# VAR-Seq workflow template: Some Descriptive Title

Project ID: VARseq_PI_Name_Organism_Jul2015
Project PI: First Last (first.last@inst.edu)
Author of Report: First Last (first.last@inst.edu)

October 21, 2016

## Contents



## 1 Introduction

### 1.1 Background and objectives

This report describes the analysis of a VAR-Seq project studying the genetic differences among several strains ... from *organism* ....

### 1.2 Experimental design

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

## 2 Load workflow environment

### 2.1 Load packages and sample data

The *systemPipeR* package needs to be loaded to perform the analysis steps shown in this report (**?**).

```
library(systemPipeR)
```

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

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

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

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

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

```
source("systemPipeVARseq_Fct.R")
```

### 2.2 Experiment definition provided by `targets` file

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

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

```
                  FileName1                FileName2 SampleName Factor SampleLong
1  ./data/SRR446027_1.fastq ./data/SRR446027_2.fastq        M1A     M1  Mock.1h.A
2  ./data/SRR446028_1.fastq ./data/SRR446028_2.fastq        M1B     M1  Mock.1h.B
3  ./data/SRR446029_1.fastq ./data/SRR446029_2.fastq        A1A     A1   Avr.1h.A
4  ./data/SRR446030_1.fastq ./data/SRR446030_2.fastq        A1B     A1   Avr.1h.B
5  ./data/SRR446031_1.fastq ./data/SRR446031_2.fastq        V1A     V1   Vir.1h.A
6  ./data/SRR446032_1.fastq ./data/SRR446032_2.fastq        V1B     V1   Vir.1h.B
7  ./data/SRR446033_1.fastq ./data/SRR446033_2.fastq        M6A     M6  Mock.6h.A
8  ./data/SRR446034_1.fastq ./data/SRR446034_2.fastq        M6B     M6  Mock.6h.B
9  ./data/SRR446035_1.fastq ./data/SRR446035_2.fastq        A6A     A6   Avr.6h.A
10 ./data/SRR446036_1.fastq ./data/SRR446036_2.fastq        A6B     A6   Avr.6h.B
11 ./data/SRR446037_1.fastq ./data/SRR446037_2.fastq        V6A     V6   Vir.6h.A
12 ./data/SRR446038_1.fastq ./data/SRR446038_2.fastq        V6B     V6   Vir.6h.B
13 ./data/SRR446039_1.fastq ./data/SRR446039_2.fastq       M12A    M12 Mock.12h.A
14 ./data/SRR446040_1.fastq ./data/SRR446040_2.fastq       M12B    M12 Mock.12h.B
15 ./data/SRR446041_1.fastq ./data/SRR446041_2.fastq       A12A    A12  Avr.12h.A
16 ./data/SRR446042_1.fastq ./data/SRR446042_2.fastq       A12B    A12  Avr.12h.B
17 ./data/SRR446043_1.fastq ./data/SRR446043_2.fastq       V12A    V12  Vir.12h.A
18 ./data/SRR446044_1.fastq ./data/SRR446044_2.fastq       V12B    V12  Vir.12h.B
```

## 3 Read preprocessing

### 3.1 Read quality filtering and trimming

The following removes reads with low quality base calls (here Phred scores below 20) from all FASTQ files.

```
args <- systemArgs(sysma="param/trimPE.param", mytargets="targetsPE.txt")[1:4] # Note: subsetting!
filterFct <- function(fq, cutoff=20, Nexceptions=0) {
    qcount <- rowSums(as(quality(fq), "matrix") <= cutoff)
    fq[qcount <= Nexceptions] # Retains reads where Phred scores are >= cutoff with N exceptions
}
preprocessReads(args=args, Fct="filterFct(fq, cutoff=20, Nexceptions=0)", batchsize=100000)
writeTargetsout(x=args, file="targets_PEtrim.txt", overwrite=TRUE)
```

### 3.2 FASTQ quality report

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

```
args <- systemArgs(sysma="bwa.param", mytargets="targets.txt")
fqlist <- seeFastq(fastq=infile1(args), batchsize=100000, klength=8)
pdf("./results/fastqReport.pdf", height=18, width=4*length(fqlist))
seeFastqPlot(fqlist)
dev.off()
```

