Using the IM Program

By Jody Hey Department of Genetics, Rutgers University

For questions – check out the *Isolation with Migration* discussion group
http://groups.google.com/group/Isolation-with-Migration

What's new since the last update?

- The option to print histograms of migration counts and mean times has been changed. Instead of recording the mean time of migration for each migration parameter and locus, the times of all migration events are recorded. It turns out that using the mean introduces a number of problems.

Table of Contents

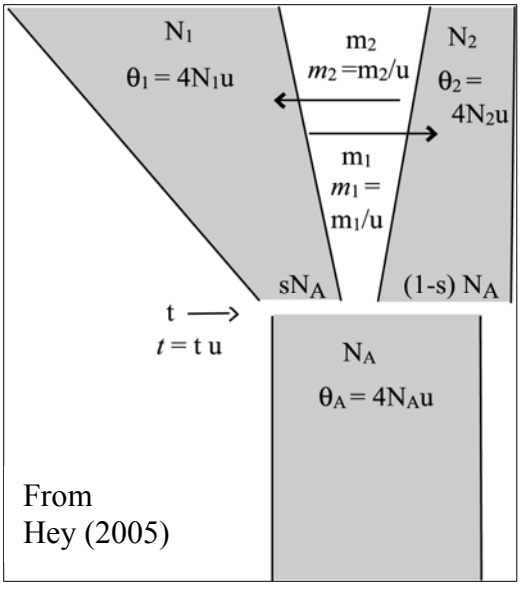
Overview

IM applies the Isolation with Migration model to genetic data drawn from a pair of closely related populations or species. The IM program is descended from the original MDIV program (Nielsen and Wakeley, 2001). The program can be used to generate estimates of the marginal posterior probability densities for each of the model parameters. This document explains how to run the program. New users should first read the "Introduction_to_IM_and_IMa" document.

That basic model, called an 'Isolation with Migration' model has 6 demographic parameters, and is described in the "Introduction_to_IM_and_IMa" document. Unlike the **IMa** program, the **IM** program can also implement a model with changing population sizes.

Isolation with Migration under Changing Population Size (IM program only)

Important limitations of the basic model are that it cannot account for changes in population sizes, and it cannot account for the sizes of founding populations. Both issues are addressed by the inclusion of an additional parameter, *s*, which is the proportion of the ancestral population that founds descendant population 1 ($0 < s < 1$). The remaining fraction of the ancestral population (1-*s*) constitute the founders of population 2. The founding events happen at t, at which time population 1 begins with an effective size of sN_A individuals and population two begins with an effective size of $(1\text{-}s)N_A$ individuals.



During the time between the split and the time of sampling (i.e. t generations) population 1 moves from its starting size to an effective size of N_1 and population 2 moves from its starting size to its current effective size of N_2. In this way each population can have had a history of growth, or shrinkage, from its starting point. Population size change is assumed to have been exponential. The coalescent under changing population size has been addressed by Griffiths and Tavare (1994) and the development and application of

the Isolation with Migration (with population size change) model is given in Hey (2005).

Running IM and Understanding Runtime Information

The program is run by typing and entering a command at a command line prompt. If your computer is running MS Windows, then this will be done in a 'command prompt' window. It is usually simplest to have the program and data files in the same directory (folder), and for the command prompt window to be open in that directory (folder).

The distributed version of the code and the windows executable will handle up to 100 loci and 200 chains. For larger models, values for MAXLOCI and MAXCHAINS in the IM.h file need to be increased and the program recompiled (see **Compilation).** Data is loaded into dynamic memory, so the actual size of a memory model during the run can be large and hard to predict (see **Compilation).**

When running, the program writes back to the screen the current parameter values and updating rates, at intervals of a certain number of steps (default is 10,000 steps). This information includes the following:

- The current step number and the current values for the probability of the data, given the genealogies, and the probability of the genealogies given the current parameters. These probabilities are not particularly useful, but it is good to check that they are actual numbers (non-numerical values indicate a floating point problem and the run should be halted).
- The update rates for all model parameters. These are the percentage of proposed updates that were accepted based on the corresponding Metropolis-Hastings criteria.
- The current locus specific probabilities and the values of the locus specific mutation rate scalar parameters.
- The current values of the other model parameters.
- A table of autocorrelations for the primary model parameters, for a series of lag values. This table begins to appear after 100,000 steps or so.

- Current effective sample size (ESS) estimates. These do not begin to become stable until after about a million steps or so, depending on the data.
- If Metropolis-coupling has been invoked then there are several series of numbers
    - The heating terms for each chain (beta values)
    - The swap rates between successive chains. If these are lower than 5% then they are not doing much good.
    - Actual swap counts between successive chains.
    - The swap rates between chain 0 and the other chains.
    - Actual swap counts between chain 0 and the other chains.

It is a good idea to keep on eye on things, especially during the beginning of a run to be sure that all parameters are being updated and that parameter values are changing. However, even if parameters have a nontrivial update rate (e.g. if > 5% of updates are accepted), it is quite possible that the chain is mixing poorly. Note that even if things are not mixing well at the beginning of a run, the mixing properties may change as the system approaches stationarity, so it is usually advisable to watch a run for awhile before deciding whether or not to restart with different terms.

In general the best approach to ensure mixing is to run the program long enough so that there are no obvious trends in the trendline plots and so the lowest ESS value among the parameters is at least 50 *and* to run the program multiple times (with a different random number seed) to see if results are similar.

It is also important to keep in mind that the parameter estimates provided by the program are based on histograms (i.e. bar charts) that are recorded during the simulation. You may have good mixing, but if the run is not long then the histograms provided in the output file may still provide only a rough idea of where the true peaks are on the posterior. Often this is not too bad. It is far better to have a rough but unbiased idea where the true peak lies, than to have a peak that appears to be well resolved but that actually has come from a chain that never actually mixed. In the latter case, the apparent parameter estimates can be *way* off from the correct ones.

