GATK run report

Vikas Gupta, Niraj Shah and Stig U. Andersen

March 25, 2014

Contents

GATK is an acronym for Genome analysis toolkit, a detailed documentation can be found at `http://www.broadinstitute.org/gatk/guide/`. **GATK** pipeline can used for detecting the variants from multi-sample data. One of the problem with the variant detection is the misaligned reads, this issue has been addressed in GATK by considering the per-base quality score which represents the probability that the base called in the read is true sequenced base. These per-base quality scores are quite inaccurate (Mark A DePristo et.al 2011) and depends on the experimental setup and technology used. These errors are eliminated from the GATK using empirically accuarate per-base error model.

1. Introduction

Here we have utilized sequencing data from MG20, Gifu, Burttii and 28 additional accessions. Using this data we aim to annotate the genetic variants such as SNPs and Indels in the *Lotus japonicus*.

2. WorkFlow

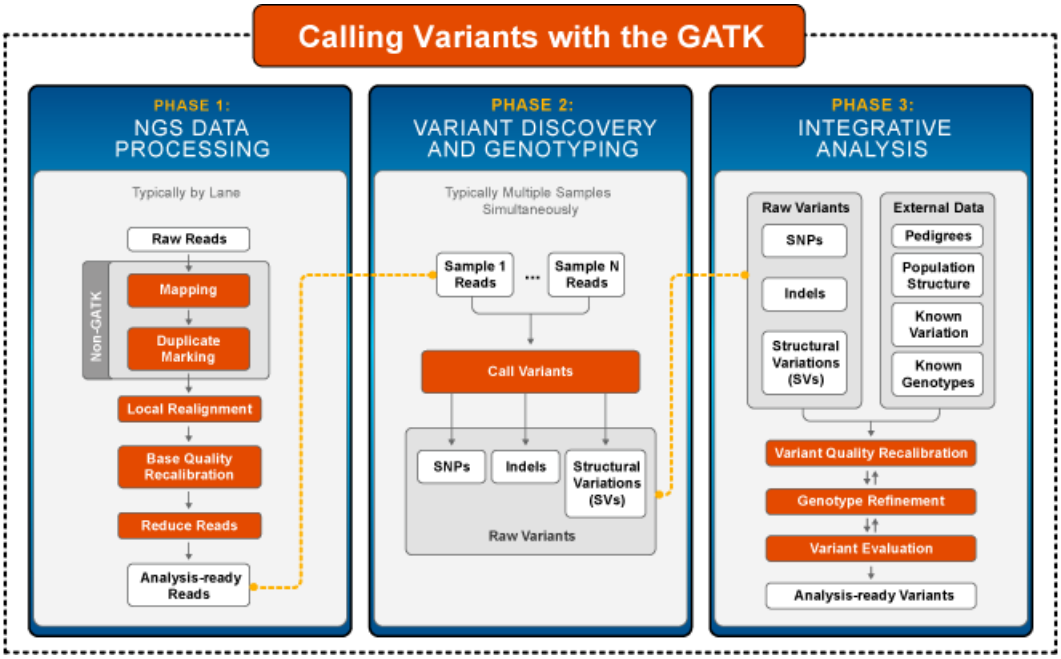

Figure 1: Three sections of GATK pipeline

We have divided work here into four major points:
1. Initial read mapping
2. Local realignment around indels
3. Base quality score recalibration
4. SNP and indel detection

2..1 Fastq Mapping

Fastq files were mapped to the Lotus genome 3.0 using the BWA version-0.7.5a-r405 with default parameters and with appropriate insert size for the pair-end libraries. GATK pipeline does not support files without read groups so these were added whenever necessary with Picard *AddOrReplaceReadgroup* function. All the mapped files are placed at genome.au.dk. `/home/vgupta/LotusGenome/100_data/01_Jin_BamFile/02_withReadGroup`.

2..2 Duplicate Marking

Duplicates were filtered using Picard toolkit version 1.96. Two things to remember when using Picard for duplicate filtering:
1. Set *VALIDATION_STRINGENCY=LENIENT* otherwise Picard will not accept umapped read positions.
2. Use *TMP_DIR* variable to a folder where you have sufficient space.

