Genome diversity analysis

‘Plant pathogenomics’ training school
of the COST Action SUSTAIN
Norwich, 3-7April 2017

Pierre Gladieux
INRA Montpellier
pierre.gladieux@inra.fr
@PetrusGladioli

Christophe Lemaire
Université Angers
christophe.lemaire@univ-angers.fr
@Krostif_Angers

Introduction

This course will present the essential points for analysing population subdivision, computing summary statistics of polymorphism and divergence, and testing for natural selection based on genomic data.

We will use a series of programs and Python packages to carry out several exercise. SNP calling will be covered, but not as an exercise, as it requires important computational resources and computing time.

We provide the solution to the different exercises in the corresponding directories. Do not cheat, ask us if you are stuck. We wrote the scripts using a simple syntax, and commenting on the lines of code. Programs are relatively inefficient and slow, but they work, and they can be optimized if you want.

1. SNP calling pipeline

That’s an edit of Daniel Croll’s pipeline (https://github.com/crolllab/wheat-blast) used in the Islam et al. pub http://bmcbiol.biomedcentral.com/articles/10.1186/s12915-016-0309-7

Throughout, variables start with the $ sign: $variable.

We performed all analyses using the genome of the wheat blast strain BR32 as a reference, instead of the rice blast strain 70-15 as in the Islam et al. paper.
Advice: set up your pipeline using a few strains only, and with a subset of the reads only, so that your pipelines run fast and you can quickly see whether they run OK.

We will use our pipeline to make three distinct dataset for pop gen analyses:
- SNPs in wheat blast isolates from Brazil and Bangladesh
  wheat_blast_brazil_bangladesh.snp-only.filters_maxmissing30.vcf
- Variants (SNPs + indels) and invariant positions in wheat blast isolates from Brazil.
  wheat_blast_brazil.with_inv.filters.vcf
- Variants (SNPs + indels) and invariant positions in wheat blast isolates from Brazil, including an isolate of M. grisea as outgroup.
  wheat_blast_brazil_outgroup.with_inv.filters.vcf

1.1. Download raw data

Links to sequencing reads repositories can be found in Table S1 of Islam et al.
http://bmcbiol.biomedcentral.com/articles/10.1186/s12915-016-0309-7

I copied links to the sequencing reads and used command
wget $your_url to download the reads to my favorite computing cluster.
I made one folder $isolate per isolate and put reads R1 and R2 in this folder.

Illumina reads were quality-trimmed. Several options available. Suggestion: trimmomatic.

1.2. Download and index reference genome

The reference genome sequence (isolate BR32) is available at http://genome.jouy.inra.fr/gemo.
Also download GFF3 annotation file, because we'll need it to align transcriptome data using tophat.

1.2.1. Indexing for bowtie2:

```
bowtie2-build BR32.fa BR32
```

1.2.2. Indexing for GATK:

see http://gatkforums.broadinstitute.org/gatk/discussion/1601/how-can-i-prepare-a-fasta-file-to-use-as-reference

- Use CreateSequenceDictionary.jar from Picard to create a .dict file from a fasta file.

```
java -jar CreateSequenceDictionary.jar R= BR32.fa \
                                       O= BR32.dict
```

- Use the faidx command in samtools to prepare the fasta index file.

```
samtools faidx BR32.fa
```

1.2. Alignment of sequencing data against the reference genome

1.2.1. Completely sequenced genomes

We aligned quality-trimmed Illumina short read data against the reference genome. We used bowtie2.

