(p)IPHULA - (parallel) Inference of Population History Using a Likelihood Approach

(p)IPHULA Manual
Version 1.16 (May 2007)

Heiko A. Schmidt
Center for Integrative Bioinformatics Vienna (CIBIV),
Max F. Perutz Laboratories (MFPL),
Dr. Bohr-Gasse 9, A-1030 Vienna, Austria.
email: `heiko.schmidt @ univie.ac.at`

Arndt von Haeseler
Center for Integrative Bioinformatics Vienna (CIBIV),
Max F. Perutz Laboratories (MFPL),
Dr. Bohr-Gasse 9, A-1030 Vienna, Austria.
email: `arndt.von.haeseler @ univie.ac.at`

Jutta Buschbom
Institute of Forest Genetics and Forest Plant Breeding,
Federal Research Centre for Forestry and Forest Products,
Sieker Landstr. 2, D-22927 Großhansdorf, Germany.
email: `j.buschbom @ holz.uni-hamburg.de`

General Information

(p)IPHULA is a software that infers parameters of population history in a maximum likelihood framework using Monte Carlo simulation.

(p)IPHULA is written in ANSI/ISO C. It will run on most personal computers and workstations if compiled by an appropriate C compiler. IPHULA has been parallelized (pIPHULA) using the Message Passing Interface standard (MPI, Snir *et al.*, 1998; Gropp *et al.*, 1998). For parallel execution of pIPHULA it is necessary to have an implementation of MPI installed on your system. Please read the *Installation* section (3) for more details.

We suggest that this documentation should be read before using (p)IPHULA the first time.

(p)IPHULA is pronounced ['i:fula:] (ee-foo-lah) or ['pi:fula:] (pea-foo-lah), depending on whether one is speaking about the sequential (IPHULA) or the parallel version (pIPHULA), respectively.

To find out what's new in the current version please read the *Version History* (Section 10).

Contents

1 Legal Stuff

2 Introduction

Weiss and von Haeseler (1998) introduced a three-parameter model (θ, τ, ρ), the WvH98 model, of population size change to model population history and infer demographic parameter values based on a given set of aligned DNA-sequences.

WvH98 models an event of exponential population size change in a Wright-Fischer population (i.e., a population with randomly mating individuals in non-overlapping generations, in which individuals of the next generation are drawn from a large gene pool and no selection occurs). It takes into account the population size $\theta = 4N_0\mu$ before the growth or decline event started, with N_0 denoting the initial effective population size and μ the mutation rate per gene and generation. The starting point τ of the population size change event is considered in units of $\frac{1}{\mu}$ in the past and the population growth factor ρ represents the ratio between the current and the initial population size at time τ in the past (i.e., $\rho > 1$ growth, $\rho = 1$ constant size, and $\rho < 1$ decline, cf. Fig. 2.1).

Population size changes produce typical distributions of neutral genetic variation in populations that provide information on the three parameters of the model. Inference is based on the mean pairwise difference K and the number of polymorphisms S observed in DNA sequence data. A maximum likelihood approach is applied to estimate the model parameters based on K and S. Since there is no closed form available to estimate these model parameters directly from the data, Weiss and von Haeseler (1998) have suggested a Monte Carlo approach to obtain maximum likelihood estimates of the parameters. This approach approximates the likelihood surface for the three relevant population parameters by coalescent simulations on a 3D-grid. Each simulation is conducted in two steps: first, a genealogy is produced. Its coalescent times are determined according to

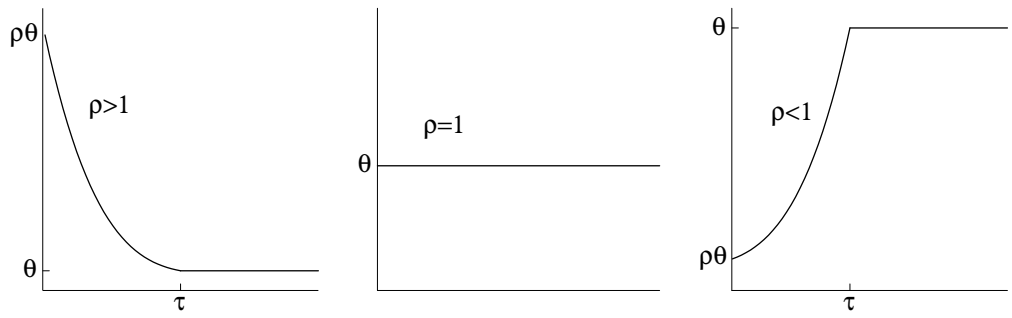

Figure 2.1: Relationship of the population growth parameters ρ, θ, and τ in cases of exponential growth (left), stable size (middle), and decline (right). Figure adopted from Weiss and von Haeseler (1998).

the coalescent approximation for populations with deterministically varying population sizes (Griffiths and Tavaré, 1994). At each coalescent time two lineages are randomly merged, resulting in a genealogy for the given number of individuals. In a second step a sample of sequences is generated by evolving an ancestral sequence along the genealogy. Here, the mutation process runs irrespective of the current genealogy and is determined by the nucleotide substitution model parameters set by the user. The, thus, simulated DNA sequence data sets are analyzed with regard to the K- and S-statistics. The frequency with which the simulated data sets produce mean pairwise distances (K) close to and numbers of polymorphisms (S) identical to those found in the original data set approximates the likelihood $\mathrm{lik}(\theta, \tau, \rho | K, S)$.

