# Stan:
## Probabilistic Modeling & Bayesian Inference


### Development Team

Andrew Gelman, **Bob Carpenter**, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Allen Riddell, Marco Inacio, Jeffrey Arnold, Mitzi Morris, Rob Trangucci, Rob Goedman, Brian Lau, Jonah Sol Gabry, Robert L. Grant, Krzysztof Sakrejda, Aki Vehtari, Rayleigh Lei, Sebastian Weber, Charles Margossian, Vincent Picaud, Imad Ali, Sean Talts, Ben Bales, Ari Hartikainen, Matthijs Vàkàr, Andrew Johnson, Dan Simpson



Stan 2.17 (November 2017) `http://mc-stan.org`

# StanCon 2018

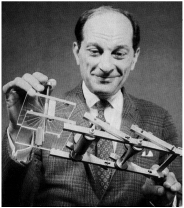
- 10–12 January 2018
- Asilomar (Pacific Grove, CA — 2 hrs south of SFO)
- Early registration still available
- Tutorials by Stan developers at all levels
- 6 keynotes from science and business
- Breakout “unconference” sessions

- http://mc-stan.org/events/stancon2018/

# What is Stan?

# Who is Stan?

- Named in honor of **Stanislaw Ulam** (1909–1984)
- Co-inventor of the **Monte Carlo method**



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

image from G. C. Geisler (2000) Los Alamos report LA-UR-2532

# What is Stan?

- Stan is an **imperative** probabilistic programming language
  - cf., BUGS: declarative; Church: functional; Figaro: object-oriented
- Stan **program**
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)
- Stan **inference**
  - MCMC for full Bayesian inference
  - VB for approximate Bayesian inference
  - MLE for penalized maximum likelihood estimation

# Why Choose Stan?

- **Expressive**
  - Stan is a full imperative programming language
  - continuously differentiable log densities
- **Robust**
  - usually works; signals when it doesn't
- **Efficient**
  - effective sample size / time (i.e., information)
  - multi-core and GPU code complete on branches
- **Ongoing open source development**
- **Community support!**

# Intro
# Probability

# Probability is Epistemic

- **John Stuart Mill** (*Logic* 1882, Part III, Ch. 2):
  - ... the probability of an event is not a quality of the event itself, but a mere name for the degree of ground which we, or some one else, have for expecting it.
  - Every event is **in itself certain**, not probable; if we knew all, we should either know positively that it will happen, or positively that it will not.
  - ... its probability to us means the **degree of expectation of its occurrence**, which we are warranted in entertaining by our present evidence.
- Probabilities **quantify uncertainty**
- Statistical reasoning is **counterfactual**

# Random Variables

- Random variables are the currency of probability theory
- Random variables typically take numbers as values
- Imagine a bin filled with balls represneting the way the world might be
- A ball records the value of every random variable
- Examples
  - the sum of the three best among a roll of four dice (d6)
  - time before the next traffic accident on a given highway
  - prevalence of a disease in a population

# Bayesian Data Analysis

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

Gelman et al., *Bayesian Data Analysis*, 3rd edition, 2013

# Bayesian Methodology

- Set up **full probability model**
  - for **all** observable & unobservable quantities
  - consistent w. problem knowledge & data collection
- **Condition** on observed data (where Stan comes in!)
  - to caclulate posterior probability of unobserved quantities (e.g., parameters, predictions, missing data)
- **Evaluate**
  - model fit and implications of posterior
- **Repeat** as necessary

*Ibid.*

# Properties of Bayesian Inference

- Explores full range of parameters consistent with prior info and data*
  - * if such agreement is possible
  - Stan automates this procedure with diagnostics
- Inferences can be plugged in directly for
  - parameter estimates minimizing expected error
  - predictions for future outcomes with uncertainty
  - event probability updates conditioned on data
  - risk assesment / decision analysis conditioned on uncertainty

# Where do Models Come from?

- Sometimes model comes first, based on substantive considerations
  - toxicology, economics, ecology, physics, . . .
- Sometimes model chosen based on data collection
  - traditional statistics of surveys and experiments
- Other times the data comes first
  - observational studies, meta-analysis, . . .
- Usually its a mix

# Model Checking

- Do the inferences make sense?
  - are parameter values consistent with model's prior?
  - does simulating from parameter values produce reasoable fake data?
  - are marginal predictions consistent with the data?
- Do predictions and event probabilities for new data make sense?
- ***Not***: Is the model true?
- ***Not***: What is Pr[model is true]?
- ***Not***: Can we "reject" the model?

