BAYES 2013
21-23 May ERASMUS UNIVERSITY ROTTERDAM

Arlenda
Advance your business with statistics

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

$$p(\boldsymbol{\theta} \mid \mathbf{y}) = \frac{p(\mathbf{y} \mid \boldsymbol{\theta}) \;\; p(\boldsymbol{\theta})}{p(\mathbf{y})}$$

$$p(\boldsymbol{\theta} \mid \mathbf{y}) \propto \mathcal{L}(\boldsymbol{\theta} \mid \mathbf{y}) \; . \; p(\boldsymbol{\theta})$$

$$\text{Posterior} \propto \text{Likelihood} \;\; \times \;\; \text{Prior}$$

- Prediction of a new observation

$$p(\tilde{y} \mid \mathbf{y}) = \boxed{\int_{\boldsymbol{\theta}} \boxed{p(\tilde{y} \mid \boldsymbol{\theta})} \, p(\boldsymbol{\theta} \mid \mathbf{y}) \, d\boldsymbol{\theta}}$$

Likelihood given the parameters

Predictive density integrating out the parameter distribution

Bayesian principle

- Let's consider that θ is the parameter of interest (ex: treatment effect)

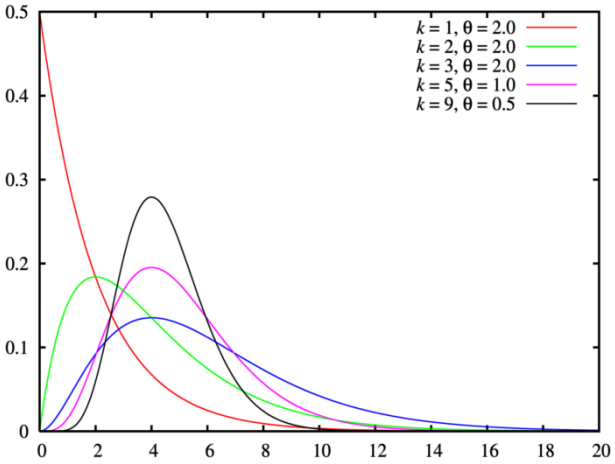
θ is treated as random variables

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

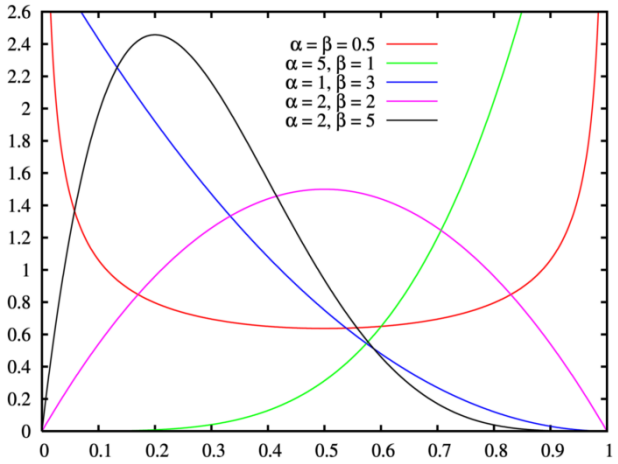
Bayesian principle

Examples of prior distributions

Gamma distributions



Beta distributions



- Prior **distribution** -> Specify the domain of plausible values

  -> Specify the weights given to these values

- 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 θ: $p(y|\theta)$
   - Based solely on data

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

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

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

$$\text{draw } \boldsymbol{\theta}^{(s)} \text{ from the joint posterior density } p(\boldsymbol{\theta} \mid \mathbf{y}),$$

$$\text{draw } \tilde{y}^{(s)} \text{ from the model } p(\tilde{y} \mid \boldsymbol{\theta}^{(s)}),$$

$$s = 1, ..., n^* \text{ is the number of samples to draw.}$$

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)

  } Adaptative rejection sampling

  - 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 Metropolis-Hasting algorithm

- Basic algorithm

For $s = 1$ to n^*

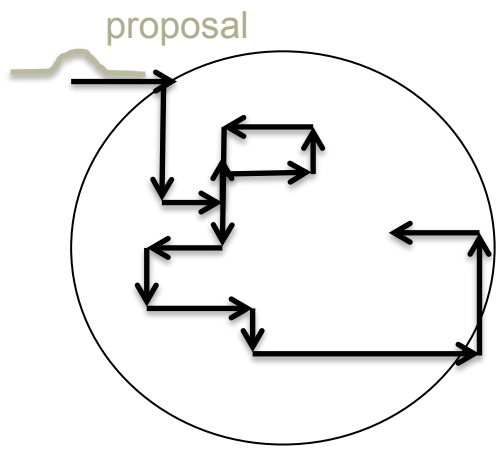
1. From a ~~symmetric~~ proposal distribution $q(\boldsymbol{\theta}^{(s)} \mid \boldsymbol{\theta}^{(s-1)})$, draw a new candidate vector $\boldsymbol{\theta}_t$,

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

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

End

- Can be easily used for drawing univariate parameters conditional to the previous values (s-1) of the others



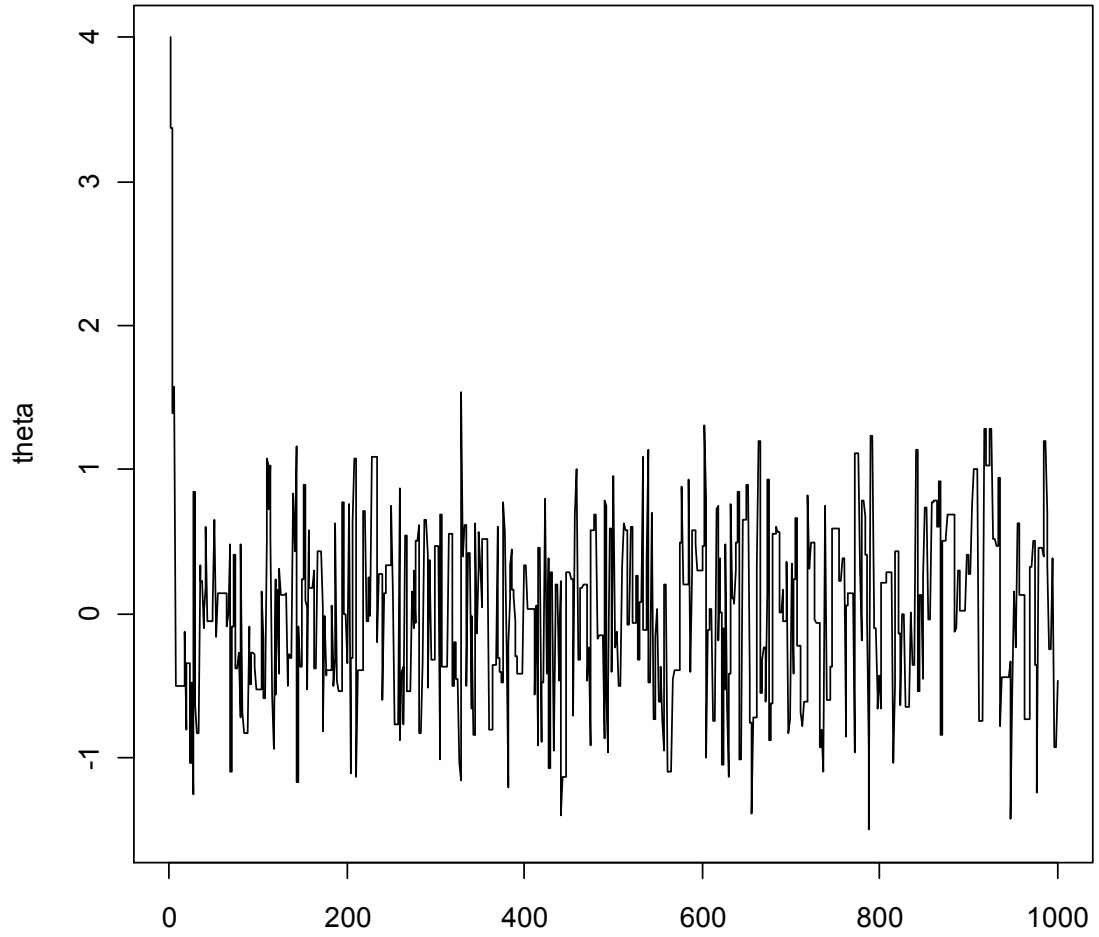
Example of hand-made MCMC simulations

```
#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
```

