CompPopGenWorkshop2019 Documentation

Stephan Schiffels

May 09, 2019

Contents:

1 Setting up 3
- 1.1 The Computational Infrastructure for this Workshop . . . 3
- 1.2 Basic Usage . . . 3
- 1.3 Notebooks . . . 4
- 1.4 Working with Bash . . . 5
- 1.5 Back to Notebooks . . . 7
- 1.6 Some Plotting with Python . . . 7

2 Bam Files 11
- 2.1 samtools: Converting, Filtering SAM and BAM Files . . . 11
- 2.2 Bash programming . . . 12
- 2.3 Basic programming in AWK . . . 13
- 2.4 Calculating coverage with samtools . . . 14
- 2.5 EXTRA: Genotype calling from bam files . . . 14

3 Principal Components Analysis (PCA) 17
- 3.1 Genotype Data . . . 17
- 3.2 How PCA works . . . 18
- 3.3 Preparing the parameter file . . . 18
- 3.4 Population lists vs. Projection . . . 19
- 3.5 Running smartPCA . . . 19
- 3.6 Plotting modern populations . . . 20
- 3.7 Adding ancient individuals . . . 24

4 F Statistics and Treemix 27

5 Admixture modelling with *qpWave* and *qpAdm* 29
- 5.1 Overview: summarizing a matrix of F4 statistics . . . 29
- 5.2 Data . . . 30
- 5.3 Preparing "left" and "right" populations . . . 30
- 5.4 Preparing the parameter file . . . 31
- 5.5 Running the programs . . . 31
- 5.6 Reading the *qpWave* log file . . . 32
- 5.7 Reading the *qpAdm* log file . . . 34

6 Estimating Admixture times with *Alder* 37
- 6.1 Overview . . . 37

6.2 Data . . . 37
6.3 Preparing the parameter file . . . 38
6.4 Running the programs . . . 38
6.5 Reading the main output file (weighted LD table) . . . 39
6.6 Reading the log file . . . 39
6.7 Plotting weighted LD decay curve in R . . . 42

7 Estimating Admixture Graphs with `qpGraph` 43
7.1 Overview method . . . 43
7.2 Data . . . 44
7.3 Preparing the parameter file . . . 44
7.4 Preparing the graph topology . . . 45
7.5 Running the program . . . 45
7.6 Reading the output files . . . 46
7.7 Adding new groups to the scaffold graph . . . 47
7.8 Continuing to fit new groups . . . 47
7.9 Test the robustness of the graph topology . . . 47
7.10 Adding admixture edges . . . 47

8 MSMC 51
8.1 Data . . . 51
8.2 Generating consensus sequences for each sample . . . 51
8.3 Combining samples . . . 52
8.4 Running MSMC2 for estimating the effective population size . . . 54
8.5 Estimating population separation history . . . 54
8.6 Plotting in Python . . . 55

This is the documentation for the Workshop held during January 22-24, 2019 at the Max-Planck-Institute for the Science of Human History (MPI-SHH) in Jena, organised by Stephan Schiffels (MPI-SHH), Choongwon Jeong (MPI-SHH) and Benjamin Peter (MPI-EVA). Important contributions came from Kay Prüfer and Cosimo Posth!

CHAPTER 1

Setting up

1.1 The Computational Infrastructure for this Workshop

For the practical sessions and exercises of this workshop, we have set up a Cloud Server with 32 CPUs and 64Gb of Memory in total with a separate user account for each participant. As participant, you will have received your own username and password for accessing the server. The simplest way to access the server is via Jupyter:

https://c107-224.cloud.gwdg.de/hub/login

where you can enter your username and password and are then logged into Jupyter. For reference, you find documentation for the Jupyter interface here

More expert users can also access the server via custom terminal client software (such as `Terminal` on Mac OS X, or `putty` on Windows), via:

```
ssh <USERNAME>@c107-224.cloud.gwdg.de
```

1.2 Basic Usage

When you first access Jupyter, you will get a file browser view of your home directory on the server. In the beginning, your home directory will be empty, and will be populated with notebooks and files throughout this workshop.

To create a new text file, click on *New* (in the upper right corner) and then *Text File*, which opens a text editor within your browser. You can now add content into the file, or edit existing content and save. The filename can be changed by clicking into the Filename on top. You can now go back to your file browser window and update using the button with the two arrows in the upper right corner, and you should see your text file saved in your home directory.

You can also use Jupyter to open a Terminal within the browser: Click on *New* and then *Terminal*, which will open a terminal window in a separate browser tab. You can enter Unix Bash commands to change directories, view files or execute programs (as we will learn below).

Finally, you can create new Folders by clicking on *New* and then *Folder*. To rename the new folder, click on the checkbox beside the new folder, and click the *Rename* button on top, which appeared. To change into the new folder,

click on it. To move back, click on the parent folder appearing on top of the file browser.

Exercise

Create a new folder called `hello`, and a text file within that folder using Jupyter. Name that text file `hello.txt` and fill it with arbitrary content, such as "Hello, World!". Then open a terminal and output the contents of the new text file typing `cat hello/hello.txt` followed by ENTER.

Note: While the Jupyter terminal and Jupyter Text Files are different ways to interact with the server, both access the same file system. So files created with the Text Editor are saved in your home directory, and can be accessed via the terminal, and vice versa: Files created via the Terminal can be accessed via the Text Editor, by simpling clicking on them in the Jupyter File Browser.

1.3 Notebooks

Notebook can be loaded for different underlying kernels: bash, python and R. Notebooks are useful to document interactive data analysis. It combines code cells with markdown cells. A markdown cell can contain text, math or headings.

Exercise

Create a new bash notebook. Then select in the dropdown list above "Markdown". Type `# Bash Exercises` into the cell, press Shift-Enter and watch. Then type `This is text with *italic* and **bold** letters`. Then again type Shift-ENTER and watch.

Hint: To change the cells, double click into them.

Code cells can be used to write arbitrary code, execute it and get the results printed back into the Notebook.

Exercise

A new empty Code cell should have been added to the Notebook in the last step. Click into this code cell and type `ls`. This should output the current directories and files into the notebook. Into a new cell enter `NAME="Hello World"` and in the line below (same cell) `echo $NAME`. Then again Shift-ENTER.

You can use Bash notebooks to perform standard Unix tasks and run programs throughout this workshop. That way, you have always documented what you did.

In Python 3 notebooks you can plot things: Create a new python3 notebook, and use this boilerplate code in the first cell:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

Then plot something:

Exercise

Create a simple plot using `plt.plot([1, 2, 3], [5, 2, 6])`

1.4 Working with Bash

Bash denotes both a scripting language and the interactive system that is used with Terminals (and Bash notebooks). In the following, we will learn some basic use cases and commands.

1.4.1 Simple commands

The most important commands in a Unix shell are `ls`, `cd` and `mkdir`. In general, all commands in a Unix shell are entered by typing the command and then typing ENTER.

Exercise

Open a Terminal and execute the `ls` command by typing `ls`. It should list you the contents of your home directory. Then run `mkdir testdir` to create a new directory. Then type `cd testdir` to change into that new directory.

Let's learn some more commands. Above you have already used `cat` to output the contents of a file to the screen. Another important command is `echo`, which also prints stuff to the screen, but not from a file but from a string that you give it. For example:

Exercise

try the command `echo "Hello, how are you?"` in your terminal.

Another important command is `grep`, which lets you filter out lines of a file that contain certain strings. As a basic example, consider this file listing genotyped individuals: `/data/pca/genotypes_small.ind`. You can for example now list all French individuals via `grep French /data/pca/genotypes_small.ind`.

Exercise

try the above listing of French individuals. Also try other population names, like "Polish" or "Lebanese". Instead of just `grep`, try `grep -c` and see what that does (find out using `man grep` on a terminal.

Hint: In bash, you can use tab-expansion. Instead of heaving to spell out `/data/pca/genotypes_small.ind`, you can try typing `/da<TAB>pca/geno<TAB>`.

In general, in order to get help on any bash command, including the above, you can use `man` to review the documentation. For example, in a Terminal window, run `man mkdir` to view the documentation of the `mkdir` command. Use Space to move forward through the documentation, or the UP- and DOWN- keys. Use `q` to quit the view.

Warning: Use `man <COMMAND>` only in the Terminal, not in a bash-notebook!

1.4.2 Pipes

We will use Pipes in several places in this workshop. The basic idea is to combine multiple bash commands into powerful workflows. As an example, we'll use a simple bash pipeline to count the number of populations in our individual file. We need some new commands for that. First, let's look at the structure of the file at question. The command `head /data/pca/genotypes_small.ind` outputs:

```
    Yuk_009 M    Yukagir
Yuk_025 F    Yukagir
Yuk_022 F    Yukagir
Yuk_020 F    Yukagir
  MC_40 M    Chukchi
Yuk_024 F    Yukagir
Yuk_023 F    Yukagir
  MC_16 M    Chukchi
  MC_15 F    Chukchi
  MC_18 M    Chukchi
```

So this file contains three columns, with variable numbers of leading whitespace in each row. For counting the number of populations, we need to first cut out the third column of this file. A useful command for this is the command `awk {print $3}`, which you will learn more about later. Let's now build our first pipe. We will pipe the output of `head /data/pca/genotypes_small.ind` into `awk {print $3}`, by running:

```
head /data/pca/genotypes_small.ind | awk '{print $3}'
```

This means "Take the output of the first command, and pipe it into the input of the second command. The result is:

```
Yukagir
Yukagir
Yukagir
Yukagir
Chukchi
Yukagir
Yukagir
Chukchi
Chukchi
Chukchi
```

OK, so now we have to sort these population names, and of course there is a command for that: `sort`.

Exercise

Build a pipeline that extracts the third column of the "ind" file and sorts it. Don't use `head` in the beginning, but pipe the entire file through the awk script. You already know which command outputs an entire file!

Hint: Use `head` frequently to test pipelines, by putting it at the end of a pipeline and only look at the first 10 rows of your pipeline output.

OK, so finally, we need to remove duplicates from the sorted population names, and the appropriate command for that is the `uniq` command (which works only on sorted input).

Exercise

Extend the pipeline from above to output unique population labels.

The first ten rows of that pipeline output (verfiable with `head`) should read:

```
Abkhasian
Adygei
Albanian
Aleut
Aleut_Tlingit
Altaian
Ami
Armenian
Atayal
Balkar
```

The final step is to count the lines. The command for that is `wc -l`, which counts the lines from its input.

Exercise

Extend the pipeline one last time, by piping the output into `wc -l`.

We can now do one final step and augment the pipeline we have built to output not only the total number of populations, but also the number of individuals per population. The trick here is to use the `uniq -c` command, instead of just `uniq`. That will output the unique population labels alongside the number of times it is seen. The final pipeline then reads:

```
cat /data/pca/genotypes_small.ind | awk '{print $3}' | sort | uniq -c
```

You can run this and store the result in a file, using standard redirection. This is achieved via the operator `> FILENAME`. So for example, you can extend the above pipeline with `> population_frequencies.txt` to output the result in a file called `population_frequencies.txt`.

1.5 Back to Notebooks

Exercise

You should now try to implement the step-by-step build up of that pipeline in a bash notebook. You can find my own example here.

1.6 Some Plotting with Python

As a final exercise, we now want to plot the population frequencies. As a first step, we again open a python notebook and include the already known boilerplate in a code cell:

```
%matplotlib inline
import matplotlib.pyplot as plt
```

In addition, we need to load the pandas library:

```
import pandas as pd
```

We can now load the `population_frequencies.txt` (or however you have called it) into python, using pandas `read_csv` function.

Exercise

Look up some documentation for the `read_csv()` function, by typing in a python notebook `?pd.read_csv`. This should open a little extra window with help information. For reading the population frequencies, you will need options `delim_whitespace` and `names`. Look them up.

You can now load in the data, using the command:

```
dat = pd.read_csv(FILENAME, delim_whitespace=True, names=["nr", "pop"])
```

Exercise

Execute the above command using the correct filename (make sure you specify the correct path and directory to it, if the file is not in the same directory as your notebook).

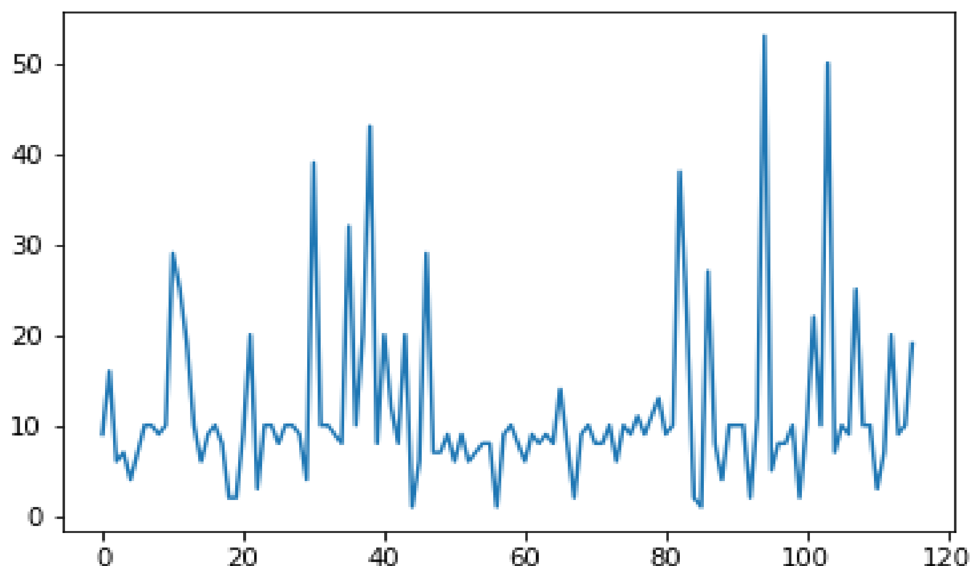
Note: The column names, here "nr" and "pop" are arbitrary and just denote the names for the two columns in the file.

You can verify that loading this data has succeeded by just typing `dat` into a new code cell and checking that it outputs a nicely formatted dataframe with two columns and an index.

