Mutation Load in Black Grouse

Rebecca S. Chen

Table of contents

1 Mutation load in black grouse

2 Introduction

This webpage/document contains a summary of the workflow used in the manuscript titled "Predicted deleterious mutations reveal the genomic mechanisms underlying fitness variation in a lekking bird", Chen et al. 2024 (in preparation). Please see the github repository for all scripts and more detailed descriptions of the data and analyses.

Figure 2.1: Black grouse

3 Main goals

In the current study, we used a long-term dataset to (i) quantify the fitness effects of homozygous and heterozygous individual genomic mutation loads; (ii) compare the fitness effects of deleterious mutations in coding versus noncoding regions; and (iii) unravel the behavioural and / or ornamental pathways through which deleterious mutations impact lifetime reproductive success. We used whole genome resequencing, phenotypic and fitness data of 190 male black grouse sampled annually across five study sites in Central Finland.

Mutation load can be defined as a statistic that summarizes the selection and dominance coefficients of deleterious mutations as a function of their frequencies in a population [1]. As we do not have selection and dominance coefficients of mutations in wild populations, we use a proxy for mutation load calculated as the number of deleterious mutations for a given individual.

There are different types of load, e.g. the realized load (expressed load) which reduces fitness in the current generation, and the potential/masked load (inbreeding load) which quantifies the potential fitness loss due to (partially) recessive deleterious mutations that may become expressed in future generations depending on the population's demography. The genetic load is made up of realized plus masked load.

4 Calculating genetic load

There are generally two most commonly used computational approaches to identify putative deleterious variants from whole genome re-sequencing data. In general, these tools attempt to predict the effect of a mutation on the function or evolutionary fitness of a protein. The two are distinct but can be related; for instance, a loss of function mutation will be strongly selected against if the gene is essential but will tend to be less evolutionary deleterious if the gene is non-essential or if the variant only slightly alters protein function. We used two common approaches:

- Genomic Evolutionary Rate Profiling (GERP): This approach uses multi-species genome alignments to identify genomic sites that are strongly conserved over millions of years of evolution, as non-synonymous mutations at these sites have a high likelihood of being deleterious. [3]
- SNP effect (SnpEff): This approach predicts the consequences of genomic variants on protein sequences and identifies loss of function and missense variants. [2].

5 This webpage / document

This webpage can also be found in PDF format on github. Note that in the PDF format, code is not folded which will end up in a lengthy document, and the html looks aesthetically better ;). In both documents, you will find some of the scripts for the analysis performed in this study. Note that not all bioinformatic steps are put on here (only from inferring mutations onwards). You can find the complete set of analyses with their explanations in the github repo.

6 SnpEff

6.1 Introduction

SnpEff annotates genetic variants and predicts the functional effects. The output includes a VCF file with annotations that indicate what kind of mutation it is (e.g. introduction of a stop codon) and the predicted effect (low, moderate, high, modifier). In this study, we focus on high impact mutations, which include loss of function (LoF) and nonsense mediate decay (NMD) mutations.

6.2 Methods

6.2.1 Building the database

As the black grouse (*Lyrurus tetrix*) is no common model species with a pre-built database, a custom database was built from the annotation files in .gff format provided by Cantata Bio.

To build a custom database, five files are required: the gff file containing the gene annotation, the reference genome, and then three files containing information about the coding regions (cds.fa; a fasta file containing the coding regions only), the genes (genes.fa; a fasta file containing the genes only) and a file with the protein sequences (proteins.fa; a fasta file with the protein sequences). Two softwares were used to construct these three fasta files: gff3_to_fasta and AGAT.

```
gff3_to_fasta -g
 ↪  data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.annotation.gff
 ↪  \
    -f
 ↪  data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.RepeatMasked.fasta
 ↪  -st cds -d complete -o data/genomic/refgenome/lyrurus_tetrix/cds.fa

gff3_to_fasta -g
 ↪  data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.annotation.gff
 ↪  \
 -f
 ↪   data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.RepeatMasked.fasta
 ↪   -st gene -d complete -o
 ↪   data/genomic/refgenome/lyrurus_tetrix/genes.fa
```

Similarly, the protein sequences were constructed with AGAT

```
agat_sp_extract_sequences.pl --gff
 ↪  data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.annotation.gff
 ↪  -f \

 ↪  data/genomic/refgenome/PO2979_Lyrurus_tetrix_black_grouse.RepeatMasked.fasta
 ↪  -p -o \
    data/genomic/refgenome/lyrurus_tetrix/protein.fa
```

Then, the database was built (and automatically checked).

```
java -jar snpEff.jar build -gff3 -v
 ↪  data/genomic/refgenome/lyrurus_tetrix
```

Once the database is ready, we can run SnpEff to create the annotated vcf file.

```
java -Xmx8g -jar snpEff.jar ann -stats  \
-no-downstream -no-intergenic -no-intron -no-upstream -no-utr -v \
lyrurus_tetrix data/genomic/intermediate/ltet_snps_filtered.vcf >
 ↪  data/genomic/intermediate/snpef/ltet_ann_snp_output.vcf
```

6.2.2 Ancestral alleles

SnpEff annotates mutations according to the change from the reference allele to the focal allele. Hence, it assumes that the reference allele is the ‘better’ one and that a mutation that changes the transcription of this reference allele is detrimental. To allow this assumption to be better met, we used the ancestral genome as a reference, instead of the reference genome itself (i.e. we polarized the genome). This ancestral genome is constructed by cactus, and represents the most recent common ancestor between black grouse *(L. tetrix)* and *Lagoplus leucura* (white tailed ptarmigan). This way, any derived allele was assumed to be ‘deleterious’ compared to the ancestral allele, as opposed to a reference-non reference comparison.

6.2.3 Filtering

We then used SnpSift to filter annotated mutations based on the four impact categories: modifier, low, moderate and high impact using the following commands.

```
## High impact
zcat output/ancestral/ltet_filtered_ann_aa.vcf.gz | java -jar
 ↪  src/SnpSift.jar filter " ( ANN[*].IMPACT = 'HIGH' )" >
 ↪  data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_HIGH.vcf
gzip data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_HIGH.vcf

## Moderate
zcat output/ancestral/ltet_filtered_ann_aa.vcf.gz | java -jar
 ↪  src/SnpSift.jar filter " ( ANN[*].IMPACT = 'MODERATE')" >
 ↪  data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_moderate.vcf
gzip data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_moderate.vcf

## Low
zcat output/ancestral/ltet_filtered_ann_aa.vcf.gz | java -jar
 ↪  src/SnpSift.jar filter " ( ANN[*].IMPACT = 'LOW') " >
 ↪  data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_low.vcf
gzip data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_low.vcf
```

6.3 Results

We identified 5,341 high impact mutations:

Existing of mostly LoF mutations and gained stop codons (non-mutually exclusive)

The mutations in the 'high impact' category were used to calculate individual genomic mutation load estimates.

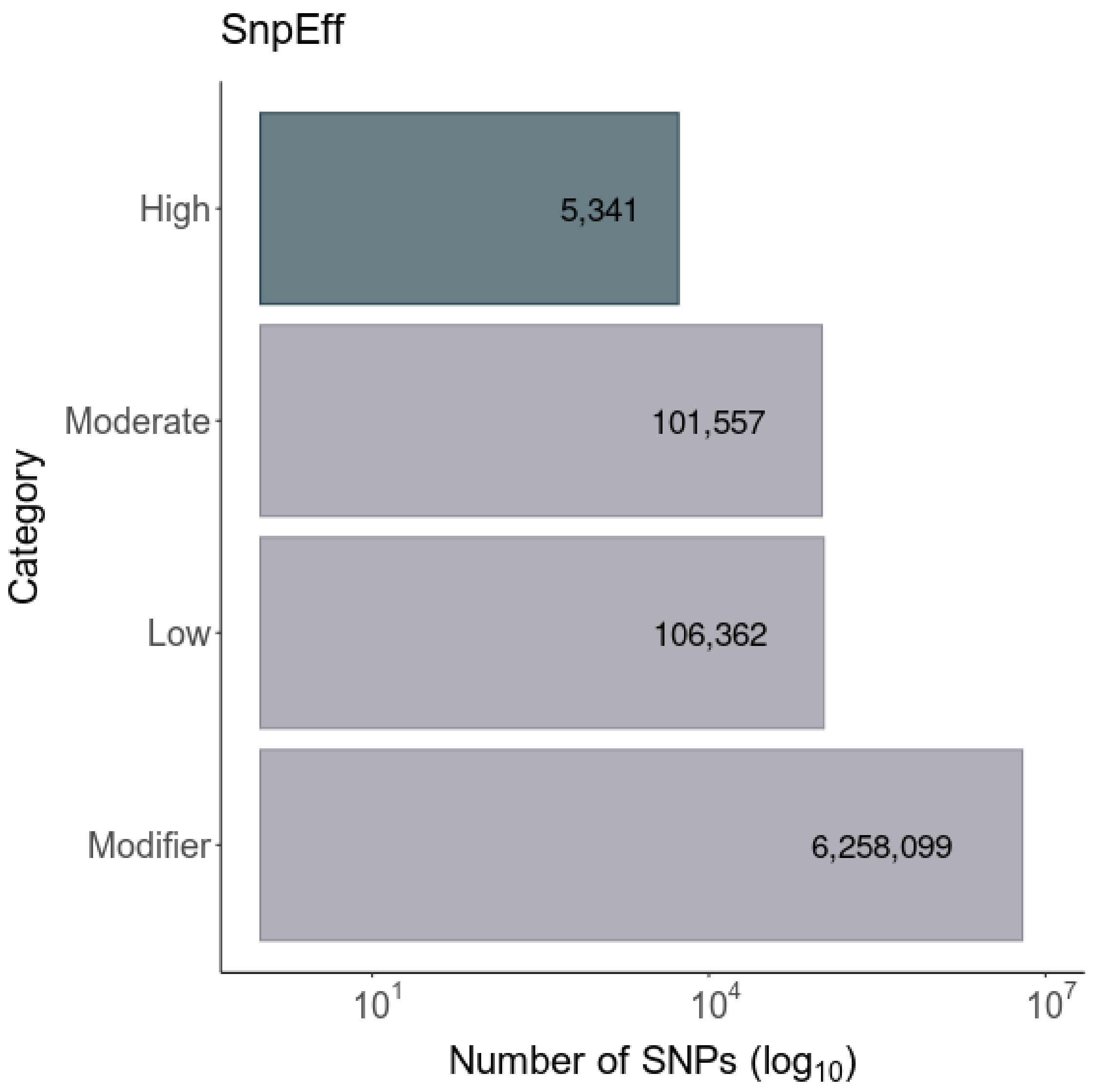


Figure 6.1: SnpEff annotation

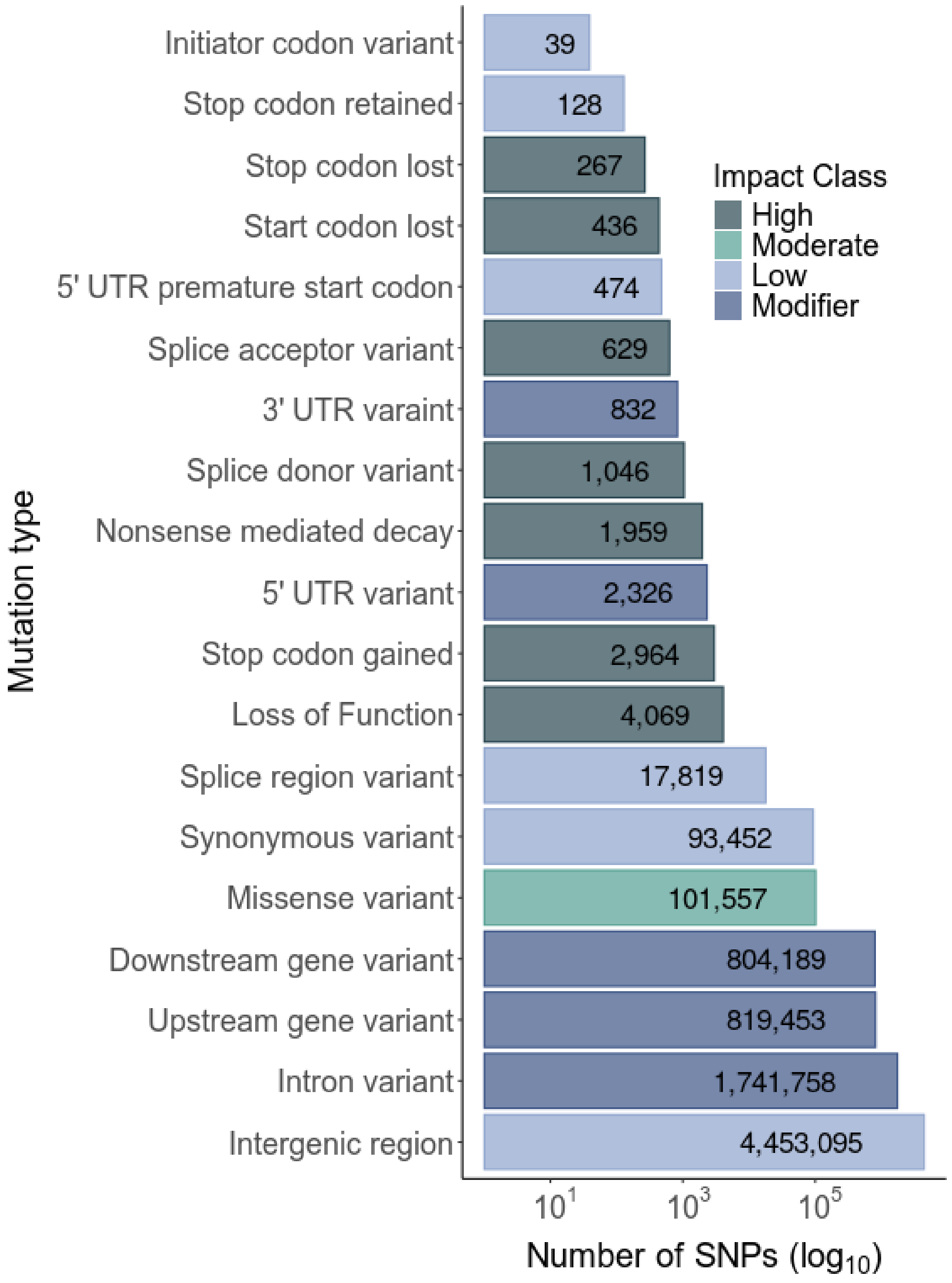


Figure 6.2: Detailed SnpEff annotation

7 GERP

7.1 Introduction

GERP++ annotates a focal genome based on evolutionary conserveration, where regions in the genome that show higher conservation across multiple different species are expected to face higher selective constraint. GERP score calculation, which indicate the reduction in the number of substitutions compared to neutral expectations, is done based on a multi-species alignment file. Higher GERP scores indicate higher evolutionary constraint. First, we create this MAF file using Progressive Cactus, then we calculate GERP scores genome-wide, and select genomic positions with SNPs in our population.