3 Installation

The source code of the (p)IPHULA software is 100% identical across platforms. However, installation procedures differ. There is a source distribution (`piphula-X.XX.tar.gz`, `piphula-X.XX.zip`) and binary distributions (`piphula-X.XX-linux.tar.gz`, `piphula-X.XX-macosx.tar.gz`, `piphula-X.XX-windows.zip`) with executables for Linux, Mac OS X, and Windows respectively. (`X.XX` has to be substituted by the current version number.)

3.1 UNIX/Source Distribution

Get the file `piphula-X.XX.tar.gz` (`.zip` file for Windows). Decompress it first (using the `gunzip` command in a Terminal or another program capable of handling `gz` format) and then untar the file with

```
gunzip piphula-X.XX.tar.gz
tar xvf piphula-X.XX.tar
```

The newly created directory `piphula-X.XX` contains five subdirectories called `doc`, `data`, `R-scripts`, `bin`, and `src`. The `doc` directory contains this manual in PDF format. The `data` directory provides example input files. The `src` directory contains the ANSI/ISO C sources of (p)IPHULA. An R-script for plotting results is found in the folder `R-scripts`. In a Terminal (e.g., `xterm` on Unix/Linux, `Terminal.app` on MacOS X, or `Command prompt` on Windows) switch to this directory by typing

```
cd piphula-X.XX
```

To compile we recommend the GNU gcc compiler (included in most Linux distributions, the free Xcode package for Mac OS X or the free `cygwin` or `minGW` package for Windows). If gcc is installed just type

```
sh ./configure
make
make install
```

and the executable `iphula` is compiled and put into the `/usr/local/bin` directory. You might need administrator rights to run the `make install` step. Alternatively, you can copy the executable to another directory in your search path that you can access.

If you want to have `iphula` installed automatically into another directory you can set this by setting the `--prefix=/name/of/the/wanted/directory` directive at the `sh ./configure` command line.

The parallel version should have been built and installed as well, if `configure` found a known MPI compiler/installation (cf. 3.3.1 MPI-Parallel (p)IPHULA).

Then type

```
make clean
```

and everything will be nicely cleaned up.

If your compiler is not the GNU gcc compiler and not found by `configure` you will have to modify that, by setting the `CC` variable (e.g. `setenv CC cc` under `csh` or `CC=cc; export CC` under `sh`/`bash`) before running `sh ./configure`. If you still cannot compile properly then your compiler or its runtime library is most probably not ANSI compliant (e.g., very old SUN compilers). In most cases, however, you will succeed to compile by changing some parameters in the `makefile`. Ask your local Unix expert/system administrator for help.

3.2 Binary Distributions

3.2.1 Linux

Get the file `piphula-X.XX-linux.tar.gz`. After extracting this file as described in section 3.1 (if you have problems, please ask your local Linux expert), you will find a folder called `piphula-X.XX` on your hard disk. This folder contains the subfolders `doc`, `data`, `R-scripts`, and `src`. The `doc` folder contains this manual in PDF format. The `data` folder contains example input files. The folder `R-scripts` contains the R-script for plotting results. The precompiled executable as well as the ANSI/ISO C sources of (p)IPHULA are found in the folder `src`.

Rename the Linux executable `iphula-linux-gcc-static` to `iphula` and copy it to a folder in your `PATH`, e.g., `/usr/local/bin` is generally a good choice.

The Linux executable has been compiled using the GNU C compiler (`http://gcc.gnu.org`). If you need a compiler to prepare the executable yourself, you might download one from the list below (section 3.4). Then proceed as described in the UNIX/Source Installation section (3.1).

3.2.2 Mac OS X

Get the file `piphula-X.XX-macosx.tar.gz`. After extracting this file as described in section 3.1 (if you have problems, please ask your local Mac OS X expert), you will find a folder called `piphula-X.XX` on your hard disk. This folder contains the subfolders `doc`, `data`, `R-scripts`, and `src`. The `doc` folder contains this manual in PDF format. The `data` folder provides example input files. The folder `R-scripts` contains the R-script

for plotting results. The precompiled execultable as well as the ANSI/ISO C sources of (p)IPHULA are found in the folder `src`.

Rename the Mac OS X executable `iphula-macosx-Xcode` to `iphula` and copy it to a folder in your `PATH`, e.g., `/usr/bin` or `/sw/bin` are generally good choices.

The Mac OS X executable has been compiled using Apple's Xcode Tools (`http://developer.apple.com/macosx/`). If you need a compiler to prepare the executable yourself, you might download one from the list below (section 3.4). Then proceed as described in the *UNIX/Source Installation* section (3.1).

3.2.3 Older Mac OSes

Due to missing access we do not support Mac OS 9/Classic anymore. We recommend changing to Mac OS X, which combines the advantages of UNIX systems with the Mac OS's graphical interface.

If you want to prepare an executable for your Mac OS yourself, you might get Metrowerks' CodeWarrior (refer to section 3.4) to compile the C sources included in all (p)IPHULA distribution packages.

3.2.4 Windows 95/98/NT/etc.

Get the file `piphula-X.XX-windows.zip`. After un-zipping this file (if you have problems, please ask your local Windows expert), you will find a folder called `piphula-X.XX` on your hard disk. This folder contains the subfolders `doc`, `data`, `R-scripts`, and `src`. The `doc` folder contains this manual in PDF format. The `data` folder provides example input files. The folder `R-scripts` contains the R-script for plotting results. The precompiled execultable as well as the ANSI/ISO C sources of (p)IPHULA are found in the folder `src`.

