Stan

a Probabilistic Programming Language

Core Development Team (in order of joining):

Andrew Gelman, **Bob Carpenter**, Matt Hoffman, Daniel Lee,
Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo,
Peter Li, Allen Riddell, Marco Inacio, Jeffrey Arnold,
Mitzi Morris, Rob Trangucci, Rob Goedman, Brian Lau,
Jonah Sol Gabry, Alp Kucukelbir, Robert L. Grant,
Dustin Tran, Alp Kucukelbir, **Krzysztof Sakrejda**,
Aki Vehtari, Rayleigh Lei, Sebastian Weber

Stan 2.9.0 (April 2016) http://mc-stan.org

Stan

Get the Slides

`http://mc-stan.org/workshops/ness2016`

Example I

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

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**
  - $p(y \mid \theta) \; = \; \prod_{n=1}^{N} \text{Bernoulli}(y_n \mid \theta) \; = \; \prod_{n=1}^{N} \theta^{y_n} \, (1 - \theta)^{1 - y_n}$

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
  for (n in 1:N)
    y[n] ~ bernoulli(theta);        // likelihood
}
```

A Stan Program

- Defines log (posterior) density up to constant, so...
- Equivalent to define log density directly:

```
model {
  increment_log_prob(0);
  for (n in 1:N)
    increment_log_prob(log(theta^y[n])
                               * (1 - theta)^(1 - y[n])));
}
```

- Equivalent to (a) drop constant prior, (b) vectorize likelihood:

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

Simulating Repeated Binary Trials

- R Code: `rbinom(10, N, 0.3)`
  - **N = 10** trials (10% to 50% success rate)

    2 2 1 3 3 2 3 2 2 5
  - **N = 100** trials (27% to 34% success rate)

    29 34 27 31 25 31 27 29 32 26
  - **N = 1000** trials (29% to 32% success rate)

    291 297 289 322 305 296 294 297 314 292
  - **N = 10,000** trials (29.5% to 30.7% success rate)

    3014 3031 3017 2886 2995 2944 3067 3069 3051 3068
- Deviation goes down at rate: $\mathcal{O}(1/\sqrt{N})$

R: Simulate 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

> sum(y) / N

[1] 0.4
```

RStan: Fit

```
> library(rstan);

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

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

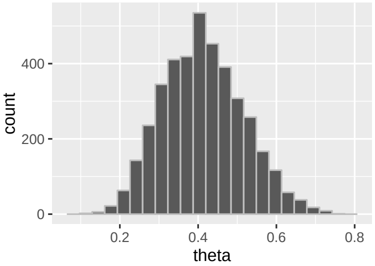
```
Inference for Stan model: bern.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000,
total post-warmup draws=4000.

        mean se_mean   sd    10%    90% n_eff Rhat
theta   0.41    0.00 0.10   0.28   0.55  1580    1
lp__  -15.40    0.02 0.71 -16.26 -14.89  1557    1

Samples drawn using NUTS(diag_e) at Thu Apr 21 19:38:16 2016.
```

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
```

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;
```



Binomial Distribution

- Don't know order of births, only total.
- If $y_1, \ldots, y_N \sim \text{Bernoulli}(\theta)$,

  then $(y_1 + \cdots + y_N) \sim \text{Binomial}(N, \theta)$
- The analytic form is

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

  where the binomial coefficient normalizes for permutations (i.e., which subset of n has $y_n = 1$),

$$\binom{N}{y} = \frac{N!}{y!\,(N-y)!}$$

Calculating Laplace’s Answers

```
transformed data {
  int male;
  int female;
  male <- 251527;
  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_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: θ is 99% certain to lie in $(0.508, 0.512)$
- Q2: Laplace "morally certain" boys more prevalent

Parameter Estimates

- Estimate **probability** that a parameter value in interval, e.g.,

$$\Pr[\theta \in (0.508, 0.512) \mid y]$$

- Conditions on observed data y

- **Bayesian parameter estimates are probabilistic**

Event Probabilities

- Random variable defined by **indicator** function

```
theta_gt_half <- (theta > 0.5);
```

- Indicators are random variables
  - with boolean (0 or 1) values
  - defined in terms of parameters (and data)
- For Laplace’s problem, calculus shows

$$\Pr[\theta \leq 0.5 \mid y] \approx 10^{-42}$$

- **Event probabilities are expectations of indicators**

Part I
Bayesian Inference

Observations, Counterfactuals, and Random Variables

- Assume we observe data $y = y_1, \ldots, y_N$
- Statistical modeling assumes even though y is observed, the values could have been different
- John Stuart Mill first characterized this **counterfactual** nature of statistical modeling in:

  A System of Logic, Ratiocinative and Inductive (1843)
- In measure-theoretic language, y modeled as a **random variable**

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
  - to caclulate posterior probability of unobserved quantities (e.g., parameters, predictions, missing data)
- **Evaluate**
  - model fit and implications of posterior
- **Repeat** as necessary

Ibid.

Where do Models Come from?