We can now go ahead and plot this. As a first step, we can just try the simplest way of plotting:

```
plt.plot(dat["nr])
```

which should yield something like this:



Now this already shows that the majority of populations has something like 10 individuals, but we would like to also

display the population labels, and sort the values. As first step, we first create a sorted version of this dataframe using the following command:

```
dat_sorted = dat.sort_values(by="nr")
```

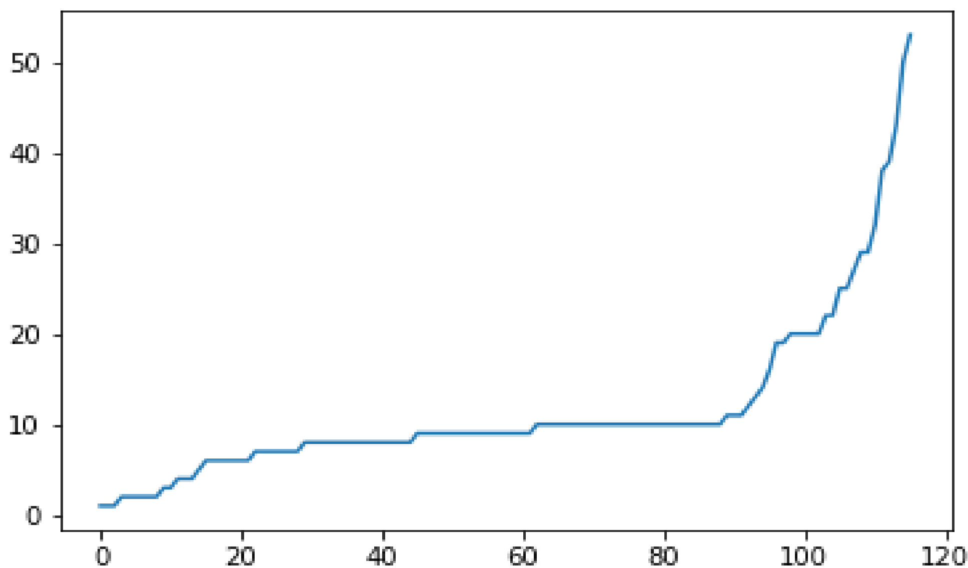
Exercise

Look up the sort_values method with ?dat.sort_values and see what it does and what the by= option does.

We can now make a sorted plot. For that we actually need two arguments to plt.plot, one for specifying the x coordinate of each point, and one for the y coordinate for each point (before we didn't need that because plt.plot assumes a default x value if none is given, which is the index in the dataframe. Now this got resorted, so we have to spcify it). Here is the command:

```
x = range(len(dat_sorted))
y = dat_sorted["nr"]
plt.plot(x, y)
```

Here, the range(len(dat_sorted)) just produces an array which looks like [0, 1, 2, 3, 4, ..., so it simply creates a list from 1 to the number of populations minus one. This command produces a sorted version of the previous plot:



OK, so now the final step is to add the labels. We of course have the labels already in our dataframe, under the "pop" column. The function that adds labels is the plt.xticks function.

Exercise

Look up the documentation for the plt.xticks function, similarly as in the previous exercise.

For the final plot, we now put this together, and we also increase the figure size a bit to accommodate all the population labels. Together we have:

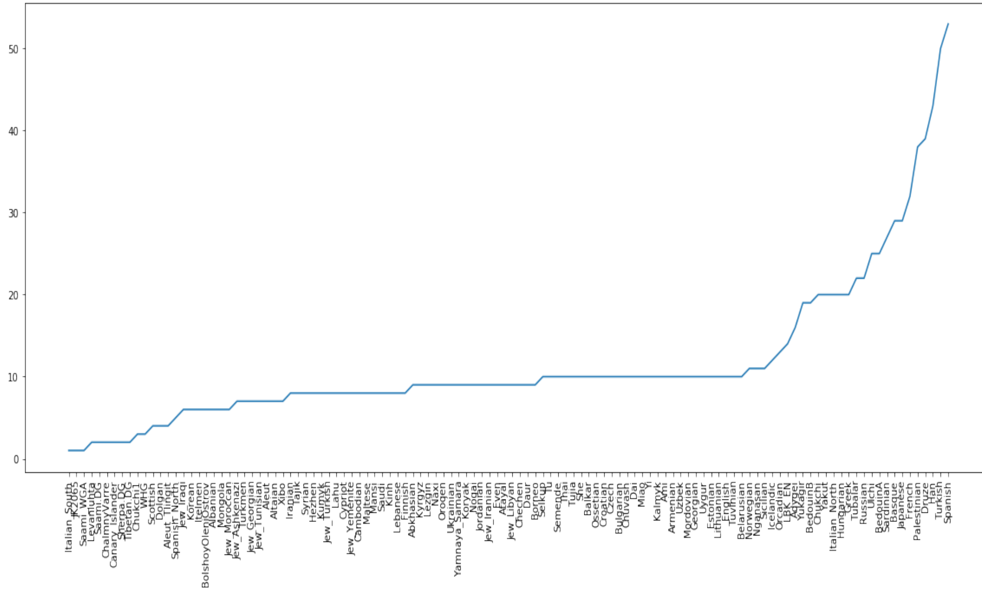
```
dat_sorted = dat.sort_values(by="nr")
y = dat_sorted["nr"]
```

(continues on next page)

(continued from previous page)

```
x = range(len(y))
xticks = dat_sorted["pop"]
plt.figure(figsize=(20,8))
plt.plot(x, y)
plt.xticks(x, xticks, rotation="vertical");
```

which gives this final plot:



You can find a solution notebook for this part here.

CHAPTER 2

Bam Files

BAM is a binary format for reads and accompanying information about their alignments to a reference genome. They are for instance generated by bwa, a popular alignment program. SAM is the human readable version of BAM. A full description of both formats is available here: https://samtools.github.io/hts-specs/SAMv1.pdf

2.1 samtools: Converting, Filtering SAM and BAM Files

A commandline-tool called samtools can be used to work with BAM and SAM. You can run samtools without any parameters to get an overview of parameters and options. The command `man samtools` shows you a longer documentation.

The basic pattern of usage for samtools is

```
samtools [command] [-paramter] [-parameter] bam/sam-file
```

A useful command is `view` which converts a BAM file to SAM. The command `tview` allows you to view the alignments.

I prepared two files that you can try these commands on:

```
/data/bam/workshop1.bam
/data/bam/workshop2.bam
```

Exercise

Run `samtools tview` on workshop2.bam and use space to scroll until you see an alignment. Try `?` to see further options.

Hint: If you want to find a position with lots of data, check out `samtools tview -p 17:6944949 /data/bam/workshop1.bam`. You can also add the reference genome, which will render the reads dif-

ferently: `samtools tview -p 17:6944949 /data/bam/workshop1.bam --reference /data/bam/whole_genome.fa`

Exercise

Run `samtools view workshop1.bam | less`. What chromosome do the first couple of sequences align to? How does the output change if you use `-h` which adds the SAM header information?

The view command can also be instructed to print specific regions (as long as the bam file is sorted and indexed): `samtools view workshop1.bam 17` will only print alignments on chromosome 17 and `samtools view workshop1.bam 17:6944949-6947242` only alignments overlapping the specified coordinates.

Samtools view also allows for alignments to be filtered. You can specify which alignments should be printed using the option `-f` or exclude alignments with `-F`. These options are followed by a integer for the flags that should be set/not set (see format description pdf). `samtools view -f 16` for instance, selects all sequences that align on the minus strand.

2.2 Bash programming

To get some basic statistics from BAM-files it is useful to know a couple of shell-commands and how to chain them together. You can get a full decription on these commands by running `man [command]`. Commands can be chained together in bash to form longer commands using the pipe-operator `|`. The output of each command in the chain will be used as input for the next command.

```
command1 | command2 | command3
```

Here are some basic shell-commands:

head and tail The commands `head` and `tail` give you the first 10 lines of the beginning and 10 lines of the end of a file, respectively. Parameter `-n` can be used to specify how many lines.

wc The command `wc` (for "word count") counts the number of characters, words and lines in a document. Use parameter `-c` for just the character count, `-w` for just the words and `-l` for just the number of lines.

Exercise

Run `samtools view /data/bam/workshop1.bam | head`. Count the number of lines that are printed using `wc`. Specify the number of lines to be printed with `head -n`.

Exercise

How many sequences in total are aligned workshop1.bam? How many of these are single reads (not paired end)?

---

cut Will only print specific columns of a text table. Parameter `-f` specifies which column you want printed. `cut -f 1,2` will for instance print the first and second column from a file.

Exercise

Run `samtools view /data/bam/workshop1.bam | cut -f 3`. What does this column of the SAM format mean?

---

tr This command (for "transpose") will substitute a set of characters by diffent characters. For instance, `tr 'ABC' 'abc'` will substitute all occurrences of A, B or C by a, b and c, respectively. The command can also delete characters. `tr -d "N"` will for instance delete all occurrences of "N" from the input.

grep grep will only print lines matching a specific pattern. Example: `grep AAAACCCC` will print all lines containing "AAAACCCC". `grep "^Word"` will print lines starting with "Word".

Exercise

Convert the sequences from the SAM output of workshop1.bam to lowercase using tr. Use grep to check for sequences that contain the character "N".

---

sort Sorts the input alphabetically. Use option `-k` to specify the column to sort by and `-n` if you want to sort numerically

uniq Only print unique occurrences of lines on the input. Input must be sorted (see `sort`). Use option `-c` if you'd like to get counts of occurrences.

Exercise

How many sequences align to each chromosome in workshop1.bam and workshop2.bam? Seeing how many sequences align to chromosome X and chromosome 7 (which is similar in size to X) for workshop2.bam, would you say this individual is male or female?

2.3 Basic programming in AWK

awk is a simple programming language that is particularly useful when processing line-wise input. The basic format of any awk program looks like this:

```
BEGIN{ }
{ }
END{ }
```

Everything in curly brackets in the first line is going to be done before the first line is read. The middle line specifies everything that should be done for each line. The last line says what should be done after the last line. A simple awk program that counts the number of lines may be written like this:

```
BEGIN{ line=0 }
{ line=line+1 }
END{ print line }
```

Since awk keeps track of the number of a line in the variable /NR/, you can simplify this program to just one line:

```
END{ print NR }
```

The formatting of awk programs doesn't matter. This makes it easy to specify programs on one line inbetween other shell-commands. For instance:

```
# long version:
samtools view workshop1.bam 17 | awk 'BEGIN{ line=0 }{ line=line+1 }END{ print line }'

# simplified version:
samtools view workshop1.bam 17 | awk 'END{ print NR }'
```

Awk can also select specific columns (like `cut` does). To refer to a specific column, you add a `$` before the number of the column. This prints the first column from a file:

```
{ print $1 }
```

To count the characters in each line (like `wc -c`), you can use the function length():

```
{ print length($1) }
```

Exercise

Calculate the average size of sequences in workshop1.bam and workshop2.bam. Select only sequences that are not paired-end.

Exercise

Calculate the number of GC and AT bases in workshop2.bam. Extra question: is the GC content in workshop1.bam different and why?

2.4 Calculating coverage with samtools

`samtools depth` gives the number of sequences covering sites. With `-a`, all positions are given, also those not covered. Example:

```
samtools depth /data/bam/workshop1.bam | less
```

Exercise

Calculate the average coverage on chrX and chr7 for workshop2.bam.

Exercise

Calculate the average coverage for the region 17:6944949-6947242 on workshop1.bam.

2.5 EXTRA: Genotype calling from bam files

When several sequences overlap a position in the nuclear genome, then the genotype of the carrier can be inferred. How this is done best in every case goes beyond the scope of the workshop. However, when ancient DNA damage is low, you can use samtools together with a program called `bcftools` to produce genotype calls in VCF format.

```
samtools mpileup -v -f [reference_genome] -I [input-bam] | bcftools call -m > output.
↪vcf
```

The VCF format is described here: http://www.internationalgenome.org/wiki/Analysis/vcf4.0/

Exercise

Run the above command on workshop1.bam using the reference genome /data/bam/whole_genome.fa. Have a look at the output and see how many differences you observe to the human reference.

The solution notebook for this session is here

CHAPTER 3

Principal Components Analysis (PCA)

Principal components analysis (PCA) is one of the most useful techniques to visualise genetic diversity in a dataset. The methodology is not restricted to genetic data, but in general allows breaking down high-dimensional datasets to two or more dimensions for visualisation in a two-dimensional space.

3.1 Genotype Data

This lesson is also our first contact with the genotype data used in this and most of the following lessons. The dataset that we will work with contains 1,340 individuals, each represented by 593,124 single nucleotide polymorphisms (SNPs). Those SNPs have exactly two different alleles, and each individual has one of four possible values at each genotype: homozygous reference, heterozygous, homozygous alternative, or missing. Those four values are encoded 2, 1, 0 and 9 respectively.

The data is laid out as a matrix, with columns indicating individuals, and rows indicating SNPs. The data itself comes in the so-called "EIGENSTRAT" format, which is defined in the Eigensoft package used by many tools used in this workshop. In this format, a genotype dataset consists of three files, usually with the following file endings:

`*.snp` The file containing the SNP positions. It consists of six columns: SNP-name, chromosome, genetic positions, physical position, reference allele, alternative allele.

`*.ind` The file containing the names of the individuals. It consists of three columns: Individual Name, Sex (encoded as M(ale), F(emale), or U(nknown)), and population name.

`*.geno` The file containing the genotype matrix, with individuals laid out from left to right, and SNP positions laid out from top to bottom.

Exercise

Explore the three files used in the workshop using the terminal. They are located unser `/data/pca/genotypes_small.*`. Use the bash terminal, and use `less -S <FILENAME>` to view each file and skim through it to get a feeling for the data.

Exercise

Start a new bash notebook and look at the first 20 rows of both the `ind` and the `snp` file using `head -20`. For the `geno` file, restrict to the first 100 columns using `cut -c 1-100`, which you can also use in a pipe together with `head -20`.

Exercise

Confirm that there are 1,340 individuals in the dataset. Confirm that this number equals the number of rows in the `ind` file, and the number of columns in the `geno` file..

