

Introduction to *VariantAnnotation*

Valerie Obenchain

June 21, 2015

Contents

1	Introduction	1
2	Variant Call Format (VCF) files	2
2.1	Data import and exploration	2
2.1.1	Header information	3
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	14
6.1	Create a SnpMatrix	14
6.2	Write out VCF files	16
7	Performance	16
8	References	17
9	Session Information	18

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`.

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

found header lines for 3 fixed fields: ALT, QUAL, FILTER

found header lines for 22 info fields: LDAF, AVGPOST, RSQ, ERATE, THETA, CIEND, CIPOS, END, HOMLEN, HOMSEQ, SVLEN, SVTYPE, AC, AN, AA, AF, AMR_AF, ASN_AF, AFR_AF, EUR_AF, VT, SNPSOURCE

found header lines for 3 geno fields: GT, DS, GL

```
> 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...
CIPOS	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 AVGPST ... VT SNPSOURCE
geno(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
      Number      Type      Description
  <character> <character> <character>
GT          1      String      Genotype
DS          1      Float Genotype dosage from MaCH/Thunder
GL          .      Float      Genotype Likelihoods
```

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:
      seqnames      ranges strand | paramRangeID
      <Rle>        <IRanges> <Rle> | <factor>
rs7410291      22 [50300078, 50300078] * | <NA>
rs147922003    22 [50300086, 50300086] * | <NA>
rs114143073    22 [50300101, 50300101] * | <NA>
      REF          ALT      QUAL      FILTER
  <DNAStrngSet> <DNAStrngSetList> <numeric> <character>
rs7410291      A          G      100      PASS
rs147922003    C          T      100      PASS
rs114143073    G          A      100      PASS
```

```
-----
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 `DNAStrngSet` and `qual` is a numeric vector.

```
> ref(vcf)[1:5]

A DNAStrngSet 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]

[1] 100 100 100 100 100
```

ALT is a `DNASTringSetList` (allows for multiple alternate alleles per variant) or a `DNASTringSet`. When structural variants are present it will be a `CharacterList`.

```
> alt(vcf)[1:5]

DNASTringSetList of length 5
[[1]] G
[[2]] T
[[3]] A
[[4]] T
[[5]] T
```

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.

```
> geno(vcf)

List of length 3
names(3): GT DS GL

> sapply(geno(vcf), class)

      GT      DS      GL
"matrix" "matrix" "matrix"
```

Let's take a closer look at the genotype dosage (DS) variable. The header provides the variable definition and type.

```
> geno(header(vcf))["DS",]

DataFrame with 1 row and 3 columns
      Number      Type      Description
  <character> <character> <character>
DS           1      Float Genotype dosage from MaCH/Thunder
```

These data are stored as a 10376×5 matrix. Each of the five samples (columns) has a single value per variant location (row).

```
> DS <- geno(vcf)$DS
> dim(DS)

[1] 10376      5

> DS[1:3,]

      HG00096 HG00097 HG00099 HG00100 HG00101
rs7410291      0      0      1      0      0
rs147922003     0      0      0      0      0
rs114143073     0      0      0      0      0
```

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.

```
> fivenum(DS)
```

```
[1] 0 0 0 0 2
```