Rename the Windows executable `iphula-windows-mingw` to `iphula` and copy it to a folder in Windows' search path.

The Windows executable has been compiled using MinGW (`http://www.mingw.org`). If you need a compiler to prepare the executable yourself, you might download one from the list below (section 3.4). Then proceed as described in the UNIX/Source Installation section (3.1).

If you got a Linux partition on your PC we recommend to install and use (p)IPHULA under Linux (see section 3.2.1), since (p)IPHULA will run faster under Linux than under Windows.

3.3 Compilation for Parallel Implementation

3.3.1 MPI-Parallel (p)IPHULA

To compile and run the parallelized IPHULA you need to have an implementation of the Message Passing Interface (MPI) library, a widely used message passing library standard,

installed on your system. Implementations of the MPI libraries are available for almost all parallel platforms and computer systems, and there are free implementations for most platforms as well.

To find an MPI implementation suitable for your platform visit the following web sites:

- `http://en.wikipedia.org/wiki/Message_Passing_Interface`
- `http://www.lam-mpi.org/mpi/implementations/`
- `http://www-unix.mcs.anl.gov/mpi/implementations.html`
- `http://WWW.ERC.MsState.Edu/labs/hpcl/projects/mpi/implementations.html`

Although MPI is also available on Macintosh and Windows systems, the developers never ran the parallel version on these platforms.

To install the parallel version of (p)IPHULA you need the source distribution for (p)IPHULA and install the package on your computer as described above (3.1). The `configure` should configure the Makefiles appropriately. If there is no known MPI compiler found on the system the parallel version is not configured. (If problems occur ask your local system administrator for help.)

Now you should be able to compile the parallel version of (p)IPHULA using the following commands:

```
sh ./configure
make
make install
```

and the executable `mpiiphula` is compiled and put into the `/usr/local/bin` directory. If you want to have the executable installed into another directory please proceed as described in section 3.1.

If your compiler is none out of `mpcc` (IBM), `hcc` (LAM), `mpicc_lam` (LAM under LINUX), `mpicc_mpich` (MPICH under LINUX), and `mpicc` (LAM, MPICH, HP-UX, etc.) and not found by `configure` you will have to modify that by setting the `MPICC` variable (e.g. `setenv MPICC /another/mpicc` under `csh` or `MPICC=/another/mpicc; export MPICC` under `sh`) before running `sh ./configure`.

Some compilers (e.g., IBM) have problems to compile the SPRNG random number generator source code. In this case you can use the old 'leapfrog generator' by setting the `CFLAGS` variable to `-DNOSPRNG` (e.g., `setenv CFLAGS '-DNOSPRNG'` under `csh` or `CFLAGS='-DNOSPRNG'; export CFLAGS` under `sh`/`bash`) before running `sh ./configure`.

The way to start `mpiiphula` depends on the MPI implementation installed. So please refer to your MPI manual or ask your local MPI expert for help.

Since the master process does not require a lot of CPU time, it can be scheduled sharing one processor with a worker process. Thus, you can run `mpiiphula` by assigning $n + 1$ processes.

3.3.2 OpenMP-Parallel (p)IPHULA

OpenMP (Dagum and Menon, 1998) is a different standard to parallelize software. It is, however, restricted to multi-processor machines since it is based on threads running on the same machine. Furthermore, the parallelization itself is done by OpenMP-aware compilers, and hence dependent on their performance.

The (p)IPHULA source code contains all compiler directives needed to produce an OpenMP-based thread-parallel executable by compiling it with an OpenMP compiler.

Testing the performance of OpenMP-mediated parallelism showed amazingly bad performance on an IBM Regatta system, even for high numbers of independent tasks on only 4 CPUs. Unfortunately, this behavior had to be attributed to the performance of the OpenMP compiler.

Since the MPI implementation by far outperformed the OpenMP version, the developers decided not to maintain the OpenMP approach. The compiler directives will remain in the code and should be usable in the future.

A hybrid use of OpenMP and MPI parallelism simultaneously in pIPHULA is not available.

3.4 ANSI/ISO C Compilers

If there is no binary version of (p)IPHULA available for your computer or if you want to compile (p)IPHULA yourself, an ANSI/ISO C compiler is needed to produce an executable suitable for your machine. There are a number of free compilers available for the different operating systems:

GCC - Gnu Compiler Collection (UNIX/Linux, Mac OS X, Windows), available for most OSes, widely spread in the UNIX/Linux community where it is included in virtually every distribution. GCC is also the basis to most compilers mentioned below. (`http://gcc.gnu.org`)

ICC - Intel C++ Compiler (Linux, Windows), ICC has shown good performance. It is free to a certain extent - please refer to the license. (`http://www.intel.com/software/products/compilers/`)

MinGW - Minimalist GNU for Windows (Windows), a GCC based package that provides all basic tools necessary for software development with Gnu tools under Windows (`http://www.mingw.org`)

CygWin - GNU+Cygnus+Windows (Windows), a GCC based package that provides the (almost) full Gnu environment and tools known from UNIX/Linux systems (including X-Windows). Unfortunately, executables compiled with CygWin need CygWin installed to be run on other Windows computers. (`http://www.cygwin.com`)

Xcode Tools (Mac OS X), the GCC based software development tools from Apple for Mac OS X. It is free, but you have to register. Please check the license. (`http://developer.apple.com/macosx/`)

CodeWarrior (Mac OS Classic/9/X, Windows), this is a commercial development environment from Metrowerks. Although not supported by the (p)IPHULA developers, CodeWarrior is able to produce executables for older Mac OSes. (`http://www.metrowerks.com/MW/Develop/CodeWarrior.htm`)

For MPI (message passing interface) libraries which are needed to build the parallel version of (p)IPHULA refer to the *Parallel (p)IPHULA* section (3.3.1).

