Fast Bayesian modeling in Stan using rstan

mc-stan.org

Hamiltonian Monte Carlo

Speed
(rotation-invariance + convergence + mixing)

Flexibility of priors

Stability to initial values

See Radford Neal's chapter in the
Handbook of MCMC

Hamiltonian Monte Carlo

Tuning is tricky

One solution is the No U-Turn Sampler (NUTS)

Stan is a C++ library for NUTS (and variational inference, and L-BFGS)

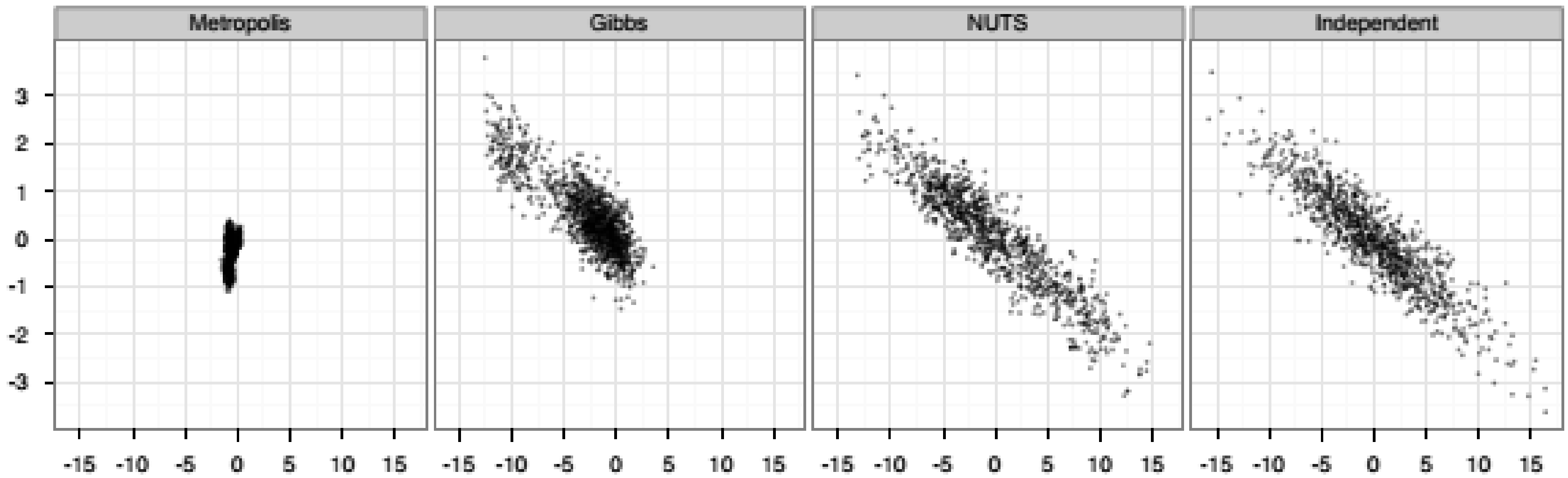


Figure 7: *Samples generated by random-walk Metropolis, Gibbs sampling, and NUTS. The plots compare 1,000 independent draws from a highly correlated 250-dimensional distribution (right) with 1,000,000 samples (thinned to 1,000 samples for display) generated by random-walk Metropolis (left), 1,000,000 samples (thinned to 1,000 samples for display) generated by Gibbs sampling (second from left), and 1,000 samples generated by NUTS (second from right). Only the first two dimensions are shown here.*

http://www.stat.columbia.edu/~gelman/research/published/nuts.pdf

rstan

```
stan( file='model.stan',
      data=list.of.data,
      chains=4,
      iter=10000,
      warmup=2000,
      init=list.of.initial.values,
      seed=1234 )
```

Some simulations

Collaboration with Furr, Carpenter, Rabe-Hesketh, Gelman
arxiv.org/pdf/1601.03443v1.pdf
rstan v StataStan v JAGS v Stata

