# Genome Analysis Workshop 2018 Documentation

***Release 1.0***

**Stephan Schiffels**

**Jan 09, 2019**

# Contents:

This is the documentation for the 2018 Workshop delivered at the Next Generation Genomics 2018 Symposium Workshops in Helsinki by Stephan Schiffels

CHAPTER 1

# Notes on Infrastructure

The practical exercises of this workshop will be running on a separate virtual machine (VM) for each participant. Access to this virtual machine can be obtained using various channels, including a custom terminal client software (such as Terminal on Mac OS X) and Jupyter, a browser based system for interactive computing. For the following, I will assume access via Jupyter.

## 1.1 How to Connect to Jupyter

To connect to the Jupyter interface for your virtual machine, open your favourite Browser, go to the VM Server entry page, and click on the Jupyter link indicated for your particular machine. You will be asked to enter the password you have been given for your personal VM.

## 1.2 Basic Usage

When you first access Jupyter, you will get a file browser view of the `~/work` folder on your VM. In the beginning that list will be empty, and will be populated with notebooks and files throughout this workshop.

To create a new text file, click on *New* 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 under `~/work` on your VM.

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.

---

**Note:** To use the terminal, some basic knowledge of Unix is required. For this workshop, you will frequently use tools such as `pwd` to view the current directory, `ls` to view the contents of the current directory, `cd` to change the directory, `cat` to output the contents of a file, `grep` to search in a text file and other commands.

---

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 `fun`, and a text file within that folder using Jupyter. Fill the text file with arbitrary content, such as "Hello, World!". Then open a terminal and output the new text file using the `cat` command.

---

## 1.3 Notebooks

Notebook can be loaded for different underlying kernels: bash, python 2, python3 and R. In this tutorial, we will use bash and python 3. 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 `# This is a heading` into the cell, press Shift-Enter and watch. Then type `This is text with \*italic\* and \*\*bold\*\* letters`. 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`.

---

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
```

---

**Exercise**

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

---

CHAPTER 2

# 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.

**Hint:** You can find the solution notebooks for all exercises in this and subsequent sessions here

## 2.1 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 3,902 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. They are located unser `~/share/genotype_data`. You can use the bash terminal, and use `less -S <FILENAME>` to view each file and skim through it to get a feeling for the data.

Alternatively, you can create use a Bash notebook, use `cd` as above, and then use the unix tools `head` in combination with `cut` to show portions of the files (see solutions notebook `bash_commands`).

---

**Exercise**

Confirm that there are 1,351 individuals in the dataset. (Advanced) Count how many different populations there are. Hint: You can use the Unix tools `awk '{print $3}'`, `sort` and `uniq -c` to achieve that (see solutions notebook `bash_commands`).

---

## 2.2 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.

## 2.3 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.

## 2.4 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:

```
/home/training/share/WestEurasia.poplist.txt
/home/training/share/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 any population name that you don't recognise to get a feeling for the ethnic groups represented here.

---

## 2.5 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 (as done in the solution notebook `bash_commands`). Alternatively, you can use the Jupyter file editor to create the parameter file.

---

... and run smartPCA using the command `smartpca -p <PARAMS_FILE>`

---

**Exercise**

Run `smartpca` with the prepared parameter file.

---

---

**Note:** Running `smartPCA` with this dataset takes between 15 and 30 minutes.

---

---

**Hint:** `smartpca` outputs a flurry of log messages that may be useful later. If you run the program within a Jupyter Notebook, you can always go back later and view the log, as it is saved within the notebook. If you choose to run it through a terminal, you should direct the output into a file, e.g. like this `smartpca -p PARAMS_FILE > output.log`.

---

To facilitate further processing, I have put the results file into `~/share/pca_results/pca.WestEurasia.*` and `~/share/pca_results/pca.AllEurasia.*`

## 2.6 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 Python3 interface, and here I opted for matplotlib.

I suggest that you start a new Jupyter Notebook with the Python3 language, 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 going to the terminal and running `head EVEC_FILE`, where `EVEC_FILE` should obviously replaced with the actual filename of the PCA run. 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. Please see the `02_pca_python` solution notebook if you need help. You should now have the pca data loaded into a dataframe.

---

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

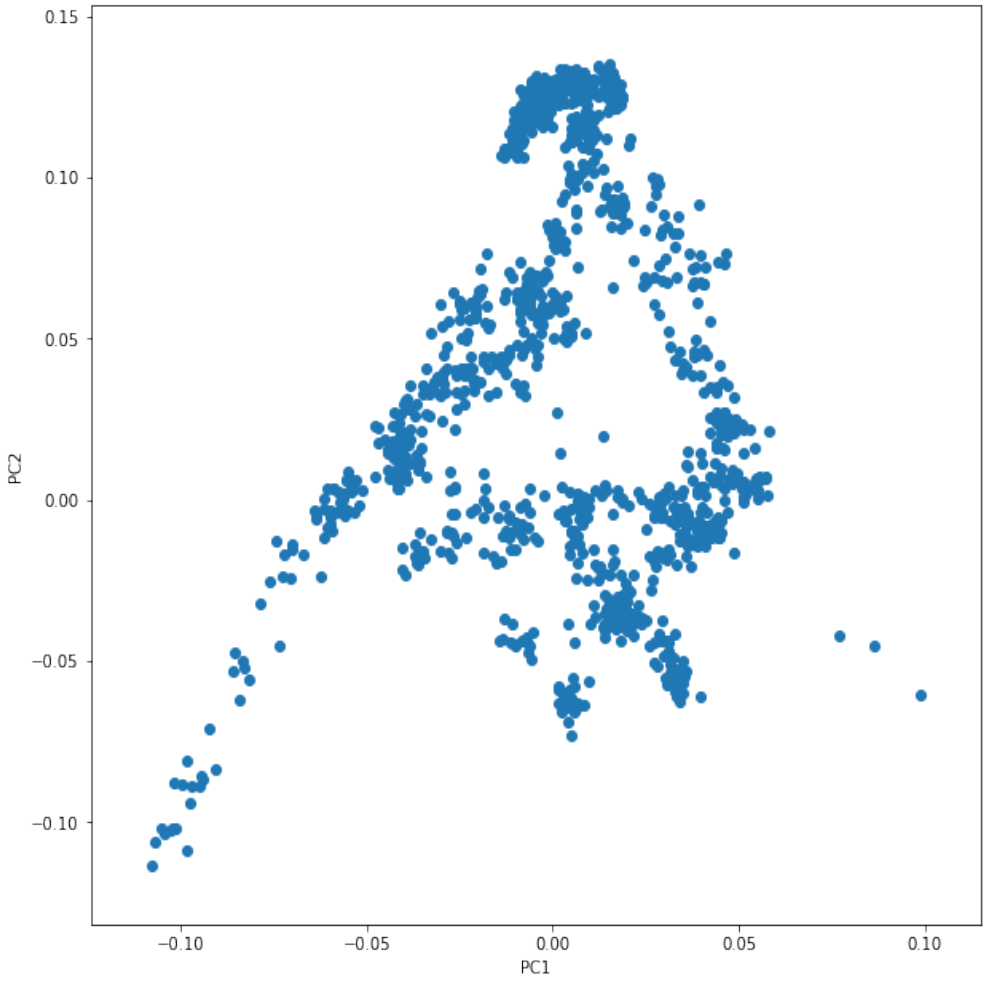
```
Name     PC1     PC2     PC3     PC4     Population
        I001    -0.0192    0.0353   -0.0024   -0.0084    Ignore_Iran_
↪Zoroastrian(PCA_outlier)
        I002    -0.0237    0.0372   -0.0018   -0.0133    Ignore_Iran_
↪Zoroastrian(PCA_outlier)
   IREJ-T006    -0.0226    0.0417    0.0045    0.0003    Iran_Non-Zoroastrian_
↪Fars
```