Metropolis sampling

```
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

Burn-in

- 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

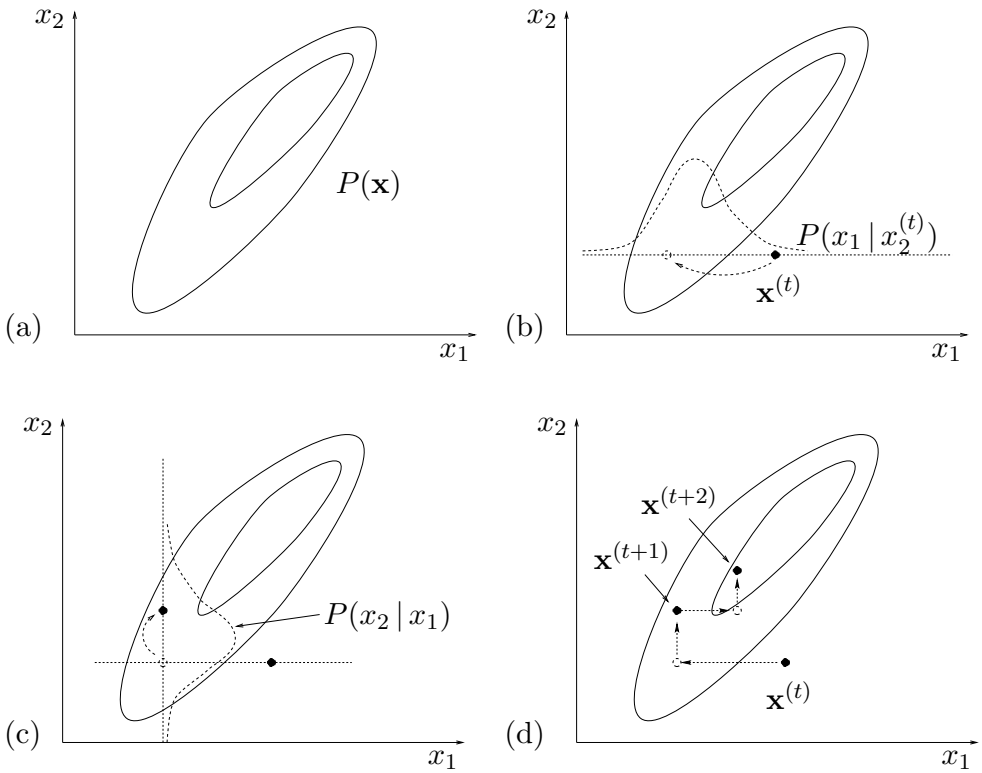
$$\text{For } j = 1 \text{ to } m$$

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

$$\text{End}$$

  - 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

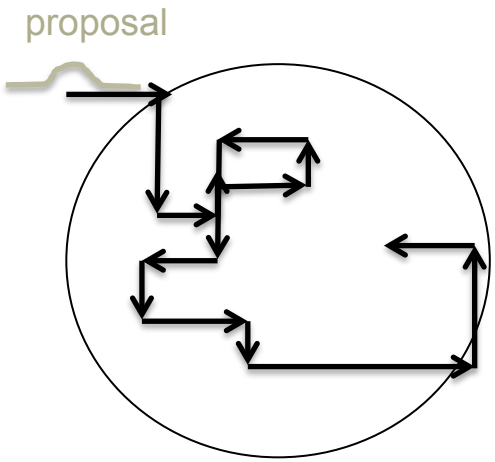
Gibbs sampling vizualized

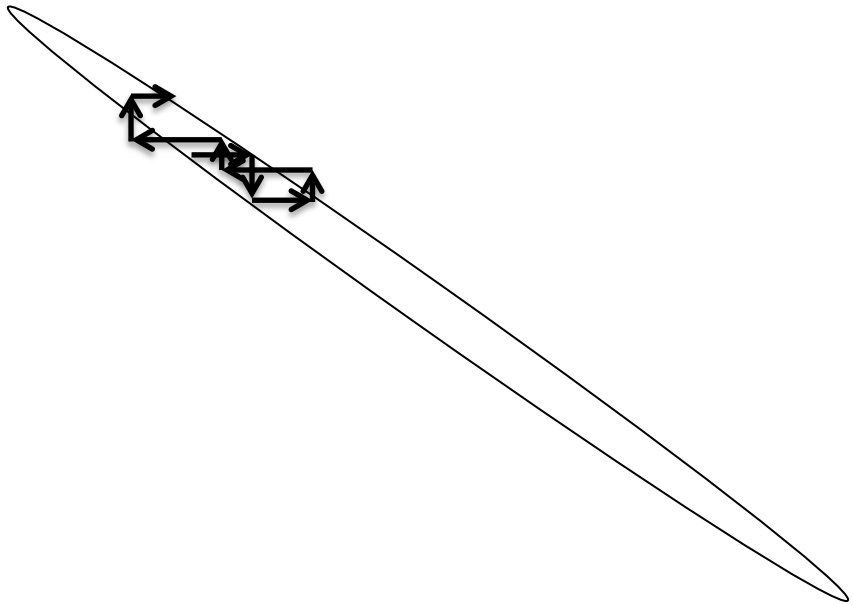


David J.C. MacKay, Information Theory, Inference, and Learning Algorithms, Cambridge University Press, 2003

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 → 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: mu[i] <- alpha + beta * (x[i] - 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)

  → The target density defines a potential energy function using Hamiltonian equations

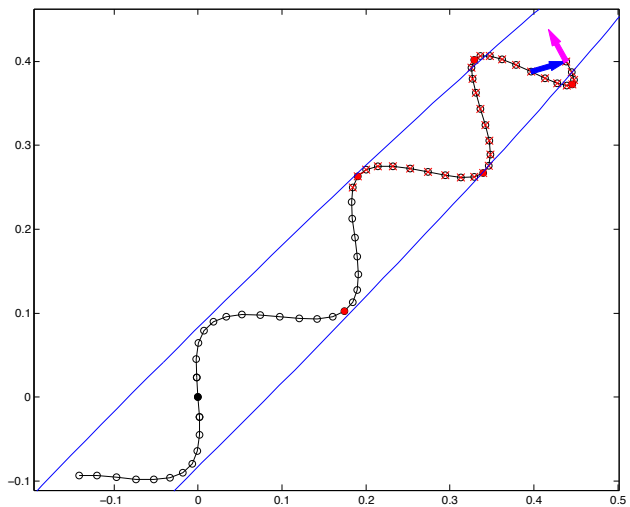
  → One sampling iteration consists in moving on the posterior following these dynamics

  - instead of moving using a simpler proposal distribution

- Theory terminology is often hard for statisticians without a good knowledge of physics

Hamiltonian Monte-Carlo

- 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



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

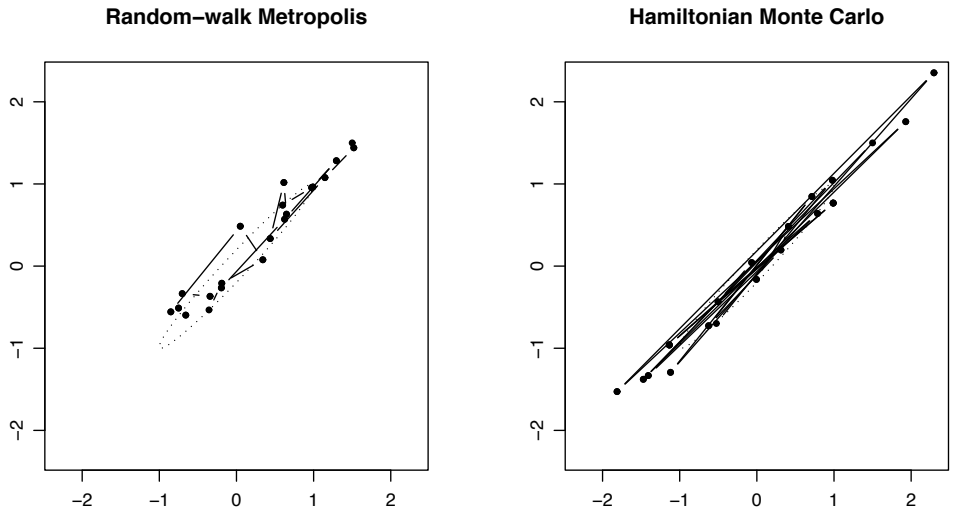
Hoffman, Matthew D. and Andrew Gelman. In press. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. Journal of Machine Learning Research.

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

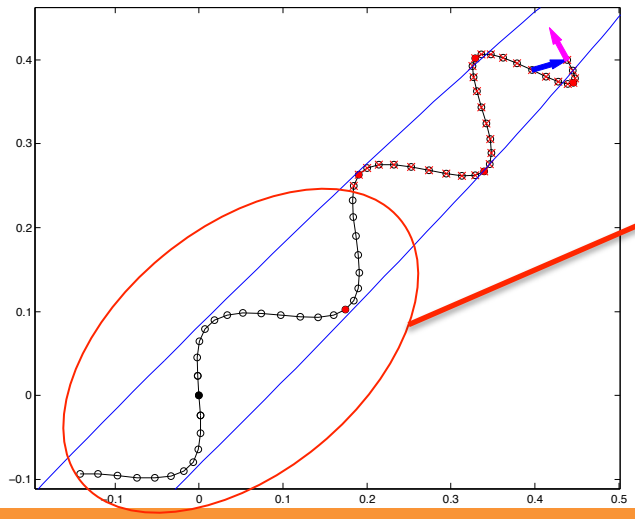
