APEMoST Documentation

Release 1.3HEAD

Johannes Buchner

January 08, 2012

CONTENTS

CHAPTER

ONE

AUTOMATED PARAMETER ESTIMATION AND MODEL SELECTION TOOLKIT

APEMoST is a free, fast MCMC engine that allows the user to apply Bayesian inference for parameter estimation and model selection.

1.1 Quick links:

1. ***General information*** What is APEMoST? What is Bayesian Inference?
2. ***User manual*** Includes *how to get APEMoST*.
3. *FAQ*. Also see here for contact details.

1.2 Full Table of Contents:

1.2.1 Automated Parameter Estimation and Model Selection Toolkit (APEMoST)

If you are unfamiliar with Bayesian inference using MCMC (Markov Chain Monte Carlo) sampling/updates, you can find some information in the Links section.

The program was first written by Michael Gruberbauer (at the time University of Vienna). Later it was rewritten for parallelization by Johannes Buchner (at the time University of Vienna, Technical University of Vienna), which was funded by Werner W. Weiss (University of Vienna). Now it is maintained by Johannes Buchner.

Specification

APEMoST enables the user to perform parameter estimation in a simple way: Only three things have to be provided:

1. The likelihood function

   Specified as a C function (usually ~15 lines of code)

2. The parameter space

   Specified as a text file, containing the borders and starting points

3. A data file

   A table of data the likelihood function can work on is read in and available as a matrix.

Fundamentals

APEMoST is written in ANSI C and uses the GNU Scientific Library (GSL). This makes it a very fast sampler. In fact, if run on a million iterations, less than a minute is spent in APEMoST, the rest of the time the CPU evaluates the likelihood function.

APEMoST utilizes all CPUs on a computer, by using OpenMP.

Features

- **Automatic stepwidth calibration** of the proposal distribution.

  The stepwidth of each parameter is optimally calibrated for the desired acceptance rate.

- **Simulated Tempering and Parallel Tempering**:

  A number of chains (usually 20) is run and swap their positions. This can be disabled by setting the number of chains to 1.

- **Automagic**:

  For example, the frequency of swaps, the betas of the hottest chain and many more values are calculated to have a sane value the user usually doesn't have to mess with.

  Furthermore, we discovered a way of predicting the stepwidths of chains once two chains are calibrated. This allows skipping time consuming calibration and yet reaching the desired acceptance rates

We also have some differences to other software.

Ready to dig in? **Get started** with the user manual, it explains APEMoST step by step.

1.2.2 User Manual

A general introduction, frequently asked questions and the license can be found at *the main site*. The website of APEMoST is http://apemost.sourceforge.net/. The latest version of this manual can be found there aswell.

Contents

Prerequisites

You need to have the following software installed:

- gcc (>= 4.1, for OpenMP)
- GNU scientific library (libgsl0-dev or gsl-devel)
- Boehm garbage collector (boehmgc or gc)

  The garbage collector is optional, but can be enabled by compiling with make WITH_GARBAGE_COLLECTOR=1

For the version control (keeping track of different versions, experimenting with code and patches)

- git (git-core or git)

For generating the documentation (make doc):

- doxygen
- sphinx (the python package)

It may be handy to keep this manual open, and the GSL manual as well as the API documentation.

Getting APEMoST

You can get a release of APEMoST from the Sourceforge project page http://sourceforge.net/projects/apemost/ or fetch the latest code using git.

The general concept

A quick overview of the components. All of these will be explained in detail in the following sections.

There are 2 pieces of software:

1. The MCMC engine, APEMoST

   Its code can be found in the folder src/

2. The user-provided likelihood function

   The code has to be placed in apps/

Say your project is called "foo". The idea is that you implement one function, calc_model, in apps/foo.c . Then, you run

```
$ CCFLAGS="-DMAX_ITERATIONS=1000000" make foo.exe
```

and this compiles APEMoST and your function together. All code is compiled at once to get the best optimization. You can specify the algorithms behaviour and many other things with CCFLAGS. This will be explained later.

Furthermore, when you APEMoST (that has your likelihood function in it), you need two more files that you need to place where you start your program foo.exe (in the working directory). They are called data and params.

1. The definition of the parameter space, "params"

   Each line characterizes a dimension of parameter space. The columns are:

   (a) Start value

   (b) Min value

   (c) Max value

   (d) Name (a string)

   (e) Initial step width (before calibration)

Use a name without whitespaces. You can set the step width to -1, which results in automatically using 10% of (max - min).

2. A data table, "data"

   This file will be read into a matrix that is accessible for the likelihood calculation.

