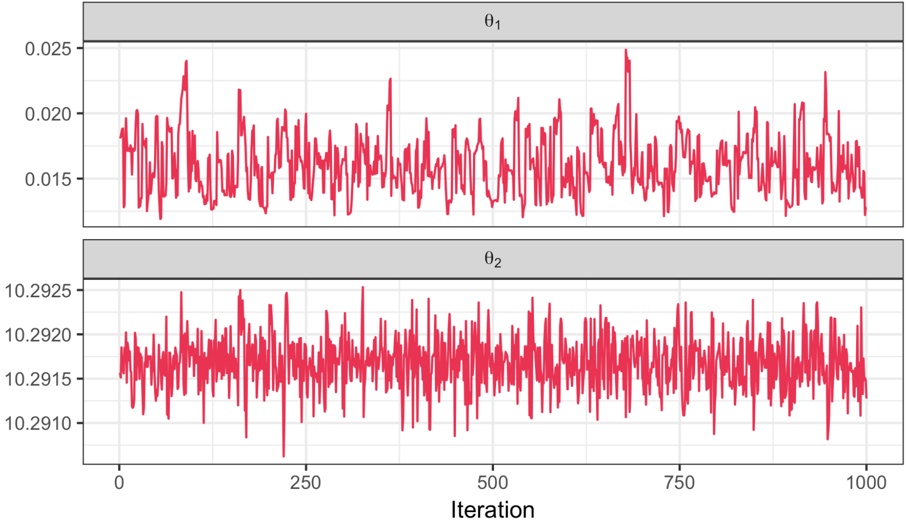
Quantitative Marketing SIG: A Gentle Introduction to Estimation Bayesian Models Using Stan

Jason M.T. Roos

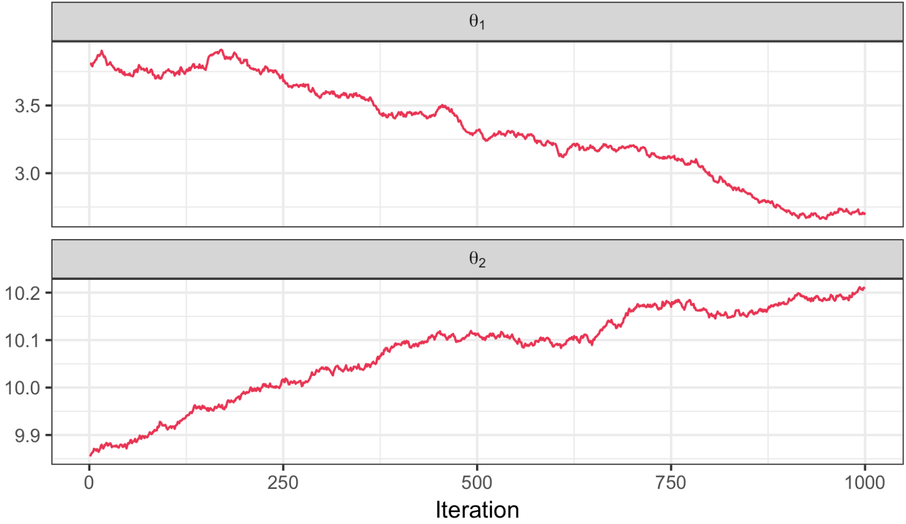
EMAC 2018

Motivation

Why we're talking about Stan today



Why we're talking about Stan today



Why we're talking about Stan today

- Highly efficient sampling from complex Bayesian models that Gibbs and Metropolis-Hastings might fail at
- Interfaces to R, Python, Matlab, etc.
- Coding errors limited to the model (not sampling algorithm)
- Diagnostic tools to evaluate if your sampler works
- Might require you to learn a new programming language (not necessarily)

Roadmap

Part 1: Introduction to Hamiltonian Monte Carlo and Stan

- Sampling from complex Bayesian models using standard methods is inefficient and error-prone
- Hamiltonian Monte Carlo (HMC) offers huge improvements
- Intuition for how HMC works
- Implementing an HMC sampler in Stan