    Explore the posterior distribution more efficiently using several leapfrogs to reduce autocorrelation



Neal, R. MCMC using Hamiltonian dynamics, in Handbook of Markov Chain Monte Carlo, Brooks et al, Chapman & Hall, 2010

NUTS

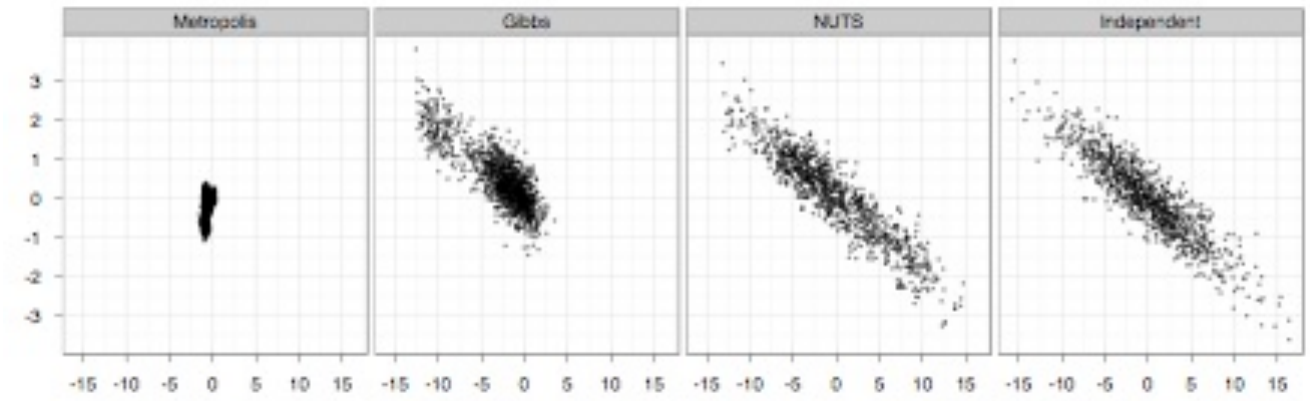
- Using the leapfrog function, two parameters have to be tuned
  - The size of the leap (the step)
  - The number of leaps
- Tuning them is a complex task that may require many additional runs
- The No-U-Turn Sampler (NUTS) is an improvement of HMC that have routines to tune these parameters on-the-fly



Choose randomly among the valid candidates the next sample/position

NUTS vs. random Walk

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



Hoffman, Matthew D. and Andrew Gelman. In press. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. Journal of Machine Learning Research.

Diagnostic tools

ESS and Geweke

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

$$\mathrm{ESS} = \frac{n}{\tau} = \frac{n}{1 + 2\sum_{k=1}^{\infty} \rho_k(\theta)}$$

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

MCSE

- **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 $\hat{\sigma}_i/\sqrt{m}$. , the procedures use the following formula to include ESS :

$$\widehat{\mathrm{Var}}(\bar{\theta}_i) = \frac{1+2\sum_{k=1}^{\infty}\rho_k(\theta_i)}{n} \cdot \frac{\sum_{t=1}^{n}\left(\theta_i^t - \bar{\theta}_i\right)^2}{(n-1)}$$

  - 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-Rubin-Brook

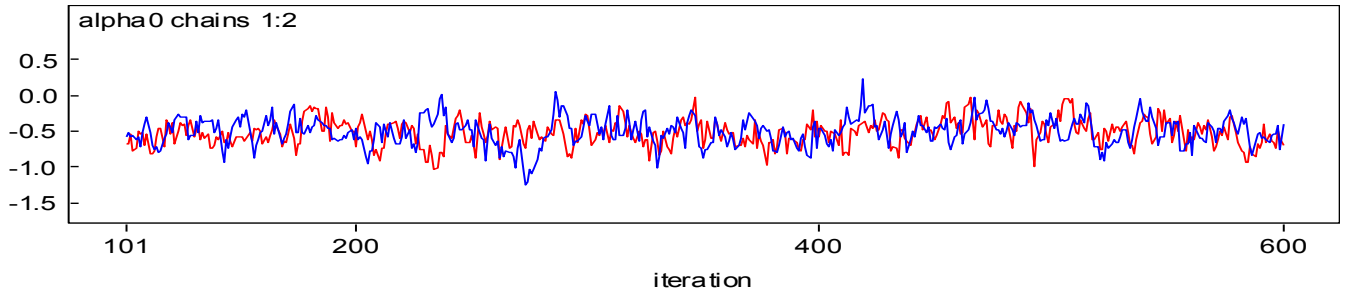
Diagnostic tool:

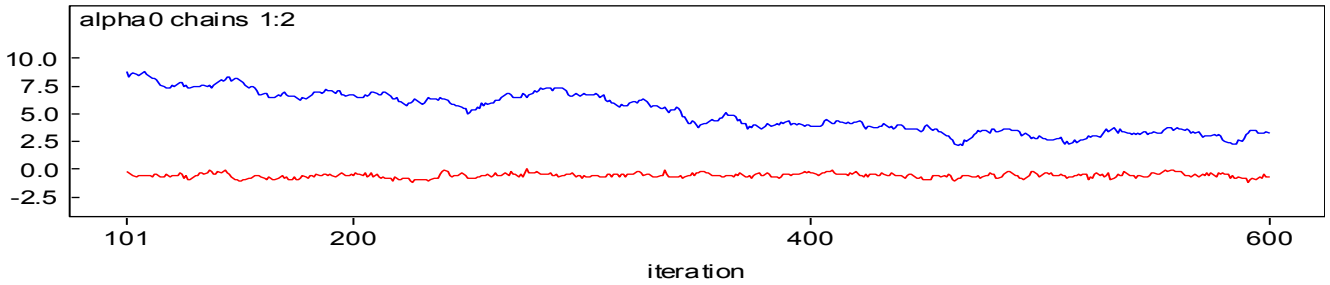
Gelman-Rubin Diagnostic

- 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.

