Introduction to *VariantAnnotation*

Valerie Obenchain

November 30, 2016

**Contents**

1 Introduction 1

2 Variant Call Format (VCF) files 2
   2.1 Data import and exploration 2
   2.1.1 Header information 2
   2.1.2 Genomic positions 3
   2.1.3 Genotype data 4
   2.1.4 Info data 5
   2.2 Import data subsets 7
   2.2.1 Select genomic coordinates 7
   2.2.2 Select VCF fields 8

3 Locating variants in and around genes 9

4 Amino acid coding changes 10

5 SIFT and PolyPhen Databases 12

6 Other operations 13
   6.1 Create a SnpMatrix 13
   6.2 Write out VCF files 15

7 Performance 16

8 References 17

9 Session Information 17

1 Introduction

This vignette outlines a work flow for annotating and filtering genetic variants using the *VariantAnnotation* package. Sample data are in VariantCall Format (VCF) and are a subset of chromosome 22 from 1000 Genomes. VCF text files contain meta-information lines, a header line with column names, data lines with information about a position in the genome, and optional genotype information on samples for each position. The 1000 Genomes page describes the VCF format in detail.

Data are read in from a VCF file and variants identified according to region such as coding, intron, intergenic, spliceSite etc. Amino acid coding changes are computed for the non-synonymous variants and SIFT and PolyPhen databases provide predictions of how severely the coding changes affect protein function.
2 Variant Call Format (VCF) files

2.1 Data import and exploration

Data are parsed into a VCF object with readVcf.

```r
> library(VariantAnnotation)
> fl <- system.file("extdata", "chr22.vcf.gz", package="VariantAnnotation")
> vcf <- readVcf(fl, "hg19")
> vcf
```

class: CollapsedVCF
dim: 10376 5
rowRanges(vcf):
  GRanges with 5 metadata columns: paramRangeID, REF, ALT, QUAL, FILTER
info(vcf):
  DataFrame with 22 columns: LDAF, AVGPOST, RSQ, ERATE, THETA, CIEND...
info(header(vcf)):

```
Number Type Description
LDAF 1 Float MLE Allele Frequency Accounting for LD
AVGPOST 1 Float Average posterior probability from MaCH/...
RSQ 1 Float Genotype imputation quality from MaCH/Th...
ERATE 1 Float Per-marker Mutation rate from MaCH/Thunder
THETA 1 Float Per-marker Transition rate from MaCH/Thu...
CIEND 2 Integer Confidence interval around END for impre...
CIPO4 2 Integer Confidence interval around POS for impre...
END 1 Integer End position of the variant described in...
HOMLEN . Integer Length of base pair identical micro-homo...
HOMSEQ . String Sequence of base pair identical micro-ho...
SVLEN 1 Integer Difference in length between REF and ALT...
SVTYPE 1 String Type of structural variant
AC . Integer Alternate Allele Count
AN 1 Integer Total Allele Count
AA 1 String Ancestral Allele, ftp://ftp.1000genomes....
AF 1 Float Global Allele Frequency based on AC/AN
AMR_AF 1 Float Allele Frequency for samples from AMR ba...
ASN_AF 1 Float Allele Frequency for samples from ASN ba...
AFR_AF 1 Float Allele Frequency for samples from AFR ba...
EUR_AF 1 Float Allele Frequency for samples from EUR ba...
VT 1 String indicates what type of variant the line ...
SNPSOURCE . String indicates if a snp was called when analy...
geno(vcf)):
  SimpleList of length 3: GT, DS, GL
geno(header(vcf)):
  Number Type Description
  GT 1 String Genotype
  DS 1 Float Genotype dosage from MaCH/Thunder
  GL . Float Genotype Likelihoods
```

2.1.1 Header information

Header information can be extracted from the VCF with header(). We see there are 5 samples, 1 piece of meta information, 22 info fields and 3 geno fields.
> header(vcf)

class: VCFHeader
samples(5): HG00096 HG00097 HG00099 HG00100 HG00101
meta(1): META
fixed(1): ALT
info(22): LDAF AVGPOST ... VT SNPSOURCE
genom(3): GT DS GL 

Data can be further extracted using the named accessors.