- Sometimes model comes first, based on substantive considerations
  - toxicology, economics, ecology, . . .
- 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

(Donald) Rubin’s Philosophy

- All statistics is inference about missing data
- Question 1: What would you do if you had all the data?
- Question 2: What were you doing before you had any data?

(as relayed in course notes by Andrew Gelman)

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)

Using Bayesian Inference

- Finds parameters consistent with prior info and data*
    - * if such agreement is possible
- Automatically includes uncertainty and variability
- Inferences can be plugged in directly
    - risk assesment
    - decision analysis

Notation for Basic Quantities

- **Basic Quantities**
  - y: observed data
  - θ: 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 θ fixed (prob function)
  - Likelihood is function of θ 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 out parameters from posterior

$$p(\tilde{y}|y) \;=\; \int_{\Theta} p(\tilde{y}|\theta)\, p(\theta|y)\, d\theta.$$

- Averages predictions $p(\tilde{y}|\theta)$, weight by posterior $p(\theta|y)$
  - $\Theta = \{\theta \mid p(\theta|y) > 0\}$ is support of $p(\theta|y)$
- Allows continuous, discrete, or mixed parameters
  - integral notation shorthand for sums and/or integrals

Event Probabilities

- Recall that an event A is a collection of outcomes
- Suppose event A is determined by indicator on parameters

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

- e.g., $f(\theta) = \mathrm{I}(\theta_1 > \theta_2)$ for $\Pr[\theta_1 > \theta_2 \mid y]$
- Bayesian event probabilities calculate posterior mass

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

- Not frequentist, because involves parameter probabilities

Detour: Beta Distribution

- For parameters $\alpha, \beta > 0$ and $\theta \in (0, 1)$,

$$\text{Beta}(\theta|\alpha, \beta) \;\propto\; \theta^{\alpha-1}\,(1-\theta)^{\beta-1}$$

- Euler's Beta function is used to normalize,

$$\text{B}(\alpha, \beta) \;=\; \int_0^1 u^{\alpha-1}(1-u)^{\beta-1}du \;=\; \frac{\Gamma(\alpha)\,\Gamma(\beta)}{\Gamma(\alpha+\beta)}$$

  so that

$$\text{Beta}(\theta|\alpha, \beta) \;=\; \frac{1}{\text{B}(\alpha, \beta)}\,\theta^{\alpha-1}\,(1-\theta)^{\beta-1}$$

- Note: $\text{Beta}(\theta|1, 1) = \text{Uniform}(\theta|0, 1)$

- Note: $\Gamma()$ is continuous generalization of factorial

Laplace Turns the Crank

- From Bayes's rule, the posterior is

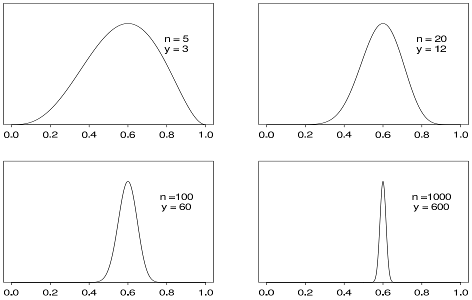
$$p(\theta|y, N) \;=\; \frac{\textsf{Binomial}(y|N, \theta)\,\textsf{Uniform}(\theta|0, 1)}{\int_{\Theta} \textsf{Binomial}(y|N, \theta')\, p(\theta')\; d\theta'}$$

- Laplace calculated the posterior analytically

$$p(\theta|y, N) \;=\; \textsf{Beta}(\theta \mid y + 1,\; N - y + 1).$$

Beta Distribution — Examples

- Unnormalized posterior density assuming uniform prior and y successes out of n trials (all with mean 0.6).



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

Estimation

- Posterior is Beta$(\theta \mid 1 + 241\,945,\ 1 + 251\,527)$
- Posterior mean:

$$\frac{1 + 241\,945}{1 + 241\,945 + 1 + 251\,527} \approx 0.490291\mathbf{3}$$

- Maximum likelihood estimate same as posterior mode (because of uniform prior)

$$\frac{241\,945}{241\,945 + 251\,527} \approx 0.490291\mathbf{2}$$

- As number of observations approaches ∞,
  MLE approaches posterior mean

Event Probability Inference

- What is probability that a male live birth is more likely than a female live birth?

$$\begin{aligned}
\Pr[\theta > 0.5] &= \int_{\Theta} \mathrm{I}[\theta > 0.5]\, p(\theta|y,N) d\theta \\
&= \int_{0.5}^{1} p(\theta|y,N) d\theta \\
&= 1 - F_{\theta|y,N}(0.5) \\
&\approx 10^{-42}
\end{aligned}$$

- $\mathrm{I}[\phi] = 1$ if condition ϕ is true and 0 otherwise.
- $F_{\theta|y,N}$ is posterior cumulative distribution function (cdf).

