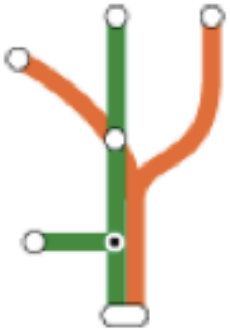
What is Stan?

Stan consists of the following largely independent components



Graphical Model Compiler

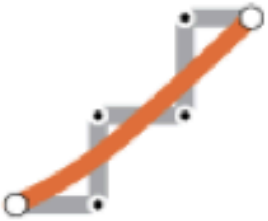
Parser and Code Generator

- Directed Graphical Model Compiler

No U-Turn Sampler

Automatic Step Size and Number Adaptation

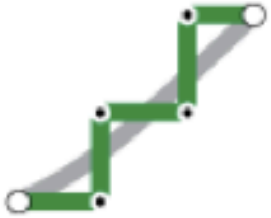
- (Adaptive) Hamiltonian Monte Carlo Sampling



Hamiltonian Express

Unconstrained Parameters

- Hamiltonian Monte Carlo Sampling



Gibbs Local

Discrete or Constrained Parameters

- Gibbs Sampling for Discrete Parameters
- Reverse Mode Algorithmic Differentiation
- Probability Distributions
- Special Functions
- Matrices and Linear Algebra

Models may be written directly using C++ or in the BUGS-like language which is compiled to C++.

Stan: a program for Bayesian data analysis with complex models

Andrew Gelman, Bob Carpenter, and Matt Hoffman
with Ben Goodrich, Michael Malecki, and Daniel Lee
Department of Statistics, Columbia University, New York

and Chad Scherrer, Pacific Northwest National Laboratories

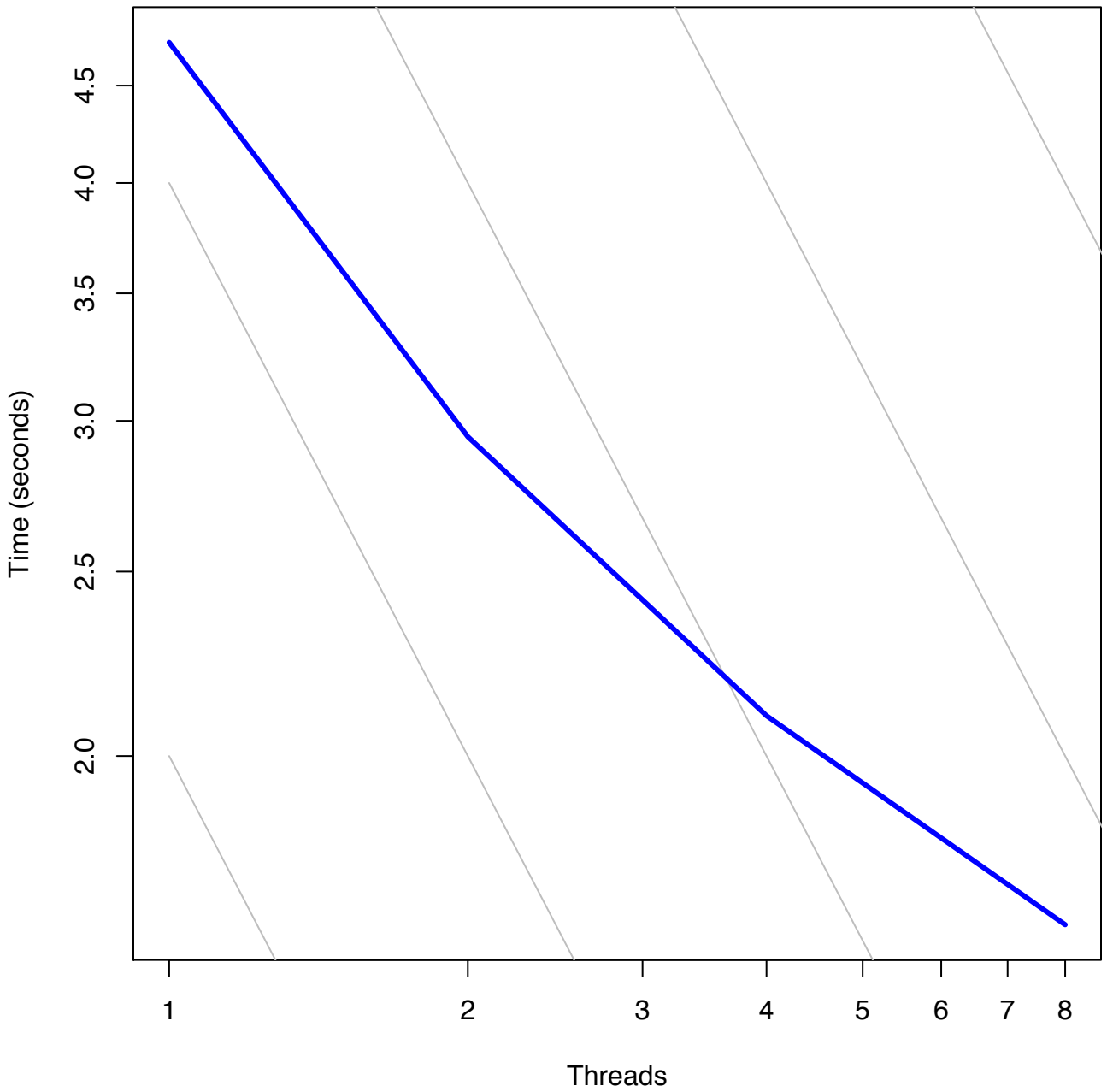
Project "Petascale Hierarchical Modeling via Parallel Execution"
Funded by the Department of Energy Office of Advanced Scientific Computing Research

17 Oct 2011

Passage: a parallel sampler generator for hierarchical Bayesian modeling

- Chad Scherrer, Pacific Northwest National Laboratories, Iavor Diatchki and Levent Erkok, Galois, Inc.
- Domain specific embedded language for graphical models
  - Brings in power of Haskell (lazy evaluation, functional)
  - Abstract syntax tree for efficient proportional conditionals
  - Higher-order functions for commonly used submodels
- Generates openMP code, parallel C API for multi-core
  - Graphical model node coloring uncovers parallel structure
  - Non-interfering blocks of parameters updated in parallel

