Bayesian Analysis using Stan

June 17, 2019
way-too-early o'clock

Daniel Lee
daniel@generable.com

Ground Rules / Expectations

- Please ask questions!
- 45 minutes: feel free to stand up, get more coffee
- Expectation: can't learn Stan in 45 minutes.

Overall Goals

- Stan is awesome!
    - Technical breakthroughs
    - Modeling flexibility
- It's not easy.
    - Introduces a new dimension of difficulty
- There are resources.
    - Forums: https://discourse.mc-stan.org

Who's heard of Stan?

Who's written a Stan program?

What is Stan?

What is Stan?

1. Language

2. Algorithms

3. Interfaces

Language for Statistical Models

- **Goal:** specify statistical models

$$x$$

data

Language for Statistical Models

- **Goal:** specify statistical models

$$\underbrace{\theta}_{\text{parameters}}, \underbrace{x}_{\text{data}}$$

Language for Statistical Models

- **Goal:** specify statistical models

$$\underset{\text{model}}{p}(\underset{\text{parameters}}{\theta}, \underset{\text{data}}{x})$$

Language for Statistical Models

- **Goal:** specify statistical models

$$p(\theta, x)$$

model parameters data

- Stan is a language
    - statically typed, imperative
    - users define programs: data, parameters, **log joint pdf**
- User can specify any *differentiable* joint probability distribution function over data and parameters

Language for Statistical Models

- **Goal:** specify statistical models

$$\underbrace{p}_{\text{model}}(\underbrace{\theta}_{\text{parameters}}, \underbrace{x}_{\text{data}})$$

What's the problem?

Example: Hello World

```
data {
}
parameters {
}
model {
  print("hello world!");
}
```

Example: Logistic Regression

```
data {
  int<lower=0> N;
  vector[N] x;
  int<lower=0,upper=1> y[N];
}
parameters {
  real alpha;
  real beta;
}
model {
  y ~ bernoulli_logit(alpha + beta * x);
}
```

Users define the statistical model

$$p(\theta, x)$$

Inference algorithms use $p(\theta, x)$

- Bayesian inference; Markov Chain Monte Carlo (MCMC)

- Approximate Bayesian inference

- Optimization

Inference algorithms use $p(\theta, x)$

- Bayesian inference; Markov Chain Monte Carlo (MCMC)
  - $p(\theta \mid x)$ approximated with $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}\}$
- Approximate Bayesian inference
  - ex: $\hat{p}(\theta \mid x) \approx q(\hat{\phi})$ where $\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \; D_{\mathrm{KL}}\left(q(\theta \mid \phi) \,||\, p(\theta, x)\right)$
- Optimization
  - $\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \; p(\theta, x)$ (only holds when there's a single optima)

Inference algorithms use $p(\theta, x)$

- **Bayesian inference; Markov Chain Monte Carlo (MCMC)**
  - $p(\theta \,|\, x)$ approximated with $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(N)}\}$
- Approximate Bayesian inference
  - ex: $\hat{p}(\theta \,|\, x) \approx q(\hat{\phi})$ where $\hat{\phi} = \underset{\phi}{\operatorname{argmin}} \; D_{\mathrm{KL}}\left(q(\theta \,|\, \phi) \,||\, p(\theta, x)\right)$
- Optimization
  - $\hat{\theta} = \underset{\theta}{\operatorname{argmax}} \; p(\theta, x)$ (only holds when there's a single optima)

Interfaces

- CmdStan, RStan, PyStan
- C++ API
- C++ automatic differentiation library
- RStanArm, brms, prophet, …

Stan: mc-stan.org

- Language

- Inference algorithms

- Interfaces

- Open-source github.com/stan-dev
  core: BSD
  interfaces: GPL or BSD

Why is Stan so awesome?

