Analysis of ChIP-Seq Data with R/Bioconductor

...

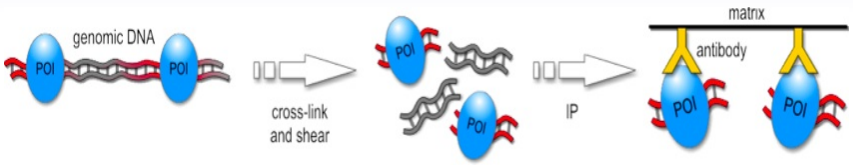
Thomas Girke

December 7, 2014

Outline

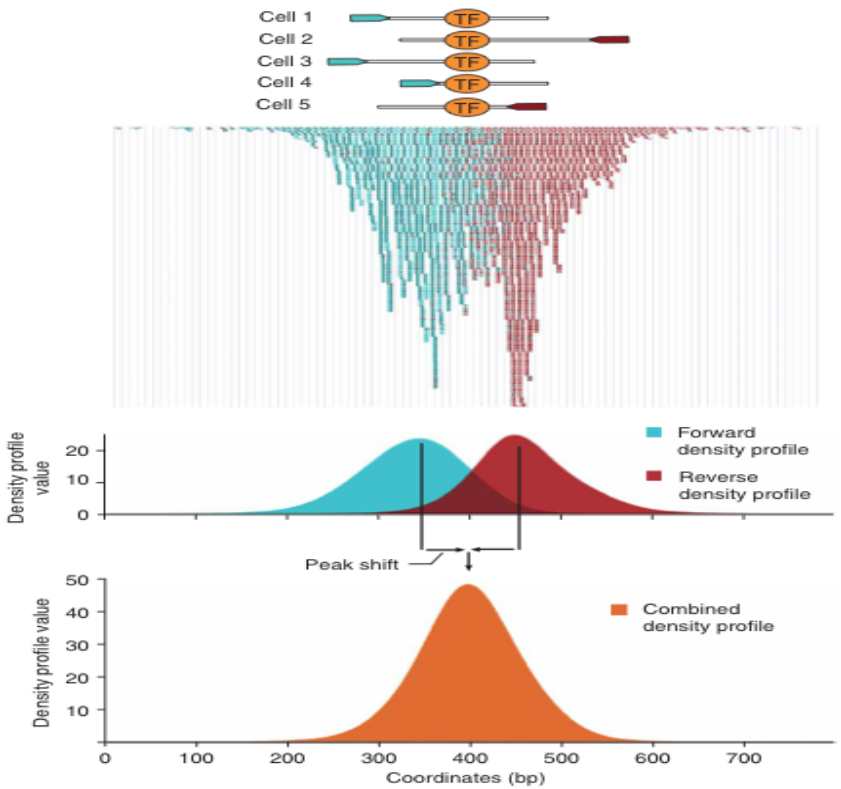
Outline

ChIP-Seq Technology



⇓

ChIP-Seq



ChIP-Seq Workflow

- Read mapping
- Peak calling
- Peak annotation/filtering
- Differential binding analysis
- Motif enrichment analysis in sequences under peaks

Peak Callers (Command-line Tools)

- CisGenome
- ERANGE
- FindPeaks
- F-Seq
- GLITR
- MACS
- PeakSeq
- QuEST
- SICER
- SiSSRs
- spp
- USeq
- ...

Pepke et al. (2009) Computation for ChIP-seq and RNA-seq studies. Nat Methods 6, 22-32. Link

Outline

General Purpose Resources for ChIP-Seq Analysis in R

- GenomicRanges Link: high-level infrastructure for range data
- Rsamtools Link: BAM support
- DiffBind Link: Differential binding analysis of ChIP-Seq peak data
- rtracklayer Link: Annotation imports, interface to online genome browsers
- DESeq Link: RNA-Seq analysis
- edgeR Link: RNA-Seq analysis
- chipseq Link: Utilities for ChIP-Seq analysis
- ChIPpeakAnno Link: Annotating peaks with genome context information
- MotifDb Link: Collection of motif databases
- motifStack Link: Stacked logo plots
- PWMEnrich Link: PWM enrichment analysis
- Bioc Workflow Link: Overview of resources
- ...

Peak Calling in R

- BayesPeak Link: hidden Markov models (HMM) and Bayesian statistics
- PICS Link: probabilistic inference
- MOSAiCS Link: model-based analysis of ChIP-Seq data
- iSeq Link: Hidden Ising Models
- ChIPseqR Link
- CSAR Link: tests based on Poisson distribution
- ChIP-Seq Link
- SPP Link
- NarrowPeaks Link
- ...

Outline

Outline

Data Sets and Experimental Variables

To make the following sample code work, please follow these instructions:

- Download and unpack the sample data Link. Afterwards direct your R session into the new project directory called `Rchipseq`.
- The sample data contains slimmed down versions of four FASTQ files from the ChIP-Seq experiment published by Kaufmann et al. (2010) (GSE20176 Link), a shortened GFF3 annotation file Link and the corresponding reference genome from *Arabidopsis thaliana*.
- Start the analysis by opening in your R session the `Rchipseq.R` script Link which contains the code shown in this slide show in pure text format.

The FASTQ files are organized in the provided `targets.txt` file. This is the only file in this analysis workflow that needs to be generated manually, *e.g.* in a spreadsheet program. To import `targets.txt`, we run the following commands from R:

```
> targets <- read.delim("./data/targets.txt")
> targets
```

```
         FileName SampleName Factor Factor_long
1 SRR038845.fastq    AP1IND1   sig1   sig1_long
2 SRR038846.fastq    AP1IND2   sig1   sig1_long
3 SRR038848.fastq   AP1UIND1   bgr1   bgr1_long
4 SRR038850.fastq   AP1UIND2   bgr1   bgr1_long
```

Outline

Align Reads and Output Indexed Bam Files

Note: Rsubread is for Linux and OS X only. Windows users want to skip the mapping step.

```
> library(Rsubread); library(Rsamtools)
> dir.create("results") # Note: all output data will be written to directory 'result
> buildindex(basename="./results/tair10chr.fasta", reference="./data/tair10chr.fasta
> input <- paste("./data/", targets$FileName, sep="")
> output <- paste("./results/", targets$FileName, ".sam", sep="")
> reference <- "./results/tair10chr.fasta"
> for(i in seq(along=targets$FileName)) {
+         align(index=reference, readfile1=input[i], output_file=output[i], output_f
+         asBam(file=output[i], destination=gsub(".sam", "", output[i]), overwrite=T
+         unlink(output[i])
+ }
```

