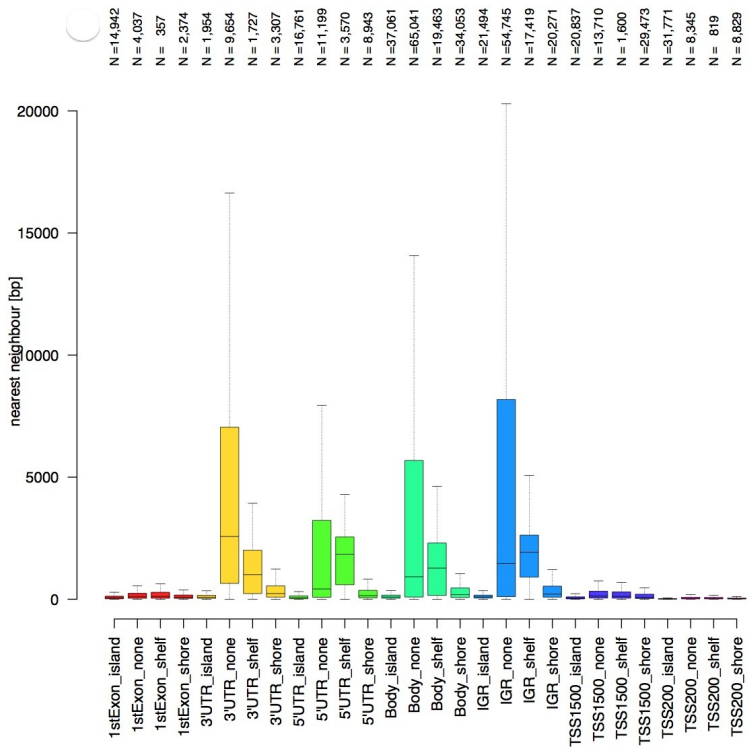

Figure 12: A study specific plot showing the number of probes found in each feature. Note the different spacing on the 450k array between neighbouring probes for different features (1st exon, 3′UTR, 5′UTR, gene body, intergenic region, transcription start sites) and their relation to the nearest CpG island (in the island, in a shore, in a shelf, not associated). Intergenic regions, 3′UTRs and gene bodies with no CGI relation are very sparsely spaced; whereas probes in the 1st exon and TSSs are very closely spaced.

the Probe Lasso DMR Hunter determines which feature/CGI category fulfils this criterion first and what quantile of the relevant feature/CGI category it corresponds to. This quantile is applied to all feature/CGI categories to define feature/CGI category-specific lasso sizes (see Figures 13 and 14).*Note a minimum lasso size would rarely be appropriate.

Next, operating solely on the significant probes in the dataset, an appropriately-sized lasso is thrown around each probe; if the lasso captures a user-specified number of significant probes (including itself), that probe (and the probes

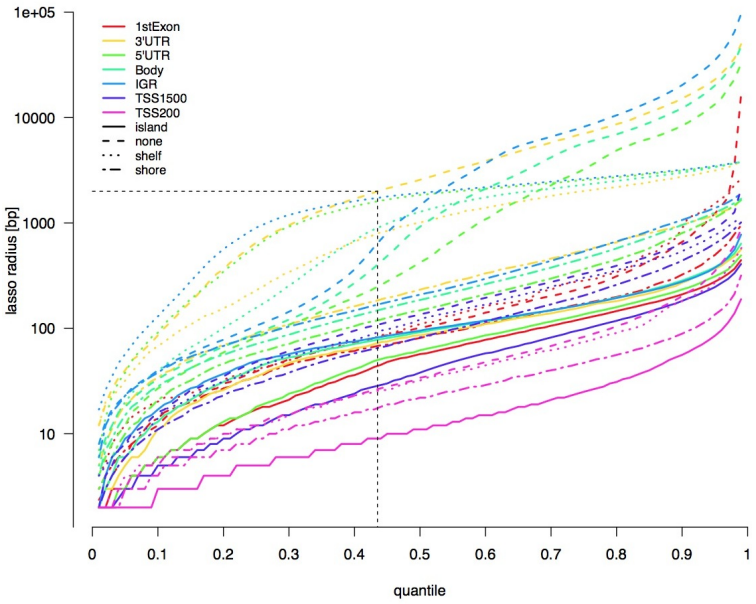

Figure 13: Defining the lasso for each feature type. This illustrates when someone has specified a minimum lasso of 10bp and how this sets the quantile for all feature/CGI relation-specific lassos to $\tilde{4}8\%$. These lassos are then cast out (centred on each probe) to capture a minimum number of significant probes. Probes that fulfil this are set aside.