Hint: You can use `head -1` to restrict to the first line of a file, and `wc -c` to count the number of characters (i.e. columns) in that file.

Exercise

Count how many individuals per population there are. Hint: You can use the Unix tools `awk '{print $3}'`, `sort` and `uniq -c` to achieve that.

You can find my solution notebook to these exercises here

3.2 How PCA works

To understand how PCA works, consider a single individual and its representation by its 593,124 markers. Formally, each individual is a point in a 593,124-dimensional space, where each dimension can take only the three possible genotypes indicated above, or have missing data. To visualise this high-dimensional dataset, we would like to project it down to two dimensions. But as there are many ways to project the shadow of a three-dimensional object on a two dimensional plane, there are many (and even more) ways to project a 593,124-dimensional cloud of points to two dimensions. What PCA does is figuring out the "best" way to do this project in order to visualise the major components of variance in the data.

3.3 Preparing the parameter file

For actually running the analysis, we use a software called `smartPCA` from the Eigensoft package. As many other tools from this and related packages, `smartPCA` reads in a parameter file which specifies its input and output files and options. The basic format of the parameter file with one extra option (lsqproject) looks like this:

```
genotypename: <GENOTYPE_DATA>.geno
snpname: <GENOTYPE_DATA>.snp
indivname: <GENOTYPE_DATA>.ind
evecoutname: <OUT_FILE>.evec
evaloutname: <OUT_FILE>.eval
poplistname: <POPULATION_LIST_FILE>.txt
lsqproject: YES
numoutevec: 4
numthreads: 1
```

Here, the first three parameters specify the input genotype files, as discussed above. The next two rows specify two output file names, typically with ending `*.evec` and `*.eval`. The parameter line beginning with `poplistname` contains a file with a list of populations used for calculating the principal components (see below). The option `lsqproject` is important for applications including ancient DNA with lots of missing data, which I will not elaborate on. For the purpose of this workshop, you should use `lsqproject:  YES`. The last option `numoutevec` specifies the number of principal components that we compute.

3.4 Population lists vs. Projection

The parameter named `poplistname` is a very crucial one. It specifies the populations whose individuals are used to calculate the principal components. Why not just all of them you ask? For two reasons: First, there are simply too many of them. As you may have found out in the exercise above there are more than 500 ancient and modern populations available in the dataset, and we don't want to use all of them, since the computation would take too long. More importantly, however, we generally try to avoid using ancient samples to compute principal components, to avoid specific ancient-DNA related artefacts affecting the computation.

So what happens to individuals that are not in populations listed in the population list? Well, fortunately, they are not just ignored, but "projected". This means that after the principal components have been computed, *all* individuals (not just the one in the list) are projected onto these principal components. That way, we can visualise ancient populations in the context of modern genetic variation. While that may sound a bit problematic at first (surely there must be variation in ancient populations that is not represented well by modern populations), but it turns out to be nevertheless one of the most useful tools for this purpose. The advantage of avoiding ancient-DNA artefacts and batch effects to affect the visualisation outweighs the disadvantage of missing some private genetic variation components in the ancient populations themselves. Of course, that argument breaks down once the analysed populations become too ancient and detached from modern genetic variation. But for our purposes it will work just fine.

For this workshop, I prepared two population lists:

```
/data/pca/WestEurasia.poplist.txt
/data/pca/AllEurasia.poplist.txt
```

As you can tell from the names of the files, they specify two sets of modern populations representing West Eurasia or all of Europe and Asia, respectively.

Exercise

Look through both of the population lists and google some population names that you don't recognise to get a feeling for the ethnic groups represented here.

3.5 Running smartPCA

Now go ahead and prepare a parameter file according to the layout described above.

Hint: Put all filenames with their absolute path into the parameter file. To prepare the parameter file, you can use the so-called "Heredoc" syntax in bash, if you are familiar with it (see also solution notebook below). Alternatively, you can use the Jupyter file editor to create the parameter file.

… and run smartPCA using the command `smartpca -p PARAMS_FILE > smartpca.log`. Here, I'm using bash redirection of the log output of `smartpca` into a log file called `smartpca.log`, which may be useful for trouble shooting.

Exercise

Run smartpca with the prepared parameter file.

Note: Running smartPCA with this dataset and on a single CPU takes between 30 and 60 minutes.

To facilitate further processing, I have put the results file into /data/pca/results/pca.WestEurasia.* and /data/pca/results/pca.AllEurasia.*

3.6 Plotting modern populations

There are several ways to make nice publication-quality plots (Excel is usually not one of them). Popular tools include R and matplotlib . Both frameworks can be used within the Jupyter Notebook interface, and here I opted for matplotlib.

I suggest that you start a new Jupyter Python Notebook, and load a couple of essential libraries in the first code cell:

```
%matplotlib inline
import pandas as pd
import matplotlib.pyplot as plt
```

Let's have a look at the main results file from smartpca, the *.evec file, for example by running this simple bash command in the notebook using !head EVEC_FILE, where EVEC_FILE should obviously replaced with the actual filename of the PCA run.

Hint: You can run any bash command in a python notebook by preceding it with the ! sign. This comes in very handy at times!

You should find something like:

```
#eigvals:      6.289      3.095      2.693      2.010
     I001     -0.0192       0.0353     -0.0024     -0.0084      Ignore_Iran_
↪Zoroastrian(PCA_outlier)
     I002     -0.0237       0.0372     -0.0018     -0.0133      Ignore_Iran_
↪Zoroastrian(PCA_outlier)
IREJ-T006     -0.0226       0.0417      0.0045      0.0003      Iran_Non-Zoroastrian_Fars
IREJ-T009     -0.0214       0.0404      0.0024     -0.0064      Iran_Non-Zoroastrian_Fars
IREJ-T022     -0.0165       0.0376     -0.0003     -0.0106      Iran_Non-Zoroastrian_Fars
IREJ-T023     -0.0226       0.0376     -0.0031     -0.0101      Iran_Non-Zoroastrian_Fars
IREJ-T026     -0.0203       0.0373     -0.0009     -0.0103      Iran_Non-Zoroastrian_Fars
IREJ-T027     -0.0241       0.0392      0.0025     -0.0072      Iran_Non-Zoroastrian_Fars
```

The first row contains the eigenvalues for the first 4 principal components (PCs), and all further rows contain the PC coordinates for each individual. The first column contains the name of each individual, the last row the population. To load this dataset with python, we use the pandas package, which facilitates working with data in python. To load data using pandas, use the read_csv() function.

Exercise

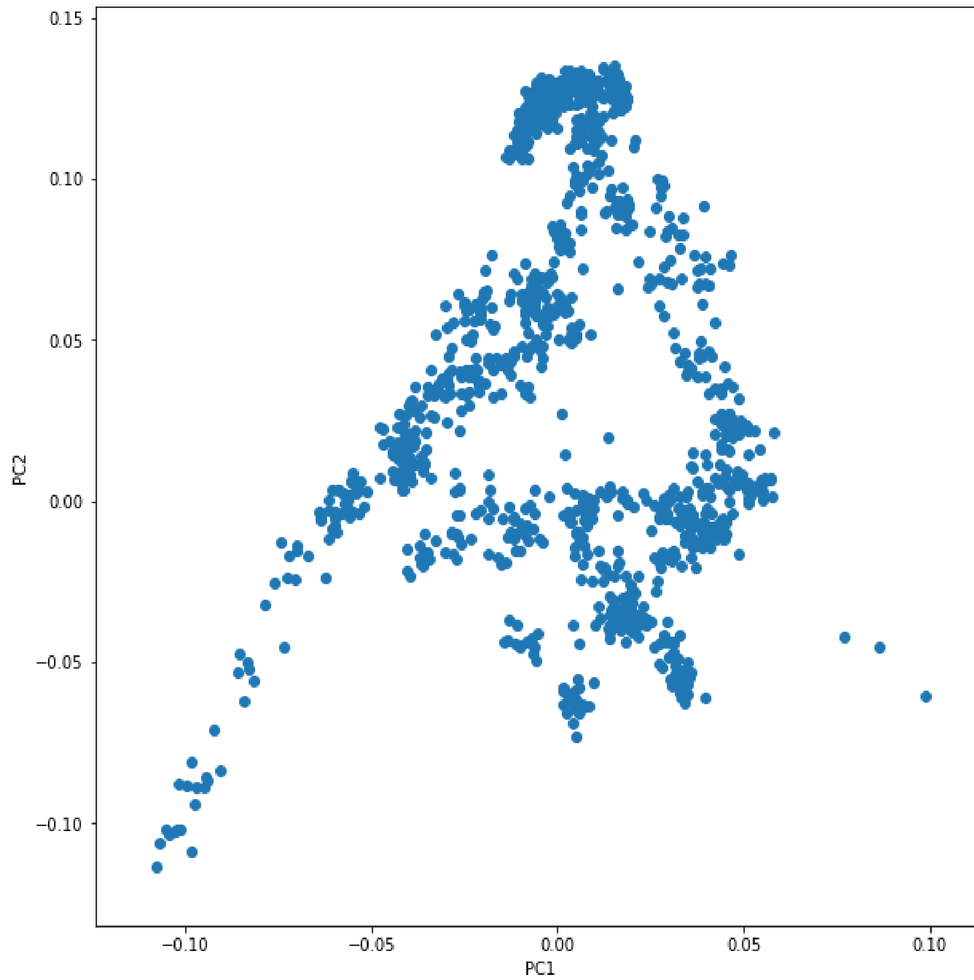
Load one of the two PCA results files with ending *.evec. You need to skip the first row and name the columns manually. Use "Name", "PC1", . . . "PC4", "Population" for the column names. Google documentation for read_csv()

to ensure that tabs and spaces are considered field delimiters, that the first row is skipped, and that the column names are correctly entered.

You should now have a pandas dataframe which looks like this:

```
Name     PC1     PC2      PC3      PC4      Population
        I001     -0.0192      0.0353     -0.0024     -0.0084     Ignore_Iran_
↪Zoroastrian(PCA_outlier)
        I002     -0.0237      0.0372     -0.0018     -0.0133     Ignore_Iran_
↪Zoroastrian(PCA_outlier)
   IREJ-T006     -0.0226      0.0417      0.0045      0.0003     Iran_Non-Zoroastrian_
↪Fars
   IREJ-T009     -0.0214      0.0404      0.0024     -0.0064     Iran_Non-Zoroastrian_
↪Fars
   IREJ-T022     -0.0165      0.0376     -0.0003     -0.0106     Iran_Non-Zoroastrian_
↪Fars
   IREJ-T023     -0.0226      0.0376     -0.0031     -0.0101     Iran_Non-Zoroastrian_
↪Fars
   IREJ-T026     -0.0203      0.0373     -0.0009     -0.0103     Iran_Non-Zoroastrian_
↪Fars
   IREJ-T027     -0.0241      0.0392      0.0025     -0.0072     Iran_Non-Zoroastrian_
↪Fars
```

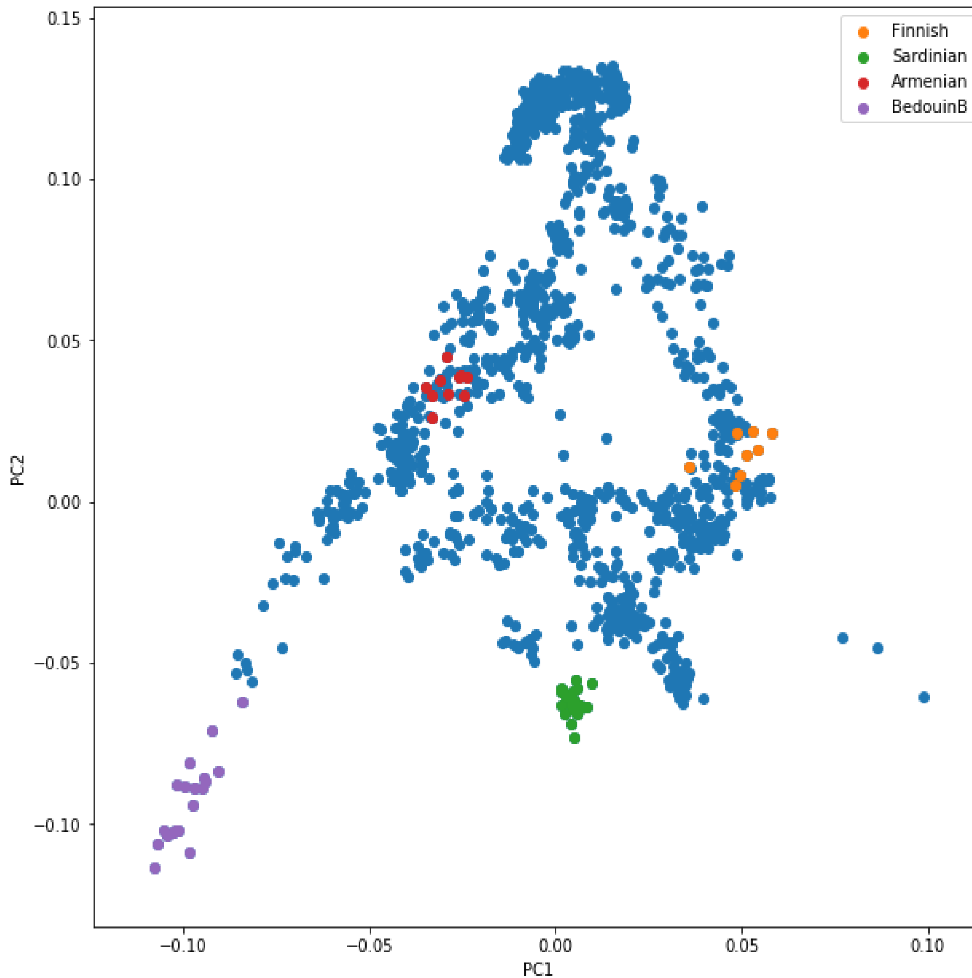
Let's say you called this dataframe `pcaDat`. You can now very easily produce a plot of PC1 vs. PC2 for all samples , by running `plt.scatter(x=pcaDat["PC1"], y=pcaDat["PC2"])`, which in my case yields a boring figure like this:



Now, obviously, we would like to highlight the different populations by color. A quick and dirty solution is to simply plot a different subset of the data on top, like this:

```
plt.scatter(x=pcaDat["PC1"], y=pcaDat["PC2"], label="")
for pop in ["Finnish", "Sardinian", "Armenian", "BedouinB"]:
    d = pcaDat[pcaDat["Population"] == pop]
    plt.scatter(x=d["PC1"], y=d["PC2"], label=pop)
plt.legend()
```

This sequence of commands gives us:



OK, but how do we systematically show all the populations? There are too many of those to separate them all by different colors, or by different symbols, so we need to combine colours and symbols and use all the combinations of them to show all the populations. To do that, we first need to load the population list that we want to focus on for now, which are the same lists as used above for running the PCA. In case of the West Eurasian PCA, you can load the file using `pd.read_csv("/data/pca/WestEurasia.poplist.txt", names=["Population"]).sort_values(by="Population")`. Next, we need to associate a color number and a symbol number with each population. To keep things simple, I would recommend to simply cycle through all combinations automatically. This code snippet looks a bit magic, but it does the job:

```
nPops = len(popListDat)
nCols = 8
nSymbols = int(nPops / nCols)
colorIndices = [int(i / nSymbols) for i in range(nPops)]
symbolIndices = [i % nSymbols for i in range(nPops)]
popListDat = popListDat.assign(colorIndex=colorIndices, symbolIndex=symbolIndices)
```

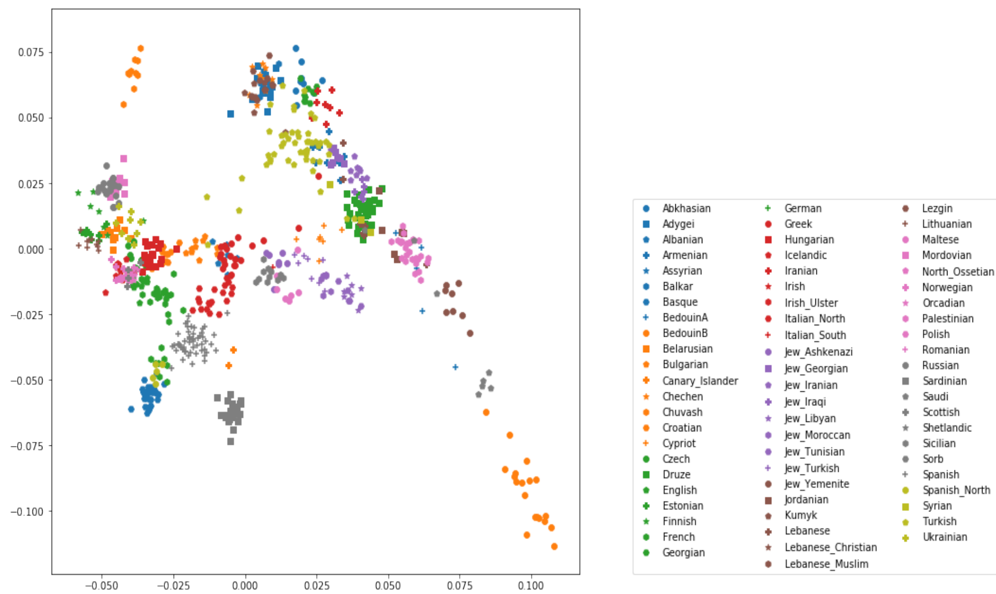
You should check that this worked by viewing the resulting `popListDat` variable (just type its name into a new Jupyter notebook cell). Now we can produce the full PCA plot, which uses a for loop to cycle through all populations in the `popListDat` dataframe, and plots each listed population in turn, with its assigned color and symbol. To prepare, we need a list of colors and symbols. Here, I am using the default color sequence from `matplotlib` and a manual sequence of symbols, which for the sake of simplicity I simply put here for you to copy-paste:

```
symbolVec = ["8", "s", "p", "P", "*", "h", "H", "+", "x", "X", "D", "d", "<", ">", "^
↪", "v"]
colorVec = [u'#1f77b4', u'#ff7f0e', u'#2ca02c', u'#d62728', u'#9467bd',
            u'#8c564b', u'#e377c2', u'#7f7f7f', u'#bcbd22', u'#17becf']
```

With this, the final plot command is:

```
for i, row in popListDat.iterrows():
    d = pcaDat[pcaDat.Population == row["Population"]]
    plt.scatter(x=-d["PC1"], y=d["PC2"], c=colorVec[row["colorIndex"]],
                marker=symbolVec[row["symbolIndex"]], label=row["Population"])
plt.legend(loc=(1.1, 0), ncol=3)
```

which produces a nice plot like this (note that I've flipped the x axis to make the correlation with Geography more apparent):



3.7 Adding ancient individuals

Of course, until now we haven't yet included any of the actual ancient test individuals that we want to analyse, but with plot command above you can very easily add them, by simply adding a few manual plot command before the legend, but outside of the foor loop.

Exercise

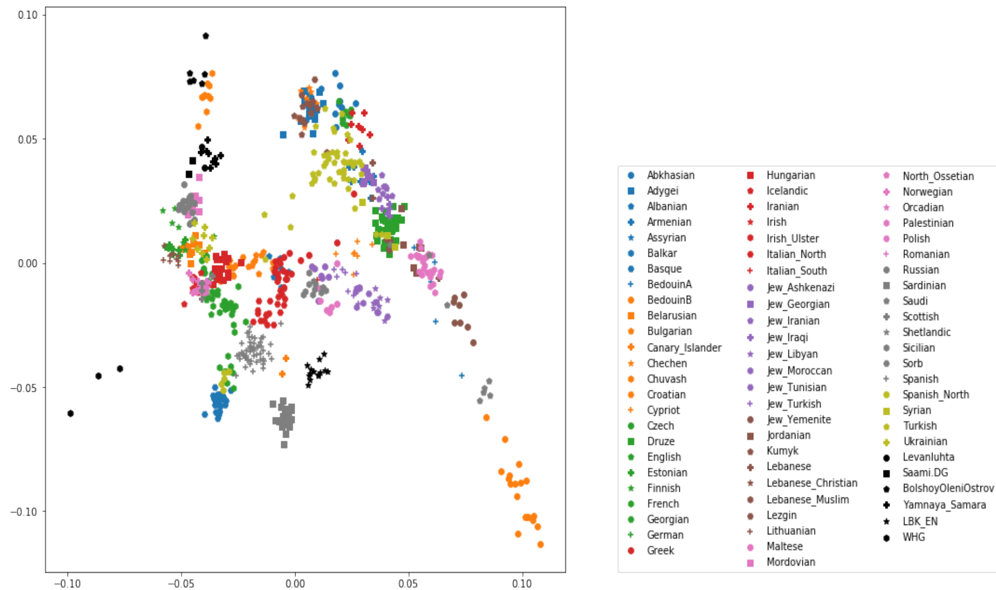
Add two ancient populations to this plot, named "Levanluhta", "JK2065" (the third individual from Levanluhta with different ancestry) and "BolshoyOleniOstrov", using the same technique of selecting populations from the big dataset and plotting them as used in case of the modern populations. Use "black" as colour, and different symbols for each additional population. While you're at it, go ahead and also add the population called "Saami.DG".

Finally, we are going to learn something about deeper European history, by also adding some Neolithic and Mesolithic populations:

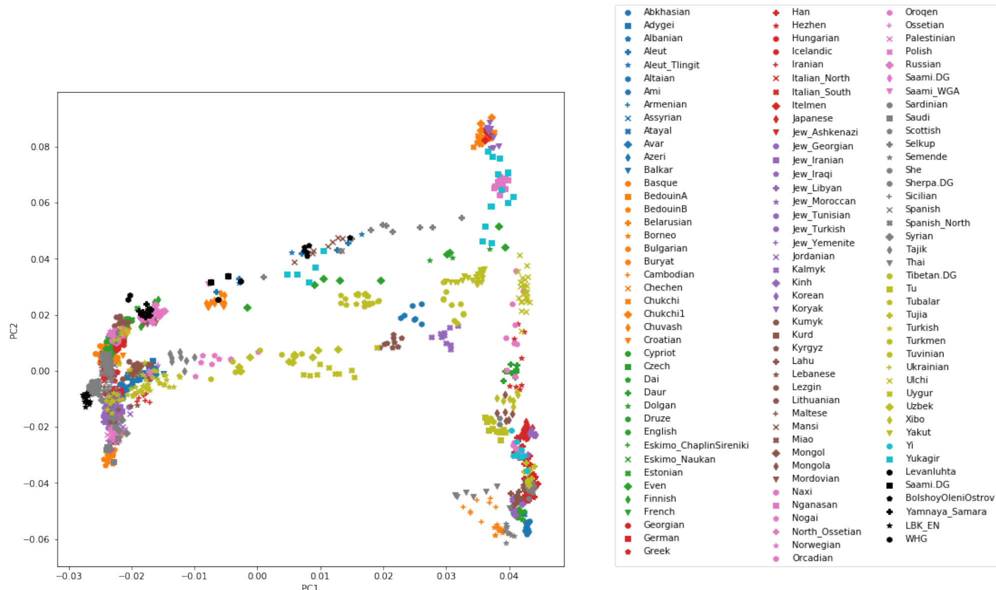
Exercise

Add three more populations to the plot, called "WHG" (short for Western Hunter-Gatherers), "LBK_EN" (short for Linearbandkeramik Early Neolithic, from about 6,000 years ago), and "Yamnaya_Samara", a late Neolithic population from the Russian Steppe, about 4,800 years ago. It can be shown that modern European genetic diversity is formed by a mixture of these three divergence ancient groups (Lazaridis2014, Haak2015).

The final plot should look like this:



You can carry out similar commands to plot the All Eurasia case, which should look like this:



You can fine the solution notebook here.

CHAPTER 4

F Statistics and Treemix

Slides for this lesson are available `here`.

The workbook for this module is available here

And a full solutions notebook for this session available here.

Please also see https://bodkan.net/admixr for documentation on how to run Admixtools with the fantastic `admixr` package in R.

CHAPTER 5

Admixture modelling with *qpWave* and *qpAdm*

In the previous session, we learned how to calculate D/F_4 statistics and how to interpret output. F_4 statistics are extremely useful to test simple phylogenetic hypotheses, such as whether the relationship of four test populations fits into a tree. However, an individual f_4-statistic cannot provide specifics of this "non-treeness": such as i) what is the magnitude and direction of gene flow, ii) how many gene flows are required to model the relationship between the populations used, and iii) whether the populations used are sufficient to model their relationships (or an unsampled lineage is required).

QpWave and its derivative qpAdm are tools for summarizing information from multiple F-statistics to make such sophisticated

1) Detecting the minimum number of independent gene pools to explain a set of target populations (*qpWave*)
2) Testing sufficiency of an admixture model within the resolution of data (*qpAdm*)
3) Estimating admixture proportions (*qpAdm*)

5.1 Overview: summarizing a matrix of F4 statistics

In the *qpWave/qpAdm* scheme, you choose m "right" populations (or outgroups) and n "left" populations (targets for *qpWave*, target and references for *qpAdm*). Taking the first population on each side as the point of comparison, you build a $(m-1) \times (n-1)$ matrix of f_4 statistics in the following way:

$$F_{ij} = F_4(L_1, L_j; R_1, R_j)$$

Given that you provide a sufficient number of distinct outgroup (m > n), this matrix has maximum n-1 independent columns (= rank) and minimum zero nontrivial column (i.e. zero matrix). For example, two unadmixed Native American left populations are a sister group to each other against all non-Native American right populations. Then, any $f_4(\text{American1}, \text{American2}; \text{non-American1}, \text{non-American2})$ will be zero.

Assuming that this matrix has rank r, i.e. there are r independent columns and the rest is a linear combination of the r columns, we can model F matrix into product of two matrices, A and B.

$$F = A \cdot B$$

Where A is an $(m-1) \times r$ matrix, and B is an $r \times (n-1)$ matrix. You can think that columns of A matrix represents r independent F_4 statistic columns. In turn, j-th column of B matrix represents the weight values for combining A columns to reproduce the j-th column of the original F matrix.

The observed f_4 statistic is an estimate of true parameter, F_4 , with a noise/error. Therefore, the actual model fitting becomes to estimate A and B matrices in the following formula:

$$F = A \cdot B + E$$

where E = error matrix:

$$E = \text{F-observed} - A \cdot B$$

Assuming that $(m-1) \times (n-1)$ entries of the E matrix follows a multivariate normal distribution with mean zero (because it is the error matrix), you can write down log-likelihood and compare different models based on log-likelihood.

`qpAdm` is a special case of `qpWave`, in which you assume that the first left population ("target)" is a mixture of the remaining left populations ("references"), and therefore not independent. It becomes a *qpWave* case with rank (n-2) with additional constraint for scaling to make sum of admixture coefficient to 1.

5.2 Data

In this session, we will use a small `EIGENSTRAT` format genotype data, including the following populations:

```
/data/qpAdm/popgen_qpAdm_test_190120.geno
/data/qpAdm/popgen_qpAdm_test_190120.ind
/data/qpAdm/popgen_qpAdm_test_190120.snp
```

5.3 Preparing "left" and "right" populations

Both "left" and "right" population lists are a simple text file, including one population name per line. In the EIGENSTRAT format input genotype data, population name matches entries in the third column of the `.ind` file. "Right" population list includes outgroup populations that are distantly but (potentially) differentially related to "left" populations. "Left" population list includes populations of your interest. For `qpWave`, the order of left populations does not have specific meaning. However, for `qpAdm`, the first left population (one at the top of the file) is the target of admixture modeling, and the remaining ones serve as reference populations for the target.

For this session, let's generate two "left" and one "right" population list files using the following code bits:

```
target="Corded_Ware_Germany"
refs="Yamnaya_Samara LBK_EN"
og1s="Mbuti Natufian Onge Iran_N Villabruna Mixe Ami Nganasan Itelmen"

echo ${refs} | sed s/" "/"\n"/g > left1.pops
echo ${target} ${refs} | sed s/" "/"\n"/g > left2.pops
echo ${og1s} | sed s/" "/"\n"/g > right.pops
```

Exercise

If you run `qpAdm` using left2.pops file, which population is your target for admixture modeling? What admixture modeling are you proposing by using this list? How would you write the left population list file if you want to test three-way admixture model, including "WHG" (Mesolithic Western European Hunter-gatherers) as the third reference?