4 Usage

4.1 Program Execution

In a terminal application (e.g., `xterm` on Unix/Linux, `Terminal.app` on Mac OS X, or the `Command Console` on Windows) the non-parallel version of (p)IPHULA is called by typing `iphula` on the line prompt. The program starts by providing its version number and general information. It immediately enters the first submenu for setting the simulation parameters.

```
******   IPHULA 1.16 (sequential binary)   ******

This program makes inference of population history
parameters based on the likelihood of a parameter set
given the number of variable positions in the sample
and the mean pairwise distance.
For more details and reference issues see:
G.Weiss & A.v.Haeseler (1998) Genetics:149, pp.1539-1546

Current parameters of the substitution process:
-----------------------------------------------

base frequency of A  : 0.327
                  C  : 0.338
                  G  : 0.114
                  T  : 0.221
ts/tv ratio kappa    : 15.5
pyr/pur ts ratio xi  : 1.85
rate heterogeity     : no

Do you want to accept these values for the substitution process?
type 'y' to accept, 'n' to change, or 'q' to abort:
```

4.2 Input/Output Conventions

Extension	file content
`.iphula`	report file including results
`.matrix`	output of the θ - τ matrix for each ρ value
`.params`	output of run parameter settings (for associated R-script)

The file types are described in detail below. In the following "`FILEPREFIX`" denotes the prefix, which is either the default name (`panel`), the name of the input parameter file given with the `-infile` option, or explicitely set with the `-prefix` option.

4.2.1 Program input

The parameter settings for the simulation run are generally entered manually following the prompts of the menu, no input file is needed. In batch mode the settings can be entered by piping a text file with the parameter values (the same way you otherwise would enter them manually) into the program. Such file can also be given to the program using the `-infile` commandline option (see section 5.3).

4.2.2 Report file

The report file of pIPHULA (`FILEPREFIX.iphula`) logs information on the version number of the program used, date and time of the start of the run as well as the duration of the run, a summary of the simulation settings and the matrix of the simulation results.

4.2.3 Matrix output

In the θ/τ matrix output files of pIPHULA (`FILEPREFIX_RHO.matrix`) the simulation results for each ρ value (substituting the `RHO` term in the file name) can be found. The values of τ differ over the columns, and for θ along the rows.

	τ_{min}	...	τ_{max}
θ_{min}			
.			
.			
.			
θ_{max}			

The first column and row contain the different values of θ and τ respectively, thus, leaving the upper left field empty. All other fields contain the frequencies of successful simulation runs. The columns are separated by Tabs.

The matrices are in a format that can be read by the R routine distributed with pIPHULA for graphical presentation of the results (see section 6.1).

4.2.4 Parameter output

The parameter output file (`FILEPREFIX.params`) summarizes the parameter values of ρ, θ and τ that define the simulation grid, as well as the number of simulation repeats. This output file is needed by the R routine for graphical presentation of the results provided with pIPHULA (see section 6.1).

4.3 Simulation Settings

A (p)IPHULA run is configured through a couple of successive menus. First, one can change the *parameters of the substitution process*: base frequencies, transition:transversion ratio κ, the ratio ξ between pyrimidine and purine transitions, and the numbers of categories and the shape parameter α for a discrete Γ model. By default the Jukes-Cantor model is implemented with all base frequencies being equal, equal transition and transversion rates (i.e., a ts/tv ratio of 0.5), the pyrimidine:purine transition rate ratio set to 1 and rate homogeneity (At the moment rate heterogeneity is not implemented yet. Thus, only $\alpha = 1$ is possible.)

```
Default parameters of the substitution process:
-----------------------------------------------

base frequency of A  : 0.250
                  C  : 0.250
                  G  : 0.250
                  T  : 0.250
ts/tv ratio kappa    : 0.5
pyr/pur ts ratio xi  : 1.00
rate heterogeneity   : no

Do you want to accept these values for the substitution process?
type 'y' to accept, 'n' to change, or 'q' to abort:
```

Next, the dimensions and characteristics of the aligned data set to be simulated need to be defined, that is, the number of sequences and their lengths. In this step of the menu also the observed variability in the original data set as reflected by the summary statistics S (the number of variable positions in the data set) and K (the mean pairwise distance over all sequences) needs to be entered. K is calculated as the average number of differences between two sequences in the data set.