(continues on next page)

(continued from previous page)

```
   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:

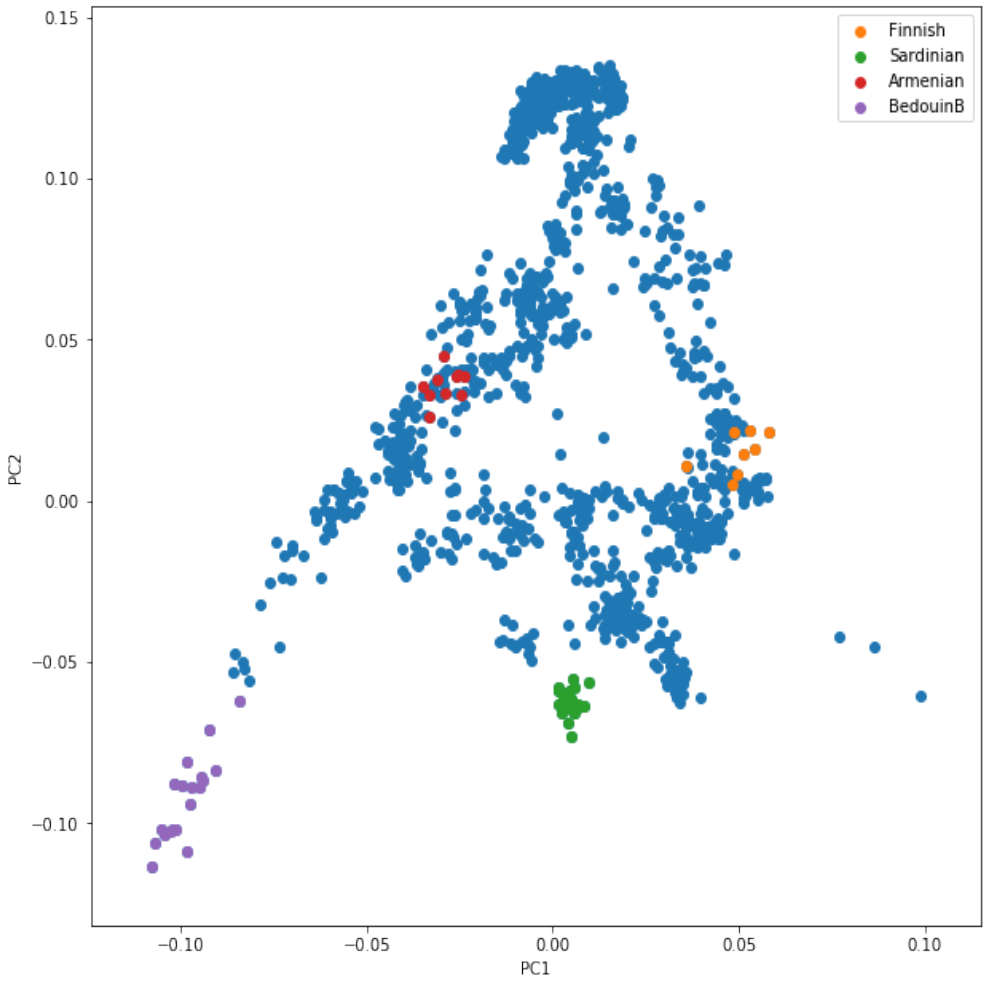
```
plt.scatter(x=pcaDat["PC1"], y=pcaDat["PC2"], label="")
for pop in ["Finnish", "Sardinian", "Armenian", "BedouinB"]:
    d = pcaDat[pcaDat["Population"] == pop]
    plt.scatter(x=d["PC1"], y=d["PC2"], label=pop)
```

(continues on next page)

(continued from previous page)

```
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("~/share/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

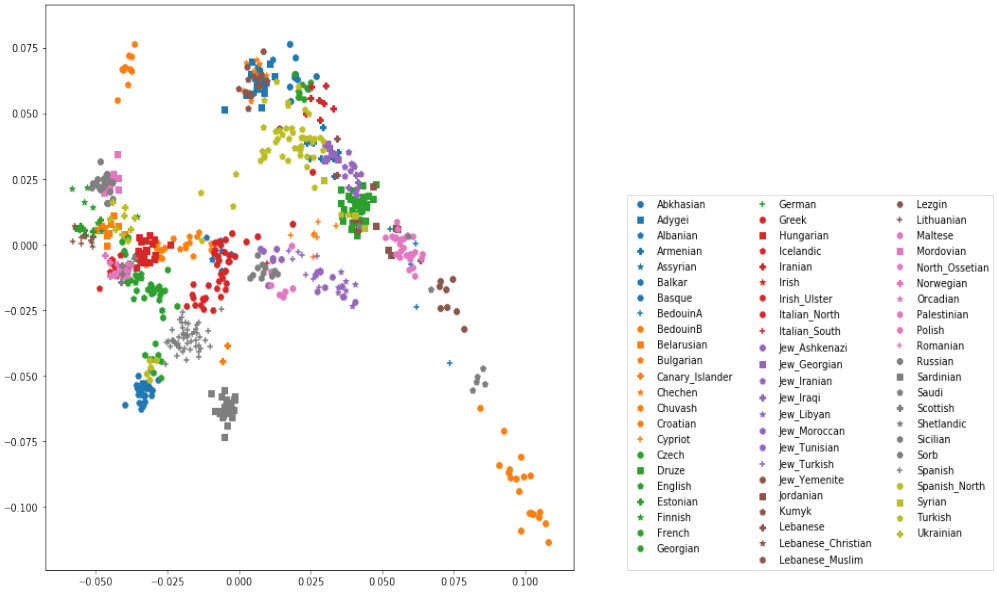
in the `popListDat` dataframe, and plots each listed population in turn, with its assigned color and symbol. To prepare, we need a list of colors and symbols. Here, I am using the default color sequence from `matplotlib` and a manual sequence of symbols, which for the sake of simplicity I simply put 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):




