Big Data need Big Model

Model Complexity vs. Sample Size

Big data in practice
Stan: A platform for Bayesian inference

Andrew Gelman, Bob Carpenter, Matt Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, Allen Riddell, ...

Department of Statistics, Columbia University, New York (and other places)

10 Nov 2014
Eminem - Stan (Short Version) ft. Dido - YouTube
www.youtube.com/watch?v=aSLZFdqwh7E

Artists: Eminem, Dido
Album: No Angel
Eminem - Stan (Short Version) ft. Dido - YouTube
www.youtube.com/watch?v=aSLZFdqwh7E

Artists: Eminem, Dido
Album: No Angel
Released: 1999

Stan (song) - Wikipedia, the free encyclopedia
en.wikipedia.org/wiki/Stan_(song)
"Stan" is the third single from the The Marshall Mathers LP, recorded in 1999 by American rapper Eminem and featuring British singer Dido. It peaked at number ...
Thank You - Rock City - Robert Browning - Murder ballad

Stan: Project Home Page
mc-stan.org/
Stan modeling language and C++ library for Bayesian inference. NUTS adaptive HMC (MCMC) sampling, automatic differentiation, R, shell interfaces. Gelman.

Urban Dictionary: stan
www.urbandictionary.com/define.php?term=stan
Based on the central character in the Eminem song of the same name, a "stan" is an overzealous maniacal fan for any celebrity or athlete.

Gelman Carpenter Hoffman Lee Goodrich Betancourt... Stan: A platform for Bayesian inference
Stan is a probabilistic programming language implementing full Bayesian statistical inference with

- MCMC sampling (NUTS, HMC)

and penalized maximum likelihood estimation with

- Optimization (BFGS)

Stan is coded in C++ and runs on all major platforms (Linux, Mac, Windows).

Stan is freedom-respecting, open-source software (new BSD core, GPLv3 interfaces).

Interfaces

Download and getting started instructions, organized by interface:

- **RStan v2.5.0** (R)
- **PyStan v2.5.0** (Python)
- **CmdStan v2.5.0** (shell, command-line terminal)
- **MatlabStan** (MATLAB)
- **Stan.jl** (Julia)
Ordered probit

data {
    int<lower=2> K;
    int<lower=0> N;
    int<lower=1> D;
    int<lower=1,upper=K> y[N];
    row_vector[D] x[N]; }

parameters {
    vector[D] beta;
    ordered[K-1] c; }

model {
    vector[K] theta;
    for (n in 1:N) {
        real eta;
        eta <- x[n] * beta;
        theta[1] <- 1 - Phi(eta - c[1]);
        for (k in 2:(K-1))
            theta[k] <- Phi(eta - c[k-1]) - Phi(eta - c[k]);
        theta[K] <- Phi(eta - c[K-1]);
        y[n] ~ categorical(theta);
    }
}
data {
  ...
  real x_meas[N]; // measurement of x
  real<lower=0> tau; // measurement noise
}
parameters {
  real x[N]; // unknown true value
  real mu_x; // prior location
  real sigma_x; // prior scale
  ...
}
model {
  x ~ normal(mu_x, sigma_x); // prior
  x_meas ~ normal(x, tau); // measurement model
  y ~ normal(alpha + beta * x, sigma);
  ...
}
Stan overview

- Fit open-ended Bayesian models
Stan overview

- Fit open-ended Bayesian models
- Specify log posterior density in C++
Fit open-ended Bayesian models
Specify log posterior density in C++
Code a distribution once, then use it everywhere
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
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
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, Python, Matlab, Julia; postprocessing
People

Stan: A platform for Bayesian inference
People

- Stan core (15)
People

- Stan core (15)
- Research collaborators (30)
- Developers (100)
People

- Stan core (15)
- Research collaborators (30)
- Developers (100)
- User community (1000)
People

- Stan core (15)
- Research collaborators (30)
- Developers (100)
- User community (1000)
- Users (10000)
Funding

- National Science Foundation
- Institute for Education Sciences
- Department of Energy
- Novartis
- YouGov
Roles of Stan

- Bayesian inference for unsophisticated users (alternative to Stata, Bugs, etc.)
- Bayesian inference for sophisticated users (alternative to programming it yourself)
- Fast and scalable gradient computation
- Environment for developing new algorithms

Stan: A platform for Bayesian inference
Roles of Stan

- Bayesian inference for unsophisticated users (alternative to Stata, Bugs, etc.)
Roles of Stan

- Bayesian inference for unsophisticated users (alternative to Stata, Bugs, etc.)
- Bayesian inference for sophisticated users (alternative to programming it yourself)
Roles of Stan

- Bayesian inference for unsophisticated users (alternative to Stata, Bugs, etc.)
- Bayesian inference for sophisticated users (alternative to programming it yourself)
- Fast and scalable gradient computation
Roles of Stan

- Bayesian inference for unsophisticated users (alternative to Stata, Bugs, etc.)
- Bayesian inference for sophisticated users (alternative to programming it yourself)
- Fast and scalable gradient computation
- Environment for developing new algorithms
If the election were held today, who would you vote for?