Outline

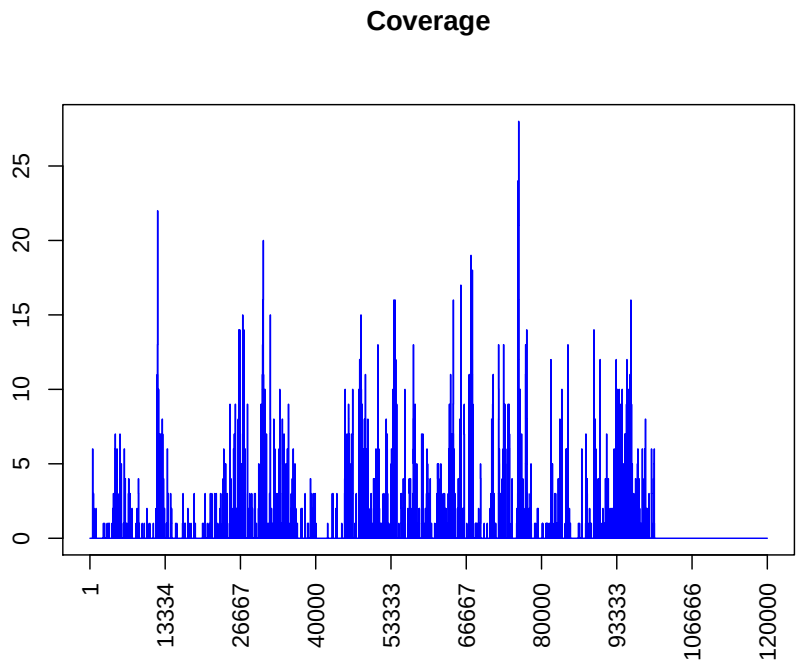
Important Resources for ChIP-Seq Analysis

Coverage and peak slicing

```
> library(rtracklayer); library(GenomicRanges); library(Rsamtools); library(GenomicA
> samples <- as.character(targets$FileName)
> samplespath <- paste("./results/", samples, ".bam", sep="")
> aligns <- readGAlignmentsFromBam(samplespath[3])
> cov <- coverage(aligns)
> islands <- slice(cov, lower = 15)
> islands[[1]][1:12,]

Views on a 30427671-length Rle subject
views:
      start   end width
 [1] 11998 12003     6 [22 22 22 22 22 22]
 [2] 27064 27077    14 [15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [3] 30619 30623     5 [16 16 16 16 16]
 [4] 30709 30725    17 [16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 20 20]
 [5] 31956 31975    20 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
 [6] 48015 48029    15 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15]
 [7] 53839 53844     6 [16 16 16 16 16 16]
 [8] 54031 54066    36 [16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16 16
 [9] 64384 64394    11 [16 16 16 16 16 16 16 16 16 15 15]
[10] 65692 65719    28 [15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15 15
[11] 67505 67508     4 [19 19 19 19]
[12] 67743 67770    28 [18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18 18
```

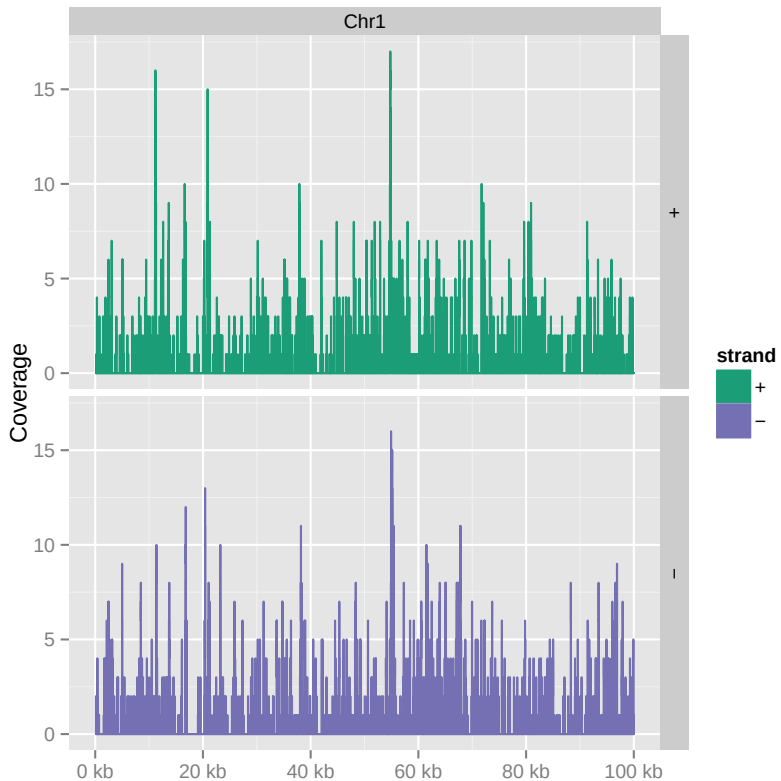
Coverage Plot with Base Graphics

```
> plotCovBase <- function(mycov=cov, mychr=1, mypos=c(1,1000), mymain="Coverage", ...) {
+     op <- par(mar=c(8,3,6,1))
+     plot(as.numeric(mycov[[mychr]][mypos[1]:mypos[2]]), type="l",
+     lwd=1, col="blue", ylab="", main=mymain, xlab="", xaxt="n", ...)
+     axis(1, las = 2, at=seq(1,mypos[2]-mypos[1], length.out= 10),
+     labels=as.integer(seq(mypos[1], mypos[2], length.out= 10)))
+     par(op)
+ }
> plotCovBase(mycov=cov, mychr="Chr1", mypos=c(1,120000)) # Remember: read data is trunctated to first 100kbp
```