Mathematics vs. Simulation

- Luckily, we don't have to be as good at math as Laplace
- Nowadays, we calculate all these integrals by computer using tools like Stan

> If you wanted to do foundational research in statistics in the mid-twentieth century, you had to be bit of a mathematician, whether you wanted to or not. . . . if you want to do statistical research at the turn of the twenty-first century, you have to be a computer programmer.
>
> —from Andrew's blog

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?

$$\begin{aligned}\Pr[\theta_1 > \theta_2 \mid y, N] &= \int_{\Theta} \mathrm{I}[\theta_1 > \theta_2]\, p(\theta|y, N)\, d\theta \\ &\approx 0.91\end{aligned}$$

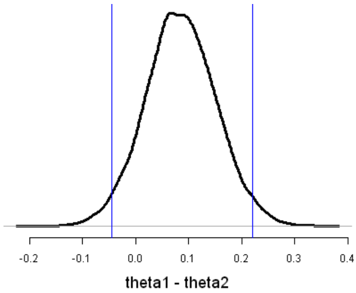
- Directly interpretable result; *not* a frequentist procedure

Stan Binomial Comparison

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

Visualizing Posterior Difference

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



- Vertical bars: central 95% posterior interval $(-0.05, 0.22)$

Technical Interlude
Conjugate Priors

Conjugate Priors

- Family $\mathcal{F}$ is a conjugate prior for family $\mathcal{G}$ if
  - prior in $\mathcal{F}$ and
  - likelihood in $\mathcal{G}$,
  - entails posterior in $\mathcal{F}$
- Before MCMC techniques became practical, Bayesian analysis mostly involved conjugate priors
- Still widely used because analytic solutions are more efficient than MCMC

Beta is Conjugate to Binomial

- Prior: $p(\theta|\alpha,\beta) = \text{Beta}(\theta|\alpha,\beta)$
- Likelihood: $p(y|N,\theta) = \text{Binomial}(y|N,\theta)$
- Posterior:

$$
\begin{aligned}
p(\theta|y,N,\alpha,\beta) &\propto p(\theta|\alpha,\beta)\, p(y|N,\theta) \\
&= \text{Beta}(\theta|\alpha,\beta)\, \text{Binomial}(y|N,\theta) \\
&= \frac{1}{\text{B}(\alpha,\beta)}\theta^{\alpha-1}\,(1-\theta)^{\beta-1}\;\binom{N}{y}\theta^{y}\,(1-\theta)^{N-y} \\
&\propto \theta^{y+\alpha-1}\,(1-\theta)^{N-y+\beta-1} \\
&\propto \text{Beta}(\theta|\alpha+y,\;\beta+(N-y))
\end{aligned}
$$

Chaining Updates

- Start with prior Beta$(\theta|\alpha, \beta)$
- Receive binomial data in K statges $(y_1, N_1), \ldots, (y_K, N_K)$
- After (y_1, N_1), posterior is Beta$(\theta|\alpha + y_1,\ \beta + N_1 - y_1)$
- Use as prior for (y_2, N_2), with posterior
  Beta$(\theta|\alpha + y_1 + y_2,\ \ \beta + (N_1 - y_1) + (N_2 - y_2))$
- Lather, rinse, repeat, until final posterior
  Beta$(\theta|\alpha + y_1 + \cdots + y_K,\ \ \beta + (N_1 + \cdots + N_K) - (y_1 + \cdots + y_K))$
- Same result as if we'd updated with combined data
  Beta$(y_1 + \cdots + y_K,\ \ N_1 + \cdots + N_K)$

Part II

(Un-)Bayesian Point Estimation

MAP Estimator

- For a Bayesian model $p(y, \theta) = p(y|\theta)\, p(\theta)$, the max a posteriori (MAP) estimate maximizes the posterior,

$$\begin{aligned}
\theta^* &= \text{arg max}_\theta \; p(\theta|y) \\
&= \text{arg max}_\theta \; \frac{p(y|\theta)p(\theta)}{p(y)} \\
&= \text{arg max}_\theta \; p(y|\theta)p(\theta). \\
&= \text{arg max}_\theta \; \log p(y|\theta) + \log p(\theta).
\end{aligned}$$

- *not* Bayesian because it doesn't integrate over uncertainty
- *not* frequentist because of distributions over parameters

MAP and the MLE

- MAP estimate reduces to the MLE if the prior is uniform, i.e.,

  $$p(\theta) = c$$

  because

  $$\begin{aligned} \theta^* &= \arg\max_\theta \, p(y|\theta)\, p(\theta) \\ &= \arg\max_\theta \, p(y|\theta)\, c \\ &= \arg\max_\theta \, p(y|\theta). \end{aligned}$$

Penalized Maximum Likelihood

- The MAP estimate can be made palatable to frequentists via philosophical sleight of hand
- Treat the negative log prior $-\log p(\theta)$ as a "penalty"
- e.g., a Normal$(\theta|\mu,\sigma)$ prior becomes a penalty function

$$\lambda_{\theta,\mu,\sigma} \; = \; -\left(\log\sigma + \frac{1}{2}\left(\frac{\theta-\mu}{\sigma}\right)^2\right)$$

- Maximize sum of log likelihood and negative penalty

$$\begin{aligned}\theta^* &= \text{arg max}_\theta \; \log p(y|\theta,x) - \lambda_{\theta,\mu,\sigma} \\ &= \text{arg max}_\theta \; \log p(y|\theta,x) + \log p(\theta|\mu,\sigma)\end{aligned}$$

