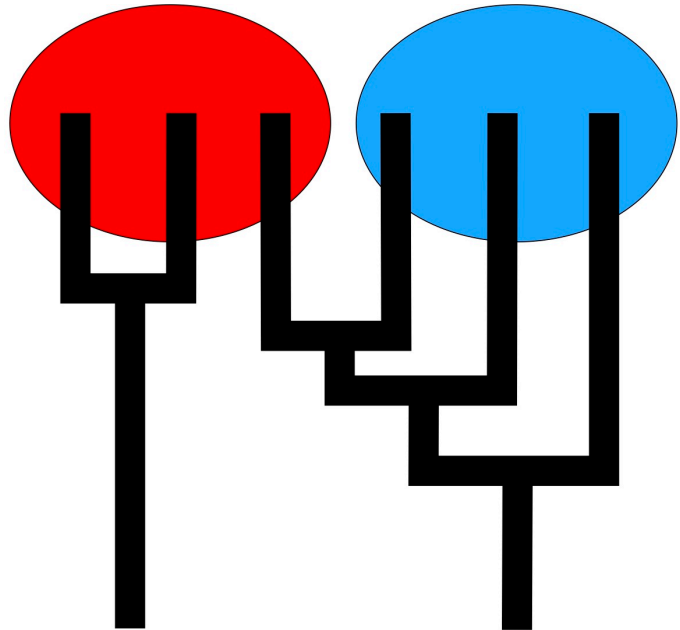


Peter Beerli

**Talk: Analysis of genomic
sequence data using finite sites
models**

**Tutorial: Population
model selection
using a combination
of VCF and
reference data**



[Tutorial link](#)

I will assume that the users of the tutorial know some basic UNIX commands, such as the bash or zsh shell, that they can run Python 3 scripts [you will need python installed on your system]. Installation of *migrate-n* on Macs and Windows/UNIX is described in the Installation section.

Overview of the activity:

1. This tutorial discusses the use of *migrate-n* for model selection.
2. We assume that you have a genomic data set in VCF format and a reference sequence; as an example, we use a small dataset using a VCF dataset of contemporary human populations. We use a VCF of two African populations; the reference is chromosome 23.
3. We need python3 to extract and combine the VCF with the reference sequence and generate a *migrate-n* dataset. In the future, this will be incorporated into *migrate-n*, but currently, we need to preprocess. Our handling of the VCF data is also rather rudimentary (if you want to use your VCF data with *migrate-n* and conversion fails, contact us so that we

can improve the translation script).

4. To successfully run the tutorial in a short time, we would need a cluster computer; we will shortcut some of the steps, and also give hints on how to troubleshoot problems.
5. A first run of migrate, we will discuss how to specify models for migrate-n; we will explain the layout of the output.
6. *Migrate-n* uses Bayesian model comparison using marginal likelihoods, in the last few minutes of the tutorial, we will compare a few simple models that were run with the human dataset.

Installation

Best to be done before the meeting! If you downloaded the ssb-tutorial.zip file and unpacked it, you should see a directory migrate-5.0.7 and a tutorial directory. This file is part of the tutorial.

Installation of *migrate*

a. on a Mac computer:

- from source: If you do not have the necessary tools installed, you may need to install them using a system like "homebrew" [you would need minimally (first install *brew*; then *brew install autoconf automake libtool pkg-config gcc open-mpi zlib*)

```
cd migrate-5.0.7
autoreconf -fi
./configure
make
# The binaries migrate-n and migrate-n-mpi are in the src directory
# and should be copied
# where you want the executable. For example, in ~/bin
```

- as binary: copy the appropriate binary for your machine from *migrate-5.0.7/bin* to wherever you need it [I usually use ~/bin, test the file: by ~/bin/migrate-n -version, if it shows the version it should be fine.]

b. on a Windows computer:

We will use the LINUX software that can be easily run your Windows11 computer. If you never used that you may need to install several packages to make this work.

Open ****PowerShell (Administrator)**** and run:

```
wsl --install
```

Reboot if prompted. After reboot, start Ubuntu from the Start Menu and complete the initial setup.

