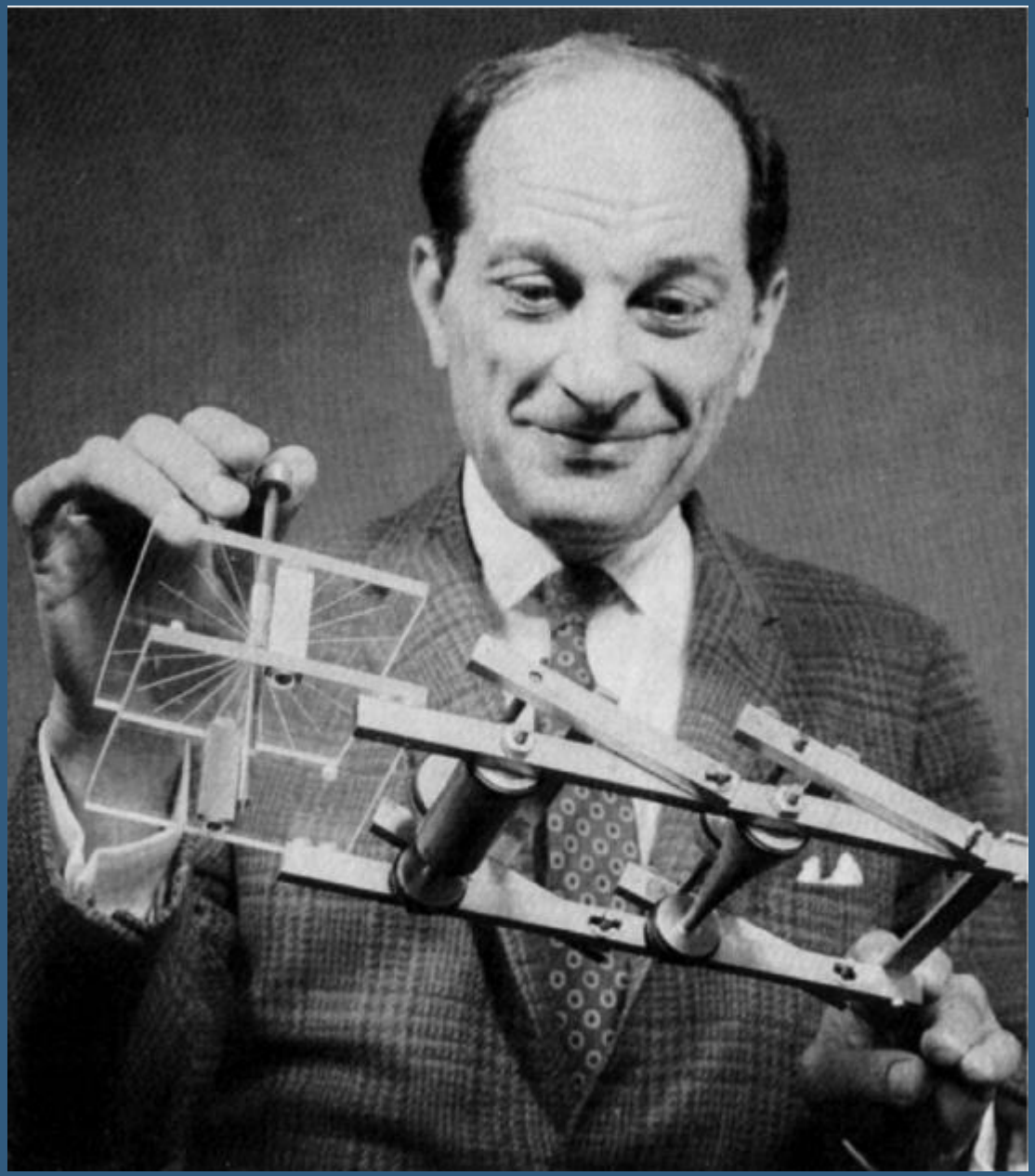
Fast Bayesian modeling in Stan using StataStan

mc-stan.org

Robert Grant

Kingston University + St George's,
University of London
www.robertgrantstats.co.uk

Collaborative work including:
Bob Carpenter
Daniel Furr
Andrew Gelman
Daniel Lee
Sophia Rabe-Hesketh



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)

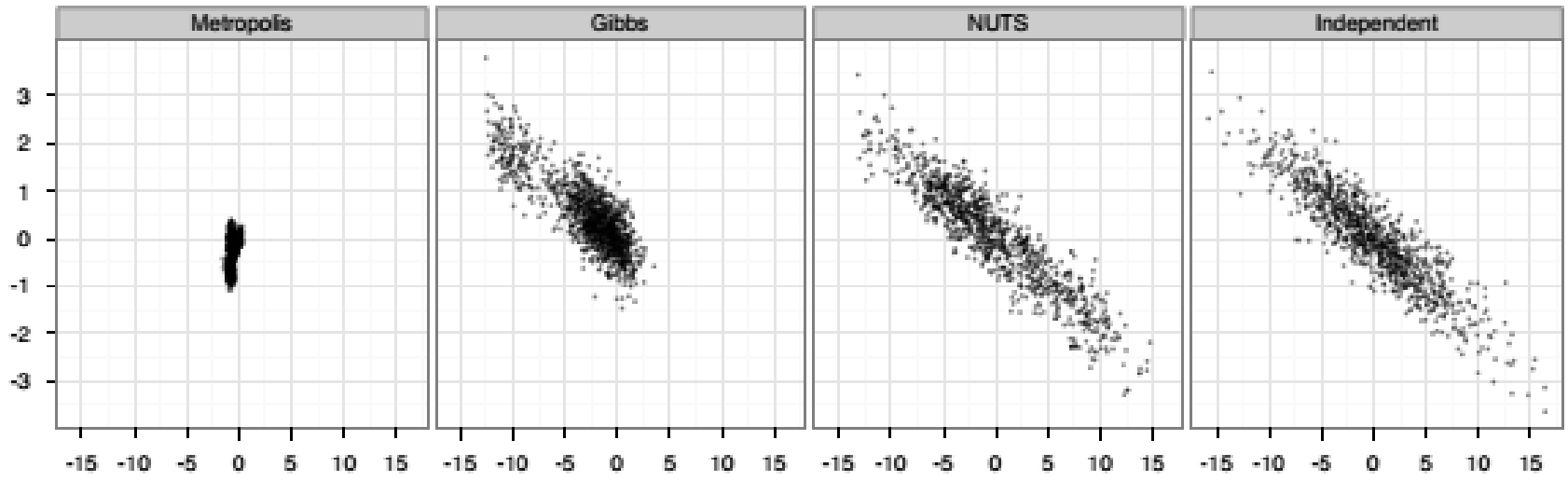


Figure 7: *Samples generated by random-walk Metropolis, Gibbs sampling, and NUTS. The plots compare 1,000 independent draws from a highly correlated 250-dimensional distribution (right) with 1,000,000 samples (thinned to 1,000 samples for display) generated by random-walk Metropolis (left), 1,000,000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

Some simulations

Daniel Furr is first author;
paper forthcoming

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）

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

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

$$\delta_i \sim \text{N}(0, \tau^2)$$

Hierarchical Rasch model (includes hyperpriors)

```
bayesmh y mu i.person i.item, noconstant likelihood(logit) ///
  mcmcsize(1000) burnin(100000) thinning(100)              ///
  prior({y:i.person},   normal(0, {sigma}^2))              ///
  prior({y:i.item},     normal(0, {tau}^2))                ///
  prior({y:mu},         normal(0, 25))                     ///
  prior({sigma},        uniform(0, 5))                     ///
  prior({tau},          uniform(0, 5))
```