MCMC Algorithm Breakthrough

- The No-U-Turn Sampler (NUTS). Matthew D. Hoffman, Andrew Gelman. Journal of Machine Learning Research. 2014. http://jmlr.org/papers/v15/hoffman14a.html
  - Improved on Hamiltonian Monte Carlo (HMC).
    - 2 tuning parameters: stepsize, number of steps
    - Performance (very) sensitive to tuning parameters
  - NUTS
    - 1 tuning parameter: stepsize
    - Works for many models

250 dimensional Normal distribution

2d projection

1000 draws

Hoffman and Gelman. "The No-U-Turn Sampler." JLMR. 2014.

Quick aside: HMC is not enough

Hoffman and Gelman. "The No-U-Turn Sampler." JLMR. 2014.

Stan: algorithm + autodiff

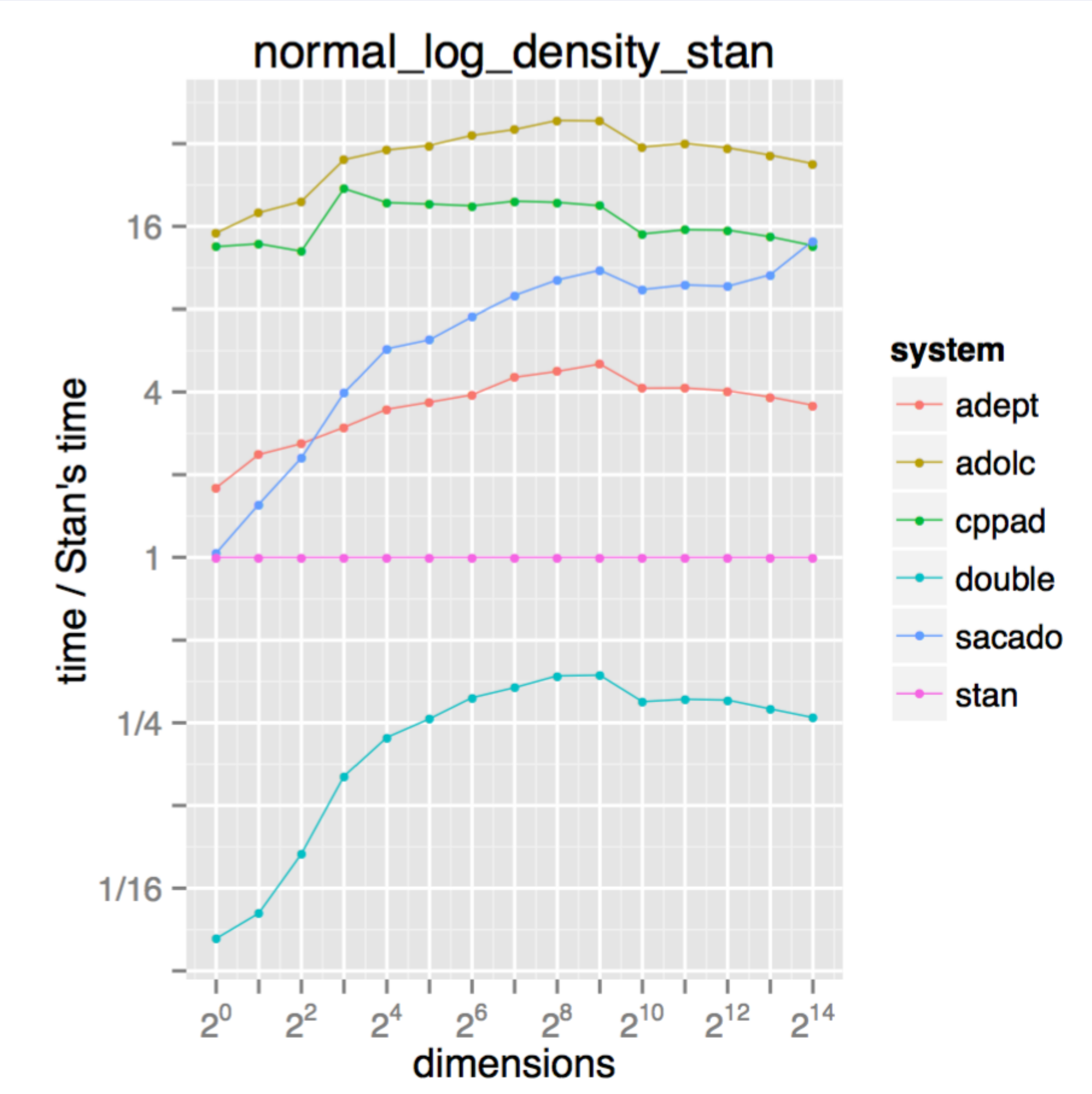
- The No-U-Turn Sampler requires the function and **gradients**

