Supplementary Material

Pyicos: A versatile toolkit for the analysis of high-throughput sequencing data

Sonja Althammer[1,‡], Juan González-Vallinas[1,‡], Cecilia Ballaré[2], Miguel Beato[1,2], Eduardo Eyras[1,3,*]

[1]Universitat Pompeu Fabra. Dr. Aiguader 88, E08003, Barcelona, Spain

[2]Centre for Genomic Regulation (CRG). Dr. Aiguader 88, E08003 Barcelona, Spain

[3]Catalan Institution for Research and Advanced Studies (ICREA), Passeig Lluís Companys 23, E08010, Barcelona, Spain

[‡] These authors contributed equally

[*] Corresponding author

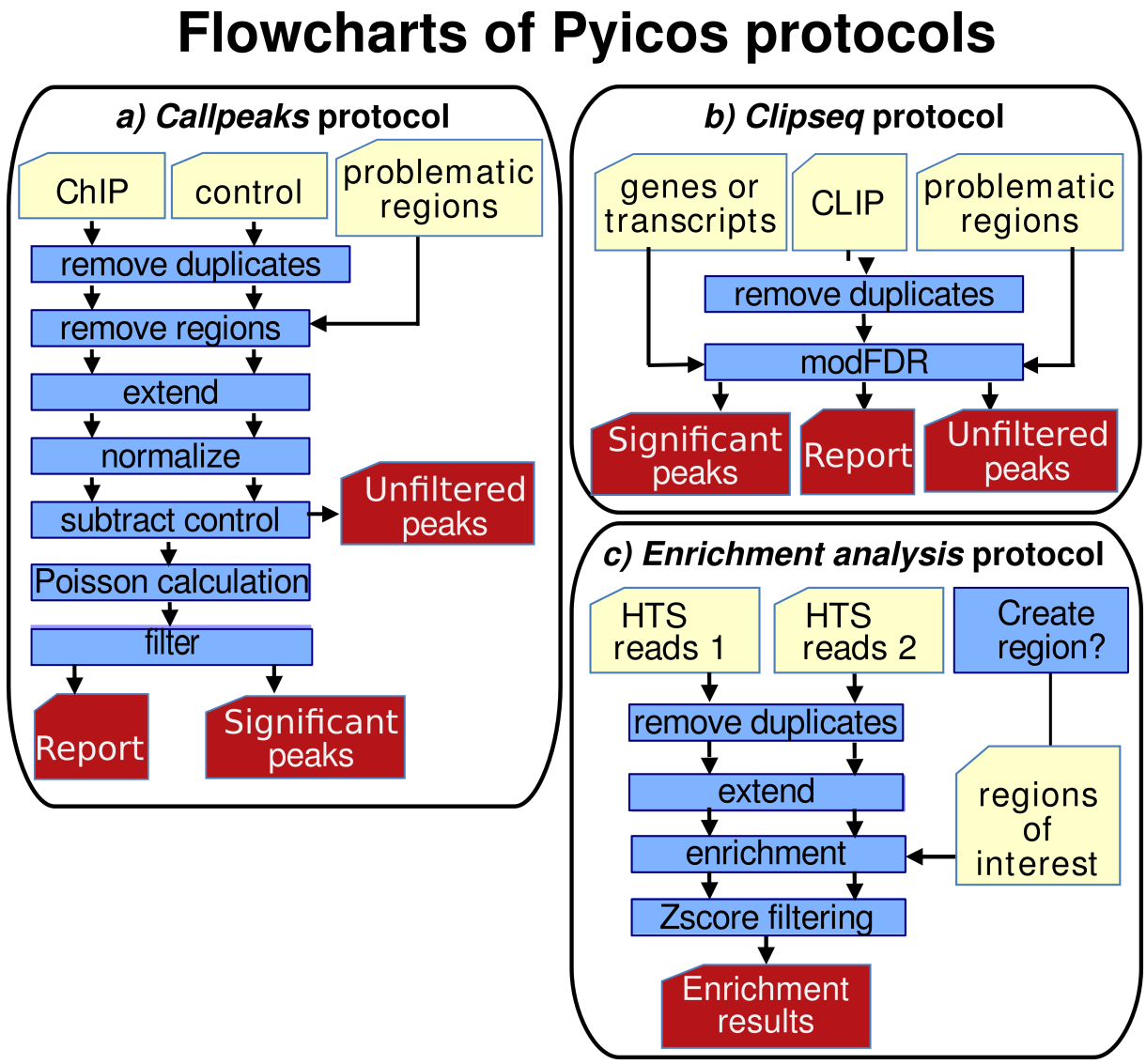


Figure S1. Flowcharts of Pyicos protocols. We propose three protocols for the analysis of high-throughput sequencing data: **a)** *Callpeaks* protocol for genome-wide peak calling using punctuated ChIP-Seq data. **b)** *Clipseq* to detect significant clusters within a gene or transcript, for CLIP-Seq datasets. **c)** *Enrichment analysis*, to detect regions with significant changes in read density between two conditions. Alternatively, Pyicos architecture allows easy creation of new protocols using the already existing operations.

Protocol files

Pyicos protocol files follow the INI file format (http://en.wikipedia.org/wiki/INI_file). In order to be identified as a Pyicos Protocol file, all Pyicos Protocols should start with a header line that reads [Pyicotrocol]. Semicolons (;) indicate the start of a comment. Everything between the semicolon and the end of the line is ignored. The operations to execute should be specified in the operation line, separated by commas. For instance: operations=extend, subtract, normalize. We provide below example protocol files for Chip-Seq punctuated data (Callpeaks.ptcl), CLIP-Seq data (Clipseq.ptcl) and for enrichment analysis of RNA-Seq or ChIP-Seq data (Enrichment.ptcl)

Callpeaks.ptcl

```
[Pyicotrocol]

; This is a comment

experiment         = kidney1.bed
control            = liver1.bed
region             = blacklist.bed

experiment_format   = bed
control_format      = bed
region_format       = bed

open_experiment     = true
open_control        = true
open_region         = true

output             = significant_peaks.bedpk

operations         = remove_duplicates, remove, extend, normalize, subtract, trim, poisson, filter, split

duplicates=0           ; Number of duplicates that are tolerated
frag_size=150          ; Estimated fragment size
correction=0.8         ; Fraction of the genome that is mappable
trim_proportion=0.1    ; Fraction of the cluster height below which the peak is trimmed
split_proportion=0.1   ; Fraction of the lower maximum; if the read coverage between two maxima falls
below it the peak will be split
height_limit=100       ; After this value the poisson calculation will not assign lower p-values to the peaks
anymore
poisson_test=height    ; alternatively you can use numtags for broad peaks
```

Clipseq.ptcl

```
[Pyicotrocol]

; This is a comment

experiment=kidney1.bed
output=peaks_called.pk

experiment_format=bed
output_format=bed_spk
; stranded format is recommended for clipseq experiment analysis in order to merge overlapping reads with the same orientation in a cluster.
; ! Still clusters made up of either plus or minus reads are detected in a region independet of its strand !

region=regions.bed
; if the region is not provided, regions of interest will be infered from the data.
; Required BED format.


operations=  modfdr
```

Enrichment.ptcl

```
[Pyicotrocol]

; This is a comment

experiment=liver1.bed       ; experiment and experiment_b are the conditions / cell lines you want to compare
experiment_b=kidney2.bed
replica_a=liver2.bed        ; if replica_a is not provided, a theoretical background model will be calculated

output=result_counts.bed
; output_format not necessary here,
; the enrichment test always outputs a BED like counts file

experiment_format=bed

region=regions.bed
; if the region is not provided, regions of interest will be inferred from the data.
; Required BED format.

operations= enrichment, zscore, plot
```

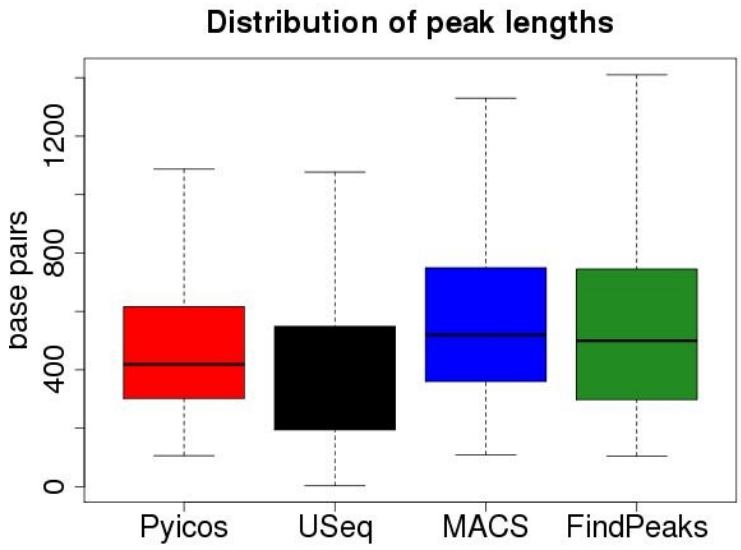


Figure S2. Length distribution of the top 3000 NRSF peaks selected by each method. The peaks were ranked for each method as described in Methods. Pyicos peaks were ranked according to the peak read-count score.

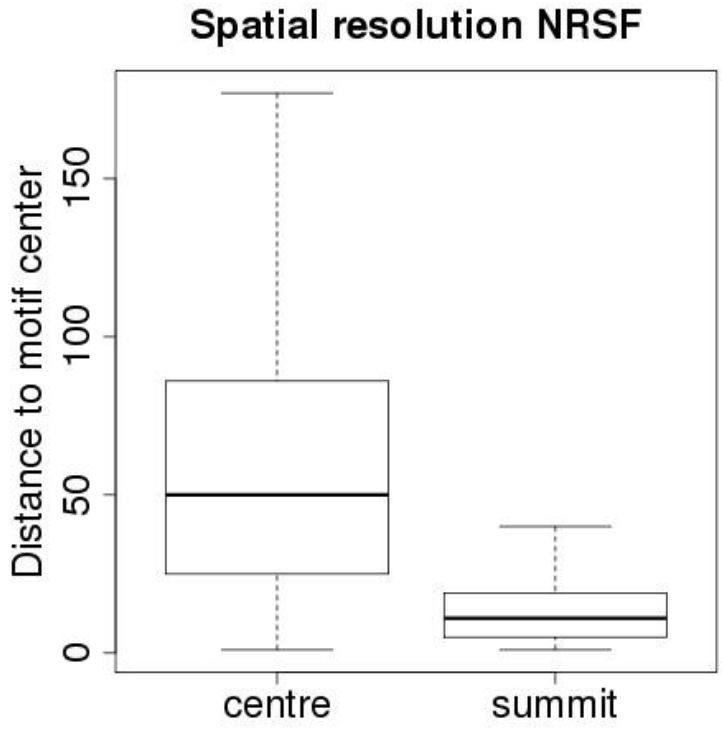


Figure S3. Distance of peak centre and summit from motif centre. We compared the distributions of distances between the motif centre and the peak-centre (left boxplot), and between the motif centre and the peak summit (right boxplot). Peak summit is defined as the point with highest read pile-up. The motif centre was measured as the midpoint of the motif found in the peak. In this calculation the NRSF ChIP-Seq data and the Pyicos predictions were used.

Enrichment analysis

The enrichment analysis is used to detect regions that are enriched in signal between two conditions, or cell-lines. We calculate a Z-Score as a measure of the significance of this enrichment, based on the assumption that cases of non-significant enrichment between two conditions *1* and *2* will appear with a similar distribution as the enrichment between the sample *1* and its replicate *1'*. If no such replicate *1'* is provided, we create two theoretical replicates, *1'* and *2',* with the same sizes as *1* and *2,* by mixing the reads from *1* and *2* randomly and assigning them with to 1' and 2' with either equal probability or with probability proportional to their relative sizes (this is an option in *Pyicos*). Differences between these theoretical replicates are considered to be non-significant. Accordingly, the Z-Score is calculated for each of the M and A values of the comparison between the signals C_1 and C_2 , from conditions 1 and 2, respectively:

$M = \log_2(\frac{C_1}{C_2})$, and

$A = \frac{1}{2}\log_2(C_1 \cdot C_2)$

where the signals C are expressed in terms of the read-count, the RPKM (Reads per kilobase per Million of mapped reads), the TMM normalized read-count (Robinson and Oshlack 2010), or the TRPK density as defined below.

