Analysis of PRO-seq data

Michael J. Guertin

January 16, 2019

Contents

1 Processing PRO-seq reads **2**
1.1 Processing FASTQ files . . . 2
1.2 Separate by strand, shift, and convert to `bigWig` . . . 3
1.3 Calling TSS coordinates . . . 3
1.4 Counting in gene coordinates . . . 5
1.5 Replicate correlations . . . 7
1.6 Python script: normalize_bedGraph.py . . . 8

2 Differential expression upon heat shock **11**

3 ChIP-seq and proximity analysis **20**
3.1 Calling ChIP-seq peaks . . . 20
3.2 Using existing ChIP-seq peaks . . . 21
3.3 Motif enrichment at HSF peaks . . . 22
3.4 Python script: HOMER_MEME_conversion.py . . . 23

4 HSF and CTCF proximity analysis **23**
4.1 Most of information to ask questions is in this data frame . . . 28

5 UCSC tracks to browse **30**

6 Categorical designation and plotting exercise **32**
6.1 CTCF/HSF/gene orientation subsetting . . . 34

7 HSF peak-centric analyses **37**

8 Next **42**

List of Figures

1 PRO-seq PCA . . . 6
2 Replicate correlations . . . 7
3 Heat shock responsive genes . . . 11
4 *de novo* motif analysis . . . 22
5 motif database matching . . . 24
6 Proximity of heat shock responsive genes to HSF1 binding sites. . . . 28
7 Empirical HSF distance determination . . . 29
8 Proximity of genes to HSF1 binding separated by CTCF proximity . . . 30
9 Proximity of genes to HSF1 binding separated by CTCF proximity only genes within 30kb of HSF binding site . . . 31
10 Exercise Example 1 . . . 33
11 Exercise Example 2 . . . 33
12 Exercise Example 3 . . . 35

13 Exercise Example 4 36
14 ChIP-seq composites at HSF binding sites 40
15 ChIP-seq composites at HSF binidng sites categorized 40
16 ChIP-seq intensities for HSF1 during NHS and HS, and NHS CTCF 41
17 Fraction of HSF peaks within the specified distance from a CTCF peak summit 41

1 Processing PRO-seq reads

1.1 Processing FASTQ files

All the files should be named with the cell type, conditions, description, the replicate number. No spaces. This makes it so that downstream analyses can be automated. The PRO-seq data is GSE89230 (Vihervaara *et al.*, 2017).

`fastq-dump` is an SRA tool (Kodama *et al.*, 2011).
Until I manage to generate a Docker container, I explicitly indicate software dependencies preceding the relevant chunks. Any CS PhD student will be able to help with software installs, which can be very annoying for a beginner. My advice: read the `README` files. **Software installations**:

`sra-tools`: `https://github.com/ncbi/sra-tools`

```
#make a new directory to operate within

cd ~
mkdir HSF
cd HSF

#PRO-seq GSE89230
#GSM2361445: PRO-seq K562 HS30, replicate 2; Homo sapiens
fastq-dump SRR4454570
#GSM2361444: PRO-seq K562 HS30, replicate 1; Homo sapiens
fastq-dump SRR4454569
#GSM2361443: PRO-seq K562 NHS, replicate 2; Homo sapiens
fastq-dump SRR4454568
#GSM2361442: PRO-seq K562 NHS, replicate 1; Homo sapiens
fastq-dump SRR4454567

gzip *fastq

mv SRR4454570.fastq.gz K562_30HS_rep2.fastq.gz #STOPPEDHERE
mv SRR4454569.fastq.gz K562_30HS_rep1.fastq.gz
mv SRR4454568.fastq.gz K562_NHS_rep2.fastq.gz
mv SRR4454567.fastq.gz K562_NHS_rep1.fastq.gz

#you also need to download the human genome:
#one can use the following command on a Linux:
wget http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz

#use curl (or install wget) on a Mac:
curl -OL http://hgdownload.cse.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz

#unzip it
gunzip hg38.fa.gz
```

We first use `cutadapt` to discard reads fewer than 18 bases (Martin, 2011). Note that these do not have an 8 base unique molecular identifier (UMI) to filter out PCR duplicates. Recall that the 5 prime adapter sequence for PRO-seq is: CCUUGGCACCCGAGAAUUCCA (Mahat *et al.*, 2016) `https://www.nature.com/articles/nprot.2016.086/tables/1`

We then implement `fastx_tools` to make the longest read length 30 bases `https://github.com/agordon/fastx_toolkit`. Trimming to 30 bases is not necessary.

We then align to the `hg38` genome using `bowtie2` (Langmead *et al.*, 2009); the first task is to build the genome index with `bowtie2`. This only has to be performed once per genome. Then we convert the `sam` file to the compressed and sorted `BAM` format using `samtools` (Li *et al.*, 2009). We are only aligning single end reads for this experiment

Software installations:

`cutadapt`: `https://cutadapt.readthedocs.io/en/stable/`

`fastx_toolkit`: `https://github.com/agordon/fastx_toolkit`

`bowtie2`: `http://bowtie-bio.sourceforge.net/bowtie2/index.shtml`

`samtools`: `http://www.htslib.org`

```
#build index genome
bowtie2-build hg38.fa hg38

for i in *.fastq.gz
do
    name=$(echo $i | awk -F"/" '{print $NF}' | \
           awk -F".fastq.gz" '{print $1}')
    echo $name
    cutadapt -m 18 -a TGGAATTCTCGGGTGCCAAGG ${name}.fastq.gz | \
       fastx_trimmer -f 1 -l 30 | \
       fastx_reverse_complement -z -o ${name}.processed.fastq.gz
    bowtie2 -p 3 -x hg38 -U ${name}.processed.fastq.gz | \
       samtools view -b - | \
       samtools sort - -o ${name}.sorted.bam
done
```

1.2 Separate by strand, shift, and convert to `bigWig`

First we separate the `BAM` by strand. Here we are not implementing enzyme bias scaling of the PRO-seq reads, but `seqOutBias` (Martins *et al.*, 2018) has some useful features (e.g. –shift-counts) and mappability filtering, so we will use `seqOutBias` to make `bigWig` files. The time consuming intermediate files are only made once, so only the first run is slow. A *wonderfully written* and *exceptionally complete* description of the methods, algorithms, software, and biological implications exist as a PDF vignette (`https://guertinlab.github.io/seqOutBias/seqOutBias_vignette.pdf`), a website vignette (`https://guertinlab.github.io/seqOutBias_Vignette/`), a user guide (`https://guertinlab.github.io/seqOutBias/seqOutBias_user_guide.pdf`), and a paper (Martins *et al.*, 2018).

Software installations:

`seqOutBias`: `https://github.com/guertinlab/seqOutBias`

```
for i in *.sorted.bam
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F".sorted.bam" '{print $1}')
    echo $name
    samtools view -bh -F 20 ${name}.sorted.bam > ${name}_pro_plus.bam
    samtools view -bh -f 0x10 ${name}.sorted.bam > ${name}_pro_minus.bam
    seqOutBias hg38.fa \
        ${name}_pro_plus.bam --no-scale --skip-bed \
        --bw=${name}_plus_body_0-mer.bigWig --tail-edge --read-size=30
    seqOutBias hg38.fa \
        ${name}_pro_minus.bam --no-scale --skip-bed \
        --bw=${name}_minus_body_0-mer.bigWig --tail-edge --read-size=30
done
```

1.3 Calling TSS coordinates

Assigning predominant transcription start sites is an important step when analyzing PRO-seq data. We will start with all Ensembl annotations and choose the one with highest promoter proximal pausing density using all K562 merged data. The following code retrieves the gene annotation (`GTF`) file, extracts all annotated transcription start sites (TSS), combines all data into a single `bigWig` per strand, and extracts all genic annotations.

```
#retrieve gene annotation file
wget ftp://ftp.ensembl.org/pub/release-95/gtf/homo_sapiens/Homo_sapiens.GRCh38.95.gtf.gz
gunzip Homo_sapiens.GRCh38.95.gtf.gz
```

```
#parse all TSS--exons 1
grep 'exon_number "1"' Homo_sapiens.GRCh38.95.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$14,$20,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' > Homo_sapiens.GRCh38.95.tss.bed

#combine all PRO data from HEK

pfiles=$(ls *plus*bam)
mfiles=$(ls *minus*bam)

seqOutBias hg38.fa --tallymer hg38_30.4.2.2.tbl \
    $pfiles --no-scale --bw=K562_plus_combined_no_scale.bigWig --tail-edge --read-size=30
seqOutBias hg38.fa --tallymer hg38_30.4.2.2.tbl \
    $mfiles --no-scale --bw=K562_minus_combined_no_scale.bigWig --tail-edge --read-size=30


mkdir combined_bigWig
mv *combined_no_scale.bigWig combined_bigWig


#extract all complete gene annotations
awk '$3 == "gene"' Homo_sapiens.GRCh38.95.gtf | \
    sed 's/^/chr/' | \
    awk '{OFS="\t";} {print $1,$4,$5,$14,$10,$7}' | \
    sed 's/";//g' | \
    sed 's/"//g' > Homo_sapiens.GRCh38.95.bed
```

Operating on combined `bigWig` and gene annotation files in R. The following chunk loads the functions from our `seqOutBias` paper and a set of functions that are specific for this project. Note that I have not polished these functions and some of the project-specific functions may replace the `seqOutBias` functions that are named identically. Note the libraries that need installation in R.

```
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#you need to make this directory
dir = paste0(path.expand("~"),'/HSF/')

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
source('https://raw.githubusercontent.com/guertinlab/seqOutBias/master/docs/R/seqOutBias_functions.R')

setwd(dir)

#get this from guertin@virginia.edu and put in your HSF directory
source('MJG_sathyan_functions.R')

#these need to be installed
library(DESeq2)
library(lattice)
library(bigWig)
library(grid)
library(zoo)
library(latticeExtra)
library(data.table)
```

```
setwd(dir)

#clean up annotations
tss.test = read.table('Homo_sapiens.GRCh38.95.tss.bed')
tss.test.100 = bed.100.interval(tss.test)
tss.test.100[,1] = as.character(tss.test.100[,1])
chr_keep = c( paste0("chr",c(1:22)), "chrX") #taking out chrY here is a good idea for XX cells
tss.test.100 = tss.test.100[tss.test.100[,1] %in% chr_keep,]
tss.test.100[tss.test.100[,1]=="chrMT",1] = "chrM"

#load combined bigWigs
loaded.bw.plus = load.bigWig(paste0(dir,
                                    'combined_bigWig/K562_plus_combined_no_scale.bigWig'))
loaded.bw.minus = load.bigWig(paste0(dir,
                                     'combined_bigWig/K562_minus_combined_no_scale.bigWig'))

#count in pause region
mod.inten = bed6.region.bpQuery.bigWig(loaded.bw.plus, loaded.bw.minus, tss.test.100)
mod.inten.df = data.frame(cbind(tss.test.100, mod.inten))
mod.inten.df = as.data.table(mod.inten.df)
```

```
#select the TSS with highest read count
df.tss = mod.inten.df[mod.inten.df[, .I[which.max(mod.inten)], by=xy]$V1]

save(df.tss, file = paste(dir, 'df.tss.k562.Rdata'))

#load('df.tss.vcap.Rdata')

#combine with transcription termination site. One per gene
bed6 = read.table("Homo_sapiens.GRCh38.95.bed")
gene.file = bed6[bed6[,1] %in% chr_keep,]
gene.file[,1] = as.character(gene.file[,1])
gene.file[,6] = as.character(gene.file[,6])
gene.file = gene.file[!duplicated(paste(gene.file[,1],gene.file[,2],gene.file[,3])),]

#ad hoc filtering of small genes and problem annotations
gene.file = gene.file[(gene.file[,3] - gene.file[,2] )> 200,]
gene.file = gene.file[gene.file[,4] != 'Metazoa_SRP' &
                      gene.file[,4] != 'U3' & gene.file[,4] != '7SK',]
gene.file = gene.file[!duplicated(gene.file[,4]),]

#merge
df.tss.pre = merge(gene.file, df.tss, by.x='V4', by.y='xy')[,c(2,3,4,10,1,6,8,9)]

colnames(df.tss.pre) = c('chr', 'start', 'end', 'gene', 'xy', 'strand', 'tssplus', 'tssminus')


df.tss.pre[df.tss.pre$strand == '+',]$start = df.tss.pre[df.tss.pre$strand == '+',]$tssplus - 20
df.tss.pre[df.tss.pre$strand == '-',]$end = df.tss.pre[df.tss.pre$strand == '-',]$tssminus + 20

df.tss.pre = df.tss.pre[,c(1:3,5,4,6)]
colnames(df.tss.pre) = c('V1', 'V2','V3','V4','V5','V6')

gene.file = df.tss.pre
dim(gene.file)
gene.file = gene.file[(gene.file[,3] - gene.file[,2] )> 200,]
gene.file = gene.file[(gene.file[,3] - gene.file[,2] ) < 2500000,]

dim(gene.file)

#I may be missing some weird annotation error that puts
#the 5' end of a gene incredibly far aawy from the true tss
```

1.4 Counting in gene coordinates

Section 1.3 defines each gene annotation for downstream analyses. The following section will analyze the reads that align to these annotations. The principle components analysis should confirm that replicates cluster and treatments separate.

```
df.K562 = get.raw.counts.interval(gene.file, dir, file.prefix = 'K562_')

#use this for making the bedGraphs and loading into UCSC
estimateSizeFactorsForMatrix(df.K562)
write.table(estimateSizeFactorsForMatrix(df.K562),
            file = 'norm.bedGraph.sizeFactor.txt', quote =F, col.names=F)

#PCA for experiments

df.K562.deseq.raw = run.deseq.table(df.K562)
rld_K562 = rlogTransformation(df.K562.deseq.raw)
plotPCA(rld_K562, intgroup="condition")
x = plotPCA(rld_K562, intgroup="condition", returnData=TRUE)
percentVar = round(100 * attr(x, "percentVar"))

#this function was updated in the ATAC analysis to allow for different # reps
plotPCAlattice(x, file = 'PCA_K562_lattice_guertin.pdf', reps = 2)


save(df.K562, file = paste0(dir, 'df.K562.Rdata'))
```

Conclusion: Despite variation in read depth, the replicates are consistent as represented by PCA.

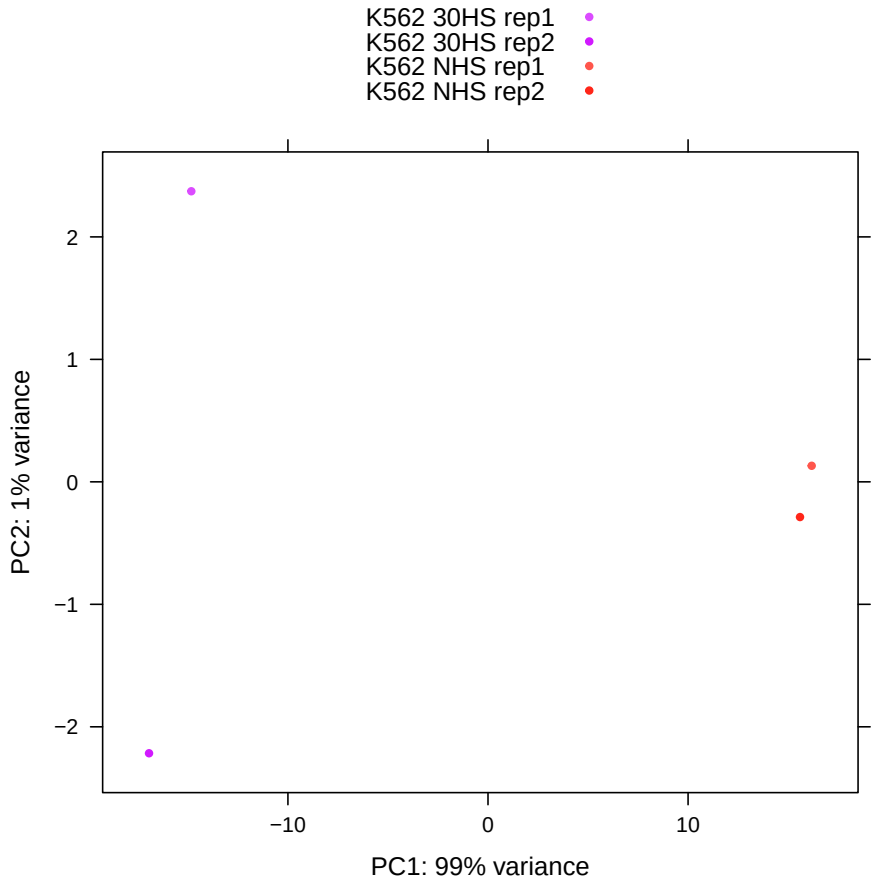


Figure 1: Principle components analysis indicates that heat shock dominates PC1 and replicates cluster together.