## 2.7 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".

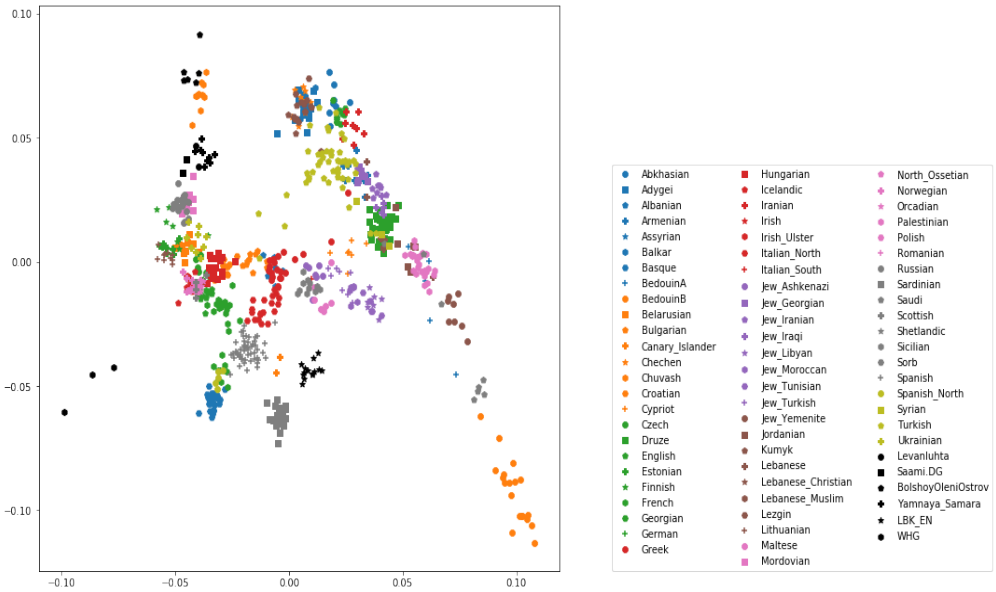
---

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

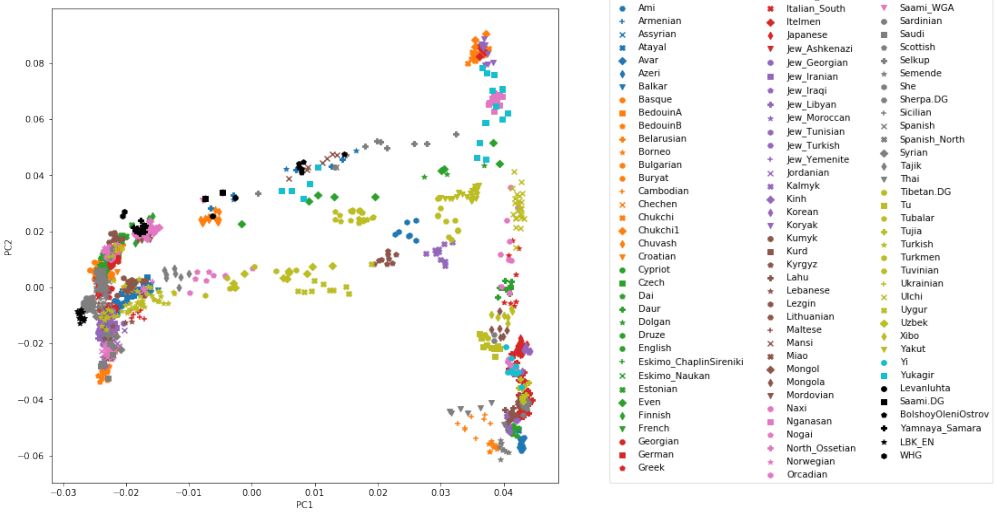
**Exercise**

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:



CHAPTER 3

# F Statistics

## 3.1 F3 Statistics

F3 statistics are a useful analytical tool to understand population relationships. F3 statistics, just as F4 and F2 statistics measure allele frequency correlations between populations and were introduced by Nick Patterson in his Patterson 2012

F3 statistics are used for two purposes: i) as a test whether a target population (C) is admixed between two source populations (A and B), and ii) to measure shared drift between two test populations (A and B) from an outgroup (C).

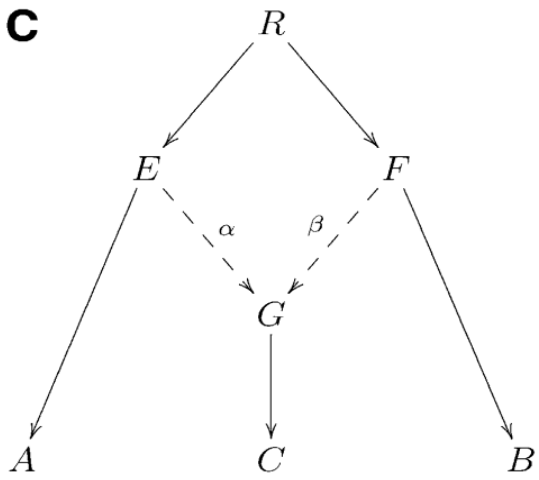
F3 statistics are in both cases defined as the product of allele frequency differences between population C to A and B, respectively:

$$F3(A, B; C) = \langle (c - a)(c - b) \rangle$$

Here, $\langle \cdot \rangle$ denotes the average over all genotyped sites, and $a, b$ and $c$ denote the allele frequency for a given site in the three populations $A$, $B$ and $C$.

## 3.2 Admixture F3 Statistics

It can be shown that if that statistics is negative, it provides unambiguous proof that population C is admixed between populations A and B, as in the following phylogeny (taken from Figure 1 from Patterson 2012):




Intuitively, an F3 statistics becomes negative if the allele frequency of the target population (C) is on average intermediate between the allele frequencies of A and B. Consider as an extreme example a genomic site where $a = 0, b = 1$ and $c = 0.5$. Then we have $(c-a)(c-b) = -0.25$, which is negative. So if the entire statistics is negative, it suggests that in many positions, the allele frequency $c$ is indeed intermediate, suggesting admixture between the two sources.

---

**Note:** If an F3 statistics is *not* negative, it does *not* proof that there is no admixture!

---

We will use this statistics to test if Finnish are admixed between East and West, using different Eastern and Western sources. In the West, we use French, Icelandic, Lithuanian and Norwegian as source, and in the East we use Nganasan and one of the populations analysed in this workshop, *Bolshoy Oleni Ostrov*, a 3,500 year old group from the Northern Russian Kola-peninsula.

We use the software `qp3Pop` from AdmixTools, which similar to `smartpca` takes a parameter file:

```
genotypename:   input genotype file (in eigenstrat format)
snpname:        input snp file      (in eigenstrat format)
indivname:      input indiv file    (in eigenstrat format)
popfilename:    a file containing rows with three populations on each line A, B and C.
inbreed: YES
```

Here, the last option is necessary if we are analysing pseudo-diploid ancient data (which is the case here).

To prepare the `popfilename`, open a new file using Jupyter and enter:

```
Nganasan French Finnish
Nganasan Icelandic Finnish
Nganasan Lithuanian Finnish
Nganasan Norwegian Finnish
BolshoyOleniOstrov French Finnish
BolshoyOleniOstrov Icelandic Finnish
BolshoyOleniOstrov Lithuanian Finnish
BolshoyOleniOstrov Norwegian Finnish
```

---

**Exercise**

Prepare the parameter file with the input data as in the PCA session (see *Principal Components Analysis (PCA)*) and then run `qp3Pop -p PARAMETER_FILE`, where `PARAMETERFILE` should be replaced by your parameter file name. This will take about 3 minutes (see the `~/share/solutions/bash_commands` notebook if you need a

hint).