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);
}
```

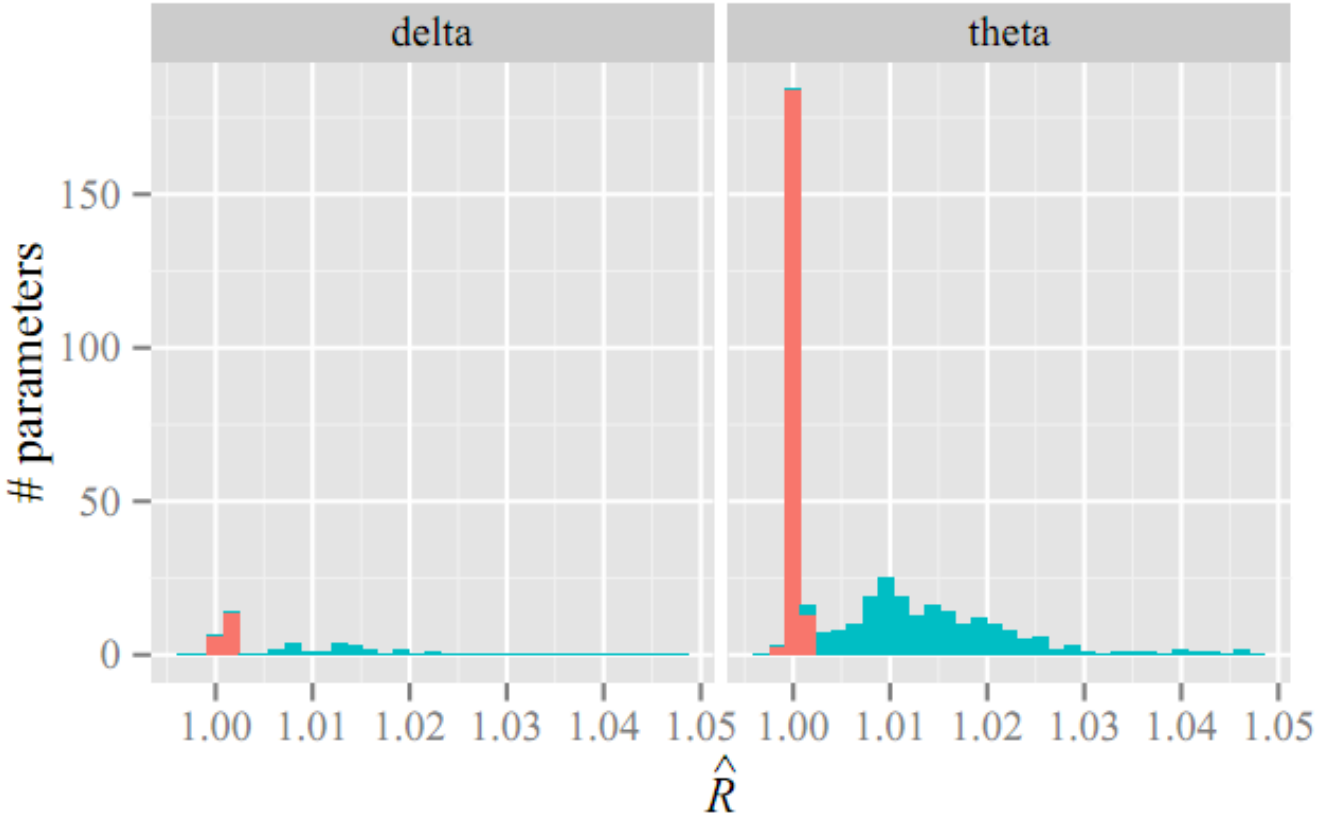


Figure 1: Histograms of convergence statistics for the Rasch model (20 δ's and 200 θ's).

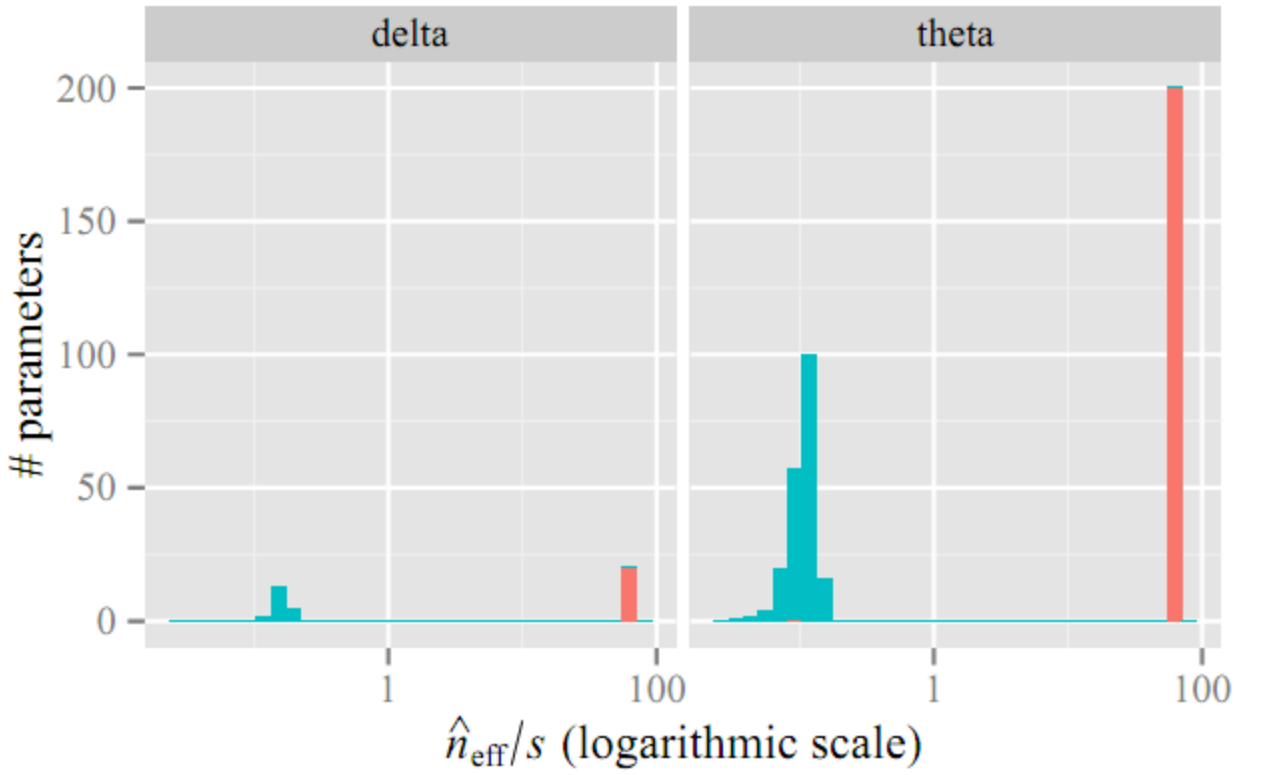


Figure 2: Histograms of efficiency statistics for the Rasch model (20 δ's and 200 θ's).

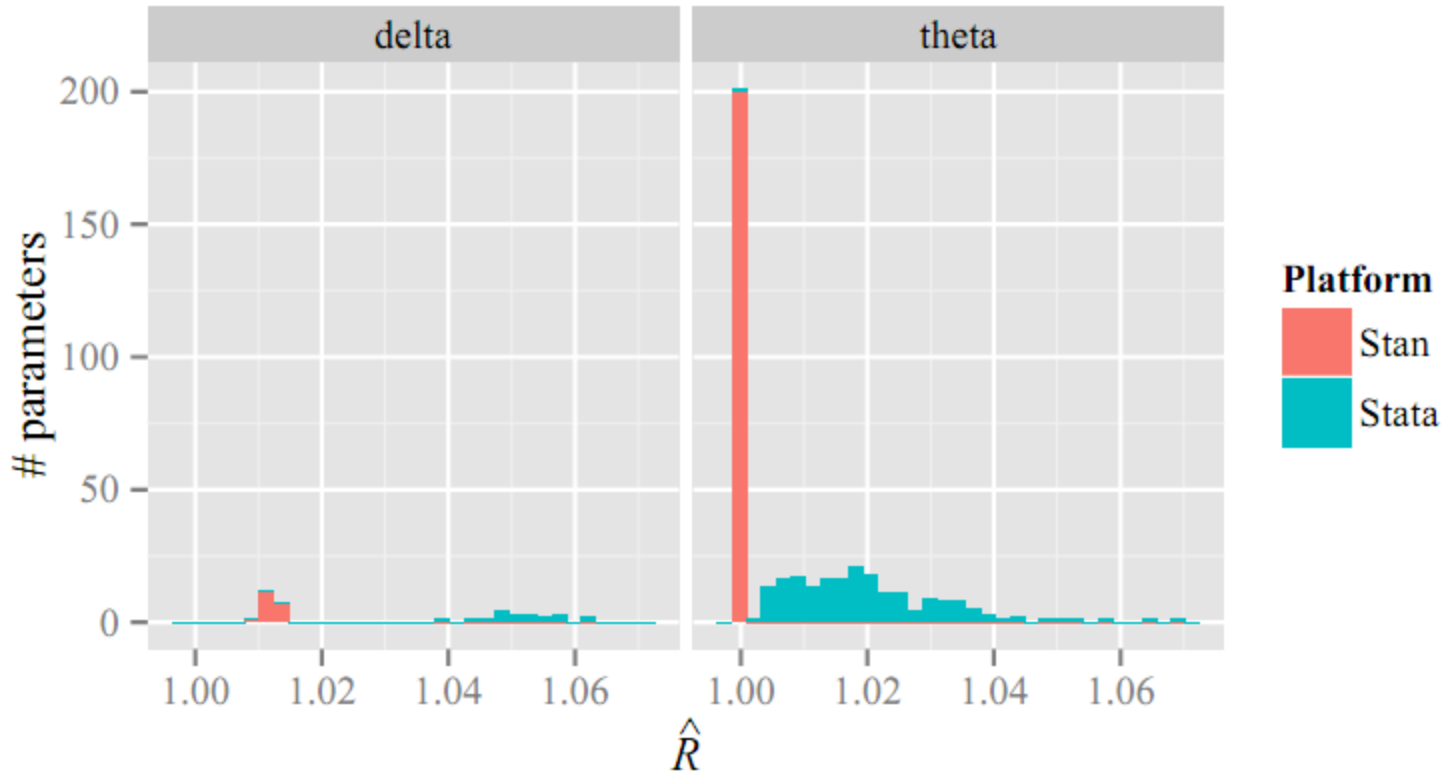


Figure 3: Histograms of convergence statistics for the hierarchical Rasch model (20 δ's and 200 θ's)