# Model Improvement

- Expanding the model
  - hierarchical and multilevel structure . . .
  - more flexible distributions (overdispersion, covariance)
  - more structure (geospatial, time series)
  - more modeling of measurement methods and errors
  - . . .
- Including more data
  - breadth (more predictors or kinds of observations)
  - depth (more observations)

# Notation for Basic Quantities

- **Basic Quantities**
  - $y$: observed data
  - $\theta$: parameters (and other unobserved quantities)
  - $x$: constants, predictors for conditional (aka "discriminative") models
- **Basic Predictive Quantities**
  - $\tilde{y}$: unknown, potentially observable quantities
  - $\tilde{x}$: constants, predictors for unknown quantities

# Naming Conventions

- **Joint**: $p(y, \theta)$
- **Sampling / Likelihood**: $p(y|\theta)$
  - Sampling is function of $y$ with $\theta$ fixed (prob function)
  - Likelihood is function of $\theta$ with $y$ fixed (*not* prob function)
- **Prior**: $p(\theta)$
- **Posterior**: $p(\theta|y)$
- **Data Marginal (Evidence)**: $p(y)$
- **Posterior Predictive**: $p(\tilde{y}|y)$

# Bayes's Rule for Posterior

$$
\begin{aligned}
p(\theta|y) &= \frac{p(y,\theta)}{p(y)} && \text{[def of conditional]} \\
&= \frac{p(y|\theta)\,p(\theta)}{p(y)} && \text{[chain rule]} \\
&= \frac{p(y|\theta)\,p(\theta)}{\int_\Theta p(y,\theta')\;d\theta'} && \text{[law of total prob]} \\
&= \frac{p(y|\theta)\,p(\theta)}{\int_\Theta p(y|\theta')\,p(\theta')\;d\theta'} && \text{[chain rule]}
\end{aligned}
$$

- *Inversion:* Final result depends only on sampling distribution (likelihood) $p(y|\theta)$ and prior $p(\theta)$

# Bayes's Rule up to Proportion

- If data $y$ is fixed, then

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

- Posterior proportional to likelihood times prior
- Equivalently, posterior proportional to joint
- The nasty integral for data marginal $p(y)$ goes away

# Posterior Predictive Distribution

- Predict new data $\tilde{y}$ based on observed data $y$
- Marginalize parameters $\theta$ out of posterior and likelihood

$$\begin{aligned} p(\tilde{y} \mid y) &= \mathbb{E}[p(\tilde{y}|\theta) \mid Y = y] \\ &= \int p(\tilde{y}|\theta)\, p(\theta|y)\, d\theta. \end{aligned}$$

- Weights predictions $p(\tilde{y}|\theta)$, by posterior $p(\theta|y)$
- Integral notation shorthand for sums and/or integrals

# Posterior Event Probabilities

- Recall that an event $A$ is a collection of outcomes
- So $A$ may be defined by an indicator $f$ on parameters

$$f(\theta) = \begin{cases} 1 & \text{if } \theta \in A \\ 0 & \text{if } \theta \notin A \end{cases}$$

  - $f(\theta) = \mathrm{I}(\theta_1 > \theta_2)$ for $\Pr[\theta_1 > \theta_2 \mid y]$,
  - $f(\theta) = \mathrm{I}(\theta \in (0.50, 0.52)$ for $\Pr[\theta \in (0.50, 0.52) \mid y]$

- Defined by posterior expectation of indicator $f(\theta)$

$$\Pr[A \mid y] \;=\; \mathbb{E}[f(\theta) \mid y] \;=\; \int_{\Theta} f(\theta)\, p(\theta|y)\, d\theta.$$

# Repeated Binary Trials

# Repeated Binary Trial Model

- **Data**
  - $N \in \{0, 1, \ldots\}$: number of trials (constant)
  - $y_n \in \{0, 1\}$: trial $n$ success (known, modeled data)
- **Parameter**
  - $\theta \in [0, 1]$ : chance of success (unknown)
- **Prior**
  - $p(\theta) \;=\; \text{Uniform}(\theta \mid 0, 1) = 1$
- **Likelihood**
  - $p(y \mid \theta) \;=\; \prod_{n=1}^{N} \text{Bernoulli}(y_n \mid \theta) \;=\; \prod_{n=1}^{N} \theta^{y_n} \, (1 - \theta)^{1-y_n}$
- **Posterior**
  - $p(\theta \mid y) \propto p(\theta) \, p(y \mid \theta)$

# Stan Program

```
data {
  int<lower=0> N;                     // number of trials
  int<lower=0, upper=1> y[N];         // success on trial n
}
parameters {
  real<lower=0, upper=1> theta;       // chance of success
}
model {
  theta ~ uniform(0, 1);              // prior
  y ~ bernoulli(theta);               // likelihood
}
```

# A Stan Program...

- defines log (posterior) density up to constant, so...
- equivalent to define log density directly:

```
model {
  target += 0;
  for (n in 1:N)
    target += log(y[n] ? theta : (1 - theta));
}
```

- equivalent to drop constant prior and vectorize likelihood:

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

# R: Simulate Data

- Generate data

```
> theta <- 0.30;
> N <- 20;
> y <- rbinom(N, 1, 0.3);

> y

 [1] 1 1 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0 0 0 1
```

