MCMC_CLIB User Manual

Andrei Kramer

November 20, 2013

This manual lists how to set up an ODE model for parameter sampling using the MCMC_CLIB software package. We have provided very general example files, which illustrate all uses of the software package. In the provided example the data is given in arbitrary units and only makes sense in relation to a reference experiment. Since the reference data also depends on the unknown parameters, the ratio has to be taken every time the model is used in the likelihood function. The example shows how to model unknown but stable initial conditions of perturbation experiments. Note, that this manual does not provide documentation on the mathematical structure of the sampler. This has to be done in the given order. The provided Makefile might be useful when setting up a new model

Contents

1 Setup

This section provides all necessary commands to generate a shared library which can be loaded by MCMC_CLIB. The initial value problem for the given ODE model will be solved using CVODES. We use VFGEN [2] for this purpose.

1.1 Make Model `model.vf`

VFGEN takes an `xml` file as input and exports into many different formats including CVODES and MATLAB. The extension `.vf` is arbitrary.

We use the xml-element `Parameter` to define model parameters θ and inputs u. This list of parameters is ordered and the MCMC_CLIB software will assume that the first m appearing parameters are the unknown sampling parameters θ, where m is defined implicitely by setting a prior density in the MCMC_CLIB configuration file, see Section 1.4. The remaining parameters will be used as known input parameters. These inputs model the (known) experimental conditions, they differ from oen experiment to another.

To model output functions, we use the `Function` element, while fluxes can be defined using the `Expression` element.

1.2 Make cvodes model sources `model_cvs.{c,h}`

Export the model:

```
$ vfgen cvodes:sens=yes,func=yes model.vf
```

1.3 Make a Shared Library `model.so`

Compile the `vfgen` sources with GCC:

```
$ gcc -shared -fPIC -O2 -Wall -o model.so  model_cvs.c
```

1.4 Write a Configuration File `data.cfg`

The configuration file is a plain text file; it contains some mandatory and some optional elements. Optional elements can be set for convenience, the same elements can be set using command line arguments. Command line arguments take precedence over the entries in the configuration file.

The mandatory elements concern the data, the experimental conditions under which the data was recorded and the prior knowledge about the parameters. Most of the mandatory elements can be described as blocks of numerical entries or matrices. This type of entry is done using tags of the form `[entry_type][/entry_type]`:

`[time]` a row of timepoints at which measurements have occured. This array has a meaning for all state variables. If some outputs were not observed at some time instance, this is noted in the `data` tag. The closing tag is `[/time]`.

`[reference_input]` this block contains a row of known input parameters which define the model conditions corresponding to the reference experiment. If this block is present, then the experiment is considered relative. The closing tag is `[/reference_input]` and similar for the following.

`[input]` contains one row per experiment, with columns corresponding to the input parameters.

`[reference_data]` block containing the measurements for the reference experiment; has one row per time point. columns represent output functions.

`[data]` The data block can take data from any number of performed experiments. Each experiment is associated with an input vector. Columns represent the output functions defined in the `vf` file. Rows represent measurement time points and experiment conditions (see Table 1): measurements observed at timepoint t_i ($i = 0, \ldots, T-1$) for input u_j ($j = 0, \ldots, N-1$) will be in row $k = jT + i$ ($k = 0, \ldots, NT-1$). Example, with T time points and N experimental conditions, we get:

line	input row	timepoint
1	1	1
2	1	2
⋮	1	⋮
	1	T
	2	1
	2	2
	2	⋮
	2	T
	⋮	⋮
NT	N	T

Table 1: Here we used line numbering starting at 1; we get NT lines for N inputs (experiments) and T timepoints at which measurements have occurred. Each line contains a row of data points. If a measurement was omitted you can enter any placeholder in the omitted slot and set the standard diviation of this point to `inf` (∞).

`[sd_data]` This block is of the same size and structure as the data block and contains the standard deviations (uncertainties) of the data. Enter `inf` for any omitted measurement.