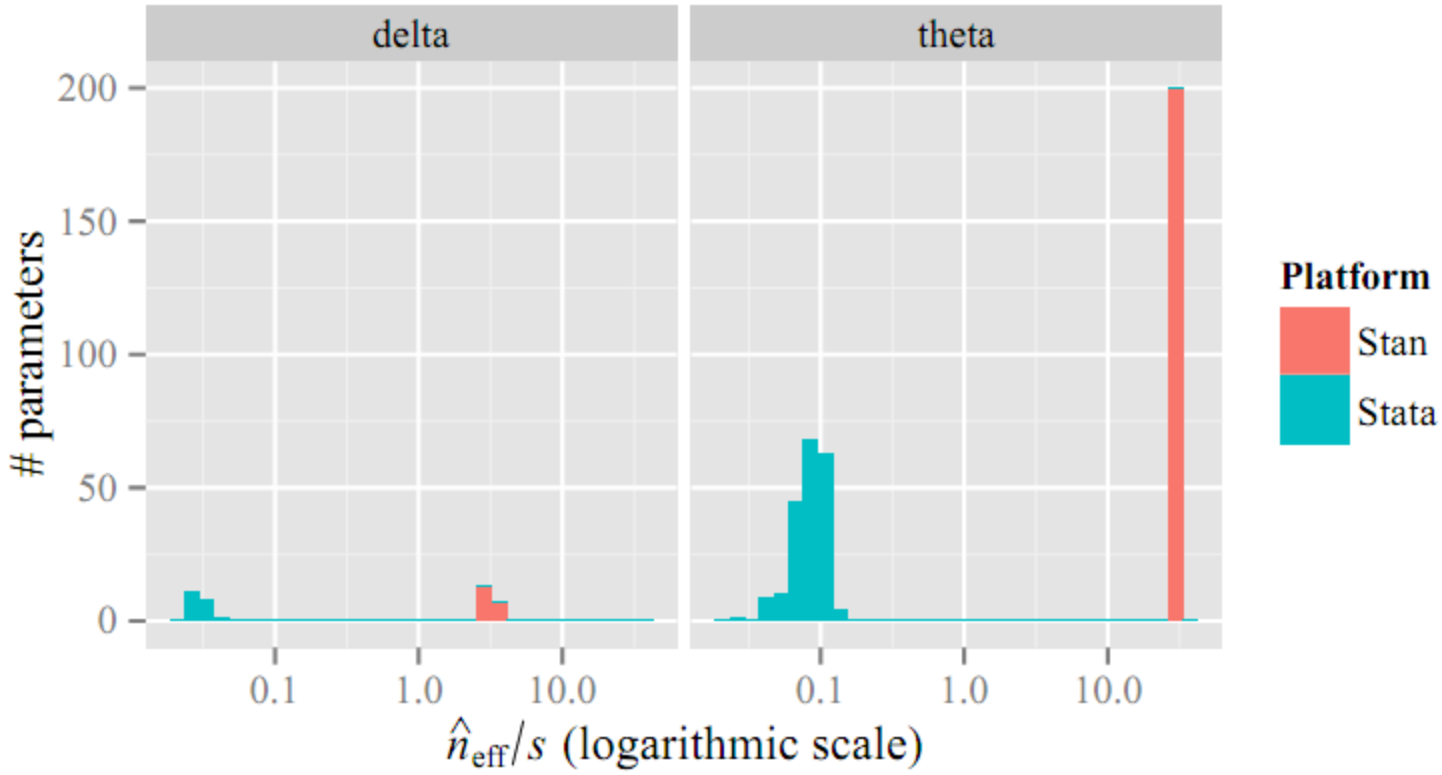
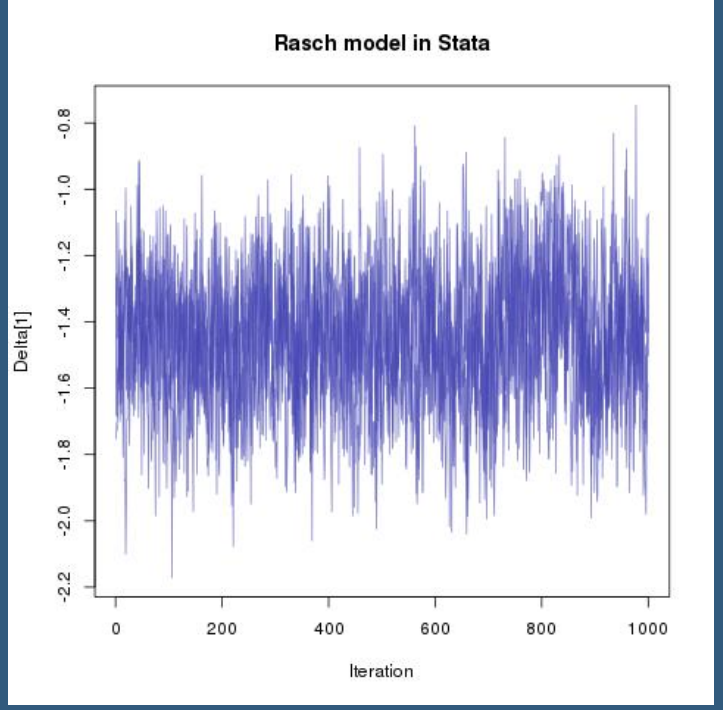
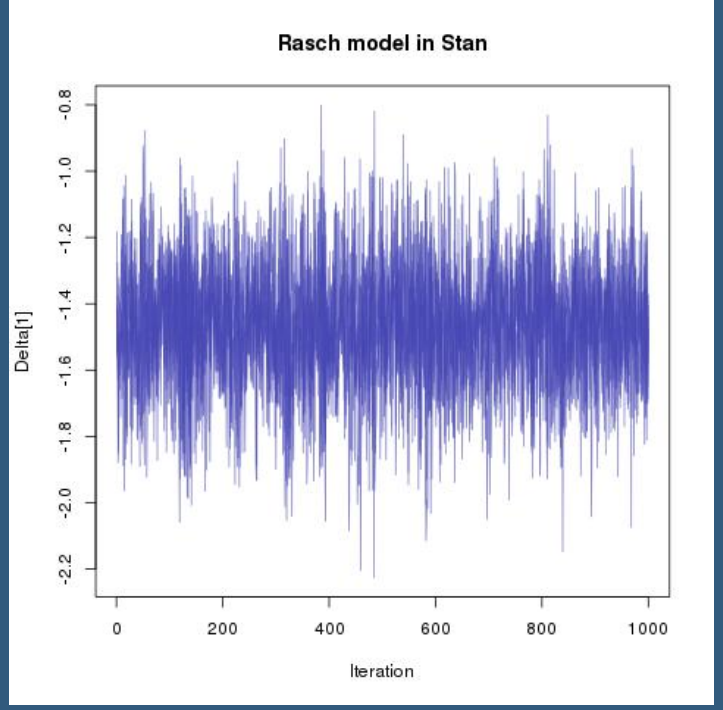
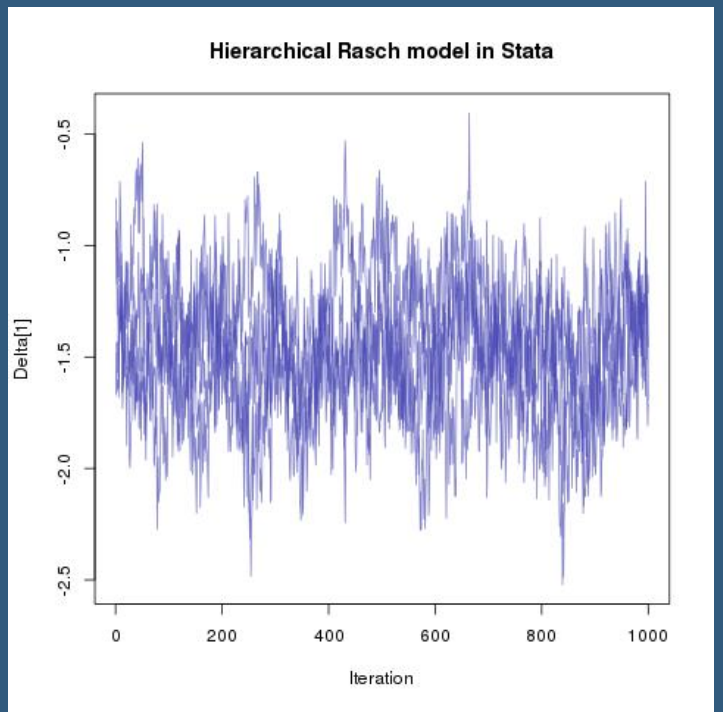
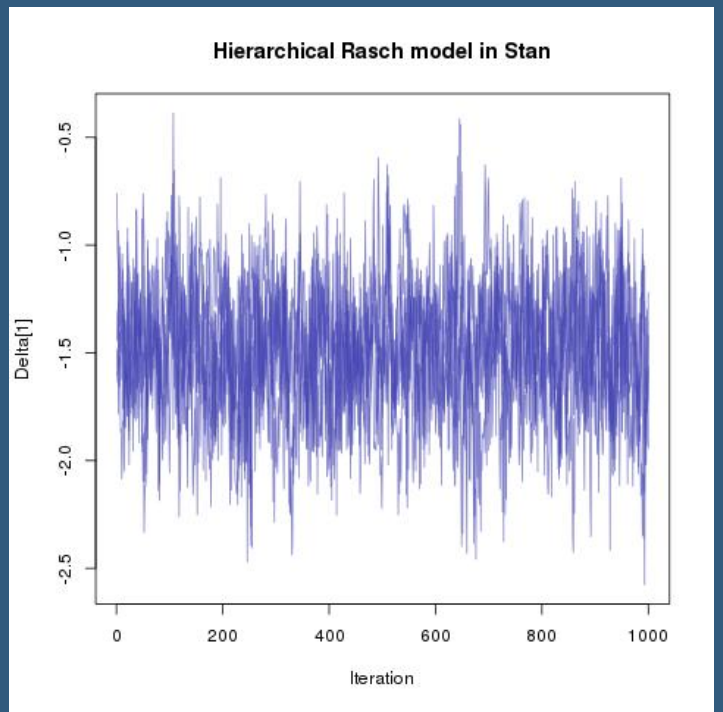


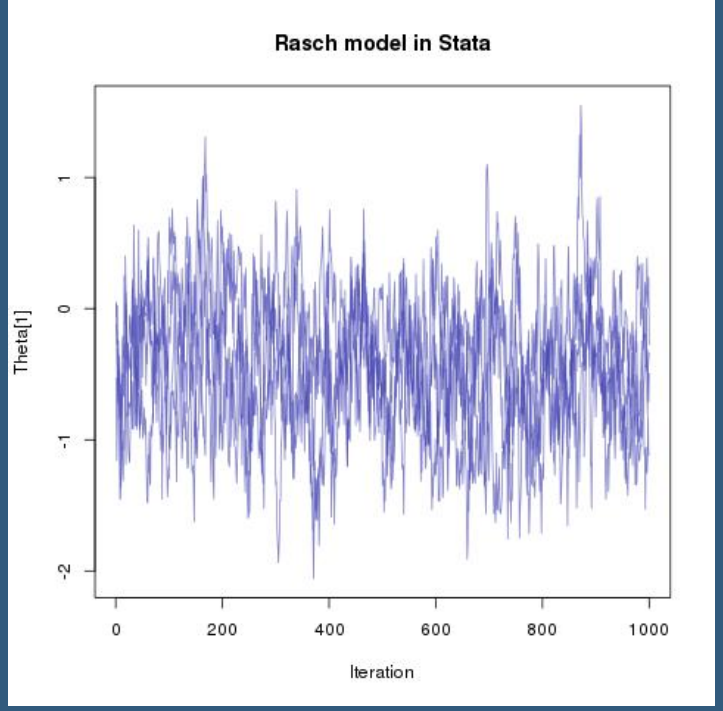
Figure 4: Histograms of efficiency statistics for the hierarchical Rasch model (20 δ's and 200 θ's).

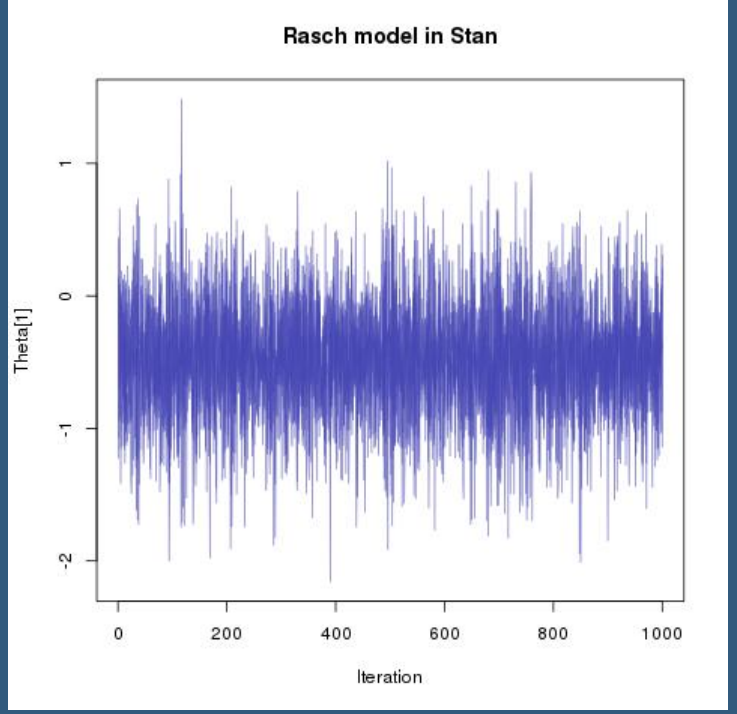

Rasch model in Stata
Delta[1]
-0.8
-1.0
-1.2
-1.4
-1.6
-1.8
-2.0
-2.2
0
200
400
600
