

# Subclonal variant calling with multiple samples and prior knowledge using shearwater

Moritz Gerstung

October 30, 2018

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>The statistical model</b>	<b>1</b>
2.1	Definition . . . . .	1
2.2	Testing for variants . . . . .	2
2.2.1	The OR model . . . . .	2
2.2.2	The AND model . . . . .	3
2.3	Estimating $\rho$ . . . . .	4
2.4	Using a prior . . . . .	5
<b>3</b>	<b>Using shearwater</b>	<b>6</b>
3.1	Minimal example . . . . .	6
3.2	More realistic example . . . . .	8

## Introduction

The shearwater algorithm was designed for calling subclonal variants in large ( $N = 10 \dots 1,000$ ) cohorts of deeply ( $\sim 100x$ ) sequenced unmatched samples. The large cohort allows for estimating a base-specific error profile on each position, which is modelled by a beta-binomial. A prior can be used to selectively increase the power of calling variants on known mutational hotspots. The algorithm is similar to deepSNV, but uses a slightly different parametrization and a Bayes factors instead of a likelihood ratio test.

If you are using shearwater, please cite

- Gerstung M, Papaemmanuil E, Campbell PJ (2014). “Subclonal variant calling with multiple samples and prior knowledge.” *Bioinformatics*, **30**, 1198-1204.

## The statistical model

### Definition

Suppose you have an experimental setup with multiple unrelated samples. Let the index  $i$  denote the sample,  $j$  the genomic position and  $k$  a particular nucleotide. Let  $X_{ijk}$  and  $X'_{ijk}$  denote the counts of nucleotide  $k$  in sample  $i$  on position  $j$  in forward and reverse read orientation, respectively. We assume that

$$\begin{aligned} X &\sim \text{BetaBin}(n, \mu, \rho) \\ X' &\sim \text{BetaBin}(n', \mu', \rho). \end{aligned} \tag{1}$$

are beta-binomially distributed. To test if there is a variant  $k$  in sample  $i$ , we compare the counts to a compound reference  $X_{ijk} = \sum_{h \in H} X_{hjk}$  and  $X'_{ijk} = \sum_{h \in H} X'_{hjk}$ . The subset of indices  $H$  is usually chosen such that  $H = \{h : h \neq j\}$ , that is the row sums  $X_{ijk}$  and  $X'_{ijk}$ . To reduce the effect of true

variants in other samples entering the compound reference, one may also choose  $H$  such that it only includes sample  $h$  with variant allele frequencies below a user defined threshold, typically 10%. We model the compound reference again as a beta-binomial,

$$\begin{aligned} \mathbf{X} &\sim \text{BetaBin}(\mathbf{n}, \nu, \rho) \\ \mathbf{X}' &\sim \text{BetaBin}(\mathbf{n}', \nu', \rho). \end{aligned} \quad (2)$$

## Testing for variants

Testing for the presence of a variant can now be formulated as a model selection problem in which we specify a null model and an alternative. Here we consider two options, "OR" and "AND".

The OR model

The OR model is defined in the following way:

$$\begin{aligned} M_0 : \quad &\mu = \nu \quad \vee \quad \mu' = \nu' \\ M_1 : \quad &\mu = \mu' > \nu, \nu'. \end{aligned} \quad (3)$$

Under the null model  $M_0$ , the mean rates of the beta-binomials are identical in sample  $i$  and the compound reference on at least one strand. Under the alternative model  $M_1$ , the mean rates  $\mu, \mu'$  are identical on both strands and greater than the mean in the compound reference on both strands.

Here we use the following point estimates for the parameters:

$$\begin{aligned} \hat{\mu} &= (X + X') / (n + n') \\ \hat{\nu} &= \mathbf{X} / \mathbf{n} \\ \hat{\nu}' &= \mathbf{X}' / \mathbf{n}' \\ \hat{\nu}_0 &= (X + \mathbf{X}) / (n + \mathbf{n}) \\ \hat{\nu}'_0 &= (X' + \mathbf{X}') / (n' + \mathbf{n}') \\ \hat{\mu}_0 &= X / n \\ \hat{\mu}'_0 &= X' / n'. \end{aligned} \quad (4)$$