- Calculate MLE as sample mean from data

```
> sum(y) / N

[1] 0.4
```

# RStan: Bayesian Posterior

```
> library(rstan);

> fit <- stan("bern.stan",
              data = list(y = y, N = N));

> print(fit, probs=c(0.1, 0.9));

Inference for Stan model: bern.
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    10%    90% n_eff Rhat
theta   0.41    0.00 0.10   0.28   0.55  1580    1
```

# RStan: Posterior Sample

```
> theta_draws <- extract(fit)$theta;
> mean(theta_draws);

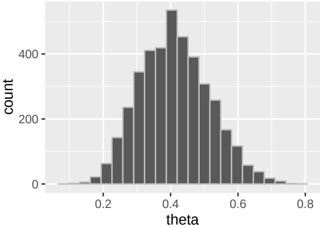
[1] 0.4128373

> quantile(theta_draws, probs=c(0.10, 0.90));

      10%       90%
0.2830349 0.5496858
```

# Marginal Posterior Histograms

```
theta_draws_df <- data.frame(list(theta = theta_draws));
plot <-
  ggplot(theta_draws_df, aes(x = theta)) +
  geom_histogram(bins=20, color = "gray");
plot;
```




- Displays the full posterior *marginal* distribution $p(\theta \mid y)$

# RStan: MAP, penalized MLE

- Stan's optimization for estimation; two views:
  - max posterior mode, aka max a posteriori (MAP)
  - max penalized likelihood (MLE)

```
> library(rstan);
> N <- 5;
> y <- c(0,1,1,0,0);
> model <- stan_model("bernoulli.stan");
> mle <- optimizing(model, data=c("N", "y"));
...
> print(mle, digits=2)
$par                  $value  (log density)
theta                 [1] -3.4
  0.4
```

# Plug in Posterior Draws

- Extracting the posterior draws

```
> theta_draws <- extract(fit)$theta;
```

- Calculating posterior mean (estimator)

```
> mean(theta_draws);

[1] 0.4128373
```

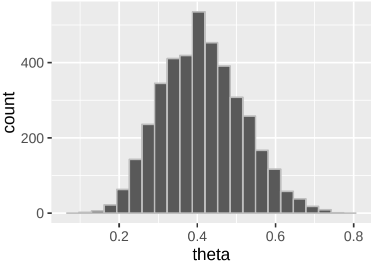
- Calculating posterior intervals

```
> quantile(theta_draws, probs=c(0.10, 0.90));

      10%       90%
0.2830349 0.5496858
```

# ggplot2: Plotting

```
theta_draws_df <- data.frame(list(theta = theta_draws));
plot <-
  ggplot(theta_draws_df, aes(x = theta)) +
  geom_histogram(bins=20, color = "gray");
plot;
```



# Default Priors and Vectorization

- All parameters are uniform by default
- Probability functions can be vectorized (more efficient)
- Thus

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

reduces to

```
y ~ bernoulli(theta);
```

## Real Example

# Male Birth Ratio

# Birth Rate by Sex

- **Laplace**'s data on live births in Paris from 1745–1770:

| *sex* | *live births* |
|---|---|
| female | 241 945 |
| male | 251 527 |

- **Question 1** (Estimation)
  What is the birth rate of boys vs. girls?
- **Question 2** (Event Probability)
  Is a boy more likely to be born than a girl?
- Bayes (1763) set up the "Bayesian" model
- Laplace (1781, 1786) solved for the posterior

# Bayes's Binomial Model

- Data
  - $y$: total number of male live births (251,527)
  - $N$ : total number of live births (493,472)
- Parameter
  - $\theta \in (0,1)$: proportion of male live births
- Likelihood

$$p(y|N,\theta) \;=\; \text{Binomial}(y|N,\theta) \;=\; \binom{N}{y}\theta^{y}(1-\theta)^{N-y}$$

- Prior

$$p(\theta) \;=\; \text{Uniform}(\theta \mid 0,1) \;=\; 1$$

# Calculating Laplace's Answers

```
transformed data {
  int male = 251527;
  int female = 241945;
}
parameters {
  real<lower=0, upper=1> theta;
}
model {
  male ~ binomial(male + female, theta);
}
generated quantities {
  int<lower=0, upper=1> theta_gt_half = (theta > 0.5);
}
```

# And the Answer is...

```
> fit <- stan("laplace.stan", iter=100000);
> print(fit, probs=c(0.005, 0.995), digits=3)

                    mean    0.5%   99.5%
theta               0.51   0.508   0.512
theta_gt_half       1.00   1.000   1.000
```

- Q1: $\theta$ is 99% certain to lie in $(0.508, 0.512)$
- Q2: Laplace "morally certain" boys more prevalent

## Example 2

# A-B Testing

# Bayesian "Fisher Exact Test"

- Suppose we observe the following data on handedness

| | *sinister* | *dexter* | TOTAL |
|---|---|---|---|
| *male* | 9 ($y_1$) | 43 | 52 ($N_1$) |
| *female* | 4 ($y_2$) | 44 | 48 ($N_2$) |

- Assume likelihoods Binomial$(y_k|N_k,\theta_k)$, uniform priors
- Are men more likely to be lefthanded?

