ChIP-Seq Workflow Template

First/last name (first.last@ucr.edu)

Last update: 09 May, 2018

Introduction

Users want to provide here background information about the design of their ChIP-Seq project.

Background and objectives

This report describes the analysis of several ChIP-Seq experiments studying the DNA binding patterns of the transcriptions factors ... from *organism* ....

Experimental design

Typically, users want to specify here all information relevant for the analysis of their NGS study. This includes detailed descriptions of FASTQ files, experimental design, reference genome, gene annotations, etc.

Generate workflow environment

Load workflow environment with sample data into your current working directory. The sample data are described here.

In the workflow environments generated by `genWorkenvir` all data inputs are stored in a `data/` directory and all analysis results will be written to a separate `results/` directory, while the `systemPipeChIPseq.Rmd` script and the `targets` file are expected to be located in the parent directory. The R session is expected to run from this parent directory. Additional parameter files are stored under `param/`.

To work with real data, users want to organize their own data similarly and substitute all test data for their own data. To rerun an established workflow on new data, the initial `targets` file along with the corresponding FASTQ files are usually the only inputs the user needs to provide.

```
library(systemPipeRdata)
genWorkenvir(workflow="chipseq")
setwd("chipseq")
```

Alternatively, this can be done from the command-line as follows:

```
$ Rscript -e "systemPipeRdata::genWorkenvir(workflow='chipseq')"
$ cd chipseq
```

Now download the latest `systemPipeChIPseq.Rmd` script for this course. From within R this can be done as follows.

```
download.file("https://raw.githubusercontent.com/tgirke/GEN242/gh-pages/_vignettes/12_ChIPseqWorkflow/sy
```

Or from the command-line one can do this with `wget`.

```
$ wget -O systemPipeChIPseq.Rmd https://raw.githubusercontent.com/tgirke/GEN242/gh-pages/_vignettes/12_C
```

Now log in to a computer node on the HPCC/biocluser. The following command sequence will connect the user from the command-line to a computer node on the cluster.

```
$ srun --x11 --partition=short --mem=2gb --cpus-per-task 1 --ntasks 1 --time 2:00:00 --pty bash -l
```

Load desired R version from module system (here R-3.4.2).

```
$ module load R/3.4.2
```

Now open the R markdown script `systemPipeChIPseq.Rmd`in your R IDE (*e.g.* nvim-r or RStudio) and run the workflow as outlined below.

Note, Tmux sessions should always run on one of the headnodes and never on the computer nodes themsleves. This is important since Tmux sessions are persistent meaning they don't close automatically when a computer job finishes. Thus, they are not controlled by the queueing system.

To check the environment of R session, one can execute the following commands from R. The first line returns the node name of the R session.

```
system("hostname") # should return name of a compute node starting with i or c
getwd() # checks current working directory of R session
dir() # returns content of current working directory
```

Required packages and resources

The `systemPipeR` package needs to be loaded to perform the analysis steps shown in this report (H Backman and Girke 2016).

```
library(systemPipeR)
```

If applicable users can load custom functions not provided by `systemPipeR`. Skip this step if this is not the case.

```
source("systemPipeChIPseq_Fct.R")
```

Read preprocessing

Experiment definition provided by `targets` file

The `targets` file defines all FASTQ files and sample comparisons of the analysis workflow.

```
targetspath <- system.file("extdata", "targets_chip.txt", package="systemPipeR")
targets <- read.delim(targetspath, comment.char = "#")
targets[1:4,-c(5,6)]
```

```
##                   FileName SampleName Factor SampleLong SampleReference
## 1 ./data/SRR446027_1.fastq        M1A     M1  Mock.1h.A
## 2 ./data/SRR446028_1.fastq        M1B     M1  Mock.1h.B
## 3 ./data/SRR446029_1.fastq        A1A     A1   Avr.1h.A             M1A
## 4 ./data/SRR446030_1.fastq        A1B     A1   Avr.1h.B             M1B
```

