FourCSeq analysis workflow

Felix A. Klein
European Molecular Biology Laboratory (EMBL),
Heidelberg, Germany
felix.klein@embl.de

October 14, 2015

Contents

1 Introduction

This vignette shows an example workflow of a 4C sequencing analysis. In an typical setting 4C sequencing data has been generated for different viewpoints in several replicates of multiple conditions. We focus on the analysis of a subset of the data which was recently published [1]. The data set comprises one viewpoint in replicates of 3 conditions.

> For further information on the underlying method and if you use *FourCSeq* in published research please consult and cite:
>
> Felix A. Klein, Simon Anders, Tibor Pakozdi, Yad Ghavi-Helm, Eileen E. M. Furlong, Wolfgang Huber
> FourCSeq: Analysis of 4C sequencing data
> Bioinformatics (2015). doi:10.1093/bioinformatics/btv335 [2]

2 Preprocessing

The analysis with *FourCSeq* starts from binary alignment/map (BAM)-files. If you already have seperate bam files for each viewpoint you can skip this section, which shows a possible way how to generate these bam files.

Usually many viewpoints are multiplexed in one sequencing lane. To demultiplex or just trim off the primer sequence the *FourCSeq* contains the python script "demultiplex.py" in the folder "extdata/python". To run the python script you have to install the HTSeq python package (http://www-huber.embl.de/users/anders/HTSeq/doc/install.html).

Then you can run the command:

```
python pathToScriptFile/demultiplex.py --fastq YourFASTQFile --barcode YourBarcodeFile
```

The barcode file is a FASTA file containing the primer sequences that have been used to generate the 4C library. The read starts are matched against these sequences and if a unique match is found the primer sequence is trimmed and the remaining read is saved in a FASTQ file with the viewpoint name attached to the original file name, e.g. for the FASTQ input 4c_library.fastq and a primer sequence named "viewpoint1" in the primer FASTA file, the script will generate the output file 4c_library_viewpoint1.fastq for reads matching to the "viewpoint1" primer sequence.

Here is an example content of a primer FASTA file, containing one sequence for the "testdata" viewpoint:

```
>testdata
ATTTTCTCATCCATATAAATACTA
```

For additional parameters that can be passed to demulitiplex.py have a look at the help documentation of the python script by running:

```
python pathToScriptFile/demultiplex.py --help
```

If you don't know where the python script in the package is installed use the following command in R.

```
system.file("extdata/python/demultiplex.py", package="FourCSeq")
```

After demultiplexing the files can be aligned with standard alignment software generating bam output.

3 Initialization of the FourC object

As first step we need to load the required libraries.

```
library(FourCSeq)
```

To start the analysis we need to make a *FourC* object. The *FourC* object is created from a *list* `metadata` containing information about the experiment and a *DataFrame* `colData` containing information about the samples. We now look at this in more detail.

For `metadata` the following information is required:

1. `projectPath`, directory where the project will be saved.
2. `fragmentDir`, subdirectory of the project directory where to save the information about restriction fragments.
3. `referenceGenomePath`, path to the FASTA file of the reference genome or a *BSgenome* object.
4. `reSequence1` and `reSequence2`, restriction enzyme recognition sequence of the first and second restriction enzyme used in the 4C protocol, respectively.
5. `primerFile`, path to a FASTA file containing the primer sequences of the viewpoints used for preparing the 4C libraries (names of the primer have to match the names of the viewpoints provided in `colData`).
6. `bamFilePath`, path to a directory where the bam files are stored.

For demonstration purposes example files of the ap viewpoint, containing only a small region of the first 6900 bases on chromosome chr2L of the dm3 *Drosophila* genome, are saved in the *FourCSeq* package. Later on we load a processed (FourC) object that contains the whole data for chr2L and chr2R (chr2R is the viewpoint chromosome of the ap example viewpoint). We get the path to these files using the `system.file` function. For your own data you have to adjust the file path to the directory where your files are stored.

```
referenceGenomeFile = system.file("extdata/dm3_chr2L_1-6900.fa",
                                  package="FourCSeq")
referenceGenomeFile
```

```
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/dm3_chr2L_1-6900.fa"
```

```
bamFilePath = system.file("extdata/bam",
                          package="FourCSeq")
bamFilePath
```

```
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/bam"
```

```
primerFile = system.file("extdata/primer.fa",
                         package="FourCSeq")
primerFile
```

```
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/primer.fa"
```

We also take a look at the content of the primer file.

```
writeLines(readLines(primerFile))
```

```
>testdata
ATTTTCTCATCCATATAAATACTA
```

The primer file contains one sequence, namely for the "ap" viewpoint.

Next we create `metadata` using "exampleData" as directory for the projectPath and the two restriction enzyme cutting sequences of DpnII (GATC) and NlaIII (CATG) that were used in the experiment.

```
metadata <- list(projectPath = "exampleData",
                 fragmentDir = "re_fragments",
                 referenceGenomeFile = referenceGenomeFile,
                 reSequence1 = "GATC",
                 reSequence2 = "CATG",
                 primerFile = primerFile,
                 bamFilePath = bamFilePath)
metadata
```

```
## $projectPath
## [1] "exampleData"
##
## $fragmentDir
## [1] "re_fragments"
##
## $referenceGenomeFile
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/dm3_chr2L_1-6900.fa"
##
## $reSequence1
## [1] "GATC"
##
## $reSequence2
## [1] "CATG"
##
## $primerFile
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/primer.fa"
##
## $bamFilePath
## [1] "/private/tmp/RtmpsX1Ogj/Rinst500b56ee9d89/FourCSeq/extdata/bam"
```

After creating `metadata` we now look at `colData`

For each library the following information has to be provided to `colData`:

1. `viewpoint`, name of the viewpoint (has to match the viewpoint names in the provided primer file in `metadata`).
2. `condition`, experimental condition.
3. `replicate`, replicate number.
4. `bamFile`, file name of the bam file.
5. `sequencingPrimer`, was the 4C library sequenced from the side of the first restriction enzyme cutting site or the second. The allowed values are "first" or "second"

```
colData <- DataFrame(viewpoint = "testdata",
                     condition = factor(rep(c("WE_68h", "MESO_68h", "WE_34h"),
                                            each=2),
```

```
                                        levels = c("WE_68h", "MESO_68h", "WE_34h")),
                    replicate = rep(c(1, 2),
                                    3),
                    bamFile = c("CRM_ap_ApME680_WE_6-8h_1_testdata.bam",
                                "CRM_ap_ApME680_WE_6-8h_2_testdata.bam",
                                "CRM_ap_ApME680_MESO_6-8h_1_testdata.bam",
                                "CRM_ap_ApME680_MESO_6-8h_2_testdata.bam",
                                "CRM_ap_ApME680_WE_3-4h_1_testdata.bam",
                                "CRM_ap_ApME680_WE_3-4h_2_testdata.bam"),
                    sequencingPrimer="first")
colData
```

```
## DataFrame with 6 rows and 5 columns
##     viewpoint condition replicate                                  bamFile
##   <character>  <factor> <numeric>                              <character>
## 1    testdata    WE_68h         1   CRM_ap_ApME680_WE_6-8h_1_testdata.bam
## 2    testdata    WE_68h         2   CRM_ap_ApME680_WE_6-8h_2_testdata.bam
## 3    testdata  MESO_68h         1 CRM_ap_ApME680_MESO_6-8h_1_testdata.bam
## 4    testdata  MESO_68h         2 CRM_ap_ApME680_MESO_6-8h_2_testdata.bam
## 5    testdata    WE_34h         1   CRM_ap_ApME680_WE_3-4h_1_testdata.bam
## 6    testdata    WE_34h         2   CRM_ap_ApME680_WE_3-4h_2_testdata.bam
##   sequencingPrimer
##        <character>
## 1            first
## 2            first
## 3            first
## 4            first
## 5            first
## 6            first
```

After having the necessary information in the required form, we create the *FourC* object.

```
fc <- FourC(colData, metadata)
fc
```

```
## class: FourC
## dim: 0 6
## metadata(7): projectPath fragmentDir ... primerFile bamFilePath
## assays(0):
## rownames: NULL
## rowRanges metadata column names(0):
## colnames(6): testdata_WE_68h_1 testdata_WE_68h_2 ... testdata_WE_34h_1
##   testdata_WE_34h_2
## colData names(5): viewpoint condition replicate bamFile sequencingPrimer
```

We now have an *FourC* object that contains all the required metadata.

4 Fragment reference

As the next step the provided reference genome is *in-silico* digested using the provided restriction enzyme recognition sequences. The resulting fragment reference is stored as `rowRanges` of the *FourC* object.

```
fc <- addFragments(fc)


fc
```

```
## class: FourC
## dim: 2 6
## metadata(7): projectPath fragmentDir ... primerFile bamFilePath
```

```
## assays(0):
## rownames: NULL
## rowRanges metadata column names(4): leftSize rightSize leftValid rightValid
## colnames(6): testdata_WE_68h_1 testdata_WE_68h_2 ... testdata_WE_34h_1
##   testdata_WE_34h_2
## colData names(5): viewpoint condition replicate bamFile sequencingPrimer
```

```
rowRanges(fc)
```

```
## GRanges object with 2 ranges and 4 metadata columns:
##       seqnames       ranges strand |  leftSize rightSize leftValid rightValid
##          <Rle>    <IRanges>  <Rle> | <numeric> <numeric> <logical>  <logical>
##   [1]    chr2L [5305, 6022]      * |       160       554         1          1
##   [2]    chr2L [6027, 6878]      * |       251       597         1          1
##   -------
##   seqinfo: 1 sequence from an unspecified genome
```

Now the *FourC* object contains a *GRanges* object in the `rowRanges` slot with the information on the fragments.

By setting `save` to `TRUE` in `addFragments`, the results of the *in-silico* digestion can be saved in the provided fragmentDir folder in the project directory, which are both defined in `metadata`. The first file (valid_fragments.txt) contains the information for all fragments of the first restriction enzyme in the following columns:

1. Chromosome
2. Fragment start
3. Fragment end
4. Size of the left fragment end
5. Size of the right fragment end
6. Information whether the left fragment end is valid
7. Information whether the right fragment end is valid

The second file contains the locations of all cutting sites of the second restriction enzyme in the following columns:

1. Chromosome
2. Cutting site start
3. Cutting site end

Additionally, bedgraph files (re_sites_Seqeunce1/Sequence2.bed) are produced in the same folder for displaying the cutting sites in a genome viewer of choice (e.g. IGV or UCSC).

4.1 Adding the viewpoint information

To find the viewpoint fragment and extract the genomic position of the viewpoint, the primers are mapped to the reference genome and fragment reference. Because this can be time consuming for many sequences the results of `findViewpointFragments` is saved in the project directory provided in `metadata`. In a second step this data is loaded by `addViewpointFrags` and the `colData` of the *FourC* object is updated with the corresponding information of each viewpoint.

```
findViewpointFragments(fc)

fc <- addViewpointFrags(fc)
```

The mapped primer fragments are also saved in the file "primerFragments.txt" in the provided fragmentDir folder in project directory both defined in `metadata`. In contains one row per primer and the following columns:

1. Viewpoint
2. Chromosome
3. Fragment start position
4. Fragment end position
5. Width of the whole fragment
6. Size of the left fragment end
7. Size of the right fragment end

8. Information whether the left fragment end is valid
9. Information whether the right fragment end is valid
10. Primer start position
11. Primer end position
12. Fragment side on which the primer matches

4.2 Adding the viewpoint information manually

If the primer file is missing, this information can also be added manually. The information has to contain the viewpoint chromosome name `chr`, the start position of the viewpoint fragment `start`, and its end position `end`

```
colData(fc)$chr = "chr2L"
colData(fc)$start = 6027
colData(fc)$end = 6878
```

5 Counting reads at fragment ends

To filter out non-informative reads, we use several criteria motivated by the 4C sequencing protocol. In the function `countFragmentOverlaps`, only reads mapping exactly to the end of a fragment with the correct orientation are counted and assigned to the corresponding fragment in this step (Figure 1). The counting is strand specific, taking the orientation of the reads into account (Figure 1). The count values are stored as matrices in the `assays` slot of the *FourC* object. They are named `countsLeftFragEnd` and `countsRightFragEnd`.

