Stan: a program for Bayesian data analysis with complex models

Andrew Gelman, Bob Carpenter, and Matt Hoffman, Jiqiang Guo, Ben Goodrich, and Daniel Lee
Department of Statistics, Columbia University, New York
and others

12 Oct 2012
Stan overview

- Fit open-ended Bayesian models
- Specify log posterior density in C++
- Code a distribution once, then use it everywhere
- Hamiltonian No-U-Turn sampler
- Autodiff
- Runs from R; postprocessing
Sparse data

Exposure of 72 ppm

Exposure of 144 ppm
Validation using predictive simulations

Graph showing the relationship between time (min) and alveolar/inhaled concentration ratio.
Public opinion: Health care reform

Should federal gov't spend more money on health care for the uninsured (2004 survey)?

<table>
<thead>
<tr>
<th>Income</th>
<th>18-29</th>
<th>30-44</th>
<th>45-64</th>
<th>65+</th>
</tr>
</thead>
<tbody>
<tr>
<td>Under $20,000</td>
<td><img src="image1" alt="Map" /></td>
<td><img src="image2" alt="Map" /></td>
<td><img src="image3" alt="Map" /></td>
<td><img src="image4" alt="Map" /></td>
</tr>
<tr>
<td>$20-40,000</td>
<td><img src="image5" alt="Map" /></td>
<td><img src="image6" alt="Map" /></td>
<td><img src="image7" alt="Map" /></td>
<td><img src="image8" alt="Map" /></td>
</tr>
<tr>
<td>$40-75,000</td>
<td><img src="image9" alt="Map" /></td>
<td><img src="image10" alt="Map" /></td>
<td><img src="image11" alt="Map" /></td>
<td><img src="image12" alt="Map" /></td>
</tr>
<tr>
<td>$75-150,000</td>
<td><img src="image13" alt="Map" /></td>
<td><img src="image14" alt="Map" /></td>
<td><img src="image15" alt="Map" /></td>
<td><img src="image16" alt="Map" /></td>
</tr>
<tr>
<td>Over $150,000</td>
<td><img src="image17" alt="Map" /></td>
<td><img src="image18" alt="Map" /></td>
<td><img src="image19" alt="Map" /></td>
<td><img src="image20" alt="Map" /></td>
</tr>
</tbody>
</table>

U.S. avg. is 73%

The state is left blank where a category represents less than 1% of the voters of a state.
Public opinion: School vouchers

2000: Do you support school vouchers?

- Income under $20,000
- $20–40,000
- $40–75,000
- $75–150,000
- Over $150,000

All voters
White Catholics
White evangelical Protestants
White non-evang. Protestants
White other/no religion
Blacks
Hispanics
Other races

Stan for Bayesian data analysis

Gelman, Carpenter, Hoffman, Guo, Goodrich, Lee, ...
Hierarchical time series model
Open-ended
Tree rings

![Graph showing temperature trends over time.](Image)

- Axes: Temperature (Celsius) vs. Year
- Years: 1500 to 2000
- Trends: Wave-like fluctuations with a peak around 1750.
Speed dating

- Each person meets 10–20 “dates”
- Rate each date on Attractiveness, Sincerity, Intelligence, Ambition, Fun to be with, Shared interests
- Outcomes
  - Do you want to see this person again? (Yes/No)
  - How much do you like this person (1–10)
- How important are each of the 6 attributes?
- Logistic or linear regression
- Hierarchical model: coefficients vary by person
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
Figure 2: Example of a trajectory generated during one iteration of NUTS. The blue ellipse is a contour of the target distribution, the black open circles are the positions $\theta$ traced out by the leapfrog integrator and associated with elements of the set of visited states $B$, the black solid circle is the starting position, the red solid circles are positions associated with states that must be excluded from the set $C$ of possible next samples because their joint probability is below the slice variable $u$, and the positions with a red “x” through them correspond to states that must be excluded from $C$ to satisfy detailed balance. The blue arrow is the vector from the positions associated with the leftmost to the rightmost leaf nodes in the rightmost height-3 subtree, and the magenta arrow is the (normalized) momentum vector at the final state in the trajectory. The doubling process stops here, since the blue and magenta arrows make an angle of more than 90 degrees. The crossed-out nodes with a red “x” are in the right half-tree, and must be ignored when choosing the next sample.

