An introduction to STAN and SAS PROC MCMC

Pierre Lebrun
Astrid Jullion
Outline

- Bayesian basics
- Presentation of common samplers in SAS proc MCMC
- Presentation of the No-U-Turn Sampler in Stan
- Overview of some diagnostic tools to check sampled chains
- Proc MCMC
- Stan / rstan
  - Installation guide
  - Use
- Examples in proc MCMC and Stan
  - Poisson random model for the EPIL data with highly correlated parameters
  - Right-censored survival model for KIDNEY data
Bayes Theorem

- Posterior distribution of the parameters

\[ p(\theta \mid y) = \frac{p(y \mid \theta) \cdot p(\theta)}{p(y)} \]

\[ p(\theta \mid y) \propto L(\theta \mid y) \cdot p(\theta) \]

Posterior \( \propto \) Likelihood \( \times \) Prior

- Prediction of a new observation

\[ p(\tilde{y} \mid y) = \int_{\theta} p(\tilde{y} \mid \theta) \cdot p(\theta \mid y) \, d\theta \]

Likelihood given the parameters

Predictive density integrating out the parameter distribution
Let’s consider that $\theta$ is the parameter of interest (ex: treatment effect)

**$\theta$ is treated as random variables**

1. **Prior distribution** of parameter $\theta$ : $p(\theta)$
   - Distribution of $\theta$ before any data are observed
   - Reasonable opinion concerning the plausibility of different values of $\theta$
   - Ideally based on all available evidence/knowledge (or belief)
   - Or deliberately select a non-informative prior
Examples of prior distributions

**Gamma distributions**

- Prior distribution -> Specify the domain of plausible values
- Specify the weights given to these values

**Beta distributions**

- Prior distributions do not have to be a Normal (not only prior mean and prior variance)
- Prior distributions ≠ initial values.
Bayesian principle

2. **Likelihood:**
   - Conditional probability of the data given $\theta$: $p(y\mid \theta)$
   - Based solely on data

3. **Posterior distribution:**
   - Distribution of $\theta$ after observed data have been taken into account: $p(\theta\mid y)$
   - Final opinion about $\theta$

4. **Predictive distribution:**
   - Given the model and the posterior distribution of its parameters, what are the plausible values for a future observation $y^*$? $p(y^*\mid \theta)$
Sampling

- When it is not possible to identify a known distribution for the posterior of parameters
  - Rely on sampling from the complete joint posterior

- But, a MCMC sampler is cumbersome and time consuming to program and tune
  - So, use an existing ‘multi-purpose sampler’ already existing
    - BUGS based (Win/OpenBUGS, JAGS)
    - SAS based (proc MCMC)
    - R/C++ based (Stan, JAGS)
  - Or use very good approximations of the posterior
    - INLA
Predictions

- For a majority of hierarchical (unbalanced) linear or nonlinear models, the predictive distribution is non-tractable.
  - Often, the posterior of the parameters is not identified.
- In this case, the integral in the prediction formula could be resolved using Monte-Carlo simulations if samples of the parameter posterior distribution are available.

\[
\begin{align*}
& \text{draw } \theta^{(s)} \text{ from the joint posterior density } p(\theta \mid y), \\
& \text{draw } \tilde{y}^{(s)} \text{ from the model } p(\tilde{y} \mid \theta^{(s)}), \\
& s = 1, \ldots, n^* \text{ is the number of samples to draw.}
\end{align*}
\]
Markov Chain Monte Carlo basics
Bayesian analysis using SAS

- **SAS** allows some Bayesian analysis with:
  - GENMOD (generalized linear models)
  - PHREG (Cox proportional hazards models)
  - LIFEREG (accelerated failure time models)
  - MIXED (prior statement to sample from variance components distribution)

- **Proc MCMC:**
  - Nearly any models
  - Program your likelihood, your prior and tune your MCMC algorithm
  - Algorithms:
    - Metropolis-Hasting
    - Independent sampler
    - Conjugate updater using Gibbs whenever possible
Basic algorithm

For $s = 1$ to $n^*$

1. From a symmetric proposal distribution $q(\theta^{(s)} | \theta^{(s-1)})$, draw a new candidate vector $\theta_i$,

2. compute the acceptance probability: $P_a = \min \left( 1, \frac{p(\theta^{(s)} | \text{data}) \cdot q(\theta^{(s-1)} | \theta^{(s-1)})}{p(\theta^{(s-1)} | \text{data}) \cdot q(\theta^{(s)} | \theta^{(s-1)})} \right)$

3. keep $\theta^{(s)}$ with probability $P_a$ or assign the old value $\theta^{(s)} = \theta^{(s-1)}$ otherwise.

End

Can be easily used for drawing univariate parameters conditional to the previous values ($s-1$) of the others
Example of hand-made MCMC simulations

#Posterior distribution

logposterior=function(theta) -abs(theta)^3

#Number of generated values in the chain

M=1000

#Starting value for theta:
theta=c()
theta[1]=4
#Count the number of acceptations
n.accept=0
for (i in 2:M) {
    # Draw a value from the proposal symmetric distribution
    theta.prop <- rnorm(1, theta[i-1], 1.6)
    # Compute the probability
    prob <- min(1, exp(logposterior(theta.prop) - logposterior(theta[i-1])))
    accept <- (runif(1) <= prob)
    if (accept) {
        n.accept <- n.accept + 1
        theta[i] <- theta.prop
    }
    else theta[i] <- theta[i-1]
    # Compute the acceptance rate
    round(n.accept/(M-1), 2)
MCMC simulations

Acceptance rate = 0.43
If the problem is not ill-conditionned, the Markov chain should eventually converge to the desired distribution.