> samples(header(vcf))
[1] "HG00096" "HG00097" "HG00099" "HG00100" "HG00101"

> geno(header(vcf))

DataFrame with 3 rows and 3 columns

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;character&gt;</td>
<td>&lt;character&gt;</td>
<td>&lt;character&gt;</td>
</tr>
<tr>
<td>GT</td>
<td>1</td>
<td>String Genotype</td>
</tr>
<tr>
<td>DS</td>
<td>1</td>
<td>Float Genotype dosage from MaCH/Thunder</td>
</tr>
<tr>
<td>GL</td>
<td>.</td>
<td>Float Genotype Likelihoods</td>
</tr>
</tbody>
</table>

2.1.2 Genomic positions

rowRanges contains information from the CHROM, POS, and ID fields of the VCF file, represented as a GRanges. The paramRangeID column is meaningful when reading subsets of data and is discussed further below.

> head(rowRanges(vcf), 3)

GRanges object with 3 ranges and 5 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges strand</th>
<th>paramRangeID</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;Rle&gt;</td>
<td>&lt;IRanges&gt;</td>
<td>&lt;factor&gt;</td>
</tr>
<tr>
<td>rs7410291</td>
<td>22 [50300078, 50300078] *</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>rs147922003</td>
<td>22 [50300086, 50300086] *</td>
<td>&lt;NA&gt;</td>
</tr>
<tr>
<td>rs114143073</td>
<td>22 [50300101, 50300101] *</td>
<td>&lt;NA&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REF</th>
<th>ALT</th>
<th>QUAL</th>
<th>FILTER</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DNAStringSet&gt;</td>
<td>&lt;DNAStringSetList&gt;</td>
<td>&lt;numeric&gt;</td>
<td>&lt;character&gt;</td>
</tr>
<tr>
<td>rs7410291</td>
<td>A</td>
<td>G</td>
<td>100</td>
</tr>
<tr>
<td>rs147922003</td>
<td>C</td>
<td>T</td>
<td>100</td>
</tr>
<tr>
<td>rs114143073</td>
<td>G</td>
<td>A</td>
<td>100</td>
</tr>
</tbody>
</table>

seqinfo: 1 sequence from hg19 genome; no seqlengths

Individual fields can be pulled out with named accessors. Here we see REF is stored as a DNAStringSet and qual is a numeric vector.

> ref(vcf)[1:5]

A DNAStringSet instance of length 5

width seq
[1] 1 A
[2] 1 C
[3] 1 G
[4] 1 C
[5] 1 C

> qual(vcf)[1:5]
ALT is a DNAStringSetList (allows for multiple alternate alleles per variant) or a DNAStringSet. When structural variants are present it will be a CharacterList.

2.1.3 Genotype data

Genotype data described in the FORMAT fields are parsed into the geno slot. The data are unique to each sample and each sample may have multiple values variable. Because of this, the data are parsed into matrices or arrays where the rows represent the variants and the columns the samples. Multidimensional arrays indicate multiple values per sample. In this file all variables are matrices.

DS is also known as 'posterior mean genotypes' and range in value from \([0, 2]\). To get a sense of variable distribution, we compute a five number summary of the minimum, lower-hinge (first quartile), median, upper-hinge (third quartile) and maximum.
The majority of these values (86%) are zero.

```r
> length(which(DS==0))/length(DS)
[1] 0.8621627
```

View the distribution of the non-zero values.

```r
> hist(DS[DS != 0], breaks=seq(0, 2, by=0.05),
+       main="DS non-zero values", xlab="DS")
```

2.1.4 Info data

In contrast to the genotype data, the info data are unique to the variant and the same across samples. All info variables are represented in a single DataFrame.

```r
> info(vcf)[1:4, 1:5]
```

```
DataFrame with 4 rows and 5 columns

<table>
<thead>
<tr>
<th></th>
<th>LDAF</th>
<th>AVGPOST</th>
<th>RSQ</th>
<th>ERATE</th>
<th>THETA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt;numeric&gt;</td>
<td>&lt;numeric&gt;</td>
<td>&lt;numeric&gt;</td>
<td>&lt;numeric&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
<tr>
<td>rs7410291</td>
<td>0.3431</td>
<td>0.9890</td>
<td>0.9856</td>
<td>2e-03</td>
<td>0.0005</td>
</tr>
</tbody>
</table>
```
We will use the info data to compare quality measures between novel (i.e., not in dbSNP) and known (i.e., in dbSNP) variants and the variant type present in the file. Variants with membership in dbSNP can be identified by using the appropriate SNPlocs package for hg19.

```r
> library(SNPlocs.Hsapiens.dbSNP.20101109)
> rd <- rowRanges(vcf)
> seqlevels(rd) <- "ch22"
> ch22snps <- getSNPlocs("ch22")
> dbsnpchr22 <- sub("rs", "", names(rd)) %in% ch22snps$RefSNP_id
> table(dbsnpchr22)
```