2..3 Realiging the reads

Duplicate filtered reads are mapped back on the genome using RealignerTargetCreator and IndelRealigner commands from the GATK. All the fastq files must follow the sanger quality encoding. IndelRealigner locally aligns the reads to minimize the mismatches. Also many reads are misaligned due to presence of insertions and deletions, leading to many false discoveries of SNPs.

2..4 Unified genotyper

Realigned bam files are parsed through the unified genotyper to procude a primary list of snp and indel list. -glm BOTH option must be used to predict indels together with SNPs. A higher degree of calling confidence cut-off can be used to insure minimal false positives. Unified genotyper uses a Bayesian genotype likelihood model to find genotypes and allele frequency in a population of N samples. It provides a posterior probability of there being a segregating variant allele at each locus as well as for the genotype of each sample.

2..5 Base Recalibrator

In this step, base quality scores are recalibrated. Given a set of high confidence SNPs from the earlier step, program calculates an empirical probability of error given the particular covariates seen at this site, where p(error) = num mismatches / num observations.

3. Callable loci

Genome-wide Callable loci destribution

4. SNP destribution

Genome-wide SNP density destribution

/Users/vgupta/Desktop/03_Lotus_annotation/2013_week35/snpDen

Figure 2: SNP distribution

5. Heterozygosity destribution

Genome-wide heterozygosity distribution

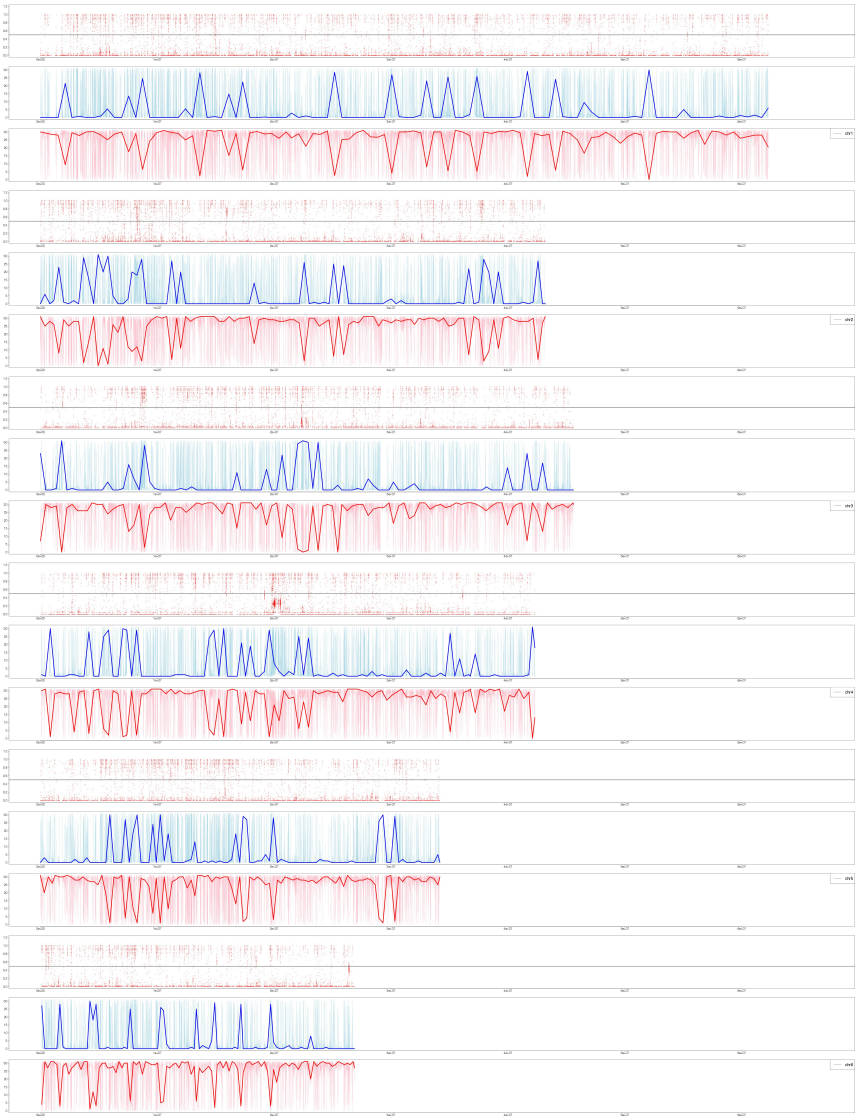

Figure 3: Heterozygosity distribution

5..1 Mapping on Lotus Genome

5..2 Average coverage on Lotus Genome

FileName	AverageCoverage	StdDev
MG20	51.4679	137.669
Gifu	80.7166	163.611
mg004	7.12603	38.2242
mg010	7.89245	45.8791
mg012	5.15874	31.085
mg019	8.33885	39.0382
mg023	8.88417	51.2091
mg036	9.49761	43.2652
mg049	9.33341	47.7517
mg051	8.35398	55.9712
mg062	7.66547	55.2581
mg072	6.96558	49.5173
mg073	7.84491	40.8285
mg077	6.85289	48.9573
mg080	7.74727	41.1546
mg082	7.99317	35.8655
mg083	5.75298	30.0052
mg086	7.50583	33.8896
mg089	7.30731	38.3547
mg093	6.58203	29.0308
mg095	NULL	NULL