Input File Format (format is identical for both the IM and IMa programs)

IM requires one input file that contains the data for all loci to be considered. The input file should be prepared in the same operating system that the program will be running in. This is to ensure that the end-of-line character differences between unix/linux and dos/windows (and older Macs) do not cause problems. The format is as follows:

- line 1 - arbitrary text, usually explaining the content of the file
  - After line 1, but before line 2, comments can be included to provide explanatory information. Each line of comment must begin with a '#'
- line 2 - two population names, for populations 1 and 2 respectively, separated by one or more spaces
- line 3 - an integer, the number of loci in the data set
- line 4 - basic information for locus 1. This line contains at least five items separated by one or more spaces
  1. Locus name (no spaces within the name)
  2. n1, the sample size for population 1
  3. n2, the sample size for population 2
  4. the length of the sequence (if SSM model – a number is needed here, but it is ignored, if HapSTR, the length pertains to the sequence portion of the data)
  5. Letter indicating the mutation model (I- IS, H - HKY, S - SSM, J - joint SSM and IS = HapSTR) - if this letter is not included on this line, the default is IS. If SSM (S) or HapSTR (J), the letter is followed immediately (no spaces) by the number of linked STR markers within the locus.
  6. Inheritance scalar - For example: 1 for autosome, 0.75 for X-linked, 0.25 for Y-linked or mtDNA.
  7. The mutation rate per year for the locus (not per base pair). This can be left blank, but is needed for estimating parameters on demographic scales. If there are multiple STRs in the locus then there can be multiple mutation rates on this line separated by spaces. If the locus is a HapSTR, then the first mutation rate given applies to the sequence portion of the locus with subsequent values corresponding to STR markers included in the locus.

8. If the mutation rate is given, it can be followed by a range of mutation rates that can be used (with ranges for other loci in the analysis) to set priors on the ratios of mutation rate scalars. The range is entered with an open parentheses, the lowest value, a comma, the highest value, and a closed parentheses (e.g. '(0.00001, 0.00004)'. The range must bracket the rate. For a locus with multiple mutation rates, and multiple ranges, each range follows its corresponding mutation rate immediately on line. See option '`-j7`' for more explanation.

- line 5 - data for gene copy # 1 from population 1. For this line and all other data lines, the first 10 spaces are devoted to the sample name. The sequence or allele length (for SSM model) begins in column 11 of the file. The sequence for a given sample is given all on one line without gaps. For SSM or HapSTR data, the allele length assumes a step size of 1. This means that data from STRs that are multiples of lengths greater than 1 must be converted to counts of the number of base repeats (e.g. for a dinucleotide 'CACACACACACA' the length would be 6). If the data is for an SSM model locus and there are multiple STRs, then there will be one integer on each line for each STR, separated by a space. If the locus is HapSTR (joint IS and SSM) then the STR data is given on the line, beginning at column 11, followed by the sequence data. For SSM data, as for other types of data, only one gene copy is represented on each line of the data file. This is true even if the original data consists of diploid genotypes. In other words, diploid genotype data must be broken up and listed, with one data line for each gene copy.
- lines 6 thru line (n1+n2 +4) - the remainder of the data for locus 1. Each line contains the data for one sample. The data for population 1 samples are given in lines 5 thru line (n1 + 4). The data for population 2 begins on line (n1+5) and proceeds to line (n1+n2+4).
- Additional lines for additional loci - If there is more than one locus, then the data for locus 2 begins on line (n1+n2+5) with a line similar to line 4 presenting the basic information for locus 2. The sample names and sample sizes for locus 2 and the inheritance scalars and mutation model for locus 2 need not be the same as for locus 1.

Finally, the last line of data should end with a newline so that the file ends on a blank line.

Here is an example for a tiny three locus data set. In this made-up example the mutation rate per year is known and specified for locus 1, but not for loci 2 and 3.

```
Example data  for IM
# im test data
population1 population2
3
locus1 1 1 13 I 1 0.0000000008 (0.0000000001, 0.0000000015)
pop1_1    ACTACTGTCATGA
pop2_1    AGTACTATCACGA
hapstrexample 2 1 4 J2 0.75
pop1_1    13 34 GTAC
pop1_2    12 35 GTAT
pop2_1    12 37 GTAT
strexample 2 2 1 S1 1  0.00001 (0.000001, 0.00005)
strpop11a 23
strpop11b 26
strpop21a 25
strpop21b 31
```

Output File

The primary results of a run are contained in an output file. The file is divided into several sections. Most of the tables within the file are formatted with tab stops, and are probably most easily viewed by opening the file in a spreadsheet program. The basic output (apart from additional output indicated at runtime) includes three main sections.

INPUT AND STARTING INFORMATION

This section lists the starting information from the input file and the command line settings.

RUN INFORMATION

This section summarizes information on the run, beginning with the length and time of the run and the number of measurements made. These are followed by the highest values observed for the probability of the data, given the genealogy and the parameters, P(D|G, Params) and the highest values for the probability of the genealogy, given the parameters, P(G|Params). Note that these values are not of much use. They should not be confused, for example, with the likelihoods associated with the maximum likelihood estimates of the parameters. Those likelihoods can only be obtained by integrating over all the genealogies.

Next is listed a table of the means and variances of the times of common ancestry of the genealogies. These can be useful, and if desired the program can be set to record the entire posterior distribution of common ancestor times. The units on these values are the same as for t .

Next are listed the rates at which parameter and genealogy updates were accepted, over the course of the run. For genealogies these include the total update rate (G) and the rate at which the branching (B) pattern was updated.

