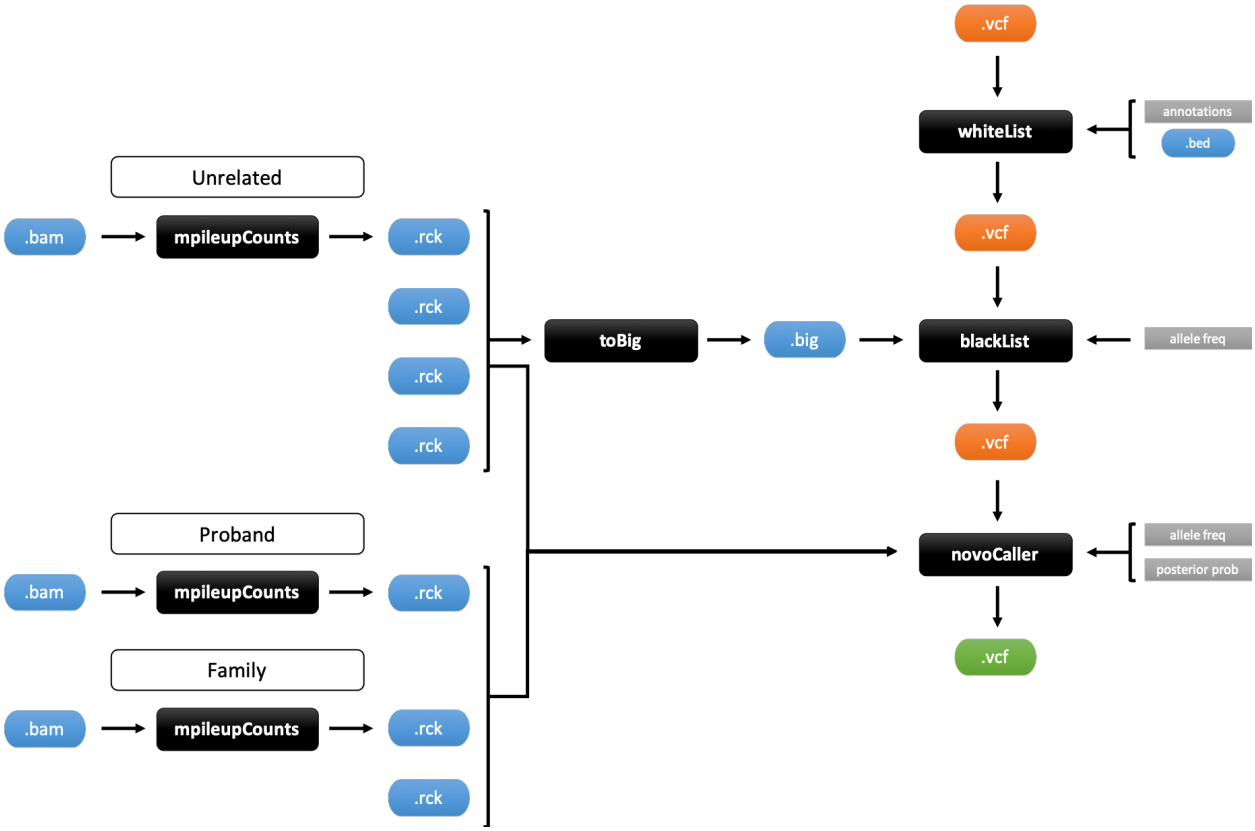
granite-suite

Release 0.2.0

Mar 09, 2023

Contents

granite is a collection of software to work with genomic variants. The suite provides inheritance mode callers and utilities to filter and refine variants called by other methods in VCF format.



Contents:

CHAPTER 1

Install

A ready-to-use docker image is available to download.

```
docker pull b3rse/granite:<version>
```

To run locally, Python 3.6+ is required together with the following libraries:

- *numpy*
- *pysam*
- *bitarray*
- *pytabix*
- *h5py*
- *matplotlib*

Additional software needs to be available in the environment:

- *samtools*
- *bgzip*
- *tabix*

To install the program from source:

```
git clone https://github.com/dbmi-bgm/granite
cd granite
make configure && make build
```

To install the program with pip:

```
pip install granite-suite
```

CHAPTER 2

File Formats

The program is compatible with standard BED, BAM and VCF formats (VCFv4.x).

2.1 ReadCountKeeper (.rck)

RCK is a tabular format that allows to efficiently store counts by strand (ForWard-ReVerse) for reads that support REFerence allele, ALTernate alleles, INSertions or DELetions at CHRomosome and POSition. RCK files can be further compressed with *bgzip* and indexed with *tabix* for storage, portability and faster random access. 1-based.

Tabular format structure:

```
#CHR    POS    COVERAGE    REF_FW    REF_RV    ALT_FW    ALT_RV    INS_FW    INS_RV    DEL_FW␣
↪ DEL_REV
13      1      23          0         0         11        12        0         0         0␣
↪ 0
13      2      35          18        15        1         1         0         0         0␣
↪ 0
```

Commands to compress and index files:

```
    bgzip PATH/TO/FILE
    tabix -b 2 -s 1 -e 0 -c "#" PATH/TO/FILE.gz
```

2.2 BinaryIndexGenome (.big)

BIG is a hdf5-based binary format that stores boolean values for each genomic position as bit arrays. Each position is represented in three complementary arrays that account for SNVs (Single-Nucleotide Variants), insertions and deletions respectively. 1-based.

hdf5 format structure:

```
e.g.
chr1_snv: array(bool)
chr1_ins: array(bool)
chr1_del: array(bool)
chr2_snv: array(bool)
...
...
chrM_del: array(bool)
```

note: hdf5 keys are built as the chromosome name based on reference (e.g. chr1) plus the suffix specifying whether the array represents SNVs (_snv), insertions (_ins) or deletions (_del).