The results are in the output that you can view in the Notebook. The crucial bit should look like this:

```
                    Source 1            Source 2              Target           f_3␣
↪      std. err           Z    SNPs
result:              Nganasan              French             Finnish     -0.004539␣
↪      0.000510      -8.894  442567
result:              Nganasan            Icelandic             Finnish     -0.005297␣
↪      0.000563      -9.404  427954
result:              Nganasan           Lithuanian             Finnish     -0.005062␣
↪      0.000590      -8.574  426231
result:              Nganasan            Norwegian             Finnish     -0.004744␣
↪      0.000569      -8.332  428161
result:    BolshoyOleniOstrov              French             Finnish     -0.002814␣
↪      0.000444      -6.341  402958
result:    BolshoyOleniOstrov            Icelandic             Finnish     -0.002590␣
↪      0.000486      -5.323  386418
result:    BolshoyOleniOstrov           Lithuanian             Finnish     -0.001523␣
↪      0.000536      -2.840  384134
result:    BolshoyOleniOstrov            Norwegian             Finnish     -0.001553␣
↪      0.000502      -3.092  386203
```

This output shows as first three columns the three populations A, B (sources) and C (target). Then the f3 statistics, which is negative in all cases tested here, a standard error, a Z score and the number of SNPs involved in the statistics.

The Z score is key: It gives the deviation of the f3 statistic from zero in units of the standard error. As general rule, a Z score of -3 or more suggests a significant rejection of the Null hypothesis that the statistic is not negative. In this case, all of the statistics are significantly negative, proving that Finnish have ancestral admixture of East and West Eurasian ancestry. Note that the statistics does not suggest *when* this admixture happened!

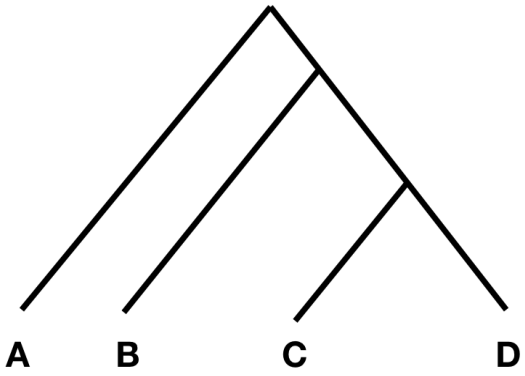
## 3.3 F4 Statistics

A different way to test for admixture is by "F4 statistics" (or "D statistics" which is very similar), also introduced in Patterson 2012.

F4 statistics are also defined in terms of correlations of allele frequency differences, similarly to F3 statistics (see above), but involving four different populations, not just three. Specifically we define

$$F4(A, B; C, D) = \langle (a - b)(c - d) \rangle.$$

To understand the statistics, consider the following tree:




In this tree, without any additional admixture, the allele frequency difference between A and B should be completely independent from the allele frequency difference between C and D. In that case, F4(A, B; C, D) should be zero, or at least not statistically different from zero. However, if there was gene flow from C or D into A or B, the statistic should be different from zero. Specifically, if the statistic is significantly negative, it implies gene flow between either C and B, or D and A. If it is significantly positive, it implies gene flow between A and C, or B and D.

The way this statistic is often used, is to put a divergent outgroup as population A, for which we know for sure that there was no admixture into either C or D. With this setup, we can then test for gene flow between B and D (if the statistic is positive), or B and C (if it is negative).

Here, we can use this statistic to test for East Asian admixture in Finns, similarly to the test using Admixture F3 statistics above. We will use the `qpDstat` program from AdmixTools for that. We need to again prepare a population list file, this time with four populations (A, B, C, D). I suggest you open a new file and fill it with:

```
Mbuti Nganasan French Finnish
Mbuti Nganasan Icelandic Finnish
Mbuti Nganasan Lithuanian Finnish
Mbuti Nganasan Norwegian Finnish
Mbuti BolshoyOleniOstrov French Finnish
Mbuti BolshoyOleniOstrov Icelandic Finnish
Mbuti BolshoyOleniOstrov Lithuanian Finnish
Mbuti BolshoyOleniOstrov Norwegian Finnish
```

You can then use this file again in a parameter file, similar to the one prepared for `qp3Pop` above:

```
genotypename:    input genotype file (in eigenstrat format)
snpname:         input snp file      (in eigenstrat format)
indivname:       input indiv file    (in eigenstrat format)
popfilename:     a file containing rows with three populations on each line A, B and C.
f4mode: YES
```

Note that you cannot give the "inbreed" option here.

---

**Exercise**

Prepare the parameter file as suggested above and then run `qpDstat -p PARAMETER_FILE`, where `PARAMETERFILE` should be replaced by your parameter file name. This will take about 3 minutes (see the `~/share/solutions/bash_commands` notebook if you need a hint).

---

The results should be (skipping some header lines):

```
result:      Mbuti   Nganasan     French    Finnish      0.002363     19.016   29254
↪27852 593124
result:      Mbuti   Nganasan  Icelandic    Finnish      0.001721     11.926   28915
↪27894 593124
result:      Mbuti   Nganasan Lithuanian    Finnish      0.001368      9.664   28745
↪27933 593124
result:      Mbuti   Nganasan  Norwegian    Finnish      0.001685     11.663   28933
↪27934 593124
result:      Mbuti BolshoyOleniOstrov     French    Finnish      0.001962     16.737
↪ 27249  26175 547486
result:      Mbuti BolshoyOleniOstrov  Icelandic    Finnish      0.001084      7.776
↪ 26876  26282 547486
result:      Mbuti BolshoyOleniOstrov Lithuanian    Finnish      0.000554      3.942
↪ 26683  26380 547486
result:      Mbuti BolshoyOleniOstrov  Norwegian    Finnish      0.000952      6.707
↪ 26873  26351 547486
```

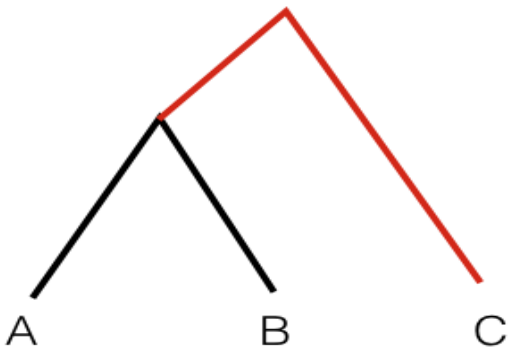
Here, the key columns are columns 2, 3, 4 and 5, denoting A, B, C and D, and column 6 and 7, which denote the F4 statistic and the Z score, measuring significance in difference from zero.

As you can see, in all cases, the Z score is positive and larger than 3, indicating a significant deviation from zero, and implying gene flow between Nganasan and Finnish, and BolshoyOleniOstrov and Finnish, when compared to French, Icelandic, Lithuanian or Norwegian.

## 3.4 Outgroup F3 Statistics

Outgroup F3 statistics are a special case how to use F3 statistics. The definition is the same as for Admixture F3 statistics, but instead of a target C and two source populations A and B, one now gives an outgroup C and two test populations A and B.

