Section 4.
How Stan Works

Bob Carpenter

Columbia University

Part I
What Stan Does

Full Bayes: No-U-Turn Sampler

- Adaptive **Hamiltonian Monte Carlo** (HMC)
  - **Potential Energy**: negative log posterior
  - **Kinetic Energy**: random standard normal per iteration
- Adaptation **during warmup**
  - step size adapted to target total acceptance rate
  - mass matrix estimated with regularization
- Adaptation **during sampling**
  - simulate forward and backward in time until U-turn
- **Slice sample** along path

(Hoffman and Gelman 2011, 2014)

Posterior Inference

- Generated quantities block for **inference**
  (predictions, decisions, and event probabilities)
- **Extractors** for draws in sample in RStan and PyStan
- Coda-like **posterior summary**
  - posterior mean w. MCMC std. error, std. dev., quantiles
  - split-$\hat{R}$ multi-chain convergence diagnostic (Gelman/Rubin)
  - multi-chain effective sample size estimation (FFT algorithm)
- Model comparison with **WAIC**
  - in-sample approximation to cross-validation

Penalized MLE

- Posterior **mode finding** via L-BFGS optimization
  (uses model gradient, efficiently approximates Hessian)
- **Disables Jacobians** for parameter inverse transforms
- **Standard errors** on unconstrained scale
  (estimated using curvature of penalized log likelihood function
- Models, data, initialization as in MCMC
- **Very Near Future**
  - Standard errors **on constrained scale**
    (sample unconstrained approximation and inverse transform)

"Black Box" Variational Inference

- **Black box** so can fit any Stan model
- Multivariate **normal approx to unconstrained** posterior
  - covariance: diagonal mean-field or full rank
  - not Laplace approx — around posterior mean, not mode
  - transformed back to constrained space (built-in Jacobians)
- Stochastic **gradient-descent** optimization
  - ELBO gradient estimated via Monte Carlo + autdiff
- Returns **approximate posterior** mean / covariance
- Returns **sample** transformed to constrained space

Posterior Analysis: Estimates

- For each parameter (and `lp__`)
  - Posterior mean
  - Posterior standard deviation
  - Posterior MCMC error esimate: sd/N_{eff}
  - Posterior quantiles
  - Number of effective samples
  - $\hat{R}$ convergence statistic

- . . . and much much more in ShinyStan

Stan as a Research Tool

- Stan can be used to **explore algorithms**
- Models transformed to **unconstrained support** on $\mathbb{R}^n$
- Once a model is compiled, have
  - **log probability, gradient** (soon: Hessian)
  - data I/O and parameter initialization
  - model provides variable names and dimensionalities
  - transforms to and from constrained representation (with or without Jacobian)

Part III
How Stan Works

Model: Read and Transform Data

- Only done once for optimization or sampling (per chain)
- Read data
  - read data variables from memory or file stream
  - validate data
- Generate transformed data
  - execute transformed data statements
  - validate variable constraints when done

Model: Log Density

- *Given* parameter values on unconstrained scale
- Builds expression graph for log density (start at 0)
- Inverse transform parameters to constrained scale
  - constraints involve non-linear transforms
  - e.g., positive constrained x to unconstrained $y = \log x$
- account for curvature in change of variables
  - e.g., unconstrained y to positive $x = \log^{-1}(y) = \exp(y)$
  - e.g., add log Jacobian determinant, $\log \left| \frac{d}{dy} \exp(y) \right| = y$
- Execute model block statements to increment log density

Model: Log Density Gradient

- Log density evaluation builds up expression graph
  - templated overloads of functions and operators
  - efficient arena-based memory management
- Compute gradient in backward pass on expression graph
  - propagate partial derivatives via chain rule
  - work backwards from final log density to parameters
  - dynamic programming for shared subexpressions
- Linear multiple of time to evalue log density

Model: Generated Quantities

- **Given** parameter values
- Once per iteration (not once per leapfrog step)
- May involve (pseudo) random-number generation
  - Executed generated quantity statements
  - Validate values satisfy constraints
- Typically used for
  - Event probability estimation
  - Predictive posterior estimation
- Efficient because evaluated with `double` types (no autodiff)