captured by its lasso) is set aside, in essence, a "mini-DMR". Next, Probe Lasso DMR Hunter attempts to close-up gaps between neighbouring mini-DMRs whose lasso boundaries are either overlapping or less than a user-specified distance apart (e.g., 1000bp), effectively defining a "final DMR". The coordinates for each DMR are calculated by throwing an appropriately-sized lasso around each probe in the DMR and taking the minimum and maximum genomic coordinates.

Finally, Probe Lasso DMR Hunter pulls back all probes within the DMR coordinates and returns them as data frame containing genomic annotation and association statistics of each MVP. Additionally, a p-value for each DMR is calculated using Stouffler's method. This method combines the p-values of individual probes within a DMR by weighting them according to the underlying correlation structure of methylation scores between probes; p-values of probes whose methylation scores are correlated are down-weighted,

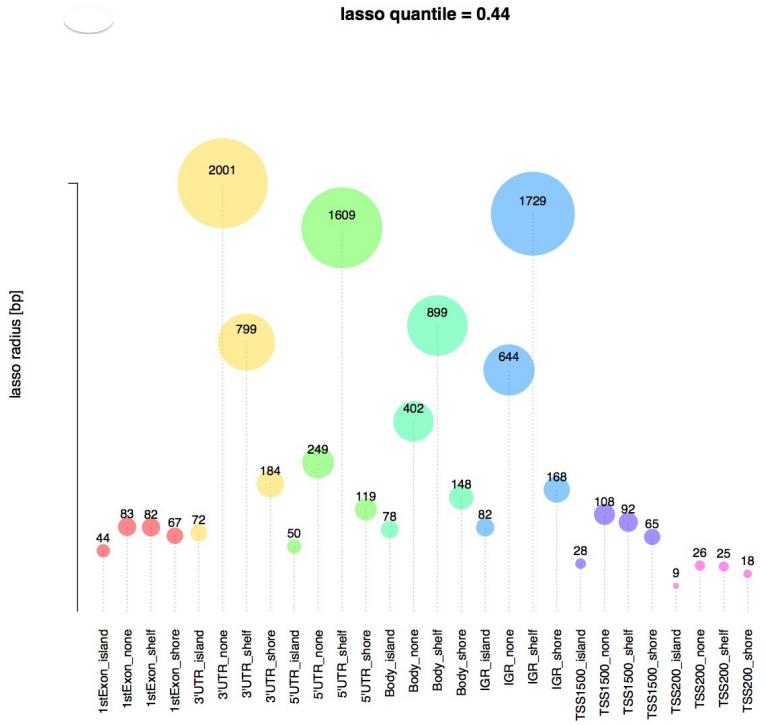

Figure 14: This image shows the radius size for each feature.

while independent probes are not penalised. The concept of the probe lasso is visualized as a cartoon in Figure 15.

5.6.2 SNP filtering

SNPs can be filtered from the DMR hunter using data obtained from the 1000 Genomes Project. This data includes SNP information for 4 populations (European, American, Asian and African). (The1000GenomesProjectConsortium, 2012)

5.6.3 Output

5.6.4 Usage

```
> lasso=champ.lasso(fromFile=TRUE, limma=limma, image=FALSE,bedFile=FALSE)
> if(!is.null(lasso))
+ {
```

