BCGES short courses, session 4: Multi-sample calling, VCFtools and variant annotation

Contents

1 Multi-sample variant calling (about 1h30)

Multi-sample calling has several advantages:

- Borrow information from well covered samples to identify indels and other small variants, in turn improving the calling for other samples.

- Uniform calls across samples, which improves sample comparability.

- Computational improvements associated with the joint management of potentially large collection of samples.

Both `GATK` and `samtools` provide options for multi-sample calling and we will explore how these scripts are setup today.

1.1 Installing the reference fasta file (5 minutes)

As far as I can tell, and unlike `samtools`, `GATK` requires that the fasta file matches the sequence dictionary as specified in the header of the BAM file.

Hence a required element for variant calling with human data is the reference sequence in fasta format. It is also necessary to index this sequence in order to provide a "dictionary" of what sequences are present, and how long they are. This file is large and is therefore not on the `github` of the class. It can be downloaded using a `ftp` site and the `wget` utility

```
wget ftp://ftp.1000genomes.ebi.ac.uk/vol1/ftp/technical/reference/human_g1k_v37.fasta.gz
gunzip human_g1k_v37.fasta.gz
samtools faidx human_g1k_v37.fasta
```

These three steps download the compressed reference, unzip it and generate an index for that file.

1.2 Samtools `pileup` and `mpileup` (30 minutes)

`samtools` is less feature-rich that `GATK` but it remains a dependable, solid piece of software that can provide very reliable callsets. It has been used for large scale projects like UK10K, and there is no reason to doubt the quality of these calls. Calling samples requires the presence of the reference sequence, in fasta format. The compressed file `chr21.fasta` is available on the github site to make it easy. It is however necessary to uncompress this file using `gunzip`.

Once this is done, the script `multi_sample_calling_samtools.sh` describes the steps that generate a VCF file using `samtools`.
Exercise: Have a look at the scripts and make sure you can run them (in the simplest case just calling `sh script_name.sh` might do). They may need minor tweaking to point to the appropriate locations. It is important to understand all the options that are used.

Now we can look at the output and make sure we understand what is there. For example, using a simple `awk` script, let us look at position 21:43847096:

```
awk '{if ( ($2 == "POS") || ($2 == 43847096)) print $1,$2,$4,$5,$9, $10, $15}' \
            results/UBASH31A_samtools.vcf

## #CHROM POS REF ALT FORMAT HG00250 HG00255
## 21 43847096 G A GT:PL:DP:GQ 0/1:119,0,54:9:70 0/0:0,27,169:9:37
```

Make sure that you understand the meaning of all the fields in the output vcf file. Of note, you can see the total depth for each sample in the output VCF file, but I do not think that an option exists in `samtools` to show the depth per allele, unlike GATK (see below).

1.3 GATK v2.0 Unified Genotyper (30 minutes)

The GATK Unified Genotyped (UG) has been the standard approach for GATK for a while, and still delivers high quality and reliable calls. For reasons that will be outlined in the next subsection, GATK has now transitioned toward a more powerful system called HaplotypeCaller.But there is nothing wrong with still using the UG module of GATK, and this subsection shows how to run such tools in the context of multi-sample calling.

The Unified Genotyper (UG) version of the GATK pipeline is implemented in the script called `multi_sample_calling_GATK_UG.sh`. The key output of this script is the file `results/GATK_msamplecalling.vcf`. Note that GATK reorders the samples by alphabetical order, so the order may not match what was specified in the input BAM list, and it may also not match the VCF output of `samtools`.

Exercise: As for the `samtools` subsection, make sure you can run these scripts, tweak them a bit and understand the options that I used. Don't hesitate to look at the manual page for the UG tool to help you out.

As before for `samtools`, we can look at the output VCF file and make sure we understand what is shown:

```
awk '{if ( ($2 == "POS") || ($2 == 43847096)) print $1,$2,$4,$5, $9, $11, $16}' \
          results/GATK_msamplecalling.vcf

## #CHROM POS REF ALT FORMAT HG00250 HG00255
## 21 43847096 G A GT:AD:DP:GQ:PL 0/1:3,6:9:78:178,0,78 0/0:10,0:10:27:0,27,358
```

Exercise: Make sure you fully understand the output from GATK. It differs from GATK in terms of what is shown, but there are also differences in terms of content (look for example at the sequencing depth for HG00255 at this locus). In general, can you make sense of why the read depth would differ? Can you find a GATK update related to this point in the release notes of the version 3.2? Observe also that the genotype likelihoods also differ significantly, even with the same input data.

1.4 GATK v3.0 HaplotypeCaller version of multi-sample calling (30 minutes)

The issue with the UG approach to large cohort is, to a large extent, computational.

2 VCFtools (about 45 minutes)

VCFtools is a swiss army type tool very handy for VCF manipulations. It contains a suite of Perl scripts, which we won't use today, as well as a binary executable, which will be the focus of this section. The documentation for the binary executable is located here and here.