Part 2: Alina Ferecatu on hierarchical logit and models of bounded rationality

Part 3: Hernan Bruno on multivariate Tobit and two-stage "hurdle" models

Setup

- **Goal:** Sample from some distribution (the target)
  - Typically a Bayesian posterior distribution, $\pi(\theta|y, x)$
  - But generally any distribution $\pi(\theta)$
- **Requirements:**
  - All elements $\theta_i \in \theta$ (parameters) are continuous
    - Discrete parameters cannot be sampled
    - Usually they can be integrated out before sampling
  - Target distribution can be evaluated at any permitted value of θ
    - With or without normalizing constant

Common approaches for Bayesian models

- Metropolis-Hastings (MH) or Gibbs sampling
- Typical problems:
  - High parameter correlation kills efficiency
  - Finite chains may dramatically over- and under-sample certain regions, with biased inferences
    - Convergence guaranteed *asymptotically*
- Many alternatives alleviate these problems
  - One discussed today: Hamiltonian Monte Carlo (HMC)

Hamiltonian Monte Carlo

Hamiltonian mechanics

- Idealized physical model of random particle motion
- A particle's potential energy is $-\log \pi(\theta)$
  - Start thinking of θ as the parameter vector and $\pi(\theta)$ its density
- Particle has mass M, momentum p, and position θ
  - Start thinking of p as the random step in standard random-walk Metropolis, and M as its step size
- Total energy of the physical system is constant

Hamiltonian equations

- Hamilton's equations describe the particle's motion in continuous time

$$\text{Change in position:} \quad \frac{\mathrm{d}\theta}{\mathrm{d}t} = M^{-1}p$$

$$\text{Change in momentum:} \quad \frac{\mathrm{d}p}{\mathrm{d}t} = \nabla_\theta \log\left[\pi\left(\theta\right)\right]$$

- Motion is like "a frictionless puck that slides over a surface of varying height" (Neal 2011, p.2)
- I prefer: "a frictionless skateboarder in an empty swimming pool"

one_particle.mp4

Idealized version of Hamiltonian MC (HMC) sampling

- Take a particle and give it a random shove (momentum p)
  - Let it move for a while and then stop it
  - Record its position (value of the parameter vector θ)
- Give it another random shove
  - Let it more for a while and then stop it
  - Record its position again
- Repeat

Idealized version of Hamiltonian MC (HMC) sampling

hmct1.mp4...

Idealized version of Hamiltonian MC (HMC) sampling

- Take a particle and give it a random shove (momentum p)
  - Let it move for a while and then stop it
  - Record its position (value of the parameter vector θ)
- Give it another random shove
  - Let it more for a while and then stop it
  - Record its position again
- Repeat

Idealized version of Hamiltonian MC (HMC) sampling

hmct2.mp4...

Idealized version of Hamiltonian MC (HMC) sampling

- Take a particle and give it a random shove (momentum p)
    - Let it move for a while and then stop it
    - Record its position (value of the parameter vector θ)
- Give it another random shove
    - Let it more for a while and then stop it
    - Record its position again
- Repeat

Idealized version of Hamiltonian MC (HMC) sampling

hmct3.mp4...

Idealized version of Hamiltonian MC (HMC) sampling

hmct4.mp4...

Idealized version of HMC is just that…an ideal

- It would generate exact samples from target distribution
- However:
  - Analytical solutions to this continuous time model aren't available
  - Numerical approximation is necessary
- Solution:
  - Discretize the model by dividing time into discrete steps
  - Simulate the particle's motion in discrete time

Discretized version of HMC

- Discretize time into small steps of length ϵ, leading to sampling trajectories

$$\theta^{(t)} \rightarrow \theta^{(t+\epsilon)} \rightarrow \theta^{(t+2\epsilon)} \rightarrow \theta^{(t+3\epsilon)} \rightarrow \theta^{(t+4\epsilon)} \rightarrow \cdots \rightarrow \theta^{(t+1)}$$

  - Monte Carlo samples are t and $t+1$
  - $t+\epsilon$, $t+2\epsilon$, etc. are intermediate "leapfrog" steps
- No longer a trajectory in $\pi(\theta)$, but close
  - Must correct for discrepancy between continuous model and discrete approximation
  - Occasionally reject samples (as in MH) to correct for discrepancy

ani.mp4

Costs and benefits of standard HMC

- Benefits:
  - Rarely rejects proposals, lower autocorrelation
  - Almost always more efficient than Gibbs or MH
- Costs:
  - Need to compute $\nabla_\theta \log \pi(\theta)$, the gradient (first derivative) of the log of the target density, with respect to the parameters θ, for all L intermediate steps
  - Calculus is hard
    - However: Automatic differentiation will save us

Detour: What is automatic differentiation?