The Z-Score of a region r is calculated based on the distribution of M values for the replicas within a window of size S_w on the A axis. This window is assigned to a region according to the proximity of the A values: given the A value of the region $A(r)$, we assign *r* to the window that has a mean A value closest to $A(r)$. This window describes the null distribution of M values (also referred to as background) against which we compare the observed M value for *r*. If we define as μ_{MB} the mean and as σ_{MB} the standard deviation of the M values from the background window *B*, and $M(r)$ is the M value for region r under conditions 1 and 2, the Z-Score of r is calculated as follows:

$$Z\,score(r) = \frac{M(r) - \mu_{MB}}{\sigma_{MB}}$$

The resolution can be adjusted by changing the window size and the window step through the parameters `binsize` and `binstep`, respectively. By default, the `binsize` is set to 0.3 and `binstep` is 3, which means that each window contains 30% of the regions, and the difference between one window and the next is just 3 data points. The `binstep` is the resolution at which the window slides through the background data, i.e. at every background window we remove three points and add three points according their ordering along the A axis. We compared the performance of Pyicos on the benchmarking microarray dataset using different parameters of `binsize` and `binstep` and found that the highest AUC is achieved with a the values that we use as default.

As mentioned above, the signal can be expressed in terms of the read-count, the RPKM (Reads per kilobase per Million of mapped reads), the TMM normalized read-count (Robinson and Oshlack, 2010), or the TRPK density. The RPKM density is defined as (Mortazavi et al., 2006):

$$RPKM = 10^9 \frac{n(S,r)}{N_S \cdot L_r}$$

Where $n(S,r)$ is the read count from sample *S* that fall in region r , N_S is the total number of reads in sample S and L_r is the length of the region *r*. Alternatively, one can define the area described by the reads in a given region and redefined the signal as a read-density per kilobase and per million of units of read density (RdPKM):

$$RdPKM = 10^9 \frac{a(S,r)}{A_S \cdot L_r}$$

Where $a(S,r)$ is the area defined by the reads from sample S that fall within region r , A_S s the total area of the reads in sample S and L_r is the length of the region *r*. The areas are defined by the number of bases occupied by a read. The RdPKM measure is convenient when the reads have different lengths or when the reads do not fall entirely in the selected regions.

The Trimmed Mean of M-values (TMM) normalization factor for counts was proposed in (Robinson and Oshlack 2010). We implemented the calculation of the TMM normalization factor, trimming a fraction of the regions with the highest absolute M values, i.e. regions with great difference in the number of reads between samples, and discarding a fraction of the lowest A values, i.e. regions with very low average numbers of reads from both samples. The fraction of discarded regions are parameters that can be specified by the user. The values used for the enrichment analysis liver vs. kidney are mentioned below. The normalization factor TMM is calculated as follows:

$$\log_2(TMM) = \frac{\sum_{r \in R} w_r M'(r)}{\sum_{r \in R} w_r}$$

where R is the set of regions analysed after trimming for extreme M and A values; and where the weight factors w_r are defined as:

$$w_r = \frac{N_1 - n(1,r)}{N_1 \cdot n(1,r)} + \frac{N_2 - n(2,r)}{N_2 \cdot n(2,r)}; n(1,r), n(2,r) > 0$$

with $n(1,r)$ and $n(2,r)$ the read counts that fall in region r from sample 1 *and 2*, respectively, with $n(1,r), n(2,r) > 0$; and where N_1 and N_2 are the total number of reads in sample 1 and 2, respectively. Moreover, the modified M value $M'(r)$ is:

$$M'(r) = \log_2 \frac{\frac{n(1,r)}{N_1}}{\frac{n(2,r)}{N_2}}$$

We combined the TMM normalization with the normalization by region length:

$$TRPK = 10^3 \frac{TMM \cdot n(S,r)}{L_r}$$

Where TMM is the normalization factor, $n(S,r)$ is the read count from sample S within region r and L_r is the length of the region r .

Validation of the MA calculation

In order to test tour implementation of the differential expression analysis, we decided to try to replicate the same results as using edgeR with the TMM normalization. We used the same read-count table from (Robinson and Oshlack, 2010) with data from (Marioni et al. 2008) and applied *Pyicos* EA and edgeR, both with the TMM normalization. Our implementation, gives a TMM factor of 0.66 using the trimming proportions of M_trim = 0.25 and A_trim = 0.05. Moreover, the calculated M and A values for each gene after applying the TMM normalization from either method were identical.

Normality test on the MA plots

When calculating the enriched regions from the M-A distributions, we calculate a z-score locally per subset of background data points within a sliding window of A. This relies in the assumption that the distribution of M values within this window are normally distributed. We verified this by using the D'Agostino's K-squared test of normality on the sliding windows. Starting from a distribution of M-A values for 16,915 regions, we used sub-windows including 846 regions at every step, i.e. 5%. Moreover, at every step, we slid the window one region at a time according to the order of regions along the A axis from left to right. We then observed that all distributions of M contained in a window were normal (p-value >0.01) (Supplementary Image S4). We also observed that the bigger the value of A, the lower the p-value of the normality test, which is due to the decreasing dispersion of the M distribution in the windows for large values of A.

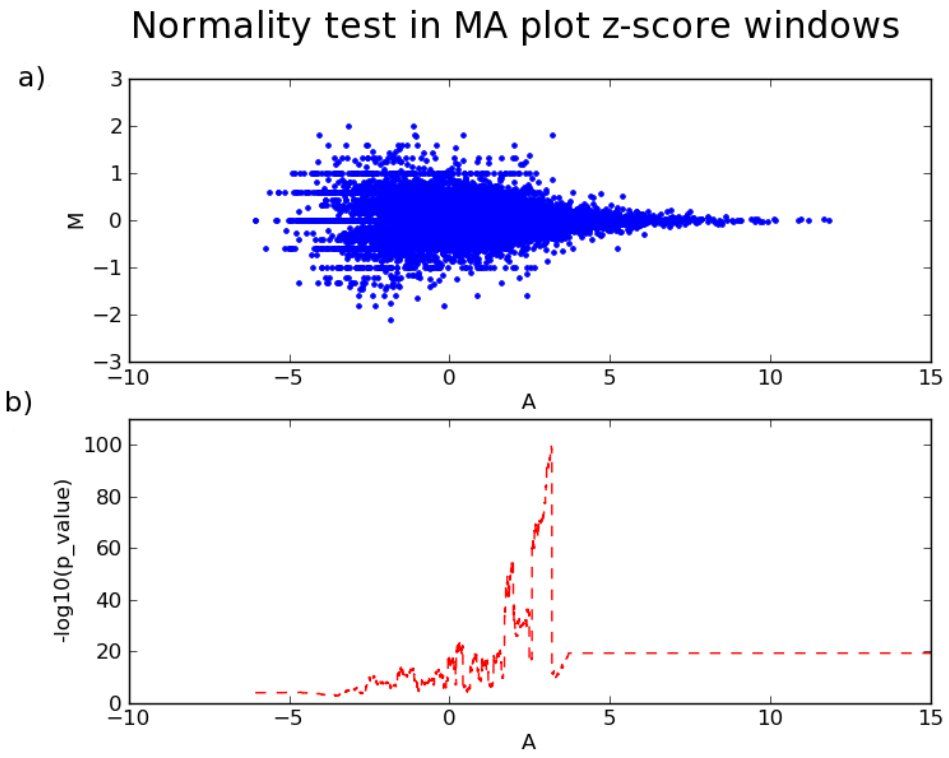


Figure S4. Normality test on the enrichment windows. a) The MA plot for 2 of the liver replica samples from (Marioni et al., 2008). **b)** Distribution of $-\log_{10}$(p-value) for the sliding windows of data points along the A axis using the D'Agostino's K-squared test of normality. All p-values are above 0.01; hence the distribution in each window is normal.

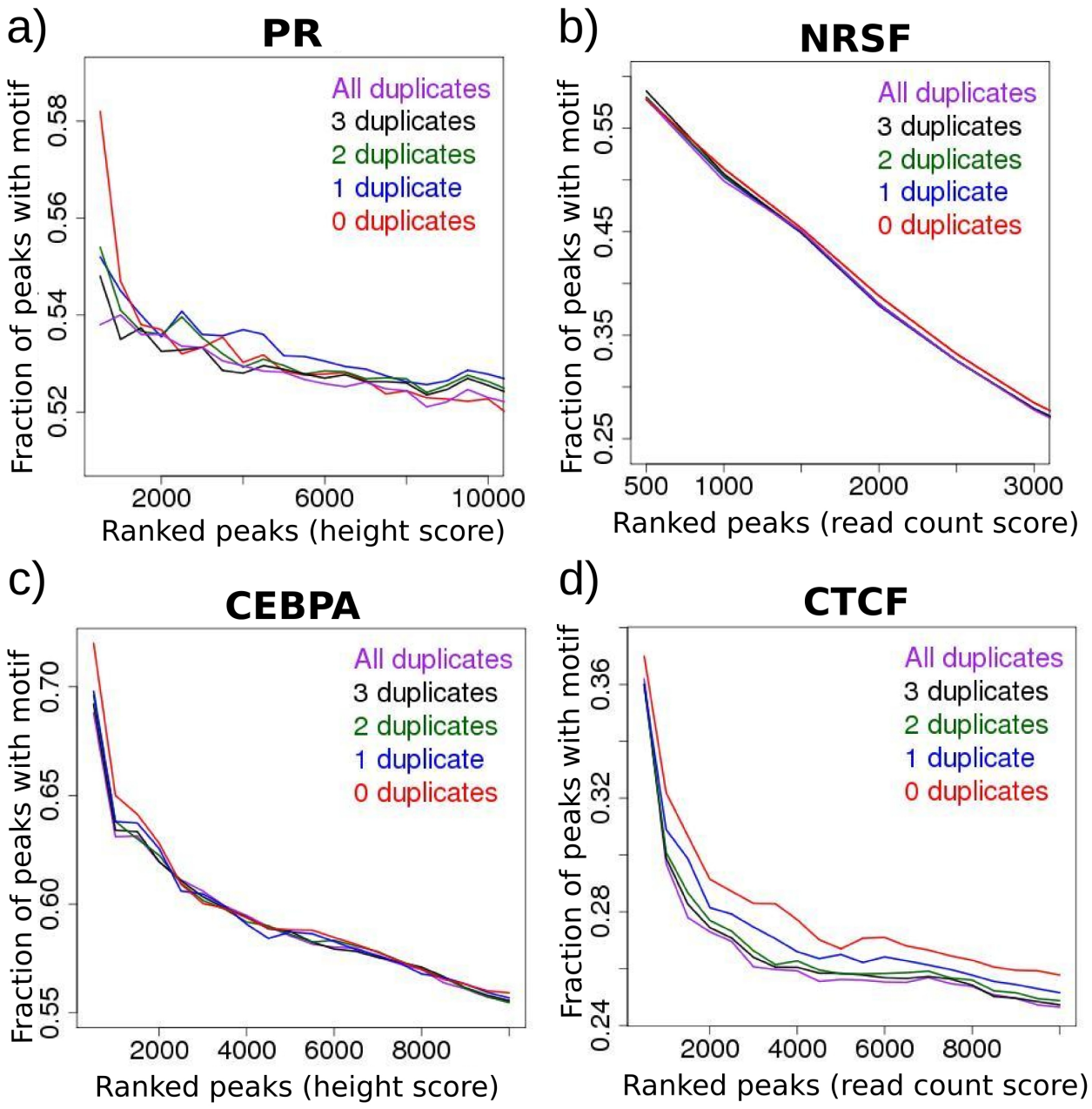


Figure S5. Results of Pyicos protocol *callpeaks* with different numbers of tolerated duplicates on **a)** PR **b)** NRSF **c)** CEBPA and **d)** CTCF ChIP-Seq data. To rank the peaks along the x axis, we use the peak height score for PR and CTCF and the peak read-count score for NRSF and CEBPA, as these are the scores the give best peak detection (see Supplementary Figure S10) and Methods. The y-axis shows the fraction of peaks with the corresponding motif, calculated as explained in Methods.