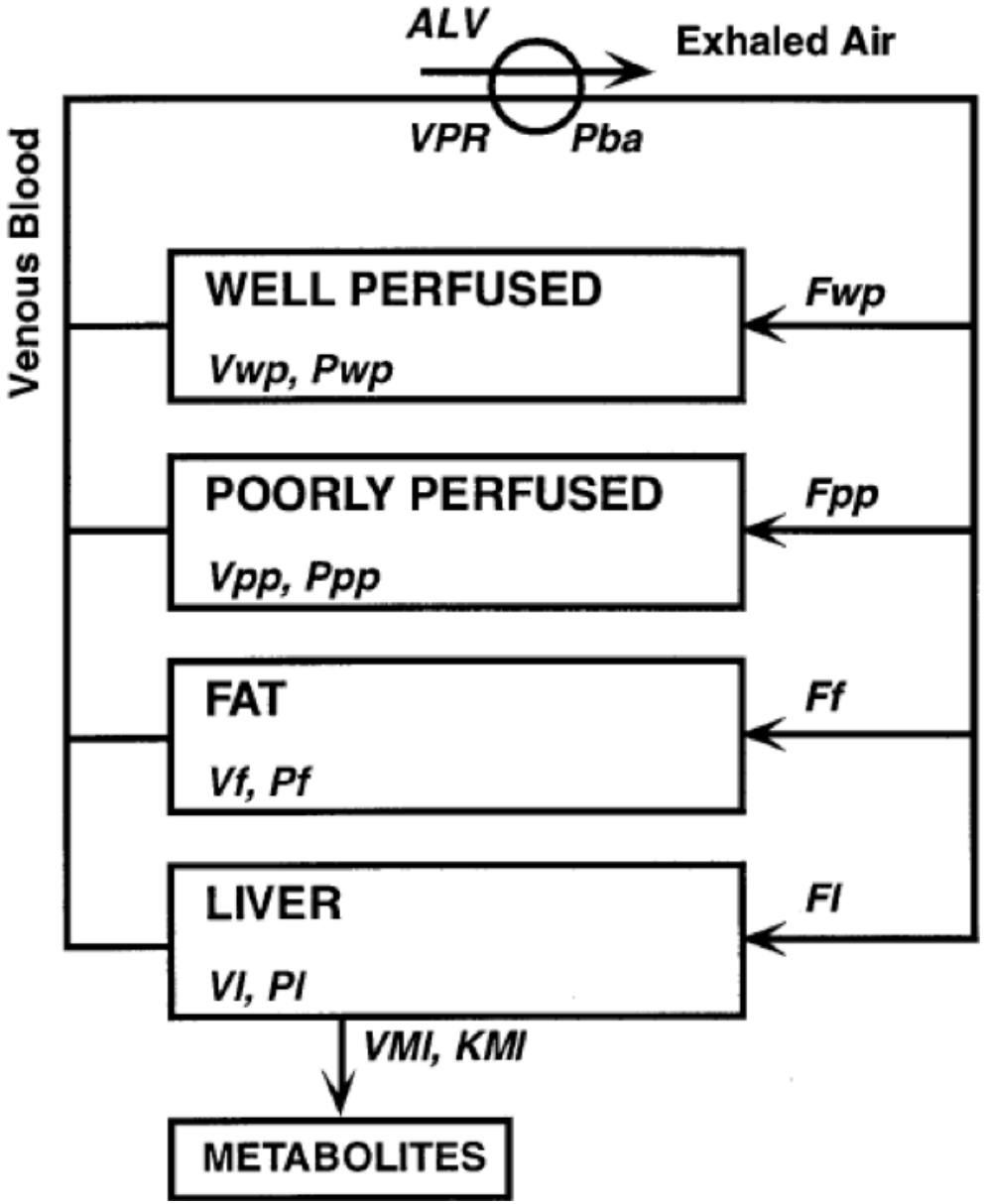
Practical gains from Passage's parallelism