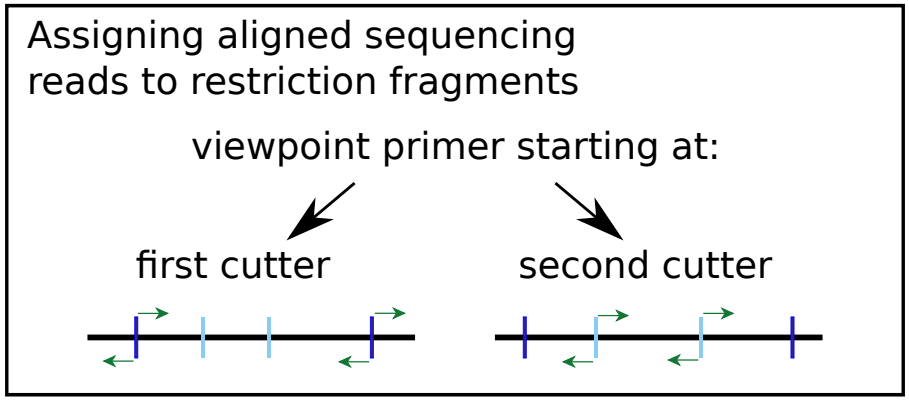


Figure 1: If the sequencing primer starts at the first restriction enzyme cutting site, reads that start at the fragment ends and are oriented towards the fragment middle are kept for analysis (green arrows). If the sequencing primer starts at the second restriction enzyme cutting site, reads that start directly next to the cutting site of the second restriction enzyme and are directed towards the ends of the fragment are kept for analysis (green arrows).

If the sequence of the restriction enzyme has not been trimmed in the demultiplexing step (for viewpoints using a primer of the first cutting site) this can be done during the following step to make sure that reads start at the fragment's end. In this example case we trim the first 4 bases of each read by setting `trim` to 4 to remove the GATC sequence of the first restriction enzyme. (For the viewpoint primer starting from the second cutting site the reads can be extended to overlap the cutting site, if the cutting site has been trimmed. In this case reads are counted with the `countFragmentOverlapsSecondCutter` function.)

Additionally we filter out read that have a mapping quality below 30 by setting `minMapq` to 30.

```
fc <- countFragmentOverlaps(fc, trim=4, minMapq=30)

## reading bam files
## calculating overlaps
```

The counts from both fragment end are added by using the function `combineFragEnds`.

```
fc <- combineFragEnds(fc)
```

We take a look at the *FourC* object and see that it now contains 3 data matrices (called "assays"): `counts`, `countsLeftFragEnd` and `countsRightFragEnd` These matrices can be accessed by the `assay` or `assays` functions.

```
fc
```

```
## class: FourC
## dim: 2 6
## metadata(7): projectPath fragmentDir ... primerFile bamFilePath
## assays(3): counts countsLeftFragmentEnd countsRightFragmentEnd
## rownames: NULL
## rowRanges metadata column names(4): leftSize rightSize leftValid rightValid
## colnames(6): testdata_WE_68h_1 testdata_WE_68h_2 ... testdata_WE_34h_1
##   testdata_WE_34h_2
## colData names(21): viewpoint condition ... mappedReads mappingRatio
```

```
assays(fc)
```

```
## List of length 3
## names(3): counts countsLeftFragmentEnd countsRightFragmentEnd
```

```
head(assay(fc, "counts"))
```

```
##      testdata_WE_68h_1 testdata_WE_68h_2 testdata_MESO_68h_1 testdata_MESO_68h_2
## [1,]                 0                13                   4                   0
## [2,]                 6                 0                   0                   2
##      testdata_WE_34h_1 testdata_WE_34h_2
## [1,]                 0                 0
## [2,]                 8                 0
```

For the rest of the vignette we now load the dataset of the "ap" viewpoint that was created for the whole chromosmes 2L and 2R of the dm3 reference genome. We adjust the project path and look at the *FourC* object.

```
data(fc)
metadata(fc)$projectPath
```

```
## [1] "pathToProjectFolder"
```

```
metadata(fc)$projectPath <- "exampleData"


fc
```

```
## class: FourC
## dim: 57253 6
## metadata(7): projectPath fragmentDir ... primerFile bamFilePath
## assays(3): counts countsLeftFragmentEnd countsRightFragmentEnd
## rownames: NULL
## rowRanges metadata column names(4): leftSize rightSize leftValid rightValid
## colnames(6): ap_WE_68h_1 ap_WE_68h_2 ... ap_WE_34h_1 ap_WE_34h_2
## colData names(21): viewpoint condition ... mappedReads mappingRatio
```

```
assays(fc)
```

```
## List of length 3
## names(3): counts countsLeftFragmentEnd countsRightFragmentEnd
```

```
head(assay(fc, "counts"))
```

```
##      ap_WE_68h_1 ap_WE_68h_2 ap_MESO_68h_1 ap_MESO_68h_2 ap_WE_34h_1 ap_WE_34h_2
## [1,]           0          13             4             0           0           0
## [2,]           6           0             0             2           8           0
## [3,]           7           0             0             0           0           0
## [4,]           0           3             0             7          19          23
## [5,]           0           4             0             0           0           0
```

```
## [6,]          35           0           0           3           8          39
```

We can see, that the dimensions of the object now represent all fragments on chromosmes 2L and 2R of the dm3 reference genome.

The content of each assay can be saved as bigWig or bedGraph files. By default the `counts` assay is exported.

```
writeTrackFiles(fc)
```

```
## [1] "Successfully created bw files of the counts data."
```

```
writeTrackFiles(fc, format='bedGraph')
```

```
## [1] "Successfully created bedGraph files of the counts data."
```

Because 4C data sometimes contains many spikes due to possible PCR artifacts, the data can be smoothed for visualization.

```
fc <- smoothCounts(fc)
```

```
## 5 chr2L
## 5 chr2R
```

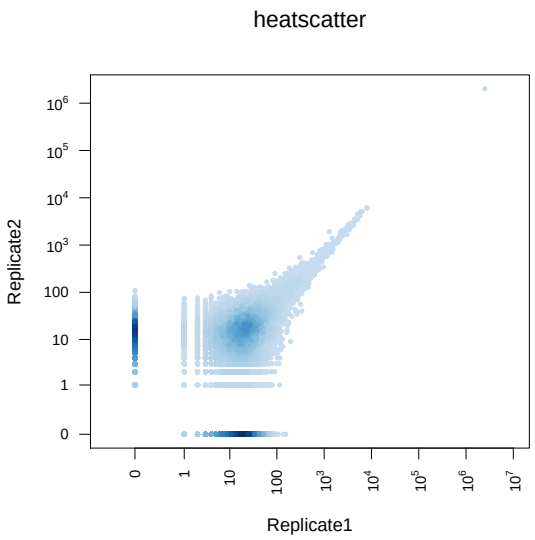
```
fc
```

```
## class: FourC
## dim: 57253 6
## metadata(7): projectPath fragmentDir ... primerFile bamFilePath
## assays(4): counts countsLeftFragmentEnd countsRightFragmentEnd counts_5
## rownames: NULL
## rowRanges metadata column names(4): leftSize rightSize leftValid rightValid
## colnames(6): ap_WE_68h_1 ap_WE_68h_2 ... ap_WE_34h_1 ap_WE_34h_2
## colData names(21): viewpoint condition ... mappedReads mappingRatio
```

We see that after the smoothing step there is a new assay, `counts_5`, of smoothed values.

