Stan: A (Bayesian) Directed Graphical Model Compiler

Bob Carpenter

with Matt Hoffman, Ben Goodrich, Daniel Lee
Jiqiang Guo, Michael Malecki, and Andrew Gelman

Columbia University, Department of Statistics

NYC Machine Learning Meetup: January 2012

The Big Picture

- ***Application***: Fit rich Bayesian statistical models

- *Problem*: Gibbs too slow, Metropolis too problem-specific
- *Solution*: Hamiltonian Monte Carlo

- *Problem*: Interpreters too slow, won't scale
- *Solution*: Compilation

- *Problem*: Need gradients of log posterior for HMC
- *Solution*: Reverse-mode algorithmic differentation

The Big Picture (cont.)

- *Problem*: Existing algo-diff slow, limited, unextensible
- *Solution*: Our own algo-diff

- *Problem*: Algo-diff requires fully templated functions
- *Solution*: Our own density library, Eigen linear algebra

- *Problem*: Need unconstrained parameters for HMC
- *Solution*: Variable transforms w. Jacobian determinants

The Big Picture (cont.)

- *Problem*: Need ease of use of BUGS
- *Solution*: Support directed graphical model language

- *Problem*: Need to tune parameters for HMC
- *Solution*: Auto tuning, adaptation

- *Problem*: Efficient up-to-proportion calcs
- *Solution*: Density template metaprogramming

The Big Picture (conclusion)

- *Problem*: Poor error checking in model
- *Solution*: Static model typing, informative exceptions

- *Problem*: Poor boundary behavior
- *Solution*: Calculate limits (e.g. $\lim_{x \to 0} x \log x$)

- *Problem*: Restrictive licensing (e.g., closed, GPL, etc.)
- *Solution*: Open-source, BSD license

Bayesian Data Analysis

- "By Bayesian data analysis, we mean practical methods for making inferences from data using probability models for quantities we observe and about which we wish to learn."
- "The essential characteristic of Bayesian methods is their **explict use of probability for quantifying uncertainty** in inferences based on statistical analysis."

[Gelman et al., *Bayesian Data Analysis*, 2003]

The Process

1. Set up full probability model
   - for all observable & unobservable quantities
   - consistent w. problem knowledge & data collection
2. Condition on observed data
   - caclulate posterior probability of unobserved quantities conditional on observed quantities
3. Evaluate
   - model fit
   - implications of posterior

[*Ibid.*]

Basic Quantities

- Basic Quantities
  - y: observed data
  - $\tilde{y}$: unknown, potentially observable quantities
  - θ: parameters (and other unobserved quantities)
  - x: constants, predictors for conditional models
- Random models for things that could've been otherwise
  - All Stats: Model data y as random
  - Bayesian Stats: Model parameters θ as random

Basic Distributions

- Joint: $p(y, \theta)$
- Sampling / Likelihood: $p(y|\theta)$
- Prior: $p(\theta)$
- Posterior: $p(\theta|y)$
- Data Marginal: $p(y)$
- Posterior Predictive: $p(\tilde{y}|y)$

y observed data, θ parameters, $\tilde{y}$ predictions

Bayes's Rule: The Big Inversion

- Suppose the data y is fixed (i.e., observed). Then

$$
\begin{aligned}
p(\theta|y) &= \frac{p(y,\theta)}{p(y)} = \frac{p(y|\theta)\,p(\theta)}{p(y)} \\
&= \frac{p(y|\theta)\,p(\theta)}{\int p(y,\theta)\ d\theta} \\
&= \frac{p(y|\theta)\,p(\theta)}{\int p(y|\theta)\,p(\theta)\ d\theta} \\
&\propto p(y|\theta)\,p(\theta) = p(y,\theta)
\end{aligned}
$$

- Posterior proportional to likelihood times prior (i.e., joint)

Directed Graphical Models

- Directed acyclic graph
- Nodes are data or parameters
- Edges represent dependencies
- Generative model
  - Start at top
  - Sample each node conditioned on parents
- Determines joint probability