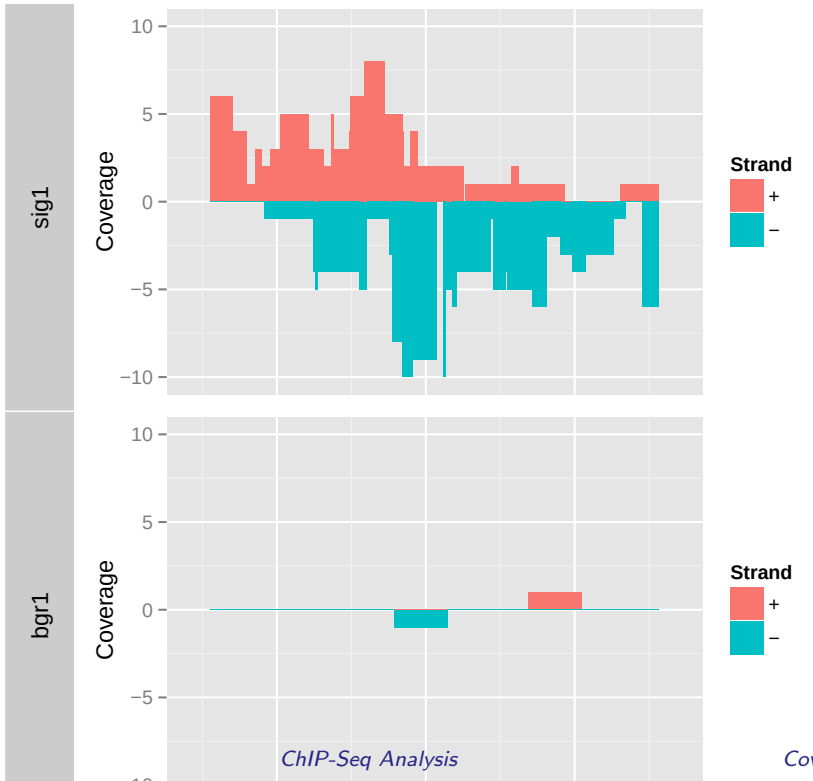
Coverage Plot with *ggbio*

```
> library(ggbio)
> myloc <- c("Chr1", 1, 100000)
> ga <- readGAlignmentsFromBam(samplespath[1], use.names=TRUE, param=ScanBamParam(which=GRanges(myloc[1], IRang
> autoplot(ga, aes(color = strand, fill = strand), facets = strand ~ seqnames, stat = "coverage")
```



Mirror Plot for +/- Coverage of Sample Peak

```
> library(ggplot2)
> plotCov <- function(bampath, mychr, mystart, myend, ymin, ymax) {
+         ga <- readGAlignmentsFromBam(bampath, use.names=TRUE, param=ScanBamParam(which=GRanges(mychr, IRanges
+         pos <- as.numeric(coverage(ga[strand(ga)=="+"])[[mychr]])[mystart:myend]
+         pos <- data.frame(Chr=rep(mychr, length(pos)), Strand=rep("+", length(pos)), Position=mystart:myend, 
+         neg <- as.numeric(coverage(ga[strand(ga)=="-"])[[mychr]])[mystart:myend]
+         neg <- data.frame(Chr=rep(mychr, length(neg)), Strand=rep("-", length(neg)), Position=mystart:myend, 
+         covdf <- rbind(pos, neg)
+         ggplot(covdf, aes(Position, Coverage, fill=Strand)) + geom_bar(stat="identity", position="identity") +
+ }
> p1 <- plotCov(bampath=samplespath[1], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> p2 <- plotCov(bampath=samplespath[3], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> tracks(sig1=p1, bgr1=p2)
```



Import Aligned Read Data

Import aligned reads (bam files) and extend to 200bp mappings. Note: to remove PCR duplicates and secondary alignments, one can flag them by assigning the `flag_pcr` filter to the `param` argument of the `readGAlignmentsFromBam` function. Subsequently, one would filter on the `flag` column of the alignment object.

```
> chip_signal_list <- sapply(samplespath, list)
> for(i in seq(along=samplespath)) {
+         flag_pcr <- ScanBamParam(flag=scanBamFlag(isDuplicate=FALSE, isNotPrimaryRead=FALSE), what=c("qual",
+                                  # To use flag_pcr, assign it to param argument in next line.
+         aligns <- readGAlignmentsFromBam(samplespath[i], param=NULL)
+         chip_signal_list[[i]] <- as(aligns, "GRanges")
+ }
> chip_signal_list[["./results/SRR038845.fastq.bam"]][1:4,]

GRanges object with 4 ranges and 0 metadata columns:
      seqnames     ranges strand
         <Rle>  <IRanges>  <Rle>
  [1]     Chr1 [121, 156]      -
  [2]     Chr1 [121, 156]      -
  [3]     Chr1 [216, 251]      +
  [4]     Chr1 [295, 330]      +
  -------
  seqinfo: 7 sequences from an unspecified genome

> chip_signal_list <- sapply(names(chip_signal_list), function(x) trim(resize(chip_signal_list[[x]], width = 20
```

Outline

Naive Peak Calling by Coverage Value

Compute coverage and call peaks

```
> Nreads <- sapply(names(chip_signal_list), function(x)  length(chip_signal_list[[x]]))
> # countBam(BamFileList(samplespath)) # Similar result, but obtained in a more memory efficient manner.
> normfactor <- 10^6/Nreads
> chip_signal_list <- sapply(names(chip_signal_list), function(x) coverage(chip_signal_list[[x]]) * normfactor[
> chip_signal_list[["./results/SRR038845.fastq.bam"]][1:2,]

RleList of length 2
$Chr1
numeric-Rle of length 30427671 with 4193 runs
  Lengths:              139               17               59               10               69
  Values :  6.4233914222031 12.8467828444062  6.4233914222031 9.63508713330464 22.4818699777108  35.3286528221
$Chr2
numeric-Rle of length 19698289 with 19656 runs
  Lengths:              833                1                1                1                2
  Values :                0 9.63508713330464 16.0584785555077 19.2701742666093 22.4818699777108  102.774262755

> chip_peak_list <- sapply(names(chip_signal_list), function(x) slice(chip_signal_list[[x]], lower=5))
> chip_peak_list[[1]][[1]][1:3]

Views on a 30427671-length Rle subject
views:
    start  end width
[1]     1  689   689 [ 6.423391  6.423391  6.423391  6.423391  6.423391  6.423391  6.423391  6.423391  6.42339
[2]   764  963   200 [9.635087 9.635087 9.635087 9.635087 9.635087 9.635087 9.635087 9.635087 9.635087 9.63508
[3]  1463 1697   235 [12.84678 12.84678 12.84678 12.84678 12.84678 12.84678 12.84678 12.84678 12.84678 12.8467

> s <- start(chip_peak_list[[1]][[1]][1:3]) # shows how to return start/end/width of peaks
> c <- as.list(chip_peak_list[[1]][[1]][1:3]) # shows how to return coverage values of peaks
```