For isolates with paired-end reads:

```
bowtie2 --very-sensitive-local \
        --phred33 \
        --rg-id $isolate \
        --rg PL:ILLUMINA \
        --rg PU: \
        --rg SM:$isolate \
        -x $path_to_refgenome/BR32 \
        -1 $R1reads \
        -2 $R2reads \
        -X 1000 \
        -S $isolate_alignedreads.sam
```

For the one isolate (outgroup) with single-end reads:

```
bowtie2 --very-sensitive-local \
        --phred33 \
        --rg-id $isolate \
        --rg PL:ILLUMINA \
        --rg PU: \
        --rg SM:$isolate \
        -x $path_to_refgenome/BR32 \
```

```
-U $R1reads \
-X 1000 \
-S $isolate_alignedreads.sam
```

1.2.2. RNAseq data:

For all strains collected from the outbreak, an approach called "Field pathogenomics" was used. This data represents transcriptome sequences (RNA) and must be analyzed differently than complete genome sequencing data. We used the RNA-seq aligner tophat2.

```
tophat -G $path_to_refgenome/BR32.gff --min-intron-length 10 \
    --rg-id $isolate --rg-sample $isolate --rg-library $isolate \
    --keep-fasta-order --output-dir $isolate \
    $path_to_refgenome/BR32 R1.fq.gz R2.fq.gz
```

1.3. Sort and index bam files using samtools

Completely sequenced genomes
in each $isolate folder, run:

```
samtools view -uS -h $isolate_alignedreads.sam | samtools sort -@ 4 - \
    $isolate_alignedreads_sorted
samtools index -b $isolate_alignedreads_sorted.bam
```

RNAseq data
in each $isolate folder, run:

```
samtools index -b accepted_hits.bam
```

1.4. Raw variant calling using the Genome Analysis Toolkit GATK

We identified variants in the genomes of the different strains using the Genome Analysis Toolkit (GATK) from the Broad Institute.

1.4.1. Completely sequenced genomes

Perform variant calling using the GATK HaplotypeCaller.
Use option emitRefConfidence to keep all sites, i.e. not only SNPs, but non-SNP variants (e.g. indels) and invariant sites as well.

```
java -jar GenomeAnalysisTK.jar \
    -T HaplotypeCaller \
    -R $path_to_refgenome/BR32.fa \
    -ploidy 1 \
    --input_file $isolate_alignedreads_sorted.bam \
    --emitRefConfidence BP_RESOLUTION \
    --variant_index_type LINEAR \
    --variant_index_parameter 128000 \
    -o $isolate_alignedreads.g.vcf
```

1.4.2. RNAseq data:

To prepare the aligned reads from tophat for variant calling.

```
java -jar GenomeAnalysisTK.jar \
    -T SplitNCigarReads \
```

```
    -R $path_to_refgenome/BR32.fa \
    -I $isolate/accepted_hits.bam \
    -o $isolate/accepted_hits.cigarN.bam \
    -rf ReassignOneMappingQuality -RMQF 255 -RMQT 60 \
    -U ALLOW_N_CIGAR_READS
```

Perform variant calling using the GATK HaplotypeCaller.
This time, we're not keeping all sites. Only sites with variants.

```
java -jar GenomeAnalysisTK.jar \
    -T HaplotypeCaller \
    -R $path_to_refgenome/BR32.fa \
    -ploidy 1 \
    --input_file $isolate/accepted_hits.cigarN.bam \
    --emitRefConfidence GVCF \
    --variant_index_type LINEAR \
    --variant_index_parameter 128000 \
    -o $isolate4/alignedreads.g.vcf
```

1.5 Joint variant calling and filtration using the Genome Analysis Toolkit GATK

We generated a joint variant call set using the GATK GenotypeGVCFs.
Note that we made one data set including RNAseq data and completely sequenced genomes, and the other one with only completely sequenced genomes.
Note that the --variant option is repeated as many times as there are isolates. Use a for loop in python. Edit the list of VCF files to loop through, depending on what you want to include in your dataset (e.g. include Bangladeshi samples, or not).

```
java -jar GenomeAnalysisTK.jar \
        -T GenotypeGVCFs \
        -R $path_to_refgenome/BR32.fa \
        --variant $path_to_isolate1/$isolate1_alignedreads.g.vcf \
        --variant $path_to_isolate2/$isolate2_alignedreads.g.vcf \
        …
        --variant $path_to_isolateN/$isolateN_alignedreads.g.vcf \
        #completely sequenced genomes
        --includeNonVariantSites \
        -o wheat_blast_brazil_outgroup_with_inv.vcf
        #or, for completely sequenced genomes + RNAseq data
        -o wheat_blast_brazil_bangladesh_outgroup.vcf
```