Following the update rates, the output file has a table of parameter autocorrelations and ESS (effective sample size) estimates. These are the same as appear on the computer screen during the course of the run. ESS values are estimates of the number of independent points that have been sampled for each parameter. They can be a valuable guide to how well the Markov Chain is mixing and how long the chain should be run for. However they should be used with caution (see **Assessing the Autocorrelation of Parameter Values**)

If multiple chains were run with Metropolis-coupling, then the next section summarizes the Beta values and observed swapping rates.

Following the autocorrelations and correlations there are listed results on differences between parameters. Over the course of the run all relevant pairwise comparisons between parameters of similar type are made, and a record is made of the proportion of the time that one is larger than the other. The result is a posterior probability estimate that can be used directly as a statistical assessment of whether or not one parameter is larger than another (Thanks to Robbie Young for suggesting this).

MARGINAL HISTOGRAMS

The primary results of the analysis are the records of the residence time for each parameter in each of 1000 evenly sized bins that span the prior distribution for each parameter. A table is first provided that summarizes features of the histogram for each parameter.

- Minbin - the midpoint value of the lowest bin.
- Maxbin - the midpoint value of the highest bin.

- HiPt - the value of the bin with the highest count (i.e. highest residence time).
- HiSmth - the value of the bin with the highest count, after the counts have been smoothed by taking a running average of 9 points centered on each bin.
- Mean - the mean value of the parameter.
- 95Lo - the estimated point to which 2.5% of the total area lies to the left.
- 95Hi - the estimated point to which 2.5% of the total areas lies to the right
- HPD90Lo - the lower bound of the estimated 90% highest posterior density (HPD) interval. The 90% HPD interval is the shortest span (on the X axis) that contains 90% of the posterior probability. A question mark, '?', is added if the HPD interval did not appear to be contiguous (in which case the HPD estimates are not reliable) which can happen with multiple peaks or if the surface is rough.
- HPD90Hi - the upper bound of the estimated 90% HPD interval.
- Tail? - this provides a rough guess of whether the posterior distribution appears to be complete ('complete') or if it is flat and non-zero at the upper end ('flat'), of if the curve is declining at the upper limit ('falling') or if it is actually rising at the upper limit ('rising').

Command Line Options – *not* the same as in earlier versions of the program

Executing the program with no flags, or with only '-h' will write the following text to the screen:

```
Command line usage: - upper or lower case letters can be used
-a  number of steps between record keeping (default is 10)
-b  duration of burn
    - if integer, the number of burnin steps
    - if floating point, the time in hours of burnin period
-bh ramped heating scheme for burn, -bh default is ~100,000 steps
-d  number of genealogy updates per step (default is 1)
-e  time (hours) between save of checkpoint file (e0 for only after burn)
-f  heat mode: l linear (default); t twostep; a adaptive twostep; g geometric
-g1 first heating parameter,effect depends on heating mode (default 0.05)
-g2 second heating parameter,effect depends on heating mode
-h  comment for output file (no spaces)
-i  input file name (no spaces)
-j  run options:
    0  likelihood() functions return 1 - posterior should equal prior
    1  for 4Nu priors (q1,q2,qa) use command-line values as actual piors
         default: priors = product of command-line values and data estimates
    2  treat inheritance scalars as parameters.(default= input file value or 1)
    3  include ranges on mutation rates as priors on mutation rate scalars
    4  set t to a very large value, mimics two population island model
    5  set t = 0 to mimic one large panmictic population of size thetaA
    6  set theta1 = theta2 = thetaA
    7  set m1 = m2
    8  each locus has a pair of migration rates, m1 and m2
    9  turn on population size change (include splitting parameter)
-k  with multiple chains (-n) the number of chain swap attempts per step
-l  duration of run
    - if integer, the number of steps in the chains
    - if floating point, the time in hours between outputs.
        run continues until file IMrun is no longer present
        in the directory, or if present, does not begin with 'y'
-m1 maximum migration rate from population 1 to population 2
-m2 maximum migration rate from population 2 to population 1
-n  number of chains
-o  output file name (no spaces)  default is 'outfile.txt'
-p  output options (default is no options):
    0 - print file of values of basic parameters (file can be enormous!)
    1 - print TMRCA histogram for each locus
    2 - print histogram of parameters on demographic scales
    3 – print histograms of # migration events and migration times
    4 - print ASCII-based trends of parameters
    5 - print ASCII-based posterior density curves
    6 - print smoothed distributions of marginal log likelihoods
-q1  scalar for theta1 maximum
-q2  scalar for theta2 maximum (default =  scalar for theta1 maximum)
-qA  scalar for thetaA maximum (default =  scalar for theta1 maximum)
-s  random number seed (default is taken from current time)
-sl  lower limit of range for population split parameter s
-su  upper limit of range for population split parameter s
-t  maximum time of population splitting
-u  generation time in years  (default is 1)
-w  minimum time of population splitting
-v  window width adjust for t updating - reduce window size w/ multiple loci
-x '*.dck' file name for starting from a previous run
-y  restart as if burn just ended, use with -x and '*.dck' file
-z  number of steps between output to the monitor (default 10000)
```

Explanation of Command Line Terms

`-a  number of steps between record keeping (default is 10)`

Because the parameter values over the course of the run are highly autocorrelated, there is no need to record values at every step. There is no harm in recording at every step, but recording at longer intervals saves a little time.