BUGS Declarative Model Language

- Declarative specification of directed graphical models
- Variables are (potentially) random quantities
- Full set of arithmetic, functional, and matrix expressions
- Sampling: `y ~ Foo(theta);`
- Assignment: `y <- bar(x);`
- For Loops: `for (n in 1:N) { ... }`
- Constants modeled if on left of sampling
  - usually modeled: outcomes
  - not usually modeled: predictors, data sizes

Normal (Sampling)

```
for (n in 1:N)
   y[n] ~ normal(0,1);
```

- Sampling: data (N), params (y)

Normal (Full)

```
mu ~ normal(0,10);
sigma_sq ~ inv_gamma(1,1);
for (n in 1:N)
   y[n] ~ normal(mu,sigma_sq);
```

- Estimation: data (y, N), params (μ, σ)
- Sampling: data (μ, σ^2, N), params (y)

Naive Bayes

- ```
  pi ~ Dirichlet(alpha);
  for (d in 1:D) {
     z[d] ~ Discrete(pi);
     for (n in 1:N[d])
         w[d,n] ~ Discrete(phi[z[d]]);
  }
  for (k i 1:K)
    phi[k] ~ Dirichlet(beta);
  ```
- Estimation: data $(w, z, D, N, \alpha, \beta)$, params (π, ϕ)
- Prediction: data $(w, D, N, \pi, \phi, \alpha, \beta)$, params (z)
- Clustering: data (w, D, N, α, β), params (z, ϕ, π)
  ```

Supervision: Full, Semi-, and Un-

- How variable is used
  - Supervised: declared as data
  - Unsupervised: declared as parameter
  - Semi-supervised: partly data, partly parameter
- Full probability model does not change
- E.g., Semi-supervised naive Bayes
  - partly estimation, known categories $z[n]$ supervised
  - partly clustering, unknown $z[n]$ unsupervised

Latent Dirichlet Allocation

```
for (d in 1:D) {
    theta[d] ~ Dirichlet(alpha);
    for (n in 1:N[d]) {
        z[d,n] ~ Discrete(theta[d]);
        w[d,n] ~ Discrete(phi[z[d,n]]);
    }
}
for (k i 1:K)
   phi[k] ~ Dirichlet(beta);
```

- Clustering: data $(w, \alpha, \beta, D, K, N)$, params (θ, ϕ, z)

(Blei et al. 2003)

Logistic Regression

- ```
  for (k in 1:K)
    beta[k] ~ cauchy(0,2.5);
  for (n in 1:N)
    y[n] ~ bern(inv_logit(transpose(beta) * x[n]))
  ```
- Estimate: data (y, x, K, N), params (β)
- Predict: data (β, x, K, N), params (y)
- Pluggable prior
  - Cauchy, fat tails (allows concentration around mean)
  - Normal (L2), strong due to relatively thin tails
  - Laplace (L1), sparse only with point estimates
  ```

BUGS to Joint Probability

- BUGS Model

```
mu ~ normal(0,10);
for (n in 1:N)
    y[n] ~ normal(mu,1);
```

- Joint Probability

$$
\begin{aligned}
p(\mu, y) &= \text{Normal}(\mu | 0, 10) \\
&\quad \times \prod_{n=1}^{N} \text{Normal}(y_n | 0, 1)
\end{aligned}
$$

Monte Carlo Methods

- For integrals that are impossible to solve analytically
- But for which sampling and evaluation is tractable
- Compute plug-in estimates of statistics based on randomly generated variates (e.g., means, variances, quantiles/intervals, comparisons)
- Accuracy with M (independent) samples proportional to

$$\frac{1}{\sqrt{M}}$$

e.g., 100 times more samples per decimal place!