$$
\begin{aligned}
\mathrm{Pr}[\theta_1 > \theta_2 \mid y, N] &= \int_\Theta \mathrm{I}[\theta_1 > \theta_2]\, p(\theta|y,N)\, d\theta \\
&\approx \frac{1}{M} \sum_{m=1}^{M} \mathrm{I}[\theta_1^{(m)} > \theta_2^{(m)}].
\end{aligned}
$$

# Stan Binomial Comparison

```
data {
  int y[2];
  int N[2];
}
parameters {
  vector<lower=0,upper=1> theta[2];
}
model {
  y ~ binomial(N, theta);
}
generated quantities {
  real boys_minus_girls = theta[1] - theta[2];
  int boys_gt_girls = theta[1] > theta[2];
}
```

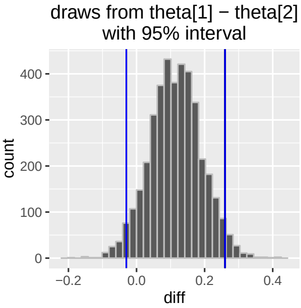
# Binomial Comparison Results

| | mean | 2.5% | 97.5% |
|---|---|---|---|
| theta[1] | 0.22 | 0.12 | 0.35 |
| theta[2] | 0.11 | 0.04 | 0.21 |
| boys_minus_girls | 0.12 | -0.03 | 0.26 |
| boys_gt_girls | 0.93 | 0.00 | 1.00 |

- $\Pr[\theta_1 > \theta_2 \mid y] \approx 0.93$
- $\Pr\left[(\theta_1 - \theta_2) \in (-0.03, 0.26) \mid y\right] = 95\%$

# Visualizing Posterior Difference

- Plot of posterior difference, $p(\theta_1 - \theta_2 \mid y, N)$ (men - women)




- Vertical bars: central 95% posterior interval $(-0.03, 0.26)$

# What is Stan?

# What is Stan?

- Stan is an **imperative** probabilistic programming language
  - cf., BUGS: declarative; Church: functional; Figaro: object-oriented
- Stan **program**
  - declares data and (constrained) parameter variables
  - defines log posterior (or penalized likelihood)
- Stan **inference**
  - MCMC for full Bayesian inference
  - VB for approximate Bayesian inference
  - MLE for penalized maximum likelihood estimation

# Platforms and Interfaces

- **Platforms**: Linux, Mac OS X, Windows
- **C++ API**: portable, standards compliant (C++11)
- **Interfaces**
  - **CmdStan**: Command-line or shell interface (direct executable)
  - **RStan**: R interface (Rcpp in memory)
  - **PyStan**: Python interface (Cython in memory)
  - **MatlabStan**: MATLAB interface (external process)
  - **Stan.jl**: Julia interface (external process)
  - **StataStan**: Stata interface (external process)
  - **MathematicaStan**: Stata interface (external process)

# Higher-Level Interfaces

- **R Interfaces**
  - **RStanArm**: regression modeling with R expressions
  - **ShinyStan**: web-based posterior visualization, exploration
  - **Loo**: approximate leave-one-out cross-validation
- **Jupyter Containers**
  - **Docker** versions for R, Python, Julia
  - **SageMath**: free online server (R)
- From others
  - **Prohet** (Facebook): time-series analysis (R and Python)
  - **brms** (Bürkner): regression modeling with R expressions

- **rethinking** (McElreath): simplified Stan embedded in R

# Who's Using Stan?

- 2500+ **users group** registrations; 20,000+ **downloads** (per version just in Rstudio); 1000+ Google scholar citations
- **Biological sciences**: clinical drug trials, entomology, pharmacology, toxicology, botany, neurology, genomics, agriculture, botany, fisheries, genomics, cellular biology, epidemiology, population ecology, neurology
- **Physical sciences**: astrophysics, particle physics, molecular biology, oceanography, climatology, biogeochemistry, materials science
- **Social sciences**: econometrics (macro and micro), population dynamics, cognitive science, psycholinguistics, social networks, political science, survey sampling

- **Other**: materials engineering, finance, actuarial science, sports, public health, recommender systems, educational testing, fleet maintenance, sports

# Documentation

- *Stan User's Guide and Reference Manual*
  - 600+ pages
  - example models, modeling and programming advice
  - introduction to Bayesian and frequentist statistics
  - complete language specification and execution guide
  - descriptions of algorithms (NUTS, R-hat, n_eff)
  - guide to built-in distributions and functions
- Installation and getting started manuals by interface
  - RStan, PyStan, CmdStan, MatlabStan, Stan.jl, StataStan, MathematicaStan
- Many written and video tutorials by users and developers

# Model Sets Translated to Stan

- BUGS examples (most of all 3 volumes)
- Gelman and Hill (2009) *Data Analysis Using Regression and Multilevel/Hierarchical Models*.
- Wagenmakers and Lee (2014) *Bayesian Cognitive Modeling*.
- Kéry and Schaub (2014) *Bayesian Population Analysis Using WinBUGS*.
- Kruschke (2014) *Doing Bayesian Data Analysis*.

