MODEST MCMC TUTORIAL

Antti Solonen

Version 0.3, 01/2008

Laboratory of Applied Mathematics, Lappeenranta University of Technology

1 Introduction and example model

MCMC (Markov Chain Monte Carlo) methodology provides a Bayesian approach to the nonlinear parameter estimation task. With MCMC it is possible to examine the distribution of unknown parameters in nonlinear models, whereas traditional fitting techniques only produces single estimates for the parameters. For linear models, the distribution of the parameters can be given as an exact formula, but for nonlinear models numerical techniques (such as MCMC) have to be employed.

The shape and size of the distribution of the parameters give valuable information to the modeler, related to correlations, uncertainty and identifiability of parameters, for example. It is also possible to derive the distribution for the response curves of the model - instead of one "fit" we get an "area" where the model prediction lies with certain probability.

This document explains how the Modest MCMC Package works and what different options it includes. The package is an extension to the Modest statistical software (refer to [1] for details), written in Fortran90. The reader is assumed to have a basic understanding about Bayesian statistics, MCMC methods and parameter estimation in nonlinear models. A brief introduction is given below. Also an example model that is used to demonstrate the package is given. For more on the topic, refer to [2], for example.

1.1 Model Parameters in Bayesian Terms

The general form of the mathematical models discussed here is given as two equations:

$$s = f(x,\theta) \tag{1}$$
$$y = g(s) + \epsilon \tag{2}$$

where s is the model *state*, x are the design variables and θ are the unknown parameters in the model. The observation function g maps the model state to the *response* y that is observed. The error term ϵ represents the *measurement error.*

In Bayesian statistics, the goal is to find the *posterior distribution* of parameters θ given measurements y, denoted by $\pi(\theta|y)$. The posterior distribution is defined using the Bayes' rule

$$\pi(\theta|y) = \frac{L(y|\theta)p(\theta)}{\int L(y|\theta)p(\theta)d\theta} \tag{3}$$

where L is the *likelihood* and p is the *prior.* The likelihood here is defined by assuming independent and identically distributed Gaussian errors for N observations y_i:

$$L(y|\theta) = \prod_{i=1}^{N} L(y_i|\theta) = (2\pi\sigma^2)^{-n/2} \exp(-\frac{1}{2\sigma^2} SS(\theta)).$$

The sum of squares $SS(\theta) = \sum_{i=1}^{N} (y_i - g(f(x_i, \theta)))^2$ is a measure of a distance from model to observations. In traditional parameter estimation, this is minimized with some iterative optimization method.

With MCMC methods, a set of samples from the posterior distribution $\pi(\theta|y)$ is created. Thus, MCMC will find many parameter values with which the model fits the data with the accuracy of the measurement error.

1.2 Example Model: Chemical Kinetics

The MCMC package is demonstrated using a simple chemical reaction model, described as an ODE system below.

$$A \xrightarrow{k_1} B \xrightarrow{k_2}$$

$$\begin{aligned} \frac{dA}{dt} &= -k_1 A \\ \frac{dB}{dt} &= k_1 A - k_2 B \end{aligned}$$

where k_1 and k_2 are reaction rates. The temperature dependency of the reaction rate is expressed in chemical kinetics using the Arrhenius' law, in which

$$\begin{aligned} k_1 &= k_1^{mean} e^{-E_1 z} \\ k_2 &= k_2^{mean} e^{-E_2 z} \\ z &= 1/R(1/T - 1/T_{mean}) \end{aligned}$$

The frequency factors k_i^{mean} denote the average reaction rate is some average temperature T_{mean}. R is the general gas constant and E_1 and E_2 denote the activation energies for the reaction.

The estimated parameters here are $\theta = (k_1^{mean}, E_1, k_1^{mean}, E_2)$. T denotes the temperature in which the measurements are made. We have measurements for the components A and B in two different temperatures.

2 Modest MCMC Package

The MCMC run for unknown model parameters is initiated after an estimation task in Modest (see [1]). That is, we have the following files (see appendix 1 for the files).