7.2 Methods

7.2.1 Creating the MAF

We use the publicly available 363 avian genomes multi-alignment file as a starting point, and then reduce this file to exclude species of the Neoaves clade. All the GERP analyses were done using the cactus container.

```
### Remove subtrees that are not needed for ltet analysis

#set cactus scratch directory
CACTUS_SCRATCH=$(pwd)/scratch/

# enter the container
apptainer shell --cleanenv \
  --fakeroot --overlay ${CACTUS_SCRATCH} \
  --bind ${CACTUS_SCRATCH}/tmp:/tmp,$(pwd) \
  --env PYTHONNOUSERSITE=1 \
  docker:quay.io/comparative-genomics-toolkit/cactus:v2.5.1

# get stats on the original 363-avian multi-alignment file
```

```
halStats data/genomic/intermediate/cactus/363-avian-2020.hal >
 ↪  output/cactus/stats_original_363_hal.txt

# copy the original file to then edit it
cp data/genomic/intermediate/cactus/363-avian-2020.hal
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal

# remove subtrees to exclude neoaves
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc1
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc57
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc69
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc318 #starts with Heliornis_fulica
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc319 #starts with Psophia_crepitans
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc320 #starts with Charadrius_vociferus
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc321 #starts with Opisthocomus_hoazin
halRemoveSubtree data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  birdAnc322 #stars with birdAnc57, so the big chunk of passerines but
 ↪  also a

#some individual ancestral genomes left to exclude
halRemoveGenome data/genomic/intermediate/cactus//363-avian-reduced.hal
 ↪  birdAnc322
halRemoveGenome data/genomic/intermediate/cactus//363-avian-reduced.hal
 ↪  birdAnc1

# get stats of our subset of genomes

halStats data/genomic/intermediate/cactus/363-avian-reduced.hal >
 ↪  output/cactus/stats_reduced_363_hal.txt

#extract the reduced file
halExtract data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal
```

Next, we add two genomes: *Lyrurus tetrix* and *Lagopus lecura*, using the `cactus prepare` function. To add the *Lagopus lecura* genome to the hal file, the assembly needs to be downloaded from NCBI which can be found on NCBI: https://0-www-ncbi-nlm-nih-gov.brum.beds.ac.uk/datasets/genome/GCF_019238085.1/.

```
# In this file, we will use the cactus-update-prepare function to create
↪  two scripts that will allow us to add two genomes to our dataset

# set scratch directory
CACTUS_SCRATCH=$(pwd)/scratch/

#enter container
apptainer shell --cleanenv \
  --fakeroot --overlay ${CACTUS_SCRATCH} \
  --bind ${CACTUS_SCRATCH}/tmp:/tmp,$(pwd) \
  --env PYTHONNOUSERSITE=1 \
  src/containers/cactus_v2.5.1.sif

sh

#first add Lyrurus tetrix, branchlengths will be corrected in a later
↪  step

cactus-update-prepare \
  add branch \
  --parentGenome birdAnc334 \
  --childGenome Tympanuchus_cupido \
  data/genomic/intermediate/cactus/363-avian-reduced.hal \
  scripts/2_cactus/input_ltet.txt \
  --cactus-prepare-options \
  '--alignCores 4' \
  --topBranchLength 0.01 \
  --outDir scratch/tmp/steps-output \
  --jobStore scratch/tmp/js \
  --ancestorName AncX >
   ↪  scripts/2_cactus/3_cactus_update_lyrurus_steps.sh

# then add Lagopus leucura

cactus-update-prepare \
  add branch \
  --parentGenome AncX \
  --childGenome Lyrurus_tetrix \
  data/genomic/intermediate/cactus/363-avian-reduced.hal \
  scripts/2_cactus/input_lleu.txt \
  --cactus-prepare-options \
  '--alignCores 4' \
  --topBranchLength 0.01 \
  --outDir scratch/tmp/steps-output \
  --jobStore scratch/tmp/js \
  --ancestorName AncY >
   ↪  scripts/2_cactus/4_cactus_update_lagopus_steps.sh
```

This is just the preparation step, and two files will be outputted that can be used to update the .hal file. This is what these files look like (and then they have to be executed).

```
## Preprocessor
cactus-preprocess scratch/tmp/js/0 scratch/tmp/steps-output/seq_file.in
 ↪  scratch/tmp/steps-output/seq_file.out --inputNames Lyrurus_tetrix
 ↪  --realTimeLogging --logInfo --retryCount 0 --maskMode none

## Alignment

### Round 0
cactus-blast scratch/tmp/js/1 scratch/tmp/steps-output/seq_file.out
 ↪  scratch/tmp/steps-output/AncX.cigar --root AncX --restart
cactus-align scratch/tmp/js/2 scratch/tmp/steps-output/seq_file.out
 ↪  scratch/tmp/steps-output/AncX.cigar
 ↪  scratch/tmp/steps-output/AncX.hal --root AncX  --maxCores 8
hal2fasta scratch/tmp/steps-output/AncX.hal AncX --hdf5InMemory >
 ↪  scratch/tmp/steps-output/AncX.fa

### Round 1
cactus-blast scratch/tmp/js/3 scratch/tmp/steps-output/seq_file.out
 ↪  scratch/tmp/steps-output/birdAnc334.cigar --root birdAnc334
 ↪  --includeRoot
cactus-align scratch/tmp/js/4 scratch/tmp/steps-output/seq_file.out
 ↪  scratch/tmp/steps-output/birdAnc334.cigar
 ↪  scratch/tmp/steps-output/birdAnc334.hal --root birdAnc334
 ↪  --maxCores 4 --includeRoot

## Alignment update
```

```
halAddToBranch data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  scratch/tmp/steps-output/AncX.hal
 ↪  scratch/tmp/steps-output/birdAnc334.hal birdAnc334 AncX
 ↪  Tympanuchus_cupido Lyrurus_tetrix 0.01 1.0 --hdf5InMemory

## Alignment validation
halValidate --genome birdAnc334
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  --hdf5InMemory
halValidate --genome AncX
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  --hdf5InMemory
halValidate --genome Tympanuchus_cupido
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  --hdf5InMemory
halValidate --genome Lyrurus_tetrix
 ↪  data/genomic/intermediate/cactus/363-avian-reduced.hal
 ↪  --hdf5InMemory
```

For subsequent steps, the resulting hal file is converted to maf format per scaffold for both GERP++ and the neutral tree calculation. Note that we only focus on the 30 largest scaffolds of the black grouse genome which over >95% of the genome, and only autosomal scaffolds.

This conversion is done within an R script

```
hal =  "data/genomic/intermediate/cactus/363-avian-reduced.hal"
library(dplyr); library(data.table)
scafs <- fread("data/genomic/refgenome/30_largest.scafs.tsv")
output_dir = "output/cactus/maf_per_scaf"
sif = "src/containers/cactus_v2.6.12.sif"
scratch = "scripts/2_cactus/scratch"
tmp_js = "scripts/2_cactus/scratch/tmp/js/wiggle"

hal_to_maf_per_scaf <- function(hal, scaf, outdir, scratch, i, sif,
 ↪  tmp_js){
    system(paste0('/vol/apptainer/bin/apptainer run --cleanenv
     ↪  --fakeroot --overlay ', scratch, ' --bind ', scratch,
     ↪  '/tmp:/tmp,', scratch, ' --env PYTHONNOUSERSITE=1 ', sif, '
     ↪  cactus-hal2maf ', tmp_js, '/js_', i, ' --restart ', hal, ' ',
     ↪  outdir, '/maf_', i, '.maf --refGenome Lyrurus_tetrix
     ↪  --refSequence ', scaf, ' --dupeMode single
     ↪  --filterGapCausingDupes --chunkSize 1000000 --noAncestors'))
}

for (i in 1:30){
  hal_to_maf_per_scaf(hal = hal,
  scaf = scafs$scaf[i],
  outdir = output_dir,
  scratch = scratch,
  sif = sif,
  i = i,
  tmp_js = tmp_js)
}
```

Lastly, the final phylogenetic tree has to be recalculated according to the updated tree, and the branch lengths have to be calculated in substitutions/site (rather than million years ago). This analysis can be found on github and will not be included here as it contains many small steps integrated with in a snakemake workflow.

7.2.2 Calculate GERP scores

GERP scores were calculated per scaffold using snakemake using the following rule:

```
rule call_gerp:
    input:
      maf = "output/cactus/maf_per_scaf/maf_{scaf}.maf",
      tree = "output/tree_cactus_updated.txt"
    output:
      rates = "output/gerp/maf_{scaf}.maf.rates"
    params:
      refname = "Lyrurus_tetrix"
    log: "logs/gerp_{scaf}.log"
    shell:
      """
```

```
        gerpcol -t {input.tree} -f {input.maf} -e {params.refname} -j -z
 ↪  -x ".rates" &> {log}
        """
```

7.2.3 Overlap GERP scores with SNPs

We are only interested in genomic locations where SNPs were found in our population. Therefore, we convert our VCF file and GERP files to bed format to overlap the SNPs with bedtools using the following commands (integrated in snakemake):

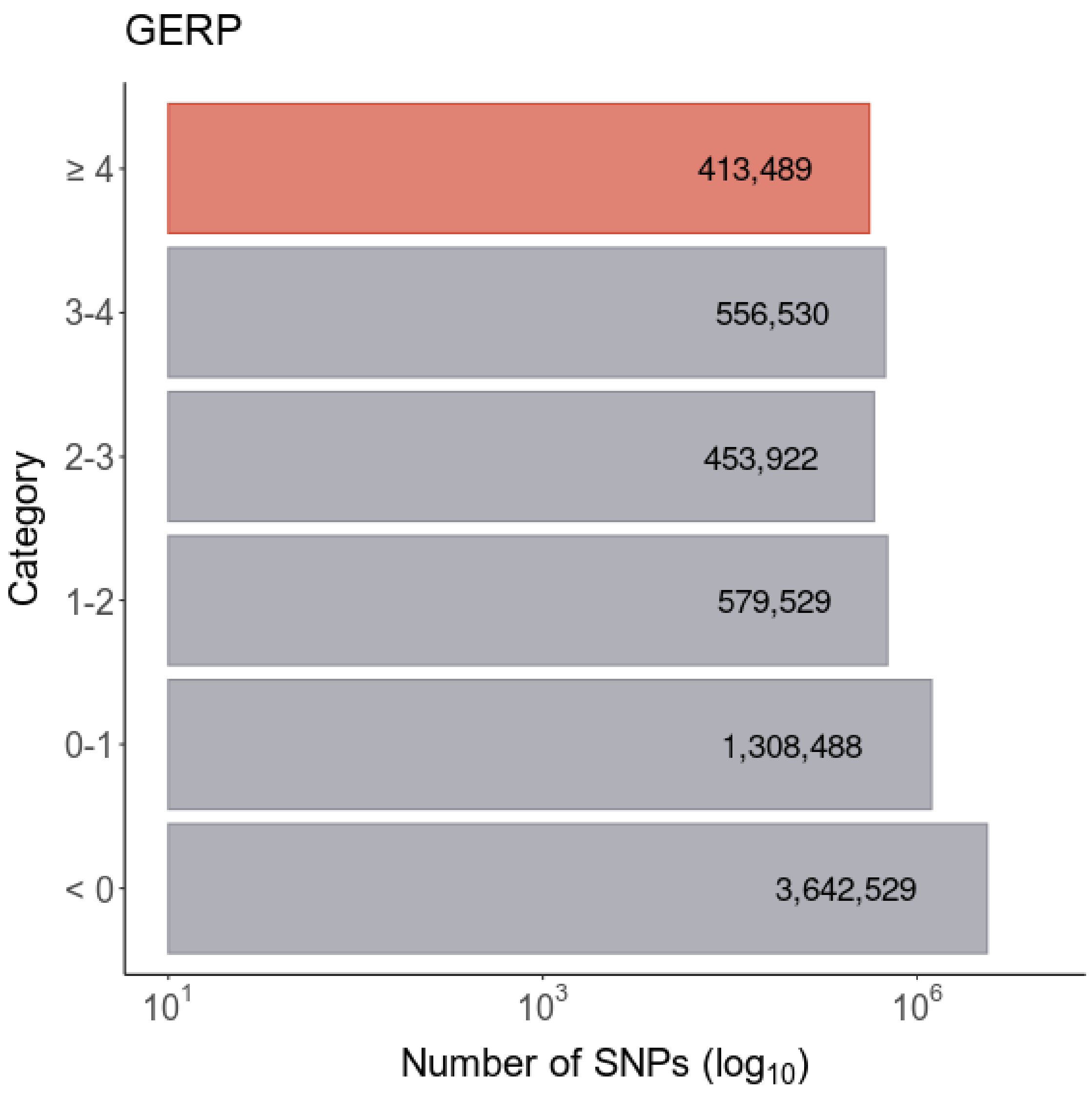
```
rule vcf_to_bed:
    input:
        vcf = "output/ancestral/ltet_filtered_ann_aa.vcf.gz"
    output:
        bed = "output/ancestral/ltet_filtered_ann_aa.bed"
    shell:
        """
        convert2bed -i vcf < {input.vcf} > {output.bed}

rule gerp_to_bed:
    input:
        rates =
 ↪  "output/gerp/maf_per_scaf/biggest_30/maf_{nscaf}.maf.rates"
    output:
        bed = "output/gerp/beds/gerp_scaf_{nscaf}.bed"
    params:
        outdir = "output/gerp/beds"
    log: "logs/gerp_to_bed_{nscaf}"
    shell:
        """
        Rscript --vanilla scripts/6_snpeff_gerp/2_gerp/gerp_to_bed.R
         ↪  {input.rates} {output.bed} {params.outdir} &> {log}
        """

rule bed_overlap_snps:
    input:
        bed = "output/gerp/beds/gerp_scaf_{nscaf}.bed",
        snps = "output/ancestral/ltet_filtered_ann_aa.bed"
    output:
        tsv = "output/gerp/beds/gerp_overlapSNP_scaf_{nscaf}.tsv.gz"
    params:
        tsv = "output/gerp/beds/gerp_overlapSNP_scaf_{nscaf}.tsv"
    shell:
        """
        bedtools intersect \
        -a {input.bed} \
        -b {input.snps} \
        -wa -wb |
        cut -f 6-10 --complement > {params.tsv}

        gzip {params.tsv}
        """
```