However, the first sample position that is provided to the sampler (= initial value) might be far from this distribution:

- the starting position has a very low density.

A burn-in period is then generally envisaged:

- It consists in running the sampler for, say, 5000 iterations, to make it converge, and then continue the sampling.
- Throwing away the first 5000 samples, the remaining samples should represent a sample from the posterior.

See diagnostics.
Gibbs sampling

- If the full conditional posterior distribution of subsets of parameters can be identified, use Gibbs Sampling
  - Use this conditional distribution as proposal and accept every draws

For $j = 1$ to $m$

- Draw a sample from $\theta_j^{(s)} \sim p(\theta_j \mid \theta_1^{(s-1)}, \theta_2^{(s-1)}, \ldots, \theta_{j-1}^{(s-1)}, \theta_{j+1}^{(s-1)}, \ldots, \theta_m^{(s-1)}, \text{data})$

End

- Most algorithms, including proc MCMC or BUGS based samplers, have rules and algorithms to derive the full conditional posteriors to use Gibbs sampling

- They choose automatically if e.g. Gibbs or Metropolis-Hasting have to be used
Gibbs sampling visualized

(a) $P(x)$

(b) $P(x_1 | x_2^{(t)})$

(c) $P(x_2 | x_1)$

(d) $x^{(t+2)}$, $x^{(t+1)}$, $x^{(t)}$

MCMC with correlations

- Sampling a multivariate distribution from a univariate proposal

- Very slow exploration of the parameter space
  - especially if correlation is present

- As sample j is very close to sample j-1 \(\rightarrow\) autocorrelation

- Convergence can be very slow as well
How to improve exploration?

- Take a multivariate distribution as proposal
  - If correlations can be roughly estimated, the sampling can be improved to account for the dependency structure
    
    Still, it does not work well with exotic distribution (e.g. banana shaped, etc.)
  
  - Sampling by block easier if interesting blocking of similar parameters can be identified
    
    E.g. in a regression, sample the regressors and the variance in two blocks

- Thin the samples
  
  - Keeping only one sample out of, say, 10, to obtain a ‘faster’ exploration of the distribution

- Transform the model to obtain uncorrelated parameters
  
  E.g. in BUGS: \( \text{mu}[i] \leftarrow \text{alpha} + \text{beta} \times (\text{x}[i] - \text{x.mean}) \)

- Overrelaxation method
Hamiltonian Monte-Carlo

- The idea is to avoid the random walk behavior of MCMC algorithms
  - Uses Hamiltonian dynamics
  - Auxiliary momentum vector

So, a state (one sample) has a position and a momentum (mass*velocity)

- A potential energy ($\propto$ to the posterior density height)
- A kinetic energy (momentum & mass)

$\rightarrow$ The target density defines a potential energy function using Hamiltonian equations

$\rightarrow$ One sampling iteration consists in moving on the posterior following these dynamics
  - instead of moving using a simpler proposal distribution

- Theory terminology is often hard for statisticians without a good knowledge of physics
Two types of proposals are used iteratively

1. randomize the momentum variable (give a velocity)
2. move on the posterior using Hamiltonian equations
   - Leapfrog function

Discard the momentum variables and keep only the sequence of position (i.e. samples)

Hamiltonian Monte-Carlo

Why it is performant?

- Rely on the gradient of the current location of the posterior to better know the direction to take towards the next sample
- Leapfrog functions are used to discretize the Hamiltonian equations
  Computers can work with them very efficiently
  Explore the posterior distribution more efficiently using several leapfrogs to reduce autocorrelation

Neal, R. MCMC using Hamiltonian dynamics, in Handbook of Markov Chain Monte Carlo, Brooks et al, Chapman & Hall, 2010
Using the leapfrog function, two parameters have to be tuned:
- The size of the leap (the step)
- The number of leaps

Tuning them is a complex task that may require many additional runs.

The No-U-Turn Sampler (NUTS) is an improvement of HMC that have routines to tune these parameters on-the-fly.
NUTS vs. random Walk

- How NUTS performs compared to the other samplers and compared to an i.i.d. sampler assuming it is available

Diagnostic tools
ESS and Geweke

- Both available even when only one chain is available
- Effective sample size

\[
\text{ESS} = \frac{n}{\tau} = \frac{n}{1 + 2 \sum_{k=1}^{\infty} \rho_k(\theta)}
\]

ESS corrects the number of samples obtained, by the autocorrelations present in the chains

- Geweke diagnostic
  - The Geweke test compares values in the early part of the Markov chain to those in the latter part of the chain in order to detect failure of convergence.
  - Similar to a two-sided t-test to compare 2 means, with standard errors that can be adjusted for autocorrelations
Monte Carlo Standard Errors

- MCSE = Monte Carlo Standard Errors of the mean: accuracy of the posterior estimates
- SD = posterior standard deviations computed on the chain
- Given an effective sample size of m, the MC standard error for the mean is \( \hat{\sigma} / \sqrt{m} \), the procedures use the following formula to include ESS:

\[
\hat{\text{Var}}(\bar{\theta}) = \frac{1 + 2 \sum_{k=1}^{\infty} \rho_k(\theta_i) \cdot \sum_{i=1}^{n} (\theta_i - \bar{\theta})^2}{n(n-1)}
\]

- If the values in the “MCSE/SD” column are small, it means that only a fraction of the posterior variability is due to the simulation.
Gelman argues that the best way to identify non-convergence is to simulate multiple sequences for over-dispersed starting points/initial values.