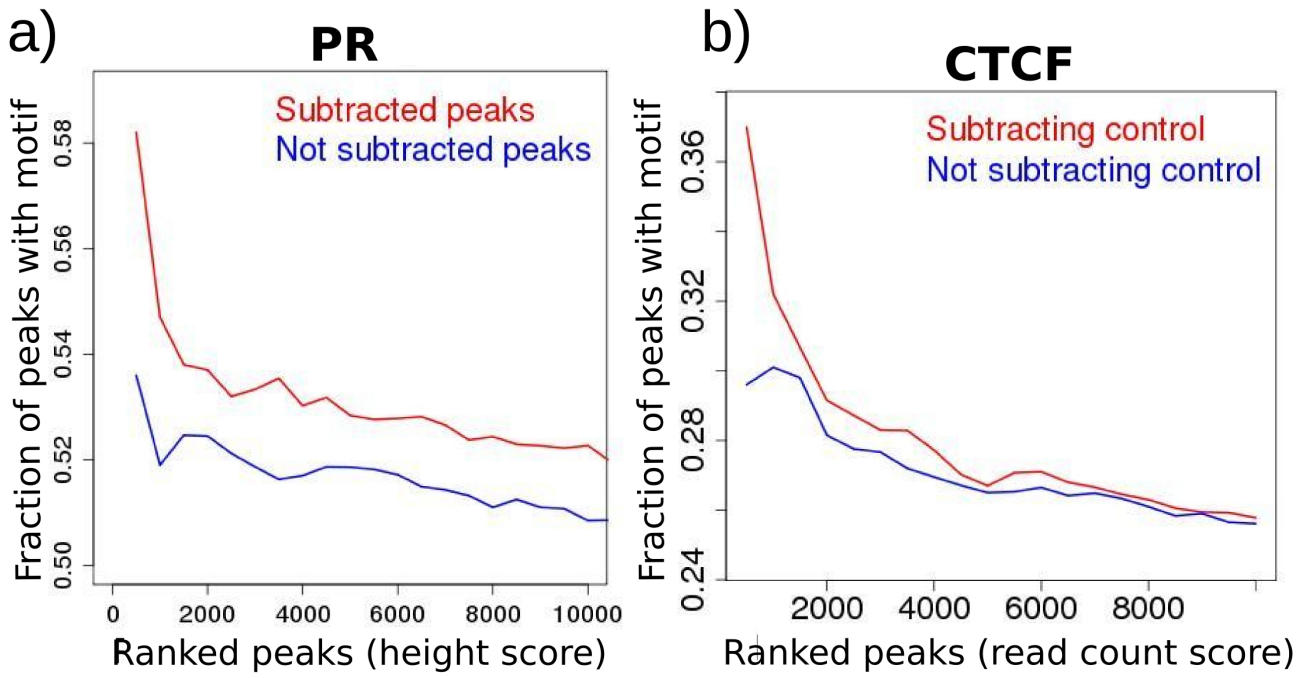


Figure S6. Results of Pyicos protocol callpeaks subtracting or not subtracting the control on **a)** PR and **b)** CTCF. On the x-axes we ranked the peaks according to the appropriate score for the dataset (height score or read count score as in Methods) (see also Supplementary Figure S10). The y-axes show the fraction of peaks with the corresponding motif, calculated as explained in Methods.

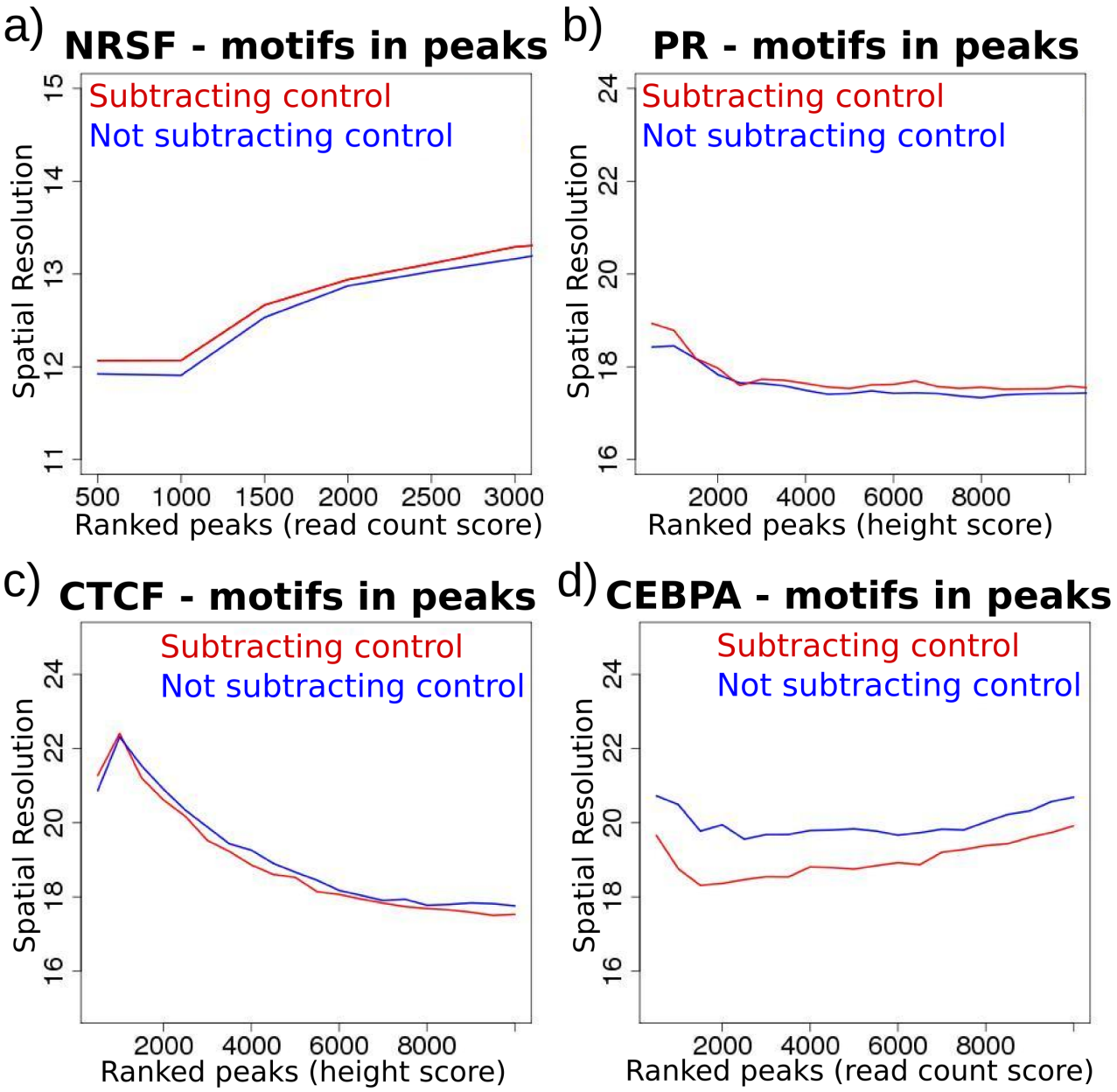


Figure S7. Maintenance of spatial resolution after subtraction on **a)** NRSF **b)** PR **c)** CTCF and **d)** CEBPA ChIP-Seq data. Pyicos protocol callpeaks was run with and without subtracting the control. Spatial resolution is defined as the distance between motif centre and peak summit. On the x-axes we ranked the peaks according to the appropriate score for each dataset as explained in Methods (see also Supplementary Figure S10). The y-axes show the fraction of peaks with the corresponding motif, calculated as explained in Methods.

a) **b)**

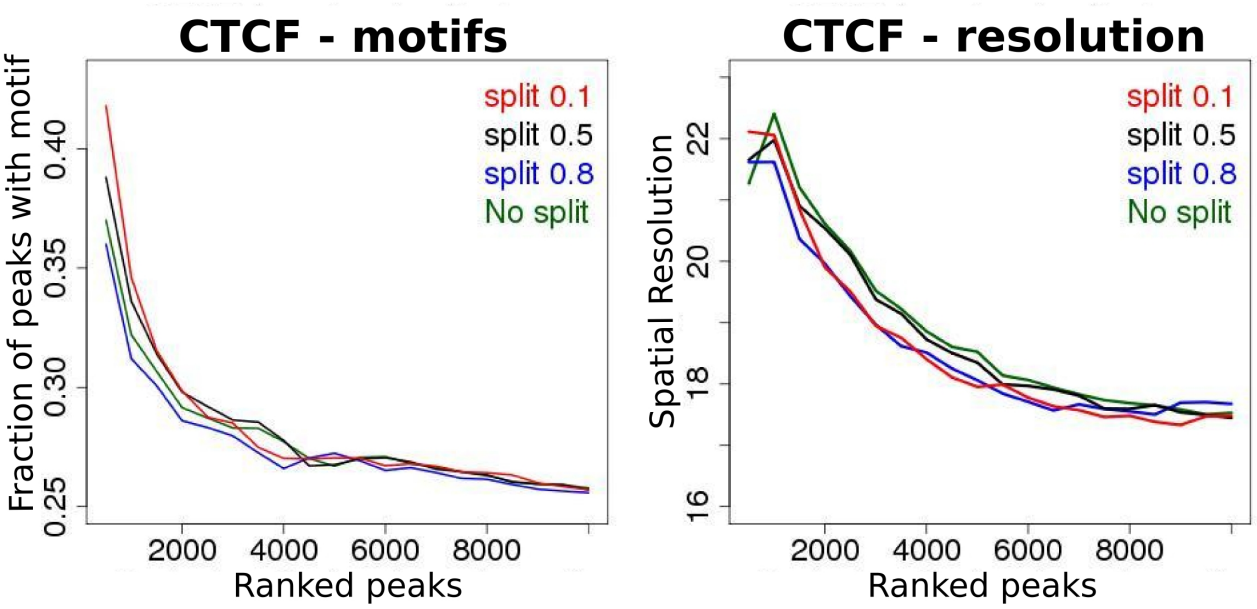


Figure S8. Results of Pyicos protocol callpeaks when CTCF peaks where split using different parameters. a) Fraction of peaks with motifs along peaks ranked by height score. **b)** Spatial resolution, defined as distance between motif centre and peak summit (as in Figure S3), for each split parameter along the same ranking of peaks.

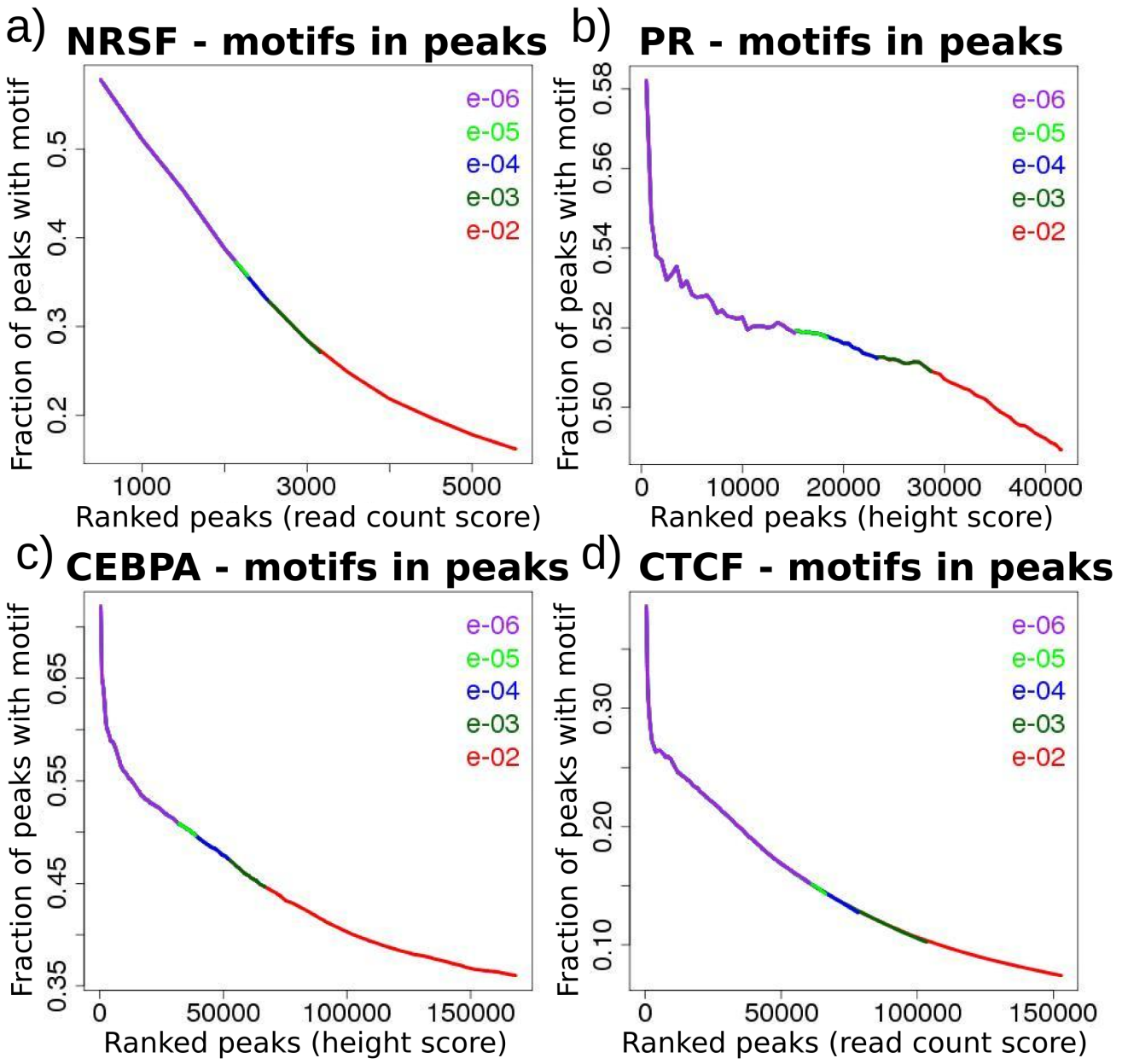


Figure S9. P-values from *Pyicos* Poisson analysis along the ranking of peaks for **a)** NRSF **b)** PR **c)** CEBPA and **d)** CTCF ChIP-Seq data. On the x-axes we ranked the peaks according to the appropriate score for the dataset (height score or read count score) as explained in Methods. The y-axes show the fraction of peaks with the corresponding motif, calculated as explained in Methods.