2.3 Pedigree in JSON format

When the program requires pedigree information, the expected format is as follow:

```
[
  {
    "individual": "NA12877",
    "sample_name": "NA12877_sample",
    "gender": "M",
    "parents": []
  },
  {
    "individual": "NA12878",
    "sample_name": "NA12878_sample",
    "gender": "F",
    "parents": []
  },
  {
    "individual": "NA12879",
    "sample_name": "NA12879_sample",
    "gender": "F",
    "parents": ["NA12878", "NA12877"]
  }
]
```

where `individual` is the unique identifier for member inside the pedigree, `sample_name` is the corresponding sample ID in VCF file, and `parents` is the list of unique identifiers for member parents if any.

CHAPTER 3

Inheritance Mode Callers

3.1 novoCaller

novoCaller is a Bayesian calling algorithm for *de novo* mutations. The model uses read-level information both in pedigree (trio) and unrelated samples to rank and assign a probabilty to each call. The software represents an updated and improved implementation of the original algorithm described in Mohanty et al. 2019.

3.1.1 Arguments

```
usage: granite novoCaller [-h] -i INPUTFILE -o OUTPUTFILE -u UNRELATEDFILES -t
                          TRIOFILES [--ppthr PPTHR] [--afthr AFTHR]
                          [--aftag AFTAG] [--bam] [--MQthr MQTHR]
                          [--BQthr BQTHR] [--ADthr ADTHR]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  -u UNRELATEDFILES, --unrelatedfiles UNRELATEDFILES
                        TSV index file containing SampleID<TAB>Path/to/file
                        for unrelated files used to train the model (BAM or
                        bgzip and tabix indexed RCK)
  -t TRIOFILES, --triofiles TRIOFILES
                        TSV index file containing SampleID<TAB>Path/to/file
                        for family files, the PROBAND must be listed as FIRST
                        (BAM or bgzip and tabix indexed RCK)
  --ppthr PPTHR         threshold to filter by posterior probabilty for de
                        novo calls (>=) [0]
  --afthr AFTHR         threshold to filter by population allele frequency
                        (<=) [1]
```

(continues on next page)

(continued from previous page)

```
      --aftag AFTAG         TAG (TAG=<float>) or TAG field to be used to filter by
                            population allele frequency
      --bam                 by default the program expect bgzip and tabix indexed
                            RCK files for "--triofiles" and "--unrelatedfiles",
                            add this flag if files are in BAM format instead
                            (SLOWER)
      --MQthr MQTHR         (only with "--bam") minimum mapping quality for an
                            alignment to be used (>=) [0]
      --BQthr BQTHR         (only with "--bam") minimum base quality for a base to
                            be considered (>=) [0]
      --ADthr ADTHR         threshold to filter by alternate allele depth in
                            parents. This will ignore and set to "0" the posterior
                            probability for variants with a number of alternate
                            reads in parents higher than specified value
```

3.1.2 Input

novoCaller accepts files in VCF format as input. Files must contain genotype information for trio in addition to standard VCF columns. Column IDs for trio must match the sample IDs provided together with the list of RCK/BAM files (`--triofiles`).

Required VCF format structure:

```
#CHROM   POS   ID   REF   ALT   QUAL   FILTER   INFO   FORMAT   PROBAND_ID   MOTHER_
↪ID   FATHER_ID   ...
```

3.1.3 Trio and unrelated files

By default novoCaller expect bgzip and tabix indexed RCK files. To use BAM files directly specify `--bam` flag.

note: using BAM files directly will significantly slow down the software since pileup counts need to be calculated on the spot at each position and for each bam.

3.1.4 Output

novoCaller generates output in VCF format. Two new tags are used to report additional information for each call. *RSTR* stores reads counts by strand at position for reference and alternate alleles. *novoPP* stores posterior probabilty calculated for the call. Variants are sorted by posterior probability in desceding order.

RSTR tag definition (FORMAT):

```
##FORMAT=<ID=RSTR,Number=4,Type=Integer,Description="Read counts by strand for ref␣
↪and alt alleles (Rf,Af,Rr,Ar)">
```

novoPP tag definition (INFO):

```
##INFO=<ID=novoPP,Number=1,Type=Float,Description="Posterior probability from␣
↪novoCaller">
```

note: novoCaller model assumptions do not apply to unbalanced chromosomes (e.g. sex and mithocondrial chromosomes), therefore the model does not assign a posterior probabilty. When filtering by posterior probabilty (`--ppthr`), these variants are treated as if their posterior probabilty was 0.