We removed all non-SNP variants, for the dataset including completely sequenced genomes and RNAseq data:

```
java -jar GenomeAnalysisTK.jar \
    -T SelectVariants \
    -R $path_to_refgenome/BR32.fa \
    -selectType SNP \
    --variant wheat_blast_brazil_bangladesh_outgroup.vcf \
    -o wheat_blast_brazil_bangladesh_outgroup.snp-only.vcf
```

Finally, we applied several hard filters to remove low-quality SNPs.

```
QUAL=5000.0
QD=5.0
MQ=20.0
```

```
ReadPosRankSum_lower=-2.0
ReadPosRankSum_upper=2.0
MQRankSum_lower=-2.0
MQRankSum_upper=2.0
BaseQRankSum_lower=-2.0
BaseQRankSum_upper=2.0

java -jar GenomeAnalysisTK.jar \
    -T VariantFiltration \
    -R $path_to_refgenome/BR32.fa \
    #completely sequenced genomes
    --variant wheat_blast_brazil_outgroup_with_inv.vcf \
    #or, for completely sequenced genomes + RNAseq data
    --variant wheat_blast_brazil_bangladesh_outgroup.snp-only.vcf \
    --filterExpression "QD < '+QD+'" --filterName "QDFilter" \
    --filterExpression "QUAL < $QUAL" --filterName "QualFilter" \
    --filterExpression "MQ < $MQ+" --filterName "MQ" \
    --filterExpression "ReadPosRankSum <  $ReadPosRankSum_lower+" \
     --filterName "ReadPosRankSum" \
    --filterExpression "ReadPosRankSum > $ReadPosRankSum_upper" \
    --filterName "ReadPosRankSum" \
    --filterExpression "MQRankSum < $MQRankSum_lower" \
    --filterName "MQRankSum" \
    --filterExpression "MQRankSum > $MQRankSum_upper" \
    --filterName "MQRankSum" \
   --filterExpression "BaseQRankSum < $BaseQRankSum_lower" \
   --filterName "BaseQRankSum" \
   --filterExpression "BaseQRankSum > $BaseQRankSum_upper" \
   --filterName "BaseQRankSum" \
   #completely sequenced genomes
   -o wheat_blast_brazil_outgroup_with_inv.filters.vcf
  #or, for completely sequenced genomes + RNAseq data
  --variant wheat_blast_brazil_bangladesh_outgroup.snp-only.filters.vcf
```

1.6. Selection of SNPs for phylogenomic analyses using vcftools

1.6.1. Completely sequenced genomes

We used vcftools to produce a dataset without the outgroup sequence.

```
vcftools --vcf wheat_blast_brazil_outgroup_with_inv.filters.vcf \
--remove-indv BR29_Mgrisea --recode --out wheat_blast_brazil_with_inv.filters
```

1.6.2. RNAseq

Remove outgroup from the RNAseq dataset, as we won't need it for this workshop.

```
vcftools --vcf wheat_blast_brazil_bangladesh_outgroup.snp-only.filters.vcf \
--remove-indv BR29_Mgrisea --recode --out wheat_blast_brazil_bangladesh.snp-only.filters
```

Generate a SNP dataset for population structure analyses. We only retained SNPs that were called in at least 30% of all analyzed strains. We also removed sites without data (many were caused by the fact that a divergent outgroup sequence was included in our vcf).

```
vcftools --vcf wheat_blast_brazil_bangladesh.snp-only.filters.vcf \
```

```
--max-missing 0.3 --remove-filtered-all --recode \
--out wheat_blast_brazil_bangladesh.snp-only.filters_maxmissing30
```