#in a Ubuntu shell install the following tools:

```
sudo apt update
```

```
sudo apt install -y \  
    build-essential \  
    autoconf automake libtool pkg-config \  
    zlib1g-dev \  
    openmpi-bin libopenmpi-dev \  
    git wget unzip
```

#download the tutorial

```
wget https://peterbeerli.com/tutorials/ssb-tutorial-2026.zip
```

```
unzip ssb-tutorial-2026.zip
```

```
cd ssb-tutorial-2026/migrate-5.0.7
```

```
autoreconf -fi
```

```
./configure
```

```
make
```

The binaries migrate-n and migrate-n-mpi are in the src directory

and should be copied

where you want the executable. For example, in ~/bin

Tutorial

Learning goals: Students are able to generate a *migrate* input file from a VCF data file, and can specify different population genetic models and compare them in a Bayesian model selection approach.

Preparing your dataset

I added a simple VCF dataset to the tutorial directory. I suggest that we use this dataset for our tutorial. The tutorial should work the same way with your own dataset. [The VCF dataset was created using stdpopsim -- see the file stdpopsim.log for details]

Currently the VCF dataset option does not exist in *migrate*. In future version you will be able directly to use VCF data, but for now we have a python script to convert the VCF data into a

migrate format.

```
python vcf2mig.py --help
```

will deliver

```
syntax: vcf2mig --vcf vcffile.vcf
        <--ref|--abbrevref> ref1.fasta,ref2.fasta,... | --linksnp n
        <--popspec numpop ind1 ind2 .... | --pop populationfile.txt>
        <--chrom chr1,chr2,...>
        <--bound start,stop>
        --out migrateinfile
```

Details:

`--vcf vcffile` : a VCF file that is uncompressed or .gz, currently only few VCF options are allowed, simple reference and alternative allele, diploid and haploid data can be used

`--abbrevref ref1.fasta,ref2.fasta,...` : reference in fasta format for more info see next option, returns snps + invariant c

`--ref ref1.fasta,ref2.fasta,...` : reference in fasta format several references can be given, for example for each chromosome, if this option is NOT present then the migrate dataset will contain only the SNPs

`--allowindel` if there are indels or deletions they will be used and not

`--linksnp <number|chrom>`: cannot not be used with `--ref`; defines linkage the keyword 'chrom' will link all snps within one chromos the 'number' specifies the distance among snps that are 1 read from first to last snp, so if number=1000 and the fi then all snps within the x+1000 will belong to the linkag If this option and the `--ref` are are missing, then the re will contain single, unlinked snps

`--popspec numpop ind1,ind2,...` : specify the population structure, number with the number of individuals for each population This option excludes the option `--pop`; if the numbers do then the options takes precedence and distributes accordi

`--pop popfile`: specify a file that contains a single line with (use space numpop ind1 ind2 ... This option exlcudes the option `--popspec`

`--chrom chr1,chr2,...` specify subset of chromosomes in vcf file if all chromosomes are used ignore this option

`--bound start,stop` specifies the left and right bound of the reference se snps are only reported within the bound

`--out migratedatafile`: specify a name for the converted dataset in migra

```
--strict: replaces all characters that are not ACGTN? with ?
```

Example:

```
vcf2mig.py --vcf vcffile.vcf.gz --ref ref.fasta --popspec 2 10,10 --out mig
vcf2mig.py --vcf vcffile.vcf --popspec 3 10,10,10 --out migratefile
vcf2mig.py --vcf vcffile.vcf --linksnp 10000 --popspec 2 5,10 --out migrate
```

```
You specified:['/Users/beerli/bin/vcf2mig.py', '--help']
```

There are three main modes of the script. (1) generate a complete sequence using a reference to fill in the non-SNP sites; (2) use the VCF data to generate a SNP dataset; (3) use the reference to generate information about the base frequencies for linked or unlinked SNPs.

(1) Full reference sequence

This will allow the use of phylogenetic-style mutation models, such as JC69, F84, HKY, and Tamura-Nei, including gamma-deviated site-rate variation. This will result in analyses that differ from those you would perform with Site Frequency Spectra, which seems to be the common analysis approach when we have large amounts of genomic data. This allows us to consider finite-site mutation models that are more appropriate for analyses of highly variable species than the infinite-sites or no-mutation models.