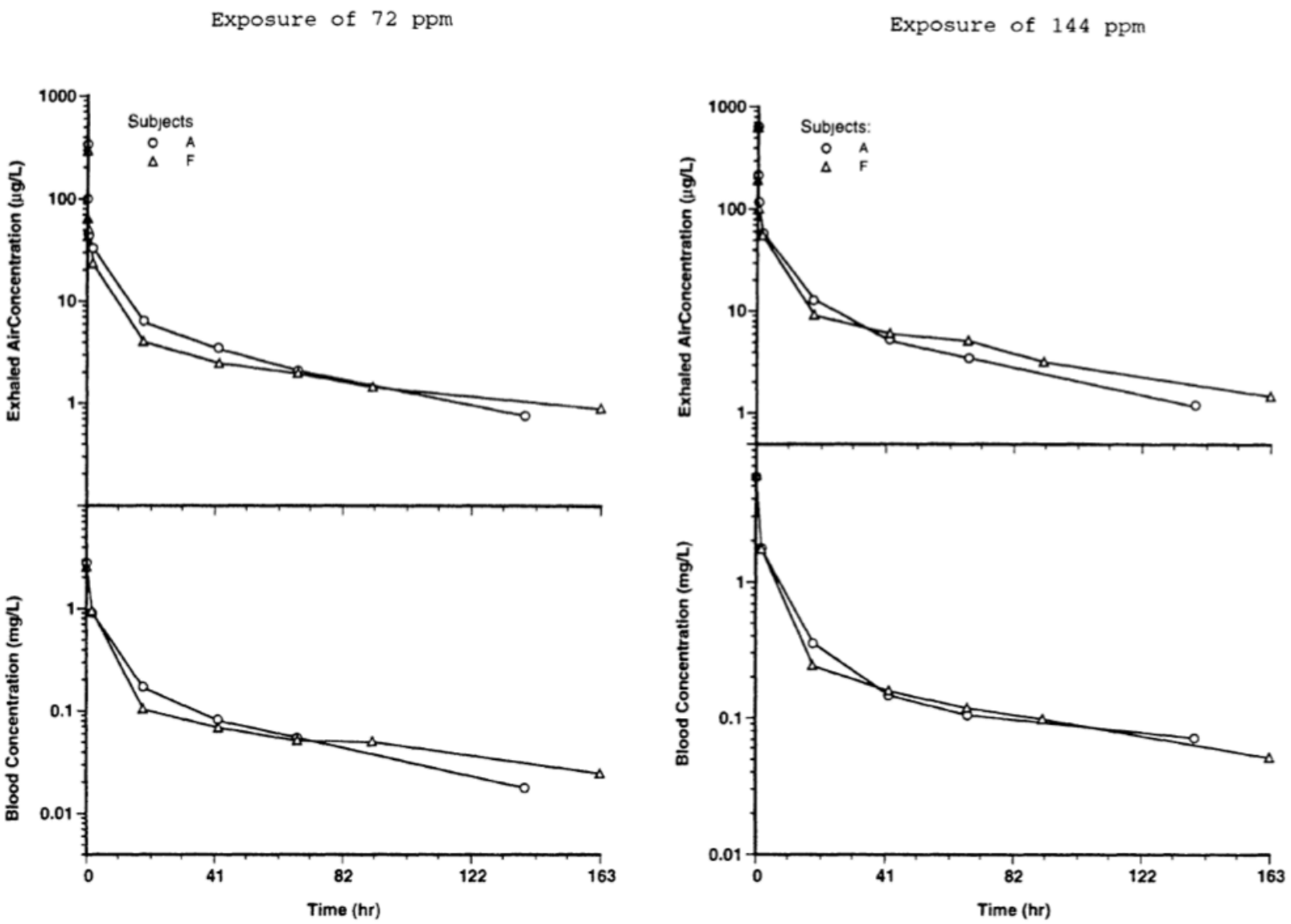
Stan for Bayes

- Example from toxicology
- What is Bayesian data analysis?
- Motivation for Stan
- How Stan works
- Next steps