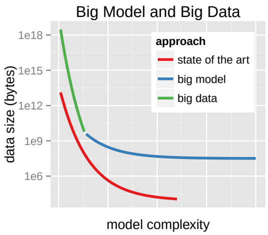
# Books about Stan

- Gelman and Hill (2018) *Regression and Other Stories*. Cambridge.
- Hilbe, de Souza, and Ishida (2017) *Bayesian Models for Astrophysical Data Using R, JAGS, Python, and Stan*. Cambridge.
- Matsuura (2016) *Bayesian Statistical Modeling Using Stan and R*. Kyoritsu. (Japanese)
- Faraway (2016) *Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric Regression Models*, 2nd Edition. CRC.
- McElreath (2016) *Statistical Rethinking: A Bayesian course with R and Stan*. CRC.
- Korner-Nievergelt et al. (2015) *Bayesian Data Analysis in Ecology Using Linear Models with R, BUGS, and Stan*. Academic Press.

- Kruschke (2014) *Doing Bayesian Data Analysis, Second Edition: A Tutorial with R, JAGS, and Stan*. Academic Press.

- Gelman et al. (2013) *Bayesian Data Analysis*, 3rd Edition. CRC.

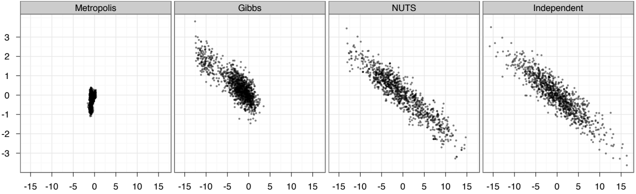
# Scaling and Evaluation




- Types of Scaling: data, parameters, **models**
- Time to converge and per effective sample size:

  0.5–∞ times faster than BUGS & JAGS
- Memory usage: 1–10% of BUGS & JAGS

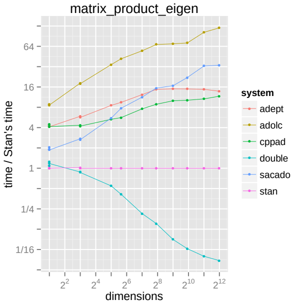
# NUTS vs. Gibbs and Metropolis

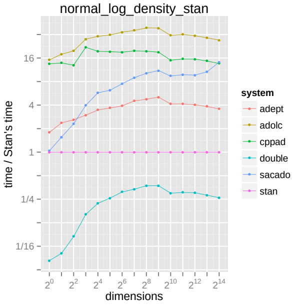



- Two dimensions of highly correlated 250-dim normal
- **1,000,000 draws** from Metropolis and Gibbs (thin to 1000)
- **1000 draws** from NUTS; 1000 independent draws

# Stan's Autodiff vs. Alternatives

- Among **C++ open-source** offerings: Stan is **fastest** (for gradients), **most general** (functions supported), and **most easily extensible** (simple OO)



# Stan Language

# Stan is a Programming Language

- **Not** a graphical specification language like BUGS or JAGS
- Stan is a Turing-complete imperative programming language for specifying differentiable log densities
  - reassignable local variables and scoping
  - full conditionals and loops
  - functions (including recursion)
- With automatic "black-box" inference on top (though even that is tunable)
- Programs computing same thing may have different efficiency

# Basic Program Blocks

- **`data`** (once)
  - *content*: declare data types, sizes, and constraints
  - *execute*: read from data source, validate constraints
- **`parameters`** (every log prob eval)
  - *content*: declare parameter types, sizes, and constraints
  - *execute*: transform to constrained, Jacobian
- **`model`** (every log prob eval)
  - *content*: statements definining posterior density
  - *execute*: execute statements

# Derived Variable Blocks

- **`transformed data`** (once after data)
  - *content*: declare and define transformed data variables
  - *execute*: execute definition statements, validate constraints
- **`transformed parameters`** (every log prob eval)
  - *content*: declare and define transformed parameter vars
  - *execute*: execute definition statements, validate constraints
- **`generated quantities`** (once per draw, `double` type)
  - *content*: declare and define generated quantity variables; includes pseudo-random number generators
    (for posterior predictions, event probabilities, decision making)
  - *execute*: execute definition statements, validate constraints

# Model: Read and Transform Data

- Only done once for optimization or sampling (per chain)
- Read data
  - read data variables from memory or file stream
  - validate data
- Generate transformed data
  - execute transformed data statements
  - validate variable constraints when done

# Model: Log Density

- *Given* parameter values on unconstrained scale
- Builds expression graph for log density (start at 0)
- Inverse transform parameters to constrained scale
  - constraints involve non-linear transforms
  - e.g., positive constrained $x$ to unconstrained $y = \log x$
- account for curvature in change of variables
  - e.g., unconstrained $y$ to positive $x = \log^{-1}(y) = \exp(y)$
  - e.g., add log Jacobian determinant, $\log |\frac{d}{dy} \exp(y)| = y$
- Execute model block statements to increment log density