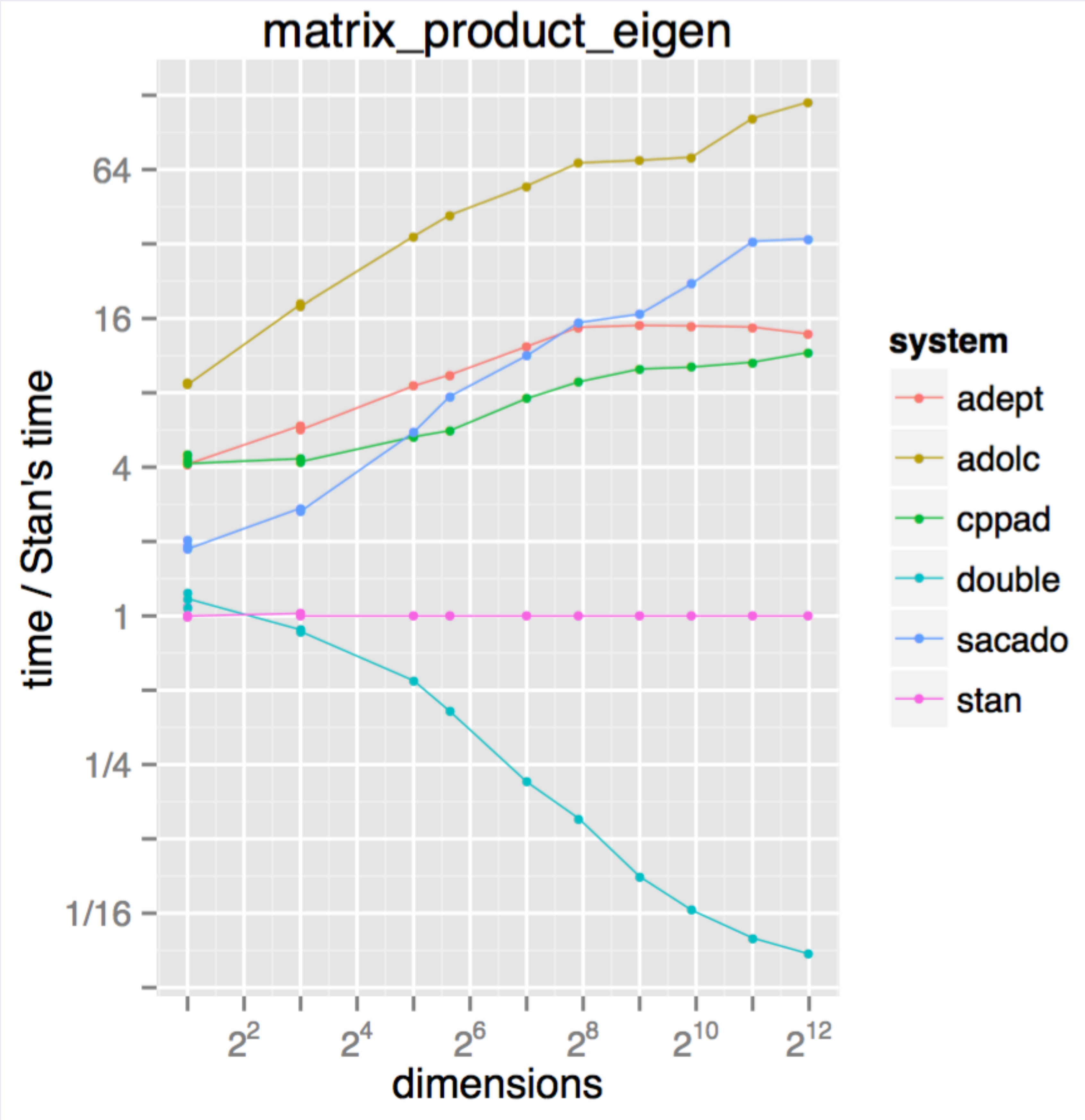
  1. $\log f(\theta, x) = \log p(\theta, x) + C$ **i.e.** $f(\theta, x) \propto p(\theta, x)$
  2. $\dfrac{d \log f(\theta, x)}{d\theta}$ **gradients** with respect to parameters, **all parameters**