`-b  duration of burn`

The Markov chain begins with parameters and genealogies set to values that are consistent with the data but that may be very far removed from values that have nontrivial probabilities. To avoid having the initial conditions dominate the results, the chain should be run until it has reached a point that is independent of the starting conditions. The minimal length of a burn-in chain depends on the data set and cannot be known prior to looking at the results of some runs. In practice, burn-in lengths of 100,000 steps seem sufficient for most data sets unless the SMM model is used. If the value is an integer (e.g. '`-b100000`') the duration of the burn-in chain is the number of steps. If the value is a floating point number (e.g. '`-b1.0`') then a burn-in chain is run for that length of time, in hours.

`-bh ramped heating scheme for burn, default is ~100,000 steps`

This implements a burn with a set sequence of heating values, starting with lots of heating, and finishing with an extended time of no additional heating. The idea is to mix the chain(s) strongly at the start, with the hope that they will more quickly become independent of the starting point. If '`-bh`' is followed by an integer, x, then the duration of the burnin is $14x$ steps, with the first $9x$ of those steps taken with a sequence of preset heating values, and $5x$ steps taken with no heating (or with the final heating level specified for a chain, if multiple chains are used).

`-d  number of genealogy updates per step (default is 1)`

The updating of genealogies may occur at low rates, and there will usually be very strong correlations between genealogies and parameter values. Together these factors cause the overall rate of chain mixing to be slow, sometimes very slow. One way to overcome this is to update genealogies more often - multiple times per step rather than just once per step. Thus a value of '`-d10`' would lead to 10 update attempts for each locus for each step. This option can be useful at times. However the updating of genealogies is the slowest step in the entire process, so that a run with '`-d10`'

will take nearly 10 times as long as a run with '-d1' in which genealogies are updated at every step. Still this is a reasonable option for saving some time if things are mixing fairly well. If mixing is poor, then a better option is two run multiple chains with Metropolis-coupling. One can also do both multiple genealogy updates per step and Metropolis-coupling.

`-e  time (hours) between save of checkpoint file`

This causes a file to be saved (called a checkpoint file) that contains everything about the current state of the run. The name of the file is the output file name with the extension '.dck' added. This file is very large, and although it is a text file it is not really readable or useful as output. The file can be used to restart a run using the `-x` flag (see below). The first time the checkpoint file is saved is immediately after the burn-in is done. The file is also saved at time intervals (in hours) set by the number given with the '-e' flag (e.g. '`-e 1`' causes the file to be saved every hour after the burn-in is complete). A checkpoint file will also be saved at the very end of the run. If the run duration is set with '`-l`' using a floating point number (see below), then a checkpoint file will be saved every time that a results file is saved, as well as when the checkpoint interval is reached (if that is less than the run duration interval set by '`-l`'). If '`-e`' is followed by a zero, '0', then the checkpoint file will be saved after the burn-in is done and whenever a results file is written. This is probably the typical usage, unless one wants to have a checkpoint file saved more often than the results file. When the checkpoint file is saved, the previous file is renamed with the extension '.old' added to the end of it. This way there are usually the two most recent checkpoint files to use for restarting, if needed.

It is important to note that because the checkpoint file contains so much information, that the format of the checkpoint file changes with each update of the program, so that the latest version of the program cannot read checkpoint files generated with previous versions of the program.

`-f  heating mode`

Small single locus data sets often do not require heating, whereas multiple locus data sets or large data sets often do. STR data almost always require heating and metropolis-coupling. If multiple Markov-coupled chains are run, then it is important to have a heating

scheme that leads to sufficient rates of swapping of the chains. When there are n chains, they are numbered from 0 to n-1. For chain i, the Metropolis criteria for parameter updating are raised to a power β_i (beta(i)), where $\beta_i < 1$. For all heating schemes chain 0 is not heated and chains 1 thru n-1 have successively greater values of β. Swapping is done randomly among chains with similar heating values, but most accepted swaps are between chains with the smallest difference in heating values. Chains with low values of β will be strongly heated, but will have lower swapping rates with chains that are much less heated. In general, swapping rates for chain 0 that are less than 5% do not seem to be very helpful. For data sets where good mixing is achieved with a small number of chains, rates between 10% and 90% between chain 0 and chain 1, and generally between chains i and i+1, seem to be ideal. For difficult problems, 20 or 30 or more chains are sometimes needed. In these cases it is best to arrange for very slight heating and very high swap rates (> 90%) in the chains closest to chain 0. In these cases the geometric mode for selecting heating levels is recommended (see below).

Trial and error is required in order to find a useful heating scheme. A user may have to try several values, and wait for the program to run for awhile, to see their effect on chain swapping rates. Also, keep in mind that the rate of swapping among chains cannot always be assessed very well when a chain is just starting out, and sometimes one will not know the swapping rate until a chain has run for a million or more steps.

One difficulty with finding an adequate heating scheme is that the swapping rates between successive chains are often quite low, for low numbered chains (e.g. between chain 0 and chain 1), even for slight heating, whereas higher numbered chains often have higher swapping rates. The trick is to get sufficient swapping with chain 0, and yet also have sufficient heating in high numbered chains, so that the heated chains do really explore the parameter space.

Four different heating schemes have been designed: linear, twostep, twostep-adaptive, and geometric. The values used by these functions are given in the command line using '`-g1`', and '`-g2`'.

The linear scheme ('-fl') is the simplest and requires only that the user input a value for '-g1'. Under the linear scheme, each successive chain receives an additional heating increment. If the heating term is h, then the heating term β_i, for chain i, is given by

$$\beta_i = 1/(1 + i \times h)$$

For example, consider this simple linear scheme for five chains: `-fl -n 5 -g1 0.05`
For more than just a few chains, it is sometimes better to use the twostep scheme ('-ft') , which adds an additional multiplier increment of heating, for each chain i>1. This scheme requires two terms, h1 given by the '-g1' flag and h2 by the '-g2' flag. The formula for the heating term for chain i is

$$\beta_i = 1/(1 + h1 + h1 \times h2 \times (i - 1)$$

For example, this can be useful: `-ft -n 5 -g1 0.05 -g2 2`

The program also includes an adaptive twostep scheme ('-fa') which changes the heating parameters for the twostep scheme periodically so that the overall rate of swapping between low numbered chains is between 20% and 80%. Values are updated every 1000 steps during the burn-in period, and every 10000 steps during the main run. Changing the heating value during the run violates the Metropolis criterion for chain swapping, unless each time the heating rates are changed is also followed by an extensive burn-in period without swapping. For this reason the adaptive scheme should not be used for primary analyses, but rather for finding useful heating parameters to use under the two step ('-ft') model.

