RStan: Efficient MCMC in R

Jack Jewson

Warwick R user group: 2 March 2017

Problem

You're doing a statistical analysis:

- Want to impose some prior information.
- Ultimately you care about prediction.
- So you decide to be Bayesian.
- However… you don't have conjugate priors.
- You could spend ages coding up a Metropolis-Hastings, optimising acceptance rates and proposals.
- Or you could use *RStan*!

HMC

- HMC is a method to produce proposals for a MH algorithm that are accepted with high probability.
- Rather than have a proposal distribution we appeal to Hamiltonian dynamics.
- Consider the target distribution as an inverted ice rink:
  - Give the particle some momentum.
  - It slides around the ice rink spending most time where the density is high.
  - Taking snap shots of this trajectory gives a proposal sample for the posterior.
  - We then correct using Metropolis-Hastings.

Original paper Alder and Wainwright (1959) or see Neal and others (2011) for an easier read

NUTS

- HMC, like RWMH, requires some tuning, the number and size of the leapfrog steps.
- The "No-U-Turn Sampler" or NUTs (Hoffman and Gelman (2014)), optimises these adaptively.
- Too small number and size of steps leads to RW type behaviour while too big the trajectory starts to come back on itself.
- NUTS builds up a set of likely candidate points and stops immediately when a trajectory starts to come back on itself.

Advantages of *Stan*

- Can produce high dimensional proposals that are accepted with high probability without having to spend time tuning.
- Has inbuilt diagnostics to analyse the MCMC output.
- Built in *c++* so runs quickly but outputs to *R*.

Example

- What to build a Bayesian linear regression model using LASSO shrinkage.
- Build a *Stan* model:
    - Data: n, p, Y, X, prior parameters, hyper-parameters
    - Parameters: β, σ^2
    - Model: Gaussian likelihood, Laplacian and Gamma priors.
    - Output: Posterior samples, posterior predictive samples.
- see example *bayes_LASSO.stan* model file.

Data

```
data {

    int<lower=0> n;
    int<lower=0> p;
    matrix[n,p+1] X;
    vector[n] y;
    real<lower=0> a;
    real<lower=0> b;
    real<lower=0> w;
}
```

Parameters

```
parameters {

    vector[p+1] beta;
    real<lower=0> sigma;
}
```

Transformed parameters (optional)

```
transformed parameters {

    vector[n] linpred;
    linpred = X*beta;
}
```

Model

```
model {

    beta ~ double_exponential(0,w);
    sigma ~ gamma(a,b);

    y~ normal(linpred,sigma);
}
```

or without vectorisation,

```
    for(i in 1:n){
       y[i]~normal(X[i,]*beta,sigma);
    }
```

Generated quantities (optional)

```
generated quantities {

    vector[n] y_predict;
    for(i in 1:n){
      y_predict[i] = normal_rng(linpred[i],sigma);
    }
}
```

- evaluates this code once for every element of the posterior sample.

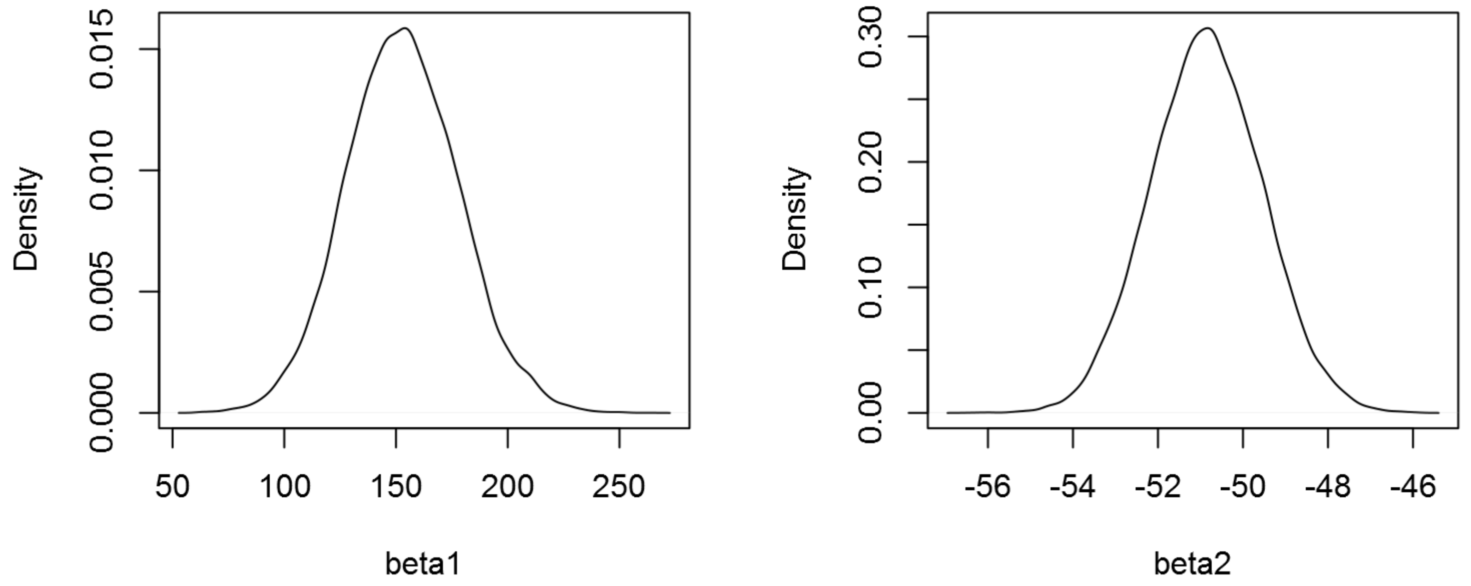
Running in R

```r
library(rstan)
setwd("C:/Users/jack/Documents/OxWaSP_Yr2/Presentations")
data<-list(n=102,p=3,X=cbind(Prestige$education, Prestige$women,
      Prestige$prestige,rep(1,102)),y=Prestige$income,a=10,b=10,w=100)
chain1 <- stan(file="bayes_LASSO.stan",data=data,iter=50000, chains=1,
              cores=1)
params<-extract(chain1)
```