FILE	DESCRIPTION
boxom.f	Model code, here the ODE function that is feeded into an ODE solver, describes the function f in equation 1.
boxoi.f	Initial assignments
boxoo.f	States that are observed, defines the function g in equation 2.
boxo.nml	Nml-file describing the estimation task

In addition, we can specify another nml-file called *mcmcinit.nml* that specifies the options for the MCMC run. The MCMC options can also be defined in the model nml-file (here *boxo.nml*). The complete file with all usable options is given in Appendix 1 with the default values for the options. Note that only the options that differ from the default values need to be specified in the mcmc namelist. The options are listed and explained in detail in this chapter.

The fortran files are compiled to an executable with links to Modest and MCMC libraries.

2.1 GENERAL OPTIONS

NAME [DEFAULT]	DESCRIPTION
nsimu [0]	Length of the MCMC chain: how many samples from the distribution of the parameters are created (0: no MCMC run is made)
printint [1000]	Interval to print statistics
filepars [0]	Binary: 0 means reading initial values for parameters from Modest, 1 means reading initial values from files. See below for a description about how to give initial values in files.
priorsfile [”]	File from which Gaussian priors for the parameters are read (0: no prior). See below for more details about assigning priors.
chainfile [’chain.mat’]	MAT-file where the MCMC chain is written (samples given in rows, variables in columns)
ssfile [’sschain.mat’]	MAT-file where the sums of squares for created samples are written
sstype [1]	The method for calculating the sum of squares in the likelihood. 1: normal sum of squares, 2: logarithmic scale. See below for description.
sstrans [1.0]	Uses transformation $ss = \sum_i (y_i^{sstrans} - f_i^{sstrans})^2$ where y is data and f is model prediction. Applied only if sstype=1.

If **filepar** is set to 1, Modest MCMC tries to read initial parameter values from file *mcmcpar.dat* and the initial covariance from file *mcmccov.dat*.

If **priorsfile** is defined, the corresponding file is opened and Gaussian prior parameters for the unknowns are read from it. In priorsfile, the first row specifies the prior means and second row the prior variances. Other priors can be specified by editing the *priorratio* subroutine in the *prior.f* source file.

The **sstype** and **sstrans** parameters describes how the sum of squares is calculated and how the error term is treated. If sstype=1, we use additive Gaussian iid error model and calculate $ss = \Sigma_i (y_i - f_i)^2$. If sstype=2, the noise is assumed to be lognormal and the sum of squares is calculated in a logarithmic scale: $ss = \sum_i (\log y_i - \log f_i)^2$.

2.2 BURN-IN AND INITIALIZATION

NAME [DEFAULT]	DESCRIPTION
doburnin [0]	Binary, 0: do not alter the proposal distribution during the burn-in phase, 1: allow different tricks during the burn-in
burnintime [0]	Length of the burn-in period (0: no burn-in)
scalelimit [0.05]	If acceptance ratio ($\alpha <$ scalelimit) or if ($\alpha > 1$ - scalelimit) scale the proposal distribution down or up respectively by dividing / multiplying with *scalefactor*. This improves mixing if the proposal covariance is too large or small. 0: no scaling. Scaling is done only during the burn-in.
scalefactor [2.5]	Factor used in proposal scaling during the burn-in
greedy [1]	Binary, 1: use *greedy adaptation*, where the proposal covariance is updated using only accepted points during burn-in (0: no greedy adaptation). This helps in getting the adaptive sampler moving. See adaptive methods' options for adaptation details.
badaptint [-1]	Adaptation interval used during the burn-in phase. Negative value uses the value given in *adaptint*. See adaptive methods' options for adaptation details.
condmaxini [1.0]	Regularizes a singular initial covariance in a way described below.

The burn-in period is an initial period of the MCMC run which is discarded when the final analyses are made from the MCMC output. Burn-in is used in order to give the sampler time to move to the region where the distribution has high probability.

Initial proposal covariance is calculated in Modest MCMC using linearization: $C = (J^T J)^{-1}$ where J is the Jacobian matrix calculated at the initial point. If $J^T J$ is singular, we have to regularize it to be able to calculate the proposal covariance. This is done in Modest MCMC by looking at the condition number of $J^T J$, which is the ratio between the largest and smallest singular value of the matrix. The singular values are sought using the singular value decomposition, where $J^T J = USV^T$, S containing the singular values in decreasing order in the diagonal. If ($cond(J^T J) >$ condmaxini), the smallest singular value is set to be ($s_{end} = s_1$/condmaxini), thus forcing the condition number to condmaxini, that assures that C can be calculated.