Today: rstan v rjags
robertgrantstats.co.uk/rstan_v_jags.R

Rasch model (item-response)

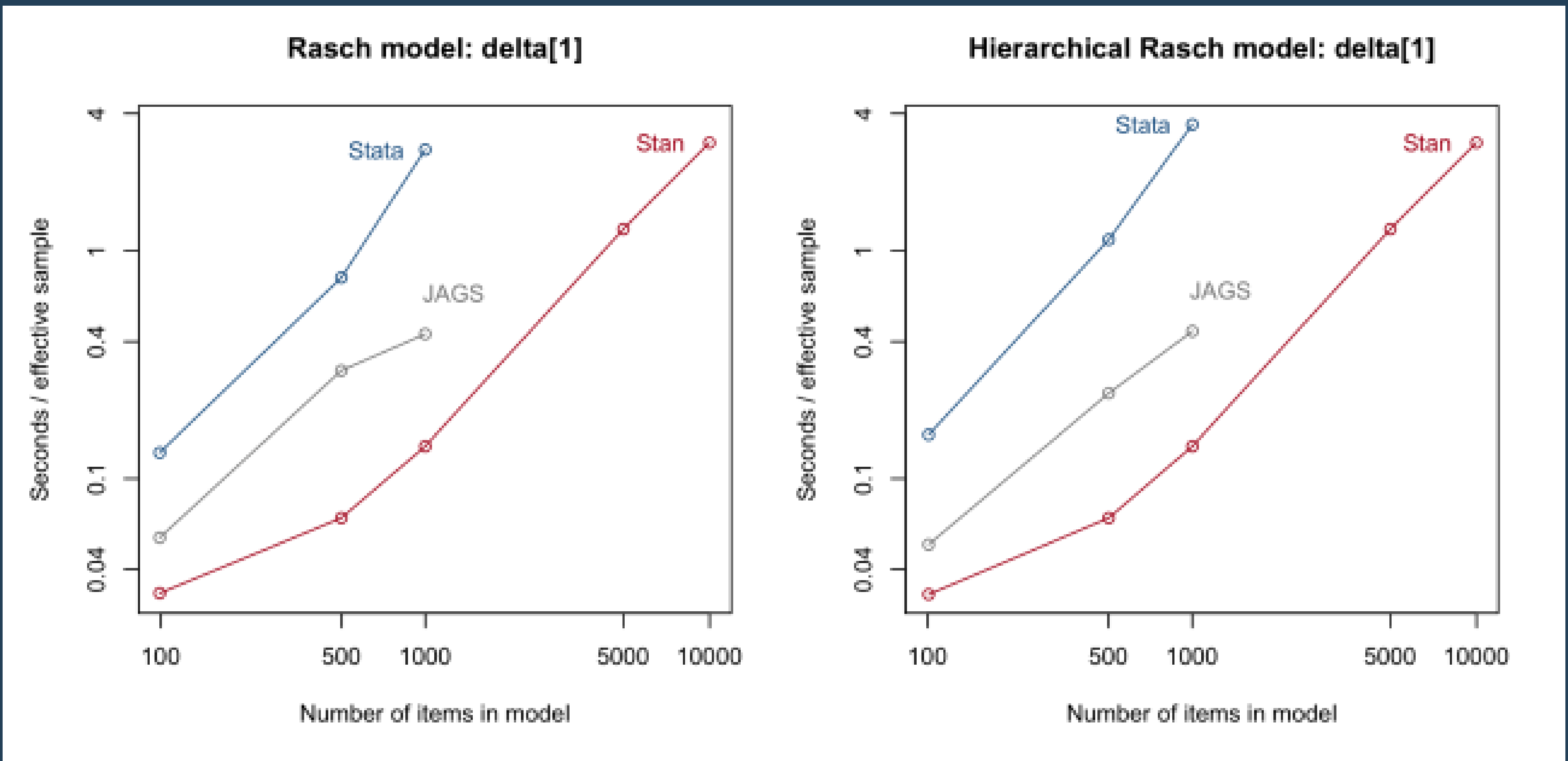
$$\Pr(y_{ip} = 1 | \theta_p, \delta_i) = \text{logit}^{-1}(\theta_p + \delta_i)$$