The intuition is that the behavior of all of the chains should be basically the same, if convergence occurs.

As Gelman and Rubin put it, the variance within the chains should be the same as the variance across the chains.

This can be diagnosed pretty easily through traceplots of multiple chains. You want to see if it looks like that the mean and the variance of all the chains are the same.
Examples where convergence seems reasonable (top) and unreasonable (bottom)
Gelman-Rubin-Brook diagnostic (~F-test ANOVA)

- If convergence, the dispersion within the chains should be equal to the dispersion observed between the chains
- Need several chains!
- Pooled within chain variance

\[ W = \frac{1}{m} \sum_{j=1}^{m} s_j^2 \]

\[ s_j^2 = \frac{1}{n - 1} \sum_{i=1}^{n} (\theta_{ij} - \bar{\theta}_j)^2 \]

- Between chain variance

\[ B = \frac{n}{m - 1} \sum_{j=1}^{m} (\bar{\theta}_j - \bar{\theta})^2 \]

\[ \bar{\theta} = \frac{1}{m} \sum_{j=1}^{m} \bar{\theta}_j \]

- Total variance

\[ \hat{\text{Var}}(\theta) = (1 - \frac{1}{n})W + \frac{1}{n}B \]
- The R statistic

\[ \hat{R} = \sqrt{\frac{\text{Var}(\theta)}{W}} \]

- R should be very close to 1 in case of convergence
- Local optima convergence (with poor initial values all in the same area) may not be identified, even if R is close to 1
- To do with all chains of parameters
- Potential scale reduction factor (PSRF) : Compute R through the iterations (including burn-in) : Gelman-Rubin-Brooks plot
Other classical tools

- **Trace plots**

- **Densities**

- **Autocorrelations**
Other classical tools

- SAS default output for one chain/parameter
Intervals

- Quantile-based 95% credible Interval: 2.5% and 97.5% quantiles

- Alternative:
  - Highest posterior density (HPD) interval
  - Shortest interval containing 95% of the posterior probability: 
    $[\theta_0 ; \theta_1]$ such that:

\[
\int_{\theta_0}^{\theta_1} p(\theta|data)d\theta = 0.95
\]
Intervals

• **Bayesian credible interval**: 95% most plausible/credible values

• **Frequentist Confidence interval**: “If we repeat the same experiment a large number of times, the confidence interval will cover the true value in 95% of the cases.”

- If the posterior distribution is approximately symmetric, the HPD and quantile-based credible interval are very similar.
Diagnostic tools

- Estimates and intervals

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>se_mean</th>
<th>sd</th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
<th>n_eff</th>
<th>Rhat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[1]</td>
<td>-0.6</td>
<td>0</td>
<td>0.6</td>
<td>-1.9</td>
<td>-1.0</td>
<td>-0.6</td>
<td>-0.2</td>
<td>0.5</td>
<td>1793</td>
<td>1</td>
</tr>
<tr>
<td>beta[2]</td>
<td>-0.6</td>
<td>0</td>
<td>0.4</td>
<td>-1.3</td>
<td>-0.8</td>
<td>-0.5</td>
<td>-0.3</td>
<td>0.2</td>
<td>1748</td>
<td>1</td>
</tr>
</tbody>
</table>

> HPDinterval(mc,0.95)

<table>
<thead>
<tr>
<th></th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[1]</td>
<td>-1.762674</td>
<td>0.5780758</td>
</tr>
<tr>
<td>beta[2]</td>
<td>-1.284624</td>
<td>0.2594099</td>
</tr>
</tbody>
</table>

Posterior Intervals

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alpha</th>
<th>Equal-Tail Interval</th>
<th>HPD Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta0</td>
<td>0.050</td>
<td>-1.8694</td>
<td>-1.9106</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.5550</td>
<td>0.4782</td>
</tr>
<tr>
<td>beta1</td>
<td>0.050</td>
<td>-1.3911</td>
<td>-1.3095</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0.2186</td>
<td>0.2707</td>
</tr>
</tbody>
</table>
Convergence Diagnostic Summary

1) You can never prove that something has converged, you can only tell when something has not converged.

2) If your model has not converged and you are confident that you haven’t made a stupid mistake, then the best thing to do may be to just let the model run a long time.

3) For models with large numbers of parameters you should let the model run for a long time.

4) There are a number of “easy to implement” tricks (mostly reparameterizations) that will help to speed convergence.
Proc MCMC
ods graphics on;
proc mcmc data=mcmc.c12 outpost=mcmc.predcmax nbi=1000 nmc=10000 thin=5 seed=2466810
monitor=(_parms_ mu test45 test80 test85 )
    STATS(ALPHA=(0.1 0.2))=ALL ;
parms alpha 0 beta 0;
parms sigma2 1;
prior alpha beta ~ normal(mean = 0, var = 1e6);
prior sigma2 ~ igamma(shape = 0.00000001, scale = 0.00000001);
mu = alpha + beta*ln_dose_;  
model ln_cmax_ ~ normal(mu, var = sigma2);
    test45 = exp(alpha + beta*log(45)) ;
    test80 = exp(alpha + beta*log(80)) ;
    test85 = exp(alpha + beta*log(85)) ;
run;
ods graphics off;
ods graphics on;
proc mcmc data=mcmc.c12 outpost=mcmc.predcmax nbi=1000 nmc=10000 seed=2466810
  monitor=(_parms_ mu test45 test80 test85 ) STATS(ALPHA=(0.1 0.2))=ALL ;