We call the script like this:

```
vcf2mig.py --vcf chr22-3pop.vcf.gz --popspec 3 4,4,4 --ref chr22sim.fasta -
```

But before we run the script, we peek into the raw VCF file and the reference sequence to get an idea what we need to do, here the first few lines of the VCF file (1.vcf.gz -- if you want to look yourself use for example *emacs* or *zless* because the file is compressed and your editor needs to know that:

```
##fileformat=VCFv4.2
##source=tskit 0.6.4
##FILTER=<ID=PASS,Description="All filters passed">
##contig=<ID=chr22,length=50818468>
##FORMAT=<ID=GT,Number=1,Type=String,Description="Genotype">
#CHROM POS ID REF ALT QUAL FILTER INFO FORMAT YRI
chr22 15288558 0 G C . PASS . GT
chr22 15288622 1 T G . PASS . GT
chr22 15289025 2 T C . PASS . GT
chr22 15289265 3 T C . PASS . GT
chr22 15289830 4 A G . PASS . GT
```

The VCF file is rather simple with reference and alternative allele, my script ignores the Ancestral Allele specification, it recognizes the diploid (or haploid) data case and uses the header file to get the names of the individual in the data, here we have 5 individual with YRI and 5 with CHB stem, we recognize these as two populations, each has 5 diploid individuals each.

The reference file needs to be a standard FASTA file, it can contain multiple entries. For example, for each chromosome. Our example has only a single chromosome, and given that the VCF we use comes from a simulator (*stdpopsim*) there may be other issues. Here the first few bytes of the FASTA reference file:

[illegible]

We then call your script [the current version is very picky about spelling and the two dashes (--)]:

```
vcf2mig.py --vcf chr22-3pop.vcf.gz --popspec 3 4,4,4 --ref chr22sim.fasta -
```

where the option **--popspec** gives the number of populations and the number of (diploid) individuals. Chr22 has ~50Mb nad it seems that the reference sequences has about 15Mb of unspecified (N) nucleotides at the beginning, for our tutorial I extracted 1Mb from the region 30Mb to 31Mb, this should allow to run the examples on machines with lower amounts of RAM (I did tests with my Macbook using 24 sequences each 50Mb long, but this needs considerable amount of RAM -- I had 128 GB -- running this large dataset on HPC at FSU failed because th constraint of RAM for each task is around 5 GB.)

This will generate a file that contains the reference sequence augmented by the VCF; but is not yet ready to analyze, the first few lines look like this:

```

3 1 chr22-3pop.vcf.gz
# VCF file used:      chr22-3pop.vcf.gz
# Translated from VCF 2025-12-30
# Reference file: chr22sim.fasta
# Migrate input file: infile.migrate
# References augmented with VCF data file!
# Using references augmented VCF data with bound [30000001,31000000]
(s1000000)
# individual name length is 10!
  8 Pop1
YRI_0:1      GACTCTGTCGCAAAAAAAAAAATTAATGTAACCATTTAGCCATTACCCAGTTCCTTAAAT..
YRI_0:2      GACTCTGTCGCAAAAAAAAAAATTAATGTAACCATTTAGCCATTACCCAGTTCCTTAAAT..
YRI_1:1      GACTCTGTCGCAAAAAAAAAAATTAATGTAACCATTTAGCCATTACCCAGTTCCTTAAAT..
YRI_1:2      GACTCTGTCGCAAAAAAAAAAATTAATGTAACCATTTAGCCATTACCCAGTTCCTTAAAT..
...