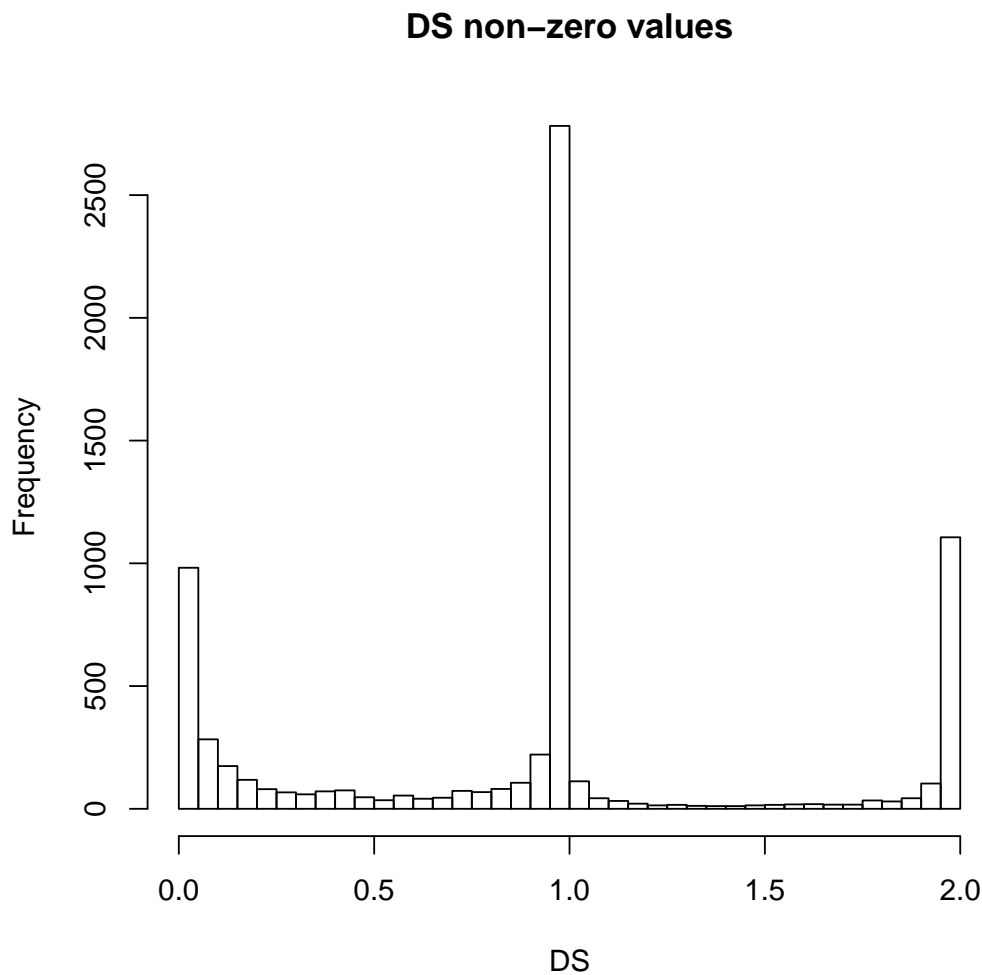
The majority of these values (86%) are zero.

```
> length(which(DS==0))/length(DS)
```

```
[1] 0.8621627
```

View the distribution of the non-zero values.

```
> 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`.

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

DataFrame with 4 rows and 5 columns

	LDAF	AVGPOST	RSQ	ERATE	THETA
	<numeric>	<numeric>	<numeric>	<numeric>	<numeric>
rs7410291	0.3431	0.9890	0.9856	2e-03	0.0005
rs147922003	0.0091	0.9963	0.8398	5e-04	0.0011
rs114143073	0.0098	0.9891	0.5919	7e-04	0.0008
rs141778433	0.0062	0.9950	0.6756	9e-04	0.0003

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.