   If you have your data elsewhere, you can also use a symlink.

General remarks

The program operates on logarithmic probability values (loglikelihood). Everywhere and exclusively. Double precision is used.

The values read from files can be of many formats (everything scanf can read).

Phase 0 - Preparation

In this phase of your progress, you will want to

- specify your likelihood function from a formula
- compile the program the first time
- get used to CCFLAGS

Equipped with the GSL manual, take a look at the file apps/simplesin.c. You will write something very similar.

You have to implement two functions:

```
void calc_model(mcmc * m, const gsl_vector * old_values);

void calc_model_for(mcmc * m, const unsigned int i, const double old_value) {
        calc_model(m, NULL);
}
```

The second one, calc_model_for, is only used in the calibration: Only the ith parameter has been changed. The old values are given as parameters in case you can do some optimizations (e.g. if the parameter is just an offset, simply add/subtract something from the probability). The default (as above) is to just recalculate everything it using calc_model().

The first function, calc_model() has to

- look at the parameter values

  get_params(m) returns all parameters as a vector.

  get_params_for(m, i) returns the ith parameter value.

- look at the data table

  m->data is the matrix read in from the file "data".

  You can do read operations on this matrix, e.g. gsl_matrix_get(m->data, i, j)

- calculate and set the prior

  The program has to keep track of the prior, since we need the probability both with and without the prior.

  Use set_prior(m, myprior);

- calculate and set the probability

The probability has to contain the prior, and has to incorporate beta.

Example: set_prob(m, get_prior(m) + get_beta(m) * myprob);

As your MCMC papers will tell you, the priors should not be exponentiated by beta.

Keep in mind that everything is logarithmic (loglikelihoods!).

m, more precisely the mcmc structure, represents one chain.

First compilation

Lets try to compile your program. I'll take simplesin as an example (replace simplesin with your project name).

I run:

```
$ make simplesin.exe
$ make eval_simplesin.exe
$ make benchmark_simplesin.exe
```

If everything works out, I get three executables: simplesin.exe, eval_simplesin.exe and benchmark_simplesin.exe (replace simplesin with your project name).

Lets see if our loglikelihood function is correct, and evaluate it at.

We change into a empty directory we want to work from, and put two files there.

In "data" (for example, also see data):

```
101     0.67
102     1.01
103     7.9e-1
104     1.34
and so on
```

In "params" (see above at params):

```
0       0       2       amplitude       -1
0       0       0.3     frequency       -1
0       0       1.0     phase   -1
0       0       2       offset  -1
```

You can specify the values in different formats (e.g. 0.13, 1.3e-1) and use tabs or spaces as you like (I would recommend tabs). Now we can try out the likelihood function (replace apemost-directory with where the apemost code and the Makefile is):

```
$ apemost-directory/eval_simplesin.exe
(you enter:)    1 0.2 1 0
(output:)       -1.480898044165363e+01  0.000000000000000e+00
(quit with Ctrl-C or Ctrl-D)
```

The first is the probability, the second the prior.

You may get this error, which can be a little confusing:

```
gsl: ../gsl/gsl_vector_double.h:177: ERROR: index out of range
Default GSL error handler invoked.
Aborted
```

This means you tried to access a element beyond the size of the vector (or matrix). In that case, the function expects a different number of parameters than the params file provides. Although this is less relevant for the first read, you can also benchmark your likelihood function with the benchmark_simplesin.exe you produced. It takes the number of evaluations as arguments.

The third way of accessing the MCMC engine is the really interesting one:

```
$ apemost-directory/simplesin.exe
$ apemost-directory/simplesin.exe check
```

This also outputs some inline help about the phases.

You can find the main() functions of these three programs in apps/generic_main.c, apps/eval.c and apps/benchmark.c.

CCFLAGS

It is essential that you understand the CCFLAGS variable. This will be the main "interface" how you tinker with the program, change its default values and its behaviour.

For compilation, you can do something like:

```
CCFLAGS="-DMAX_ITERATION=100000 -DWITHOUT_GARBAGE_COLLECTOR" make simplesin.exe
```

This tells the compiler to set preprocessor values. Here, I call e.g. WITHOUT_GARBAGE_COLLECTOR a "flag", and you "set the flag" by appending it to your CCFLAGS string with "-DFLAG" and you "set the flag to a value" using -DFLAG=value.

The check subcommand outputs the values currently set (after compilation):

```
$ apemost-directory/simplesin.exe check
```

A full list of flags can be found in the API documentation, with their meaning and default values. This is a good resource that you should keep open.