The geometric heating scheme ('-fg') is also intended to create successively greater heating levels for successively higher numbered chains. Two terms are required. h1, given by '-g1' on the command line, which specifies the degree of non-linearity, where 0<h1 (also usually h1 <1). The case of h1=1 gives a linear decline and h1<1 gives β values that decline slowly for low numbered chains, and fast for high numbered chains. The second term, h2 given by '-g2', is the value of β for the highest numbered chain. The formula is

$$\beta_i = 1 - \frac{(1 - h2) \times i \times h1^{n-1-i}}{n - 1}$$

To determine h1, on the basis of a particular heating value for chain 1, β_1, the following formula can be used

$$h1 = \frac{(1-\beta_1)(n-1)}{(1-h2)^{1/(n-2)}}$$

For example, with h2, the heating level of the highest numbered chain set to 0.6, and a desired level of heating for chain 1 of 0.999, then with 15 chains, h1 should be set to 0.773. This will create a series in which β drops very slowly among the lowest numbered chains, and more precipitously for the highest numbered chains. A useful starting point is

```
-f g  -n 6 -g1 0.8 -g2 0.9
```

Note that some data sets require large numbers of chains and very small β values for low numbered chains. In these cases it is best to play around with the geometric scheme.

`-g1 first heating parameter`

See the text for '`-f`'

`-g2 second heating parameter`

See the text for '`-f`'

`-h comment for output file (no spaces)`

This is if you want to add some explanatory text to appear in the output file. It is optional.

`-i input file name (no spaces within the name)`

For example, `-i mydatafile`

If no input file name is specified, the default is infile.txt. If no file of this name is found, the program stops.

`-j run options`

These are various program options to set the model

0. Causes all likelihood functions to return a value of 1. This makes the analysis not dependent on the data and, if the program is working properly should return the prior distributions in the resulting histograms. This is mostly used for debugging, but it can be useful for checking the priors on things that are not explicitly layed out. For example, when histograms are plotted for time and number of migration events, it is useful to do a run first with the –j1 option to see what the priors are for these distributions. Invoking this option causes the upper bounds on the prior distributions for the theta parameters to be set directly (see the '`-q`' flag).

1. Sets the manner for establishing the upper bound on the theta parameters. This option changes the way the -q1, -q2 and -qa command line terms are interpreted. By default the program uses an estimate of variation in each population and multiply this times the scalars given with –q1, -q2 and –qa. This is sometimes easier than doing preliminary runs and guessing about the range of these parameters. Using the '-j1' option causes the terms that are given by –q1, -q2 and –qa to be used as the actual maximum values for these parameters.
2. Treat inheritance scalars as model parameters. The default treats inheritance scalars as constants as given in the input file (or set to 1, if not specified in the input file). Setting '`-j2`' causes these terms to be treated as parameters that are sampled on a wide log scale centered at 1, much as are the mutation rate scalar parameters (Hey and Nielsen 2004). This option has no meaning when data includes a single locus, and is not likely to be informative unless there is a large amount of data on polymorphism within populations.
3. If mutation rate range priors are included on in the input file, for multiple mutation rates, then the ratios of these limits are used as limits on the ratios of the mutation rate scalars. If two or more loci in the analysis have a prior range, then this allows the prior information on mutation rates to be included in the analysis and to shape the findings. Great care must be taken in selecting a prior range, as it must include all of the sources of uncertainty in the mutation rate. One possibility is to use a likelihood approach. For example, consider a locus that is going to be used in the analysis, and suppose that there is information on the mutation rate for that locus based on a comparison between other species (not in the analysis). Let x be the amount of observed divergence observed between those species, and let τ be the number of years since those species are thought to have separated (in this example assume that τ is large and does not include a component of coalescent variation from the ancestor). Then develop the math for the probability of x, $p(x \mid \tau, r)$. Now decide what the prior range should be on τ (based on whatever information was used to get τ - this will depend very strongly on the source of τ). Then integrate $p(x \mid \tau, r)$ over the prior range of τ to get $p(x \mid r)$ (you can assume a uniform distribution of τ, or something else –

whatever makes the most sense), and define this quantity as the likelihood, $L(r \mid x)$. Now solve this expression for the most likely value of *r* (this will be the value of *r* you enter into the IM input file) and for the 95% confidence intervals (using standard likelihood assumptions) and use this interval as the prior range on *r* in the input file.

4. Set *t* to a very large value and causes *t* to not be updated. This makes the model a two-island equilibrium model.
5. Set $t = 0$. When invoked, the only parameter that is updated is θ_A. This is a single, constant size population model.
6. Causes $\theta_1 = \theta_2 = \theta_A$. In effect there is only one population mutation rate parameter.
7. Causes $m_1 = m_2$. The effect is for there to be only one migration rate parameter.
8. Causes each locus to have its own pair of migration rate parameters. This can complicate things a fair bit, and generates much for output with large numbers of loci, but does allow a user to try and see if different genes have had different amounts of gene flow.
9. Causes the model to include the splitting parameter *s*. This invokes exponential population size for descendant populations as described in Hey (2005). If this is invoked, *s* will be included in the regular histogram tables. Also, histograms will be generated for the product of $s \times \theta_A$ and $(1\text{-}s) \times \theta_A$. These are printed after any ASCII plots that are called for and after any TMRCA histograms that were called for.