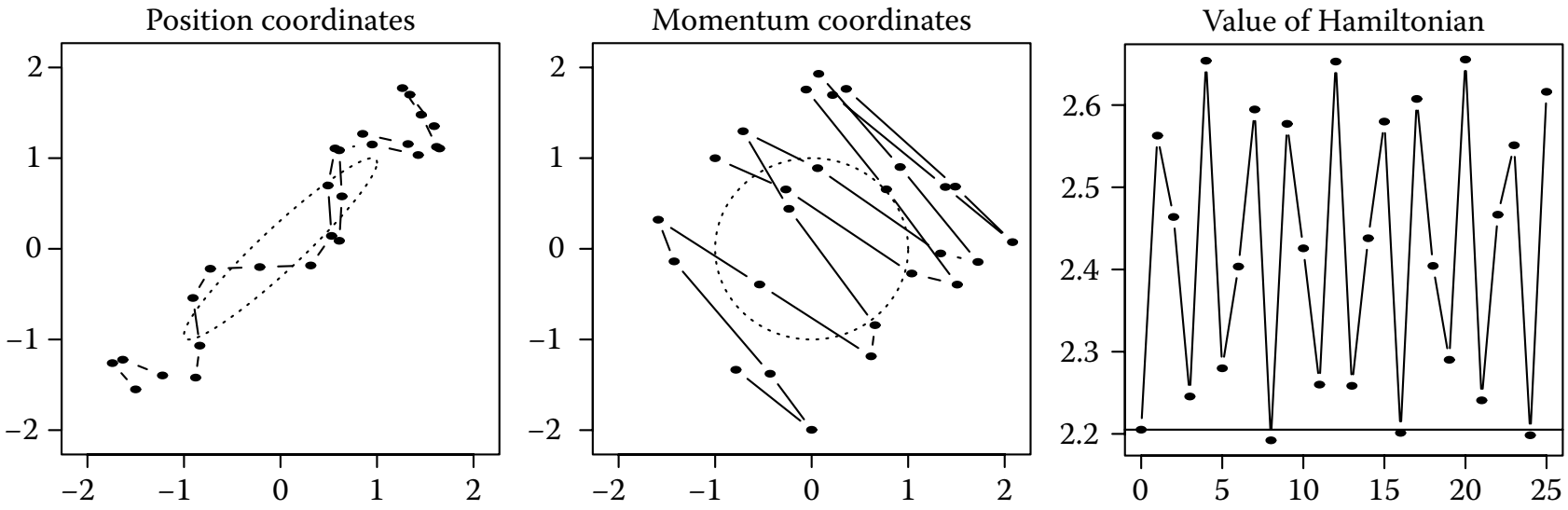
Optimize: L-BFGS

- Initialize unconstrained parameters and Hessian
  - Random values on unconstrained scale uniform in $(-2, 2)$
    - or user specified on constrained scale, transformed
  - Hessian approximation initialized to unit matrix
- While not converged
  - Move unconstrained parameters toward optimum based on Hessian approximation and step size (Newton step)
  - If diverged (arithmetic, support), reduce step size, continue
  - else if converged (parameter change, log density change, gradient value), return value
  - else update Hessian approx. based on calculated gradient

Sample: Hamiltonian Flow

- Generate random **kinetic energy**
  - random Normal$(0, 1)$ in each parameter
- Use negative log posterior as **potential energy**
- Hamiltonian is kinetic plus potential energy
- **Leapfrog Integration**: for *fixed* stepsize (time discretization), number of steps (total time), and mass matrix,
  - update momentum half-step based on potential (gradient)
  - update position full step based on momentum
  - update momentum half-step based on potential
- Numerical solution of Hamilton's first-order version of Newton's second-order diff-eqs of motion (force $=$ mass $\times$ acceleration)