As these files are still very large, we loop over scaffolds within snakemake with an R script to count the number of mutations per individual per scaffold, both in homozygosity and heterozygosity using the following R formula (which is used for each scaffold separately and outputs a tsv file used for calculating mutation load in the next script).

```
calculate_gerp_load <- function(gerp_vcf, scafno){
  ## metadata on filenames and ids
  filenames <- fread("data/genomic/raw/metadata/idnames.fam")
  ids <- fread("data/genomic/raw/metadata/file_list_all_bgi_clean.csv")

  #merge
  idnames <- left_join(filenames[,c("V1")], ids[,c("loc", "id")], by =
 ↪  c("V1" = "loc"))

  file <- read_tsv(gerp_vcf, col_names = c("chr", "start", "pos",
 ↪  "neutral_rate_n", "rs_score", "ancestral", "derived", "qual",
 ↪  "info","format", idnames$id) )#rename columns

  # only get GT info, PL and DP are filtered by already anyway
  gt <- c(11:ncol(file))
  select_n3 <- function(x){x = substr(x,1,3)}
  file[gt] <- lapply(file[gt], select_n3)

  # replace genotype with RS value but separate per zygosity, do per ID
  gerp_load <- list()
  for( id in 11:ncol(file)){
    subset_id <- file[,c(1:10, id)]
    subset_id <- subset_id %>% mutate(gerp_cat = as.factor(case_when(
        rs_score < 0 ~ "< 0", #changed
        rs_score >= 0 & rs_score < 1 ~ "0-1",
        rs_score >= 1 & rs_score < 2 ~ "1-2",
        rs_score >= 2 & rs_score < 3 ~ "2-3",
        rs_score >= 3 & rs_score < 4 ~ "3-4",
        rs_score >= 4 ~ "4-5"
    )))
    gerp_load_id <- list()
    for (i in c("< 0", "0-1", "1-2", "2-3", "3-4", "4-5")){#changed
        cat_subset <- subset(subset_id, gerp_cat == i)
        het_data <- subset(cat_subset, cat_subset[[11]] == "1/0" |
↪   cat_subset[[11]] == "0/1")
        hom_data <- subset(cat_subset, cat_subset[[11]] == "1/1")
        n_genotyped <- nrow(cat_subset) - nrow(subset(cat_subset,
↪   cat_subset[[11]] == "./."))
        n_total <- nrow(cat_subset)
        df <- data.frame(id = colnames(file[id]),
                         gerp_cat = i,
                         scafno = scafno,
                         n_total = n_total,
                         n_genotyped = n_genotyped,
                         het_data = nrow(het_data),
                         hom_data = nrow(hom_data))

        gerp_load_id[[i]] <- df

        }
        gerp_load_id <- do.call(rbind.data.frame, gerp_load_id)
        rownames(gerp_load_id) <- NULL

    gerp_load[[id]] <- gerp_load_id
    }
  gerp_load <- do.call(rbind.data.frame, gerp_load)

    return(gerp_load)}
```

7.3 Results

We identified 413,489 mutations with a GERP score higher than or equal to 4:



These mutations were used to calculate mutation load in the next script.

8 Calculating mutation load

8.1 Introduction

There are various types of load and in general, mutation load can be divided between potential and realized load. Realized load (also known as expressed load) only includes the deleterious mutations that are expressed in the individual [1]. Potential load (also known as inbreeding or masked load) is the fitness reduction due to deleterious mutations, of which not all are expressed on an individual level and therefore quantifies recessive deleterious mutations that could be expressed in future generations [1].

However, to be able to distinguish between realized and potential load, you need to know dominance coefficients, which we do not. Therefore, we focus on homozygous and heterozygous load instead, which consists of mutations in homo- and heterozygosity in an individual instead. The total load sums the number of mutations contributing to the load, where heterozygous mutations are counted ones (one per allele) and homozygous mutations twice (one per allele).

From here on onwards, the majority of analyses are computed within R (instead of bash scripts/snakemake).

8.2 Mutation load (SnpEff)

Here, we load in the .vcf file outputted by SnpSift with only high impact SnpEff mutations, add column names, include only the 29 largest autosomal scaffolds, exclude warning messages, convert the genotype columns into only 1/1, 1/0, 0/1, 0/0 and ./. values, calculate load per individual, and then merge the load estimates of all individuals together.

```r
### load packages ###
pacman::p_load(dplyr, data.table)

### function to calculate load ###
calculate_load_snpeff <- function(vcf, output_vcf, loadtype){
  ## metadata on filenames and ids
  filenames <- fread("data/genomic/raw/metadata/idnames.fam")
  ids <- fread("data/genomic/raw/metadata/file_list_all_bgi_clean.csv")
```

```
#merge
idnames <- left_join(filenames[,c("V1")], ids[,c("loc", "id")], by =
↪  c("V1" = "loc"))

names(vcf) <- c(c("CHROM", "POS", "ID", "REF", "ALT", "QUAL",
 ↪  "FILTER", "INFO","FORMAT"), idnames$id)#rename columns

# only select 29 largest autosomal scaffolds
scaf <- fread("data/genomic/refgenome/30_largest.scafs.tsv")
scaf$scaf <- gsub(":", ";", scaf$scaf)
scaf$scaf <- gsub("\\.", "=", scaf$scaf)
scaf <- subset(scaf, scaf_no != 4)

vcf <- subset(vcf, CHROM %in% scaf$scaf)

# exclude warning messages

vcf <- subset(vcf, !grepl("WARNING", INFO))

# only get GT info, PL and DP are filtered by already anyway
gt <- c(10:ncol(vcf))
select_n3 <- function(x){x = substr(x,1,3)}
vcf[gt] <- lapply(vcf[gt], select_n3)

# calculate load
load <- list()
# loop over ids
for( id in 10:(ncol(vcf))){
  # subset per id
  subset_id <- vcf[,c(1:9, id)]

  # filter for snps in het and hom
  het_data <- subset(subset_id, subset_id[[10]] == "1/0" |
↪  subset_id[[10]] == "0/1")
  hom_data <- subset(subset_id, subset_id[[10]] == "1/1")

  # count amount of snps in het and hom
  het_load_sum <- nrow(het_data)
  hom_load_sum <- nrow(hom_data)
```

```
    # count no of snps successfully genotyped
    n_genotyped <- nrow(subset_id) - nrow(subset(subset_id,
 ↪  subset_id[[10]] == "./."))
    n_total <- nrow(subset_id)

    # collect data in df
    df <- data.frame(id = colnames(vcf[id]),
                     n_total = n_total,
                     n_genotyped = n_genotyped,
                     het_load = het_load_sum / n_genotyped,
                     hom_load = hom_load_sum / n_genotyped,
                     total_load = (het_load_sum*0.5 + hom_load_sum) /
                      ↪  n_genotyped,
                     loadtype = loadtype)
    load[[id]] <- df
  }
  # convert list to df
  load <- do.call(rbind.data.frame, load)

  if(output_vcf == TRUE){
    out <- list(load = load, vcf = vcf)
    return(out)
  }

  if(output_vcf==FALSE){
    return(load)}
}

##### load high impact mutations (filtered by snpsift) #####

high <-
 ↪  read.table("data/genomic/intermediate/snpef/ltet_ann_aa_snp_output_HIGH.vcf.gz")

## calculate load
# in this function, we give the columns names, filter for only the
 ↪  largest 29 autosomal scaffolds and exclude annotations with warning
 ↪  messages

high_load <- calculate_load_snpeff(high, output_vcf = TRUE, loadtype =
 ↪  "high")
```

8.3 Mutation load (GERP)

Here, we load in the .bed files that contain GERP scores from SNPs, filter for those with GERP values >= 4, add column names, convert the genotype columns into only 1/1, 1/0, 0/1, 0/0 and ./. values, calculate load per individual, and then merge the load estimates of all individuals together. Note this step is quite time-intensive as the .bed files are large in filesize!

```
# load in all bed files with gerp scores that overlap a SNP
gerp_snp_scafs <- list.files(path = "output/gerp/beds", pattern =
 ↪  "gerp_overlapSNP*", full.names = T)
gerp_snp_scafs <- gerp_snp_scafs[-22] #empty, scaffold 29 has no SNPs
 ↪  with gerp scores

gerp_snp <- data.frame()
for (i in 1:length(gerp_snp_scafs)){
  scaf <- read.table(gerp_snp_scafs[i])
  scaf <- scaf %>% filter(V5 >= 4)
  gerp_snp <- rbind(gerp_snp, scaf)
}

## function to calculate load

calculate_load_gerp <- function(vcf, output_vcf, loadtype){

  ## metadata on filenames and ids
  filenames <- fread("data/genomic/raw/metadata/idnames.fam")
  ids <- fread("data/genomic/raw/metadata/file_list_all_bgi_clean.csv")

  #merge
  idnames <- left_join(filenames[,c("V1")], ids[,c("loc", "id")], by =
 ↪  c("V1" = "loc"))

  names(vcf) <- c("chr", "start", "pos", "neutral_rate_n", "rs_score",
   ↪  "ancestral", "derived", "qual", "info","format", idnames$id)
   ↪  #rename columns

  # only get GT info, PL and DP are filtered by already anyway
  gt <- c(11:ncol(vcf))
  select_n3 <- function(x){x = substr(x,1,3)}
  vcf[gt] <- lapply(vcf[gt], select_n3)
```

```r
# calculate load
load <- list()
# loop over ids
for( id in 11:(ncol(vcf))){
  # subset per id
  subset_id <- vcf[,c(1:10, id)]

  # filter for snps in het and hom
  het_data <- subset(subset_id, subset_id[[11]] == "1/0" |
↪ subset_id[[11]] == "0/1")
  hom_data <- subset(subset_id, subset_id[[11]] == "1/1")

  # count amount of snps in het and hom
  het_load_sum <- nrow(het_data)
  hom_load_sum <- nrow(hom_data)

  # count no of snps successfully genotyped
  n_genotyped <- nrow(subset_id) - nrow(subset(subset_id,
↪ subset_id[[11]] == "./."))
  n_total <- nrow(subset_id)

  # collect data in df
  df <- data.frame(id = colnames(vcf[id]),
                   n_total = n_total,
                   n_genotyped = n_genotyped,
                   het_load = het_load_sum / n_genotyped,
                   hom_load = hom_load_sum / n_genotyped,
                   total_load = (het_load_sum*0.5 + hom_load_sum) /
                    ↪ n_genotyped,
                   loadtype = loadtype)
  load[[id]] <- df
}
# convert list to df
load <- do.call(rbind.data.frame, load)

if(output_vcf == TRUE){
  out <- list(load = load, vcf = vcf)
  return(out)
}
```

```
  if(output_vcf==FALSE){
  return(load)}
}

## calculate load
gerp_45 <- calculate_load_gerp(gerp_snp, output_vcf = TRUE, loadtype =
 ↪  "gerp45")
gerp <- gerp_45_load_check$vcf
```

8.3.1 Combine loads

Note: the analyses done above was also done for other mutation categories, e.g. low and moderate impact classes and GERP scores between 3-4. All load scores are then combined into a single file:

```
load <- rbind(high_load$load[,c("id", "het_load", "hom_load",
 ↪  "total_load", "loadtype")] ,
              moderate_load[,c("id", "het_load", "hom_load",
 ↪  "total_load", "loadtype")],
              low_load[,c("id", "het_load", "hom_load", "total_load",
 ↪  "loadtype")],
              lof_load[,c("id", "het_load", "hom_load", "total_load",
 ↪  "loadtype")],
              missense_load[,c("id", "het_load", "hom_load",
 ↪  "total_load", "loadtype")],
              gerp_34_load[,c("id", "het_load", "hom_load",
 ↪  "total_load", "loadtype")],
              gerp_45_load[,c("id", "het_load", "hom_load",
 ↪  "total_load", "loadtype")])

save(load, file =
 ↪  "output/load/all_loads_combined_da_nosex_29scaf.RData")
write.table(load, file =
 ↪  "output/load/all_loads_combined_da_nosex_29scaf.tsv", sep="\t",
 ↪  row.names = F)
```

We can then calculate the correlation between the two load estimates and test for lek effects on load.

```r
library(dplyr)
load(file = "../output/load/all_loads_combined_da_nosex_29scaf.RData")

cor.test(load$total_load[which(load$loadtype == "gerp45")],
 ↪  load$total_load[which(load$loadtype == "high")])
```

```
	Pearson's product-moment correlation

data:  load$total_load[which(load$loadtype == "gerp45")] and load$total_load[which(load$load
t = 1.7867, df = 188, p-value = 0.07559
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.01338135  0.26666622
sample estimates:
      cor
0.1292181
```

```r
### Test for lek effects ####
load("../data/phenotypes/phenotypes_lifetime.RData")
pheno_load <- left_join(pheno_wide, load, by = "id")

summary(lm(total_load ~ site, data = subset(pheno_load, loadtype ==
 ↪  "gerp45")))
```

```
Call:
lm(formula = total_load ~ site, data = subset(pheno_load, loadtype ==
    "gerp45"))

Residuals:
       Min         1Q     Median         3Q        Max
-2.426e-03 -2.162e-04  4.882e-05  3.303e-04  1.308e-03

Coefficients:
              Estimate Std. Error  t value Pr(>|t|)
(Intercept)  1.519e-01  6.684e-05 2272.733   <2e-16 ***
siteLEH     -8.574e-05  1.171e-04   -0.732    0.465
siteNYR     -5.353e-05  9.711e-05   -0.551    0.582
siteSAA     -5.700e-05  1.200e-04   -0.475    0.635
```

```
siteTEE      4.854e-05  1.337e-04    0.363    0.717
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.0005177 on 185 degrees of freedom
Multiple R-squared:  0.006348,  Adjusted R-squared:  -0.01514
F-statistic: 0.2955 on 4 and 185 DF,  p-value: 0.8807
```

```
summary(lm(total_load ~ site, data = subset(pheno_load, loadtype ==
↪  "high")))
```

```
Call:
lm(formula = total_load ~ site, data = subset(pheno_load, loadtype ==
    "high"))

Residuals:
       Min         1Q     Median         3Q        Max
-0.0098650 -0.0020586  0.0003317  0.0020222  0.0083354

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.1653480  0.0003800 435.179   <2e-16 ***
siteLEH     -0.0001091  0.0006656  -0.164    0.870
siteNYR     -0.0001785  0.0005521  -0.323    0.747
siteSAA     -0.0009170  0.0006820  -1.344    0.180
siteTEE     -0.0005430  0.0007599  -0.715    0.476
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.002943 on 185 degrees of freedom
Multiple R-squared:  0.01131,   Adjusted R-squared:  -0.01007
F-statistic: 0.5289 on 4 and 185 DF,  p-value: 0.7146
```

9 Modelling fitness

9.1 Introduction

Now that we have mutation load (total, homozygous and heterozygous) estimates for each individual, based on SnpEff and GERP, we can model their effects on lifetime mating success (LMS).