Read quality filtering and trimming

The following example shows how one can design a custom read preprocessing function using utilities provided by the `ShortRead` package, and then apply it with `preprocessReads` in batch mode to all FASTQ samples

referenced in the corresponding SYSargs instance (args object below). More detailed information on read preprocessing is provided in systemPipeR's main vignette.

```
args <- systemArgs(sysma="param/trim.param", mytargets="targets_chip.txt")
filterFct <- function(fq, cutoff=20, Nexceptions=0) {
    qcount <- rowSums(as(quality(fq), "matrix") <= cutoff)
    fq[qcount <= Nexceptions] # Retains reads where Phred scores are >= cutoff with N exceptions
}
preprocessReads(args=args, Fct="filterFct(fq, cutoff=20, Nexceptions=0)", batchsize=100000)
writeTargetsout(x=args, file="targets_chip_trim.txt", overwrite=TRUE)
```

FASTQ quality report

The following seeFastq and seeFastqPlot functions generate and plot a series of useful quality statistics for a set of FASTQ files including per cycle quality box plots, base proportions, base-level quality trends, relative k-mer diversity, length and occurrence distribution of reads, number of reads above quality cutoffs and mean quality distribution. The results are written to a PDF file named fastqReport.pdf.

```
args <- systemArgs(sysma="param/tophat.param", mytargets="targets_chip.txt")
library(BiocParallel); library(BatchJobs)
f <- function(x) {
    library(systemPipeR)
    args <- systemArgs(sysma="param/tophat.param", mytargets="targets_chip.txt")
    seeFastq(fastq=infile1(args)[x], batchsize=100000, klength=8)
}
funs <- makeClusterFunctionsSLURM("slurm.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="00:20:00", ntasks=1, ncpus=1, memory="2G
register(param)
fqlist <- bplapply(seq(along=args), f)
pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
seeFastqPlot(unlist(fqlist, recursive=FALSE))
dev.off()
```

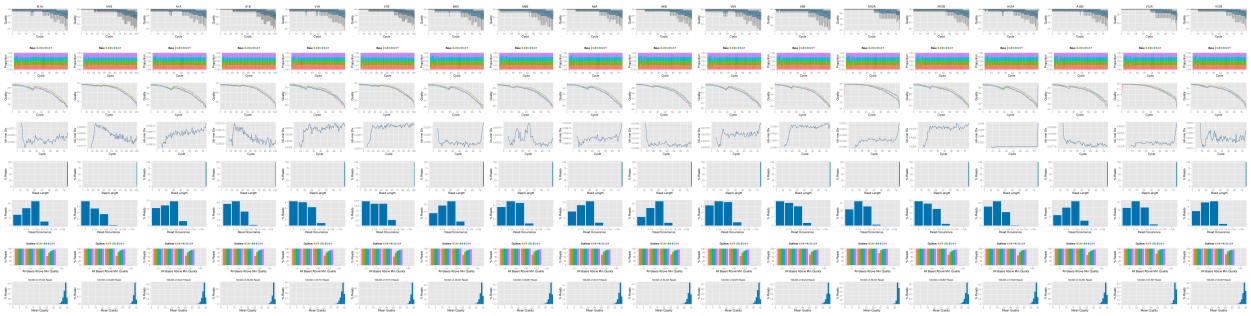

Figure 1: FASTQ quality report for 18 samples

Alignments

Read mapping with Bowtie2

The NGS reads of this project will be aligned with Bowtie2 against the reference genome sequence (Langmead and Salzberg 2012). The parameter settings of the aligner are defined in the bowtieSE.param file. In ChIP-Seq experiments it is usually more appropriate to eliminate reads mapping to multiple locations. To

achieve this, users want to remove the argument setting `-k 50 non-deterministic` in the `bowtieSE.param` file.

The following submits 18 alignment jobs via a scheduler to a computer cluster.

```
args <- systemArgs(sysma="param/bowtieSE.param", mytargets="targets_chip_trim.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
moduleload(modules(args)) # Skip if a module system is not used
system("bowtie2-build ./data/tair10.fasta ./data/tair10.fasta") # Indexes reference genome
resources <- list(walltime="1:00:00", ntasks=1, ncpus=cores(args), memory="10G")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="slurm.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
writeTargetsout(x=args, file="targets_bam.txt", overwrite=TRUE)
```

Alternatively, one can run the alignments sequentially on a single system.

```
runCommandline(args)
```