Proper Bayesian Point Estimates

- Choose estimate to minimize some loss function
- To minimize expected squared error (L2 loss), $\mathbb{E}[(\theta - \theta')^2 \mid y]$, use the posterior mean

$$\hat{\theta} \;=\; \text{arg min}_{\theta'} \mathbb{E}[(\theta - \theta')^2 \mid y] \;=\; \int_{\Theta} \theta \times p(\theta|y)\, d\theta.$$

- To minimize expected absolute error (L1 loss), $\mathbb{E}[|\theta - \theta'|]$, use the posterior median.
- Other loss (utility) functions possible, the study of which falls under decision theory
- All share property of involving full Bayesian inference.

Point Estimates for Inference?

- Common in machine learning to generate a point estimate θ^*, then use it for inference, $p(\tilde{y}|\theta^*)$

- This is **defective** because it

  underestimates uncertainty.

- To properly estimate uncertainty, apply full Bayes

- A major focus of statistics and decision theory is estimating uncertainty in our inferences

Philosophical Interlude

What is Statistics?

Exchangeability

- Roughly, an exchangeable probability function is such that for a sequence of random variables $y = y_1, \ldots, y_N$,

  $$p(y) = p(\pi(y))$$

  for every N-permutation π (i.e, a one-to-one mapping of $\{1, \ldots, N\}$)

- i.i.d. implies exchangeability, but not vice-versa

Exchangeability Assumptions

- Models almost always make some kind of exchangeability assumption
- Typically when other knowledge is not available
  - e.g., treat voters as conditionally i.i.d. given their age, sex, income, education level, religous affiliation, and state of residence
  - But voters have many more properties (hair color, height, profession, employment status, marital status, car ownership, gun ownership, etc.)
  - Missing predictors introduce additional error (on top of measurement error)

Random Parameters: Doxastic or Epistemic?

- Bayesians treat distributions over parameters as epistemic (i.e., about knowledge)
- They do *not* treat them as being doxastic (i.e., about beliefs)
- Priors encode our knowledge before seeing the data
- Posteriors encode our knowledge after seeing the data
- Bayes’s rule provides the way to update our knowledge
- People like to pretend models are ontological (i.e., about what exists)

Arbitrariness: Priors vs. Likelihood

- Bayesian analyses often criticized as subjective (arbitrary)
- Choosing priors is no more arbitrary than choosing a likelihood function (or an exchangeability/i.i.d. assumption)
- As George Box famously wrote (1987),

"All models are wrong, but some are useful."

- This does not just apply to Bayesian models!

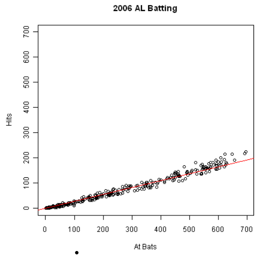
Part III
Hierarchical Models

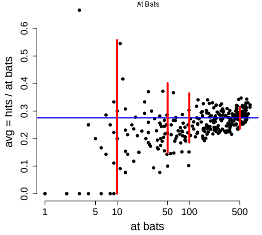
Baseball At-Bats

- For example, consider baseball batting ability.
  - Baseball is sort of like cricket, but with round bats, a one-way field, stationary "bowlers", four bases, short games, and no draws
- Batters have a number of "at-bats" in a season, out of which they get a number of "hits" (hits are a good thing)
- Nobody with higher than 40% success rate since 1950s.
- No player (excluding "bowlers") bats much less than 20%.
- Same approach applies to hospital pediatric surgery complications (a BUGS example), reviews on Yelp, test scores in multiple classrooms, . . .

Baseball Data

- Hits versus at bats for the 2006 American League season
- Not much variation in ability!
- Ignore skill vs. at-bats relation
- Note uncertainty of MLE





Pooling Data

- How do we estimate the ability of a player who we observe getting 6 hits in 10 at-bats? Or 0 hits in 5 at-bats? Estimates of 60% or 0% are absurd!
- Same logic applies to players with 152 hits in 537 at bats.
- *No pooling*: estimate each player separately
- *Complete pooling*: estimate all players together (assume no difference in abilities)
- *Partial pooling*: somewhere in the middle
  - use information about other players (i.e., the population) to estimate a player's ability

Hierarchical Models

- Hierarchical models are principled way of determining how much pooling to apply.
- Pull estimates toward the population mean based on amount of variation in population
  - low variance population: more pooling
  - high variance population: less pooling
- In limit
  - as variance goes to 0, get complete pooling
  - as variance goes to ∞, get no pooling

Hierarchical Batting Ability

- Instead of fixed priors, estimate priors along with other parameters
- Still only uses data once for a single model fit
- Data: y_n, B_n: hits, at-bats for player n
- Parameters: θ_n: ability for player n
- Hyperparameters: α, β: population mean and variance
- Hyperpriors: fixed priors on α and β (hardcoded)

Hierarchical Batting Model (cont.)