Here, we build three sets of models:

1. The effect of total load on LMS
2. The effects of homo- and heterozygous load on LMS
3. The direct and indirect effects of total load on mating success through the sexual traits

We use Bayesian GLMMs using the R package 'brms' to compute these models

9.2 Methods

9.2.1 Total load

The general structure of the total load models is as follows:

`LMS ~ scale(total_load) + core + (1|site)`

This is what the script for each load type looks like:

```r
# load pheno data lifetime
load(file = "output/load/pheno_loads_lifetime.RData")

# load mutation load measures
load("output/load/all_loads_combined_da_nosex_29scaf_plus_per_region.RData")
 ↪  #loads no sex chr only 30 scaf

# subset only the relevant method/loadtype
load <- load_per_region %>% filter(loadtype == method)
```

```
# combine
pheno_wide_load <- left_join(pheno_wide, load, by = "id")
pheno_wide_load <- subset(pheno_wide_load, !is.na(total_load)) #some ids
 ↪  without genotypes, excluded for wgr

#### model ####
brm_load_t <- brm(LMS_min ~ scale(total_load) + core + (1|site), data =
 ↪  pheno_wide_load,
                  family = "zero_inflated_poisson",
                  prior = prior(normal(0,1), class = b),
                  cores =8, control = list(adapt_delta = 0.99,
                   ↪  max_treedepth = 15),
                  iter = iter, thin = thin, warmup = warm, seed = 1908)

save(brm_load_t, file = out)
```

We can check out the performance of each model using the following loop:

```
output_total <- list.files(path = "output/models/total_hom_het/",
                        pattern = "lms*", full.names=T)

diagnose_summary <- list()
for (i in 1:length(output)){
  #load fit
  load(file = output[[i]])
  #get posteriors
  posterior <- as.array(fit)
  log_ps <- log_posterior(fit)
  nuts <- nuts_params(fit) #divergence
  #get only beta and sd
  betas <- variables(fit)[grep("b_", variables(fit))]
  sd <- variables(fit)[grep("sd_", variables(fit))]

  #global patterns in divergence
  diverge_beta <- mcmc_parcoord(posterior, np = nuts, pars= betas)
  diverge_sd <- mcmc_parcoord(posterior, np = nuts, pars= sd)

  #identify collinearity between parameters
  collin_beta <- mcmc_pairs(posterior, np = nuts, pars= betas)
  collin_sd <- mcmc_pairs(posterior, np = nuts, pars= sd)
```

```
    #traceplot
    trace_beta <- mcmc_trace(posterior, pars = betas, np = nuts)
    trace_sd <- mcmc_trace(posterior, pars = sd, np = nuts)

    #rhat
    rhat <- mcmc_rhat(brms::rhat(fit))

    #effective sample size
    neff <- mcmc_neff(neff_ratio(fit))

    #autocorrelation
    autocor_beta <- mcmc_acf(posterior, pars = betas)
    autocor_sd <- mcmc_acf(posterior, pars=sd)

    #quick glance results
    areas <- mcmc_areas(fit, pars=betas)

    #combine in list
    diagnosis <- list(diverge_beta = diverge_beta,
                      diverge_sd = diverge_sd,
                      collin_beta = collin_beta,
                      collin_sd = collin_sd,
                      trace_beta = trace_beta,
                      trace_sd = trace_sd,
                      rhat = rhat,
                      neff = neff,
                      autocor_beta = autocor_beta,
                      autocor_sd = autocor_sd,
                      areas = areas)


    modelname <- sub(".*/", "", output[i])
    modelname <- sub(".RData", "", modelname)

    # add to summary
    diagnose_summary[[modelname]] <- diagnosis
  }
```

This is what these plots look like for total GERP load effects:

Diagnostics GERP total load model

9.2.2 Hom and het load

The general structure of the homozygous and heterozygous load models is as follows:

`LMS ~ scale(het_load) + scale(hom_load) + core + (1|site)`

This is what the script for each load type looks like:

```
# load pheno data lifetime
load(file = "output/load/pheno_loads_lifetime.RData")

# load mutation load measures
load("output/load/all_loads_combined_da_nosex_29scaf_plus_per_region.RData")
 ↪  #loads no sex chr only 30 scaf

# subset only the relevant method/loadtype
load <- load_per_region %>% filter(loadtype == method)

# combine
pheno_wide_load <- left_join(pheno_wide, load, by = "id")
pheno_wide_load <- subset(pheno_wide_load, !is.na(total_load)) #some ids
 ↪  without genotypes, excluded for wgr

#### model ####
brm_load_het_hom <- brm(LMS_min ~ scale(het_load) + scale(hom_load) +
 ↪  core + (1|site), data = pheno_wide_load,
                  family = "zero_inflated_poisson",
                  prior = prior(normal(0,1), class = b),
                  cores =8, control = list(adapt_delta = 0.99,
                   ↪  max_treedepth = 15),
                  iter = iter, thin = thin, warmup = warm, seed = 1908)

save(brm_load_het_hom, file = out)
```

The same diagnostics were applied to each model.

9.2.3 Direct and indirect effects

Next, we build models that are based on annual values. There are two sets of models: the first quantifies the effect of load on the six sexual traits (attendance, fighting rate, centrality, lyre size, blue chroma and red eye comb size) in six separate models. The second set analyses the effect of the six traits and load on annual mating success (MS).

The direct effect is the effect of load on MS while correcting for all mediators. The indirect effect is calculated using a mediation analysis, where this effect is calculated as the product of the effect of the predictor (the total load) on the mediator (the sexual trait) and the effect of the mediator on the response variable (AMS).

9.2.3.1 Set 1: load on traits

For each trait, we run this model:

`scale(trait) ~ scale(total_load) + age_cat + (1|year) + (1|site/id)`

```
### load data ###

load(file = "data/phenotypes/phenotypes_annual.RData")

# load mutation load measures
load("output/load/all_loads_combined_da_nosex_29scaf_plus_per_region.RData")
↪  #loads no sex chr only 30 scaf

# subset only the relevant method/loadtype
load <- load_per_region %>% filter(loadtype == method)

# merge files
pheno <- left_join(pheno_long, load, by = "id")
pheno$born <- pheno$year - pheno$age
pheno <- pheno %>% mutate(age_cat = as.factor(case_when(age == 1 ~
↪  "yearling", age > 1  ~ "adult")))

### modelling ####
formula <- formula(paste0("scale(", response, ") ~ scale(total_load) +
↪  age_cat + (1|year) + (1|site/id)"))

fit <- brm(formula,
             family = "gaussian",
            data = pheno,
            cores =8,
            control = list(adapt_delta = 0.99, max_treedepth = 15),
            prior = prior(normal(0,1), class = b),
            iter = iterations,
            thin = thin, warmup = burn)
```

```
save(fit, file = out)
```

9.2.3.2 Set 2: trait + load on MS

Then, for both approaches we run the following model:

`MS ~ scale(total_load) + scale(lyre) + scale(eyec) + scale(blue) + scale(dist) + scale(attend) + scale(fight) + age_cat + (1|year) + (1|site/id)`

```
### load data ###

load(file = "data/phenotypes/phenotypes_annual.RData")

# load mutation load measures
load("output/load/all_loads_combined_da_nosex_29scaf_plus_per_region.RData")
 ↪  #loads no sex chr only 30 scaf

# subset only the relevant method/loadtype
load <- load_per_region %>% filter(loadtype == method)

# merge files
pheno <- left_join(pheno_long, load, by = "id")
pheno$born <- pheno$year - pheno$age

pheno <- pheno %>% mutate(age_cat = as.factor(case_when(age == 1 ~
 ↪  "yearling", age > 1  ~ "adult")))

### modelling ####
formula <- formula("MS ~ scale(total_load) + scale(lyre) + scale(eyec) +
 ↪  scale(blue) + scale(dist) + scale(attend) + scale(fight) + age_cat +
 ↪  (1|year) + (1|site/id)")

fit <- brm(formula,
           family = "zero_inflated_poisson",
           data = pheno,
           cores =8,
           control = list(adapt_delta = 0.99, max_treedepth = 15),
           prior = prior(normal(0,1), class = b),
           iter = iterations,
           thin = thin, warmup = burn)
```

```r
save(fit, file = out)
```

9.2.3.3 Direct / indirect effects

The direct/indirect effects are then calculated after loading in all model outputs:

```r
### load packages ####

pacman::p_load(brms, bayesplot, dplyr, data.table)

### load models ###

#### gerp ####
load(file = "output/models/annual/traits/model_attend_gerp45.RData")
fit_gerp_attend <- fit
load(file = "output/models/annual/traits/model_fight_gerp45.RData")
fit_gerp_fight <- fit
load(file = "output/models/annual/traits/model_dist_gerp45.RData")
fit_gerp_dist <- fit
load(file = "output/models/annual/traits/model_eyec_gerp45.RData")
fit_gerp_eyec <- fit
load(file = "output/models/annual/traits/model_blue_gerp45.RData")
fit_gerp_blue <- fit
load(file = "output/models/annual/traits/model_lyre_gerp45.RData")
fit_gerp_lyre <- fit

load(file = "output/models/annual/ams/model_trait_ams_gerp45.RData")
fit_gerp_ams <- fit

rm(fit)

### snpeff
load(file = "output/models/annual/traits/model_attend_high.RData")
fit_high_attend <- fit
load(file = "output/models/annual/traits/model_fight_high.RData")
fit_high_fight <- fit
load(file = "output/models/annual/traits/model_dist_high.RData")
fit_high_dist <- fit
load(file = "output/models/annual/traits/model_eyec_high.RData")
```

```r
fit_high_eyec <- fit
load(file = "output/models/annual/traits/model_blue_high.RData")
fit_high_blue <- fit
load(file = "output/models/annual/traits/model_lyre_high.RData")
fit_high_lyre <- fit

load(file = "output/models/annual/ams/model_trait_ams_high.RData")
fit_high_ams <- fit

rm(fit)

### indirect effects loop #####
get_indirect <- function(mediator, method, trait_model, ams_model){
  treatment = "b_scaletotal_load"
  path1 <- as_draws_df(trait_model, variable =treatment)
  path1 <- path1$b_scaletotal_load

  path2 <- as_draws_df(ams_model, variable = mediator)
  path2 <- unlist(c(path2[,1]))

  indirect <- path1*path2

  direct <- as_draws_df(ams_model, variable =treatment)
  direct <- direct$b_scaletotal_load

  total <- indirect + direct

  effect_attend <- data.frame(treatment = treatment,
                              mediator = mediator,
                              method = method,
                              indirect_median = round(median(indirect),
                               ↪  2),
                              indirect_lower = round(quantile(indirect,
                               ↪  probs = c(.025)), 2),
                              indirect_upper = round(quantile(indirect,
                               ↪  probs = c(.975)), 2),
                              direct_median = round(median(direct), 2),
                              direct_lower = round(quantile(direct,
                               ↪  probs = c(.025)), 2),
                              direct_upper = round(quantile(direct,
                               ↪  probs = c(.975)), 2),
                              total_median = round(median(total), 2),
                              total_lower = round(quantile(total, probs
                               ↪   = c(.025)), 2),
                              total_upper = round(quantile(total, probs
                               ↪   = c(.975)), 2),
                              path1_median = round(median(path1), 2),
                              path1_lower = round(quantile(path1, probs
                               ↪   = c(.025)), 2),
                              path1_upper = round(quantile(path1, probs
                               ↪   = c(.975)), 2),
                              path2_median = round(median(path2), 2),
                              path2_lower = round(quantile(path2, probs
                               ↪   = c(.025)), 2),
                              path2_upper = round(quantile(path2, probs
                               ↪   = c(.975)), 2),
                              indirect_lower_80 =
                                  ↪   round(quantile(indirect, probs =
                                  ↪   c(.1)), 2),
                              indirect_upper_80 =
                                  ↪   round(quantile(indirect, probs =
                                  ↪   c(.9)), 2),
                              direct_lower_80 = round(quantile(direct,
                               ↪   probs = c(.1)), 2),
                              direct_upper_80 = round(quantile(direct,
                               ↪   probs = c(.9)), 2))

  return(effect_attend)
}

effects <- data.frame(rbind(get_indirect(mediator="b_scalelyre", method
 ↪   = "gerp45",
                                         trait_model=fit_gerp_lyre,
                                          ↪   ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scaleeyec",  method
                             ↪   = "gerp45",
                                         trait_model=fit_gerp_eyec,
                                          ↪   ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scaleblue",  method
                             ↪   = "gerp45",
                                         trait_model=fit_gerp_blue,
                                          ↪   ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scaleattend",
                             ↪  method = "gerp45",
                                         trait_model=fit_gerp_attend,
                                          ↪  ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scalefight",
                             ↪  method = "gerp45",
                                         trait_model=fit_gerp_fight,
                                          ↪  ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scaledist",  method
                             ↪  = "gerp45",
                                         trait_model=fit_gerp_dist,
                                          ↪  ams_model = fit_gerp_ams),
                            get_indirect(mediator="b_scalelyre", method
                             ↪  = "high",
                                         trait_model=fit_high_lyre,
                                          ↪  ams_model = fit_high_ams),
                            get_indirect(mediator="b_scaleeyec",  method
                             ↪  = "high",
                                         trait_model=fit_high_eyec,
                                          ↪  ams_model = fit_high_ams),
                            get_indirect(mediator="b_scaleblue",  method
                             ↪  = "high",
                                         trait_model=fit_high_blue,
                                          ↪  ams_model = fit_high_ams),
                            get_indirect(mediator="b_scaleattend",
                             ↪  method = "high",
                                         trait_model=fit_high_attend,
                                          ↪  ams_model = fit_high_ams),
                            get_indirect(mediator="b_scalefight",
                             ↪  method = "high",
                                         trait_model=fit_high_fight,
                                          ↪  ams_model = fit_high_ams),
                            get_indirect(mediator="b_scaledist",  method
                             ↪  = "high",
                                         trait_model=fit_high_dist,
                                          ↪  ams_model = fit_high_ams)))


write.csv(effects, file =
    ↪  "output/models/annual/direct_indirect_summary.csv", quote=F,
    ↪  row.names = F)
```

9.3 Results

We find significant effects of total GERP and total SnpEff load:

```r
library(readxl); library(dplyr); library(kableExtra)
totals <- read.csv("../output/models/intervals/total_gerp45_high.csv")
totals %>% kbl()
```

parameter	outer_width	inner_width	point_est	ll	l	m	h	hh	model
b_scaletotal_load	0.95	0.8	median	-0.27	-0.25	-0.21	-0.17	-0.14	GERP
b_scaletotal_load	0.95	0.8	median	-0.18	-0.16	-0.11	-0.06	-0.04	SnpEff

We also find significant effects of both hom and het GERP and SnpEff load:

```r
homhet <- read.csv("../output/models/intervals/hom_het_gerp45_high.csv")
homhet %>% kbl()
```

parameter	outer_width	inner_width	point_est	ll	l	m	h	hh	model	l
b_scalehom_load	0.95	0.8	median	-0.76	-0.70	-0.57	-0.45	-0.39	Hom	G
b_scalehet_load	0.95	0.8	median	-0.78	-0.72	-0.60	-0.48	-0.41	Het	G
b_scalehom_load	0.95	0.8	median	-0.17	-0.15	-0.09	-0.04	-0.01	Hom	S
b_scalehet_load	0.95	0.8	median	-0.24	-0.21	-0.15	-0.09	-0.06	Het	S