If you were to write a new calibration algorithm, or use a different adaptive MCMC algorithm, you would use "#ifdef MYFLAG" preprocessor directives and enable/disable the use of the algorithm by a flag.

The perhaps most important flag is DEBUG, which enables some debug output.

Note: Smart readers will notice that you have to rebuild the program when you want to change a flag something.

In this phase of your progress, you learned how to

- specify your likelihood function from a formula
- compile the program the first time
- get used to CCFLAGS

Very good! You get a cookie.

Phase 1 - Calibration, first chain (beta = 1)

A good MCMC sampling should have a good acceptance rate. Different sources state different things, something between 30% and 80% should be right.

To reach this acceptance rate, a calibration algorithm tinkers with the stepwidths of the proposal distribution (lets assume the default, a multivariate normal distribution).

The inline help shows which flags are relevant:

```
$ apemost-directory/simplesin.exe help calibrate_first
```

You will want to enable the DEBUG flag, otherwise you won't see much if stuff goes wrong. Ideally, you don't have to care about it, practically you will want to see which stepwidths scale up, which scale down.

You can run the calibration with:

```
$ apemost-directory/simplesin.exe calibrate_first
```

The result of the calibration will be a file that stores the calibrated stepwidths, "calibration_results". The rows are defined as:

```
beta    param1_stepwidth        param2_stepwidth        ...     param1_value    param2_value    .
```

Each chain will get one such row in the next phase. For now, just one row in this file.

Also, the program suggests a new params file ("params_suggested") that contains the new stepwidths (last column). If you use these stepwidths in your params file, this will make your next calibrate_first run go faster.

Possible problems in this phase

- stepwidth gets too large

  You may want to increase the parameter space.

  This can also mean the posterior distribution is independent of this parameter!

- stepwidth gets too small
- calibration fails

  You can increase ITER_LIMIT.

- calibration takes too long and doesn't find a good end point.

  Bad.

  Among many things, you can try altering MAX_AR_DEVIATION.

  Among the less recommended, but possible solutions are:

  – manually setting some stepwidths

  You can also add another calibration algorithm to APEMoST (we'd be happy).

Note: You can watch the progress of the calibration by plotting the file "calibration_progress.data". The columns are defined as:

1. parameter number (starting with 0)
2. number of iteration done
3. stepwidth ([0..1], normalized to parameter space)
4. acceptance rate
5. accuracy of the acceptance rate estimate (-1 if not available)

For example:

```
0       200     0.058824        0.590000        -1.000000
1       200     0.050000        0.590000        -1.000000
2       200     0.058824        0.590000        -1.000000
3       200     0.058824        0.590000        -1.000000
0       400     0.069204        0.395000        -1.000000
and so on
```

Phase 1 - Calibration, other chains (beta <= 1)

Now we just have to do the same with the hot chains.

There are some interesting facts about the hot chains in APEMoST, for example

1. Per default, beta is not distributed equally, but using a chebyshev scheme

   This proved to be quite good so far, I tested out several methods (also see my bachelor thesis).

2. The hottest chain's beta is automatically determined so that the stepwidths will be maximally the size of the parameter space.

   If the beta_0 seems suspicously low, you can set the flag BETA_0 to something sensible, 0.01 is often used.

3. There is a mechanism that allows skipping the calibration of all but two chains

   This saves you plenty of time, and possible calibration failures with the hottest chain.

   You can enable it with the flag SKIP_CALIBRATE_ALLCHAINS.

   Although this technique, developed by us (see my bachelor thesis), is not based on a sound mathematical proof (yet?), I have yet to see a scenario where this technique is inappropriate.

If you want a different number of chains, set N_BETA.

With our knowledge from the previous chapter, we look up the inline help:

```
$ apemost-directory/simplesin.exe help calibrate_rest
```

and run:

```
$ apemost-directory/simplesin.exe calibrate_rest
```

Note: You should also be aware that when you change a flag, the likelihood function or the parameter space, you may have to do the calibration again, as the stepwidths will not be appropriate anymore.