Gelman-Rubin-Brook

Examples where convergence seems reasonable (top) and unreasonable (bottom)





Gelman-Rubin-Brook

- 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

$$W = \frac{1}{m}\sum_{j=1}^{m} s_j^2 \qquad s_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(\theta_{ij} - \bar{\theta}_j)^2$$

  - Between chain variance

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

  - Total variance

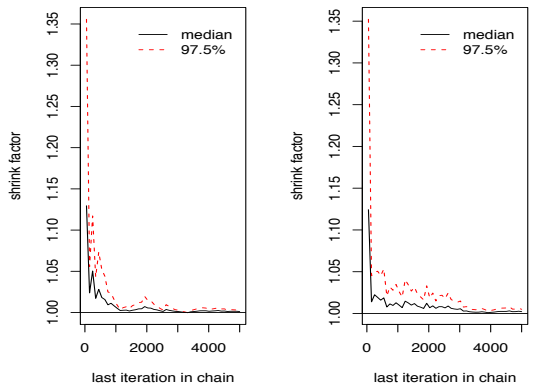
$$\hat{\mathrm{Var}}(\theta) = (1 - \frac{1}{n})W + \frac{1}{n}B$$

Gelman-Rubin-Brook

- The R statistic

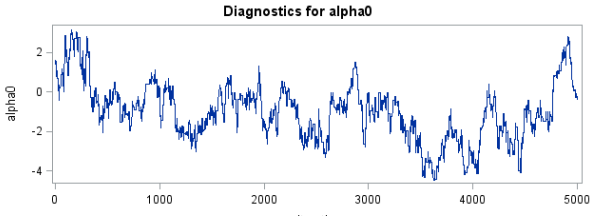
$$\hat{R} = \sqrt{\frac{\hat{\mathrm{Var}}(\theta)}{W}}$$

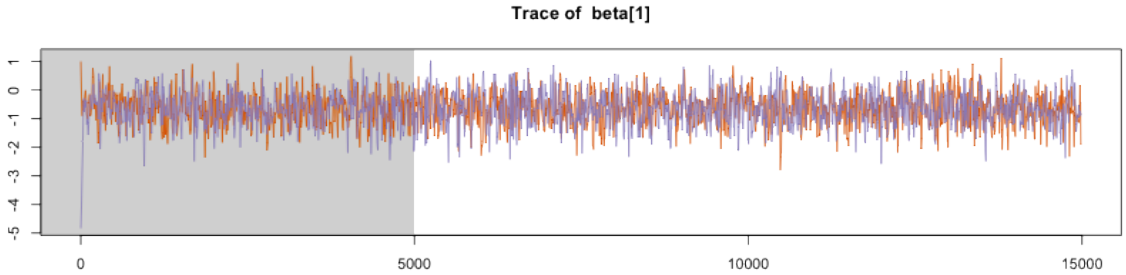
- R should be very close to 1 in case of convergence
- Local optima convergence (with poor initial values all in the same area) may not be identified, even if R is close to 1
- To do with all chains of parameters
- Potential scale reduction factor (PSRF) : Compute R through the iterations (including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



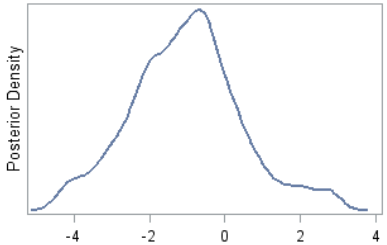
Other classical tools

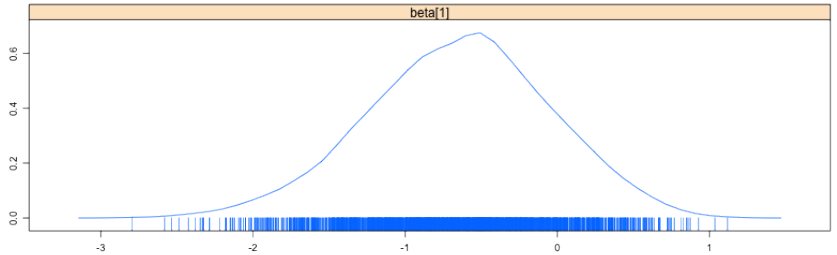
- Trace plots



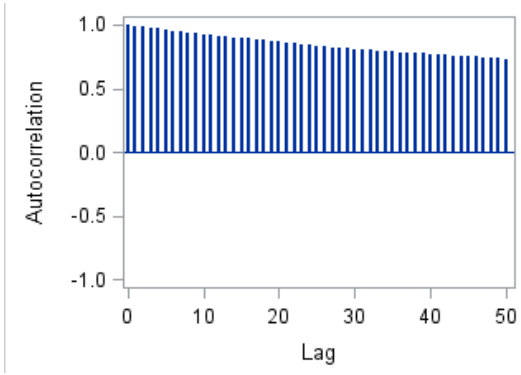


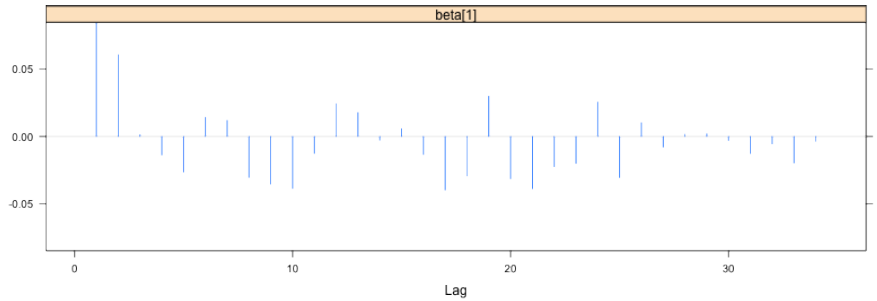
- Densities





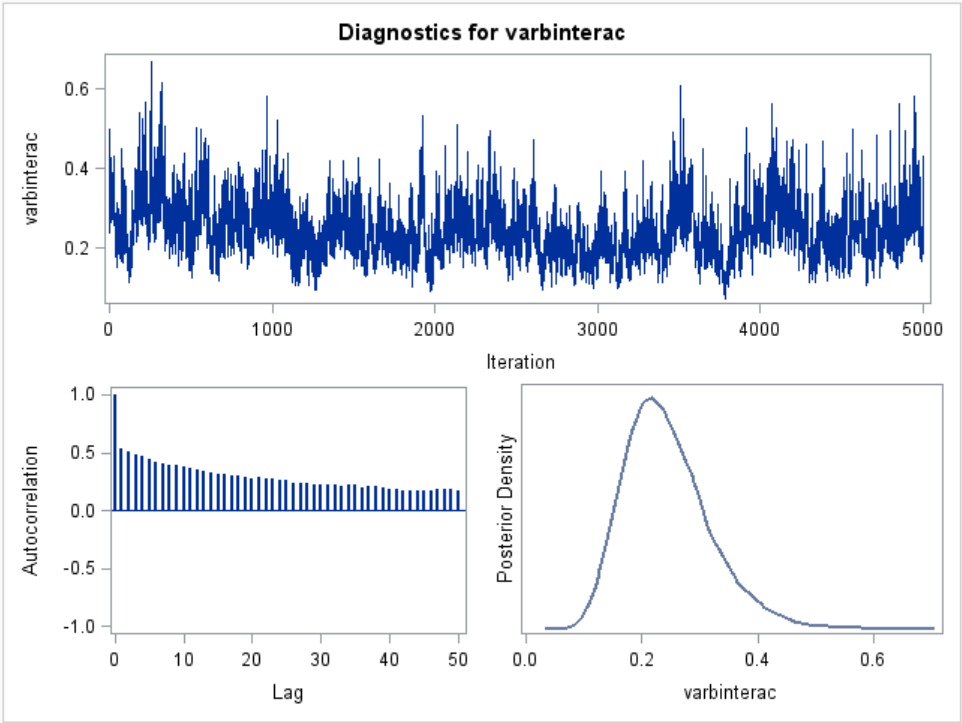
- Autocorrelations





Other classical tools

- SAS default output for one chain/parameter



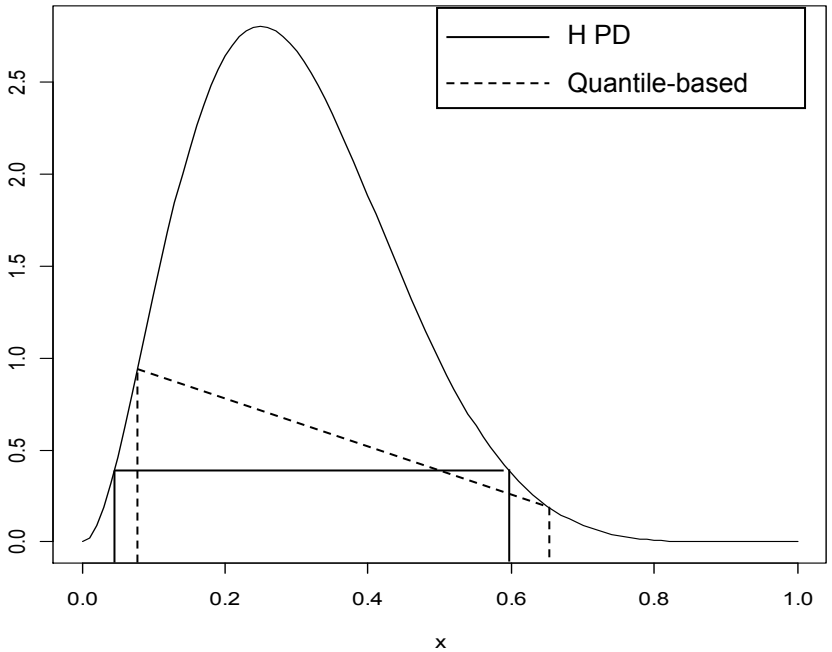
Intervals

- 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

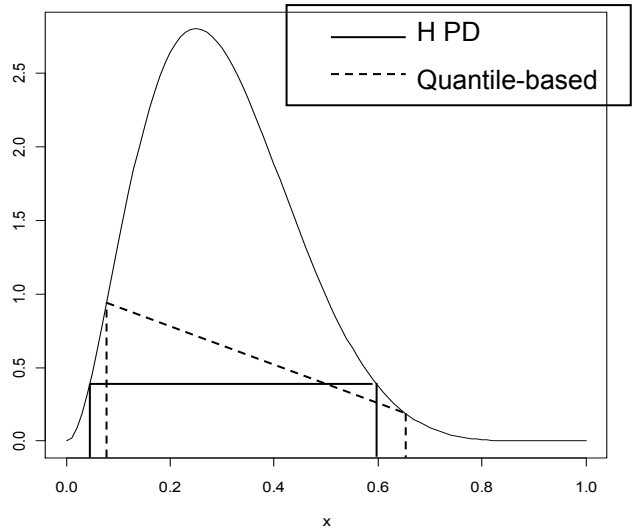
- Estimates and intervals