Reproducibility between replicates can be assessed using a scatter plot of the count values. We therefore generate such a scatter plot for two columns of the *FourC* object.

```
plotScatter(fc[,c("ap_WE_68h_1", "ap_WE_68h_2")],
            xlab="Replicate1", ylab="Replicate2", asp=1)
```



They show good agreement for higher count values.

6 Detecting interactions

In the following step the count values are first transformed with a variance stabilizing transformation. After this step the variance between replicates no longer depends strongly on the average count value, thereby allowing a consistent statistical treatment over a wide range of count values. On these transformed counts, the general decay of the 4C signal with genomic distance from the viewpoint is fitted using a symmetric monotone fit. The residuals of the fit are used to calculate z-scores: the z-scores are the fit residuals divided by the median absolute deviation (MAD) of all the sample's residuals.

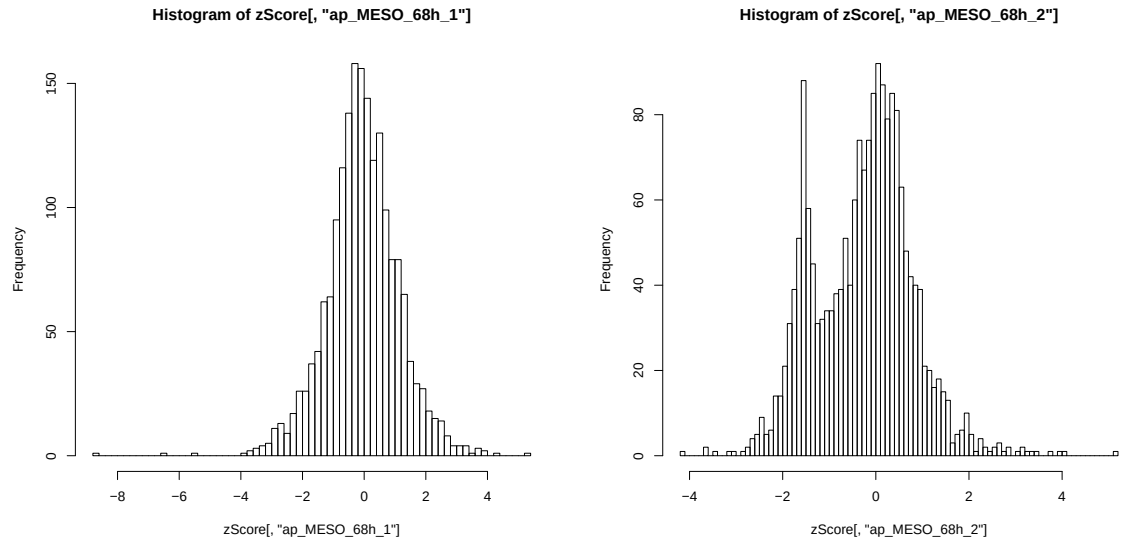
This is done the `getZScores` function. The data is filtered so that only fragments with a median count of at least 40 count are kept for the analysis. Also fragments that are close to the viewpoint, and hence show an extremely high count value are filtered out. If no minimum distance from the viewpoint is defined this distance is automatically estimated by choosing the borders of the initial signal decrease around the viewpoint. For more details and information about additional parameters that may be specified for the `getZScores` function type ?getZScores.

```
fcf <- getZScores(fc)
fcf
```

After calling getZScores, a new *FourC* object is returned that has been filtered to contain only fragments that were kept for analysis according to the above criteria. It also contains additional information added by the `getZScores` function (see ?getZScores for details).

We take a look at the distribution of z-scores, which are stored in the `assay` "zScores" of the `FourC` object.

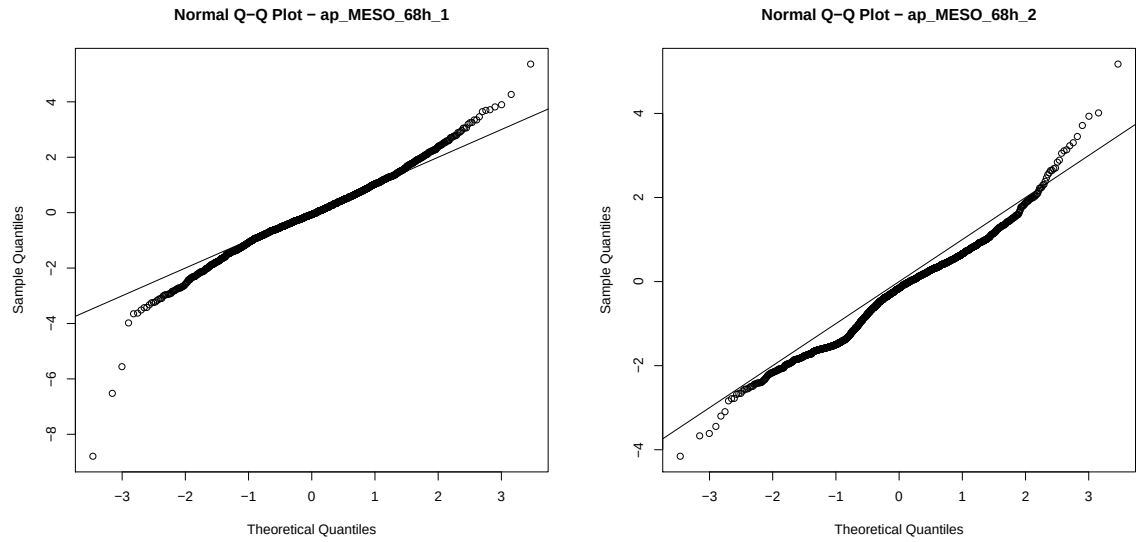
```
zScore <- assay(fcf, "zScore")

hist(zScore[,"ap_MESO_68h_1"], breaks=100)
hist(zScore[,"ap_MESO_68h_2"], breaks=100)
```



For the second replicate two peaks are observed in the histogram. The peak close to -2 is due to fragments with 0 counts in the second library, which has a lower coverage. Since we are interested in finding strong interactions on the positive side of the distribution, we can continue and capture the strongest contacts. However, if the influence of low values would further shift the distribution to negative values this might lead to errors in the calculation of z-scores. It is therefore important to check the distribution of values after calculating the z-scores.

