



The *ChAMP* Package

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

April 13, 2016

The *ChAMP* package is a pipeline that integrates currently available 450k and EPIC 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. First of all *ChAMP* is both available for both 450k or EPIC array. Also this package contains multiple novel functions. 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. Also another effective DMR detection function *Bumphunter* is also integrated and set default option (?).

For the purpose of dealing with cell heterogeneity problem, we included two functions `RefFreeEWAS` (?) and `RefbaseEWAS` (?) into *ChAMP*. Finally, *ChAMP* has an additional function to analyse 450k or EPIC 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 ungraded Bioconductor to newest version (3.2) and installed *minfi*, *DNAcopy*, *impute*, *marray*, *limma*, *preprocessCore*, *RPMM*, *sva*, *IlluminaHumanMethylation450kmanifest*, *plyr*, *GenomicRanges*, *RefFreeEWAS*, *qvalue*, *doParallel*, *bumphunter*, *quadprog*, *isva* 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', 'isva', 'quadprog', 'bumphunter', 'doParallel',
+ 'qvalue', 'RefFreeEWAS', 'GenomicRanges', 'plyr'))
```

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
```

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 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)`. In `champ.DMR()` function, if *Bumphunter* DMR detection method is assigned, user may use parallel method to accelerate the speed. If your server or computer has more cores, you may specify more threads at the same time to make function faster, but which may cost more memory.

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

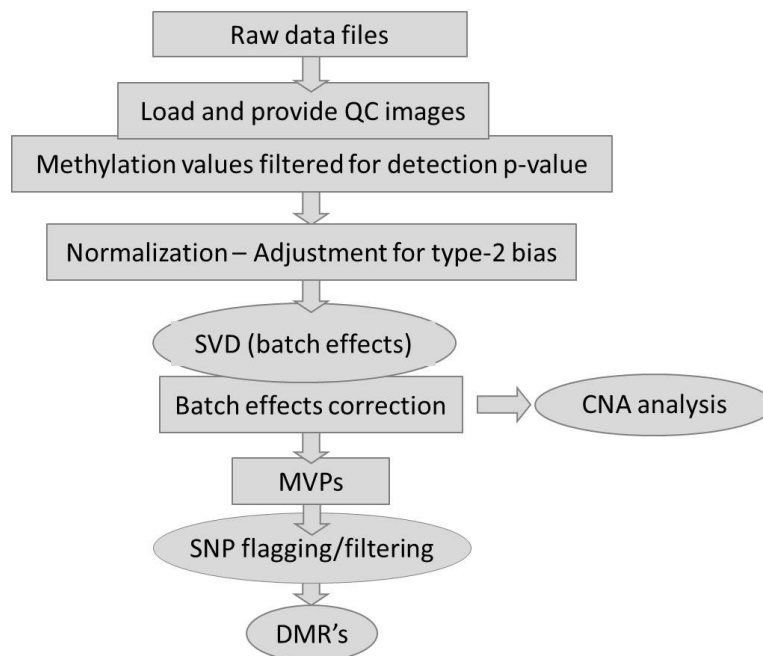


Figure 1: An overview of the *ChAMP* pipeline showing all major steps available in the pipeline. These steps can be run as a full pipeline or individually as functions combined with other packages. This vignette will explain in detail how to proceed.

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()
> myDMR <- champ.DMR()
> myRefBase <- champ.refbase()
> myRefFree <- champ.reffree()
> champ.CNA()
```

Also we included a Simulation EPIC dataset in *ChAMPdata* package, which contains 16 simulation samples and manually modified CpGs and DMRs. We simply use this dataset to show our EPIC array pipeline, which is also very easy to use.