3.1.5 Examples

Calls *de novo* variants. This will return the calls ranked and sorted by calculated posterior probabilty.

```
granite novoCaller -i file.vcf -o file.out.vcf -u file.unrelatedfiles -t file.
↪triofiles
```

It is possible to filter-out variants with posterior probabilty lower than `--ppthr`.

```
granite novoCaller -i file.vcf -o file.out.vcf -u file.unrelatedfiles -t file.
↪triofiles --ppthr <float>
```

It is possible to filter-out variants with population allele frequency higher than `--afthr`. Allele frequency must be provided for each variant in INFO column.

```
granite novoCaller -i file.vcf -o file.out.vcf -u file.unrelatedfiles -t file.
↪triofiles --afthr <float> --aftag tag
```

Filters can be combined.

```
granite novoCaller -i file.vcf -o file.out.vcf -u file.unrelatedfiles -t file.
↪triofiles --afthr <float> --aftag tag --ppthr <float>
```

3.2 comHet

comHet is a calling algorithm for compound heterozygous mutations. The model uses genotype-level information in pedigree (trio) and VEP-based annotations to call possible compound heterozygous pairs. VEP annotations are used to assign variants to genes and transcripts, genotype information allows to refine calls based on inheritance mode. Calls are further flagged as "Phased" or "Unphased", where "Phased" means that genotype information supports in-trans inheritance for alternate alleles from parents.

3.2.1 Arguments

```
usage: granite comHet [-h] -i INPUTFILE -o OUTPUTFILE --trio TRIO [TRIO ...]
                      [--VEPtag VEPTAG] [--sep SEP] [--filter_cmpHet]
                      [--allow_undef] [--SpliceAItag SPLICEAITAG] [--impact]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  --trio TRIO [TRIO ...]
                        list of sample IDs for trio, PROBAND is required and
                        must be listed FIRST (e.g. --trio PROBAND_ID
                        [PARENT_ID] [PARENT_ID])
  --VEPtag VEPTAG       by default the program will search for "CSQ" TAG
                        (CSQ=<values>), use this parameter to specify a
                        different TAG to be used (e.g. VEP)
  --sep SEP             by default the program uses "&" as separator for
                        subfields in annotating VCF (e.g.
                        ENST00000643759&ENST00000643774), use this parameter
```

(continues on next page)

(continued from previous page)

```
                        to specify a different separator to be used
    --filter_cmpHet     by default the program returns all variants in the
                        input VCF file. This flag will produce a shorter
                        output containing only variants that are potential
                        compound heterozygous
    --allow_undef       by default the program ignores variants with undefined
                        genotype in parents. This flag extends the output to
                        include these cases
    --SpliceAItag SPLICEAITAG
                        by default the program will search for SpliceAI delta
                        scores (DS_AG, DS_AL, DS_DG, DS_DL) to calculate the
                        max delta score for the variant. If a max value is
                        already defined, use this parameter to specify the TAG
                        | TAG field to be used
    --impact            use VEP "IMPACT" or "Consequence" terms to assign an
                        impact to potential compound heterozygous. If
                        available, SpliceAI and ClinVar "CLNSIG" information
                        is used together with VEP
```

3.2.2 Input

comHet accepts files in VCF format as input. Files must contain genotype information for trio members to be used in addition to standard VCF columns. Column IDs for trio must match the sample IDs provided as argument (`--trio`). Proband genotype information is mandatory. If available, parents information will be used to improve specificity by ruling-out false calls based on inheritance mode. VEP annotations for "Gene" and "Feature" are also required in INFO column for transcripts.

Required VCF format structure:

```
#CHROM   POS   ID   REF   ALT   QUAL   FILTER   INFO   FORMAT   PROBAND_ID   [MOTHER_
↪ID]   [FATHER_ID]
```

3.2.3 Output

comHet generates output in VCF format. The program adds a VEP-like tag to INFO field to report information for calls associated to each variant. *comHet* stores information for each compound heterozygous pair (cmpHet) that involves the variant.

comHet tag definition (INFO):

```
##INFO=<ID=comHet,Number=.,Type=String,Description="Putative compound heterozygous␣
↪pairs. Subembedded:'cmpHet':Format:'phase|gene|transcript|mate_variant'">
```

comHet tag definition (INFO) with `--impact`:

```
##INFO=<ID=comHet,Number=.,Type=String,Description="Putative compound heterozygous␣
↪pairs. Subembedded:'cmpHet':Format:'phase|gene|transcript|impact_gene|impact_
↪transcript|mate_variant'">
```

A cmpHet is defined for each gene and for each possible mate variant. Multiple cmpHets are listed separated by comma.

Example:

```
comHet=Phased|ENSG00000069424||STRONG_PAIR||chr1:6051661C>T,
↪Phased|ENSG00000069424|ENST00000652845|STRONG_PAIR|STRONG_PAIR|chr1:6082358C>T,
↪Phased|ENSG00000084636|ENST00000373672&ENST00000488897|STRONG_PAIR|STROING_
↪PAIR|chr1:6051661G>A
```

All shared transcripts for a given pair are listed in `transcript` field. If the pair does not share any transcript, the field is empty.