```
> 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)
```

```
dbsnpchr22
FALSE TRUE
 6259 4117
```

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

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

DataFrame with 3 rows and 3 columns

	Number	Type
	<character>	<character>
VT	1	String
LDAF	1	Float
RSQ	1	Float

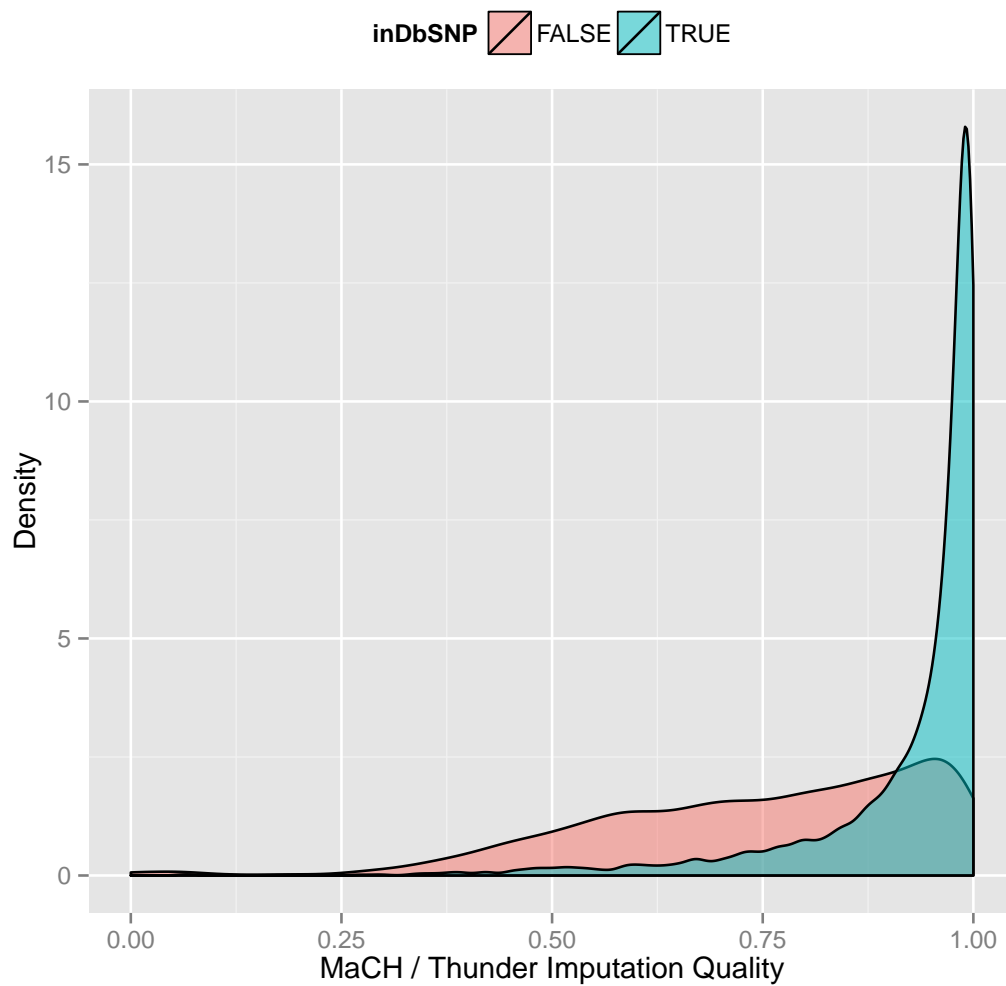
	Description
	<character>
VT	indicates what type of variant the line represents
LDAF	MLE Allele Frequency Accounting for LD
RSQ	Genotype imputation quality from MaCH/Thunder

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

```
> 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.

```
> 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.

```
> 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.

```
> tab <- TabixFile(fl)  
> vcf_rng <- readVcf(tab, "hg19", param=rng)
```

found header lines for 3 *fixed* fields: ALT, QUAL, FILTER

found header lines for 22 *info* fields: LDAF, AVGPOST, RSQ, ERATE, THETA, CIEND, CIPOS, END, HOMLEN, P

found header lines for 3 *geno* fields: GT, DS, GL

The paramRangesID column distinguishes which records came from which param range.

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

GRanges object with 3 ranges and 5 metadata columns:

seqnames	ranges	strand	paramRangeID
<Rle>	<IRanges>	<Rle>	<factor>
rs114335781	22 [50301422, 50301422]	*	gene_79087
rs8135963	22 [50301476, 50301476]	*	gene_79087
22:50301488_C/T	22 [50301488, 50301488]	*	gene_79087

REF	ALT	QUAL	
<DNASTringSet>	<DNASTringSetList>	<numeric>	
rs114335781	G	A	100
rs8135963	T	C	100
22:50301488_C/T	C	T	100

FILTER	
<character>	
rs114335781	PASS
rs8135963	PASS
22:50301488_C/T	PASS

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.