```
> # myLoad <- champ.load(directory = testDir,arraytype="EPIC")
> # We simulated EPIC data from beta value instead of .idat file,
> # but user may use above code to read .idat files directly.
> # Here we we started with myLoad.
> data(EPICSimData)
> myNorm <- champ.norm(arraytype="EPIC")
> champ.SVD()
> batchNorm <- champ.runCombat()
> myrefbase <- champ.refbase()
> myreffree <- champ.reffree()
> limma <- champ.MVP(arraytype="EPIC")
> myDMR <- champ.DMR(arraytype="EPIC")
>
> # champ.CNA(arraytype="EPIC")
> # champ.CNA() function call for intensity data, which is not included in
> # out Simulation data.
```

Thus, it's very easy to use *ChAMP* on EPIC dataset as well, users just need to specify the 'arraytype' parameter as 'EPIC' in `champ.load()`, `champ.norm()`, `champ.MVP()`, `champ.DMR()`, `champ.CNA()` functions, then *ChAMP* will automatically perform analysis with EPIC annotation.

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 be useful for identifying sample mixups. As such, for 450k bead array data, before filtering for low quality probes the dataset will include 485,512 probes. And for EPIC bead array data, before filtering probes the dataset will include 867,531 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 be changed with the filterBeads parameter or the frequency can be adjusted with the beadCutoff parameter.

```
> 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.

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.

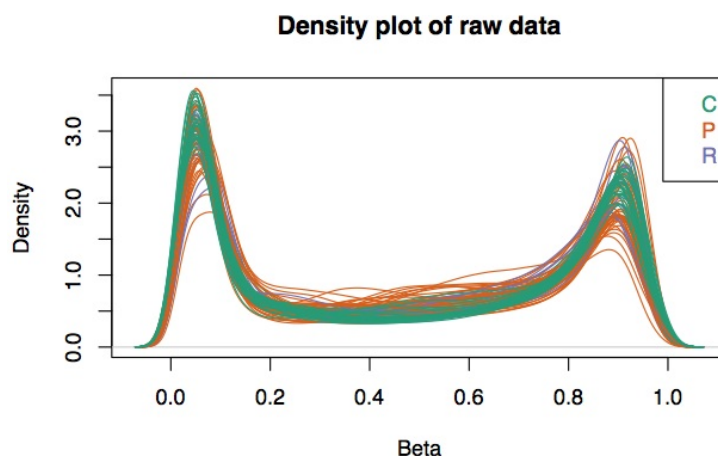


Figure 3: An example of a density plot showing the density of unnormalised beta values for all the samples that have been uploaded. They are coloured based on the Sample Group as defined in the sample sheet (Figure 4). This plot may identify potential outliers that have significantly different profiles.

5.1.4 Usage

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

```
[read.450k.sheet] Found the following CSV files:
```

```
[1] "/home/pogb.cancer.ucl.ac.uk/regmtyu/R/x86_64-redhat-linux-gnu-library/3.2/ChAMPd
```


	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_bIC	1	C	blood		A01	7766130072	R01C01		
10	22_bIC	1	C	blood		B01	7766130072	R02C01		
11	24_bIT	1	T	blood		C01	7766130072	R03C01		
12	22_bIT	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.

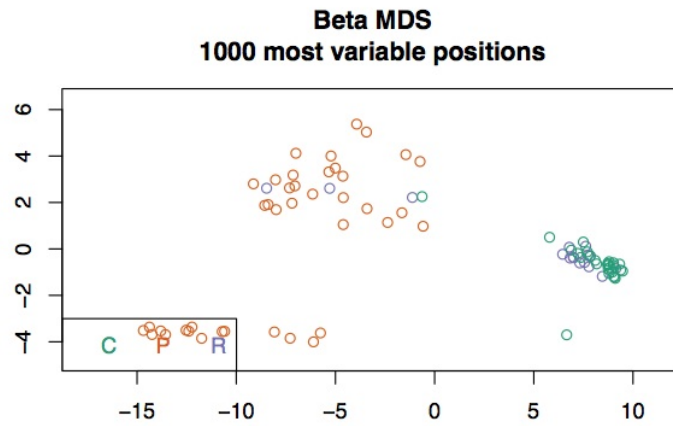


Figure 5: An example of a MDS (multidimensional scaling) plot. This plot allows a visualisation of the similarity of samples based on the top 1000 most variable probes amongst all samples. The samples are coloured by Sample_Group (as defined in the sample sheet Figure 4.)