$$\begin{aligned}
y_n &\sim \text{Binomial}(B_n, \theta_n) \\
\theta_n &\sim \text{Beta}(\alpha, \beta) \\
\frac{\alpha}{\alpha + \beta} &\sim \text{Uniform}(0, 1) \\
(\alpha + \beta) &\sim \text{Pareto}(1.5)
\end{aligned}$$

- Sampling notation syntactic sugar for:

  $p(y, \theta, \alpha, \beta) = \text{Pareto}(\alpha+\beta|1.5) \prod_{n=1}^{N} \left(\text{Binomial}(y_n|B_n, \theta_n)\ \text{Beta}(\theta_n|\alpha, \beta)\right)$

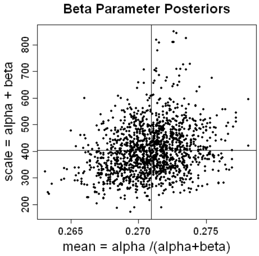
- Pareto provides power law: $\text{Pareto}(u|\alpha) \propto \frac{\alpha}{u^{\alpha+1}}$
- Should use more informative hyperpriors!

Stan Program

```
data {
  int<lower=0> N;                 // items
  int<lower=0> K[N];              // initial trials
  int<lower=0> y[N];              // initial successes
}
parameters {
  real<lower=0, upper=1> phi;          // population chance of suc
  real<lower=1> kappa;                 // population concentration
  vector<lower=0, upper=1>[N] theta;   // chance of success
}
model {
  kappa ~ pareto(1, 1.5);                          // hyperprior
  theta ~ beta(phi * kappa, (1 - phi) * kappa);    // prior
  y ~ binomial(K, theta);                          // likelihood
}
```

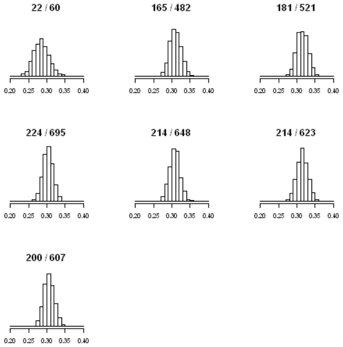
Hierarchical Prior Posterior

- Draws from posterior (crosshairs at posterior mean)
- Prior population mean: 0.271
- Prior population scale: 400
- Together yield prior std dev of 0.022
- Mean is better estimated than scale (typical)



Posterior Ability (High Avg Players)

- Histogram of posterior draws for high-average players
- Note uncertainty grows with lower at-bats



Multiple Comparisons

- Who has the highest ability (based on this data)?
- Probabilty player n is best is

Average	*At-Bats*	Pr[best]
.347	521	0.12
.343	623	0.11
.342	482	0.08
.330	648	0.04
.330	607	0.04
.367	60	0.02
.322	695	0.02

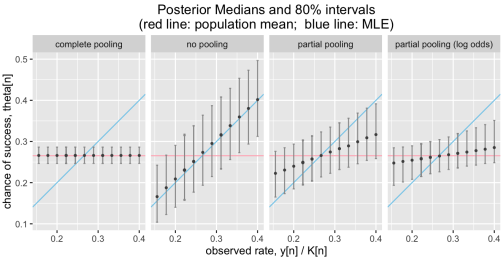
- No clear winner—sample size matters.
- In last game (of 162), Mauer (Minnesota) edged out Jeter (NY)

Efron & Morris (1975) Data

```
   FirstName   LastName Hits At.Bats Rest.At.Bats Rest.Hits
1    Roberto   Clemente   18      45          367       127
2      Frank   Robinson   17      45          426       127
3      Frank     Howard   16      45          521       144
4        Jay  Johnstone   15      45          275        61
5        Ken      Berry   14      45          418       114
6        Jim    Spencer   14      45          466       126
7        Don  Kessinger   13      45          586       155
8       Luis   Alvarado   12      45          138        29
9        Ron      Santo   11      45          510       137
10       Ron    Swaboda   11      45          200        46
11      Rico Petrocelli   10      45          538       142
```

Pooling Estimates

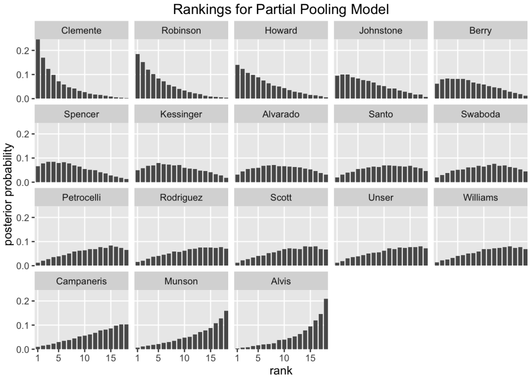
- Case Study: Repeated Binary Trials (mc-stan.org)



Ranking

```
generated quantities {
  int<lower=1, upper=N> rnk[N];      // rank of player n
  {
    int dsc[N];
    dsc <- sort_indices_desc(theta);
    for (n in 1:N)
      rnk[dsc[n]] <- n;
  }
```