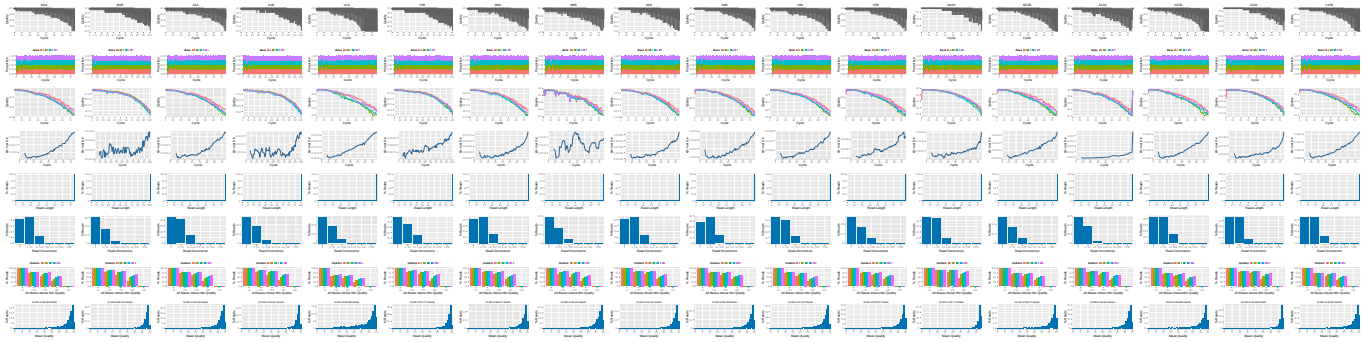

Figure 1: QC report for 18 FASTQ files.

# 4 Alignments

## 4.1 Read mapping with BWA

The NGS reads of this project are aligned against the reference genome sequence using the highly variant tolerant short read aligner BWA (**??**). The parameter settings of the aligner are defined in the `bwa.param` file.

```
args <- systemArgs(sysma="bwa.param", mytargets="targets.txt")
sysargs(args)[1] # Command-line parameters for first FASTQ file
```

Runs the alignments sequentially (*e.g.* on a single machine)

```
bampaths <- runCommandline(args=args)
```

Alternatively, the alignment jobs can be submitted to a compute cluster, here using 72 CPU cores (18 `qsub` processes each with 4 CPU cores).

```
moduleload(modules(args))
system("bwa index -a bwtsw ./data/tair10.fasta")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", cores(args)), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
```

Check whether all BAM files have been created

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

## 4.2 Read mapping with gsnap

An alternative variant tolerant aligner is gsnap from the *gmapR* package (**?**). The following code shows how to run this aligner on multiple nodes of a comute cluster that uses Torque as scheduler.

```
library(gmapR); library(BiocParallel); library(BatchJobs)
gmapGenome <- GmapGenome(reference(args), directory="data", name="gmap_tair10chr", create=TRUE)
args <- systemArgs(sysma="gsnap.param", mytargets="targetsPE.txt")
f <- function(x) {
    library(gmapR); library(systemPipeR)
    args <- systemArgs(sysma="gsnap.param", mytargets="targetsPE.txt")
    gmapGenome <- GmapGenome(reference(args), directory="data", name="gmap_tair10chr", create=FALSE)
    p <- GsnapParam(genome=gmapGenome, unique_only=TRUE, molecule="DNA", max_mismatches=3)
    o <- gsnap(input_a=infile1(args)[x], input_b=infile2(args)[x], params=p, output=outfile1(args)[x])
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
d <- bplapply(seq(along=args), f)
writeTargetsout(x=args, file="targets_gsnap_bam.txt")
```

### 4.3 Read and alignment stats

The following generates a summary table of the number of reads in each sample and how many of them aligned to the reference.

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

### 4.4 Create symbolic links for viewing BAM files in IGV

The `symLink2bam` function creates symbolic links to view the BAM alignment files in a genome browser such as IGV. The corresponding URLs are written to a file with a path specified under `urlfile`, here IGVurl.txt.

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

## 5 Variant calling

The following performs variant calling with `GATK`, `BCFtools` and *VariantTools* in parallel mode on a compute cluster (**??**). If a cluster is not available, the `runCommandline()` function can be used to run the variant calling with `GATK` and `BCFtools` for each sample sequentially on a single machine, or `callVariants` in case of *VariantTools*. Typically, the user would choose here only one variant caller rather than running several ones.