- **Automatic differentiation**
  - Provides gradients of any (continuous) function (even non-analytic)
  - Reverse mode: **O(1) time complexity** with respect to number of parameters

Autodiff comparison

- For open-source C++ packages:
  Stan is **fastest** (for gradients),
  most **general** (functions supported), and most easily **extensible** (simple OO)
  https://arxiv.org/abs/1509.07164





Stan: algorithm + autodiff + language

- **Great! We have an algorithm + autodiff.**
    - Who wants to write statistical models in C++?
    - Btw, this was Stan circa 2011, pre v1.0.
- **Enter the Stan language.**
    - Any Stan program can be autodiffed!
    - No-U-Turn Sampler can be applied to any Stan program!

Why is Stan so awesome?

For any model, we can run the No-U-Turn Sampler!

- A high-level language for specifying statistical models
- Does a whole lot of $#*! under the hood
- Produces MCMC draws
    - Estimate the pdf with draws from the posterior

So... what's the bad news?

Difficulties

- **Modeling:** Not all models match data;
  fails to produce good output
- **Computation:** New diagnostics
- **Skills:** Programming in Stan;
  leaky abstraction -- mcmc, computation, autodiff
- **No built-in models:** Blessing and a curse;
  no vetted, canned models with robust, built-in
  regularization / approximations

Language basics

RECAP

STATISTICAL INFERENCE

Want

posterior distribution of parameters given data $$p(\theta \mid x)$$

Given

joint model $$p(\theta, x)$$

data $$x$$

parameters $$\theta$$

WHY MCMC?

$$p(\theta \mid x) = \frac{p(\theta, x)}{p(x)}$$

$$= \frac{p(\theta, x)}{\int p(\theta, x) d\theta}$$

WHY MCMC?

$$
\begin{aligned}
p(\theta \mid x) &= \frac{p(\theta, x)}{p(x)} \\
&= \frac{p(\theta, x)}{\int p(\theta, x) d\theta}
\end{aligned}
$$

WHY MCMC?

$$p(\theta \mid x) \propto \boldsymbol{p(\theta, x)}$$

- MCMC generates draws from the posterior distribution
- We write the joint distribution, Stan does the MCMC
- Stan estimates expectations

$$\mathbb{E}[f(x,\theta)] = \int f(x,\theta) \times p(\theta \mid x) dx$$

IN STAN, WE DEFINE

- Joint model of data and parameters: $\log p(\theta, x)$
  - Define data x
  - Define parameters θ

STAN LANGUAGE

GENERAL PROPERTIES OF STAN LANGUAGE

- Whitespace does not matter
- Comments
  - // or #
  - /* … */
- semicolon (;)
- Variables are typed and scoped
- Compile-time vs run-time errors

BLOCKS: STRUCTURE OF A STAN PROGRAM

```
functions
```

```
data
```

```
transformed data
```

```
parameters
```

```
transformed parameters
```

```
model
```

```
generated quantities
```