Check whether all BAM files have been created

```
file.exists(outpaths(args))
```

Read and alignment stats

The following provides an overview of the number of reads in each sample and how many of them aligned to the reference.

```
read_statsDF <- alignStats(args=args)
write.table(read_statsDF, "results/alignStats.xls", row.names=FALSE, quote=FALSE, sep="\t")
read.delim("results/alignStats.xls")
```

Create symbolic links for viewing BAM files in IGV

The `symLink2bam` function creates symbolic links to view the BAM alignment files in a genome browser such as IGV without moving these large files to a local system. The corresponding URLs are written to a file with a path specified under `urlfile`, here `IGVurl.txt`.

```
symLink2bam(sysargs=args, htmldir=c("~/.html/", "somedir/"),
            urlbase="http://biocluster.ucr.edu/~tgirke/",
            urlfile="./results/IGVurl.txt")
```

Utilities for coverage data

The following introduces several utilities useful for ChIP-Seq data. They are not part of the actual workflow.

Rle object stores coverage information

```
library(rtracklayer); library(GenomicRanges); library(Rsamtools); library(GenomicAlignments)
aligns <- readGAlignments(outpaths(args)[1])
cov <- coverage(aligns)
cov
```

Resizing aligned reads

```
trim(resize(as(aligns, "GRanges"), width = 200))
```

Naive peak calling

```
islands <- slice(cov, lower = 15)
islands[[1]]
```

Plot coverage for defined region

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

Peak calling with MACS2

Merge BAM files of replicates prior to peak calling

Merging BAM files of technical and/or biological replicates can improve the sensitivity of the peak calling by increasing the depth of read coverage. The `mergeBamByFactor` function merges BAM files based on grouping information specified by a `factor`, here the `Factor` column of the imported targets file. It also returns an updated `SYSargs` object containing the paths to the merged BAM files as well as to any unmerged files without replicates. This step can be skipped if merging of BAM files is not desired.

```
args <- systemArgs(sysma=NULL, mytargets="targets_bam.txt")
args_merge <- mergeBamByFactor(args, overwrite=TRUE)
writeTargetsout(x=args_merge, file="targets_mergeBamByFactor.txt", overwrite=TRUE)
```

Peak calling without input/reference sample

MACS2 can perform peak calling on ChIP-Seq data with and without input samples (Zhang et al. 2008). The following performs peak calling without input on all samples specified in the corresponding `args` object. Note, due to the small size of the sample data, MACS2 needs to be run here with the `nomodel` setting. For real data sets, users want to remove this parameter in the corresponding `*.param` file(s).

```
args <- systemArgs(sysma="param/macs2_noinput.param", mytargets="targets_mergeBamByFactor.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
runCommandline(args)
file.exists(outpaths(args))
writeTargetsout(x=args, file="targets_macs.txt", overwrite=TRUE)
```

Peak calling with input/reference sample

To perform peak calling with input samples, they can be most conveniently specified in the `SampleReference` column of the initial `targets` file. The `writeTargetsRef` function uses this information to create a `targets`

file intermediate for running MACS2 with the corresponding input samples.

```
writeTargetsRef(infile="targets_mergeBamByFactor.txt", outfile="targets_bam_ref.txt", silent=FALSE, ove
args_input <- systemArgs(sysma="param/macs2.param", mytargets="targets_bam_ref.txt")
sysargs(args_input)[1] # Command-line parameters for first FASTQ file
# unlink(outpaths(args_input)) # Note: if output exists then next line will not be run
runCommandline(args_input)
file.exists(outpaths(args_input))
writeTargetsout(x=args_input, file="targets_macs_input.txt", overwrite=TRUE)
```

The peak calling results from MACS2 are written for each sample to separate files in the `results` directory. They are named after the corresponding files with extensions used by MACS2.