Having a more complicated, the analogous algorithm that eliminates the slice variable seems empirically to be slightly less efficient than the algorithm presented in this paper.
Solving problems

- **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 differentiation

- **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
"One practical impediment to the use of Hamiltonian Monte Carlo is the need to select suitable values for the leapfrog stepsize, $\epsilon$, and the number of leapfrog steps $L$. Tuning HMC will usually require preliminary runs with trial values for $\epsilon$ and $L$. Unfortunately, preliminary runs can be misleading..."
Created by Matt Hoffman
Run the HMC steps until they start to turn around (bend with an angle $> 180^\circ$)
Computationally efficient
Requires no tuning of $\#$ steps
Complications to preserve detailed balance
Figure 2: Example of a trajectory generated during one iteration of NUTS. The blue ellipse is a contour of the target distribution, the black open circles are the positions $\theta$ traced out by the leapfrog integrator and associated with elements of the set of visited states $B$, the black solid circle is the starting position, the red solid circles are positions associated with states that must be excluded from the set $C$ of possible next samples because their joint probability is below the slice variable $u$, and the positions with a red “x” through them correspond to states that must be excluded from $C$ to satisfy detailed balance. The blue arrow is the vector from the positions associated with the leftmost to the rightmost leaf nodes in the rightmost height-3 subtree, and the magenta arrow is the (normalized) momentum vector at the final state in the trajectory. The doubling process stops here, since the blue and magenta arrows make an angle of more than 90 degrees. The crossed-out nodes with a red “x” are in the right half-tree, and must be ignored when choosing the next sample.

Being more complicated, the analogous algorithm that eliminates the slice variable seems empirically to be slightly less efficient than the algorithm presented in this paper.
### NUTS vs. Gibbs and Metropolis

#### 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.

#### 4.4 Comparing the Efficiency of HMC and NUTS

Figure 6 compares the efficiency of HMC (with various simulation lengths $\lambda \approx L$) and NUTS (which chooses simulation lengths automatically). The x-axis in each plot is the target $\delta$ used by the dual averaging algorithm from section 3.2 to automatically tune the step size $\epsilon$. The y-axis is the effective sample size (ESS) generated by each sampler, normalized by the number of gradient evaluations used in generating the samples. HMC's best performance seems to occur around $\delta = 0$, suggesting that this is indeed a reasonable default value for a variety of problems. NUTS's best performance seems to occur around $\delta = 0.6$, but does not seem to depend strongly on $\delta$ within the range $\delta \in [0.45, 0.65]$. $\delta = 0.6$ therefore seems like a reasonable default value for NUTS.

On the two logistic regression problems NUTS is able to produce effectively independent samples about as efficiently as HMC can. On the multivariate normal and stochastic volatility problems, NUTS with $\delta = 0.6$ outperforms HMC's best ESS by about a factor of three.

As expected, HMC's performance degrades if an inappropriate simulation length is chosen. Across the four target distributions we tested, the best simulation lengths $\lambda$ for HMC varied by about a factor of 100, with the longest optimal $\lambda$ being 17.62 (for the multivariate normal) and the shortest optimal $\lambda$ being 0.17 (for the simple logistic regression). In practice, finding a good simulation length for HMC will usually require some number of preliminary runs. The results in Figure 6 suggest that NUTS can generate samples at least as efficiently as HMC, even discounting the cost of any preliminary runs needed to tune HMC's simulation length.

