PhUSE 2009

Paper PO08

Safety analysis using Bayesian simulation methods in SAS 9.2

Armin Gemperli, BIOP AG, Switzerland

ABSTRACT

Berry and Berry (2004) proposed a hierarchical model for the analysis of frequency counts of Adverse Events with one active treatment and one control group. For parameter estimation a simulation-based Bayesian approach has been suggested. SAS® offers in version 9.2 for the first time a procedure that is able to fit models of any complexity using Bayesian simulation. This procedure – called Proc MCMC – is shipped as experimental version and offers all features to implement the Berry & Berry model in SAS. Proc MCMC allows various ways to code the problem. A successful implementation requires some understanding of the functioning of PROC MCMC as will be demonstrated.

INTRODUCTION

SAS introduces with version 9.2 Bayesian fitting of linear models via Gibbs sampling in the GENMOD, LIFEREG and PHREG procedures. For non-linear models an experimental version of Proc MCMC has been provided, allowing great flexibility in model formulation, similarly to Proc NLMIXED or Proc MODEL. So far, Bayesian models have been best fit fully outside of SAS or via an SAS-interface to more powerful Bayesian computing packages such as WinBUGS. This is going to change soon, although the current version of Proc MCMC lacks some of the cleverness of other Bayesian packages like WinBUGS or AMOS®.

The handling of Proc MCMC will be explained and illustrated by fitting the Berry & Berry (2004) model to safety data. The model of Berry & Berry has achieved high recognition as it offers a single analysis combining all Adverse Events and their nesting within body systems. This way the model allows borrowing strength across body systems and avoids the problem of multiple testing, simultaneously.

MODEL FOR ASSESSING DRUG SAFETY

Berry and Berry (2004) describe a statistical analysis of frequency counts of Adverse Events by treatments group and preferred term. Since separate analyses for every preferred term would lead to an increase in wrongly statistically significant comparisons, a single model is suggested that combines all preferred terms and nests them in their respective system organ class. The modelled counts of Adverse Events within a system organ class do borrow strength across the preferred terms.

For N_T patients in the treatment group and N_C controls, the frequency of Adverse Event j, within body system class b is modelled as Binomial(N_T, logit(θ_{bj}+ γ_{bj})), for the treatment and Binomial(N_C, logit(γ_{bj})) for the control group. The parameter θ_{bj}, which is the log-odds ratio for treatment versus control is defined as $\theta_{bj} \sim \pi_b\, 1[\theta_{bj}=0] + (1-\pi_b) * N(\mu_{\theta b}, (\sigma_{b\theta})^2)$. Via parameter π_b we control the probability of no treatment difference (that is $\theta_{bj}=0$). It is modelled as Beta(α_π, β_π). The normal part of the mixture is further defined as $\mu_{\theta b} \sim N(\mu_{\theta 0},(\tau_{\theta 0})^2)$ and $(\sigma_{b\theta})^2 \sim IG(\alpha_\theta, \beta_\theta)$.

Then, $\sigma_{bj} \sim N(\mu_{\gamma b},(\sigma_\gamma)^2)$ and its hyperparameters $\mu_{\gamma b}$ and $(\sigma_\gamma)^2$ are distributed as $N(\mu_0, (\tau_{\gamma 0})^2)$ and $IG(\alpha_{\sigma\gamma}, \beta_{\sigma\gamma})$, respectively. Finally hyperpriors for the remaining parameters are defined as:

$\mu_{\gamma 0} \sim N(\mu_{\gamma 00}, (\tau_{\gamma 00})^2)$

$(\tau_{\gamma 0})^2 \sim IG(\alpha_{\tau\gamma}, \beta_{\tau\gamma})$

$\alpha_\pi \sim Exp(\alpha\lambda_\alpha)I(\alpha>1)$

$\beta_\pi \sim Exp(\beta\lambda_\beta)I(\beta>1)$

$\mu_{\theta 0} \sim N(\mu_{\theta 00}, (\tau_{\theta 00})^2)$

$(\tau_{\theta 0})^2 \sim IG(\alpha_{\tau 0}, \beta_{\tau 0})$

BAYESIAN COMPUTATION

Bayesian statistics needs several ingredients: Besides data and a model for the data, it requires an a-priori guess about that model. The Bayesian paradigm tells us that the probability of the model (given the data) is proportional to the likelihood of the model times the a-priori guess:

