
Getting Started with CoaSim/Python

An introduction to the simulator CoaSim

Thomas Mailund
mailund@birc.au.dk

Copyright © 2006 Thomas Mailund • Bioinformatics ApS

Permission is granted to make and distribute verbatim copies of this manual provided the copyright notice and this permission notice are preserved in all copies.

About CoaSim

CoaSim is a tool for simulating the coalescent process with recombination and geneconversion, under either constant population size or exponential population growth. It effectively constructs the ancestral recombination graph for a given number of chromosomes and uses this to simulate samples of SNP and micro-satellite haplotypes or genotypes.

CoaSim comes in two flavours: A graphical user interface version for easy use by novice users, and a script based version (using either Guile-Scheme or Python) for efficient batch simulations. This document is an introduction to the Python based version.

Installing CoaSim

CoaSim is distributed as RPM files, binary tar files, or as source code. For most users, we recommend installing from the RPM files, since building the tool from source requires setting up the right build environment and having access to the needed development tools. If you are not familiar with UNIX C++ development—using the Automake suite of tools and Python's distutils system—we do not recommend that you try building from source.

Installing the RPM Files. The RPM files contains binary versions of the program, compiled to an Intel x86 Linux platform. To install the Scheme version, run

```
> rpm -Uvh coasim-python-x.y.z-r.i386.rpm
```

Since the RPM files installs in the directory `/usr/local/`, installing the RPM package requires root access.

Installing the Binary tar Files. The binary tar files contains a pre-compiled version of CoaSim that should just be untared in the root. Notice that you need to have root access to do this:

```
> cd /
> tar zxf /path/to/tarfile/coasim-python-x.y.z.linux-i686.tar.gz
```

If you do not have root access, you can untar CoaSim in a different directory, but in that case it is necessary for you to set your PYTHONPATH so Python can locate the CoaSim module, e.g. in bash

```
> export PYTHONPATH=/path/to/CoaSim/Python:$PYTHONPATH
```

or in csh

```
> setenv PYTHONPATH /path/to/CoaSim/Python:$PYTHONPATH
```

Installing from the Source Files. The source code is distributed in a tar-file; to build the Python version from the source files untar the file, build the Core module

```
> tar xzf coasim-python-x.y.z.tar.gz
> cd coasim-python-version/Core
> ./configure
> make
```

and then build the Python module:

```
> cd ../Python
> python setup.py build
```

This will build a module that you can import into your Python scripts. You will need to either install it or make sure you have the path to it in your PYTHONPATH, e.g. in bash

```
> export PYTHONPATH=/path/to/CoaSim/Python:$PYTHONPATH
```

or in csh

```
> setenv PYTHONPATH /path/to/CoaSim/Python:$PYTHONPATH
```

Running CoaSim

Once installed, you can load CoaSim into python using the `import` command:

```
> python
Python 2.3.4 (#1, Feb  2 2005, 12:11:53)
[GCC 3.4.2 20041017 (Red Hat 3.4.2-6.fc3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import CoaSim
>>>
```

to get a short overview of the functionality in this module, use

```
>>> help(CoaSim)
```

Using CoaSim

Running CoaSim will in most cases consist of three steps: Set up the simulation parameters, including the markers (type of marker, mutation rates and similar), demographic parameters, rates for recombination, etc.; running the simulation obtaining and ARG; and extracting the needed information from the ARG (*Ancestral Recombination Graph*), e.g. the resulting sequences, the timing of the various events, or the local coalescent trees embedded in the ARG.

Before using the Python based CoaSim *it is necessary that you have installed the module*, not just built it. If the module has not been properly installed, it will not be able to locate the module and load it into Python. If it is not possible to install the modules globally, you can install the locally but you will then need to

set the PYTHONPATH environment variable to let Python know where the module is installed.

Once CoaSim is successfully installed, it can be loaded into Python and used as any other python module. Controlling the simulations through Python makes CoaSim a very flexible and powerful simulation tool. However, with flexibility inevitably is associated some complexity, and while running simple simulations through the Python interface is quite simple, the more complex simulation tasks require a bit of knowledge about the Python programming language and the Python modules supplied with CoaSim. To keep this ‘getting started’ guide short, we do not attempt to explain the Python interface to CoaSim in detail here—for this we refer to the *CoaSim/Python Manual*—instead we give a short introduction to running very simple simulations.

A very simple use of CoaSim is to simulate coalescent trees. A script for that is shown here:

```
from CoaSim import simulate, SNPMarker

markers = [SNPMarker(0.5,0,1)]
noLeaves = 10

arg = simulate(markers, noLeaves)

tree = arg.intervals[0].tree
print tree
```

This might look a bit complicated, but if you break it down in the three phases mentioned at the beginning of this section—setting up parameters, running a simulation, and analysing the result—it is really not.

The first line is not really part of any of the three phases, but simply loads the CoaSim module into Python and imports `simulate` and `SNPMarker` into the global namespace. The next two lines define the list of markers (polymorphic sites) the simulated ARG should contain, and the number of leaves the ARG should contain. For the markers, we specify a list of a single marker, a *Single Nucleotide Polymorphism* (SNP) marker, positions at the middle of the genomic region we consider (position 0.5), with the 1-allele frequency to be between 0 and 1, i.e. unrestricted.