Example output

A output of the calibration phase can look like this (ideal case, DEBUG turned off, SKIP_CALIBRATE_ALLCHAINS turned on):

```
$ ../simplesin.exe calibrate_first
Initializing 20 chains
Starting markov chain calibration
wrote calibration results for 1 chains to calibration_results
new suggested parameters file has been written
$ ../simplesin.exe calibrate_rest
Initializing 20 chains
Calibrating chains
Calibrating second chain to infer stepwidth factor
        Chain  1 - beta = 0.993277      steps: Vector4d[0.052377;0.000060;0.036247;0.037842]
stepwidth factors: Vector4d[0.887411;0.887411;1.044013;0.887411]
automatic beta_0: 0.013294
        Chain  1 - beta = 0.993271      steps: Vector4d[0.046480;0.000053;0.037842;0.033582]
        Chain  2 - beta = 0.973269      steps: Vector4d[0.046955;0.000054;0.038229;0.033925]
        Chain  3 - beta = 0.940538      steps: Vector4d[0.047765;0.000055;0.038889;0.034510]
        Chain  4 - beta = 0.895972      steps: Vector4d[0.048939;0.000056;0.039844;0.035358]
        Chain  5 - beta = 0.840786      steps: Vector4d[0.050519;0.000058;0.041131;0.036500]
        Chain  6 - beta = 0.776486      steps: Vector4d[0.052569;0.000060;0.042800;0.037981]
        Chain  7 - beta = 0.704825      steps: Vector4d[0.055177;0.000063;0.044923;0.039866]
        Chain  8 - beta = 0.627758      steps: Vector4d[0.058466;0.000067;0.047601;0.042242]
        Chain  9 - beta = 0.547388      steps: Vector4d[0.062611;0.000072;0.050976;0.045237]
        Chain 10 - beta = 0.465906      steps: Vector4d[0.067866;0.000078;0.055254;0.049033]
        Chain 11 - beta = 0.385536      steps: Vector4d[0.074605;0.000085;0.060741;0.053902]
        Chain 12 - beta = 0.308470      steps: Vector4d[0.083405;0.000095;0.067906;0.060260]
        Chain 13 - beta = 0.236809      steps: Vector4d[0.095192;0.000109;0.077502;0.068776]
        Chain 14 - beta = 0.172508      steps: Vector4d[0.111531;0.000128;0.090805;0.080581]
        Chain 15 - beta = 0.117323      steps: Vector4d[0.135241;0.000155;0.110109;0.097712]
        Chain 16 - beta = 0.072756      steps: Vector4d[0.171737;0.000197;0.139823;0.124080]
        Chain 17 - beta = 0.040026      steps: Vector4d[0.231543;0.000265;0.188514;0.167290]
        Chain 18 - beta = 0.020023      steps: Vector4d[0.327367;0.000375;0.266531;0.236523]
        Chain 19 - beta = 0.013294      steps: Vector4d[0.401759;0.000460;0.327099;0.290271]
all chains calibrated.
```

```
        Chain  0 - beta = 1.000000      steps: Vector4d[0.052201;0.000060;0.036125;0.037715]
        Chain  1 - beta = 0.993271      steps: Vector4d[0.046480;0.000053;0.037842;0.033582]
        Chain  2 - beta = 0.973269      steps: Vector4d[0.046955;0.000054;0.038229;0.033925]
        Chain  3 - beta = 0.940538      steps: Vector4d[0.047765;0.000055;0.038889;0.034510]
        Chain  4 - beta = 0.895972      steps: Vector4d[0.048939;0.000056;0.039844;0.035358]
        Chain  5 - beta = 0.840786      steps: Vector4d[0.050519;0.000058;0.041131;0.036500]
        Chain  6 - beta = 0.776486      steps: Vector4d[0.052569;0.000060;0.042800;0.037981]
        Chain  7 - beta = 0.704825      steps: Vector4d[0.055177;0.000063;0.044923;0.039866]
        Chain  8 - beta = 0.627758      steps: Vector4d[0.058466;0.000067;0.047601;0.042242]
        Chain  9 - beta = 0.547388      steps: Vector4d[0.062611;0.000072;0.050976;0.045237]
        Chain 10 - beta = 0.465906      steps: Vector4d[0.067866;0.000078;0.055254;0.049033]
        Chain 11 - beta = 0.385536      steps: Vector4d[0.074605;0.000085;0.060741;0.053902]
        Chain 12 - beta = 0.308470      steps: Vector4d[0.083405;0.000095;0.067906;0.060260]
        Chain 13 - beta = 0.236809      steps: Vector4d[0.095192;0.000109;0.077502;0.068776]
        Chain 14 - beta = 0.172508      steps: Vector4d[0.111531;0.000128;0.090805;0.080581]
        Chain 15 - beta = 0.117323      steps: Vector4d[0.135241;0.000155;0.110109;0.097712]
        Chain 16 - beta = 0.072756      steps: Vector4d[0.171737;0.000197;0.139823;0.124080]
        Chain 17 - beta = 0.040026      steps: Vector4d[0.231543;0.000265;0.188514;0.167290]
        Chain 18 - beta = 0.020023      steps: Vector4d[0.327367;0.000375;0.266531;0.236523]
        Chain 19 - beta = 0.013294      steps: Vector4d[0.401759;0.000460;0.327099;0.290271]
calibration summary has been written
wrote calibration results for 20 chains to calibration_results
$
```

A more readable output (especially when you used DEBUG) is available in the file “calibration_summary”.