### 5.1 Variant calling with GATK

The following creates in the inital step a new targets file (`targets_bam.txt`). The first column of this file gives the paths to the BAM files created in the alignment step. The new targets file and the parameter file `gatk.param` are used to create a new `SYSargs` instance for running GATK. Since GATK involves many processing steps, it is executed by a bash script `gatk_run.sh` where the user can specify the detailed run parameters. All three files are expected to be located in the current working directory. Samples files for `gatk.param` and `gatk_run.sh` are available in the subdirectory `./inst/extdata/` of the source file of the *systemPipeR* package. Alternatively, they can be downloaded directly from here.

```
writeTargetsout(x=args, file="targets_bam.txt")
system("java -jar CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.dict")
# system("java -jar /opt/picard/1.81/CreateSequenceDictionary.jar R=./data/tair10.fasta O=./data/tair10.di
args <- systemArgs(sysma="gatk.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
writeTargetsout(x=args, file="targets_gatk.txt")
```

## 5.2 Variant calling with BCFtools

The following runs the variant calling with BCFtools. This step requires in the current working directory the parameter file `sambcf.param` and the bash script `sambcf_run.sh`.

```
args <- systemArgs(sysma="sambcf.param", mytargets="targets_bam.txt")
resources <- list(walltime="20:00:00", nodes=paste0("1:ppn=", 1), memory="10gb")
reg <- clusterRun(args, conffile=".BatchJobs.R", template="torque.tmpl", Njobs=18, runid="01",
                  resourceList=resources)
waitForJobs(reg)
writeTargetsout(x=args, file="targets_sambcf.txt")
```

## 5.3 Variant calling with VariantTools

```
library(gmapR); library(BiocParallel); library(BatchJobs)
args <- systemArgs(sysma="vartools.param", mytargets="targets_gsnap_bam.txt")
f <- function(x) {
    library(VariantTools); library(gmapR); library(systemPipeR)
    args <- systemArgs(sysma="vartools.param", mytargets="targets_gsnap_bam.txt")
    gmapGenome <- GmapGenome(systemPipeR::reference(args), directory="data", name="gmap_tair10chr", create
    tally.param <- TallyVariantsParam(gmapGenome, high_base_quality = 23L, indels = TRUE)
    bfl <- BamFileList(infile1(args)[x], index=character())
    var <- callVariants(bfl[[1]], tally.param)
    sampleNames(var) <- names(bfl)
    writeVcf(asVCF(var), outfile1(args)[x], index = TRUE)
}
funs <- makeClusterFunctionsTorque("torque.tmpl")
param <- BatchJobsParam(length(args), resources=list(walltime="20:00:00", nodes="1:ppn=1", memory="6gb"),
register(param)
d <- bplapply(seq(along=args), f)
writeTargetsout(x=args, file="targets_vartools.txt")
```

# 6 Filter variants

The function `filterVars` filters VCF files based on user definable quality parameters. It sequentially imports each VCF file into R, applies the filtering on an internally generated `VRanges` object and then writes the results to a new subsetted VCF file. The filter parameters are passed on to the corresponding argument as a character string. The function applies this filter to the internally generated `VRanges` object using the standard subsetting syntax for two dimensional objects such as: `vr[filter, ]`. The parameter files (`filter_gatk.param`, `filter_sambcf.param` and `filter_vartools.param`), used in the filtering steps, define the paths to the input and output VCF files which are stored in new `SYSargs` instances.

### 6.1 Filter variants called by GATK

The below example filters for variants that are supported by ≥x reads and ≥80% of them support the called variants. In addition, all variants need to pass ≥x of the soft filters recorded in the VCF files generated by GATK. Since the toy data used for this workflow is very small, the chosen settings are unreasonabley relaxed. A more reasonable filter setting is given in the line below (here commented out).

```
library(VariantAnnotation)
args <- systemArgs(sysma="filter_gatk.param", mytargets="targets_gatk.txt")
filter <- "totalDepth(vr) >= 2 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))>=1
# filter <- "totalDepth(vr) >= 20 & (altDepth(vr) / totalDepth(vr) >= 0.8) & rowSums(softFilterMatrix(vr))
filterVars(args, filter, varcaller="gatk", organism="A. thaliana")
writeTargetsout(x=args, file="targets_gatk_filtered.txt")
```