<table>
<thead>
<tr>
<th>dbsnpchr22</th>
<th>FALSE</th>
<th>TRUE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>6259</td>
<td>4117</td>
</tr>
</tbody>
</table>

Info variables of interest are 'VT', 'LDAF' and 'RSQ'. The header offers more details on these variables.

```r
> info(header(vcf))[c("VT", "LDAF", "RSQ"),]
```

DataFrame with 3 rows and 3 columns

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VT</td>
<td>String</td>
</tr>
<tr>
<td>LDAF</td>
<td>Float</td>
</tr>
<tr>
<td>RSQ</td>
<td>Float</td>
</tr>
</tbody>
</table>

Description

- **VT**: indicates what type of variant the line represents
- **LDAF**: MLE Allele Frequency Accounting for LD
- **RSQ**: Genotype imputation quality from MaCH/Thunder

Create a data frame of quality measures of interest ...

```r
> metrics <- data.frame(QUAL=qual(vcf), inDbSNP=dbsnpchr22, 
+ VT=info(vcf)$VT, LDAF=info(vcf)$LDAF, RSQ=info(vcf)$RSQ)
```

and visualize the distribution of qualities using ggplot2. For instance, genotype imputation quality is higher for the known variants in dbSNP.

```r
> library(ggplot2)
> ggplot(metrics, aes(x=RSQ, fill=inDbSNP)) +
+ geom_density(alpha=0.5) +
+ scale_x_continuous(name="MaCH / Thunder Imputation Quality") +
+ scale_y_continuous(name="Density") +
+ theme(legend.position="top")
```
2.2 Import data subsets

When working with large VCF files it may be more efficient to read in subsets of the data. This can be accomplished by selecting genomic coordinates (ranges) or by specific fields from the VCF file.

2.2.1 Select genomic coordinates

To read in a portion of chromosome 22, create a GRanges with the regions of interest.

```r
> rng <- GRanges(seqnames="22", ranges=IRanges(
+   start=c(50301422, 50989541),
+   end=c(50312106, 51001328),
+   names=c("gene_79087", "gene_644186"))
```

When ranges are specified, the VCF file must have an accompanying Tabix index file. See `?indexTabix` for help creating an index.

```r
> tab <- TabixFile(f1)
> vcf_rng <- readVcf(tab, "hg19", param=rng)
```
The `paramRangesID` column distinguishes which records came from which param range.

```r
> head(rowRanges(vcf_rng), 3)
```

GRanges object with 3 ranges and 5 metadata columns:

<table>
<thead>
<tr>
<th>seqnames</th>
<th>ranges</th>
<th>strand</th>
<th>paramRangeID</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs114335781</td>
<td>22 [50301422, 50301422]</td>
<td>*</td>
<td>gene_79087</td>
</tr>
<tr>
<td>rs8135963</td>
<td>22 [50301476, 50301476]</td>
<td>*</td>
<td>gene_79087</td>
</tr>
<tr>
<td>22:50301488_C/T</td>
<td>22 [50301488, 50301488]</td>
<td>*</td>
<td>gene_79087</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REF</th>
<th>ALT</th>
<th>QUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs114335781</td>
<td>G</td>
<td>A</td>
</tr>
<tr>
<td>rs8135963</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>22:50301488_C/T</td>
<td>C</td>
<td>T</td>
</tr>
</tbody>
</table>

FILTER

<table>
<thead>
<tr>
<th>&lt;character&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs114335781</td>
</tr>
<tr>
<td>rs8135963</td>
</tr>
<tr>
<td>22:50301488_C/T</td>
</tr>
</tbody>
</table>

---

seqinfo: 1 sequence from hg19 genome; no seqlengths

### 2.2.2 Select VCF fields

Data import can also be defined by the fixed, info and geno fields. Fields available for import are described in the header information. To view the header before reading in the data, use `ScanVcfHeader`.

```r
> hdr <- scanVcfHeader(f1)
> ## e.g., INFO and GENO fields
> head(info(hdr), 3)
```

DataFrame with 3 rows and 3 columns

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LDAF</td>
<td>1</td>
<td>MLE Allele Frequency Accounting for LD</td>
</tr>
<tr>
<td>AVGPOST</td>
<td>1</td>
<td>Average posterior probability from MaCH/Thunder</td>
</tr>
<tr>
<td>RSQ</td>
<td>1</td>
<td>Genotype imputation quality from MaCH/Thunder</td>
</tr>
</tbody>
</table>

```r
> head(geno(hdr), 3)
```