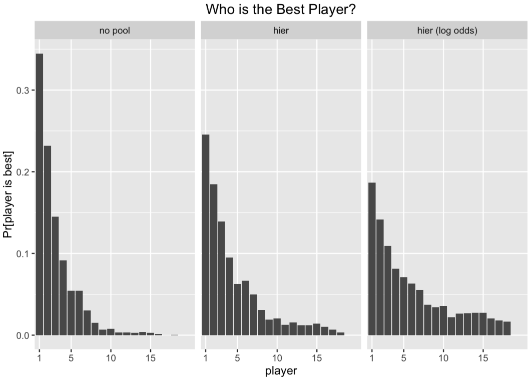
Posterior Ranks



Who is Best? Stan Code

```
generated quantities {
  ...
  int<lower=0, upper=1> is_best[N];  // Pr[player n highest chanc
  ...
  for (n in 1:N)
    is_best[n] <- (rnk[n] == 1);
  ...
```

Who is Best? Posterior

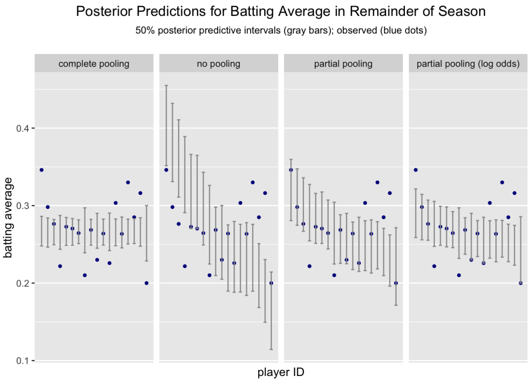


Posterior Predictive Inference

- How do we predict new outcomes (e.g., rest of season)?

```
data {
  int<lower=0> K_new[N];       // new trials
  int<lower=0> y_new[N];       // new successes
  ...
generated quantities {
  int<lower=0> z[N];  // posterior prediction
  for (n in 1:N)
    z[n] <- binomial_rng(K_new[n], theta[n]);
```

Posterior Predictions

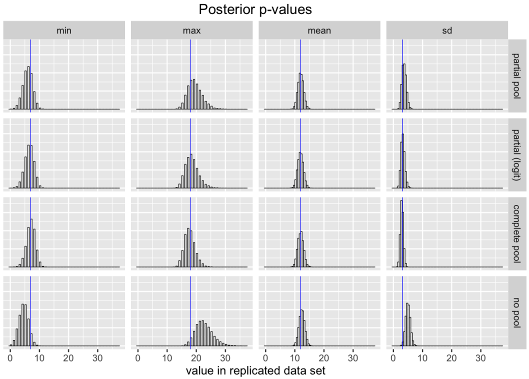


Posterior Predictive Check

- Replicate data from paraemters

```
generated quantities {
  ...
  for (n in 1:N)
    y_rep[n] <- binomial_rng(K[n], theta[n]);
  for (n in 1:N)
    y_pop_rep[n] <- binomial_rng(K[n],
                                 beta_rng(phi * kappa,
                                          (1 - phi) * kappa));
  min_y_rep <- min(y_rep);
  sd_y_rep <- sd(to_vector(y_rep));
  p_min <- (min_y_rep >= min_y);
  p_sd <- (sd_y_rep >= sd_y);
}
```

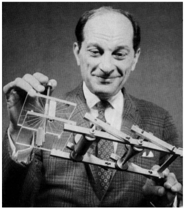
Posterior p-Values



Part IV
Stan Overview

Stan’s Namesake

- Stanislaw Ulam (1909–1984)
- Co-inventor of Monte Carlo method (and hydrogen bomb)



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

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++03)
- **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) [under testing]
- **Posterior Visualization & Exploration**
  - **ShinyStan**: Shiny (R) web-based

Higher-Level Interfaces

- **R Interfaces**
  - **RStanArm**: Regression modeling with R expressions
  - **ShinyStan**: Web-based posterior visualization, exploration
  - **Loo**: Approximate leave-one-out cross-validation
- **Containers**
  - Dockerized Jupyter (iPython) Notebooks (R, Python, or Julia)

Who's Using Stan?

- 1600+ **users group** registrations; 10,000 manual **downloads** (2.5.0); 300+ Google scholar citations
- **Biological sciences**: clinical drug trials, entomology, opthalmology, neurology, genomics, agriculture, botany, fisheries, cancer biology, epidemiology, population ecology, neurology
- **Physical sciences**: astrophysics, molecular biology, oceanography, climatology, biogeochemistry
- **Social sciences**: population dynamics, psycholinguistics, social networks, political science, surveys
- **Other**: materials engineering, finance, actuarial, sports, public health, recommender systems, educational testing, equipment maintenance

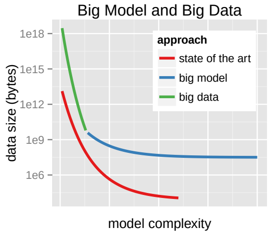
Documentation

- *Stan User's Guide and Reference Manual*
  - 500+ 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
  - RStan vignette

