Stan: An under-the-hood tour

Marcus A Brubaker
University of Toronto / York University

UNIVERSITY OF TORONTO

LASSONDE SCHOOL OF ENGINEERING

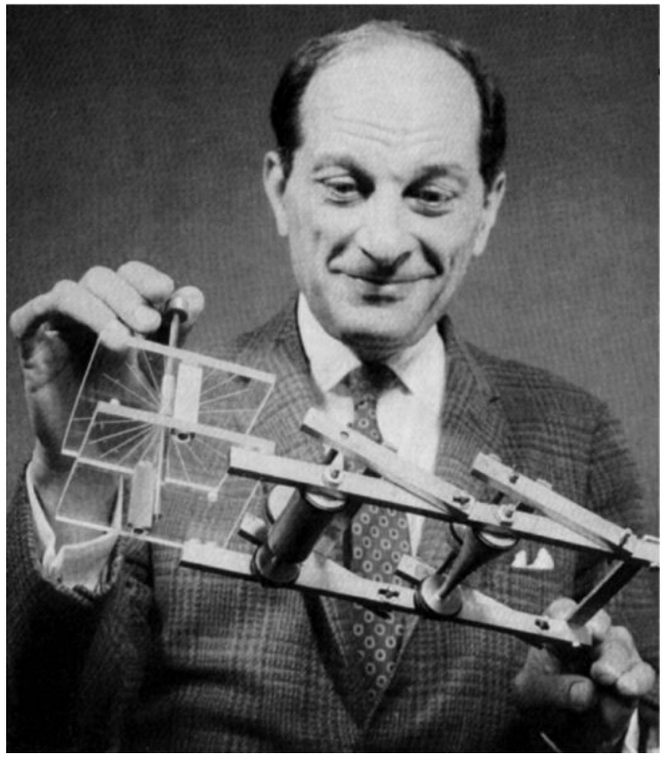
YORK UNIVERSITÉ UNIVERSITY

Part 1: Stan Basics

Stan Basics

Who is Stan?

- Stan is not an acronym



Stanislaw Ulam
(1909 - 1984)

Stan Basics

Who is Stan?

- Started at Columbia University by Andrew Gelman, Bob Carpenter, Matt Hoffman, Daniel Lee, Ben Goodrich, et al around 2011
- I joined in 2012 as one of the first “external” contributors and since then, many other people have contributed to Stan
- Development is still focused around Columbia University but with a growing ecosystem of interface developers and casual contributors
  - Stan recently joined NumFOCUS

Stan Basics

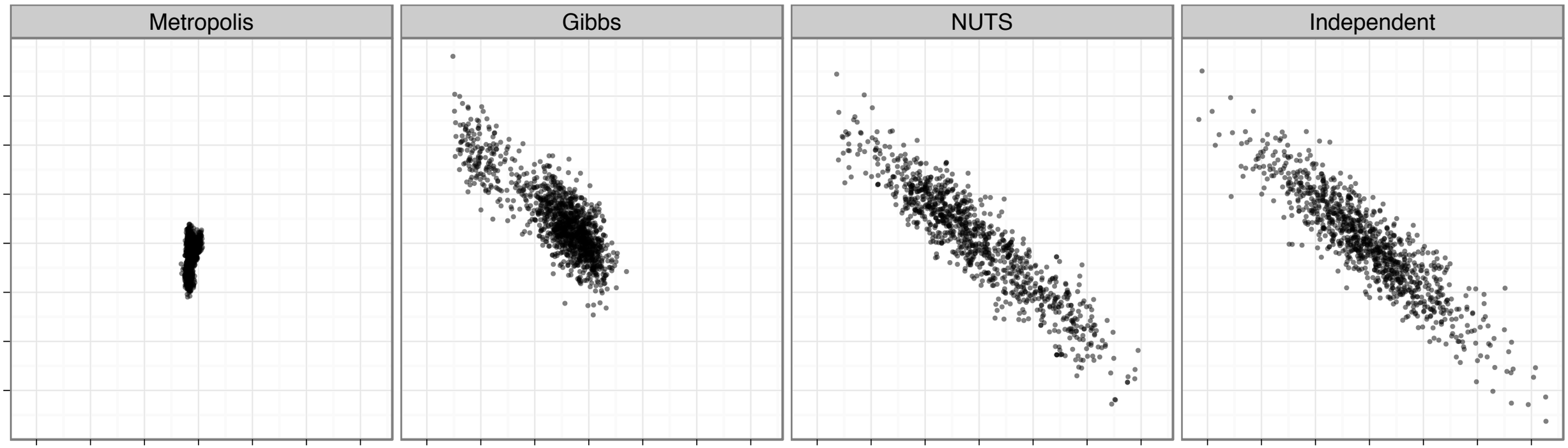
Why Stan?

- Stan was started because at the time existing tools for Bayesian inference (specifically MCMC) were quite poor
- Practically all packages used variations on two algorithms
  - Metropolis-Hastings with Gaussian proposal distributions
  - Gibbs sampling
- These algorithms were known to perform poorly on a wide range of problems but are easy to understand, implement and tune
  - With many models (highly correlated, hierarchical, non-conjugate priors, etc) these methods fail to converge in a reasonable time and only occasionally can this be detected

Stan Basics

Why Stan?

- Consider a simple 2D correlated Gaussian distribution



Stan Basics

What is Stan?