$$\theta_p \sim \text{N}(0, \sigma^2)$$

Hierarchical Rasch model (includes hyperpriors)

$$\Pr(y_{ip} = 1 | \mu, \theta_p, \delta_i) = \text{logit}^{-1}(\mu + \theta_p + \delta_i)$$
$$\theta_p \sim \text{N}(0, \sigma^2)$$
$$\delta_i \sim \text{N}(0, \tau^2)$$

StataStan vs Stata vs rjags



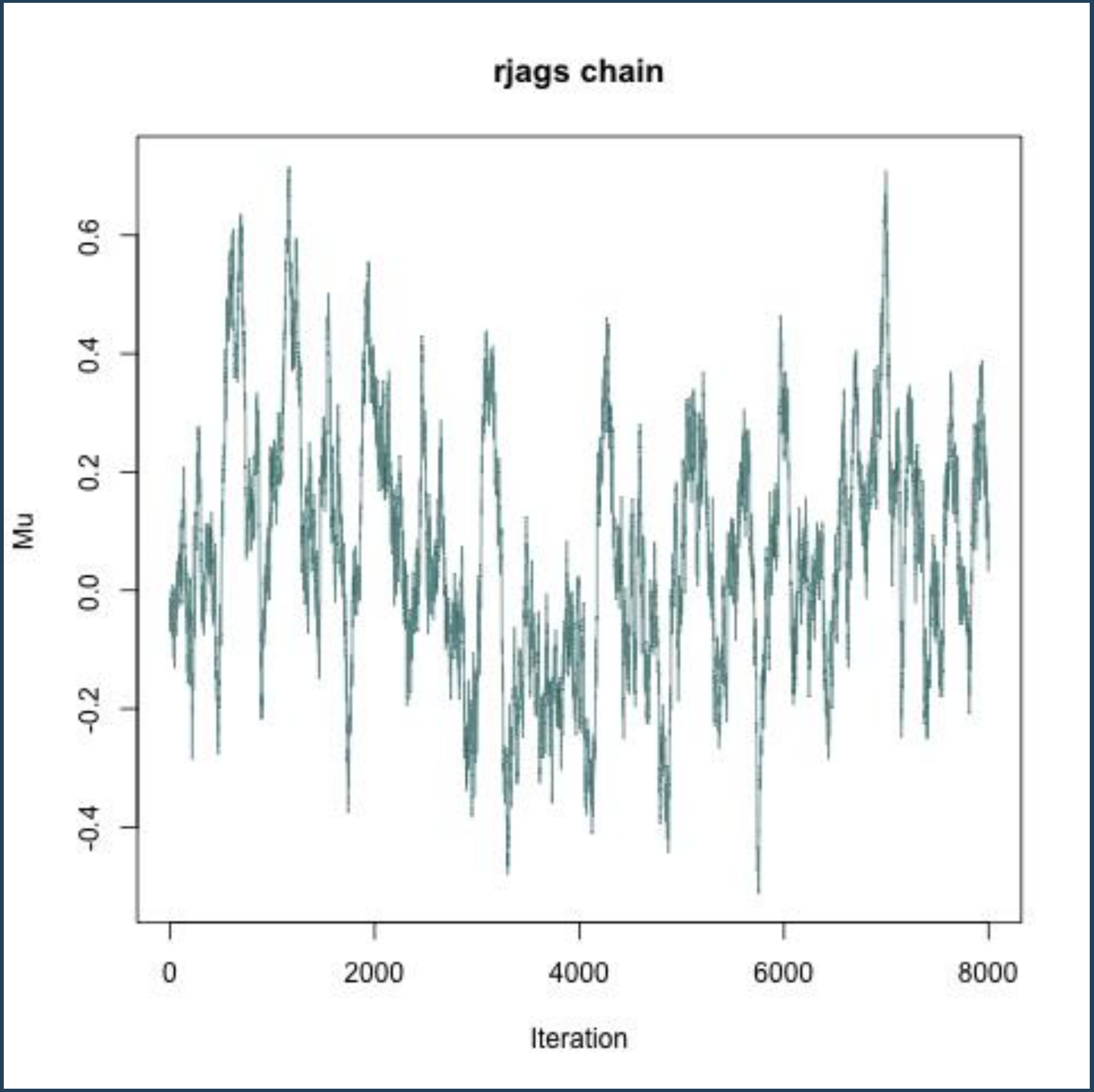
rstan vs rjags

Seconds:	Rasch	H-Rasch
rstan	180	210
rjags	558	1270

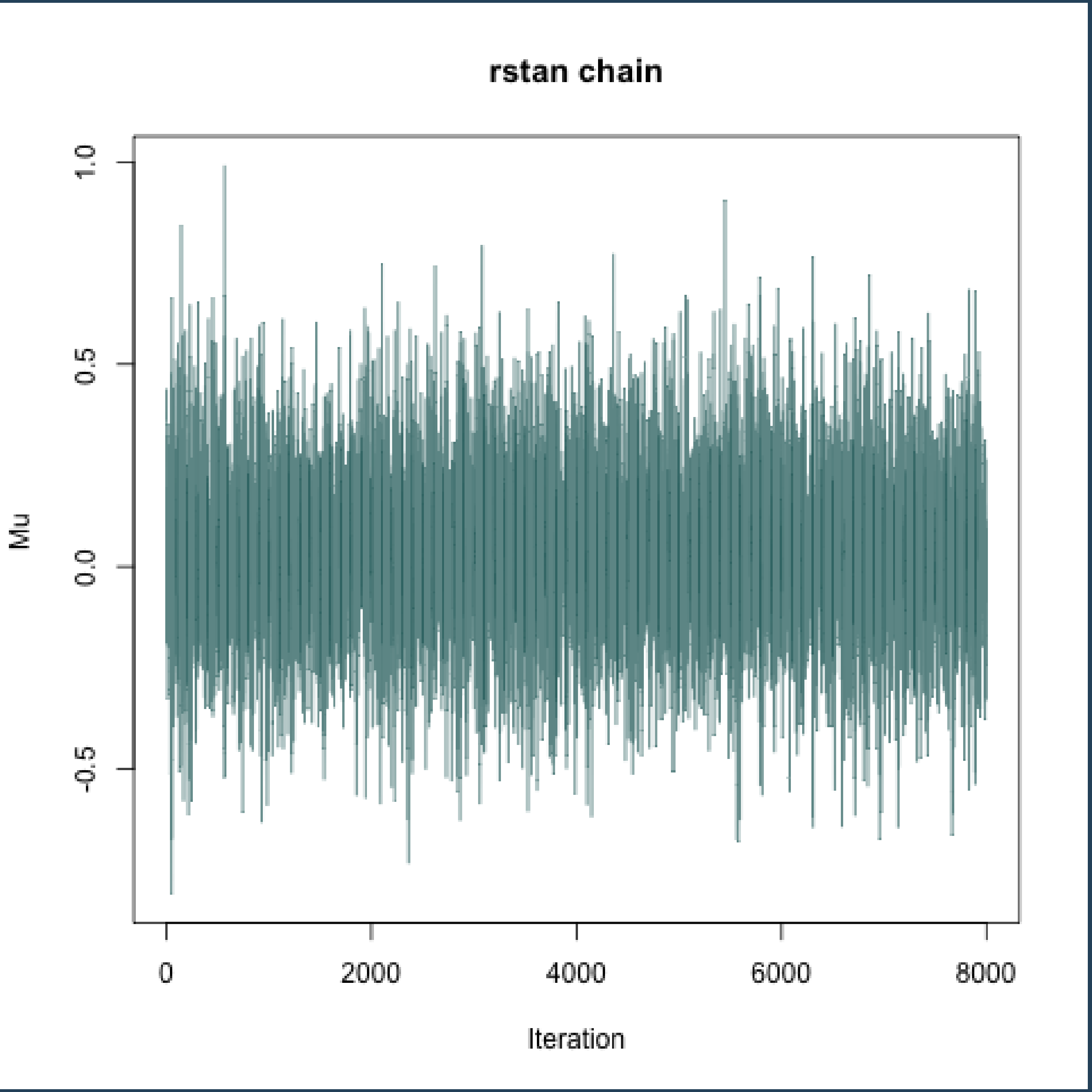
ESS (sigma):	Rasch	H-Rasch
rstan	22965	21572
rjags	7835	8098