Sample: Leapfrog Example



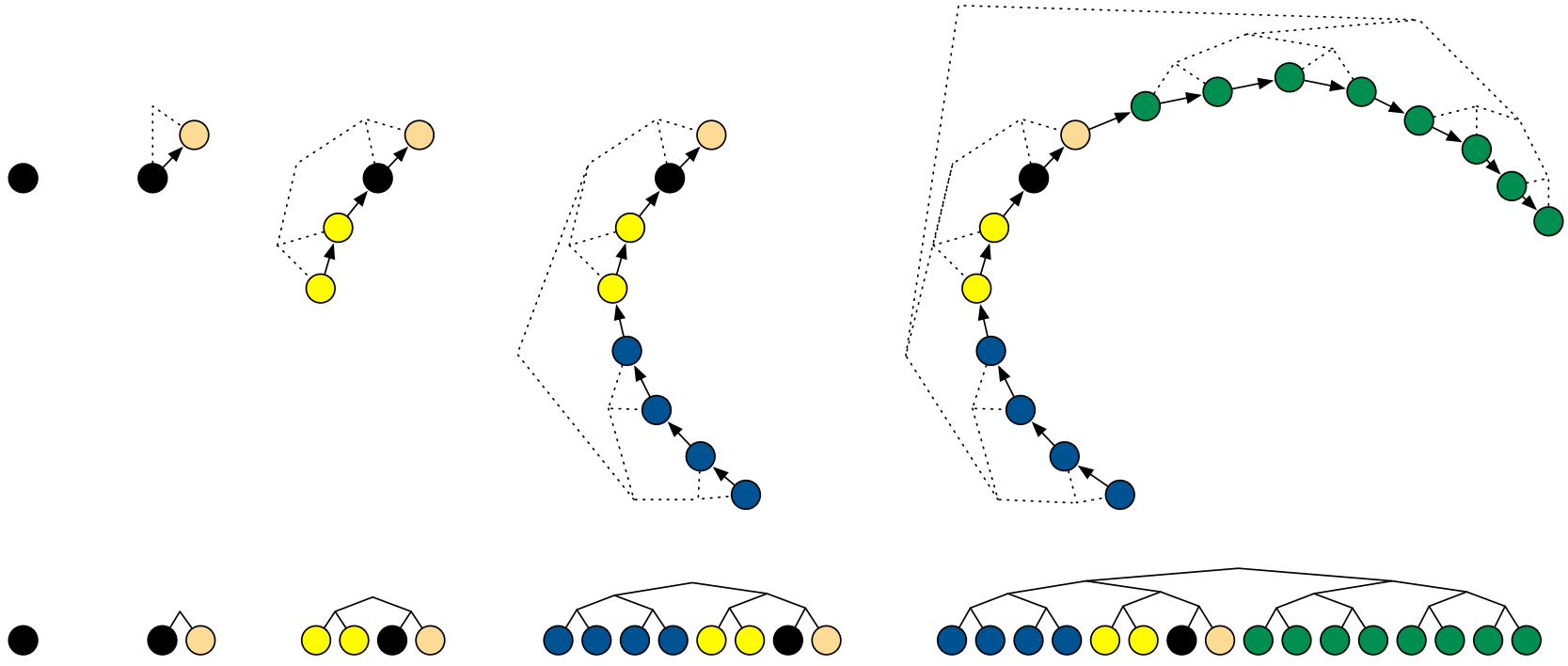
- Trajectory of 25 leapfrog steps for correlated 2D normal (ellipses at 1 sd from mean), stepsize of 0.25, initial state of $(-1, 1)$, and initial momentum of $(-1.5, -1.55)$.

Radford Neal (2013) MCMC using Hamiltonian Dynamics. In *Handbook of MCMC*. (free online at `http://www.mcmchandbook.net/index.html`)

Sample: No-U-Turn Sampler (NUTS)

- Adapts Hamiltonian simulation time
  - goal to maximize mixing, maintaining detailed balance
  - too short devolves to random walk
  - too long does extra work (i.e., orbits)
- For exponentially increasing number of steps up to max
  - Randomly choose to extend forward or backward in time
  - Move forward or backward in time number of steps
    - stop if any subtree (size 2, 4, 8, ...) makes U-turn
    - remove all current steps if subtree U-turns (not ends)
- Randomly select param with density above slice (or reject)