- Stan is an *imperative* probabilistic programming language used to define Bayesian posterior distributions or, more generally, any probability distribution which can be computed numerically.
- The goal of Stan is to enable efficient Bayesian inference in a flexible and intuitive manner without requiring large amounts of programming knowledge or mathematical background
  - Simple model changes (e.g., to add data or parameters, change priors, etc) should be easy
  - “Black box” inference

Stan Basics

What is Stan?

- A **program** in the Stan language defines:
  - Input data, parameters and
  - The log probability of those parameters given the data

$$\log p(\Theta|\mathcal{D})$$

- Stan also provides a variety of tools for **inference**:
  - MCMC with HMC/NUTS for sampling from $p(\Theta|\mathcal{D})$
  - Numerical optimization methods for MAP estimation
  - Variational Bayes for approximate inference

Stan Basics

What is Stan?

- Currently Stan consists of 250k lines of heavily templated C++03 (BSD license)
  - 200k lines consists of the Stan Math library, an automatic differentiation library which supports a wide range of scalar, vector and matrix operations and has optimized implementations of many probability distributions
  - 50k lines are the core Stan library and inference algorithms themselves
  - User interfaces wrap these libraries

Stan Basics

What is Stan?

- Stan has a variety of user interfaces
  - Command line (CmdStan), R (RStan), Python (PyStan) are the primary (and best supported) ones
  - Other interfaces exist for Matlab, Julia and Stata
- Runs on Linux, OS X and Windows
- Detailed documentation as well as a growing library of books introduce or discuss Stan
- Number of users is hard to estimate, but the manual for a recent release (v2.5.0) was downloaded over 10,000 times
  - Used to analyze LIGO data!

Stan Basics

How does Stan work?

- Consider flipping a coin and trying to estimate whether it's biased
- To analyze this problem with Stan we need to:

1. Write code in the Stan language which defines the data (the results of the coin flips), the parameters (the probability of the coin coming up heads), and the relationship between the parameters and the data (Bernoulli).
2. Compile that code into a program.
3. Perform various types of inference using that program.

Stan Basics

How does Stan work?

- The model in the Stan language

```
data {
  int<lower = 0> N;
  int<lower = 0, upper = 1> y[N];
}
parameters {
  real<lower = 0, upper = 1> theta;
}
model {
  y ~ bernoulli(theta);
}
```

Stan Basics

How does Stan work?

- Compiling the model with PyStan

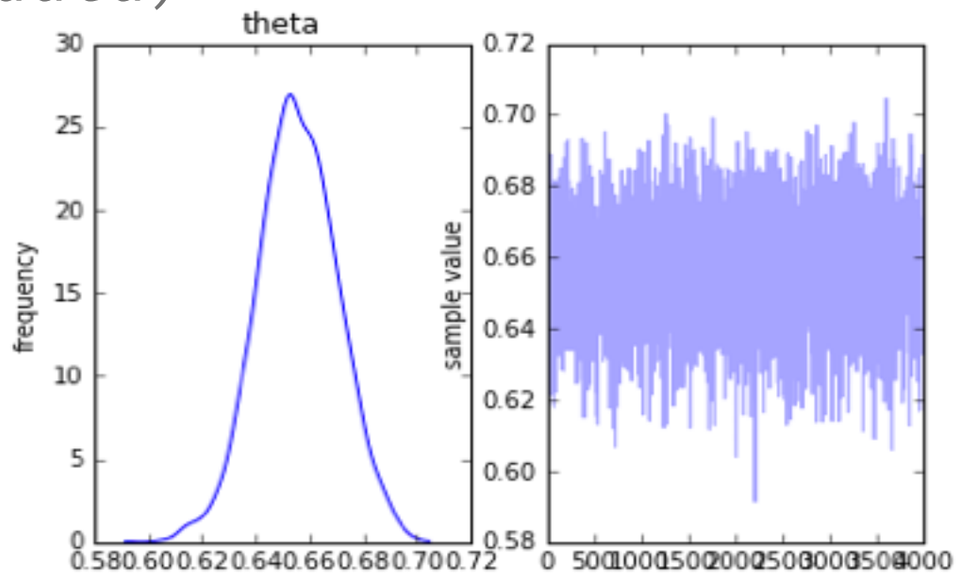
```
model_code = """
data {
  int<lower = 0> N;
  int<lower = 0, upper = 1> y[N];
}
parameters {
  real<lower = 0, upper = 1> theta;
}
model {
  y ~ bernoulli(theta);
}
"""

sm = pystan.StanModel(model_code=model_code)
```

Stan Basics

How does Stan work?

- Draw samples using MCMC and look at the results

```
>> mcmc_samples = sm.sampling(data=data)
>> print(mcmc_samples)
>> mcmc_samples.plot()
```




```
Inference for Stan model: bernoulli_1380e11a6c8b9c7fddb72f95376a4375.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

        mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
theta   0.66  5.2e-4   0.01   0.63   0.65   0.66   0.67   0.68  806.0    1.0
lp__  -645.6    0.02    0.7 -647.7 -645.7 -645.3 -645.1 -645.1  974.0    1.0
```

Stan Basics

How does Stan work?