mg097	7.11818	37.69
mg101	9.87511	55.1488
mg107	9.3095	47.0785
mg109	6.96913	35.0317
mg112	9.66309	39.9193
mg113	7.14424	48.0236
mg118	7.18516	49.812
mg123	5.03514	40.5754

Table 2: Average Coverage of re-sequenced lines

5..3 Inbreeding coefficient

A good example of calculating the F (Inbreeding co-efficient) is explained at `http://www.uwyo.edu/dbmcd/popecol/maylects/fst.html`. Hardy-Weinberg Equilibrium calculates values of p and q based on the dominant alleles and using p and q values calculates expected genotype counts. Based on the diffences on genotypic counts, we can catogerize a popultation as perfect fit, homozygtes excessive (inbreed) or homozygotes deficient (isolate breaking).

6. Parsing VCF

Note: Total depth on a given position doesn't add up to sum of individual sample read depth. Explanation is copied from `http://www.broadinstitute.org/gatk/gatkdocs/org_broadinstitute_sting_gatk_walkers_annotator_Coverage.html`.

While the sample-level (FORMAT) DP field describes the total depth of reads that passed the caller's internal quality control metrics (like MAPQ > 17, for example), the INFO field DP represents the unfiltered depth over all samples. Note though that the DP is affected by downsampling (-dcov), so the max value one can obtain for N samples with -dcov D is N * D.

6..1 Heterozygosity

We have measured the level of heterozygosity using the final vcf file produced by GATK. It reports a total of 394,861,527 base-pairs which includes all the callable sites. 5,280,517 markers(SNPs and Indels) were detected. Expect a small fraction of 40,000, all 1.39 million makers from prior analysis were recovered. MG20 echotype has been inbreed for many generations and expected to be highly homozygous but due to error in the mapping and assembly, we found a high fraction of markers were heterozygous across the genome. To remove these potentially false positive markers, we filtered all the markers where genotype call was not 0/0 (reference based homozygous) for the MG20 resulted into a set of 3,989,842 markers among MG20, Gifu, Burttii and 28 re-sequenced. Lines in the analysis seems to highly homozygous hence can be analysed as haploid.