Phase 2 - Running

If you made it this far, you have almost won! You have calibrated chains (with the burn in already done).

In this phase the program will do the actual sampling, parallel tempering and write out

1. The visited parameter values of chain0 (beta = 1)

   The files are named by the scheme paramname-chain-0.prob.dump. These just consist of the visited values for each iteration (doubles for rejects).

   These will be used for parameter estimation.

2. The probabilities of all chains

   The files are named by the scheme prob-chain<chain number>.dump. They consist of two columns:

   (a) posterior probability including prior (as set by the likelihood function

   (b) likelihood (excluding prior) as calculated by the likelihood function, but the prior subtracted.

   These will be used for the data probability and model selection.

3. “acceptance_rate.dump” allows you to watch the acceptance rates.

   Its first column is the iteration count, the succeeding columns are the number of accepts.

   For convenience, a gnuplot file, acceptance_rate.dump.gnuplot is written that allows you to make a nice plot and press “refresh” in the gnuplot window to watch the progress while the program runs (also try “set key left”).

   On the one hand it would be nice to have the acceptance rates as percentages, but this way we present two pieces of information at once: The acceptance rate can be inferred by subtracting the previous row, or estimated by adding 0.5*x to the plot. But it also allows us to see when chains get seriously stuck (the plot goes horizontal).

The first two are called “dump files”. They can easily reach hundreds of megabytes. Unless you specify –append, the existing dump files will be overwritten.

The online help is as always available with:

```
$ apemost-directory/simplesin.exe help run
```

and run:

```
$ apemost-directory/simplesin.exe run
```

It is important to realise that the speed of the calculation is *only* limited by the loglikelihood function, and not by the output written to stdout or the files.

Stopping the run

Note Bene: The last line of output files may be invalid. A analysis tool that looks at the output in real time should ignore it. Read on for why:

For speed purposes, the output to the files is unbuffered. This means the last two lines could be:

```
3.592794839126184e-01(newline)
3.367089(no newline)
```

And the rest not yet written. This is done efficiently by the operating system, which operates on blocks, not on lines.

To force a flush, you can send the USR1 signal to the program:

> $ killall -SIGUSR1 simplesin.exe

Which will cause the program to flush all files, and then continue to run.

To stop the program, press Ctrl-C or send the TERM signal using "kill". This will also cause a flush, and the files will be cleanly finished.

Unless you specified MAX_ITERATIONS, the program will happily run forever.

You can also pause and continue the program using normal job control (see the manual of your shell on how to send STOP and CONT signals).

Speeding up the run You can use the benchmark program to evaluate the speed of your loglikelihood function. For example, *pow(a*b, 2)* is faster than *a*a*b*b*.

You can also get speed improvements from setting N_PARAMETERS. The program will then expect the given number of parameters. This allows the compiler to do loop unrolling.

At this point, you are probably waiting for the program to reach a million iterations. You deserve a banana (APEmost, get it?).

Phase 3 - Analyse

Since we not only want to fill our hard disks, at some point we will want to analyse our data.

In this phase, all the dump files are read in again. This is often not limited by the CPU, but the hard disk speed. As noted in the FAQ, you can analyse your files independently on a different computer, or paste several dump files together.

So far, APEMoST can produce the following statistics:

1. Marginal distribution histograms

   This gets you the pretty pictures you are looking for, i.e. the full posterior probabilities for each parameter.

   The files are – appropriately – named "paramname.histogram".

   NBINS and HISTOGRAMS_MINMAX are flags you might be interested in.

For convenience, a gnuplot file is written, "marginal_distributions.gnuplot". If you remove the leading '#' and run it with gnuplot, it will give you a nice graphic of all histograms. For your publication you probably want to use a eps file or a different plot program.

2. MCMC error estimate

   Essentially, this tells you how much the mean of a histogram changes over time. The sigma should be less than 1% of the histogram sigma. (It will say "** high!" if that is not the case.)

   The formula is from here.

   You should include this estimate in your publication.

   This does not do a clean overlapping batch estimate, just analyses a batch of the length sqrt(total number of iterations) after another. since the number of iterations is high, this should be sufficient (batch length > 500).