- **outpost**: dataset with the chain of monitored parameters
- **nbi**: number of burn-in values
- **nmc**: number of sampled values
- **seed**: for analysis repeatability
- **monitor**: parameters to monitor
- **STATS**: saved statistics and level of alpha for the posterior intervals
parms alpha 0 beta 0;
  parms sigma2 1;
prior alpha beta ~ normal(mean = 0, var = 1e6);
prior sigma2 ~ igamma(shape = 0.00000001, scale = 0.00000001);

- **parms**: initial values
- **prior**: prior distributions
SAS code (4)

\[
\begin{align*}
\text{mu} & = \alpha + \beta \text{ln}_\text{dose}_-; \\
\text{model ln_cmax_-} & \sim \text{normal}(\mu, \text{var} = \sigma^2); \\
\text{test45} & = \exp(\alpha + \beta \text{log}(45)) \\
\text{test80} & = \exp(\alpha + \beta \text{log}(80)) \\
\text{test85} & = \exp(\alpha + \beta \text{log}(85)) \\
\end{align*}
\]

- **model**: likelihood function
Truncated distribution:
- prior alpha ~ normal(mean = 0, sd = 1, lower = 3, upper = 45);

Censored data:
- Likelihood:
\[
L(\theta) = \prod_{T_i \in unc} \Pr(T = T_i | \theta) \prod_{i \in l.c.} \Pr(T < T_i | \theta) \prod_{i \in r.c.} \Pr(T > T_i | \theta) \prod_{i \in r.c.} \Pr(T_{i,d} < T < T_{i,r} | \theta)
\]

For right censored data, the likelihood is the product of the likelihood under uncensored data and the likelihood under censored data.

```sas
if uncensored then ll = logpdf('normal', x, mu, s);
else if leftcensored then ll = logcdf('normal', xl, mu, s);
else if rightcensored then ll = logsdf('normal', xr, mu, s);
else ll = log(cdf('normal', xr, mu, s) - cdf('normal', xl, mu, s));
model general(ll);
```
**PARM statement**

- Each statement forms a block of parameters, where the parameters are updated simultaneously in each iteration.

- If high posterior correlations, putting parameters in the same block improves the mixing of the chain: the efficiency that the posterior parameter space is explored by the Markov chain.

- Possibilities:
  - sample all parameters simultaneously by putting them all in a single PARMS statement
  - sample parameters individually by putting each parameter in its own PARMS statement
  - sample certain subsets of parameters together by grouping each subset in its own PARMS statements.

- There are no theoretical results that can help determine an optimal “blocking” for an arbitrary parametric model. A rule followed in practice is to form small groups of correlated parameters that belong to the same context in the formulation of the model.
**RANDOM statement**

- Used for hierarchical models

E.g. for univariate random effect:

```sas
random u ~ normal(mu, var=s2u) subject=index monitor=(u_1-u_3 u_23);
random u ~ normal(mu, var=s2u) subject=index monitor=(u);
```

E.g. for bivariate random effect:

```sas
array w[2];
array mu[2];
array cov[2,2];
random w ~ mvn(mu, cov) subject=zipcode;
```
**Proc MCMC in SAS 9.3**

- **PREDDIST statement**

  - The PREDDIST statement creates a new SAS data set that contains random samples from the posterior predictive distribution of the response variable.

How to handle several chains?

%macro gmcmc;
%do i=1 %to &nchain;
  data _null_;  
    set init;
    if Chain=&i;
    %do j = 1 %to &nparm;
      call symputx("init&j", %scan(&var, &j));
    %end;
    stop;
  run;
%end;
%mend;

data init;
  input Chain beta0 beta1 sigma2;
  datalines;
  1 10 -5 1
  2 -15 10 20
  3 0 0 50
; /* define constants */
%let nchain = 3;
%let nparm = 3;
%let nsim = 50000;
%let var = beta0 beta1 sigma2;

proc MCMC data=onedataset outpost=out&i init=reinit nbi=0 nmc=&nsim stats=none seed=7;
  parms beta0 &init1 beta1 &init2;
  parms sigma2 &init3;
  prior beta0 beta1 ~ normal(0, var = 1e6);
  prior sigma2 ~ igamma(3/10, scale = 10/3);
  mu = beta0 + beta1*height;
  model weight ~ normal(mu, var = sigma2);
  run;
%end;
%mend;

Additional to the basic diagnostics (1-chain, density, autocorrelation)

data all;
  set out1(in=in1) out2(in=in2) out3(in=in3);
  if in1 then Chain=1;
  if in2 then Chain=2;
  if in3 then Chain=3;
run;

%gelman(all, &nparm, &var, &nsim);

data GelmanRubin(label='Gelman-Rubin Diagnostics');
  merge _Gelman_Parms _Gelman_Ests;
run;

/* plot the trace plots of three Markov chains. */
%macro trace;
  %do i = 1 %to &nparm;
    proc sgplot data=all cycleattrs;
      series x=Iteration y=%scan(&var, &i) / group=Chain;
    run;
  %end;
%mend;
%trace;
Stan installation (Windows)
For a Windows installation

- (probably easier on Linux-based systems)

1) Go to the website http://mc-stan.org/
   - All the information in the next slides comes from there