5.4 Preparing the parameter file

We will use `qpWave` and `qpAdm` programs in the `ADMIXTOOLS` package. Parameter files for both programs have the same structure, similar to parameter files for other programs included in the package.:

```
genotypename: <qpWave_qpAdm_test>.geno
snpname: <qpWave_qpAdm_test>.snp
indivname: <qpWave_qpAdm_test>.ind
popleft: left1.pops
popright: right.pops
details: YES
maxrank: 7
```

You can write two parameter files, using `left1.pops` and `left2.pops` as the left population file, respectively, using a simple loop in bash:

```
pt1=($(pwd)"/")
fn1="/home/choongwon/qpAdm/popgen_qpAdm_test_190120"

for i in $(seq 1 2); do
    echo 'genotypename: '${fn1}'.geno' > test${i}.par
    echo 'snpname: '${fn1}'.snp' >> test${i}.par
    echo 'indivname: '${fn1}'.ind' >> test${i}.par
    echo 'popleft: '${pt1}'left'${i}'.pops' >> test${i}.par
    echo 'popright: '${pt1}'right.pops' >> test${i}.par
    echo -e 'details: YES\nmaxrank: 7' >> test${i}.par
done
```

An important optional parameter is `allsnps:  YES`. In default setting, both programs use only the SNPs that are not missing in any of the left or right populations. Therefore, all f_4 statistics used in the program are calculated across the exactly same set of SNPs. When `allsnps:  YES` is set, each f_4 statistic is calculated using SNPs that are present in the four populations included in the test.

5.5 Running the programs

Running command lines are similar to the other programs (e.g. `qpDstat` or `smartpca`). Let's run the following three tests:

```
qpWave -p test1.par > qpWave.test1.log
qpWave -p test2.par > qpWave.test2.log
qpAdm -p test2.par > qpAdm.test2.log
```

Exercise

Run `qpWave` and `qpAdm` with the prepared parameter file.

Note: Running `qpWave` or `qpAdm` with this dataset takes 1-2 minutes, using a single core (no multithreading is available).

5.6 Reading the *qpWave* log file

Both `qpWave` and `qpAdm` log files begin by iterating the parameter file, showing the current version of the program, listing left and right populations in the given order:

```
### THE INPUT PARAMETERS
##PARAMETER NAME: VALUE
genotypename: /home/choongwon/qpAdm/popgen_qpAdm_test_190120.geno
snpname: /home/choongwon/qpAdm/popgen_qpAdm_test_190120.snp
indivname: /home/choongwon/qpAdm/popgen_qpAdm_test_190120.ind
popleft: /home/choongwon/qpAdm/left1.pops
popright: /home/choongwon/qpAdm/right.pops
details: YES
maxrank: 7
## qpWave version: 410

left pops:
Yamnaya_Samara
LBK_EN

right pops:
Mbuti
Natufian
...
```

The next block of the log file shows quantity of data used in the analysis:

```
  0       Yamnaya_Samara    9
  1               LBK_EN    6
  2                Mbuti   10
  3             Natufian    6
  4                 Onge   11
  5               Iran_N    5
  6           Villabruna    1
  7                 Mixe   10
  8                  Ami   10
  9             Nganasan   33
 10              Itelmen    6
jackknife block size:     0.050
snps: 593124  indivs: 107
number of blocks for block jackknife: 711
dof (jackknife):   612.975
...
```

Exercise

How many SNPs are used in the analysis according to the test1 log file?

The next block contains the actual test results for `qpWave`. Because we have two left populations in test1 setting (Yamnaya_Samara and LBK_EN) and 9 right populations, `qpWave` operates on 8-by-1 f_4 matrix. Therefore, the maximum rank of the matrix is 1 (i.e. two left populations are differentially related to the outgroups). Rank 0 here means that the f_4 matrix is indistinguishable from a zero vector (i.e. two left populations are symetrically related to all outgroups listed).:

```
f4rank: 0 dof:       8 chisq:   609.975 tail:      1.67697608e-126 dofdiff:      0␣
↪chisqdiff:     0.000 taildiff:                   1
f4rank: 1 dof:       0 chisq:     0.000 tail:                    1 dofdiff:      8␣
↪chisqdiff:   609.975 taildiff:      1.67697608e-126
```

This block contains multiple sub-blocks, showing results from rank zero to min(maximum rank, maxrank value in the parfile). You can see the structure better with test2 log file, where you have three left populations and thus maximum rank value two:

```
f4rank: 0 dof:      16 chisq:   690.774 tail:      1.18836874e-136 dofdiff:      0␣
↪chisqdiff:     0.000 taildiff:                   1
f4rank: 1 dof:       7 chisq:    31.660 tail:       4.69503474e-05 dofdiff:      9␣
↪chisqdiff:   659.114 taildiff:      4.23672911e-136
f4rank: 2 dof:       0 chisq:     0.000 tail:                    1 dofdiff:      7␣
↪chisqdiff:    31.660 taildiff:       4.69503474e-05
```

Each line contains results for a certain rank value ("f4rank: r-val"). For each rank, it returns results for two different tests.

1) A comparison of the rank "r-val" model and the "full" model (i.e. rank = maxrank): dof/chisq/tail
2) A comparison of the rank "r-val" model and the rank "r-val - 1" model: dofdiff/chisqdiff/taildiff

The "full" model provides a perfect fit to data because one free parameter is assigned to each entry of the f_4 matrix. Small *p*-value ($<< 0.05$) for the "tail:" cell means that the reduced rank model under consideration fits data substantially worse than the full model. That means, you need higher rank to properly explain data. Again, that means that you need more streams of independent ancestry to explain the left populations!

To obtain the minumum number of distinct ancestries required, it is often convenient to go over the "taildiff: " cell from higher to lower ranks. If the presented *p*-value is significant (i.e. < 0.05), it means that the current rank fits data significantly better than the simpler model with rank-1.

Exercise

How many streams of distinct ancestries are required to explain left1 and left2 population sets? What would that mean for the left2 populations, given that our goal is to check if Corded Ware individuals from Germany can be modeled as a mixture of Yamnaya and LBK?

Exercise

Let's make another list of left populations, this time including WHG as an additional population. Then, let's run qpWave on this list of four left populations. What is the rank of the f_4 matrix?

Last, each non-zero rank model is followed by the estimates of A and B matrices. In case of left2 populations and rank 2,:

```
B:
          scale     1.000     1.000
       Natufian    -1.270     0.487
           Onge     0.547    -0.560
         Iran_N     0.343    -0.517
     Villabruna     0.652     2.647
           Mixe     1.570    -0.158
            Ami     0.613     0.046
       Nganasan     1.032     0.077
```

(continues on next page)

(continued from previous page)

```
        Itelmen     1.279    -0.379
A:
          scale   906.839  3447.047
 Yamnaya_Samara     0.296    -1.383
         LBK_EN    -1.383    -0.296
```

The product of A and B matrices is the expected f_4 matrix. Based on this number, you can have a good idea on which right populations are useful to distinguish between left populations.

5.7 Reading the *qpAdm* log file

`qpWave` and `qpAdm` are essentially the same program. You can see this in the first part of the `qpAdm` log file, which simply repeats the corresponding `qpWave` results. The first block of the log file shows *qpWave* test results for the full rank and (full rank - 1), including degree of freedom, chi-square value, *p*-value, and A/B matrices.

The next block shows the main results of `qpAdm`: admixture coefficient estimates and standard errors:

```
best coefficients:     0.765     0.235
Jackknife mean:      0.764524556     0.235475444
      std. errors:     0.034     0.034

error covariance (* 1000000)
      1133      -1133
     -1133       1133
```

In the case of two-way admixture model, admixture coefficients are perfectly correlated. That is, if the coefficient for the reference 1 is alpha, that for the reference two is (1 - alpha).

The next block presents model fit measures (i.e. *p*-values) for the proposed admixture model and all "submodels" of it:

```
    fixed pat  wt  dof     chisq       tail prob
           00  0     7   31.536     4.95055e-05     0.765     0.235
           01  1     8   75.490     3.93452e-13     1.000    -0.000
           10  1     8  389.320               0     0.000     1.000
best pat:          00      4.95055e-05              -  -
best pat:          01      3.93452e-13  chi(nested):    43.954 p-value for nested␣
↪model:     3.36113e-11
```

"tail prob" cell shows the comparison of the proposed admixture model (i.e. maxrank - 1) and the full model (i.e. maxrank). If the value is small, it suggests that the target population significantly deviate from the proposed admixture model (= a linear combination of reference populations).

"fixed pat" cell shows which (sub)model is being tested. "0" means that the corresponding reference population is included in the model, while "1" means that the reference is fixed to have zero admixture coefficient.

Exercise

Judging from the tail prob of two submodels, "01" and "10", which reference seems genetically closer to Corded Ware? Does it match with the admixture coefficients of the two-way admixture model?

"p-value for the nested model" is a useful metric for exploring whether there is a small set of references that can adequately explain the target population. Here it is provided only for the "best" submodel, but you can calculate it easily using dof and chisq values in the table.

Exercise

In R shell, let's run the following command:

1 - pchisq(75.490-31.536, df=8-7)

What number do you get? Can you calculate nested p-value for the worse submodel "10"?

Although the above two blocks provide all of the essential results for `qpAdm`, the following two blocks actually harbor quite useful information. The next "dscore" block shows which outgroup is useful to distinguish between reference populations and which outgroup makes the model deviate from data when the model does not fit well. For each outgroup except for the "base" outgroup (i.e. one at the top of the right population list), you get the following sub-block:

```
## dscore:: f_4(Base, Fit, Rbase, right2)
details:  Yamnaya_Samara        Natufian    -0.000610   -3.188475
details:          LBK_EN        Natufian     0.001895    8.003780
dscore:        Natufian f4:    -0.000021 Z:    -0.119830
```

The first two lines ("details") show the actual entries of f_4 matrix and Z-score, calculated by dividing the f_4 by its block jackknifing standard error value. That is, the above two lines mean:

```
f4(Corded_Ware_Germany, Yamnaya_Samara; Mbuti, Natufian)
f4(Corded_Ware_Germany, LBK_EN        ; Mbuti, Natufian)
```

The last "dscore" line makes a linear combination of the above lines, using the admixture coefficient estimates:

```
f4(CW, Model; Mbu, Nat) = 0.765 * f4(CW, Yam; Mbu, Nat) + 0.235 * f4(CW, LBK; Mbu,
↪Nat)
-0.0000213 = -0.000610 * 0.765 + 0.001895 * 0.235
```

If you are testing a three-way model, you get:

```
## dscore:: f_4(Base, Fit, Rbase, right2)
details:  Yamnaya_Samara        Natufian    -0.000630   -3.182212
details:          LBK_EN        Natufian     0.001961    8.018318
details:             WHG        Natufian     0.000319    1.297454
dscore:        Natufian f4:    -0.000045 Z:    -0.255258
```

Exercise

Let's recover the "dscore" value by taking a linear combination of the above three numbers.

The last block is a linear combination of "dscore" rows to obtain the contrast between two non-base outgroups (in z-score scale). For example, if you want to obtain f4(Corded Ware, Model; Natufian, Onge):

```
f4(CW, Model; Nat, Ong) = f4(CW, Model; Mbu, Ong) - f4(CW, Model; Mbu, Nat)
```

You now know how to run *qpWave*/*qpAdm* and how to interpret results!

CHAPTER 6

Estimating Admixture times with *Alder*

6.1 Overview

Admixture between two distinct gene pools generate an linkage disequilibrium (LD) that extends even megabases across genomes. This "admixture LD" extends far longer than the typical LD in non-recently-admixed human populations, which often decays within few tends of kilobases. Once formed, admixture LD decays, like the typical LD, over time due to recombination. Therefore, a relationship between the strength of LD the distance between markers provides a powerful window to look into the temporal aspect of admixture. Such a feature is by definition not visible by F-statistics, which does not consider LD.

The demographic admixture process is far more complicated than any simple model can fully capture, but even a surprisingly simple model can provides a useful insight into the characteristics of the admixture process. Here we explore the `alder` program that is based on the simplest possible model: a single pulse-like admixture.

6.2 Data

In this session, we will use a small `EIGENSTRAT` format genotype data, including the following populations:

```
/data/alder/popgen_alder_test_190120.geno
/data/alder/popgen_alder_test_190120.ind
/data/alder/popgen_alder_test_190120.snp
```

Exercise

How many individuals are in the data set? How many groups? Could you count the number of individuals in each population?

6.3 Preparing the parameter file

We will use `alder` v1.03 in this session. Parameter file for alder takes a format similar to that for the programs in the ADMIXTOOLS package.:

```
genotypename: <your_genotype_data>.geno
snpname: <your_genotype_data>.snp
indivname: <your_genotype_data>.ind
admixpop: TARGET
refpops: REF1;REF2
raw_outname: <output>.txt
```

The "admixpop" argument is for the name of your target population. The name should match with the group ID in the 3rd column of `.ind` file.

The "refpops" argument is for a list of reference populations for testing admixture in the target. The standard setting is to provide two populatuions (semicolon separated). You can provide more than two (also semicolon separated). In that case, all pairs of references will be run one by one. You can also provide only a single reference population: alder will run only the so-called "1-reference" model. This will still provide you with an estimate of admixture date in case that you do not have a good proxy for the contributing ancestry. However, "1-ref" mode is prone to false positives and does not provide a proper test of admixture signal.

Two important optional parameters are `mindis:  NUMBER` and `jackknife:  YES`. The "mindis" option sets up the minimum distance (in Morgan) between two SNP bins that you want to include in exponential curve fitting. If not set up, `alder` calculate a correlation of LD decay between the target and each reference population and take the minimum distance that shows no correlation in both references. If it continues beyond 2 cM (due to either a strong bottleneck or admixture in a reference), the program aborts. `jackknife:  YES` is to output 22 additional weighted LD files, each representing a leave-one-chromosome-out jackknifing results. It is useful if you want to do exponential fitting on your side.