DataFrame with 3 rows and 3 columns

<table>
<thead>
<tr>
<th>Number</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GT</td>
<td>1</td>
<td>Genotype</td>
</tr>
<tr>
<td>DS</td>
<td>1</td>
<td>Genotype dosage from MaCH/Thunder</td>
</tr>
<tr>
<td>GL</td>
<td>.</td>
<td>Genotype Likelihoods</td>
</tr>
</tbody>
</table>

To subset on "LDAF" and "GT" we specify them as character vectors in the info and geno arguments to `ScanVcfParam`. This creates a `ScanVcfParam` object which is used as the param argument to `readVcf`. 
## Return all 'fixed' fields, "LAF" from 'info' and "GT" from 'geno'

```r
svp <- ScanVcfParam(info="LAF", geno="GT")
vcf1 <- readVcf(fl, "hg19", svp)
names(geno(vcf1))
```

[1] "GT"

To subset on both genomic coordinates and fields the ScanVcfParam object must contain both.

```r
svp_all <- ScanVcfParam(info="LAF", geno="GT", which=rng)
svp_all
```

class: ScanVcfParam
vcfWhich: 1 elements
vcfFixed: character() [All]
vcfInfo: LDAF
vcfGeno: GT
vcfSamples:

### Locating variants in and around genes

Variant location with respect to genes can be identified with the `locateVariants` function. Regions are specified in the `region` argument and can be one of the following constructors: CodingVariants, IntronVariants, FiveUTRVariants, ThreeUTRVariants, IntergenicVariants, SpliceSiteVariants or PromoterVariants. Location definitions are shown in Table 1.

<table>
<thead>
<tr>
<th>Location</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>coding</td>
<td>falls within a coding region</td>
</tr>
<tr>
<td>fiveUTR</td>
<td>falls within a 5’ untranslated region</td>
</tr>
<tr>
<td>threeUTR</td>
<td>falls within a 3’ untranslated region</td>
</tr>
<tr>
<td>intron</td>
<td>falls within an intron region</td>
</tr>
<tr>
<td>intergenic</td>
<td>does not fall within a transcript associated with a gene</td>
</tr>
<tr>
<td>spliceSite</td>
<td>overlaps any portion of the first 2 or last 2 nucleotides of an intron</td>
</tr>
<tr>
<td>promoter</td>
<td>falls within a promoter region of a transcript</td>
</tr>
</tbody>
</table>

Table 1: Variant locations

For overlap methods to work properly the chromosome names (seqlevels) must be compatible in the objects being compared. The VCF data chromosome names are represented by number, i.e., '22', but the TxDb chromosome names are preceded with 'chr'. Seqlevels in the VCF can be modified with the `seqlevels` function.

```r
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
seqlevels(vcf) <- "chr22"
rd <- rowRanges(vcf)
loc <- locateVariants(rd, txdb, CodingVariants())
head(loc, 3)
```

GRanges object with 3 ranges and 9 metadata columns:

| seqnames | ranges strand | LOCATION LOCSTART LOCEND QUERYID TXID CDSID GENEID PRECEDEID |
|----------|---------------|-----------|---------|---------|-------|-----|------|-------|
| chr22    | [50301422, 50301422] - | coding 939 939 |
| chr22    | [50301476, 50301476] - | coding 885 885 |
| chr22    | [50301488, 50301488] - | coding 873 873 |
Locate variants in all regions with the `AllVariants()` constructor,

```r
> allvar <- locateVariants(rd, txdb, AllVariants())
```

To answer gene-centric questions data can be summarized by gene regardless of transcript.

```r
> ## Did any coding variants match more than one gene?
> splt <- split(mcols(loc)$GENEID, mcols(loc)$QUERYID)
> table(sapply(splt, function(x) length(unique(x)) > 1))

FALSE  TRUE
965  15
```

```r
> ## Summarize the number of coding variants by gene ID.
> splt <- split(mcols(loc)$QUERYID, mcols(loc)$GENEID)
> head(sapply(splt, function(x) length(unique(x))), 3)

113730 1890 23209
```

4 Amino acid coding changes

`predictCoding` computes amino acid coding changes for non-synonymous variants. Only ranges in query that overlap with a coding region in the subject are considered. Reference sequences are retrieved from either a `BSgenome` or fasta file specified in `seqSource`. Variant sequences are constructed by substituting, inserting or deleting values in the `varAllele` column into the reference sequence. Amino acid codes are computed for the variant codon sequence when the length is a multiple of 3.