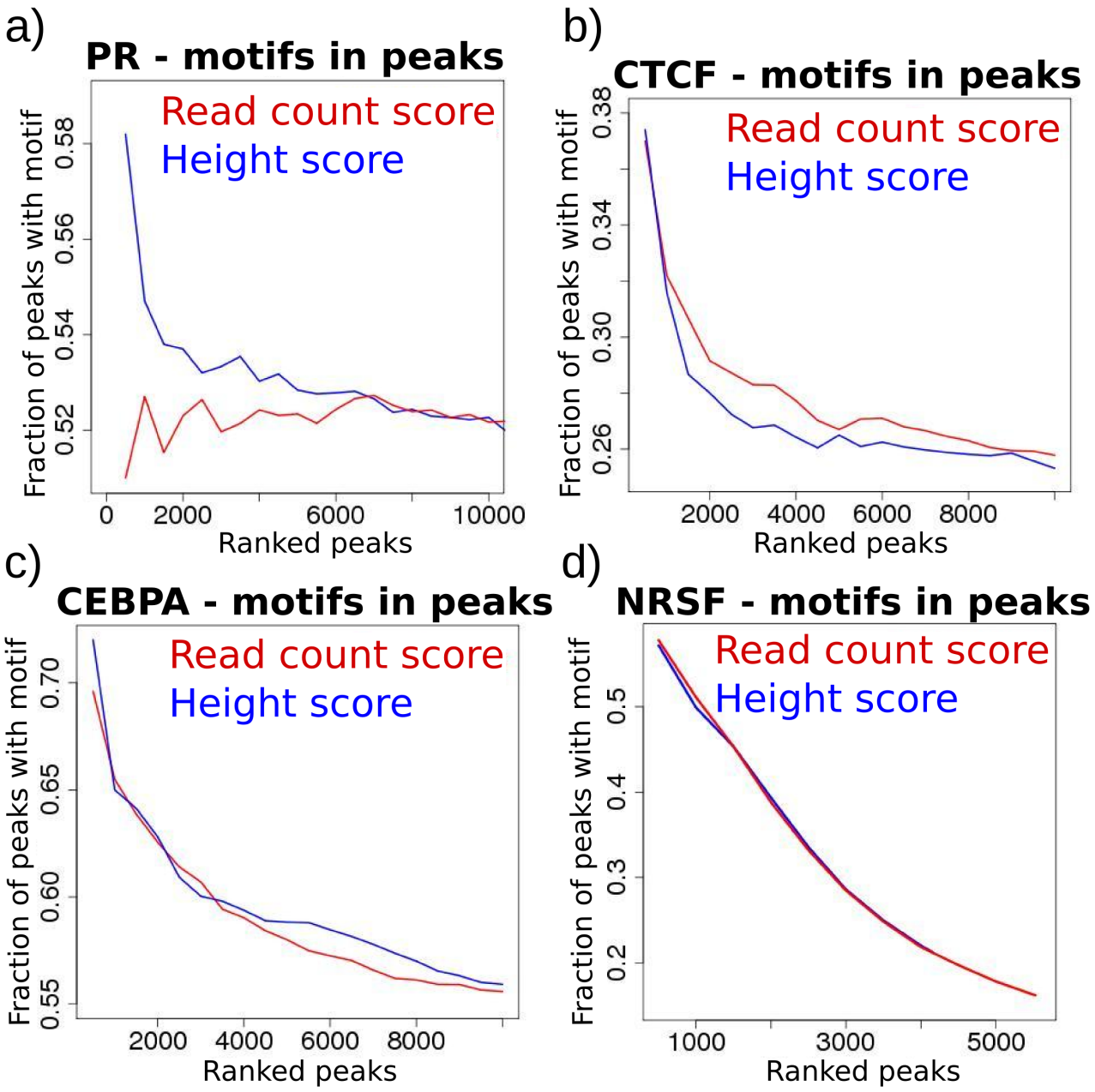


Figure S10. Ranking according to height score or read count score, as explained in Methods, for **a)** PR **b)** CTCF **c)** CEBPA and **d)** NRSF ChIP-Seq data. On the x-axes we ranked the peaks according to each scoring system. The y-axes show the fraction of peaks with the corresponding motif, calculated as explained in Methods.

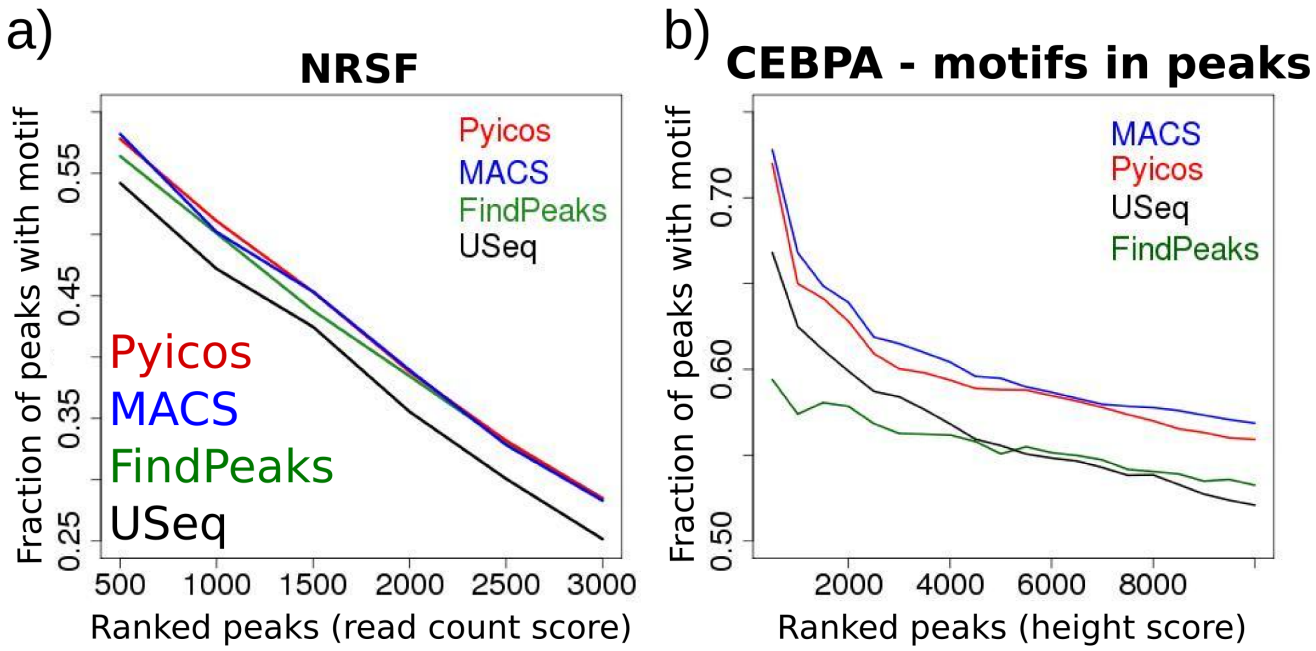


Figure S11. Comparing the four peak-calling methods in terms of fraction of peaks with motif on **a)** NRSF ranked by read count score and **b)** CEBPA ranked by height score. On the x-axes we ranked the peaks according to the appropriate score for the dataset (height score or read count score) as described in Methods. The y-axes show the fraction of peaks with the corresponding motif, calculated as explained in Methods.

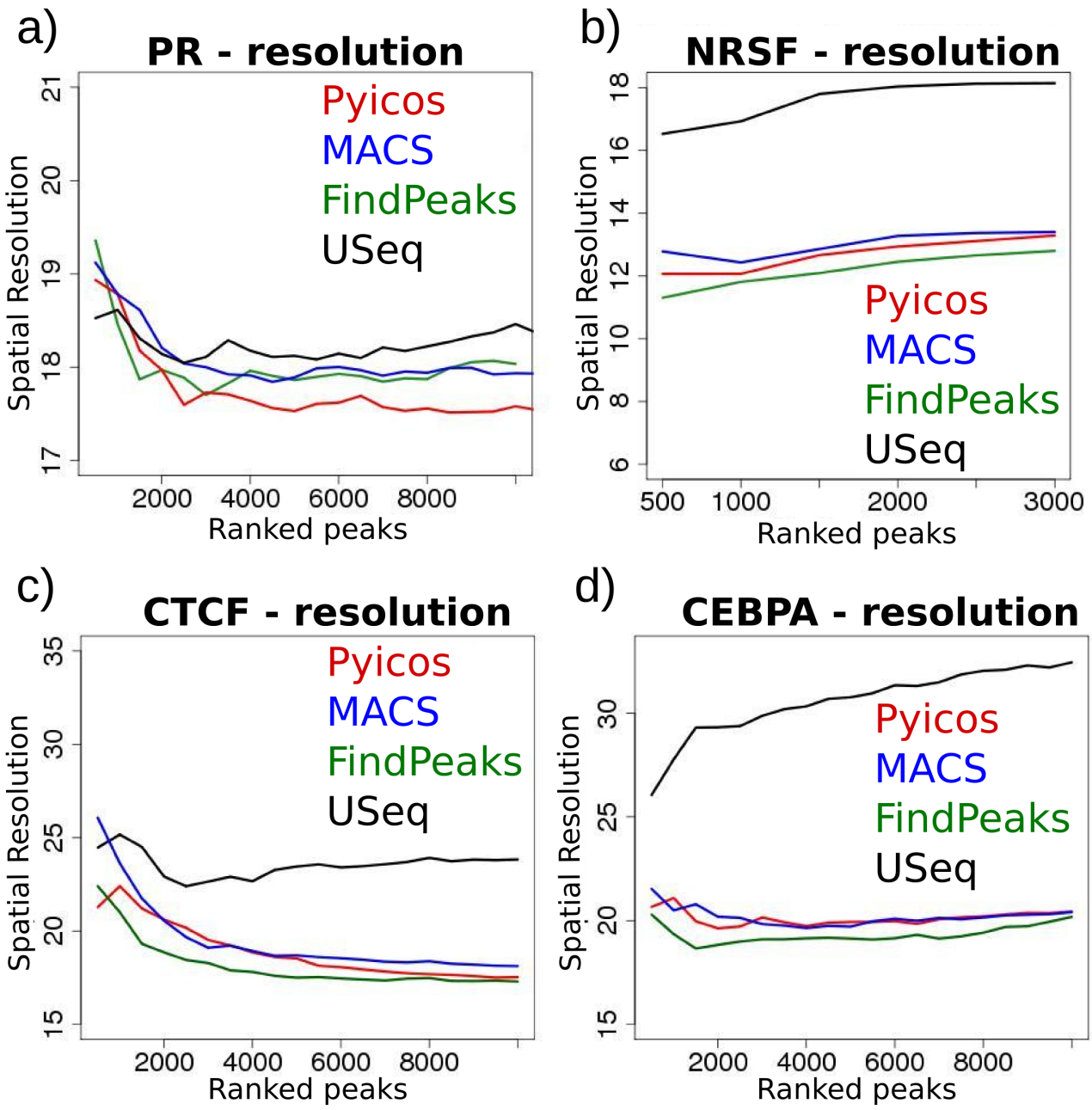


Figure S12. **Comparing the four methods in terms of spatial resolution** on **a)** PR **b)** NRSF **c)** CTCF and **b)** CEBPA ChIP-Seq data. On the x-axes we ranked the peaks according to the appropriate score for the dataset (height score or read count score) as described in Methods. The y-axes show the spatial resolution, defined as the distance between motif centre and peak summit (as in Figure S3).