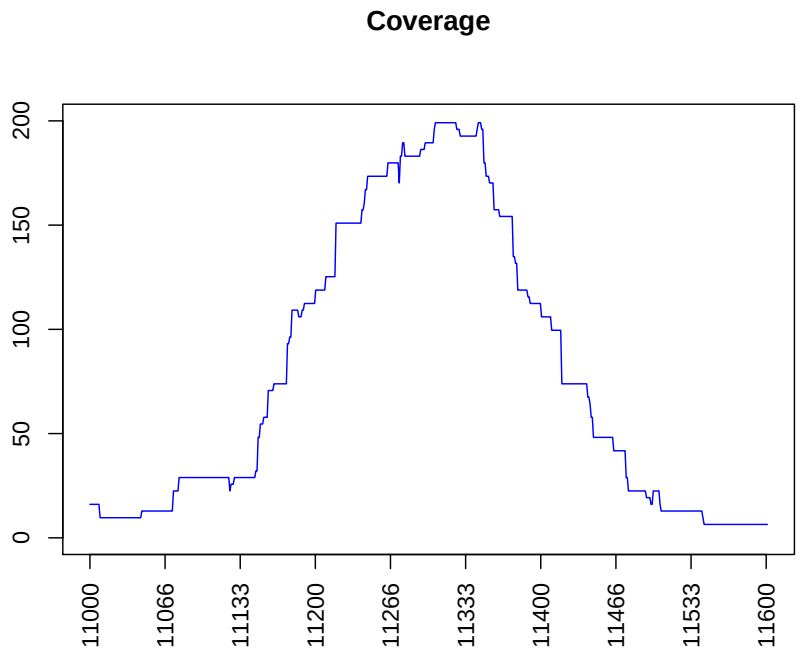
Peak Calling with BayesPeak

Compute coverage and call peaks

```
> library(BayesPeak)
> sig <- as.data.frame(as(readGAlignmentsFromBam(samplespath[1]), "GRanges"))[,-4]
> bgr <- as.data.frame(as(readGAlignmentsFromBam(samplespath[3]), "GRanges"))[,-4]
> colnames(sig) <- c("chr", "start", "end", "strand")
> colnames(bgr) <- c("chr", "start", "end", "strand")
> raw.output <- bayespeak(treatment=sig, control=bgr, start = 1, end = 100000)

..............

> # unreliable.jobs <- log(raw.output$QC$lambda1) < 1.5 # Removal of false positives due to overfitting.
> # bpeaks <- as.data.frame(summarise.peaks(raw.output, method = "lowerbound", exclude.jobs = unreliable.jo
> bpeaks <- as.data.frame(summarise.peaks(raw.output, method = "lowerbound"))
> source("./data/Fct/chipseqFct.R") # Imports the rangeCoverage function.
> sigcovDF <- rangeCoverage(summaryFct=viewMeans, myname="sig_", peaksIR=bpeaks[,1:3], sig=sig, readextend=
> bgrcovDF <- rangeCoverage(summaryFct=viewMeans, myname="bgr_", peaksIR=bpeaks[,1:3], sig=bgr, readextend=
> bpeaksDF <- cbind(bpeaks, sigcovDF[,-1], bgrcovDF[,-1])
> bpeaksDF[1:4,]

  space start   end width        PP   sig_cov sig_cov.pos sig_cov.neg     bgr_cov bgr_cov.pos bgr_cov.neg
1  Chr1  8301  8401   101 0.5072765 0.1907938   0.1271959  0.06359793 0.039605386 0.039605386 0.000000000
2  Chr1 11151 11351   201 0.9998914 0.6391434   0.5912077  0.04793576 0.009950607 0.009950607 0.000000000
3  Chr1 13601 13751   151 0.9998761 0.4041207   0.2339646  0.17015606 0.092718570 0.079473060 0.013245510
4  Chr1 16601 16951   351 0.9999604 0.4941070   0.2379034  0.25620365 0.011396422 0.005698211 0.005698211
```

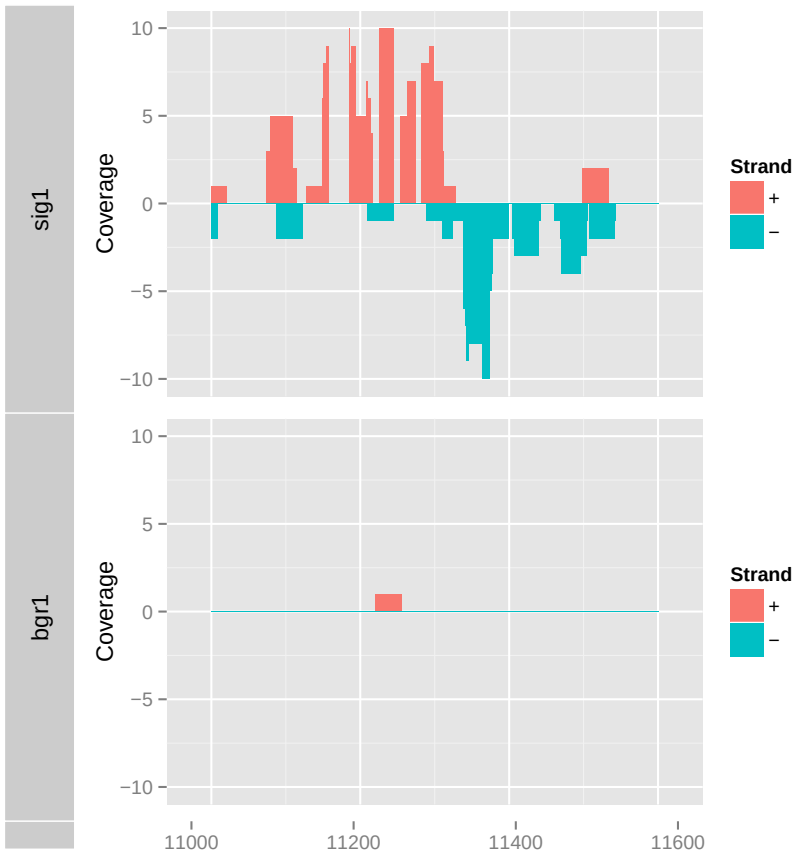
Coverage Plot

```
> plotCovBase(mycov=chip_signal_list[[1]], mychr="Chr1", mypos=c(11000, 11600), yli
```



Coverage Plot with plotCov

```
> p1 <- plotCov(bampath=samplespath[1], mychr="Chr1", mystart=11000, myend=11600, yn
> p2 <- plotCov(bampath=samplespath[3], mychr="Chr1", mystart=11000, myend=11600, yn
> tracks(sig1=p1, bgr1=p2)
```



Identify Common Peaks Among Two Methods

Compares results from simple cutoff method with BayesPeak results

```
> simple_peak <- as.data.frame(as(chip_peak_list[[1]], "IRangesList"))
> colnames(simple_peak)[2] <- "space"
> commonpeaks <- subsetByOverlaps(as(bpeaks, "RangedData"), as(simple_peak, "RangedData"), minoverlap=100)
> bpeaksDF[bpeaksDF$start %in% start(commonpeaks),][1:4,]
```

	space	start	end	width	PP	sig_cov	sig_cov.pos	sig_cov.neg	bgr_cov	bgr_cov.pos	bgr_cov.neg
1	Chr1	8301	8401	101	0.5072765	0.1907938	0.1271959	0.06359793	0.039605386	0.039605386	0.000000000
2	Chr1	11151	11351	201	0.9998914	0.6391434	0.5912077	0.04793576	0.009950607	0.009950607	0.000000000
3	Chr1	13601	13751	151	0.9998761	0.4041207	0.2339646	0.17015606	0.092718570	0.079473060	0.013245510
4	Chr1	16601	16951	351	0.9999604	0.4941070	0.2379034	0.25620365	0.011396422	0.005698211	0.005698211

Exercise 1: Compare Results with Published Peaks

Task 1 Import peaks predicted by Kaufmann et al. (2010).

