cobindR package vignette

October 29, 2019

Many transcription factors (TFs) regulate gene expression by binding to specific DNA motifs near genes. Often the regulation of gene expression is not only controlled by one TF, but by many TFs together, that can either interact in a cooperative manner or interfere with each other. In recent years high thoughput methods, like ChIP-Seq, have become available to produce large amounts of data, that contain potential regulatory regions. *In silico* analysis of trancription factor binding sites can help to interpret these enormous datasets in a convenient and fast way or narrow down the results to the most significant regions for further experimental studies.

cobindR provides a complete set of methods to analyse and detect pairs of TFs, including support of diverse input formats and different background models for statistical testing. Several visualization tools are implemented to ease the interpretation of the results. Here we will use a case study to demonstrate how to use cobindR and its various methods properly to detect the TF pair Sox2 and Oct4.

cobindR needs to be loaded together with the package Biostring, which provides methods for sequence manipulation.

```
> library(cobindR)
> library(Biostrings)
```

1 Configuration

Before starting the analysis, it is recommended to create a configuration file in YAML format, that contain all the parameters of the *in silico* experiment. The initial step in cobindR is to create a configuration instance.

```
> #run cobindR
> cfg <- cobindRConfiguration( fname =
+              system.file('extdata/config_default.yml',
+              package='cobindR'))
```

```
Reading the configuration file:  /tmp/RtmpNWG5Qf/Rinst79af4093ec/cobindR/extdata/config_defa
```

Parameter settings

When creating a configuration instance without a configuration file, a warning is issued. The configuration instance will then contain the default settings, that usually needs subsequent adjustment.

```
> #run cobindR
> cfg <- cobindRConfiguration()
```

The folder containing the binding motifs in PFM files has to be provided. All valid PFM files in the specified folder are loaded. Files ending with "*.pfm" or "*.cm" should be in the jaspar database format. Files ending with "*.tfpfm" need to have the Transfac database format.

```
> pfm_path(cfg) <-
+   system.file('extdata/pfms',package='cobindR')
```

The set of pairs for the co-binding analysis should be given as a list. Each pair should contain the motif names as provided in the PFM files. The order of the motif names in the pair is irrelevant.

```
> pairs(cfg) <- c('ES_Sox2_1_c1058 ES_Oct4_1_c570')
```

Alternatively, the package MotifDb can be used to retrieve the PWMs. To use MotifDb the parameter `pfm_path` should be set to 'MotifDb'. The pairs should then be given in the following format: `source:name`. E.g. JASPAR_CORE:KLF4, where JASPAR_CORE is the source and KLF4 is the transcription factor name.

```
> pfm_path(cfg) <- 'MotifDb'
> pairs(cfg) <- c('JASPAR_CORE:CREB1 JASPAR_CORE:KLF4',
+                 'JASPAR_CORE:CREB1 JASPAR_CORE:KLF4')
```

The parameters `sequence_type`, `sequence_source` and `sequence_origin` are used to configure the sequence input of the experiment. In this example `sequence_type` is set to 'fasta' to use sequences saved in fasta format. Other possibilites for `sequence_type` are "geneid" or "chipseq". In this case, where fasta is the input source, `sequence_source` should contain the path of the fasta file. Comments regarding the sequence can be written to `sequence_origin`.

```
> sequence_type(cfg) <- 'fasta'
> sequence_source(cfg) <- system.file('extdata/sox_oct_example_vignette_seqs.fasta',
+                                     package='cobindR')
> sequence_origin(cfg) <- 'Mouse Embryonic Stem Cell Example ChIP-Seq Oct4 Peak Sequences'
> species(cfg) <- 'Mus musculus'
```