Identify consensus peaks

The following example shows how one can identify consensus preaks among two peak sets sharing either a minimum absolute overlap and/or minimum relative overlap using the `subsetByOverlaps` or `olRanges` functions, respectively. Note, the latter is a custom function imported below by sourcing it.

```
source("http://faculty.ucr.edu/~tgirke/Documents/R_BioCond/My_R_Scripts/rangeoverlapper.R")
peak_M1A <- outpaths(args)["M1A"]
peak_M1A <- as(read.delim(peak_M1A, comment="#")[,1:3], "GRanges")
peak_A1A <- outpaths(args)["A1A"]
peak_A1A <- as(read.delim(peak_A1A, comment="#")[,1:3], "GRanges")
(myol1 <- subsetByOverlaps(peak_M1A, peak_A1A, minoverlap=1)) # Returns any overlap
myol2 <- olRanges(query=peak_M1A, subject=peak_A1A, output="gr") # Returns any overlap with OL length i
myol2[values(myol2)["OLpercQ"][,1]>=50] # Returns only query peaks with a minimum overlap of 50%
```

Annotate peaks with genomic context

Annotation with `ChIPpeakAnno` package

The following annotates the identified peaks with genomic context information using the `ChIPpeakAnno` and `ChIPseeker` packages, respectively (Zhu et al. 2010; Yu, Wang, and He 2015).

```
library(ChIPpeakAnno); library(GenomicFeatures)
args <- systemArgs(sysma="param/annotate_peaks.param", mytargets="targets_macs.txt")
# txdb <- loadDb("./data/tair10.sqlite")
txdb <- makeTxDbFromGFF(file="data/tair10.gff", format="gff", dataSource="TAIR", organism="Arabidopsis
ge <- genes(txdb, columns=c("tx_name", "gene_id", "tx_type"))
for(i in seq(along=args)) {
    peaksGR <- as(read.delim(infile1(args)[i], comment="#"), "GRanges")
    annotatedPeak <- annotatePeakInBatch(peaksGR, AnnotationData=genes(txdb))
    df <- data.frame(as.data.frame(annotatedPeak), as.data.frame(values(ge[values(annotatedPeak)$feature
    write.table(df, outpaths(args[i]), quote=FALSE, row.names=FALSE, sep="\t")
}
writeTargetsout(x=args, file="targets_peakanno.txt", overwrite=TRUE)
```

The peak annotation results are written for each peak set to separate files in the `results` directory. They are named after the corresponding peak files with extensions specified in the `annotate_peaks.param` file, here `*.peaks.annotated.xls`.

Annotation with ChIPseeker package

Same as in previous step but using the ChIPseeker package for annotating the peaks.

```
library(ChIPseeker)
for(i in seq(along=args)) {
    peakAnno <- annotatePeak(infile1(args)[i], TxDb=txdb, verbose=FALSE)
    df <- as.data.frame(peakAnno)
    write.table(df, outpaths(args[i]), quote=FALSE, row.names=FALSE, sep="\t")
}
writeTargetsout(x=args, file="targets_peakanno.txt", overwrite=TRUE)
```

Summary plots provided by the ChIPseeker package. Here applied only to one sample for demonstration purposes.

```
peak <- readPeakFile(infile1(args)[1])
covplot(peak, weightCol="X.log10.pvalue.")
peakHeatmap(outpaths(args)[1], TxDb=txdb, upstream=1000, downstream=1000, color="red")
plotAvgProf2(outpaths(args)[1], TxDb=txdb, upstream=1000, downstream=1000, xlab="Genomic Region (5'->3'
```

Count reads overlapping peaks

The countRangeset function is a convenience wrapper to perform read counting iteratively over serveral range sets, here peak range sets. Internally, the read counting is performed with the summarizeOverlaps function from the GenomicAlignments package. The resulting count tables are directly saved to files, one for each peak set.

```
library(GenomicRanges)
args <- systemArgs(sysma="param/count_rangesets.param", mytargets="targets_macs.txt")
args_bam <- systemArgs(sysma=NULL, mytargets="targets_bam.txt")
bfl <- BamFileList(outpaths(args_bam), yieldSize=50000, index=character())
countDFnames <- countRangeset(bfl, args, mode="Union", ignore.strand=TRUE)
writeTargetsout(x=args, file="targets_countDF.txt", overwrite=TRUE)
```