2. Analysis of population subdivision

2.1. Model-based Bayesian clustering using program Structure.

Structure homepage: http://web.stanford.edu/group/pritchardlab/software/structure_v.2.3.1.html
« The program structure implements a model-based clustering method for inferring population structure using genotype data consisting of unlinked markers. (…) Briefly, we assume a model in which there are K populations (where K may be unknown), each of which is characterized by a set of allele frequencies at each locus. Individuals in the sample are assigned (probabilistically) to populations, or jointly to two or more populations if their genotypes indicate that they are admixed. It is assumed that within populations, the loci are at Hardy-Weinberg equilibrium, and linkage equilibrium. Loosely speaking, individuals are assigned to populations in such a way as to achieve this. »

*2.1.1. **Exercise 1:** Prepare dataset for Structure based on vcf file*

wheat_blast_brazil_bangladesh.snp-only.filters.vcf

The datafile should have the following format:

```
indiv1  genotype_SNP1   genotype_SNP2   …   genotype_SNPn
indiv2  genotype_SNP1   genotype_SNP2   …   genotype_SNPn
…
indivn  genotype_SNP1   genotype_SNP2   …   genotype_SNPn
```

However, the datafile would be much too big if all SNPs were included. Therefore, subsampling is needed. Subsample one SNP per gene at max, and take a subset so that to get ~1k SNPs in the end. Genomic coordinates of predicted genes can be found in gff file refgenome_wheatblast_BR32/BR32.gff.

*2.1.2. **Exercise 2:** Run structure for K ranging from 1 to 5, with ten repeats per K value.*

Edit mainparams file so that it fits the specification of the input file you have prepared in Exercise 1.

For each K value, check that the MCMC has converged towards a single ‘mode’ (i.e. a single clustering solution). Use script plot_barplot_structure_singleK.py to plot stacked barplots for all repeats of a given K.

```
python ./plot_barplot_structure_singleK.py structure_output_K $K
```

Select one repeat per K value for final plotting of clustering solution at all K. List selected output files in list_plot.txt.

```
python ./plot_barplot_structure.py structure_output_K
```

2.2. Network and recombination analysis using program Splitstree.

Splitstree homepage http://www.splitstree.org/
« SplitsTree4 is the leading application for computing unrooted phylogenetic networks from molecular sequence data. Given an alignment of sequences, a distance matrix or a set of trees, the program will compute a phylogenetic tree or network using methods such as split decomposition, neighbor-net, consensus network, super networks methods or methods for computing hybridization or simple recombination networks. »

The objective is to use the phylogenetic network approach 'neighbor-net' implemented in Splitstree to visualize evolutionary relationships while taking into account the possibility of recombination. This method allows visualizing conflicting phylogenetic signals, indicated by the presence of reticulations in the network. Conflicts can be caused by recombination, or incomplete lineage sorting. However, the existence of a recombination signal can still be tested, using the PHI-test, « a simple, powerful test for detecting recombination that can be used regardless of sample history. » The PHI-test calculates the pairwise homoplasy index (PHI) as the mean of the refined incompatibility scores obtained for nearby nucleotide sites along the sequences. Null hypothesis is clonality, and a lower level of homoplasy is expected under strict clonality.

*2.2.1. **Exercise 3:** Make dataset for Splitstree based on vcf file* wheat_blast_brazil_bangladesh.snp-only.filters.vcf

Input file in fasta format. Concatenate all scaffolds. Only SNPs without missing data are included.

*2.2.2. **Exercise 4:** In Splitstree, build a neighbor-net network and test for recombination using PHI-test.*

2.3. Compute summary statistics for polymorphism using egglib library in Python

Egglib homepage http://mycor.nancy.inra.fr/egglib/
« EggLib is a C++/Python library and program package for evolutionary genetics and genomics. Main features are sequence data management, sequence polymorphism analysis, and coalescent simulations. EggLib is a flexible Python module with a performant underlying C++ library and allows fast and intuitive development of Python programs and scripts. »

Egglib library allows the analysis of the sequence polymorphism with the objective of calculating polymorphism summary statistics. In the current state, most of the functionalities are accessible through the egglib.stats.ComputeStats class, meaning that we will have to prepare fasta files (i.e. pseudo Sanger sequences).