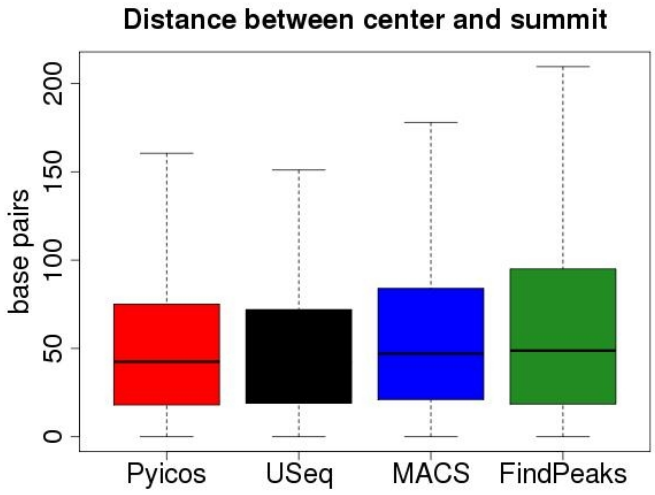


Figure S13. Distance between peak summit and peak centre for each peak-calling method on NRSF ChIP-Seq data. Peak summit and centre as defined in Figure S3.

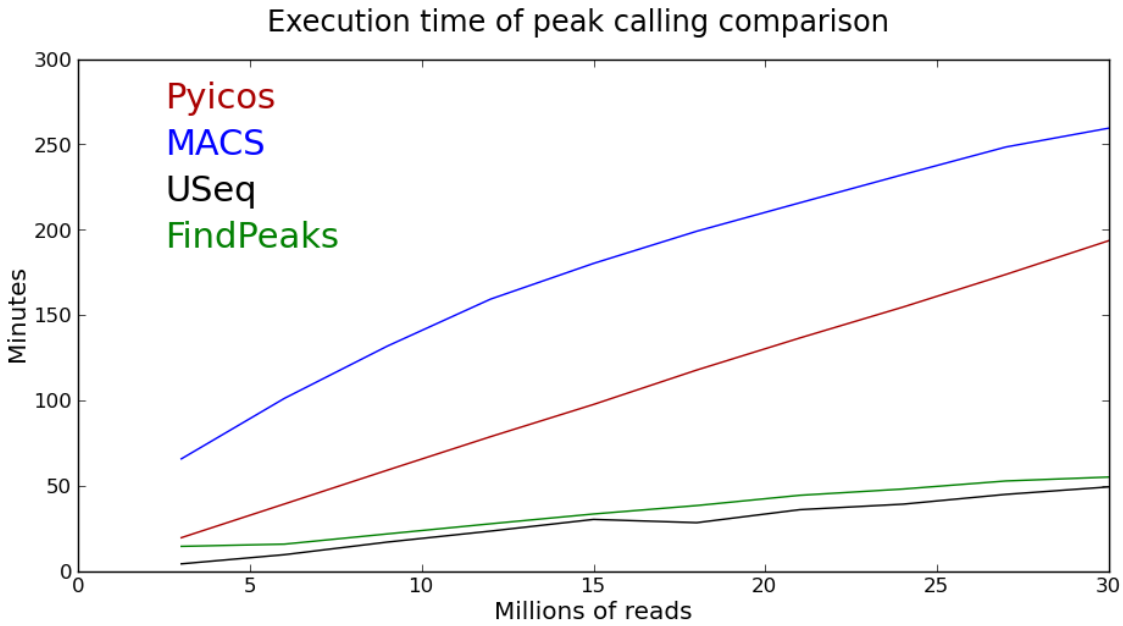
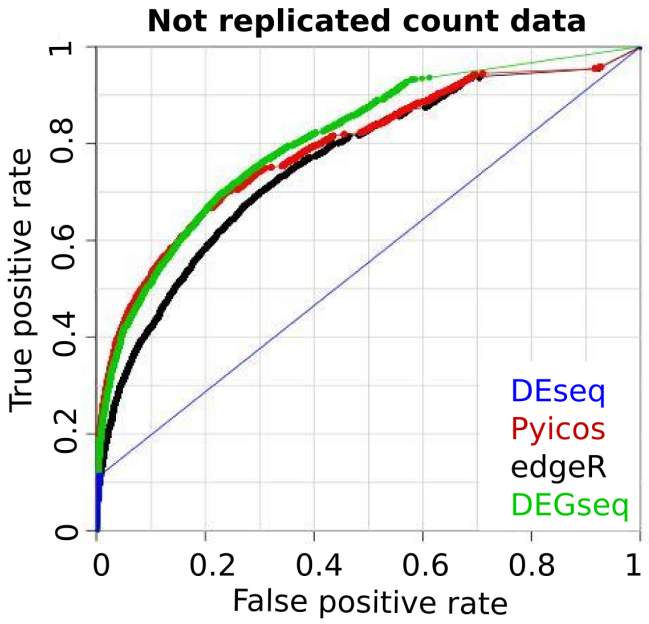
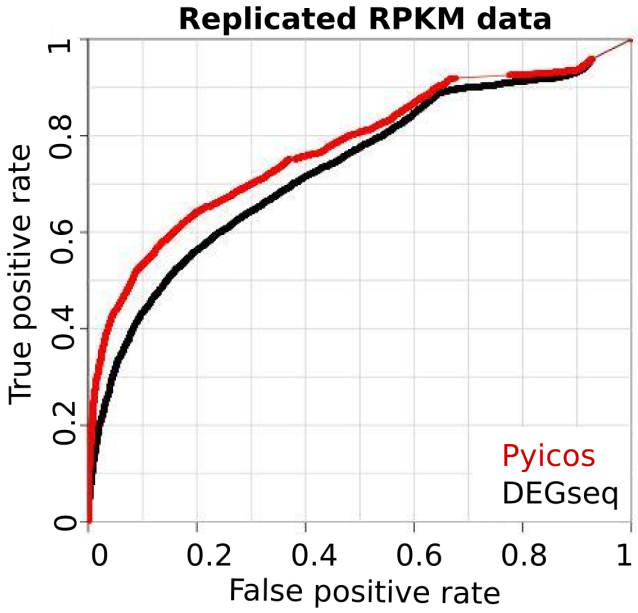


Figure S14. CPU time performance of peak calling on CEBPA. Time of execution of different methods. Methods have been run on conditions as similar as possible (see in Methods), all producing wiggle files.

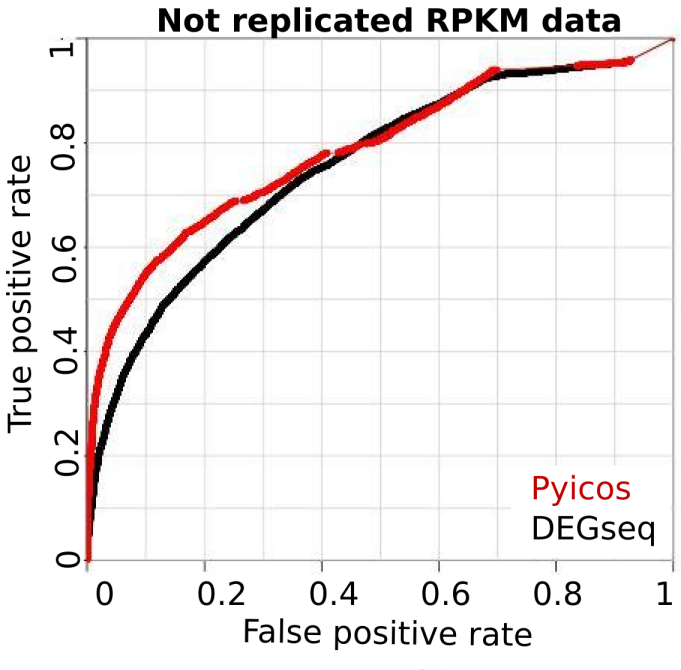
a)


Not replicated count data
True positive rate
False positive rate
0
0.2
0.4
0.6
0.8
1
DEseq
Pyicos
edgeR
DEGseq

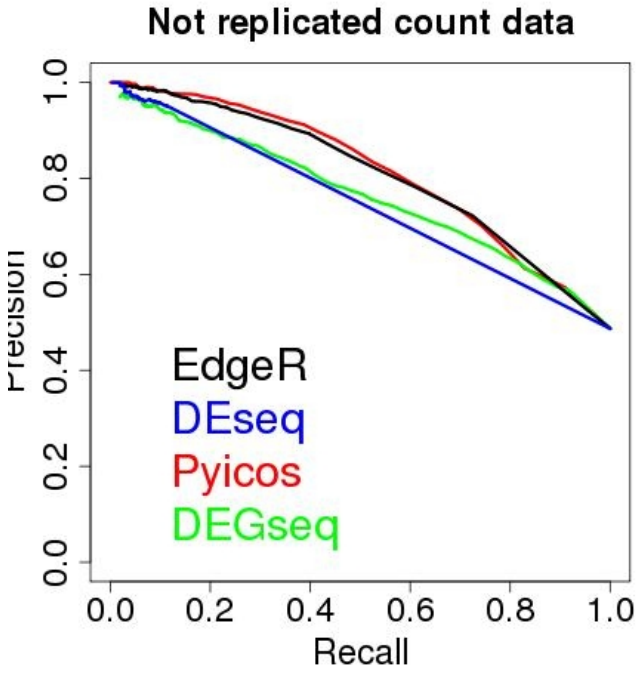
b)


Replicated RPKM data
True positive rate
False positive rate
0
0.2
0.4
0.6
0.8
1
Pyicos
DEGseq

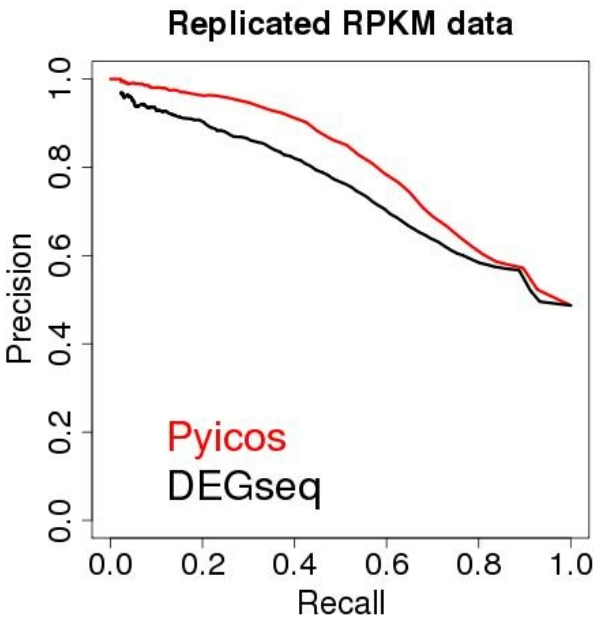
c)


Not replicated RPKM data
True positive rate
False positive rate
0
0.2
0.4
0.6
0.8
1
Pyicos
DEGseq

d)


Not replicated count data
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0
EdgeR
DEseq
Pyicos
DEGseq

e)



f)

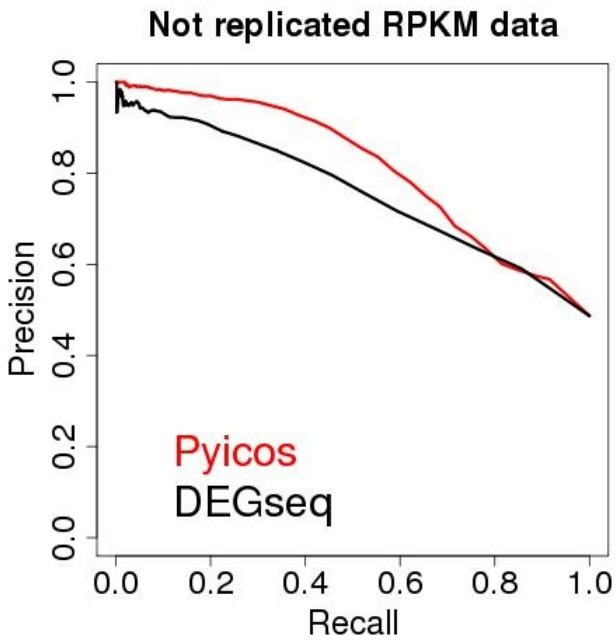


Figure S15. Prediction of differentially expressed genes. ROC curves (as in Methods) for the accuracy assessment on the array benchmarking set (as in Methods) for **a)** not replicated count data **b)** replicated RPKM data, and **c)** not replicated RPKM data; and precision-recall curves (as in Methods) on the same benchmarking set for **d)** not replicated count data **e)** replicated RPKM data, and **f)** not replicated RPKM data.