Differential binding analysis

The runDiff function performs differential binding analysis in batch mode for several count tables using edgeR or DESeq2 (Robinson, McCarthy, and Smyth 2010; Love, Huber, and Anders 2014). Internally, it calls the functions run_edgeR and run_DESeq2. It also returns the filtering results and plots from the downstream filterDEGs function using the fold change and FDR cutoffs provided under the dbrfilter argument.

```
args_diff <- systemArgs(sysma="param/rundiff.param", mytargets="targets_countDF.txt")
cmp <- readComp(file=args_bam, format="matrix")
dbrlist <- runDiff(args=args_diff, diffFct=run_edgeR, targets=targetsin(args_bam),
                   cmp=cmp[[1]], independent=TRUE, dbrfilter=c(Fold=2, FDR=1))
writeTargetsout(x=args_diff, file="targets_rundiff.txt", overwrite=TRUE)
```

GO term enrichment analysis

The following performs GO term enrichment analysis for each annotated peak set.

```
args <- systemArgs(sysma="param/macs2.param", mytargets="targets_bam_ref.txt")
args_anno <- systemArgs(sysma="param/annotate_peaks.param", mytargets="targets_macs.txt")
annofiles <- outpaths(args_anno)
gene_ids <- sapply(names(annofiles), function(x) unique(as.character(read.delim(annofiles[x])[,"geneId"]
load("data/GO/catdb.RData")
BatchResult <- GOCluster_Report(catdb=catdb, setlist=gene_ids, method="all", id_type="gene", CLSZ=2, cu
```

Motif analysis

Parse DNA sequences of peak regions from genome

Enrichment analysis of known DNA binding motifs or *de novo* discovery of novel motifs requires the DNA sequences of the identified peak regions. To parse the corresponding sequences from the reference genome, the `getSeq` function from the `Biostrings` package can be used. The following example parses the sequences for each peak set and saves the results to separate FASTA files, one for each peak set. In addition, the sequences in the FASTA files are ranked (sorted) by increasing p-values as expected by some motif discovery tools, such as `BCRANK`.

```
library(Biostrings); library(seqLogo); library(BCRANK)
args <- systemArgs(sysma="param/annotate_peaks.param", mytargets="targets_macs.txt")
rangefiles <- infile1(args)
for(i in seq(along=rangefiles)) {
    df <- read.delim(rangefiles[i], comment="#")
    peaks <- as(df, "GRanges")
    names(peaks) <- paste0(as.character(seqnames(peaks)), "_", start(peaks), "-", end(peaks))
    peaks <- peaks[order(values(peaks)$X.log10.pvalue., decreasing=TRUE)]
    pseq <- getSeq(FaFile("./data/tair10.fasta"), peaks)
    names(pseq) <- names(peaks)
    writeXStringSet(pseq, paste0(rangefiles[i], ".fasta"))
}
```

Motif discovery with `BCRANK`

The Bioconductor package `BCRANK` is one of the many tools available for *de novo* discovery of DNA binding motifs in peak regions of ChIP-Seq experiments. The given example applies this method on the first peak sample set and plots the sequence logo of the highest ranking motif.

```
set.seed(0)
BCRANKout <- bcrank(paste0(rangefiles[1], ".fasta"), restarts=25, use.P1=TRUE, use.P2=TRUE)
toptable(BCRANKout)
topMotif <- toptable(BCRANKout, 1)
weightMatrix <- pwm(topMotif, normalize = FALSE)
weightMatrixNormalized <- pwm(topMotif, normalize = TRUE)
pdf("results/seqlogo.pdf")
seqLogo(weightMatrixNormalized)
dev.off()
```

