An Introduction to *VariantTools*

Michael Lawrence, Jeremiah Degenhardt

October 2, 2012

Contents

1 Introduction

This vignette outlines the basic usages of the *VariantTools*package and the general workflow for loading data, calling single sample variants and tumor-specific somatic mutations or other sample-specific variant types (eg RNA editing). Most of the functions operate on alignments (BAM files) or datasets of called variants. The user is expected to have already aligned the reads with a separate tool, e.g., GSNAP via *gmapR*.

2 Calling single-sample variants

2.1 Basic usage

For our example, we take paired-end RNA-seq alignments from two lung cancer cell lines from the same individual. H1993 is derived from a metastatis and H2073 is derived from the primary tumor.

Below, we call variants from a region around the p53 gene:

```
> library(VariantTools)
> bams <- LungCancerLines::LungCancerBamFiles()
> bam <- bams$H1993
> tally.param <- VariantTallyParam(gmapR::TP53Genome(),
+                                  readlen = 100L,
+                                  high_base_quality = 23L,
+                                  which = range(p53))
> called.variants <- callVariants(bam, tally.param)
```

In the above, we load the genome corresponding to the human p53 gene region and the H1993 BAM file (stripped down to the same region). We pass the BAM, genome, read length and quality cutoff to the `callVariants` workhorse. The read length is not strictly required, but it is necessary for one of the QA filters. The value given for the high base quality cutoff is appropriate for Sanger and Illumina 1.8 or above. By default, the high quality counts are used by the likelihood ratio test during calling.

The returned `called_variants` is a variant *GRanges*, in the same form as that returned by `bam_tally` in the *gmapR* package. Unsurprisingly, `callVariants` uses `bam_tally` internally to generate the per-nucleotide counts (pileup) from the BAM file. The result is then filtered to generate the variant calls. The *VCF* class holds similar information; however, we favor the simple tally *GRanges*, because it has a separate record for each ALT, at each position. *VCF*, the class and the file format, has a single record for a position, collapsing over multiple ALT alleles, and this is much less convenient for our purposes.

If we subset the variants by those in an actual p53 exon (not an intron), we find two: one with strong evidence for a homozygous mutation, and another with much weaker evidence (low coverage).

```
> subsetByOverlaps(called.variants, p53, ignore.strand = TRUE)
```

```
GRanges with 2 ranges and 20 metadata columns:
      seqnames             ranges strand |     location         ref
         <Rle>          <IRanges>  <Rle> |  <character> <character>
  [1]     TP53 [1012027, 1012027]      + | TP53:1012027           T
  [2]     TP53 [1013309, 1013309]      + | TP53:1013309           C
             alt   ncycles ncycles.ref     count count.ref count.total
      <character> <integer>   <integer> <integer> <integer>   <integer>
  [1]           C         2           0         2         0           2
  [2]           G       126           0       934         0         936
      high.quality high.quality.ref high.quality.total mean.quality
         <integer>        <integer>          <integer>    <numeric>
  [1]            2                0                  2     39.50000
```

```
[2]          889                0                889     36.26884
    mean.quality.ref count.pos count.pos.ref count.neg count.neg.ref
           <numeric> <integer>     <integer> <integer>     <integer>
[1]              <NA>         1             0         1             0
[2]              <NA>       409             0       525             0
    cycleCount.0.10 cycleCount.10.90 cycleCount.90.100
          <integer>        <integer>         <integer>
[1]                0                2                 0
[2]               58              800                76
---
seqlengths:
    TP53
 2025767
```

The next section goes into further detail on the process, including the specific filtering rules applied, and how one might, for example, tweak the parameters to avoid calling low-coverage variants, like the one above.

2.2 Step by step

The `callVariants` method for BAM files, introduced above, is a convenience wrapper that delegates to several low-level functions to perform each step of the variant calling process: generating the tallies, basic QA filtering and the actual variant calling. Calling these functions directly affords the user more control over the process and provides access to intermediate results, which is useful e.g. for diagnostics and for caching results. The workflow consists of three function calls that rely on argument defaults to achieve the same result as our call to `callVariants` above. Please see their man pages for the arguments available for customization.