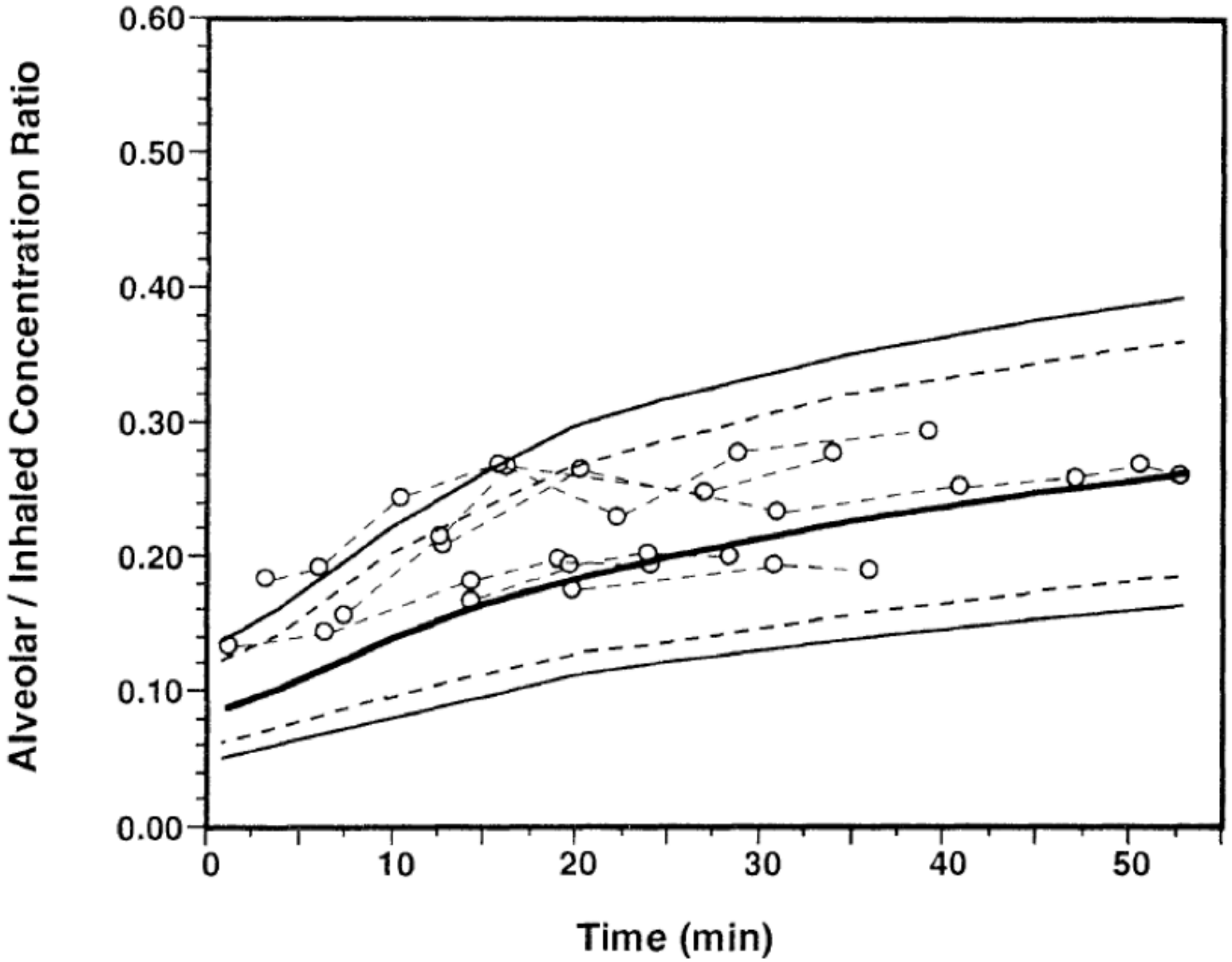
Example from toxicology



Not a lot of data



External validation



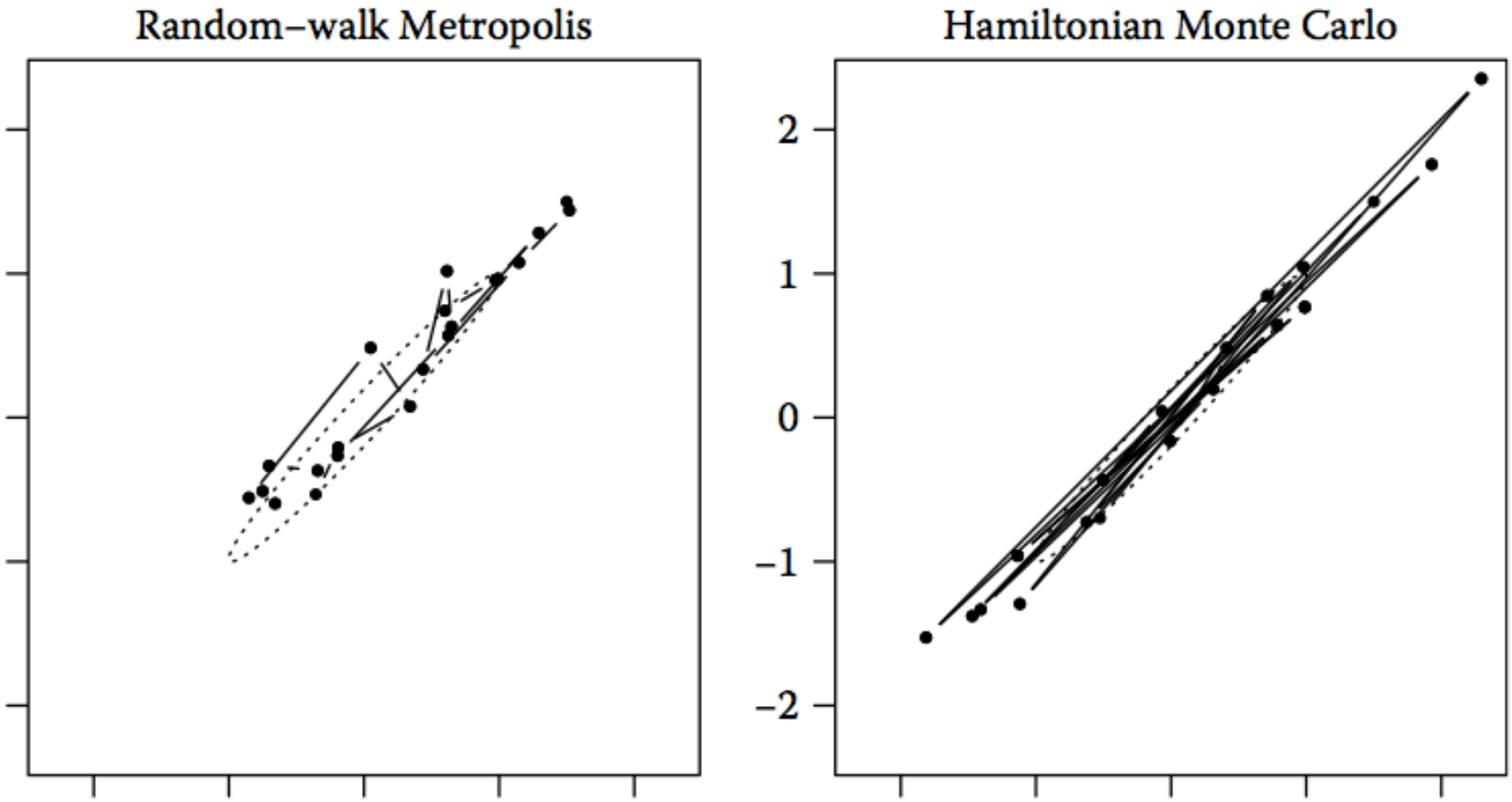
Steps of Bayesian data analysis

- Model building
- Inference
- Model checking
- Model understanding and improvement

Background on Bayesian computation

- Point estimates and standard errors
- Hierarchical models
- Posterior simulation
- Markov chain Monte Carlo (Gibbs sampler and Metropolis algorithm)
- Hamiltonian Monte Carlo