```
          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
beta[1]  -0.6 0        0.6  -1.9  -1.0  -0.6  -0.2   0.5  1793    1
beta[2]  -0.6 0        0.4  -1.3  -0.8  -0.5  -0.3   0.2  1748    1
```

> HPDinterval(mc,0.95)

```
            lower       upper
beta[1]  -1.762674   0.5780758
beta[2]  -1.284624   0.2594099
```

Posterior Intervals					
Parameter	Alpha	Equal-Tail Interval		HPD Interval	
beta0	0.050	-1.8694	0.5550	-1.9106	0.4782
beta1	0.050	-1.3911	0.2186	-1.3095	0.2707



Convergence checks

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 reparamerizations) that will help to speed convergence.

Proc MCMC

SAS code (1)

```
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;
```

SAS code (2)

```
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

SAS code (3)

```
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 : intial values
- prior: prior distributions

SAS code (4)

```
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)) ;
```

- model : likelihood function

Proc MCMC in SAS 9.3

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

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

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

```
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);
```

Proc MCMC in SAS 9.3

- **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.

Proc MCMC in SAS 9.3

- **RANDOM statement**

  - Used for hierarchical models

  E.g. for univariate random effect:

```
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:

```
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.

  - **PREDDIST OUTPRED**=SAS-data-set < **NSIM**=n > <**COVARIATES**=SAS-data-set > < **STATISTICS**=options > **;**

How to handle several chains ?

```
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;
```

```
%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;

     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;
%gmcmc;
```

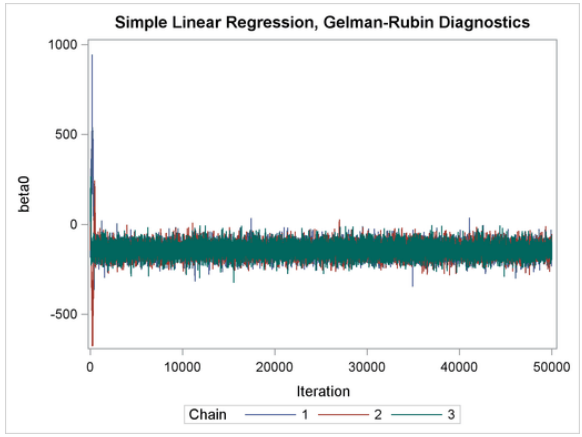
http://support.sas.com/documentation/cdl/en/statug/63962/HTML/default/viewer.htm#statug_mcmc_sect066.htm

Multiple chains diagnostics with SAS

- Additional to the basic diagnistics (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)

Stan installation

- 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

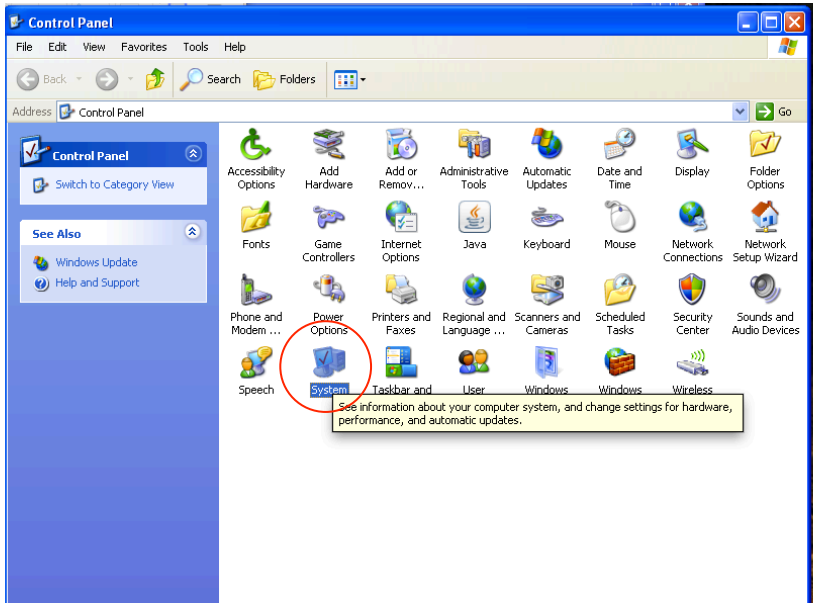
- R is readily available on http://www.r-project.org
- Rstudio (http://www.rstudio.com) is not mandatory, but is recommended to help
  - edit files (R scripts, report Sweave files, possibly C/C++ files)
  - manage projects
  - manage packages
  - etc.
- Both R and Rstudio are free and open source !

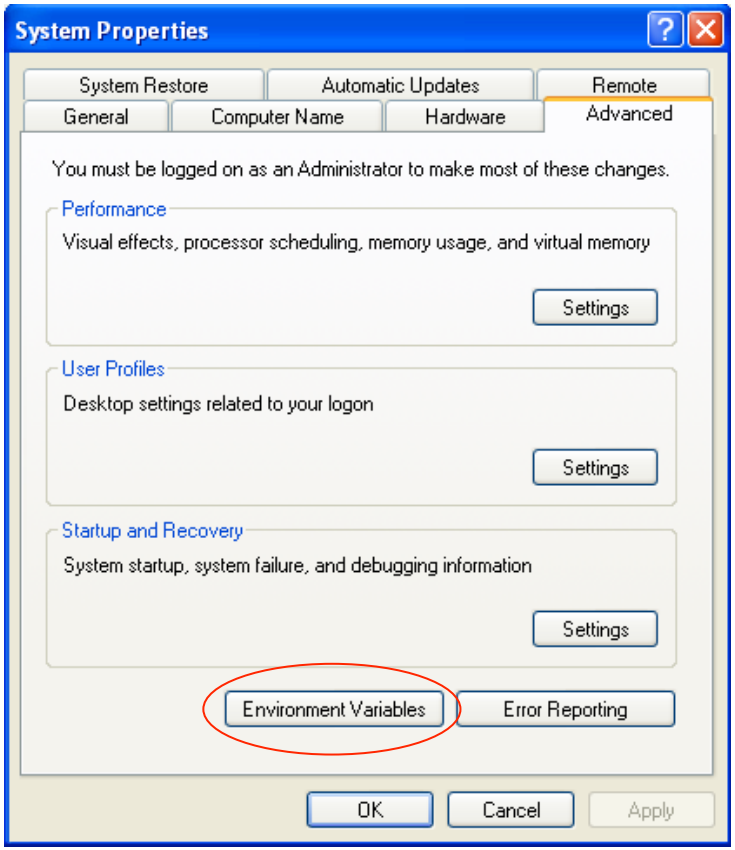
Stan: prerequisites

- 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 developped 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 personnal 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

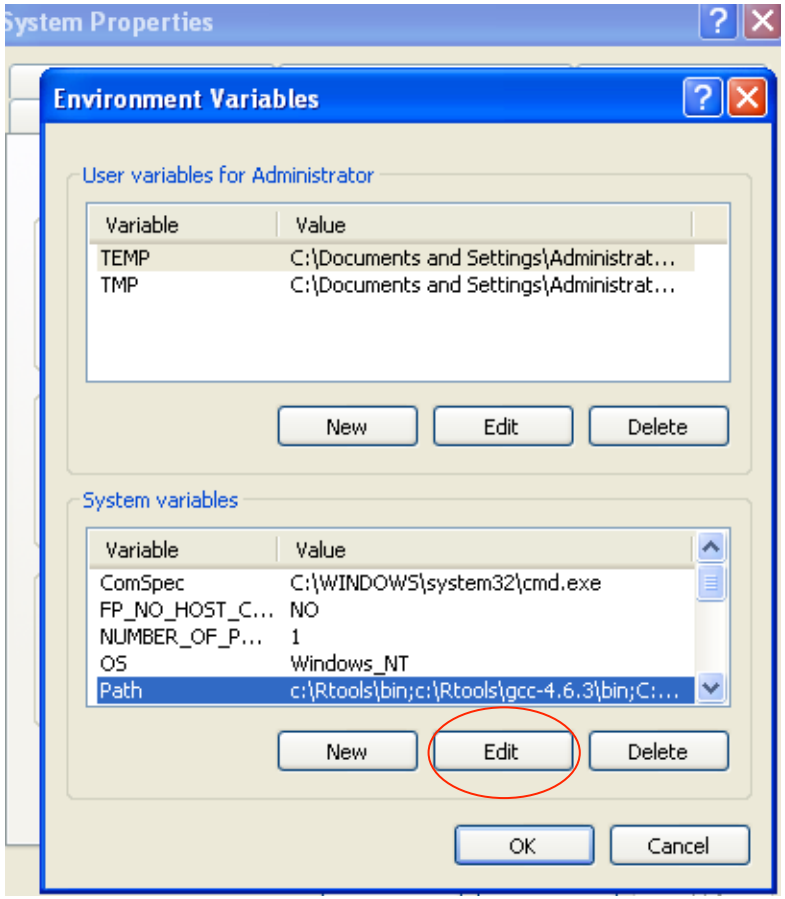
Stan: prerequisites

- On some computers, the last Rtools installation dialog may fail
  - Possible to edit the Path manually





Stan: prerequisites



- 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