2) Follow the installation of the prerequisites

3) The simpler is to call Stan from R
Stan: prerequisites

- R is readily available on http://www.r-project.org
- Rstudio (http://www.rstudio.com) is not mandatory, but is recommended to help
  - edit files (R scripts, report Sweave files, possibly C/C++ files)
  - manage projects
  - manage packages
  - etc.
- Both R and Rstudio are free and open source!
Stan needs a C++ compiler
- Several options exists, the simplest is to rely on Rtools (http://www.r-project.org)

What is Rtools?
- Rtools is developed to compile R packages and build R for Windows
- Rtools contains a C/C++ compiler for Windows (gcc)
- As all C/C++ compiler, gcc does not like ‘blank’ character in its path
  - Avoid to install it in ‘Program Files’
  - Install it on the root: ‘C:\Rtools’
  - If not admin, install it on your personal folder

At the end of the installation, Rtools asks for a Path update: if possible, do it
- The Path will make R (and any softwares) aware of the existence of gcc
On some computers, the last Rtools installation dialog may fail
  - Possible to edit the Path manually
- Click edit
- Add ‘c:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;’ or the path were Rtools has been installed in the beginning of the Path if not already present
- If there are other gcc compiler(s) named gcc or g++, then you should take care that the last pathes in the Path overwrite the firsts
Stan: prerequisites

- Rstan needs the Rcpp and inline packages
  - `install.packages('inline')`
  - `install.packages('Rcpp')`

- Try them with

```r
# using library inline to compile a C++ code on the fly
library(inline)
library(Rcpp)
src <- 'std::vector<std::string> s;
s.push_back("hello");
s.push_back("world");
return Rcpp::wrap(s);
'

hellofun <- cxxfunction(body = src, includes = '', plugin = 'Rcpp', verbose = FALSE)
cat(hellofun(), '\n')
```

- If warnings, safely ignore if it still writes ‘hello world’ in the R prompt
## Stan installation

- **Last chance if it does not work**
  - ask R to update its PATH for the local session only, to make it aware of gcc
    
    ```
    > Sys.setenv("PATH" = "c:\Rtools\bin;c:\Rtools\gcc-4.6.3\bin;")
    ```
  - Warning: if other pathes were needed for other libraries, they are deleted until R is restarded

- **Once gcc is working**
  - you can develop C/C++ codes to improve computations for some of the R bottlenecks (e.g. a for loop is efficient is C or C++)
  - Install rstan package from it repository (not in CRAN)
    
    ```
    > options(repos = c(getOption("repos"), rstan = "http://wiki.stan.googlecode.com/git/R"))
    ```
    ```
    > install.packages('rstan', type = 'source')
    ```
Stan installation

- Warning about unavailability of rstan for R 3.X.X can be ignored
- Other warnings during compilation of rstan can be ignored if the package can be loaded
  - library(rstan)
Stan language basics
Stan language

- Blocks
  - In C++, all variable types must be defined
  - Stan inherits from this properties

- `data { ... declarations ... }

- `transformed data { ... declarations ... statements ... }

- `parameters { ... declarations ... }

- `transformed parameters { ... declarations ... statements ... }

- `model { ... declarations ... statements ... }

- `generated quantities { ... declarations ... statements ... }
Stan language

- All blocks but ‘model’ are optional
  - Order matters

- A variable that is declared in one block can be used in the subsequent blocks, but not before

- Block ‘parameters’ and ‘transformed parameters’
  - Define the type and the domain of each parameter

```stan
parameters {
  real  a0;
  real  b1[N];
  real  b[N, T];
  real<lower=0> sigmasq_b;
  real<lower=0,upper=50> sigmasq_a;
  int nu;
}
```

```stan
transformed parameters {
  #Executed at each leapfrog
  real<lower=0> sigma_b;
  sigma_b <- sqrt(sigmasq_b);
}
```
**Block ‘model’**

- Contains (possibly) priors
- Contains likelihood definition in a BUGS-like style

```stan
model {
    real[N] mu_hat; // tmp variable declaration

    alpha ~ normal(0, 1000);
    beta ~ normal(0, 1000);
    sigma ~ uniform(0, 1000);

    for(i in 1:N){
        mu_hat[i] <- alpha + beta * year[i];
        y[i] ~ normal(mu_hat[i], sigma);
    }
}
```
‘Non informative’ = no prior distribution = uniform over the domain

model {
  real[N] mu_hat; #tmp variable declaration

  #alpha ~ normal(0, 1000);
  #beta ~ normal(0, 1000);
  #sigma ~ uniform(0, 1000);

  for(i in 1:N){
    mu_hat[i] <- alpha + beta * year[i];
    y[i] ~ normal(mu_hat[i], sigma);
  }
}

- The sampler does not need a prior to know the variables domain
  - Already given in the block ‘parameters’

- This is one main advantage, e.g. to more easily define multivariate hyper priors... just do nothing
Vectorization

- (most) Stan distribution are vectorized
- it means that Stan can sample vector from a seemingly univariate distribution

```stan
parameters {
  real beta[2];
}

model {
  beta ~ normal(0, 1000); # Or,
  for (i in 1:N) {
    <model statement>
  }
  for (n in 1:N) y[n] ~ bernoulli(theta);
  # is equivalent to the vectorized form,
  y ~ bernoulli(theta);
}
```
Block ‘generated quantities’

- Computed once per sample
  - If a (transformed) variable does not play a role in the model, it is more efficient to compute the transformation in this block rather than in the block ‘transformed parameters’

- Does not affect the sampled values

- Allows obtaining
  - posterior estimation of combination/transformation of parameters
  - predictions for new data
  - compute deviance or log likelihood for model comparison
  ...