The first step is to tally the variants from the BAM file. By default, this will return observed differences from the reference, excluding N calls and only counting reads above 13 in mapping quality (MAPQ) score. There are three cycle bins: the first 10 bases, the final 10 bases, and the stretch between them (these will be used in the QA step).

```
> raw.variants <- tallyVariants(bam, tally.param)
```

Next, basic QA filters are applied. These include a minimum read count (2) check, minimum unique cycle count (2) check, and Fisher Exact Test on the per-strand counts vs. reference for strand bias (p-value cutoff: 0.001). If there are at least three cycle bins in the tallies, at least one read must present the variant in an internal cycle bin. The intent is to ensure that we have sufficient data and that the data are not due to strand-specific nor cycle-specific artifacts.

```
> qa.variants <- qaVariants(raw.variants)
```

The final step is to actually call the variants. The `callVariants` function uses a binomial likelihood ratio test for this purpose. The ratio is $P(D|p = p_{lower})/P(D|p = p_{error})$, where $p_{lower} = 0.2$ is the assumed lowest variant frequency and $p_{error} = 0.001$ is the assumed error rate in the sequencing (default: 0.001).

```
> called.variants <- callVariants(qa.variants)
```

2.3 Diagnosing the filters

The calls to `qaVariants` and `callVariants` are essentially filtering the tallies, so it is important to know, especially when faced with a new dataset, the effect of each filter and the effect of the individual parameters on each filter.

The filters are implemented as modules and are stored in a *FilterRules* object from the *IRanges* package. We can create those filters directly and rely on some *FilterRules* utilities to diagnose the filtering process.

Here we construct the *FilterRules* that implements the `qaVariants` function. Again, we rely on the argument defaults to generate the same answer.

```
> qa.filters <- VariantQAFilters()
```

We can now ask for a summary of the filtering process, which gives the number of variants that pass each filter, separately and then combined:

```
> summary(qa.filters, raw.variants)

   <initial>      nonNRef   cycleCount fisherStrand     cycleBin
        3924         3924         1385         3852         3486
     <final>
        1281
```

Now we retrieve the variants that pass the filters:

```
> qa.variants <- subsetByFilter(raw.variants, qa.filters)
```

We could do the same, except modify a filter parameter, such as the p-value cutoff for the Fisher Exact Test for strand bias:

```
> qa.filters.custom <- VariantQAFilters(fisher.strand.p.value = 1e-4)
> summary(qa.filters.custom, raw.variants)

   <initial>      nonNRef   cycleCount fisherStrand     cycleBin
        3924         3924         1385         3876         3486
     <final>
        1305
```