800
1000
Iteration

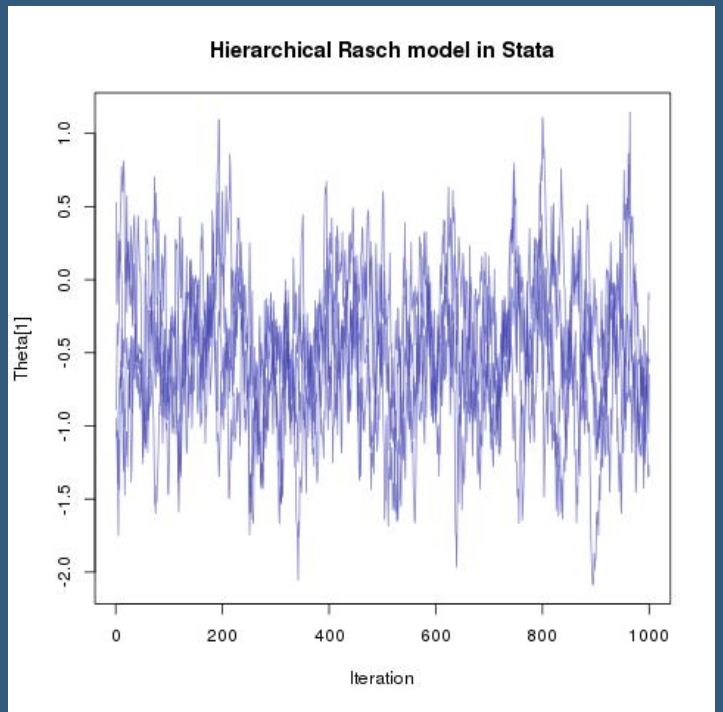

Rasch model in Stan
Delta[1]
-0.8
-1.0
-1.2
-1.4
-1.6
-1.8
-2.0
-2.2
0
200
400
600
800
1000
Iteration

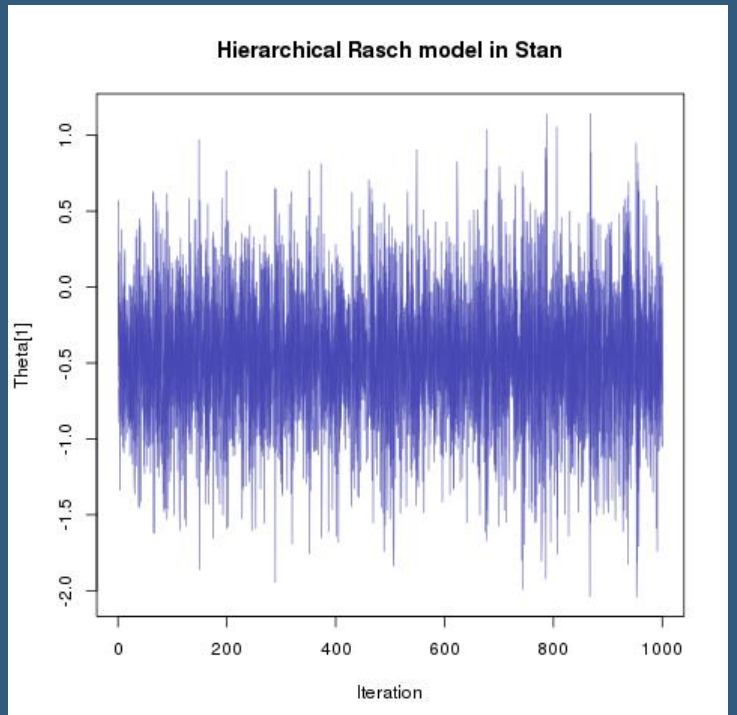

Hierarchical Rasch model in Stata
Delta[1]
-0.5
-1.0
-1.5
-2.0
-2.5
0
200
400
600
800
1000
Iteration

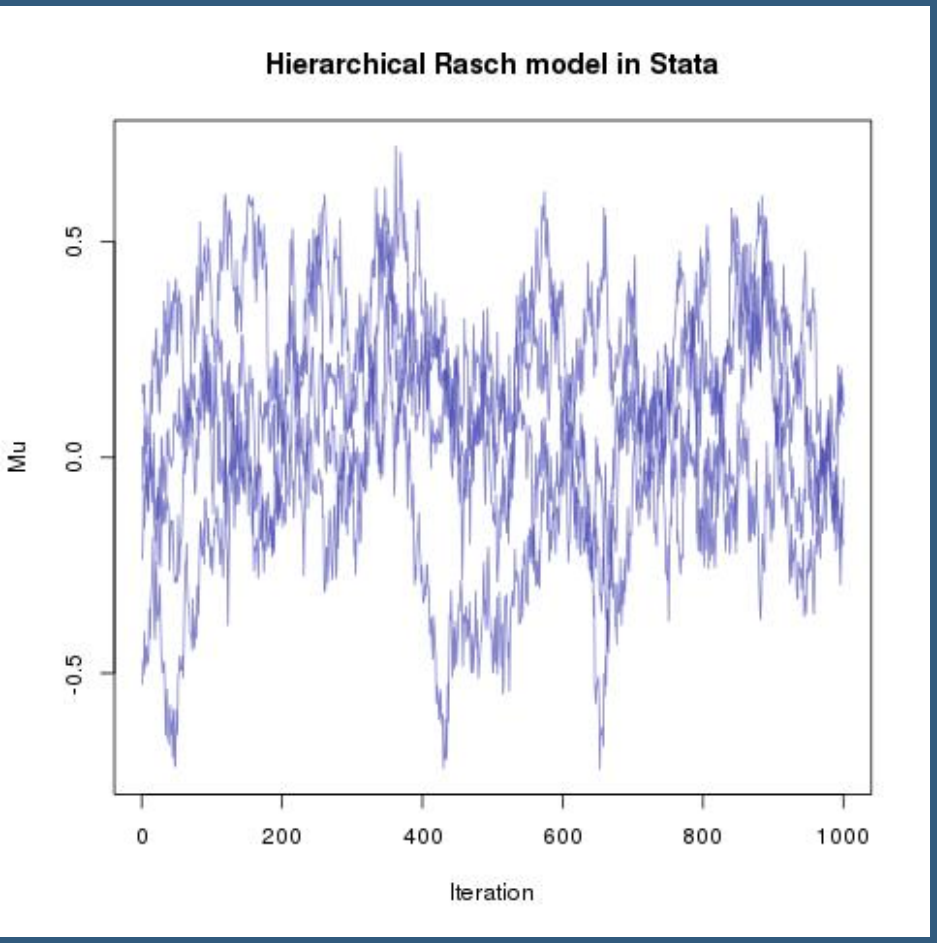

Hierarchical Rasch model in Stan
Delta[1]
-0.5
-1.0
-1.5
-2.0
-2.5
0
200
400
600
800
1000
Iteration

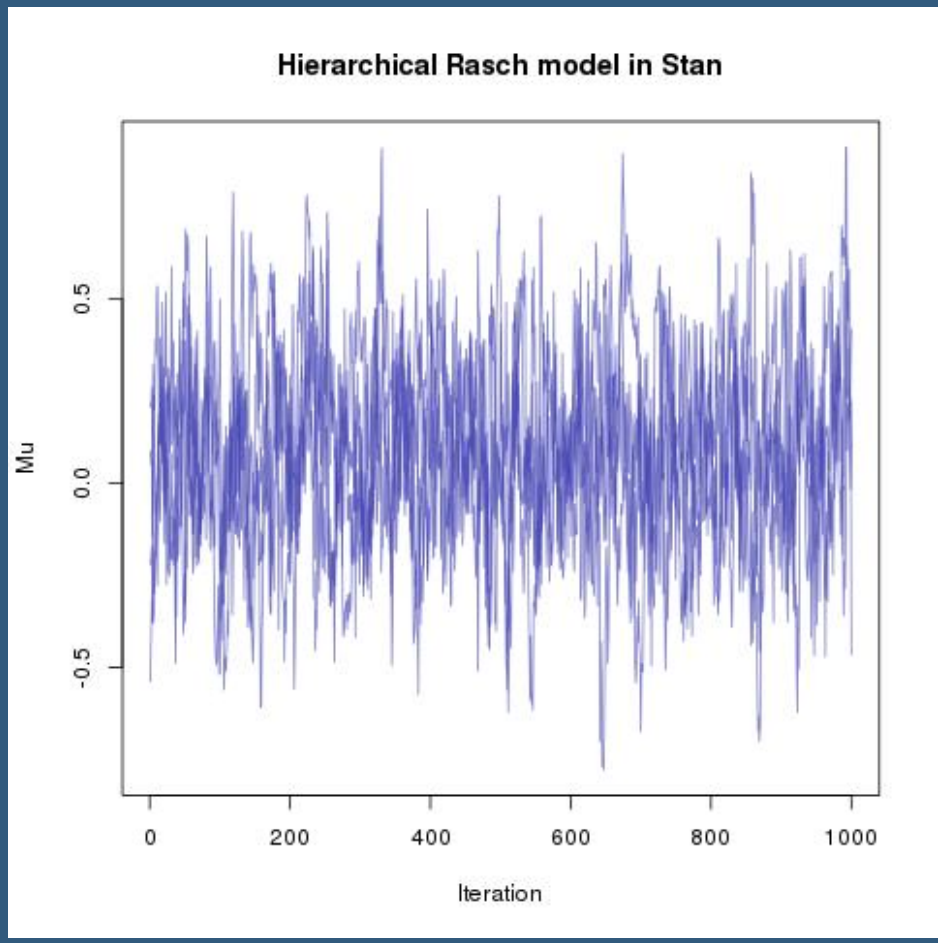