- Find the MAP parameters

```
>> map_soln = sm.optimizing(data=data)
>> print(map_soln)

OrderedDict([(u'theta', array(0.6559999574881735))])
```

Background Interlude:
Bayesian Inference with MCMC

Bayesian Inference with MCMC

Expectation and Monte Carlo methods

- Performing Bayesian inference is ultimately about computing expectations under the posterior distribution

$$E[f(\Theta)|\mathcal{D}] = \int f(\Theta)p(\Theta|\mathcal{D})d\Theta$$

- **Monte Carlo method** (remember Stanislaw?) for computing integrals uses random samples $\Theta_i \sim p(\Theta|\mathcal{D})$

$$E[f(\Theta)|\mathcal{D}] = \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N} f(\Theta_i)$$

$$\approx \frac{1}{N}\sum_{i=1}^{N} f(\Theta_i)$$

Approximation error decreases as $\frac{1}{\sqrt{N}}$

Bayesian Inference with MCMC

Sampling with MCMC

- Sampling directly from the posterior is often just as intractable as computing the integral.
- **Markov Chain Monte Carlo** is a way to *approximately* sample from intractable distributions by making small random changes over and over again.
  - Think about shuffling a deck of cards by randomly moving one card at a time.
  - If the small random changes satisfy a set of technical conditions, then in theory one can prove that you can draw samples from any distribution
  - But there is a difference between theory and practice…

Bayesian Inference with MCMC

Sampling with MCMC

- A Markov Chain is a sequence of random variables

$$\Theta_1, \Theta_2, \Theta_3, \ldots$$

- The theory says that if you do things right the distribution of Θ_i will approach the desired distribution $p(\Theta|\mathcal{D})$ as i goes to infinity
  - However, it doesn't say anything about how quickly it approaches that distribution
  - Infinity turns out to be a **really** long time for some methods…

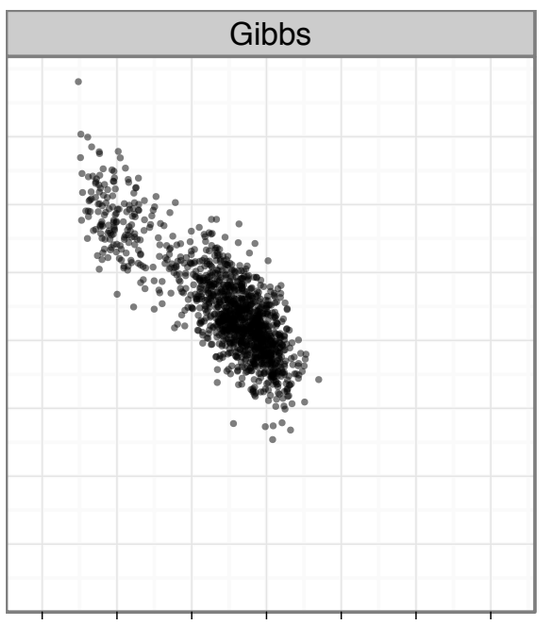
Bayesian Inference with MCMC

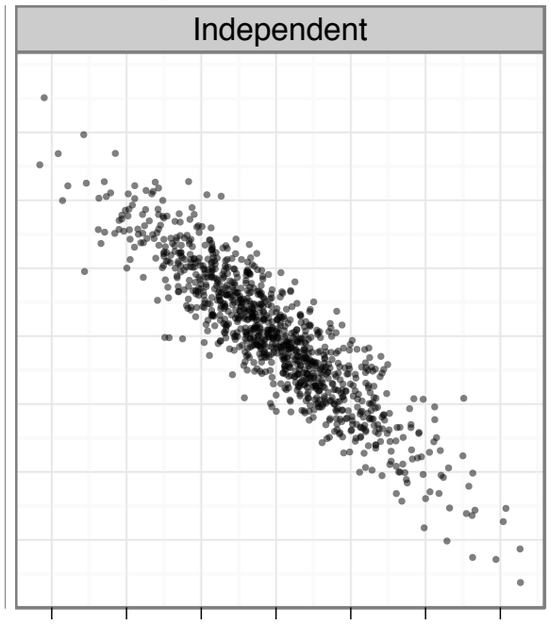
Sampling with MCMC (Gibbs sampling)

- Assume there are two parameters $\Theta = (\theta_1, \theta_2)$
- Gibbs sampling uses conditional distributions of the parameters

$$p(\theta_1|\theta_2, \mathcal{D}) \qquad p(\theta_2|\theta_1, \mathcal{D})$$

- Must be able to sample from the conditionals which limits the models and/or sampling efficiency
- What happens when two parameters are highly correlated?





Bayesian Inference with MCMC

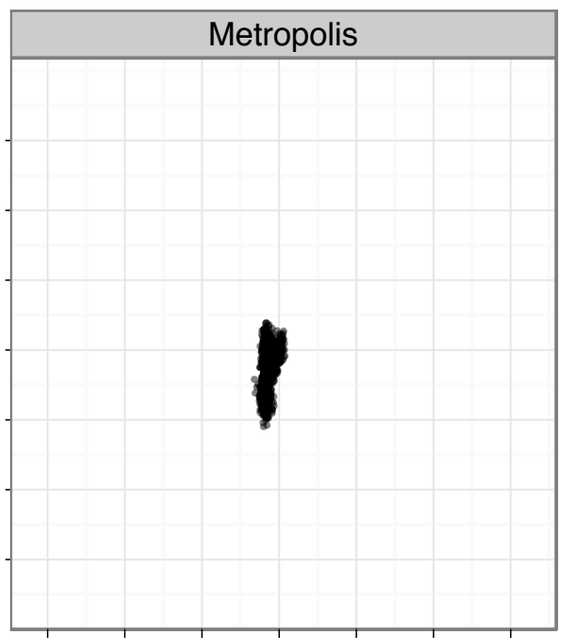
Sampling with MCMC (Gauss Metropolis)