When the `sequence_type` is set to "geneid" then `sequence_source` should contain the path of a file that contains a plain list of ENSEMBL gene identifiers. The parameters `downstream` and `upstream` define the downstream and

upstream region of the TSS that should be extracted. In this case mouse genes are analysed, so it is important to set the parameter `species` to "Mus musculus". If human sequences are used `species` should be set to "Homo sapiens". For other species see http://www.ensembl.org/info/about/species.html.

```
> tmp.geneid.file <- tempfile(pattern = "cobindR_sample_seq",
+                       tmpdir = tempdir(), fileext = ".txt")
> write(c('#cobindR Example Mouse Genes','ENSMUSG00000038518',
+                  'ENSMUSG00000042596','ENSMUSG00000025927'),
+                  file = tmp.geneid.file)
> species(cfg) <- 'Mus musculus'
> sequence_type(cfg) <- 'geneid'
> sequence_source(cfg) <- tmp.geneid.file
> sequence_origin(cfg) <- 'ENSEMBL genes'
> upstream(cfg) <- downstream(cfg) <- 500
```

When the `sequence_type` is set to "chipseq" then `sequence_source` should contain the path of a file in bed format. Since the sequences are obtained from the BSgenome package, the BSgenome species name together with its assembly number must be specified in the configuration value `species`.

```
> sequence_type(cfg) <- 'chipseq'
> sequence_source(cfg) <-
+   system.file('extdata/ucsc_example_ItemRGBDemo.bed',
+                                   package='cobindR')
> sequence_origin(cfg) <- 'UCSC bedfile example'
> species(cfg) <- 'BSgenome.Mmusculus.UCSC.mm9'
```

Background sequences can either be provided by the user by setting the option `bg_sequence_type` to "geneid" or "chipseq"; or artificial sequences can be generated automatically using a Markov model ("markov"), via local shuffling of the input sequences ("local") or via the program ushuffle ("ushuffle").

In the case of "local" shuffling each input sequence is divided into small windows (e.g. window of 10bp length). The shuffling is then only done within each window. That way the nucleotide composition of the foreground is locally conserved.

In order to use ushuffle (http://digital.cs.usu.edu/~mjiang/ushuffle/) it must be installed separately on your machine and be callable from the command line using "ushuffle". For this purpose download ushuffle and compile it as it is described on its website. Then rename "main.exe" to "ushuffle" and move it to your bin folder.

The options for the background generation are given in the following format: `model.k.n`, whereas model is either "markov", "local" or "ushuffle". In case of "markov" k determines the length of the markov chain, in case of "local" k determines the window length and for "ushuffle" k determines the k-let counts that should be preserved from the input sequences (see ushuffle website for more details). n determines the number of background sequences that should be created.

```
> bg_sequence_type(cfg) <-  'markov.3.200'# or 'ushuffle.3.3000' or 'local.10.1000'
```

2 Finding pairs of transcription factors

After creating a valid configuration object, a cobindr object has to be created to start the analysis of transcription factor pairs. In this example, the PWM matching functionality of the Biostrings package is applied via the function `search.pwm`. The "min.score" option is the threshold for the binding site detection (see Biostrings manual for more detail). Here the threshold is set to 80% of the highest possible score for a given PWM. `find.pairs` is then used to find pairs of binding sites.

```
> cobindR.bs <- cobindr( cfg,
+               name='cobind test using sampled sequences')
> cobindR.bs <- search.pwm(cobindR.bs, min.score = '80')
> cobindR.bs <- find.pairs(cobindR.bs, n.cpu = 3)
```

Alternatively, the RTFBS package (http://compgen.bscb.cornell.edu/rtfbs/) can be used via the function `rtfbs`. If the option "fdrThreshold" is set to a value greater than 0 ("fdrThreshold" should be between 0 and 1), the FDR thresholding approach of RTFBS is used.

```
> cobindR.bs = rtfbs(cobindR.bs)
```

Complementary to the two motif-based prediction methods, de-novo motif prediction can be performed using rGADEM. An optional p-value threshold can be provided via the configuration value "pValue".

```
> cobindR.bs = search.gadem(cobindR.bs, deNovo=TRUE)
```