3.2.4 Examples

Calls compound heterozygous variants.

```
granite comHet -i file.vcf -o file.out.vcf --trio PROBAND_ID [PARENT_ID] [PARENT_ID]
```

It is possible to add impact information for gene (`impact_gene`) and for shared transcripts (`impact_transcript`). `impact_gene` is the worst impact calculated at gene level while considering all its associated transcripts. `impact_transcript` is the worst impact calculated considering only transcripts that are shared between the two mates, if any. VEP annotations for "IMPACT" or "Consequence" must be provided in INFO column in order to assign an impact. If available, SpliceAI and ClinVar "CLNSIG" information is used together with VEP to refine the assignment.

```
granite comHet -i file.vcf -o file.out.vcf --trio PROBAND_ID [PARENT_ID] [PARENT_ID] -
↪-impact
```

It is possible to reduce the output to only variants that are potential compound heterozygous.

```
granite comHet -i file.vcf -o file.out.vcf --trio PROBAND_ID [PARENT_ID] [PARENT_ID] -
↪-filter_cmpHet
```

3.2.5 Impact

A variant is considered to have a potential STRONG impact if VEP impact is HIGH or MODERATE, spliceAI score is >= 0.8, or ClinVar assignment is Pathogenic | Likely Pathogenic. If both variants are STRONG, the pair is assigned as a STRONG_PAIR. If only one of the two variants is STRONG, the pair is assigned as a MEDIUM_PAIR. If none of the variants is STRONG, the pair is assigned as a WEAK_PAIR.

CHAPTER 4

Variant Filtering

4.1 whiteList

whiteList allows to select and filter-in a subset of variants from input VCF file based on specified annotations and positions. The software can use provided VEP, ClinVar or SpliceAI annotations. Positions can be also specfied as a BED format file.

4.1.1 Arguments

```
usage: granite whiteList [-h] -i INPUTFILE -o OUTPUTFILE [--SpliceAI SPLICEAI]
                         [--SpliceAItag SPLICEAITAG] [--CLINVAR]
                         [--CLINVARonly CLINVARONLY [CLINVARONLY ...]]
                         [--CLINVARtag CLINVARTAG] [--VEP] [--VEPtag VEPTAG]
                         [--VEPrescue VEPRESCUE [VEPRESCUE ...]]
                         [--VEPremove VEPREMOVE [VEPREMOVE ...]]
                         [--VEPsep VEPSEP] [--BEDfile BEDFILE]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  --SpliceAI SPLICEAI   threshold to whitelist variants by SpliceAI delta
                        scores value (>=)
  --SpliceAItag SPLICEAITAG
                        by default the program will search for SpliceAI delta
                        scores (DS_AG, DS_AL, DS_DG, DS_DL) to calculate the
                        max delta score for the variant. If a max value is
                        already defined, use this parameter to specify the TAG
                        | TAG field to be used
  --CLINVAR             flag to whitelist all variants with a ClinVar entry
```

(continues on next page)

(continued from previous page)

```
                        [ALLELEID]
    --CLINVARonly CLINVARONLY [CLINVARONLY ...]
                        ClinVar "CLNSIG" terms or keywords to be saved. Sets
                        for whitelist only ClinVar variants with specified
                        terms or keywords
    --CLINVARtag CLINVARTAG
                        by default the program will search for ClinVar
                        "ALLELEID" TAG, use this parameter to specify a
                        different TAG to be used
    --VEP               use VEP "Consequence" annotations to whitelist exonic
                        and relevant variants (removed by default variants in
                        intronic, intergenic, or regulatory regions)
    --VEPtag VEPTAG     by default the program will search for "CSQ" TAG
                        (CSQ=<values>), use this parameter to specify a
                        different TAG to be used (e.g. VEP)
    --VEPrescue VEPRESCUE [VEPRESCUE ...]
                        additional terms to overrule removed flags to rescue
                        and whitelist variants
    --VEPremove VEPREMOVE [VEPREMOVE ...]
                        additional terms to be removed
    --VEPsep VEPSEP     by default the program expects "&" as separator for
                        subfields in VEP (e.g.
                        intron_variant&splice_region_variant), use this
                        parameter to specify a different separator to be used
    --BEDfile BEDFILE   BED format file with positions to whitelist
```

4.1.2 Examples

Whitelists variants with ClinVar entry. If available, ClinVar annotation must be provided in INFO column.

```
granite whiteList -i file.vcf -o file.out.vcf --CLINVAR
```

Whitelists only "Pathogenic" and "Likely_pathogenic" variants with ClinVar entry. ClinVar "CLNSIG" annotation must be provided in INFO column.

```
granite whiteList -i file.vcf -o file.out.vcf --CLINVAR --CLINVARonly Pathogenic
```

Whitelists variants based on SpliceAI annotations. This filters in variants with SpliceAI score equal/higher than `--SpliceAI`. If available SpliceAI annotation must be provided in INFO column.

```
granite whiteList -i file.vcf -o file.out.vcf --SpliceAI <float>
```

Whitelists variants based on VEP "Consequence" annotations. This withelists exonic and functional relevant variants by removing variants flagged as "intron_variant", "intergenic_variant", "downstream_gene_variant", "upstream_gene_variant", "regulatory_region_", "non_coding_transcript_". It is possible to specify additional *terms* to remove using `--VEPremove` and terms to rescue using `--VEPrescue`. To use VEP, annotation must be provided for each variant in INFO column.

```
granite whiteList -i file.vcf -o file.out.vcf --VEP
granite whiteList -i file.vcf -o file.out.vcf --VEP --VEPremove <str> <str>
granite whiteList -i file.vcf -o file.out.vcf --VEP --VEPrescue <str> <str>
granite whiteList -i file.vcf -o file.out.vcf --VEP --VEPrescue <str> <str> --
↪VEPremove <str>
```

Whitelists variants based on positions specified as a BED format file.

```
granite whiteList -i file.vcf -o file.out.vcf --BEDfile file.bed
```

Combine the above filters.

```
granite whiteList -i file.vcf -o file.out.vcf --BEDfile file.bed --VEP --VEPrescue
↪<str> <str> --CLINVAR --SpliceAI <float>
```

4.2 blackList

blackList allows to filter-out variants from input VCF file based on positions set in BIG format file and/or provided population allele frequency.