- Blocks start and end with braces (`{ }`)
- `model` block is required
- Variables declared in each block have scope over all subsequent statements

STAN TYPES

VARIABLE DECLARATION

- Each variable has a type (**static type**)
- Only values of that type can be assigned to the variable (strongly typed)
- Declaration of variables happen at the top of a block (including local blocks)

- Start by learning about the types in the context of the data block

SCALAR DATA TYPES

`real`

- scalar
- continuous

```
data {
  real y;
}
```

`int`

- scalar
- integer
- can't be used in `parameters` or `transformed parameters` blocks

```
data {
  int n;
}
```

CONSTRAINING SCALAR VARIABLES

- Validates data is within range
- lower bound, upper bound, both
- inclusive
- can use infinite constraints:
  `positive_infinity()`
  `negative_infinity()`
- bounds can be expressions

```
data {
  int<lower = 1> m;
  int<lower = 0, upper = 1> n;
  real<lower = 0> x;
  real<upper = 0> y;
  real<lower = -1, upper = 1> rho;
}
```

VECTOR DATA TYPES

- Contains real values
- Indexing starts at 1
- Declared with size

- `vector[3] a;`
  column vector
- `row_vector[4] b;`
  row vector
- `simplex[5] c;`
  vector, sums to 1, non-negative entries
- `unit_vector[5] d;`
  vector with norm of 1
- `ordered[6] e;`
  vector in ascending order
- `positive_ordered[7] f;`
  positive, ordered vector

MATRIX DATA TYPES

- Contains real values
- Indexing starts at 1
- Declared with size

- ```
  matrix[3,4] A;
  3x4 matrix
  A[1] returns a 4-row_vector
  ```
- ```
  corr_matrix[3] Sigma;
  square, symmetric matrix
  positive definite
  entries between -1 and 1
  diagonal 1
  ```
- ```
  cholesky_factor_corr[K] L;
  represents Cholesky factor
    of correlation matrix
  K x K lower-triangular
  positive diagonal entries
  rows are length 1
  L L^T is a correlation matrix
  ```
- ```
  cov_matrix[3] Omega;
  symmetric, square, positive definite
  ```
- ```
  cholesky_factor_cov[M,N] L;
  cholesky_factor_cov[4] L; (square matrix)
  L L^T is a covariance matrix
  ```
  ```

ARRAYS

- All types can be made arrays
- Arrays can be multi-dimensional
- Examples
  - `real a[5];`
  - `vector[5] b[3];`
  - `int N[2,3];`
  - `vector<lower=0>[5] c[L,M,N];`

RECAP: STAN TYPES

Scalar types

- `real`
- `int`

Vector types

- `vector, row_vector`
- `simplex, unit_vector`
- `ordered, positive_ordered`

Matrix types

- `matrix`
- `corr_matrix, cov_matrix`
- `cholesky_factor_corr, cholesky_factor_cov`

Bounds

- `lower, upper, both`

Arrays

DATA

THE DATA BLOCK

- **Declare** data only
- Within the block, can't do anything else
- Data read in from Stan interface in order declared
- All data declared must be passed by the Stan interface
- Data is validated; happens once per execution

RECAP: DATA AND TRANSFORMED DATA

- data and transform data is the data in $p(\theta, x)$
- Both blocks are executed once for the whole program
- Execution is fast
- Both blocks validate data
- Variables in transformed data are not saved

LOOPS, CONDITIONALS, BLOCKS, HELPER FUNCTIONS

- For loop: `for (n in 1:N) …`
- while loop: `while (cond) …`
- conditional: `if (cond) … else if (cond) … else …`
- blocks: `{ … }` (local variables can be declared at the top)
- helper functions:
  - `print("message", expression, …)` – prints message and expressions
  - `reject("message")` – throws an error (used for debugging) with the message specified

PARAMETERS AND TRANSFORMED PARAMETERS

PARAMETERS

- **Declare** variables in the same way as data
- int parameters are not allowed
  Not differentiable; Stan language limited by inference algorithms
- Parameters with constraints has implicit transforms;
  changes constrained parameters to unconstrained parameters:
  guarantees sampling is over the range provided
- Can't define the value of a parameter
  (no assignment)

EXERCISES

1. Improper posterior. Run this model. (no data)