1.5 Replicate correlations

```
#replicate concordance:
rep1.corr = df.K562[,seq(1, ncol(df.K562), by = 2)]
rep2.corr = df.K562[,seq(2, ncol(df.K562), by = 2)]

colnames(rep1.corr) = gsub('_', ' ',
                           sapply(strsplit(colnames(rep1.corr), '_rep'), '[', 1))
colnames(rep2.corr) = gsub('_', ' ',
                           sapply(strsplit(colnames(rep2.corr), '_rep'), '[', 1))


lattice.pairwise.scatter(rep1.corr, rep2.corr,
          file = 'Fig_pairwise_scatter_K562_reps1v2_lattice.pdf', r.1 = '1', r.2 ='2')

#one can repeat the analyses for all pairwise replicate combinations

pairwise.scatters.each.time.point.reps.MJG(rep1.corr, rep2.corr,
          filename= "Fig_pairwise_scatter_K562_reps.pdf")

#
```

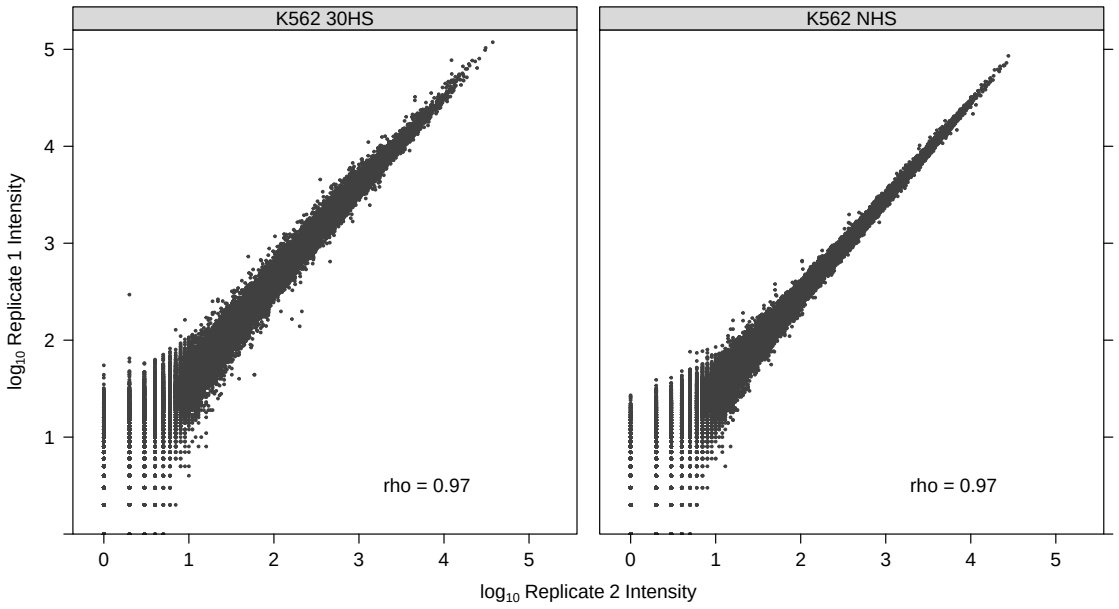


Figure 2: All PRO-seq replicates correlate with Spearman coefficients above 0.95.

Combine replicates to load into a browser and use bigWigs for plotting composite profiles. Here I include an executable `Python` program to scale `bedGraph` files. I have had trouble with `genomeCoverageBed` (Quinlan and Hall, 2010) in the past, so I wrote this to bypass the issue. I can send you the file directly if the code chunk doesn't copy/paste properly.

1.6 Python script: normalize_bedGraph.py

```
#! /usr/bin/python
import sys
import getopt
import os
import itertools
import random
import os
import glob
from decimal import *
def function(infile, scale, outfilename):
    outfile = open(outfilename, 'w')
    for line in file(infile):
            #print line
        splitline=line.split()
        outfile.write("%s\t%s\t%s\t%s\n"%(splitline[0], splitline[1], splitline[2], Decimal(splitline[3])*Decimal(scale)))
    outfile.close()

def main(argv):
    try:
        opts, args = getopt.getopt(argv, "i:s:o:h", ["infile=", "scale=", "outfilename=","help"])
    except getopt.GetoptError, err:
        print str(err)
        sys.exit(2)
    infile = False
    scale = False
    outfilename = False
    for opt, arg in opts:
        if opt in ('-i', '--infile'):
            infile = arg
        elif opt in ('-s', '--scale'):
            scale = arg
        elif opt in ('-o', '--outfilename'):
            outfilename = arg
        elif opt in ('-h', '--help'):
            print '\n./normalize_bedGraph.py -i HEK293T_minus.bg -s 1.2 -o HEK293T_minus.scaled.bg'
            sys.exit()
    if infile and scale and outfilename:
        print infile
        function(infile, scale, outfilename)
if __name__ == "__main__":
    main(sys.argv[1:])
```

Combine replicates for viewing on UCSC and for making composite profiles. This uses UCSC tools (Kuhn *et al.,* 2012). I download the executable binaries directly by following the appropriate links to my architecture and software (see links below).

Software installations:

`bigWigToBedGraph`: `http://hgdownload.soe.ucsc.edu/admin/exe/`

`bedGraphToBigWig`: `http://hgdownload.soe.ucsc.edu/admin/exe/`

`bigWigMerge`: `http://hgdownload.soe.ucsc.edu/admin/exe/`

```
#retrieve the chromosome size file
wget https://hgdownload-test.gi.ucsc.edu/goldenPath/hg38/bigZips/hg38.chrom.sizes

for i in K*_rep*_pro_*.bam
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F"_pro_" '{print $1}')
    strand=$(echo $i | awk -F"pro_" '{print $NF}' | awk -F".bam" '{print $1}')
    invscale1=$(grep ${name} norm.bedGraph.sizeFactor.txt | awk -F" " '{print $2}')
    invscale=$(echo $invscale1 |bc)
    scaletrue=$(bc <<< "scale=4 ; 1.0 / $invscale")
    echo $name
    echo $strand
    echo $scaletrue
    echo file_to_scale
    echo ${name}_${strand}_body_0-mer.bigWig
    bigWigToBedGraph ${name}_${strand}_body_0-mer.bigWig ${name}_${strand}_body_0-mer.bg
    echo normalizing
    python normalize_bedGraph.py -i ${name}_${strand}_body_0-mer.bg \
       -s $scaletrue -o ${name}_${strand}_scaled.bg
    bedGraphToBigWig ${name}_${strand}_scaled.bg hg38.chrom.sizes ${name}_pro_${strand}_scaled.bigWig
    rm ${name}_${strand}_scaled.bg
    rm ${name}_${strand}_body_0-mer.bg
done
```

Next merge bigWigs of the same condition

```
for i in *rep1*_scaled.bigWig
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F"_rep1" '{print $1}')
    strand=$(echo $i | awk -F"pro_" '{print $NF}' | awk -F"_scaled.bigWig" '{print $1}')
    echo $name
    echo $strand
    files=$(ls ${name}_rep*${strand}_scaled.bigWig)
    echo $files
    touch temp.txt
    if [ "$strand" == "plus" ]
    then
        echo "track type=bedGraph name=${name}_plus color=255,0,0 alwaysZero=on visibility=full" >> temp.txt
    fi
    if [ "$strand" == "minus" ]
    then
        echo "track type=bedGraph name=${name}_minus color=0,0,255 alwaysZero=on visibility=full" >> temp.txt
    fi
    count=$(ls ${name}_rep*${strand}_scaled.bigWig | wc -l | bc)
    scaleall=$(bc <<< "scale=4 ; 1.0 / $count")
    echo $count
    echo $scaleall
    bigWigMerge $files tmp.bg
    python normalize_bedGraph.py -i tmp.bg \
       -s $scaleall -o ${name}_${strand}_scaled.bg
    LC_COLLATE=C sort -k1,1 -k2,2n ${name}_${strand}_scaled.bg > ${name}_${strand}_scaled_sorted.bg
    cat temp.txt ${name}_${strand}_scaled_sorted.bg > ${name}_${strand}_scaled.bedGraph
    rm tmp.bg
    rm temp.txt
    rm ${name}_${strand}_scaled.bg
    rm ${name}_${strand}_scaled_sorted.bg
    head ${name}_${strand}_scaled.bedGraph
    bedGraphToBigWig ${name}_${strand}_scaled.bedGraph hg38.chrom.sizes ${name}_${strand}_combined_normalized.bigWig
    gzip ${name}_${strand}_scaled.bedGraph
done
```

2 Differential expression upon heat shock

We observe many activated and repressed genes.

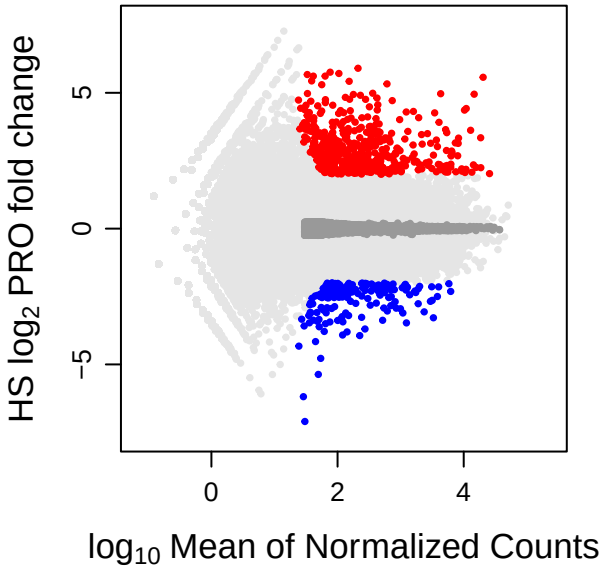


Figure 3: Genes are activated and repressed by heat stress.

```
df.hs.deseq = run.deseq.list(df.K562[,c(
'K562_NHS_rep1',
'K562_NHS_rep2',
'K562_30HS_rep1',
'K562_30HS_rep2')])

save(df.hs.deseq, file = paste0(dir, 'df.hs.deseq'))
```

```
dir = paste0(path.expand("~"),'/HSF/')
setwd(dir)

load(file = paste0(dir, 'df.K562.Rdata'))
load(file = paste0(dir, 'df.hs.deseq'))


categorize.deseq.df.hsf <- function(df, fdr = 0.05, log2fold = 0.0, treat = 'Auxin') {
    df.activated = df[df$padj < fdr & !is.na(df$padj) & df$log2FoldChange > log2fold,]

    df.repressed = df[df$padj < fdr & !is.na(df$padj) & df$log2FoldChange < -log2fold,]
#Here is where I define the unchanged class of genes
    df.unchanged = df[df$padj > 0.75 & df$baseMean > 30 & !is.na(df$padj) & abs(df$log2FoldChange) < 0.25,]

    df.dregs = df[!(df$padj < fdr & !is.na(df$padj) & df$log2FoldChange > log2fold) &
                  !(df$padj < fdr & !is.na(df$padj) & df$log2FoldChange < -log2fold) &
                  !(df$padj > 0.75 & df$baseMean > 30 &!is.na(df$padj) & abs(df$log2FoldChange) < 0.1), ]
    df.unchanged$arfauxin = paste(treat, 'Unchanged')
    df.activated$arfauxin = paste(treat, 'Activated')
    df.repressed$arfauxin = paste(treat, 'Repressed')
    df.dregs$arfauxin = paste(treat, 'All Other Genes')
    df.effects.lattice =
    rbind(df.activated,
          df.unchanged,
          df.repressed,
          df.dregs)
    df.effects.lattice$arfauxin = factor(df.effects.lattice$arfauxin)
    df.effects.lattice$arfauxin = relevel(df.effects.lattice$arfauxin,
                                          ref = paste(treat, 'Unchanged'))
    df.effects.lattice$arfauxin = relevel(df.effects.lattice$arfauxin,
                                          ref = paste(treat, 'All Other Genes'))
    return(df.effects.lattice)
}

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#change these parameters to get out new gene lists

fdr.variable = 0.00001
log2diff = 2

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
df.hs.deseq.effects.lattice =
    categorize.deseq.df.hsf(df.hs.deseq,
                        fdr = fdr.variable, log2fold = log2diff, treat = '30min HS')

hs.activated.bed = convert.deseq.to.bed(df.hs.deseq.effects.lattice
                                        [df.hs.deseq.effects.lattice$arfauxin ==
                                         '30min HS Activated',], gene.file)
hs.repressed.bed = convert.deseq.to.bed(df.hs.deseq.effects.lattice
                                        [df.hs.deseq.effects.lattice$arfauxin ==
                                         '30min HS Repressed',], gene.file)
write.table(hs.activated.bed, file = 'hs.activated.bed', quote=FALSE,
            col.names=FALSE, row.names=FALSE, sep ='\t')
write.table(hs.repressed.bed, file = 'hs.repressed.bed', quote=FALSE,
            col.names=FALSE, row.names=FALSE, sep ='\t')

hs.activated = length(df.hs.deseq.effects.lattice$arfauxin
                                [df.hs.deseq.effects.lattice$arfauxin =='30min HS Activated'])

head(df.hs.deseq.effects.lattice
                                [df.hs.deseq.effects.lattice$arfauxin =='30min HS Activated',
                                 c(2, 6,7)], 50)

## DataFrame with 50 rows and 3 columns
##                                       log2FoldChange
##                                            <numeric>
## chr18:21650897-21704780_ABHD3               2.324613
## chr19:637105-637537_AC004449.1              3.767571
## chr22:19124309-19128449_AC004471.2          3.218911
## chr19:999796-1002757_AC004528.1             4.022628
## chr1:35569813-35577729_AC004865.2           2.627984
## ...                                              ...
## chr12:123960717-123961244_AC068790.7        2.316490
## chr17:26989189-26999941_AC069061.2          2.749749
## chr3:146504570-146506560_AC069528.2         3.721594
```

```
## chr10:71878356-71879107_AC073370.1         3.362337
## chr2:47690716-47691246_AC079250.1          2.131193
##                                                                                 padj
##                                                                            <numeric>
## chr18:21650897-21704780_ABHD3        0.00000000000000000000000000000000000000000000002106581
## chr19:637105-637537_AC004449.1       0.00000000178229626065801537838597561405618030505931967217
## chr22:19124309-19128449_AC004471.2   0.00000000000000000000000000000000000000000000324326381845
## chr19:999796-1002757_AC004528.1      0.00000000000000000000000001699259279322817439372727369227
## chr1:35569813-35577729_AC004865.2    0.00000000000000000013367484837790896639983084794704416285 7
## ...                                                                              ...
## chr12:123960717-123961244_AC068790.7                  0.000000034315060463755295730986082537 03
## chr17:26989189-26999941_AC069061.2                    0.00000000000000000000000000000000222828
## chr3:146504570-146506560_AC069528.2                   0.00000000177997106082392066595850993054
## chr10:71878356-71879107_AC073370.1                    0.00000001689255698126270032448488446399
## chr2:47690716-47691246_AC079250.1                     0.00000000943163578871944125386417648316
##                                          arfauxin
##                                          <factor>
## chr18:21650897-21704780_ABHD3        30min HS Activated
## chr19:637105-637537_AC004449.1       30min HS Activated
## chr22:19124309-19128449_AC004471.2   30min HS Activated
## chr19:999796-1002757_AC004528.1      30min HS Activated
## chr1:35569813-35577729_AC004865.2    30min HS Activated
## ...                                             ...
## chr12:123960717-123961244_AC068790.7 30min HS Activated
## chr17:26989189-26999941_AC069061.2   30min HS Activated
## chr3:146504570-146506560_AC069528.2  30min HS Activated
## chr10:71878356-71879107_AC073370.1   30min HS Activated
## chr2:47690716-47691246_AC079250.1    30min HS Activated

hs.repressed = length(df.hs.deseq.effects.lattice$arfauxin
                            [df.hs.deseq.effects.lattice$arfauxin =='30min HS Repressed'])

head(df.hs.deseq.effects.lattice
                            [df.hs.deseq.effects.lattice$arfauxin =='30min HS Repressed',
                             c(2, 6,7)], 50)

## DataFrame with 50 rows and 3 columns
##                                    log2FoldChange
##                                         <numeric>
## chr9:137007227-137028288_ABCA2            -2.208441
## chr7:90345873-90346218_AC002064.3         -3.785589
## chr22:20981361-20981755_AC002470.1        -3.917969
## chr17:40517026-40527002_AC004585.1        -2.769528
## chr17:10291820-10317926_AC005291.2        -2.352419
## ...                                             ...
## chr14:91418266-91421176_AL133153.2        -2.321326
## chr6:43213801-43223860_AL133375.1         -2.155578
## chr10:125718771-125719365_AL158835.2      -2.739518
## chr1:228270443-228274397_AL353593.1       -2.404148
## chr9:112748902-112750456_AL390067.1       -2.404260
##                                                                                 padj
##                                                                            <numeric>
## chr9:137007227-137028288_ABCA2                                0.000000000000000000000661059031
## chr7:90345873-90346218_AC002064.3                             0.000000000001898765410901776362
## chr22:20981361-20981755_AC002470.1                            0.000000000000000000000006483013
## chr17:40517026-40527002_AC004585.1                            0.000016590683079432790169465729
## chr17:10291820-10317926_AC005291.2                            0.000000015912553197450438901399
## ...                                                                              ...
## chr14:91418266-91421176_AL133153.2   0.000000139088666598605080861058929352080948405046001 43
## chr6:43213801-43223860_AL133375.1    0.00000000000000000000000000000000000000000000008014426
## chr10:125718771-125719365_AL158835.2 0.000004447844716723128971852356527971750210781465284 53
## chr1:228270443-228274397_AL353593.1  0.000000000054906736852948376397971660944405646867 1697
## chr9:112748902-112750456_AL390067.1  0.000000002497599949567886352049945930162665186369252 98
##                                          arfauxin
##                                          <factor>
## chr9:137007227-137028288_ABCA2       30min HS Repressed
## chr7:90345873-90346218_AC002064.3    30min HS Repressed
## chr22:20981361-20981755_AC002470.1   30min HS Repressed
## chr17:40517026-40527002_AC004585.1   30min HS Repressed
## chr17:10291820-10317926_AC005291.2   30min HS Repressed
## ...                                             ...
## chr14:91418266-91421176_AL133153.2   30min HS Repressed
## chr6:43213801-43223860_AL133375.1    30min HS Repressed
## chr10:125718771-125719365_AL158835.2 30min HS Repressed
## chr1:228270443-228274397_AL353593.1  30min HS Repressed
## chr9:112748902-112750456_AL390067.1  30min HS Repressed

hs.unchanged = length(df.hs.deseq.effects.lattice$arfauxin
                            [df.hs.deseq.effects.lattice$arfauxin =='30min HS Unchanged'])

hs.dregs = length(df.hs.deseq.effects.lattice$arfauxin
                [df.hs.deseq.effects.lattice$arfauxin =='30min HS 30min HS All Other Genes'])

pdf("MA_plot_HS_response_in_classes_panels.pdf", useDingbats = FALSE, width=4.83, height=3.33);
print(xyplot(df.hs.deseq.effects.lattice$log2FoldChange ~
               log(df.hs.deseq.effects.lattice$baseMean, base=10),
```

```
             groups=df.hs.deseq.effects.lattice$arfauxin,
             col=c("grey90", "grey60", "red", "blue"),
             scales="free",
             aspect=1,
             ylim=c(-7.2, 7.2),
             xlim=c(-1,5.2),
             par.strip.text=list(cex=1.0, font = 1),
             pch=20,
             cex=0.5,
             ylab=expression("HS log"[2]~"PRO fold change"),
             xlab=expression("log"[10]~"Mean of Normalized Counts"),
            par.settings=list(par.xlab.text=list(cex=1.1,font=2),
                              par.ylab.text=list(cex=1.1,font=2),
                              strip.background=list(col="grey85"))))
dev.off()

## pdf
##   2

save(df.hs.deseq.effects.lattice, file = paste0(dir, 'df.hs.deseq.effects.lattice.Rdata'))
```