C1	C2	C3	C4	T1	T2
0.0013429122	0.0022162171	0.0003563249	0.0002842360	0.0003831007	0.0011946152
T3	T4				
0.0014953286	0.0015447610				

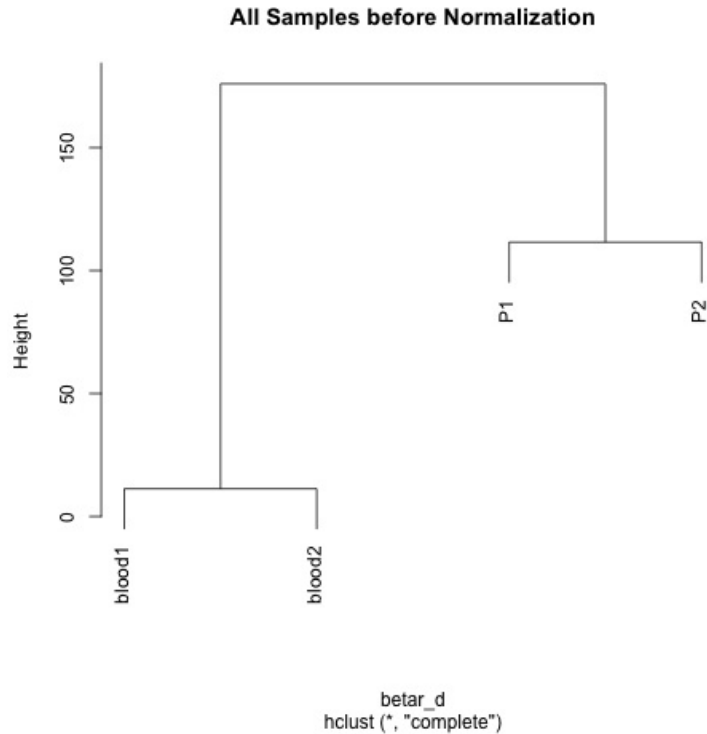


Figure 6: An example of a cluster plot. This offers a second way to visualise the similarity of samples based on all probes using hierarchical clustering.

5.2 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.

5.2.1 Usage

```
> myNorm=champ.norm()
```

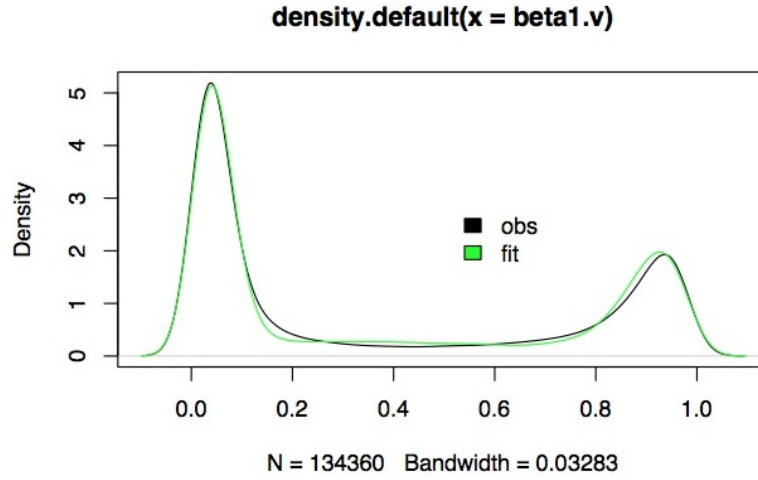


Figure 7: An example of the Type 1 fit with BMIQ.