The calculation of the linearization $C = (J^T J)^{-1}$ can be avoided by giving the initial proposal covariance to MODEST by hand in file *mcmccov.dat* and setting filepars=1.

2.3 ADVANCED MCMC METHODS

NAME [DEFAULT]	DESCRIPTION
doadapt [1]	Binary, 1: use the adaptive MCMC methods, 0: no adaptations. See below for a brief introduction to adaptive MCMC methods.
adaptint [1000]	Adaptation interval in the adaptive algorithms (0: do not use adaptation).
adapthist [0]	History length in the Adaptive Proposal (AP) algorithm (0: use the whole chain). See below for description about AP.
adaptend [0]	Step when the adaptation is stopped (0: do adaptation during the whole run)
initcmatn [0]	Describes how we take the initial covariance matrix into account in the adaptation. 0: discard the initial covariance when the first adaptation is made, > 0: consider the initial covariance as if it had been calculated from *initcmatn* points. The larger the value is, the more we trust the initial covariance and the less effect the adaptation has on the proposal.
condmax [0.0]	Regularizes a singular proposal covariance calculated during adaptation in the same way as in *condmaxini* described above
drscale [0.0]	Second stage Delayed Rejection (DR) proposal is formed by scaling the original proposal by dividing with this (0.0: do not use DR). See below for more about the DR algorithm.

Adaptive MCMC methods use the history of the sampling process (the chain created so far) to update the proposal distribution during the computation. The Modest MCMC package implements three adaptive algorithms: the Adaptive Proposal (AP), Adaptive Metropolis (AM) and Delayed Rejection

Adaptive Metropolis (DRAM). The **adaptint** tells how often the adaptation is done (by default at every 1000th step). If **adapthist**=0, the AM algorithm, where the proposal covariance is calculated from the whole chain calculated so far, is used. Otherwise the AP algorithm is used, where the covariance is calculated using a fixed number of previously created points. Note that only the AM algorithm can be proven to asymptotically result to the correct distribution.

The Delayed Rejection (DR) method, tuned with the **drscale** option, can be used alone or with adaptation (DRAM). The idea of DR is that if we reject a point during sampling we propose another move (2nd stage proposal) from another proposal distribution. In Modest MCMC this 2nd stage proposal distribution is just a scaled version of the original proposal: it is formed by dividing the original proposal covariance with drscale. This often improves the estimates and helps in getting the sampler moving.

2.4 ERROR VARIANCE SAMPLING AND PREDICTIVE CURVES

NAME [DEFAULT]	DESCRIPTION
updatesigma [0]	Binary, 0: no error variance sampling, 1: use conjugate prior and sample the error variance using Gibbs sampling (see below for details)
N0 [1.0]	Conjugate prior parameter for error variance sampling (see below), applicable if updatesigma=1
S02 [0.0]	Conjugate prior parameter for error variance sampling (see below), applicable if updatesigma=1
s2file ['s2chain.mat']	MAT-file to save the sampled error variances for each measured component, applicable if updatesigma=1
dumpint [0]	Every *dumpint*:th model prediction is saved into a file called *[projectname].dmp*. This is needed in plotting predictive curves (see chapter 3). If updatesigma=1, also the corresponding error variances are saved into a file called *[s2file].dmp*

Assuming that the measurement error is Gaussian and measurements in different points are independent and identically distributed, the likelihood of the unknown parameters θ gets a Gaussian form:

$$p(\mathbf{y}|\theta) = (2\pi\sigma^2)^{-n/2} e^{-0.5\sigma^{-2}SS_\theta}$$

where σ^2 is the measurement error variance. Normally σ^2 is thought to be fixed during the MCMC run and it is estimated from the residuals, for example. However, the error variance can also be treated as a random variable - we can let the variance "float" and sample it along with the parameters. By assigning a conjugate prior for the variance, we can write the conditional distribution for the variance (given the model parameters) as the inverse Gamma distribution (for details, see [2]):

$$\sigma^2|(\theta, y) \sim \Gamma^{-1}\left(\frac{n_0+n}{2}, \frac{n_0 S_0^2 + SS_\theta}{2}\right).$$

This is done if **updatesigma** is set to 1. In that case, the parameters **N0** and **S02** (n_0 and S_0^2 in the formulas) must be assigned. The parameter n_0 can be thought to represent the number of observations equivalent to the information given by the prior and S_0^2 represents the prior error variance. The higher values n_0 gets, the more we trust our prior variance S_0^2 (see figure 2.4). Making n_0 smaller allows larger variation for the error variance. If **S02**=0.0 is set in Modest, the mean squared error (MSE) is used as S02.