It is sometimes useful to have a gene list in the PDF, so I am printing them here. Running with different parameters (FDR and fold difference) also dynamically generates the bed files that conform to the new thresholds.

```
echo HSF activated genes
cat hs.activated.bed
echo
echo HSF repressed genes
cat hs.repressed.bed

## HSF activated genes
## chr2 158548425 158548426 AC005042.2 ENST00000393397 +
## chr2 210171518 210171519 AC006994.2 ENST00000412065 +
## chr2 165794851 165794852 AC009495.2 ENST00000425688 +
## chr17 83104255 83104256 AC144831.1 ENST00000573312 +
## chr22 24423597 24423598 ADORA2A ENST00000424232 +
## chr17 6996982 6996983 ALOX12 ENST00000480801 +
## chr15 84816847 84816848 ALPK3 ENST00000258888 +
## chr5 75611476 75611477 ANKDD1B ENST00000506596 +
## chr17 42851184 42851185 AOC3 ENST00000308423 +
## chr17 42865922 42865923 AOC4P ENST00000569586 +
## chr16 268727 268728 ARHGDIG ENST00000412541 +
## chr17 44170706 44170707 ASB16 ENST00000293414 +
## chr10 119651676 119651677 BAG3 ENST00000369085 +
## chr19 12751802 12751803 BEST2 ENST00000553030 +
## chr6 26421391 26421392 BTN2A3P ENST00000465856 +
## chr11 95143637 95143638 BUD13P1 ENST00000603676 +
## chr19 47309874 47309875 C5AR1 ENST00000355085 +
## chr5 170232946 170232947 C5orf58 ENST00000593851 +
## chr9 136946434 136946435 C8G ENST00000465773 +
## chr9 133459998 133459999 CACFD1 ENST00000489519 +
## chr1 174999955 174999956 CACYBP ENST00000473925 +
## chr11 86374887 86374888 CCDC81 ENST00000531271 +
## chr2 56184625 56184626 CCDC85A ENST00000407595 +
## chr19 41877305 41877306 CD79A ENST00000221972 +
## chr6 14117641 14117642 CD83 ENST00000379153 +
## chr20 59989106 59989107 CDH26 ENST00000370991 +
## chr9 133061978 133061979 CEL ENST00000372080 +
## chr15 78565520 78565521 CHRNA5 ENST00000299565 +
## chrX 46573932 46573933 CHST7 ENST00000276055 +
## chr19 19538248 19538249 CILP2 ENST00000586018 +
## chrX 10156945 10156946 CLCN4 ENST00000421085 +
## chr3 45026183 45026184 CLEC3B ENST00000296130 +
## chr20 63295108 63295109 COL20A1 ENST00000422202 +
## chr4 41981793 41981794 DCAF4L1 ENST00000333141 +
## chr10 72274019 72274020 DDIT4 ENST00000471240 +
## chr19 48933682 48933683 DHDH ENST00000221403 +
## chr9 33025286 33025287 DNAJA1 ENST00000495015 +
## chr1 78004935 78004936 DNAJB4 ENST00000476396 +
## chr7 157337010 157337011 DNAJB6 ENST00000417758 +
## chr7 64862999 64863000 EEF1DP4 ENST00000446006 +
## chr19 1286169 1286170 EFNA2 ENST00000215368 +
## chr1 225810095 225810096 EPHX1 ENST00000272167 +
## chr11 46719236 46719237 F2 ENST00000311907 +
```

```
## chr1 46394349 46394350 FAAH ENST00000243167 +
## chr5 80487969 80487970 FAM151B ENST00000502608 +
## chr1 32247422 32247423 FAM167B ENST00000373582 +
## chr12 109734767 109734768 FAM222A ENST00000358906 +
## chr7 128672398 128672399 FAM71F2 ENST00000477515 +
## chr5 168529116 168529117 FBLL1 ENST00000338333 +
## chr8 22042398 22042399 FGF17 ENST00000518533 +
## chr5 171419656 171419657 FGF18 ENST00000274625 +
## chr12 2795069 2795070 FKBP4 ENST00000543769 +
## chr8 101052054 101052055 FLJ42969 ENST00000514926 +
## chr2 241776825 241776826 GAL3ST2 ENST00000192314 +
## chr22 37823382 37823383 GALR3 ENST00000249041 +
## chr4 56593004 56593005 GLDCP1 ENST00000510885 +
## chr16 56191390 56191391 GNAO1 ENST00000262493 +
## chr2 240605445 240605446 GPR35 ENST00000403859 +
## chrX 102712445 102712446 GPRASP2 ENST00000486814 +
## chr16 163122 163123 HBZP1 ENST00000354915 +
## chr2 47527537 47527538 HCG2040054 ENST00000435331 +
## chr19 589893 589894 HCN2 ENST00000251287 +
## chr18 24460629 24460630 HRH4 ENST00000256906 +
## chr1 162790723 162790724 HSD17B7 ENST00000367915 +
## chr10 38356380 38356381 HSD17B7P2 ENST00000494540 +
## chr3 184115352 184115353 HSP90AA5P ENST00000438353 +
## chr4 13336482 13336483 HSP90AB2P ENST00000507090 +
## chr4 87891843 87891844 HSP90AB3P ENST00000505987 +
## chr14 64540850 64540851 HSPA2 ENST00000247207 +
## chr4 127782322 127782323 HSPA4L ENST00000508549 +
## chr1 161606291 161606292 HSPA7 ENST00000445535 +
## chr8 30237382 30237383 HSPA8P11 ENST00000508840 +
## chr7 85027828 85027829 HSPA8P16 ENST00000454171 +
## chr12 4097451 4097452 HSPA8P5 ENST00000456928 +
## chr7 10451311 10451312 HSPA8P8 ENST00000419827 +
## chr3 137880295 137880296 HSPA8P9 ENST00000485881 +
## chr7 76302688 76302689 HSPB1 ENST00000447574 +
## chr11 111912736 111912737 HSPB2-C11orf52 ENST00000527616 +
## chr17 42121431 42121432 HSPB9 ENST00000565659 +
## chr5 21882585 21882586 HSPD1P1 ENST00000503199 +
## chr12 8015024 8015025 HSPD1P12 ENST00000429127 +
## chr12 56511002 56511003 HSPD1P4 ENST00000547203 +
## chr4 144845625 144845626 HSPD1P5 ENST00000511127 +
## chr2 197500379 197500380 HSPE1 ENST00000409468 +
## chr2 197500413 197500414 HSPE1-MOB4 ENST00000604458 +
## chr10 1022668 1022669 IDI2-AS1 ENST00000420381 +
## chr22 22327351 22327352 IGLV1-50 ENST00000390291 +
## chr15 40406074 40406075 IVD ENST00000558610 +
## chr11 64291302 64291303 KCNK4 ENST00000538846 +
## chr1 180913154 180913155 KIAA1614 ENST00000496210 +
## chr16 2964275 2964276 KREMEN2 ENST00000575885 +
## chr9 136949551 136949552 LCN12 ENST00000484304 +
## chr13 113276616 113276617 LDHBP1 ENST00000455730 +
## chr5 132873444 132873445 LEAP2 ENST00000483190 +
## chr16 89159220 89159221 LINC00304 ENST00000321214 +
## chr21 34157724 34157725 LINC00310 ENST00000630751 +
## chr14 96943579 96943580 LINC00618 ENST00000435624 +
## chr13 79566727 79566728 LINC01068 ENST00000620874 +
## chr1 87129854 87129855 LINC01140 ENST00000471417 +
## chr1 246789094 246789095 LINC01341 ENST00000426089 +
## chr15 74812716 74812717 LMAN1L ENST00000470711 +
## chr4 109848202 109848203 LRIT3 ENST00000594814 +
## chrX 30230436 30230437 MAGEB3 ENST00000361644 +
## chr19 40771648 40771649 MIA ENST00000597600 +
## chr19 40771648 40771649 MIA-RAB4B ENST00000600729 +
## chr6 159790466 159790467 MRPL18 ENST00000367034 +
## chr21 34073549 34073550 MRPS6 ENST00000477091 +
## chr19 1354977 1354978 MUM1 ENST00000591337 +
## chr19 17420047 17420048 MVB12A ENST00000528659 +
## chr21 39488327 39488328 MYL6P2 ENST00000450079 +
## chr7 141811805 141811806 MYL6P4 ENST00000477911 +
## chr12 69825518 69825519 MYRFL ENST00000552032 +
## chr1 160367067 160367068 NHLH1 ENST00000302101 +
## chr19 48900355 48900356 NUCB1 ENST00000443560 +
## chr1 26921726 26921727 NUDC ENST00000321265 +
## chr10 72893584 72893585 OIT3 ENST00000622652 +
```

```
## chr19 36111151 36111152 OVOL3 ENST00000633214 +
## chr11 289172 289173 PGGHG ENST00000409655 +
## chr1 11934694 11934695 PLOD1 ENST00000449038 +
## chr6 159800294 159800295 PNLDC1 ENST00000609334 +
## chr6 36243203 36243204 PNPLA1 ENST00000312917 +
## chr21 14757588 14757589 POLR2CP1 ENST00000453699 +
## chr3 9947404 9947405 PRRT3-AS1 ENST00000431558 +
## chr9 122375322 122375323 PTGS1 ENST00000373698 +
## chr19 40778216 40778217 RAB4B ENST00000594800 +
## chr17 80976454 80976455 RPL12P37 ENST00000486874 +
## chr12 104265309 104265310 RPL18AP3 ENST00000462885 +
## chr11 62405103 62405104 SCGB1A1 ENST00000534397 +
## chr11 75562253 75562254 SERPINH1 ENST00000529643 +
## chr9 34318407 34318408 SERPINH1P1 ENST00000419794 +
## chr11 86294760 86294761 SETP17 ENST00000534163 +
## chr21 39445875 39445876 SH3BGR ENST00000458295 +
## chr1 201888680 201888681 SHISA4 ENST00000464117 +
## chr3 121914925 121914926 SLC15A2 ENST00000489886 +
## chr12 100357079 100357080 SLC17A8 ENST00000323346 +
## chr6 160129761 160129762 SLC22A1 ENST00000540443 +
## chr4 83477524 83477525 SLC25A14P1 ENST00000502867 +
## chr21 34073570 34073571 SLC5A3 ENST00000381151 +
## chr2 28810908 28810909 SPDYA ENST00000449210 +
## chr20 57329813 57329814 SPO11 ENST00000345868 +
## chr19 38391991 38391992 SPRED3 ENST00000590962 +
## chr11 64186185 64186186 STIP1 ENST00000536973 +
## chr1 1331314 1331315 TAS1R3 ENST00000339381 +
## chr2 161416331 161416332 TBR1 ENST00000463544 +
## chr1 168281091 168281092 TBX19 ENST00000367821 +
## chrX 103356452 103356453 TCEAL9 ENST00000372656 +
## chr12 110614164 110614165 TCTN1 ENST00000397659 +
## chr6 40378337 40378338 TDRG1 ENST00000448433 +
## chr3 100185687 100185688 TMEM30CP ENST00000637814 +
## chr12 71938987 71938988 TPH2 ENST00000333850 +
## chr17 81637198 81637199 TSPAN10 ENST00000574882 +
## chr19 35666813 35666814 UPK1A ENST00000379013 +
## chrX 75274127 75274128 UPRT ENST00000373373 +
## chr2 74421744 74421745 WDR54 ENST00000465134 +
## chr5 77088080 77088081 ZBED3-AS1 ENST00000514288 +
## chr5 178895894 178895895 ZFP2 ENST00000361362 +
## chr18 21704779 21704780 ABHD3 ENST00000579982 -
## chr19 4358447 4358448 AC007292.3 ENST00000593524 -
## chr2 47691245 47691246 AC079250.1 ENST00000456046 -
## chr1 229434093 229434094 ACTA1 ENST00000366684 -
## chr14 70535014 70535015 ADAM20 ENST00000256389 -
## chr1 186396775 186396776 AL596220.1 ENST00000623025 -
## chrX 55030976 55030977 ALAS2 ENST00000396198 -
## chr6 134950121 134950122 ALDH8A1 ENST00000265605 -
## chr1 113904827 113904828 AP4B1 ENST00000369567 -
## chr17 7177630 7177631 ASGR1 ENST00000570576 -
## chr19 17405574 17405575 BST2 ENST00000252593 -
## chr17 56833895 56833896 C17orf67 ENST00000397861 -
## chr19 51390562 51390563 C19orf84 ENST00000570516 -
## chr22 19034563 19034564 CA15P1 ENST00000481698 -
## chr22 18638404 18638405 CA15P2 ENST00000611168 -
## chr2 183608118 183608119 CACYBPP2 ENST00000453625 -
## chr11 105101430 105101431 CARD16 ENST00000525374 -
## chr2 27629011 27629012 CCDC121 ENST00000324364 -
## chr3 42773242 42773243 CCDC13 ENST00000435327 -
## chr13 102758931 102758932 CCDC168 ENST00000322527 -
## chr22 20592217 20592218 CCDC74BP1 ENST00000508880 -
## chr12 123972823 123972824 CCDC92 ENST00000238156 -
## chr19 18896143 18896144 CERS1 ENST00000623927 -
## chr10 73358853 73358854 CFAP70 ENST00000355577 -
## chrX 47629843 47629844 CFP ENST00000396992 -
## chr22 41240880 41240881 CHADL ENST00000216241 -
## chr11 90223058 90223059 CHORDC1 ENST00000457199 -
## chrX 155334541 155334542 CLIC2 ENST00000465553 -
## chr8 27614691 27614692 CLU ENST00000522413 -
## chr19 40226640 40226641 CNTD2 ENST00000430325 -
## chr17 50200190 50200191 COL1A1 ENST00000507689 -
## chr3 48595158 48595159 COL7A1 ENST00000328333 -
## chr17 48038029 48038030 COPZ2 ENST00000580174 -
```

```