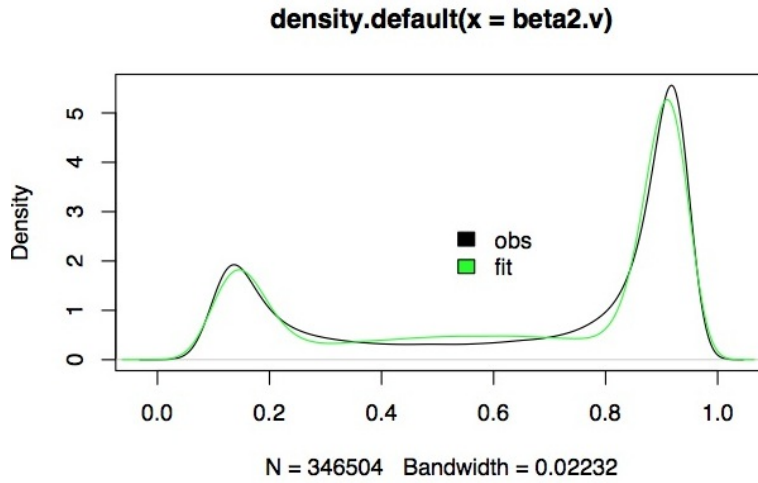


Figure 8: An example of the Type 2 fit with BMIQ.

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 ef-

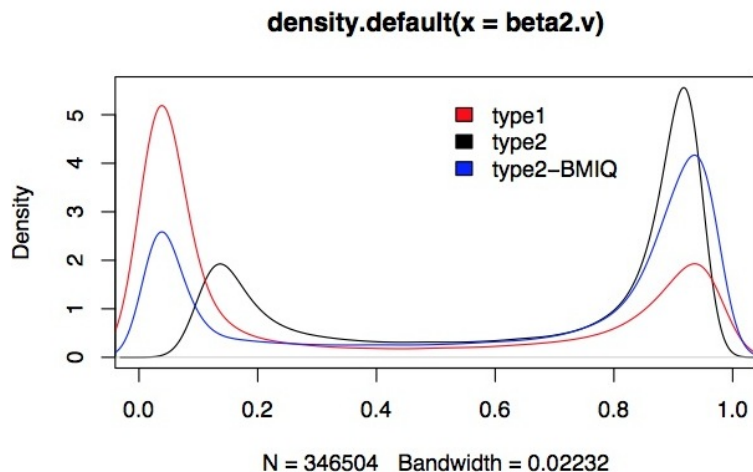


Figure 9: An overview of both type 1 and 2 fit achieved with BMIQ.

fects). 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.

1	Sample_Name	Sex	Cell Type	
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.

5.3.1 Usage

```
> champ.SVD()

[1] 4
[1] 0
[1] 4
      [,1] [,2]
[1,] "TRUE" "Sample_Well"
[2,] "TRUE" "Sample_Group"
[3,] "TRUE" "Slide"
[4,] "TRUE" "Array"
null device
      1
```

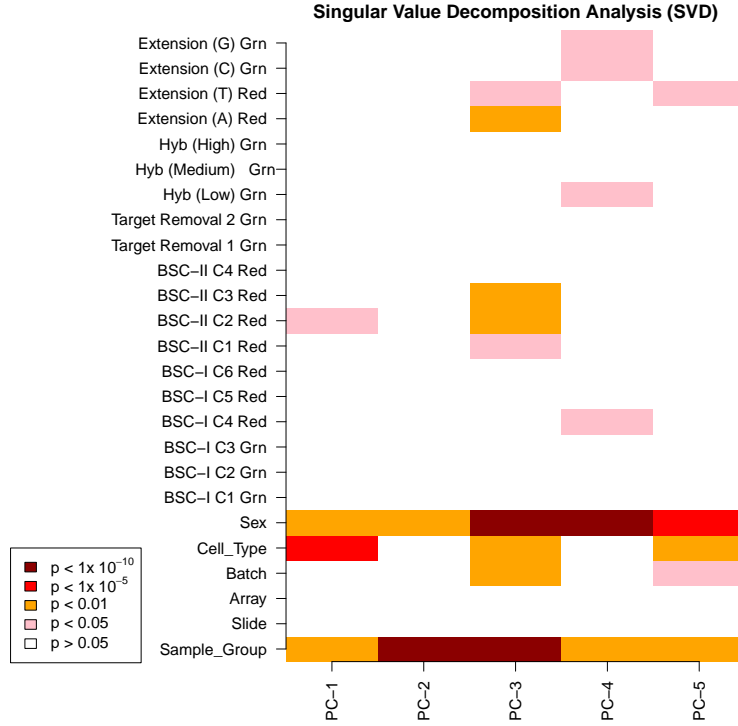


Figure 11: An example of the SVD output. This heatmap shows all the components of variation that have been included in the analysis. The first 18 are internal controls on the bead chip representing potential technical variation, The remaining covariates are obtained from data in the sample sheet (Figure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

5.4 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.