Using these values, the Bayes factor is approximated by

$$\begin{aligned} \frac{\Pr(D | M_0)}{\Pr(D | M_1)} &= \frac{\Pr(X | \hat{\nu}_0) \Pr(X' | \hat{\mu}'_0) \Pr(\mathbf{X} | \hat{\nu}_0)}{\Pr(X | \hat{\mu}) \Pr(X' | \hat{\mu}) \Pr(\mathbf{X} | \hat{\nu})} \\ &+ \frac{\Pr(X | \hat{\mu}_0) \Pr(X' | \hat{\nu}'_0) \Pr(\mathbf{X}' | \hat{\nu}'_0)}{\Pr(X | \hat{\mu}) \Pr(X' | \hat{\mu}) \Pr(\mathbf{X}' | \hat{\nu}')} \\ &- \frac{\Pr(X | \hat{\nu}_0) \Pr(\mathbf{X} | \hat{\nu}_0) \Pr(X' | \hat{\nu}'_0) \Pr(\mathbf{X}' | \hat{\nu}'_0)}{\Pr(X | \hat{\mu}) \Pr(\mathbf{X} | \hat{\nu}) \Pr(X' | \hat{\mu}) \Pr(\mathbf{X}' | \hat{\nu}')} \end{aligned} \quad (5)$$

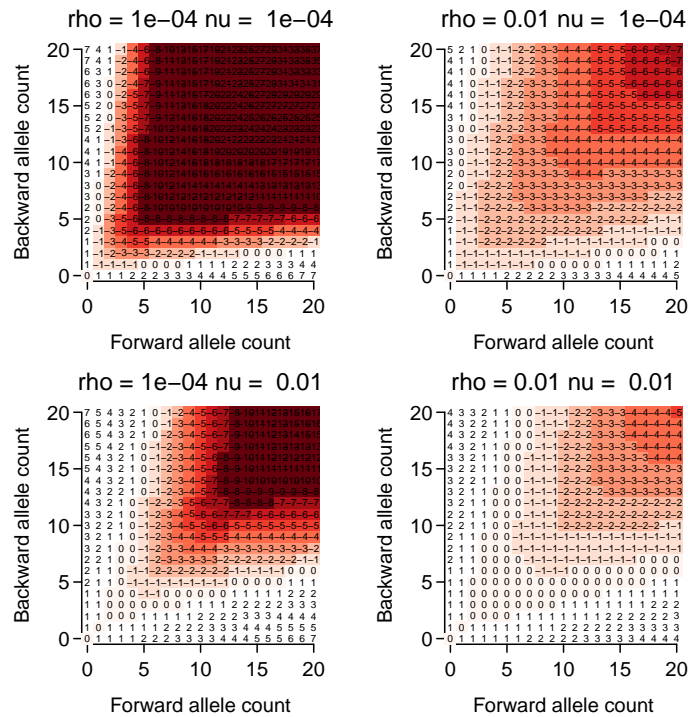
**Example** The Bayes factors can be computed using the `bbb` command:

```
library(deepSNV)
library(RColorBrewer)
n <- 100 ## Coverage
n_samples <- 1000 ## Assume 1000 samples
x <- 0:20 ## Nucleotide counts
X <- cbind(rep(x, each = length(x)), rep(x, length(x))) ## All combinations forward and reverse
par(bty="n", mgp = c(2, .5, 0), mar=c(3, 3, 2, 2)+.1, las=1, tcl=-.33, mfrow=c(2, 2))
for(nu in 10^c(-4, -2)){ ## Loop over error rates
  ## Create counts array with errors
  counts = aperm(array(c(rep(round(n_samples*n* c(nu, 1-nu, nu, 1-nu))), each=nrow(X)), cbind(n -
  dim=c(nrow(X), 4, 2)), c(3, 1, 2))
  for(rho in c(1e-4, 1e-2)){ ## Loop over dispersion factors
    ## Compute Bayes factors
    BF = bbb(counts, rho=rho, model="OR", return="BF")
```

```

## Plot
image(z=log10(matrix(BF[2,,1], nrow=length(x))),
      x=x,
      y=x,
      breaks=c(-100,-8:0),
      col=rev(brewer.pal(9,"Reds")),
      xlab = "Forward allele count",
      ylab="Backward allele count",
      main = paste("rho =", format(rho, digits=2), "nu = ", format(nu, digits=2), sep=""),
      font.main=1)
text(X[,1],X[,2],ceiling(log10(matrix(BF[2,,1], nrow=length(x))))), cex=0.5)
}

```



Here we have used a coverage of  $n = 100$  on both strands and computed the Bayes factors assuming 1,000 samples to estimate the error rate  $\nu = \nu'$  from. Shown are results for fixed values of  $\rho = \{10^{-4}, 10^{-2}\}$ .