The `query` argument to `predictCoding` can be a `GRanges` or VCF. When a `GRanges` is supplied the `varAllele` argument must be specified. In the case of a VCF, the alternate alleles are taken from `alt(<VCF>)` and the `varAllele` argument is not specified.

The result is a modified query containing only variants that fall within coding regions. Each row represents a variant-transcript match so more than one row per original variant is possible.

```r
> library(BSgenome.Hsapiens.UCSC.hg19)
> coding <- predictCoding(vcf, txdb, seqSource=Hsapiens)
> coding[5:7]

GRanges object with 3 ranges and 17 metadata columns:

```
seqnames ranges strand | paramRangeID
<IRanges> <Rle> <Rle> | <factor>
22:50301584_C/T chr22 [50301584, 50301584] - | <NA>
rs114264124 chr22 [50302962, 50302962] - | <NA>
rs149209714 chr22 [50302995, 50302995] - | <NA>
```

<table>
<thead>
<tr>
<th>REF</th>
<th>ALT</th>
<th>QUAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;DNAStringSet&gt;</td>
<td>&lt;DNAStringSetList&gt;</td>
<td>&lt;numeric&gt;</td>
</tr>
</tbody>
</table>
Using variant rs114264124 as an example, we see varAllele A has been substituted into the refCodon CGG to produce varCodon CAG. The refCodon is the sequence of codons necessary to make the variant allele substitution and therefore often includes more nucleotides than indicated in the range (i.e. the range is 50302962, 50302962, width of 1). Notice it is the second position in the refCodon that has been substituted. This position in the codon, the position of substitution, corresponds to genomic position 50302962. This genomic position maps to position 698 in coding region-based coordinates and to triplet 233 in the protein. This is a non-synonymous coding variant where the amino acid has changed from R (Arg) to Q (Gln).

When the resulting varCodon is not a multiple of 3 it cannot be translated. The consequence is considered a frameshift and varAA will be missing.

> ## CONSEQUENCE is 'frameshift' where translation is not possible
> coding[mcols(coding)$CONSEQUENCE == "frameshift"]

GRanges object with 2 ranges and 17 metadata columns:

```
sequences ranges strand | <Rle> <IRanges> <Rle> |
22:50317001_G/GCACT chr22 [50317001, 50317001] + |
22:50317001_G/GCACT chr22 [50317001, 50317001] + |
```
5 SIFT and PolyPhen Databases

From predictCoding we identified the amino acid coding changes for the non-synonymous variants. For this subset we can retrieve predictions of how damaging these coding changes may be. SIFT (Sorting Intolerant From Tolerant) and PolyPhen (Polymorphism Phenotyping) are methods that predict the impact of amino acid substitution on a human protein. The SIFT method uses sequence homology and the physical properties of amino acids to make predictions about protein function. PolyPhen uses sequence-based features and structural information characterizing the substitution to make predictions about the structure and function of the protein.

Collated predictions for specific dbSNP builds are available as downloads from the SIFT and PolyPhen web sites. These results have been packaged into SIFT.Hsapiens.dbSNP132.db and PolyPhen.Hsapiens.dbSNP131.db and are designed to be searched by rsid. Variants that are in dbSNP can be searched with these database packages. When working with novel variants, SIFT and PolyPhen must be called directly. See references for home pages.

Identify the non-synonymous variants and obtain the rsids.

```r
> nms <- names(coding)
> idx <- mcols(coding)$CONSEQUENCE == "nonsynonymous"
> nonsyn <- coding[idx]
> names(nonsyn) <- nms[idx]
> rsids <- unique(names(nonsyn)[grep("rs", names(nonsyn), fixed=TRUE)])
```

Detailed descriptions of the database columns can be found with ?SIFTDbColumns and ?PolyPhenDbColumns. Variants in these databases often contain more than one row per variant. The variant may have been reported by multiple sources and therefore the source will differ as well as some of the other variables.

It is important to keep in mind the pre-computed predictions in the SIFT and PolyPhen packages are based on specific gene models. SIFT is based on Ensembl and PolyPhen on UCSC Known Gene. The TxDb we used to identify the coding snps was based on UCSC Known Gene so we will use PolyPhen for predictions. PolyPhen provides predictions using two different training datasets and has considerable information about 3D protein structure. See ?PolyPhenDbColumns or the PolyPhen web site listed in the references for more details.

Query the PolyPhen database,