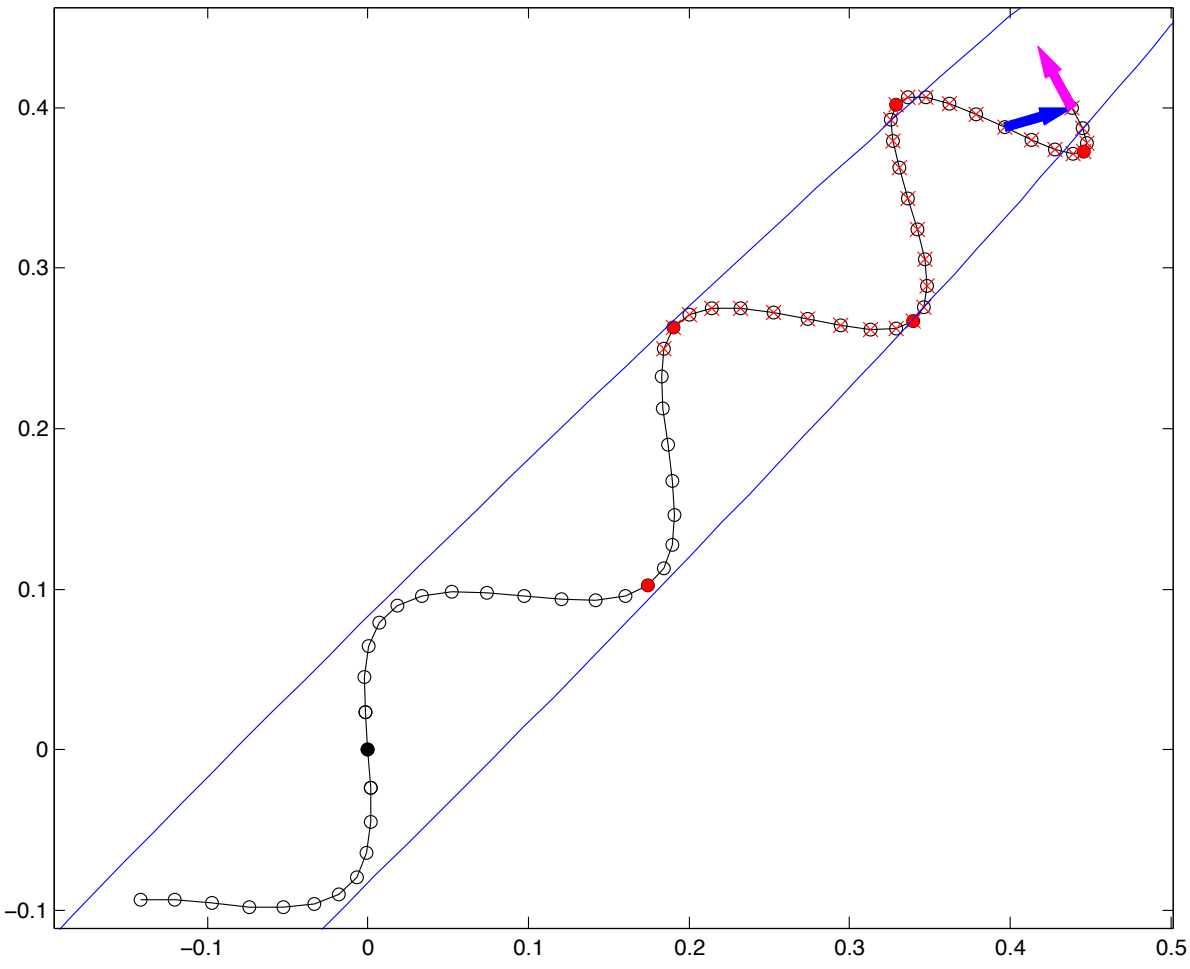
Sample: NUTS Binary Tree



- Example of repeated doubling building binary tree forward and backward in time until U-turn.