Probability(Model given the data) is proportional to the Probability(Data under the model) times Probability(Model),

or: Posterior probability $\propto$ Likelihood $\times$ Prior probability

In a software implementation, such as Proc MCMC, we have to provide the right side of the equation: data, a description of the model and the prior probability of the model. Via the Bayesian formula the left side is computed and represented via histograms. This way, we derive the probability of the model under the sampled data.

The statistical model is defined in terms of parameters. Hence, what is denoted “model” in the formula above can be replaced by the term “parameter”. Unfortunately, derivation of the right hand side is commonly not possible via analytic methods, but via simulation based methods of which Markov chain Monte Carlo (MCMC) is by far the most popular. In MCMC, the posterior probability of every parameter is computed in turn, keeping the other model parameters fixed, and eventually, the entire set of parameters converges to the posterior distribution.

MCMC requires a substantial amount of monitoring, maybe revision of the specifications and ideally some extra sensitivity analysis. Amongst the technicalities that needs to be specified are the length of the chain (number of iterations), and the blocks of parameters to be updated jointly (see next section below). The quality of the sampler is assessed via measures of convergence (convergence must be achieved), correlation between successive iterations (must be low), and acceptance rate of the Metropolis-Hastings algorithm (should be within reasonable limits).

BLOCK UPDATE VERSUS SINGLE PARAMETER UPDATES

Typically MCMC iteratively updates one model parameter at a time, keeping the remaining parameters fixed. Proc MCMC offers the possibility to update entire set of parameters simultaneously, so called block updating. These blocks are typically defined by a set of parameters, which are closely connected, and therefore highly correlated. The advantage of block updates is that convergence to the posterior distribution can be achieved much faster, with the danger that parameter updating may fail totally.

THE METROPOLIS-HASTINGS ALGORITHM

Proc MCMC updates parameters via the Metropolis-Hastings algorithm. In this algorithm first a new value for the parameter is proposed. Then this proposal is accepted or revised based on whether the new value increases the likelihood. SAS currently offers only proposals which are sampled from a multivariate normal, or t-, distribution. If the proposed parameter value is rejected, the sampler continues with the old parameter value. To work smoothly, the acceptance rate should neither be too low nor two high. SAS adapts the variance of the proposal distribution during a tuning period. The goal is to adjust the proposal distribution so that the acceptance rate is close to 0.45 or 0.234 for single parameter and block updates, respectively. The user must carefully choose the length of the tuning period and inspect the final acceptance rate for every (block of) parameter. Single parameter updates and blocks with a low number of parameters might be easier to tune (if close to convergence), but do not easily reach convergence.

A major limitation of Proc MCMC is that only one type of parameter proposal for the Metropolis-Hastings algorithm is supported. WinBUGS, on the other hand, first checks the model for common patterns and then chooses the best suitable algorithm to do the updates. The Metropolis-Hastings algorithm is the most general one among many possible algorithms, but not the most efficient one for a large number of problems.

As an example of failure of the Metropolis-Hastings step recall the prior distribution of the parameter $\theta_{bj} \sim \pi_b\, 1[\theta_{bj}=0] + (1-\pi_b) * N(\mu_{\theta b}, (\sigma_{b\theta})^2)$ in the Berry & Berry (2004) model. That is, θ_{bj} is a mixture distribution: with probability $(1-\pi_b)$ it comes from a normal $N(\mu_{\theta b}, (\sigma_{b\theta})^2)$ and with probability π_b it is 0. During simulation we update θ_{bj} by keeping all remaining parameters, π_b, $\mu_{\theta b}$, $(\sigma_{b\theta})^2$ fixed. In the Metropolis-Hastings step we first propose a new value of θ_{bj} and accept or reject it, following some well defined rules. Since the proposal is sampled from a normal or t-distribution, we are never going to experience a sampled value of exactly 0. A big part of the distribution of θ_{bj} and the whole point in formulating a mixture distribution will be left out! One possibility around this limitation is to re-formulate the model. We have done that by introducing the binary auxiliary variable $z_{bj} \sim Ber(\pi_b)$. θ_{bj} will be sampled from a normal $N(\mu_{\theta b}, (\sigma_{b\theta})^2)$ and multiplied with z_{bj} afterwards to build the frequency of Adverse Events in the treatment group as $Binomial(N_T, logit(z_{bj}\theta_{bj} + \gamma_{bj}))$.