```
parameters {
  real theta;
}
model {
}
```

```
fit <- stan("example.stan")
```

2. Proper posterior. Put lower and upper bound on theta.
   Run new model.

MODEL

WRITING A JOINT MODEL

- Data and parameters of model defined
- Model block: log joint probability

- `target +=`
  – directly increments the log probability
- sampling statements provide convenient, often efficient shortcuts

EXAMPLE: BERNOULLI COIN FLIP

One way of writing this model:

$$\theta \sim \mathrm{Beta}(1, 1)$$
$$y_i \sim \mathrm{Bernoulli}(\theta)$$

Another is:

$$\Pr(\theta, y_1, y_2, \ldots) = \prod_i \theta^{y_i} (1 - \theta)^{1 - y_i}$$

EXAMPLE MODEL: BERNOULLI COIN

- Start with data:
  - N: number of flips
  - y: N-array of int between 0 and 1
- Parameters:
  - theta: real between 0 and 1
- Model
  - use "target += " within a loop

EXAMPLE MODEL: BERNOULLI COIN

```
data {
  int N;
  int<lower=0, upper=1> y[N];
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  target += log(1);
  for (n in 1:N)
    target += log(if_else(y[n] == 1,
                          theta, 1 - theta)));
}
```

EXAMPLE MODEL: BERNOULLI COIN (DISTRIBUTION FUNCTION)

```
data {
  int N;
  int<lower=0, upper=1> y[N];
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  for (n in 1:N)
    target += bernoulli_lpmf(y[n] | theta);
}
```

VECTORIZATION

- Vectorized statements save calculations
  (where it can be saved)
- What functions are vectorized?
  Plural in manual: reals, vectors, ints

EXAMPLE MODEL: BERNOULLI COIN (VECTORIZED)

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

ANY ARBITRARY MODEL CAN BE WRITTEN USING

```
target +=
```

TARGET +=

- define models by incrementing log probability of model
- why log probability?
  - numeric stability
- Available functions
  - math
  - matrix
  - probability distributions

“SAMPLING” STATEMENTS

- Syntactic sugar for distribution functions:

```
target += foo_lpdf(lhs | arg1, arg2, …))
target += foo_lpmf(lhs | arg1, arg2, …))
```

- Sampling statements:

```
lhs ~ foo(arg1, arg2, …)
```

- Drops constant term

- **Stan does not do rejection sampling.**
  There is no “drawing” from the distribution

EXAMPLE MODEL: BERNOULLI COIN (SAMPLING)

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

EXAMPLE MODEL: BERNOULLI COIN (VECTORIZED SAMPLING)

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

FUNCTIONS

USER-DEFINED FUNCTIONS

```
functions {


}
data { ... }
```

- First block in Stan program
- Types in signature a little different: lose dimensions
- All arguments mandatory
- Must return
- Can forward declare, if necessary

EXAMPLE FUNCTION

```
functions {
  int fib(int n);

  int fib(int n) {
    if (n > 2)
      return n;
    else
      return fib(n - 1) + fib(n - 2);
  }

}
...
```

EXAMPLE DISTRIBUTION (WILL ERROR IN 2.11, BUT THIS IS THE SYNTAX)

```
functions {
  real foo_lpdf(real y, real theta) {
    …
  }
}
...
model {
   ...
   y ~ foo(theta);
   target += foo_lpdf(y | theta);
}
```

Example

Overall Goals

- Stan is awesome!
  - Technical breakthroughs
  - Modeling flexibility
- It's not easy.
  - Introduces a new dimension of difficulty
- There are resources.
  - Forums: https://discourse.mc-stan.org

Thank you!

daniel@generable.com

@djsyclik

mc-stan.org