- While computing the value of a function, obtain exact values of derivatives of that function
- Not magic: Exploit the chain rule from calculus:

$$(f \circ g)' = (f' \circ g)g' \qquad \text{...or...}$$

$$\frac{\partial}{\partial x} f(g(x)) = f'(g(x))g'(x)$$

- Example: mean μ of normal distribution:

$$\begin{array}{ccccccccc} -\frac{1}{2}(y-\mu)^2 & = & \mu & \rightarrow & y-(\cdot) & \rightarrow & (\cdot)^2 & \rightarrow & -\frac{1}{2}(\cdot) \\ & & & & \downarrow & & \downarrow & & \downarrow \\ \frac{\partial}{\partial \mu} & = & & & -1 & \times & 2(y-\mu) & \times & -\frac{1}{2} \end{array}$$

Comparison of HMC and RW Metropolis

hmc.rw.mp4

Stan

Stan for Hamiltonian Monte Carlo

- In its simplest form, Stan implements an HMC sampler
  - You specify the target distribution $\pi(\theta)$ in a way Stan can understand
  - Stan generates and compiles C++ code to evaluate $\pi(\theta)$ *and* $\nabla_\theta \log \pi(\theta)$
  - Stan adapts the HMC step size during a burn-in phase
- HMC samplers are (notoriously?) difficult to tune
  - The the total length of the path followed by the particle (integration length) affects sampling efficiency

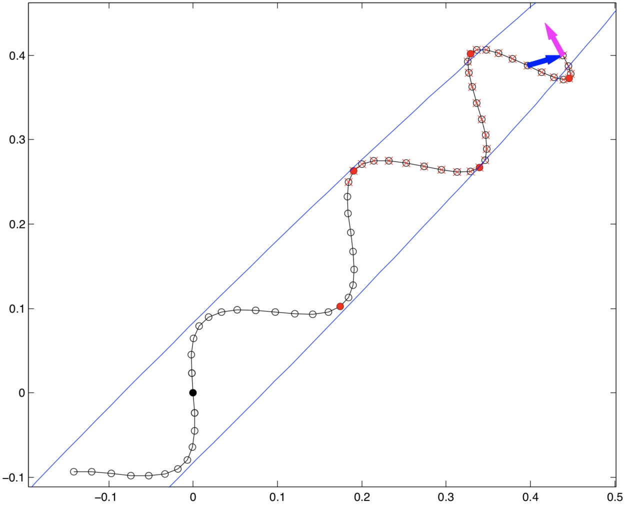
An inefficient HMC sampler

badL.mp4

Stan's NUTS sampler

- Stan also implements the No U-Turn Sampler
- Stops the particle when the sampler detects it has started making a U-Turn
- Only tuning parameter needed is ϵ (the step size) which Stan tunes during burn-in

Stopping before a U-turn



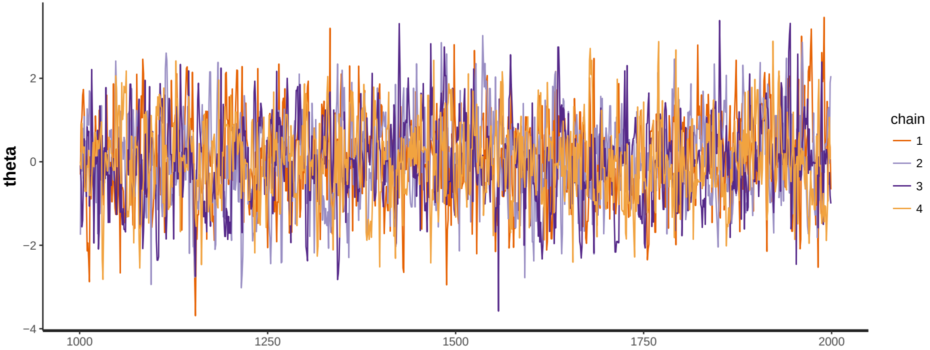
Basics of a Stan model

```
parameters {
  real theta;
}
model {
  theta ~ normal(0, 1);
}
```

```
sm <- stan_model(model_code = ...)
fit <- sampling(sm)
```