## chr11 111910892 111910893 CRYAB ENST00000525823 -
## chr1 74733060 74733061 CRYZ ENST00000370870 -
## chr12 76879022 76879023 CSRP2 ENST00000546966 -
## chr16 30910207 30910208 CTF2P ENST00000412003 -
## chr7 99679997 99679998 CYP3A5 ENST00000339843 -
## chr19 42217700 42217701 DEDD2 ENST00000602075 -
## chrX 107352842 107352843 DNAJA1P3 ENST00000399417 -
## chr19 14518419 14518420 DNAJB1 ENST00000254322 -
## chr2 191882204 191882205 DNAJB1P1 ENST00000429417 -
## chr8 104466919 104466920 DPYS ENST00000351513 -
## chr15 43221050 43221051 EPB42 ENST00000441366 -
## chr20 35292303 35292304 FAM83C ENST00000374408 -
## chr12 1594164 1594165 FBXL14 ENST00000339235 -
## chr4 118194750 118194751 FKBP4P1 ENST00000507284 -
## chr21 46138571 46138572 FTCD ENST00000446405 -
## chr19 18896095 18896096 GDF1 ENST00000247005 -
## chr5 138274670 138274671 GFRA3 ENST00000274721 -
## chr11 94401418 94401419 GPR83 ENST00000243673 -
## chr4 144140692 144140693 GYPA ENST00000324022 -
## chr4 144019344 144019345 GYPB ENST00000508618 -
## chr4 143905562 143905563 GYPE ENST00000437468 -
## chr1 222548102 222548103 HHIPL2 ENST00000343410 -
## chr2 190319889 190319890 HIBCH ENST00000392332 -
## chr5 176899331 176899332 HK3 ENST00000292432 -
## chr5 43313223 43313224 HMGCS1 ENST00000511774 -
## chr15 82849716 82849717 HOMER2 ENST00000558090 -
## chr14 102087043 102087044 HSP90AA1 ENST00000553585 -
## chr11 27891032 27891033 HSP90AA2P ENST00000530115 -
## chr4 170604983 170604984 HSP90AA6P ENST00000325407 -
## chr6 31815064 31815065 HSPA1L ENST00000375654 -
## chr11 123062031 123062032 HSPA8 ENST00000524590 -
## chrX 121205013 121205014 HSPA8P1 ENST00000425843 -
## chr9 72008135 72008136 HSPB1P1 ENST00000423240 -
## chrX 49234500 49234501 HSPB1P2 ENST00000448722 -
## chr2 197499878 197499879 HSPD1 ENST00000476746 -
## chr3 36768798 36768799 HSPD1P6 ENST00000388967 -
## chr13 31161926 31161927 HSPH1 ENST00000320027 -
## chr15 85794917 85794918 KLHL25 ENST00000559131 -
## chr9 136742003 136742004 LCN10 ENST00000494294 -
## chr2 135837179 135837180 LCT ENST00000264162 -
## chr1 225911381 225911382 LEFTY1 ENST00000492457 -
## chr20 5504526 5504527 LINC00654 ENST00000589201 -
## chr21 32575880 32575881 LINC00846 ENST00000334165 -
## chr3 72100956 72100957 LINC00877 ENST00000488545 -
## chr12 132610542 132610543 LRCOL1 ENST00000616042 -
## chr21 46228767 46228768 LSS ENST00000450351 -
## chr2 99304741 99304742 LYG1 ENST00000409448 -
## chrX 20116631 20116632 MAP7D2 ENST00000452324 -
## chr1 11047232 11047233 MASP2 ENST00000400898 -
## chr19 54189567 54189568 MBOAT7 ENST00000338624 -
## chrX 139692155 139692156 MCF2 ENST00000520602 -
## chr22 37486400 37486401 MFNG ENST00000356998 -
## chr4 139280337 139280338 MGARP ENST00000398955 -
## chr7 112738692 112738693 MIPEPP1 ENST00000413533 -
## chr12 57051174 57051175 MYO1A ENST00000433964 -
## chr13 32428310 32428311 N4BP2L1 ENST00000459716 -
## chr12 57240691 57240692 NDUFA4L2 ENST00000393825 -
## chr10 124418935 124418936 OAT ENST00000368845 -
## chr1 36450450 36450451 OSCP1 ENST00000235532 -
## chr1 111427720 111427721 OVGP1 ENST00000369732 -
## chr5 132227814 132227815 P4HA2 ENST00000401867 -
## chr17 27003258 27003259 PDLIM1P3 ENST00000582370 -
## chr1 47191043 47191044 PDZK1IP1 ENST00000371885 -
## chr14 74955625 74955626 PGF ENST00000553716 -
## chr11 33076459 33076460 PIGCP1 ENST00000527583 -
## chr6 85660605 85660606 PKMP3 ENST00000405497 -
## chr19 48110562 48110563 PLA2G4C ENST00000595899 -
## chr5 146104368 146104369 PLAC8L1 ENST00000311450 -
## chr19 42196584 42196585 POU2F2 ENST00000532176 -
## chr10 68010861 68010862 POU5F1P5 ENST00000445059 -
## chr4 158723395 158723396 PPID ENST00000307720 -
## chr6 166308382 166308383 PRR18 ENST00000322583 -
## chr17 63998930 63998931 PRR29-AS1 ENST00000577545 -
```

```
## chr12 56688161 56688162 PTGES3 ENST00000436399 -
## chr1 89104766 89104767 PTGES3P1 ENST00000439531 -
## chr14 102077471 102077472 RN7SL472P ENST00000492799 -
## chr19 1570859 1570860 RN7SL477P ENST00000488440 -
## chr12 56670745 56670746 RN7SL809P ENST00000482040 -
## chr1 1074306 1074307 RNF223 ENST00000453464 -
## chr6 53337345 53337346 RPS16P5 ENST00000406501 -
## chr16 66925535 66925536 RRAD ENST00000420652 -
## chr17 44315314 44315315 RUNDC3A-AS1 ENST00000588097 -
## chr17 50129821 50129822 SAMD14 ENST00000330175 -
## chr12 108633958 108633959 SELPLG ENST00000550948 -
## chr15 74433954 74433955 SEMA7A ENST00000543145 -
## chr1 173917377 173917378 SERPINC1 ENST00000617423 -
## chr1 110391025 110391026 SLC16A4 ENST00000472422 -
## chr12 262835 262836 SLC6A13 ENST00000343164 -
## chr14 23154392 23154393 SLC7A8 ENST00000528860 -
## chr3 43351961 43351962 SNRK-AS1 ENST00000422681 -
## chr10 72088772 72088773 SPOCK2 ENST00000373109 -
## chr15 41894076 41894077 SPTBN5 ENST00000320955 -
## chr22 40856615 40856616 ST13 ENST00000455824 -
## chrX 86086338 86086339 STIP1P3 ENST00000394563 -
## chr15 43618799 43618800 STRC ENST00000450892 -
## chr15 43718183 43718184 STRCP1 ENST00000509801 -
## chr10 73663802 73663803 SYNPO2L ENST00000606523 -
## chr6 159789595 159789596 TCP1 ENST00000544255 -
## chr21 32585532 32585533 TCP10L ENST00000300258 -
## chr19 5567923 5567924 TINCR ENST00000448587 -
## chr10 96513917 96513918 TLL2 ENST00000469598 -
## chr1 1540356 1540357 TMEM240 ENST00000378733 -
## chr20 45833744 45833745 TNNC2 ENST00000372557 -
## chr11 6619447 6619448 TPP1 ENST00000531754 -
## chr17 3597097 3597098 TRPV1 ENST00000571088 -
## chr4 48114417 48114418 TXK ENST00000506073 -
## chr12 124914600 124914601 UBC ENST00000546120 -
## chr3 48563772 48563773 UCN2 ENST00000273610 -
## chr1 156299188 156299189 VHLL ENST00000339922 -
## chr11 57649937 57649938 YPEL4 ENST00000529776 -
## chr5 850504 850505 ZDHHC11 ENST00000508951 -
## chr5 766951 766952 ZDHHC11B ENST00000508859 -
## chr7 1160177 1160178 ZFAND2A ENST00000401903 -
##
## HSF repressed genes
## chr4 7754090 7754091 AFAP1-AS1 ENST00000608442 +
## chr21 46037052 46037053 AP001476.2 ENST00000451618 +
## chr1 147600196 147600197 BCL9 ENST00000497938 +
## chr17 4899763 4899764 C17orf107 ENST00000521575 +
## chr3 112086408 112086409 C3orf52 ENST00000480282 +
## chr7 26638457 26638458 C7orf71 ENST00000614528 +
## chr11 65890784 65890785 CCDC85B ENST00000312579 +
## chr11 69641165 69641166 CCND1 ENST00000539241 +
## chr12 4273772 4273773 CCND2 ENST00000261254 +
## chr19 51225064 51225065 CD33 ENST00000436584 +
## chr14 104865280 104865281 CEP170B ENST00000556508 +
## chr10 11742366 11742367 ECHDC3 ENST00000379215 +
## chr9 136662927 136662928 EGFL7 ENST00000308874 +
## chr5 138465490 138465491 EGR1 ENST00000239938 +
## chr19 16889190 16889191 F2RL3 ENST00000248076 +
## chr1 161599571 161599572 FCGR2C ENST00000508651 +
## chr17 45221789 45221790 FMNL1 ENST00000331495 +
## chr11 68684475 68684476 GAL ENST00000265643 +
## chr19 3585553 3585554 GIPC3 ENST00000322315 +
## chr19 45668244 45668245 GIPR ENST00000590918 +
## chr1 27392644 27392645 GPR3 ENST00000374024 +
## chr5 177426678 177426679 GRK6 ENST00000355472 +
## chr2 74834581 74834582 HK2 ENST00000290573 +
## chrX 150983286 150983287 HMGB3 ENST00000325307 +
## chr12 54037854 54037855 HOXC4 ENST00000507650 +
## chr12 54032853 54032854 HOXC5 ENST00000312492 +
## chr5 95768999 95769000 HSPD1P11 ENST00000508643 +
## chr1 67207616 67207617 IL23R ENST00000425614 +
## chr8 19939441 19939442 LPL ENST00000311322 +
## chr19 35309806 35309807 MAG ENST00000593348 +
## chr13 91347942 91347943 MIR17HG ENST00000581816 +
```

```
## chr20 36541497 36541498 MYL9 ENST00000346786 +
## chr3 185959962 185959963 NMRAL2P ENST00000422108 +
## chr3 47011542 47011543 NRADDP ENST00000437305 +
## chr3 196639757 196639758 NRROS ENST00000461791 +
## chr14 24398786 24398787 NYNRIN ENST00000382554 +
## chr11 120236640 120236641 POU2F3 ENST00000260264 +
## chr18 9615264 9615265 PPP4R1-AS1 ENST00000582435 +
## chr3 191461163 191461164 PYDC2 ENST00000518817 +
## chr19 49542602 49542603 RCN3 ENST00000593483 +
## chr10 47706203 47706204 RHEBP2 ENST00000450289 +
## chr4 185143278 185143279 SLC25A4 ENST00000491736 +
## chr17 7281735 7281736 SLC2A4 ENST00000572485 +
## chr1 28507366 28507367 SNORA73A ENST00000364938 +
## chr1 28508559 28508560 SNORA73B ENST00000363217 +
## chr12 49323273 49323274 TROAP ENST00000549534 +
## chr3 42658814 42658815 ZBTB47 ENST00000505904 +
## chr9 137028287 137028288 ABCA2 ENST00000265662 -
## chr1 161199055 161199056 ADAMTS4 ENST00000367996 -
## chr1 159076900 159076901 AIM2 ENST00000368130 -
## chr3 198080670 198080671 ANKRD18DP ENST00000435620 -
## chr11 3642315 3642316 ART5 ENST00000359918 -
## chr17 3964463 3964464 ATP2A3 ENST00000309890 -
## chrX 40062936 40062937 BCOR ENST00000442018 -
## chr22 23180171 23180172 BCRP8 ENST00000412037 -
## chr12 64286076 64286077 C12orf56 ENST00000541802 -
## chr18 2655394 2655395 CBX3P2 ENST00000579647 -
## chr16 726867 726868 CCDC78 ENST00000474647 -
## chr12 9760900 9760901 CD69 ENST00000228434 -
## chr1 159900162 159900163 CFAP45 ENST00000368099 -
## chr15 84408004 84408005 CSPG4P5 ENST00000558731 -
## chr5 39424867 39424868 DAB2 ENST00000503513 -
## chr15 84401730 84401731 DNM1P51 ENST00000558149 -
## chr2 201623429 201623430 ENO1P4 ENST00000416471 -
## chr17 43546314 43546315 ETV4 ENST00000585508 -
## chr1 16073631 16073632 FAM131C ENST00000494078 -
## chr7 6939602 6939603 FAM86LP ENST00000478302 -
## chr11 65900418 65900419 FOSL1 ENST00000448083 -
## chr3 10290913 10290914 GHRL ENST00000491589 -
## chr13 99307404 99307405 GPR183 ENST00000376414 -
## chr7 128405947 128405948 IMPDH1 ENST00000496200 -
## chr5 170297783 170297784 LCP2 ENST00000519149 -
## chr14 55796687 55796688 LINC00520 ENST00000560336 -
## chr20 62775411 62775412 LINC00659 ENST00000412500 -
## chr18 77982838 77982839 LINC01029 ENST00000583106 -
## chr19 19628528 19628529 LPAR2 ENST00000588461 -
## chr1 32336032 32336033 MARCKSL1 ENST00000329421 -
## chr5 67003952 67003953 MAST4-AS1 ENST00000451496 -
## chr16 1872143 1872144 MEIOB ENST00000496541 -
## chr17 43833169 43833170 MPP3 ENST00000496503 -
## chr16 4474541 4474542 NMRAL1 ENST00000573571 -
## chr9 137459333 137459334 NSMF ENST00000371475 -
## chr10 49762322 49762323 OGDHL ENST00000374103 -
## chr11 5469421 5469422 OR51A10P ENST00000307165 -
## chr1 25871252 25871253 PAQR7 ENST00000374296 -
## chr11 31811307 31811308 PAX6 ENST00000533333 -
## chr5 177497600 177497601 PDLIM7 ENST00000486828 -
## chr15 40291352 40291353 PLCB2 ENST00000560701 -
## chr19 821951 821952 PLPPR3 ENST00000359894 -
## chr7 39013550 39013551 POU6F2-AS2 ENST00000420243 -
## chr9 120842910 120842911 PSMD5 ENST00000476949 -
## chr10 46635465 46635466 RHEBP1 ENST00000448647 -
## chr7 149125985 149125986 RN7SL521P ENST00000488398 -
## chr17 17384011 17384012 RPL13P12 ENST00000392730 -
## chr7 93118022 93118023 SAMD9 ENST00000620985 -
## chr7 93148344 93148345 SAMD9L ENST00000437805 -
## chr7 100573065 100573066 SAP25 ENST00000611464 -
## chr21 43427127 43427128 SIK1 ENST00000270162 -
## chr10 99620363 99620364 SLC25A28 ENST00000370495 -
## chr13 80340950 80340951 SPRY2 ENST00000377104 -
## chr19 3606839 3606840 TBXA2R ENST00000375190 -
## chr4 38856816 38856817 TLR6 ENST00000436693 -
## chr5 139482295 139482296 TMEM173 ENST00000511850 -
## chr2 677438 677439 TMEM18 ENST00000281017 -
```

```
## chr6 28218633 28218634 TOB2P1 ENST00000469761 -
## chr15 23039568 23039569 TUBGCP5 ENST00000614508 -
## chr11 68004068 68004069 UNC93B1 ENST00000531152 -
## chr19 45076372 45076373 ZNF296 ENST00000303809 -
## chr20 63969864 63969865 ZNF512B ENST00000369888 -
## chr19 34964048 34964049 ZNF792 ENST00000404801 -
```