3. Model selection / data probability

   This will output the model probability and will let you compare this model to others.

   example output:

```
Model probability ln(p(D|M, I)): [about 10^-59] -135.52659

Table to compare support against other models (Jeffrey):
 other model ln(p(D|M,I)) | supporting evidence for this model
 ---------------------------------
       >  -135.5       negative (supports other model)
  -135.5 .. -145.5      Barely worth mentioning
  -145.5 .. -158.5      Substantial
  -158.5 .. -169.5      Strong
  -169.5 .. -181.5      Very strong
       <  -181.5       Decisive
```

   If you have evaluated another model, look up its logarithmic (ln) model probability in this table.

Pretty neat, eh? No cookie now, you got your histograms.

Possible problems in this phase

1. Straight peaks in the histograms

   These mean a chain got stuck. Bad.

   Either run until this peak vanishes, change the calibration, ... I think you could increase NBINS, and take a average of the neighbouring bins, throwing away extreme outliers.

2. The results may be unexpected, or you are not sure if they are correct

   Thinking about it, or simulating the data with the resulting parameters may help.

3. Some possible values in the parameter space may have not been detected

   This is one real mean danger, because you probably will never know. A as high number iteration as possible helps.

   If two or more peaks have been detected already, you can try to find out after how many iterations the last peak showed up. Maybe you should run for another so many iterations.

   You can try to increase or decrease BETA_0, the beta value of the hottest chain.

   You can also try to tinker with the calibration or the proposal distribution (e.g. using a distribution with a wider tail such as logit).

   It is a good idea to run the sampling several times and also with different starting points.

4. The heights of different, independent peaks in the histograms do not correctly represent the probability relations.

   This will almost always be the case. Since the runtime is finite, the frequency of visits will be distorted.

   You should evaluate the likelihood function at the peaks to get their real values. The eval executable and peaks.exe will help you with this.

   peaks.exe will retrieve the median and quartiles of any independent peak in the marginal distribution. (independent means 1% of parameter space is unused in between). Since peaks.exe does not use a histogram, it is exact! Prefer it to measuring out the histogram.

Hacking APEMoST

Feel free to read all the source, write and change algorithms and everything.

Feedback, ideas, remarks and problems are welcome and will be added to the *FAQ*.

As the *license* states, since we worked so hard on APEMoST and you get it for free, you are expected to contribute changes back to us, so everyone can profit.

Ideally, get familiar with git, which is the version control system in use. Some resources are here:

- http://cworth.org/hgbook-git/tour/
- http://git-scm.com/ http://book.git-scm.com/1_welcome_to_git.html
- http://zrusin.blogspot.com/2007/09/git-cheat-sheet.html

The most important commands are "git pull", "git commit" and "git format-patch". The last allows you to send us a patch of your changes, so everyone can profit from it.

You can also set up your own repository (which is very easy, e.g. on github), and just tell me that you will contribute there. This will allow me to pull your changes.

If this is all too much for you – before you decide not to contribute back – a tarball or zip file is also welcome. The contact address can be found at the `contact page`.

That said, a version control system is really useful to stay on top of things (e.g. trying out some code). Consider using it for your other projects. If you don't like git, try hg, which has better GUIs. There is also a hg-git bridge.

Other topics

Random generator

(Pseudo-) random number generation is a very important topic and should be addressed. We use the default random generator from GSL. This can be influenced with environment variables, for example setting GSL_RANDOM_SEED and GSL_RNG_TYPE. See the GSL manual.

Only one random generator is used for the whole program, so setting the seed will not result in multiple, synchronized random generators.

Set a different seed for different runs, otherwise you will always obtain the same results!

For example, you can use a random number as the initial seed. If you use bash:

```
export GSL_RANDOM_SEED=$RANDOM
```

Mention in your publication that you set or varied the seed. Otherwise you may be victim to systematic errors!

Concluding remarks

None.

1.2.3 FAQ - Frequently Asked Questions

Contents

- *Is APEMoST free? How is APEMoST licensed? What are my obligations?*

  See the license.

- *What do I have to do in my publication?*

  State the version of APEMoST, and that you used it (see next question).

  Also double-check that your results make sense.

  Include error estimates in your publication.

  Mention that you set the PRNG seed.

  If you made modifications to APEMoST, contribute them back or publish them otherwise. If you don't, others can not check your results and uncover errors.

  You can publish your likelihood function, parameter and data file in the appendix of your paper. If you don't, others may have a hard time to reconstruct your results. You are also encouraged to send them to us. We will publish them on the sourceforge website so you can just put a link (that always stays the same) in your paper. This also allows us to show how APEMoST is used by others.