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 - Bumphunter and ProbeLasso

This function computes and returns a data frame of probes binned into discrete differentially methylated regions (DMRs), with accompanying p-values. There are two algorithms integrated to do this job, Bumphunter and ProbeLasso. For Bumphunter algorithms, it would firstly cluster all probes into small clusters, then apply random permutation method to estimate candidate DMRs. This method is very user friendly and is not relying on any output of previous functions. The permutation steps in Bumphunter algorithms may be a little bit slow, but user may assign more core to accelerate it by parallel more threads. The result of bumphunter algorithm is a dataframe contain all detected DMRs, with their length, clusters, number of CpGs.e.g annotated. Also, Bumphunter algorithm will return the annotation of all DMR contained CpGs.

For ProbeLasso method, the final 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 13); 2) a cumulative quantile plot illustrating how the user-specified parameters affect the sizes of all windows employed for DMR-calling (see Figure 14) 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 15).

Differentially methylated regions (DMRs) are extended and discrete 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 pre-defined - 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

ProbeLasso 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

For Bumhunter algorithm, it will replying on no previous output. Firstly, Bumhunter all cluster all probes into small clusters. The size and distances between CpGs could be specified by users. Then Bumhunter will filter all clusters contain too few CpGs in it, the default threshold is 7, but can be specified by user as well. After filtering, Bumhunter will select candidate DMRs based on probes' differential t value between two groups of samples, and their location in one cluster. Finally random permutation technic will return significance of each candidate DMR and p value will be returned. For more information, user may turn to *Bumhunter* package.

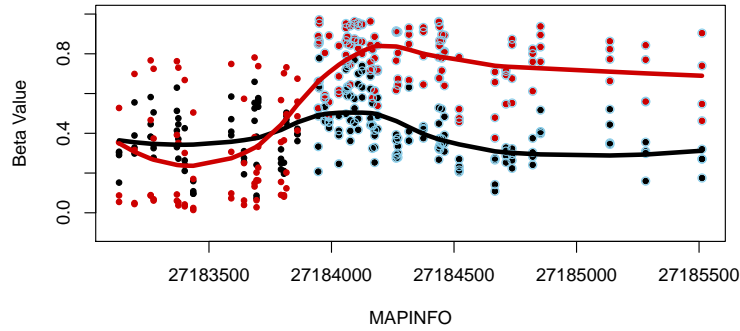


Figure 12: This image is one example of DMR generated from Bumhunter algorithm via `champ.DMR()` function. This DMR is located on chromosom 7. Totally there are 49 CpGs contained in this cluster, but real DMR stars from index 18 to the end of this cluster, thus totally this is a DMR with 32 probes. As we can see the DMR probes are marked as blue circle. Case and control are separatly plotted as red and black. Two loess lines are drawn across two points. Obviously, the last part of the cluster is a DMR between case and control.

For ProbeLasso algorithm, because the ProbeLasso 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 probes on the Infinium 450k or EPIC 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 13 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 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 14 and 15).*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 captured by its lasso) is set aside, in essence, a "mini-DMR". Next, ProbeLasso 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, ProbeLasso 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 Stouffer'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,