- Two dimensions of highly correlated 250-dim distribution
- 1M samples from Metropolis, 1M from Gibbs (thin to 1K)
- 1K samples from NUTS, 1K independent draws
250-D normal and logistic regression models

- Vertical axis shows effective #sims (big is good)
- (Left) NUTS; (Right) HMC with increasing $t = \epsilon L$
Figure 6: Effective sample size (ESS) as a function of $\delta$ and (for HMC) simulation length $L$ for the multivariate normal, logistic regression, hierarchical logistic regression, and stochastic volatility models. Each point shows the ESS divided by the number of gradient evaluations for a separate experiment; lines denote the average of the points' y-values for a particular $\delta$. Leftmost plots are NUTS's performance, each other plot shows HMC's performance for a different setting of $L$.

The trajectory length (measured in number of states visited) grows as the acceptance rate target $\delta$ grows, which is to be expected since a higher $\delta$ will lead to a smaller step size $\epsilon$, which in turn will mean that more leapfrog steps are necessary before the trajectory doubles back on itself and satisfies equation 9.

- Hierarchical logistic regression and stochastic volatility
- Simulation time is step size $\epsilon$ times #steps $L$
- NUTS can beat optimally tuned HMC
Solving more problems in Stan

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

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

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

- **Problem**: Limited error checking, recovery
  - **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
A simple hierarchical model in Stan

data {
  int<lower=0> J; // number of schools
  real y[J]; // estimated treatment effects
  real<lower=0> sigma[J]; // s.e. of effect estimates
}

parameters {
  real mu;
  real<lower=0> tau;
  real eta[J];
}

transformed parameters {
  real theta[J];
  for (j in 1:J)
    theta[j] <- mu + tau * eta[j];
}

model {
  eta ~ normal(0, 1);
  y ~ normal(theta, sigma);
}
Example: the “Kumaraswamy distribution”

\[ p(\theta|a, b) = a b \theta^{\theta-1}(1 - \theta^a)(b - 1) \text{ for } a, b > 0 \text{ and } \theta \in (0, 1) \]

model {
  // Put priors on a and b here if you want

  // Put in the rest of your model

  // Kumaraswamy log-likelihood
  lp__ <- lp__ + N*(log(a)+log(b))+(a-1)*sum_log_theta;
  for (i in 1:N)
    lp__ <- lp__ + (b-1)*log1m(pow(theta[i],a));
}
Check that it worked

R code:

N <- 1000
a <- 3
b <- 2
theta <- rbeta(N,1,b)^(1/a)
Kumaraswamy <- stan (file="Kumaraswamy.stan", data=list(N=N,x=x))
print (Kumaraswamy)

Result:

Inference for Stan model: Kumaraswamy.
4 chains: each with iter=2000; warmup=1000; thin=1; 2000 saved.

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>sd</th>
<th>2.5%</th>
<th>25%</th>
<th>50%</th>
<th>75%</th>
<th>97.5%</th>
<th>n_eff</th>
<th>Rhat</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>2.9</td>
<td>0.1</td>
<td>2.7</td>
<td>2.9</td>
<td>2.9</td>
<td>3.0</td>
<td>3.1</td>
<td>766</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1.9</td>
<td>0.1</td>
<td>1.8</td>
<td>1.9</td>
<td>1.9</td>
<td>2.0</td>
<td>2.1</td>
<td>787</td>
<td>1</td>
</tr>
<tr>
<td>lp__</td>
<td>282.4</td>
<td>1.0</td>
<td>279.7</td>
<td>282.1</td>
<td>282.7</td>
<td>283.1</td>
<td>283.4</td>
<td>683</td>
<td>1</td>
</tr>
</tbody>
</table>
Scalability

- Big data
- Big models
What do we mean by Artificial Intelligence?

Bayesian Artificial Intelligence

Kevin B. Korb
Ann E. Nicholson
A possible model of Bayesian data analysis

Computer does step 2 (Bayesian inference); homunculus does step 1 (model building) and step 3 (model checking)
- Automatic Bayesian inference as good as (or better than) “programming it yourself”
- Open architecture, active user and developer communities (mc-stan.org)
- Goals:
  - Fit more models
  - Run faster
  - Tools for Bayesian data analysis