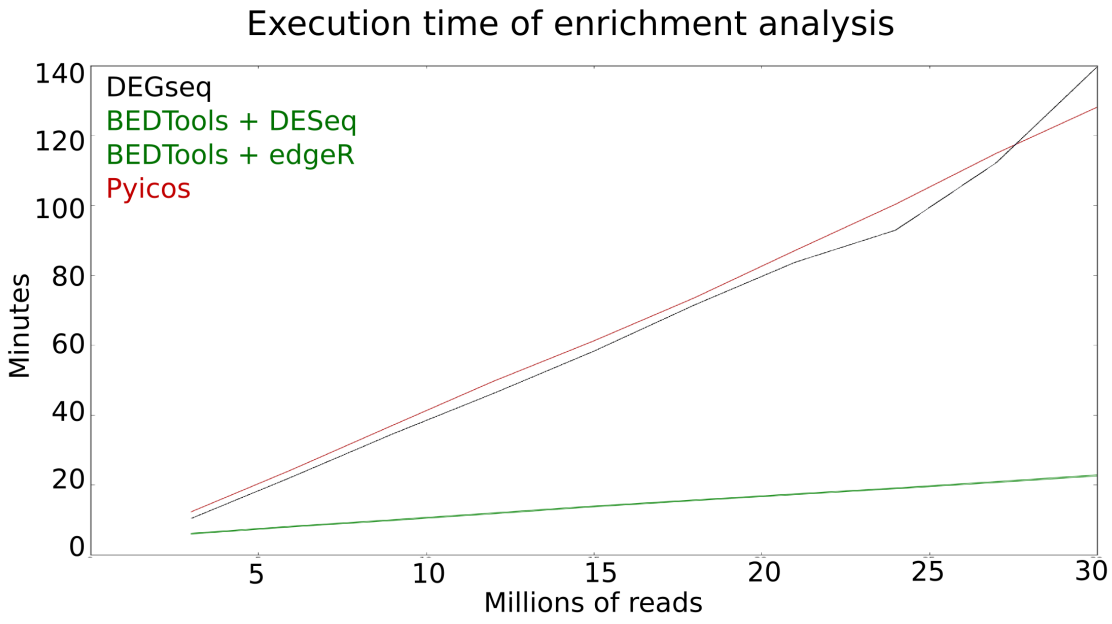


Figure S16. CPU time performance of enrichment analysis on CEBPA. Methods have been run on conditions as similar as possible (see below in this Supplementary Material).

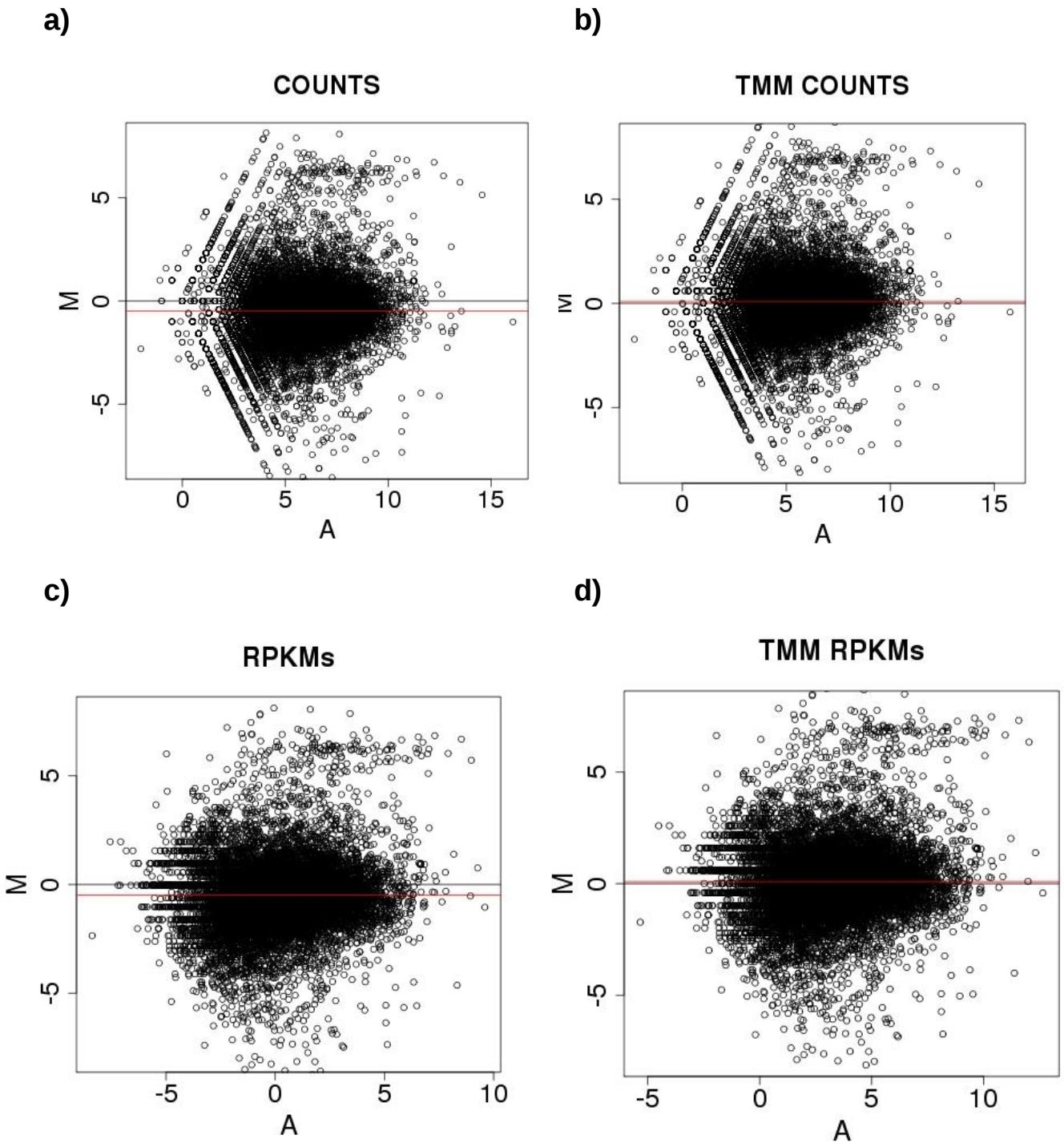
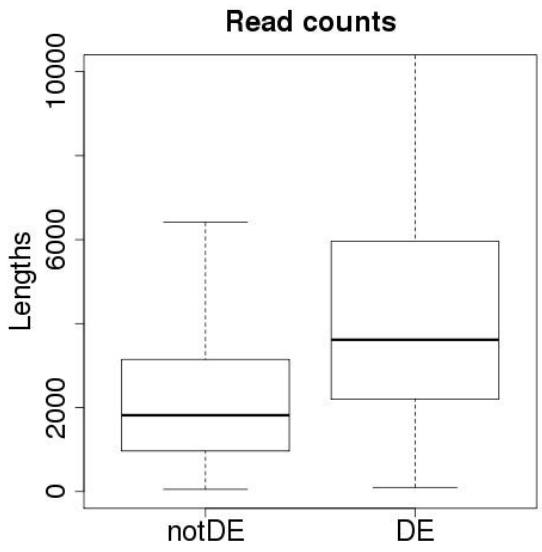
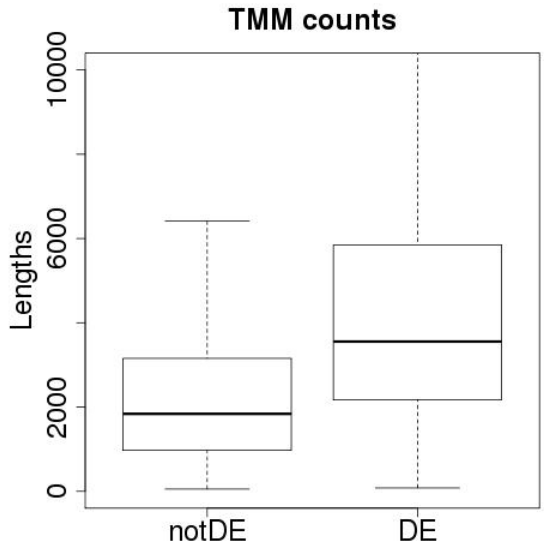


Figure S17. Correction of M-medians by TMM normalization. MA plots (explained in Supplementary Material) for the liver versus kidney comparison using **a)** not normalized count data **b)** TMM normalized count data **c)** RPKM and **d)** TRPK, defined as Tmm-normalized Read Per Kilobase (see definition in this Supplementary Material). The median of the M-values is shown as a red line.

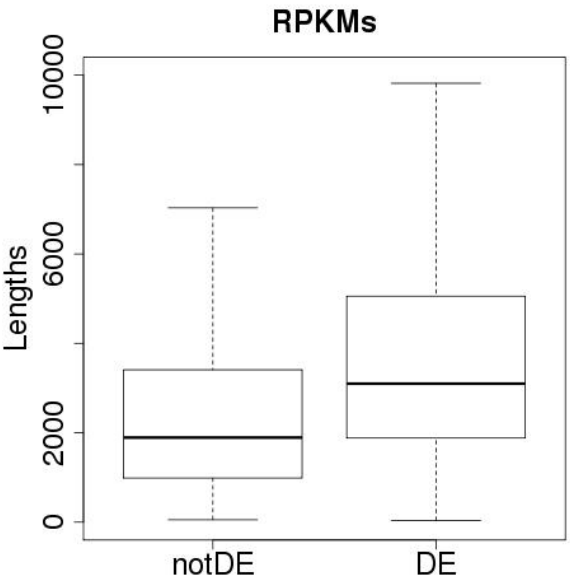
a)


Read counts
Lengths
0
2000
6000
10000
notDE
DE

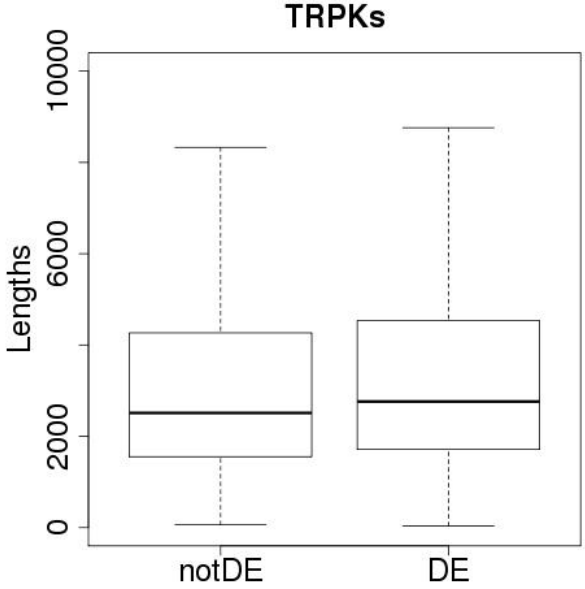
b)


TMM counts
Lengths
0
2000
6000
10000
notDE
DE

c)


RPKMs
Lengths
0
2000
6000
10000
notDE
DE

d)


TRPKs
Lengths
0
2000
6000
10000
notDE
DE

e)

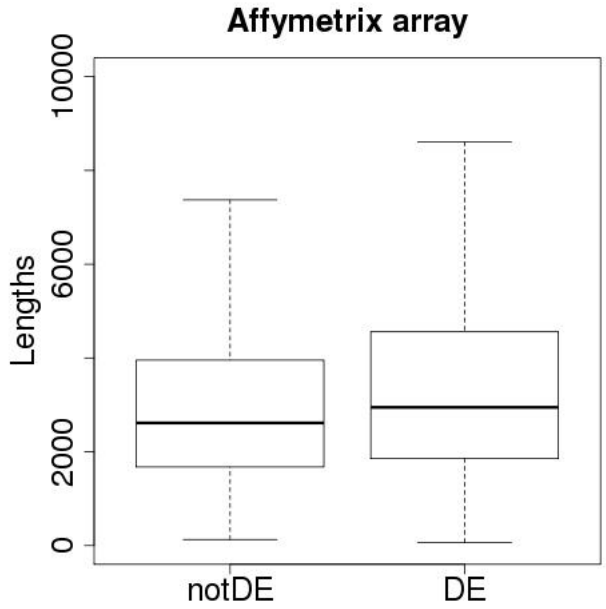


Figure S18. Length-bias of differentially expressed (DE) genes. Distributions of average transcript lengths of differentially expressed (DE) and non-DE genes between liver and kidney, predicted based on **a)** count data **b)** TMM normalized count data **c)** RPKM and **d)** TRPK and **e)** microarray.