Subsequently, the ranges and step sizes of the parameters of population history used in the simulation have to be set. These values define the position, size and mesh width of the grid in the parameter space. These parameters are the original equilibrium population size θ ($= k \times$ *population size* $\times$ *mutation rate*) before the size change event with $k = 2$ for haploid organisms or organelles and $k = 4$ for diploid organisms, the point in time τ at which the population started to change in size (i.e., the time span that passed

since the event, scaled by the mutation rate), and ρ calculated as the ratio of the extant population size to the population size before the population size change event. Note, that if $\rho = 1$ (i.e., constant population size, no change), τ remains undefined.

```
Population history parameter:
------------------------------

The simulations will run on a grid of RHO, THETA and TAU having
- RHO   with   8 grid points:
         0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000
- THETA with  10 grid points from   1.0 to  10.0
- TAU   with  10 grid points from   0.5 to   5.0

Simulate with these grid parameters?
Type 'y' to accept, 'n' to change, or 'q' to abort:  y
```

The parameters can be either changed or the defaults can be selected.

Finally, the number of simulations per grid point (i.e., per set of parameter values) is chosen and the tolerance limit δ defined. If the difference between the observed value of K and the simulation result of K is smaller than δ, the simulation result is considered to be identical to the input value and counted as hit.

Finally all values have to be accepted with `y` to start the simulation or the parameters might be corrected at this point.

(p)IPHULA indicates its progess on the screen, compare

```
Simulating grid position: Rho=0.001000 Theta=1.000000 Tau=0.500000
Simulating grid position: Rho=0.001000 Theta=1.000000 Tau=1.000000
Simulating grid position: Rho=0.001000 Theta=1.000000 Tau=1.500000
...
```

until it finishes. A report of the run can then be found in the `FILEPREFIX.iphula`, output of the results matrices in the files `FILEPREFIX_RHO.matrix` (with RHO being replaced by the different ρ values used in the simulation) and a summary of the parameter settings in `FILEPREFIX.params`.

4.4 Running pIPHULA in Parallel

For those who want to run pIPHULA in parallel, e.g., on a cluster we will examplify briefly how this is done for the MPI implementations LAM-MPI (`http://www.lam-mpi.org`) and MPICH (`http://www-unix.mcs.anl.gov/mpi/mpich`). There are many (free) MPI implementations (refer to `http://en.wikipedia.org/wiki/Message_Passing_Interface`) that all differ to some extent. Hence, we recommend that you contact your local system administrator and read the respective manual carefully.

The following assumes that you are already logged into a cluster node or parallel computer on with you want to run your parallel IPHULA analysis.

4.4.1 Using LAM-MPI

In the following the use of parallel IPHULA is briefly described for a cluster with LAM-MPI (LAM = Local Area Multicomputer) installed. This description is based on LAM-MPI's *One-Step Tutorial* (`http://www.lam-mpi.org/tutorials/one-step/lam.php`).

Hostfile:

First one has to generate a hostfile (e.g., `lamhosts`) containing all machines the parallel program should run on, say `gold`, `copper`, and a 2 processor machine called `silver`. Then `lamhosts` may look like this:

```
gold
copper cpu=1
silver cpu=2
```

The `cpu` option specifies the number of CPUs or cores that may be used on the respective machine. The `cpu` option is not necessary if a host has only one CPU/core to use.

Please ensure that using secure shell (ssh) you can login to all nodes from all other nodes without a password.

Bootability (optional):

To verify whether the cluster is bootable run:

```
> recon -v lamhosts
n-1<12583> ssi:boot:base:linear: booting n0 (copper)
n-1<12583> ssi:boot:base:linear: booting n1 (gold)
n-1<12583> ssi:boot:base:linear: booting n2 (silver)
n-1<12583> ssi:boot:base:linear: finished
```

(The output of the commands might slightly differ due to different setups.)

Boot the cluster:

Start the LAM-cluster with

```
> lamboot -v lamhosts

LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University

n-1<12585> ssi:boot:base:linear: booting n0 (copper)
n-1<12585> ssi:boot:base:linear: booting n1 (gold)
n-1<12585> ssi:boot:base:linear: booting n2 (silver)
n-1<12585> ssi:boot:base:linear: finished
```

Check the LAM-cluster:

Check whether the cluster runs properly with

```
> tping -c1 N
  1 byte from n0 (o): 0.000 secs

1 message, 1 byte (0.001K), 0.000 secs (7.026K/sec)
roundtrip min/avg/max: 0.000/0.000/0.000
```

Run parallel IPHULA:

You have to have the MPI-parallel IPHULA executable (`mpiiphula` in the `src` folder) built as described in section 3.3.1. If you have not done this yet, please perform that step first.

Start the parallel pIPHULA executable preferably with an input file as, for example, `test-input` found in the folder `data` (see section 5.3), since some clusters might not allow for interactive input. Use a command like:

```
>mpirun -np 4 mpiiphula -infile=input
```

The `-np` option tells the program how many parallel processes it should start on the LAM-cluster.

For settings, parameters, simulation design, and execution of pIPHULA refer to sections 4.1, 4.3, and 5.

To start the job in the background using a Bourne shell compatible shell (e.g. `sh`, `ksh`, `bash`) while collecting all output in a log-file, say `output.log` start pIPHULA with

```
>mpirun -np 4 mpiiphula -infile=input > output.log
```

This usually allows the user to exit the machine while the job remains running.

Monitoring an MPI application:

You can use `mpitask` and `mpimsg` to get more information about your running MPI programs. Please refer to your MPI documentation for more details.

Cleaning LAM:

With `lamclean` all your processes and messages are removed without rebooting the LAM-cluster:

```
> lamclean -v
killing processes, done
closing files, done
sweeping traces, done
cleaning up registered objects, done
sweeping messages, done
```

Then another MPI program like another pIPHULA analysis of a different program can be started without restarting the LAM-cluster from the beginning.