To get a glance at the additional variants we are discarding compared to the previous cutoff, we can subset the filter sets down to the Fisher strand filter, evaluate the old and new filter, and compare the results:

```
> fs.original <- eval(qa.filters["fisherStrand"], raw.variants)
> fs.custom <- eval(qa.filters.custom["fisherStrand"], raw.variants)
> raw.variants[fs.original != fs.custom]

GRanges with 24 ranges and 20 metadata columns:
       seqnames             ranges strand   |     location         ref
          <Rle>          <IRanges>  <Rle>   |  <character> <character>
   [1]     TP53 [1010944, 1010944]      +   | TP53:1010944           T
   [2]     TP53 [1011428, 1011428]      +   | TP53:1011428           C
   [3]     TP53 [1011435, 1011435]      +   | TP53:1011435           A
   [4]     TP53 [1011467, 1011467]      +   | TP53:1011467           T
   [5]     TP53 [1012605, 1012605]      +   | TP53:1012605           T
   [6]     TP53 [1013712, 1013712]      +   | TP53:1013712           T
   [7]     TP53 [1013961, 1013961]      +   | TP53:1013961           T
   [8]     TP53 [1017881, 1017881]      +   | TP53:1017881           T
   [9]     TP53 [1017955, 1017955]      +   | TP53:1017955           T
   ...      ...                 ...    ... ...          ...         ...
  [16]     TP53 [1018524, 1018524]      +   | TP53:1018524           T
  [17]     TP53 [1018529, 1018529]      +   | TP53:1018529           T
  [18]     TP53 [1018669, 1018669]      +   | TP53:1018669           G
  [19]     TP53 [1018722, 1018722]      +   | TP53:1018722           G
  [20]     TP53 [1018738, 1018738]      +   | TP53:1018738           G
```

```
[21]     TP53 [1018754, 1018754]      +   | TP53:1018754          T
[22]     TP53 [1018807, 1018807]      +   | TP53:1018807          T
[23]     TP53 [1018843, 1018843]      +   | TP53:1018843          T
[24]     TP53 [1018963, 1018963]      +   | TP53:1018963          T
            alt   ncycles ncycles.ref     count count.ref
     <character> <integer>   <integer> <integer> <integer>
 [1]           G        16         112        23       629
 [2]           G         7         140         8       634
 [3]           C        13         136        41       596
 [4]           C         6         142        12       744
 [5]           G         6         124         8       778
 [6]           G         7         139         7       905
 [7]           C         6         134         9       699
 [8]           C         5          90         6       385
 [9]           C         6         130         7       778
 ...         ...       ...         ...       ...       ...
[16]           C         6         133         9      1002
[17]           C         4         128         9       959
[18]           T         9          93        14       712
[19]           A         4          89         4       605
[20]           T         3          90         4       687
[21]           G         3          88         5       705
[22]           C         8          93        18       417
[23]           G        10         108        14       509
[24]           G         9         117        11       563
     count.total high.quality high.quality.ref high.quality.total
       <integer>    <integer>        <integer>          <integer>
 [1]         653            0              416                416
 [2]         643            0              493                493
 [3]         638            1              454                455
 [4]         758            0              621                621
 [5]         788            0              668                668
 [6]         913            0              836                836
 [7]         711            0              593                593
 [8]         393            0              327                327
 [9]         792            0              712                712
 ...         ...          ...              ...                ...
[16]        1013            0              842                842
[17]         972            0              828                828
[18]         728            0              577                577
[19]         611            0              543                543
[20]         691            0              657                657
[21]         713            0              662                662
[22]         436            1              326                327
[23]         525            0              437                437
[24]         577            0              473                473
     mean.quality mean.quality.ref count.pos count.pos.ref count.neg
        <numeric>        <numeric> <integer>     <integer> <integer>
 [1]         <NA>         32.82692        23           444         0
 [2]         <NA>         34.63692         8           258         0
 [3]           27         32.52423         7           270        34
```

```
  [4]         <NA>         34.83575         0           380        12
  [5]         <NA>         36.20060         8           317         0
  [6]         <NA>         35.49282         7           331         0
  [7]         <NA>         34.29174         0           379         9
  [8]         <NA>         35.21407         0           290         6
  [9]         <NA>         36.63062         0           526         7
  ...          ...              ...       ...           ...       ...
 [16]         <NA>         34.66627         0           554         9
 [17]         <NA>         34.34058         0           526         9
 [18]         <NA>         35.87868         0           305        14
 [19]         <NA>         36.68877         4            80         0
 [20]         <NA>         38.35312         4           112         0
 [21]         <NA>         36.18580         5           117         0
 [22]           37         33.53374         1           212        17
 [23]         <NA>         35.12815        14           287         0
 [24]         <NA>         34.08457        11           285         0
      count.neg.ref cycleCount.0.10 cycleCount.10.90 cycleCount.90.100
          <integer>       <integer>        <integer>         <integer>
  [1]           185               0               19                 4
  [2]           376               0                7                 1
  [3]           326               0               19                22
  [4]           364               0                4                 8
  [5]           461               0                6                 2
  [6]           574               0                6                 1
  [7]           320               0                9                 0
  [8]            95               0                4                 2
  [9]           252               0                6                 1
  ...           ...             ...              ...               ...
 [16]           448               0                4                 5
 [17]           433               0                9                 0
 [18]           407               0                5                 9
 [19]           525               0                3                 1
 [20]           575               0                1                 3
 [21]           588               0                5                 0
 [22]           205               1               17                 0
 [23]           222               0               14                 0
 [24]           278               0               11                 0
 ---
 seqlengths:
     TP53
  2025767
```

We can also manipulate the filters that call the variants that have already passed the basic QA checks.

```
> calling.filters <- VariantCallingFilters()
> summary(calling.filters, qa.variants)

      <initial>       readCount likelihoodRatio         <final>
           1281              75              23              20
```

2.4 Extending and customizing the workflow

Since the built-in filters are implemented using *FilterRules*, it is easy to mix and match different filters, including those implemented externally to the *VariantTools* package. This is the primary means of extending and customizing the variant calling workflow.

3 Exporting the calls as VCF

VCF is a common file format for communicating variants. To export our variants to a VCF file, we first need to coerce the *GRanges* to a *VCF* object. Then, we use `writeVcf` from the *VariantAnnotation* package to write the file (indexing is highly recommended for large files).

```
> vcf <- variantGR2Vcf(called.variants, sample.id = "H1993",
+                      project = "VariantTools_Vignette")

> writeVcf(vcf, "H1993.vcf", index = TRUE)
```