Output from a basic Stan model

```
stan_trace(fit)
```



Bayesian linear regression example

- Data y and X
  - n observations in y and X
  - p columns in X
- Likelihood: $y|X \sim N(\alpha + X\beta, \sigma^2)$
- Priors:
  - $\alpha, \beta \sim N(0, 1)$
  - $\sigma \sim Expo(1)$

Stan model for linear regression

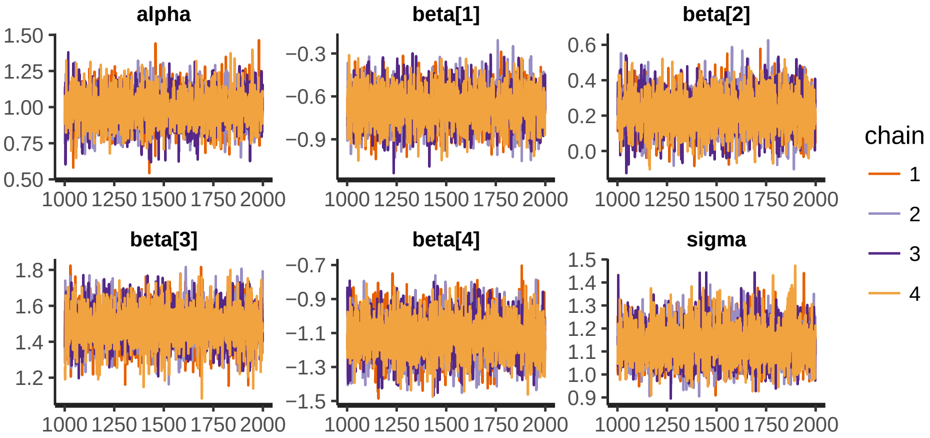
```
data {
  int<lower = 0> n;
  int<lower = 0> p;
  vector[n] y;
  matrix[n, p] X;
}
parameters {
  real alpha;
  vector[p] beta;
  real<lower = 0> sigma;
}
model {
  alpha ~ normal(0, 1);
  beta ~ normal(0, 1);
  sigma ~ exponential(1);

  y ~ normal(alpha + X * beta, sigma);
}
```

Compiling and sampling from R

```r
library(rstan)
sm <- stan_model(file = 'my_model.stan')

X <- ...
y <- ...
d <- list(n = nrow(X), p = ncol(X),
          X = X, y = y)

fit <- sampling(sm, data = d)
```