The AND model

The AND model is defined in the following way:

$$\begin{aligned}
 M_0 : \quad & \mu = \nu \quad \wedge \quad \mu' = \nu' \\
 M_1 : \quad & \mu = \mu' > \nu, \nu'.
 \end{aligned}
 \tag{6}$$

Here the null model states that the error rates  $\nu = \mu$  and  $\nu' = \mu'$  are identical on both strands, which is more restrictive and hence in favour of the alternative.

In this case the Bayes factor is approximately

$$\frac{\Pr(D | M_0)}{\Pr(D | M_1)} = \frac{\Pr(X|\hat{\nu}_0) \Pr(\mathbf{X}|\hat{\nu}_0) \Pr(X'|\hat{\nu}'_0) \Pr(\mathbf{X}'|\hat{\nu}'_0)}{\Pr(X|\hat{\mu}) \Pr(\mathbf{X}|\hat{\nu}) \Pr(X'|\hat{\mu}) \Pr(\mathbf{X}'|\hat{\nu}')}
 \tag{7}$$

Example The behaviour of the AND model can be inspected by the following commands

```

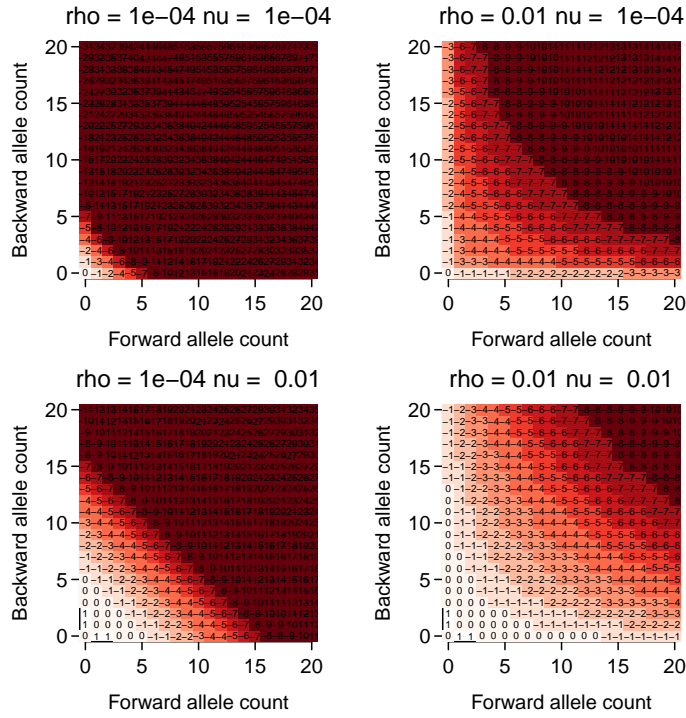
par(bty="n", mgp = c(2, .5, 0), mar=c(3,3,2,2)+.1, las=1, tcl=-.33, mfrow=c(2,2))
for(nu in 10^c(-4,-2)){ ## Loop over error rates
  ## Create counts array with errors

```

```

counts = aperm(array(c(rep(round(n_samples*n* c(nu,1-nu,nu,1-nu)), each=nrow(X)), cbind(n -
dim=c(nrow(X) ,4,2)), c(3,1,2))
for(rho in c(1e-4, 1e-2)){ ## Loop over dispersion factors
  ## Compute Bayes factors, mode = "AND"
  BF = bbb(counts, rho=rho, model="AND", return="BF")
  ## Plot
  image(z=log10(matrix(BF[2,,1], nrow=length(x))),
        x=x,
        y=x,
        breaks=c(-100,-8:0),
        col=rev(brewer.pal(9,"Reds")),
        xlab = "Forward allele count",
        ylab="Backward allele count",
        main = paste("rho =", format(rho, digits=2), "nu = ", format(nu, digi
font.main=1)
text(X[,1],X[,2],ceiling(log10(matrix(BF[2,,1], nrow=length(x)))), cex=0.5)
}
}

```



One realises that for small dispersion the Bayes factor depends mostly on the sum of the forward and reverse strands in the AND model.