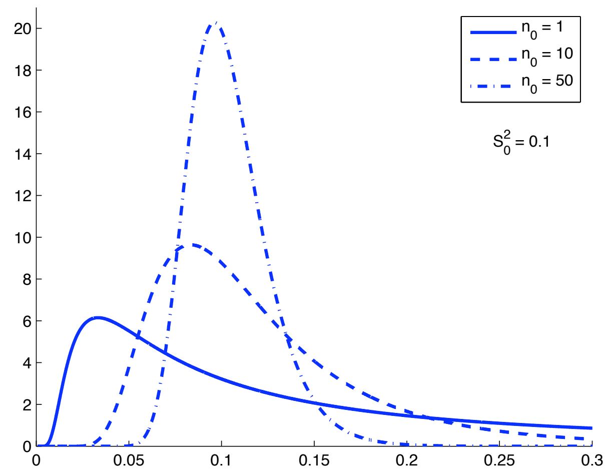

Figure 1: Priors for error variance with different parameters n_0 and S_0^2

3 MATLAB Functions

In this section a few MATLAB functions for examining the output of the MCMC run are presented. First the plotting function **mcmcplot** is introduced, with which one can plot the sampled parameter values. Then the functions for creating predictive distribution plots are discussed.

3.1 Plotting the Parameters

To examine the output of the MCMC algorithm, we can use the **mcmcplot** function in MATLAB. Many types of parameter plots can be created. The general steps for plotting the sampled parameters are as follows.

1. RUN MODEST MCMC to produce *chain.mat*.
2. RUN **chain=loadchain('chain.mat')** to load the created chain into the MATLAB environment (see "help loadchain").
3. RUN **mcmcplot(chain,inds,names,plottype, ...)** to produce the plots. The parameter *inds* denotes the parameter index vector (which parameters are plotted), *names* contains the parameter names as strings and *plottype* gives the type of the plot produced. If *inds*=[], all parameters are plotted and if *names*=[], no names are printed to plots.

The type of the plot is defined by the **plottype** parameter. The possible options with descriptions are listed below.

- **plottype='pairs'** gives the pairwise scatter plots for the parameters, where every parameter is plotted against all other parameters. In addition, one-dimensional kernel density estimates and confidence regions (eg. 95%) based on two-dimensional kernel density estimation can be plotted. The kernel density estimates are controlled with optional parameters *smo* and *rho* given to the mcmcplot function (in this order). The parameter *rho* defines the correlation used in kernel density estimation (default value is the correlation coefficient calculated from the points). With the parameter *smo* (kernel variance) it is possible to adjust the smoothness of the one and two dimensional density plots - the greater the value, the smoother densities and confidence regions we get and vice versa. If *rho*=*smo*=0 is given, no density estimates are calculated and plotted. See figure 3.1.

- **plottype='dens'** gives the one-dimensional kernel density estimates for the parameters one by one. The kernel variance *smo* can be given as an optional parameter. If it is not given, it is calculated from the sampled points.

- **plottype='chain'** plots the sample paths of each variable and a lowess smoothed curve for the points.

- **plottype='hist'** plots the histograms for each variable. The number of bins can be given as an optional parameter (default is 20 bins).

- **plottype='acf'** gives the autocorrelation plots for parameters. Autocorrelation gives the average correlation between members that are a certain steplength away from each other. From the autocorrelation plot one can approximately read how the chain should be thinned (only every ith value taken from the chain) in order to get independent samples.