Use Egglib to Estimate genomic variability using the number of polymorphic sites (S), nucleotide diversity (π), Watterson's estimate of the population mutation rate θ, the number of haplotypes K, gene diversity He (i.e. expected heterozygosity).

We know focus on completely sequenced genomes only. Bangladeshi samples are excluded, as they introduce missing data.

*2.3.1. **Exercise 5:** Make fasta file using biopython for all coding sequences using* wheat_blast_brazil.with_inv.filters.vcf *and annotation file*

Write two scripts. First script writes, for each scaffold, fasta files including genomic sequences of all isolates. In other words, we want all positions in the genome, not only SNPs. Just read in the VCF file, converting genotype values (0, 1, …, 4 or . for missing data) into A,T,C,G and N nucleotides. Second script extracts coding sequences from scaffold sequences, using Biopython. Read in the GFF file to get gene coordinates, then open scaffold sequences, and cut the section you want.

The Biopython Project is an international association of developers of freely available Python tools for computational molecular biology. Use Biopython to manipulate sequences. What is convenient with Biopython is that it computes the reverse complement, for genes on the - strand, it provides multiple format conversion options, and simple utilities to read and write sequences.

*2.3.2. **Exercise 6:** Compute summary stats for polymorphism at all genes using Egglib library*

Create an instance of the class egglib.stats.ComputeStats and program the calculation of the following statistics: S, ls, thetaW, Pi, K and He. The documentation for the ComputeStats class provides a list of statistics that can be computed. The constructor of the class does not take arguments, but the instances have an add_stat() method that allows you to add statistics to calculate. Most statistics can be computed on the whole sample, so it is not necessary to pass additional arguments in addition to the name of the statistic to be computed. Warning: the list of statistics to calculate is reset to zero if the configure() method is used to specify parameters.

Use numpy library to compute average and standard deviation, and also to get the list of genes in the top 0.1% of Pi.

*2.3.3. **OPTIONAL Exercise 7:** Compute summary stats for polymorphism in non-overlapping or sliding 100kb windows using Egglib library [no script provided for this exercise]*

2.4. Test for departures from the standard neutral model for all genes using the McDonald-Kreitman test.

For each gene, McDonald–Kreitman test is based on a two-way contingency table:

	Fixed	Polymorphic
Synonymous	Ds	Ps
Nonsynonymous	Dn	Pn

Ds: the number of synonymous substitutions per gene
Dn: the number of non-synonymous substitutions per gene
Ps: the number of synonymous polymorphisms per gene
Pn: the number of non-synonymous polymorphisms per gene

The null hypothesis of the McDonald–Kreitman test is that the ratio of nonsynonymous to synonymous variation within a species is going to equal the ratio of nonsynonymous to synonymous variation between species (i.e. Dn/Ds = Pn/Ps).

The ratio of nonsynonymous to synonymous divergence is going to be lower than the ratio of nonsynonymous to synonymous polymorphism (i.e. Dn/Ds < Pn/Ps) when negative selection is at work, and deleterious mutations strongly affect polymorphism.

The ratio of nonsynonymous to synonymous polymorphism is going to be lower than the ratio of nonsynonymous to synonymous divergence (i.e. Dn/Ds > Pn/Ps) under positive selection since beneficial mutations are rapidly fixed and hence contribute more to divergence than to polymorphism.

Proceed in two steps. First, use egglib on coding ingroup sequences to estimate Pn and Ps. Then, use egglib on one ingroup coding sequence and the outgroup coding sequence to estimate Dn and Ds.

*2.4.1. **Exercise 8:** Compute Dn and Ds at all genes using Egglib library*

Edit the script written for exercise 6.

For each gene:

Make a ReadingFrame object matching the bounds of the coding sequence.