Radford Neal (2011) on Hamiltonian Monte Carlo

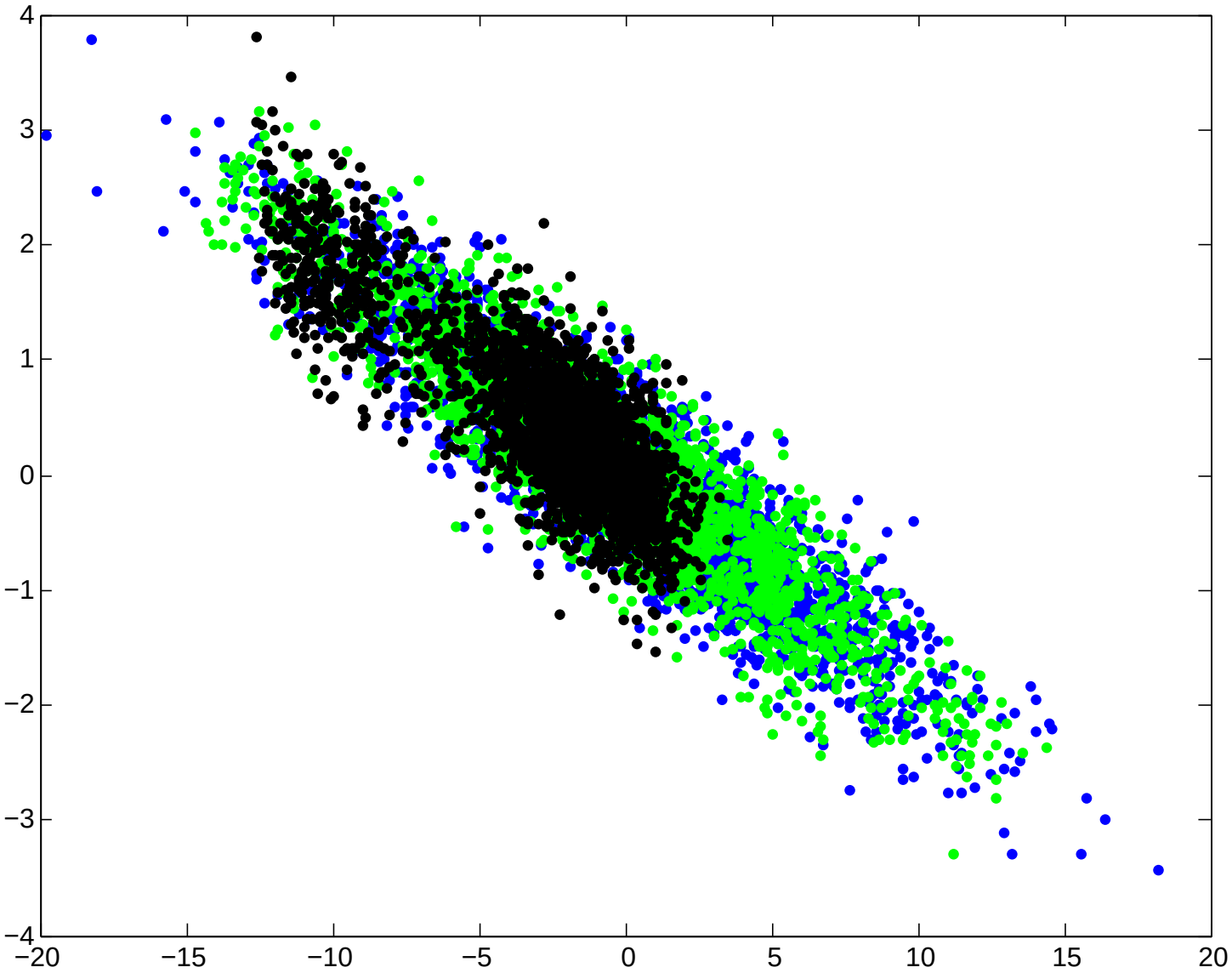


"One practical impediment to the use of Hamiltonian Monte Carlo is the need to select suitable values for the leapfrog stepsize, ϵ, and the number of leapfrog steps L . . . Tuning HMC will usually require preliminary runs with trial values for ϵ and L . . . Unfortunately, preliminary runs can be misleading . . ."

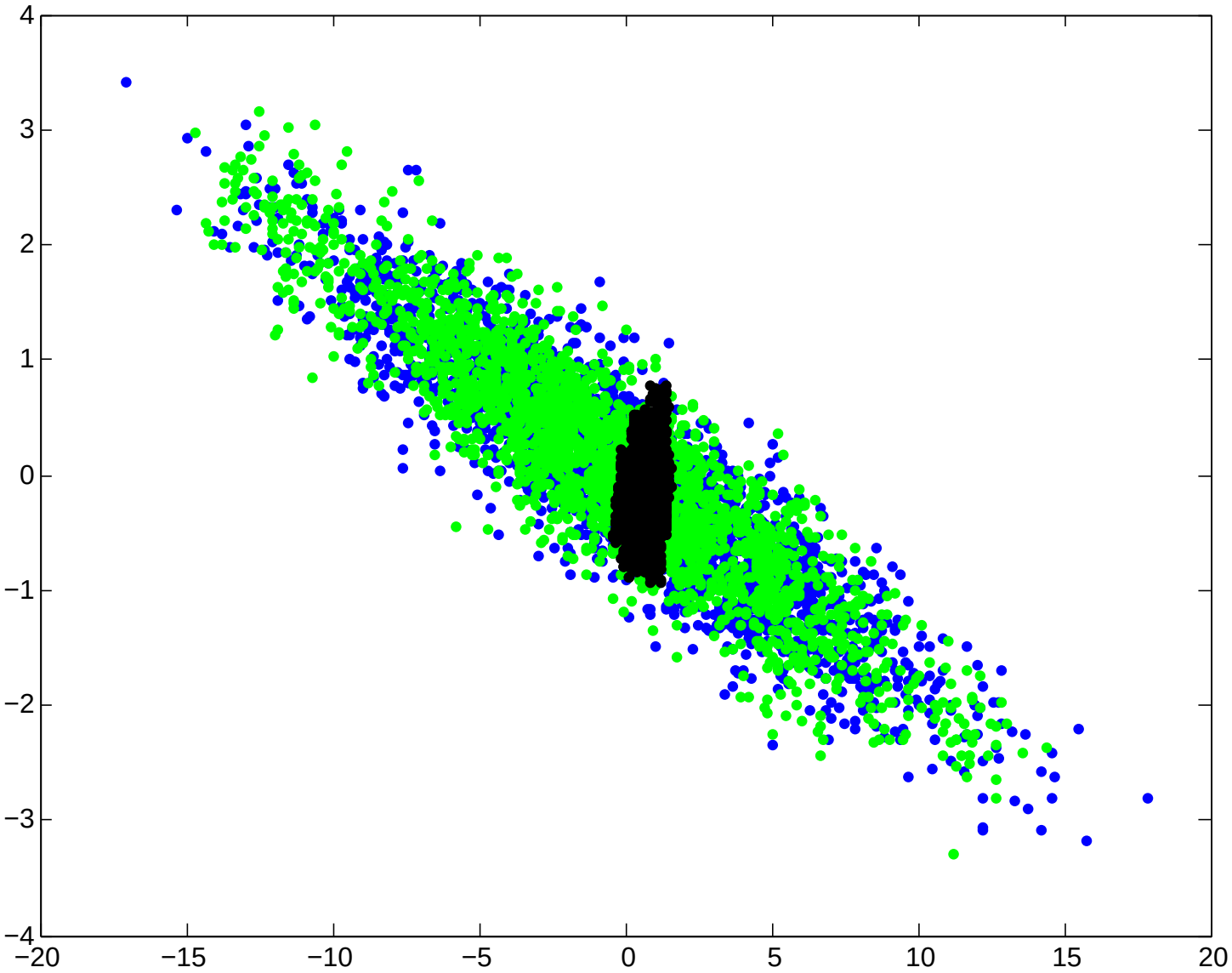
The No U-Turn Sampler

- Created by Matt Hoffman (recent computer science Ph.D.)
- Run the HMC steps until they start to turn around (bend with an angle $> 180°$)
- Computationally efficient
- Requires no tuning of #steps
- Complications to preserve detailed balance