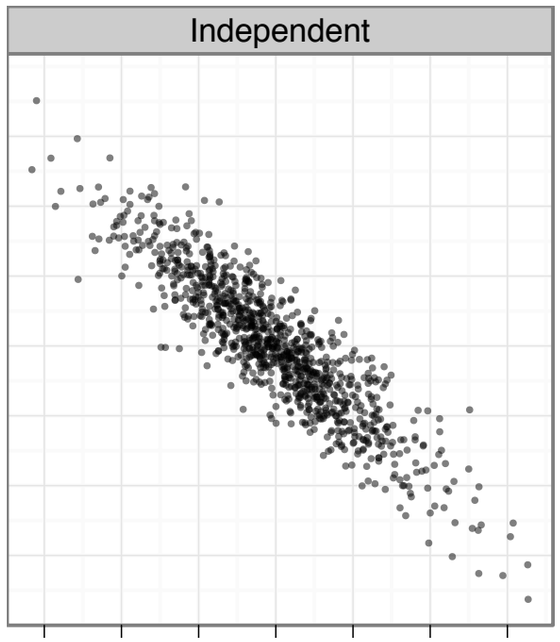
- Gauss Metropolis is the simplest version of Metropolis-Hastings

$$\hat{\Theta}_i \sim \mathcal{N}(\Theta_{i-1}, \Sigma) \qquad u \sim U(0,1)$$

$$\Theta_i = \begin{cases} \hat{\Theta}_i & u \leq \min\left(1, \frac{p(\hat{\Theta}_i|\mathcal{D})}{p(\Theta_{i-1}|\mathcal{D})}\right) \\ \Theta_{i-1} & \text{otherwise} \end{cases}$$

- Works, but is very slow: basically, a completely random walk





Bayesian Inference with MCMC

Sampling with MCMC

- How can we do better than GM or Gibbs?
- Think of MCMC as trying to find the lowest point in a region
  - Gibbs can make big moves, but can only move East-West or North-South at any one time.
  - GM can move in any direction, but the direction is chosen randomly and the distance travelled must be small.
- What if we picked the direction and distance based off the slope of the ground?
  - This is known as Langevin Monte Carlo
  - But the direction of the slope can change, so what if we followed the slope as it changed?

Bayesian Inference with MCMC

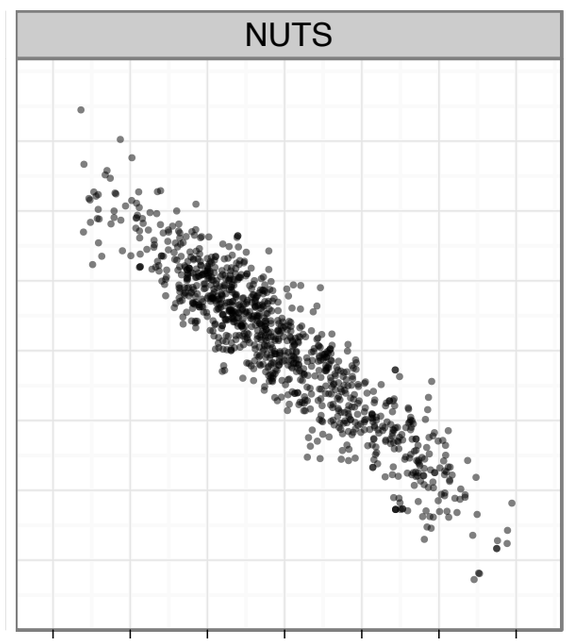
Sampling with MCMC (Hamiltonian Monte Carlo)

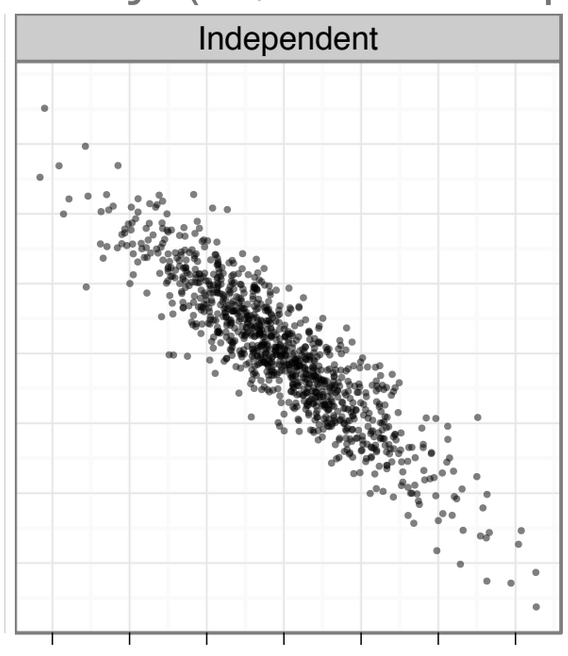
- HMC is most easily understood through a physical analogy
  - Consider placing a marble on a curved, frictionless surface and giving it a small, random push
  - Without friction the marble will trace out a periodic orbit
    - If you stop the marble after a time and give it another small random push, it will trace out a different periodic orbit
- A single orbit can be quite large, exploring the surface broadly
- Giving a new random push can produce a very different orbit
- In HMC
  - The position of marble is the current value of Θ
  - The height of the curved surface is the value of $-\log p(\Theta|\mathcal{D})$