# Model: Log Density Gradient

- Log density evaluation builds up expression graph
  - templated overloads of functions and operators
  - efficient arena-based memory management
- Compute gradient in backward pass on expression graph
  - propagate partial derivatives via chain rule
  - work backwards from final log density to parameters
  - dynamic programming for shared subexpressions
- Linear multiple of time to evalue log density

# Model: Generated Quantities

- **Given** parameter values
- Once per iteration (not once per leapfrog step)
- May involve (pseudo) random-number generation
  - Executed generated quantity statements
  - Validate values satisfy constraints
- Typically used for
  - Event probability estimation
  - Predictive posterior estimation
- Efficient because evaluated with `double` types (no autodiff)

# Variable Transforms

- Code HMC and optimization with $\mathbb{R}^n$ **support**
- Transform constrained parameters to unconstrained
  - lower (upper) bound: offset (negated) log transform
  - lower and upper bound: scaled, offset logit transform
  - simplex: centered, stick-breaking logit transform
  - ordered: free first element, log transform offsets
  - unit length: spherical coordinates
  - covariance matrix: Cholesky factor positive diagonal
  - correlation matrix: rows unit length via quadratic stick-breaking

# Variable Transforms (cont.)

- Inverse transform from unconstrained $\mathbb{R}^n$
- Evaluate log probability in model block on natural scale
- Optionally adjust log probability for change of variables
  - adjustment for MCMC and variational, not MLE
  - add log determinant of inverse transform Jacobian
  - automatically differentiable

# Variable and Expression Types

Variables and expressions are **strongly, statically typed**.

- **Primitive**: `int`, `real`
- **Matrix**: `matrix[M,N]`, `vector[M]`, `row_vector[N]`
- **Bounded**: primitive or matrix, with `<lower=L>`, `<upper=U>`, `<lower=L,upper=U>`
- **Constrained Vectors**: `simplex[K]`, `ordered[N]`, `positive_ordered[N]`, `unit_length[N]`
- **Constrained Matrices**: `cov_matrix[K]`, `corr_matrix[K]`, `cholesky_factor_cov[M,N]`, `cholesky_factor_corr[K]`
- **Arrays:** of any type (and dimensionality)

# Integers vs. Reals

- Different types (conflated in BUGS, JAGS, and R)
- Distributions and assignments care
- Integers may be assigned to reals but not vice-versa
- Reals have not-a-number, and positive and negative infinity
- Integers single-precision up to +/- 2 billion
- Integer division rounds (Stan provides warning)
- Real arithmetic is inexact and reals should not be (usually) compared with ==

# Arrays vs. Vectors & Matrices

- Stan separates arrays, matrices, vectors, row vectors
- Which to use?
- Arrays allow most efficient access (no copying)
- Arrays stored first-index major (i.e., 2D are row major)
- Vectors and matrices required for matrix and linear algebra functions
- Matrices stored column-major
- Are not assignable to each other, but there are conversion functions

# Logical Operators

| *Op.* | *Prec.* | *Assoc.* | *Placement* | *Description* |
|---|---|---|---|---|
| \|\| | 9 | left | binary infix | logical or |
| && | 8 | left | binary infix | logical and |
| == | 7 | left | binary infix | equality |
| != | 7 | left | binary infix | inequality |
| < | 6 | left | binary infix | less than |
| <= | 6 | left | binary infix | less than or equal |
| > | 6 | left | binary infix | greater than |
| >= | 6 | left | binary infix | greater than or equal |

# Arithmetic and Matrix Operators

| *Op.* | *Prec.* | *Assoc.* | *Placement* | *Description* |
|---|---|---|---|---|
| `+` | 5 | left | binary infix | addition |
| `-` | 5 | left | binary infix | subtraction |
| `*` | 4 | left | binary infix | multiplication |
| `/` | 4 | left | binary infix | (right) division |
| `\` | 3 | left | binary infix | left division |
| `.*` | 2 | left | binary infix | elementwise multiplication |
| `./` | 2 | left | binary infix | elementwise division |
| `!` | 1 | n/a | unary prefix | logical negation |
| `-` | 1 | n/a | unary prefix | negation |
| `+` | 1 | n/a | unary prefix | promotion (no-op in Stan) |
| `^` | 2 | right | binary infix | exponentiation |
| `'` | 0 | n/a | unary postfix | transposition |
| `()` | 0 | n/a | prefix, wrap | function application |
| `[]` | 0 | left | prefix, wrap | array, matrix indexing |

# Assignment Operators

| *Op.* | *Description* |
|---|---|
| `=` | assignment |
| `+=` | compound add and assign |
| `-=` | compound subtract and assign |
| `*=` | compound mulitply and assign |
| `/=` | compound divide and assign |
| `.*=` | compound elementwise mulitply and assign |
| `./=` | compound elementwise divide and assign |

- these work with all relevant matrix types
  - e.g., `matrix *= matrix;`

# Built-in Math Functions

- All built-in **C++ functions and operators**

  C math, TR1, C++11, including all trig, pow, and special log1m, erf, erfc, fma, atan2, etc.