182 repressed.

455 activated.

3 ChIP-seq and proximity analysis

ChIP-seq analysis of HSF binding. Call peaks with MACS2 (Zhang *et al.*, 2008) using the raw data from (Lo and Matthews, 2012). I am doing this differnetly than Anni, from reading the manuscript it looks like the input was used as background and then a filtering step was used with the IgG. I suspect that Anni took care and time to annotate a good peak set, so it may be best to download Anni's hg19 peaks and `liftOver` coordinates to hg38. We also use `fastaFromBed` and `slopBed` from `bedtools`.

Software installations:

`MACS2`: `https://github.com/taoliu/MACS`

`liftOver`: `http://hgdownload.soe.ucsc.edu/admin/exe/`

`bedtools`: `https://bedtools.readthedocs.io/en/latest/`

The ChIP-seq data is GSE43579 (Vihervaara *et al.*, 2013). Consider this an example for ChIP-seq peak calling, but we are not moving forward with these peaks for this vignette.

3.1 Calling ChIP-seq peaks

```
#ChIP-seq GSE43579
#GSM1065711: HSF1_non-treated cycling K562; Homo sapiens; ChIP-Seq
fastq-dump SRR650331
#GSM1065713: IgG_non-treated cycling K562; Homo sapiens; ChIP-Seq
fastq-dump SRR650333
#GSM1065714: HSF1_heat-treated cycling K562; Homo sapiens; ChIP-Seq
fastq-dump SRR650334

gzip SRR*fastq

mv SRR650331.fastq.gz K562_NHS_HSF1_chip.fastq.gz
mv SRR650333.fastq.gz K562_IgG_HSF1_chip.fastq.gz
mv SRR650334.fastq.gz K562_30HS_HSF1_chip.fastq.gz


for fq in *chip.fastq.gz
do
    name=$(echo $fq | awk -F"/" '{print $NF}' | awk -F".fastq.gz" '{print $1}')
    echo $name
    bowtie2 -p 3 -x /Volumes/GUERTIN_seq/hg38 -U ${fq} | \
       samtools view -b - | \
       samtools sort - -o ${name}.sorted.bam
done


ctrl=K562_IgG_HSF1_chip.sorted.bam
for i in K562*HS_HSF1_chip.sorted.bam
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F".sorted.bam" '{print $1}')
    echo $name
    macs2 callpeak -t $i -c $ctrl --nomodel --extsize 150 -q 0.001 -f BAM \
        --keep-dup 10 -n $name -g hs -B -m 10 100
done

for narrow in *.narrowPeak
do
```

```
    name=$(echo $narrow | awk -F"/" '{print $NF}' | awk -F"_peaks" '{print $1}')
    echo $name
    touch  tempnarrow.txt
    echo "track type=narrowPeak name=$name.peaks" >> tempnarrow.txt
    cat tempnarrow.txt $narrow > $name.narrowPeak
    rm $narrow
    rm tempnarrow.txt
done

for bdg in *treat_pileup.bdg
do
    name=$(echo $bdg | awk -F"/" '{print $NF}' | awk -F"_treat_pileup" '{print $1}')
    echo $name
    touch temp.txt
    echo "track type=bedGraph name=$name" >> temp.txt
    sort -k1,1 -k2,2n $bdg > $name.sorted.bdg
    cat temp.txt $name.sorted.bdg > $name.bedGraph
    rm temp.txt
    rm $name.sorted.bdg
    bedGraphToBigWig $name.bedGraph hg38.chrom.sizes ${name}.bigWig
    gzip $name.bedGraph
done

#i did not like what I was seeing from the peak calling, Anni's is probably more curated


#CTCF
wget https://www.encodeproject.org/files/ENCFF101MTI/@@download/ENCFF101MTI.bed.gz
gunzip ENCFF101MTI.bed.gz
mv ENCFF101MTI.bed K562_CTCF_ChIP_seq_conservative_ENCODE.bed
wget https://www.encodeproject.org/files/ENCFF233ZLL/@@download/ENCFF233ZLL.bigWig
mv ENCFF233ZLL.bigWig K562_CTCF_ChIP_seq.bigWig
mkdir ctcf_encode
mv K562_CTCF_ChIP_seq.bigWig ctcf_encode
mv K562_30HS_HSF1_chip.bigWig ctcf_encode
mv K562_NHS_HSF1_chip.bigWig ctcf_encode
#use this to view bigWig on UCSC
#https://www.encodeproject.org/files/ENCFF233ZLL/@@download/ENCFF233ZLL.bigWig?proxy=true
```

3.2 Using existing ChIP-seq peaks

I did not like what I was seeing from the peak calling, Anni's is probably more curated. There are too many peaks and when I look at them in the browser it is clearly calling background signal peaks. I don't think I can do IDR ChIP-seq peak calling without replicates for the HSF data.

The *hg19* peaks files can be directly downloaded from (Vihervaara *et al.*, 2013). Getting bed ChIP-seq peak summit files with `R`. This can be easily done in `basic shell`, but I always have to relearn the syntax...

```
hs.peaks.hg19 = read.table('sd01_U_HS_HSF1.txt', skip=1, header=FALSE, sep = '\t')
nhs.peaks = read.table('sd01_U_NHS_HSF1.txt', skip=1, header=FALSE, sep = '\t')
ctcf.peaks = read.table('K562_CTCF_ChIP_seq_conservative_ENCODE.bed', skip=0,
                        header=FALSE, sep = '\t')
hs.peaks.summit = hs.peaks.hg19
hs.peaks.summit[,2] = hs.peaks.summit[,2] + hs.peaks.summit[,5]
hs.peaks.summit[,3] = hs.peaks.summit[,2] + 1
hs.peaks.summit[,6] = '+'

nhs.peaks.summit = nhs.peaks
nhs.peaks.summit[,2] = nhs.peaks.summit[,2] + nhs.peaks.summit[,5]
nhs.peaks.summit[,3] = nhs.peaks.summit[,2] + 1
nhs.peaks.summit[,6] = '+'

ctcf.peaks.summit = ctcf.peaks
ctcf.peaks.summit[,2] = ctcf.peaks.summit[,2] + ctcf.peaks.summit[,10]
ctcf.peaks.summit[,3] = ctcf.peaks.summit[,2] + 1
ctcf.peaks.summit[,6] = '+'


write.table(hs.peaks.summit[,c(1:6)], file = 'hs.peaks.summit.bed', sep='\t',
            quote=FALSE, row.names = FALSE, col.names=FALSE)
```

```
write.table(nhs.peaks.summit[,c(1:6)], file = 'nhs.peaks.summit.bed', sep='\t',
            quote=FALSE, row.names = FALSE, col.names=FALSE)
write.table(ctcf.peaks.summit[,c(1:6)], file = 'ctcf.peaks.summit.bed', sep='\t',
            quote=FALSE, row.names = FALSE, col.names=FALSE)
```

3.3 Motif enrichment at HSF peaks

Next we wanted to validate the peak quality by finding the *de novo* with motif analysis with `meme`. The first step is to retrieve a database of position weight matrices to query any *de novo* motifs against–this is only necessary if you don't know what an HSE looks like. We will use the HOMER database (Heinz *et al.*, 2010) and `TOMTOM` software from the `MEME` suite (Bailey *et al.*, 2009).

Software installations:

`MEME suite`: `http://meme-suite.org`

```
#the HSF peaks were hg19, they need to be lifted over to hg38 reference genome
#the following is the key
wget hgdownload.soe.ucsc.edu/goldenPath/hg19/liftOver/hg19ToHg38.over.chain.gz
gunzip hg19ToHg38.over.chain.gz

for i in *s.peaks.summit.bed
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F".summit.bed" '{print $1}')
    echo $name
    liftOver $i hg19ToHg38.over.chain $name.hg38.broadPeak $name.hg38.unmapped.txt -bedPlus=6
    slopBed -i $name.hg38.broadPeak -g hg38.chrom.sizes -b 70 > $name.slop.bed
    fastaFromBed -fi hg38.fa -bed $name.slop.bed -fo $name.window.fasta
    meme -o $name.meme_chip_output -dna -revcomp -minw 5 -maxw 20 $name.window.fasta
done

for i in ctcf.peaks.summit.bed
do
    name=$(echo $i | awk -F"/" '{print $NF}' | awk -F".summit.bed" '{print $1}')
    echo $name
    slopBed -i ${name}.summit.bed -g hg38.chrom.sizes -b 70 > $name.slop.bed
    fastaFromBed -fi hg38.fa -bed $name.slop.bed -fo $name.window.fasta
    meme -o $name.meme_chip_output -dna -revcomp -minw 5 -maxw 20 $name.window.fasta
done
```

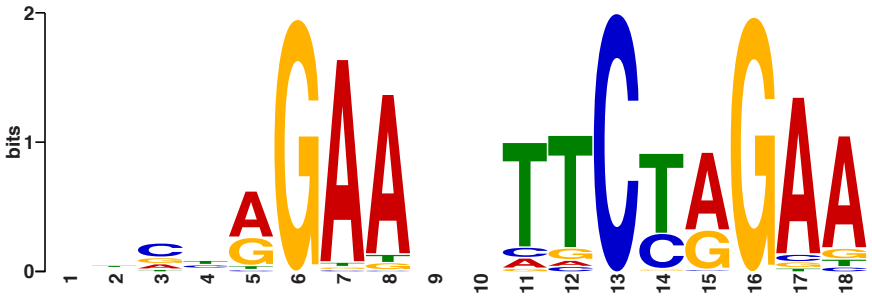


Figure 4: The heat shock element is the top motif. This figure is modified from the eps file within *hs.peaks.meme_chip_output*

3.4 Python script: HOMER_MEME_conversion.py

This script convert the position weight matrices to a compatible MEME format (Bailey *et al.*, 2006).

```
#! /usr/bin/python
import sys
import getopt
def matrix(filename, outfilename_prefix):
    infile=open(filename, 'r')
    nmotifs = 0
    outfile=open(str(outfilename_prefix) + '_meme.txt', 'w')
    outfile.write('%s\n'%("MEME version 4"))
    outfile.write('\n')
    outfile.write('%s\n'%("ALPHABET= ACGT"))
    outfile.write("\n")
    outfile.write('%s\n'%("strands: + -"))
    outfile.write("\n")
    outfile.write('%s\n'%("A 0.212 C 0.288 G 0.288 T 0.212"))
    count = 0
    while 1:
        line = infile.readline()
        if not line: break
        splitline = line.split()
        directory = filename.split('meme.txt')[0]
        if line.startswith('>'):
            count = count + 1
            outfile.write("\n")
            namemotif = str.split(splitline[1], "(")[0]
            print namemotif
            outfile.write('%s%s%s%s\n'%("MOTIF ", splitline[1], ' ', namemotif))
            str(len(splitline[0]) - 1)
            outfile.write('%s\n'%("letter-probability matrix: alength= 4"))
        else:
            outfile.write(line)
    print count
    outfile.close()
    return
def main(argv):
    try:
        opts, args = getopt.getopt(argv, "hi:o:", ["help", "input=", "out="])
    except getopt.GetoptError, err:
        print str(err)
        sys.exit(2)
    name = False
    outname = False
    for opt, arg in opts:
        if opt in ('-i', '--input'):
            name = arg
        if opt in ('-o', '--out'):
            outname = arg
        elif opt in ('-h', '--help'):
            print 'python HOMER_MEME_conversion.py -i custom.motifs -o custom.motifs'
            sys.exit()
    if name and outname:
        matrix(name, outname)
if __name__ == "__main__":
    main(sys.argv[1:])
```

Next we compare to the HOMER database with TOMTOM. Of course you know this is an HSE, but this excercise is useful if you ever need figure out the associated TF for a PSWM.

```
#HOMER
wget http://homer.ucsd.edu/homer/custom.motifs

python HOMER_MEME_conversion.py -i custom.motifs -o custom.motifs

tomtom -o hsf_chip.tomtom_output \
    hs.peaks.meme_chip_output/meme.txt \
    custom.motifs_meme.txt
```

4 HSF and CTCF proximity analysis

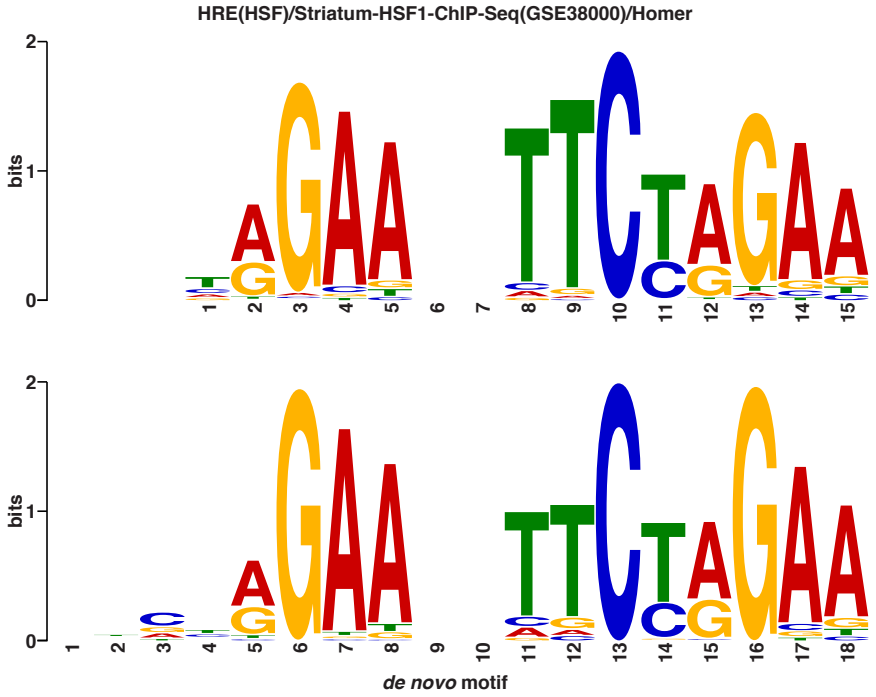


Figure 5: The heat shock element is matched. This figure is modified from the html file in *hsf_chip.tomtom_output*

```
dir = paste0(path.expand("~"),'/HSF/')
hs.peaks = read.table(paste0(dir, 'hs.peaks.hg38.broadPeak'), sep = '\t', header =FALSE)
hs.peaks = hs.peaks[grep('KI2708', hs.peaks[,1], invert = TRUE),]

load(file = paste0(dir, 'df.hs.deseq.effects.lattice.Rdata'))

colnames(df.hs.deseq.effects.lattice) =
    c(colnames(df.hs.deseq.effects.lattice)[c(1:(length(colnames(df.hs.deseq.effects.lattice))-1))],
      'arfauxin')

df.all.hs = cdf.deseq.df(df = df.hs.deseq.effects.lattice,
            genes = gene.file, chip.peaks ='hs.peaks.hg38.broadPeak',
            treat = '30min HS', tf.name = 'HSF')

##      V1        V2        V3   V4   V5 V6
## 1 chr16  89161524  89161525 1263  724  +
## 2  chr6  31827549  31827550 3566  969  +
## 3  chr2 197500097 197500098 1027  592  +
## 4  chr6  31815391  31815392 3856 1063  +
## 5  chr7  76294809  76294810 1515  795  +
## 6  chr7  27734733  27734734 1188  569  +
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed

pdf(paste0(dir, "Figure_cdf_compare_Reg_classes_HS.pdf"), width=4.2, height=3.83)

        col.lines = c("#FF0000" , "grey60", "#0000FF","grey90")
        ecdfplot(~log(abs(dis), base = 10), groups = status, data = df.all.hs,
                 auto.key = list(lines=TRUE, points=FALSE),
                 col = col.lines,
                 aspect = 1,
                 scales=list(relation="free",alternating=c(1,1,1,1)),
                 ylab = 'Cumulative Distribution Function',
                 xlab = expression('log'[10]~'HSF1 Distance from TSS'),
                 between=list(y=1.0),
                 type = 'a',
                 xlim = c(0,7.5),
                 lwd=2,
                 par.settings = list(superpose.line = list(col = col.lines, lwd=3),
                                     strip.background=list(col="grey85")),
                 panel = function(...) {
                     panel.abline(v= 200, lty =2)
                     panel.ecdfplot(...)
                 })
dev.off()

## pdf
##   2

ks.hs.ActvUnc = ks.test(x = log(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Activated']),
        y = log(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Unchanged']))$p.value

## Warning in ks.test(x = log(abs(df.all.hs$dis)[df.all.hs$status == "30min HS Activated"]), :  p-value will be approximate
in the presence of ties

ks.hs.ActvUnc

## [1] 0

ks.hs.ActvUnc = formatC(ks.hs.ActvUnc, format = "e", digits = 20)
ks.hs.RepvUnc = ks.test(x = log(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Repressed']),
        y = log(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Unchanged']))$p.value

## Warning in ks.test(x = log(abs(df.all.hs$dis)[df.all.hs$status == "30min HS Repressed"]), :  p-value will be approximate
in the presence of ties

ks.hs.RepvUnc = formatC(ks.hs.RepvUnc, format = "e", digits = 3)

save(ks.hs.ActvUnc, ks.hs.RepvUnc, file = paste0(dir, 'ks.hs.Rdata'))
```

I am proceeding with defining HSF / CTCF distances, I can use the data frame `df.all.hs` from the previous chunk to identify genes within a set distance from HSF. My plan is to use bedtools to identify CTCF and HSF proximity, so for each HSF peak I have the closer CTCF peak, then carry this

information over to the existing CDF analysis and categorize genes by HSF and CTCF distances...

Here we define the parameter `hsf.ctcf.dis` as the proximity of CTCF to HSF and thus we define the *Proximal* and *Distal* categories.