Sample	HetCount	HomoCount	HetPer	HomoPer
MG20	0	3989842	0.00	1.00
Gifu	398844	3529289	0.10	0.90
Burtii	600378	3138281	0.16	0.84
mg004	394300	2980909	0.12	0.88
mg010	299897	2869762	0.09	0.91
mg012	325414	2806793	0.10	0.90
mg019	288101	2756945	0.09	0.91
mg023	319694	2844582	0.10	0.90
mg036	224498	2417182	0.08	0.92
mg049	255672	2610175	0.09	0.91
mg051	400899	2752953	0.13	0.87
mg062	328410	2895575	0.10	0.90
mg072	296329	2839960	0.09	0.91
mg073	388106	3111348	0.11	0.89
mg077	367029	3075315	0.11	0.89
mg080	417848	3071118	0.12	0.88
mg082	365037	3067565	0.11	0.89
mg083	334598	2843448	0.11	0.89
mg086	388324	3099269	0.11	0.89
mg089	396557	3046736	0.12	0.88
mg093	356494	3000176	0.11	0.89
mg095	355525	2970284	0.11	0.89
mg097	406711	3107906	0.12	0.88
mg101	417805	3136307	0.12	0.88
mg107	398167	3131309	0.11	0.89
mg109	353309	2944567	0.11	0.89
mg112	390936	3152211	0.11	0.89
mg113	395130	2980594	0.12	0.88
mg118	364239	2964873	0.11	0.89
mg123	315600	2810258	0.10	0.90
mg128	377147	3059781	0.11	0.89

Table 3: Heterozygocity of all the lines

6..2 Marker filtering criteria

Low quality mappings should be filtered before we consolidate a high confident marker list. We need a filtering criteria to remove false positive markers, similar filtering stradegy must be used for also considering the callable fraction of the genome. To startwith we are considering three parameters for the filtering, QUAL (phred-scaled quality score) values, FILTER categories and the MQ (RMS mapping quality).

6..2.1 Counting filtering categories

[vgupta@fe1 02_withReadGroup]$cut - f7$ 20130905.$snp.vcf|$ $sort|$ $uniq - c$
228, 849, 923 .
166, 011, 565 *LowQual*

6..2.2 Average QUAL value

[vgupta@s01n14 02_withReadGroup]$grep - v''$20130905.$snp.vcf|awk'if(min == "")min = max =$6; if((6) $>$ $max)max = (6)$; if((6) $< min)min =$6; total+=(6); $count+ = 1ENDprinttotal, total/count, min, max'$
$Total$: 3.86946$e + 12Average$: 9, 799.54Min : $.Max$: 34, 359, 763, 362.97

6..2.3 MG20 homozygous positions

A total of 394,861,527 positions were reported using Unified Genotyper, of which, 307,385,387 positions were homozygous for MG20. We also filtered the vcf file for all the positions where MG20 was the only sample with the coverage, resulting positions will be refered as Callable Loci which consist a total of 301,306,971 genomic positions.

6..2.4 Variant annotation

Using SNPeff `http://snpeff.sourceforge.net/SnpEff_manual.html`, we have annotated 1,354,324 snps and 234,209 indels. VCF files for respective data are as following:
`/u/pm/data/2013_09_11_lotus_snps/20130905.snp.vcf.markers.snps.inbreed.ref`
`/u/pm/data/2013_09_11_lotus_snps/20130905.snp.vcf.markers.indels.inbreed.ref`