- Extensive library of **statistical functions**

  e.g., softmax, log gamma and digamma functions, beta functions, Bessel functions of first and second kind, etc.

- Efficient, arithmetically stable **compound functions**

  e.g., multiply log, log sum of exponentials, log inverse logit

# Built-in Matrix Functions

- **Basic arithmetic**: all arithmetic operators
- **Elementwise arithmetic**: vectorized operations
- **Solvers**: matrix division, (log) determinant, inverse
- **Decompositions**: QR, Eigenvalues and Eigenvectors, Cholesky factorization, singular value decomposition
- **Compound Operations**: quadratic forms, variance scaling, etc.
- **Ordering, Slicing, Broadcasting**: sort, rank, block, rep
- **Reductions**: sum, product, norms
- **Specializations**: triangular, positive-definite,

# Statements

- **Sampling**: `y ~ normal(mu,sigma)` (increments log probability)
- **Log probability**: increment_log_prob(lp);
- **Assignment**: `y_hat <- x * beta`;
- **For loop**: `for (n in 1:N) ...`
- **While loop**: `while (cond) ...`
- **Conditional**: `if (cond) ...; else if (cond) ...; else ...;`
- **Block**: `{ ... }` (allows local variables)
- **Print**: `print("theta=",theta);`

- **Reject**: `reject("arg to foo must be positive, found y=", y);`
- **Break, Continue**: `break`, `continue`

# "Sampling" Increments Log Prob

- A Stan program defines a log posterior
  - typically through log joint and Bayes's rule
- Sampling statements are just "syntactic sugar"
- A shorthand for incrementing the log posterior
- The following define the same* posterior
  - `y ~ poisson(lambda);`
  - `increment_log_prob(poisson_log(y, lamda));`
- * up to a constant
- Sampling statement drops constant terms

# Local Variable Scope Blocks

- ```
  y ~ bernoulli(theta);
  ```

  is more efficient with sufficient statistics

  ```
  {
    real sum_y;  // local variable
    sum_y <- 0;
    for (n in 1:N)
      sum_y <- a + y[n];   // reassignment
    sum_y ~ binomial(N, theta);
  }
  ```

- Simpler, but roughly same efficiency:

  ```
  sum(y) ~ binomial(N, theta);
  ```
  ```

# User-Defined Functions

- **functions** (compiled with model)
  - *content*: declare and define general (recursive) functions (use them elsewhere in program)
  - *execute*: compile with model

- Example

```
functions {

  real relative_difference(real u, real v) {
    return 2 * fabs(u - v) / (fabs(u) + fabs(v));
  }

}
```

# Special User-Defined Functions

- When declared with appropriate naming, user-defined functions may
  - be used in sampling statements: `real` return and suffix `_lpdf` or `_lpmf`
  - use RNGs: suffix `_rng`
  - use target accumulator: suffix `_lp`

# Differential Equation Solver

- System expressed as function
  - given state ($y$) time ($t$), parameters ($\theta$), and data ($x$)
  - return derivatives ($\partial y / \partial t$) of state w.r.t. time
- Simple harmonic oscillator diff eq

```
real[] sho(real t,          // time
           real[] y,        // system state
           real[] theta,    // params
           real[] x_r,      // real data
           int[] x_i) {     // int data
  real dydt[2];
  dydt[1] <- y[2];
  dydt[2] <- -y[1] - theta[1] * y[2];
  return dydt;
}
```

# Differential Equation Solver (cont.)

- Solution via functional, given initial state (y0), initial time (`t0`), desired solution times (`ts`)

```
mu_y <- integrate_ode(sho, y0, t0, ts, theta, x_r, x_i);
```

- Use noisy measurements of $y$ to estimate $\theta$