`[prior_mu]` The prior is assumed to be log-normal on all unknown model parameters. Since sampling is done in logarithmic space, the prior is a multivariate normal $\mathcal{N}(\mu, \Sigma)$ for the actual sampling variables. This block contains one column; it represents μ and has one row per unknown parameter. It is very important to set μ to values that make sense in *logarithmic* space.

`[prior_inverse_cov]` this tag contains the Σ parameter of the Gaussian prior. It can also be written like this: `[prior_inverse_covariance]`.

For convenience, you can set some of the command line options also as variables inside the configuration file. Some variables can only be set here. The follwong table lists all variables with example values:

variable name=value	meaning
`step_size=0.001`	Sampling algorithm's initial step size
`sample_size=1000000`	`Integer`, sample size after warmup
`acceptance=0.50`	Target acceptance for tuning.
`output=./sample/model_Y.double`	sample will be writen to this file.
`t0=-72`	t_0 for the initial conditions $x(t_0) = x_0$.

Setting t_0 to a large negative value can be used to model experiments which start in a stable but unknown equilibrium state of the model. While the input function inside the model (`vf` file) can be set to perform a perturbation at $t = 0$.

2 Run Simulation

The program has some command line options which can be shown by typing:

```
$ ./ode_smmala -h
Usage: -c ./data.cfg
                        data.cfg contains the data points and the
                        conditions of measurement.

-l ./ode_model.so
                        ode_model.so is a shared library containing
                        the CVODE functions of the model.

-s $N
                        $N sample size. default N=10.

-o ./output_file
                        file for output. Output can be binary.

-b
                        output mode: binary.

-a $a
                        target acceptance value (markov chain will
                        be tuned for this acceptance).

--seed $seed
                set the gsl pseudo random number generator seed
                to $seed.
```

An example of how to run the program is included in the package: `run_test.sh`.

```
#!/bin/bash
```

```
# start sampler with
# -b                          binary output
# -l ODEmodel11S26P4U.so      shared library of model
# -c ODEmodel11S26P4U.cfg     configurations
Model=ODEmodel11S26P4U
SampleFile="sample/${Model}_`date␣+%Y-%m-%dT%Hh%Mm`.double"

cat<<EOF
$0
 redirecting standard output to $Model.out
 redirecting standard error  to $Model.err

 output will be binary (${SampleFile})
 to load the sample in matlab or GNU Octave:

 NumOfParameters = 26;
      SampleSize = 1e6;
             fid = fopen('${SampleFile}','r');
          Sample = fread(fid,[NumOfParameters+1,SampleSize],'
             double');
                   fclose(fid);

 use <<tailf $Model.out>> to see progress (press <<Ctrl-C>> to exit
    tailf)
 redirecting standard output to $Model.out
             standard error  to $Model.err
 sampling now ...
EOF

bin/ode_smmala  -b -o $SampleFile -l $Model.so -c $Model.cfg 1>
   $Model.out 2> $Model.err
```

3 Sample Analysis

The output will be logarithmic and will contain the log posterior for each parameter vector. The sample can be read and processed using numerical computation software like GNU Octave [1] or MATLAB:

```
fid=fopen('sample.double','r');
Sample=fread(fid,[NumberOfParameters+1,SampleSize],'double');
fclose(fid);
```

To obtain model trajectories the parameter values have to be transformed:

```
LogPosteriorIndex=NumberOfParameters+1;
[~,Imax]=max(Sample(LogPosteriorIndex,:));
MaxLikelihoodParas=exp(Sample(1:NumberOfParameters,Imax));
```

References

[1] Octave community. *GNU/Octave. Octave,Software*. 2012. URL: www.gnu.org/software/octave/.

[2] Warren Weckesser. *VFGEN*. 2013. URL: http://www.warrenweckesser.net/vfgen/index.html.