- Runs 25000 warmups and 25000 samples in 3.5 seconds.
- Compiles *c*++ code the first time so that may take longer.

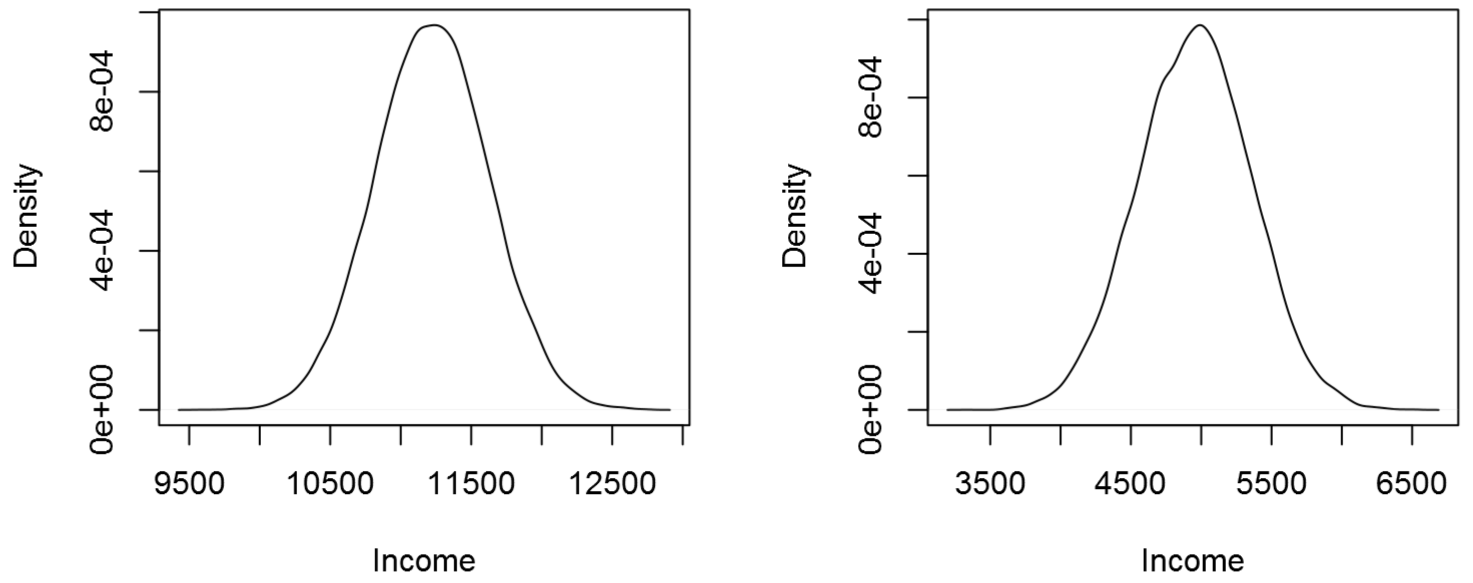
Plotting posterior distributions

```r
par(mfrow=c(1,2))
plot(density(params$beta[,1]),xlab="beta1",ylab="Density",main="")
plot(density(params$beta[,2]),xlab="beta2",ylab="Density",main="")
```



and predictive distributions

```
par(mfrow=c(1,2))
plot(density(params$y_predict[,1]),xlab="Income",ylab="Density",main="")
plot(density(params$y_predict[,100]),xlab="Income",ylab="Density",main="")
```