CODING THE SAFETY ANALYSIS MODEL IN PROC MCMC

As an illustration, the data from Mehrotra and Heyse (2001) is accessed:

```
%let N_Treat=148;
%let N_Contr=132;
data AE;
 format A $28.;
 input b j A $ Y RY X RX;
 datalines;
1 1 Astenia/fatigue 57 0.385 40 0.303
1 2 Fever 34 0.230 26 0.197
1 3 Infection-fungal 2 0.014 0 0.000
1 4 Infection-viral 3 0.020 1 0.008
1 5 Malaise 27 0.182 20 0.152
2 1 Anorexia 7 0.047 2 0.015
2 2 Cendisiasis-oral 2 0.014 0 0.000
2 3 Constipation 2 0.014 0 0.000
2 4 Diarrhea 24 0.162 10 0.076
2 5 Gastroenteritis 3 0.020 1 0.008
2 6 Nausea 2 0.014 7 0.053
2 7 Vomiting 19 0.128 19 0.144
3 1 Lymphadenopathy 3 0.020 2 0.015
```

```
4 1 Dehydration 0 0.000 2 0.015
5 1 Crying 2 0.014 0 0.000
5 2 Insomnia 2 0.014 2 0.015
5 3 Irritability 75 0.507 43 0.326
6 1 Bronchitis 4 0.027 1 0.008
6 2 Congestion-nasal 4 0.027 2 0.015
6 3 Congestion-respiratory 1 0.007 2 0.015
6 4 Cough 13 0.088 8 0.061
6 5 Infection-upper-respiratory 28 0.189 20 0.152
6 6 Laryngotracheobronchitis 2 0.014 1 0.008
6 7 Pharyngitis 13 0.088 8 0.061
6 8 Rhinorrhea 15 0.101 14 0.106
6 9 Sinusitis 3 0.020 1 0.008
6 10 Tonsillitis 2 0.014 1 0.008
6 11 Wheezing 3 0.020 1 0.008
7 1 Bite/sting 4 0.027 0 0.000
7 2 Eczema 2 0.014 0 0.000
7 3 Pruritis 2 0.014 1 0.008
7 4 Rash 13 0.088 3 0.023
7 5 Rash-diaper 6 0.041 2 0.015
7 6 Rash-measles/rubella-like 8 0.054 1 0.008
7 7 Rash-varicella-like 4 0.027 2 0.015
7 8 Urticaria 0 0.000 2 0.015
7 9 Viral-exanthema 1 0.007 2 0.015
8 1 Conjunctivitis 0 0.000 2 0.015
8 2 Otitis-media 18 0.122 14 0.106
8 3 Otorrhea 2 0.014 1 0.008
;
run;
```

b being the body system number, j the number of the Advers Event within the body system, Y and X the frequency counts for the treatment and control group, respectively and RY and RX the calculated rate.
The first line of the Proc MCMC code that implements the model by Berry and Berry (2004) looks like:

```
Proc MCMC data=AE seed=6543 outpost=postout maxtune=1000 nmc=11000 nbi=1000
          thin=6 monitor=(_parms_);
```

Via `seed` we initiate the random number generator and produce reproducible results. Proc MCMC produces a series (the chain) of parameter values. These values are written into the dataset specified under `outpost`. Later we use this dataset to draw histograms of the parameters (the posterior distribution) or determine percentiles (Bayesian p-values). We do not necessarily need to post-process our findings, as Proc MCMC already presents histograms and other summary measures from the sampled posterior distribution. It does this for every parameter listed under `monitor`. If we want to derive summary statistics for every parameter, we can simply give the argument `_parms_` to the `monitor` statement instead of list them all. We want to further specify the number of iterations produced (via `nmc`) and the number of burn-in iterations, which are the first `nbi` number of iterations used to reach convergence. Only the post-convergence parameter values at later iterations are used for the summary statistics. We also specify that only every `thin`-th iteration is outputted and all other samples discarded for the summary. A large value of `thin` reduces correlation between successive iterations at the cost of deriving a shorter sample size for post-processing.