```
# 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 <u>is</u> 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 ignore 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 parameters

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

```
transformed parameters {
#Executed at each leapfrog
real<lower=0> sigma_b;
sigma_b <- sqrt(sigmasq_b);
}
```

Stan language

- Block ‘model’
  - Contains (possibly) priors
  - Contains likelihood definition in a BUGS-like style

```
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);
        }
```

Stan language

- ‘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

Stan language

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

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

model {
  beta ~ normal(0,  1000);
  for(i in 1:N) {
   <model statement >
 }
```

```
# Or,
for (n in 1:N)  y[n] ~ bernoulli(theta);
# is equivalent to the vectorized form,
y ~ bernoulli(theta);
```

Stan Language

- 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

```
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

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

Patient	y_1	y_2	y_3	y_4	Trt	Base	Age
1	5	3	3	3	0	11	31
2	3	5	3	3	0	11	30
3	2	4	0	5	0	6	25
4	4	4	1	4	0	8	36
....							
8	40	20	21	12	0	52	42
9	5	6	6	5	0	12	37
....							
59	1	4	3	2	1	12	37

- Seizure counts in a randomised trial of anti-convulsant therapy in epilepsy
  - More on this model in the previous presentation !

SAS code

```
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;
    ods 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);
```

```
/*** model ***/
      logmu= a0+alpha_base*(logbase4-logbasebar)
                                          +alpha_trt*(trt-trtbar)
                                          +alpha_BT*(BT-BTbar)
                                          +alpha_age*(logage-logagebar)
                                          +alpha_v4*(V4-v4bar)
                                          +b+binterac;
      mu=exp(logmu);
      model y ~ poisson(mu);
/*** compute the intercept in the original scale ***/
     alpha0 = a0 - alpha_Base * logbasebar - alpha_Trt * Trtbar - alpha_BT * BTbar - alpha_Age *
 logAgebar - alpha_V4 * V4bar;

run;
```

- To try to improve convergency and mixing, centering is applied
- For this example, proc MCMC runs in about 15 sec.

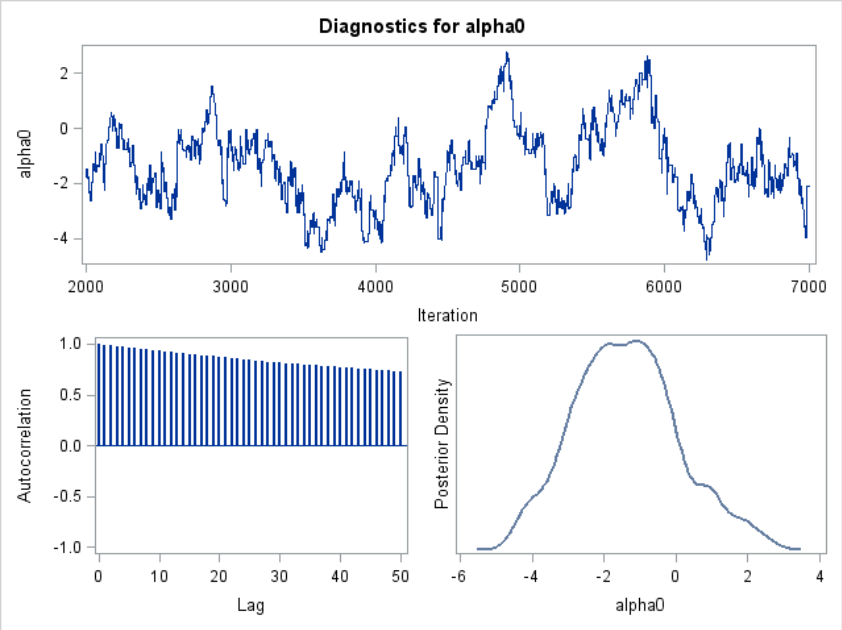
Stan code

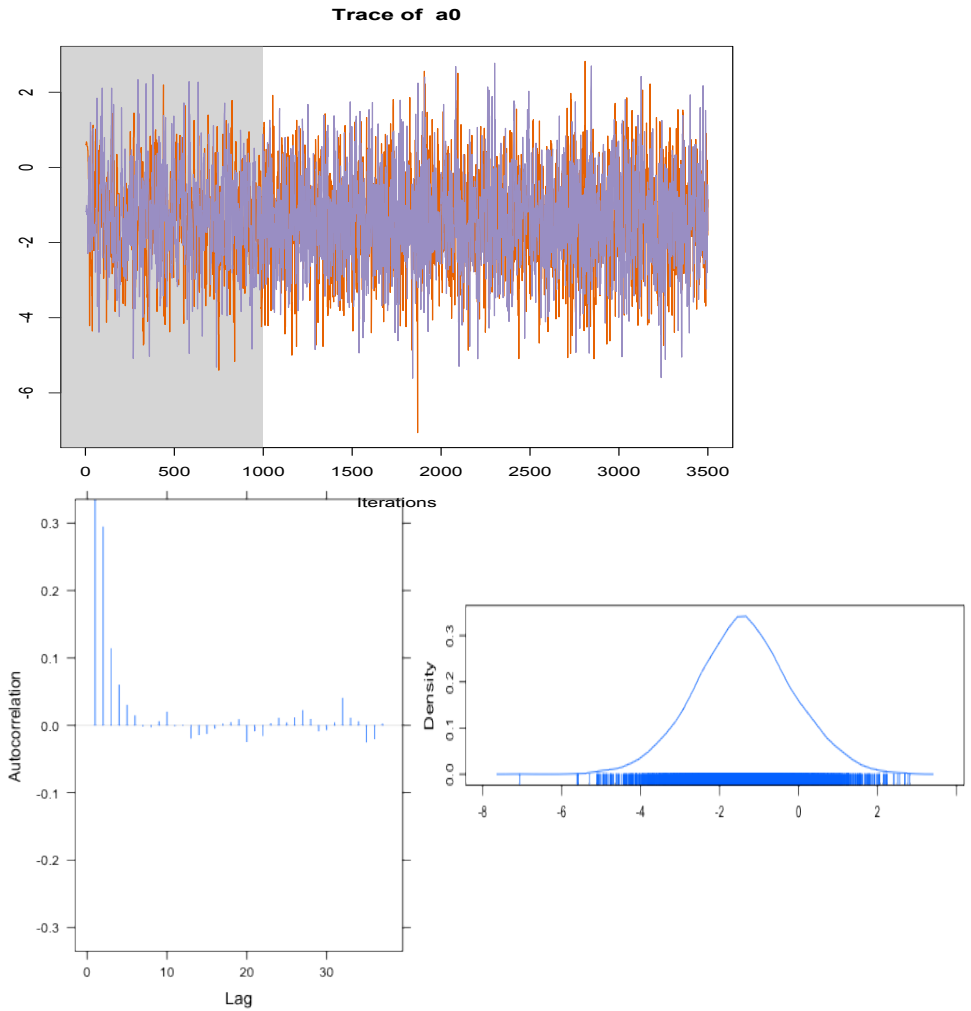
```
library(rstan)
stan_code <- '
data {
int<lower=0> N;
int<lower=0> T;
int<lower=0> y[N, T];
int<lower=0> Trt[N];
int<lower=0> V4[T];
real  log_Base4[N];
real  log_Age[N];
real  BT[N];
}
parameters {
real  a0;
real  alpha_Base;
real  alpha_Trt;
real  alpha_BT;
real  alpha_Age;
real  alpha_V4;
real  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))
acfplot(mc)
densityplot(mc)
HPDinterval(mc)
```

SAS (with centering) vs. Stan (without centering)

- To be fair, thinning is not applied with both sampler





SAS (with centering and thining 1/10) vs. Stan (without centering and no thining)