In order to apply the detrending method to detect significant pairs, the background sequences need to be generated. Afterwards the binding site prediction and the pair finding also have to be performed on the background.

```
> cobindR.bs <- generate.background(cobindR.bs)
```

```
[1] "creating background sequence..."
simulating 210 background sequences using markov models with degree  3
```

```
> cobindR.bs <- search.pwm(cobindR.bs, min.score='80',
+                          background_scan = TRUE)
```

```
finding binding sites for 4 PWMs...
finding hits for PWM ES_Sox2_1_c1058 ...
finding hits for PWM ES_Klf4_3_c1373 ...
finding hits for PWM ES_Oct4_1_c570 ...
finding hits for PWM ES_Sox2_1_c1058 ...
```

```
found  12 hits for PWM ES_Klf4_3_c1373 ...
found  38 hits for PWM ES_Oct4_1_c570 ...
found  226 hits for PWM ES_Sox2_1_c1058 ...
found  226 hits for PWM ES_Sox2_1_c1058 ...
found 502 hits in total.
```

```
> cobindR.bs <- find.pairs(cobindR.bs, background_scan = TRUE,
+                          n.cpu = 3)
```

```
Searching for pairs...
Searching for pair ES_Oct4_1_c570 ES_Sox2_1_c1058 .
Found 16 pairs.
Time difference of 0.2102537 secs
```

3 Results: Statistics and Visualizations

Several visualization methods are available in cobindR. For instance the input sequences can be analysed if subgroups exist that have different nucleotide compositions. A two-dimensional plot is created where each sequence's GC-content is scattered against its CpG-content (Fig. 1). A model-based clustering analysis is performed and if subgroups are detected in the plot, it is suggested to analyse them seperately.

```
> tmp <- testCpG(cobindR.bs, do.plot=T)
```

Furthermore, the GC or CpG content can be spatially analysed for each sequence. Since the calculation is slow, the resulting figure is not included here.

```
> plot.gc(cobindR.bs, wind.size=200, frac = 2)
```

The sequence logo of the predicted binding sites can be easily obtained. After normalizing the column sums, the matrices can be visualized via the `seqLogo` package (Fig. 2).

```
> pred.motifs <- predicted2pwm(cobindR.bs, as.pfm=TRUE)
> # normalized column sums as required by seqLogo
> pred.norm.motifs <- lapply(pred.motifs, function(x) x / colSums(x)[1])
> # load sequence logo plot function
> plot.tfbslogo(x=cobindR.bs,c('ES_Sox2_1_c1058','ES_Oct4_1_c570'))
```

To obtain a quick overview of the spatial distribution of the predicted binding sites for all input sequences, `plot.positionprofile` can be used to get a plot of the average number of binding sites relative to the position (Fig. 3).

```
> plot.positionprofile(cobindR.bs)
```

`plot.positions.simple` provides an overview for all binding sites and all input sequences. Binding sites are visualized as dots at their position along the x-axis for the corresponding input sequence along the y-axis (Fig. 4).

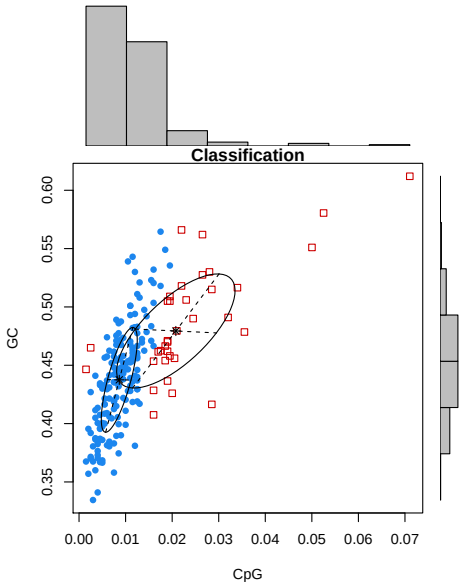

Figure 1: **GC-CpG plot for all input sequences.**

```
> plot.positions.simple(cobindR.bs)
```

The observed frequencies of two motifs occurring in the same sequence is visualized as a heatmap using the function `plot.tfbs.heatmap` (Fig. 5). A p-value is assigned to each of the motif combinations using a hypergeometric test. Overlaps with $p < 0.05$ and $p < 0.01$ are marked with '*' and '**', respectively.

```
> plot.tfbs.heatmap(cobindR.bs, include.empty.seqs=FALSE)
```