It is recommended to start with a small value of `nmc`, inititally. That way it can easily be checked that there is no error in the coding or model, before reaching for the full result and block the computer for days. SAS does not allow us to see the current state of the sampler and we have to wait until Proc MCMC has finished its job, before we gain any insight into the sampler's performance. In other packages (especially AMOS) this is much better implemented. Still, there is the (unofficial) possibility to access the output dataset specified under `outpost` at any time and check out the status of the sampler up to the current iteration. The parameters `nbi` and `thin` do not need to be specified if the data is post-processed: in this case removal of the pre-convergence, burn-in period and thinning of values to reduce serial correlation can be performed after MCMC sampling without any loss. It might be preferred to set these options to the value of 1 in order to output the complete chain and not the chopped and sliced output.

What is crucial, however, is to set the parameter `maxtune` right. It defines the number of iterations initially used for tuning of the Metropolis-Hastings algorithm. Ideally a large value is chosen. Whether PROC MCMC managed to tune the algorithm appropriately has to be verified retrospectively via the Metropolis-Hastings acceptance rate; separately for every parameter.
The next statements specify array of parameters. This way we no longer need to refer to every parameter uniquely. Since we have 8 body systems and within each body system up to a maximum of 11 preferred terms, we define arrays of size 8 for those parameters shared by every preferred term within the body system and fields of size 8×11

for parameters unique per preferred term:

```
array ptheta[8,11];
array mu_thetaa[8];
array sigma_theta[8];
array gamma[8,11];
array mu_gammaa[8];
array z[8,11];
array pi[8];
```

Then we define and initialize the model parameters. For parameters defined in an array we can use the colon modifier (*parameter*:) to refer to all numbered parameters jointly. SAS automatically produces the numbered parameters parameter1, parameter2, etc. for these array-parameters. We can – at any time – also access the parameters via this notation and abandon the array notation.

```
parms (ptheta:) 0;
parms (gamma:) 0;
parms (pi:) 0.5;
parms (mu_thetaa:) 0;
parms (sigma_theta:) 1;
parms (mu_gammaa:) 0;
parms sigma_gamma 1;
parms tau_gamma0 1;
parms alpha_pi 2;
parms beta_pi 2;
parms tau_theta0 1;
parms mu_theta0 0;
parms mu_gamma0 0;
```

We can list as many parameters as we wish after one `parms` statement. The result being that all parameters under the same `parms` statement will be updated jointly, so called block updating. It is advisable to block parameters which are similar and highly correlated. Some test runs with a small number of iterations will show which parameters are best blocked. E.g. the parameters z_1 to z_5 (or z_1 to z_{11}) are all related to body system 1. It makes sense to block them (and initialize with either 0 or 1): `parms z1 z2 z3 z4 z5 z6 z7 z8 z9 z10 z11 1;`

Some checking and early runs revealed that the Metropolis-Hastings algorithm performs badly when blocked and single parameter updates are preferred:

```
parms z1 1; parms z2 1; parms z3 1; parms z4 1; parms z5 1; parms z6 1; parms z7 1;
parms z8 1; parms z9 1; parms z10 1;
parms z11 1; parms z12 1;
```

etc. until `parms z88 1;`

Note that the parameters z were defined in a 2-dimensional array, but referenced via the automatically created dummy variables z1 to z88.