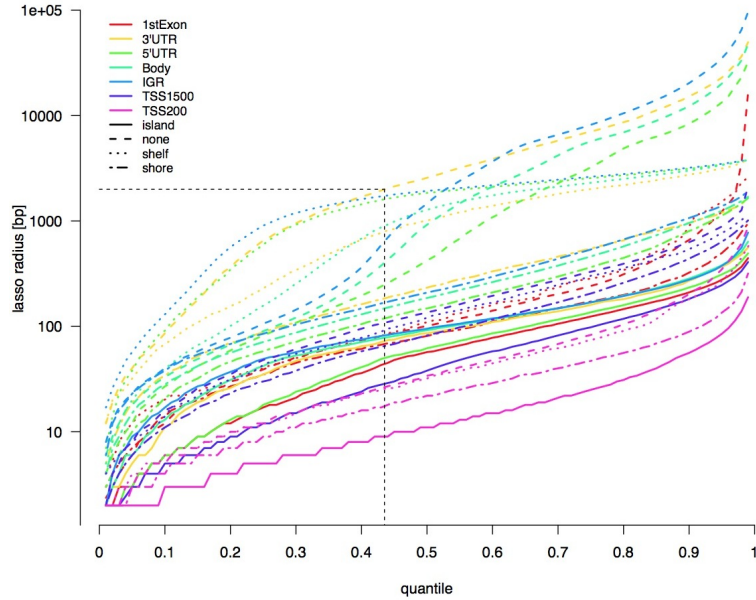


Figure 14: Defining the lasso for each feature type. This illustrates when someone has specified a minimum lasso of 10bp and how this sets the quantile for all feature/CGI relation-specific lassos to 48%. These lassos are then cast out (centred on each probe) to capture a minimum number of significant probes. Probes that fulfil this are set aside.