(Metropolis and Ulam 1949)

Monte Carlo Example

- Posterior expectation of θ:

$$\mathbb{E}[\theta|y] = \int \theta \; p(\theta|y) \; d\theta.$$

- Bayesian estimate minimizing expected square error:

$$\hat{\theta} = \arg\min_{\theta'} \mathbb{E}[(\theta - \theta')^2|y] = \mathbb{E}[\theta|y]$$

- Generate samples $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$ drawn from $p(\theta|y)$

- Monte Carlo Estimator plugs in average for expectation:

$$\mathbb{E}[\theta|y] \approx \frac{1}{M} \sum_{m=1}^{M} \theta^{(m)}$$

Monte Carlo Example II

- Bayesian alternative to frequentist hypothesis testing
- Use probability to summarize results
- Bayesian comparison: probability $\theta_1 > \theta_2$ given data y?

$$\begin{aligned}
\text{Pr}[\theta_1 > \theta_2 | y] &= \int\int \mathbb{I}(\theta_1 > \theta_2)\ p(\theta_1|y)\ p(\theta_2|y)\ d\theta_2\ d\theta_2 \\
&\approx \frac{1}{M}\sum_{m=1}^{M} \mathbb{I}(\theta_1^{(m)} > \theta_2^{(m)})
\end{aligned}$$

- (Bayesian hierarchical model "adjusts" for multiple comparisons)

Markov Chain Monte Carlo

- When sampling independently from $p(\theta|y)$ impossible
- $\theta^{(m)}$ drawn via a Markov chain $p(\theta^{(m)}|y, \theta^{(m-1)})$
- Require MCMC marginal $p(\theta^{(m)}|y)$ equal to true posterior marginal
- Leads to auto-correlation in samples $\theta^{(1)}, \ldots, \theta^{(m)}$
- Effective sample size M_{eff} divides out auto-correlation (must be estimated)
- Estimation accuracy proportional to $1/\sqrt{M_{\text{eff}}}$

Gibbs Sampling

- Samples a parameter given data and other parameters
- Requires conditional posterior $p(\theta_n|y, \theta_{-n})$
- Conditional posterior easy in directed graphical model
- Requires general unidimensional sampler for non-conjugacy
  - JAGS uses slice sampler
  - BUGS uses adaptive rejection sampler
- Conditional sampling and general unidimensional sampler can both lead to slow convergence and mixing

(Geman and Geman 1984)

Metropolis-Hastings Sampling

- Proposes new point by changing all parameters randomly
- Computes accept probability of new point based on ratio of new to old log probability (and proposal density)
- Only requires evaluation of $p(\theta|y)$
- Requires good proposal mechanism to be effective
- Acceptance requires small changes in log probability
- But small step sizes lead to random walks and slow convergence and mixing

(Metropolis et al. 1953; Hastings 1970)

Hamiltonian Monte Carlo

- Converges faster and explores posterior faster when posterior is complex
- Function of interest is log posterior (up to proportion)

$$\log p(\theta|y) \propto \log p(y|\theta) + \log p(\theta)$$

- HMC exploits its gradient

$$\begin{aligned} g &= \nabla_\theta \log p(\theta|y) \\ &= \left(\frac{d}{d\theta_1} \log p(\theta|y), \ldots \frac{d}{d\theta_K} \log p(\theta|y) \right) \end{aligned}$$

(Duane et al. 1987; Neal 1994)

HMC's Physical Analogy

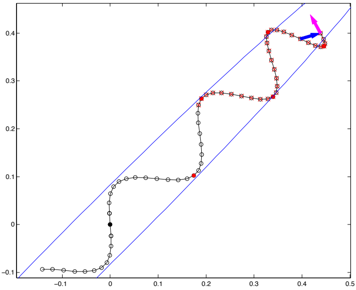
1. Negative log posterior $-\log p(\theta|y)$ is potential energy
2. Start point mass at current parameter position θ
3. Add random kinetic energy (momentum)
4. Simulate trajectory of the point mass over time t
5. Return new parameter position*

* In practice, Metropolis adjust for imprecision in trajectory simulation due to discretizing Hamiltonian dynamics