Proc MCMC processes the program code so that it loops though ever parameter, or parameter block, in turn and updates it in a Metropolis-Hastings step keeping the other parameters constant. At every parameter update it loops through the entire input dataset (plus another loop through the programming statements in Proc MCMC). This way SAS builds up the term Likelihood × Prior probability at every iteration for every parameter and does the updating as the very last step. This might be not the most time efficient approach – as not everything changes between subsequent parameters –, but it adds a lot of flexibility to define the model.

Next we define the constants – in this case the hyperprior parameters –. We enclose these statements in a `begincnst` and `endcnst` environment. The code enclosed in this environment is processed at the beginning of the first iteration only and then retained. Although not necessary, this is supposed to save us some seconds.

```
begincnst;
   mu_theta00=0; tau_theta00=10; alpha_theta=3;
   beta_theta=1; alpha_theta0=3; beta_theta0=1;
   alpha_taugamma=3; beta_taugamma=1; alpha_sigmagamma=3;
   beta_sigmagamma=1; mu_gamma00=0; tau_gamma00=10;
   lambda_alpha=1; lambda_beta=1;
endcnst;
```

Finally the prior distributions were defined. SAS provides the two statements `prior` and `hyperprior` to do so. These two statements behave the very same way and can be used interchangeably. Again, we can use the colon modifier to define identical prior distributions for an entire array of parameters.

```
prior mu_gamma0 ~ normal(mu_gamma00, var=tau_gamma00);
prior mu_theta0 ~ normal(mu_theta00, var=tau_theta00);
prior tau_theta0 ~ igamma(alpha_theta0, iscale=beta_theta0);
prior beta_pi ~ expon(iscale=lambda_beta, lower=1);
prior alpha_pi ~ expon(iscale=lambda_alpha, lower=1);
prior sigma_gamma ~ igamma(alpha_sigmagamma, iscale=beta_sigmagamma);
prior tau_gamma0 ~ igamma(alpha_taugamma, iscale=beta_taugamma);
```

```
prior mu_thetaa: ~ normal(mu_theta0, var=tau_theta0);
prior mu_gammaa: ~ normal(mu_gamma0, var=tau_gamma0);
prior sigma_theta: ~ igamma(alpha_theta, iscale=beta_theta);
prior pi: ~ beta(alpha_pi, beta_pi);
```

As we have elaborated above, the prior distributions are evaluated as the sampler steps through the input dataset, row after row. Sometimes, we do not want this. Instead we want to define the conditional distribution manually and have it evaluated only at the last observation:

```
if b=1 and j=1 then do;
 gamma=lpdfnorm(gamma[b,j], mu_gammaa[b], sigma_gamma);
 ltheta=lpdfnorm(ptheta[b,j], mu_thetaa[b], sigma_theta[b]);
end;
else do;
 gamma=lgamma+lpdfnorm(gamma[b,j], mu_gammaa[b], sigma_gamma);
 theta=ltheta+lpdfnorm(ptheta[b,j], mu_thetaa[b], sigma_theta[b]);
end;
prior gamma: ~ general(lgamma);
prior ptheta: ~ general(ltheta);
```

In the above code the likelihood for the parameters θ and γ is constructed step-wise as the sampler increments through the input dataset. That way we can sum up the log-likelihood row by row and finally define the prior via the log-likelihood using the `general` distribution function.

As a next step the prior distributions for all the z parameters are defined separately:

```
prior z1 ~ binary(p=pi[1]);
prior z2 ~ binary(p=pi[1]);
prior z3 ~ binary(p=pi[1]);
...
prior z88 ~ binary(p=pi[8]);
```

The logistic transformation is simply coded as:

```
p_y=logistic(gamma[b,j]+z[b,j]*ptheta[b,j]);
p_x=logistic(gamma[b,j]);
```

We can use most SAS function (such as logistic) and data step programming statements also within Proc MCMC. There are some SAS functions implemented for Proc MCMC exclusively: especially for calculating the (log-)density of various probability distributions.

Finally we define the model for the data. We do that in two separate model statements. Proc MCMC sums up all the log-likelihoods from all the model statements it reads. This way we can provide as many model statements as needed. Alternatively we could have used only one model statement on a self-constructed log-likelihood, as demonstrated above for the parameters θ and γ.