### Estimating $\rho$

If the dispersion parameter  $\rho$  is not specified, it is estimated at each locus using the following method-of-moment estimator:

$$\hat{\rho} = \frac{Ns^2/(1 - \hat{\nu})/\hat{\nu} - \sum_{i=1}^N 1/n_i}{N - \sum_{i=1}^N 1/n_i}$$

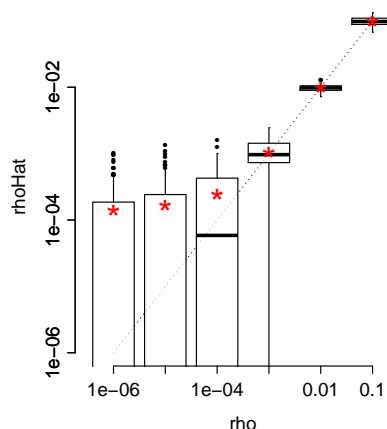
$$s^2 = \frac{N \sum_{i=1}^N n_i (\hat{\nu} - \hat{\mu}_i)^2}{(N - 1) \sum_{i=1}^N n_i}$$
(8)

This yields consistent estimates over a range of true values:

```

rho = 10^seq(-6,-1)
rhoHat <- sapply(rho, function(r){
  sapply(1:100, function(i){
    n = 100
    X = rbetabinom(1000, n, 0.01, rho=r)
    X = cbind(X, n-X)
    Y = array(X, dim=c(1000,1,2))
    deepSNV::estimateRho(Y, Y/n, Y < 1000)[1,1])
  })
})
par(bty="n", mgp = c(2,.5,0), mar=c(3,4,1,1)+.1, tcl=-.33)
plot(rho, type="l", log="y", xaxt="n", xlab="rho", ylab="rhoHat", xlim=c(0.5,6.5), lty=3)
boxplot(t(rhoHat+ 1e-7) ~ rho, add=TRUE, col="#FFFFFFAA", pch=16, cex=.5, lty=1, staplewex=0)
points(colMeans(rhoHat), pch="*", col="red", cex=2)

```



## Using a prior

shearwater calls variants if the posterior probability that the null model  $M_0$  is true falls below a certain threshold. Generally, the posterior odds is given by

$$\frac{\Pr(M_0 | D)}{\Pr(M_1 | D)} = \frac{1 - \pi(M_1)}{\pi(M_1)} \frac{\Pr(D | M_0)}{\Pr(D | M_1)} \quad (9)$$

where  $\pi = \pi(M_1)$  is the prior probability of that a variant exists. These probabilities are not uniform and may be calculated from the distribution of observed somatic mutations. Such data can be found in the COSMIC data base <http://www.sanger.ac.uk/cosmic>.

As of now, the amount of systematic, genome-wide screening data is still sparse, which makes it difficult to get good estimates of the mutation frequencies in each cancer type. However, a wealth of data exists for somatic mutations within a given gene. Assume we know how likely it is that a gene is mutated. We then model

$$\pi = \begin{cases} \pi_{\text{gene}} \times \frac{\# \text{ Mutations at given position}}{\# \text{ Mutations in gene}} & \text{if variant in COSMIC} \\ \pi_{\text{background}} & \text{else.} \end{cases} \quad (10)$$

Suppose you have downloaded the COSMIC vcf "CosmicCodingMuts\_v63\_300113.vcf.gz" from <ftp://ngs.sanger.ac.uk/production/cosmic>.

```

## Not run..
## Load TxDb
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
seqlevels(txdb) <- sub("chr", "", seqlevels(txdb))

## Make prior

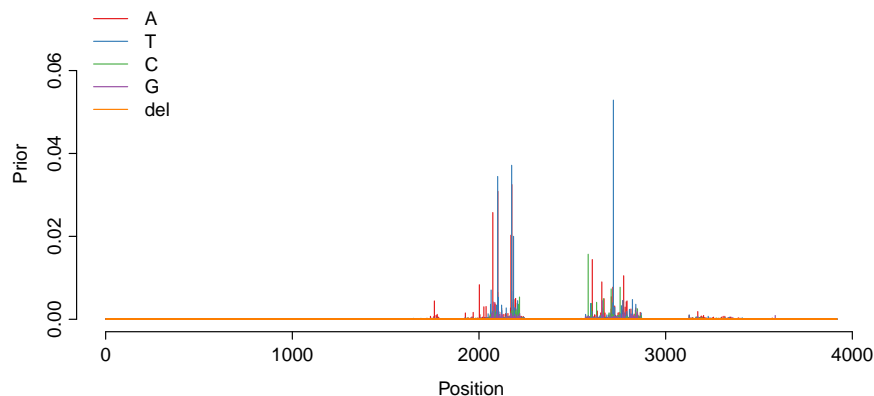
```

```
regions <- reduce(exons(txdb, filter=list(gene_id='7157'))) ## TP53 exons
cosmic <- readVcf("CosmicCodingMuts_v63_300113.vcf.gz", "hg19", param=ScanVcfParam(which=regions))
pi <- makePrior(cosmic, regions, pi.gene = 1)
```