  Hoffman and Gelman. 2014. The No-U-Turn Sampler. *JMLR*. (free online at http://jmlr.org/papers/v15/hoffman14a.html)

Sample: NUTS U-Turn



- Example of trajectory from one iteration of NUTS.
- Blue ellipse is contour of 2D normal.
- Black circles are leapfrog steps.
- Solid red circles excluded below slice
- U-turn made with blue and magenta arrows
- Red crossed circles excluded for detailed balance

Sample: HMC/NUTS Warmup

- Estimate stepsize
  - too small requires too many leapfrog steps
  - too large induces numerical inaccuracy
  - need to balance
- Estimate mass matrix
  - Diagonal accounts for parameter scales
  - Dense optionally accounts for rotation

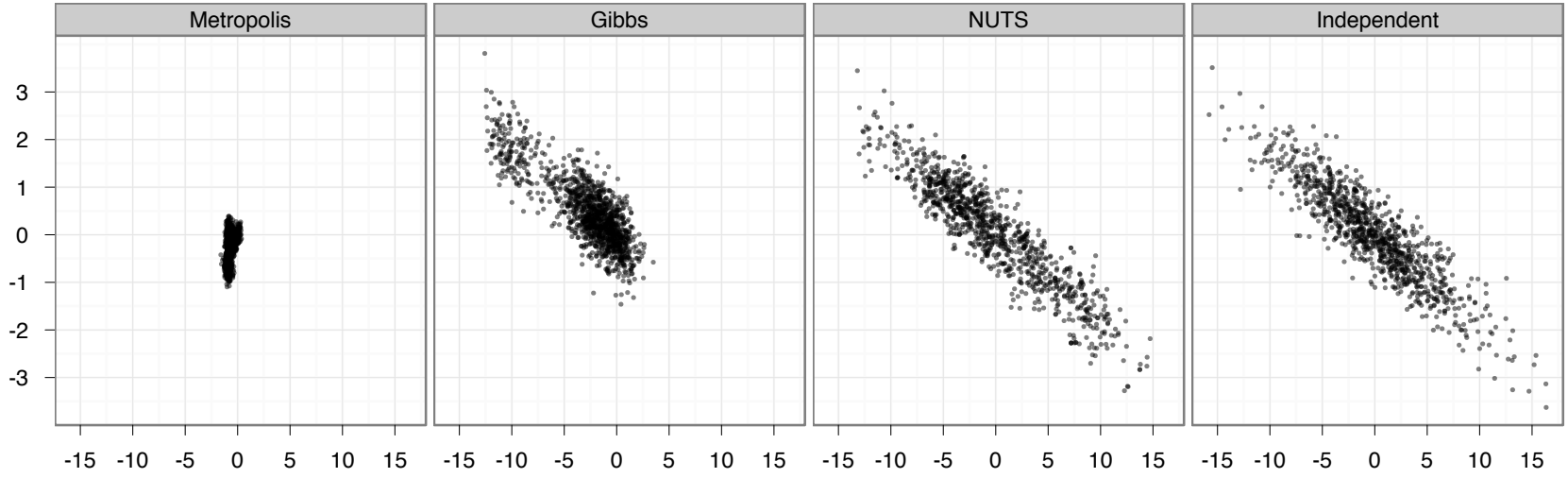
Sample: Warmup (cont.)

- Initialize unconstrained parameters as for optimization
- For exponentially increasing block sizes
  - for each iteration in block
    - generate random kinetic energy
    - simulate Hamiltonian flow (HMC fixed time, NUTS adapts)
    - choose next state (Metroplis for HMC, slice for NUTS)
  - update regularized point estimate of mass matrix
    - use parameter draws from current block
    - shrink diagonal toward unit; dense toward diagonal
  - tune stepsize (line search) for target acceptance rate