- *How do I cite APEMoST correctly?*

  A publication is in the works. Please write us a email, so we can notify you of the correct citation. In the meantime, you can use the website as a placeholder.

- *Can I run APEMoST on a Cluster/Grid?*

  Since Parallel Tempering requires the chains to intercompare and swap their status frequently, I estimated that a distributed computation would actually be slowed down by the synchronization overhead. This of course depends on how long it takes to evaluate your likelihood function, but with 300 data tuples and a 6-parametric model, we can have 1000 iterations per second (in each chain) on a single-core machine. With comparing chains every 100 iterations or so you can see that waiting for synchronisation over the network is problematic.

  Ideally, get a fast computer with many CPUs.

  You can also run the program independently on several machines, then paste the output (visited values) together, and analyze the combined runs.

  Other software packages use MPI for running on clusters.

- *Can I write my likelihood function in Fortran/C++/Python/R?*

  Quite possibly.

  For Python, check out PyAPEMoST.

  For Fortran/C++:

  - Link your program against libapemost and call set_function to tell APEMoST about your probability function.
  - Then mimic what generic_main.c does in your program (calling calibrate, run, analyse).

- *Can I run APEMoST on Linux/Unix/Mac?*

  Yes, all Unix derivates are the primary target of APEMoST.

- *Can I run APEMoST on Windows?*

We don't test APEMoST on Windows, but there is no reason you can't. You need the software APEMoST is based on, and the gcc compiler.

Alternatively, if you run into trouble, you can try working in a cygwin environment or run a Linux in a virtual machine.

- *How can I stay up to date?*

  The easiest to get notified is to contact us.

  You can follow the news on our project page and also the code changes.

- *My question has not been answered.*

  If you have a technical question, maybe the answer is in the manual?

  Otherwise feel free to contact us.

1.2.4 License

APEMoST is free and open source software. As all software using the GSL, it uses the GPLv3 (GNU Public License).

When using APEMoST, you are expected to

- Notify the authors, Johannes Buchner and Michael Gruberbauer, if you find any bugs
- Send modifications you make to the authors
- Cite APEMoST when you do a publication based on our work

We would also encourage you to send in the params file and C code you used in your publication, so we can make a neat gallery on how APEMoST is and can be used.

You can find `our contact here` (also for questions regarding the license).

Here is the license:

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.
```

```
  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.

                       TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.
```

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

```
  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.
```

```
    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
```

```
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.
```

```
  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.
```

```
  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
```

sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based. The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version. For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement). To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients. "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License. You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily

```
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
```

```
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
```

```
    under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

1.2.5 Links and other software

Contents

Introduction to MCMC and Bayesian inference

If you are unfamiliar with the concept, it can be quite overwhelming what it is all about, especially since there are many incomplete introductions. Here, we try to list several sources we consider good that cover the subject, so you can intercompare and reread in one what remains unclear in the other.

In one paragraph? Lets try. This is just to get a grasp on how the terms relate to each other.

> *You have a model that has several parameters, which should predict or describe an observation. An example of a model with three parameters (A, B, F) is "y(t) = A*sin(F*t + B)". It describes the relation between y and t.*
>
> *Assume now you can specify how likely it is that the real observation data has been observed under this model. This is called the likelihood function p(D\|M, I). Now we would need to evaluate this function everywhere in parameter space, which is simply unfeasible. We would need a method that evaluates there more densely, where the value is high. This is where Markov Chain Monte Carlo comes in (Metropolis algorithm, proposal distribution). With MCMC and the Bayesian theorem, you can not only tell what the best fitting values are, you also get the probability of each possible parameter value as a marginal distribution. And, you can compare the likelihood of one model to another (in the example above, we could add "+ C").*
>
> *Parallel Tempering, calibrations, different proposal distributions, adaptive MCMC are enhancements to improve convergence.*

Well, not quite one paragraph.

1. **A very good book on the subject:** *Bayesian logical data analysis for the physical sciences* by P.C. Gregory You can find it in Google Books and at the Cambridge Catalogue.
2. Then there is always place to promote my bachelor thesis :-)

I think it provides a pretty good overview of everything necessary. As a bonus, we detect pulsations on a star.

The tool has a different name in there, and model selection is not covered (which works very nicely on the pulsation example).

It is based on the work of Michael Gruberbauer.