Stan language

- rstan package / interface

```r
library(rstan)

#compile the model, Data is a list as in BUGS
fit <- stan(model_code = stan_code, data = Data, iter = 1000, chains = 1)

#more parameters allow using classical Hamiltonian sampler

#use the model
fit2 <- stan(fit = fit, data = Data, iter = 15000, chains =2, thin=10, warmup=5000,
     init=list(list(beta=c(1,1)),list(beta=c(-5,1))))

#print and plots
print(fit2, probs = c( 0.25, 0.5, 0.75), digits_summary=2)
plot(fit2)
traceplot(fit2)

#export as a more classical mcmc object to be able to use classical coda/MCMCpack tools
library(MCMCpack)
mc = as.mcmc(as.matrix(fit2))
acfplot(mc)
densityplot(mc)
HPDinterval(mc)
```
Examples
Epil data

- Poisson with random effects for both individual subjects and also random effects for subject by visit to model extra-Poisson variability within subjects

<table>
<thead>
<tr>
<th>Patient</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>Trt</th>
<th>Base</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>11</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>4</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>6</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>8</td>
<td>36</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>40</td>
<td>20</td>
<td>21</td>
<td>12</td>
<td>0</td>
<td>52</td>
<td>42</td>
</tr>
<tr>
<td>9</td>
<td>5</td>
<td>6</td>
<td>6</td>
<td>5</td>
<td>0</td>
<td>12</td>
<td>37</td>
</tr>
<tr>
<td>....</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>12</td>
<td>37</td>
</tr>
</tbody>
</table>

- Seizure counts in a randomised trial of anti-convulsant therapy in epilepsy
  - More on this model in the previous presentation!
PROC MCMC data=poisson.data outpost=poisson.postout thin=1 nbi=2000 nmc=5000 seed=12541
monitor=(a0 alpha_age alpha_BT alpha_base alpha_trt alpha_V4 alpha0 var_b varbinterac);
ods output PostSummaries=PostSummaries;
odsexit output PostIntervals=PostIntervals;

/*** initial values ***/
parms a0=1 alpha_base=0 alpha_trt=0 alpha_BT=0 alpha_age=0 alpha_V4=0; /*one block*/
parms var_b=1;
parms varbinterac=1;

random b ~ normal(0, var=var_b) subject=ind;
random binterac~normal(0, var=varbinterac) subject=rand;

/*** priors ***/
prior a0 ~ normal(0, var = 10000);
prior alpha_base ~ normal(0, var = 10000);
prior alpha_trt ~ normal(0, var = 10000);
prior alpha_BT ~ normal(0, var = 10000);
prior alpha_age ~ normal(0, var = 10000);
prior alpha_V4 ~ normal(0, var = 10000);
prior var_b~igamma(0.01, scale=0.01);
prior varbinterac~igamma(0.01, scale=0.01);
We model

\[ \text{logmu} = a_0 + \alpha_{\text{Base}} \times (\log\text{base}_4 - \log\text{basebar}) + \alpha_{\text{trt}} \times (\text{trt} - \text{trtbar}) + \alpha_{\text{BT}} \times (\text{BT} - \text{BTbar}) + \alpha_{\text{age}} \times (\log\text{age} - \log\text{agebar}) + \alpha_{V4} \times (V4 - \text{V4bar}) + b + \text{binterac}; \]

\[ \mu = \exp(\text{logmu}); \]

\[ \text{model y ~ poisson(} \mu \text{);} \]

//*** compute the intercept in the original scale ***//
\[ \alpha_0 = a_0 - \alpha_{\text{Base}} \times \log\text{basebar} - \alpha_{\text{Trt}} \times \text{Trtbar} - \alpha_{\text{BT}} \times \text{BTbar} - \alpha_{\text{Age}} \times \log\text{Agebar} - \alpha_{V4} \times \text{V4bar}; \]

run;

- To try to improve convergency and mixing, centering is applied
- For this example, proc MCMC runs in about 15 sec.
library(rstan)
stan_code <-'
data {
  int<lower=0> N;
  int<lower=0> T;
  int<lower=0> y[N, T];
  int<lower=0> Trt[N];
  int<lower=0> V4[T];
  real log_Base4[N];
  real log_Age[N];
  real BT[N];
}

parameters {
  real a0;
  real alpha_Base;
  real alpha_Trt;
  real alpha_BT;
  real alpha_Age;
  real alpha_V4;
  real b1[N];
  real b[N, T];
  real<lower=0> sigmasq_b;
  real<lower=0> sigmasq_b1;
}

transformed parameters {
  real<lower=0> sigma_b;
  real<lower=0> sigma_b1;
  sigma_b <- sqrt(sigmasq_b);
  sigma_b1 <- sqrt(sigmasq_b1);
}

model {
  non useful (non info) priors:
  a0 ~ normal(0, 10000);
  alpha_Base ~ normal(0, 10000);
  alpha_Trt ~ normal(0, 10000);
  alpha_BT ~ normal(0, 10000);
  alpha_Age ~ normal(0, 10000);
  alpha_V4 ~ normal(0, 10000);
  sigmasq_b1 ~ inv_gamma(.001, .001);
  sigmasq_b ~ inv_gamma(.001, .001);

  log likelihood definition
  for(n in 1:N) {
    b1[n] ~ normal(0, sigma_b1);
    for(t in 1:T) {
      b[n, t] ~ normal(0, sigma_b);
      y[n, t] ~ poisson(exp(a0 + alpha_Base * (log_Base4[n])
                       + alpha_Trt * (Trt[n]) + alpha_BT * (BT[n])
                       + alpha_Age * (log_Age[n])
                       + alpha_V4 * (V4[t]) + b1[n] + b[n, t]));
    }
  }
}'

source("Bayes 2013_Poisson Data.R")