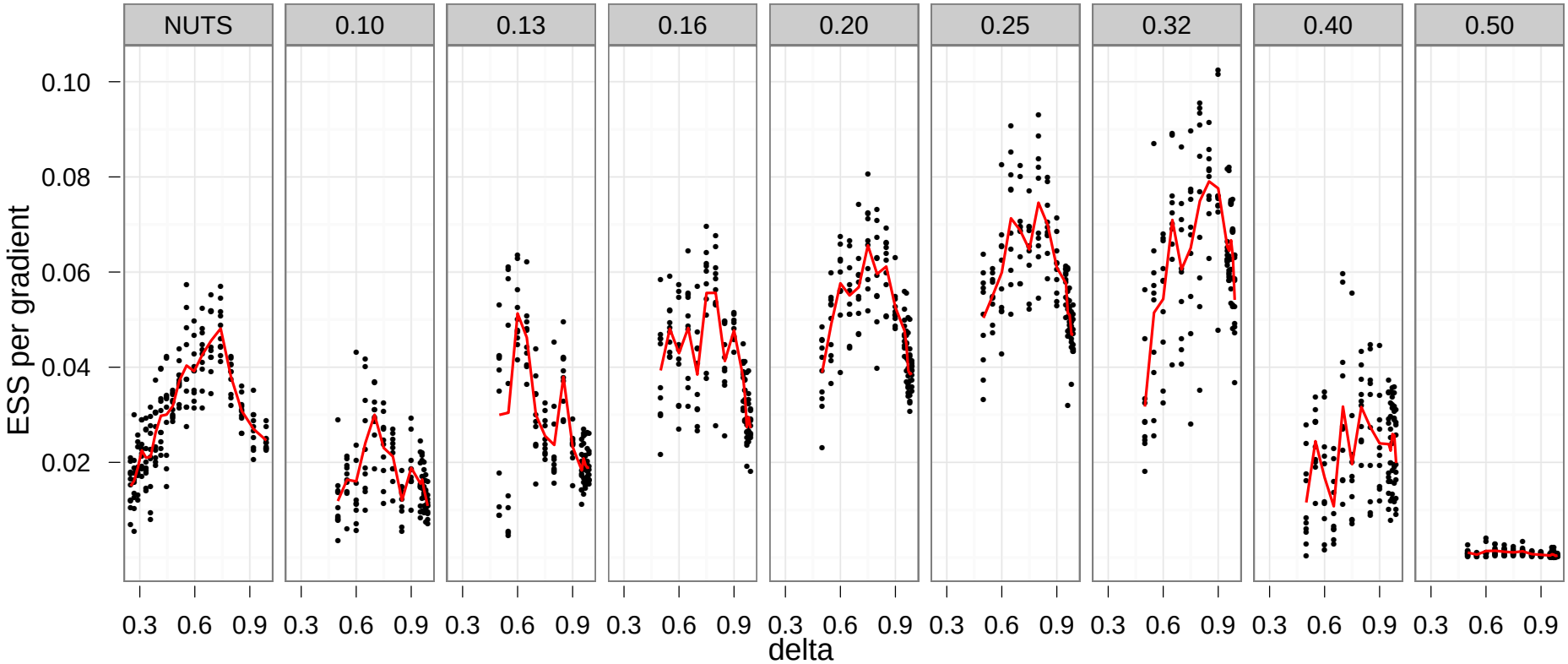
NUTS > Gibbs sampling



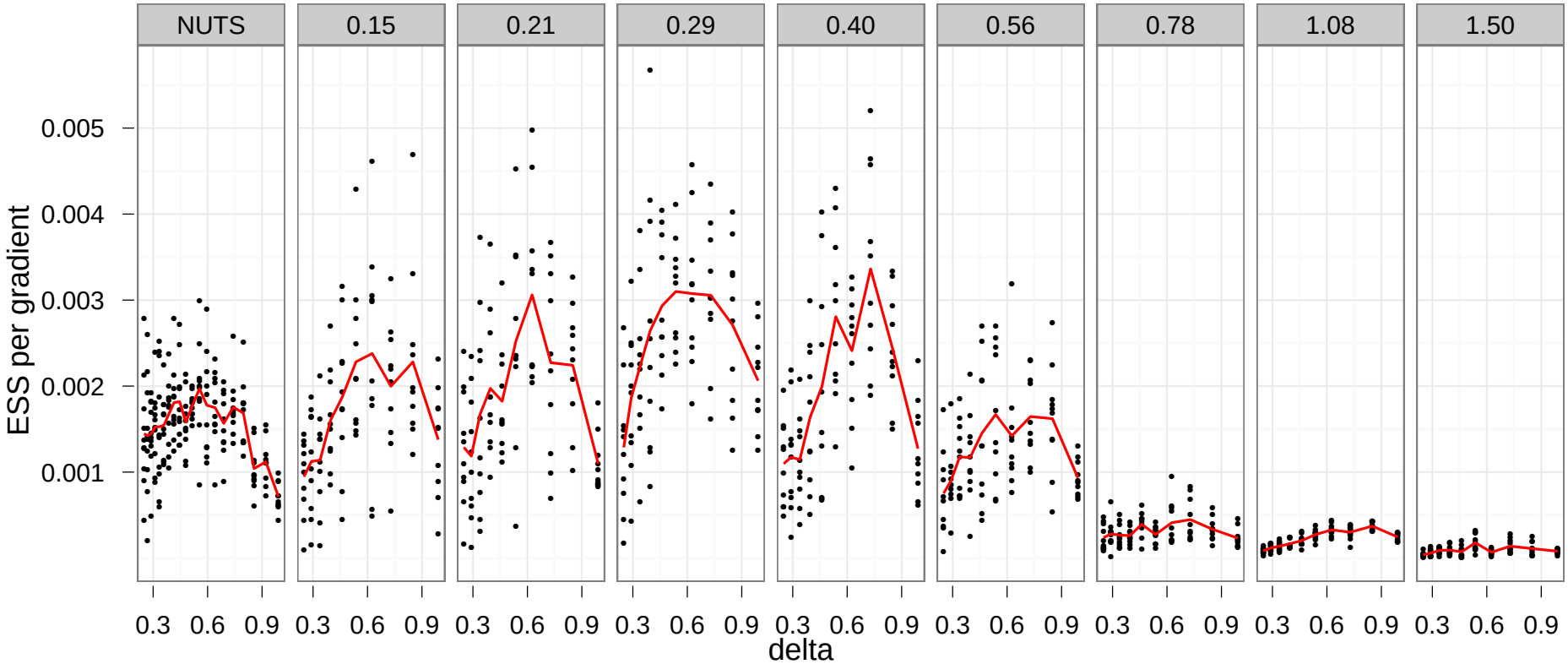
NUTS > random walk Metropolis



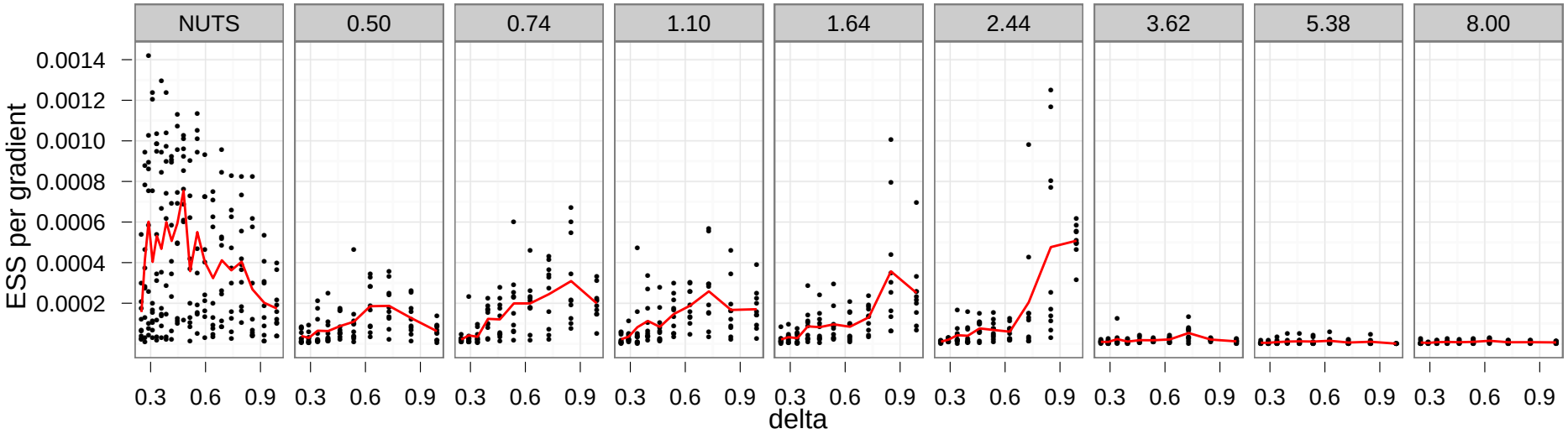