To get an intuition for this statistics, consider the following tree:




In this scenario, the statistic F3(A, B; C) measures the branch length from C to the common ancestor of A and B, coloured red. So this statistic is simply a measure of how closely two population A and B are related with each other, as measured from a distant outgroup. It is thus a similarity measure: The higher the statistic, the more genetically similar A and B are to one another.

We can use this statistic to measure for example the the genetic affinity to East Asia, by performing the statistic F3(Han, X; Mbuti), where Mbuti is a distant African population and acts as outgroup here, Han denote Han Chinese, and X denotes various European populations that we want to test.

You need to start, again, by preparing a list of population triples to be measured. I suggest the following list:

```
Han Chuvash Mbuti
Han Albanian Mbuti
Han Armenian Mbuti
Han Bulgarian Mbuti
Han Czech Mbuti
Han Druze Mbuti
Han English Mbuti
Han Estonian Mbuti
Han Finnish Mbuti
Han French Mbuti
Han Georgian Mbuti
Han Greek Mbuti
Han Hungarian Mbuti
Han Icelandic Mbuti
Han Italian_North Mbuti
Han Italian_South Mbuti
Han Lithuanian Mbuti
Han Maltese Mbuti
Han Mordovian Mbuti
Han Norwegian Mbuti
Han Orcadian Mbuti
Han Russian Mbuti
Han Sardinian Mbuti
Han Scottish Mbuti
Han Sicilian Mbuti
Han Spanish_North Mbuti
Han Spanish Mbuti
Han Ukrainian Mbuti
Han Levanluhta Mbuti
Han BolshoyOleniOstrov Mbuti
Han ChalmnyVarre Mbuti
Han Saami.DG Mbuti
```

which cycles through many populations from Europe, including the ancient individuals from Chalmny Varre, Bolshoy Oleni Ostrov and Levänluhta.

---

**Exercise**

Copy this list into a file, and prepare a parameter file for running `qp3Pop`, similar to the parameter file for admixture F3 statistics above, and run `qp3Pop` with that parameter file as above.

---

You should find this (skipping header lines from the output):

```
                    Source 1            Source 2              Target           f_
↪3       std. err           Z    SNPs
result:                  Han             Chuvash              Mbuti      0.233652␣
↪      0.002072     112.782  502678
result:                  Han            Albanian              Mbuti      0.215629␣
↪      0.002029     106.291  501734
result:                  Han            Armenian              Mbuti      0.213724␣
↪      0.001963     108.882  504370
result:                  Han           Bulgarian              Mbuti      0.216193␣
↪      0.001979     109.266  504310
result:                  Han               Czech              Mbuti      0.218060␣
↪      0.002002     108.939  504089
result:                  Han               Druze              Mbuti      0.209551␣
↪      0.001919     109.205  510853
```

(continues on next page)

(continued from previous page)

```
result:                  Han              English              Mbuti       0.216959␣
↪      0.001973     109.954  504161
result:                  Han             Estonian              Mbuti       0.220730␣
↪      0.002019     109.332  503503
result:                  Han              Finnish              Mbuti       0.223447␣
↪      0.002044     109.345  502217
result:                  Han               French              Mbuti       0.216623␣
↪      0.001969     110.012  509613
result:                  Han             Georgian              Mbuti       0.214295␣
↪      0.001935     110.721  503598
result:                  Han                Greek              Mbuti       0.215203␣
↪      0.001984     108.465  507475
result:                  Han            Hungarian              Mbuti       0.217894␣
↪      0.001999     109.004  507409
result:                  Han            Icelandic              Mbuti       0.218683␣
↪      0.002015     108.553  504655
result:                  Han        Italian_North              Mbuti       0.215332␣
↪      0.001978     108.854  507589
result:                  Han        Italian_South              Mbuti       0.211787␣
↪      0.002271      93.265  492400
result:                  Han           Lithuanian              Mbuti       0.219615␣
↪      0.002032     108.098  503681
result:                  Han              Maltese              Mbuti       0.210359␣
↪      0.001956     107.542  503985
result:                  Han             Mordovian              Mbuti       0.223469␣
↪      0.002008     111.296  503441
result:                  Han             Norwegian              Mbuti       0.218873␣
↪      0.002023     108.197  504621
result:                  Han              Orcadian              Mbuti       0.217773␣
↪      0.002014     108.115  504993
result:                  Han               Russian              Mbuti       0.223993␣
↪      0.001995     112.274  506525
result:                  Han             Sardinian              Mbuti       0.213230␣
↪      0.001980     107.711  508413
result:                  Han              Scottish              Mbuti       0.218489␣
↪      0.002039     107.145  499784
result:                  Han              Sicilian              Mbuti       0.212272␣
↪      0.001975     107.486  505477
result:                  Han        Spanish_North              Mbuti       0.215885␣
↪      0.002029     106.383  500853
result:                  Han               Spanish              Mbuti       0.213869␣
↪      0.001975     108.297  513648
result:                  Han             Ukrainian              Mbuti       0.218716␣
↪      0.002007     108.950  503981
result:                  Han          Levanluhta              Mbuti       0.236252␣
↪      0.002383      99.123  263049
result:                  Han   BolshoyOleniOstrov              Mbuti       0.247814␣
↪      0.002177     113.849  457102
result:                  Han         ChalmnyVarre              Mbuti       0.233499␣
↪      0.002304     101.345  366220
result:                  Han             Saami.DG              Mbuti       0.236198␣
↪      0.002274     103.852  489038
```

Now it's time to plot these results using python.

**Exercise**

Copy the results (all lines from the output beginning with "results:") into a text file, open a Jupyter python3 notebook and load the text file into a pandas dataframe, using `pd.read_csv(FILENAME, delim_whitespace=True, names=["dummy", "A", "B", "C", "F3", "StdErr", "Z", "SNPS"]`. View the resulting dataframe and make sure it looks correct.