Bayesian Inference with MCMC

Sampling with MCMC (Hamiltonian Monte Carlo)

- In each iteration of HMC
  1. Discard any previous "momentum" and randomly sample a new "momentum" for the parameters
  2. Perform a simulation for some amount of "time" to reach a new point on the orbit
  3. Keep the new point with some probability (ie, a Metropolis test)





Bayesian Inference with MCMC

Sampling with MCMC (Hamiltonian Monte Carlo)

- HMC can be extremely effective for a wide range of problems
- Been around since the late 80's, so why did it take so long to become widely used?
  - Need both the value and the derivatives of $p(\Theta|\mathcal{D})$
  - Doesn't naturally handle constraints on Θ, must reparameterize problems with constraints (e.g., correlations, covariance matrices, simplexes, etc)
  - There are several tuning parameters which dramatically impact performance and can be difficult to set by hand
- Stan was designed and built to automatically handle these and other issues with using HMC/MCMC

Part 2: Stan Under the Hood

Stan Under the Hood

MCMC with Stan

- HMC has two main parameters
  - Simulation step size, i.e., how carefully do we simulate
  - Length of time to perform simulation
- These can have a huge impact on sampling performance

Stan Under the Hood

MCMC with Stan (Adaptation)

- Step size adaptation
  - Large step sizes are inaccurate and lead to rejected proposals and wasted computation time
  - Excessively small step sizes can also waste computation time, even if steps are generally accepted
- Step sizes also vary between parameters (e.g., some parameters may be fast, while others must be simulated slowly)
- To estimate step sizes, Stan use a “warmup” period
  - Different step-sizes are tried until the steps are accepted 80%
  - This uses a form of stochastic optimization known as dual-averaging and is often successful, but there are no assurances

Stan Under the Hood

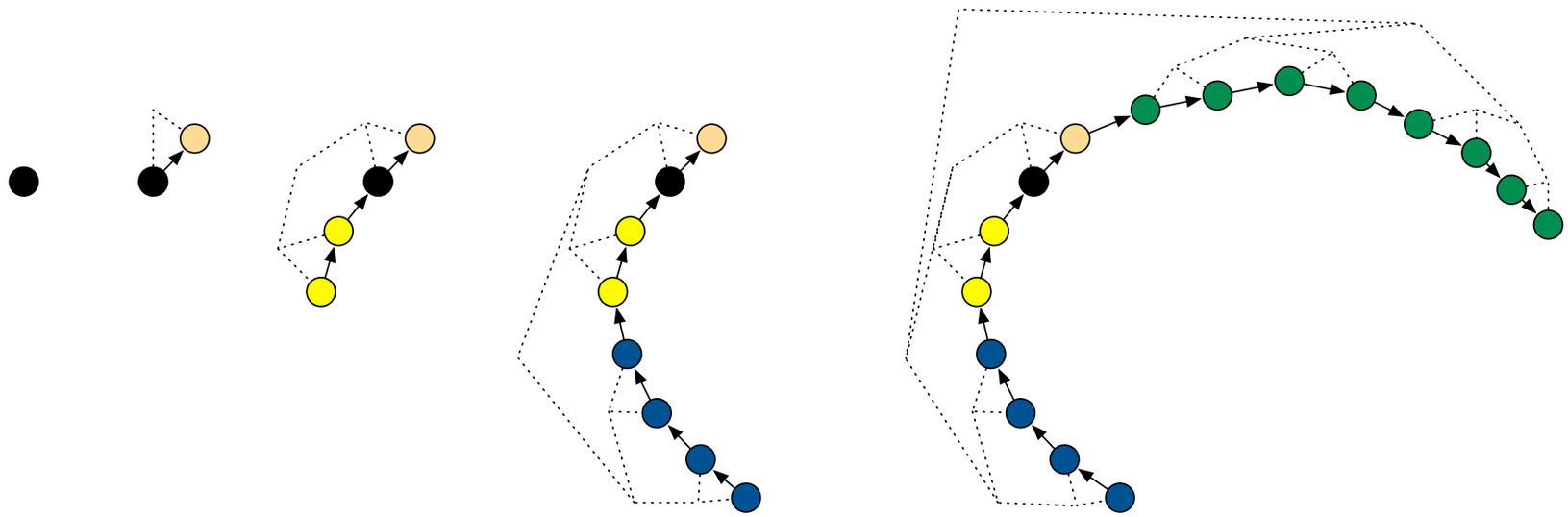
MCMC with Stan (Adaptation)

- Length of time to perform simulation
  - Short simulations are effectively Gauss Metropolis proposals
  - Long simulations can be counter productive (*periodic orbits*!)
- The “optimal” simulation time is hard to estimate
- It depends on where you are in parameter space and the specific amount of momentum
- To allow this parameter to be tuned automatically and locally, Stan uses the No-U-Turn Sampler (NUTS), a variation of HMC

Stan Under the Hood

MCMC with Stan (No-U-Turn Sampling)

- NUTS starts with the idea that once a simulation starts to "turn around" then it should be stopped
- The resulting method works by successively doubling the length of the simulation until a U-Turn is detected
- Simulation time grows exponentially