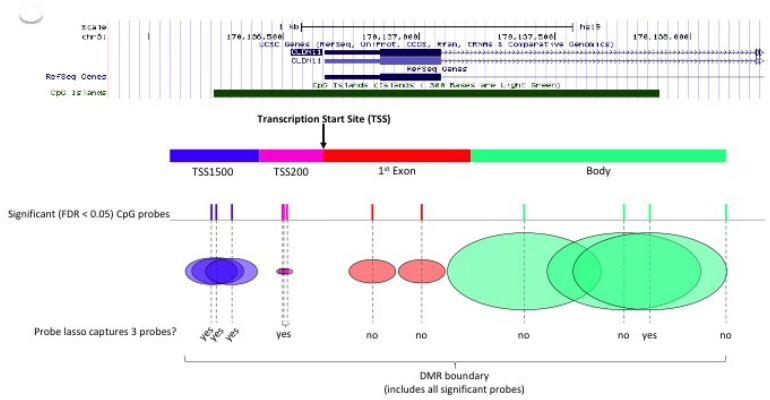

Figure 15: The details on calling a DMR. This shows the application of feature/CGI relation-specific lassos to the dataset. In the above example, six probes are 'successful' in capturing the minimum number of probes (e.g., 3) in their lasso; however, because there is less than (the user-specified) 1kb separating the first and last 'successful' probes, the DMR boundaries are defined between these (+/- their lassos!). Note that 'unsuccessful' probes are now included in the DMR.

	probeID	adj.P.Val	CHR	MAPINFO	arm	feat.rel	pol.af.f	pol.af.r	nrst.probe	lasso.radii	dmr.no	dmr.start	dmr.end	dmr.size	dmr.p
2	cg24373735	9.73E-08	1	833555	p	IGR_none	0	0	628	862	1	832693	835218	2526	3.26E-12
3	cg13938559	0.0006659	1	834183	p	IGR_none	0	0	112	862	1	832693	835218	2526	3.26E-12
4	cg12445832	0.00960585	1	834295	p	IGR_none	0	0	61	862	1	832693	835218	2526	3.26E-12
5	cg23999112	0.00044288	1	834356	p	IGR_none	0	0	61	862	1	832693	835218	2526	3.26E-12
6	cg20825023	0.1882128	1	846742	p	IGR_shore	0	0	105	172	2	846676	851273	4598	0.00015503
7	cg12549180	0.04812119	1	847816	p	IGR_shore	0	0	563	172	2	846676	851273	4598	0.00015503
8	cg11851804	0.44678109	1	848379	p	IGR_shelf	0	0	30	1733	2	846676	851273	4598	0.00015503
9	cg06623778	0.01139517	1	848409	p	IGR_shelf	0	0	30	1733	2	846676	851273	4598	0.00015503
10	cg14528386	0.03699356	1	848440	p	IGR_shelf	0	0	31	1733	2	846676	851273	4598	0.00015503

Figure 16: This is an example of the champ.lasso() output that is saved if the dataset has produced a DMR list. The columns show the probeID, adjusted p-value, chromosome, map info, chromosome arm, feature relation, SNP allele frequency for the forward strand for the selected population (pol.af.f), SNP allele frequency for the reverse strand for the selected population (pol.af.r), the distance in base pairs to the next nearest significant probe (nrst.probe), the size in base pairs of the lasso used to capture this DMR (lasso.radii), the DMR number (dmr.no), the DMR start location (dmr.start), the DMR end location (dmr.end), the DMR size (dmr.size) and the p-value of significance for the DMR (dmr.p)

```
+ head(lasso)
+ }
```

5.7 Copy Number Alterations

This function uses the HumanMethylation450 data to identify copy number alterations (Feber *et al.*, 2014). The function utilises the intensity values for each probe to count copy number and determine if copy number alterations are present. Copy number is determined using the *CopyNumber* package.

5.7.1 Procedure

Intensity values are quantile normalized and by default the ComBat normalization is run for correction of batch effects related to the slide before the copy number is calculated.

Feber (Feber *et al.*, 2014) compared the results obtained using this method to copy number data from Illumina CytoSNP array and Affymetrix SNP 6.0 arrays and found that using this method for 450k data was effective in identifying regions of gain and loss.

1	sample	chr	start	end	num.probes	seg.mean
2	X11P._qn	1	15865	1E+07	6784	-0.0395
3	X11P._qn	1	1E+07	1E+07	3	0.3169
4	X11P._qn	1	1E+07	1.1E+08	19508	-0.0506
5	X11P._qn	1	1E+08	1.1E+08	8	-0.2316
6	X11P._qn	1	1E+08	1.2E+08	1409	0.0171
7	X11P._qn	1	1E+08	1.2E+08	8	0.2655
8	X11P._qn	1	1E+08	1.6E+08	3686	-0.0333
9	X11P._qn	1	2E+08	1.6E+08	3	0.3037

Figure 17: This is an example of the champ.CNA() output that is saved for each sample. The columns show the sample name, chromosome, segment start, segment end, the number of probes in the segment and the segment mean. Typically 0.3 is used as a cut-off for the segment mean to call gains and losses.