4.2.1 Arguments

```
usage: granite blackList [-h] -i INPUTFILE -o OUTPUTFILE [-b BIGFILE]
                         [--aftag AFTAG] [--afthr AFTHR]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  -b BIGFILE, --bigfile BIGFILE
                        BIG format file with positions set for blacklist
  --aftag AFTAG         TAG (TAG=<float>) or TAG field to be used to filter by
                        population allele frequency
  --afthr AFTHR         threshold to filter by population allele frequency
                        (<=) [1]
```

4.2.2 Examples

Blacklist variants based on position set to `True` in BIG format file.

```
granite blackList -i file.vcf -o file.out.vcf -b file.big
```

Blacklist variants based on population allele frequency. This filters out variants with allele frequency higher than `--afthr`. Allele frequency must be provided for each variant in INFO column.

```
granite blackList -i file.vcf -o file.out.vcf --afthr <float> --aftag tag
```

Combine the two filters.

```
granite blackList -i file.vcf -o file.out.vcf --afthr <float> --aftag tag -b file.big
```

4.3 cleanVCF

cleanVCF allows to clean INFO field of input VCF file. The software can remove a list of TAG from INFO field, or can be used to clean VEP annotations.

4.3.1 Arguments

```
usage: granite cleanVCF [-h] -i INPUTFILE -o OUTPUTFILE [-t TAG] [--VEP]
                        [--VEPtag VEPTAG]
                        [--VEPrescue VEPRESCUE [VEPRESCUE ...]]
                        [--VEPremove VEPREMOVE [VEPREMOVE ...]]
                        [--VEPsep VEPSEP] [--SpliceAI SPLICEAI]
                        [--SpliceAItag SPLICEAITAG]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  -t TAG, --tag TAG     TAG to be removed from INFO field. Specify multiple
                        TAGs as: "-t TAG -t TAG -t ..."
  --VEP                 clean VEP "Consequence" annotations (removed by
                        default terms for intronic, intergenic, or regulatory
                        regions from annotations)
  --VEPtag VEPTAG       by default the program will search for "CSQ" TAG
                        (CSQ=<values>), use this parameter to specify a
                        different TAG to be used (e.g. VEP)
  --VEPrescue VEPRESCUE [VEPRESCUE ...]
                        additional terms to overrule removed flags to rescue
                        annotations
  --VEPremove VEPREMOVE [VEPREMOVE ...]
                        additional terms to be removed from annotations
  --VEPsep VEPSEP       by default the program expects "&" as separator for
                        subfields in VEP (e.g.
                        intron_variant&splice_region_variant), use this
                        parameter to specify a different separator to be used
  --SpliceAI SPLICEAI   threshold to save intronic annotations, from VEP
                        "Consequence", for variants by SpliceAI delta scores
                        value (>=)
  --SpliceAItag SPLICEAITAG
                        by default the program will search for SpliceAI delta
                        scores (DS_AG, DS_AL, DS_DG, DS_DL) to calculate the
                        max delta score for the variant. If a max value is
                        already defined, use this parameter to specify the TAG
                        | TAG field to be used
```

4.3.2 Examples

Remove tag from INFO field.

```
granite cleanVCF -i file.vcf -o file.out.vcf -t tag
```

Clean VEP based on VEP "Consequence" annotations. This removes annotations flagged as "intron_variant", "intergenic_variant", "downstream_gene_variant", "upstream_gene_variant", "regulatory_region_", "non_coding_transcript_". It is possible to specify additional *terms* to remove using `--VEPremove` and terms to rescue using `--VEPrescue`. VEP annotation must be provided for each variant in INFO column.

```
granite cleanVCF -i file.vcf -o file.out.vcf --VEP
granite cleanVCF -i file.vcf -o file.out.vcf --VEP --VEPremove <str> <str>
```

(continues on next page)

(continued from previous page)

```
granite cleanVCF -i file.vcf -o file.out.vcf --VEP --VEPrescue <str> <str>
granite cleanVCF -i file.vcf -o file.out.vcf --VEP --VEPrescue <str> <str> --
↪VEPremove <str>
```

The program also accepts a SpliceAI threshold that will rescue annotations for "intron_variant" by SpliceAI. SpliceAI annotation must be provided in INFO column.

```
granite cleanVCF -i file.vcf -o file.out.vcf --VEP --SpliceAI <float>
```

Combine the above filters.

```
granite cleanVCF -i file.vcf -o file.out.vcf -t tag --VEP --VEPrescue <str> <str> --
↪SpliceAI <float>
```

4.4 geneList

geneList allows to filter VEP annotations from input VCF file using a list of genes. If a transcript is not mapping to any of the genes in the list, the transcript is removed from VEP annotation in INFO field. If all transcripts are removed, the VEP tag is removed from INFO field for the variant.