Task 2 Determine how many of the published peaks have at least a 50% length overlap with the results from the BayesPeak and the naive peak calling methods.

Required information:

```
> pubpeaks <- read.delim("./data/Kaufmann_peaks100k.txt") # Published peaks for first 100kbp on chromosomes
> pubpeaks <- pubpeaks[order(pubpeaks$space, pubpeaks$start),]
> pubpeaks[1:4,]

              PeakID space start   end score_position     score length
chr1_3132   chr1_3132  Chr1  3094  3172           3132  4.656515     79
chr1_8365   chr1_8365  Chr1  8222  8425           8365 11.046212    204
chr1_11298 chr1_11298  Chr1 11149 11440          11298 52.109592    292
chr1_13686 chr1_13686  Chr1 13602 13756          13686  5.341907    155

> # Import olRanges function, which accepts two GRranges (IRanges) objects
> source("./data/Fct/rangeoverlapper.R")
```

Outline

Import Annotation Data from GFF

Annotation data from GFF

```
> library(rtracklayer); library(GenomicRanges); library(Rsamtools)
> gff <- import.gff("./data/TAIR10_GFF3_trunc.gff", asRangedData=FALSE)
> seqlengths(gff) <- end(ranges(gff[which(values(gff)[,"type"]=="chromosome"),]))
> subgene_index <- which(values(gff)[,"type"] == "gene")
> gffsub <- gff[subgene_index,] # Returns only gene ranges
> strand(gffsub) <- "*" # For strand insensitive analysis
> gffsub[1:4,1:2]

GRanges object with 4 ranges and 2 metadata columns:
      seqnames         ranges strand |   source     type
         <Rle>      <IRanges>  <Rle> | <factor> <factor>
  [1]     Chr1 [ 3631,  5899]      * |   TAIR10     gene
  [2]     Chr1 [ 5928,  8737]      * |   TAIR10     gene
  [3]     Chr1 [11649, 13714]      * |   TAIR10     gene
  [4]     Chr1 [23146, 31227]      * |   TAIR10     gene
  -------
  seqinfo: 7 sequences from an unspecified genome

> ids <- values(gffsub)[, "group"]
> gffgene <- gffsub
> gffsub <- split(gffsub, ids) # Coerce to GRangesList
```

Annotate Peaks with ChIPpeakAnno

```
> library(ChIPpeakAnno)
> annoRD <- unlist(gffsub)
> names(annoRD) <- gsub(".*=", "", values(annoRD)[, "group"])
> annoRD <- as(annoRD, "RangedData")
> peaksRD <- RangedData(space=bpeaksDF$space, IRanges(bpeaksDF$start, bpeaksDF$end))
> annotatedPeak <- annotatePeakInBatch(peaksRD, AnnotationData = annoRD)
> as.data.frame(annotatedPeak)[1:4,1:11]
```

	space	start	end	width	names	peak	strand	feature	start_position	end_position	insideFeature
1	Chr1	8301	8401	101	01 AT1G01020	01	+	AT1G01020	5928	8737	inside
2	Chr1	11151	11351	201	02 AT1G01030	02	+	AT1G01030	11649	13714	upstream
3	Chr1	13601	13751	151	03 AT1G01030	03	+	AT1G01030	11649	13714	overlapEnd
4	Chr1	16601	16951	351	04 AT1G01030	04	+	AT1G01030	11649	13714	downstream

```
> bpeaksDF[1:4,]
```

	space	start	end	width	PP	sig_cov	sig_cov.pos	sig_cov.neg	bgr_cov	bgr_cov.pos	bgr_cov.neg
1	Chr1	8301	8401	101	0.5072765	0.1907938	0.1271959	0.06359793	0.039605386	0.039605386	0.000000000
2	Chr1	11151	11351	201	0.9998914	0.6391434	0.5912077	0.04793576	0.009950607	0.009950607	0.000000000
3	Chr1	13601	13751	151	0.9998761	0.4041207	0.2339646	0.17015606	0.092718570	0.079473060	0.013245510
4	Chr1	16601	16951	351	0.9999604	0.4941070	0.2379034	0.25620365	0.011396422	0.005698211	0.005698211

Alternative Peak Annotation Approach

Alternative approach using olRanges function

```
> source("./data/Fct/rangeoverlapper.R")
> olRanges(query=gffgene, subject=as(as(bpeaks, "RangedData"), "GRanges"), output="df")[1:2,]

  space Qindex Sindex Qstart  Qend Sstart  Send OLstart OLend OLlength  OLpercQ   OLpercS OLtype
1  Chr1      2      1   5928  8737   8301  8401    8301  8401      101 3.594306 100.00000 inside
2  Chr1      3      3  11649 13714  13601 13751   13601 13714      114 5.517909  75.49669 oldown

> as.data.frame(annotatedPeak)[c(2,5),1:11] # Corresponding result from ChIPpeakAnno

  space start   end width        names peak strand   feature start_position end_position insideFeature
2  Chr1 11151 11351   201 02 AT1G01030   02      + AT1G01030          11649        13714      upstream
5  Chr1 20901 21101   201 05 AT1G01040   05      + AT1G01040          23146        31227      upstream
```

Outline

Read Counting for Peaks

Store peak ranges in GRanges container

```
> peakranges <- GRanges(seqnames = Rle(bpeaksDF$space), ranges = IRanges(bpeaksDF$st
+                 strand = Rle(strand("*")), peakIDs=paste("peak", seq(along=bpeaksDF
```

Count reads with summarizeOverlaps function from the GenomicRanges package.

```
> library(GenomicRanges)
> bfl <- BamFileList(samplespath, yieldSize=50000, index=character())
> countDF <- summarizeOverlaps(peakranges, bfl, mode="Union", ignore.strand=TRUE)
> countDF <- assays(countDF)$counts
> colnames(countDF) <- samples; rownames(countDF) <- values(peakranges)$peakIDs
> countDF[1:4,]
```