Sample: HMC/NUTS Sampling

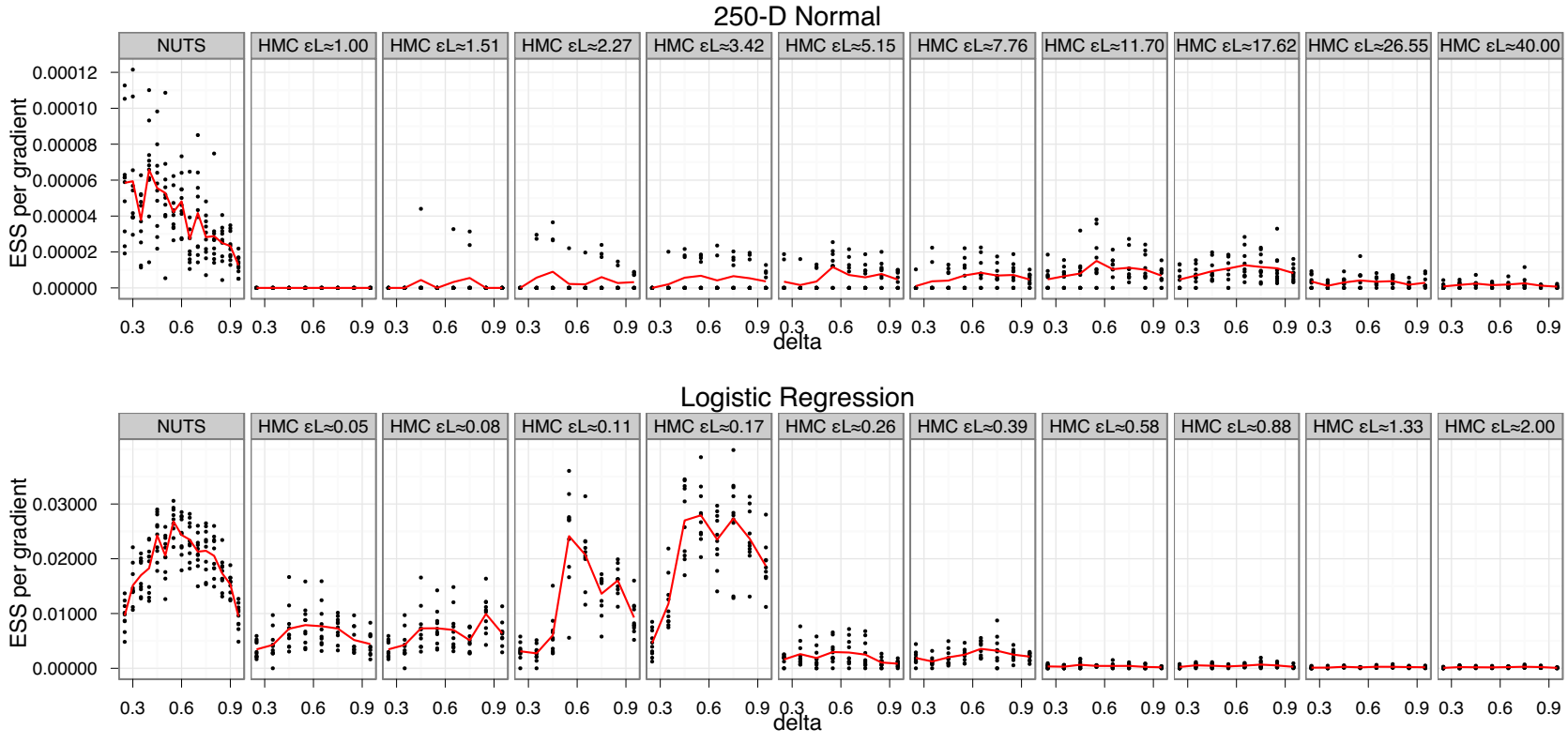
- Fix stepsize and and mass matrix
- For sampling iterations
  - generate random kinetic energy
  - simulate Hamiltonian flow
  - apply Metropolis accept/reject (HMC) or slice (NUTS)