```

The *infile* would force *migrate* run analyze for each individual 1 million sites. Although this is actually possible, it may not be desirable because it would be treated as a single locus (on my Macbook Pro M4 with 128 GB RAM the datasets runs with defaults in about 7 minutes using 660MB memory because most of the sites are aliased). The data was simulated with a recombination rate larger than zero. *migrate* has a shortcut to handle large genomic data by specifying a set of equidistant loci along the genome so that we can specify 10, 100, or any arbitrary number of loci out of the complete dataset. The shortcut takes the instruction of the number of sites

```
(s1000000)
```

and replaces it with (for example)

```
[5o1000](s1000000)
```

and you also need to change the number of loci on the first line: the first number is the number of populations and the second number is the number of loci:

```
3 1 chr22-3pop.vcf.gz
```

becomes

```
3 5 chr22-3pop.vcf.gz
```

This specifies 5 loci, each 1000 base pairs long. and once *migrate* runs this converts to

```
(0s1000) (200000s1000) (400000s1000) (600000s1000) (800000s1000)
```

One can also input that string directly in the infile instead of the shortcut.

These individual loci can be specified by hand if needed. Our dataset is now ready! *migrate* is a Bayesian inference program that uses Markov chain Monte Carlo to deliver a posterior probability density of the parameters of interest. With many loci this can be a very frustrating process because the runtime depends on the number of individuals, the number of populations, and the number of loci. If we need for a single locus 7 minutes, this translates to quite some time for, say, 1000 loci. *migrate* can be run in parallel using standard MPI interfaces, such as OPENMPI or MVAPICH2. Using a large number of parallel running CPUs cuts down the time considerably. I often run large datasets on our cluster using 501 CPUs to analyze 500 loci in parallel; this cuts down runtime even on laptops with multiple cores.

(1) SNP data set without a reference sequence

If we are not interested to reconstitute the complete sequence, we can simply translate the VCF variant calls to a SNPS (currently there is no quality control possible) We can call the script like this:

```
vcf2mig.py --vcf chr22-3pop.vcf.gz --linksnp Chrom -popspec 3 4,4,4 -out in  
vcf2mig.py --vcf chr22-3pop.vcf.gz --linksnp 100000 -popspec 3 4,4,4 -out i
```

or like this:

```
vcf2mig.py --vcf chr22-3pop.vcf.gz -popspec 3 4,4,4 -out infile.unlinkedsnp
```

The first variant creates linked snps, the first example combines all snps on a chromosome into one linkage group, the second combines all snps that are in position 0-100000, 100001-200000, ... into groups, the example data has a million sites, thus the commands generates 1 locus or 10 loci with completely linked snps. The third variant generates unlinked snps. I am not sure how usefule this is because the example generates >100,000 single SNP loci, which will take a very long time to run, and since the *migrate* uses random coalescences among individuals this seems not to be the best use of computing power because a single snp has a rather boring simple tree. with two groups (here: all individual with A and all individuals with T), thus, *migrate* spends a large amount of time to change trees that are inconsequential with the data. [I will not have time to discuss SNPs in depth and will stick to the reference sequence approach.]

Setting up a population model

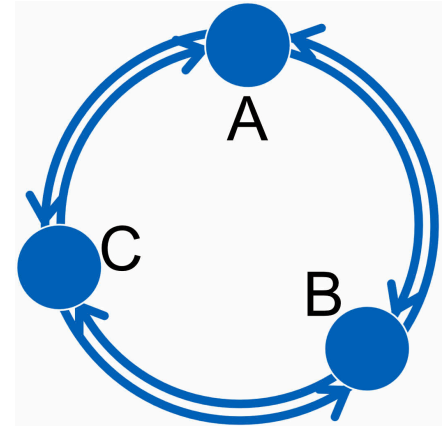
migrate uses an adjacency matrix to define the interactions among populations.

Defaults

The defaults in **migrate** are to estimate all population sizes Θ and all immigration rates M independently.

For example a 3-population model where all populations A, B, and C exchange migrants at individual rates can be specified like this:

	A	B	C
A	x	x	x
B	x	x	x
C	x	x	x



Of course, this is not all that informative, but it is the default.