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

DataFrame with 3 rows and 3 columns

	Number	Type
	<character>	<character>
LDAF	1	Float
AVGPOST	1	Float
RSQ	1	Float

Description
<character>

LDAF	MLE Allele Frequency Accounting for LD
AVGPOST	Average posterior probability from MaCH/Thunder
RSQ	Genotype imputation quality from MaCH/Thunder

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

DataFrame with 3 rows and 3 columns

	Number	Type	Description
	<character>	<character>	<character>
GT	1	String	Genotype
DS	1	Float	Genotype dosage from MaCH/Thunder
GL	.	Float	Genotype Likelihoods

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'
> svp <- ScanVcfParam(info="LDAF", geno="GT")
> vcf1 <- readVcf(fl, "hg19", svp)

found header lines for 3 fixed fields: ALT, QUAL, FILTER
found header lines for 1 info fields: LDAF
found header lines for 1 geno fields: GT

> names(geno(vcf1))
[1] "GT"
```

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

```
> svp_all <- ScanVcfParam(info="LDAF", geno="GT", which=rng)
> svp_all

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

3 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.

Location	Details
<code>coding</code>	falls <i>within</i> a coding region
<code>fiveUTR</code>	falls <i>within</i> a 5' untranslated region
<code>threeUTR</code>	falls <i>within</i> a 3' untranslated region
<code>intron</code>	falls <i>within</i> an intron region
<code>intergenic</code>	does not fall <i>within</i> a transcript associated with a gene
<code>spliceSite</code>	overlaps any portion of the first 2 or last 2 nucleotides of an intron
<code>promoter</code>	falls <i>within</i> a promoter region of a transcript

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.

```
> 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
	<Rle>	<IRanges>	<Rle>	<factor>	<integer>	<integer>
1	chr22	[50301422, 50301422]	-	coding	939	939
2	chr22	[50301476, 50301476]	-	coding	885	885
3	chr22	[50301488, 50301488]	-	coding	873	873

	QUERYID	TXID	CDSID	GENEID	PRECEDEID
	<integer>	<character>	<IntegerList>	<character>	<CharacterList>
1	24	75253	218562	79087	
2	25	75253	218562	79087	
3	26	75253	218562	79087	

	FOLLOWID
	<CharacterList>
1	
2	
3	

seqinfo: 1 sequence from an unspecified genome; no seqlengths

Locate variants in all regions with the `AllVariants()` constructor,

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

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

```
> ## 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
```

```
> ## 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
    22    15     30
```

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.

```
> 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
	<Rle>	<IRanges>	<Rle>	<factor>
22:50301584_C/T	chr22	[50301584, 50301584]	-	<NA>
rs114264124	chr22	[50302962, 50302962]	-	<NA>
rs149209714	chr22	[50302995, 50302995]	-	<NA>
	REF	ALT	QUAL	
	<DNAStringSet>	<DNAStringSetList>	<numeric>	
22:50301584_C/T	C	T	100	
rs114264124	C	T	100	
rs149209714	C	G	100	
	FILTER	varAllele	CDSLOC	PROTEINLOC
	<character>	<DNAStringSet>	<IRanges>	<IntegerList>
22:50301584_C/T	PASS	A [777, 777]		259
rs114264124	PASS	A [698, 698]		233
rs149209714	PASS	C [665, 665]		222
	QUERYID	TXID	CDSID	GENEID
	<integer>	<character>	<IntegerList>	<character>
22:50301584_C/T	28	75253	218562	79087
rs114264124	57	75253	218563	79087
rs149209714	58	75253	218563	79087
	CONSEQUENCE	REFCODON	VARCODON	
	<factor>	<DNAStringSet>	<DNAStringSet>	
22:50301584_C/T	synonymous	CCG	CCA	
rs114264124	nonsynonymous	CGG	CAG	
rs149209714	nonsynonymous	GGA	GCA	
	REFAA	VARAA		
	<AAStringSet>	<AAStringSet>		
22:50301584_C/T	P	P		
rs114264124	R	Q		
rs149209714	G	A		