Using the function `plot.tfbs.venndiagram` a Venn diagram is created that visualizes the relationship between multiple motifs. As there are only two motifs used in this example, it does not yield additional information (Fig. 6).

```
> plot.tfbs.venndiagram(cobindR.bs, pwms = c('ES_Sox2_1_c1058', 'ES_Oct4_1_c570'), include.e
```

The distribution of observed distances between two motifs over all input sequences is available via `plot.pairdistance` (Fig. 7).

```
> plot.pairdistance(cobindR.bs, pwm1='ES_Sox2_1_c1058',
+                   pwm2='ES_Oct4_1_c570')
```

Using the function `plot.pairdistribution` one can visually check whether the pair of two motifs are found in all input sequences or whether there is a subpopulation of pair-rich or -poor sequences (Fig. 8).

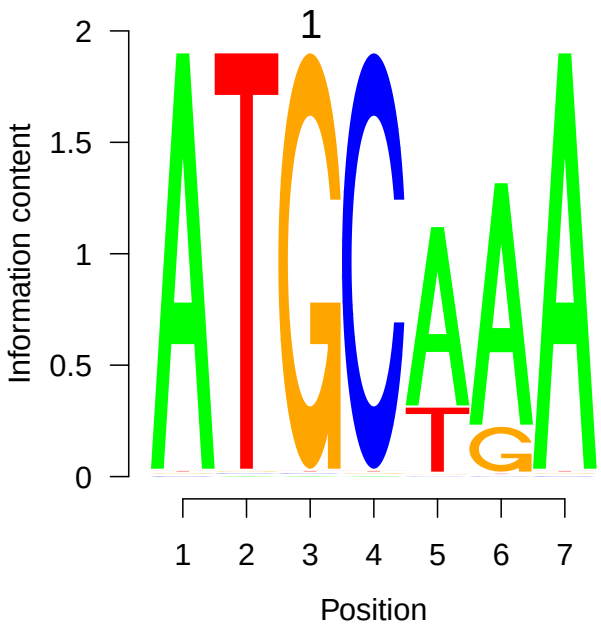

Figure 2: **Sequence Logo.**

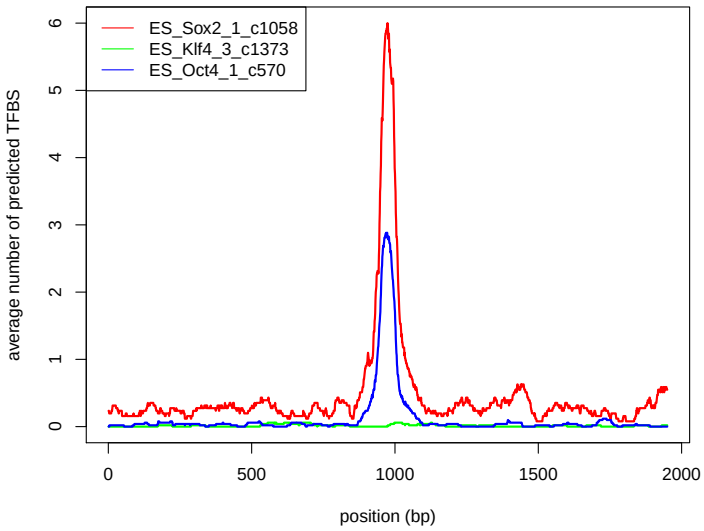

Figure 3: **Position profile for each PWM.**

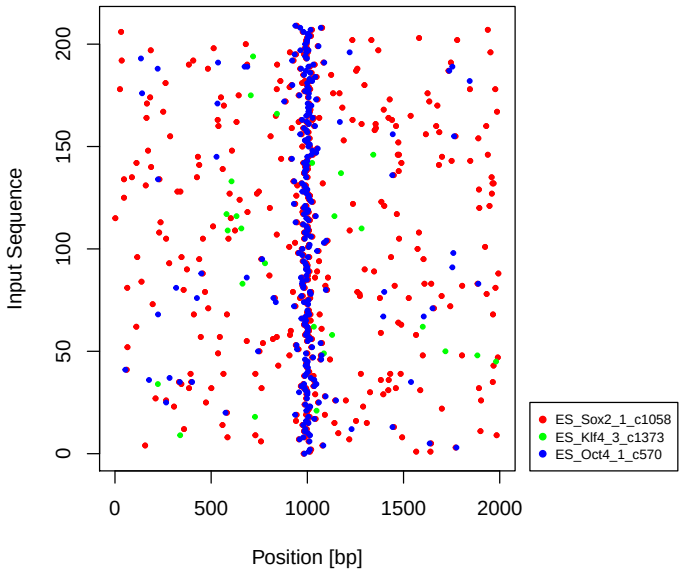

Figure 4: **Position of all binding sites for all input sequences.**

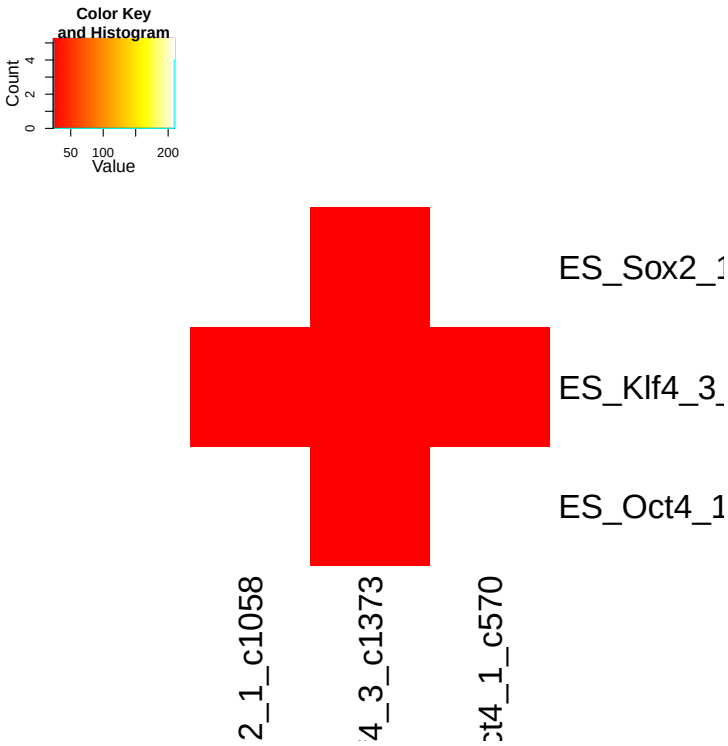

Figure 5: **Heatmap of pairwise co-occurring motifs in same sequences.**

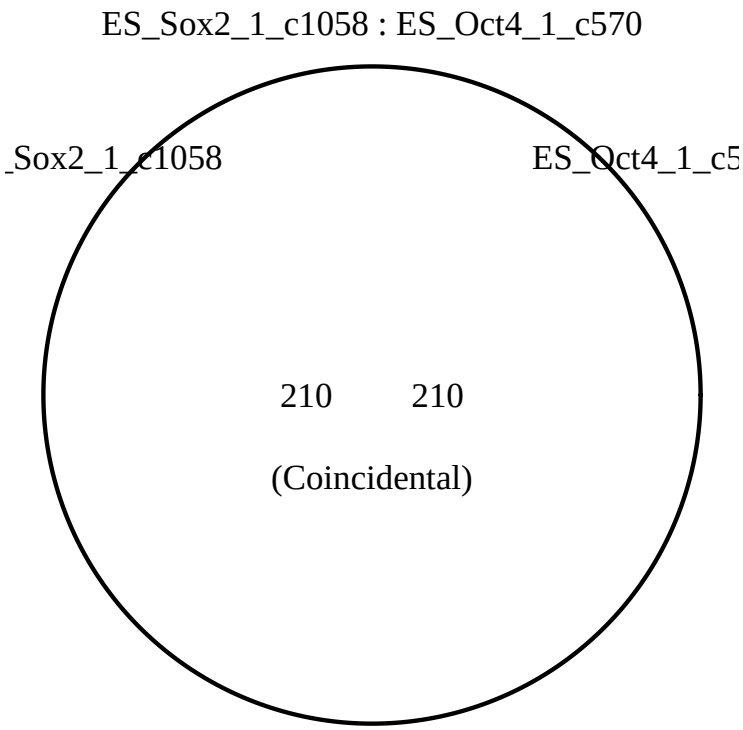

Figure 6: **Venn diagram of multiple co-occurring motifs in same sequences.**

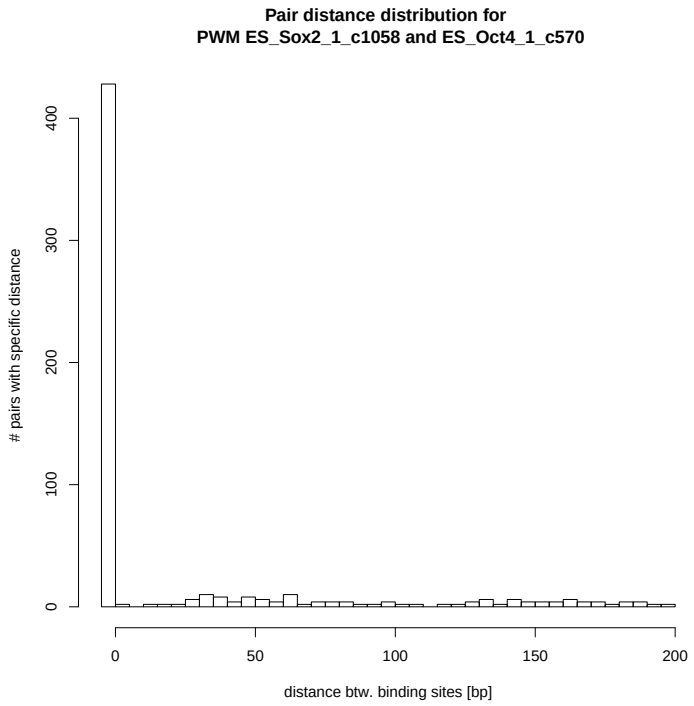

Figure 7: **Distribution of observed distances for one motif pair.**

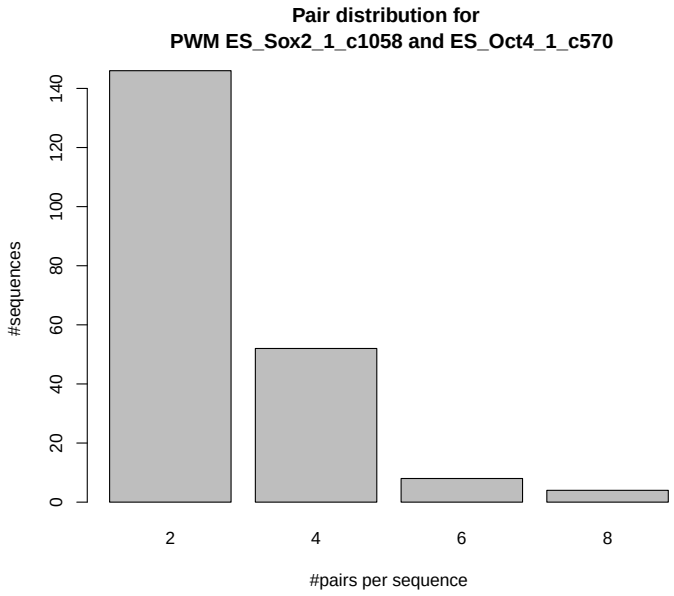

Figure 8: **Distribution of found pairs per sequence.**

```
> plot.pairdistribution(cobindR.bs, pwm1='ES_Sox2_1_c1058',
+                       pwm2='ES_Oct4_1_c570')
```

Detecting significant TF pairs with a certain distance

The distribution of distances between two motifs (Fig. 9 top left) in combination with the results from the similar procedure applied to the background sequences (top right) is available via `plot.detrending`. Furthermore, the foreground and background distance distributions are combined via the detrending procedure (bottom left) and the resulting distance profile is shown with the corresponding significance level (bottom right).

```
> plot.detrending(cobindR.bs, pwm1='ES_Sox2_1_c1058',
+                                 pwm2='ES_Oct4_1_c570', bin_length=10, abs.distance=FALSE,
```