Here you can find the posterior distributions of model set 1 (load on traits) for GERP and

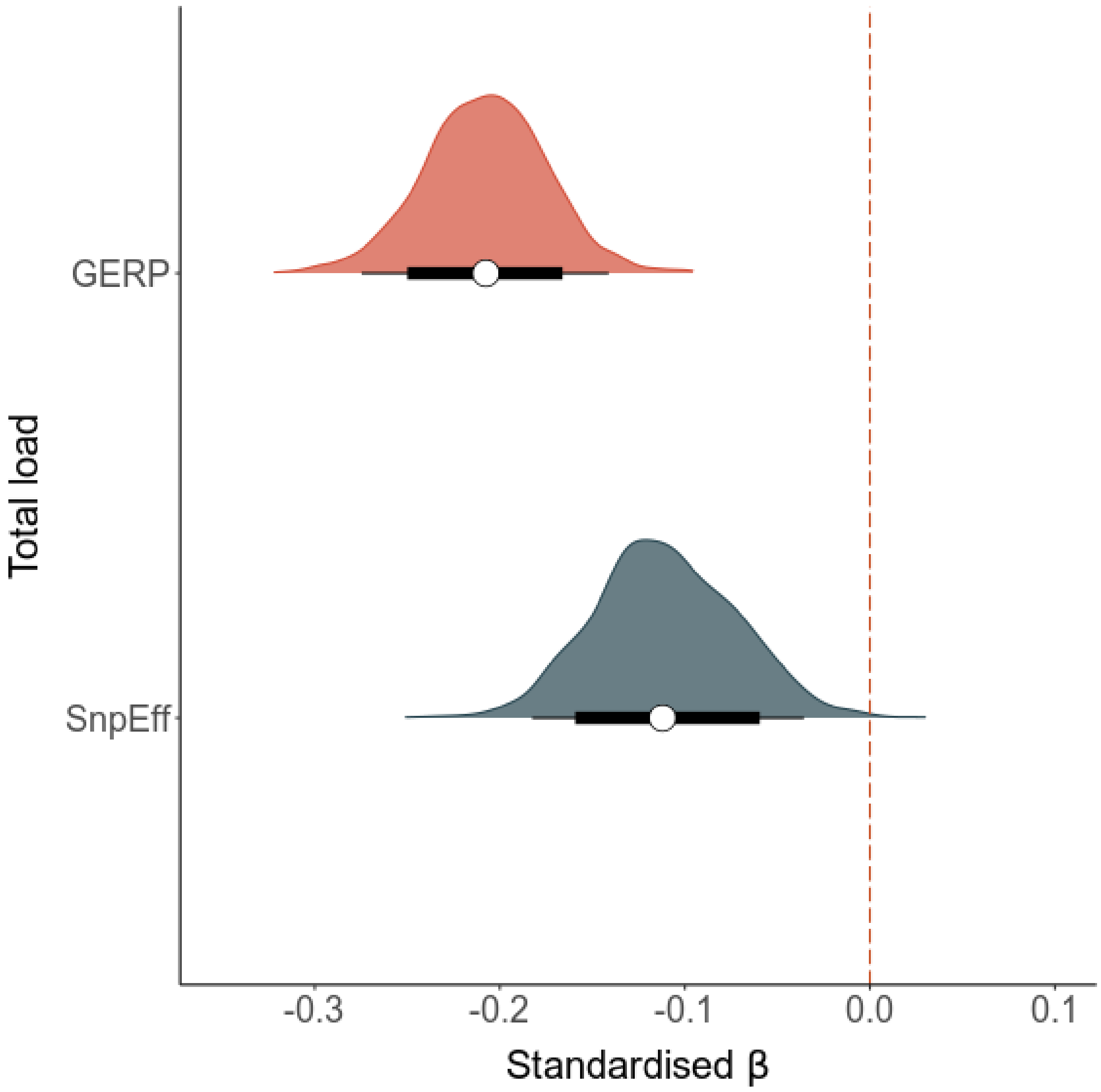


Figure 9.1: Total load results

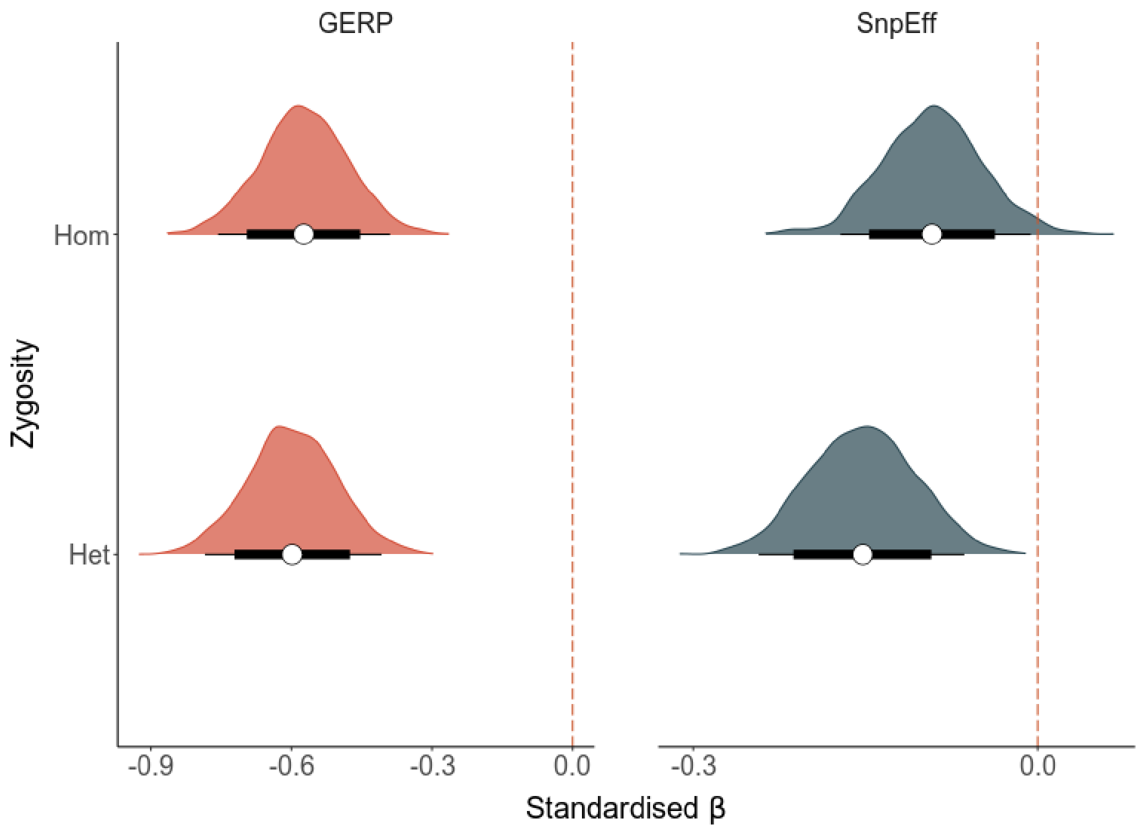
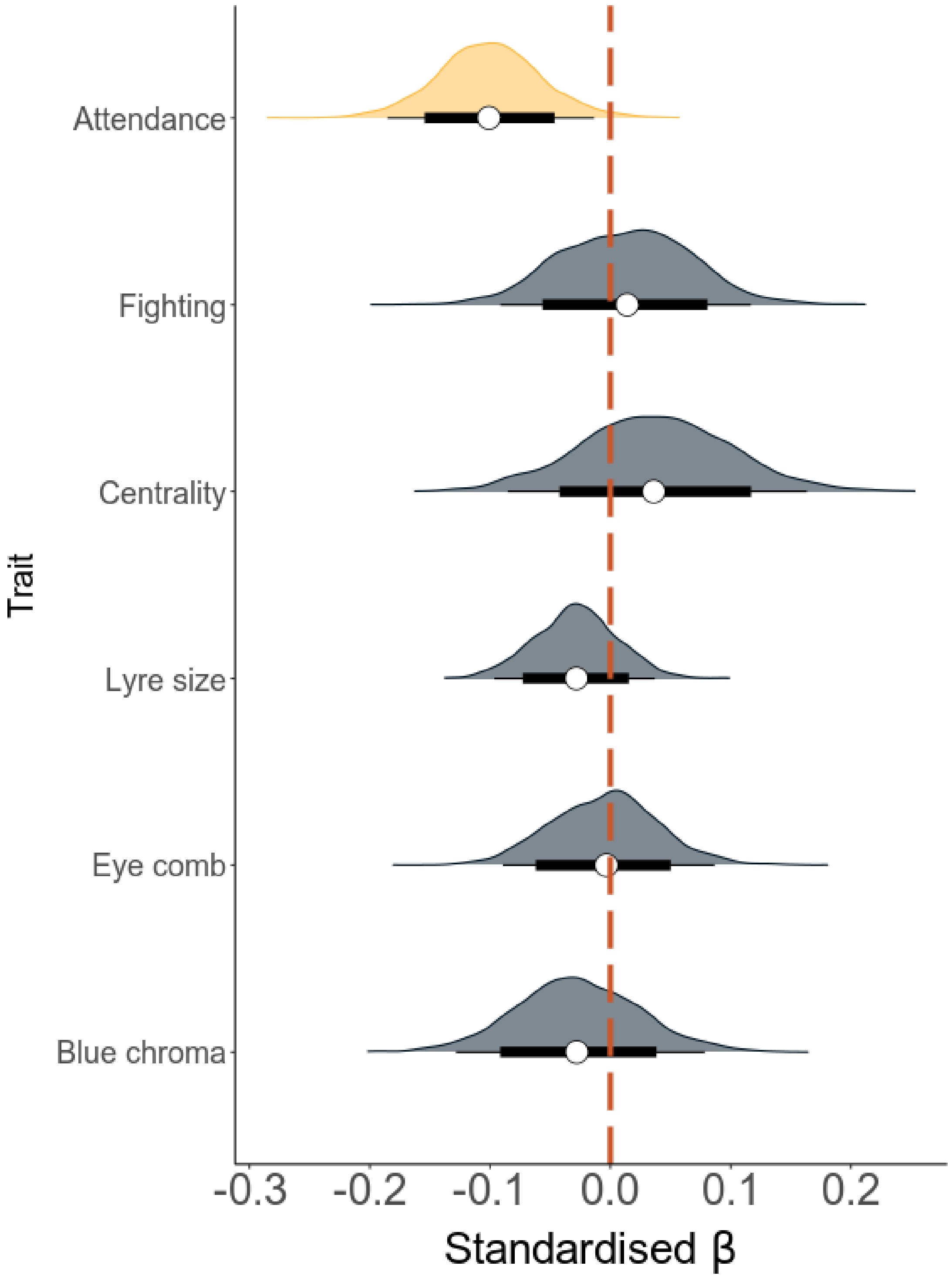


Figure 9.2: Hom het load results


Attendance
Fighting
Centrality
Lyre size
Eye comb
Blue chroma
Trait
-0.3
-0.2
-0.1
0.0
0.1
0.2
Standardised β

SnpEff:

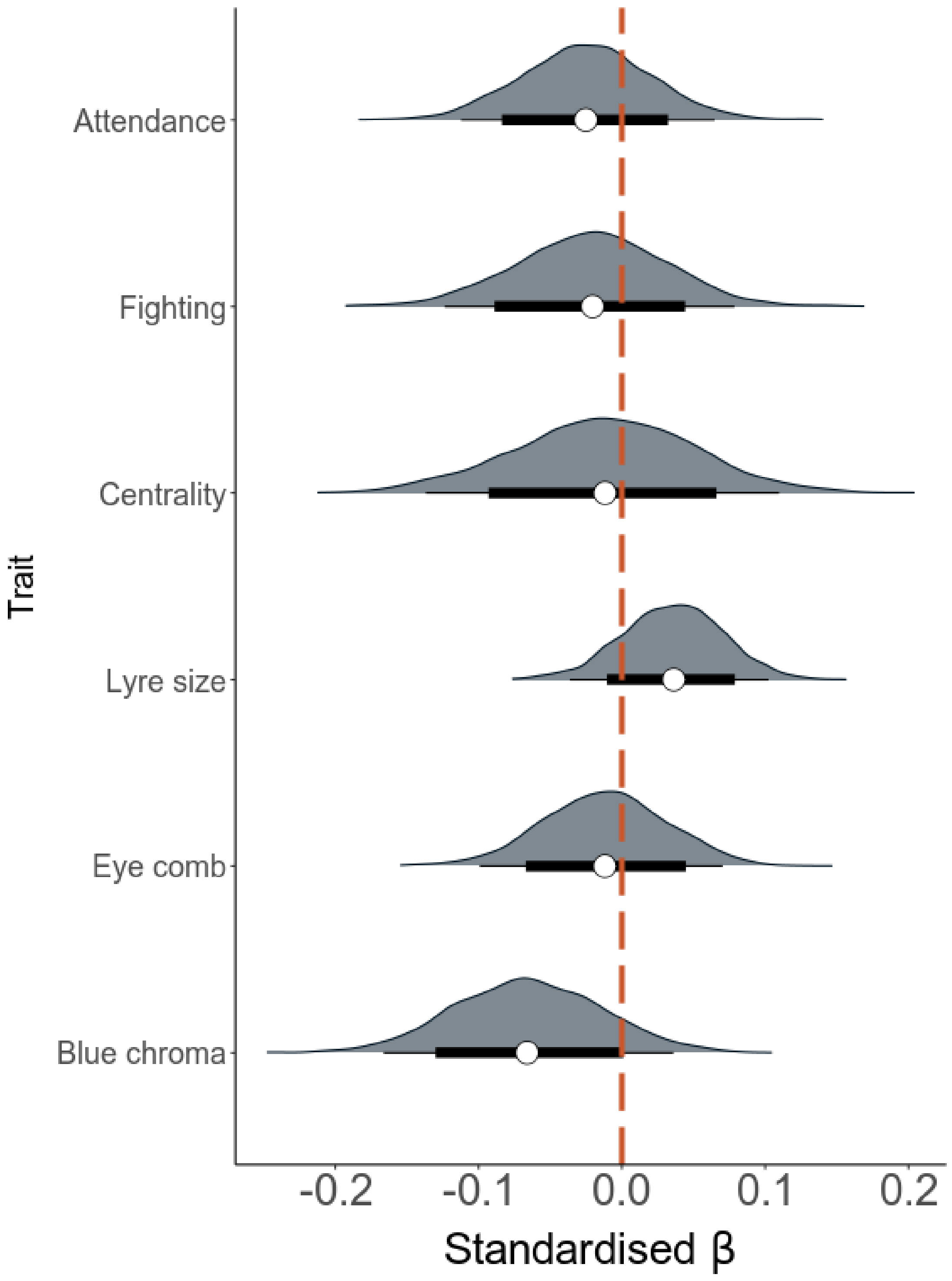
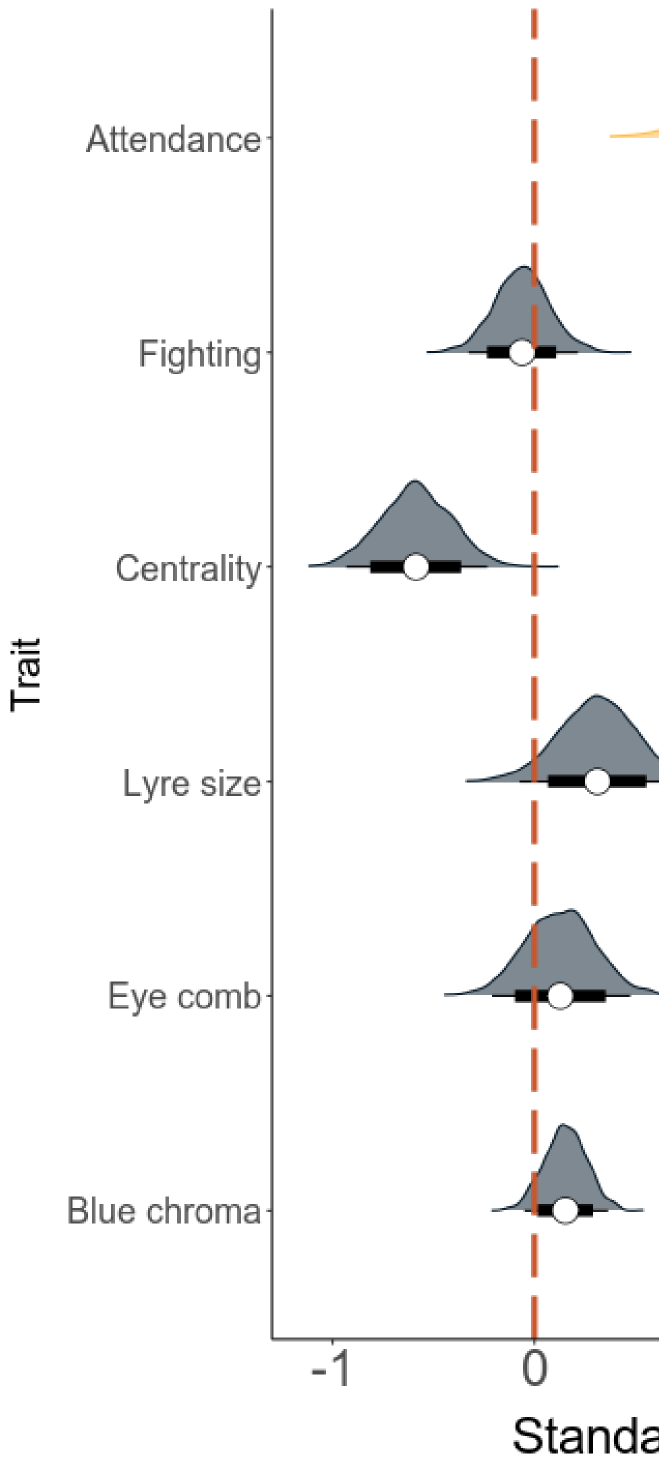