Optimizing HMC for a simple logistic regression



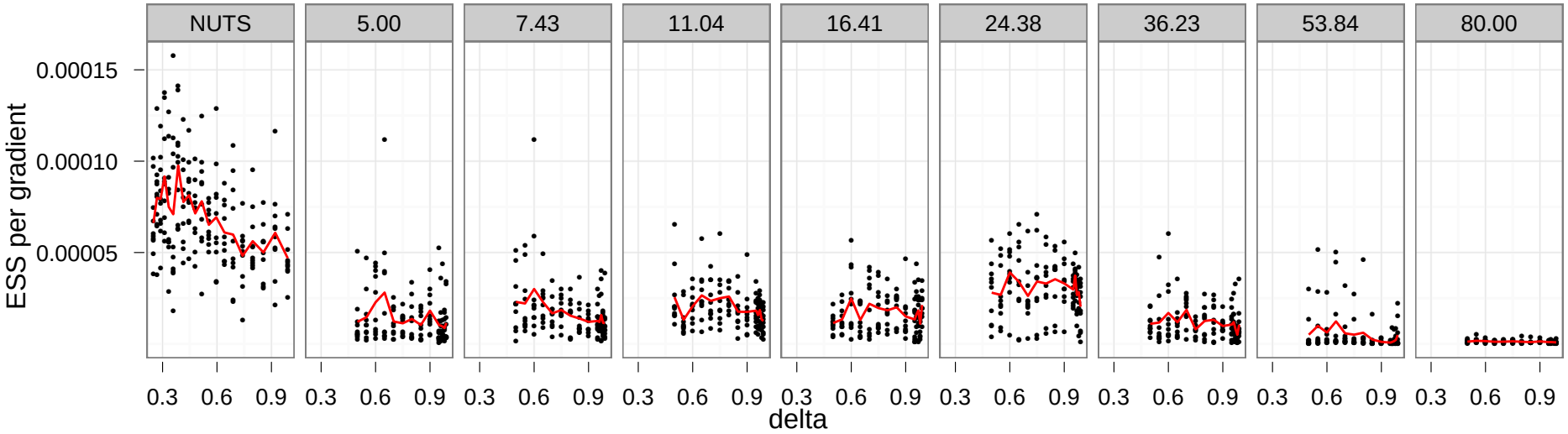
Optimizing HMC for a hierarchical logistic regression