seqinfo: 1 sequence from hg19 genome; no seqlengths

Using variant rs114264124 as an example, we see varAllele A has been substituted into the refCodon CCG 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:

	seqnames	ranges	strand	paramRangeID
	<Rle>	<IRanges>	<Rle>	<factor>
22:50317001_G/GCACT	chr22	[50317001, 50317001]	+	<NA>
22:50317001_G/GCACT	chr22	[50317001, 50317001]	+	<NA>
	REF	ALT		
	<DNAStringSet>	<DNAStringSetList>		
22:50317001_G/GCACT	G	GCACT		

```

22:50317001_G/GCACT      <NA>      G      GCACT
                        QUAL      FILTER      varAllele      CDSLOC
                <numeric> <character> <DNAStrngSet> <IRanges>
22:50317001_G/GCACT      233      PASS      GCACT [808, 808]
22:50317001_G/GCACT      233      PASS      GCACT [628, 628]
                        PROTEINLOC  QUERYID      TXID      CDSID
                <IntegerList> <integer> <character> <IntegerList>
22:50317001_G/GCACT      270      359      74357      216303
22:50317001_G/GCACT      210      359      74358      216303
                        GENEID CONSEQUENCE      REFCODON
                <character> <factor> <DNAStrngSet>
22:50317001_G/GCACT      79174 frameshift      GCC
22:50317001_G/GCACT      79174 frameshift      GCC
                        VARCHODON      REFAA      VARAA
                <DNAStrngSet> <AAStringSet> <AAStringSet>
22:50317001_G/GCACT
22:50317001_G/GCACT
-----
seqinfo: 1 sequence from hg19 genome; no seqlengths

```

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.

```

> 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,

```

> library(PolyPhen.Hsapiens.dbSNP131)
> pp <- select(PolyPhen.Hsapiens.dbSNP131, keys=rsids,

```