```r
> library(PolyPhen.Hsapiens.dbSNP131)
> pp <- select(PolyPhen.Hsapiens.dbSNP131, keys=rsids,
+ cols=c("TRAININGSET", "PREDICTION", "PPH2PROB"))
> head(pp[!is.na(pp$PREDICTION), ])
```

<table>
<thead>
<tr>
<th>RSID</th>
<th>TRAININGSET</th>
<th>OSNPID</th>
<th>OACC</th>
<th>OPOS</th>
<th>OAA1</th>
<th>OAA2</th>
<th>SNPID</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>humdiv</td>
<td>rs8139422</td>
<td>Q6UXH1-5</td>
<td>182</td>
<td>D</td>
<td>E</td>
<td>rs8139422</td>
</tr>
<tr>
<td>14</td>
<td>humvar</td>
<td>rs8139422</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>rs8139422</td>
</tr>
<tr>
<td>15</td>
<td>humdiv</td>
<td>rs74510325</td>
<td>Q6UXH1-5</td>
<td>189</td>
<td>R</td>
<td>G</td>
<td>rs74510325</td>
</tr>
<tr>
<td>16</td>
<td>humvar</td>
<td>rs74510325</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>&lt;NA&gt;</td>
<td>rs74510325</td>
</tr>
</tbody>
</table>
6 Other operations

6.1 Create a SnpMatrix

The 'GT' element in the FORMAT field of the VCF represents the genotype. These data can be converted into a SnpMatrix object which can then be used with the functions offered in snpStats and other packages making use of the SnpMatrix class.
The `genotypeToSnpMatrix` function converts the genotype calls in `geno` to a `SnpMatrix`. No `dbSNP` package is used in this computation. The return value is a named list where 'genotypes' is a `SnpMatrix` and 'map' is a `DataFrame` with SNP names and alleles at each loci. The ignore column in 'map' indicates which variants were set to NA (missing) because they met one or more of the following criteria,

- variants with >1 ALT allele are set to NA
- only single nucleotide variants are included; others are set to NA
- only diploid calls are included; others are set to NA

See `?genotypeToSnpMatrix` for more details.

```r
> res <- genotypeToSnpMatrix(vcf)
> res

$genotypes
A SnpMatrix with 5 rows and 10376 columns
Row names: HG00096 ... HG00101
Col names: rs7410291 ... rs114526001

$map
DataFrame with 10376 rows and 4 columns

   snp.names allele.1 allele.2 ignore
     <character> <DNAStringSet> <DNAStringSetList> <logical>
  1  rs7410291 A   G     FALSE
  2  rs147922003 C   T     FALSE
  3  rs114143073 G   A     FALSE
  4  rs141778433 C   T     FALSE
  5  rs182170314 C   T     FALSE
   ... ... ...     ... ...
10372 rs187302552 A   G     FALSE
10373 rs9628178 A   G     FALSE
10374 rs5770892 A   G     FALSE
10375 rs144055359 G   A     FALSE
10376 rs114526001 G   C     FALSE

In the map `DataFrame`, allele.1 represents the reference allele and allele.2 is the alternate allele.

```r
> allele2 <- res$map[, "allele.2"]
> ## number of alternate alleles per variant
> unique(elementNROWS(allele2))

[1] 1
```

In addition to the called genotypes, genotype likelihoods or probabilities can also be converted to a `SnpMatrix`, using the `snpStats` encoding of posterior probabilities as byte values. To use the values in the 'GL' or 'GP' `FORMAT` field instead of the called genotypes, use the `uncertain=TRUE` option in `genotypeToSnpMatrix`.

```r
> fl.gl <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
> vcf.gl <- readVcf(fl.gl, "hg19")
> geno(vcf.gl)
List of length 3
names(3): GT DS GL

> ## Convert the "GL" FORMAT field to a SnpMatrix
> res <- genotypeToSnpMatrix(vcf.gl, uncertain=TRUE)
> res

$genotypes
A SnpMatrix with 85 rows and 9 columns
```
Row names: NA06984 ... NA12890
Col names: rs58108140 ... rs200430748