This concludes the setup-phase. The next line is the simulation phase; it calls `simulate` to obtain an ARG. From this ARG we can, in the third and final phase, extract the list of intervals sharing the same genealogy (`arg.intervals`)—these are intervals where no recombination has occurred, in this particular case there is only one since by default the recombination rate is 0—select the first interval, and extract the genealogy for that interval, which we then print.

As another simple application, consider simulating a list of SNP sequences. A script for simulating 100 sequences with 10 SNP markers in each is shown here:

```
from CoaSim import simulate
from CoaSim.randomMarkers import makeRandomSNPMarkers
```

```

markers = makeRandomSNPMarkers(10, 0.0, 1.0)
sequences = simulate(markers, 100, rho=400).sequences

print sequences

```

The first line, once again, loads CoaSim into Python and imports the `simulate` function into the global namespace. The second line loads another module, `CoaSim.randomMarkers` and loads the method `makeRandomSNPMarkers` from that module into the global namespace. This method is used to create a list of randomly¹ positioned SNP markers in the next line. Here we choose a list with 10 markers, with the 1-allele frequency once again between 0 and 1. The next line simulates the ARG, this time with 100 leaves, and then extracts the sequences from the ARG.

The keyword argument, `rho`, sets the scaled recombination rate $\rho = 4N_e r$ to 400 (which for an effective population size N_e of 10,000 roughly correspond to 1 cM).

Printing sequences with the `print` command will print a list of lists; this is a format that is very simple for Python to read and manipulate, but is usually not suitable for other analysis tools. To print the sequences in a more traditional form of a sequence per line, with the sequences printed as space-separated numbers, we can use the `CoaSim.IO` module like this:

```

from CoaSim.IO import printMarkerPositions, printSequences
printMarkerPositions(markers)
printSequences(sequences)

```

Here, the positions of the marker list are first output as a line of space-separated numbers, and then the sequences are output. In both cases, the output is to the terminal, but using a second argument to the two print functions, we can direct the output to files. E.g. to write the positions to `positions.txt` and the sequences to `sequences.txt`, we use:

```

from CoaSim.IO import printMarkerPositions, printSequences
printMarkerPositions(markers, open('positions.txt', 'w'))
printSequences(sequences, open('sequences.txt', 'w'))

```

A common setting is simulating a set of sequences and then split them into cases and controls based on a trait mutation. A trait mutation is really just any mutation, typically on a bi-allelic marker. As such, a disease locus can be modelled simply as a SNP marker. SNP markers, however, model a certain ascertainment bias in the simulations—one that matches the typical SNP selection bias—when the mutant allele frequency is restricted.² To simulate without this bias, we can use a `TraitMarker`.

A `TraitMarker` instance is created just as a `SNPMarker` instance, and takes the same arguments—position and range of the mutant allele frequency—and can be added to the marker list for a simulation just as a SNP marker.

¹Randomly here means uniformly distributed on the interval being simulated.

²When the mutant allele frequency is unrestricted, a trait marker and a SNP marker has exactly the same semantic. Due to the different rejection-sampling approaches to restricting the allele frequencies for SNP and trait markers, respectively, restricting trait markers can be significantly more CPU demanding than restricting SNP markers.

To simulate sequence data with a single disease affecting marker, we can use:

```
from CoaSim import simulate, TraitMarker, insertSorted
from CoaSim.randomMarkers import randomPosition, makeRandomSNPMarkers
from CoaSim.diseaseModelling import singleMarkerDisease, split

markers = makeRandomSNPMarkers(10, 0.0, 1.0)
traitMarker = TraitMarker(randomPosition(), 0.2, 0.4)
markers,traitIdx = insertSorted(markers, traitMarker)

sequences = simulate(markers, 100, rho=400).sequences
diseaseModel = \
    singleMarkerDisease(traitIdx, wildTypeRisk=0.1, mutantRisk=0.3)
affected, unaffected = split(diseaseModel,sequences)
```

The first three lines just load modules and import functions and classes into the global namespace, as before. We then create 10 randomly positioned SNP markers, as we did earlier, but now also a single randomly positioned trait marker. We insert this into the list of markers, to get a new list (of 11 markers, 10 SNP markers and the trait marker) and the index where the trait marker can be found. We then simulate 100 sequences and split them into affected and unaffected using the `split` function from module `CoaSim.diseaseModelling`.

The first argument to `split` is a disease model. This parameter determines how the sequences should be interpreted and how disease status is determined. In this case we use a simple single-locus model, where the alleles of the trait marker determines the disease status based on the probability of a wild-type being affected (the `wildTypeRisk`) and the probability of a mutant being affected (the `mutantRisk`).

This disease model considers the sequences haploid, but other models consider the sequences pairwise³ as diploid individuals and determines disease status from this. For example, to simulate a dominant or recessive diploid disease, we can use

```
from CoaSim.diseaseModelling import dominantMode, split
diseaseModel = dominantModel(traitIdx)
```

or

```
from CoaSim.diseaseModelling import recessiveMode, split
diseaseModel = recessiveModel(traitIdx)
```

respectively.

Contact

For any comments or questions regarding CoaSim, please contact Thomas Mailund, at mailund@mailund.dk or mailund@birc.au.dk.

³The sequences are considered pairwise in the order they appear in the sequence list, i.e. for $i = 0, 2, 4, \dots$, `sequence[2i]` and `sequence[2i + 1]` are paired.