```
y ~ normal(mu_y, sigma);
```

  - Pharmacokinetics/pharmacodynamics (PK/PD),
  - soil carbon respiration with biomass input and breakdown

# Built-in Diff Eq Solvers

- **Non-stiff solver**: Runge-Kutta 4th/5th order (RK45)
- **Stiff solver**: backward-differentiation formula (BDF)
  - slower
  - more robust for derivatives of different scales or high curvature
- specified by suffix `_bdf` or `_rk45`

# Diff Eq Derivatives

- User defines system $\frac{\partial}{\partial t} y$
- Need derivatives of solution $y$ w.r.t. parameters $\theta$
- Couple derivatives of system w.r.t. parameters

$$\left( \frac{\partial}{\partial t} y, \;\; \frac{\partial}{\partial t} \frac{\partial}{\partial \theta} y \right)$$

- Calculate coupled system via nested autodiff of second term

$$\frac{\partial}{\partial t} \frac{\partial}{\partial \theta} y \; = \; \frac{\partial}{\partial \theta} \frac{\partial}{\partial t} y.$$

# Distribution Library

- Each distribution has
  - log density or mass function
  - cumulative distribution functions, plus complementary versions, plus log scale
  - Pseudo-random number generators
- Alternative parameterizations

  (e.g., Cholesky-based multi-normal, log-scale Poisson, logit-scale Bernoulli)
- New multivariate correlation matrix density: LKJ

  degrees of freedom controls shrinkage to (expansion from) unit matrix

# Print and Reject

- Print statements are for **debugging**
  - printed every log prob evaluation
  - print values in the middle of programs
  - check when log density becomes undefined
  - can embed in conditionals
- Reject statements are for **error checking**
  - typically function argument checks
  - cause a rejection of current state (0 density)

# Prob Function Vectorization

- Stan's probability functions are vectorized for speed
  - removes repeated computations (e.g., $-\log \sigma$ in normal)
  - reduces size of expression graph for differentation
- Consider: `y ~ normal(mu, sigma);`
- Each of `y`, `mu`, and `sigma` may be any of
  - scalars (integer or real)
  - vectors (row or column)
  - 1D arrays
- All dimensions must be scalars or having matching sizes
- Scalars are broadcast (repeated)

# Parsing and Compilation

- Stan code **parsed** to abstract syntax tree (AST)
  (Boost Spirit Qi, recursive descent, lazy semantic actions)
- C++ model class **code generation** from AST
  (Boost Variant)
- C++ code **compilation**
- **Dynamic linking** for RStan, PyStan

# What Stan Does

# Full Bayes: No-U-Turn Sampler

- Adaptive **Hamiltonian Monte Carlo** (HMC)
  - **Potential Energy**: negative log posterior
  - **Kinetic Energy**: random standard normal per iteration
  - **Multinomial**: draw along trajectory
- Adaptation **during warmup**
  - step size adapted to target total acceptance rate
  - mass matrix (scale/rotation) estimated with regularization
- Adaptation **during sampling**
  - simulate forward and backward in time until U-turn
  - **slice sample** along path

(Hoffman and Gelman 2011, 2014)

# Animation

- animated GIFs (easy to produce!)

# Why HMC?

- Gibbs and Metropolis are both random walk **diffusions**
  - $\mathcal{O}(N^2)$ in $N$ dimensions to move across posterior
  - constant factor increases with correlation
- HMC uses gradient information of log posterior
  - gradient defines vector field along which trajectory **flows**
- Eliminates random walk behavior
  - $\mathcal{O}(N^{5/4})$
  - **lower constant factor** because less sensitive to correlation

# Posterior Inference

- Generated quantities block for **inference**:
  predictions, decisions, and event probabilities
- **Extractors** for samples in RStan and PyStan
- Coda-like **posterior summary**
  - posterior mean w. MCMC std. error, std. dev., quantiles
  - split-$\hat{R}$ multi-chain convergence diagnostic (Gelman/Rubin)
  - multi-chain effective sample size estimation (FFT algorithm)
- Model comparison with **WAIC**
  - in-sample approximation to cross-validation

# MAP / Penalized MLE

- Posterior **mode finding** via L-BFGS optimization
  (uses model gradient, efficiently approximates Hessian)
- **Disables Jacobians** for parameter inverse transforms
- Models, data, initialization as in MCMC
- **Standard errors** on unconstrained scale
  (estimated using curvature of penalized log likelihood function
- **Very Near Future**
  - Standard errors **on constrained scale**)
    (sample unconstrained approximation and inverse transform)

# “Black Box” Variational Inference

- **Black box** so can fit any Stan model
- Multivariate **normal approx to unconstrained** posterior
  - covariance: diagonal mean-field or full rank
  - not Laplace approx — around posterior mean, not mode
  - transformed back to constrained space (built-in Jacobians)
- Stochastic **gradient-descent** optimization
  - ELBO gradient estimated via Monte Carlo + autdiff
- Returns **approximate posterior** mean / covariance
- Returns **sample** transformed to constrained space

# Stan as a Research Tool

- Stan can be used to **explore algorithms**
- Models transformed to **unconstrained support** on $\mathbb{R}^n$
- Once a model is compiled, have
  - **log probability, gradient, and Hessian**
  - data I/O and parameter initialization
  - model provides variable names and dimensionalities
  - transforms to and from constrained representation
    (with or without Jacobian)

# What's Next

- Distributed likelihoods: **multi-core** (MPI)
- Big matrix operations: **GPU** (OpenCL)
- Streaming data:  myemphstochastic variational inference
- Distributed data: "black box" **expectation propagation**
- Approximations, visualizations, posterior analysis tools
- **Coursera** specialization (Bob Carpenter & Andrew Gelman) mid-2018

# StanCon 2018

- 10–12 January 2018
- Asilomar (Pacific Grove, CA — 2 hrs south of SFO)
- Early registration still available
- Tutorials by Stan developers at all levels
- 6 keynotes from science and business
- Breakout “unconference” sessions

- http://mc-stan.org/events/stancon2018/