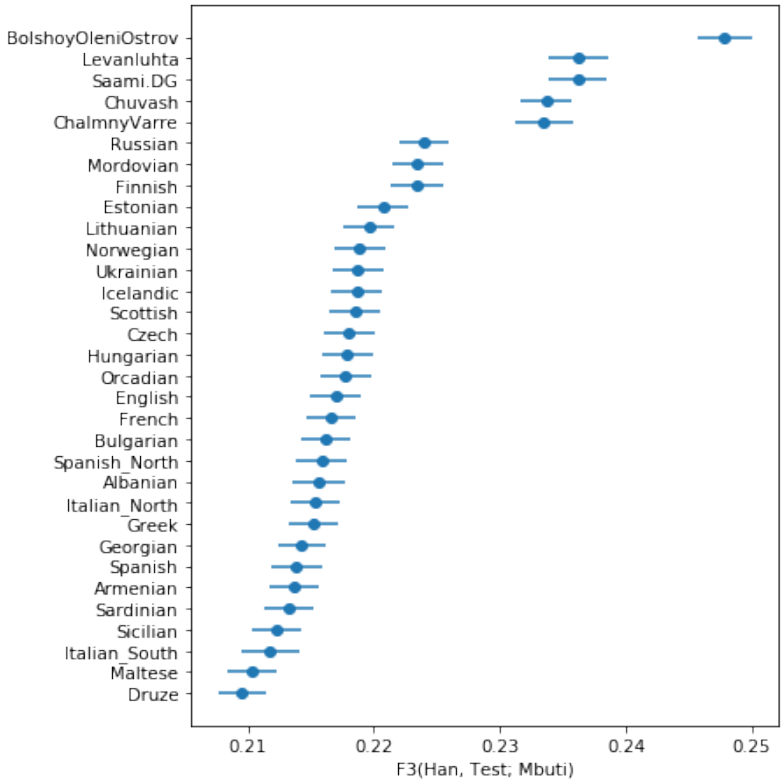
A useful way to plot these results is by sorting them by the F3 statistics, and then plotting the test populations from left to right, beginning with the largest values. This code snippet should do the trick:

```
d=f3dat_han.sort_values(by="F3")
y = range(len(d))
plt.figure(figsize=(6, 8))
plt.errorbar(d["F3"], y, xerr=d["stderr"], fmt='o')
plt.yticks(y, d["B"]);
plt.xlabel("F3(Han, Test; Mbuti)");
```

**Exercise**

Use the above code snippet to plot the Outgroup F3 data. Google the `errorbar` and `yticks` functions from matplotlib if you want to know how they works.

You should get something like this:




showing that, as expected, The ancient samples and modern Saami are most closely related to modern East Asians (as represented by Han) compared to many other Europeans.

## 3.5 Outgroup F3 Statistics Biplot

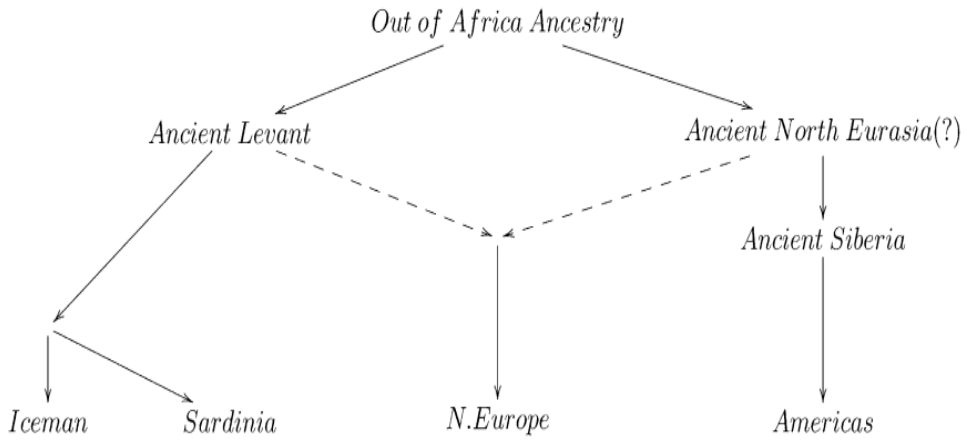
The above plot shows an intriguing cline of differential relatedness to Han in many Europeans. For example, would you have guessed that Icelandics are closer to Han than Armenians are to Han? This is very surprising, and it shows that European ancestry has a complex relationship to East Asians. To understand this better, you can read Patterson 2012, who makes some intriguing observations. Patterson and colleagues use Admixture F3 statistics and apply it to many populations world-wide. They summarise some population triples with the most negative F3 statistics in the following table:

**Table 5 Three-population test in HGDP**

| Source1 | Source2 | Target | $f_3$ | *Z*-score |
|---|---|---|---|---|
| Japanese | Italian | Uygur | −0.0259 | −74.79 |
| Japanese | Italian | Hazara | −0.0230 | −74.05 |
| Yoruba | Sardinian | Mozabite | −0.0211 | −56.95 |
| Mozabite | Surui | Maya | −0.0149 | −19.67 |
| Yoruba | San | Bantu-SA | −0.0107 | −31.39 |
| Yoruba | Sardinian | Palestinian | −0.0107 | −36.70 |
| Yoruba | Sardinian | Bedouin | −0.0104 | −33.73 |
| Druze | Yi | Burusho | −0.0090 | −27.62 |
| Sardinian | Karitiana | Russian | −0.0086 | −20.68 |
| Druze | Karitiana | Pathan | −0.0084 | −22.25 |
| Han | Orcadian | Tu | −0.0076 | −20.64 |
| Mbuti | Orcadian | Makrani | −0.0076 | −19.56 |
| Han | Orcadian | Mongola | −0.0075 | −19.21 |
| Han | French | Xibo | −0.0069 | −16.92 |
| Druze | Dai | Sindhi | −0.0067 | −21.99 |
| Sardinian | Karitiana | French | −0.0060 | −18.36 |
| Dai | Italian | Cambodian | −0.0060 | −13.16 |
| Sardinian | Karitiana | Adygei | −0.0057 | −13.03 |
| Biaka | Sardinian | Bantu-Kenya | −0.0054 | −13.42 |
| Sardinian | Karitiana | Tuscan | −0.0052 | −11.26 |
| Sardinian | Pima | Italian | −0.0045 | −12.48 |
| Druze | Karitiana | Balochi | −0.0044 | −11.58 |
| Daur | Dai | Han | −0.0026 | −13.20 |
| Han | Orcadian | Han-NChina | −0.0025 | −7.09 |
| Han | Yakut | Daur | −0.0025 | −9.05 |
| Druze | Karitiana | Brahui | −0.0025 | −6.43 |
| Hezhen | Dai | Tujia | −0.0021 | −6.97 |
| Sardinian | Karitiana | Orcadian | −0.0019 | −4.31 |
| She | Yakut | Oroqen | −0.0017 | −5.13 |

There are many interesting results here, but one of the most striking one is the finding of F3(Sardinian, Karitiana; French), which is highly significantly negative. This statistics implies that French are admixed between Sardinians and Karitiana, a Native American population from Brazil. How is that possible? We can of course rule out any recent Native American backflow into Europe.

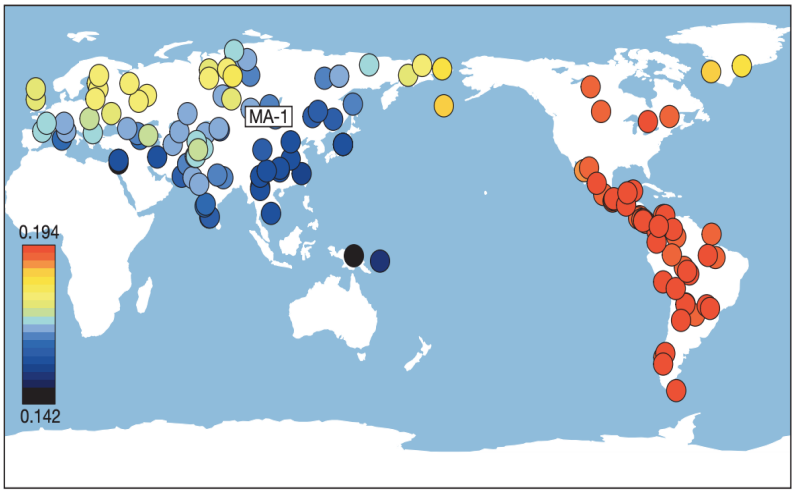
Patterson and colleagues explained this finding with hypothesising an ancient admixture event, from a Siberian population that contributed to both Europeans and to Native Americans. They termed that population the "Ancient North Eurasians (ANE)". The following admixture graph was suggested:




As you can see, the idea is that modern Central Europeans, such as French, are admixed between Southern Europeans (Sardinians) and ANE. The Ancient North Eurasians are a classic example for a "Ghost" population, a population which does not exist anymore in unmixed form, and from which we have no direct individual representative.