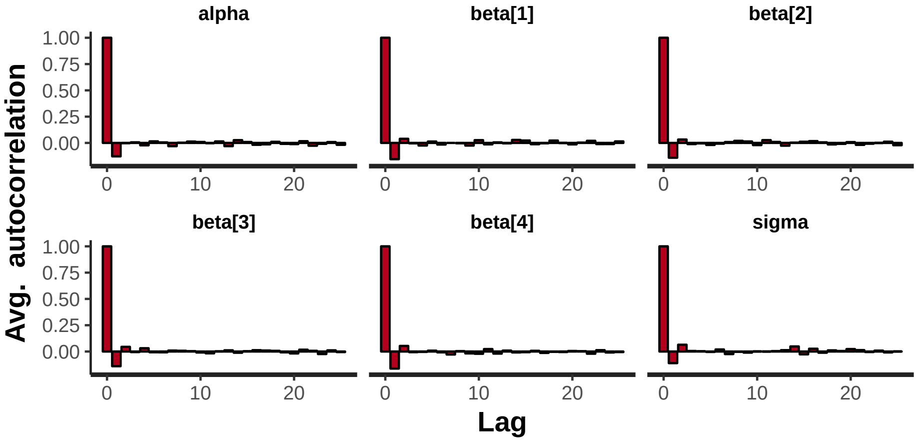
Trace plots

```
stan_trace(fit)
```



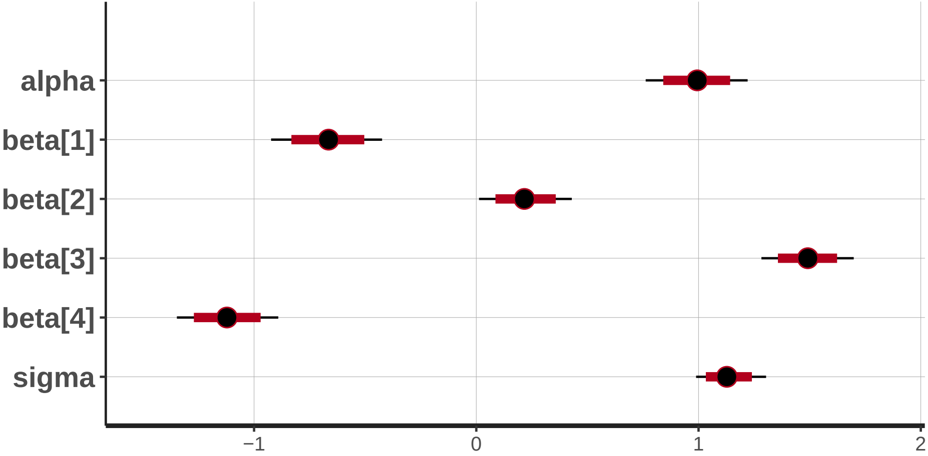
Sample autocorrelation

```
stan_ac(fit)
```



Posterior means and intervals

```
stan_plot(fit)
```



Parts of a Stan model

```
functions { ... }

data { ... }

transformed data { ... }

parameters { ... }

transformed parameters { ... }

model { ... }

generated quantities { ... }
```

Stan Inside™

What if I don't want to write my own Stan code?

```
library(rstanarm)
fit <- stan_glm(y ~ 1 + X1 + X2 + X3 + X4, data = d,
                prior = normal(0, 1),
                prior_intercept = normal(0, 1),
                prior_aux = exponential(1))
```

- `rstanarm` uses Stan to estimate complex hierarchical and non-gaussian models
- Created by Stan team, integrates nicely with `bayesplot`
  - Another alternative is `brms`, but `rstanarm` seems better so far

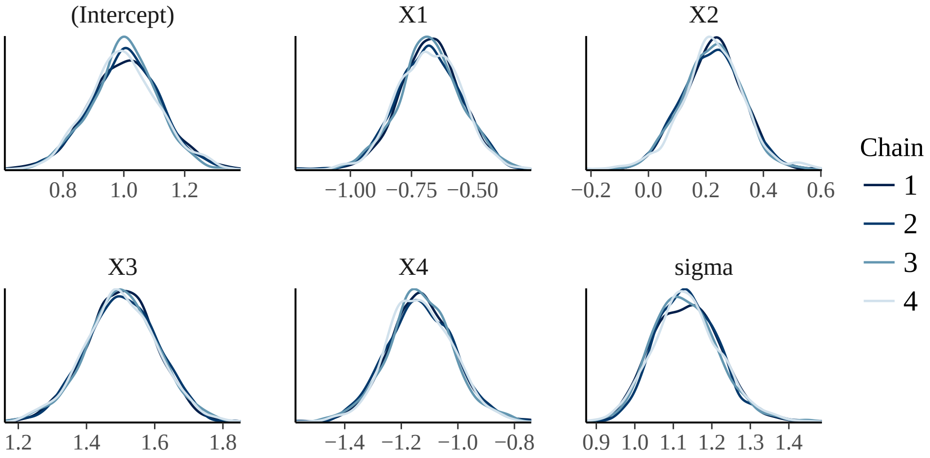
```r
summary(fit)
```

```
Model Info:

 function:     stan_glm
 family:       gaussian [identity]
 formula:      y ~ 1 + X1 + X2 + X3 + X4
 algorithm:    sampling
 priors:       see help('prior_summary')
 sample:       4000 (posterior sample size)
 observations: 100
 predictors:   5

Estimates:
                mean   sd     2.5%   25%    50%    75%    97.5%
(Intercept)     1.0    0.1    0.8    0.9    1.0    1.1    1.2
X1             -0.7    0.1   -0.9   -0.8   -0.7   -0.6   -0.4
X2              0.2    0.1    0.0    0.2    0.2    0.3    0.4
X3              1.5    0.1    1.3    1.4    1.5    1.6    1.7
X4             -1.1    0.1   -1.3   -1.2   -1.1   -1.1   -0.9
sigma           1.1    0.1    1.0    1.1    1.1    1.2    1.3
mean_PPD        1.1    0.2    0.8    1.0    1.1    1.2    1.4
log-posterior -161.3    1.8 -165.8 -162.2 -161.0 -160.0 -158.9
```

```
Diagnostics:
              mcse Rhat n_eff
(Intercept)   0.0  1.0  4000
X1            0.0  1.0  4000
X2            0.0  1.0  4000
X3            0.0  1.0  4000
X4            0.0  1.0  4000
sigma         0.0  1.0  4000
mean_PPD      0.0  1.0  4000
log-posterior 0.0  1.0  1724

For each parameter, mcse is Monte Carlo standard error, n_eff is a crude
measure of effective sample size, and Rhat is the potential scale reduction
factor on split chains (at convergence Rhat=1).
```