5.6.2 Output

5.6.3 Usage

```
> lasso <- champ.DMR(resultFiles=limma,method="ProbeLasso",arraytype="450K")
> bump <- champ.DMR(method="Bumphunter",arraytype="450K")
> if(!is.null(lasso))
+ {
+   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

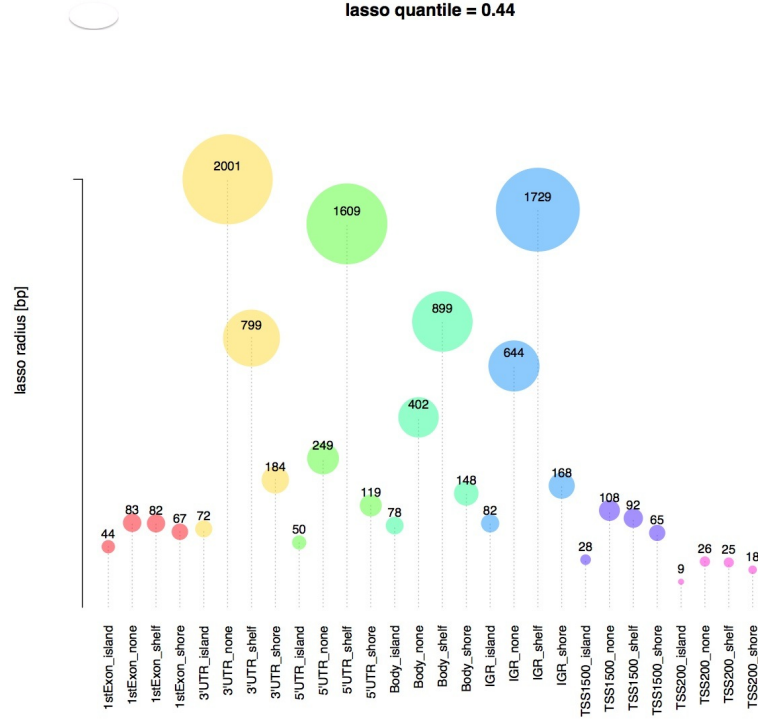


Figure 15: This image shows the radius size for each feature.

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.

5.7.2 Usage

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

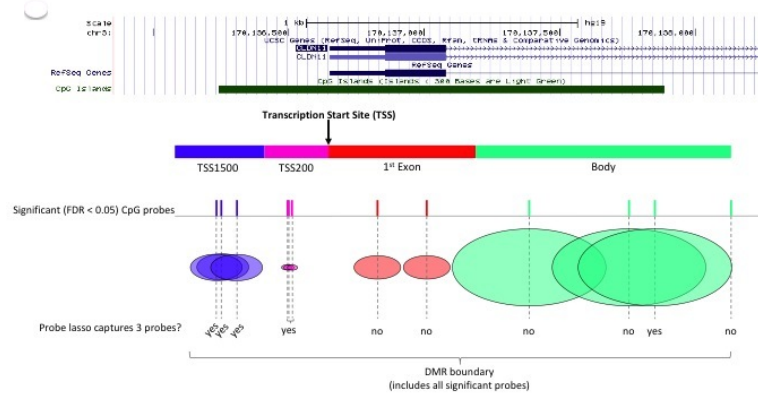


Figure 16: The details on calling a DMR with ProbeLasso Method. This shows the application of feature/CGI relation-specific lassos to the dataset. In the above example, six probes are 'successful' in capturing the minimum number of probes (e.g., 3) in their lasso; however, because there is less than (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.radius	dmr.no	dmr.start	dmr.end	dmr.size	dmr.p
1	cg24373735	9.73E-08	1	833555	p	IGR_none	0	0	628	862	1	832693	835218	2526	3.26E-12
2	cg13938959	0.00064659	1	834183	p	IGR_none	0	0	112	862	1	832693	835218	2526	3.26E-12
3	cg12445832	0.00960585	1	834295	p	IGR_none	0	0	61	862	1	832693	835218	2526	3.26E-12
4	cg13999112	0.00044288	1	834356	p	IGR_none	0	0	61	862	1	832693	835218	2526	3.26E-12
5	cg20825023	0.1882128	1	846742	p	IGR_shore	0	0	105	172	2	846676	851273	4598	0.00015503
6	cg12549180	0.04812119	1	847816	p	IGR_shore	0	0	563	172	2	846676	851273	4598	0.00015503
7	cg11851804	0.44678109	1	848379	p	IGR_shelf	0	0	30	1733	2	846676	851273	4598	0.00015503
8	cg06623778	0.01139517	1	848409	p	IGR_shelf	0	0	30	1733	2	846676	851273	4598	0.00015503
9	cg14574886	0.01660056	1	848449	n	IGR_shelf	0	0	31	1733	2	846676	851273	4598	0.00015503

Figure 17: This is an example of the champ.DMR() output on ProbeLasso algorithm 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.radius), 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)

5.8 Cell type heterogeneity correction

While many methylation sample are collected from whole blood, their methylation status actually are combination of various cell types. These cell-type specified methylation may cover true epigenome signals as well as true rela-

		chr	start	end	value	anno	cluster	indexStart	indexEnd	L	clusterL	p.value	fwcr	p.valueAnno	fwcrAnno
1															
2	cg23375985	6	30094947	30095902	2.9076575152	81.414419425	176677	78885	78812	28	28	0	0	0.000141446	0.32
3	cg13934308	12	115134148	115180338	2.7012518993	127.78384869	54030	25505	25570	51	51	9.24E-007	0.004	2.21E-005	0.090
4	cg27435917	17	46955259	46956934	4.6540933779	75.89837234	91836	49441	49860	17	17	9.24E-007	0.004	0.000170105	0.350
5	cg16164092	6	31782075	31783545	3.4552783697	83.295215981	177287	81579	81601	27	27	3.70E-006	0.016	8.32E-005	0.22
6	cg23335838	7	45961078	45962283	3.3552432084	80.598158995	191508	90279	90305	27	28	3.70E-006	0.016	9.53E-005	0.248
7	cg29193479	11	66382000	66383940	3.3716921389	87.433848778	41905	19399	19418	20	20	4.62E-006	0.016	0.000255175	0.42
8	cg17991918	5	131892142	131892523	1.546743599	20.513995294	268773	73879	73882	4	9	2.31E-005	0.02	0.021277373	0.94
9	cg18478100	6	32115964	32117565	2.4664026538	86.431892882	177465	62595	62629	35	36	7.40E-006	0.024	0.000109091	0.268
10	cg27595952	16	2050730	2051134	4.9364509399	64.173928518	78813	34951	34955	13	13	7.40E-006	0.024	0.0001133999	0.444
11	cg207763410	3	181443800	181445014	4.6869102785	62.564743999	150497	66598	66571	14	14	6.47E-006	0.024	0.0003337396	0.46