NUTS vs. Gibbs and Metropolis



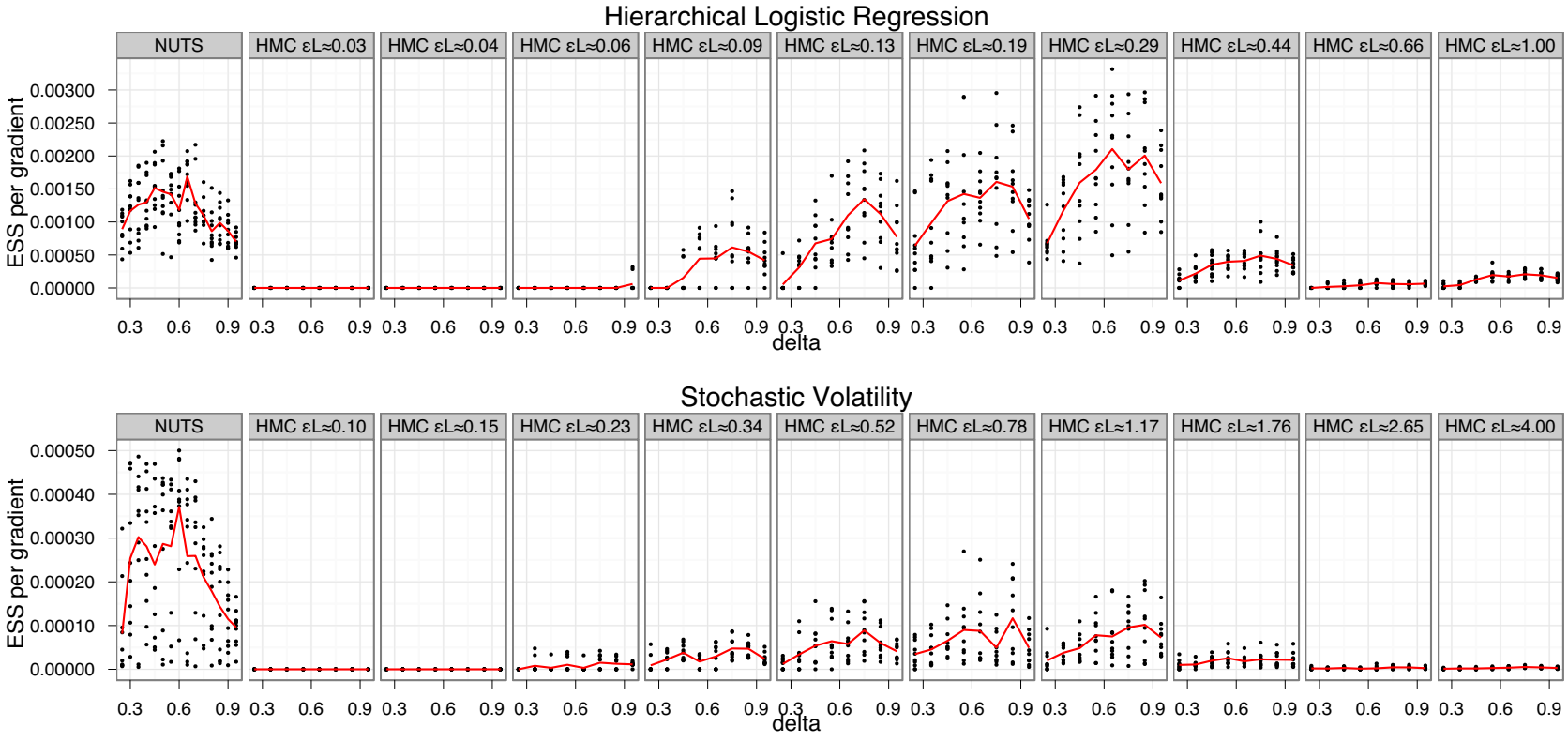
- Two dimensions of highly correlated 250-dim normal
- **1,000,000 draws** from Metropolis and Gibbs (thin to 1000)
- **1000 draws** from NUTS; 1000 independent draws

NUTS vs. Basic HMC



- 250-D normal and logistic regression models
- Vertical axis is effective sample size per sample (bigger better)
- Left) NUTS; Right) HMC with increasing $t = \epsilon L$

NUTS vs. Basic HMC II



- Hierarchical logistic regression and stochastic volatility
- Simulation time t is $\epsilon\ L$, step size (ϵ) times number of steps (L)
- NUTS can beat optimally tuned HMC (latter very expensive)

The End (Section 4)