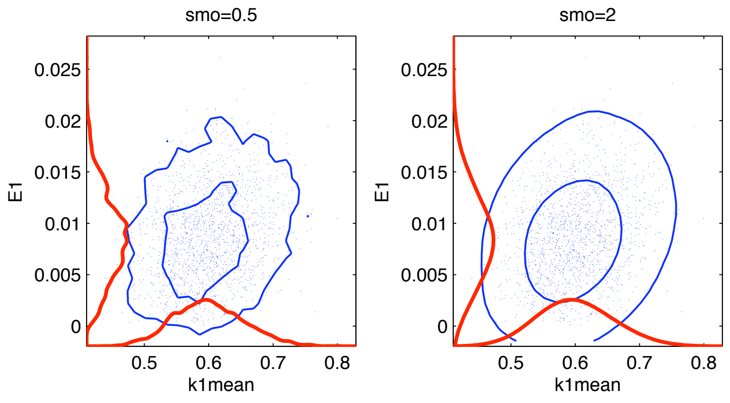

Figure 2: Pairwise scatter plot for two parameters with different smoothing factors. Produced by *boxoplots.m* script given in Appendix 1.

3.2 Predictive distributions

Besides the parameter distributions, we might be interested in how the uncertainty in parameters affects the model prediction. To form these so called predictive distributions, we can use the functions **getobs**, **dmpfile** and **mcmcpredplot** in MATLAB.

In the plots two kind of "areas" are plotted around the most probable response curve. First of all, we simulate the model response with the sampled parameter values, and form a confidence interval for the response at certain points in the x-axis. In the plots this area is plotted with darker grey color and this area represents the area where the model prediction curve lies with a certain probability.

Secondly, we add our estimate of the measurement error to the simulated responses to produce "noisy responses". Then we similarly form a confidence interval at certain points in the x-axis for these responses, and fill the area with a lighter gray color. This represents the area from which the observations (current and forthcoming) can be found with a certain probability. If *updatesigma=0* is set, we use a fixed error variance for each sampled response. If *updatesigma=1*, the error variance is sampled separately for every point in the MCMC chain.

For plotting the predictive curves, the following steps have to be carried through. See figure 3.2 for an example.

1. RUN MODEST MCMC with a specified **dumpint** to produce *chain.mat* and *boxo.dmp*. If updatesigma=1, *s2chain.mat* and *s2chain.dmp* are produced. Also *boxo.est* file is produced (name defined in the model nml file *boxo.nml*).
2. RUN **out=dmpfile('boxo.dmp','s2chain.dmp',options)**. If updatesigma=0, an empty string must be given instead of *s2chain.dmp.* The options structure is explained below.

NAME [DEFAULT]	DESCRIPTION
options.plotmode [1]	Either a string or a numeric value: numeric value corresponds to *sstrans* in MCMC options, 'normal' corresponds to sstype=1 and 'lognormal' corresponds to sstype=2.
options.lims	The calculated confidence limits, by default [0.025 0.5 0.975] which means 50% and 95% intervals
options.nn [1]	Number of noisy curves calculated per each prediction
options.negallowed [1]	1: negative predictions allowed, 0: negative predictions cut to zero

3. RUN **obs=getobs('boxo.est')** to get the observations (if you want to plot them).

4. RUN **mcmcpredplots(out,obs,options)**. If obs=[], no observations are plotted. The options structure is explained below.

NAME [DEFAULT]	DESCRIPTION
options.datas [1:nbatch]	Indices for selecting which batches are plotted (default: all batches). A new figure is created for every batch.
options.states [1:ns]	Indices for selecting which states are plotted (default: all states). The states are plotted one below the other.
options.responses [1:ny]	Indices for selecting which responses (observed components) are plotted (default: all responses). Responses and states are plotted in different figures.