Rasch model in Stata
Theta[1]
Iteration
1
0
-1
-2
0
200
400
600
800
1000

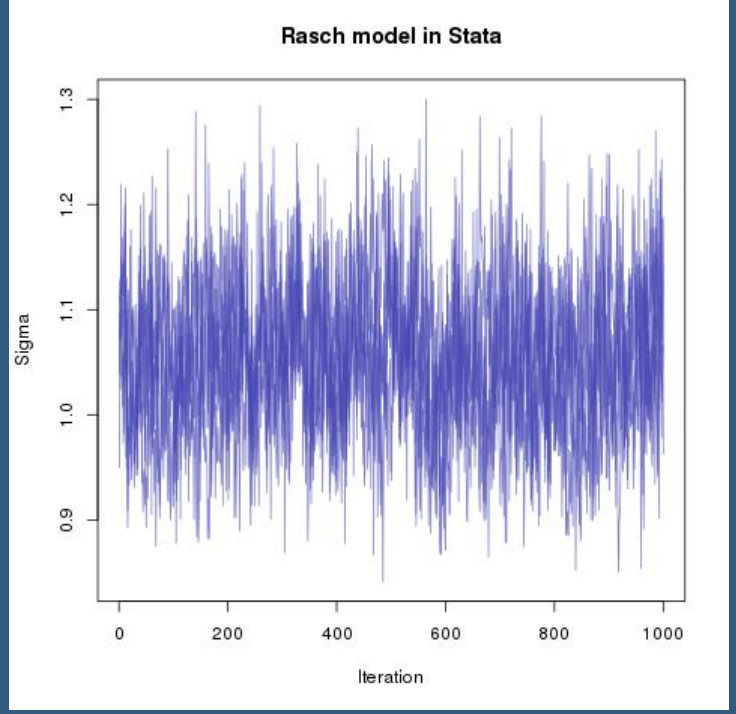

Rasch model in Stan
Theta[1]
Iteration
1
0
-1
-2
0
200
400
600
800
1000

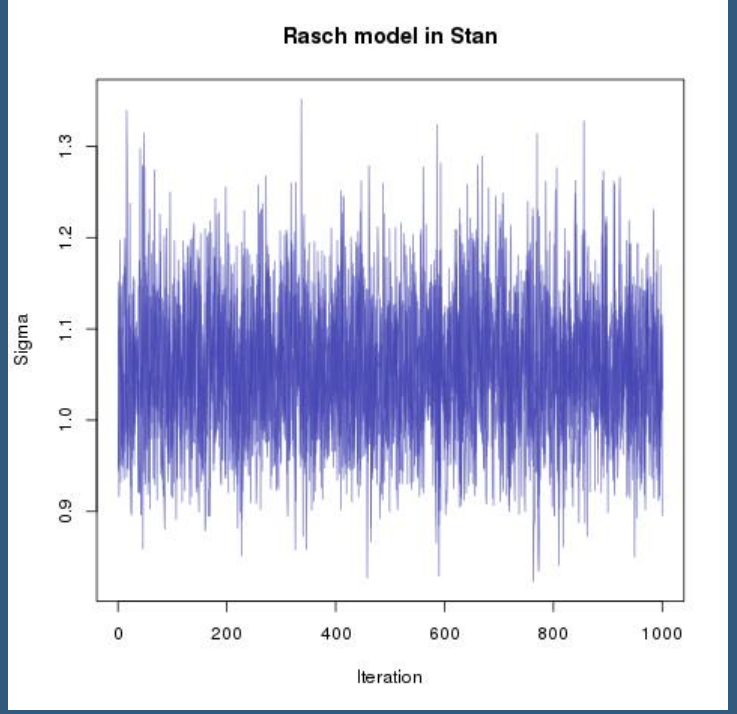

Hierarchical Rasch model in Stata
Theta[1]
Iteration
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
0
200
400
600
800
1000

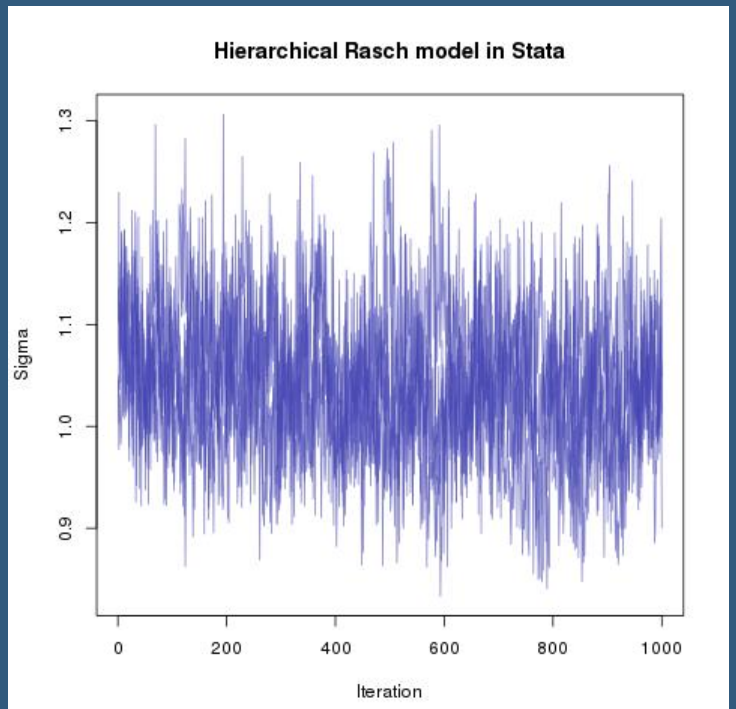

Hierarchical Rasch model in Stan
Theta[1]
Iteration
1.0