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

Posterior Summaries

Parameter	N	Mean	Standard Deviation	Percentiles 25%	50%	75%
a0	5000	1.6030	0.0783	1.5491	1.6018	1.6524
alpha_age	5000	0.4986	0.4272	0.2400	0.4954	0.8023
alpha_BT	5000	0.3552	0.1990	0.2352	0.3379	0.4909
alpha_base	5000	0.8466	0.1046	0.7707	0.8406	0.9125
alpha_trt	5000	-0.9464	0.4313	-1.2372	-0.9038	-0.6945
alpha_V4	5000	-0.0683	0.1000	-0.1369	-0.0677	0.00447
alpha0	5000	-1.3713	1.4520	-2.4103	-1.4047	-0.4387
var_b	5000	0.2354	0.0716	0.1848	0.2252	0.2741
varbinterac	5000	0.2354	0.0711	0.1843	0.2248	0.2753

Parameter	ESS
a0	14.5
alpha_age	19.6
alpha_BT	13.1
alpha_base	29.7
alpha_trt	12.1
alpha_V4	37.3
alpha0	19.8
var_b	94.9
varbinterac	97.2

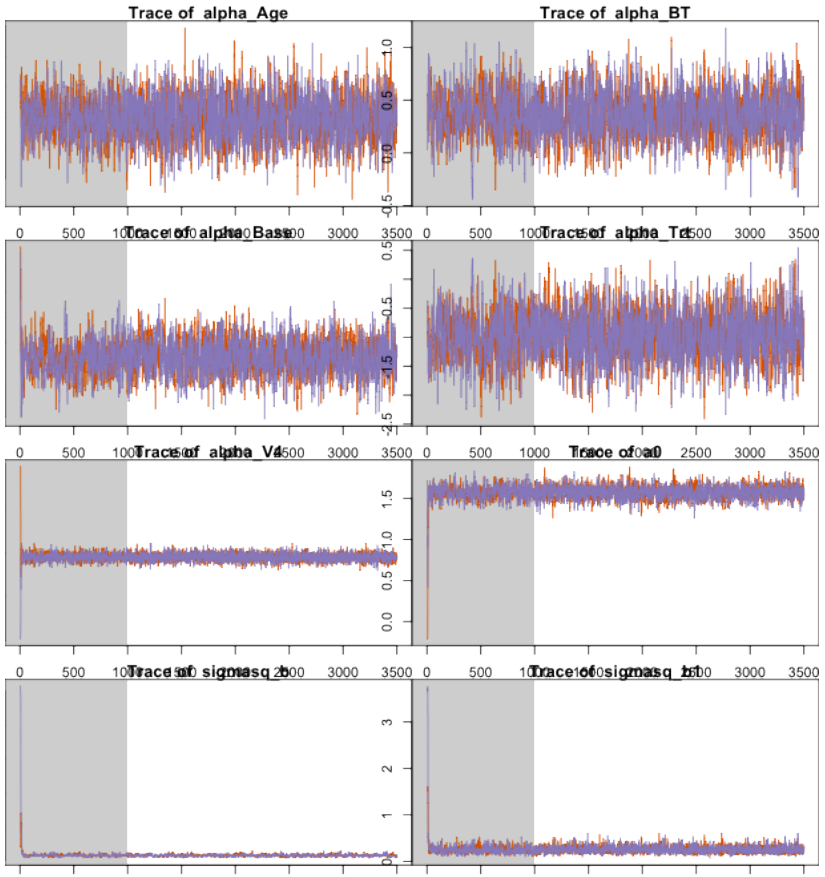
HPD intervals

Stan: alpha0 -3.615 1.225

SAS: alpha0 -4.481 1.169

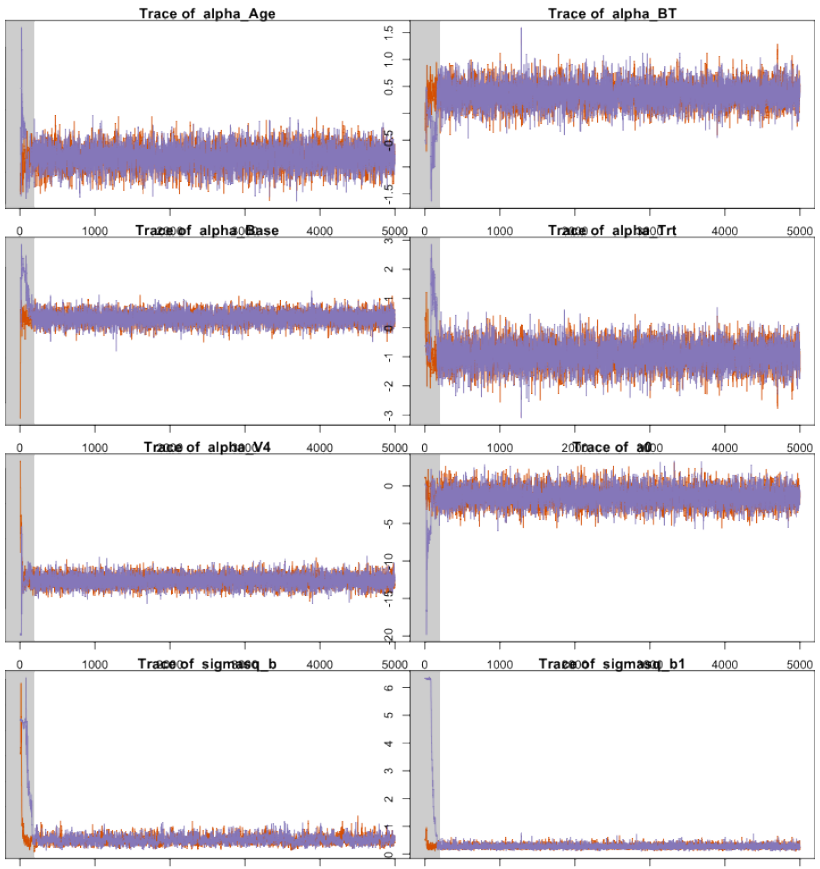
Stan: equivalence with / without centering

- Similar results, but...
- With centering: **10 sec.**
  - 'Instant' convergence



Without centering: **6 min.**

Slower convergence



Survival data

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:

$$t_i \sim E(\lambda_i)$$

$$\lambda_i = \exp(\eta_i) \qquad \eta_i = \beta_0 + trt_i \beta_1$$

$$\beta_0 \sim N(0, 0.001)$$

$$\beta_1 \sim N(0, 0.001)$$

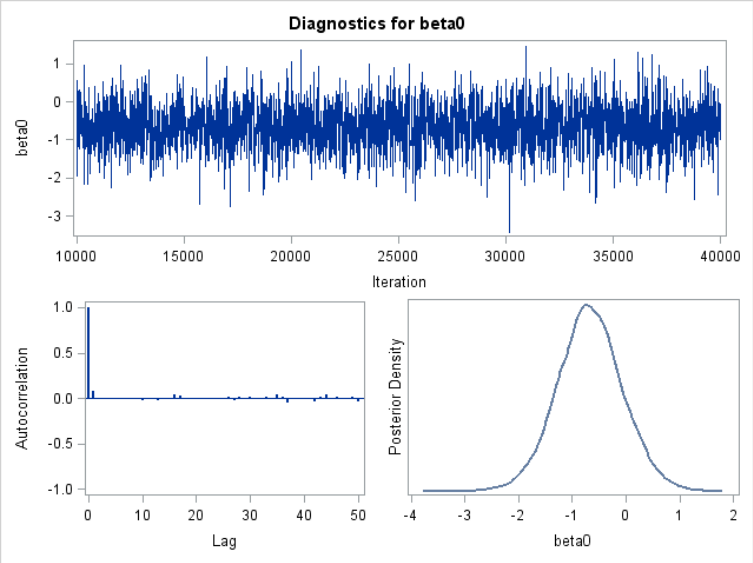
SAS model

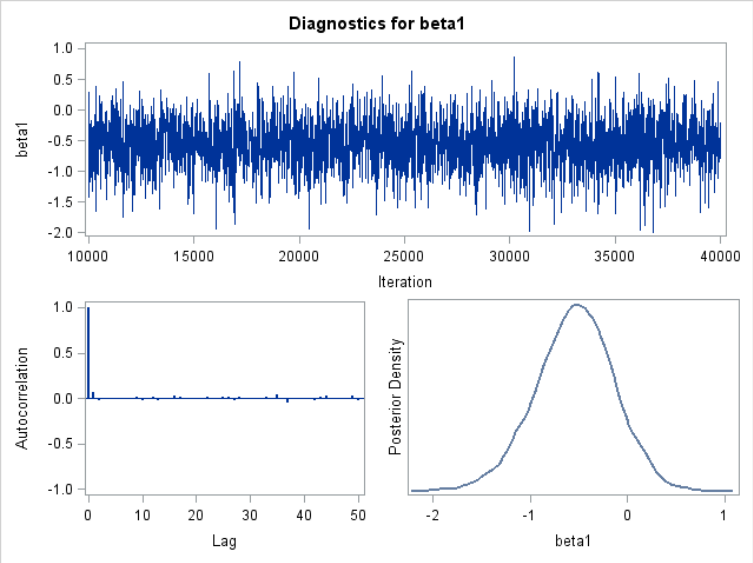
```
PROC MCMC data=survival.data outpost=postout thin=10 nbi=10000 nmc=30000
seed=12541
      monitor=(beta0 beta1);

      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;
```

$$L(\theta) = \prod_{i \in r.c.} \Pr(T > T_i|\theta) \prod_{T_i \in unc.} \Pr(T = T_i|\theta)$$