The resulting prior can be visualised:

```
## Load pi
data(pi, package="deepSNV")

## Plot
par(bty="n", mgp = c(2,.5,0), mar=c(3,3,2,2)+.1, tcl=-.33)
plot(pi[,1], type="h", xlab="Position", ylab="Prior", col=brewer.pal(5,"Set1")[1], ylim=c(0,0.075))
for(j in 2:5)
  lines(pi[,j], type="h", col=brewer.pal(5,"Set1")[j])
legend("topleft", col=brewer.pal(5,"Set1"), lty=1, bty="n", c("A","T","C","G","del"))
```



The data shows that the distribution of somatic variants is highly non-uniform, with multiple mutation hotspots.

## Using shearwater

To run shearwater you need a collection of .bam files and the set of regions you want to analyse as a GRanges() object. Additionally, you may calculate a prior from a VCF file that you can download from <ftp://ngs.sanger.ac.uk/production/cosmic>.

### Minimal example

Here is a minimal example that uses two .bam files from the deepSNV package. The data is loaded into a large array using the loadAllData() function:

```
## Load data from deepSNV example
regions <- GRanges("B.FR.83.HXB2_LAI_IIIB_BRU_K034", IRanges(start = 3120, end=3140))
files <- c(system.file("extdata", "test.bam", package="deepSNV"), system.file("extdata", "control.ba
counts <- loadAllData(files, regions, q=10)
dim(counts)

## [1] 2 21 10
```

The dimension of counts for  $N$  samples, a total of  $L$  positions is  $N \times L \times 2|B|$ , where  $|B| = 5$  is the size of the alphabet  $B = \{A, T, C, G, -\}$  and the factor of 2 for the two strand orientations.

The Bayes factors can be computed with the bbb function:

```

## Run (bbb) computes the Bayes factor
bf <- bbb(counts, model = "OR", rho=1e-4)
dim(bf)

## [1] 2 21 5

vcf <- bf2Vcf(bf, counts, regions, cutoff = 0.5, samples = files, prior = 0.5, mvcf = TRUE)
show(vcf)

## class: CollapsedVCF
## dim: 8 2
## rowRanges(vcf):
## GRanges with 4 metadata columns: REF, ALT, QUAL, FILTER
## info(vcf):
## DataFrame with 4 columns: ER, PI, AF, LEN
## info(header(vcf)):
##      Number Type Description
## ER 1 Float Error rate
## PI 1 Float Prior
## AF 1 Float Allele frequency in cohort
## LEN 1 Float Length of the alt allele
## geno(vcf):
## SimpleList of length 8: GT, GQ, BF, VF, FW, BW, FD, BD
## geno(header(vcf)):
##      Number Type Description
## GT 1 String Genotype
## GQ 1 Integer Genotype Quality
## BF 1 Float Bayes factor
## VF 1 Float Variant frequency in sample
## FW 1 Integer Forward variant read count
## BW 1 Integer Backward variant read count
## FD 1 Integer Read Depth forward
## BD 1 Integer Read Depth backward

```

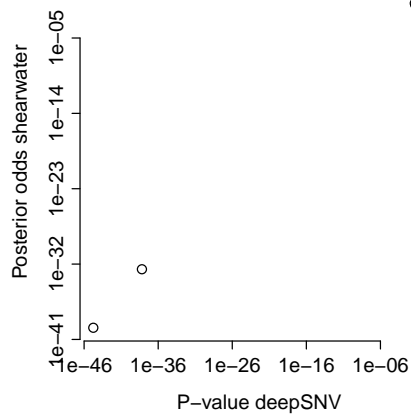
The resulting Bayes factors were thresholded by a posterior cutoff for variant calling and converted into a VCF object by `bf2Vcf`.