	SRR038845.fastq	SRR038846.fastq	SRR038848.fastq	SRR038850.fastq
peak_1	9	36	2	5
peak_2	54	69	1	9
peak_3	31	13	7	3
peak_4	68	87	2	8

```
> write.table(countDF, "./results/countDF.xls", quote=FALSE, sep="\t", col.names = T
> countDF <- read.table("./results/countDF.xls")
```

Simple RPKM Normalization

RPKM: here defined as reads per kilobase of sequence range per million mapped reads

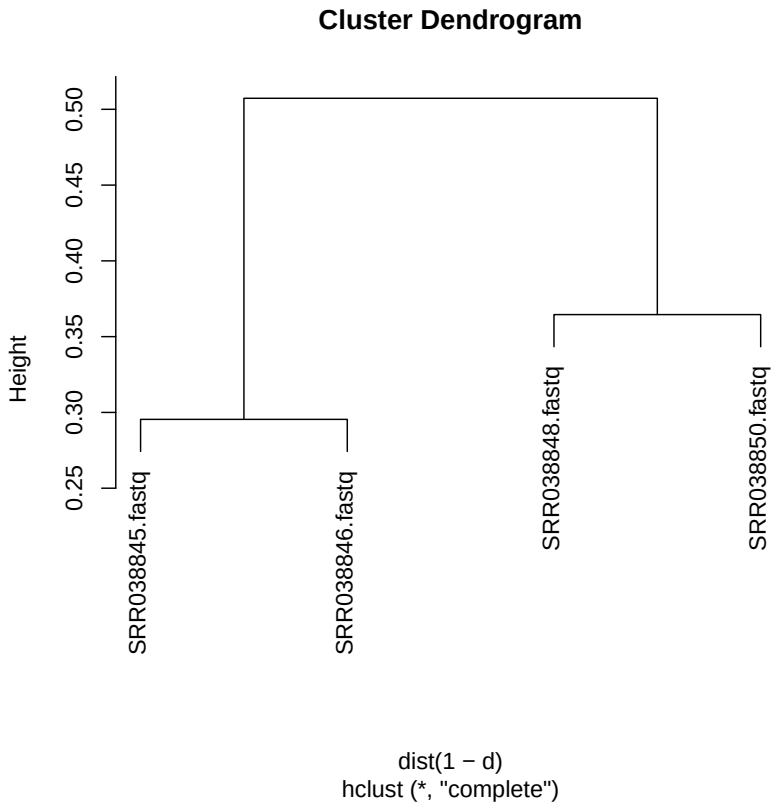
```
> returnRPKM <- function(counts, ranges) {
+         peakLengthsInKB <- width(ranges)/1000 # Number of bases per sequence range in kbp
+         millionsMapped <- sum(counts)/1e+06 # Factor for converting to million of mapped reads.
+         rpm <- counts/millionsMapped # RPK: reads per kilobase of sequence range.
+         rpkm <- rpm/peakLengthsInKB # RPKM: reads per kilobase of sequence range per million mapped reads
+         return(rpkm)
+ }
> countDFrpkm <- apply(countDF, 2, function(x) returnRPKM(counts=x, ranges=peakranges))
> countDFrpkm[1:4,]
```

	SRR038845.fastq	SRR038846.fastq	SRR038848.fastq	SRR038850.fastq
peak_1	3860.370	12947.172	623.0761	1968.3877
peak_2	11638.726	12469.436	156.5440	1780.3626
peak_3	8893.905	3127.233	1458.6582	789.9622
peak_4	8392.852	9003.387	179.2897	906.2435

QC Check

QC check by computing a sample correlating matrix and plotting it as a tree

```
> d <- cor(countDFrpkm, method="spearman")
> plot(hclust(dist(1-d))) # Sample tree
```



Identify DiffPeaks with Simple Fold Change Method

Compute mean values for replicates

```
> source("./data/Fct/colAg.R")
> countDFrpkm_mean <- colAg(myMA=countDFrpkm, group=c(1,1,2,2), myfct=mean)
> countDFrpkm_mean[1:4,]
```

```
       SRR038845.fastq_SRR038846.fastq SRR038848.fastq_SRR038850.fastq
peak_1                        8403.771                        1295.7319
peak_2                       12054.081                         968.4533
peak_3                        6010.569                        1124.3102
peak_4                        8698.119                         542.7666
```

Log2 fold changes

```
> countDFrpkm_mean <- cbind(countDFrpkm_mean, log2ratio=log2(countDFrpkm_mean[,1]/countDFrpkm_mean[,2]))
> countDFrpkm_mean <- countDFrpkm_mean[is.finite(countDFrpkm_mean[,3]), ]
> degs2fold <- countDFrpkm_mean[countDFrpkm_mean[,3] >= 1 | countDFrpkm_mean[,3] <= -1,]
> degs2fold[1:4,]
```

```
       SRR038845.fastq_SRR038846.fastq SRR038848.fastq_SRR038850.fastq log2ratio
peak_1                        8403.771                        1295.7319  2.697270
peak_2                       12054.081                         968.4533  3.637695
peak_3                        6010.569                        1124.3102  2.418461
peak_4                        8698.119                         542.7666  4.002300
```

```
> write.table(degs2fold, "./results/degs2fold", quote=FALSE, sep="\t", col.names = NA)
> degs2fold <- read.table("./results/degs2fold")
```

Identify DiffPeaks with DESeq Library

Raw count data are expected here!

```
> library(DESeq)
> countDF <- read.table("./results/countDF.xls")
> conds <- targets$Factor
> cds <- newCountDataSet(countDF, conds) # Creates object of class CountDataSet derived from eSet class
> counts(cds)[1:4, ] # CountDataSet has similar accessor methods as eSet class.

      SRR038845.fastq SRR038846.fastq SRR038848.fastq SRR038850.fastq
peak_1               9              36               2               5
peak_2              54              69               1               9
peak_3              31              13               7               3
peak_4              68              87               2               8

> cds <- estimateSizeFactors(cds) # Estimates library size factors from count data. Alternatively, one can
> cds <- estimateDispersions(cds, fitType="local") # Estimates the variance within replicates
> res <- nbinomTest(cds, "bgr1", "sig1") # Calls DEGs with nbinomTest
> res <- na.omit(res)
> res2fold <- res[res$log2FoldChange >= 1 | res$log2FoldChange <= -1,]
> res2foldpadj <- res2fold[res2fold$padj <= 0.2, ] # Here padj set very high for demo purpose
> res2foldpadj[1:2,1:8]

     id baseMean baseMeanA baseMeanB foldChange log2FoldChange        pval       padj
2 peak_2 24.58011  7.378231  41.78199   5.662874       2.501534 0.0019269419 0.02055405
4 peak_4 30.15418  7.657406  52.65094   6.875820       2.781532 0.0002409555 0.01542115
```