Stan Under the Hood

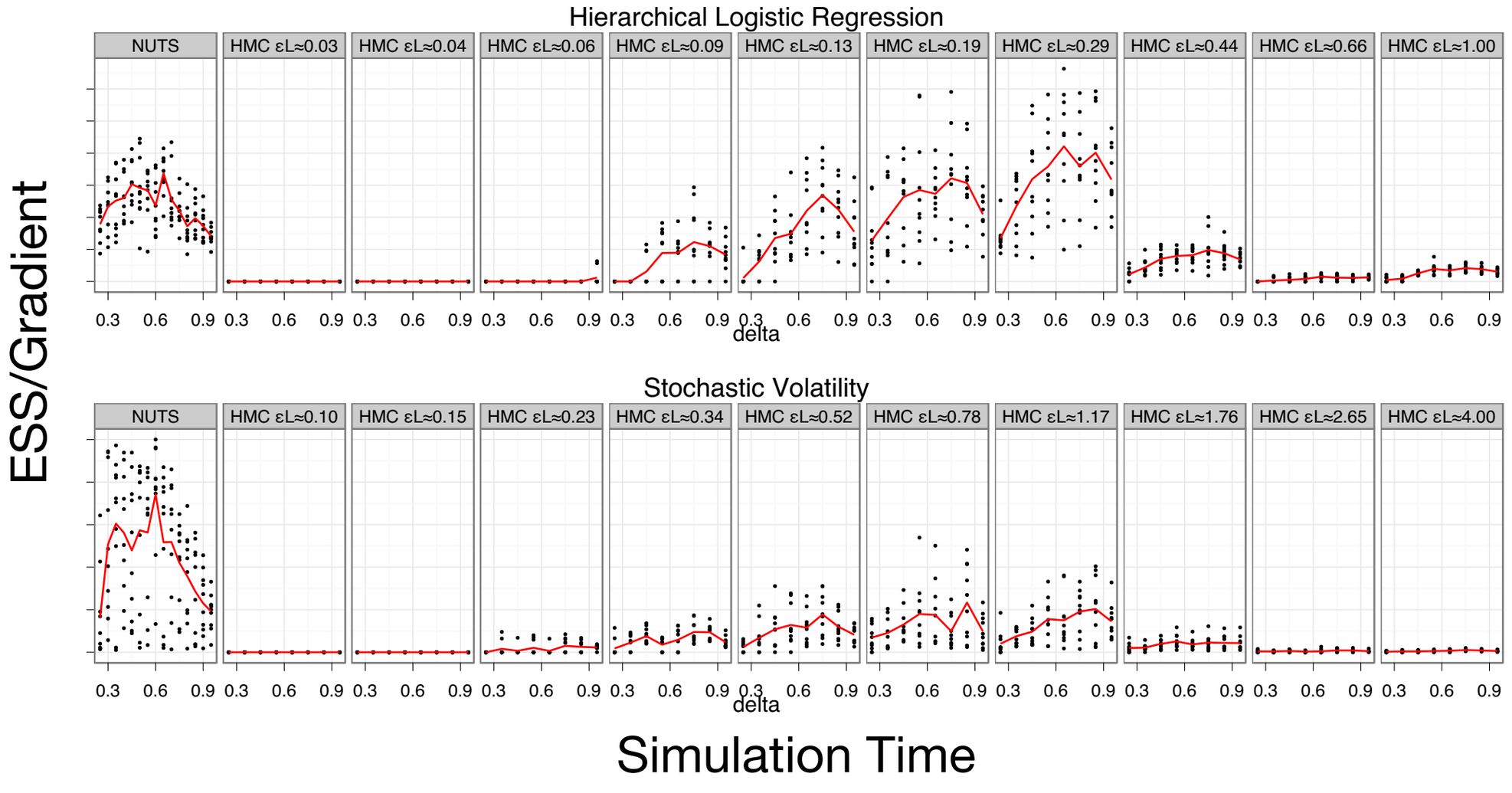
MCMC with Stan (NUTS vs Vanilla HMC)

- To compare MCMC methods we need a good metric
  - Effective Sample Size (ESS) is a measure of the number of nominally independent samples
  - A good MCMC method produces a high ESS per unit of computation
  - Gradient computation is the most expensive operation in HMC, so we use ESS/gradient as a metric

Stan Under the Hood

MCMC with Stan (NUTS vs Vanilla HMC)

- NUTS almost always outperforms optimally tuned HMC



Stan Under the Hood

Inside a Stan Model

- Previously mentioned three problems with HMC
  - Parameter tuning
  - Constrained parameters
  - Gradients
- How does Stan handle constrained parameters?
  - Constrained variables can be reparameterized as unconstrained
  - Probability theory requires that we correct for “volume changes”

$$x = f(y)$$

$$p_Y(y) = p_X(f(y)) \left| \det \frac{\partial f}{\partial y} \right|$$

Stan Under the Hood

Inside a Stan Model (Constrained Parameters)

- For instance, consider the parameter from the Bernoulli model

```
parameters {
  real<lower = 0, upper = 1> theta;
}
```

- This can be converted to an unconstrained scale with the *logit transformation*

$$\hat{\theta} = \log \frac{\theta}{1-\theta} = \text{logit}(\theta) \qquad \theta = \frac{1}{1+\exp(-\hat{\theta})} = \text{logit}^{-1}(\hat{\theta})$$

- To transform a distribution to be in the unconstrained scale

$$p_{\hat{\theta}}(\hat{\theta}) = p_{\theta}(\text{logit}^{-1}(\hat{\theta}))\left(\text{logit}^{-1}(\hat{\theta})(1-\text{logit}^{-1}(\hat{\theta}))\right)$$

Stan Under the Hood

Inside a Stan Model (Constrained Parameters)