5.7.2 Usage

```
> CNA=champ.CNA()
```

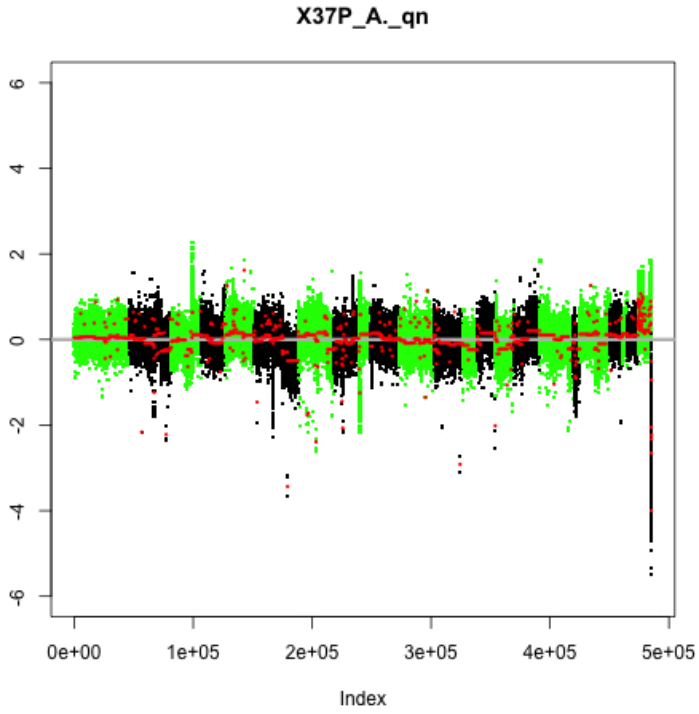

Figure 18: This image shows all the data for a single sample. Chromosomes are shown alternating green and black.

6 Further analysis

Additional options for downstream analysis are available using other packages. It is recommended that DMRs be investigated using pathway analysis software and Encode data. In addition data from previously published 450k datasets can be downloaded using *marmal-aid* and combined with study data for meta-analysis.

References

Assenov, Y., Muller, F., and Lutsik, P. (2012). RnBeads - Comprehensive DNA Methylation Analysis.

Davis, S., Du, P., Bilke, S., Triche, T., Jr., and Bootwalla, M. (2013). *methy-*

lumi: Handle Illumina methylation data. R package version 2.6.1.

Dedeurwaerder, S. e. a. (2011). Evaluation of the infinium methylation 450k technology. *Epigenomics*, **3**(6), 771–84.

Feber, A., Guilhamon, P., Lechner, M., Fenton, T., Wilson, G. A., Thirlwell, C., **T. J. Morris**, Flanagan, A. M., Teschendorff, A. E., Kelly, J. D., and Beck, S. (2014). Using high-density dna methylation arrays to profile copy number alterations. *Genome Biol*, **15**(2), R30.

Hansen, K. and Ayree, M. (2011). Analyzing Illumina 450k Methylation Arrays.

Johnson, W. E. e. a. (2007). Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, **8**(1), 118–27.

Maksimovic, J. e. a. (2012). Swan: Subset-quantile within array normalization for illumina infinium humanmethylation450 beadchips. *Genome Biol*, **13**(6), R44.

Pidsley, R., CC, Y. W., Volta, M., Lunnon, K., Mill, J., and Schalkwyk, L. C. (2013). A data-driven approach to preprocessing illumina 450k methylation array data. *BMC Genomics*, **14**, 293.

Teschendorff, A. E. e. a. (2009). An epigenetic signature in peripheral blood predicts active ovarian cancer. *PLoS One*, **4**(12), e8274.

Teschendorff, A. E. e. a. (2013). A beta-mixture quantile normalization method for correcting probe design bias in illumina infinium 450 k dna methylation data. *Bioinformatics*, **29**(2), 189–96.

The1000GenomesProjectConsortium (2012). An integrated map of genetic variation from 1,092 human genomes. *Nature*, **491**(7422), 56–65.

Wang, D. e. a. (2012). Ima: an r package for high-throughput analysis of illumina's 450k infinium methylation data. *Bioinformatics*, **28**(5), 729–30.

Zhuang, J. e. a. (2012). A comparison of feature selection and classification methods in dna methylation studies using the illumina infinium platform. *BMC Bioinformatics*, **13**, 59.