`-k  with multiple chains (-n) the number of chain swap attempts per step`

With metropolis-coupling invoked and the number of chains greater than 2, this sets the number of attempts at chain swapping per step. With many chains this can even out the apparent rate of mixing for chain 0 and the autocorrelations. Since chain swapping proceeds fairly quickly, it is easy enough to have multiple swaps. For example for n chains, the number of swaps can also be n or more. The maximum value is n(n-1)/2.

`-l  duration of run`

If an integer (e.g. '`-l2500000`') this sets the length of the chain in steps. If a floating point value, this sets the duration of the chain in hours between printings of a results file. In this case, the program will run continuously, as long as the IMrun file is present

in the same directory. IMrun is a tiny file that contains the word "yes". Its only purpose is to be present when you want a run to continue, and to be absent when you don't. When using the '`-l`' flag with a floating point value, you control the run length by deciding when to remove the IMrun file from the directory in which the program is running. At each printing of the results file, the previous results file is renamed to *.old. By taking a look at the distributions in the output files, while the program is still running, one can run the program as long as desired. To halt the program at the end of the current interval, rename the IMrun file or edit it so that the first character is not a '`y`'. Be careful not to have the output file open (e.g. in an editor or a spreadsheet program) when it is time for the program to write to the output file, as the program will probably crash if this occurs. To view the output file, while the program is still running, it is usually best to rename it or to copy it to a new file.

`-m1 maximum migration rate from population 1 to population 2`

This sets the upper limit of the prior distribution of m_1 . To remove this parameter from the model, specify '`-m1 0`'. If the maximum migration rate is set to 0 for one migration parameter, it must also be set to zero for the other migration parameter.

`-m2 maximum migration rate from population 2 to population 1`

This sets the upper limit of the prior distribution of m_2 . To remove this parameter from the model, specify '`-m2 0`'.

`-n number of chains`

The number of chains to run under Metropolis-coupling. The default is 1 (i.e. no Metropolis-coupling) and the maximum is 10. For large numbers of chains, only chains with 7 steps of each other are considered for swapping.

`-o output file name (no spaces within the name)`

The default is 'outfile.txt' however it is usually best to provide a more explanatory name.

`-p output options:`

There are various output options. All can be given at once (e.g. `-p672` invokes options 2,6 and 7). Options that print to other files besides the main output file use the output file name as a base name and add an extension.

0. prints a file of the values of the six main demographic parameters at all times during the run (extension *.surf). This file can be read by other programs that

look at the joint likelihood surface. These files can be enormous for long runs (e.g. gigabytes) and this option should not be invoked unless there is lots of room and the user has need to combine results from different runs or to examining the joint likelihood surface. However it can be very useful if you need to combine runs that were done at different times or on different machines (assuming that the prior distributions and other settings – except random number seed - are the same for the different runs).

1. prints a table in the output file of the distributions of times of the most recent common ancestor (TMRCA) for each locus. The units on these times are the same as for the *t* parameter (i.e. mutation rate times time).
2. prints histograms for the theta and time parameters with the scales adjusted by the generation time and mutation rate, as given in the input file and runtime. If one or more of your loci have mutation rate estimates provided in the input file, then this option is very useful as it generates histograms on the relevant scales.
3. print histograms of # migration events and migration times
   This prints histograms that estimate the posterior distribution for numbers and times of migration events. Histograms are provided in each direction for each locus. The histograms for the time of migration rates have the same units as t and are based on just those genealogies that had at least one migration event. The times of all migration events are recorded (in contrast to earlier releases of this program in which mean migration times were recorded). For each locus the following columns are provided
   - m1#, the number of migration events into population 1
   - p, the estimate of the posterior density for m1#
   - m1time, the time of migration events into population 1 in the genealogy. The times of all migration events are recorded.
   - p, the estimate of the posterior density of m1time
   - m2#, the number of migration events into population 2
   - p, the estimate of the posterior density for m2#
   - m2time, the time of migration events into population 2 in the genealogy. The times of all migration events are recorded.
   - p, the estimate of the posterior density of m1time

4. print plots of the parameter histograms. These plots are ASCII-based and crude, but are useful for getting a rough idea of how things look and the relation of peaks to the bounds of the prior distribution.
5. print plots of parameter value trends. These plots are ASCII-based and crude, but are still quite useful for assessing how well the parameters are mixing. A plot is also provided for Log(P) = Log(P(Data|Genealogy)) + Log(P(Genealogy|Parameters)), which is useful for an overall sense of how well the chain is mixing.
6. prints a table in the output file of smoothed estimated densities of marginal log likelihoods

`-q1 scalar for theta1 maximum`

The upper bound of the population mutation parameter for population 1. If the '`-j1`' option is used, the program generates a rough estimate of the population mutation parameters In this case this command line flag sets a scalar to be multiplied by that value for population 1.

`-q2 scalar for theta2 maximum`

As for '`-q1`'. If no value is specified for pop 2, then the value for '`-q1`' is used.

`-qA scalar for thetaA maximum`

As for '`-q1`'. If no value is specified, then the value for '`-q1`' is used.

`-s random number seed (default is taken from current time)`

An integer (e.g. '`-s 123`'). Useful in case of the need to exactly repeat a run or when starting multiple instances of the program at about the same time and you don't want the program to use the same machine time to seed different runs of the program.

`-sl lower limit of range for s`

For use with the '`–j7`'option (including the population splitting parameter, *s*). This flag is used to impose a lower bound on the prior distribution for *s*. For example, if a descendant population 1 is assumed to have been founded by a majority of the ancestral population, then *s* should not be less than 0.5 (i.e. '`-sl 0.5`').