4.4.1 Arguments

```
usage: granite geneList [-h] -i INPUTFILE -o OUTPUTFILE -g GENESLIST
                        [--VEPtag VEPTAG]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as VCF, use .vcf as
                        extension
  -g GENESLIST, --geneslist GENESLIST
                        text file listing ensembl gene (ENSG) IDs for all
                        genes to save annotations for, IDs must be listed as a
                        column
  --VEPtag VEPTAG       by default the program will search for "CSQ" TAG
                        (CSQ=<values>), use this parameter to specify a
                        different TAG to be used (e.g. VEP)
```

CHAPTER 5

Quality Check

5.1 qcVCF

qcVCF produces a report in JSON format with different quality metrics calculated for input VCF file. Both single sample and family-based metrics are available.

5.1.1 Arguments

```
usage: granite qcVCF [-h] -i INPUTFILE -o OUTPUTFILE -p PEDIGREE --samples
                     SAMPLES [SAMPLES ...] [--ti_tv] [--trio_errors]
                     [--het_hom]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as JSON, use .json as
                        extension
  -p PEDIGREE, --pedigree PEDIGREE
                        pedigree information, either as JSON file or JSON
                        representation as string
  --samples SAMPLES [SAMPLES ...]
                        list of sample IDs to get stats for (e.g. --samples
                        SampleID_1 [SampleID_2] ...)
  --ti_tv               add transition-transversion ratio and statistics on
                        substitutions to report
  --trio_errors         add statistics on mendelian errors based on trio to
                        report
  --het_hom             add heterozygosity ratio and statistics on zygosity to
                        report
```

5.2 validateVCF

validateVCF produces a report in JSON format with error models for different inheritance modes calculated for input VCF file. Additional supporting plots are generated in PNG format.

5.2.1 Arguments

```
usage: granite validateVCF [-h] -i INPUTFILE -o OUTPUTFILE -p PEDIGREE
                           [PEDIGREE ...] --anchor ANCHOR [ANCHOR ...]
                           [--het HET [HET ...]] [--novo NOVO] [--verbose]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as JSON, use .json as
                        extension
  -p PEDIGREE [PEDIGREE ...], --pedigree PEDIGREE [PEDIGREE ...]
                        pedigree information, either as JSON file or JSON
                        representation as string. It is possible to specify
                        multiple pedigrees to load as list (e.g. --pedigree
                        pedigree_1 [pedigree_2] ...)
  --anchor ANCHOR [ANCHOR ...]
                        sample ID to be used as anchor in pedigree to build
                        family. It is possible to specify multiple sample IDs
                        as list. If multiple pedigrees are specified in "--
                        pedigree", anchors are positionally matched to
                        corresponding pedigree
  --het HET [HET ...]   sample ID to be used to calculate error model for
                        heterozygous mutations. It is possible to specify
                        multiple sample IDs as list. Each sample ID must
                        correspond to anchor specified in "--anchor"
  --novo NOVO           sample ID to be used to calculate error model for de
                        novo mutations. Must correspond to anchor specified in
                        "--anchor". Requires posterior probability from
                        novoCaller
  --type TYPE [TYPE ...]
                        by default error models are calculated only for SNV.
                        It is possible to specify different types of variant
                        to use (SNV, INS, DEL, MNV, MAV) as list (e.g. --type
                        SNV INS DEL)
```

5.3 SVqcVCF

SVqcVCF produces a report in JSON format with different quality metrics calculated for input SV VCF file. Currently, this function can count DEL and DUP SVs in single- and multi-sample SV VCF files. It reports the number of DEL, DUP, and total (the sum of only DEL and DUP) SVs for each sample provided in samples. Other SVTYPEs (INS, INV, CNV, BND) are currently ignored.

5.3.1 Arguments

```
usage: granite SVqcVCF [-h] -i INPUTFILE -o OUTPUTFILE
                       --samples SAMPLES [SAMPLES ...] [--verbose]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input SV VCF file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as JSON, use .json as
                        extension
  --samples SAMPLES [SAMPLES ...]
                        list of sample IDs to get stats for (e.g. --samples
                        SampleID_1 [SampleID_2] ...)
```

CHAPTER 6

Utilities

6.1 mpileupCounts

mpileupCounts uses *samtools* to access input BAM and calculates statistics for reads pileup at each position in the specified region, returns counts in RCK format.

6.1.1 Arguments

```
usage: granite mpileupCounts [-h] -i INPUTFILE -o OUTPUTFILE -r REFERENCE
                             [--region REGION] [--MQthr MQTHR] [--BQthr BQTHR]

optional arguments:
  -i INPUTFILE, --inputfile INPUTFILE
                        input file in BAM format
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as RCK format (TSV), use
                        .rck as extension
  -r REFERENCE, --reference REFERENCE
                        reference file in FASTA format
  --region REGION       region to be analyzed [e.g. chr1:1-10000000,
                        1:1-10000000, chr1, 1], chromosome name must match the
                        reference
  --MQthr MQTHR         minimum mapping quality for an alignment to be used
                        (>=) [0]
  --BQthr BQTHR         minimum base quality for a base to be considered (>=)
                        [13]
```

6.2 toBig

toBig converts counts from bgzip and tabix indexed RCK format into BIG format. Positions are "called" by read counts or allelic balance for single or multiple files (joint calls) in specified regions. Positions "called" are set to True

(or 1) in BIG binary structure.