Create an egglib.stats.CodingDiversity object from the Align object and the ReadingFrame. Allow up to 30% of missing data. Note that the max_missing argument expects an integer (the maximum number of missing data allowed) and not a relative frequency.

Create two Align containing, respectively, synonymous and nonsynonymous sites. Use the corresponding methods of CodingDiversity. See the corresponding attributes of the instances of the CodingDiversity class.

Create an egglib.stats.Filter object to analyze alignments representing synonymous and non-synonymous variation. The Aligns returned by CodingDiversity contain integers representing the possible 64 codons (range of values from 0 to 63, with the value 64 representing all the missing data).

Create a new ComputeStats object. Specify the S statistic only (number of polymorphic sites, which will be computed on the synonymous sites (Ps) and the non-synonymous sites (Pn)). Use the configure() method of ComputeStats to allow a maximum 30% missing data and pass the Filter object created in exercise 6.

Calculate the statistics for both Align. Store the results in two variables, display them and compare them.

*2.4.2. **Exercise 9:** Compute Pn and Ps at all genes using Egglib library*

Make fasta files for all genes, including the outgroup. Use same scripts as for Exercise 5, but this time using VCF file including outgroup data (wheat_blast_brazil_outgroup.with_inv.filters.vcf).

Computing Dn and Ds is not currently implemented in Egglib. But there's a workaround. For each gene, compute Pn and Ps on a pair of sequence including one sequence for the ingroup and one sequence for the outgroup; these values should be approximately equal to Dn and Ds.

*2.4.3. **Exercise 10:** Compute contingency tables and test for equality between ratios using Fisher exact tests.*

Use scipy to perform Fisher exact tests.

2.5. Test for balancing selection.

Use an approach based on summary statistics of polymorphism and divergence. See Figure 4 in Bakker et al.

« The genetic signatures of long-term balancing selection at a locus can roughly be divided into three categories. The first signature is that the distribution of allele frequencies will be enriched for intermediate frequency alleles. This occurs because the selected locus itself is likely at moderate frequency within the population and, thus, neutral linked loci will also be at intermediate frequency. The second signature is the presence of trans-specific polymorphisms, which are polymorphisms that are shared among species. This is a result of alleles being maintained over long evolutionary time periods, sometimes for millions of years. The third signature is an increased density of polymorphic sites. This is due to linked neutral loci sharing similar deep genealogies as that of the selected site, increasing the probability of observing mutations at the neutral loci » (source).

Here, we're going to use these three expected signatures of balancing selection to identify loci having a possible history of balancing selection.

*2.5.1. **Exercise 11:** Compute the following statistics: number of segregating sites per bp, nucleotide diversity Pi, Tajima's D, the number of protein variants, the maximum number of differences per bp between pairs of sequences*

Compute S, Pi and D using Egglib:
S and Pi already calculated for Exercise 6. S and Pi are high under balancing selection (signature 3, see above).
Tajima's D is a summary of the site frequency spectrum. Tajima's D takes positive values if there is an excess of intermediate frequency alleles (signature 1, see above).

Compute the number of protein variants using Biopython:
The number of protein variants should be higher under balancing selection (signature 3, see above).

Compute the maximum number of differences per bp between pairs of sequences using your own script:
The maximum number of differences per bp between sequence pairs is a proxy for gene tree depth. Gene trees are expected to be deeper under balancing selection (signature 2, see above).

For instance, use Egglib Align objects and the following piece of code (adapted from this script):

```
for i in P1: # for each sequence in pop1...
    for j in P2: #for sequence in pop2...
        seqA = align[i][1]
        seqB = align[j][1]
        zippedSeqs = zip(seqA,seqB)
        diffs = sum(sA != sB for sA, sB in zippedSeqs if sA != "N" and sB != "N")
        sites = len([site for site in zippedSeqs if site[0] != "N" and site[1] != "N"])
        pairwiseSum += 1.0*diffs/sites
```

Summarize results using a principal component analysis in R. Identify genes harboring possible signatures of balancing selection.