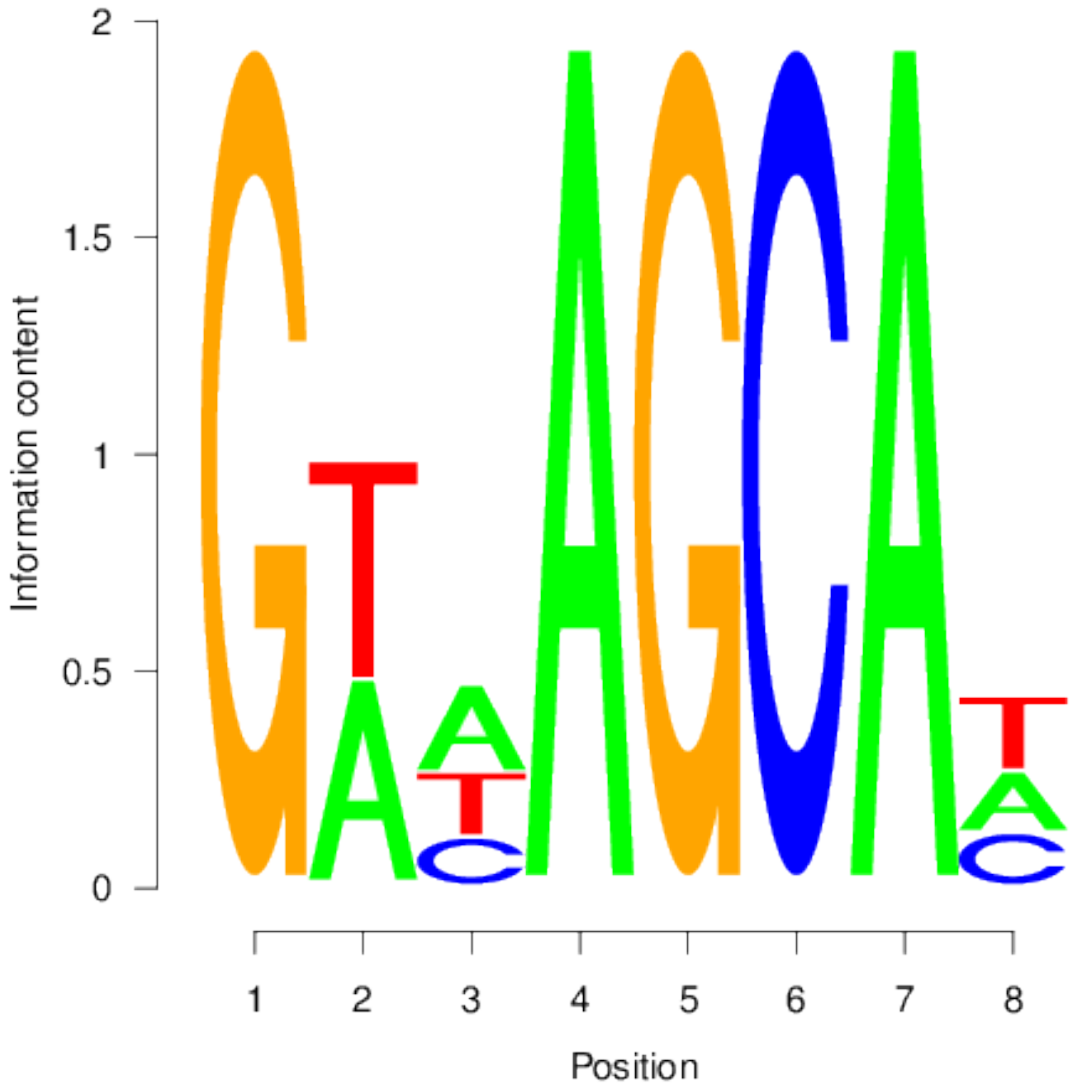

Figure 2: One of the motifs identified by `BCRANK`

Version Information

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 14.04.5 LTS
##
## Matrix products: default
## BLAS: /usr/lib/libblas/libblas.so.3.0
```