Books and Model Sets

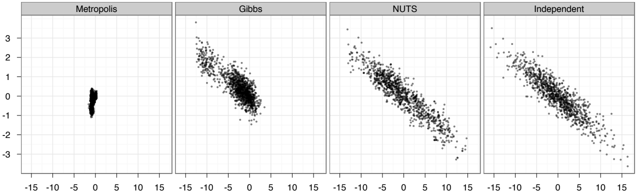
- **Model Sets** Translated to Stan
  - BUGS and JAGS examples (most of all 3 volumes)
  - Gelman and Hill (2009) *Data Analysis Using Regression and Multilevel/Hierarchical Models*
  - Wagenmakers and Lee (2014) *Bayesian Cognitive Modeling*
- **Books** with Sections on Stan
  - Gelman et al. (2013) *Bayesian Data Analysis*, 3rd Edition.
  - Kruschke (2014) *Doing Bayesian Data Analysis, Second Edition: A Tutorial with R, JAGS, and Stan*
  - Korner-Nievergelt et al. (2015) *Bayesian Data Analysis in Ecology Using Linear Models with R, BUGS, and Stan*

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

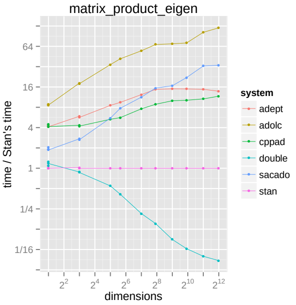
NUTS vs. Gibbs and Metropolis

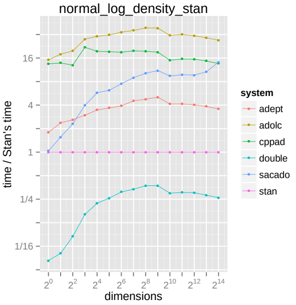


- Two dimensions of highly correlated 250-dim normal
- **1,000,000 draws** from Metropolis and Gibbs (thin to 1000)
- **1000 draws** from NUTS; 1000 independent draws

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 is a Programming Language

- **Not** a graphical specification language like BUGS or JAGS
- Stan is a Turing-complete imperative programming langauge 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

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. 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

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,

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));
  }

}
```

Differential Equation Solver

- System expressed as function
  - given state (y) time (t), parameters (θ), 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

- 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 θ

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

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

Diff Eq Derivatives

- Need derivatives of solution w.r.t. parameters
- Couple derivatives of system w.r.t. parameters

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

- Calculate coupled system via nested autodiff of second term

$$\frac{\partial}{\partial \theta} \frac{\partial y}{\partial t}$$

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

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);`

“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);
  ```

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)

Transforms: Precision

```
parameters {
  real<lower=0> tau;  // precision
  ...
}
transformed parameters {
  real<lower=0> sigma;  // sd
  sigma <- 1 / sqrt(tau);
}
```

Transforms: “Matt Trick”

```
parameters {
  vector[K] beta_raw;  // non-centered
  real mu;
  real<lower=0> sigma;
}
transformed parameters {
  vector[K] beta;  // centered
  beta <- mu + sigma * beta_raw;
}
model {
  mu ~ cauchy(0, 2.5);
  sigma ~ cauchy(0, 2.5);
  beta_raw ~ normal(0, 1);
}
```

Linear Regression (Normal Noise)

- **Likelihood**:
  - $y_n = \alpha + \beta x_n + \epsilon_n$
  - $\epsilon_n \sim \text{Normal}(0, \sigma)$ for $n \in 1{:}N$
- Equivalently,
  - $y_n \sim \text{Normal}(\alpha + \beta x_n, \sigma)$
- **Priors** (improper)
  - $\sigma \sim \text{Uniform}(0, \infty)$
  - $\alpha, \beta \sim \text{Uniform}(-\infty, \infty)$
- Stan allows **improper prior**; requires **proper posterior**.

Linear Regression: Stan Code

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

//  for (n in 1:N)
//     y[n] ~ normal(alpha + beta * x[n], sigma);
```

Logistic Regression

```
data {
  int<lower=1> K;
  int<lower=0> N;
  matrix[N,K] x;
  int<lower=0,upper=1> y[N];
}
parameters {
  vector[K] beta;
}
model {
   beta ~ cauchy(0, 2.5);          // prior
   y ~ bernoulli_logit(x * beta);  // likelihood
}
```

Time Series Autoregressive: AR(1)

```
data {
  int<lower=0> N;   vector[N] y;
}
parameters {
  real alpha;  real beta;  real sigma;
}
model {
  for (n in 2:N)
    y[n] ~ normal(alpha + beta * y[n-1], sigma);
}
```

Covariance Random-Effects Priors

```
parameters {
  vector[2] beta[G];
  cholesky_factor_corr[2] L_Omega;
  vector<lower=0>[2] sigma;

model {
  sigma ~ cauchy(0, 2.5);
  L_Omega ~ lkj_cholesky(4);
  beta ~ multi_normal_cholesky(rep_vector(0, 2),
                       diag_pre_multiply(sigma, L_Omega));
  for (n in 1:N)
    y[n] ~ bernoulli_logit(... + x[n] * beta[gg[n]]);
```

Example: Gaussian Process Estimation