In the next plots we check, whether normal assumption for calculating the p-values is justified.

```
qqnorm(zScore[,"ap_MESO_68h_1"],
       main="Normal Q-Q Plot - ap_MESO_68h_1")
abline(a=0, b=1)
qqnorm(zScore[,"ap_MESO_68h_2"],
       main="Normal Q-Q Plot - ap_MESO_68h_2")
abline(a=0, b=1)
```



As we see the approximation is satisfactory in general, even for the second replicate for which alredy observed deviations in the histogram.

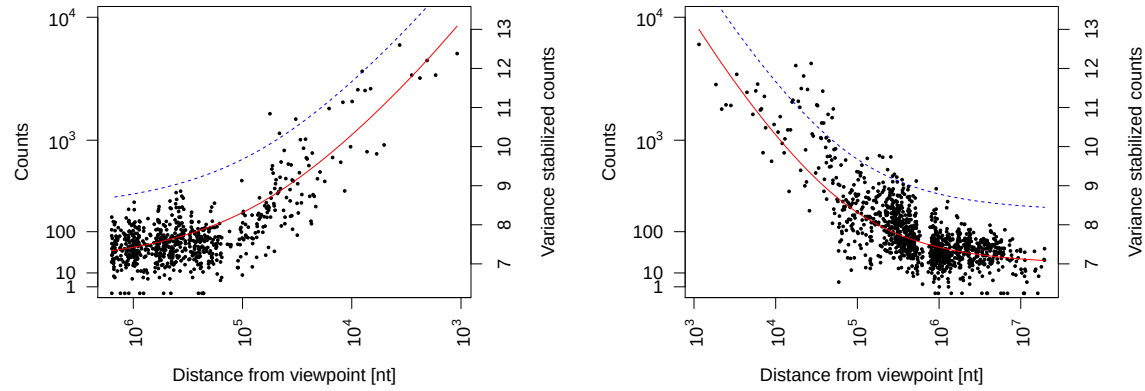
Using a conservative approach, we define interacting regions with the following thresholds: a fragment must have z-scores larger than 3 for both replicates and an adjusted p-value of 0.01 for at least one replicate. The call to `addPeaks` adds a new assay to the *FourC* object that contains booleans indicating whether an interaction has been called for a fragment or not.

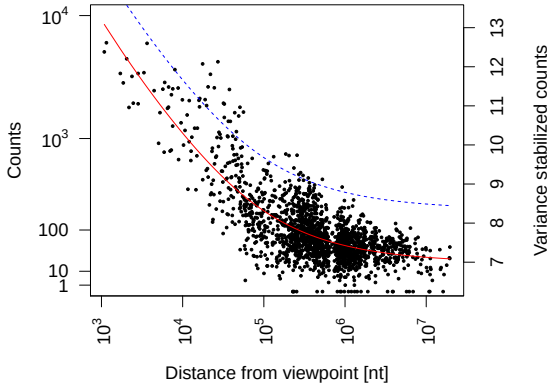
```
fcf <- addPeaks(fcf, zScoreThresh=3, fdrThresh=0.01)
head(assay(fcf, "peaks"))
```

```
##      ap_WE_68h_1 ap_WE_68h_2 ap_MESO_68h_1 ap_MESO_68h_2 ap_WE_34h_1 ap_WE_34h_2
## [1,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
## [2,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
## [3,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
## [4,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
## [5,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
## [6,]       FALSE       FALSE         FALSE         FALSE       FALSE       FALSE
```

Next we take a look at the fit for the first sample.

```
plotFits(fcf[,1], main="")
```

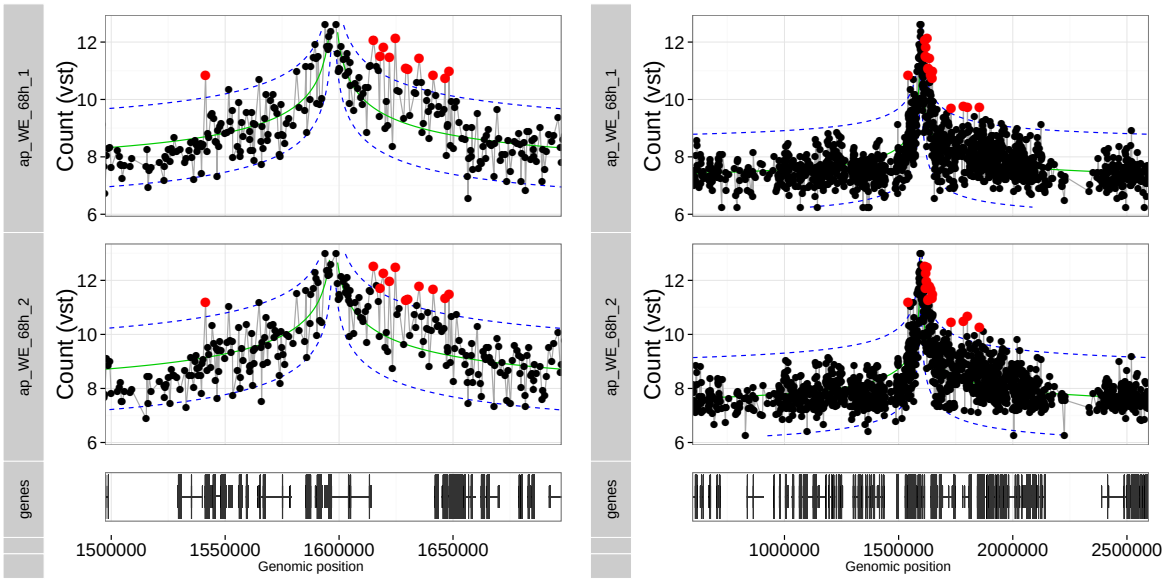




The points show the count values for the individual fragments. The red line is the fit and the blue dashed line is the fit plus (z-score threshold)*MAD for the given library, where the z-score threshold has been defined by the call to `addPeaks`. If `addPeaks` has not been called yet, a default z-score threshold of 2 is used. The first plot show the data left of the viewpoint, the second plot right of the viewpoint and for the last plot both sides have been combined by using the absolute distance from the viewpoint.