#compile
fit <- stan(model_code = stan_code, data = Data, iter = 1, chains = 1)

#run
fit2 <- stan(fit = fit, data = Data, iter = 3500, chains = 2, thin=1, warmup=1000)

plot(fit2)
traceplot(fit2, pars=c("alpha_Age","alpha_BT",
                      "alpha_Base","alpha_Trt","alpha_V4","a0",
                      "sigmasq_b","sigmasq_b1"))

library(MCMCpack) #Stan's traceplot overwritten...
mc = as.mcmc(as.matrix(fit2))
acfpplot(mc)
densityplot(mc)
HPDinterval(mc)
To be fair, thinning is not applied with both sampler.
SAS (with centering and thining 1/10) vs. Stan (without centering and no thinning)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Mean</th>
<th>se_mean</th>
<th>sd</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>n_eff</th>
<th>Rhat</th>
</tr>
</thead>
<tbody>
<tr>
<td>alpha_Age</td>
<td>0.48</td>
<td>0.01</td>
<td>0.36</td>
<td>0.24</td>
<td>0.48</td>
<td>0.71</td>
<td>2423</td>
<td>1</td>
</tr>
<tr>
<td>alpha_BT</td>
<td>0.35</td>
<td>0.00</td>
<td>0.21</td>
<td>0.20</td>
<td>0.35</td>
<td>0.49</td>
<td>2220</td>
<td>1</td>
</tr>
<tr>
<td>alpha_Base</td>
<td>0.89</td>
<td>0.00</td>
<td>0.14</td>
<td>0.80</td>
<td>0.89</td>
<td>0.97</td>
<td>2209</td>
<td>1</td>
</tr>
<tr>
<td>alpha_Trt</td>
<td>-0.95</td>
<td>0.01</td>
<td>0.42</td>
<td>-1.23</td>
<td>-0.95</td>
<td>-0.67</td>
<td>2466</td>
<td>1</td>
</tr>
<tr>
<td>alpha_V4</td>
<td>-0.10</td>
<td>0.00</td>
<td>0.09</td>
<td>-0.16</td>
<td>-0.10</td>
<td>-0.04</td>
<td>5000</td>
<td>1</td>
</tr>
<tr>
<td>a0</td>
<td>-1.39</td>
<td>0.02</td>
<td>1.24</td>
<td>-2.20</td>
<td>-1.40</td>
<td>-0.58</td>
<td>2473</td>
<td>1</td>
</tr>
<tr>
<td>sigmasq_b</td>
<td>0.13</td>
<td>0.00</td>
<td>0.03</td>
<td>0.11</td>
<td>0.13</td>
<td>0.15</td>
<td>696</td>
<td>1</td>
</tr>
<tr>
<td>sigmasq_b1</td>
<td>0.25</td>
<td>0.00</td>
<td>0.07</td>
<td>0.20</td>
<td>0.24</td>
<td>0.30</td>
<td>2619</td>
<td>1</td>
</tr>
</tbody>
</table>

HPD intervals
- Stan: alpha0: -3.615, 1.225
- SAS: alpha0: -4.481, 1.169
Stan: equivalence with / without centering

- Similar results, but...
- With centering: **10 sec.**
  - ‘Instant’ convergence
- Without centering: **6 min.**
  - Slower convergence
Survival data
From the Kidney example of R-INLA (http://www.r-inla.org/examples/volume-ii)

- Times to infection of kidney dialysis patients
- Data:
  - Time to infection in Month/10, $t_i$
  - Presence/Absence of infection
  - 2 types of catheter to be compared, $trt_i$
- Right-censored data
  - If infection, ‘time’ is the time of failure
- Model without censored data:
  $$t_i \sim E(\lambda_i)$$
  $$\lambda_i = \exp(\eta_i) \quad \eta_i = \beta_0 + trt_i \beta_1$$
  $$\beta_0 \sim N(0, 0.001)$$
  $$\beta_1 \sim N(0, 0.001)$$
SAS model

**PROC MCMC** data=survival.data outpost=postout thin=10 nbi=10000 nmc=30000 seed=12541
monitor=(beta0 beta1);

ods output PostSummaries=PostSummaries;
ods output PostIntervals=PostIntervals;

/*** initial values ***/
parms (beta0 beta1) 0;

/*** priors ***/
prior beta: ~ normal(0, var = 1000);

/*** model ***/
eta= beta0+beta1*placement;
/* if the distribution parameter is the regression, the log-likelihood for right censored data is as follows (log survival function + logpdf if event is observed ) */
llike=event*(eta)-time*exp(eta);
model general(llike);
run;

\[
L(\theta) = \prod_{i \in r.c.} \Pr(T > T_i|\theta) \prod_{T_i \in unc} \Pr(T = T_i|\theta)
\]
Using the ‘general’ statement, proc MCMC will take care to sum the posterior log-density of each observation to compute the log-posterior.
**SAS summary**

### Posterior Summaries

<table>
<thead>
<tr>
<th>Parameter</th>
<th>N</th>
<th>Mean</th>
<th>Standard Deviation</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta0</td>
<td>3000</td>
<td>-0.6699</td>
<td>0.6029</td>
<td>-1.0681</td>
<td>-0.6771</td>
<td>-0.2740</td>
</tr>
<tr>
<td>beta1</td>
<td>3000</td>
<td>-0.5419</td>
<td>0.4029</td>
<td>-0.8030</td>
<td>-0.5275</td>
<td>-0.2751</td>
</tr>
</tbody>
</table>