- Barack Obama
- Mitt Romney
- Other
- Not sure
Stan: A platform for Bayesian inference
“This week, the New York Times and CBS News published a story using, in part, information from a non-probability, opt-in survey sparking concern among many in the polling community. In general, these methods have little grounding in theory and the results can vary widely based on the particular method used.”
— Michael Link,
President, American Association for Public Opinion Research
Michael W. Link is Chief Methodologist for Research Methods at The Nielsen Company. His background includes a broad base of experience in survey research, having worked in academia (University of South Carolina, 1993-1999), not-for-profit research (RTI International, 1999-2004), government (Centers for Disease Control and Prevention, 2004-2007), and the private sector (Nielsen, 2007-present). He received his PhD in Social Science from the University of South Carolina. Michael’s research centers around developing and applying methodologies for confronting some of the most pressing issues facing survey research, including: techniques for improving survey participation and data quality (use of address-based sampling and call screening technologies), methodological issues involving use of multiple modes in data collection (email, CATI, field, mobile, meters), and obtaining participation from hard-to-survey populations (younger, isolated, racial and ethnic groups). His numerous research articles have appeared in *Public Opinion Quarterly* and other leading scientific journals.

An AAPOR member since 1993, Michael served as AAPOR Conference Chair in back-to-back years (2008 & 2010), a member of both the Cell Phone and Online task forces, an instructor for an AAPOR short-course, a reviewer for the student paper competition on several occasions, and a regular reviewer for *Public Opinion Quarterly*. He is a member of SAPOR, serving from 2007 to 2011 as President, Conference Chair, and Student Paper Competition Organizer and also a member of the AAPOR Awards Committee.

In 2011 he, along with several research colleagues, received AAPOR’s Warren J. Mitofsky Award for their work on address-based sampling designs. His current research focuses on understanding how new technologies, such as mobile and social platforms, can be used as vehicles for measuring and understanding attitudes and behaviors. He will be teaching a short course on “The Role of New Technologies in Augmenting, or Replacing Traditional Surveys” at the 2012 AAPOR conference.
Nielsen feels the heat of competition as it flubs its ratings of news broadcasts, putting ABC ahead of NBC

In spite of the goof, its global president took time to slam rival Rentrak, which collects different kind of data from viewers

NEW YORK DAILY NEWS / Sunday, October 19, 2014, 2:00 AM

MEDIA

TV Ratings by Nielsen Had Errors for Months

By BILL CARTER and EMILY STEEL OCT. 10, 2014

Nielsen, the television research firm, acknowledged on Friday that it had been reporting inaccurate ratings for the broadcast networks for the last seven months, a mistake that raises questions about the company’s increasingly criticized system for measuring TV audiences.
Xbox estimates, adjusting for demographics

Nate Silver, *New York Times*, 6 Oct: “Mr. Romney has not only improved his own standing but also taken voters away from Mr. Obama’s column.”
Xbox estimates, adjusting for demographics and partisanship

![Graph showing two-party Obama support over time from Sep. 24 to Nov. 05.](image)
Jimmy Carter Republicans and George W. Bush Democrats

Non-Monotonic Age Curve in 2008
The Formative Years

Age-Specific Weights (w)

-0.01
0.00
0.01
0.02
0.03
0.04

10 20 30 40 50 60 70

Posterior Mean
50% C.l.
95% C.l.
Lots of other applications

Astronomy, ecology, linguistics, epidemiology, soil science, ...
Steps of Bayesian data analysis

- Model building
- Inference
- Model checking
- Model understanding and improvement
Steps of Bayesian data analysis

- Model building
Steps of Bayesian data analysis

- Model building
- Inference
Steps of Bayesian data analysis

- Model building
- Inference
- Model checking
Steps of Bayesian data analysis

- Model building
- Inference
- Model checking
- Model understanding and improvement
Background on Bayesian computation

- Point estimates and standard errors
Background on Bayesian computation

- Point estimates and standard errors
- Hierarchical models
Background on Bayesian computation

- Point estimates and standard errors
- Hierarchical models
- Posterior simulation
Background on Bayesian computation

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

- **Problem:** Gibbs too slow, Metropolis too problem-specific
Solving problems

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

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

- **Problem**: Interpreters too slow, won’t scale

---

Gelman Carpenter Hoffman Lee Goodrich Betancourt...