The `plotZScores` function produces plots to display the results. To include gene annotation, we load the *TxDb.Dmelanogaster.UCS* package that contains transcript information for the dm3 genome. It can be passed as an argument to the `plotZScores` function.

```
library(TxDb.Dmelanogaster.UCSC.dm3.ensGene)

plotZScores(fcf[,c("ap_WE_68h_1", "ap_WE_68h_2")],
            txdb=TxDb.Dmelanogaster.UCSC.dm3.ensGene)

## [1] "ap"
## Successfully plotted results.
```



The plot shows the results for two different window sizes around the viewpoint. The fit is shown as green line and the dashed blue lines span the interval of ± (z-score threshold)*MAD, where the z-score threshold has been defined by the call to `addPeaks`. If `addPeaks` has not been called yet, a default z-score threshold of 2 is used. Red points represent fragments that have been called as interactions.

7 Detecting differences

In addition to detecting interactions within a sample, one might be interested in finding differences of interaction frequencies between samples from different experimental conditions.

Here we show how to detect differences between conditions. In our case the conditions are whole embryo tissue at 3-4 h and 6-8 h and mesoderm specific tissue at 6-8 h. The distance dependence, which varies between viewpoints is taken into account by calculating normalizationFactors.

```
fcf <- getDifferences(fcf,
                      referenceCondition="WE_68h")
```

```
## [1] "ap"
```

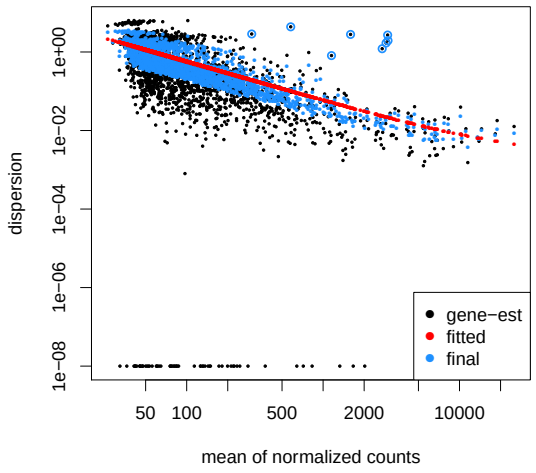
```
fcf
```

```
## class: FourC
## dim: 1872 6
## metadata(9): projectPath fragmentDir ... parameter peakParameter
## assays(13): counts countsLeftFragmentEnd ... normalizationFactors cooks
## rownames: NULL
## rowRanges metadata column names(45): leftSize rightSize ... deviance maxCooks
## colnames(6): ap_WE_68h_1 ap_WE_68h_2 ... ap_WE_34h_1 ap_WE_34h_2
## colData names(22): viewpoint condition ... mappingRatio sd
```

After calling getDifferences, the *FourC* object contains additional information about the differential test (see ?getDifferences for details.)

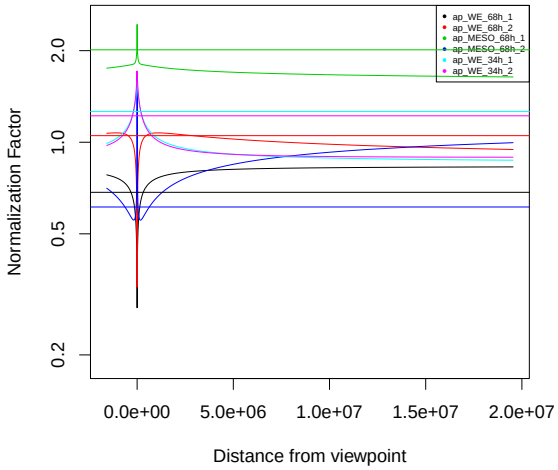
First we take a look at the dispersion fit calculated in the analysis to check if the fit worked, especially since a warning was thrown. As we can see the red fit nicely captures the trend of the black dots. The blue dots are the shrunken dispersion estimates for each fragment that are used in the differential test as measure for the variability of the data (see *DESeq2* vignette and [3] for details).

```
plotDispEsts(fcf)
```



We also take a look at the estimated values of the normalization factors plotted against the distance from the viewpoint. The horizontal lines represent the size factors of the different libraries.

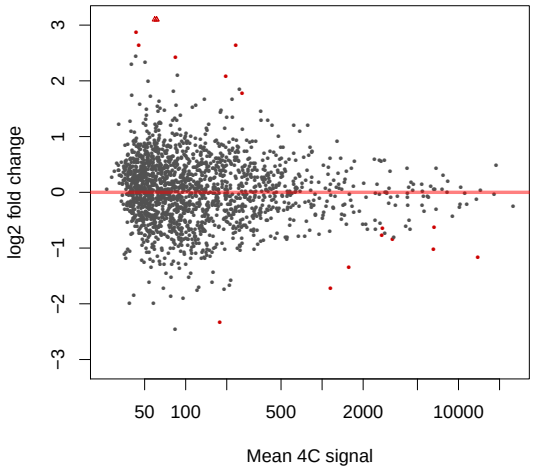
```
plotNormalizationFactors(fcf)
```



Compared to single size factors for the library size correction these values are shifted, especially in the region close to the viewpoint, where they span a range from approximately 0.3 to 3.