Identify DiffPeaks with edgeR Library

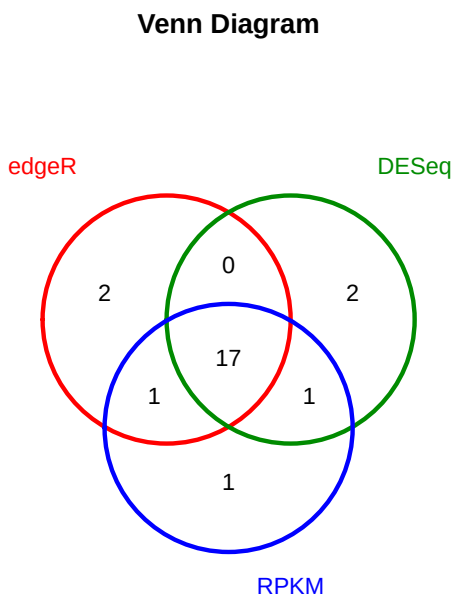
Raw count data are expected here!

```
> library(edgeR)
> countDF <- read.table("./results/countDF.xls")
> y <- DGEList(counts=countDF, group=conds) # Constructs DGEList object
> y <- estimateCommonDisp(y) # Estimates common dispersion
> y <- estimateTagwiseDisp(y) # Estimates tagwise dispersion
> et <- exactTest(y, pair=c("bgr1", "sig1")) # Computes exact test for the negative binomial distribution.
> topTags(et, n=4)

Comparison of groups:  sig1-bgr1
           logFC   logCPM       PValue          FDR
peak_32 3.829373 10.62902 2.299470e-10 1.494656e-08
peak_4  4.032856 10.72344 6.118597e-08 1.988544e-06
peak_43 3.651178 11.02320 1.227632e-07 2.659870e-06
peak_29 3.201302 11.26018 1.367281e-06 2.221832e-05

> edge <- as.data.frame(topTags(et, n=50000))
> edge2fold <- edge[edge$logFC >= 1 | edge$logFC <= -1,]
> edge2foldpadj <- edge2fold[edge2fold$FDR <= 0.01, ]
```

Merge Results and Compute Overlaps Among Methods

Here overlaps for 20 best ranking peaks of each method!

```
> bothDF <- merge(res, countDFrpkm_mean, by.x=1, by.y=0, all=TRUE); bothDF <- na.omit(bothDF)
> cor(bothDF[,"log2FoldChange"], bothDF[,"log2ratio"], method="spearman")
[1] 0.9906593
> source("./data/Fct/overLapper.R")
> setlist <- list(edgeR=rownames(edge[order(edge$FDR),][1:20,]),
+                 DESeq=res[order(res$padj),][1:20,"id"],
+                 RPKM=rownames(degs2fold[order(-degs2fold$log2ratio),][1:20,]))
> OLlist <- overLapper(setlist=setlist, sep="_", type="vennsets")
> counts <- sapply(OLlist$Venn_List, length)
> vennPlot(counts=counts)
```



Outline

Inspect Results in IGV

View peak_3 in IGV

- Download and open IGV Link
- Select in menu in top left corner `A. thaliana (TAIR10)`
- Upload the following indexed/sorted Bam files with `File -> Load from URL...`

http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038845.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038846.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038848.fastq.bam
http://faculty.ucr.edu/~tgirke/HTML_Presentations/Manuals/Rngsapps/chipseqBioc2012/results/SRR038850.fastq.bam

- To view peak_3, enter its coordinates `Chr1:16656-16956` in position menu on top.

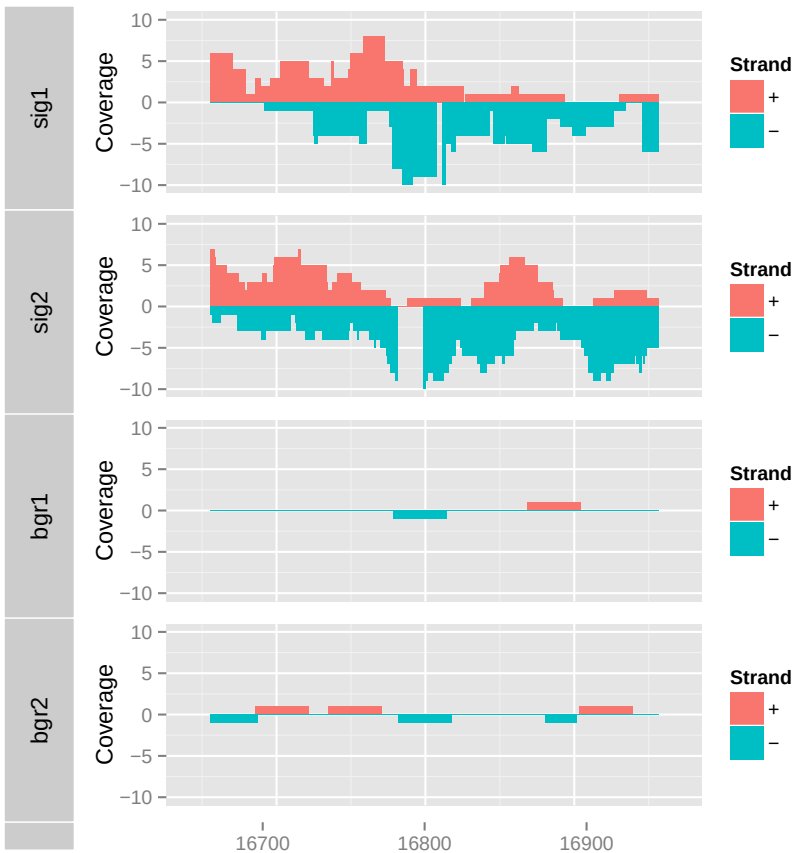
Controlling IGV from R

Create previous IGV session with required tracks automatically, and direct it to the desired position, here `Chr1:16656-16956`.

```
> library(SRAdb)
> startIGV("lm")
> sock <- IGVsocket()
> session <- IGVsession(files=samplespath,
+                       sessionFile="session.xml",
+                       genome="A. thaliana (TAIR10)")
> IGVload(sock, session)
> IGVgoto(sock, 'Chr1:16656-16956')
```

Plot Peak Programmatically with *ggbio*

```
> p1 <- plotCov(bampath=samplespath[1], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> p2 <- plotCov(bampath=samplespath[2], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> p3 <- plotCov(bampath=samplespath[3], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> p4 <- plotCov(bampath=samplespath[4], mychr="Chr1", mystart=16656, myend=16956, ymin=-10, ymax=10)
> tracks(sig1=p1, sig2=p2, bgr1=p3, bgr2=p4)
```