0.5
0.0
-0.5
-1.0
-1.5
-2.0
0
200
400
600
800
1000

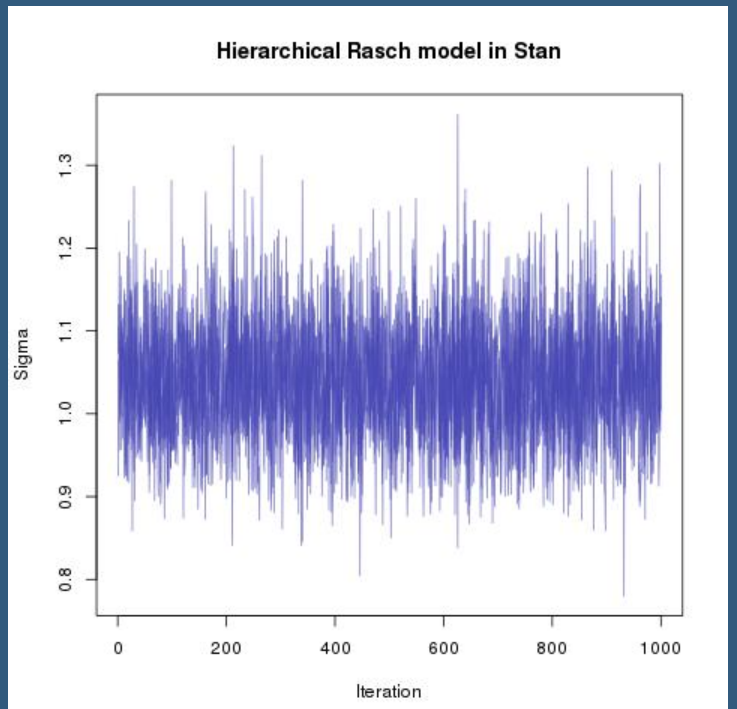

Hierarchical Rasch model in Stata
Mu
0.5
0.0
-0.5
0
200
400
600
800
1000
Iteration

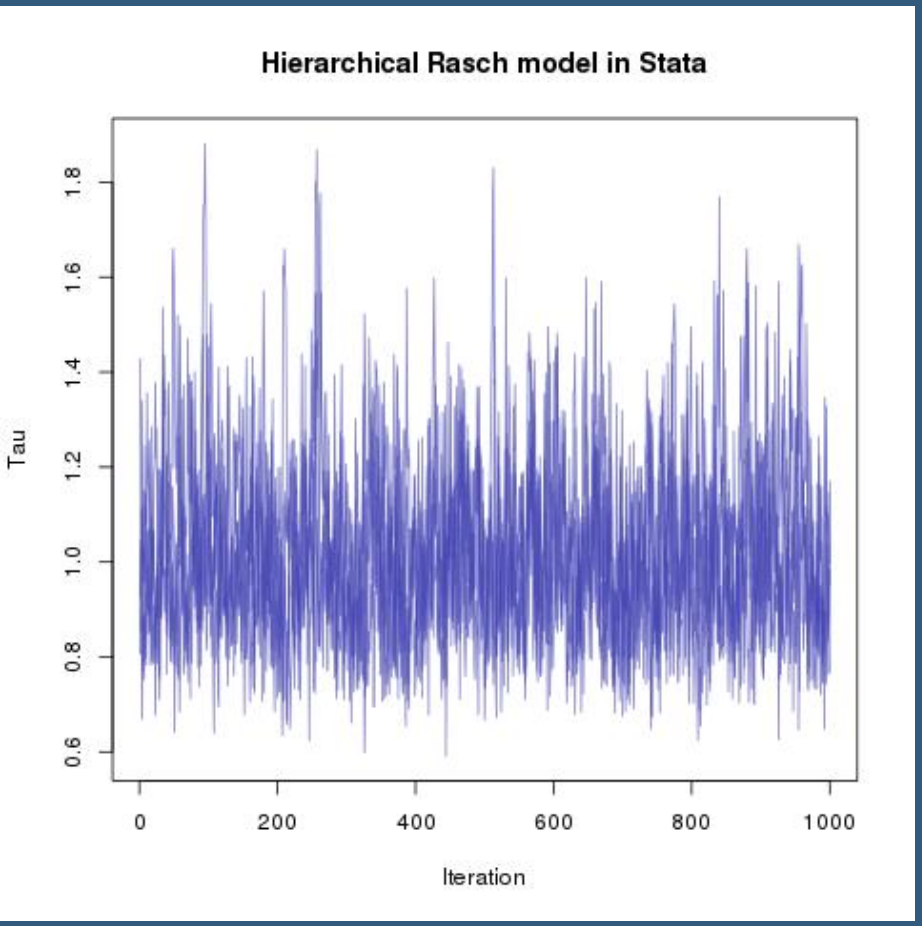

Hierarchical Rasch model in Stan
Mu
0.5
0.0
-0.5
0
200
400
600
800
1000
Iteration

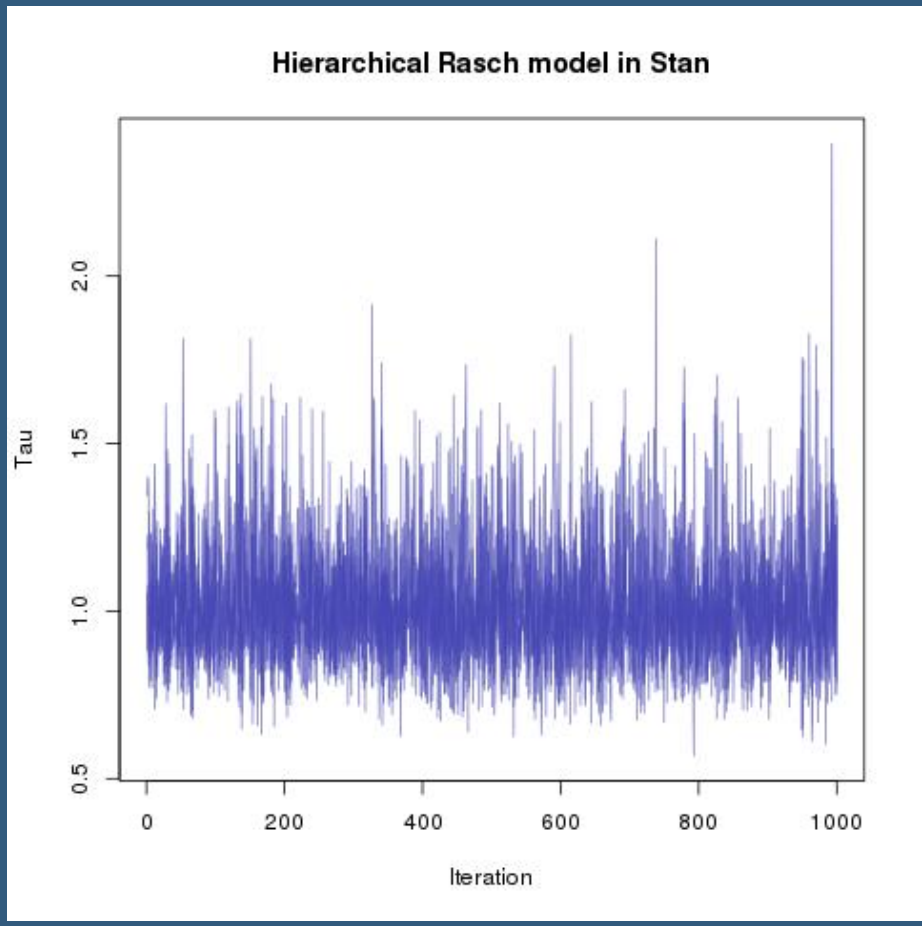

Rasch model in Stata
Sigma
Iteration
0.9
1.0
1.1
1.2
1.3
0
200
400
600
800
1000


Rasch model in Stan
Sigma
Iteration
0.9
1.0
1.1
1.2
1.3
0
200
400
600
800
1000


Hierarchical Rasch model in Stata
Sigma
Iteration
0.9
1.0
1.1
1.2
1.3
0
200
400
600
800
1000


Hierarchical Rasch model in Stan
Sigma
Iteration
0.8
0.9
1.0
1.1
1.2
1.3
0
200
400
600
800
1000


Hierarchical Rasch model in Stata
Tau
0.6
0.8
1.0
1.2
1.4
1.6
1.8
0
200
400
600
800
1000
Iteration


Hierarchical Rasch model in Stan
Tau
0.5
1.0
1.5
2.0
0
200
400
600
800
1000
Iteration

80-280 times faster is not just luxury

stan（Stata command）