`-su upper limit of range for s`

For use with the '`–j7`'option (including the population splitting parameter, *s*). This flag is used to impose an upper bound on the prior distribution for *s*. For example, if

descendant population 1 is assumed to have been founded by a minority of the ancestral population, then *s* should not be greater than 0.5 (i.e.'`-su 0.5`').

`-t maximum time of population splitting`

The upper bound on the prior distribution for *t*.

`-w minimum time of population splitting`

The lower bound of the time of population splitting (default is 0)

`-u generation time in years (default is 1)`

Specify the generation time. This is only needed if you want the population size parameters rescaled and you have used print flag '`-p4`'. This also requires that at least one locus have a mutation rate given in the input file.

`-v window width setting for t updating`

Adjust the window width for updating of *t* . Do not confuse this with the upper bound of the prior distribution for *t.* The window width only concerns the way random values are picked for the updating, and this option will only rarely be used. With multiple loci it can be necessary to reduce the window width in order to have reasonable update rates. The formula for the width is (t_max - t_min)/(5 + v × #loci), v is the value read in using this command line flag (default is 0). Sometimes the default level of mixing for *t* is too low and it can help to reduce the window size.

`-x '*.dck' file name for starting from a previous run`

If a checkpoint file has been saved by using the '`-e`' flag, then the '`-x`' flag can be used to start a run that begins at the point that the checkpoint file was written. This can be very useful if an ongoing run had to be stopped (or crashed) for some reason. The new run will begin with _all_ of the same parameters used in the run that created the checkpoint file (including the '`-e`' setting that will cause the continued writing of checkpoint files). With the exception of '`-y`', no other command line flags should be used with '`-x`'.

`-y restart as if burn just ended, with -x and '*.dck' file`

This will cause a run to start using a previously saved checkpoint file, however it will begin without any saved parameter values and it will assume that the burn-in period has just been completed. This can be very useful if a long run has been done, only to find that the burn-in was not sufficient. For example, a run may be done with a burn-in of 1,000,000 steps, only to show in the trend line plots of the output that that was

not sufficient. In such a case the user can continue to monitor the trend lines in subsequently written output files, until such time as it appears that stationarity has been reached. Then stopping the run and restarting with the last saved checkpoint file and the '`-y`' flag will begin a new run at that point, as if the burn had just been completed.

`-z the number of steps between output to the monitor.`

The default is 10,000, but if the program is quite slow for your data, and you want to look at things immediately, it is useful to set it to a small number (e.g. 100 or 1000).

Suggested Starting Point

To get things started you might try typing the following line at the command prompt and hitting enter:

```
im -i mydata -o mydata.out  -q1 10 -m1 10 -m2 10 -t 10 -b
100000 -L 0.5  -s123  -p45 –e0
```

This will cause the program to do a burn-in of 100000 steps followed by a ½ hour run. After the burn a checkpoint file will be saved. After the burn and the ½ hour run a results file will be written (mydata.out) and the checkpoint file will be updated. The program will either stop (if the IMrun file is not present) or if the IMrun file is present the program will keep doing ½ hour runs – updating the output and checkpoint files after each. The program stops after completing a ½ hour interval and there is no IMrun file present in the folder where the program is running. At each ½ hour interval the previous results file and the previous checkpoint file are renamed to *.old and new results and checkpoint files are written.

This example will cause simple curves to be printed (`-p4`) near the end of the output file. However to properly look at curves the output file should be opened in a spreadsheet program and the histograms selected and plotted. A spreadsheet (IMchart.xls)is provided in the distribution package.

This example also results in plots of parameter trend lines (`-p5`) at the end of the output file. These are useful for assessing how well the Markov chain is exploring the parameter space.

Cautions, Suggestions and Interpretations

One of the greatest challenges is knowing whether the chain is mixing sufficiently. It is possible to have seemingly high update acceptance rates for the parameters, and yet still have a slow rate of mixing. In cases like these, it is likely that a low rate of genealogy updating is the culprit. Check the rate of genealogy updating, and especially the rate of branching pattern updating (column identified with 'B%' in output file). For runs of a couple million or more steps, the ESS estimates can be taken as a rough guide to how many effectively independent observations have been made for each parameter. Be aware that, because the parameters are not independent, the lowest ESS is the critical one. It is quite possible after a long run to have an ESS value of 10 or less for the time parameter, and ESS values greater than 1000 for the other parameters.

A minimum of three runs will be needed to have a rough idea of how well the program is working and of what the marginal distributions look like. The first run is required to assess mixing and to find a range of prior distributions that include all or most of the range over which the posterior density is non-trivial. If things are well behaved, a run using something like '`-q110 -m120 -m220 -t20`' will generate curves that rise from zero, peak, and then fall to zero within the range for each of the six or seven basic demographic parameters (this is just an example - you may need different prior values). If it looks like the posterior distributions are fully contained within the bounds of the prior distributions, and if the observed maxima for any distributions are far to the left of the upper bounds, then the parameter maxima can be reduced for subsequent runs. For the second and third runs the priors can be adjusted as a function of what was observed in the first run. Both of these runs should be long and started using identical settings but different random number seeds. You will have a pretty good idea that the chains are sufficiently long if all ESS values are high and if both runs have generated similar distributions.

In cases where the posterior distributions are sampled well, and where the estimated probabilities at the margins of the prior distribution approach zero (i.e. the histogram drops to zero at the tails), the parameter estimates (based on the locations of the peaks) are also maximum-likelihood estimates. The reasons are that the prior distributions are flat (i.e. all values in the range are equally probable) and because we expect in these cases that

extending the prior of the parameters to large values, even to infinity, would not change the posterior distributions.