Outline

Sequence Motifs Enriched in Peak Sequences I

Extract peak sequences and predict enriched motifs with BCRANK library

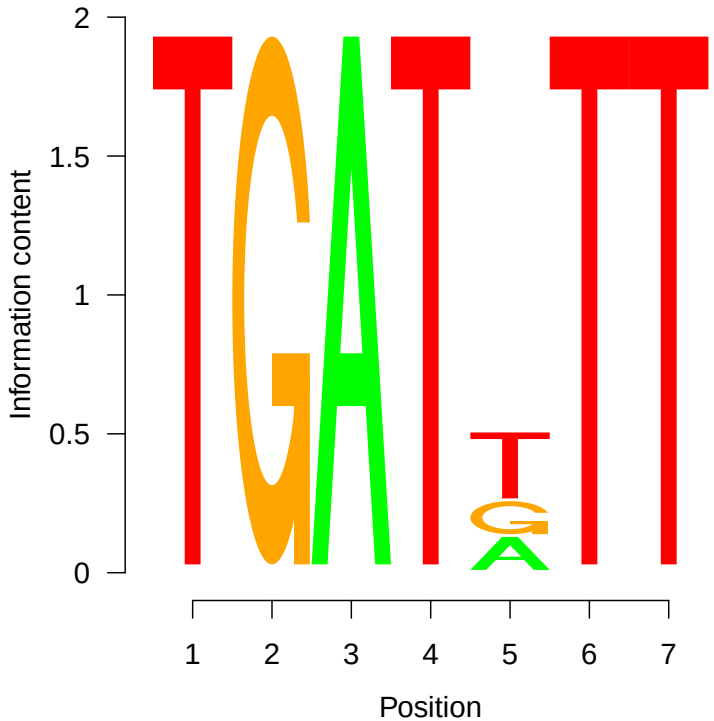
```
> library(Biostrings); library(seqLogo); library(BCRANK)
> gr <- as(as(bpeaksDF, "RangedData"), "GRanges"); gr <- gr[start(gr)>=1]
> pseq <- getSeq(FaFile("./data/tair10chr.fasta"), gr)
> names(pseq) <- paste(as.character(seqnames(gr)), start(gr), sep="_")
> writeXStringSet(pseq[1:8], "./results/pseq.fasta") # Note: reduced to 8 sequences to run quickly.
> set.seed(0)
> BCRANKout <- bcrank("./results/pseq.fasta", restarts=25, use.P1=TRUE, use.P2=TRUE)
```

```
**** Running BCRANK on 8 regions, starting from HGRMHGHVSS ****
Iteration 1 - HGRMHGHVSS: 0
Scanning sequences............
Computing scores..............

Iteration 2 - AGRMHGHVSS: 7.673544
Scanning sequences............
Computing scores..............

Iteration 3 - AGAMHGHVSS: 10.23139
Scanning sequences............
Computing scores..............

Iteration 4 - AGAAHGHVSS: 12.78924
Scanning sequences............
Computing scores..............

Iteration 5 - AGAATGHVSS: 15.34709
Scanning sequences............
Computing scores..............

Iteration 6 - AGAATGTVSS: 17.90494
Scanning sequences............
Computing scores..............
```

Sequence Motifs Enriched in Peak Sequences II

Plot BCRANK result

```
> topMotif <- toptable(BCRANKout, 1)
> weightMatrix <- pwm(topMotif, normalize = FALSE)
> weightMatrixNormalized <- pwm(topMotif, normalize = TRUE)
> seqLogo(weightMatrixNormalized)
```



Exercise 2: Motif Enrichment Analysis

Task 1 Extract from the BCRANK result stored in `weightMatrix` the motif occurrence patterns and generate with them a position weight matrix using the `PWM` function from Biostrings.

Task 2 Enumerate the motif matches in the peak sequences and the entire genome using Biostring's `countPWM` function.

Task 3 Determine which sequence type, peak or genome, shows more matches per 1kbp sequence for this motif.

Task 4 Homework: write a function for computing enrichment p-values for motif matches based on the hypergeometric distribution.

Session Information

```
> sessionInfo()

R version 3.1.2 (2014-10-31)
Platform: x86_64-unknown-linux-gnu (64-bit)
locale:
[1] C

attached base packages:
 [1] grid      stats4    parallel  stats     graphics  utils     datasets  grDevices methods   base

other attached packages:
 [1] BCRANK_1.28.0           seqLogo_1.32.1          edgeR_3.8.2             limma_3.22.1            DESe
[10] Biobase_2.26.0          RSQLite_1.0.0           DBI_0.3.1               biomaRt_2.22.0          VennD
[19] BayesPeak_1.18.0        ggbio_1.14.0            ggplot2_1.0.0           GenomicAlignments_1.2.1 Rsamt
[28] GenomeInfoDb_1.2.3      IRanges_2.0.0           S4Vectors_0.4.0         BiocGenerics_0.12.1

loaded via a namespace (and not attached):
 [1] BBmisc_1.8               BatchJobs_1.5            Formula_1.1-2            GGally_0.4.8             G
 [9] OrganismDbi_1.8.0        RBGL_1.42.0              RColorBrewer_1.0-5       RCurl_1.95-4.3           R
[17] annotate_1.44.0          base64enc_0.1-2          biovizBase_1.14.0        bitops_1.0-6             b
[25] colorspace_1.2-4         dichromat_2.0-0          digest_0.6.4             fail_1.2                 f
[33] graph_1.44.0             gridExtra_0.9.1          gtable_0.1.2             hwriter_1.3.2            h
[41] munsell_0.4.2            nnet_7.3-8               plyr_1.8.1               proto_0.3-10             r
[49] sendmailR_1.2-1          splines_3.1.2            stringr_0.6.2            survival_2.37-7          t
```

Outline

References I

Kaufmann, K., Wellmer, F., Muiño, J. M., Ferrier, T., Wuest, S. E., Kumar, V., Serrano-Mislata, A., Madueño, F., Krajewski, P., Meyerowitz, E. M., Angenent, G. C., Riechmann, J. L., Apr 2010. Orchestration of floral initiation by APETALA1. Science 328 (5974), 85–89.
URL http://www.hubmed.org/display.cgi?uids=20360106

Pepke, S., Wold, B., Mortazavi, A., Nov 2009. Computation for ChIP-seq and RNA-seq studies. Nat Methods 6 (11 Suppl), 22–32.
URL http://www.hubmed.org/display.cgi?uids=19844228