Figure 9.3: MS set 1 results SnpEff



And for model set 2 (traits on MS) for GERP and SnpEff:

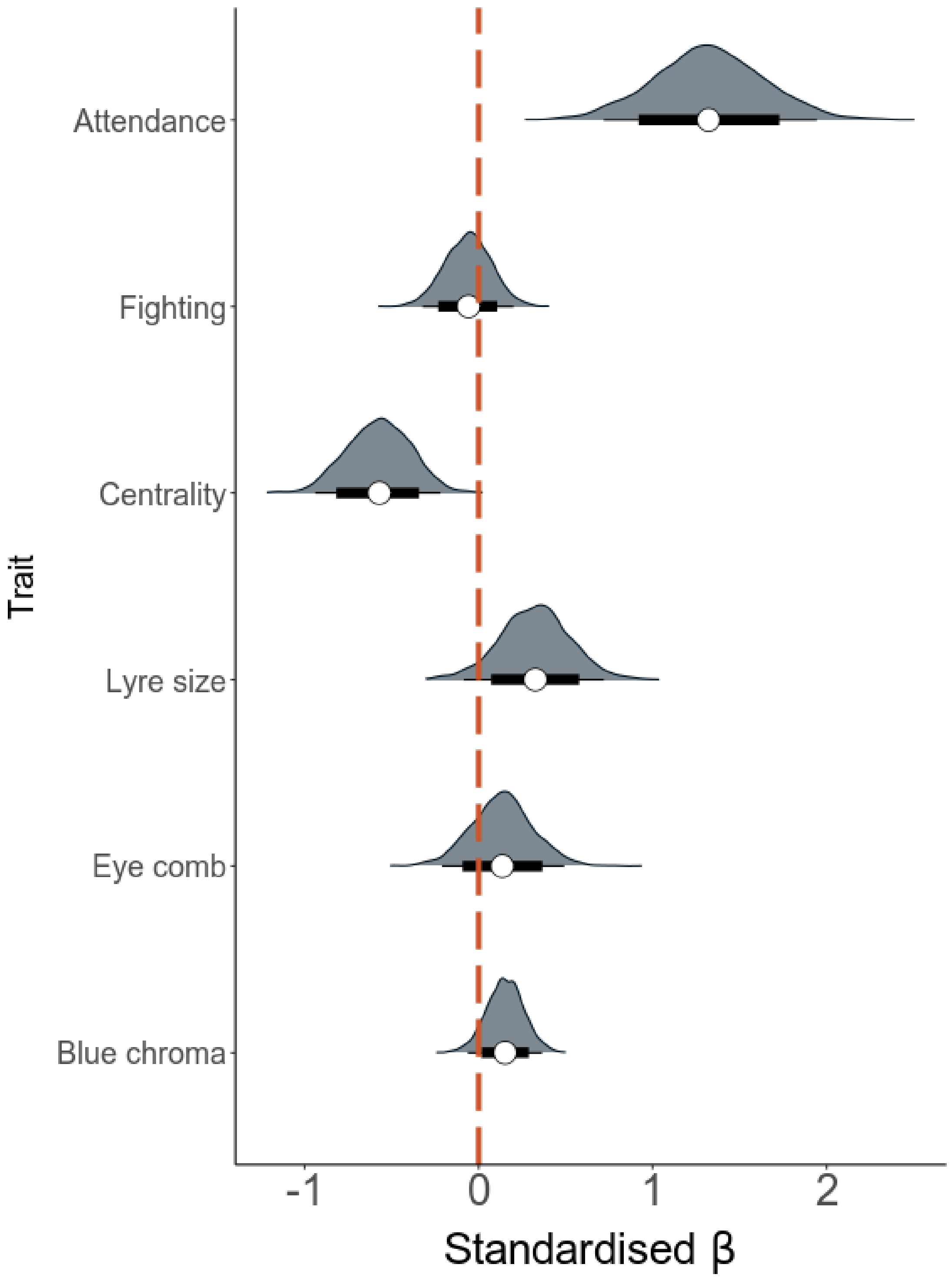


Figure 9.4: MS set 2 results SnpEff

treatment	mediator	method	indirect_median	indirect_lower	indirect_upper	direct_
b_scaletotal_load	b_scalelyre	gerp45	-0.01	-0.04	0.01	
b_scaletotal_load	b_scaleeyec	gerp45	0.00	-0.02	0.02	
b_scaletotal_load	b_scaleblue	gerp45	0.00	-0.03	0.01	
b_scaletotal_load	b_scaleattend	gerp45	-0.13	-0.28	-0.01	
b_scaletotal_load	b_scalefight	gerp45	0.00	-0.02	0.02	
b_scaletotal_load	b_scaledist	gerp45	-0.02	-0.11	0.05	
b_scaletotal_load	b_scalelyre	high	0.01	-0.01	0.05	
b_scaletotal_load	b_scaleeyec	high	0.00	-0.03	0.02	
b_scaletotal_load	b_scaleblue	high	-0.01	-0.04	0.01	
b_scaletotal_load	b_scaleattend	high	-0.03	-0.16	0.09	
b_scaletotal_load	b_scalefight	high	0.00	-0.01	0.02	
b_scaletotal_load	b_scaledist	high	0.01	-0.06	0.09	

We can check out the result of the direct and indirect effects as follows:

```
effects <-
 ↪  read.csv("../output/models/annual/direct_indirect_summary.csv")
effects %>% kbl() %>%  kable_classic_2() %>% scroll_box(width = "99%",
 ↪  height = "200px")
```

10 Load per genomic region

10.1 Introduction

Both functional noncoding and protein-coding regions can be under selective constraint48,49. However, the former include various regulatory elements such as promoters, enhancers and silencers, which play different roles in gene regulation and might therefore be expected to experience different strengths of purifying selection. To investigate whether the fitness effects of deleterious mutations vary by genomic region, we classified each deleterious mutation according to whether it was located within a promoter, transcription start site, intron or exon. Then, we calculate total load based on these subsets and model their effects on LMS.

10.1.1 Genomic regions

We use a combination of GenomicRanges and rtracklayer and other packages to divide up the genome annotation file into the four genomic regions.

10.1.1.1 Subset reference genome

```
#### Packages #####
pacman::p_load(BiocManager, rtracklayer, GenomicFeatures, BiocGenerics,
 ↪   data.table, dplyr, genomation, GenomicRanges, tibble)

#### Genome data ####
# first change scaffold names
gff_raw <-
 ↪   fread("data/genomic/annotation/PO2979_Lyrurus_tetrix_black_grouse.annotation.gff")
 ↪
gff_raw$V1 <- gsub(";", "__", gff_raw$V1)
gff_raw$V1 <- gsub("=", "_", gff_raw$V1)
write.table(gff_raw, file =
 ↪   "data/genomic/PO2979_Lyrurus_tetrix_black_grouse.annotation_editedscafnames.gff",
 ↪   sep = "\t", col.names = FALSE, quote=F, row.names = FALSE)
```

```
#read in new file
gff <-
↪  makeTxDbFromGFF("data/genomic/PO2979_Lyrurus_tetrix_black_grouse.annotation_editedscaf
↪  format="gff3", organism="Lyrurus tetrix")

## divide up between the 4 regions ###

promoters <- promoters(gff, upstream=2000, downstream=200,
↪  columns=c("tx_name", "gene_id")) # From NIOO
TSS <- promoters(gff, upstream=300, downstream=50, columns=c("tx_name",
↪  "gene_id")) # TSS as in Laine et al., 2016. Nature Communications
exons_gene <- unlist(exonsBy(gff, "gene")) # group exons by genes
introns <- unlist(intronsByTranscript(gff, use.names=TRUE))

### write out files
export(promoters, "data/genomic/annotation/promoters.gff3")
export(TSS, "data/genomic/annotation/TSS.gff3")
introns@ranges@NAMES[is.na(introns@ranges@NAMES)]<-"Unknown"
export(introns, "data/genomic/annotation/introns_transcripts.gff3")
export(exons_gene, "data/genomic/annotation/exons_gene.gff3")
```

10.1.1.2 Annotate deleterious mutations

Then, we load in all deleterious mutations for GERP and SnpEff separately and annotate the mutations according to the four regions.

```
###### Annotate high impact and GERP mutations #####
### annotate gene regions per SNP for high impact and gerp mutations ###

### load data ###

load(file = "output/load/snpeff/snpeff_high.RData")
load(file = "output/load/gerp/gerp_over4.RData")

### change scaf names
snpeff$CHROM <- gsub(";", "__", snpeff$CHROM)
snpeff$CHROM <- gsub("=", "_", snpeff$CHROM)

gerp$chr <- gsub(";", "__", gerp$chr)
gerp$chr <- gsub("=", "_", gerp$chr)
```

```r
### remove genotypes: not necessary here
snpeff <- snpeff[,c(1:9)]
gerp <- gerp[,c(1:9)]

####### Gene regions ###########

### load annotation data
annotation_dir <- "data/genomic/annotation"

promoter=unique(gffToGRanges(paste0(annotation_dir, "/promoters.gff3")))
TSS=unique(gffToGRanges(paste0(annotation_dir, "/TSS.gff3")))
introns=unique(gffToGRanges(paste0(annotation_dir,
 ↪  "/introns_transcripts.gff3")))
exons_gene=unique(gffToGRanges(paste0(annotation_dir,
 ↪  "/exons_gene.gff3")))

#### Annotate SNPeff regions ####
snpeff$end <- snpeff$POS
snpeff$start <- snpeff$POS
snpef_gr <- as(snpeff, "GRanges")

snpef_promoter <- as.data.frame(subsetByOverlaps(snpef_gr, promoter))
 ↪  %>%
  add_column("region_promoter" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

snpef_TSS <- as.data.frame(subsetByOverlaps(snpef_gr, TSS))%>%
  add_column("region_tss" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

snpef_exons <- as.data.frame(subsetByOverlaps(snpef_gr, exons_gene))%>%
  add_column("region_exon" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

snpef_introns <- as.data.frame(subsetByOverlaps(snpef_gr, introns))%>%
  add_column("region_intron" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

snpef_all <- left_join(snpeff, snpef_promoter[,c("chr", "start",
 ↪  "region_promoter")], by = c("CHROM" = "chr", "start"))%>%
  left_join(snpef_TSS[,c("chr", "start", "region_tss")], by = c("CHROM"
   ↪  = "chr", "start"))%>%
  left_join(snpef_exons[,c("chr", "start", "region_exon")], by =
   ↪  c("CHROM" = "chr", "start"))%>%
  left_join(snpef_introns[,c("chr", "start", "region_intron")], by =
   ↪  c("CHROM" = "chr", "start"))

# correct promoters
snpef_all <- snpef_all %>% mutate(region_promoter = case_when(
  region_tss == 1 & region_promoter == 1 ~ NA,
  is.na(region_tss) & region_promoter == 1 ~ 1
))

save(snpef_all, file =
 ↪  "output/load/snpeff/snpeff_high_annotated_region.RData")

#### Annotate gerp regions ####
gerp$end <- gerp$pos
gerp$start <- gerp$pos
gerp_gr <- as(gerp, "GRanges")

gerp_promoter <- as.data.frame(subsetByOverlaps(gerp_gr, promoter)) %>%
  add_column("region_promoter" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

gerp_TSS <- as.data.frame(subsetByOverlaps(gerp_gr, TSS))%>%
  add_column("region_tss" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

gerp_exons <- as.data.frame(subsetByOverlaps(gerp_gr, exons_gene))%>%
  add_column("region_exon" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

gerp_introns <- as.data.frame(subsetByOverlaps(gerp_gr, introns))%>%
  add_column("region_intron" = 1) %>% dplyr::rename(chr = seqnames)%>%
   ↪  unique()

gerp_all <- left_join(gerp, gerp_promoter[,c("chr", "start",
 ↪  "region_promoter")], by = c("chr" = "chr", "start"))%>%
   left_join(gerp_TSS[,c("chr", "start", "region_tss")], by = c("chr" =
    ↪  "chr", "start"))%>%
  left_join(gerp_exons[,c("chr", "start", "region_exon")], by = c("chr"
   ↪  = "chr", "start"))%>%
  left_join(gerp_introns[,c("chr", "start", "region_intron")], by =
   ↪  c("chr" = "chr", "start"))

# correct promoters
gerp_all <- gerp_all %>% mutate(region_promoter = case_when(
  region_tss == 1 & region_promoter == 1 ~ NA,
  is.na(region_tss) & region_promoter == 1 ~ 1
))

save(gerp_all, file = "output/load/gerp/gerp_annotated_region.RData")
```