With population size change (`-j9`) it is often the case that there is not enough information in the data to provide for discrete posterior distributions for all parameters. In these cases one or more parameters may show estimated posterior distributions that are flat or that increase steadily for increasing parameter values without revealing a peak. It is also not uncommon to have two peaks for *s* (one low and one high), corresponding to alternative similarly likely histories. In such cases it can be very helpful if an upper or lower bound on the prior for *s* is justifiable (`-su, -sl`).

If the data set is small, and sometimes even if not, it can happen that a large portion of the likelihood surface is very flat over the range of some parameters. In particular it is not uncommon to have high, flat likelihoods for high values of *t* and θ_A. One may find for example in the curve for *t* a sharp peak at a low value, and then a plateau that extends indefinitely to the right at an even higher value. This is awkward because the highest likelihood appears to be associated with an infinitely wide range of parameter values. In these situations, the data does not contain enough information to identify the model, and the results should not be relied upon.

It is tempting in situations with multiple peaks, or where the curve shows a distinct peak at a low value, and a plateau at ever increasing high values, to do more runs with a lower upper bound on the prior for that parameter so as to exclude the area to the right of the distinct peak. An example of this is shown, for the *t* parameter, in Hey (2005). This kind of changing of the prior, to get more readily interpretable plots, is a reasonable thing to do if the altered prior range makes sense given other information (outside of the data). However, in the absence of other information upon which to constrain the prior, then changing it simply because the plots don't give a clear picture, cannot really be justified.

Five things to keep in mind:

1. You may not have enough data to estimate so many parameters (you can try a simpler model with a reduced parameter set).

2. It is critically important that you assess convergence by doing multiple long runs, and by monitoring ESS values and trendlines. If you don't, then your results are probably wrong, possibly very wrong.
3. The algorithm is technically capable of providing information about the joint likelihood surface. However estimation of points in a parameter space with many dimensions (one for each parameter) takes far too long. The marginal distributions are derived from this surface and the peaks of these need not be in the same place as the peak of the joint distribution.
4. The prior distributions can indeed be used to reflect your prior expectations. If you think that values of t (or any parameter) greater than a certain value should not be open for consideration, then set the maximum of accordingly.
5. Finally - everything depends on the data. Your data may present new challenges, and provide new insights, that have not been anticipated. They may also require runtimes of days, weeks, or quite possibly much worse, in order to achieve convergence.

Compilation

A Windows executable is provided (IM.exe). This has been compiled under Microsoft Visual C++ 6.0. The source code is available if you need to change program constants or need to compile for another computer system. In particular, large data sets or data that requires extensive Metropolis-coupling may require that MAXLOCI and MAXCHAINS be increased in the im.h header file.

Runs with large data sets (e.g. more than 10 loci, with more than a hundred gene copies each) and large numbers of metropolis coupled chains (e.g. 50 chains) can use a lot of memory. This problem is much alleviated in the version released in July 2006, but it can still be an issue.

Compiling for unix/linux has been a bit of a problem in the past. Hopefully in these recent updates those bugs are gone and the code will compile and run appropriately. When using a non-Windows compiler and OS it is important to make sure that the input file has the appropriate end-of-line characters for that OS. Failure to do this will probably cause a cryptic crash.

For some unix systems that I have used, the following command line generates an executable called *im* which runs:

```
cc chain.c check.c datain.c front.c out.c pc.c stepwise.c
util.c  -lm  -o im
```


References

Griffiths, R. C., and S. Tavaré. 1994. Sampling theory for neutral alleles in a varying environment. Philosophical Transactions of the Royal Society of London, Series B 344:403-410.

Hey, J. 2005. On the Number of New World Founders: A Population Genetic Portrait of the Peopling of the Americas. PLoS Biol 3:0965-0975.

Hey, J., and R. Nielsen. 2004. Multilocus methods for estimating population sizes, migration rates and divergence time, with applications to the divergence of *Drosophila pseudoobscura* and *D. persimilis*. Genetics 167:747-760.

Won, Y. J., and J. Hey. 2005. Divergence population genetics of chimpanzees. Mol Biol Evol 22:297-307.

Appendix

```
miscellaneous errors
1  cannot open data file
3  cannot open surface file
6  cannot create treeprint file
7 cannot create output file
8  input and output file names are identical
9  cannot write checkpoint file
10 cannot open checkpoint file
11 equilibrium migration model, but migration set to 0
13 one migration prior set to zero  (if one is set to 0, must have both)
14  -p1 option not permited when each locus has its own migration rate
      parameters
18  too few chains for heating model
19 not enough information on command line
21 problem w/ number of loci indicated in data file
22  either Qmax.q1 or Qmax.q2 set to 0, but sample size for that population is
      not zero

30 - to much migration called for on a single branch of a tree
31  - roottime > TIMEMAX after makecoal
```

```
32  - roottime > TIMEMAX  in changet
34  - too much migration in startnummig()
35  - #'s of gene copies and migration events out of synch when sorting tree
      events in geteventinfo

41 error reading data
42 error in data
43 variable name mistmatch when reading checkpoint file
44 variable type mismatch when reading checkpoint file
45 checkpoint file versions do not match

51 model not recognized in changeq
55  NR vector allocation error
56  NR NSTACK too small in indexx
76 - product of uscalers != 1;
77 - could not solve for uscaler priors - ranges not compatible
79 - problem with mutation rate ranges

80 - formatting of input file causes wrong lines to be read as data
81  data not compatible with infinite sites model
82  data not compatible with infinite sites model
83  data not compatible with infinite sites model
84 - sample sizes do not have add up for one locus
91  likelihoods do not add up for stepwise model
93  problem generating hpd intervals
```