The locations and sequences of the overrepresented pairs can be exported into a plain text file,

```
> tmp.sig.pairs = get.significant.pairs(x = cobindR.bs, pwm1='ES_Sox2_1_c1058',pwm2='ES_Oct4

Found candidate pair in FOREGROUND ES_Sox2_1_c1058 ES_Oct4_1_c570 in distance -1 - -10 bp. Z
```

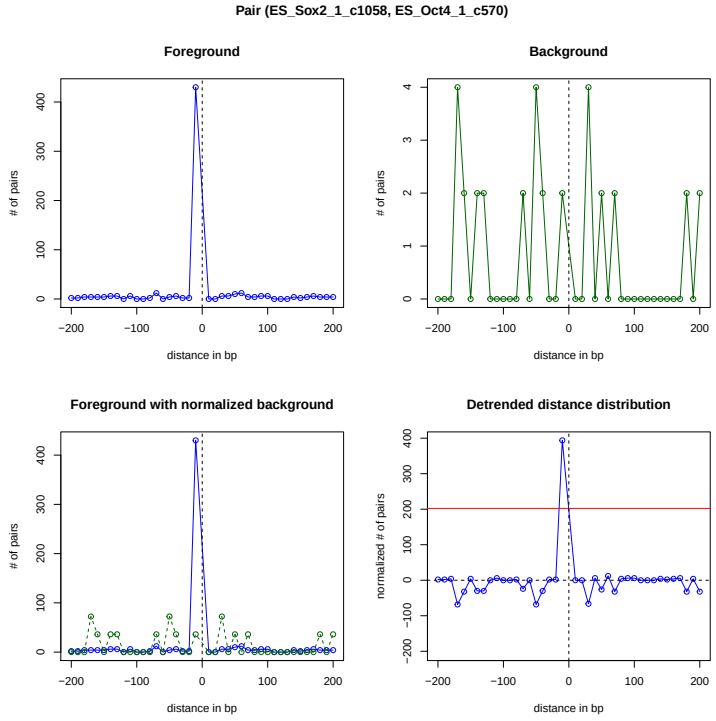

Figure 9: **Using the detrending approach a significant distance is detected for the Sox2-Oct4 pair (bottom right).**

```
> tmp.resultbs.file <- tempfile(pattern = "cobindR_detrening_result_bindingsites", tmpdir =
> write.table(tmp.sig.pairs[[1]], file=tmp.resultbs.file, sep="\t", quote=F)
> system(paste('head',tmp.resultbs.file))
> tmp.resultcp.file = gsub("bindingsites","candidates_pairs", tmp.resultbs.file)
> write.table(tmp.sig.pairs[[2]], file=tmp.resultcp.file, sep="\t", quote=F)
> system(paste('head',tmp.resultcp.file))
```

as well as the complete set of predicted binding sites.

```
> tmp.result.bs.file <- tempfile(pattern = "cobindR_bindingsite_pred",
+                       tmpdir = tempdir(), fileext = ".txt")
> write.bindingsites(cobindR.bs, file=tmp.result.bs.file, background=FALSE)

[1] "wrote binding sites to: /tmp/RtmpLAMJrd/cobindR_bindingsite_pred10bb10ae66bc.txt"

> system(paste('head',tmp.result.bs.file))
```

The foreground and background sequences can be obtained from the cobindR object for further analysis.

```
> tmp.inseq.file <- tempfile(pattern = "cobindR_input_sequences",
+                       tmpdir = tempdir(), fileext = ".fasta")
> # slotname = 'bg_sequences' to obtain the background sequences
> write.sequences(cobindR.bs, file=tmp.inseq.file,
+                                  slotname= "sequences")
> #system(paste('head',tmp.inseq.file,'n=10'))
```

Clean up the input sequence files.

```
> try(unlink(tmp.result.file))
> try(unlink(tmp.result.bs.file))
> try(unlink(tmp.inseq.file))
```