Software installation note
I hard coded R functions to call the closestBed bedtools binary, so I generate and put all the binaries in the following directory: /usr/local/bin/bedtools/

Therefore, you should be able to paste `/usr/local/bin/bedtools/closestBed` into a Terminal session and it will output the usage docs.

```
#
#example
#

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#change parameter to define CTCF proximity to HSF

hsf.ctcf.dis = 2500

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ctcf.peaks = read.table(paste0(dir, 'ctcf.peaks.summit.bed'), sep = '\t', header =FALSE)
hsf.ctcf.peaks = bedTools.closest(bed1 = hs.peaks, bed2 = ctcf.peaks, opt.string = '-D a')

## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#making a catergorical column from existing continuous data measurements

hsf.ctcf.peaks$ctcf = 'CTCF_Distal'
hsf.ctcf.peaks[abs(hsf.ctcf.peaks$dis) < hsf.ctcf.dis,]$ctcf = 'CTCF_Proximal'

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

colnames(hsf.ctcf.peaks) = c('hsf.chr', 'hsf.start', 'hsf.end',
                             'hsf.metric1', 'hsf.metric2', 'hsf.strand',
                             'ctcf.chr', 'ctcf.start', 'ctcf.end',
                             'ctcf.metric1', 'ctcf.metric2', 'ctcf.strand',
                             'ctcf.hsf.dis', 'ctcf')


#this function is not well written, comments indicate why
cdf.deseq.df.mods <- function(df, genes = gene.file,
                              peaks = hsf.ctcf.peaks,
                              treat = 'Auxin', tf.name = 'ZNF143') {
#I am copy/pasting text as opposed to writing functions.
    bed.tss.activated = filter.deseq.into.bed(df, genes, cat = paste(treat, 'Activated'))
    bed.tss.repressed = filter.deseq.into.bed(df, genes, cat = paste(treat, 'Repressed'))
    bed.tss.unchanged = filter.deseq.into.bed(df, genes, cat = paste(treat, 'Unchanged'))
    bed.tss.dregs = filter.deseq.into.bed(df, genes, cat = paste(treat, 'All Other Genes'))
    act.distance = bedTools.closest(bed1 = bed.tss.activated, bed2 = peaks, opt.string = '-D a')
    unreg.distance = bedTools.closest(bed1 = bed.tss.unchanged, bed2 = peaks, opt.string = '-D a')
    repress.distance = bedTools.closest(bed1 = bed.tss.repressed, bed2 = peaks, opt.string = '-D a')
    dregs.distance = bedTools.closest(bed1 = bed.tss.dregs, bed2 = peaks, opt.string = '-D a')

    df.up.can = cbind(act.distance, paste(treat, 'Activated'), "Canonical")
    df.un.can = cbind(unreg.distance, paste(treat, 'Unchanged'), "Canonical")
    df.down.can = cbind(repress.distance, paste(treat, 'Repressed'), "Canonical")
    df.dregs.can = cbind(dregs.distance, cat = paste(treat, 'All Other Genes'), "Canonical")

    print(head(df.up.can))
#the er column name is not useful.
    colnames(df.up.can) = c(colnames(df.up.can)[1:21], 'status', 'er')
    colnames(df.un.can) = c(colnames(df.up.can)[1:21], 'status', 'er')
    colnames(df.down.can) = c(colnames(df.up.can)[1:21], 'status', 'er')
    colnames(df.dregs.can) = c(colnames(df.up.can)[1:21], 'status', 'er')

    df.all = rbind(df.up.can, df.un.can, df.down.can, df.dregs.can)
#the col.names is hard coded.
    colnames(df.all) = c('chr.tss', 'start.tss', 'end.tss', 'gene', 'id','strand.tss', colnames(df.all)[7:23])
    return(df.all)
}

#remove chrY genes for plotting
df.dis.reg = cdf.deseq.df.mods(df = df.hs.deseq.effects.lattice,
            genes = gene.file, peaks = hsf.ctcf.peaks,
            treat = '30min HS', tf.name = 'HSF')
```

```
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
## sort -k1,1 -k2,2n a.file.bed > a.file.sorted.bed
## sort -k1,1 -k2,2n b.file.bed > b.file.sorted.bed
## /usr/local/bin/bedtools/closestBed -D a -a a.file.sorted.bed -b b.file.sorted.bed > out.file.bed
##     V1       V2       V3      V4              V5 V6 hsf.chr hsf.start
## 1 chr1  1074306  1074307  RNF223 ENST00000453464  -    chr1   1215540
## 2 chr1  1331314  1331315  TAS1R3 ENST00000339381  +    chr1   1215540
## 3 chr1  1540356  1540357 TMEM240 ENST00000378733  -    chr1   1780280
## 4 chr1 11047232 11047233   MASP2 ENST00000400898  -    chr1  10969390
## 5 chr1 11934694 11934695   PLOD1 ENST00000449038  +    chr1  11911253
## 6 chr1 26921726 26921727    NUDC ENST00000321265  +    chr1  26921514
##    hsf.end hsf.metric1 hsf.metric2 hsf.strand ctcf.chr ctcf.start ctcf.end
## 1  1215541        1165         561          +     chr1    1207130  1207131
## 2  1215541        1165         561          +     chr1    1207130  1207131
## 3  1780281        1018         390          +     chr1    1779148  1779149
## 4 10969391         907         378          +     chr1   10961185 10961186
## 5 11911254         956         526          +     chr1   11907850 11907851
## 6 26921515        1220         694          +     chr1   26878150 26878151
##   ctcf.metric1 ctcf.metric2 ctcf.strand ctcf.hsf.dis          ctcf     dis
## 1            .         1000          +        -8410   CTCF_Distal -141234
## 2            .         1000          +        -8410   CTCF_Distal -115774
## 3            .         1000          +        -1132 CTCF_Proximal -239924
## 4            .         1000          +        -8205   CTCF_Distal   77842
## 5            .         1000          +        -3403   CTCF_Distal  -23441
## 6            .         1000          +       -43364   CTCF_Distal    -212
##   paste(treat, "Activated") "Canonical"
## 1        30min HS Activated   Canonical
## 2        30min HS Activated   Canonical
## 3        30min HS Activated   Canonical
## 4        30min HS Activated   Canonical
## 5        30min HS Activated   Canonical
## 6        30min HS Activated   Canonical

pdf(paste0(dir, "Figure_cdf_CTCF_HSF_compare_Reg_classes_HS.pdf"), width=6.2, height=3.83)

        col.lines = c("#FF0000" , "grey60", "#0000FF","grey90")
        ecdfplot(~log(abs(dis), base = 10) | ctcf, groups = status, data = df.dis.reg,
                 auto.key = list(lines=TRUE, points=FALSE),
                 col = col.lines,
                 aspect = 1,
                 scales=list(relation="free",alternating=c(1,1,1,1)),
                 ylab = 'Cumulative Distribution Function',
                 xlab = expression('log'[10]~'HSF1 Distance from TSS'),
                 between=list(y=1.0),
                 type = 'a',
                 xlim = c(0,7.5),
                 lwd=2,
                 par.settings = list(superpose.line = list(col = col.lines, lwd=3),
                                     strip.background=list(col="grey85")),
                 panel = function(...) {
                     #panel.abline(v= 200, lty =2)
                     panel.abline(v= 4.477121, lty =2)
                     panel.ecdfplot(...)
                 })
dev.off()

## pdf
##   2

write.table(df.dis.reg, file = 'df.dis.reg.txt', sep = '\t',
            quote=FALSE, col.names = FALSE, row.names = FALSE)

df.hs.deseq.effects.lattice$gene = sapply(strsplit(rownames(df.hs.deseq.effects.lattice), '_'), '[', 2)

df.dis.reg.deseq = merge(df.hs.deseq.effects.lattice, df.dis.reg, by.x='gene', by.y='gene')
write.table(df.dis.reg.deseq, file = 'df.dis.reg.deseq.txt', sep = '\t',
            quote=FALSE, col.names = FALSE, row.names = FALSE)


save(df.dis.reg.deseq, file = paste0(path.expand("~"),'/HSF/df.dis.reg.deseq.Rdata'))
```

We find that HSF1 binding, as measured by ChIP-seq, is enriched proximal to the activated gene class (Kolmogorov–Smirnov two-sided p-value = $0.00000000000000000000e+00$) and the repressed gene class (Kolmogorov–Smirnov two-sided p-value = $3.395e-04$). This is similar to *Drosophila* HSF (Duarte *et al.*, 2016).

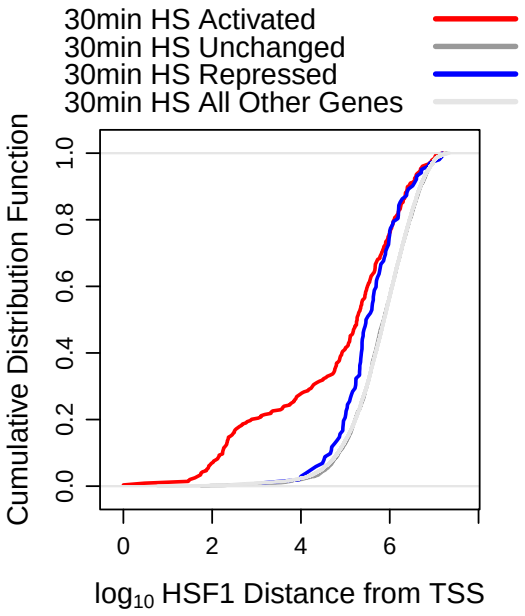


Figure 6: Activated genes are proximal to HSF1 binding sites.

4.1 Most of information to ask questions is in this data frame

I believe that the R data frame: `df.dis.reg.deseq` (exported the file df.dis.reg.deseq.txt too), is all one needs to do many analyses. All ChIP-seq peaks are on the plus strand, so the relationship between CTCF and HSF proximity is the column `ctcf.hsf.dis` and a negative distance means CTCF is closer to start of the chromosome. The distance is in base pairs. Each gene's TSS and the coding strand is included. The column `dis` indicates where HSF is binding relative to the coding strand. The categorical variables are defined with parameter settings, such as *Activated*, *CTCF_Proximal*, etc, so we can either re-run the dynamic document with different parameters and make a new PDF each time...or simply start redefining the categorical columns dynamically in R (a little practice with the data frame and you will be able to do this). Note that the `er` column is not useful. It is also noteworthy that this data frame has all the information to determine whether CTCF is oriented between HSF and a TSS or not (I think). I will write up an exercise with this data frame that will make a new catergorical column, *ctcf.orientation*, based on the orientation of CTCF relative to HSF and the TSS criteria.

I think that a first priority it to define HSF-bound genes. I am thinking about the most epirical way to do this. I am inclined to use a distance where the slope (rate of change, derivative) of the CDF function is consistently less steep than the control genes. My rationale is that at this distance, the accumulation in each class at increasing distance is the same (or higher rate in unchanged), so any HSF binding is likely incidental.

update: the following works intuitively for me...I simply subtract the activated genes CDF and repressed genes CDF and look for where the plot flattens out, this is where the CDFs are parallel (accumulating at the same rate). I can do this with the unchanged gene class once I determine the proper characteristics to use to find a set of genes that is directly comparable to the activated and repressed classes. Again, it concerns me that it traces over the all other genes class in near parity to the unregulated class.

```
act = ecdf(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Activated'])
rep = ecdf(abs(df.all.hs$dis)[df.all.hs$status == '30min HS Repressed'])

act.y = seq(0, 2000000, by=1000)
rep.y = seq(0, 2000000, by=1000)

spl <- smooth.spline(act.y, act(act.y) - rep(rep.y))
pred <- predict(spl)


pdf(paste0(dir, "empirical_distance_determination.pdf"), width=3.83, height=3.83)
plot(act.y, act(act.y) - rep(rep.y), xlim = c(0,1000000), cex=0.7,
           xlab = 'HSF distance from TSS (bp)',
           ylab = 'Activated Genes CDF - Repressed genes CDF')
      abline(v = 30000, col =2, lty =2)
      abline(h = 0.2, col =2, lty =2)
      lines(pred, col = 'blue')
dev.off()
```

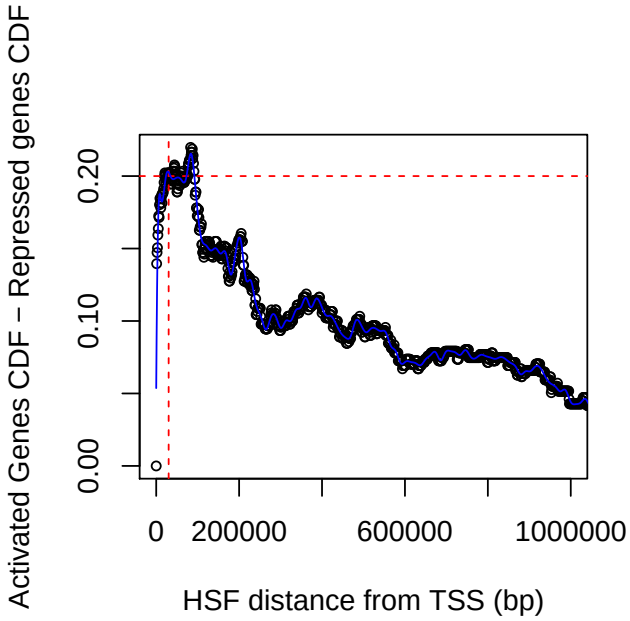


Figure 7: 30,000 bases seems like a good distance

```
load(paste0(path.expand("~"),'/HSF/df.dis.reg.deseq.Rdata'))

pdf(paste0(dir, "Figure_cdf_CTCF_HSF_compare_Reg_classes_HS_win30kb.pdf"), width=6.2, height=3.83)
        col.lines = c("#FF0000" , "grey60", "#0000FF","grey90")
        x = df.dis.reg.deseq[abs(df.dis.reg.deseq$dis) < 30000,]
        ecdfplot(~log(abs(x$dis), base = 10) | x$ctcf,
                 groups = x$status,
                 auto.key = list(lines=TRUE, points=FALSE),
                 col = col.lines,
                 aspect = 1,
                 scales=list(relation="free",alternating=c(1,1,1,1)),
                 ylab = 'Cumulative Distribution Function',
                 xlab = expression('log'[10]~'HSF1 Distance from TSS'),
                 between=list(y=1.0),
                 type = 'a',
                 xlim = c(0,7.5),
                 lwd=2,
                 par.settings = list(superpose.line = list(col = col.lines, lwd=3),
                                     strip.background=list(col="grey85")),
```

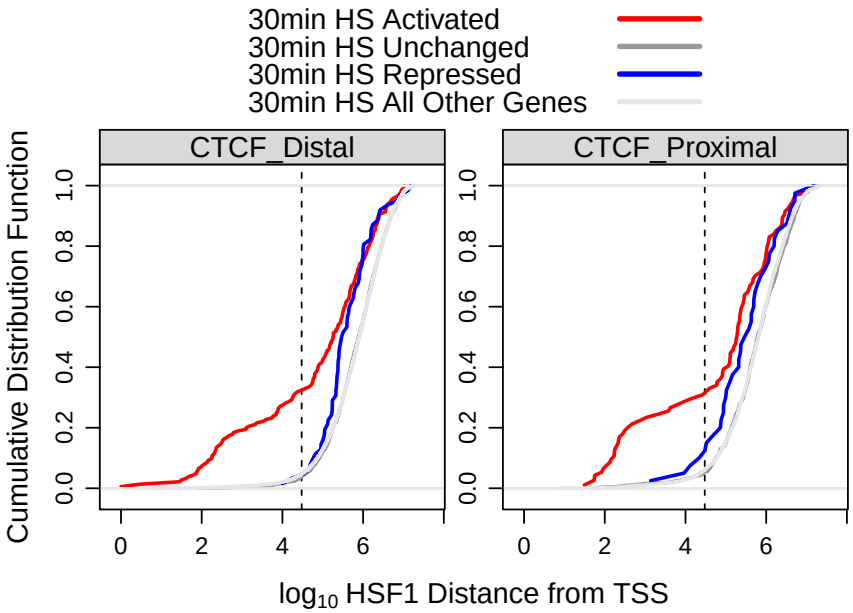


Figure 8: Activated genes are proximal to HSF1 binding sites. CTCF proximity may affect repressed genes. Next I look at the genes to the left of the 30kb line exclusively. I still have to think about whether looking at genes only 30kb from HSF is introducing biases. I am sure there are biases, but I don't want over 70% of the genes in the *activated* class to be considered HSF bound if HSF binding is likely incidental and not functional.

```
                panel = function(...) {
                    #panel.abline(v= 200, lty =2)
                    panel.abline(v= 4.477121, lty =2)
                    panel.ecdfplot(...)
                })
dev.off()

## pdf
##   2
```

5 UCSC tracks to browse

I loaded the `bedGraph` files directly to UCSC. Let me know if this link does not work. `https://genome.ucsc.edu/s/Mike%20Guertin/hg38_HSF_PRO_chip_CTCF`

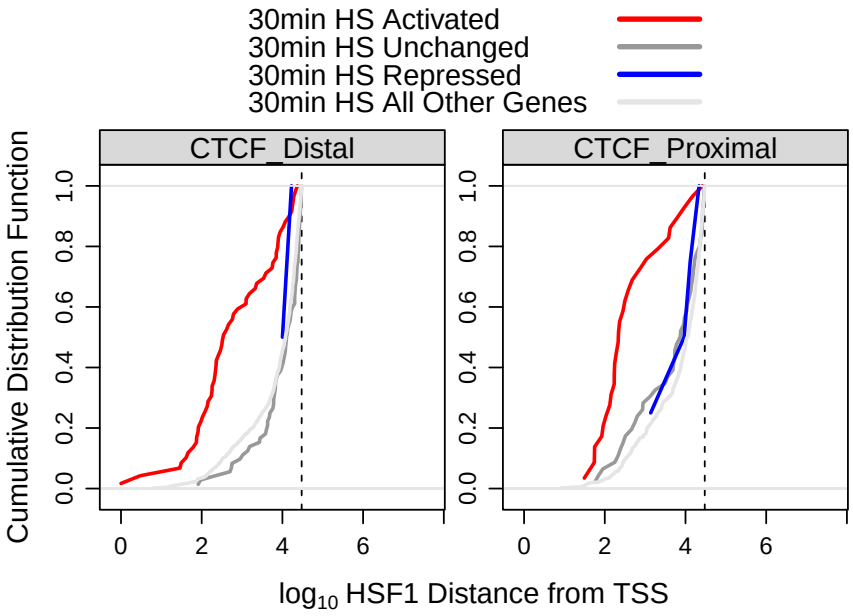


Figure 9: only genes close to HSF

6 Categorical designation and plotting exercise

I can send you *df.dis.reg.deseq.Rdata* if you do not want to execute all the all the chunks above. Having all the relevant information for each gene (or any genomic feature) in one data frame is important. First we will make a new column that identifies genes as HSF-proximal or distal based on an arbitrary distance.

```
load(paste0(path.expand("~"),'/HSF/df.dis.reg.deseq.Rdata'))

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hsf.gene.distance = 30000

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#make
df.dis.reg.deseq$hsf.gene = NA
#recall the dis column is distance of HSF from TSS
#abs takes the absolute value

df.dis.reg.deseq$hsf.gene[abs(df.dis.reg.deseq$dis) < hsf.gene.distance] = 'HSF_proximal'
df.dis.reg.deseq$hsf.gene[abs(df.dis.reg.deseq$dis) >= hsf.gene.distance] = 'HSF_distal'

#now you have a new column with this categorical variable
#you can plot panels or draw traces based on this new category
#no frills box whisker plots example below

#are HSF-proximal genes, on average, increasing expression relative to distal?
pdf(paste0(path.expand("~"),'/HSF/test_bwplot_1.pdf'), useDingbats = FALSE,
    width=4.0, height=4.0)

bwplot(df.dis.reg.deseq$log2FoldChange ~ df.dis.reg.deseq$hsf.gene,
       horizontal =FALSE,
       scales=list(relation="free",rot = 30, alternating=c(1,1,1,1)),
       ylab = expression('log'[2]*' (change upon HS)'),
       xlab = expression('HSF Distance Category'),
       pch = '|',
       aspect = 1.0,
       do.out = TRUE
 )
dev.off()

## pdf
##   2

#does the CTCF proximity variable make a difference?
pdf(paste0(path.expand("~"),'/HSF/test_bwplot_2.pdf'), useDingbats = FALSE,
    width=4.0, height=4.0)

bwplot(df.dis.reg.deseq$log2FoldChange ~ df.dis.reg.deseq$ctcf |
           df.dis.reg.deseq$hsf.gene,
       horizontal =FALSE,
     scales=list(relation="free",rot = 30, alternating=c(1,1,1,1)),
       ylab = expression('log'[2]*' (change upon HS)'),
       xlab = expression('CTCF Distance Category'),
       pch = '|',
       aspect = 1.0,
       do.out = TRUE
 )
dev.off()

## pdf
##   2
```