6.2.1 Arguments

```
usage: granite toBig [-h] [-i INPUTFILE [INPUTFILE ...]] -o OUTPUTFILE -r
                     REGIONFILE -f CHROMFILE [--ncores NCORES] --fithr FITHR
                     [--rdthr RDTHR] [--abthr ABTHR]

optional arguments:
  -f FILE, --file FILE  file to be used to call positions. To do joint calling
                        specify multiple files as: "-f file_1 -f file_2 -f ...".
                        Expected bgzip and tabix indexed RCK file
  -o OUTPUTFILE, --outputfile OUTPUTFILE
                        output file to write results as BIG format (binary
                        hdf5), use .big as extension
  -r REGIONFILE, --regionfile REGIONFILE
                        file containing regions to be used [e.g.
                        chr1:1-10000000, 1:1-10000000, chr1, 1] listed as a
                        column, chromosomes names must match the reference
  -c CHROMFILE, --chromfile CHROMFILE
                        chrom.sizes file containing chromosomes size
                        information
  --ncores NCORES       number of cores to be used if multiple regions are
                        specified [1]
  --fithr FITHR         minimum number of files with at least "--rdthr" for
                        the alternate allele or having the variant, "calls" by
                        allelic balance, to jointly "call" position (>=)
  --rdthr RDTHR         minimum number of alternate reads to count the file in
                        "--fithr", if not specified "calls" are made by
                        allelic balance (>=)
  --abthr ABTHR         minimum percentage of alternate reads compared to
                        reference reads to count the file in "--fithr" when
                        "calling" by allelic balance (>=) [15]
```

6.2.2 Examples

toBig can be used to calculate positions to blacklist for common variants by using unrelated samples. This command will set to `True` in BIG structure positions with allelic balance for alternate allele equal/higher than `--abthr` in more that `--fithr` samples (joint calling).

```
granite toBig -f file -f file -f file -f file -f ... -o file.out.big -c file.chrom.
↪sizes -r file.regions --fithr <int> --abthr <int>
```

Absolute reads count can be used instead of allelic balance to call positions. This command will set to `True` in BIG structure positions with reads count for alternate allele equal/higher than `--rdthr` in more that `--fithr` samples (joint calling).

```
granite toBig -f file -f file -f file -f file -f ... -o file.out.big -c file.chrom.
↪sizes -r file.regions --fithr <int> --rdthr <int>
```

6.3 rckTar

rckTar creates a tar archive from bgzip and tabix indexed RCK files. Creates an index file for the archive.

6.3.1 Arguments

```
usage: granite rckTar [-h] -t TTAR -f FILE

optional arguments:
  -t TTAR, --ttar TTAR  target tar to write results, use .rck.tar as extension
  -f FILE, --file FILE  file to be archived. Specify multiple files as: "-f
                        SampleID_1.rck.gz -f SampleID_2.rck.gz -f ...". Files
                        order is maintained while creating the index
```

CHAPTER 7

VCF Parser

granite library can be used directly to access and manipulate information in VCF format.

7.1 Import the library

```
from granite.lib import vcf_parser
```

7.2 Usage

The library implements the objects Vcf, Header and Variant.

7.2.1 Vcf

This is the main object and has methods to read and write VCF format.

Initialize the object

```
vcf_obj = vcf_parser.Vcf('inputfile.vcf')
```

This will automatically read the file header into a *Header* object.

Read and access variants

The method *parse_variants()* will read the file and return a generator to *Variant* objects that store variants information.

```
for vnt_obj in vcf_obj.parse_variants():
    # do something with vnt_obj
```

Write to file

The method *write_header(fo)* allows to write header definitions and columns to specified buffer (fo).

```
with open('outputfile.vcf', 'w') as fo:
    vcf_obj.write_header(fo)
```

It is possible to write only definitions or columns respectively with the methods *write_definitions(fo)* and *write_columns(fo)*.

```
with open('outputfile.vcf', 'w') as fo:
    vcf_obj.write_definitions(fo)
    vcf_obj.write_columns(fo)
```

The method *write_variant(fo, Variant_obj)* allows to write information from *Variant* object to specified buffer (fo).

```
with open('outputfile.vcf', 'w') as fo:
    vcf_obj.write_variant(fo, vnt_obj)
```

7.2.2 Header

This is the object used to store information for the header in VCF format. Methods are available to extract and modify information in the header.

Attributes

definitions *<str>*

Stores the full header information minus the last line where columns are defined.

```
vcf_obj.header.definitions
```

columns *<str>*

Stores the last header line where columns are defined.

```
# columns example
# #CHROM    POS    ID    REF    ALT    ...
vcf_obj.header.columns
```

IDs_genotypes *<list>*

Stores sample ID(s) available in the VCF as list. If multiple samples, the order from the VCF is maintained.

```
vcf_obj.header.IDs_genotypes
```

Add or remove definitions

The method *add_tag_definition(tag_definition, tag_type='INFO')* allows to add tag_definition to the header on top of the block specified by tag_type (e.g. FORMAT, INFO).

```
tag_definition = '##INFO=<ID=tag,Number=.,Type=.,Description="INFO tag definition␣
↪example">'
vcf_obj.header.add_tag_definition(tag_definition)
```

The method *remove_tag_definition(tag, tag_type='INFO')* allows to remove tag definition from the header block specified by tag_type (e.g. FORMAT, INFO).

```
# remove CSQ definition (VEP) from the header INFO block
tag = 'CSQ'
vcf_obj.header.remove_tag_definition(tag)
```