Amazingly, two years after the publication of Patterson 2012, the ANE ghost population was actually found: Raghavan et al. and colleagues, in 2014, published a paper called "Upper Palaeolithic Siberian genome reveals dual ancestry of Native Americans". A 24,000 year old boy (called MA1) from the site of "Mal'ta" in Siberia was shown to have close genetic affinity with both Europeans and in particular Native Americans, just as proposed in Patterson 2012.

The affinities are summarised nicely in this figure from Raghavan et al.:




OK, so we now know that ancestry related to Native Americans contributed to European countries. Could that possibly explain the affinity of our ancient samples and Saami to Han Chinese in some way? To test this, we will run the same Outgroup F3 statistics as above, but this time not with Han but with MA1 as test population. Specifically, we run the following population triples in `qp3Pop`:

```
MA1_HG.SG Chuvash Mbuti
MA1_HG.SG Albanian Mbuti
MA1_HG.SG Armenian Mbuti
MA1_HG.SG Bulgarian Mbuti
MA1_HG.SG Czech Mbuti
MA1_HG.SG Druze Mbuti
MA1_HG.SG English Mbuti
```

(continues on next page)

(continued from previous page)

```
MA1_HG.SG Estonian Mbuti
MA1_HG.SG Finnish Mbuti
MA1_HG.SG French Mbuti
MA1_HG.SG Georgian Mbuti
MA1_HG.SG Greek Mbuti
MA1_HG.SG Hungarian Mbuti
MA1_HG.SG Icelandic Mbuti
MA1_HG.SG Italian_North Mbuti
MA1_HG.SG Italian_South Mbuti
MA1_HG.SG Lithuanian Mbuti
MA1_HG.SG Maltese Mbuti
MA1_HG.SG Mordovian Mbuti
MA1_HG.SG Norwegian Mbuti
MA1_HG.SG Orcadian Mbuti
MA1_HG.SG Russian Mbuti
MA1_HG.SG Sardinian Mbuti
MA1_HG.SG Scottish Mbuti
MA1_HG.SG Sicilian Mbuti
MA1_HG.SG Spanish_North Mbuti
MA1_HG.SG Spanish Mbuti
MA1_HG.SG Ukrainian Mbuti
MA1_HG.SG Levanluhta Mbuti
MA1_HG.SG BolshoyOleniOstrov Mbuti
MA1_HG.SG ChalmnyVarre Mbuti
MA1_HG.SG Saami.DG Mbuti
```

where `MA1_HG.SG` is the cryptic name for the MA1 genome from Raghavan et al..

---

**Exercise**

Follow the same protocol as above: Copy the list into a file, prepare a parameter file for `qp3Pop` with that population triple list, and run `qp3Pop`. Copy the results (all lines beginning with "results:") into a file and load it into python via `pd.read_csv()`.

---

To test in what way the relationship to Han Chinese is correlated with the relationship with MA1, we will now plot the two statistics against each other in a scatter plot. We first have to merge the two outgroup-F3 datasets together. Here is the code including loading (assuming that the two F3 dataframes are called `outgroupf3dat_Han` and `outgroupf3dat_MA1`):

```
outgroupf3dat_Han = pd.read_csv("/home/training/work/outgroupF3_results_Han.txt",
                        delim_whitespace=True,
                        names=["dummy", "A", "B", "C", "F3", "stderr", "Z", "nSNPs"])
outgroupf3dat_MA1 = pd.read_csv("/home/training/work/outgroupF3_results_MA1.txt",
                        delim_whitespace=True,
                        names=["dummy", "A", "B", "C", "F3", "stderr", "Z", "nSNPs"])

outgroupf3dat_merged = outgroupf3dat_Han.merge(outgroupf3dat_MA1, on="B", suffixes=("_
↪Han", "_MA1"))
```

---

**Exercise**

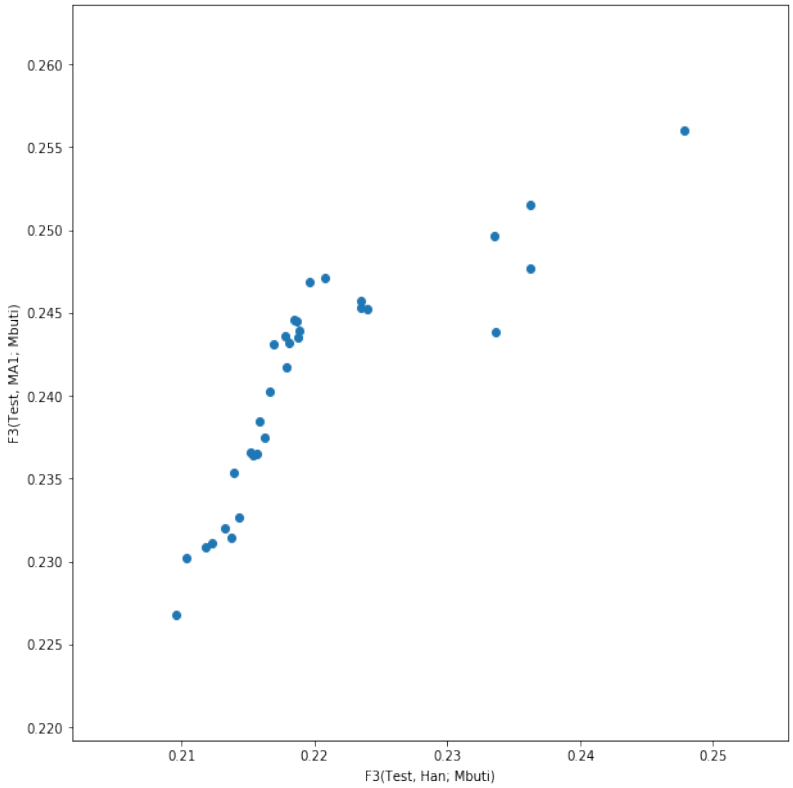
run the above merge command and check that it worked by viewing the resulting dataframe.