ESS (theta1):	Rasch	H-Rasch
rstan	32000	32000
rjags	19119	19637

rstan vs rjags



rstan vs rjags



Hierarchical Rasch model (includes hyperpriors)

```
data {
  int<lower=1> N;
  int<lower=1> I;
  int<lower=1> P;
  int<lower=1, upper=I> ii[N];
  int<lower=1, upper=P> pp[N];
  int<lower=0, upper=1> y[N];
}
parameters {
  real<lower=0, upper=5> sigma;
  real<lower=0, upper=5> tau;
  real mu;
  vector[I] delta;
  vector[P] theta;
}
model {
  vector[N] eta;
  theta ~ normal(0, sigma);
  delta ~ normal(0, tau);
  mu ~ normal(0, 5);
  for(n in 1:N) eta[n] <- mu + theta[pp[n]] + delta[ii[n]];
  y ~ bernoulli_logit(eta);
}
```

```
data {
        int<lower=1> N;
        int<lower=1> I;
        int<lower=1> P;
        int<lower=1, upper=I> ii[N];
        int<lower=1, upper=P> pp[N];
        int<lower=0, upper=1> y[N];
}
parameters {
        real<lower=0, upper=5> sigma;
        real<lower=0, upper=5> tau;
        real mu;
        vector[I] delta;
        vector[P] theta;
}
model {
        vector[N] eta;
        theta ~ normal(0, sigma);
        delta ~ normal(0, tau);
        mu ~ normal(0, 5);
        for(n in 1:N) {
                eta[n] <- mu + theta[pp[n]] + delta[ii[n]];
        }
        y ~ bernoulli_logit(eta);
}
```

```
data {
        int<lower=1> N;
        int<lower=1> I;
        int<lower=1> P;
        int<lower=1, upper=I> ii[N];
        int<lower=1, upper=P> pp[N];
        int<lower=0, upper=1> y[N];
}
parameters {
        real<lower=0, upper=5> sigma;
        real<lower=0, upper=5> tau;
        real mu;
        vector[I] delta;
        vector[P] theta;
}
model {
        vector[N] eta;
        theta ~ normal(0, sigma);
        delta ~ normal(0, tau);
        mu ~ normal(0, 5);
        for(n in 1:N) {
                eta[n] <- mu + theta[pp[n]] + delta[ii[n]];
        }
        y ~ bernoulli_logit(eta);
}
```

```
data {
        int<lower=1> N;
        int<lower=1> I;
        int<lower=1> P;
        int<lower=1, upper=I> ii[N];
        int<lower=1, upper=P> pp[N];
        int<lower=0, upper=1> y[N];
}
parameters {
        real<lower=0, upper=5> sigma;
        real<lower=0, upper=5> tau;
        real mu;
        vector[I] delta;
        vector[P] theta;
}
model {
        vector[N] eta;
        theta ~ normal(0, sigma);
        delta ~ normal(0, tau);
        mu ~ normal(0, 5);
        for(n in 1:N) {
                eta[n] <- mu + theta[pp[n]] + delta[ii[n]];
        }
        y ~ bernoulli_logit(eta);
}
```

And...

Missing data CAN be included (coarse data too) as latent variables mapped to observed values

Discrete parameters CAN be approximated with steep(ish) logistic curves

Getting started

mc-stan.org

stan-users Google Group