Extract information

The method *get_tag_field_idx(tag, field, tag_type='INFO', sep='|')* allows to get the index corresponding to value field in tag from definition, block specified by tag_type (e.g. FORMAT, INFO). sep is the fields separator used in the tag definition.

```
# return the index corresponding to 'Consequence' field
# from CSQ definition (VEP) in the header INFO block
# ##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from␣
↪Ensembl VEP. Format: Gene|Feature|Consequence|IMPACT|...">
tag, field = 'CSQ', 'Consequence'
idx <int> = vcf_obj.header.get_tag_field_idx(tag, field) # idx -> 3
```

The method *check_tag_definition(tag, tag_type='INFO', sep='|')* allows to check if a tag is in the header and if is standalone or field of another leading tag. Returns the leading tag and the field corresponding index, if any, to acces the tag. sep is the fields separator used in the tag definition.

```
# return the leading tag and index corresponding to 'Consequence' field
# from CSQ definition (VEP) in the header INFO block
# ##INFO=<ID=CSQ,Number=.,Type=String,Description="Consequence annotations from␣
↪Ensembl VEP. Format: Gene|Feature|Consequence|IMPACT|...">
tag = 'Consequence'
lead_tag <str>, idx <int> = vcf_obj.header.check_tag_definition(tag) # lead_tag -> CSQ
                                                                     # idx -> 3
```

note: tag and field are case sensitive.

7.2.3 Variant

This is the object used to store information for variants in VCF format.

Attributes

CHROM *<str>*

Stores chromosome name (e.g. 1, chr1), as in the VCF file.

```
vnt_obj.CHROM
```

POS *<int>*

Stores variant position.

```
vnt_obj.POS
```

ID *<str>*

Stores variant ID(s), as in the VCF file.

```
vnt_obj.ID
```

REF *<str>*

Stores reference allele at position.

```
vnt_obj.REF
```

ALT *<str>*

Stores alternate allele(s) at position.

```
vnt_obj.ALT
```

QUAL *<str>*

Stores phred-scaled quality score for the assertion made in ALT.

```
vnt_obj.QUAL
```

FILTER *<str>*

Stores filter status.

```
vnt_obj.FILTER
```

INFO *<str>*

Additional information for the variant.

```
vnt_obj.INFO
```

FORMAT *<str>*

Stores specification for the genotype column(s) structure.

```
vnt_obj.FORMAT
```

IDs_genotypes *<list>*

Stores sample ID(s) available in the VCF as list. If multiple samples, the order from the VCF is maintained.

```
vnt_obj.IDs_genotypes
```

GENOTYPES *<dict>*

Stores a dictionary linking genotype(s) for the variant to corresponding sample ID(s).

```
# {ID_genotype: genotype, ...}
vnt_obj.GENOTYPES
```

Format variants

The method *to_string()* returns the variant representation in VCF format.

```
vnt_vcf <str> = vnt_obj.to_string()
```

The method *repr()* returns the variant representation as *CHROM:POSREF>ALT*.

```
vnt_repr <str> = vnt_obj.repr()
```

Manipulate genotype(s)

The method *remove_tag_genotype(tag, sep=':')* allows to remove a tag from FORMAT and GENOTYPES. sep is the tags separator used in format definition and genotype(s).

```
# remove AD tag from format definition and genotype(s)
tag = 'AD'
vnt_obj.remove_tag_genotype(tag)
```

The method *complete_genotype(sep=':')* fills in the trailing fields that are missing and by default dropped in GENOTYPES. sep is the tags separator used in format definition and genotype(s).

```
vnt_obj.complete_genotype()
```

The method *empty_genotype(sep=':')* returns a empty genotype based on FORMAT structure. sep is the tags separator used in format definition and genotype(s).

```
empty <str> = vnt_obj.empty_genotype()
```

The method *add_tag_format(tag, sep=':')* allows to add a tag at the end of FORMAT structure. sep is the tags separator used in format definition and genotype(s).

```
# add RSTR tag to format
tag = 'RSTR'
vnt_obj.add_tag_format(tag)
```

The method *add_values_genotype(ID_genotype, values, sep=':')* allows to add values at the end of the genotype specified by corresponding ID. sep is the tags separator used in format definition and genotype(s).

```
vnt_obj.add_values_genotype(ID_genotype, values)
```

The method *get_genotype_value(ID_genotype, tag, sep=':')* returns value for tag from the genotype specified by corresponding ID. sep is the tags separator used in format definition and genotype(s).

```
tag_val <str> = vnt_obj.get_genotype_value(ID_genotype, tag)
```

Manipulate INFO

The method *remove_tag_info(tag, sep=';')* allows to remove a tag from INFO. sep is the tags separator used in INFO.

```
vnt_obj.remove_tag_info(tag)
```

The method *add_tag_info(tag_value, sep=';')* allows to add a tag and its value at the end of INFO. sep is the tags separator used in INFO.

```
# add tag and value to INFO
tag_value = 'tag=value'
vnt_obj.add_tag_info(tag_value)
```

The method *get_tag_value(tag, sep=';')* returns the value from tag in INFO. sep is the tags separator used in INFO.

```
tag_val <str> = vnt_obj.get_tag_value(tag)
```

note: tag and ID are case sensitive.