---

Finally, we can produce our bi-plot, using this code:

```
plt.figure(figsize=(10, 10))
plt.scatter(x=outgroupf3dat_merged["F3_Han"], y=outgroupf3dat_merged["F3_MA1"])
plt.xlabel("F3(Test, Han; Mbuti)");
plt.ylabel("F3(Test, MA1; Mbuti)");
```

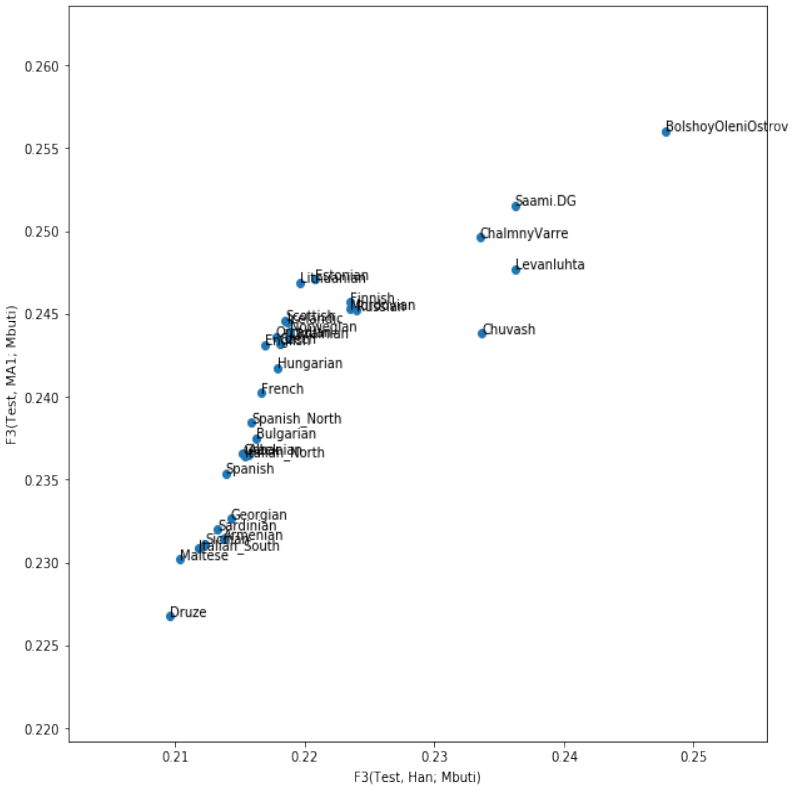
This should yield something like this:




This isn't very useful, however, as we cannot see which point is which population. We can use the annotation function from matplotlib to add text labels to each point:

```
plt.figure(figsize=(10, 10))
plt.scatter(x=outgroupf3dat_merged["F3_Han"], y=outgroupf3dat_merged["F3_MA1"])
for i, row in outgroupf3dat_merged.iterrows():
    plt.annotate(row["B"], (row["F3_Han"], row["F3_MA1"]))
plt.xlabel("F3(Test, Han; Mbuti)");
plt.ylabel("F3(Test, MA1; Mbuti)");
```

which should yield:




**Exercise**

Create this plot with the code snippets above.

The result shows that indeed the affinity to East Asians in the bulk of European contries can be explained by MA1-related ancestry. Most European countries have a linear relationship between their affinity to Han and their affinity to MA1. However, this is not true for our ancient samples from Fennoscandia and for modern Saami and Chuvash, who have extra affinity to Han not explained by MA1 (Lazaridis et al. 2014).

CHAPTER 4

# MSMC

## 4.1 Prerequisites

We first need to download some python scripts from the msmc-tools repository. To do that, go to your home directory and run `git clone https://github.com/stschiff/msmc-tools`

You should now have a directory called `msmc-tools` in your home-folder, as you can verify by running `ls`.

## 4.2 Data

All input data and intermediate files for this tutorial are at `~/share/MSMC-tutorial-files/`.

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.

## 4.3 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
```

(continues on next page)

(continued from previous page)

```
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
    ~/msmc-tools/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.

## 4.4 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
~/msmc-tools/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)!

## 4.5 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.

## 4.6 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.

## 4.7 Plotting in Python

The result files from MSMC2 look like this:

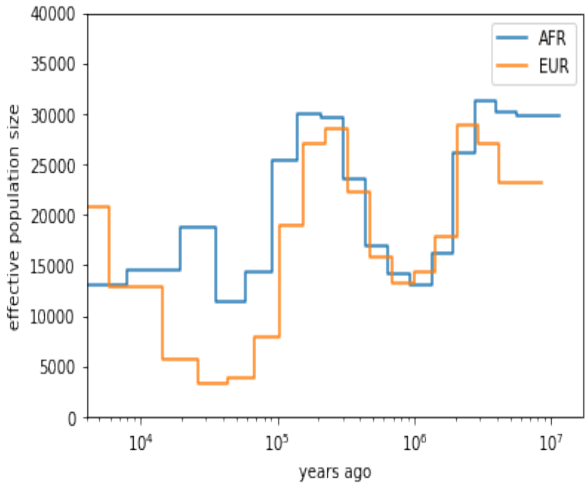
```
time_index  left_time_boundary      right_time_boundary     lambda
0   0       2.61132e-06     2.93162
1   2.61132e-06     6.42208e-06     3043.06
2   6.42208e-06     1.19832e-05     3000.32
3   1.19832e-05     2.00987e-05     8353.98
4   2.00987e-05     3.19418e-05     12250.1
5   3.19418e-05     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 `~/share/MSMC-tutorial-files`. 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 `~/share/MSMC-tutorial-files` 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:

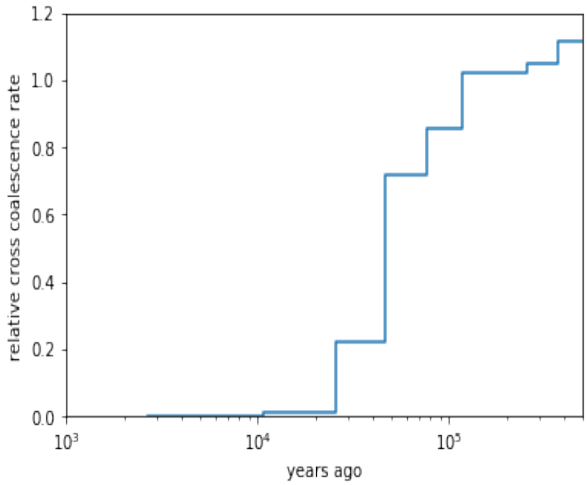
```
mu = 1.25e-8
gen = 30
crossPopDat = pd.read_csv("/path/to/EUR_AFR.combined.msmc2.final.txt", delim_
↪whitespace=True)
plt.step(crossPopDat["left_time_boundary"]/mu*gen, 2 * crossPopDat["lambda_01"] /␣
↪(crossPopDat["lambda_00"] + crossPopDat["lambda_11"]))
plt.xlim(1000,500000)
```

(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.

CHAPTER 5

# Solution Notebooks

- A jupyter notebook with all bash commands from this workshop: https://nbviewer.jupyter.org/github/stschiff/compvar-workshop-docs/blob/master/supp/bash_commands.ipynb
- A jupyter notebook with python commands for pca and f statistics: https://nbviewer.jupyter.org/github/stschiff/compvar-workshop-docs/blob/master/supp/python_pca_fstats.ipynb
- A jupyter notebook with the python commands used for plotting the MSMC results: https://nbviewer.jupyter.org/github/stschiff/compvar-workshop-docs/blob/master/supp/python_MSMC.ipynb

CHAPTER 6

# Indices and tables

- genindex
- modindex
- search