Figure 18: This is an example of the champ.DMR() output on Bumphunter algorithm that is saved if the dataset has produced a DMR list. The columns show the DMR located chromosome, start site and end site, average beta value, cluster index, start CpG index, end CpG index, number of CpG Probes in each DMR, total number of probes in each cluster, and p value calculated for each DMR.

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 19: 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.

tionships between CpGs sites and phenotypes, or themselves are the reason of some diseases. Thus many methods have be invented to deal with cell heterogeneity problem, like RefbaseEWAS and RefFreeEWAS incorporated in ChAMP.

RefbaseEWAS method is a method similar to regression calibration, for inferring changes in the distribution of white blood cells between different subpopulations (e.g. cases and controls) using DNA methylation signatures, in combination with a previously obtained external validation set consisting of signatures from purified leukocyte samples.

We integrates another cell type heterogeneity correction method, RefFreeEWAS in *ChAMP*. The new RefFreeEWAS method is similar to non-negative

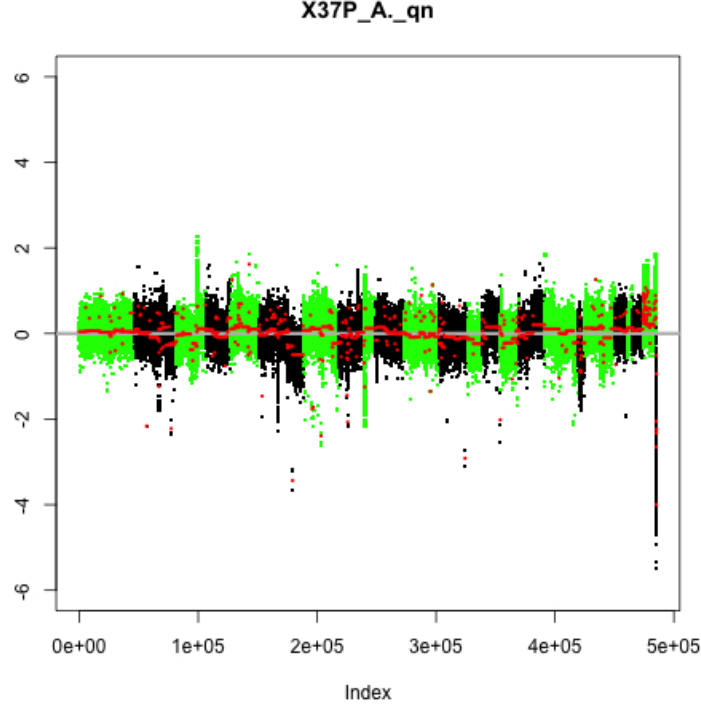


Figure 20: This image shows all the data for a single sample. Chromosomes are shown alternating green and black.

matrix factorization, with additional constraints and additional utilities. This function is specifically suitable for tissue datasets such as placenta, saliva, adipose or tumor tissue. This method closely related to surrogate variable analysis, relies on a simple projection based on singular value decomposition (SVD), as does SVA. In SVA, the residuals of a linear model are decomposed into a factor-analytic structure and the factors are used subsequently in a regression model, with iteration resulting in a final set of surrogate variables.

5.8.1 Procedure

Here we prepared two cell-type purified methylation reference, one for 27K and the other for 450K, user may use any of them to do cell heterogeneity

correction even for EPIC bead array, because the first step of the function is to extract common CpG sites between reference dataset and inputted beta valued dataset. Then by applying quadratic programming, cell proportions for each sample could be detected out. Then, linear regression will be applied to correct heterogeneity based on cell proportions of each sample.

However, though RefbaseEWAS method would return fairly accurate cell proportion, this method can ONLY be applied for whole blood samples, which contain cells in the reference. After `champ.refbase()` function, cell type heterogeneity corrected beta matrix, and cell-type specific proportions in each sample will be returned.

On the other hand, RefFreeEWAS method requires two parameters in `champ.refree()` function, one is the covariates matrix for inputted dataset, which could be a matrix or a vector. containing various phenotypes of each sample. The other important parameter is number of latent variable K, which represent numbers of latent cell type mixed in the dataset. If users don't know the correct number of latent variable (cell types), they could ignore this parameter, and `champ.refree()` will apply Random Matrix Theory from *isva* package to estimate number of latent variables.

5.8.2 Usage

```
> refbase <- champ.refbase()  
> refree <- champ.refree()
```

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., Morris, T. J., 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.
- 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.