- When a parameter is declared to have constraints in Stan, it is automatically reparameterized and the appropriate volume correction is incorporated into the posterior distribution
- Then inference algorithms (NUTS, optimization, etc) can all work directly on unconstrained variables
- Stan supports most commonly used constraints
  - Upper, lower and both bounds on scalars and vectors or matrices of scalars with such bounds
  - Unit sum, unit length and sorted vectors
  - Covariance, correlation and Cholesky factor matrices

Stan Under the Hood

Inside a Stan Model (Automatic Differentiation)

- NUTS/HMC, optimization and other methods require the ability to evaluate the log probability and its gradients
- A model in Stan is translated to C++ code implementing a templated function which computes the log probability using the Stan Math library
  - Stan Math library uses (reverse mode) automatic differentiation to be able to efficiently compute the gradient of the function
  - The library also includes optimized implementations of probability distributions, matrix operations, ODEs, and other mathematical functions
- The translated C++ code is compiled before inference can be performed

Stan Under the Hood

Inside a Stan Model (Automatic Differentiation)

- In a sense, the Stan language is a thin wrapper around the Stan Math library
- For instance, the Bernoulli model from before

```
parameters {
  real<lower = 0, upper = 1> theta;
}
model {
  y ~ bernoulli(theta);
}
```

- Translates (roughly) to C++ that looks like

```
theta = logit_inv(theta_hat);
lp__ = logit_volume_correction(theta_hat);
for (int i = 0; i < N; i++) {
  lp__ += bernoulli_log(y[i],theta);
}
```

Stan Under the Hood

Inside a Stan Model (Automatic Differentiation)

- In fact, we can actually write the Bernoulli model in a way which makes this much clearer
- For instance, these two model blocks are equivalent

```
model {
  y ~ bernoulli(theta);
}
```

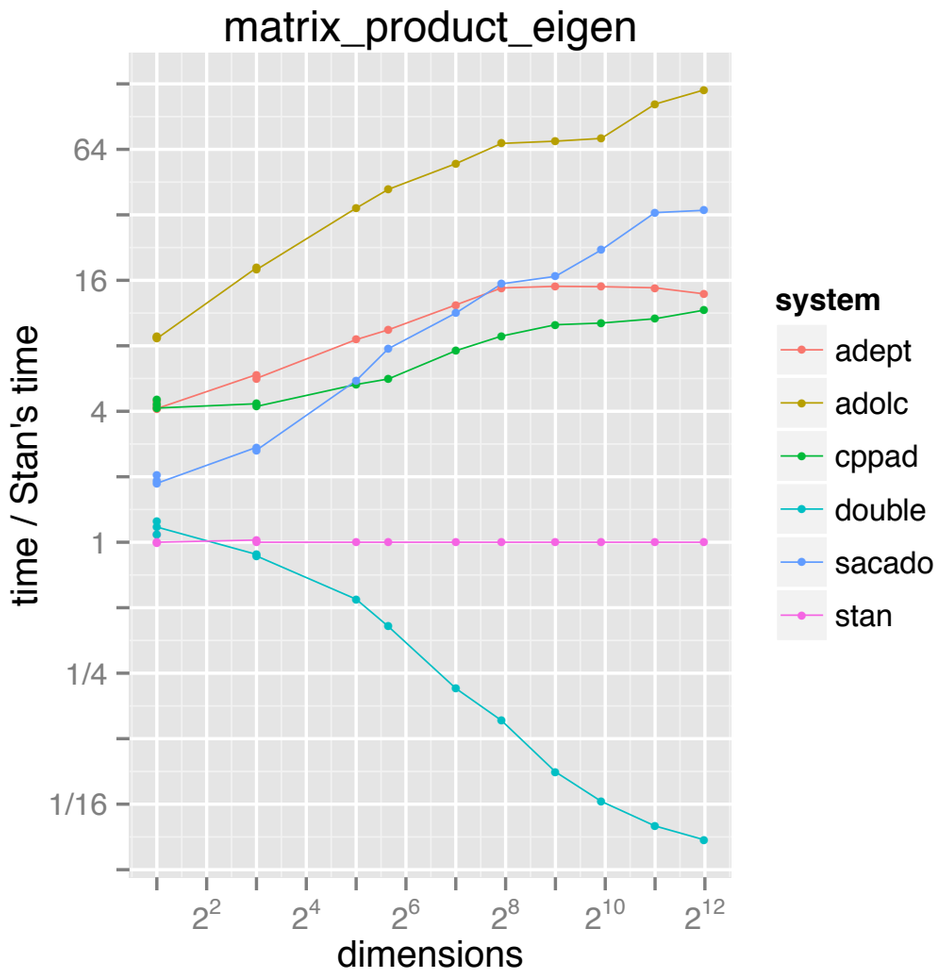
```
model {
  for (i in 1:N)
    increment_log_prob(bernoulli_log(y[i],theta));
}
```