Next we generate an MA plot using the method from the *DESeq2* package (for details see ?plotMA and choose DESeq2). The MA plot shows the log fold changes between conditions plotted over the base mean values across samples. Red dots represent fragments with an adjusted p-value below the significance level of 0.01.

```
plotMA(results(fcf, contrast=c("condition", "WE_68h", "MESO_68h")),
       alpha=0.01,
       xlab="Mean 4C signal",
       ylab="log2 fold change",
       ylim=c(-3.1,3.1))
```



To take a look at the results of the differential test we use the `getAllResults` function.

```
results <- getAllResults(fcf)
dim(results)
```

```
## [1] 1872   16
```

```
head(results)[,1:6]
```

```
## DataFrame with 6 rows and 6 columns
##    baseMean log2FoldChange_WE_68h_MESO_68h lfcSE_WE_68h_MESO_68h stat_WE_68h_MESO_68h
##   <numeric>                      <numeric>            <numeric>            <numeric>
## 1  48.28132                      0.2502771            0.6424489            0.3895673
## 2 101.02485                      0.6539174            0.6633625            0.9857617
## 3 130.67834                      0.2192849            0.6034599            0.3633794
## 4 128.70542                      0.8007692            0.5896823            1.3579672
## 5  56.21784                      0.8261480            0.6679079            1.2369191
## 6  52.95743                     -0.6949402            0.6490725           -1.0706665
##   pvalue_WE_68h_MESO_68h padj_WE_68h_MESO_68h
##                <numeric>            <numeric>
## 1              0.6968565            0.9974311
## 2              0.3242501            0.9767801
## 3              0.7163215            0.9974311
## 4              0.1744741            0.8956729
## 5              0.2161171            0.9460951
## 6              0.2843194            0.9688240
```

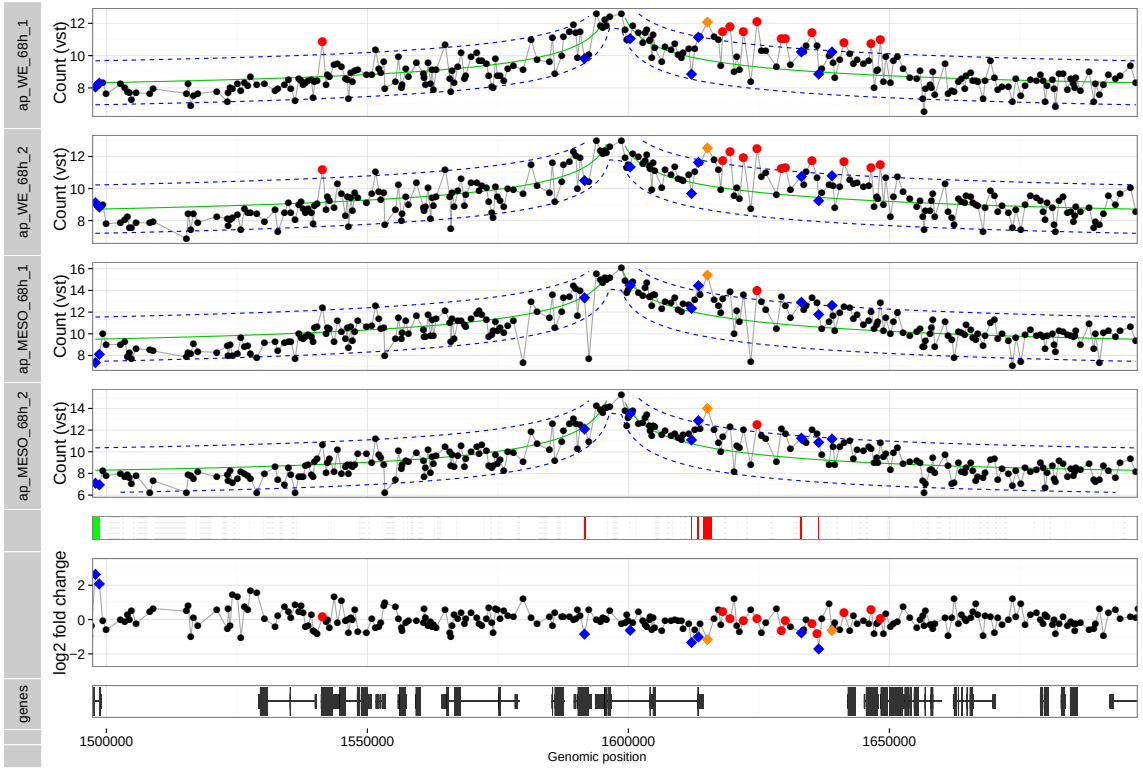
The table shows the base mean for the given fragment in the first column. Then, for every combination of conditions, 5 columns are shown. We only look at the results of the first combination by selecting the first 6 columns. The second column shows the estimated log2 fold change between the two conditions, the third the estimated standard error of the log2 fold change, the fourth the Wald test statistic, the fifth the corresponding p-value and the sixth the adjusted p-value.

The results can be visualized by creating plots with the `plotDifferences` function.

```
plotDifferences(fcf,
                txdb=TxDb.Dmelanogaster.UCSC.dm3.ensGene,
                plotWindows = 1e+05,
                textsize=16)
```

```
## [1] "ap"
```

```
## Warning:  Non Lab interpolation is deprecated
```



```
## Successfully plotted results.
```

The plot shows the results for the comparison of the two conditions. The upper two tracks show the variance stabilized counts of the first condition. The fit is shown as green line and the dashed blue lines span the interval of ± (z-score threshold)*MAD, where the z-score threshold has been defined by the call to `addPeaks`. If `addPeaks` has not been called yet, a default z-score threshold of 2 is used. Red points represent fragments that have been called as interactions, blue points represent points that show significant changes between conditions and orange points fulfill both criteria. The fifth track shows a color representation of differential interactions. A green bar means that the interaction is stronger in the first condition compared to the second and a red bar represents the opposite case. The log2 fold changes are shown also on top of a gene model track (lower panel).