### 6.2 Filter variants called by BCFtools

The following shows how to filter the VCF files generated by *BCFtools* using similar parameter settings as in the previous filtering of the GATK results.

```
args <- systemArgs(sysma="filter_sambcf.param", mytargets="targets_sambcf.txt")
filter <- "rowSums(vr) >= 2 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
# filter <- "rowSums(vr) >= 20 & (rowSums(vr[,3:4])/rowSums(vr[,1:4]) >= 0.8)"
filterVars(args, filter, varcaller="bcftools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_sambcf_filtered.txt")
```

### 6.3 Filter variants called by VariantTools

The following shows how to filter the VCF files generated by *VariantTools* using similar parameter settings as in the previous filtering of the GATK results.

```
args <- systemArgs(sysma="filter_vartools.param", mytargets="targets_vartools.txt")
filter <- "(values(vr)$n.read.pos.ref + values(vr)$n.read.pos) >= 2 & (values(vr)$n.read.pos / (values(vr)
# filter <- "(values(vr)£n.read.pos.ref + values(vr)£n.read.pos) >= 20 & (values(vr)£n.read.pos / (values(
filterVars(args, filter, varcaller="vartools", organism="A. thaliana")
writeTargetsout(x=args, file="targets_vartools_filtered.txt")
```

## 7 Annotate filtered variants

The function `variantReport` generates a variant report using utilities provided by the *VariantAnnotation* package. The report for each sample is written to a tabular file containing genomic context annotations (*e.g.* coding or non-coding SNPs, amino acid changes, IDs of affected genes, etc.) along with confidence statistics for each variant. The parameter file `annotate_vars.param` defines the paths to the input and output files which are stored in a new `SYSargs` instance.

### 7.1 Annotate filtered variants called by GATK

```
library("GenomicFeatures")
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")
```

### 7.2 Annotate filtered variants called by BCFtools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")
```

### 7.3 Annotate filtered variants called by VariantTools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_vartools_filtered.txt")
txdb <- loadDb("./data/tair10.sqlite")
fa <- FaFile(systemPipeR::reference(args))
variantReport(args=args, txdb=txdb, fa=fa, organism="A. thaliana")
```

## 8 Combine annotation results among samples

To simplify comparisons among samples, the `combineVarReports` function combines all variant annotation reports referenced in a `SYSargs` instance (here `args`). At the same time the function allows to consider only certain feature types of interest. For instance, the below setting `filtercol=c(Consequence="nonsynonymous")` will include only nonsysynonymous variances listed in the `Consequence` column of the annotation reports. To omit filtering, one can use the setting `filtercol="All"`

### 8.1 Combine results from GATK

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_gatk.xls", quote=FALSE, row.names=FALSE, sep="\t")
```

### 8.2 Combine results from BCFtools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_sambcf.xls", quote=FALSE, row.names=FALSE, sep="\t")
```

### 8.3 Combine results from VariantTools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_vartools_filtered.txt")
combineDF <- combineVarReports(args, filtercol=c(Consequence="nonsynonymous"))
write.table(combineDF, "./results/combineDF_nonsyn_vartools.xls", quote=FALSE, row.names=FALSE, sep="\t")
```

## 9 Summary statistics of variants

The function `varSummar` counts the number of variants for each feature type included in the anntation reports.

### 9.1 Summary for GATK

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
write.table(varSummary(args), "./results/variantStats_gatk.xls", quote=FALSE, col.names = NA, sep="\t")
```

### 9.2 Summary for BCFtools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
write.table(varSummary(args), "./results/variantStats_sambcf.xls", quote=FALSE, col.names = NA, sep="\t")
```

### 9.3 Summary for VariantTools

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_vartools_filtered.txt")
write.table(varSummary(args), "./results/variantStats_vartools.xls", quote=FALSE, col.names = NA, sep="\t"
```