3. Then there is always Wikipedia, although I found it hard to understand for novices.

   http://en.wikipedia.org/wiki/MCMC http://en.wikipedia.org/wiki/Bayesian_inference

4. Geyer has a good presentation on how MCMC works.

5. There are some astronomy-related papers

   Bayes in the sky: Bayesian inference and model selection in cosmology provides a good introduction and highlights cosmology as a good application for Bayesian inference with MCMC.

   and Applications of Bayesian model selection to cosmological parameters

6. ...

   If you know or have any good resources on the topic, you are welcome to `suggest links`.

Differences and other programs

Obviously, APEMoST is not the first program to do Bayesian inference or MCMC sampling.

Differences

We think it differs from existing programs by

- specifying and programming the model in a imperative language

  Although a declarative specification of the likelihood function has some beauty, we find a imperative specification to be easier to access, debug and faster to evaluate.

  The downside is that complex likelihood relations may not be expressible with APEMoST.

- not providing a GUI (so far)

  We provide a lean and mean command based structure. We think it is easier to understand and debug than other tools.

  The output (text files with double values) can be analysed with any plot program or any program written in any programming language. We do provide tools to analyse the output (e.g. creating histograms, etc.).

- a simple workflow

  The new user is not overwhelmed by dozens of features when [s]he does not have to be.

- Metropolis sampling

  Other programs usually use Gibbs Sampling, we use classic Metropolis sampling (usually without needing the Hastings extension, since the proposal distributions are normally symmetric).

- general purpose for data analysis

  We are no ‘’pure statistician’‘, but we also don’t just target Astronomy and Physics. All natural sciences are invited to use and extend our program.

- speed

  If another program converges twice as fast due to some sophisticated, experimental concept, but APEMoST runs 20 times as many iterations in the same time ...

Regarding speed we can say that APEMoST, running 2 million iterations, spends less than a minute outside the likelihood function. So, the bottleneck will definitely not be APEMoST, and C allows the most efficient implementation of your likelihood function.

Other programs

You are welcome to notify us if you know others!

- BUGS/WinBUGS/OpenBUGS/LinBUGS

  GUI-based; used by statisticians. Uses Gibbs sampling.

  The BUGS language is in wide use as declarative description of the model.

- JAGS http://calvin.iarc.fr/~martyn/software/jags/

  Similar to BUGS. Manual says it has poor performance.

- MCMC R package http://www.stat.umn.edu/geyer/mcmc/

  There is a mcmc package for the R project. Has very good slides of an introduction to MCMC sampling http://www.stat.umn.edu/geyer/mcmc/talk/mcmc.pdf

- BioBayes http://www.dcs.gla.ac.uk/biobayes/

  A Software Package for Bayesian Inference in Systems Biology

  Has a very nice, user-friendly GUI (Java). Has a very nice video presentation

- COSMOMC http://cosmologist.info/cosmomc/ http://cosmologist.info/notes/

  A mcmc program specialized to cosmological problems. Also uses the GSL. Has a nice presentation on what it is about.

- FBM Software for Flexible Bayesian Modeling http://www.cs.utoronto.ca/~radford/fbm.software.html

  ANSI-C

  "Flexible Bayesian models for regression and classifica- tion based on neural networks and Gaussian processes, and for probability density estimation using mixtures. Neural net training using early stopping is also sup- ported." "Markov chain Monte Carlo methods, and their appli- cations to Bayesian modeling, including implementations of Metropolis, hybrid Monte Carlo, slice sampling, and tempering methods. "

  This looks like the most similar approach (being ANSI-C). Looks very powerful and complete.

- less relevant software follows

- Bassist http://www.cs.helsinki.fi/research/fdk/bassist/

  generates C++ code: Bayesian model data -> posterior distribution of model parameters

- mrbayes http://mrbayes.csit.fsu.edu/

  bayesian inference with biology models (several discrete options)

- BEAST http://www.beastsoftware.org/

  modeling population models. Java

- YADAS http://www.ccs.lanl.gov/ccs6/yadas/

- There is a MCMC sampler written in Python, pyMC

- Programs that are not available for download are not listed (e.g. "bayesiananalysis", and the one from Do Kester)

Some notes: several software packages are abandoned since a few years.

I (Johannes) find it hard to get into the programs and to understand them. A toy example that works both in e.g. BUGS/JAGS, BioBayes, fbm and APEMoST would be great.

You can download the full documentation (PDF). Also available is the API reference (HTML) or (PDF).

1.2.6 News:

You can now access and write your likelihood functions in C, C++ and *Python*!

- Check out the PyAPEMoST project hosted together with PyMultiNest.

New Publication using APEMoST:

- "Gamma-ray bursts as cosmological probes: LambdaCDM vs. conformal gravity" Antonaldo Diaferio et al, 2011