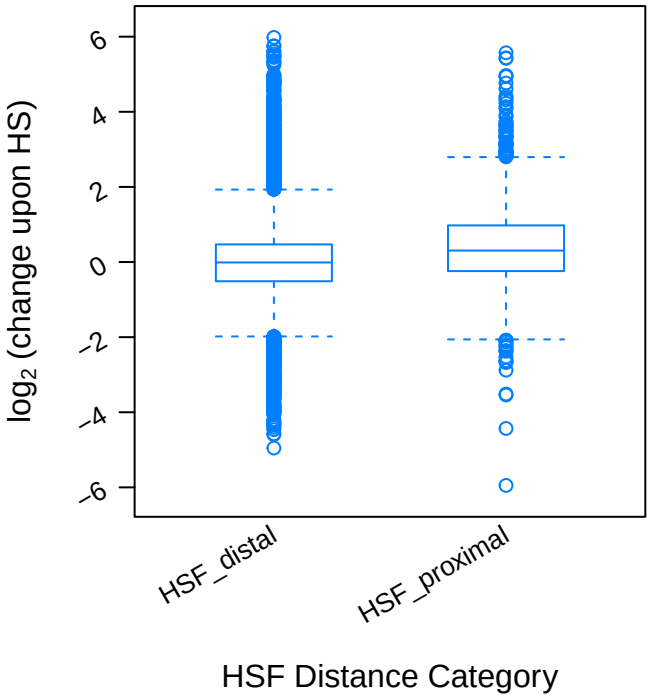


Figure 10: HSF-proximal genes tend to be activated more than distal genes

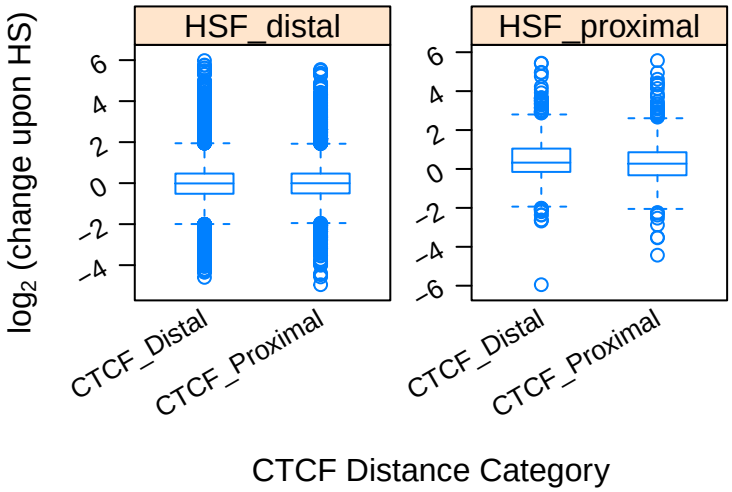


Figure 11: HSF-proximal genes that are also CTCF-proximal may be activated less than CTCF-distal genes

6.1 CTCF/HSF/gene orientation subsetting

This exercise is a bit more complicated. We will determine whether CTCF intervening between a proximal HSF binding site and a promoter acts to insulate from HSF's activation function.

```
load(paste0(path.expand("~"),'/HSF/df.dis.reg.deseq.Rdata'))

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

hsf.promoter.distance = -5000

#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
#%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

#define genes with HSF in promoter
x = df.dis.reg.deseq
x$hsf.promoter = 'HSF_nonPromoter'
x$hsf.promoter[x$dis > hsf.promoter.distance &
               x$dis < 0] = 'HSF_promoter'

#consider the orientation of each gene and CTCF/HSF
#this just looks at the orientation of the closest CTCF binding site
#this is a limitation of how the data is structured.

x$ctcf.orientation = 'CTCF_notIntervening'

#plus strand and minus strand considered separately

x$ctcf.orientation[(x$strand.tss == '+' &
                    (x$dis > 0 & x$ctcf.hsf.dis > 0 & x$ctcf.hsf.dis < x$dis)) |
 (x$strand.tss == '-' &
                    (x$dis < 0 & x$ctcf.hsf.dis < 0 & x$ctcf.hsf.dis > x$dis))] =
    'CTCF_intervening'

#considering both categories

pdf(paste0(path.expand("~"),'/HSF/test_bwplot_3.pdf'), useDingbats = FALSE,
    width=5.0, height=5.0)

bwplot(x$log2FoldChange ~ x$ctcf.orientation |
           x$hsf.promoter,
       horizontal =FALSE,
     scales=list(relation="free",rot = 30, alternating=c(1,1,1,1)),
       ylab = expression('log'[2]*' (change upon HS)'),
       xlab = expression('CTCF Orientation Category'),
       pch = '|',
       aspect = 1.0,
       do.out = TRUE
 )
dev.off()

## pdf
##   2

#if we only want to consider cases where HSF is promoter-bound
#and overlay a violin plot

y = x[x$hsf.promoter == 'HSF_promoter',]
pdf(paste0(path.expand("~"),'/HSF/test_bwplot_4.pdf'), useDingbats = FALSE,
    width=5.0, height=5.0)

bwplot(y$log2FoldChange ~ y$ctcf.orientation,
       horizontal =FALSE,
     scales=list(relation="free",rot = 30, alternating=c(1,1,1,1)),
       ylab = expression('log'[2]*' (change upon HS)'),
       xlab = expression('CTCF Orientation Category'),
       pch = '|',
       aspect = 1.0,
       do.out = TRUE,
      panel = function(..., box.ratio, col, pch) {
          panel.violin(..., col = 'white',
                       varwidth = FALSE, box.ratio = box.ratio)
           panel.bwplot(..., col = 'black', pch = '|')

       }
 )
dev.off()

## pdf
##   2
```

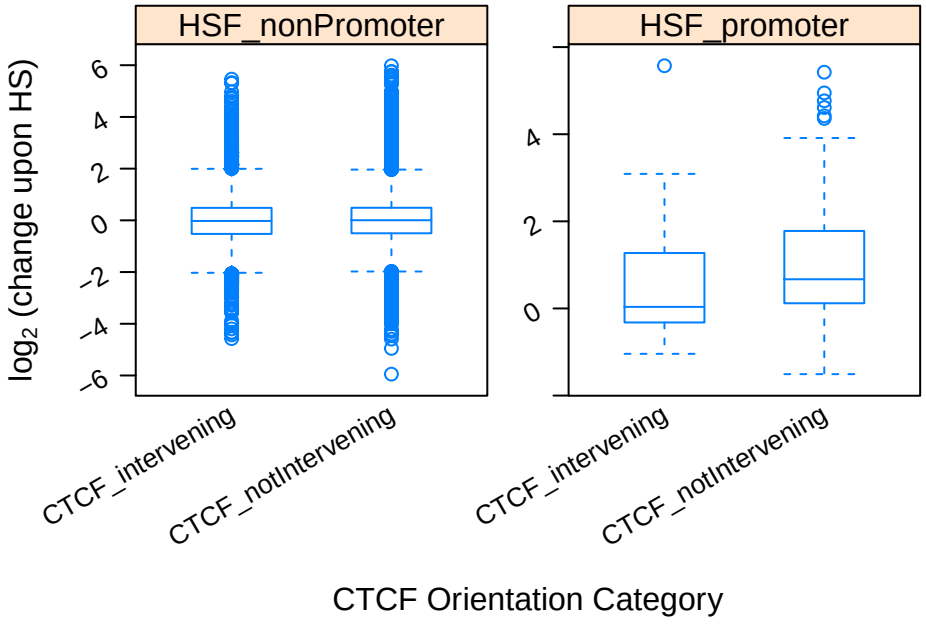


Figure 12: CTCF intervening between HSF and a TSS insulatino effect.

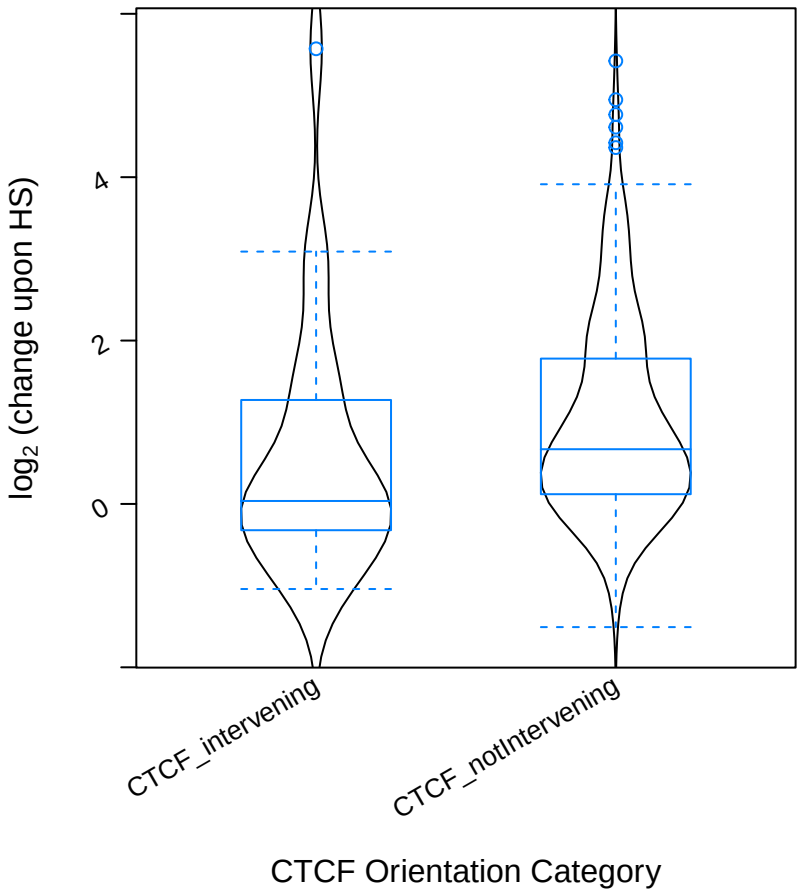


Figure 13: CTCF may insulate genes from promoter-bound HSF.