10.1.1.3 Calculate mutation load

These two files (gerp_all and snpeff_all) contain the SNP locations and additional binary columns whether the mutations was located in one of the 4 regions. We can use this file together with the files containing the genotypes to calculate mutation load based on the subsets of mutations only.

```
#### Calculate load per region #####

# load annotated gerp data
load(file = "output/load/gerp/gerp_annotated_region.RData")
gerp_all$chr <- gsub("__", ";", gerp_all$chr)
gerp_all$chr <- gsub("HRSCAF_", "HRSCAF=", gerp_all$chr)

# load annotation
load(file = "output/load/snpeff/snpeff_high_annotated_region.RData")
snpef_all$CHROM <- gsub("__", ";", snpef_all$CHROM)
snpef_all$CHROM <- gsub("HRSCAF_", "HRSCAF=", snpef_all$CHROM)

## load functions to calculate load
source("scripts/7_calculate_load/function_calculate_load.R")

#### calculate load per region as defined below

regions <- c("region_promoter", "region_tss"
 ↪  ,"region_exon","region_intron")
```

```r
#load existing combined load file
load("output/load/all_loads_combined_da_nosex_29scaf.RData") #loads no
↪  sex chr only 29 scaf

# load gt again to include genotypes
load(file = "output/load/snpeff/snpeff_high.RData")
load(file = "output/load/gerp/gerp_over4.RData")

#first gerp5
load_per_region <- load
for (region in regions){
  subset_locs <- subset(gerp_all, gerp_all[,region] == 1) #subset based
↪  on region name

  subset_locs$chr_pos <- paste0(subset_locs$chr, "_", subset_locs$pos)
↪  #make a col for the snp position
  gerp$chr_pos <- paste0(gerp$chr, "_", gerp$pos) #make a col for the
↪  snp position

  sub_genotypes <- subset(gerp, chr_pos %in% subset_locs$chr_pos)
↪  #subset genotypes based on subset

  sub_genotypes$chr_pos <- NULL #remove chr_pos again

  loadtype = paste0("gerp45", gsub("region", "", region))

  load_sub <- calculate_load_gerp(sub_genotypes, loadtype = loadtype,
↪  output_vcf = F) #calculate loads

  load_per_region <- rbind(load_per_region, load_sub[,c("id",
↪  "het_load", "hom_load", "total_load", "loadtype")])
}

#snpeff
for (region in regions){
  subset_locs <- subset(snpef_all, snpef_all[,region] == 1) #subset
↪  based on region name

  subset_locs$chr_pos <- paste0(subset_locs$CHROM, "_", subset_locs$POS)
↪  #make a col for the snp position
```

```
  snpeff$chr_pos <- paste0(snpeff$CHROM, "_", snpeff$POS) #make a col
↪  for the snp position

  sub_genotypes <- subset(snpeff, chr_pos %in% subset_locs$chr_pos)
↪  #subset genotypes based on subset

  sub_genotypes$chr_pos <- NULL #remove chr_pos again

  loadtype = paste0("high", gsub("region", "", region))
  load_sub <- calculate_load_snpeff(sub_genotypes, loadtype = loadtype,
↪  output_vcf = F) #calculate loads

  load_per_region <- rbind(load_per_region, load_sub[,c("id",
↪  "het_load", "hom_load", "total_load", "loadtype")])
}

save(load_per_region, file =
↪  "output/load/all_loads_combined_da_nosex_29scaf_plus_per_region.RData")
```

10.1.2 Modelling

We then computed the exact same models as the total load models presented before with the following model structure:

`LMS ~ scale(total_load) + core + (1|site)`

Where total load is not a measure taken from a subset of deleterious mutations located within a specific genomic region.

10.1.3 Results

Below you find the posterior distributions:

```
library(readxl); library(dplyr); library(kableExtra)
gerp <- read.csv("../output/models/intervals/regions_gerp45.csv")
gerp %>% kbl()
```

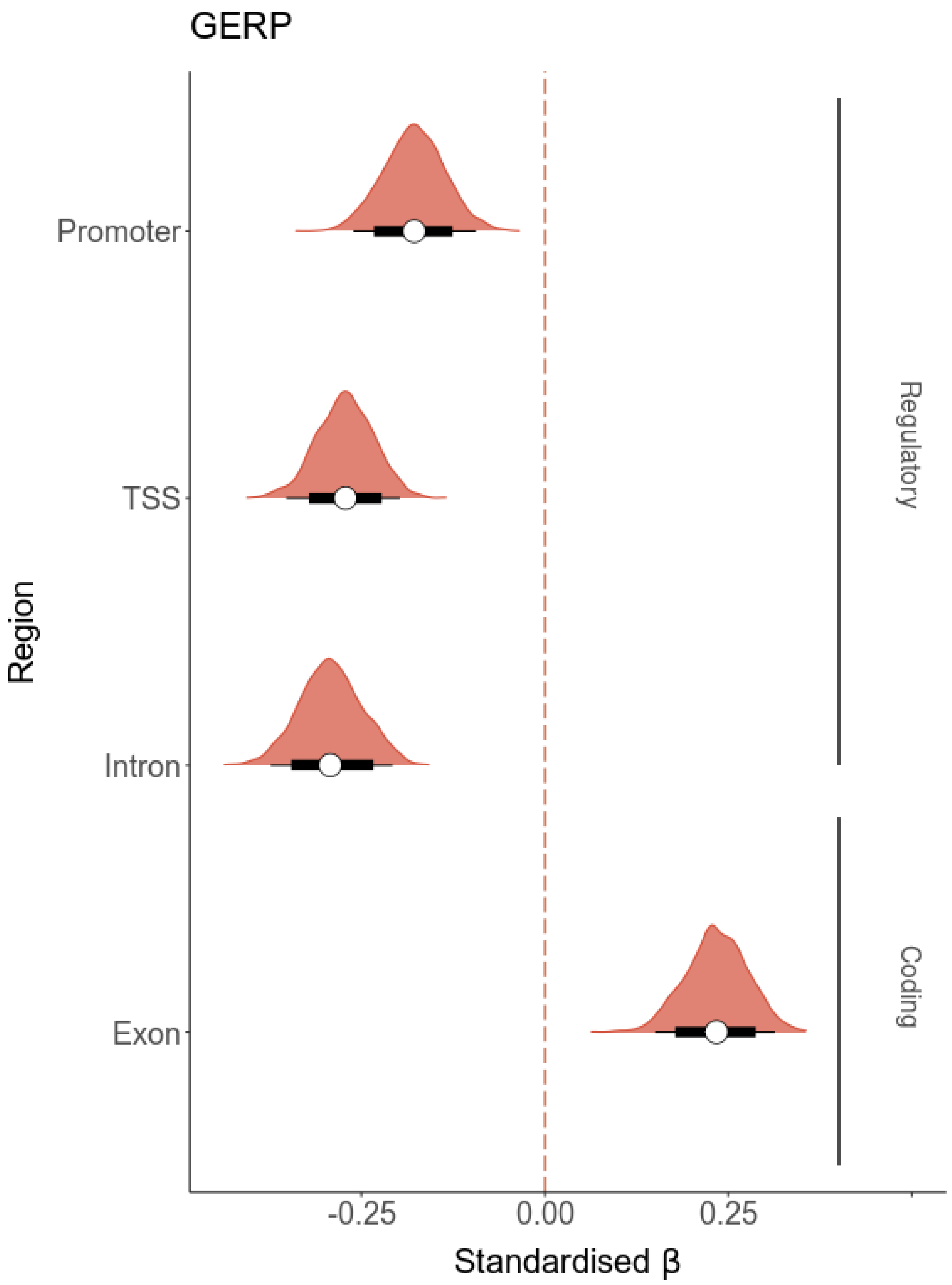


Figure 10.1: Results per genomic region GERP

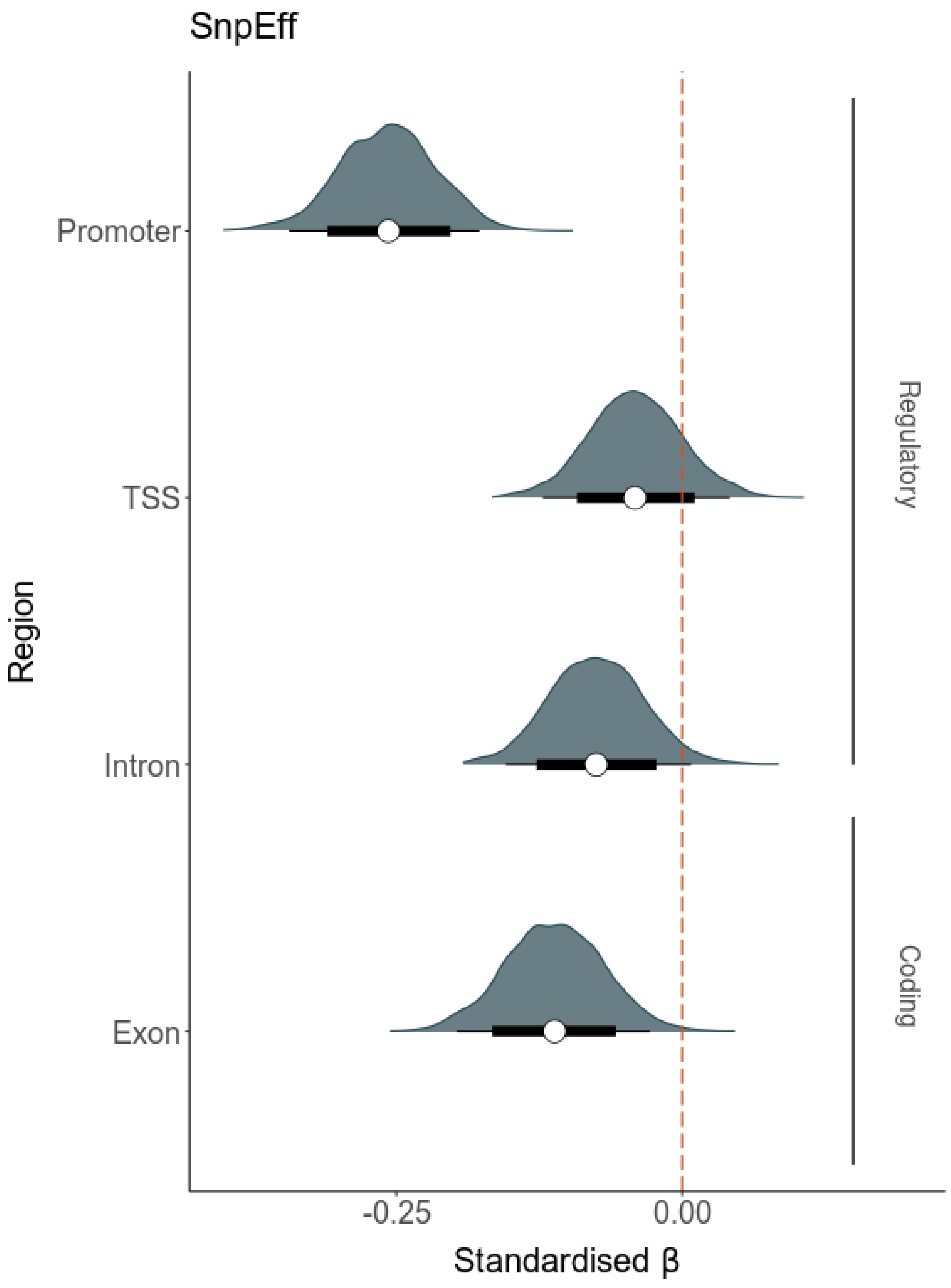


Figure 10.2: Results per genomic region SnpEff

parameter	outer_width	inner_width	point_est	ll	l	m	h	hh	region
b_scaletotal_load	0.95	0.8	median	-0.26	-0.23	-0.18	-0.13	-0.09	Promote
b_scaletotal_load	0.95	0.8	median	-0.35	-0.32	-0.27	-0.22	-0.20	TSS
b_scaletotal_load	0.95	0.8	median	-0.37	-0.35	-0.29	-0.23	-0.21	Intron
b_scaletotal_load	0.95	0.8	median	0.15	0.18	0.23	0.29	0.31	Exon

```r
high <- read.csv("../output/models/intervals/regions_high.csv")
high %>% kbl()
```

parameter	outer_width	inner_width	point_est	ll	l	m	h	hh	region
b_scaletotal_load	0.95	0.8	median	-0.34	-0.31	-0.26	-0.20	-0.18	Promote
b_scaletotal_load	0.95	0.8	median	-0.12	-0.09	-0.04	0.01	0.04	TSS
b_scaletotal_load	0.95	0.8	median	-0.15	-0.13	-0.08	-0.02	0.01	Intron
b_scaletotal_load	0.95	0.8	median	-0.20	-0.17	-0.11	-0.06	-0.03	Exon

These results indicate that regulatory regions, especially promoter regions (SnpEff) are very important!

11 Random subsampling

11.1 Introduction

Variation in the effects of mutation load among the various models presented could be attributed to variation in the effect sizes of the individual mutations, or alternatively due to the differences in the number of mutations that contribute to the mutation load estimate. Assuming that all mutations that are identified have negative fitness consequences to some extent, we can expect that a larger number of mutations explain more fitness variation. We tested for the effect of total load on LMS between GERP and SnpEff mutations and the four genomic regions by controlling for the number of mutations.

11.2 Methods

We randomly subset over all deleterioius GERP and SnpEff mutations (separately), and per genomic region for the two approaches.