Terminating the LAM-cluster:

With `lamhalt` the LAM-cluster is stopped when not needed for further parallel computation:

```
> lamhalt

LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University
```

In case of severe errors like crashes of LAM nodes, `lamhalt` may hang. Then the LAM-cluster has to be 'cleaned' with the `wipe` command for the same `lamhosts` file the LAM-cluster was started with:

```
> wipe -v lamhosts

LAM 7.0.6/MPI 2 C++/ROMIO - Indiana University

n-1<12597> ssi:boot:base:linear: booting n0 (copper)
n-1<12597> ssi:boot:base:linear: booting n1 (gold)
n-1<12597> ssi:boot:base:linear: booting n2 (silver)
n-1<12597> ssi:boot:base:linear: finished
```

This removes all hanging jobs.

4.4.2 Using MPICH

In the following the use of parallel IPHULA is briefly described for a cluster with a MPICH implementation installed (`http://www-unix.mcs.anl.gov/mpi/mpich/`).

Machinefile:

First one has to generate a machinefile (e.g., `hostfile`) containing all machines the parallel program should run on, say `gold`, `copper`, and a 2 processor machine called `silver`. Then the `hostfile` will look like this:

```
gold
copper
silver:2
```

The value behind the colon (`:`) specifies the number of CPUs or cores that may be used on the respective machine. This value is not necessary if a host has only one CPU/core to use.

Please ensure that using secure shell (ssh) you can login to all nodes from all other nodes without a password.

Configure MPD:

If you already configured MPD for your set-up, you might skip this step.

Create the MPD configuration file `.mpd.conf` in your home directory by running the following:

```
cd $HOME
touch .mpd.conf
chmod 600 .mpd.conf
```

Use any editor to edit the `.mpd.conf` in your home directory on the cluster. Add the line

```
MPD_SECRETWORD=foo123bar
```

where `foo123bar` has of course to be replaced by another secret word.

Start MPD:

Start the parallel environment with the command:

```
> mpd --pid=${HOME}/mpd.pid &
```

Run parallel IPHULA:

You have to have the MPI-parallel IPHULA executable (`mpiiphula` in the `src` folder) built as described in section 3.3.1. If you have not done this yet, please perform that step first.

Start the parallel pIPHULA executable preferable with an input file as, for example, `test-input` found in the folder `data` (see section 5.3), since some clusters might not allow for interactive input. Use a command like:

```
>mpiexec -maschinefile hostfile -n 4 mpiiphula -infile=input &
```

The `-maschinefile` option specifies the location of the above mentioned maschinefile containing the hosts and CPUs to be used. The `-n` option tells the program how many parallel processes it should start on the cluster.

For settings, parameters, simulation design, and execution of pIPHULA refer to sections 4.1, 4.3, and 5.

To start the job in the background using a Bourne shell compatible shell (e.g. `sh`, `ksh`, `bash`) while collecting all output in a log-file, say `output.log` start pIPHULA with

```
>mpiexec -maschinefile hostfile -n 4 mpiiphula -infile=input > output.log &
```

This usually allows the user to exit the machine while the job remains running.

Killing MPD:

To finish the MPD daemon determine its process ID (PID) which was written into the file `${HOME}/mpd.pid` on startup. Terminate the MPD daemon with the command `kill PID`, where `PID` must be replaced by the process ID given in `${HOME}/mpd.pid`. You may want to remove the PID file with `rm ${HOME}/mpd.pid`.

5 Hints for Running pIPHULA

5.1 Defining the Simulation Grid

The goal of simulations using pIPHULA is to find the maximum likelihood parameter values for ρ, θ and τ employing a 3-dimensional grid that encloses the global maximum. The position of this grid in the space of possible parameter values needs to be defined by the user a priori, however, generally the (approximate) location of the maximum likelihood peak is unknown. It is thus necessary to explore the likelihood surface and find the θ and τ ranges that enclose the global maximum likelihood peak in preliminary runs of the program.

We suggest to start with a ρ value that is likely to represent the demography of the population under investigation and a widely spaced grid covering a larger range of θ and τ values for a low number of simulation repeats (e. g. 100 to 1000 repetitions). Subsequently, zoom into grid areas of high likelihoods adjusting grid spacing and parameter ranges to research objectives and computational resources.

5.2 Choosing the Tolerance Limit δ

The likelihood of a specific growth parameter combination is approximated by the frequency with which both S and K are hit by a simulated data set. Possible values of S are discrete integers, thus a hit is counted if exactly the same number of polymorphic sites are found in a simulated data set as in the original data set. Since values of K are continuously distributed, a hit is counted if the difference between the simulated K value and the observed K value is smaller than δ.

An appropriate value of δ will depend on the characteristics of the analyzed data set. We suggest to start with the default setting $\delta = 0.2$, perform preliminary pIPHULA simulations and check the results. If the the proportion of $Zeros$ in the resulting matrices (`FILEPREFIX_RHO.matrix`) is too high, δ should be increased, if the hit frequencies in the matrices are too high, reduce δ.