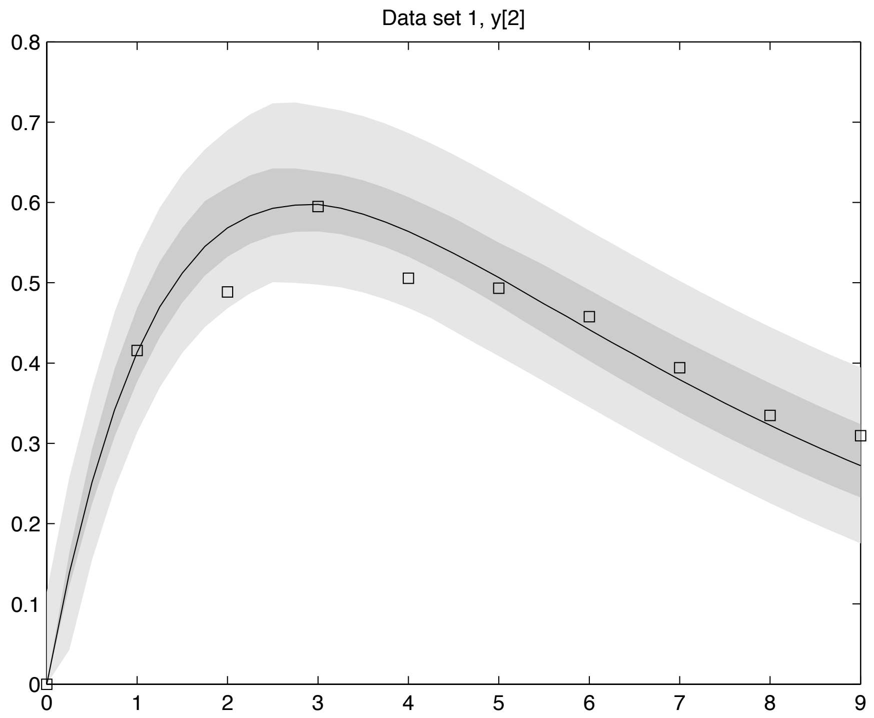

Figure 3: Predictive distribution plot for B in the example model. Produced with *boxoplot.m* script given in Appendix 1.

4 Appendix - Files

4.1 boxom.f

```
subroutine fode(ns,t,s,ds,
     &           xdata,nx,nobs,
     &           nsaux,nstatea,
     &           states0,
     &           gpar,ngpar,
     &           lpar,nlpar,
     &           iobs,iset)

      implicit none

c     arguments

       integer*4 ns,nsaux,nstatea    !n of state variables
       real*8    t           !time
       real*8    s(nstatea) !state variables
       real*8    ds(ns)      !derivatives
       integer*4 nx,nobs,ngpar,nlpar
       real*8    xdata(nx,nobs)
       real*8    states0(nstatea)
       real*8    gpar(ngpar)
       real*8    lpar(nlpar)
       integer*4 iobs,iset

c     local variables (all user defined variables must be declared here!)
      integer*4 i
      real*8 A,B,R,z,k1,k2

      include 'boxo.inc'

c     user code:

      A = s(1)
      B = s(2)

      e1 = e1*1.0d+6
      e2 = e2*1.0d+6
```

```
R  =  8.314
z  = 1.0d0/R * ( 1.0d0/Temp - 1.0d0/Tmean)
k1 = k1mean * dexp(-e1*z)
k2 = k2mean * dexp(-e2*z)

ds(1) = - k1 * A
ds(2) =   k1 * A - k2 * B

return
end
```

4.2 boxoo.f

```
subroutine observations(s,ns,yest,ny,
     &           xdata,nx,nobs,
     &           nsaux,nstatea,
     &           states0,
     &           gpar,ngpar,
     &           lpar,nlpar,
     &           iobs,iset)

      implicit none

c     arguments

       integer*4 ns,nsaux,nstatea    !n of state variables
       real*8    s(nstatea) !state variables
       integer*4 ny          !n of obs. vars
       real*8    yest(ny)    !observed variables

       integer*4 nx,nobs,ngpar,nlpar
       real*8    xdata(nx,nobs)
       real*8    states0(nstatea)
       real*8    gpar(ngpar)
       real*8    lpar(nlpar)
       integer*4 iobs,iset

c local variables (all user defined variables must be declared here!)
       integer*4 i                            ! loop indices
       include 'boxo.inc'

c      user code:

         do i=1,ny
             yest(i) = s(i)
         enddo

       return
       end
```

4.3 boxoi.f

```
      subroutine inits0(ns,t,s,
     &           xdata,nx,nobs,
     &           nsaux,nstatea,
     &           states0,
     &           gpar,ngpar,
     &           lpar,nlpar,
     &           iobs,iset)

      implicit none

c     arguments

       integer*4 ns,nsaux,nstatea   !n of state variables
       real*8    t           !time
       real*8    s(nstatea) !state variables

       integer*4 nx,nobs,ngpar,nlpar
       real*8    xdata(nx,nobs)
       real*8    states0(ns)
       real*8    gpar(ngpar)
       real*8    lpar(nlpar)
       integer*4 iobs,iset

c  local variables (all user defined variables must be declared here!)

       integer i
       include 'boxo.inc'

c  user code:

       do i = 1,ns
         s(i) = states0(i)
       enddo

       t = xdata(1,1)

       return
       end
```