7. Python Script

Listing 1: GATK pipeline

```
#------------------------------------------------------------+
#                                                            |
# 116_runGATK.py - script to run GATK analysis               |
#                                                            |
#------------------------------------------------------------+
#                                                            |
# AUTHOR: Vikas Gupta                                        |
# CONTACT: vikas0633@gmail.com                               |
# STARTED: 09/06/2013                                        |
# UPDATED: 09/06/2013                                        |
#                                                            |
# DESCRIPTION:                                               |
#                                                            |
# LICENSE:                                                   |
#  GNU General Public License, Version 3                     |
#  http://www.gnu.org/licenses/gpl.html                      |
#                                                            |
#------------------------------------------------------------+

# Example:
# python ~/script/python/116_runGATK.py -i 02_Stegodyphous_cdna.refined.fa.orf.tr_longest_frame


### import modules
import os,sys,getopt, re


### global variables
global bams, ref, picard, gatk, threads, variant, sort_bams, tmp_dir

### make a logfile
import datetime
now = datetime.datetime.now()
o = open(str(now.strftime("%Y-%m-%d_%H%M."))+'logfile','w')



### write logfile

def logfile(infile):
    o.write("Program used: \t\t%s" % "116_runGATK.py"+'\n')
    o.write("Program was run at: \t%s" % str(now.strftime("%Y-%m-%d_%H%M"))+'\n')
    o.write("Infile used: \t\t%s" % infile+'\n')


def help():
    print '''
            python 116_runGATK.py
                                -b <bams> [One bam file per sample seperated by commas]
                                -r <ref> [Reference sequence]
                                -p <picard> [Path to picard folder MarkDuplicates.jar]
```

```
52                                  -g <gatk> [Path to GATK folder containing
                                        GenomeAnalysisTK.jar]
53                                  -t <threads> [Number of threads to be used]
54
55
56              '''
57      sys.exit(2)
58
59  ### main argument to
60
61  def options(argv):
62
63      global bams, ref, picard, gatk, threads, variant, sort_bams, tmp_dir
64
65      bams = ''
66      ref = ''
67      picard = ''
68      gatk = ''
69      threads = str(1)
70      variant = ''
71      sort_bams = False
72      tmp_dir="/home/vgupta/temp"
73
74      try:
75          opts, args = getopt.getopt(argv,"hb:r:p:g:t:v:s",["bams=","ref=","picard=",
                "gatk=","threads=","variant=", "sort_bams="])
76      except getopt.GetoptError:
77          help()
78      for opt, arg in opts:
79          if opt == '-h':
80              help()
81          elif opt in ("-b", "--bams"):
82              bams = arg
83          elif opt in ("-r", "--ref"):
84              ref = arg
85          elif opt in ("-p", "--picard"):
86              picard = arg
87          elif opt in ("-g", "--gatk"):
88              gatk = arg
89          elif opt in ("-t", "--threads"):
90              threads = str(arg)
91          elif opt in ("-v", "--variant"):
92              variant = arg
93          elif opt in ("-s", "--sort"):
94              sort_bams = True
95
96
97      logfile(bams)
98
99  def Index():
100     if not os.path.isfile(ref + '.fai'):
101         os.system('nice -n 19 samtools faidx '+ ref)
102
103
104 def files():
105     files = []
106     print bams
107     for file in bams.split(','):
108         print file
```

```
109         files.append(file.strip())
110     return files
111
112 def sortBams(file_list):
113     if sort_bams == True:
114         file_list_bams = []
115         for file in file_list:
116             os.system('samtools sort '+file+' '+file+'_sorted')
117             file_list_bams.append(file+'_sorted.bam')
118         return file_list_bams
119
120     return file_list
121
122 def MarkDuplicates(file_list):
123     for file in file_list:
124         print 'Marking duplicates for', file
125         os.system('java -Xmx50g -jar '+picard+'/MarkDuplicates.jar
              VALIDATION_STRINGENCY=LENIENT TMP_DIR='+tmp_dir +' INPUT='+ file + '
              OUTPUT=' + file+'.dedup.bam' +' METRICS_FILE='+ file +'.dups')
126
127
128 def ReAlign(file_list):
129     for file in file_list:
130         print 'Realigning', file
131         os.system('java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK
              .jar --fix_misencoded_quality_scores -fixMisencodedQuals -U -T
              RealignerTargetCreator '+' -I ' + file+'.dedup.bam' +' -nt '+ threads +
              ' -R ' + ref +' -o '+ file+'.intervals')
132         os.system('java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK
              .jar --fix_misencoded_quality_scores -fixMisencodedQuals -U -T
              IndelRealigner ' + ' -targetIntervals ' + file+'.intervals '+ ' -I ' +
              file+'.dedup.bam' +' -R ' + ref \
133                   + ' -o ' + file+'.realigned.bam')
134
135
136 def UnifiedGenotyper(file_list):
137     if variant == '':
138         in_string = ''
139         ### make input string
140         for file in file_list:
141             os.system('samtools index '+file)
142             in_string += ' -I '+file
143         print 'Running UnifiedGenotyper'
144         '''
145         ### for sample
146         os.system(' java -jar '+gatk+'/GenomeAnalysisTK.jar '\
147         + ' -R ' + ref \
148         + ' -T  UnifiedGenotyper '\
149         + in_string \
150         + ' -nt ' + threads \
151         + ' -o snps.90.raw.vcf ' \
152         + '-stand_call_conf 20 ' \
153         + '-stand_emit_conf 10.0 '\
154         + '-dcov 2 ')
155
156         '''
157         os.system(' java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/
              GenomeAnalysisTK.jar '\
158         + ' -R ' + ref \
```

```
159         + ' -T  UnifiedGenotyper -glm BOTH '\
160         + in_string \
161         + ' -nt ' + threads \
162         + ' -o '+ file +'.90.vcf ' \
163         + '-stand_call_conf 90 ' \
164         + '-stand_emit_conf 10.0 '\
165         + '-dcov 200 ')
166
167
168 def recal(file_list):
169     global variant
170     if variant == '':
171         variant = file_list[-1] +'.90.vcf'
172     for file in file_list:
173         print 'Running BaseRecalibrator for ', file
174         os.system('java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK
              .jar -U -T BaseRecalibrator -rf BadCigar ' + ' -knownSites ' +variant +
               ' -I ' + file+'.realigned.bam '+ ' -R ' + ref \
175               + ' -o ' + file+'.recal.table')
176         os.system('java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK
              .jar -T PrintReads -R '+ref +' -I '+ file+'.realigned.bam  -L 20 '+ ' -
              BQSR '+file+'.recal.table' +' -o '+ file+ '.recal_reads.bam' )
177
178 def ReduceReads(file_list):
179     for file in file_list:
180         print 'Running ReduceReads for ', file
181         os.system('java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK
              .jar -U -T ReduceReads -rf BadCigar -I ' + file+ '.recal_reads.bam'+ '
              -R ' + ref \
182               + ' -o ' + file+'.reduced.bam')
183
184 def ReUnifiedGenotyper(file_list):
185     print 'Running ReUnifiedGenotyper'
186     in_string = ''
187     ### make input string
188     for file in file_list:
189         os.system('samtools index '+ file+'.reduced.bam')
190         in_string += ' -I '+file+'.reduced.bam'
191
192     os.system(' java -jar -Djava.io.tmpdir='+tmp_dir+' '+gatk+'/GenomeAnalysisTK.
          jar '\
193     + ' -R ' + ref \
194     + ' -T  UnifiedGenotyper -glm BOTH'\
195     + in_string \
196     + ' -nt ' + threads \
197     + ' -o snps.raw.vcf ')
198
199 def BuildErrorModelWithVQSR(file , var):
200     os.system('java -jar '+gatk+'/GenomeAnalysisTK.jar '\
201     + ' -T VariantRecalibrator ' \
202     + ' -R '+ ref \
203     + ' -input '+ file \
204     + ' -recalFile output.recal ' \
205     + ' -tranchesFile output.tranches ' \
206     + ' -nt ' + threads \
207     + ' -mode ' + var)
208
209 if __name__ == "__main__":
210
```

```
    options(sys.argv[1:])

    ### check if index exits
    Index()


    ### return the list of the bam files
    file_list = files()

    ### sort Bams file
    file_list = sortBams(file_list)

    ### mark duplicates
    MarkDuplicates(file_list)

    ### realign the reads
    ReAlign(file_list)

    ### call UnifiedGenotyper to make a primary list of variants
    UnifiedGenotyper(file_list)

    ### Baserecalibration
    recal(file_list)

    ### reducing BAM files
    ReduceReads(file_list)

    ### Run UnifiedGenotyper
    ReUnifiedGenotyper(file_list)


    ### BuildErrorModelWithVQSR
    #BuildErrorModelWithVQSR('snps.raw.vcf', 'SNP')
    #BuildErrorModelWithVQSR('snps.raw.vcf', 'INDEL')

    ### close the logfile
    o.close()
```