A (Simple) HMC Update

1. $m \sim \text{Norm}(0, \mathrm{I})$ $\qquad H = \frac{m^\top m}{2} - \log p(\theta|y)$
2. $\theta^{\text{new}} = \theta$
3. repeat L times:
   (a) $m = m - \frac{1}{2}\,\epsilon\,g(\theta^{\text{new}})$
   (b) $\theta^{\text{new}} = \theta^{\text{new}} + \epsilon\,m$
   (c) $m = m - \frac{1}{2}\,\epsilon\,g(\theta^{\text{new}})$
4. $H^{\text{new}} = \frac{m^\top m}{2} - \log p(\theta^{\text{new}}|y)$
5. if Unif$(0, 1) < \exp(H - H^{\text{new}})$, then θ^{new}, else θ

HMC Example Trajectory



- Blue ellipse is contour of target distribution
- Initial position at black solid circle
- Arrows indicate a U-turn in momentum

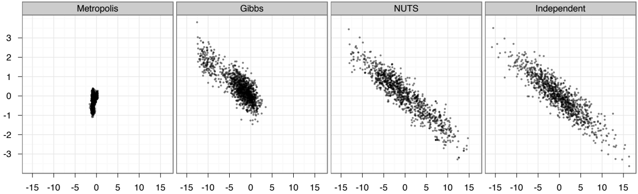
No-U-Turn Sampler (NUTS)

- HMC highly sensitive to tuning parameters
  - discretization step size ϵ
  - discretization number of steps L
- NUTS sets ϵ during burn-in by stochastic optimization (Nesterov-style dual averaging)
- NUTS chooses L online per-sample using no-U-turn idea:

  keep simulating as long as position gets further away from initial position
- Number of steps just a bit of bookkeeping on top of HMC

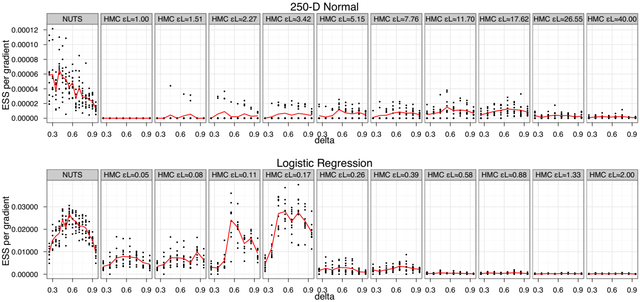
(Hoffman and Gelman, 2011)

NUTS vs. Gibbs and Metropolis



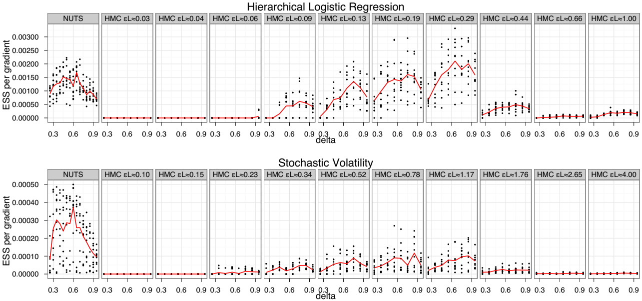
- Two dimensions of highly correlated 250-dim distribution
- 1M samples from Metropolis, 1M from Gibbs (thinned to 1K)
- 1K samples from NUTS, 1K independent draws

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)

Stan C++ Library

- Beta available from Google code; 1.0 release soon
- C++, with heavy use of templates
- HMC and NUTS continuous samplers (Metropolis in v2)
- Gibbs (bounded) and slice (unbounded) for discrete
- Model (probability, gradient) extends abstract base class
- Automatic gradient w. algorithmic differentiation
- Fully templated densities, cumulative densities, transforms
- (New) BSD licensed

Stan — Graphical Model Compiler

- Compiler for directed graphical model language ($\sim$ BUGS)
- Generates C++ model class
- Compile model from command line
- Run model from command line
  - random seeds
  - multiple chains (useful for convergence monitoring)
  - parameter initialization
  - HMC parameters and NUTS hyperparameters
  - CSV sample output