$map
DataFrame with 9 rows and 4 columns

<table>
<thead>
<tr>
<th>snp.names</th>
<th>allele.1</th>
<th>allele.2</th>
<th>ignore</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rs58108140</td>
<td>G</td>
<td>A</td>
<td>FALSE</td>
</tr>
<tr>
<td>rs189107123</td>
<td>C</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>rs180734498</td>
<td>C</td>
<td>T</td>
<td>FALSE</td>
</tr>
<tr>
<td>rs144762171</td>
<td>G</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>rs201747181</td>
<td>TC</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>rs151276478</td>
<td>T</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>rs140337953</td>
<td>G</td>
<td>T</td>
<td>FALSE</td>
</tr>
<tr>
<td>rs199681827</td>
<td>C</td>
<td></td>
<td>TRUE</td>
</tr>
<tr>
<td>rs200430748</td>
<td>G</td>
<td></td>
<td>TRUE</td>
</tr>
</tbody>
</table>

> t(as(res$genotype, "character"))[c(1,3,7), 1:5]

<table>
<thead>
<tr>
<th></th>
<th>NA06984</th>
<th>NA06986</th>
<th>NA06989</th>
<th>NA06994</th>
<th>NA07000</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs58108140</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/A&quot;</td>
</tr>
<tr>
<td>rs180734498</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;A/A&quot;</td>
<td>&quot;A/A&quot;</td>
</tr>
<tr>
<td>rs140337953</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;Uncertain&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/A&quot;</td>
</tr>
</tbody>
</table>

> ## Compare to a SnpMatrix created from the "GT" field
> res.gt <- genotypeToSnpMatrix(vcf.gl, uncertain=FALSE)
> t(as(res.gt$genotype, "character"))[c(1,3,7), 1:5]

<table>
<thead>
<tr>
<th></th>
<th>NA06984</th>
<th>NA06986</th>
<th>NA06989</th>
<th>NA06994</th>
<th>NA07000</th>
</tr>
</thead>
<tbody>
<tr>
<td>rs58108140</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/A&quot;</td>
<td>&quot;A/A&quot;</td>
</tr>
<tr>
<td>rs180734498</td>
<td>&quot;A/A&quot;</td>
<td>&quot;A/A&quot;</td>
<td>&quot;A/A&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;A/B&quot;</td>
</tr>
<tr>
<td>rs140337953</td>
<td>&quot;B/B&quot;</td>
<td>&quot;B/B&quot;</td>
<td>&quot;A/B&quot;</td>
<td>&quot;B/B&quot;</td>
<td>&quot;A/B&quot;</td>
</tr>
</tbody>
</table>

> ## What are the original likelihoods for rs58108140?
> geno(vcf.gl)$GL["rs58108140", 1:5]

$NA06984
[1] -4.70 -0.58 -0.13

$NA06986
[1] -1.15 -0.10 -0.84

$NA06989
[1] -2.05 0.00 -3.27

$NA06994
[1] -0.48 -0.48 -0.48

$NA07000
[1] -0.28 -0.44 -0.96

For variant rs58108140 in sample NA06989, the "A/B" genotype is much more likely than the others, so the SnpMatrix object displays the called genotype.

### 6.2 Write out VCF files

A VCF file can be written out from data stored in a VCF class.
> fl <- system.file("extdata", "ex2.vcf", package="VariantAnnotation")
> out1.vcf <- tempfile()
> out2.vcf <- tempfile()
> in1 <- readVcf(fl, "hg19")
> writeVcf(in1, out1.vcf)
> in2 <- readVcf(out1.vcf, "hg19")
> writeVcf(in2, out2.vcf)
> in3 <- readVcf(out2.vcf, "hg19")
> identical(rowRanges(in1), rowRanges(in3))
[1] TRUE
> identical(geno(in1), geno(in2))
[1] TRUE

### 7 Performance

Targeted queries can greatly improve the speed of data input. When all data from the file are needed define a `yieldSize` in the `TabixFile` to iterate through the file in chunks.

```r
readVcf(TabixFile(fl, yieldSize=10000))
```

`readVcf` can be used with a `ScanVcfParam` to select any combination of INFO and GENO fields, samples or genomic positions.

```r
readVcf(TabixFile(fl), param=ScanVcfParam(info='DP', geno='GT'))
```

While `readVcf` offers the flexibility to define combinations of INFO, GENO and samples in the `ScanVcfParam`, sometimes only a single field is needed. In this case the lightweight read functions (`readGT`, `readInfo` and `readGeno`) can be used. These functions return the single field as a matrix instead of a VCF object.

```r
readGT(fl)
```

The table below highlights the speed differences of targeted queries vs reading in all data. The test file is from 1000 Genomes and has 494328 variants, 1092 samples, 22 INFO, and 3 GENO fields and is located at ftp://ftp-trace.ncbi.nih.gov/1000genomes/ftp/release/20101123/.