Listing 2: vcfParser

```
#------------------------------------------------------------+
#                                                            |
# 119_vcfParser.py - script to parse vcf format file         |
#                                                            |
#------------------------------------------------------------+
#                                                            |
# AUTHOR: Vikas Gupta                                        |
# CONTACT: vikas0633@gmail.com                               |
# STARTED: 09/06/2013                                        |
# UPDATED: 09/06/2013                                        |
#                                                            |
# DESCRIPTION:                                               |
# Short script to convert and copy the wheat BACs            |
# Run this in the parent dir that the HEX* dirs exist        |
#                                                            |
# LICENSE:                                                   |
#  GNU General Public License, Version 3                     |
#  http://www.gnu.org/licenses/gpl.html                      |
#                                                            |
#------------------------------------------------------------+
```

```
21
22  # Example:
23  # python ~/Desktop/script/python/119_vcfParser.py -i snp.90.PhredQual_5000.vcf
24
25
26  ### import modules
27  import os,sys,getopt, re, classVCF
28
29
30  ### global variables
31  global ifile, HEADER
32
33  ### make a logfile
34  import datetime
35  now = datetime.datetime.now()
36  o = open(str(now.strftime("%Y-%m-%d_%H%M."))+'logfile','w')
37
38
39
40  ### write logfile
41
42  def logfile(infile):
43      o.write("Program used: \t\t%s" % "100b_fasta2flat.py"+'\n')
44      o.write("Program was run at: \t%s" % str(now.strftime("%Y-%m-%d_%H%M"))+'\n')
45      o.write("Infile used: \t\t%s" % infile+'\n')
46
47
48  def help():
49      print '''
50              python 100b_fasta2flat.py -i <ifile>
51              '''
52      sys.exit(2)
53
54  ### main argument to
55
56  def options(argv):
57      global ifile
58      ifile = ''
59      try:
60          opts, args = getopt.getopt(argv,"hi:",["ifile="])
61      except getopt.GetoptError:
62          help()
63      for opt, arg in opts:
64          if opt == '-h':
65              help()
66          elif opt in ("-i", "--ifile"):
67              ifile = arg
68
69      logfile(ifile)
70
71  ### check if file empty
72  def empty_file(infile):
73      if os.stat(infile).st_size==0:
74          sys.exit('File is empty')
75
76
77  def parseFile(ifile):
78      o = open(ifile+'.MG20filtered','w')
79      global HEADER
```

```
    count = 0
    for line in open(ifile,'r'):
        if len(line) > 1 and not line.startswith('##'):
            line = line.strip('\n')
            if line.startswith('#CHROM'):
                o.write(line+'\n')
                HEADER = line
                samples_het = []
                samples_homo = []
                sample_names = line.split('\t')[9:]
                samples_len = len(line.split('\t')) -9
                for i in range(samples_len):
                    samples_het.append(0)
                    samples_homo.append(0)
            else:
                obj = classVCF.VCF(line)
                genotypes = obj.genotypes()
                ### check if the MG20 is 0/0 reference Homozygous
                if obj.genotype(2) == '0/0':
                    o.write(line+'\n')
                    count += 1
                    genotypes = obj.genotypes()
                    for i in range(len(genotypes)):
                        if obj.genotype(i) =='0/1' or obj.genotype(i) =='1/0':
                            samples_het[i] += 1
                        elif obj.genotype(i) =='0/0' or obj.genotype(i) =='1/1':
                            samples_homo[i] += 1
    print 'Marksers used: ',count
    print 'Sample\tHetCount\tHomoCount\tHetPer\tHomoPer'

    for i in range(len(sample_names)):
        total = int(samples_het[i]) + int(samples_homo[i])
        Het_per = float(samples_het[i])/total
        Homo_per = float(samples_homo[i])/total
        print sample_names[i] + '\t' + str(samples_het[i]) + '\t' + str(
            samples_homo[i]) + '\t' + str(Het_per) + '\t' + str(Homo_per)
    o.close()

if __name__ == "__main__":

    options(sys.argv[1:])
    empty_file(ifile)

    parseFile(ifile)


    ### close the logfile
    o.close()
```