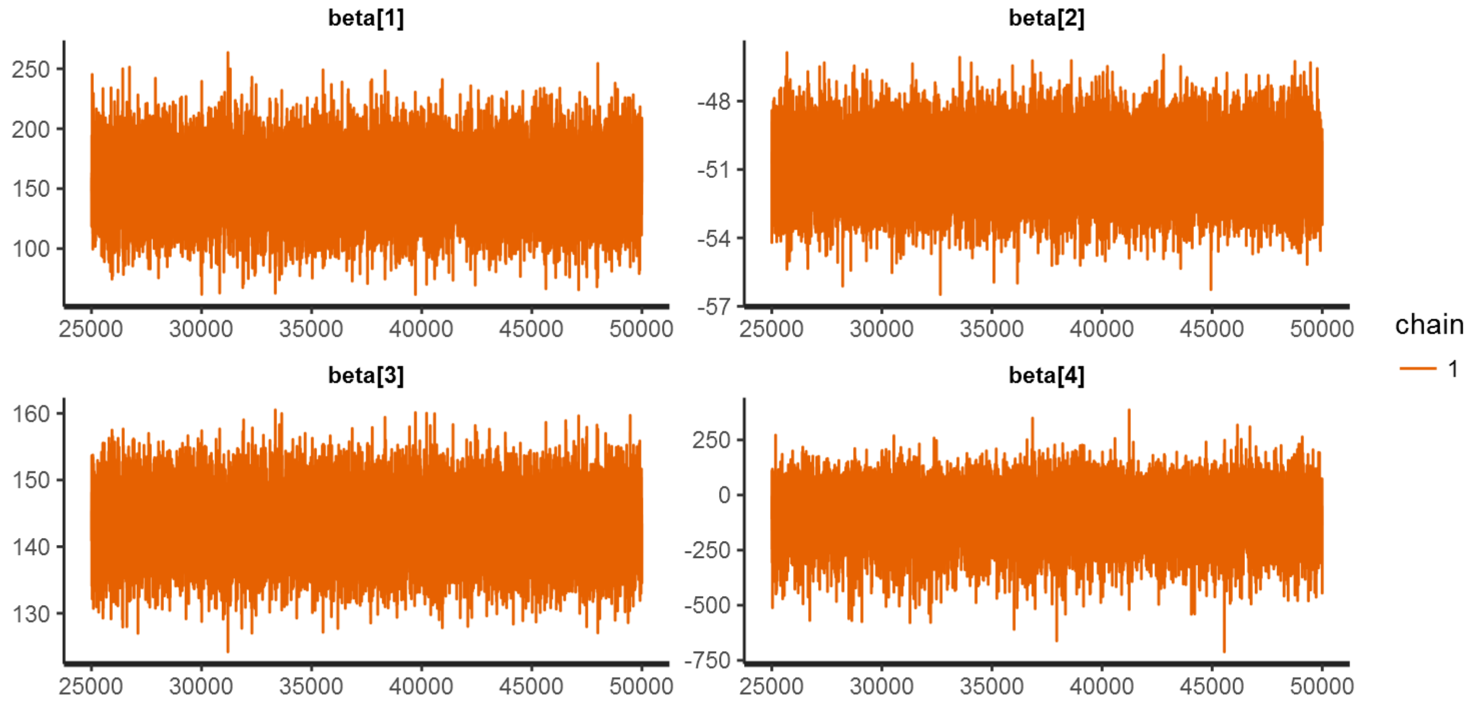
Chain Diagnostics

```r
sampler_params <- get_sampler_params(chain1, inc_warmup = FALSE)
sampler_params[[1]][1:5,]
```

```
##      accept_stat__ stepsize__ treedepth__ n_leapfrog__ divergent__
## [1,]     0.9803107  0.1240111           4           15           0
## [2,]     0.9025821  0.1240111           4           31           0
## [3,]     0.9662435  0.1240111           5           31           0
## [4,]     0.8553821  0.1240111           5           31           0
## [5,]     0.8977977  0.1240111           5           31           0
##      energy__
## [1,] 6588.685
## [2,] 6587.883
## [3,] 6589.927
## [4,] 6591.606
## [5,] 6589.096
```