5.3 Batch Mode

Running (p)IPHULA from a Unix batch file is straightforward despite the lack of command line switches. For this, you have to pipe the parameters to the standard input of the program. For example, to run (p)IPHULA with a the transition/transversion parameter equal to 10 the following lines in a shell script are sufficient:

```
iphula << EOF
y
10
200
40
1.2
y
y
10000
2
y
EOF
```

A second possibility is to create a parameter file, e.g. `input` containing all input, as for example (using the above settings)

```
y
10
200
40
1.2
y
y
10000
2
y
```

A parameter file defining all parameter settings can be found in the pIPHULA distribution package in `data/test-input`

Such a parameter file can be piped directly into (p)IPHULA, using `iphula < input` or `cat input | iphula`. You can also tell (p)IPHULA to read in a command file via a command line flag: `iphula -infile=input`.

If the simulation parameters are typed or piped into the program, the results will be written by default to files with the filename base "panel" (e.g. `panel.iphula`). If a command file is given via the `-infile` option as above, the file is named following the above example `input.iphula`. By default the prefix is identical to the infile's name but can also be changed at the command line using the `-prefix=PREFIX` option. Then `PREFIX` is the base of the output file names used to append the extensions, e.g. if `-prefix=input` then the record file will be called `input.iphula`.

Started as described above, (p)IPHULA will still print its status messages to the standard output. If this behavior is not wanted, e.g., when running a batch job in the background, the the output can be piped into a separate file like `iphula.log` by adding '> `iphula`' at the commandline. The command

`iphula < input > iphula.log &`

for example will run IPHULA with the content of the file `input` as input in the background. All status messages will end up in `iphula.log`.

6 Supporting Programs and Scripts

6.1 Graphical Output

An (incomplete) list of programs supporting pIPHULA by providing graphical presentation of results:

R: a free software environment for statistical computing and graphics that runs under UNIX/Linux, Mac OS X, and Windows (`http://www.r-project.org`).

As part of the pIPHULA distribution package the R routine `Plot-Iphula.R` is distributed and can be found in `R-scripts/Plot-Iphula.R`. On UNIX and Mac OS X operating systems the routine can also be run via a shell script (`R-scripts/plot-iphula.sh`).

Extension	**file content**
`.matrix`	input containing the θ - τ matrix for each ρ value
`.params`	input of run parameter settings
`.LNmatrix`	output of the θ - τ matrix for each ρ transformed into ln likelihood values
`.pdf`	graphical output of the results per ρ value in pdf format
`_all.pdf`	compilation of all ρ graphics on one or more pages
`_max_list.txt`	lists of likelihood maxima per ρ and over all parameters

The routine uses input from the files `FILEPREFIX_RHO.matrix` and `FILEPREFIX.params`. All input files need to be situated in the same directory as the routine.

To apply `Plot-Iphula.R` to your output data, start the statistics program `R`, load the routine with

```
source("/path/to/routine/Plot-Iphula.R")
```

and execute it:

```
plot.iphula(FILEPREFIX)
```

The shell script can also be called directly from the commandline by navigating to the directory with the shell script and typing

```
./plot-iphula.sh FILEPREFIX
```

First, the approximate likelihood values (frequency data) output by (p)IPHULA (`FILEPREFIX_RHO.matrix` files) are transformed into ln likelihoods and the resulting matrices are saved as `FILEPREFIX_RHO.LNmatrix` files.

The ln likelihood results for each ρ value are then plotted as panels. These panels are on one hand saved individually as pdf files(`FILEPREFIX_RHO.pdf`). On the other hand up to six panels are compiled onto one page and saved in a single pdf file (`FILEPREFIX_all.pdf`). The ln likelihood for a specific parameter combination ρ, θ and τ can be represented in either a gray-scale plot (black = high likelihood) or a heat plot ranging from blue to red colors. Blue colors stand for low likelihoods, white for intermediate and red for high likelihoods. To switch between gray-scale and color output, you need to comment the appropriate `image` commands in the R code in and out, respectively. The colors in all plates resulting from a pIPHULA run are represented relatively to the maximum likelihood value across all parameter settings (i. e. also across panels = ρ values). Fig. 6.1 gives an example output of `plot-iphula.R` for a pIPHULA run with six ρ settings.

Finally, a text file is generated that provides the maximum likelihood value(s) and its/their parameter settings per panel (i.e. ρ) and over all parameters.

6.2 Compilers and Other Software

A list of available compilers is provided in section 3.4. Links to lists of implementations of the Message Passing Interface (MPI) library for parallel computing are given in section 3.3.1.