Specifying models without divergence

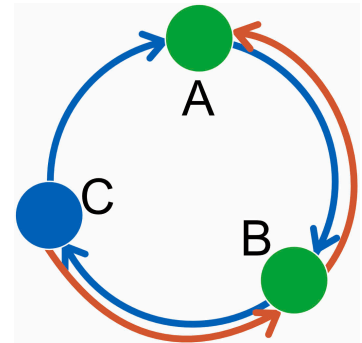
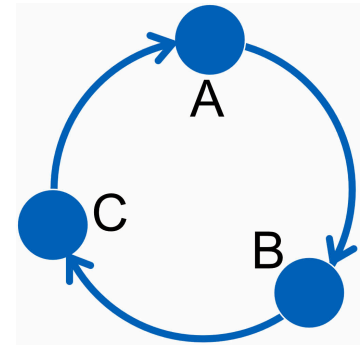
There are many ways to specify different structured models, **migrate** has a set of options to help with that:

- x or '*' means that this parameter is estimated
- 0 means that this parameter is not estimated (works only on off-diagonal elements)
- s means the migration parameter is symmetric, this implies that $M_{1 \rightarrow 2} = M_{2 \rightarrow 1}$
- c means that the value is constant and not estimated, the values needs to be specified in the the start-parameter option
- m means take the average of all parameters labeled m, Θ and M will be treated separately. A matrix full of m will be a two- parameter model (all populations share the same values for 1 Θ and 1 M).
- there are upper case variants for s and m (S, M) that use instead of the migraton parameter M the number of immigrants Nm . In most cases this is not a good choice because Θ and Nm are more strongly correlated than Θ and M .
- all other lower alphabet letters except c, d, t, m, s, t, and x can be used to form groups of parameters (similar to m but more flexible), see example.

More examples:

	A	B	C
A	x	0	x
B	x	x	0
C	0	x	x

	A	B	C
A	e	b	a
B	a	e	b
C	0	a	x



REMINDER: the populations specified on the row of the adjacency matrix are the receiving populations, populations on the columns are the sending populations, thus in the circular unidirectional stepping stone example above, population A receives migrants from population C and we estimate the parameter $M_{C \rightarrow A}$, population B receives migrants from population A and we estimate $M_{A \rightarrow B}$ etc.

Answer these questions

Click on the question to reveal the answer, but write down the answer before you peek :-).

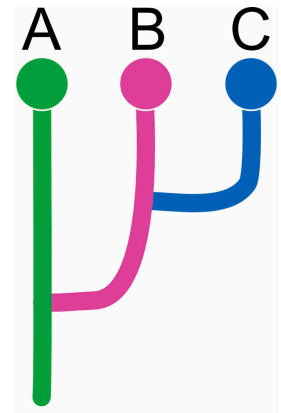
- 1. specify the matrix for a 2-population system with unidirectional migration from 2 \rightarrow 1
- 2. specify a migration matrix for a 5-population system where population 1 and 5 are on a mainland, population 2 is an island close to 1, population 4 is close to 5, and population 3 is far out in the sea but closest to 2. 'Close' means reachable by rafting, and once on an island it will be difficult to get off again.

Specifying models with divergence

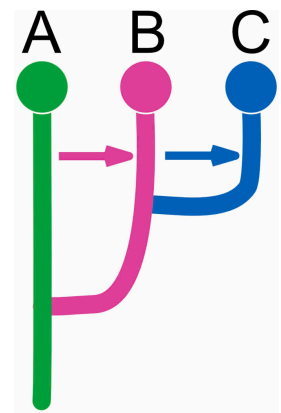
For the divergence specification we also use the adjacency matrix. For example if we have a model where the populations do not exchange migrants we could specify the matrix like this

	A	B	C
A	x	0	0
B	d	x	0
C	0	d	x

- d means divergence from the ancestral column population
- D means divergence from the ancestral column population with ongoing immigration
- t means divergence time is the same for all the matrix elements with the t.

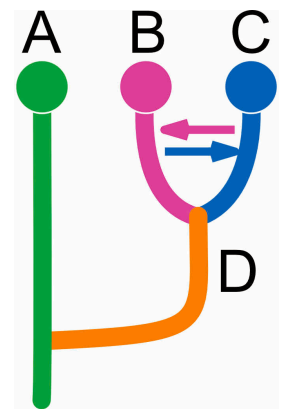


	A	B	C
A	x	0	0
B	D	x	0
C	0	D	x



Migration is problematic and models with immigration and divergence may need additional ancestral populations to allow the estimation to be identifiable. Here an example that needs ancestral population to get better estimates of divergence and migration.