Stan Integration with R

- Effective sample size calcs (variogram-based)
- Convergence monitoring (split $\hat{R}$)
- Plots of posteriors
- Statistical summaries and comparisons

- Python, MATLAB to come

Extensions to BUGS Language

- User-defined functions (JAGS, Stan)
- Data Transformations (JAGS, Stan)
- General matrix solvers (Stan)
- Local variables (Stan)

Variable Typing

- Classes of variables (Stan):
  data, transformed data, parameters, transformed parameters, derived quantities, local
- Static variable typing (Stan):
  Unconstrained: int, double, vector, row vector, matrix, list
  Constrained: (half) bounded, simplex, ordered, correlation matrix, covariance matrix

Algorithmic Differentiation

- Forward-mode fast for single derivative
- Reverse-mode uses dynamic programming to evaluate gradient in time proportional to function eval (independently of number of dimensions)
- Functional Behavior
  - Write function templating out scalar variables
  - Instantiate template with algo-dif variables
  - Call function
  - Fetch gradient

Algorithmic Differentiation (cont.)

- Override all built-in scalar ops (operators, lib functions)
  - Calculate values and partial derivates w.r.t. all arguments
  - Object-oriented design supports user extensions
- Algo-dif uses templated variables to build expression tree
- Nodes of tree represent intermediate expressions
- Nodes topologically sorted on a stack
- Custom arena-based memory management (thread localizable at 20% performance hit)
- Propagate partial derivatives down along edges

Algorithmic Differentiation (cont.)

- Non-negligible cost compared to well-coded derivatives
- Space per operation: 24 bytes + 8 bytes/argument
  - especially problematic for iterative algorithms
- Time per operation: about 4 times slower than basic function evaluation
  - Mostly due to partial derivative virtual function
- Can partially evaluate some expressions and vectorize repeated operations with shared suboperations

Variable Transforms

- HMC works best with unconstrained variables
- (Technically possible to bounce off boundaries)
- Automatically transform variables from unconstrained to constrained
- Add log of the absolute determinant of the Jacobian of the transform
- Jacobian is the matrix of output variable gradients with respect to each input variable

Example Transforms

- Lower bound 0: $x \mapsto \exp(x)$
- Constrained $(0, 1)$: $x \mapsto$ logit$^{-1}(x)$
- Simplex: $x \mapsto$ softmax(x) (or hyperspherical + Weierstrss); $K - 1$ degrees of freedom
- Ordered: $(x_1, x_2) \mapsto (x_1, x_1 + \exp(x_2))$
- Correlation Matrix: Lewandowski et al. C-vines transform; $\binom{K}{2}$ degrees of freedom
- Covariance Matrix: Scale correlation matrix; $K + \binom{K}{2}$ degrees of freedom

Calculating Prop-to Log Densities

- Only need calculations to proportion
- Drop additive terms that only have constants
- Consider log of normal distribution:

$$\log \textsf{Normal}(y|\mu, \sigma) = -\log \sqrt{2\,\pi} - 0.5 \log \sigma + \frac{(y-\mu)^2}{2\sigma^2}$$

  - Drop first term always if only need proportion
  - Drop second term if σ is constant
  - Drop third term if all arguments constant

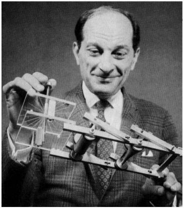
Templates for Proportionality

- Type traits to statically test fixed values

- ```
  template <typename T_out,
            typename T_loc,
            typename T_scale>
  typename promote_args<T_out,T_loc,T_scale>::type
  normal_log(T_out y, T_loc mu, T_scale sigma) {
      ...
      if (is_variable<T_scale>::value)
          result += 0.5 * log(sigma);
      ...
  }
  ```
  ```

Stan's Namesake

- Stanislaw Ulam (1909–1984)
- Co-inventor of Monte Carlo method (and hydrogen bomb)



- Ulam holding the Fermiac, Enrico Fermi's physical Monte Carlo simulator for random neutron diffusion

The End