SAS model

- 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						
				Percentiles		
Parameter	N	Mean	Standard Deviation	25%	50%	75%
beta0	3000	-0.6699	0.6029	-1.0681	-0.6771	-0.2740
beta1	3000	-0.5419	0.4029	-0.8030	-0.5275	-0.2751

Posterior Intervals					
Parameter	Alpha	Equal-Tail Interval		HPD Interval	
beta0	0.050	-1.8694	0.5550	-1.9106	0.4782
beta1	0.050	-1.3911	0.2186	-1.3095	0.2707

Effective Sample Sizes			
Parameter	ESS	Autocorrelation Time	Efficiency
beta0	2670.7	1.1233	0.8902
beta1	2747.9	1.0917	0.9160

Stan model with prior as in SAS

```
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) {
    eta[i]<- beta[1]+ beta[2]*placement[i];
    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

	mean	se_mean	sd	25%	50%	75%	n_eff	Rhat
beta[1]	-0.66	0.01	0.61	-1.06	-0.66	-0.25	2267	1
beta[2]	-0.55	0.01	0.41	-0.81	-0.54	-0.28	2265	1

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

	lower	upper
beta[1]	-1.822407	0.5413661
beta[2]	**-1.316363**	**0.2675095**

Stan model with uniform prior

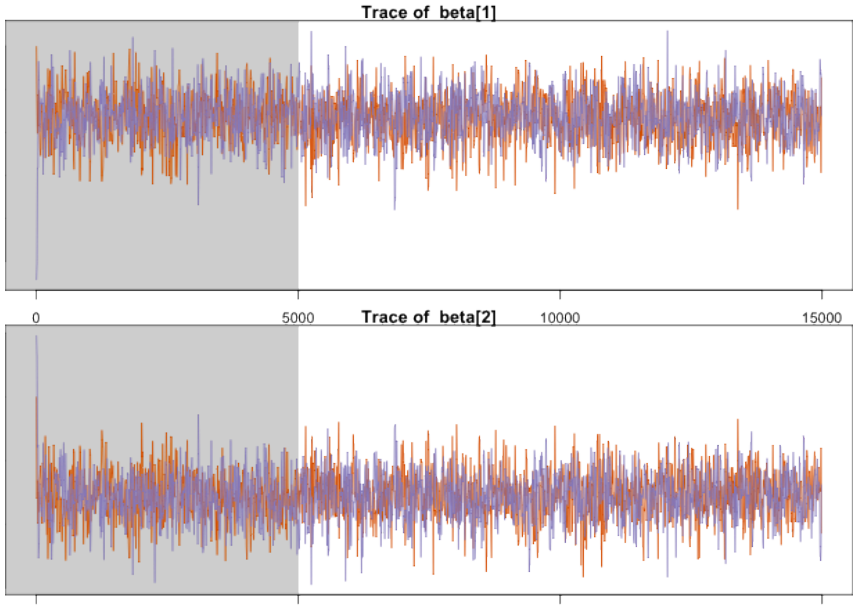
```
library(rstan)
stan_code <- '
 data {
  int<lower=0> N;
  real  time[N];
  real  event[N];
  real  placement[N];
}
parameters {
  real beta[2];
}
model {
  real eta[N];
  #beta ~ normal(0,  1000);
  for(i in 1:N) {
    eta[i]<- beta[1]+ beta[2]*placement[i];
    lp__ <- lp__ + event[i]*(eta[i])-time[i]*exp(eta[i]);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

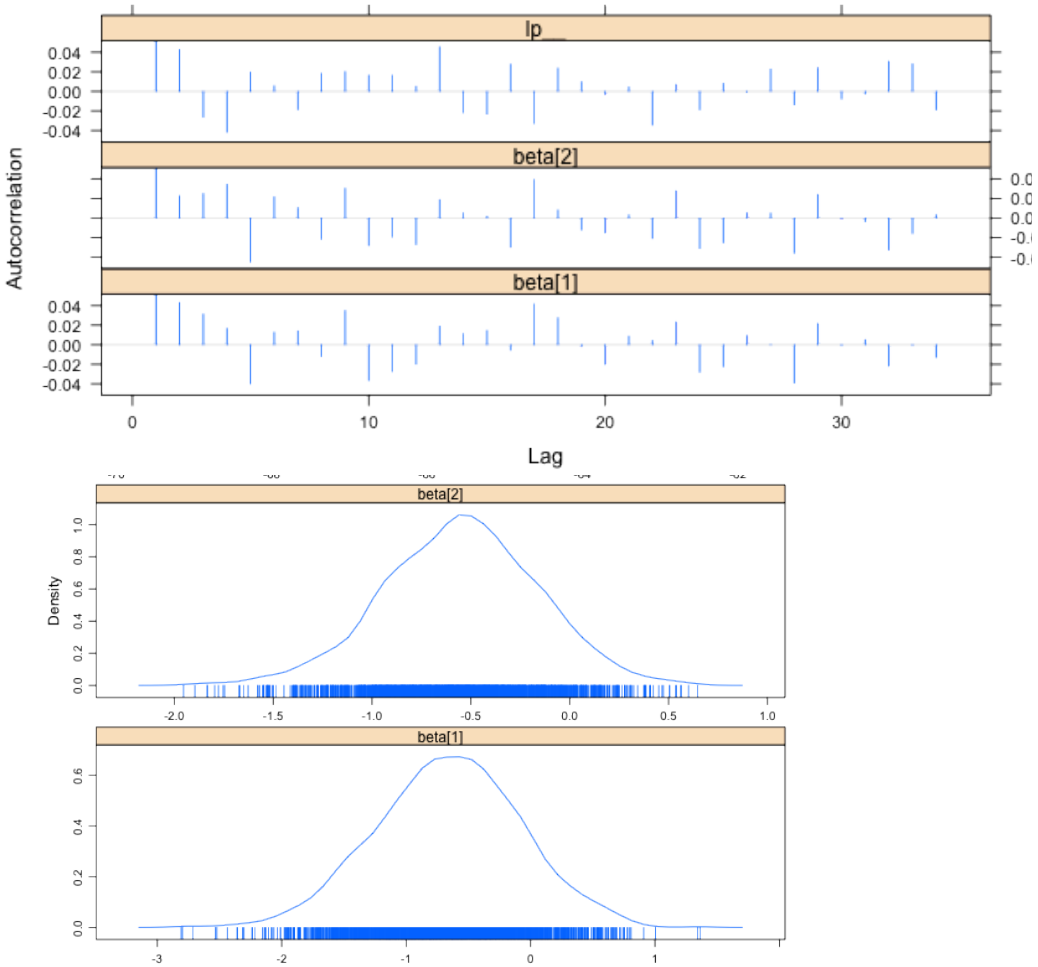
	mean	se_mean	sd	25%	50%	75%	n_eff	Rhat
beta[1]	-0.66	0.01	0.59	-1.05	-0.65	-0.27	1681	1
beta[2]	-0.55	0.01	0.39	-0.81	-0.55	-0.29	1714	1

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

	lower	upper
beta[1]	-1.758074	0.5402209
beta[2]	**-1.387087**	**0.1610000**

(Other) model check





INLA model

- 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:
               mean     sd 0.025quant 0.5quant 0.975quant   kld
(Intercept) -0.6243 0.5979    -1.8395   -0.610     0.5097 4e-04
placement   -0.5335 0.3969    -1.3253   -0.529     0.2323 0e+00
```

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

```
                 low      high
level:0.95 -1.317485 0.2390768
```

(similar to Stan and SAS results with Normal prior)

Conclusion

- 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 departement 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 !