	A	B	C	D
A	x	0	0	0
B	0	x	x	t
C	0	x	x	t
D	d	0	0	x



migrate has a menu to enter this adjacency matrix, but given the many options it may seem easier for complex scenarios to edit the parameter file (it is a text file) directly. the option is named *custom-migration* and contains the adjacency matrix, for example the example from above looks like this

```
custom-migration={ x000 0xxt 0xxt d00x}
```

If a diagonal element is set to zero the program will crash! There are also models that will not work well or fail. All models essentially need to be able to end in one common ancestor, for example we may be tempted to code an admixture example like this

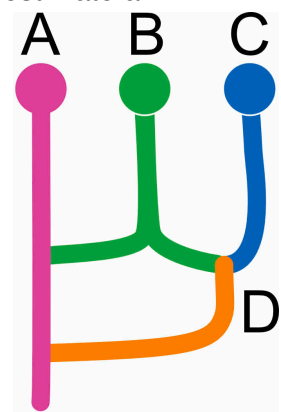
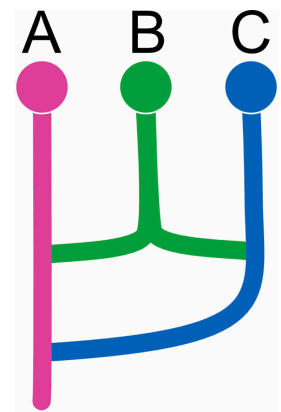
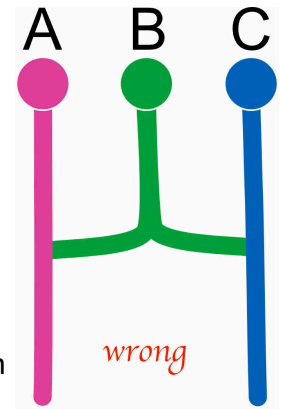
	A	B	C
A	x	0	0
B	d	x	d
C	0	0	x

but this will not work well because we force **to have two roots**, and this can lead to failures during the run, one can code this better as this (a) or this (b)

(a)	A	B	C
A	x	0	0
B	d	x	d
C	d	0	x

More complicated than (a) is this (b) model that forces a sequence of divergences. Both models should deliver similar values so, although (b) will estimate a population size for population D.

(b)	A	B	C	D
A	x	0	0	0
B	d	x	d	0
C	0	0	x	d
D	d	0	0	x



Answer these questions

- 1. specify the matrix for two current populations A, B that share a common ancestor C
- 2. If the first questions is answered using 'd' what does that mean for the divergence times?

Run a model selection exercise

We will have little time so we try to run two examples:

1. using the parmfile specified in the tutorial it uses a model that is coded as

```
1 YRI          * 0 0
2 CEU          d * 0
3 CHB          0 d *
```

run it using: **migrate-n parmfile-x00dx00dx** #do not change the menu

2. Once it is done, we will find the results in *outfile-x00dx00dx* and *outfile-x00dx00dx.pdf*
3. run another parmfile, for example: **migrate-n parmfile-x00xx00xx**
4. we can compare the marginal likelihoods of these two runs, there is a small python script that helps with calculating the numbers, but that can be done "by-hand", too: [I forgot to add the python script *bf.py* in the zipfile; use it like this:

grep " All " outfile* | sort -n -k 4,4 | python bf.py

and it will produce something like this:

Model	Log(mL)	LBF	Model-probability
1:outfile-x0dxn:	-3739322.55	-423376.19	0.0000
2:outfile-x00dx00dxn:	-3733753.34	-417806.98	0.0000
3:outfile-x00dx00dxgn:	-3315946.36	0.00	1.0000

The table includes 3 models, each run over the 1Mb sequence data broken into 10 unlinked loci (→ means a colonization from → to): 1. YRI → (CEU + CHB); 2. YRI → CEU → CHB; 3. YRI → growing CEU → growing CHB;