### Posterior Intervals

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Alpha</th>
<th>Equal-Tail Interval</th>
<th>HPD Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta0</td>
<td>0.050</td>
<td>-1.8694</td>
<td>0.5550</td>
</tr>
<tr>
<td>beta1</td>
<td>0.050</td>
<td>-1.3911</td>
<td>0.2186</td>
</tr>
</tbody>
</table>

### Effective Sample Sizes

<table>
<thead>
<tr>
<th>Parameter</th>
<th>ESS</th>
<th>Autocorrelation Time</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta0</td>
<td>2670.7</td>
<td>1.1233</td>
<td>0.8902</td>
</tr>
<tr>
<td>beta1</td>
<td>2747.9</td>
<td>1.0917</td>
<td>0.9160</td>
</tr>
</tbody>
</table>
Stan model with prior as in SAS

```r
library(rstan)
stan_code <- 'data {
  int<lower=0> N;
  real  time[N];
  real  event[N];
  real placement[N];
}
parameters {
  real beta[2];
}
model {
  real eta[N];
beta ~ normal(0, 1000);
  for(i in 1:N) {
    lp__ <- lp__ + event[i]*(eta[i]-time[i])*exp(eta[i]);
  }
}
'

source("Bayes 2013_Survival Data.R »)

fit <- stan(model_code = stan_code, data = Data, iter = 1000, chains = 1)
fit2 <- stan(fit = fit, data = Data, iter = 15000, chains = 2, thin=10, warmup=5000)
```

Here, `lp__` is a reserved word that makes clear that the model block evaluate the log posterior as a sum of observations posterior densities.

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>se_mean</th>
<th>sd</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>n_eff</th>
<th>Rhat</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[1]</td>
<td>-0.66</td>
<td>0.01</td>
<td>0.61</td>
<td>-1.06</td>
<td>-0.66</td>
<td>-0.25</td>
<td>2267</td>
<td>1</td>
</tr>
<tr>
<td>beta[2]</td>
<td>-0.55</td>
<td>0.01</td>
<td>0.41</td>
<td>-0.81</td>
<td>-0.54</td>
<td>-0.28</td>
<td>2265</td>
<td>1</td>
</tr>
</tbody>
</table>

```r
> HPDinterval(as.mcmc(as.matrix(fit2)))

<table>
<thead>
<tr>
<th></th>
<th>lower</th>
<th>upper</th>
</tr>
</thead>
<tbody>
<tr>
<td>beta[1]</td>
<td>-1.822407</td>
<td>0.5413661</td>
</tr>
<tr>
<td>beta[2]</td>
<td>-1.316363</td>
<td>0.2675095</td>
</tr>
</tbody>
</table>
```
library(rstan)
stan_code <- '
  data {
    int<lower=0> N;
    real  time[N];
    real  event[N];
    real  placement[N];
  }

  parameters {
    real beta[2];
  }

  model {
    real eta[N];
    #beta ~ normal(0,  1000);
    for(i in 1:N) {
      lp__ <- lp__ + event[i]*(eta[i])-time[i]*exp(eta[i]);
    }
  }

source("Bayes 2013_Survival Data.R »)

fit <- stan(model_code = stan_code, data = Data, iter = 1000, chains = 1)
fit2 <- stan(fit = fit, data = Data, iter = 15000, chains = 2, thin=10,warmup=5000)
(Other) model check
The model is surprisingly easy to write in INLA

source('Bayes 2013_Survival Data.R')
library(INLA)
# inla.surv() automatically handles right-censored data
# 'time' is the follow up time
formula = inla.surv(time, event) ~ placement

# The prior assumed for intercept and regression coefficient are same as by default
# Exponential model is given by
model=inla(formula,family="exponential", data= data, verbose=TRUE)

summary(model)

Fixed effects:

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>0.025quant</th>
<th>0.5quant</th>
<th>0.975quant</th>
<th>kld</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Intercept)</td>
<td>-0.6243</td>
<td>0.5979</td>
<td>-1.8395</td>
<td>-0.610</td>
<td>0.5097</td>
<td>4e-04</td>
</tr>
<tr>
<td>placement</td>
<td>-0.5335</td>
<td>0.3969</td>
<td>-1.3253</td>
<td>-0.529</td>
<td>0.2323</td>
<td>0e+00</td>
</tr>
</tbody>
</table>

m = model$marginals.fixed$placement
inla.hpdmarginal(0.95, m)

<table>
<thead>
<tr>
<th>level:0.95</th>
<th>low</th>
<th>high</th>
</tr>
</thead>
</table>
|                | -1.317485 | 0.2390768 | (similar to Stan and SAS results with Normal prior)
Parameter correlations give a hard time to the samplers
  - We don’t succeed to overcome this, using proc MCMC
    Centering regressor certainly helps but this seems not sufficient
  - Stan is readily built to take a special care of the correlations in using the gradient to define its proposals

A plurality of tools exist each with specificities

Stan is very flexible and powerful, yet it requires more coding and complex installation
  - ...while your IT department takes care of SAS and so virtually everybody can play with proc MCMC
Conclusion

All samplers (proc MCMC, Stan, INLA) can handle non-trivial likelihood by letting the user simply defines (nearly) any log posterior he wants

- The special case of differential equations?
Thank you!