4.4 boxo.nml

```
&project
  projectname = 'boxoest'
/


&files
 nsets = 2
 datafile(1) = 'boxodata.dat',
 resultfile  = 'boxo.sta'
 estfile = 'boxo.est'
/

&problem
 task  = 'est'
 model = 'ode'
 odesolver = 'odessa'
 optimizer = 'simflex'
 objfun = 'lsq'
/

&modelpar
 nstates = 2

 modelvar = 'k1mean   global   1.0
             E1       global   0.01
             k2mean   global   1.0
             E2       global   0.01
             Tmean    global   300.
             Temp     local    283.   313.;
             time     Odevar   0 file;
             s0(1:2)  initval  1.0;
                               0.0; '

  target  = 'k1mean       0.01  100.
             E1           1.e-4  1.0
             k2mean       0.01  100.
             E2           1.e-4  1.0'
```

```
/

&filepar
 combined = 1
 nobs(1)   = 18
 ncolxy(1) = 3
 nydata(1) = 2
 indx(1,1) = 1
 indy(1,1) = 2,3
/

&print
 echo = 1
 echodata = 1
 optmonit = 1
 stats = 1
 debug = 0
 jacout = 1
/

&simflex
 abstols = 1.00E-08
 reltols = 1.00E-08
 sizes = .1
 itmaxs = 100
/
```

4.5 mcmcinit.nml

```
! mcmcinit.nml -- initialization for modest mcmc
!
&mcmc

! GENERAL OPTIONS
 nsimu       = 0   ! length of the chain
 printint    = 500    ! interval to print statistics
 filepars    = 0       ! read initial values from files instead of modest
 priorsfile  = ''      ! file from which prior parameters are read
 chainfile   = 'chain.mat'    ! file to save the chain
 ssfile      = 'sschain.mat'  ! file to save sum-of-squares chain
 sstype      = 1       ! 1=Gaussian , 2=lognormal, 3=Poisson
 sstrans     = 1.0     ! ss is  (y**sstrans-f**sstrans)**2, if sstype = 1

! BURN-IN AND INITIALIZATION
 doburnin    = 0       ! do we have 'burn-in'
 burnintime  = 0    ! burn-in time
 scalelimit  = 0.05    ! when to scale
 scalefactor = 2.5     ! scale factor
 greedy      = 1       ! "greedy" burn in adaptation
 badaptint   = -1      ! burn-in adaptation interval, if < 0 use adaptint
 condmaxini  = 1.0d15  ! reqularize initial modest J'J

! ADVANCED MCMC METHODS
 doadapt     = 1       ! do we adapt
 adaptint    = 1000    ! interval for adaptation
 adapthist   = 0       ! adaptation history size (AP type adaptation)
 adaptend    = 0       ! end adaptation at this time (if >0)
 initcmatn   = 0       ! "imaginary chain size" for initial proposal cmat
 condmax     = 0       ! reqularize cond(cov(chain)) (use svd)
 drscale     = 0.0     ! DR scale

! ERROR VARIANCE SAMPLING AND PREDICTIVE DISTRIBUTIONS
 updatesigma = 0       ! update error variance
 N0          = 1       ! prior for error variance,
 S02         = 0       !    1/s^2 ~ Gamma(N0/2,2/N/S02)
 s2file      = 's2chain.mat'  ! file to save sigma2 chain
 dumpint = 0       ! interval for saving predictions
/
```

4.6 boxoplots.m

```
% Plotting Modest MCMC output

clear all; close all;

% pair plots
chain = loadchain('chain.mat');

figure;
mcmcplot(chain,[],[],'pairs');

% predictive plots
dumpfile ='boxo.dmp';
s2file   ='s2chain.dmp';
estfile  ='boxo.est';

out = dmpfile(dumpfile,s2file);
obs = getobs(estfile);

options.datas   = 1;  % first dataset
options.states  = [];  % no states
options.responses  = 2  % second response

figure;
mcmcpredplots(out,obs,options);
```

References

[1] Haario, H. *Modest User Guide* (1994), available as PDF at: http://www.it.lut.fi/project/MCMC/modest.pdf

[2] Solonen, A. *Monte Carlo Methods in Parameter Estimation of Nonlinear Models* (2006), Master's Thesis, Lappeenranta University of Technology, available online at: http://www.doria.fi/ (collection LutPub).