For two samples the Bayes factors are very similar to the p-values obtained by `deepSNV`:

```

## Shearwater Bayes factor under AND model
bf <- bbb(counts, model = "AND", rho=1e-4)
## deepSNV P-value with combine.method="fisher" (product)
dpSNV <- deepSNV(test = files[1], control = files[2], regions=regions, q=10, combine.method="fisher")
## Plot
par(bty="n", mgp = c(2,.5,0), mar=c(3,3,2,2)+.1, tcl=-.33)
plot(p.val(dpSNV), bf[1,,]/(1+bf[1,,]), log="xy",
      xlab = "P-value deepSNV",
      ylab = "Posterior odds shearwater"
)

```



## More realistic example

Suppose the bam files are in folder `./bam` and the regions of interest are stored in a `GRanges()` object with metadata column `Gene`, indicating which region (typically exons for a pulldown experiment) belongs to which gene. Also assume that we have a tabix indexed vcf file `CosmicCodingMuts_v63_300113.vcf.gz`. The analysis can be parallelized by separately analysing each gene, which is the unit needed to compute the prior using `makePrior`.

```
## Not run
files <- dir("bam", pattern="*.bam$", full.names=TRUE)
MC_CORES <- getOption("mc.cores", 2L)
vcfList <- list()
for(gene in levels(mcols(regions)$Gene)){
  rgn <- regions[mcols(regions)$Gene==gene]
  counts <- loadAllData(files, rgn, mc.cores=MC_CORES)
  ## Split into
  BF <- mcChunk("bbb", split = 200, counts, mc.cores=MC_CORES)
  COSMIC <- readVcf("CosmicCodingMuts_v63_300113.vcf.gz", "GRCh37", param=ScanVcfParam(which=))
  prior <- makePrior(COSMIC, rgn, pi.mut = 0.5)
  vcfList[[gene]] <- bf2Vcf(BF = BF, counts=counts, regions=rgn, samples = files, cutoff = 0.5)
}
## Collapse vcfList
vcf <- do.call(rbind, vcfList)
```

The `mcChunk` function splits the counts objects into chunks of size `split` and processes these in parallel using `mclapply`.

Instead of using a for loop one can also use a different mechanism, e.g. submitting this code to a computing cluster, etc.

## sessionInfo()

- R version 3.5.1 Patched (2018-07-12 r74967), x86\_64-pc-linux-gnu
- Locale: LC\_CTYPE=en\_US.UTF-8, LC\_NUMERIC=C, LC\_TIME=en\_US.UTF-8, LC\_COLLATE=C, LC\_MONETARY=en\_US.UTF-8, LC\_MESSAGES=en\_US.UTF-8, LC\_PAPER=en\_US.UTF-8, LC\_NAME=C, LC\_ADDRESS=C, LC\_TELEPHONE=C, LC\_MEASUREMENT=en\_US.UTF-8, LC\_IDENTIFICATION=C
- Running under: Ubuntu 16.04.5 LTS
- Matrix products: default
- BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
- LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so



- Base packages: base, datasets, grDevices, graphics, methods, parallel, splines, stats, stats4, utils
- Other packages: Biobase 2.42.0, BiocGenerics 0.28.0, BiocParallel 1.16.0, Biostrings 2.50.0, DelayedArray 0.8.0, GenomeInfoDb 1.18.0, GenomicRanges 1.34.0, IRanges 2.16.0, RColorBrewer 1.1-2, Rhtslib 1.14.0, Rsamtools 1.34.0, S4Vectors 0.20.0, SummarizedExperiment 1.12.0, VGAM 1.0-6, VariantAnnotation 1.28.0, XVector 0.22.0, deepSNV 1.28.0, knitr 1.20, matrixStats 0.54.0
- Loaded via a namespace (and not attached): AnnotationDbi 1.44.0, BSgenome 1.50.0, DBI 1.0.0, GenomeInfoDbData 1.2.0, GenomicAlignments 1.18.0, GenomicFeatures 1.34.0, Matrix 1.2-14, R6 2.3.0, RCurl 1.95-4.11, RSQLite 2.1.1, Rcpp 0.12.19, XML 3.98-1.16, assertthat 0.2.0, backports 1.1.2, biomaRt 2.38.0, bit 1.1-14, bit64 0.9-7, bitops 1.0-6, blob 1.1.1, compiler 3.5.1, crayon 1.3.4, digest 0.6.18, evaluate 0.12, grid 3.5.1, highr 0.7, hms 0.4.2, htmltools 0.3.6, httr 1.3.1, lattice 0.20-35, magrittr 1.5, memoise 1.1.0, pkgconfig 2.0.2, prettyunits 1.0.2, progress 1.2.0, rlang 0.3.0.1, rmarkdown 1.10, rprojroot 1.3-2, rtracklayer 1.42.0, stringi 1.2.4, stringr 1.3.1, tools 3.5.1, yaml 2.2.0, zlibbioc 1.28.0