	Pyicos	MACS	FindPeaks	USeq
Pyicos	3000	2537	2423	2413
MACS		3000	2373	2340
FindPeaks			3000	2283
USeq				3000

Table S1. Overlap of the top 3,000 NRSF peaks from the four different methods considering the 100nt region centred on summit.

	Pyicos	MACS	FindPeaks	Useq
Pyicos	1	0.97	0.87	0.98
MACS		1	0.93	0.97
FindPeaks			1	0.96
USeq				1

Table S2. Spearman correlation coefficients for the comparison of the scores from the top 3,000 NRSF peaks calculated by the four methods.

	CC
RPKMs with replica	0.75
Counts with replica	0.77
RPKMs no replica	0.7
Counts no replica	0.76

Table S3. Spearman correlation coefficients from the comparison of the Z-scores (as in Supplementary Material) calculated by Pyicos and DEGseq on the RNA-Seq data from (Marioni et al., 2008), for different configurations of input: RPKM or counts; and with or without replica.

Benchmarking of Pyicos results against other methods

Definition of peak summit

MACS, FindPeaks and Pyicos provide the summit position in their results. For USeq, we considered the summit as the centre of the sub-window in which the signal along the peak is the strongest.

Commands used for the peak calling benchmarking:

Method	Command line
Pyicos	pyicos protocol NRSF.ptl*
MACS	python /soft/bin/macs -t GSM327023_chipFC1592_uniq_hg17.bed -c GSM327024_mockFC1592_uniq_hg17.bed --name NRSF.macs
FindPeaks	java -jar SeparateReads.jar bed GSM327023_chipFC1592_uniq_hg17.bed chip/ java -jar SeparateReads.jar bed GSM327024_mockFC1592_uniq_hg17.bed mock/ java -jar SortFiles.jar bed chip_sorted chip/*gz java -jar SortFiles.jar bed mock_sorted mock/*gz java -jar FindPeaks.jar -aligner bed -alpha 1 -eff_frac 0.7 -duplicatefilter -minimum 1 -dist_type 1 80 100 60 -input chip/chr*.part.bed.gz -control mock/chr*.part.bed.gz -output findpeaks_results/ -name NRSF
USeq	java -Xmx1500M -jar USeq_7.2/Apps/Tag2Point -f NRSF/GSM327023_chipFC1592_uniq_hg17.bed -v H_sapiens_May_2004 -b java -Xmx1500M -jar USeq_7.2/Apps/Tag2Point -f NRSF/GSM327024_mockFC1592_uniq_hg17.bed -v H_sapiens_May_2004 -b java -Xmx1500M -jar USeq_7.2/Apps/PeakShiftFinder -t NRSF/GSM327023_chipFC1592_uniq_hg17_Point/ -c NRSF/GSM327024_mockFC1592_uniq_hg17_Point/ -s USeq_NRSF/ java -Xmx1500M -jar USeq_7.2/Apps/ScanSeqs -t NRSF/GSM327023_chipFC1592_uniq_hg17_Point/ -c NRSF/GSM327024_mockFC1592_uniq_hg17_Point/ -s USeq_NRSF/ -w 200 -p 78 -r /soft/bin/R java -Xmx1500M -jar USeq_7.2/Apps/EnrichedRegionMaker -f USeq_NRSF/windowData200bp.swi -i 0,2,4 -s 10,10,1 -t NRSF/GSM327023_chipFC1592_uniq_hg17_Point/ -c NRSF/GSM327024_mockFC1592_uniq_hg17_Point/ -p /soft/bin/R

* protocol file: NRSF.ptl :

```
[Pyicotrocol]
experiment=GSM327023_chipFC1592_uniq_hg17.bed
experiment_format=bed
open_experiment=true
control=GSM327024_mockFC1592_uniq_hg17.bed
control_format=bed
open_control=true
output=NRSF_pyicos.pk
operations=remove_duplicates, extend, normalize, subtract, poisson, filter

frag_size=100
correction=0.8
duplicates=0
poisson_test=numtags
```

Comparison of memory usage and CPU time performance

Commands used to compare time and memory performance of peak calling

Pyicos:

`pyicos all experiment_file.bed result_file.wig --frag-size 100 -f bed -F bed_wig -O --control control_file.bed --control-format bed --normalize --extend --subtract --filter --poisson`

Useq:

`java -jar USeq_7.2/Apps/Tag2Point -f file.bed -v H_sapiens_Mar_2006 -b` (for both experiment and control files)

`java -jar USeq_7.2/Apps/PeakShiftFinder -t experiment_Point -c control_Point -s Useq_result/`

`java -Xmx10G -jar USeq_7.2/Apps/ScanSeqs -t experiment_Point -c control_Point -s USeq_result/ -w 200 -p 69 -r /soft/bin/R` (values come from previous commands)

`java -Xmx10G -jar USeq_7.2/Apps/EnrichedRegionMaker -f USeq_result/windowData200bp.swi -i 0,2,4 -s 10,10,1 -t experiment_Point -c control_Point -p /soft/bin/R`

FindPeaks:

`java -jar SeparateReads.jar bed file.bed result/` (per experiment and control)

`java -jar SortFiles.jar bed result/ file.part.bed.gz` (Now files are separated by chromosome, so per chromosome and experiment/control)

`java -jar FindPeaks.jar -aligner bed -eff_frac 0.7 -duplicatefilter -minimum 1 -dist_type 1 80 100 60 -input experiment_chrN.part.bed.gz -control control_chrN.part.bed.gz -output findpeaks_results/ -name result`

MACS:

`macs -t experiment -c control --pvalue=0.01 --wig --space 1 --name result --nomodel --verbose 1 --tsize 36`

Commands used to compare time and memory performance of the enrichment analysis

Pyicos:

```
pyicos enrichment file_a.bed file_b.bed result. -f bed -O --replica-a replica_a.bed --region
refgenes_stranded_sorted_promoters.bed
```

Note: Count and RPKM files were calculated by Pyicos.

DEGSeq:

```
library(DEGseq)
geneExpFile <- commandArgs(TRUE)[1]

geneExpMatrix1 <- readGeneExp(file = geneExpFile, geneCol = 1, valCol = c(2, 4))
geneExpMatrix2 <- readGeneExp(file = geneExpFile, geneCol = 1, valCol = c(3, 5))

layout(matrix(c(1, 2, 3, 4, 5, 6), 3, 2, byrow = TRUE))
par(mar = c(2, 2, 2, 2))

outputDir =file.path(".","DEGseq_liver_kidney_all")

DEGexp(
geneExpMatrix1 = geneExpMatrix1, expCol1 = 2, groupLabel1 = "Liver",
geneExpMatrix2 = geneExpMatrix2, expCol2 = 2, groupLabel2 = "Kidney",
replicateExpMatrix1 = geneExpMatrix1, expColR1 = 3,
replicateExpMatrix2 = geneExpMatrix2, expColR2 =  3,
replicateLabel1 = "L1", replicateLabel2 = "L2",
method = "MATR", outputDir=outputDir)
```

DESeq:

```
library( DESeq )
print(commandArgs(TRUE)[1])
countsTable1 <- read.delim(commandArgs(TRUE)[1], header=FALSE, stringsAsFactors=TRUE)
countsTable <- countsTable1[ , -1 ]
conds <- c("L","K","L","L")
cds <- newCountDataSet( countsTable, conds )
cds=estimateSizeFactors(cds)
cds <- estimateVarianceFunctions( cds )
resTbvsN <- nbinomTest( cds, "L", "K" )
plot(
 resTbvsN$baseMean,
 resTbvsN$log2FoldChange,
 log="x", pch=20, cex=1,
 col = ifelse( resTbvsN$padj < .1, "red", "black" ) )
resTbvsN$gene <- countsTable1[,1]
x= resTbvsN$pval
write.table(x,file="DEseq_simplecounts")
x= resTbvsN$padj
write.table(x,file="DEseq_simplecounts_padj")
```

EdgeR:

```
library(edgeR)
###  replica
raw.data <- read.delim(commandArgs(TRUE)[1], header=FALSE)
D = as.matrix(raw.data[, 2:5])
f <- calcNormFactors(D)
f <- f/exp(mean(log(f)))
lib.sizes          <-          c(commandArgs(TRUE)[2],commandArgs(TRUE)[2],commandArgs(TRUE)
[2],commandArgs(TRUE)[2])
 g <- c("L","K","L","K")
 d <- DGEList(counts = D, group = g, lib.size = colSums(D) * f)
#rownames(D)=raw.data[, 1]
d$gene=raw.data[, 1]
par(mfrow=c(1,2))
maPlot(D[, 1], D[, 2], normalize = TRUE, pch = 19, cex = 0.2,ylim = c(-8, 8))
abline(h = log2(f[2]), col = "red", lwd = 4)
maPlot(d$counts[, 1]/d$samples$lib.size[1], d$counts[, 2]/d$samples$lib.size[2],normalize =
FALSE, pch = 19, cex = 0.2, ylim = c(-8, 8))
d <- estimateCommonDisp(d)
de.com <- exactTest(d)
#x=topTags(de.com)
write.table(de.com$table,file="EdgeR_simplecounts")
```

The bedpk format

Pyicos has its own compressed format for peaks, the bedpk. This is also used as an internal representation for peaks, which can be used as input and output formats. It follows the same starting fields “chromosome/tag start end” but it uses some of the original optional fields to include extra information. It is a cluster oriented format that aims to recollect information of a cluster of reads in a single comprehensive line:

Column description:

1. Chromosome
2. Start coordinate
3. End coordinate
4. Profile: This field summarizes the read-count per nucleotide of the cluster. It is written as an array of 2-tuples, where the first number of the tuple is the number of bases covered and the second number of the tuple is the number of reads that cover those bases (see example below)
5. Height: The maximum height of the cluster (3 in the example below).
6. Strand: “+” if ALL reads in the cluster are positive strand, and “-” if they are all negative; “.” otherwise.
7. Summit: The coordinate position where the maximum height is found. If there is more than one position, the midpoint is taken.
8. Area: The area covered by the cluster.
9. p-value: The significance of the cluster calculated by the Poisson operation based on peak heights or numbers of reads.

Furthermore, Pyicos can handle other various formats, providing converter capabilities. It can read eland, SAM, BED, and wiggle (fixed and variable step); and write to SAM, BED, wiggle (fixed and variable step). Some formats are obligatory for some of the operations, which is specified in the corresponding command help. Pyicos default format is a derivative of UCSC BED format.

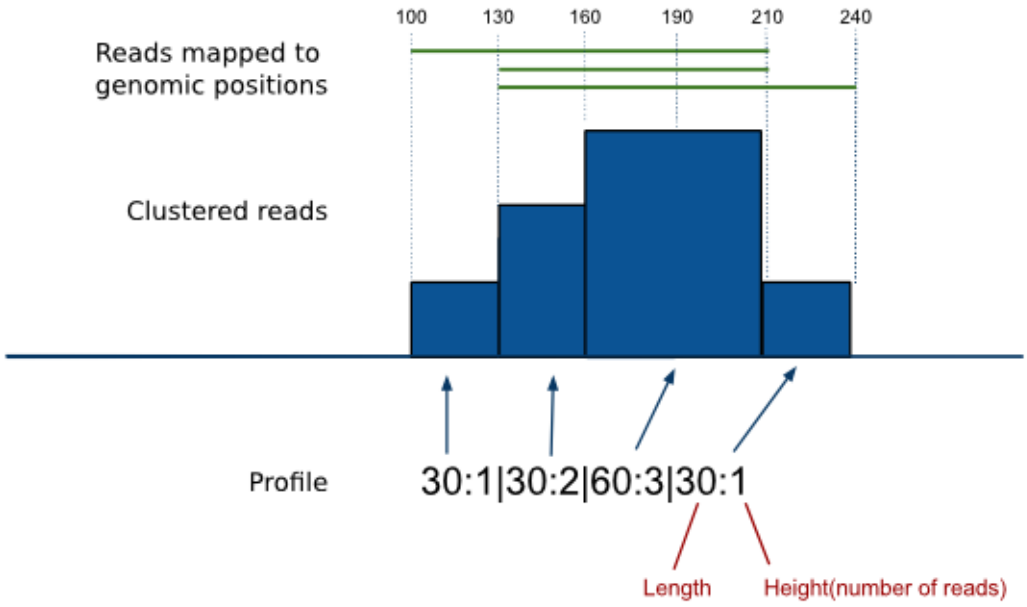


Figure S19. Illustration of the bedpk format. See description in the text. This notation can be applied to any read-count data on any coordinate system.