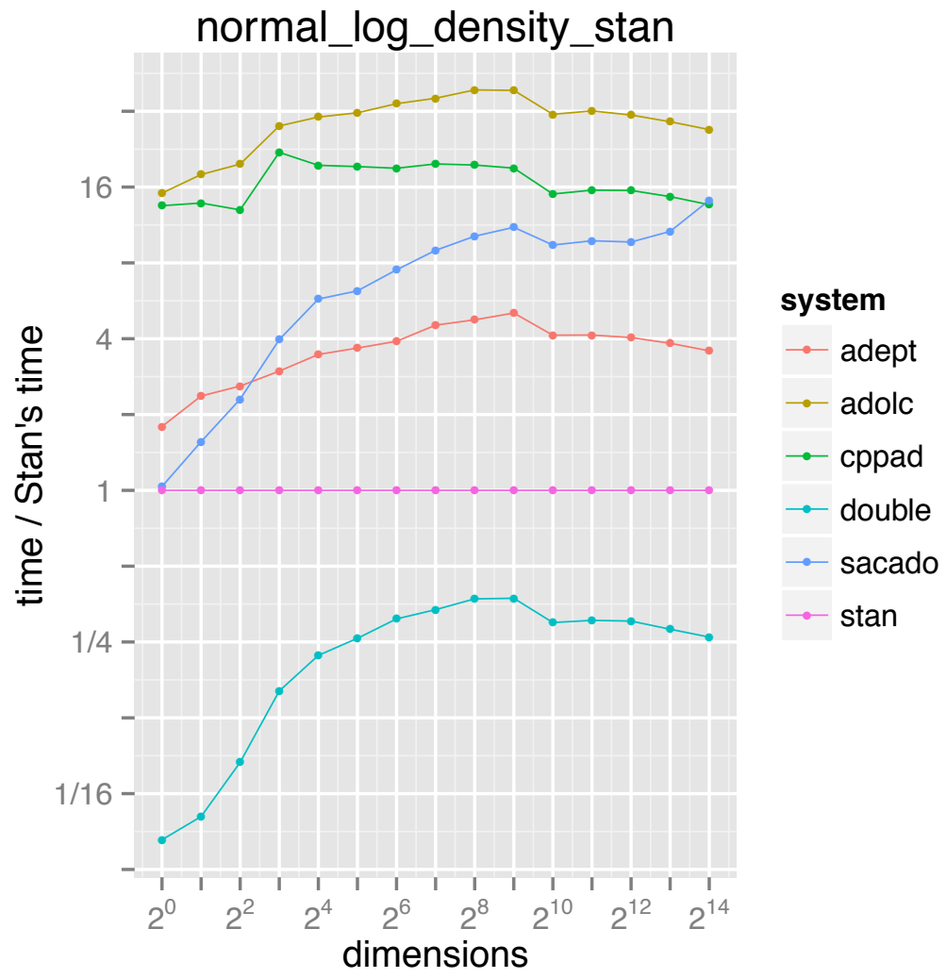
- The first model will be faster but they implement equivalent models

Stan Under the Hood

Inside a Stan Model (Automatic Differentiation)

- Stan's auto diff implementation is one of the fastest among OSS
- Also uses the least amount memory





Stan Under the Hood

Inside a Stan Model (Automatic Differentiation)

- Stan Math currently supports computing first derivatives through reverse mode autodiff
- Hopefully soon it will also support second and third order derivatives
  - Support for this has been at about 90% done for the last year or so, but there are a few sticking points for the remaining 10% which turns out to be really important

Part 3: Coming soon to Stan

Coming soon to Stan

Variational Inference

- A new technique for approximate Bayesian inference in models for which MCMC would be impractical
- Something of a hybrid between MCMC and local optimization
  - Currently implemented in the core Stan library, but not yet available in all the interfaces
  - Hopefully in the next release (v2.10.0) it will be exposed in PyStan and RStan

Coming soon to Stan

Riemann Manifold HMC w/ SoftAbs Metric

- Even *more* efficient than NUTS
- Uses the local Hessian to adapt the step-size to local curvature
  - Somewhat like NUTS adapts simulation length
- Implementation requires 2nd and 3rd order derivatives
  - Manual implementation is tedious at best
  - Storing Hessians and 3rd order derivative tensor is impractical
  - Efficient implementation uses autodiff'd functionals
    - gradient-vector and Hessian-vector products
    - gradient of trace of matrix-Hessian products
- Mostly implemented, waiting on the last 10% of high order autodiff

Coming soon to Stan

Adiabatic Monte Carlo

- RMHMC, NUTS, HMC, GM, and most other MCMC methods perform poorly when the distribution is *multi-modal*
- Adiabatic Monte Carlo gradually warps between the target distribution and a fairly simple (broad, unimodal) one making it possible to easily jump between different modes
  - Similar in spirit to simulated annealing and tempered sampling but much more theoretically elegant and effective
- Still some practical and theoretical issues to resolve and also requires 2nd and 3rd order derivatives

References and Resources

MCMC, HMC, NUTS, etc

- "MCMC Using Hamiltonian Dynamics" by R.M. Neal in *Handbook of Markov Chain Monte Carlo,* (2011).
- "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo" by M.D. Hoffman and A. Gelman, arXiv:1111.4246 (2011).
- "Riemann manifold Langevin and Hamiltonian Monte Carlo methods" by M. Girolami and B. Calderhead, Journal of the Royal Statistical Society: Series B (2011).
- "A General Metric for Riemannian Manifold Hamiltonian Monte Carlo" by M.J. Betancourt, arXiv: 1212.4693 (2012).
- "Adiabatic Monte Carlo" by M.J. Betancourt, arXiv:1405.3489 (2014).
- Basically anything by Michael Betancourt or Radford Neal…

Stan

- The Stan User's Manual and other docs at http://mc-stan.org
- "The Stan Math Library: Reverse-Mode Automatic Differentiation in C++" by B. Carpenter et al, arXiv: 1509.07164 (2015).