Stan: A platform for Bayesian inference
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
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
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

Stan: A platform for Bayesian inference
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
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
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
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 …”
The No U-Turn Sampler

- 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.
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.
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 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.
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.
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 $\delta=0.65$, 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 $\delta \in [0.45, 0.65]$. Therefore $\delta=0.6$ 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.
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 $\delta = 0.65$, 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.
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 $\lambda \approx \lambda L$) 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)
NUTS vs. Gibbs and Metropolis

Figure 7: Samples generated by random-walk Metropolis, Gibbs sampling, and NUTS.

- 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

4.4 Comparing the Efficiency of HMC and NUTS

Figure 6 compares the efficiency of HMC (with various simulation lengths $\lambda \approx 2L$) 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 $\delta = 0.65$, 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.
NUTS vs. Basic HMC

- 250-D normal and logistic regression models
- Vertical axis shows effective #sims (big is good)
- (Left) NUTS; (Right) HMC with increasing $t = \epsilon L$
NUTS vs. Basic HMC II

- 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
Solving more problems in Stan

- **Problem:** Need ease of use of BUGS
Solving more problems in Stan

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

- **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
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
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

*Stan: A platform for Bayesian inference*
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
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
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
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
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. $\lim_{x \to 0} x \log x$)
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. \( \lim_{x\to0} x \log x \))
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.)
- **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
New stuff: Differential equation models

Simple harmonic oscillator:

\[
\frac{dz_1}{dt} = -z_2 \\
\frac{dz_2}{dt} = -z_1 - \theta z_2
\]

with observations \((y_1, y_2)_t, t = 1, \ldots, T:\)

\[
y_{1t} \sim N(z_1(t), \sigma^2_1) \\
y_{2t} \sim N(z_2(t), \sigma^2_2)
\]

Given data \((y_1, y_2)_t, t = 1, \ldots, T,\)

estimate initial state \((y_1, y_2)_{t=0}\) and parameter \(\theta\)
functions {
    real[] sho(real t, real[] y, real[] theta, real[] x_r, int[] x_i) {
        real dydt[2];
        dydt[1] <- y[2];
        return dydt;
    }
}
data {
    int<lower=1> T;
    real y[T,2];
    real t0;
    real ts[T];
}
transformed data {
    real x_r[0];
    int x_i[0];
}
parameters {
    real y0[2];
    vector<lower=0>[2] sigma;
    real theta[1];
}

model {
    real z[T,2];
    sigma ~ cauchy(0,2.5);
    theta ~ normal(0,1);
    y0 ~ normal(0,1);
    z <- integrate_ode(sho, y0, t0, ts, theta, x_r, x_i);
    for (t in 1:T)
        y[t] ~ normal(z[t], sigma);
}
Run RStan with data simulated from
$\theta = 0.15$, $y_0 = (1, 0)$, and $\sigma = 0.1$:

Inference for Stan model: sho.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

<table>
<thead>
<tr>
<th></th>
<th>mean</th>
<th>se_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>y0[1]</td>
<td>1.05</td>
<td>0.00</td>
<td>0.09</td>
<td>0.87</td>
<td>0.98</td>
<td>1.05</td>
<td>1.10</td>
<td>1.23</td>
<td>1172</td>
<td>1</td>
</tr>
<tr>
<td>y0[2]</td>
<td>-0.06</td>
<td>0.00</td>
<td>0.06</td>
<td>-0.18</td>
<td>-0.10</td>
<td>-0.06</td>
<td>-0.02</td>
<td>0.06</td>
<td>1524</td>
<td>1</td>
</tr>
<tr>
<td>sigma[1]</td>
<td>0.14</td>
<td>0.00</td>
<td>0.04</td>
<td>0.08</td>
<td>0.11</td>
<td>0.13</td>
<td>0.16</td>
<td>0.25</td>
<td>1354</td>
<td>1</td>
</tr>
<tr>
<td>sigma[2]</td>
<td>0.11</td>
<td>0.00</td>
<td>0.03</td>
<td>0.06</td>
<td>0.08</td>
<td>0.10</td>
<td>0.12</td>
<td>0.18</td>
<td>1697</td>
<td>1</td>
</tr>
<tr>
<td>theta[1]</td>
<td>0.15</td>
<td>0.00</td>
<td>0.04</td>
<td>0.08</td>
<td>0.13</td>
<td>0.15</td>
<td>0.17</td>
<td>0.22</td>
<td>1112</td>
<td>1</td>
</tr>
<tr>
<td>lp__</td>
<td>28.97</td>
<td>0.06</td>
<td>1.80</td>
<td>24.55</td>
<td>27.95</td>
<td>29.37</td>
<td>30.29</td>
<td>31.35</td>
<td>992</td>
<td>1</td>
</tr>
</tbody>
</table>
Big Data, Big Model, Scalable Computing