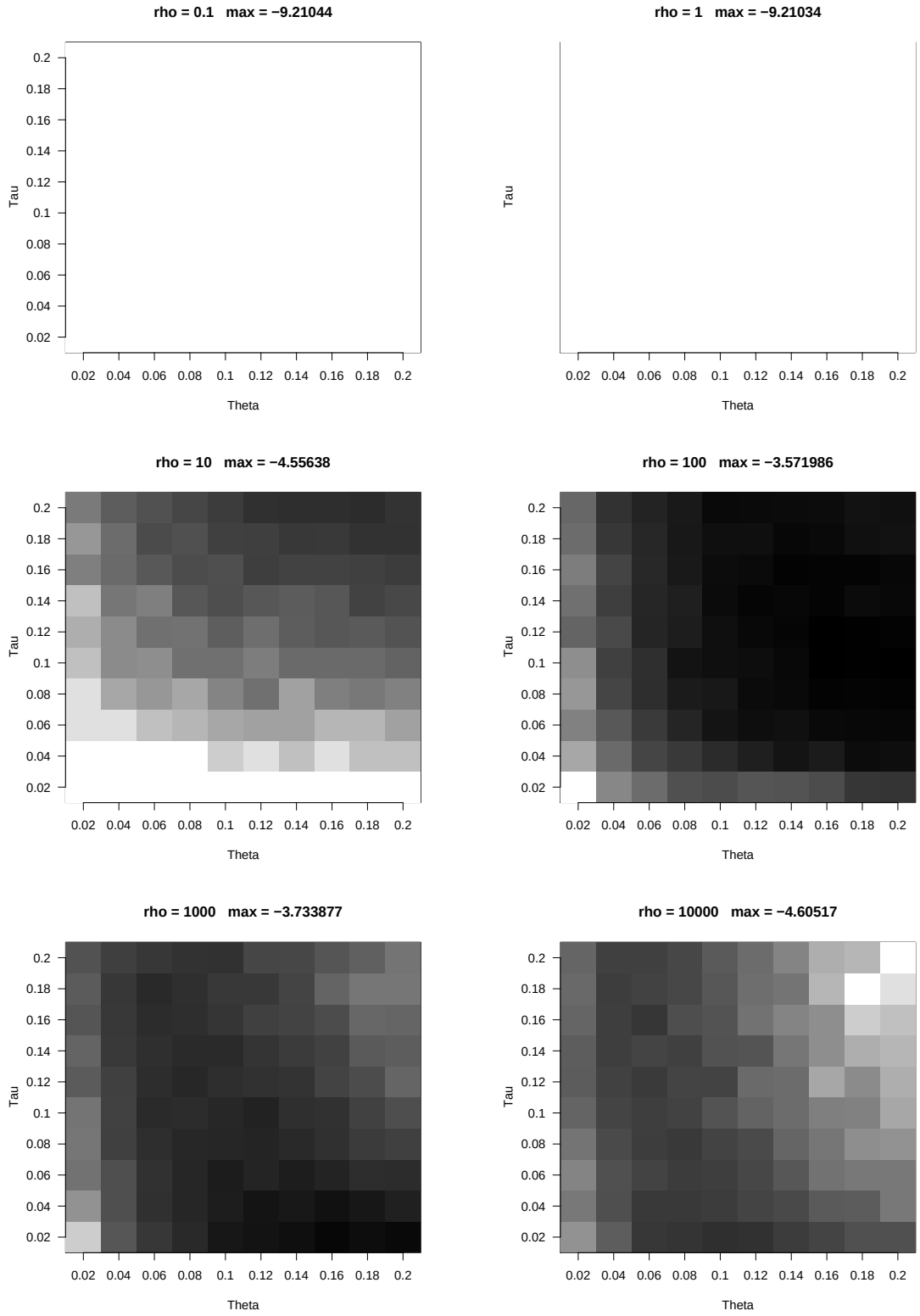

Figure 6.1: ln likelihood surface of the gene *accn2* for a sample of European humans (darker = higher likelihoods). 28

7 (p)IPHULA References and Further Reading

For a more extensive description of the theory behind the population size change model used in pIPHULA and the implementation of the simulations we recommend to refer to

> Weiss, G. and A. von Haeseler (1998) Inference of Population History Using a Likelihood Approach. *Genetics*, 149, 1539-1546.

This paper also describes an application of the previous non-parallel version of (p)Iphula that might provide further insight into possible analysis with the (p)IPHULA program.

8 Acknowledgments and Credits

We would like to thank Gunter Weiss for providing helpful comments on the IPHULA code. AvH and JB were supported by a grant of the *Deutsche Forschungsgemeinschaft* (DFG Ha 1628/7-1). HAS and AvH acknowledge financial support by the *Wiener Wissenschafts-, Forschungs- und Technologiefonds* (WWTF).

As a scalable random number generator we use source code from the SPRNG (Scalable Pseudo Random Number Generator) library (Mascagni and Srinivasan, 2000).

Bibliography

Dagum, L. and Menon, R. (1998) OpenMP: An industry-standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, **5**, 46–55.

Griffiths, R. C. and Tavaré, S. (1994) Sampling theory for neutral alleles in a varying environment. *Phil. Trans. R. Soc. Lond. B*, **344**, 403–410.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W. and Snir, M. (1998) *MPI: The Complete Reference - The MPI Extensions*, volume 2. 2nd edition, The MIT Press, Cambridge, Massachusetts.

Mascagni, M. and Srinivasan, A. (2000) SPRNG: A scalable library for pseudorandom number generation. *ACM Trans. Math. Software*, **26**, 436–461.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. (1998) *MPI: The Complete Reference - The MPI Core*, volume 1. 2nd edition, The MIT Press, Cambridge, Massachusetts.

Weiss, G. and von Haeseler, A. (1998) Inference of population history using a likelihood approach. *Genetics*, **149**, 1539–1546.

9 Known Bugs

If you observe any strange behavior or other bugs please contact the developers!

10 Version History

pIPHULA 1.16 first official release of (p)IPHULA.

- PDF Manual added.
- R-scripts included to produce graphics from (p)IPHULA results.
- Changes in the output format.
- Several minor changes and bug fixes.

pIPHULA 1.10 first pre-release of (p)IPHULA.

pIPHULA 1.x improved and parallelized version of IPHULA.

IPHULA first implementation of the WvH98 model to infer parameters of populations history as described in Weiss and von Haeseler (1998).