ModFDR method for the CLIP-Seq analysis

CLIP-Seq reads can be mapped to the genome when the RNA binding protein (RBP) acts at the pre-mRNA level, or to the mature transcripts when the RBP acts on the mRNA. The reads are clustered according to position overlap. A particular type of enrichment analysis applicable to CLIP-Seq reads is the modified FDR, introduced in (Yeo et al., 2009). This operation is implemented in *Pyicos* and, as for the enrichment analysis; it can be applied using as coordinate system any regions in the genome, genic regions (i.e. exons and introns) or mature transcripts, where the reads are mapped to the genome or to the mRNA sequences, respectively.

When there is a control available, one can perform in general an FDR analysis over pre-defined regions. These regions can be entire chromosomes, windows along chromosomes, genic regions or mRNA lengths. We classify all positions of the region according to the read-count per position. Accordingly, we define $n(i)$ as the number of positions with a coverage of i reads, such that the total number of reads in the region N is the sum of all $n(i)$. We can then define the probability of obtaining a particular coverage h as:

$$P(h)=\frac{1}{N}\sum_{h>i} n(i)$$

Calculating every $P(h)$ for both, experiment e and control c, we can determine the false discovery rate (FDR) for positions of density h:

$$FDR(h)=\frac{P_c(h)}{P_e(h)}$$

A variation of the FDR method was introduced in the analysis of the CLIP-Seq data for FOX2 (Yeo et al., 2009). This method is used when there is no control sample to compare to. We randomize then the positions of the reads in a given pre-defined region, keeping fixed the length and the number of reads and avoiding internal repeat elements (optional in *Pyicos*); generating for the region a new randomized coverage coverage distribution, $n_R(i)$, representing the number of positions with read-coverage i after randomization. We can then calculate for a read coverage h the randomized probability $P_R(h)$ as

$$P_R(h)=\frac{1}{N}\sum_{h>i} n_R(i)$$

If we repeat this process k times (by default we use k=100), we can obtain calculate $P_R(h,a)$ for each iteration a=1,...,k. Given the mean and standard deviation for all these iterations:

$$m_R(h)=\frac{1}{k}\sum_{a=1}^{k} P_R(h,a)$$

$$s_R(h)=\sqrt{\frac{1}{k}\sum_{a=1}^{k}\left(P_R(h,a)-m_r(h)^2\right)}$$

we can define the modified FDR for a given density of reads h as:

$$mFDR(h) = \frac{m_R(h) + s_R(h)}{P(h)}$$

where $P(h)$ is the probability for the experiment as defined above. We then calculate the lowest density h such that mFDR(h) < t, where t is the desired FDR limit, which is taken by default to be 0.01.

The subtraction algorithm

We define nucleotide intervals covered by overlapping reads in computer memory by storing its label in the genome or transcriptome (Ex: chromosome 1, GeneA...) and the region start coordinates. We then store in memory n 2-tuples that represent sub-intervals of the interval with different amounts of overlapping reads covering them:

$$t_1=(l_1,r_1), t_2=(l_2,r_2) \ldots t_n=(l_n,r_n)$$

Where the 2-tuple t_i is a sub-interval, l_i is the number of positions covered in t_i and r_i is the number of reads that overlap in this sub-interval. An example of this will be: chr1 550 {(50, 1), (15, 2) (20, 3)}, meaning: Chromosome 1 starting in position 550, has 1 read from position 550 to 600, 2 reads from position 601 to position 615 and 3 reads ovelapping between positions 616 and position 636.

We propose a novel algorithm to subtract 2 intervals A and B (A-B=C) in $O(B*\log(A)+B)$ time, where A is the number of sub-intervals t_a in A; and B is the number of sub-intervals t_b in B. The algorithm can be divided in 2 steps: the **subtract-intervals** step (see below), where the actual subtraction happens and which accounts for the execution time $B*\log(A)$; and the **clean-intervals** step (see below), which joins together overlapping intervals with the same r and deletes redundant information, which accounts for the $+B$ in the execution time. The algorithm achieves single nucleotide precision, while offering fast and scalable performance.

First the sub-intervals in A are copied to the “result” collection of sub-intervals, called C. The algorithm iterates through the sub-intervals in B until the start of the sub-interval is smaller than the rightmost sub-interval in A. When this is detected the subtraction ends. There are three cursors that store the position of the subtraction: One cursor that iterates through B (cursor-B), a cursor that iterates through C (cursor-C) and a “slow” cursor that saves the position of the sub-interval B (slow-cursor) that have been passed by the A intervals, allowing for less iteration and faster execution.

The objective of this algorithm is to pair sub-intervals iteratively as efficiently as possible and then subtract t_b to t_a when they overlap. There are 4 different conformations of the interval, which result in different number of

of intervals after subtraction (Supplementary Figure S20).

Subtract-intervals

```
1 c-sub-intervals ← copy of all a-sub-intervals
2 while number of sub-interval in B is bigger than B-cursor
3       while sub-B is behind the slow-cursor
4             advance slow-cursor
5             if the end of interval B is reached, go to clean-intervals
6       C-cursor ← slow-cursor
7       while number of sub-C is bigger than C-cursor and sub-B started before sub-C finished
8                       case sub-B and sub-C
9                       conformation-1 (B covers all C sub-interval)
10                                    sub-C.height ← (sub-A.height - sub-B.height)
11                      conformation-2 (B starts before C and finishes before C)
12                                    sub-C.height ← (sub-A.height - sub-B.height)
13                                    sub-C.end ← sub-A.start
14                                    add new sub-interval (new-sub) to C in position C-cursor + 1
15                                    new-sub.height ← sub-A.height
16                                    new-sub.start ← sub-B.end
17                                    new-sub.end ← sub-A.end
18                      conformation-3 (B starts after C and finishes after C)
19                                    sub-C.height ← (sub-A.height - sub-B.height)
20                                    sub-C.start ← sub-B.start
21                                    sub-C.end ← sub-B.end
22                                    add new sub-interval to C (new-sub) in position C-cursor
23                                    new-sub.height ← sub-A.height
24                                    new-sub.start ← sub-A.start
25                                    new-sub.end ← sub-B.start
26                                    advance C-cursor #Cursor advances so next sub-C is not the new one
27                      conformation-4 (B starts after C and finishes before C)
28                                    sub-C.height ← (sub-A.height - sub-B.height)
29                                    sub-C.start ← sub-B.start
30                                    sub-C.end ← sub-B.end
31                                    add new sub-interval to C (new-sub1) in position C-cursor + 1
32                                    new-sub1.height ← sub-A.height
33                                    new-sub.start ← sub-B.start
34                                    new-sub.end ← sub-B.end
35                                    add new sub-interval to C (new-sub2) in position C-cursor
36                                    new-sub2.height ← sub-A.height
37                                    new-sub2.start ← sub-A.start
38                                    new-sub2.end ← sub-B.start
39                      advance C-cursor
40              advance B-cursor
41 clean-intervals
```

clean-intervals

```
1  while the first sub-interval height <= 0
2            pop first sub-interval
3  while last sub-interval height >= 0
4            delete last sub-interval
5  for all sub-intervals in C
6            if the height sub-C is equal to the previous height
7                  sub-C.start = previous-start
8                  delete previous-sub
9            else:
10                previous_length ← sub-C.start
11                previous_height ← sub-C.height
```

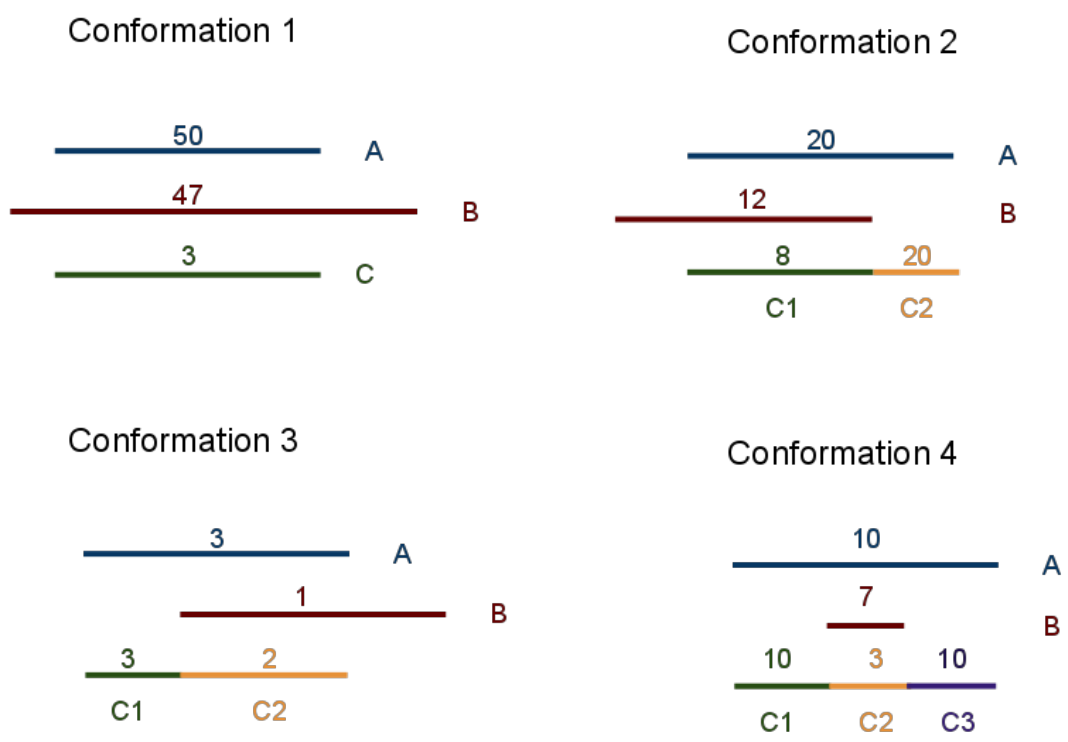


Figure S20. All 4 possible conformations when subtracting interval B from A, resulting in one (C), two (C1 and C2) or three (C1, C2 and C3) intervals of read clusters upon subtraction. For example, in conformation 1, the interval A has 50 reads within it, while the interval B has 47. Since the interval B fully covers the extension of the interval A, the result will be a single interval C of value 3. In the case of conformation 2, since the interval B (12 reads) does not fully cover A (20 reads), the subtraction results in 2 intervals, C1 (8 reads), with coordinates corresponding to the area of the intersection of A and B, and an interval C2, with the remaining part of area A and same number of overlapping reads. The conformation 3 is the mirror image of conformation 2. Conformation 4 can be understood as a combination of conformations 2 and 3.

The split algorithm

Splitting clusters is a useful operation when trying to automatically identify sub-regions in a read covered dense area. The split algorithm is performed in 2 steps. First, all possible local maxima are detected by scanning the cluster once. Then, between each pair of local maxima detected, local minima candidates are detected. These local minima represent a potential point of split. For each of the potential split points we calculate the decrease in height with respect to the lowest of the two flanking local maxima. If this reduction in height is equal or greater than the value set by the –***split-proportion*** flag, the cluster is split in that position into two clusters. For example, if the *split-proportion* cut-off is 0.9, then if a local minima is 0.9 times smaller than the smallest of the flanking local maxima, the peak is split. By default read-clusters are not split in the *callpeaks* protocol, but it can be added to it or used as an independent operation.

References

Marioni,J.C. et al. (2008) RNA-seq: an assessment of technical reproducibility and comparison with gene expression arrays. *Genome Res*, **18**, 1509-1517.

Robinson,M.D. and Oshlack,A. (2010) A scaling normalization method for differential expression analysis of RNA-seq data. *Genome Biol*, **11**, R25.

Yeo,G.W. et al. (2009) An RNA code for the FOX2 splicing regulator revealed by mapping RNA-protein interactions in stem cells. *Nat Struct Mol Biol*, **16**, 130-137.