Listing 3: classVCF

```


import re

### split line
def split_line(line):
    return line.strip().split('\t')

class VCF:
```

```
10      def __init__(self, line):
11
12          tokens = split_line(line)
13          self.CHROM = tokens[0]
14          self.POS = tokens[1]
15          self.ID = tokens[2]
16          self.REF = tokens[3]
17          self.ALT = tokens[4]
18          self.QUAL = tokens[5]
19          self.FILTER = tokens[6]
20          self.INFO = tokens[7]
21          self.FORMAT = tokens[8]
22
23          self.GENOTYPE = []
24
25          for i in tokens[9:]:
26              self.GENOTYPE.append(i)
27
28
29      def __str__(self):
30          return self.CHROM+'\t'+self.POS
31
32      def chroms(self):
33          return self.CHROM
34
35      def poss(self):
36          return self.POS
37
38      def ids(self):
39          return self.ID
40
41      def refs(self):
42          return self.REF
43
44      def alts(self):
45          return self.ALT
46
47      def quals(self):
48          return self.QUAL
49
50      def filters(self):
51          return self.FILTER
52
53      def infos(self):
54          return self.INFO
55
56      def formats(self):
57          return self.FORMAT
58
59      def genotypes(self):
60          return self.GENOTYPE
61
62      def depth(self):
63          match = re.search(r'DP=.+;',self.INFO)
64          if match:
65              return match.group().split(';')[0].replace('DP=','')
66          else:
67              return 0
68
```

```
69      def genotype(self, i):
70          if len(self.GENOTYPE[i].split(':')) > 1:
71              return self.GENOTYPE[i].split(':')[0]
72          else:
73              return 'NONE'
74
75      def genotypeDepth(self, i):
76          if len(self.GENOTYPE[i].split(':')) > 1:
77              if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index('DP'
                    ) and self.genotype(i) != './.':
78                  return self.GENOTYPE[i].split(':')[self.FORMAT.split(':').index('DP
                        ')]
79              else:
80                  return 'NONE'
81          else:
82              return 0
83
84      def genotypeQual(self, i):
85          if len(self.GENOTYPE[i].split(':')) > 1:
86              if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index('GQ'
                    ) and self.genotype(i) != './.':
87                  return self.GENOTYPE[i].split(':')[self.FORMAT.split(':').index('GQ
                        ')]
88              else:
89                  return 0
90          else:
91              return 0
92
93      def genotypeDepthSUM(self):
94          geno_sum = 0
95          for i in self.GENOTYPE:
96              if len(i.split(':')) > 1:
97                  if len(self.GENOTYPE[i].split(':')) > self.FORMAT.split(':').index(
                        'DP'):
98                      geno_sum += int(i.split(':')[self.FORMAT.split(':').index('DP')
                            ])
99          return geno_sum
100
101     def genotypeCalls(self):
102         geno_call = 0
103         for i in self.GENOTYPE:
104             if len(i.split(':')) > 1:
105                 geno_call += 1
106         return geno_call
107
108     def genotypeCallsHete(self):
109         geno_call_hete = 0
110         for i in self.GENOTYPE:
111             if len(i.split(':')) > 1:
112                 if i.split(':')[0] == '0/1' or i.split(':')[0] == '1/0':
113                     geno_call_hete += 1
114         return geno_call_hete
115
116     def genotypeCallsHomo(self):
117         geno_call_homo = 0
118         for i in self.GENOTYPE:
119             if len(i.split(':')) > 1:
120                 if i.split(':')[0] == '0/0' or i.split(':')[0] == '1/1':
121                     geno_call_homo += 1
```

```
122             return geno_call_homo
123
124         def InbreedingCoeffs(self):
125             match = re.search(r'InbreedingCoeff=.+;',self.INFO)
126             if match:
127                 return match.group().split(';')[0].replace('InbreedingCoeff=','')
128             else:
129                 return 0
130
131         def HaplotypeScores(self):
132             match = re.search(r'HaplotypeScore=.+;',self.INFO)
133             if match:
134                 return match.group().split(';')[0].replace('HaplotypeScore=','')
135             else:
136                 return 0
137
138         def variants(self):
139             if self.ALT == '.':
140                 return 0
141             else:
142                 return 1
```