## 10 Venn diagram of variants

The venn diagram utilities defined by the *systemPipeR* package can be used to identify common and unique variants reported for different samples and/or variant callers. The below generates a 4-way venn diagram comparing four sampes for each of the two variant callers.

```
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_gatk_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID
vennset_gatk <- overLapper(varlist, type="vennsets")
args <- systemArgs(sysma="annotate_vars.param", mytargets="targets_sambcf_filtered.txt")
varlist <- sapply(names(outpaths(args))[1:4], function(x) as.character(read.delim(outpaths(args)[x])$VARID
vennset_bcf <- overLapper(varlist, type="vennsets")
pdf("./results/vennplot_var.pdf")
vennPlot(list(vennset_gatk, vennset_bcf), mymain="", mysub="GATK: red; BCFtools: blue", colmode=2, ccol=c(
dev.off()
```

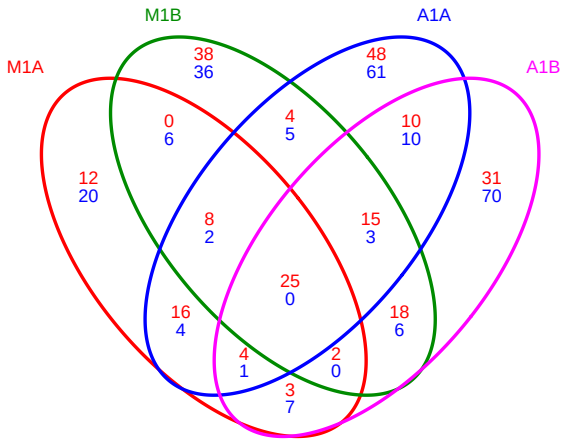


Figure 2: Venn Diagram for 4 samples from GATK and BCFtools.

## 11 Version Information

```
toLatex(sessionInfo())
```

- R version 3.3.1 (2016-06-21), `x86_64-apple-darwin13.4.0`
- Locale: `C/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8`
- Base packages: base, datasets, grDevices, graphics, methods, parallel, stats, stats4, utils
- Other packages: Biobase 2.34.0, BiocGenerics 0.20.0, BiocParallel 1.8.0, BiocStyle 2.2.0, Biostrings 2.42.0, DESeq2 1.14.0, GenomeInfoDb 1.10.0, GenomicAlignments 1.10.0, GenomicRanges 1.26.1, IRanges 2.8.0, Rsamtools 1.26.1, S4Vectors 0.12.0, ShortRead 1.32.0, SummarizedExperiment 1.4.0, XVector 0.14.0, ape 3.5, ggplot2 2.1.0, knitr 1.14, systemPipeR 1.8.1
- Loaded via a namespace (and not attached): AnnotationDbi 1.36.0, AnnotationForge 1.16.0, BBmisc 1.10, BatchJobs 1.6, Category 2.40.0, DBI 0.5-1, Formula 1.2-1, GO.db 3.4.0, GOstats 2.40.0, GSEABase 1.36.0, GenomicFeatures 1.26.0, Hmisc 3.17-4, Matrix 1.2-7.1, RBGL 1.50.0, RColorBrewer 1.1-2, RCurl 1.95-4.8, RSQLite 1.0.0, Rcpp 0.12.7, XML 3.98-1.4, acepack 1.4.0, annotate 1.52.0, assertthat 0.1, backports 1.0.3, base64enc 0.1-3, biomaRt 2.30.0, bitops 1.0-6, brew 1.0-6, checkmate 1.8.1, chron 2.3-47, cluster 2.0.5, codetools 0.2-15, colorspace 1.2-7, data.table 1.9.6, digest 0.6.10, edgeR 3.16.0, evaluate 0.10, fail 1.3, foreign 0.8-67, formatR 1.4, genefilter 1.56.0, geneplotter 1.52.0, graph 1.52.0, grid 3.3.1, gridExtra 2.2.1, gtable 0.2.0, highr 0.6, htmltools 0.3.5, hwriter 1.3.2, labeling 0.3, lattice 0.20-34, latticeExtra 0.6-28, limma 3.30.0, locfit 1.5-9.1, magrittr 1.5, munsell 0.4.3, nlme 3.1-128, nnet 7.3-12, pheatmap 1.0.8, plyr 1.8.4, rjson 0.2.15, rmarkdown 1.1, rpart 4.1-10, rtracklayer 1.34.0, scales 0.4.0, sendmailR 1.2-1, splines 3.3.1, stringi 1.1.2, stringr 1.1.0, survival 2.39-5, tibble 1.2, tools 3.3.1, xtable 1.8-2, yaml 2.1.13, zlibbioc 1.20.0

# 12 Funding

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