```
// make your data

clear
set obs 10
gen y=0
replace y=1 in 2
replace y=1 in 10
count
global N=r(N)
```

stan（Stata command）

```
data {
      int<lower=0> N;
      int<lower=0,upper=1> y[N];
}
parameters {
      real<lower=0,upper=1> theta;
}
model {
      theta ~ beta(1,1);
      for (n in 1:N) {
            y[n] ~ bernoulli(theta);
      }
}
```

stan（Stata command）

```
// Version 1: write a separate model file

// call Stan, providing the modelfile option

stan y, modelfile("bernoulli.stan") ///
     cmd("$cmdstandir") globals("N")
```

stan (Stata command)

```
/* Version 2: specify the model inline, the John Thompson
way (in a comment block), naming THIS do-file in the
thisfile option */

/*
data {
...
}
*/

// call Stan with the inline and thisfile options.
// modelfile now tells it where to save your model

stan y, inline modelfile("inline-bernoulli.stan") ///
      thisfile("bernoulli.do") ///
      cmd("$cmdstandir") globals("N") load mode
```

stan (Stata command)

```
/* Version 3: use the comment block, but don't provide
thisfile - Stata will go looking for it in c(tmpdir),
which saves you typing in the do-file name and path, but
might not work sometimes */

/*
data {
...
}
*/
// call Stan with the inline and thisfile options.
// modelfile now tells it where to save your model

stan y, inline modelfile("inline-bernoulli.stan") ///
      cmd("$cmdstandir") globals("N") load mode
```

stan (Stata command)

```
/* Version 4: inline model, the Charles Opondo way */

tempname writemodel
file open `writemodel' using "mystanmodel.stan", write
#delimit ;
foreach line in
      "data { "
      "  int<lower=0> N; "
...
      #delimit cr
      file write `writemodel' "`line'" _n
}
file close `writemodel'

stan y, modelfile("mystanmodel.stan") ///
      cmd("$cmdstandir") globals("N") load mode
```

stan_schools

Shortcut command to plug your data into the BUGS Example called “schools”

```
stan_schools y lrt vr female, ///
    globals("N M") hetvar(lrt) ///
    clusterid(school) ///
    rslopes(s_denom s_gender) ///
    betapriors("normal(0,100)") ///
    stanopts(cmdstandir("$cdir") mode)
```

https://github.com/stan-dev/example-models/wiki/BUGS-Examples

stan_rasch

Shortcut command to plug your data into the Rasch IRT model

```
stan_rasch y, globals("N P I") ///
    personid(pp) itemid(ii) ///
    deltapriors("normal(0,100)") ///
    thetapriors("normal(0,100)") ///
    stanopts(cmdstandir("$cdir") mode)
```

stan_c14cal

Shortcut command to plug your data into Andrew Millard's radiocarbon calibration

```
stan_c14cal m xdate sigma, ///
    globals("mintheta" "maxtheta" ///
            "nDate") ///
    curve("c1986") ///
    stanopts(cmdstandir("$cdir") mode)
```

http://community.dur.ac.uk/a.r.millard/BUGS4Arch.html

Spin-offs: windowsmonitor

Shows the result of a long winexec call (like Stan) in Stata while it's running

This just happens under Linux and Mac, but not Windows...

```
windowsmonitor, waitsecs(20)///
command("ping 127.0.0.1 -n 30")
```

More spin-offs using the inline code idea

statar

statacpp

Getting started

http://mc-stan.org/interfaces/cmdstan.html

https://github.com/stan-dev/statastan