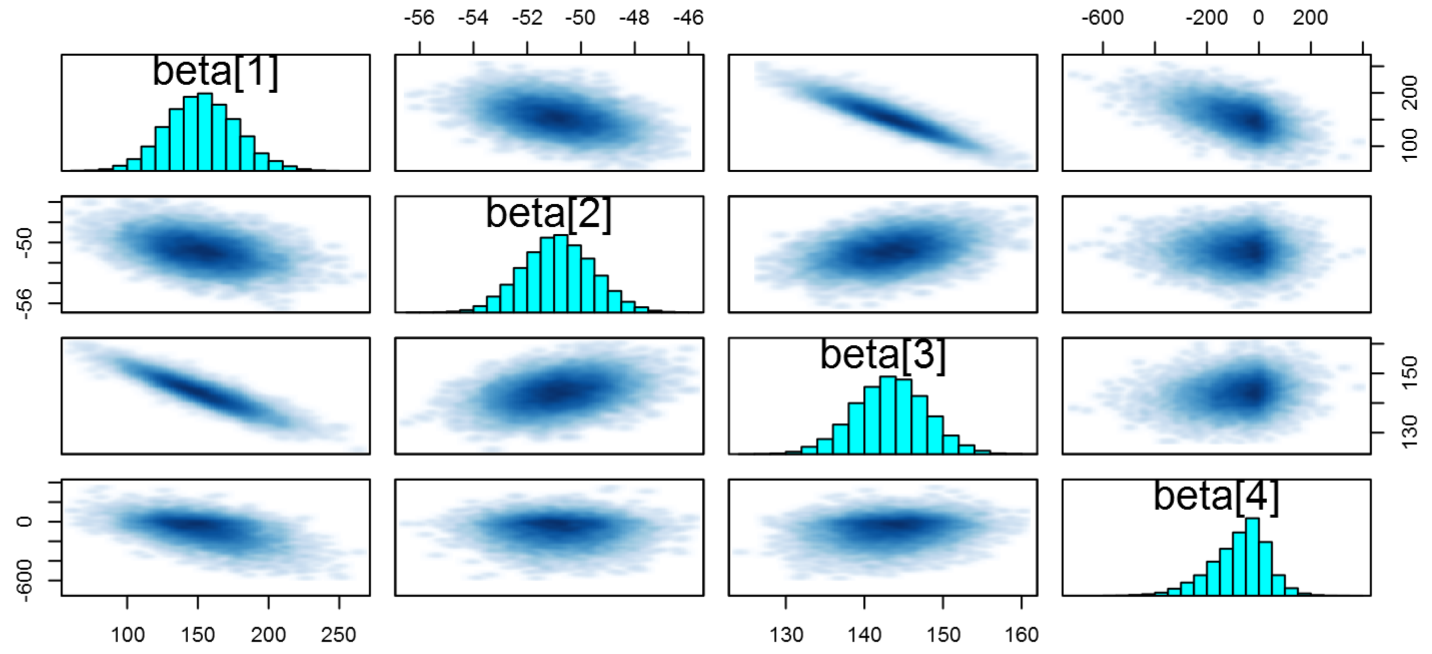
Chain Diagnostics

```
traceplot(chain1,pars="beta" )
```



Chain Diagnostics

```
pairs(chain1,pars="beta")
```



more Chain Diagnostics

Stan can also extract various other diagnostics from the chain

- Credibility intervals.
- effective sample size and the Markov chain squared error.
- Comparison plots between the values of the chain and various chain properties, log-likelihood, acceptance rate and step size.

for more information see *Rstan Diagnostic Plots*

When *Stan* goes wrong

- "Divergent transitions after warmup"
- Means *Stan* is taking steps that are too big.
- Can fix by manually increasing the desired average acceptance probably, *adapt_delta*, above it's default of 0.8

```
chain1 <- stan(file="bayes_LASSO.stan",data=data,iter=50000, chains=1,
              cores=1,control = list(adapt_delta = 0.99,
                                     max_treedepth = 15))
```

- This will slow your chain down but may result in a better sample
- see http://mc-stan.org/misc/warnings.html for more info.

Self-made function

- *Stan* also has compatibility with self-made function.
- Useful if your prior or likelihood is not standard.

```
model {

    beta ~ double_exponential(0,w);
    sigma ~ gamma(a,b);

    for(i in 1:n){
      increment_log_prob(-0.5*fabs(1-(exp(normal_log(y[i],linpred[i],
            sigma))/y_dense[i])));
    }
}
```

Conclusion

- Don't waste your time coding and tuning RWMH!
- *Stan* will run faster, is automatically tuned and should produce a superior samples.
- Inbuilt functions make analysing your chain easy.

For more information

- *Stan* manual: http://mc-stan.org/documentation/ (only 601 pages long).
- Google groups: http://mc-stan.org/community/.
- R-package documentation: https://cran.r-project.org/web/packages/rstan/index.html.

References

Alder, Berni J, and T E Wainwright. 1959. “Studies in Molecular Dynamics. I. General Method.” *The Journal of Chemical Physics* 31 (2). AIP: 459–66.

Hoffman, Matthew D, and Andrew Gelman. 2014. “The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo.” *Journal of Machine Learning Research* 15 (1): 1593–1623.

Neal, Radford M, and others. 2011. “MCMC Using Hamiltonian Dynamics.” *Handbook of Markov Chain Monte Carlo* 2: 113–62.