```
## LAPACK: /usr/lib/lapack/liblapack.so.3.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8
##  [4] LC_COLLATE=en_US.UTF-8     LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C                  LC_ADDRESS=C
## [10] LC_TELEPHONE=C             LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    parallel  methods   stats     graphics  utils     datasets  grDevices base
##
## other attached packages:
##  [1] ape_4.1                    ggplot2_2.2.1              systemPipeR_1.10.0
##  [4] ShortRead_1.34.0           GenomicAlignments_1.12.0   SummarizedExperiment_1.6.0
##  [7] DelayedArray_0.2.0         matrixStats_0.52.2         Biobase_2.36.0
## [10] BiocParallel_1.10.0        Rsamtools_1.28.0           Biostrings_2.44.0
## [13] XVector_0.16.0             GenomicRanges_1.28.0       GenomeInfoDb_1.12.0
## [16] IRanges_2.10.0             S4Vectors_0.14.0           BiocGenerics_0.22.0
## [19] BiocStyle_2.4.0
##
## loaded via a namespace (and not attached):
##  [1] edgeR_3.18.0            splines_3.4.0           latticeExtra_0.6-28     RBGL_1.52.0
##  [5] GenomeInfoDbData_0.99.0 yaml_2.1.14             Category_2.42.0         RSQLite_1.1-2
##  [9] backports_1.0.5         lattice_0.20-35         limma_3.32.0            digest_0.6.12
## [13] RColorBrewer_1.1-2      checkmate_1.8.2         colorspace_1.3-2        htmltools_0.3.5
## [17] Matrix_1.2-8            plyr_1.8.4              GSEABase_1.38.0         XML_3.98-1.6
## [21] pheatmap_1.0.8          biomaRt_2.32.0          genefilter_1.58.0       zlibbioc_1.22.0
## [25] xtable_1.8-2            GO.db_3.4.1             scales_0.4.1            brew_1.0-6
## [29] tibble_1.3.0            annotate_1.54.0         GenomicFeatures_1.28.0  lazyeval_0.2.0
## [33] survival_2.41-3         magrittr_1.5            memoise_1.1.0           evaluate_0.10
## [37] fail_1.3                nlme_3.1-131            hwriter_1.3.2           GOstats_2.42.0
## [41] graph_1.54.0            tools_3.4.0             BBmisc_1.11             stringr_1.2.0
## [45] sendmailR_1.2-1         munsell_0.4.3           locfit_1.5-9.1          AnnotationDbi_1.38.0
## [49] compiler_3.4.0          grid_3.4.0              RCurl_1.95-4.8          rjson_0.2.15
## [53] AnnotationForge_1.18.0  bitops_1.0-6            base64enc_0.1-3         rmarkdown_1.5
## [57] codetools_0.2-15        gtable_0.2.0            DBI_0.6-1               knitr_1.15.1
## [61] rtracklayer_1.36.0      rprojroot_1.2           stringi_1.1.5           BatchJobs_1.6
## [65] Rcpp_0.12.10
```

Funding

This project was supported by funds from the National Institutes of Health (NIH) and the National Science Foundation (NSF).

References

H Backman, Tyler W, and Thomas Girke. 2016. "systemPipeR: NGS workflow and report generation environment." *BMC Bioinformatics* 17 (1): 388. doi:10.1186/s12859-016-1241-0.

Langmead, Ben, and Steven L Salzberg. 2012. "Fast Gapped-Read Alignment with Bowtie 2." *Nat. Methods*

9 (4). Nature Publishing Group: 357–59. doi:10.1038/nmeth.1923.

Love, Michael, Wolfgang Huber, and Simon Anders. 2014. "Moderated Estimation of Fold Change and Dispersion for RNA-seq Data with DESeq2." *Genome Biol.* 15 (12): 550. doi:10.1186/s13059-014-0550-8.

Robinson, M D, D J McCarthy, and G K Smyth. 2010. "EdgeR: A Bioconductor Package for Differential Expression Analysis of Digital Gene Expression Data." *Bioinformatics* 26 (1): 139–40. doi:10.1093/bioinformatics/btp616.

Yu, Guangchuang, Li-Gen Wang, and Qing-Yu He. 2015. "ChIPseeker: An R/Bioconductor Package for ChIP Peak Annotation, Comparison and Visualization." *Bioinformatics* 31 (14): 2382–3. doi:10.1093/bioinformatics/btv145.

Zhang, Y, T Liu, C A Meyer, J Eeckhoute, D S Johnson, B E Bernstein, C Nussbaum, et al. 2008. "Model-Based Analysis of ChIP-Seq (MACS)." *Genome Biol.* 9 (9). doi:10.1186/gb-2008-9-9-r137.

Zhu, Lihua J, Claude Gazin, Nathan D Lawson, Hervé Pagès, Simon M Lin, David S Lapointe, and Michael R Green. 2010. "ChIPpeakAnno: A Bioconductor Package to Annotate ChIP-seq and ChIP-chip Data." *BMC Bioinformatics* 11 (11~may): 237. doi:10.1186/1471-2105-11-237.