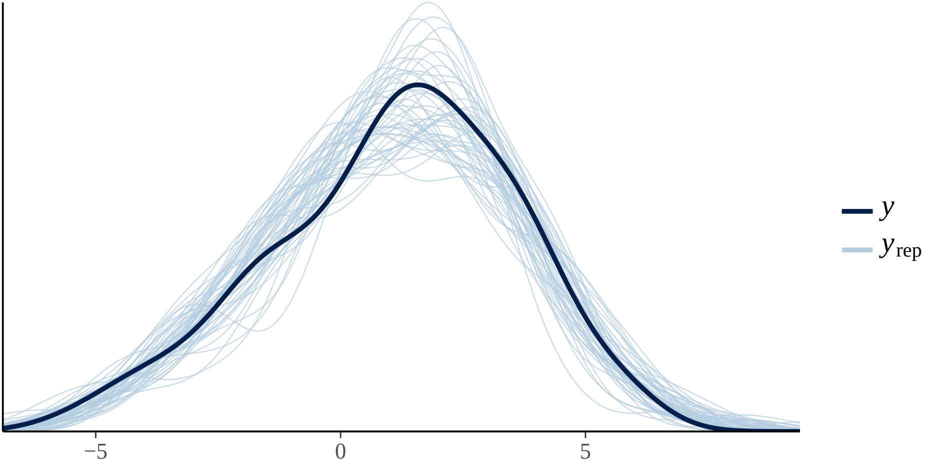
Density overlays

```
fit %>% as.array() %>% bayesplot::mcmc_dens_overlay()
```



Posterior predictive checks

```
pp_check(fit)
```



Automatic integration with loo

```
loo(fit)

Computed from 4000 by 100 log-likelihood matrix

         Estimate   SE
elpd_loo   -157.0  5.4
p_loo         5.5  0.7
looic       314.1 10.9
------
Monte Carlo SE of elpd_loo is 0.0.

All Pareto k estimates are good (k < 0.5).
See help('pareto-k-diagnostic') for details.
```

Conclusion

Why Stan is so important

- Coding errors confined to model specification
- If Stan fails, more likely due to a problem with your model than Stan
  - Numerically ill-conditioned
  - Non-identified
  - Improper posterior
- Nothing privileged about conjugacy
  - Choose priors based on what makes sense for the model
- Stan best practices and defaults should be MCMC best practices and defaults
  - Sampling diagnostics based on output from HMC
  - $\hat{R}$ for assessing convergence
  - Model comparison via the `loo` package

Roadmap

Part 1: Introduction to Hamiltonian Monte Carlo and Stan

- Sampling from complex Bayesian models using standard methods is inefficient and error-prone
- Hamiltonian Monte Carlo offers huge improvements
- Intuition for how HMC works
- Implementing an HMC sampler in Stan

Part 2: Alina Ferecatu on hierarchical logit and models of bounded rationality

Part 3: Hernan Bruno on multivariate Tobit and two-stage "hurdle" models