Let's write two parameter files, one without mindis argument, and the other including `mindis:  0.005`. For both tests, we use `Uygur` as the target and `Daur` and `Georgian` as references.:

```
pt1=($(pwd)"/")
fn1="/data/alder/popgen_alder_test_190120"
target="Uygur"
refs="Daur;Georgian"

for i in $(seq 1 2); do
    echo 'genotypename: '${fn1}'.geno' > test${i}.par
    echo 'snpname: '${fn1}'.snp' >> test${i}.par
    echo 'indivname: '${fn1}'.ind' >> test${i}.par
    echo 'admixpop: '${target} >> test${i}.par
    echo 'refpops: '${refs} >> test${i}.par
    echo 'raw_outname: '${pt1}'alder.test'${i}'.raw.txt' >> test${i}.par
    if [ "$i" -eq 2 ]; then echo 'mindis: 0.005' >> test${i}.par; fi
done
```

6.4 Running the programs

Running command line is similar to the other programs (e.g. `qpDstat` or `smartpca`). Let's run the following two tests:

```
alder -p test1.par > test1.log
alder -p test2.par > test2.log
```

If you want to see the log file being written in real time, you can try pipe + tee command instead of re-direction:

```
alder -p test1.par | tee test1.log
alder -p test2.par | tee test2.log
```

Note: Running `alder` with this small dataset takes a few minutes, using a single core. Multithreading is also available.

6.5 Reading the main output file (weighted LD table)

The main output file, the name of which is designated by the "raw_outname:" argument, looks like below:

```
#   d (cM)      weighted LD         bin count
#    0.050       0.00153525           7478597
#    0.100       0.00073147           6079776
#    0.150       0.00049288           5940521
#    0.200       0.00038920           5914194
...
#    0.600       0.00023054           5700318
#    0.650       0.00022210           5702649
     0.700       0.00020821           5687007
     0.750       0.00020796           5670508
...
    49.900      -0.00000575           3938767
    49.950      -0.00000674           3910622
       inf       0.00000416       -1089862760
# last row: affine term computed from pairs of SNPs on different chroms
```

Exercise

What do you think would "#" at the beginning of some rows mean?

6.6 Reading the log file

The first block of the log file repeats the parameter settings and input files. There are a few default settings that are good to know:

```
Data filtering:
            mincount: 4

Curve fitting:
             binsize: 0.000500
              mindis: auto
              maxdis: 0.500000

Computational options:
         num_threads: 1
```

Alder calculates LD between a pair of SNPs across individuals. The "mincount:" argument dictates the minimum number of individuals that you have genotype data for both SNPs. For computational efficiency, alder first split

genome into small non-overlapping bins and use the distance between two bins for all pairs of SNPs between the two. The "binsize" argument dictates the size of these bins. "First-bin-then-measure-distance" approach of alder produces bigger variance of actual distance between pairs of SNPs in the same bin than "first-measure-distance-then-bin" approach of rolloff, but it is computationally much faster. You can also multithreads alder by setting the "num_threads:" argument.

If you did not set up the "mindis:" argument, the next block describes the minimum distance between bins that alder found:

```
Checking LD correlation of test pop Uygur with ref pop Georgian
  binsize: 0.1 cM
  (distances are rounded down to bins; bin starting at 0 is skipped)

d (cM)    LD corr (scaled)   bin count
 0.100     0.119 +/- 0.007      186942
 0.200     0.034 +/- 0.010      364453
 0.300     0.012 +/- 0.004     5760262
 0.400     0.018 +/- 0.006     2861516
 0.500     0.005 +/- 0.004    11396910   losing significance (1)
 0.600     0.006 +/- 0.005    11395248   losing significance (2)
lost significance; computing bias-corrected LD corr polyache
 0.600     0.216 +/- 0.186   <-- approx bias-corrected LD corr

Decay curves will be fit starting at the folliowing min distances (cM):
  (to override, specify the 'mindis' parameter)

                Daur   0.700
            Georgian   0.600
```

The program will use 0.7 cM for the minimum distance between bins (inclusive) to fit exponential decay.

The next block describes the main results of alder: exponential fit of weighted LD decay using two references:

```
---- fit on data from 0.50 to 50.00 cM (using inter-chrom affine term) ----
d>0.50        decay:       19.94 +/- 1.96           z = 10.16 *
d>0.50      amp_tot:  0.00021347 +/- 0.00001428
d>0.50      amp_exp:  0.00021139 +/- 0.00001462   z = 14.46 *
d>0.50      amp_aff:  0.00000416 +/- 0.00000174

---- fit on data from 0.60 to 50.00 cM (using inter-chrom affine term) ----
d>0.60        decay:       19.32 +/- 1.98           z = 9.77 *
d>0.60      amp_tot:  0.00020732 +/- 0.00001400
d>0.60      amp_exp:  0.00020524 +/- 0.00001434   z = 14.31 *
d>0.60      amp_aff:  0.00000416 +/- 0.00000174

---- fit on data from 0.70 to 50.00 cM (using inter-chrom affine term) ----
d>0.70        decay:       18.82 +/- 1.97           z = 9.58 *
d>0.70      amp_tot:  0.00020218 +/- 0.00001398
d>0.70      amp_exp:  0.00020010 +/- 0.00001431   z = 13.98 *
d>0.70      amp_aff:  0.00000416 +/- 0.00000174

---- fit on data from 0.80 to 50.00 cM (using inter-chrom affine term) ----
d>0.80        decay:       18.45 +/- 1.98           z = 9.30 *
d>0.80      amp_tot:  0.00019840 +/- 0.00001406
d>0.80      amp_exp:  0.00019632 +/- 0.00001438   z = 13.66 *
d>0.80      amp_aff:  0.00000416 +/- 0.00000174

---- fit on data from 0.90 to 50.00 cM (using inter-chrom affine term) ----
```

(continues on next page)

(continued from previous page)

```
d>0.90       decay:       17.97 +/- 2.11          z = 8.52 *
d>0.90     amp_tot:  0.00019337 +/- 0.00001429
d>0.90     amp_exp:  0.00019129 +/- 0.00001465    z = 13.06 *
d>0.90     amp_aff:  0.00000416 +/- 0.00000174
```

This block shows estimates using the above-found minimum distance (0.7 cM), as well as values around it (0.2 cM range). The most important number is "decay:", which shows the estimate for admixture date and its block jackknifing standard error. It corresponds to the "n" parameter in the following exponential curve formula:

```
.. math::
```

E[Y] = M times e^-nd + K

Exercise

In which unit is the decay parameter estimate? Let's convert it to the unit of years.

Another important estimate is "amp_exp:", which corresponds to the "M" parameter in the above formula. Naively speaking, it becomes bigger when your choice of references matches the true sources better.

Exercise

What will the "d" parameter represent? In which unit will it be then?

Exercise

Check the log file of "test2" run, for which we provided "mindis: 0.005" argument. Do you see the difference in the results on this block? Combining the results from the two runs, do you see any trend in the decay parameter estimates over the minimum distance values?

Then, alder repeats fitting with the "1-ref" models. In these runs, alder uses one of the two references and the target itself to set up weight.

The last block summarizes test results. The last lines will tell repeat the most important estimates and tell you whether alder considers the test of admixture as either successful or failed.:

```
Test SUCCEEDS (z=9.58, p=1e-21) for Uygur with {Daur, Georgian} weights

DATA:   success (warning: decay rates inconsistent)     1e-21   Uygur   Daur   ␣
↪Georgian        9.58     6.34     8.57     28%     18.82 +/- 1.97  0.00020010 +/- 0.
↪00001431    16.22 +/- 2.56  0.00004300 +/- 0.00000456       21.55 +/- 2.52  0.
↪00005970 +/- 0.00000650

DATA:   test status     p-value test pop        ref A    ref B   2-ref z-score   1-ref␣
↪z-score A 1-ref z-score B max decay diff %        2-ref decay      2-ref amp_exp     ␣
↪    1-ref decay A    1-ref amp_exp A 1-ref decay B   1-ref amp_exp B
```

Exercise

Let's set up new parameter files that use Daur and French as two references (instead of Georgian). Then run alder and compare the results with the current runs.

6.7 Plotting weighted LD decay curve in R

For an easy plotting in R, let's re-format the main output file (the LD table) using the follwoing bash commands:

```
for i in $(seq 1 2); do
    echo -e 'Dist\tweightedLD\tnpairs\tuse' > alder.test${i}.txt
    head -n -2 alder.test${i}.raw.txt | tail -n +2 | awk '{OFS="\t"} {if ($1 == "#")
↪print $2,$3,$4,"N"; else print $1,$2,$3,"Y"}' >> alder.test${i}.txt
done
```

This is to make it into a proper table format and remove irregular lines (the last two).

In R, you can import this file, create a scatter plot, and add the fitted curve based on the estimates in the log file. For example, let's use "test1" results:

```
fn1 = "alder.test1.txt"

n = 18.82       ## decay parameter
M = 0.00020524  ## amp_exp
K = 0.00000416  ## amp_aff

d1 = read.table(fn1, header=T)
xv = as.vector(d1$Dist)         ## distance between SNP bins (cM)
yv = as.vector(d1$weightedLD)   ## Weighted LD
fv = as.vector(d1$use) == "Y"   ## a boolean vector marking if each bin is used in
↪fitting
prdv = M * exp(-1 * n * xv / 100) + K

plot(xv[fv], yv[fv], xlab="Genetic distance (cM)", ylab = "weighted LD", pch=4, col=
↪"blue")
points(xv[fv], prdv[fv], type="l", lwd=1.5, col="red")
```

Exercise

How does the exponential decay curve look? Does it seem to fit data well?

Slides for this lesson are available `here`.

CHAPTER 7

Estimating Admixture Graphs with `qpGraph`

Based on previous analyses we can try to reconstruct a model that includes the described observations in an admixture graph, which combines in an unique model the relationships between multiple groups that can be single individuals or larger populations. For that we are going to use `qpGraph` (also known as "ADMIXTUREGRAPH") part of the package ADMIXTOOLS (Patterson et al. 2012. _qpGraph_ relies on an user-defined topology of the admixture graph and calculates the best-fitting admixture proportions and branch lengths based on the observed f-statistics.

7.1 Overview method

The general approach of an admixture graph is to reconstruct the genetic relationships between different groups through a phylogenetic tree allowing for the addition of admixture events. The method operates on a defined graph's topology and estimates *f2*, *f3*, and *f4*-statistic values for all pairs, triples, and quadruples of groups, compared to the expected allele frequency correlation of the tested groups. For a given topology, `qpGraph` provides branch lengths (in units of genetic drift) and mixture proportions. Groups that share a more recent common ancestor will covary more than others in their allele frequencies due to common genetic drift. We should keep in mind that the model is based on an unrooted tree and that while we show graphs with a selected outgroup as the root, the results should not depend on the root position.

Note: The reconstructed graph is never meant to reflect a comprehensive population history of the region under study but the best fitted model to the limit of the available groups and method's resolution.

As reported in the Method section of Lipson and Reich 2017:

> "Our usual strategy for building a model in ADMIXTUREGRAPH is to start with a small, well-understood subgraph and then add populations (either unadmixed or admixed) one at a time in their best-fitting positions. This involves trying different branch points for the new population and comparing the results. If a population is unadmixed, then if it is placed in the wrong position, the fit of the model will be poorer, and the inferred split point will move as far as it can in the correct direction, constrained only by the specified topology. Thus, searching over possible branching orders allows us to find a (locally) optimal topology. If no placement provides a good fit (in the sense that the residual errors are large), then we infer the presence of an admixture event, in which case we test for the best-fitting split points of the

two ancestry components. After a new population is added, the topology relating the existing populations can change, so we examine the full model fit and any inferred zero-length internal branches for possible local optimizations."

7.2 Data

For this session, we are going to use modern and ancient DNA data analysed in the recently published study about the settlement of the Americas Posth et al. 2018.

The genotype data is in "EIGENSTRAT" format and exclude from the "1240K panel" a subset of SNPs that are transitions in CpG sites. Those Cytosines are prone to be methylated and if deamination (the typical chemical modification of ancient DNA) occurs at the same position Cytosine is directly converted into Thymine without becoming Uracil. Thus, the resulting CtoT modification cannot be removed with an enzymatic reaction like performing uracil-DNA glycosylase (UDG) treatment. This results in additional noise in ancient DNA data that can be reduced by excluding those SNPs. This is especially relevant when analysing ancient samples that have been processed using different laboratory protocols.

The path to the data we are going to work on is the following:

```
/data/qpGraph/Posth2018_Americas.packedancestrymapgeno
/data/qpGraph/Posth2018_Americas.ind
/data/qpGraph/Posth2018_Americas.snp
```

Exercise

Check how many SNPs are in this dataset.

7.3 Preparing the parameter file

In order to run `qpGraph` we need to prepare a parameter file that we can call "parQpgraph" and where we change "filename" with the names above:

```
DIR: /data/qpGraph
genotypename: DIR/<filename>.packedancestrymapgeno
snpname: DIR/<filename>.snp
indivname: DIR/<filename>.ind
outpop:  NULL
useallsnps: YES
blgsize: 0.05
forcezmode: YES
lsqmode: YES
diag:  .0001
bigiter: 6
hires: YES
lambdascale: 1
```

Let's go through the most relevant parameter options. `outpop:  NULL` does not use an outgroup population to normalize f-stats by heterozygosity e.g. selecting a group in the graph in which SNPs must be polymorphic. `useallsnps:  YES` each comparison uses all SNPs overlapping in that specific test, otherwise the program looks only at the overlapping SNPs between all groups. `blgsize:  0.05` is the block size in Morgans for Jackknife. `diag:  .0001` uses the entire matrix form of the objective function to avoid the basis dependence of the least-squares version of the computation. `lambdascale:  1` in order to preserve the standard scaling of the f-statistics

without an extra denominator. `lsqmode: YES` otherwise unstable for large graphs. `hires: YES` controls output when more decimals are desired.