7 HSF peak-centric analyses

I need to clean up these functions and make them more flexible:

```
window.step.chip <- function(bed, bigWig, halfWindow, step) {
  windowSize = (2*halfWindow) %/% step
  midPoint = floor((as.numeric(as.character(bed[,2])) + as.numeric(as.character(bed[,3]))) / 2)
  start = (midPoint - halfWindow)
  end = start + windowSize*step
  if ((as.numeric(as.character(bed[1,2])) + as.numeric(as.character(bed[1,3]))) %% 2 == 0) {
    bed[,2] = start
    bed[,3] = end
  } else {
    bed[,2] = start
    bed[,3] = end
    bed[,2][bed[,6] == '-'] = bed[,2][bed[,6] == '-'] + 1
    bed[,3][bed[,6] == '-'] = bed[,3][bed[,6] == '-'] + 1
  }
  #matrix.comp = bed.region.bpQuery.bigWig(bigWig, bed, op = "avg")
  matrix.comp = bed.step.probeQuery.bigWig(bigWig, bed,
                                           gap.value = 0, step = step)
  print(head(matrix.comp))
  res = do.call(rbind, matrix.comp)
  return(list(res, matrix.comp))
}


composites.chip <- function(path.dir, composite.input, region=20, step=1, grp = 'DNase') {
  vec.names = c('chr','start','end')
  hmap.data = list()
  composite.df=data.frame(matrix(ncol = 6, nrow = 0))
  for (mod.bigWig in Sys.glob(file.path(path.dir, "*.bigWig"))) {
      factor.name = strsplit(strsplit(mod.bigWig,
                "/")[[1]][length(strsplit(mod.bigWig, "/")[[1]])], '\\.')[[1]][1]
      print(factor.name)
      vec.names = c(vec.names, factor.name)
      wiggle = load.bigWig(mod.bigWig)
      bpquery = window.step.chip(composite.input, wiggle, region, step)
      subsample = subsampled.quantiles.metaprofile((bpquery[[1]]))
      #alternative functions: bootstrapped.confinterval.metaprofile,
      #confinterval.metaprofile, subsampled.quantiles.metaprofile
      mult.row = ncol(bpquery[[1]])
      hmap.data[[factor.name]] = bpquery[[1]]
      df.up <- data.frame(matrix(ncol = 6, nrow = mult.row))
      df.up[, 1] <- colMeans(bpquery[[1]])
      df.up[, 2] <- seq((-1 * region) + 0.5 * step, region - 0.5 * step, by = step)
      df.up[, 3] <- matrix(data = factor.name, nrow=mult.row, ncol=1)
      df.up[, 4] <- subsample$top
      df.up[, 5] <- subsample$bottom
      df.up[, 6] <- matrix(data = grp, nrow=mult.row, ncol=1)
      composite.df = rbind(composite.df, df.up)
      unload.bigWig(wiggle)
  }
  colnames(composite.df) <- c('est', 'x', 'cond', 'upper', 'lower', 'grp')
  composite.df = composite.df[composite.df[,2] >= -1000 & composite.df[,2] <= 1000,]
  for (cond in (1:length(hmap.data))) {
      rownames(hmap.data[[cond]]) = paste(composite.input[,1], ':',
                                          composite.input[,2], '-', composite.input[,3], sep='')
      colnames(hmap.data[[cond]]) = seq((-1 * region) + 0.5 * step,
                                        region - 0.5 * step, by = step)
  }
  return(list(composite.df, hmap.data))
}


composites.func.panels.plot <- function(dat, fact = 'Factor',
                                                    summit = 'Summit', num=800,
              col.lines = c(rgb(0,0,1,1/2), rgb(1,0,0,1/2),  rgb(0.1,0.5,0.05,1/2), rgb(0,0,0,1/2),
              rgb(1/2,0,1/2,1/2), rgb(0,1/2,1/2,1/2), rgb(1/2,1/2,0,1/2)), fill.poly = c(rgb(0,0,1,1/4),
              rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4)),
              ypl = list(c(0,20), c(0, 5), c(0,1))) {
  count = length(unique(dat$grp))
  ct.cons = 0
  lst.cons = list()
  unique(dat$grp)[order(unique(dat$grp))]
  for (i in unique(dat$grp)[order(unique(dat$grp))]) {
      ct.cons= ct.cons + 1
      lst.cons[[ct.cons]] = c(min(dat[dat$grp == i,]$lower), max(dat[dat$grp == i,]$upper))
  }
  pdf(paste('composite_', fact, '_signals_', summit, '_peaks.pdf', sep=''), width=7.43,
      height=3.8)
  print(xyplot(est ~ x|cond, group = cond, data = dat,
               type = 'l',
               scales=list(x=list(cex=0.8,relation = "free"), y =list(cex=0.8, relation="free")),
               xlim=c(-(num),(num)),
               #ylim = lst.cons,
```

```
              ylim= ypl,
              col = col.lines,
              auto.key = list(points=F, lines=T, cex=0.8),
              par.settings = list(superpose.symbol = list(pch = c(16),
                   col=col.lines, cex =0.5),
                   strip.background=list(col="grey80"),
                  superpose.line = list(col = col.lines, lwd=c(2,2,2,2,2),
                      lty = c(1,1,1,1,1,1,1,1,1))),
              cex.axis=1.0,
              par.strip.text=list(cex=0.9, font=1, col='black'),
              aspect=1.0,
              between=list(y=0.5, x=0.5),
              #lwd=2,
              ylab = list(label = paste(fact," ChIP Intensity", sep=''), cex =0.8),
              xlab = list(label = paste("Distance from ", summit, " center",sep=''), cex =0.8),
              upper = dat$upper,
              fill = fill.poly,
              lower = dat$lower,
              #strip = function(..., which.panel, bg) {
              #  bg.col = c("grey85")
              #  strip.default(..., which.panel = which.panel, bg = rep(bg.col, length = which.panel)[which.panel])
            #},
              panel = function(x, y, ...){
                  panel.superpose(x, y, panel.groups = 'my.panel.bands', ...)
                  panel.xyplot(x, y, ...)
      }
  ))
  dev.off()
}



composites.func.panels.plot.2 <- function(dat, fact = 'Factor',
                                          summit = 'Summit', num=800,
             col.lines = c(rgb(0,0,1,1/2), rgb(1,0,0,1/2),  rgb(0.1,0.5,0.05,1/2), rgb(0,0,0,1/2),
             rgb(1/2,0,1/2,1/2), rgb(0,1/2,1/2,1/2), rgb(1/2,1/2,0,1/2)), fill.poly = c(rgb(0,0,1,1/4),
             rgb(1,0,0,1/4), rgb(0.1,0.5,0.05,1/4),rgb(0,0,0,1/4), rgb(1/2,0,1/2,1/4)),
             ypl = list(c(0,20), c(0, 5), c(0,1))) {
  count = length(unique(dat$grp))
  ct.cons = 0
  lst.cons = list()
  unique(dat$grp)[order(unique(dat$grp))]
  for (i in unique(dat$grp)[order(unique(dat$grp))]) {
      ct.cons= ct.cons + 1
      lst.cons[[ct.cons]] = c(min(dat[dat$grp == i,]$lower), max(dat[dat$grp == i,]$upper))
  }
  pdf(paste('composite_', fact, '_signals_', summit, '_peaks.pdf', sep=''), width=7.43,
      height=3.8)
  print(xyplot(est ~ x|cond, group = grp, data = dat,
              type = 'l',
              scales=list(x=list(cex=0.8,relation = "free"), y =list(cex=0.8, relation="free")),
              xlim=c(-(num),(num)),
              #ylim = lst.cons,
              ylim= ypl,
              col = col.lines,
              auto.key = list(points=F, lines=T, cex=0.8),
              par.settings = list(superpose.symbol = list(pch = c(16),
                   col=col.lines, cex =0.5),
                   strip.background=list(col="grey80"),
                  superpose.line = list(col = col.lines, lwd=c(2,2,2,2,2),
                      lty = c(1,1,1,1,1,1,1,1,1))),
              cex.axis=1.0,
              par.strip.text=list(cex=0.9, font=1, col='black'),
              aspect=1.0,
              between=list(y=0.5, x=0.5),
              #lwd=2,
              ylab = list(label = paste(fact," ChIP Intensity", sep=''), cex =0.8),
              xlab = list(label = paste("Distance from ", summit, " center",sep=''), cex =0.8),
              upper = dat$upper,
              fill = fill.poly,
              lower = dat$lower,
              #strip = function(..., which.panel, bg) {
              #  bg.col = c("grey85")
              #  strip.default(..., which.panel = which.panel, bg = rep(bg.col, length = which.panel)[which.panel])
            #},
              panel = function(x, y, ...){
                  panel.superpose(x, y, panel.groups = 'my.panel.bands', ...)
                  panel.xyplot(x, y, ...)
      }
  ))
  dev.off()
}


composite.hsf1 = composites.chip(paste0(path.expand("~"),'/HSF/ctcf_encode'), hs.peaks[,1:6], region = 2000, step = 50, grp = 'CTCF')
composites.func.panels.plot(composite.hsf1[[1]], fact = 'Factor', summit= 'HSF1_peak_summit',
                            num = 800, ypl = list(c(1,25), c(1, 7), c(1,3)))
```

```
composite.hsf1.ctcf.distal = composites.chip(paste0(path.expand("~"),'/HSF/ctcf_encode'),
                                hsf.ctcf.peaks[hsf.ctcf.peaks$ctcf == 'CTCF_Distal',][,1:6],
                                region = 2000, step = 50, grp = 'CTCF_Distal')
composite.hsf1.ctcf.proximal = composites.chip(paste0(path.expand("~"),'/HSF/ctcf_encode'),
                                hsf.ctcf.peaks[hsf.ctcf.peaks$ctcf == 'CTCF_Proximal',][,1:6],
                                region = 2000, step = 50, grp = 'CTCF_Proximal')

both.hsf1.ctcf = rbind(composite.hsf1.ctcf.distal[[1]], composite.hsf1.ctcf.proximal[[1]])
composites.func.panels.plot.2(both.hsf1.ctcf, fact = 'Factor', summit= 'HSF1_peak_summit_2',
                            num = 800, ypl = list(c(1,26), c(1, 20), c(1,3)))



raw.intensity.chip <- function(path.dir, composite.input, region=200) {
  vec.names = c(colnames(composite.input), 'fctr', 'intensity')
  print(vec.names)
  composite.df=data.frame(matrix(ncol = ncol(composite.input) + 2, nrow = 0))
  colnames(composite.df) = vec.names
  composite.input[,2] = composite.input[,2] - region/2
  composite.input[,3] = composite.input[,2] + region
  for (mod.bigWig in Sys.glob(file.path(path.dir, "*.bigWig"))) {
      factor.name = strsplit(strsplit(mod.bigWig,
                "/")[[1]][length(strsplit(mod.bigWig, "/")[[1]])], '\\.')[[1]][1]
      print(factor.name)
      #vec.names = c(vec.names, factor.name)
      wiggle = load.bigWig(mod.bigWig)
      x = bed.region.bpQuery.bigWig(wiggle, composite.input, op = "avg")
      y = cbind(composite.input, factor.name, x)
      colnames(y) = vec.names
      composite.df = rbind(composite.df, y)
      unload.bigWig(wiggle)
  }
  return(composite.df)
}


raw.chip.intensity = raw.intensity.chip(paste0(path.expand("~"),'/HSF/ctcf_encode'),
                                    hsf.ctcf.peaks, region = 200)

pdf(paste0(path.expand("~"),'/HSF/test_bwplot_5.pdf'), useDingbats = FALSE,
    width=8.0, height=4.0)

bwplot(log(intensity, base = 2) ~ ctcf|fctr, data = raw.chip.intensity,
       horizontal =FALSE,
       scales=list(relation="free",rot = 30, alternating=c(1,1,1,1)),
       ylab = expression('ChIP-seq intensity'),
       xlab = expression('HSF Distance Category'),
       pch = '|',
       par.settings = list(superpose.symbol = list(pch = c(16),
                    col='black', cex =0.5),
                    box.umbrella = list(lty = 1, col="black", lwd=2),
                    box.rectangle = list(col = 'black', lwd=1.6),
                    plot.symbol = list(col='black', lwd=1.6, pch =19, cex = 0.5),
                    strip.background=list(col="grey80")
                   ),
       aspect = 1.0,
       do.out = TRUE
 )
dev.off()



pdf(paste0(dir, "Figure_cdf_CTCFvsHSF.pdf"), width=4, height=3.83)

        col.lines = c("#FF0000" , "grey60", "#0000FF","grey90")
        ecdfplot(~log(abs(ctcf.hsf.dis), base = 10), data = hsf.ctcf.peaks,
                 auto.key = list(lines=TRUE, points=FALSE),
                 col = col.lines,
                 aspect = 1,
                 scales=list(relation="free",alternating=c(1,1,1,1)),
                 ylab = 'Cumulative Distribution Function',
                 xlab = expression('log'[10]~'CTCF Distance from HSF1'),
                 between=list(y=1.0),
                 type = 'a',
                 xlim = c(0,7.5),
                 lwd=2,
                 par.settings = list(superpose.line = list(col = col.lines, lwd=3),
                                     strip.background=list(col="grey85")),
                 panel = function(...) {
                     panel.abline(h= 0.32, lty =2)
                     panel.abline(v= 3.39794, lty =2)
                     panel.ecdfplot(...)
                 })
dev.off()
```

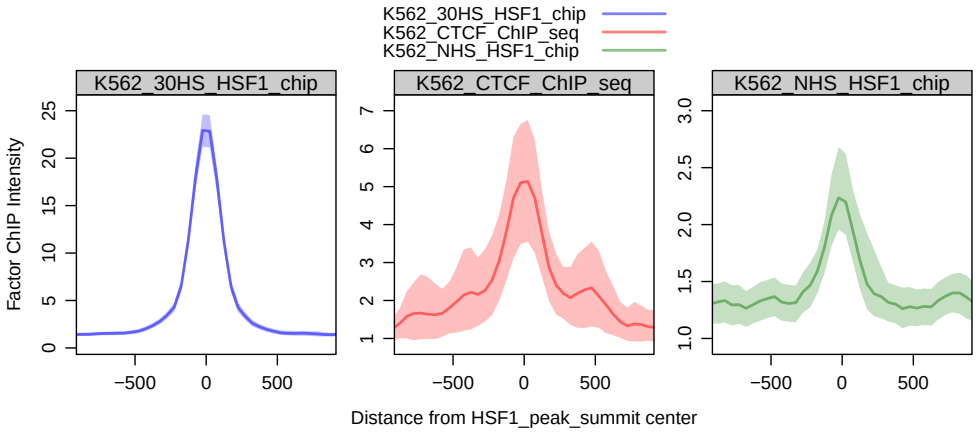


Figure 14: CTCF and HSF intensities peaks near HSF1 peak summits.

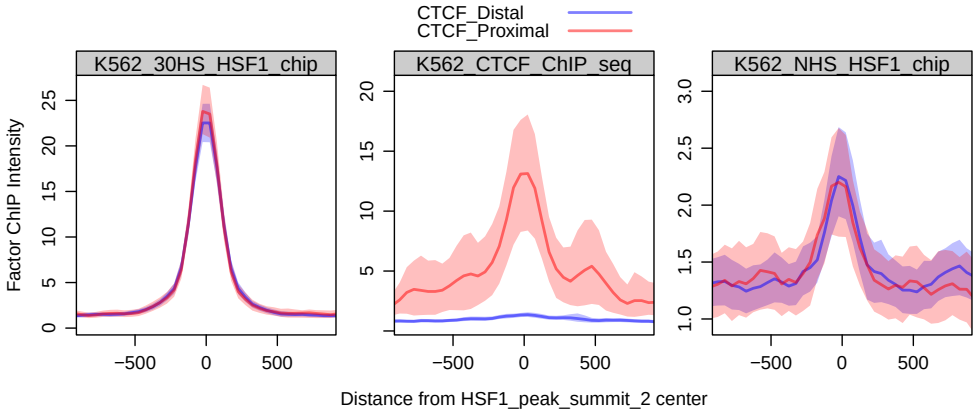


Figure 15: CTCF and HSF intensities peaks near HSF1 peak summits, categorized by CTCF proximity.

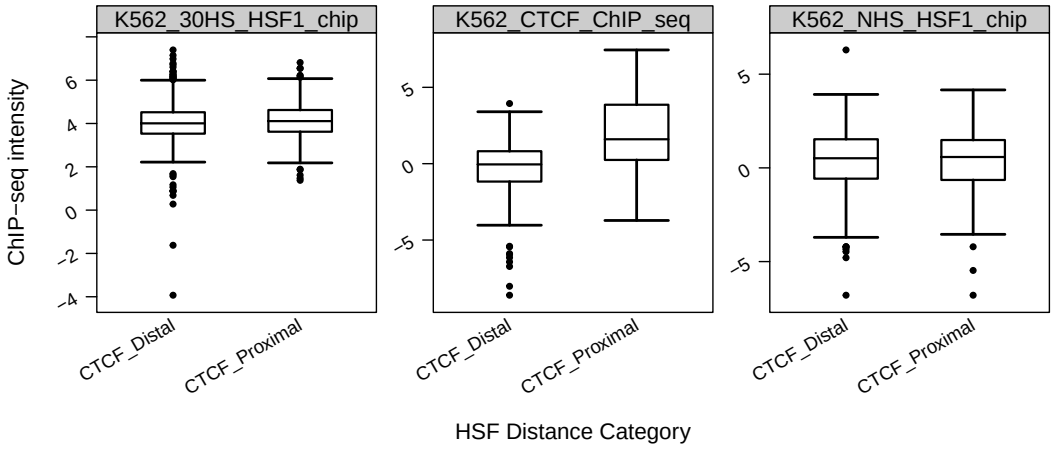


Figure 16: each panel is a different ChIP-seq experiment. The CTCF ChIP is a positive control; CTCF signal is indeed higher at HSF1 binding sites that are categorized as within 2.5kb of a CTCF peak.

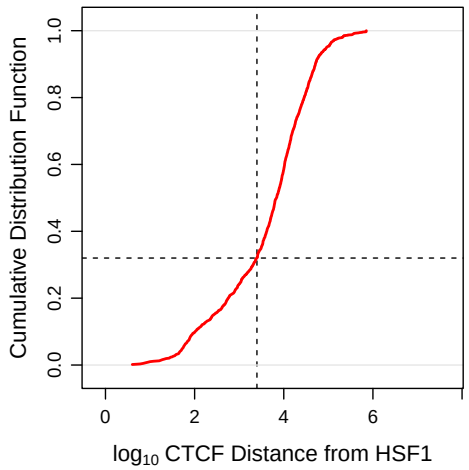


Figure 17: 32% of HSF1 peaks are defined as CTCF-proximal using a 2.5kb threshold. It is unclear whether this is a large proportion compared to other factors.

8 Next

I am going to look more closely at your emails and Word document to figure out what to look at.

I am reasonably happy with the idea of using 30kb as a distance (i.e. ignore all genes greater than 30kb from HSF binding sites). That said, I will continue to make the document dynamic so we can just change this one value and re-run to make the figures.

I will also code all the KS statistics into the document text for the CTCF proximal and distal traces.

I am writing this in `knitR`, so it is a dynamic document and we can change the initialized fold-changes and FDR values and re-make the PDF with numbers of repressed and activated genes changing accordingly and the figures are updated with the new thresholds. The only additional code someone needs to reproduce this document is `MJG_sathyan_functions.R`, which I am actively maintaining for another project, but it will be in Github when the other paper is in bioRXiv. I can provide it to you if you want to try to learn analysis, just let me know. I think jumping in to this may be a bit steep without intro to R or an intro CS class.

References

Bailey TL, Boden M, Buske FA, Frith M, Grant CE, Clementi L, Ren J, Li WW, Noble WS (2009). "MEME SUITE: tools for motif discovery and searching." *Nucleic acids research*, p. gkp335.

Bailey TL, Williams N, Misleh C, Li WW (2006). "MEME: discovering and analyzing DNA and protein sequence motifs." *Nucleic acids research*, **34**(suppl 2), W369–W373.

Duarte FM, Fuda NJ, Mahat DB, Core LJ, Guertin MJ, Lis JT (2016). "Transcription factors GAF and HSF act at distinct regulatory steps to modulate stress-induced gene activation." *Genes & development*.

Heinz S, Benner C, Spann N, Bertolino E, Lin YC, Laslo P, Cheng JX, Murre C, Singh H, Glass CK (2010). "Simple combinations of lineage-determining transcription factors prime cis-regulatory elements required for macrophage and B cell identities." *Molecular cell*, **38**(4), 576–589.

Kodama Y, Shumway M, Leinonen R (2011). "The Sequence Read Archive: explosive growth of sequencing data." *Nucleic acids research*, **40**(D1), D54–D56.

Kuhn RM, Haussler D, Kent WJ (2012). "The UCSC genome browser and associated tools." *Briefings in bioinformatics*, **14**(2), 144–161.

Langmead B, Trapnell C, Pop M, Salzberg S (2009). "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome." *Genome Biology*, **10**(3), R25. ISSN 1465-6906. doi: 10.1186/gb-2009-10-3-r25. URL `http://genomebiology.com/2009/10/3/R25`.

Li H, Handsaker B, Wysoker A, Fennell T, Ruan J, Homer N, Marth G, Abecasis G, Durbin R, *et al.* (2009). "The sequence alignment/map format and SAMtools." *Bioinformatics*, **25**(16), 2078–2079.

Lo R, Matthews J (2012). "High-resolution genome-wide mapping of AHR and ARNT binding sites by ChIP-Seq." *Toxicological sciences*, **130**(2), 349–361.

Mahat DB, Kwak H, Booth GT, Jonkers IH, Danko CG, Patel RK, Waters CT, Munson K, Core LJ, Lis JT (2016). "Base-pair-resolution genome-wide mapping of active RNA polymerases using precision nuclear run-on (PRO-seq)." *Nature protocols*, **11**(8), 1455.

Martin M (2011). "Cutadapt removes adapter sequences from high-throughput sequencing reads." *EMBnet. journal*, **17**(1), pp–10.

Martins AL, Walavalkar NM, Anderson WD, Zang C, Guertin MJ (2018). "Universal correction of enzymatic sequence bias reveals molecular signatures of protein/DNA interactions." *Nucleic acids research*.

Quinlan AR, Hall IM (2010). "BEDTools: a flexible suite of utilities for comparing genomic features." *Bioinformatics*, **26**(6), 841–842.

Vihervaara A, Mahat DB, Guertin MJ, Chu T, Danko CG, Lis JT, Sistonen L (2017). "Transcriptional response to stress is pre-wired by promoter and enhancer architecture." *Nature communications*, **8**(1), 255.

Vihervaara A, Sergelius C, Vasara J, Blom MA, Elsing AN, Roos-Mattjus P, Sistonen L (2013). "Transcriptional response to stress in the dynamic chromatin environment of cycling and mitotic cells." *Proceedings of the National Academy of Sciences*, **110**(36), E3388–E3397.

Zhang Y, Liu T, Meyer CA, Eeckhoute J, Johnson DS, Bernstein BE, Nusbaum C, Myers RM, Brown M, Li W, *et al.* (2008). "Model-based analysis of ChIP-Seq (MACS)." *Genome Biol*, **9**(9), R137.