The first thing I propose to do is to compare the output of the GATK v2.0 output (Unified Genotyper, or UG) with the output from `samtools`.

```
vcftools --vcf results/UBASH31A_samtools.vcf  --freq --out results/samtools
vcftools --vcf results/GATK_msamplecalling.vcf --freq --out results/GATK_UG
```

Note that from this output it is easy to compare the number of calls for both algorithms.

```
wc -l samtools.frq GATK_UG.frq

##  135 samtools.frq
##  112 GATK_UG.frq
##  247 total
```

So where does the difference come from? A first thing is the fact that the call to GATK used a minimum quality filter of 30, which was not applied to the `samtools` output. `VCFtools` allows you to recode VCF files after having applied relevant filters. Try for example:

```
vcftools --vcf results/UBASH31A_samtools.vcf --minQ 30 --out results/samtools_filtered --recode
```

Note the use of the `recode` argument. Without this, no new VCF file will be created. I suggest you try, and you will see nothing but a log file.

Exercise: Using the VCF output from `VCFtools`, create a new file with the frequency values for each site. Is the result more consistent with the output from GATK UG in terms of number of polymorphic sites? See how you could have created the same frequency file without the intermediate VCF file `results/samtools_filtered.recode.vcf`. Make sure that this "direct" frequency file is identical to the one that used the intermediate VCF file.

```
vcftools --vcf results/samtools_filtered.recode.vcf --freq --out results/freq_from_intermediate.vcf
vcftools --vcf results/UBASH31A_samtools.vcf --minQ 30 --freq --out results/freq_direct_from_samtool
```

Optional exercise: For the subset of variants present in both `samtools` and GATK UG output, generate a graph to compare the estimated variant frequencies (using `R` most likely, but feel free to use something else if you prefer).
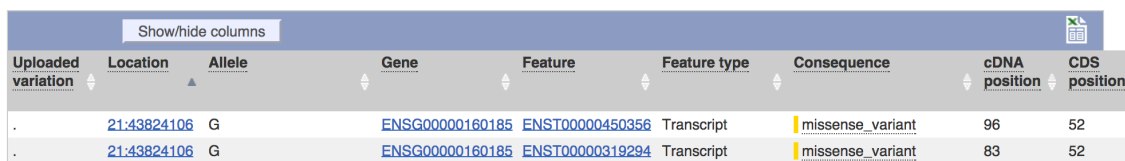
3 Variant annotation (about 45 minutes)

3.1 Running the variant effect predictor (VEP)

After having called variants, the next step consists of annotating them. A widely used and reliable tool is the Variant Effect Predictor (VEP), which is part of the tools provided by Ensembl. The simplest way to interact with the VEP is the web based interface. One possible input format, and probably the most widely used, is VCF. There is no need to copy all the fields, only the first 8 will be sufficient. For example you could use:

```
awk '{if ($1 == 21) print $1,$2,$3,$4,$5,$6,$7,$8}' \
         results/UBASH31A_samtools.vcf > results/samtools_for_VEP.vcf
```

Exercise: Note that there is a trap here. As the name indicates, these sequencing reads are supposed to cover the gene *UBASH31A*, however, none of the annotated variants is actually located in that gene. Can you figure out what went wrong? Fix it and download the output of the VEP in `csv` format (small excel like icon on the right side of the blue banner (see Figure 1).

Uploaded variation	Location	Allele	Gene	Feature	Feature type	Consequence	cDNA position	CDS position
.	21:43824106	G	ENSG00000160185	ENST00000450356	Transcript	missense_variant	96	52
.	21:43824106	G	ENSG00000160185	ENST00000319294	Transcript	missense_variant	83	52

Figure 1: Small icon on the right side is what you need to download the output in csv format
.

Running the VEP directly from the web is a very practical thing to do but it is not effective for very large amount of data (tens of thousand of variants...). It is possible to run it directly from your computer. If you do that, the first time you run the VEP, it will download locally all the files that you need. The installation has been stremalined extensively by the Ensembl team and the `INSTALL.pl` will fetch and install the minimum set of tools required for the VEP to run.

```
wget -O VEP.zip https://github.com/Ensembl/ensembl-tools/archive/release/76.zip
unzip VEP.zip
cd ensembl-tools-release-76/scripts/variant_effect_predictor
perl INSTALL.pl
```

You can read more about this process at this location.
Optional Exercise: Install the VEP locally, including the cached file for the NCBI build 37 of the human genome. Run the VEP against the VCF files that was generated by `samtools`. Note that this step can be time consuming, and you may hit small technical issues on these computers mostly set up for teaching, but it should be possible to make these scripts work on most linux installs.

3.2 Interpreting the output of the VEP

You should have now downloaded the output of the VEP in csv format, but in case that is not the case my output is available (`samtools_output_VEP.csv`). `R` is probably the best tool to read a csv file like the one generated by the VEP.
Exercise: Using `R`, load the output of the VEP. Look at the number of lines and understand the logic of the output, in particular why the number of lines exceeds the number of variants that were provided as input.

3.3 ANNOVAR as an alternative

Another software widely used is ANNOVAR. It is also written in perl, and you can find all the details here. In class we will discuss the advantages and weaknessed of ANNOVAR compared to the VEP. They

do deliver broadly comparable results, but the main difference comes from differences in the underlying databases (i.e. GENCODE, Ensembl, UCSC...).