7.4 Preparing the graph topology

We start by constructing a scaffold graph based on previously published studies (Lipson and Reich 2017, Moreno-Mayar et al. 2018 and Scheib et al. 2018). The following can be saved in a file called "Figure3a":

```
root    R
label    Mbuti.DG    Mbuti.DG
label    Han.DG    Han.DG
label    Onge.DG    Onge.DG
label    MA1    MA1
label    USR1   USR1
label    USA_SanNicolas_4900BP    USA_SanNicolas_4900BP
label    Canada_Lucier_4800BP_500BP    Canada_Lucier_4800BP_500BP

edge    a R Mbuti.DG
edge    b R nonAfrica
edge    c1 nonAfrica EastEurasia
edge    c2 EastEurasia EastAsia
edge    c3 EastEurasia Onge.DG
edge    c4 EastAsia EastAsia2
edge    c5 EastAsia EastAsia3
edge    c6 EastAsia2 Han.DG
edge    c7 EastAsia2 EastAsia4
edge    c8 nonAfrica WestEurasia
edge    c9 WestEurasia E_HG2
edge    c10 WestEurasia E_HG3
admix   ANE E_HG2 EastAsia3
edge    c11 ANE MA1
admix   FA EastAsia4 E_HG3
edge    c12 FA Beringia
edge    c13 Beringia USR1
edge    c14 Beringia NA
edge    c15 NA Canada_Lucier_4800BP_500BP
edge    c16 NA NA2
edge    c17 NA2 USA_SanNicolas_4900BP
```

7.5 Running the program

To run `qpGraph` we need to specify the parameter file "parQpgraph" and three output files that are `.ggg`, `.dot` and `.out`:

```
qpGraph -p parQpgraph -g Figure3a -o Figure3a.ggg -d Figure3a.dot > Figure3a.out
```

Note: Running `qpGraph` with this dataset takes 1-2 minutes.

7.6 Reading the output files

First let's inspect the `.out` file. If we open that on the terminal with `less Figure3a.out` you can directly go to the bottom of the file with `shift-g` sequence. Among other values we should see reported the total number of individuals used for all groups `indivs` and the total number of `snps` available from at least one individual.

Note: as mentioned above, not all those SNPs are used for each f-statistic

```
outpop: NULL
population:   0             Mbuti.DG    5
population:   1               Han.DG    4
population:   2             Onge.DG    2
population:   3                 MA1    1
population:   4                USR1    1
population:   5 USA_SanNicolas_4900BP   17
population:   6 Canada_Lucier_4800BP_500BP    6
before setwt numsnps: 882908  outpop: NULL
setwt numsnps: 681581
number of blocks for moving block jackknife: 713
snps: 681581  indivs: 36
lambdascale:     1.000
```

At the very bottom of the `.out` file are reported the outlier f4-statistics, which show the lowest or highest Z-scores. Those are calculated based on the difference between the fitted and the observed f4 values. The only worst f4-statistic identified in the model we just run is some un-modelled affinity between "USA_SanNicolas_4900BP" and "USR1" that is anyway below 3 standard deviations:

```
outliers:
                                                    Fit          Obs          Diff  ␣
↪Std. error         Z


worst f-stat:       Mbu        USR        USA        Can       0.000000     -0.001849 ␣
↪  -0.001849      0.000681     -2.717
```

We can now visualize the graph in the `.dot` file using the program *graphviz* with the following command and then look at the resulting `.png` from the jupyter file browser. Dashed arrows represent admixture edges while solid arrows drift edges reported in units of FST $\times 1,000$. On the very top, the worst f4-statistic is again reported:

```
dot -Tpng Figure3a.dot -o Figure3a.png
```

Finally we can have a look at the `.ggg` file, which provides detailed proportions for admixture edges and drift lengths for each branch.

Note: generally it is important to not have zero-length edges because it might signify that the modelled edge does not exist. Also terminal edges for ancient groups, especially if composed by a single individual, are artificially long and should not be considered.

7.7 Adding new groups to the scaffold graph

Once confirmed that the scaffold graph has a good fit, we carry on adding the new Central and South American groups released in Posth et. al. 2018. To the file `Figure3a` add the following *labels* `Belize_MayahakCabPek_9300BP, PERu_Cuncaicha_9000BP, Peru_Lauricocha_8600BP` as well as the following additional *edges*, save it with a new name like `Figure3b` and run `qpGraph` as shown before:

```
edge    c18 NA2 CA
edge    c19 CA Belize_MayahakCabPek_9300BP
edge    c20 CA SA
edge    c21 SA SA2
edge    c22 SA2 PERu_Cuncaicha_9000BP
edge    c23 SA2 Peru_Lauricocha_8600BP
```

The worst f4-statistic is -2.809 and despite in the `.dot` file once converted into `.png` there are some zero-length branches a more careful examination of the `.ggg` file indicates that those edges are in fact different from zero.

7.8 Continuing to fit new groups

Now we can create a file called `Figure3c` where we add the last three group *labels* `Brazil_LapaDoSanto_9600BP, Argentina_ArroyoSeco2_7700BP, Chile_LosRieles_5100BP`. From node "SA" add an *edge* to form a new node called "SA3" that splits into `Brazil_LapaDoSanto_9600BP` and a new node `SA4`. Finally "SA4" splits itself into the two Southern Cone populations that are `Argentina_ArroyoSeco2_7700BP` and `Chile_LosRieles_5100BP`. After running it we can visualise the resulting `.dot` file as a `.png`. That is the final graph reported in Figure 3 of Posth et al. 2018!

7.9 Test the robustness of the graph topology

Starting from the final graph `Figure3c` we can try, for example, to invert `Belize_MayahakCabPek_9300BP` with `USA_SanNicolas_4900BP` in a file called `Figure3c.v2` to test for branching patterns between North and Central American groups. For more advanced modelling we can instead invert the entire "SA3" node with `Belize_MayahakCabPek_9300BP` and call the file `Figure3c.v3` to test for Central-South America branching patterns.

Exercise::

What do you observe when inspecting the respective `.out` files? Which of the models fit and which not? How do you interpret that?

7.10 Adding admixture edges

We finally want to add in our working graph the oldest genome published so far from South America called `CHIle_LosRieles_10900BP`. We initially try to position it as departing from each node without invoking admixture. One example is the following `FigureS5a` file that we can copy and run as seen before:

```
root     R
label     Mbuti.DG     Mbuti.DG
label     Han.DG     Han.DG
label     Onge.DG     Onge.DG
label     MA1     MA1
label     USR1     USR1
label     USA_SanNicolas_4900BP     USA_SanNicolas_4900BP
label     Canada_Lucier_4800BP_500BP     Canada_Lucier_4800BP_500BP
label     Belize_MayahakCabPek_9300BP     Belize_MayahakCabPek_9300BP
label     PERu_Cuncaicha_9000BP     PERu_Cuncaicha_9000BP
label     Peru_Lauricocha_8600BP     Peru_Lauricocha_8600BP
label     Brazil_LapaDoSanto_9600BP     Brazil_LapaDoSanto_9600BP
label     Argentina_ArroyoSeco2_7700BP     Argentina_ArroyoSeco2_7700BP
label     Chile_LosRieles_5100BP     Chile_LosRieles_5100BP
label     CHIle_LosRieles_10900BP     CHIle_LosRieles_10900BP

edge     a R Mbuti.DG
edge     b R nonAfrica
edge     c1 nonAfrica EastEurasia
edge     c2 EastEurasia EastAsia
edge     c3 EastEurasia Onge.DG
edge     c4 EastAsia EastAsia2
edge     c5 EastAsia EastAsia3
edge     c6 EastAsia2 Han.DG
edge     c7 EastAsia2 EastAsia4
edge     c8 nonAfrica WestEurasia
edge     c9 WestEurasia E_HG2
edge     c10 WestEurasia E_HG3
admix    ANE E_HG2 EastAsia3
edge     c11 ANE MA1
admix    FA EastAsia4 E_HG3
edge     c12 FA Beringia
edge     c13 Beringia USR1
edge     c14 Beringia NA
edge     c15 NA Canada_Lucier_4800BP_500BP
edge     c16 NA NA2
edge     c17 NA2 USA_SanNicolas_4900BP
edge     c18 NA2 CA
edge     c19 CA Belize_MayahakCabPek_9300BP
edge     c20 CA SA
edge     c21 SA SA2
edge     c22 SA SA3
edge     c23 SA2 SA6
edge     c24 SA2 CHIle_LosRieles_10900BP
edge     c25 SA6 PERu_Cuncaicha_9000BP
edge     c26 SA6 Peru_Lauricocha_8600BP
edge     c27 SA3 Brazil_LapaDoSanto_9600BP
edge     c28 SA3 SA4
edge     c29 SA4 Argentina_ArroyoSeco2_7700BP
edge     c30 SA4 Chile_LosRieles_5100BP
```

At the bottom of the newly produced `.out` file there are several f4-statistics that have Z-scores below -3 or above 3. The worst one is the following statistics that indicates some un-modelled affinity between the two Chilean samples `CHIle_LosRieles_10900BP` and `Chile_LosRieles_5100BP`:

```
worst f-stat:        Han        Chi        Bra        CHI        0.000000        0.003372 ␣
↪    0.003372      0.000862      3.912
```

We can then model a contribution from the oldest Chilean individual into the younger one. Change the last part of `FigureS5a` with the following **edges** and one **admixture event**, save it as `FigureS5a.v2` and run it again:

```
edge    c24 SA2 SA7
edge    c25 SA7 CHIle_LosRieles_10900BP
edge    c26 SA6 PERu_Cuncaicha_9000BP
edge    c27 SA6 Peru_Lauricocha_8600BP
edge    c28 SA3 Brazil_LapaDoSanto_9600BP
edge    c29 SA3 SA4
edge    c30 SA4 Argentina_ArroyoSeco2_7700BP
edge    c31 SA4 SA5
admix   SA8 SA7 SA5
edge    c32 SA8 Chile_LosRieles_5100BP
```

In the `.out` file we see that most of the outlier f4-statistics are gone while the worst statistic is still present but reduced (Zscore=3.2). This suggests the presence of un-modelled affinity between the two oldest South American groups `Brazil_LapaDoSanto_9600BP` and `CHIle_LosRieles_10900BP`, that might represent a shared `Anzick-1`-related ancestry that we investigate in detail in Figure 4 and Figure 5 of Posth et al. 2018. The resulting admixture graph suggests that a component from the oldest Chilean individual contributed at least marginally to the younger individual despite being more than 5,000 years apart!

CHAPTER 8

MSMC

8.1 Data

All input data and intermediate files for this tutorial are at `/data/msmc/`.

For this lesson, we will use two trios from the 69 Genomes dataset by Complete Genomics. You will find the so called "MasterVarBeta" files for six individuals for chromosome 1 in the `cg_data` subdirectory in the tutorial files. Some information on the six samples: The first three form a father-mother-child trio from the West-African Yoruba, a people living in Nigeria. Here, NA19240 is the offspring, and NA19238 and NA19239 are the two parents. The second three samples form a father-mother-child trio from Utah (USA), with European ancestry. Here, NA12878 is the offspring, and NA12891 and NA12892 are the parents.

8.2 Generating consensus sequences for each sample

We will use the `masterVar`-files for each of the 6 samples, and use the `cgCaller.py` script in the `msmc-tools` repository to generate a VCF and a mask file for each individual from the `masterVar` file. For this, I suggest you write a little shell script that loops over all individuals:

```
#!/usr/bin/env bash

MASTERVARDIR=/path/to/sequence_data
OUTDIR=/path/to/output_files
CHR=chr1
for IND in NA19238 NA19239 NA19240 NA12878 NA12891 NA12892; do
    MASTERVAR=$(ls $MASTERVARDIR/masterVarBeta-$IND-*.tsv.chr1.bz2)
    OUT_MASK=$OUTDIR/$IND.$CHR.mask.bed.gz
    OUT_VCF=$OUTDIR/$IND.$CHR.vcf.gz
    cgCaller.py $CHR $IND $OUT_MASK $MASTERVAR | gzip -c > $OUT_VCF
done
```

Here, we restrict analysis only on chromosome 1 (which is called `chr1` in the Complete Genomics data sets). Normally, you would also loop over chromosomes 1-22 in this script.

The line `MASTERVAR=$(ls ...)` uses bash command substitution to look for the masterVar file and store the result in the variable `$MASTERVAR`.

Copy the code above into a shell script, named for example *runCGcaller.sh*, adjust the paths, make it executable via `chmod u+x runCGcaller.sh` and run it. You should see log messages indicating the currently processed position in the chromosome. Chromosome 1 has about 250 million sites, so you can estimate the waiting time.

When finished (should take 10-20 minutes for all 6 samples), you should now have one `*.mask.bed.gz` and one `*.vcf.gz` file for each individual.

8.3 Combining samples

Some explanation on the generated files: The VCF file in each sample contains all sites at which at least one of the two chromosomes differs from the reference genome. Here is a sample:

```
##fileformat=VCFv4.1
##FORMAT=<ID=GT,Number=1,Type=String,Description="Phased Genotype">
#CHROM      POS     ID      REF     ALT     QUAL    FILTER  INFO    FORMAT  NA19238
chr1        38232   .       A       G       .       PASS    .       GT      1/1
chr1        41981   .       A       G       .       PASS    .       GT      1/1
chr1        47108   .       G       C       .       PASS    .       GT      1/0
chr1        47292   .       T       G       .       PASS    .       GT      1/0
chr1        49272   .       G       A       .       PASS    .       GT      1/1
chr1        51673   .       T       C       .       PASS    .       GT      1/0
chr1        52058   .       G       C       .       PASS    .       GT      1/0
```

This alone would not be enough information. MSMC is a Hidden Markov Model which uses the density of heterozygous sites (`1/0` genotypes) to estimate the time to the most recent common ancestor. However, for a density you need not only a numerator but also a denominator, which in this case is the number of non-heterozygous sites, so typically homozygous reference alleles. Those are not part of this VCF file, for efficiency reasons. That's why we have a Mask-file for each sample, that gives information in which regions in the genome could be called. Regions with not enough coverage or too low quality will be excluded. The first lines of such a mask look like this:

```
chr1        11093   11101
chr1        11137   11154
chr1        11203   11235
chr1        11276   11288
chr1        11319   11371
chr1        11378   11387
chr1        11437   11453
chr1        11481   11504
chr1        11511   11527
chr1        11568   11637
```

which gives a very detailed view on which regions could be called (2nd and 3rd column are begin and end).