First, we load in our data and make different subsets of the data: all GERP mutations, GERP mutations per region (4 dataframes) and the same for SnpEff mutations

```r
### load packages
pacman::p_load(tidyverse, data.table)

#load data mutations
load(file = "output/load/snpeff/snpeff_high.RData")
load(file = "output/load/gerp/gerp_over4.RData")

# load annotation data
load(file = "output/load/snpeff/snpeff_high_annotated_region.RData")
snpef_all$CHROM <- gsub("__", ";", snpef_all$CHROM)
snpef_all$CHROM <- gsub("HRSCAF_", "HRSCAF=", snpef_all$CHROM)
snpef_all$chr_pos <- paste0(snpef_all$CHROM, "_", snpef_all$POS)

load(file = "output/load/gerp/gerp_annotated_region.RData")
```

```
gerp_all$chr <- gsub("__", ";", gerp_all$chr)
gerp_all$chr <- gsub("HRSCAF_", "HRSCAF=", gerp_all$chr)
gerp_all$chr_pos <- paste0(gerp_all$chr, "_", gerp_all$pos)

### make subsets of mutations based on region
# subset snpef based on annotation
snpeff_exons <- subset(snpef_all, region_exon == 1)
snpeff_tss <- subset(snpef_all, region_tss == 1)
snpeff_introns <- subset(snpef_all, region_intron == 1)
snpeff_promoter <- subset(snpef_all, region_promoter == 1 &
 ↪  is.na(region_tss))

snpeff$chr_pos <- paste0(snpeff$CHROM, "_", snpeff$POS)

snpeff_exons_gt <- subset(snpeff, chr_pos %in% snpeff_exons$chr_pos)
snpeff_tss_gt <- subset(snpeff, chr_pos %in% snpeff_tss$chr_pos)
snpeff_introns_gt <- subset(snpeff, chr_pos %in% snpeff_introns$chr_pos)
snpeff_promoter_gt <- subset(snpeff, chr_pos %in%
 ↪  snpeff_promoter$chr_pos)

# subset gerp based on annotation
gerp_exons <- subset(gerp_all, region_exon == 1)
gerp_tss <- subset(gerp_all, region_tss == 1)
gerp_introns <- subset(gerp_all, region_intron == 1)
gerp_promoter <- subset(gerp_all, region_promoter == 1 &
 ↪  is.na(region_tss))

gerp$chr_pos <- paste0(gerp$chr, "_", gerp$pos)

gerp_exons_gt <- subset(gerp, chr_pos %in% gerp_exons$chr_pos)
gerp_tss_gt <- subset(gerp, chr_pos %in% gerp_tss$chr_pos)
gerp_introns_gt <- subset(gerp, chr_pos %in% gerp_introns$chr_pos)
gerp_promoter_gt <- subset(gerp, chr_pos %in% gerp_promoter$chr_pos)
```

We then create a function to execute the subsetting:

```
## create function to subset X number of SnpEffs and model its effect on
 ↪  fitness

random_draws <- function(geno, n_draws, n_mutations, file, method,
 ↪  emperical_beta){
  source("scripts/theme_ggplot.R")
  all_draws <- data.frame()

  if(method == "GERP"){
    for (i in 1:n_draws){
      draw <- geno[sample(nrow(geno), n_mutations),] #randomly draw snps
      ## load functions
      source("scripts/7_calculate_load/function_calculate_load.R")
      load <- calculate_load_gerp(draw, output_vcf = F, loadtype =
↪    "random_draw") #calculate load
      model_out <- model_load(load, i)
      model_out$method <- method

      all_draws <- rbind(all_draws, model_out)
    }}

  if(method == "High impact"){
    for (i in 1:n_draws){
      draw <- geno[sample(nrow(geno), n_mutations),] #randomly draw snps
      ## load functions
      source("scripts/7_calculate_load/function_calculate_load.R")
      load <- calculate_load_snpeff(draw, output_vcf = F, loadtype =
↪    "random_draw") #calculate load
      model_out <- model_load(load, i)
      model_out$method <- method

      all_draws <- rbind(all_draws, model_out)
    }}

  ### conclusion
  all_draws <- all_draws %>% mutate(conclusion = as.factor(case_when(
    beta < 0 & pval < 0.05 ~ "Significantly negative",
    beta > 0 & pval < 0.05 ~ "Significantly positive",
    TRUE ~ "Insignificant"
  )))

  return(all_draws)
  save(all_draws, file = file)
}

### the model_load function looks like follows:
```

```
model_load <- function(load, ndraw){
  #join with phenotypes
  load("data/phenotypes/phenotypes_lifetime.RData") #LMS
  pheno <- pheno_wide %>% mutate(core = as.factor(case_when(is.na(LMS) ~
 ↪  "no core", !is.na(LMS) ~ "core")))

  data_pheno <- left_join(load, pheno[,c("id", "LMS", "LMS_min", "core",
 ↪  "site", "born")], by = "id")

  model <- glmmTMB::glmmTMB(LMS_min ~ scale(total_load) + core +
 ↪  (1|site), family = "poisson", ziformula = ~1, data = data_pheno)
  summary <- summary(model)

  beta <- summary$coefficients$cond["scale(total_load)","Estimate"]
  se <- summary$coefficients$cond["scale(total_load)","Std. Error"]
  zval <- summary$coefficients$cond["scale(total_load)","z value"]
  pval <- summary$coefficients$cond["scale(total_load)","Pr(>|z|)"]

  out <- data.frame(ndraw = ndraw,
                    beta = beta,
                    se = se,
                    zval = zval,
                    pval = pval)

  return(out)
}
```

Then we run this function for the 10 subsets of mutations, with varying number of mutations that get sampled:

```
# run functions

# all mutations
draws_gerp <- random_draws(geno = gerp, n_draws = 5000, n_mutations =
 ↪  1000, file = "output/random_draws/all_gerp.RData", method="GERP",
 ↪  emperical_beta = -0.21)
save(draws_gerp, file = "output/random_draws/all_gerp.RData")
```

```
draws_high <- random_draws(geno = snpeff, n_draws = 5000, n_mutations =
 ↪  1000, file = "output/random_draws/all_high.RData", method = "High
 ↪  impact", emperical_beta = -0.07)
save(draws_high, file = "output/random_draws/all_high.RData")

# per region
# high
draws_high_promoter <- random_draws(geno = snpeff_promoter_gt, n_draws =
 ↪  5000, n_mutations = 500, file =
 ↪  "results/random_draws/promoter_high.RData", method = "High impact",
 ↪  emperical_beta = 0)
save(draws_high_promoter, file =
 ↪  "output/random_draws/promoter_high.RData")

draws_high_tss <- random_draws(geno = snpeff_tss_gt, n_draws = 5000,
 ↪  n_mutations = 500, file = "results/random_draws/tss_high.RData",
 ↪  method = "High impact", emperical_beta = 0)
save(draws_high_tss, file = "output/random_draws/tss_high.RData")

draws_high_intron <- random_draws(geno = snpeff_introns_gt, n_draws =
 ↪  5000, n_mutations = 500, file =
 ↪  "results/random_draws/intron_high.RData", method = "High impact",
 ↪  emperical_beta = 0)
save(draws_high_intron, file = "output/random_draws/intron_high.RData")

draws_high_exon <- random_draws(geno = snpeff_exons_gt, n_draws = 5000,
 ↪  n_mutations = 500, file = "results/random_draws/exon_high.RData",
 ↪  method = "High impact", emperical_beta = 0)
save(draws_high_exon, file = "output/random_draws/exon_high.RData")

# gerp
draws_gerp_promoter <- random_draws(geno = gerp_promoter_gt, n_draws =
 ↪  5000, n_mutations = 500, file =
 ↪  "results/random_draws/promoter_gerp.RData", method = "GERP",
 ↪  emperical_beta = 0)
save(draws_gerp_promoter, file =
 ↪  "output/random_draws/promoter_gerp.RData")

draws_gerp_tss <- random_draws(geno = gerp_tss_gt, n_draws = 5000,
 ↪  n_mutations = 500, file = "results/random_draws/tss_gerp.RData",
 ↪  method = "GERP", emperical_beta = 0)
```

```r
save(draws_gerp_tss, file = "output/random_draws/tss_gerp.RData")

draws_gerp_intron <- random_draws(geno = gerp_introns_gt, n_draws =
  ↪ 5000, n_mutations = 500, file =
  ↪ "results/random_draws/intron_gerp.RData", method = "GERP",
  ↪ emperical_beta = 0)
save(draws_gerp_intron, file = "output/random_draws/intron_gerp.RData")

draws_gerp_exon <- random_draws(geno = gerp_exons_gt, n_draws = 5000,
  ↪ n_mutations = 500, file = "results/random_draws/exon_gerp.RData",
  ↪ method = "GERP", emperical_beta = 0)
save(draws_gerp_exon, file = "output/random_draws/exon_gerp.RData")
```

11.3 Results

We can then plot the results as histograms:

```r
### load packages ####
pacman::p_load(tidyverse, ggpubr, extrafont, cowplot, data.table)

source("scripts/theme_ggplot.R")

#### total GERP #####
load(file="output/random_draws/all_gerp.RData")
load(file="output/random_draws/all_high.RData")

sum_gerp <- draws_gerp %>%
  summarize(lower_95 = quantile(beta, probs=c(0.025)),
            upper_95 = quantile(beta, probs=c(0.975)),
            lower_80 = quantile(beta, probs=c(0.1)),
            upper_80 = quantile(beta, probs=c(0.9)),
            mean = mean(beta))

ggplot(draws_gerp, aes(x = beta)) +
  xlim(-1.2,1.2)+
  ylim(-50, 600)+
  geom_histogram(aes(fill = beta < 0, col = beta < 0), linewidth=0.5,
   ↪ bins=40)+
  scale_fill_manual(values =alpha(c("grey60", clr_gerp), 0.7)) + #
  scale_color_manual(values =c("grey60", clr_gerp)) +
  geom_segment(data=sum_gerp, aes(x = lower_95,
                                  xend = upper_95,
                                  y = 0), col = "black", linewidth=1)+
  geom_segment(data=sum_gerp, aes(x = lower_80,
                                  xend = upper_80,
                                  y = 0), col = "black", linewidth=3)+
  geom_point(data=sum_gerp,aes(x = mean, y = 0), fill="white",  col =
  ↪   "black", shape=21, size = 6)+
  geom_vline(xintercept = 0, col = "#ca562c", linetype="longdash",
  ↪   linewidth=0.6)+
  theme(plot.title = element_text(margin=margin(0,0,30,0)),
        panel.border = element_blank(),
        panel.grid = element_blank(),
        panel.spacing = unit(3,"lines"),
        strip.background = element_blank(),
        legend.position="none")+
  labs(x = expression("Standardised"~beta), y = "# draws", title =
  ↪   "GERP") -> total_gerp
```

We can then repeat this for the other subsets, leading to the following plots:

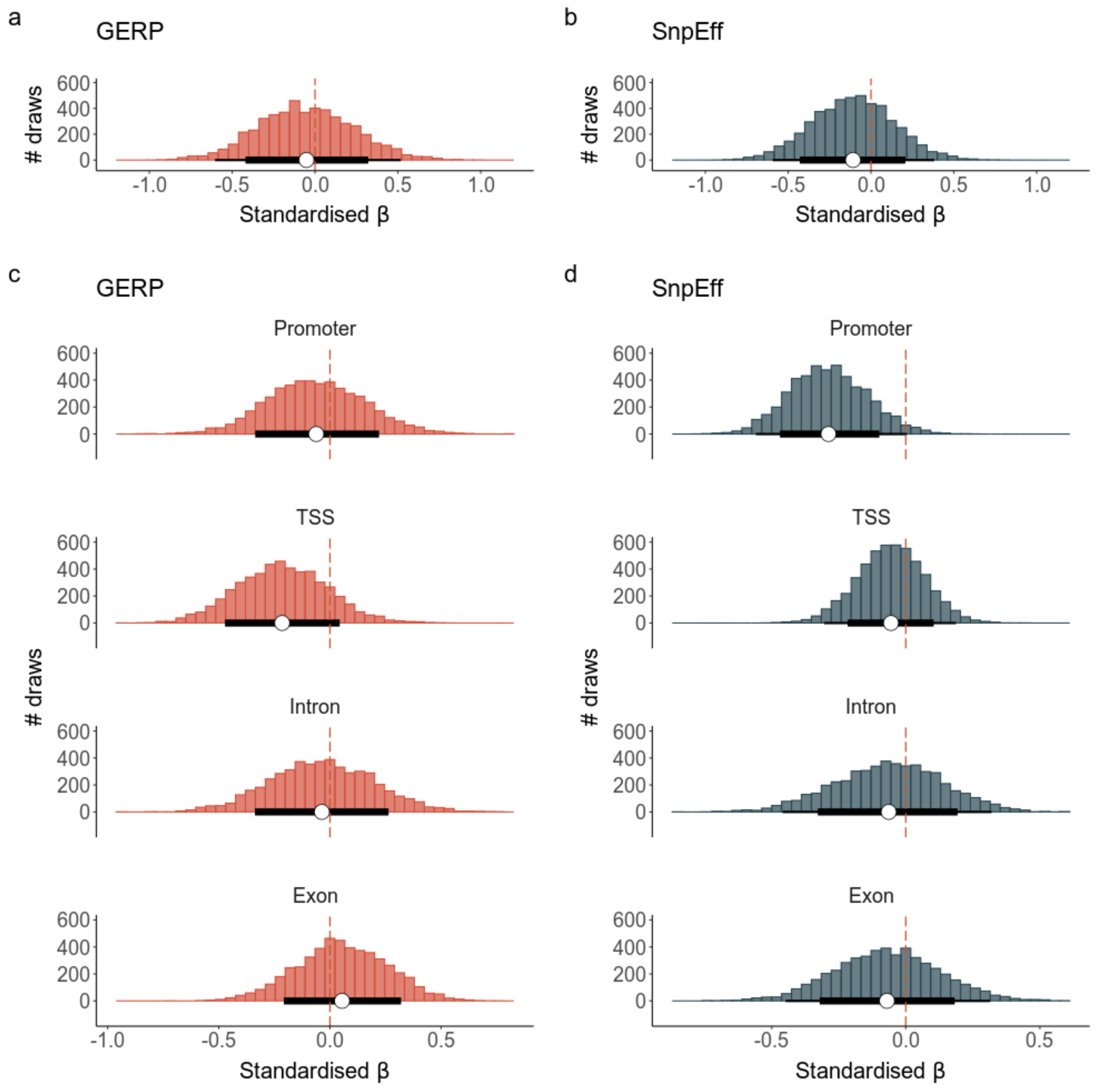


Figure 11.1: Random subsets

References

[1] Giorgio Bertorelle et al. "Genetic Load: Genomic Estimates and Applications in Non-Model Animals". In: *Nature Reviews Genetics* 23.8 (Aug. 2022), pp. 492–503. ISSN: 1471-0056, 1471-0064. DOI: 10.1038/s41576-022-00448-x. (Visited on 09/09/2022).

[2] Pablo Cingolani et al. "A Program for Annotating and Predicting the Effects of Single Nucleotide Polymorphisms, SnpEff". In: *Fly* 6.2 (Apr. 2012). _eprint: https://doi.org/10.4161/fly.19695, pp. 80–92. ISSN: 1933-6934. DOI: 10.4161/fly.19695.

[3] Eugene V. Davydov et al. "Identifying a High Fraction of the Human Genome to Be under Selective Constraint Using GERP++". In: *PLoS Computational Biology* 6.12 (Dec. 2010). Ed. by Wyeth W. Wasserman, e1001025. ISSN: 1553-7358. DOI: 10.1371/journal.pcbi.1001025. (Visited on 07/20/2022).