![Graph showing the relationship between total ratings and seconds per sample for different numbers of items. The graph includes three lines: one for 10 Items, another for 100 Items, and the third for 1000 Items. The x-axis represents total ratings on a logarithmic scale, ranging from $10^2$ to $10^6$. The y-axis represents seconds per sample, ranging from $10^{-3}$ to $10^0$. The graph illustrates that as the number of items increases, the time required to sample from the posterior distribution also increases.](image_url)
Thinking about scalability

Hierarchical item response model:

<table>
<thead>
<tr>
<th># items</th>
<th># raters</th>
<th># groups</th>
<th># data</th>
<th>time</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2,000</td>
<td>100</td>
<td>40,000</td>
<td>:02m</td>
<td>16MB</td>
</tr>
<tr>
<td>40</td>
<td>8,000</td>
<td>200</td>
<td>320,000</td>
<td>:16m</td>
<td>92MB</td>
</tr>
<tr>
<td>80</td>
<td>32,000</td>
<td>400</td>
<td>2,560,000</td>
<td>4h:10m</td>
<td>580MB</td>
</tr>
</tbody>
</table>

Also, Stan generated 4x effective sample size per iteration

Gelman Carpenter Hoffman Lee Goodrich Betancourt ...

Stan: A platform for Bayesian inference
Thinking about scalability

Hierarchical item response model:

<table>
<thead>
<tr>
<th># items</th>
<th># raters</th>
<th># groups</th>
<th># data</th>
<th>Stan time</th>
<th>Stan memory</th>
<th>JAGS time</th>
<th>JAGS memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2,000</td>
<td>100</td>
<td>40,000</td>
<td>:02m</td>
<td>16MB</td>
<td>:03m</td>
<td>220MB</td>
</tr>
<tr>
<td>40</td>
<td>8,000</td>
<td>200</td>
<td>320,000</td>
<td>:16m</td>
<td>92MB</td>
<td>:40m</td>
<td>1400MB</td>
</tr>
<tr>
<td>80</td>
<td>32,000</td>
<td>400</td>
<td>2,560,000</td>
<td>4h:10m</td>
<td>580MB</td>
<td>:??m</td>
<td>?MB</td>
</tr>
</tbody>
</table>

Also, Stan generated 4x effective sample size per iteration.
Thinking about scalability

Hierarchical item response model:

<table>
<thead>
<tr>
<th># items</th>
<th># raters</th>
<th># groups</th>
<th># data</th>
<th>Stan time</th>
<th>memory</th>
<th>JAGS time</th>
<th>memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>2,000</td>
<td>100</td>
<td>40,000</td>
<td>:02m</td>
<td>16MB</td>
<td>:03m</td>
<td>220MB</td>
</tr>
<tr>
<td>40</td>
<td>8,000</td>
<td>200</td>
<td>320,000</td>
<td>:16m</td>
<td>92MB</td>
<td>:40m</td>
<td>1400MB</td>
</tr>
<tr>
<td>80</td>
<td>32,000</td>
<td>400</td>
<td>2,560,000</td>
<td>4h:10m</td>
<td>580MB</td>
<td>:??m</td>
<td>?MB</td>
</tr>
</tbody>
</table>

Also, Stan generated 4x effective sample size per iteration
Future work
Future work

- Programming
Future work

- Programming
  - Faster gradients and higher-order derivatives
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions
- Statistical algorithms
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions
- Statistical algorithms
  - Riemannian Hamiltonian Monte Carlo
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions
- Statistical algorithms
  - Riemannian Hamiltonian Monte Carlo
  - (Penalized) mle
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions

- Statistical algorithms
  - Riemannian Hamiltonian Monte Carlo
  - (Penalized) mle
  - (Penalized) marginal mle
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions

- Statistical algorithms
  - Riemannian Hamiltonian Monte Carlo
  - (Penalized) mle
  - (Penalized) marginal mle
  - Black-box variational Bayes
Future work

- Programming
  - Faster gradients and higher-order derivatives
  - Functions
- Statistical algorithms
  - Riemannian Hamiltonian Monte Carlo
  - (Penalized) mle
  - (Penalized) marginal mle
  - Black-box variational Bayes
  - Data partitioning and expectation propagation