There is one more mask that we need, which is the mappability mask. This mask defines regions in the reference genome in which we trust the mapping to be of high quality because the reference sequence is unique in that area. The mappability mask for chromosome 1 for the human reference GRCh37 is included in the Tutorial files. For all other chromosomes, this README includes a link to download them, but we won't need them in this tutorial.

For generating the input files for MSMC, we will use a script called `generate_multihetsep.py`, which merges VCF and mask files together, and also performs simple trio-phasing. I will first show a command line that generates and MSMC input file for a single diploid sample *NA12878*:

```
#!/usr/bin/env bash

INDIR=/path/to/VCF/and/mask/files
OUTDIR=/path/to/output_files
MAPDIR=/path/to/mappability/mask
generate_multihetsep.py --chr 1 --mask $INDIR/NA12878.mask.bed.gz \
    --mask $MAPDIR/hs37d5_chr1.mask.bed $INDIR/NA12878.vcf.gz > $OUTDIR/NA12878.chr1.
↪multihetsep.txt
```

Here we have added the mask and VCF file of the NA12878 sample, and the mappability mask. I suggest you don't actually run this because we won't need this single-sample processing.

To process these two trios, we will use the two offspring samples only to phase the four parental chromosomes. You can do this with the trio option:

```
#!/usr/bin/env bash

INDIR=/path/to/VCF/and/mask/files
OUTDIR=/path/to/output_files
MAPDIR=/path/to/mappability/mask
generate_multihetsep.py --chr 1 \
    --mask $INDIR/NA12878.chr1.mask.bed.gz --mask $INDIR/NA12891.chr1.mask.bed.gz --
↪mask $INDIR/NA12892.chr1.mask.bed.gz \
    --mask $INDIR/NA19240.chr1.mask.bed.gz --mask $INDIR/NA19238.chr1.mask.bed.gz --
↪mask $INDIR/NA19239.chr1.mask.bed.gz \
    --mask $MAPDIR/hs37d5_chr1.mask.bed --trio 0,1,2 --trio 3,4,5 \
    $INDIR/NA12878.chr1.vcf.gz $INDIR/NA12891.chr1.vcf.gz $INDIR/NA12892.chr1.vcf.gz \
    $INDIR/NA19240.chr1.vcf.gz $INDIR/NA19238.chr1.vcf.gz $INDIR/NA19239.chr1.vcf.gz \
    > $OUTDIR/EUR_AFR.chr1.multihetsep.txt
```

Here we have first input all 6 calling masks, plus one mappability mask, then the two trio specifications (see `~/msmc-tools/generate_multihetsep.py -h` for details), and then the 6 VCF files.

The first lines of the resulting "multihetsep" file should look like this:

```
1   68306   44      TTTCTCCT,TTTCCTTC
1   68316   10      CCCTTCCT,CCCTCTTC
1   87563   13      CCTTTTTT
1   570089  259     TTTTCCCC
1   752566  1058    AAAAAGAA
1   752721  83      GGGGGAGA
1   756781  596     GGGGGGGA
1   756912  113     AGAAAAAA
1   757103  26      CCCCCCCT
1   757734  84      TTTTTCTT
```

This is the input file for MSMC. The first two columns denote chromosome and position of a segregating site within the samples. The fourth column contains the 8 alleles in the 8 parental haplotypes of the four parents we put in. When there are multiple patterns separated by a comma, it means that phasing information is ambiguous, so there are multiple possible phasings. This can happen if all three members of a trio are heterozygous, which makes it impossible to separate the paternal and maternal allele.

The third column is special and I get a lot of questions about that column, so let me explain it as clearly as possible: The third column contains the number of called sites *since the previous segregating site, including the current site*. So for example, in the first row above, the first segregating site is at position 68306, but not all 68306 sites up to that site were called homozygous reference, but only 44. This is very important for MSMC, because it would otherwise assume that there was a huge homozygous segment spanning from 1 through 68306. Note that the very definition given above also means that the third column is always greater or equal to 1 (which is actually enforced by MSMC)!

8.4 Running MSMC2 for estimating the effective population size

MSMC's purpose is to estimate coalescence rates between haplotypes through time. This can then be *interpreted* for example as the inverse effective population size through time. If the coalescence rate is estimated between subpopulations, another interpretation would be how separated the two populations became through time. In this tutorial, we will use both interpretations.

As a first step, we will use MSMC2 to estimate coalescence rates within the four African haplotypes alone, and within the four European haplotypes alone. Here is a short script running both these cases:

```
#!/usr/bin/env bash

INPUTDIR=/path/to/multihetsep/files
OUTDIR=/path/to/output/dir

msmc2 -p 1*2+15*1+1*2 -o $OUTDIR/EUR.msmc2 -I 0,1,2,3 $INPUTDIR/EUR_AFR.chr1.
↪multihetsep.txt
msmc2 -p 1*2+15*1+1*2 -o $OUTDIR/AFR.msmc2 -I 4,5,6,7 $INPUTDIR/EUR_AFR.chr1.
↪multihetsep.txt
```

Let's go through the parameters one by one. The `-p 1*2+15*1+1*2` option defines the time segment patterning. By default, MSMC uses 32 time segments, grouped as `1*2+25*1+1*2+1*3`, which means that the first 2 segments are joined (forcing the coalescence rate to be the same in both segments), then 25 segments each with their own rate, and then again two groups of 2 and 3, respectively. MSMC2 run time and memory usage scales quadratically with the number of time segments. Here, since we are only analysing a single chromosome, you should reduce the number of segments to avoid overfitting. That's why I set 18 segments, with two groups in the front and back. Grouping helps avoiding overfitting, as it reduces the number of free parameters.

The `-o` option denotes an output prefix. The three files generated by msmc will be called like this prefix with endings `.final.txt`, `.loop.txt` and `.log`.

The `-I` option denotes the 0-based indices of the haplotypes analysed. In our case we have 8 haplotypes, the first four being of European ancestry, the latter of African ancestry. In the first run we estimate coalescence rates within the European chromosomes (indices 0,1,2,3), and in the second case within the African chromosomes (indices 4,5,6,7). The last argument to `msmc2` is the multihetsep file. Normally you would run it on all 22 chromosomes, and in that case you would simply give all those 22 files in a row.

On one processors, each of those runs will take about one hour, so that's too long to actually run it, but you should at least test whether it starts alright and then kill the job using CTRL-C. The output files of the runs are available in the tutorial files.

8.5 Estimating population separation history

Above we have run MSMC on each population individually. In order to better understand when and how the two ancestral populations separated, we will use MSMC to estimate the coalescence rate across populations. Here is a script for this run:

```
#!/usr/bin/env bash

INPUTDIR=/path/to/multihetsep/files
OUTDIR=/path/to/output/dir

msmc2 -I 0-4,0-5,1-4,1-5 -s -p 1*2+15*1+1*2 -o $OUTDIR/AFR_EUR.msmc2 $INPUTDIR/EUR_
↪AFR.chr1.multihetsep.txt
```

Here, I am running on all pairs between the first two parental chromosomes in each subpopulation, so `-I 0-4,0-5,1-4,1-5`. If you wanted to analyse all eight haplotypes (will take consiberably longer), you would have had to type `-I 0-4,0-5,0-6,0-7,1-4,1-5,1-6,1-7,2-4,2-5,2-6,2-7,3-4,3-5,3-6,3-7`.

The `-s` flag tells MSMC to skip sites with ambiguous phasing. As a rule of thumb: For population size estimates, we have found that unphased sites are not so much of a problem, but for cross-population analysis we typically remove those.

8.6 Plotting in Python

For plotting in python, we first need to import the pandas library via:

```
import pandas as pd
```

The result files from MSMC2 look like this:

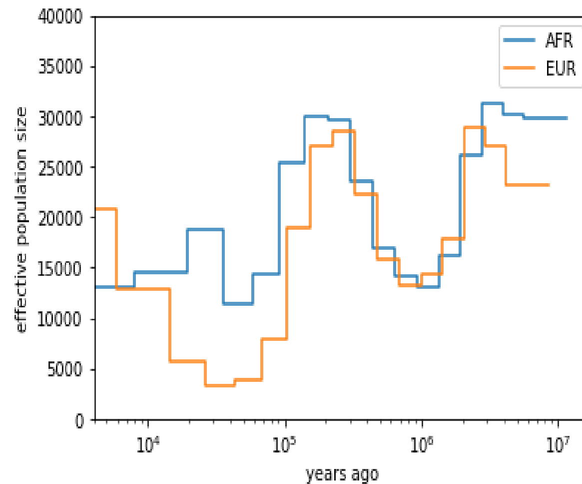
```
time_index  left_time_boundary      right_time_boundary     lambda
0   0       2.61132e-06     2.93162
1   2.61132e-06     6.42208e-06     3043.06
2   6.42208e-06     1.19832e-05     3000.32
3   1.19832e-05     2.00987e-05     8353.98
4   2.00987e-05     3.19418e-05     12250.1
5   3.19418e-05     4.92247e-05     8982.41
...
```

Here, the first column denotes a simple index of all time segments, the second and third indicate the scaled start and end time for each time interval. The last column contains the scaled coalescence rate estimate in that interval.

Let's first plot the effective population sizes with the following python code:

```
mu = 1.25e-8
gen = 30
afrDat = pd.read_csv("/path/to/AFR.msmc2.final.txt", delim_whitespace=True)
eurDat = pd.read_csv("/path/to/EUR.msmc2.final.txt", delim_whitespace=True)
plt.step(afrDat["left_time_boundary"]/mu*gen, (1/afrDat["lambda"])/(2*mu), label="AFR
↪")
plt.step(eurDat["left_time_boundary"]/mu*gen, (1/eurDat["lambda"])/(2*mu), label="EUR
↪")
plt.ylim(0,40000)
plt.xlabel("years ago");
plt.ylabel("effective population size");
plt.gca().set_xscale('log')
plt.legend()
```

Obviously, you have to adjust the path to the final result files under `/data/msmc`. The code produces this plot:



You can see that both ancestral population had similar effective population sizes before 200,000 years ago, after which the European ancestors experienced a severe population bottleneck. Of course, this is relatively low resolution because we are only analysing one chromosome, but the basic signal is already visible. Note that here we have scaled times and rates using a generation time of 30 years and a mutation rate of 1.25e-8, which are the same values as used in the initial publication on MSMC

For the cross-population results, we would like to plot the coalescence rate across populations relative to the values within the populations. However, since we have obtained these three rates independently, we have allowed MSMC2 to choose different time interval boundaries in each case, depending on the observed heterozygosity within and across populations. We therefore first have to use the script `~/msmc-tools/combinedCrossCoal.py`:

```
#!/usr/bin/env bash

DIR=/path/to/msmc/results

combineCrossCoal.py $DIR/EUR_AFR.msmc2.final.txt $DIR/EUR.msmc2.final.txt \
    $DIR/AFR.msmc2.final.txt > $DIR/EUR_AFR.combined.msmc2.final.txt
```

The resulting file (also available under `/data/msmc` looks like this:

```
time_index  left_time_boundary      right_time_boundary     lambda_00       lambda_01␣
↪       lambda_11
0   1.1893075e-06   4.75723e-06     1284.0425703    2.24322 2650.59574175
1   4.75723e-06     1.15451e-05     3247.01877925   2.24322 2940.90417746
2   1.15451e-05     2.12306e-05     7798.2270432    99.0725 2526.98957475
3   2.12306e-05     3.50503e-05     11261.3153077   2271.31 2860.21608183
4   3.50503e-05     5.47692e-05     8074.85679367   4313.17 3075.15793155
```

Here, instead of just one columns with coalescence rates, as before, we now have three. The first is the rate within population 0, the second across populations, the third within population 1.

OK, so we can now plot the relative cross-coalescence rate as a function of time:

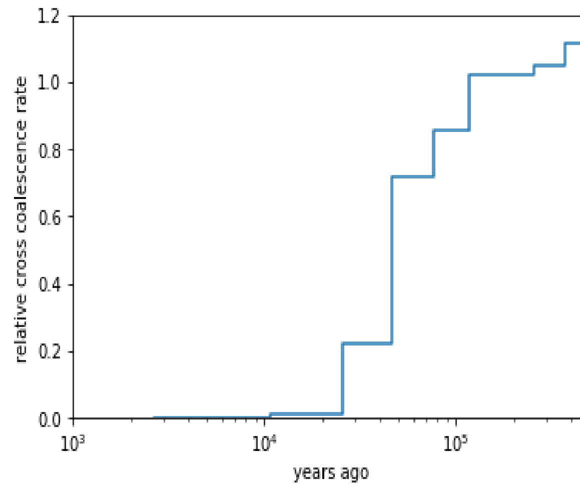
```
mu = 1.25e-8
gen = 30
crossPopDat = pd.read_csv("/path/to/EUR_AFR.combined.msmc2.final.txt", delim_
↪whitespace=True)
plt.step(crossPopDat["left_time_boundary"]/mu*gen, 2 * crossPopDat["lambda_01"] /␣
↪(crossPopDat["lambda_00"] + crossPopDat["lambda_11"]))
plt.xlim(1000,500000)
```

(continues on next page)

(continued from previous page)

```
plt.xlabel("years ago");
plt.ylabel("relative cross coalescence rate");
plt.gca().set_xscale('log')
```

which produces this plot:



where you can see that the separation of (West-African) and European ancestors began already 200,000 years ago. The two populations then became progressively more separated over time, reaching a mid-point of 0.5 around 80,000 years ago. Since about 45,000 years, the two population seem fully separated on this plot. Note that even in simulations with a sharp separation, MSMC would not produce an infinitely sharp separation curve, but introduces a "smear" around the true separation time, so this plot is compatible also with the assumption that the two populations where already fully separated around 60,000 years ago, even though the relative cross-coalescence rate is not zero at that point yet.