Optimizing HMC for a stochastic volatility model



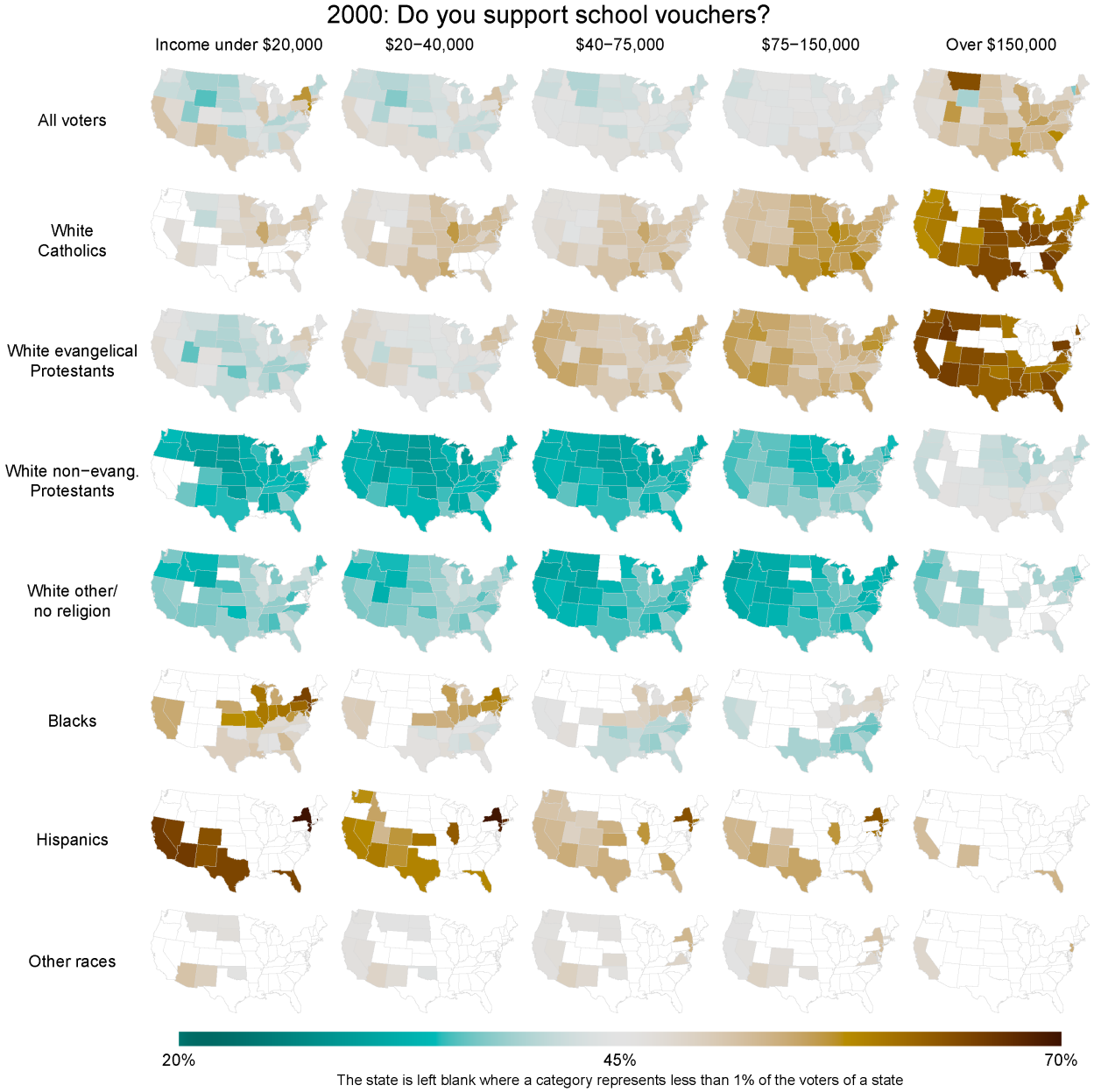
Optimizing HMC for a model with high correlations



Why did we develop Stan?

- WinBugs/OpenBugs/Jags don't work for big data or complex models
- Programming MCMC myself is slow and buggy
- Gibbs/Metropolis can be slow; Hamiltonian Monte Carlo requires gradients and more programming
- Goal: automate the programming steps so the user can focus on building, checking, understanding, and improving models
- "The user" is me!

Example: a deep-interaction model



Some applications

- Virtual database query (hierarchical models with deep interactions)
- Multilevel models with varying intercepts and slopes
- Reconstructing climate from tree rings (hierarchical spline models)

What's in Stan

- Hamiltonian Monte Carlo
- Autodif
- The no-U-turn sampler
- Adaptive tuning
- Gibbs sampler for discrete parameters
- StanBugs and StanC++ compilers

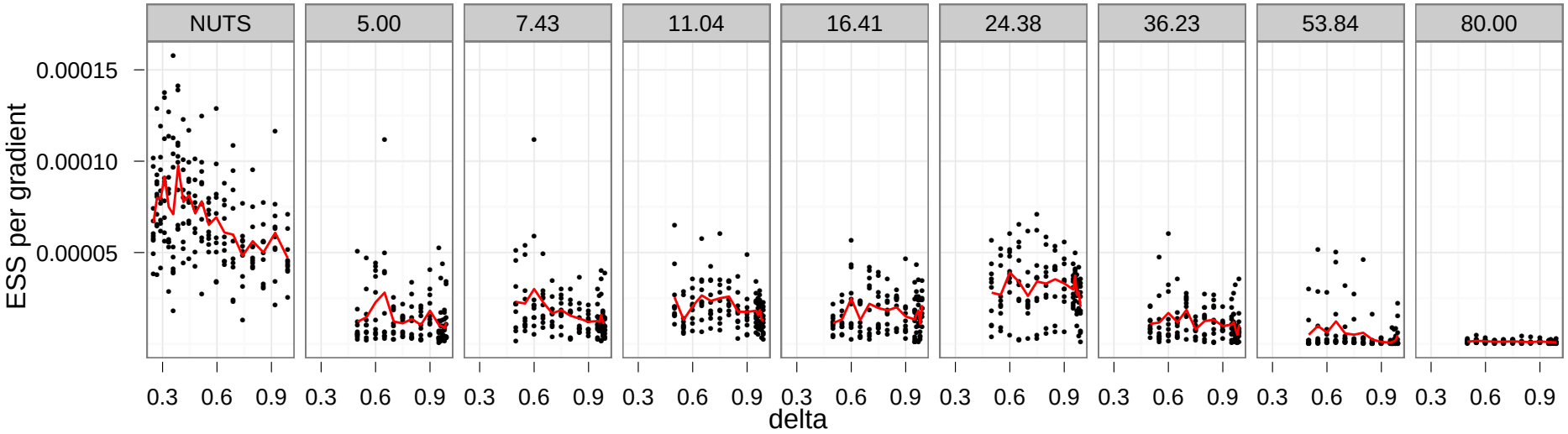
A simple hierarchical model in Stan

```
data {
    int(0,) J;                  // number of schools
    double y[J];                // estimated treatment effect (school j)
    double(0,) sigma_y[J];      // standard error of estimate (school j)
    double sigma_xi;            // prior scale (coefficient)
}
parameters {
    double mu;                  // intercept coefficient (for y)
    double xi;                  // slope coefficient (for y)
    double eta[J];              // predictor (school j)
    double(0,) sigma_eta;       // deviation of eta
}
model {
    sigma_eta ~ cauchy(0,1);
    for (j in 1:J)
        eta[j] ~ normal(0, sigma_eta);
    mu ~ normal(0,10);
    xi ~ normal(0, sigma_xi);
    for (j in 1:J)
        y[j] ~ normal(mu + xi * eta[j], sigma_y[j]);
}
```

Next steps

- Automatic model testing, cross-validation, and predictive checking
- Parallel implementation
- Adapting to do optimization and variational Bayes
- Scalability
  - Big data
  - Big models

Summary



- Automatic Bayesian inference as good as (or better than) "programming it yourself"
- Open architecture for further improvements
- `http://hackage.haskell.org/package/passage`
- `http://code.google.com/p/stan/`