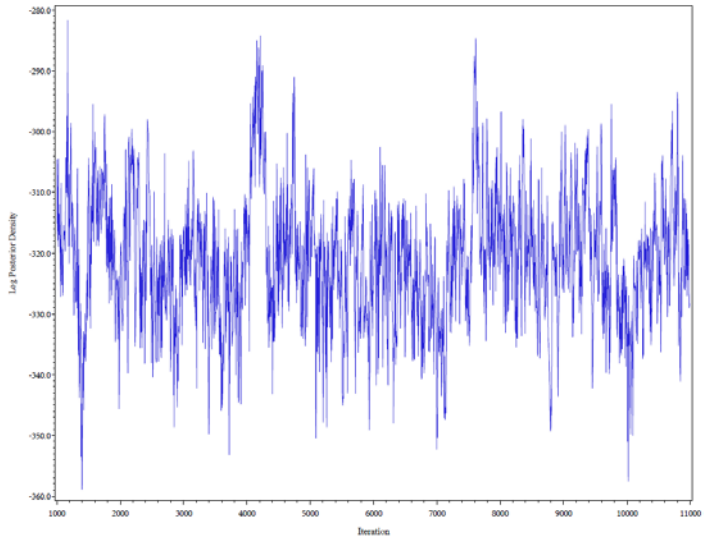
```
model Y ~ binomial(n=&N_Treat, p=p_y);
model X ~ binomial(n=&N_Contr, p=p_x);
run;
```

For ease of coding we have introduced arrays of size 8×11 for the preferred term specific parameters. This adds some overhead as a lot of parameters introduced this way are not relevant for the model. E.g. there is no 6th Adverse Event within body system 1, hence theta[1,6], gamma[1,6] and z[1,6] are defined but not applicable. This does not pose any problem to Proc MCMC. The prior distribution of these parameters is defined, but they do not impact the likelihood: the term Likelihood × Prior is proportional to the prior and hence the simulated posterior consists of samples from the prior.

RESULTS

The outpost dataset includes a column for every model parameter and one column named “iteration”, starting after the burn-in period (`nbi`) and counting all iteration, leaving out the thinned iterations (`thin`). It lists the chain of every sampled parameter value, plus the log prior, log likelihood and log posterior density. The latter is helpful in getting an overall feeling about convergence or correlation between successive iterations.

Figure 1: MCMC-trace of log-posterior density

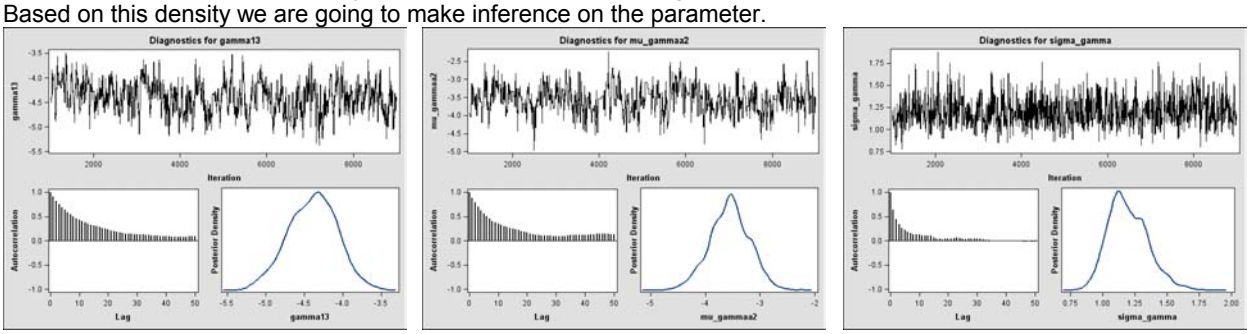


Proc MCMC outputs a number of indicators which help in assessing the quality of the sampler:

- The Metropolis-Hastings acceptance rate for every block of parameters and every phase (that is a number of successive iterations for which the acceptance rate is evaluated and the proposal distribution suitably tuned) within the tuning period (`maxtune`).
- The Metropolis-Hastings acceptance rate for every block of parameters for the burn-in period (`nbi`).
- Monte-Carlo standard errors (and ratio Monte-Carlo standard error / Standard deviation) for every parameter.
- Autocorrelation between successive iterations for every parameter. Ideally, the autocorrelation drops quickly to zero for moderate lags. If the autocorrelation is too high, thinning (interleaving of iterations) can still be performed post-hoc on the outpost dataset.
- Convergence diagnostics: These give indications on whether the sampler has reached convergence within the specified burn-in period (`nbi`). SAS provides seven tests that help do decide if convergence has been reached. If convergence is not attained, some early iterations can still be chopped off retrospectively – if the chain is long enough – in order to have only post convergence values included in the parameter summary.

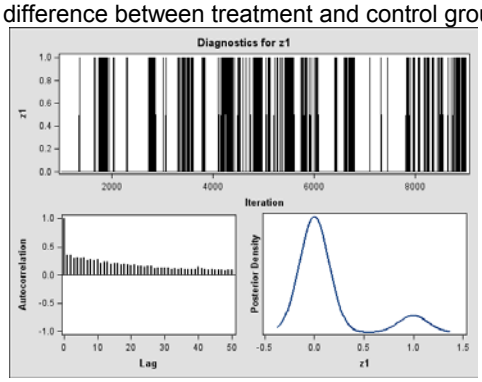
SAS further provides graphical diagnostics separately for every parameter.

Figure 2: Examples of graphical diagnostics and summaries provided by Proc MCMC. The trace shows us how good mixing is and if convergence has been reached, or if there still are long-term trends. The autocorrelation plot (here shown without thinning) indicates the lag up to which autocorrelation is still substantially high. It is recommended to thin the data by re-sampling every x-th datapoint in the output dataset; with x being the smallest lag at which the autocorrelation is low. The density plot shows a smoothed histogram of the posterior distribution of the parameter. Based on this density we are going to make inference on the parameter.



Once we are certain that the sampler is of good quality (convergence reached, good mixing and low autocorrelation), we can have a look at statistical summaries of the sampled posterior distribution for every parameter. These summaries are provided in the SAS output. Alternatively, we process the output dataset to derive parameter estimates and Bayesian p-values.

Figure 3: Diagnostics for parameter z1 (auxiliary binary variable on whether frequencies of asthenia/fatigue differ between the treatment and the control group). Due to the binary nature of the parameter being sampled the trace looks dissimilar to those depicted in Figure 2. The density plot is a smoothed bar-chart with two bars only, at 0 and 1. By calculating the frequencies of zeros and ones in the output dataset, we can derive an a-posteriori estimate of the difference between treatment and control group in asthenia/fever.



EXTENSIONS

A weakness of Proc MCMC is the Metropolis-Hastings algorithm. The proposals used for this algorithm are not flexibly stated. They need adaptation during implementation and a careful examination of the acceptance rate. Even if the conditional distribution is of simple form (such as conjugate), Proc MCMC does not perform anything smarter than a Metropolis-Hastings step. In this case it makes sense to program a tailored updating scheme different from Metropolis-Hastings. These User Defined Samplers (UDS) are supported in Proc MCMC via the UDS statement. It is implemented as follows:

1) Use Proc FCMP to program a subroutine that updates the parameter of interest.
2) Declare the parameters used by this subroutine in a `parms` statement.
3) Define the prior distribution for every parameter used by this subroutine in a `prior` statement.

The Berry and Berry model is formulated in a way to provide conjugate prior distributions whenever possible. For sampling from conjugate distributions, there exist well known algorithms which are more efficient than Metropolis-Hastings. Since Proc MCMC does not exploit this structure as it should, the programmer should implement a tailored updating scheme.

