`ode_smmala`

MCMC_CLIB User Manual

Andrei Kramer

November 29, 2016

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. This manual does not provide only the necessary documentation on the mathematical background.

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. Alongside the ODE model, the user also supplies data, and information about how it was obtained (sort of). Data often has to be normalised to some reference signal. In many cases, a full quantification is too expensive, so two (relative) signals are measured. They are meaningful in relation to each other, but not on their own. In systems biology, this is the typical case. We broadly distinguish between three cases: (a) a control experiment was performed (system was in nominal state), (b) a time series was obtained, where the relative changes between the points are meaningful, and (c) a time series of several output signals was obtained and normalisation happens between those signals at specified time points. Relative data is often represented in the form of percentages in publications (not in mol or other physical units).

We consider the following system:

$$\dot{x} = f(x, t; \rho, u)\,, \qquad x(t_0; \rho, u) = x_0\,, \tag{1}$$

$$y_{ijk} \stackrel{!}{=} h_i(x(t_j; \rho, u_k) + \varkappa_{ijk}\,, \qquad \varkappa \sim \mathcal{N}(0, \sigma_{ijk})\,, \tag{2}$$

where, ρ are the model parameters und u are input parameters. A dynamic input signal can be modeled within f and is not made explicit here. Notation: the semicolon is used to separate primary arguments from parameters (contsant in one given simulation). Currently, the initial conditions may not depend on the parameters or inputs.

The output functions h have access to one model simulation result, while normalisation occurs between two simulations. Output functions are specified in the model, normalisation is performed by MCMC_CLIB internally. The user merely supplies the index structure of the normalisation.

1.1 Make Model `model.{vf,xml}`

VFGEN takes an `xml` file as input and exports into many different formats including CVODES and MATLAB. The extension `.vf` is arbitrary. Any other tool that exports the model into a CVODES compatible header and source file is of course just as good. VFGEN performs the necessary analytical computations for the Jacobian and parameter-Jacobian.

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 on the following page. The remaining parameters will be used as known input parameters. These inputs model the (known) experimental conditions, they differ from one 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 `sd_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 u.

`[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 ($j = 0, \ldots, C-1$) will be in row $k = jT + i$

($k = 0, \ldots, CT - 1$). For 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 CT lines for C 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]`.

`[norm_t]` If you omit the `reference_data` tag because you have an uncontrolled, relative time series, you can specify a time point for each experiment, where normalisation for all the measured outputs shall occur. In the simplest case, this is a column vector with one line per experiment, giving the index of normalisation time. Both `norm_t` and `norm_f` are explained in more detail in Section 2

`[norm_f]` Sometimes, it makes sense to normalise the outputs between each other, within the same experiment. In that case, `norm_t` must contain $C \times F$ elements (normalisation times), while `norm_f` specifies by which output function index a given output function is to be normalised.

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 Time Series Normalisation

It is possible to specify a rather complex normalisation scheme using the elements `[norm_f]` and `[norm_t]`. Let us assume the following as the most general time series normalisation procedure

$$\texttt{normalised}\tilde{y}_{i,j,b} := \frac{h_i(t_j; \rho, u_b)}{h_{l(i,b)}(t_{k(i,b)}; \rho, u_b)} = \frac{y_{ijb}}{y_{lkb}}, \tag{3}$$

each output y is normalised by dividing the output function h at a fixed time point t_k of some other output function h_l, where k and l are different for each output channel i and experiment b.

An example configuration could look like this:

```
[norm_t]
2
[/norm_t]
```

This represents the following normalisation:

$$y_{ijk} \leftarrow \frac{y_{ijk}}{y_{i2k}} \tag{4}$$

If a different normalisation point is used for the different inputs, this notation applies:

```
[norm_t]
2 # Experiment 0
0 # Experiment 1
2 # Experiment 2
5 # Experiment 3
[/norm_t]
```

This defines 4 experiments, where in experiment 0, y is normalised at t_2, while while y_2 is normalised at t_0, the first measurement instance.

Finally, the most complex form of normalisation can be specified like this:

```
[norm_f]
0 1 2 3 4 5 # This is ordered just like the data would be
1 1 1 1 4 5 # and specifies the data index to normalise with.
2 2 2 2 2 2 # Sometimes it is the same for all data-points.
4 4 4 4 4 4 # 4 experiments by 6 outputs
[/norm_f]

[norm_t]
2 2 2 2 2 2 # This is also ordered just like the data
3 3 3 3 2 2 # but specifies the time index to normalise at.
3 3 3 3 3 3 # Each measurement time t is normalised as any other
3 3 3 3 3 3 # so this is just a C by F matrix again.
[/norm_t]
```

Here, we have 6 output functions and $C = 4$ experiments, no reference experiment. The table lists the necessary k and l index for each output function:

$$\tilde{y}_{i,j,b} := \frac{y_{ijb}}{y_{lkb}}, \tag{5}$$

$$\texttt{norm\_f}_{b,i} = l(i,b), \tag{6}$$

$$\texttt{norm\_t}_{b,i} = k(i,b), \tag{7}$$

where l is the output function we normalise with and k is the time index we normalise at. Each line represents a different experiment in the data set. We can infer the following information from the given example:

$$\begin{array}{rcl}
h_0(t;\rho,u_0) & \text{is normalised by} & h_0(t_2;\rho,u_0) \\
h_1(t;\rho,u_0) & \text{is normalised by} & h_1(t_2;\rho,u_0) \\
h_2(t;\rho,u_0) & \text{is normalised by} & h_2(t_2;\rho,u_0) \\
\vdots & \vdots & \vdots \\
h_0(t;\rho,u_1) & \text{is normalised by} & h_1(t_3;\rho,u_1) \\
h_1(t;\rho,u_1) & \text{is normalised by} & h_1(t_3;\rho,u_1) \\
\vdots & \vdots & \vdots \\
h_4(t;\rho,u_1) & \text{is normalised by} & h_4(t_2;\rho,u_1) \\
\vdots & \vdots & \vdots \\
h_5(t;\rho,u_4) & \text{is normalised by} & h_4(t_3;\rho,u_1)
\end{array}$$

3 Run Simulation

The program has some command line options, you can inspect the full list by requesting help: `./ode_smmala -h`

```
~/mcmc_clib $ ./bin/ode_smmala -h
Usage:
```

```
-c ./data.cfg
                        data.cfg contains the data points and the conditions of measu

-l ./ode_model.so
                        ode_model.so is a shared library containing the CVODE functic

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

-r, --resume
                        resume from last sampled MCMC point. Only the last MCMC posit
-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

--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
```

4 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.