```

+      cols=c("TRAININGSET", "PREDICTION", "PPH2PROB"))
> head(pp[!is.na(pp$PREDICTION), ])

```

	RSID	TRAININGSET	OSNPID	OACC	OPOS	OAA1	OAA2	SNPID
13	rs8139422	humdiv	rs8139422	Q6UXH1-5	182	D	E	rs8139422
14	rs8139422	humvar	rs8139422	<NA>	<NA>	<NA>	<NA>	rs8139422
15	rs74510325	humdiv	rs74510325	Q6UXH1-5	189	R	G	rs74510325
16	rs74510325	humvar	rs74510325	<NA>	<NA>	<NA>	<NA>	rs74510325
21	rs73891177	humdiv	rs73891177	Q6UXH1-5	207	P	A	rs73891177
22	rs73891177	humvar	rs73891177	<NA>	<NA>	<NA>	<NA>	rs73891177

	ACC	POS	AA1	AA2	NT1	NT2	PREDICTION	BASEDON	EFFECT
13	Q6UXH1-5	182	D	E	T	A	possibly damaging	alignment	<NA>
14	Q6UXH1-5	182	D	E	<NA>	<NA>	possibly damaging		<NA>
15	Q6UXH1-5	189	R	G	C	G	possibly damaging	alignment	<NA>
16	Q6UXH1-5	189	R	G	<NA>	<NA>	possibly damaging		<NA>
21	Q6UXH1-5	207	P	A	C	G	benign	alignment	<NA>
22	Q6UXH1-5	207	P	A	<NA>	<NA>	benign		<NA>

	PPH2CLASS	PPH2PROB	PPH2FPR	PPH2TPR	PPH2FDR	SITE	REGION	PHAT	DSCORE
13	neutral	0.228	0.156	0.892	0.258	<NA>	<NA>	<NA>	0.951
14	<NA>	0.249	0.341	0.874	<NA>	<NA>	<NA>	<NA>	<NA>
15	neutral	0.475	0.131	0.858	0.233	<NA>	<NA>	<NA>	1.198
16	<NA>	0.335	0.311	0.851	<NA>	<NA>	<NA>	<NA>	<NA>
21	neutral	0.001	0.86	0.994	0.61	<NA>	<NA>	<NA>	-0.225
22	<NA>	0.005	0.701	0.981	<NA>	<NA>	<NA>	<NA>	<NA>

	SCORE1	SCORE2	NOBS	NSTRUCT	NFILT	PDBID	PDBPOS	PDBCH	IDENT	LENGTH
13	1.382	0.431	37	0	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
14	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
15	1.338	0.14	36	0	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
16	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
21	-0.45	-0.225	1	0	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
22	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>

	NORMACC	SECSTR	MAPREG	DVOL	DPROP	BFACT	HBONDS	AVENHET	MINDHET
13	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
14	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
15	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
16	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
21	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
22	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>

	AVENINT	MINDINT	AVENSIT	MINDSIT	TRANSV	CODPOS	CPG	MINDJNC	PFAMHIT
13	<NA>	<NA>	<NA>	<NA>	<NA>	1	2	0	<NA>
14	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
15	<NA>	<NA>	<NA>	<NA>	<NA>	1	0	1	<NA>
16	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>
21	<NA>	<NA>	<NA>	<NA>	<NA>	1	0	0	<NA>
22	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>	<NA>

	IDPMAX	IDPSNP	IDQMIN	COMMENTS
13	18.261	18.261	48.507	chr22:50315363_CA
14	<NA>	<NA>	<NA>	chr22:50315363_CA
15	19.252	19.252	63.682	chr22:50315382_CG
16	<NA>	<NA>	<NA>	chr22:50315382_CG
21	1.919	<NA>	60.697	chr22:50315971_CG
22	<NA>	<NA>	<NA>	chr22:50315971_CG

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.

```
> 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.

```
> allele2 <- res$map[["allele.2"]]
> ## number of alternate alleles per variant
> unique(elementLengths(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*.

```
> fl.gl <- system.file("extdata", "gl_chr1.vcf", package="VariantAnnotation")
> vcf.gl <- readVcf(fl.gl, "hg19")
```

```

found header lines for 3 "fixed" fields: ALT, QUAL, FILTER
found header lines for 22 "info" fields: LDAF, AVGPOST, RSQ, ERATE, THETA, CIEND, CIPOS, END, HOMLEN,
found header lines for 3 "geno" fields: GT, DS, GL

> 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
  snp.names      allele.1      allele.2  ignore
<character> <DNAStringSet> <DNAStringSetList> <logical>
1 rs58108140      G      A      FALSE
2 rs189107123     C      TRUE
3 rs180734498     C      T      FALSE
4 rs144762171     G      TRUE
5 rs201747181     TC     TRUE
6 rs151276478     T      TRUE
7 rs140337953     G      T      FALSE
8 rs199681827     C      TRUE
9 rs200430748     G      TRUE

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

      NA06984  NA06986  NA06989  NA06994  NA07000
rs58108140 "Uncertain" "Uncertain" "A/B"      "Uncertain" "Uncertain"
rs180734498 "Uncertain" "Uncertain" "Uncertain" "Uncertain" "Uncertain"
rs140337953 "Uncertain" "Uncertain" "Uncertain" "Uncertain" "Uncertain"

> ## 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]

      NA06984 NA06986 NA06989 NA06994 NA07000
rs58108140 "A/B"  "A/B"  "A/B"  "A/A"  "A/A"
rs180734498 "A/B"  "A/A"  "A/A"  "A/A"  "A/B"
rs140337953 "B/B"  "B/B"  "A/B"  "B/B"  "A/B"