We now integrate the results with known gene annotation for the apterous (ap) gene, which is the closest gene contacted by the viewpoint. We extract the log2 fold change of the signal at the ap promoter. The flybase gene id of the ap gene is "FBgn0000099". The `genes` function return a `GRanges` object with the genomic coordinates of the gene.

```
apId <- "FBgn0000099"
apGene <- genes(TxDb.Dmelanogaster.UCSC.dm3.ensGene,
                vals=list(gene_id=apId))
apGene
```

```
## GRanges object with 1 range and 1 metadata column:
##               seqnames             ranges strand |     gene_id
##                  <Rle>          <IRanges>  <Rle> | <character>
##   FBgn0000099    chr2R [1593707, 1614335]      - | FBgn0000099
##   -------
##   seqinfo: 15 sequences (1 circular) from dm3 genome
```

The `promoters` function extends the transcription start site (TSS) in both directions and returns a `GRanges` object with the resulting genomic coordinates.

```
apPromotor <- promoters(apGene, upstream = 500, downstream=100)
apPromotor
```

```
## GRanges object with 1 range and 1 metadata column:
##               seqnames             ranges strand |     gene_id
##                  <Rle>          <IRanges>  <Rle> | <character>
##   FBgn0000099    chr2R [1614236, 1614835]      - | FBgn0000099
##   -------
##   seqinfo: 15 sequences (1 circular) from dm3 genome
```

We now want to find the results for the fragment that overlaps with the ap promoter. Therefore we get the genomic coordinates of the fragments stored in the `FourC` object with `rowRanges` and find the overlap with `findOverlaps`.

```
frags <- rowRanges(fcf)

if(length(frags) != nrow(results))
  stop("Number of rows is not the same for the fragment data and results table.")

ov <- findOverlaps(apPromotor, frags)
ov
```

```
## Hits object with 1 hit and 0 metadata columns:
##       queryHits subjectHits
##       <integer>   <integer>
##   [1]         1         743
##   -------
##   queryLength: 1
##   subjectLength: 1872
```

The overlap shows which fragments (subjectHits) overlaps the ap promoter (queryHits).

Finally we look at the results of the fragment overlapping the ap promoter by subsetting results using the `subjectHits` function and look only at the first comparison of 6-8h whole embryo and mesoderm specific tissue.

```
results[subjectHits(ov),1:6]
```

```
## DataFrame with 1 row and 6 columns
##    baseMean log2FoldChange_WE_68h_MESO_68h lfcSE_WE_68h_MESO_68h stat_WE_68h_MESO_68h
##   <numeric>                      <numeric>            <numeric>           <numeric>
## 1  13899.69                      -1.168942            0.1111559           -10.51624
##   pvalue_WE_68h_MESO_68h padj_WE_68h_MESO_68h
##                <numeric>           <numeric>
## 1           7.271647e-26        1.334347e-22
```

We can see that from comparison of these conditions that there is a significant log2 fold change of -1.1689418.

8 Session Info

```
sessionInfo()
```

```
## R version 3.2.2 Patched (2015-10-08 r69496)
## Platform: x86_64-apple-darwin10.8.0 (64-bit)
## Running under: OS X 10.6.8 (Snow Leopard)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
##  [1] splines   stats4    parallel  stats     graphics  grDevices utils     datasets
##  [9] methods   base
##
## other attached packages:
##  [1] XVector_0.10.0                          TxDb.Dmelanogaster.UCSC.dm3.ensGene_3.2.2
##  [3] GenomicFeatures_1.22.0                  AnnotationDbi_1.32.0
##  [5] FourCSeq_1.4.0                          LSD_3.0
##  [7] DESeq2_1.10.0                           RcppArmadillo_0.6.100.0.0
##  [9] Rcpp_0.12.1                             SummarizedExperiment_1.0.0
## [11] Biobase_2.30.0                          ggplot2_1.0.1
## [13] GenomicRanges_1.22.0                    GenomeInfoDb_1.6.0
## [15] IRanges_2.4.0                           S4Vectors_0.8.0
## [17] BiocGenerics_0.16.0
##
## loaded via a namespace (and not attached):
##  [1] gtools_3.5.0             Formula_1.2-1            highr_0.5.1
##  [4] latticeExtra_0.6-26      RBGL_1.46.0              BSgenome_1.38.0
##  [7] Rsamtools_1.22.0         RSQLite_1.0.0            lattice_0.20-33
## [10] biovizBase_1.18.0        digest_0.6.8             RColorBrewer_1.1-2
## [13] colorspace_1.2-6         ggbio_1.18.0             Matrix_1.2-2
## [16] plyr_1.8.3               OrganismDbi_1.12.0       XML_3.98-1.3
## [19] biomaRt_2.26.0           fda_2.4.4                genefilter_1.52.0
## [22] zlibbioc_1.16.0          xtable_1.7-4             scales_0.3.0
## [25] BiocParallel_1.4.0       annotate_1.48.0          nnet_7.3-11
## [28] proto_0.3-10             survival_2.38-3          magrittr_1.5
## [31] evaluate_0.8             GGally_0.5.0             MASS_7.3-44
## [34] foreign_0.8-66           graph_1.48.0             BiocInstaller_1.20.0
## [37] tools_3.2.2              BiocStyle_1.8.0          formatR_1.2.1
## [40] stringr_1.0.0            munsell_0.4.2            locfit_1.5-9.1
## [43] cluster_2.0.3            lambda.r_1.1.7           Biostrings_2.38.0
## [46] futile.logger_1.4.1      grid_3.2.2               RCurl_1.95-4.7
```

```
## [49] dichromat_2.0-0          VariantAnnotation_1.16.0 labeling_0.3
## [52] bitops_1.0-6             codetools_0.2-14         gtable_0.1.2
## [55] DBI_0.3.1                reshape_0.8.5            reshape2_1.4.1
## [58] GenomicAlignments_1.6.0  gridExtra_2.0.0          knitr_1.11
## [61] rtracklayer_1.30.0       Hmisc_3.17-0             futile.options_1.0.0
## [64] stringi_0.5-5            geneplotter_1.48.0       rpart_4.1-10
## [67] acepack_1.3-3.3
```

References

[1] Yad Ghavi-Helm, Felix a. Klein, Tibor Pakozdi, Lucia Ciglar, Daan Noordermeer, Wolfgang Huber, and Eileen E. M. Furlong. Enhancer loops appear stable during development and are associated with paused polymerase. *Nature*, 512(7512):96–100, July 2014. URL: http://www.nature.com/doifinder/10.1038/nature13417, doi:10.1038/nature13417.

[2] Felix A. Klein, Tibor Pakozdi, Simon Anders, Yad Ghavi-Helm, Eileen E. M. Furlong, and Wolfgang Huber. Fourcseq: Analysis of 4c sequencing data. *Bioinformatics*, 2015. URL: http://bioinformatics.oxfordjournals.org/content/early/2015/05/30/bioinformatics.btv335.abstract, arXiv:http://bioinformatics.oxfordjournals.org/content/early/2015/05/30/bioinformatics.btv335.full.pdf+html, doi:10.1093/bioinformatics/btv335.

[3] Mike I. Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-Seq data with DESeq2. *bioRxiv preprint*, 2014. doi:10.1101/002832.