As an example we show how the parameter $\mu_{\theta 0}$ can be efficiently updated using UDS. Recall $\mu_{\theta b} \sim N(\mu_{\theta 0}, (\tau_{\theta 0})^2)$ and $\mu_{\theta 0} \sim N(\mu_{\theta 00}, (\tau_{\theta 00})^2)$. These are the only two terms where the parameter $\mu_{\theta 0}$ is used. Since the normal prior is the conjugate distribution of a normal we can easily identify the full conditional distribution as: $\mu_{\theta 0} | \mu_{\theta b}, (\tau_{\theta 0})^2, \mu_{\theta 00}, (\tau_{\theta 00})^2 \sim N((\mu_{\theta 00}/(\tau_{\theta 00})^2 + m\, \mu_{\theta .}/(\tau_{\theta 0})^2)/(1/(\tau_{\theta 00})^2 + m/(\tau_{\theta 0})^2), 1/(1/(\tau_{\theta 00})^2 + m/(t_{\theta 0})^2))$, with m=8 being the number of body systems and $\mu_{\theta .} = (\mu_{\theta 1} + \ldots + \mu_{\theta 8})/8$.

```
proc fcmp outlib=sasuser.funcs.uds;
      subroutine mu_theta0update(mu_theta0, tau_theta0, mu_theta00, tau_theta00, m,
                                 sum_mut);
      outargs mu_theta0;
      var = 1/(1/tau_theta00 + m/tau_theta0);
      mean = (mu_theta00/tau_theta00 + sum_mut/tau_theta0) * var;
      mu_theta0 = rand("normal", mean, sqrt(var));
      endsub;
run;
```

Since the subroutine is saved in the OUTLIB= library, we have to specify the CMPLIB option to let SAS search at the right place for the mu_theta0update subroutine: `options cmplib=sasuser.funcs;`

In Proc MCMC we use the subroutine by specifying:

```
UDS mu_theta0update(mu_theta0, tau_theta0, mu_theta00, tau_theta00, m,
                                 sum_mut);
parm mu_theta0 / uds;
```

All the other parameters used in the routine are defined as before. The parameter constant m=8 and sum_mut need to be newly introduced. This is best done in the cnst-environment:

```
m=8;
sum_mut = sum(of mu_thetaa1 mu_thetaa2 mu_thetaa3 mu_thetaa4 mu_thetaa5 mu_thetaa6
mu_thetaa7 mu_thetaa8);
```

All the rest of the code stays the same, as previously defined, including the prior statement for mu_theta0. The UDS option in the parms statement for mu_theta0 makes sure the own defined subroutine is used for the parameter update and not the SAS internal Metropolis-Hastings algorithm.

CONCLUSION

SAS provides with the experimental version of Proc MCMC a marvelous addition to the statistical modeling procedures. Proc MCMC is very flexible, yet it is nicely integrated into the entire SAS system and anybody understanding SAS (and Bayesian Modeling) will not have problems using Proc MCMC. The uncompromising SAS integration is also a weakness: On-line monitoring of the sampler (which often runs for days) is hardly possible. This is better solved in other MCMC software packages. Any MCMC implementation is highly adaptive; it needs monitoring and adjustment.

The main weakness of Proc MCMC is the restriction to perform a Metropolis within Gibbs sampler. This is the most general form of Bayesian simulation and certainly a good start. For specialized problems it is far from ideal, however. Fortunately, SAS provides a way to overcome this setback by the possibility to program own samplers with Proc FCMP.

The released – experimental – version of Proc MCMC will certainly be improved. Currently, Proc MCMC is not user friendly enough for beginners in Bayesian computation and not flexible enough for Bayesian power users. Still, anybody with interest in Bayesian computation and SAS will warmly welcome the new addition to the SAS Proc family.

REFERENCES

Berry SM, Berry DA (2004) Accounting for multiplicities in assessing drug safety: a three-level hierarchical mixture model. Biometrics 60(2) 418-426.

Carlin BP, Louis TA (2000) Bayes and Empirical Bayes Methods for Data Analysis. Chapman & Hall/CRC.

Gelman A, Carlin J, Stern H, Rubin D (2004) Bayesian Data Analysis. Chapman & Hall/CRC.

Mehotra DV, Heyse JF (2004) Multiplicity considerations in clinical safety analyses. Statistical Methods in Medical Research 13, 227-238.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Armin Gemperli
BIOP AG
Centralbahnstrasse 29
CH-4051 Basel
Armin.Gemperli@biop.ch