> ## 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")

found header lines for 3 fields: ALT, QUAL, FILTER
found header lines for 6 fields: NS, DP, AF, AA, DB, H2
found header lines for 4 fields: GT, GQ, DP, HQ

> writeVcf(in1, out1.vcf)
> in2 <- readVcf(out1.vcf, "hg19")

found header lines for 3 fields: ALT, QUAL, FILTER
found header lines for 6 fields: NS, DP, AF, AA, DB, H2
found header lines for 4 fields: GT, GQ, DP, HQ

> writeVcf(in2, out2.vcf)
> in3 <- readVcf(out2.vcf, "hg19")

found header lines for 3 fields: ALT, QUAL, FILTER
found header lines for 6 fields: NS, DP, AF, AA, DB, H2
found header lines for 4 fields: GT, GQ, DP, HQ

> 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.

```
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.

```
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.

```
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.

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

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

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

n records	readGT	readVcf (GT and ALT)	readVcf (all)
100	0.082	0.128	0.501
1000	0.609	0.508	5.878
10000	5.972	6.164	68.378
100000	78.593	81.156	693.654

Table 2: Targeted queries (time in seconds)

8 References

Wang K, Li M, Hakonarson H, (2010), ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. *Nucleic Acids Research*, Vol 38, No. 16, e164.

McLaren W, Pritchard B, RiosD, et. al., (2010), Deriving the consequences of genomic variants with the Ensembl API and SNP Effect Predictor. *Bioinformatics*, Vol. 26, No. 16, 2069-2070.

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

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

9 Session Information

R version 3.2.1 (2015-06-18)

Platform: x86_64-unknown-linux-gnu (64-bit)

Running under: Ubuntu 14.04.2 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
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
```

attached base packages:

```
[1] stats4      parallel  stats      graphics  grDevices  utils
[7] datasets   methods   base
```

other attached packages:

```
[1] snpStats_1.18.0
[2] Matrix_1.2-1
[3] survival_2.38-2
[4] PolyPhen.Hsapiens.dbSNP131_1.0.2
[5] RSQLite_1.0.0
[6] DBI_0.3.1
[7] BSgenome.Hsapiens.UCSC.hg19_1.4.0
[8] BSgenome_1.36.1
[9] rtracklayer_1.28.5
[10] TxDb.Hsapiens.UCSC.hg19.knownGene_3.1.2
[11] GenomicFeatures_1.20.1
[12] AnnotationDbi_1.30.1
[13] Biobase_2.28.0
[14] ggplot2_1.0.1
[15] SNPlocs.Hsapiens.dbSNP.20101109_0.99.6
[16] VariantAnnotation_1.14.4
[17] Rsamtools_1.20.4
[18] Biostrings_2.36.1
[19] XVector_0.8.0
[20] GenomicRanges_1.20.5
[21] GenomeInfoDb_1.4.1
[22] IRanges_2.2.4
[23] S4Vectors_0.6.0
[24] BiocGenerics_0.14.0
```

loaded via a namespace (and not attached):

```
[1] Rcpp_0.11.6          futile.logger_1.4.1
[3] plyr_1.8.3           bitops_1.0-6
[5] futile.options_1.0.0  tools_3.2.1
[7] zlibbioc_1.14.0      biomaRt_2.24.0
[9] digest_0.6.8         lattice_0.20-31
[11] gtable_0.1.2         proto_0.3-10
[13] stringr_1.0.0        grid_3.2.1
[15] XML_3.98-1.2         BiocParallel_1.2.5
```

```
[17] reshape2_1.4.1      lambda.r_1.1.7
[19] magrittr_1.5         splines_3.2.1
[21] MASS_7.3-41          scales_0.2.5
[23] GenomicAlignments_1.4.1 colorspace_1.2-6
[25] BiocStyle_1.6.0      labeling_0.3
[27] stringi_0.4-1        RCurl_1.95-4.6
[29] munsell_0.4.2
```