`yieldSize` is used to define chunks of 100, 1000, 10000 and 100000 variants. For each chunk size three function calls are compared: `readGT` reading only GT, `readVcf` reading both GT and ALT and finally `readVcf` reading in all the data.

```r
library(microbenchmark)
fl <- "ALL.chr22.phase1_release_v3.20101123.snps_indels_svs.genotypes.vcf.gz"
y <- c(100, 1000, 10000, 100000)

## readGT() input only 'GT':
fun <- function(fl, yieldSize) readGT(TabixFile(fl, yieldSize))
lapply(y, function(i) microbenchmark(fun(fl, i), times=5))

## readVcf() input only 'GT' and 'ALT':
fun <- function(fl, yieldSize, param)
    readVcf(TabixFile(fl, yieldSize), "hg19", param=param)
param <- ScanVcfParam(info=NA, geno="GT", fixed="ALT")
lapply(y, function(i) microbenchmark(fun(fl, i, param), times=5))

## readVcf() input all variables:
fun <- function(fl, yieldSize) readVcf(TabixFile(fl, yieldSize), "hg19")
lapply(y, function(i) microbenchmark(fun(fl, i), times=5))
```
Introduction to VariantAnnotation

<table>
<thead>
<tr>
<th>n records</th>
<th>readGT</th>
<th>readVcf (GT and ALT)</th>
<th>readVcf (all)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>0.082</td>
<td>0.128</td>
<td>0.501</td>
</tr>
<tr>
<td>1000</td>
<td>0.609</td>
<td>0.508</td>
<td>5.878</td>
</tr>
<tr>
<td>10000</td>
<td>5.972</td>
<td>6.164</td>
<td>68.378</td>
</tr>
<tr>
<td>100000</td>
<td>78.593</td>
<td>81.156</td>
<td>693.654</td>
</tr>
</tbody>
</table>

Table 2: Targeted queries (time in seconds)

8 References


SIFT home page: http://sift.bii.a-star.edu.sg/

PolyPhen home page: http://genetics.bwh.harvard.edu/pph2/

9 Session Information

R version 3.3.2 (2016-10-31)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 16.04.1 LTS

locale:
[1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C
[3] LC_TIME=en_US.UTF-8 LC_COLLATE=C
[5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
[7] LC_PAPER=en_US.UTF-8 LC_NAME=C
[9] LC_ADDRESS=C LC_TELEPHONE=C

attached base packages:
[1] stats4  parallel  stats  graphics  grDevices  utils
[7] datasets  methods  base

other attached packages:
[1] snpStats_1.24.0
[2] Matrix_1.2-7.1
[3] survival_2.40-1
[4] PolyPhen.Hsapiens.dbSNP131_1.0.2
[5] RSQLite_1.1
[6] BSgenome.Hsapiens.UCSC.hg19_1.4.0
[7] BSgenome_1.42.0
[8] rtracklayer_1.34.1
[10] GenomicFeatures_1.26.0
[11] AnnotationDbi_1.36.0
Introduction to *VariantAnnotation*

[12] ggplot2_2.2.0
[13] SNPlocs_Hsapiens.dbSNP.20101109_0.99.7
[14] VariantAnnotation_1.20.2
[15] Rsamtools_1.26.1
[16] Biostrings_2.42.1
[17] XVector_0.14.0
[18] SummarizedExperiment_1.4.0
[19] Biobase_2.34.0
[21] GenomeInfoDb_1.10.1
[22] IRanges_2.8.1
[23] S4Vectors_0.12.1
[24] BiocGenerics_0.20.0

loaded via a namespace (and not attached):
[1] Rcpp_0.12.8      plyr_1.8.4
[3] bitops_1.0-6     tools_3.3.2
[5] zlibbioc_1.20.0   biomaRt_2.30.0
[7] digest_0.6.10    tibble_1.2
[9] memoise_1.0.0    gtable_0.2.0
[11] lattice_0.20-34  DBI_0.5-1
[13] grid_3.3.2       XML_3.98-1.5
[15] BiocParallel_1.8.1 splines_3.3.2
[17] scales_0.4.1     GenomicAlignments_1.10.0
[19] assertthat_0.1   BiocStyle_2.2.1
[21] colorspace_1.3-1 labeling_0.3
[23] RCurl_1.95-4.8   lazyeval_0.2.0
[25] munsell_0.4.3