```
data {
  int<lower=1> N;  vector[N] x; vector[N] y;
} parameters {
  real<lower=0> eta_sq, inv_rho_sq, sigma_sq;
} transformed parameters {
  real<lower=0> rho_sq; rho_sq <- inv(inv_rho_sq);
} model {
  matrix[N,N] Sigma;
  for (i in 1:(N-1)) {
    for (j in (i+1):N) {
      Sigma[i,j] <- eta_sq * exp(-rho_sq * square(x[i] - x[j]));
      Sigma[j,i] <- Sigma[i,j];
  }}
  for (k in 1:N) Sigma[k,k] <- eta_sq + sigma_sq;
  eta_sq, inv_rho_sq, sigma_sq ~ cauchy(0,5);
  y ~ multi_normal(rep_vector(0,N), Sigma);
}
```

Posterior Predictive Inference

- Parameters θ, observed data y and data to predict $\tilde{y}$

$$p(\tilde{y}|y) = \int_{\Theta} p(\tilde{y}|\theta)\ p(\theta|y)\ d\theta$$

-
  ```
data {
  int<lower=0> N_tilde;
  matrix[N_tilde,K] x_tilde;
  ...
parameters {
  vector[N_tilde] y_tilde;
  ...
model {
  y_tilde ~ normal(x_tilde * beta, sigma);
```

Predict w. Generated Quantities

- Replace sampling with pseudo-random number generation

```
generated quantities {
  vector[N_tilde] y_tilde;

  for (n in 1:N_tilde)
    y_tilde[n] <- normal_rng(x_tilde[n] * beta, sigma);
}
```

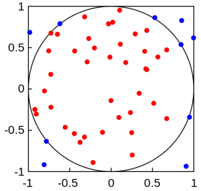
- Must include noise for predictive uncertainty
- PRNGs only allowed in generated quantities block
  - more computationally efficient per iteration
  - more statistically efficient with i.i.d. samples (i.e., MC, not MCMC)

Part V

Monte Carlo Integration

Monte Carlo Calculation of π

- Computing $\pi = 3.14\ldots$ via simulation is *the* textbook application of Monte Carlo methods.
- Generate points uniformly at random within the square
- Calculate proportion within circle ($x^2 + y^2 < 1$) and multiply by square's area (4) to produce the area of the circle.
- This area is π (radius is 1, so area is $\pi r^2 = \pi$)



Plot by Mysid Yoderj courtesy of Wikipedia.

Monte Carlo Calculation of π (cont.)

- R code to calcuate π with Monte Carlo simulation:

```
> x <- runif(1e6,-1,1)
> y <- runif(1e6,-1,1)

> prop_in_circle <- sum(x^2 + y^2 < 1) / 1e6

> 4 * prop_in_circle
[1] 3.144032
```

Accuracy of Monte Carlo

- Monte Carlo is *not* an approximation!
- It can be made exact to within any ϵ
- Monte Carlo draws are i.i.d. by definition
- Central limit theorem: expected error decreases at rate of

$$\frac{1}{\sqrt{N}}$$

- 3 decimal places of accuracy with sample size 1e6
- Need $100\times$ larger sample for each digit of accuracy

General Monte Carlo Integration

- MC can calculate arbitrary definite integrals,

$$\int_a^b f(x)\,dx$$

- Let d upper bound $f(x)$ in (a,b); tightness determines computational efficiency
- Then generate random points uniformly in the rectangle bounded by (a,b) and $(0,d)$
- Multiply proportion of draws (x,y) where $y < f(x)$ by area of rectangle, $d \times (b-a)$.
- Can be generalized to multiple dimensions in obvious way

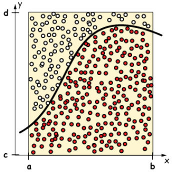


Image courtesy of Jeff Cruzan, http://www.drcruzan.com/NumericalIntegration.html

Expectations of Function of R.V.

- Suppose $f(\theta)$ is a function of random variable vector θ
- Suppose the density of θ is $p(\theta)$
  - *Warning*: θ overloaded as random and bound variable
- Then $f(\theta)$ is also random variable, with expectation

$$\mathbb{E}[f(\theta)] = \int_{\Theta} f(\theta)\; p(\theta)\; d\theta.$$

  - where Θ is support of $p(\theta)$ (i.e., $\Theta = \{\theta \mid p(\theta) > 0\}$

QoI as Expectations

- Most Bayesian quantities of interest (QoI) are expectations over the posterior $p(\theta \mid y)$ of functions $f(\theta)$
- **Bayesian parameter estimation**: $\hat{\theta}$
  - $f(\theta) = \theta$
  - $\hat{\theta} = \mathbb{E}[\theta|y]$ minimizes expected square error
- **Bayesian parameter (co)variance estimation**: $\mathrm{var}[\theta \mid y]$
  - $f(\theta) = (\theta - \hat{\theta})^2$
- **Bayesian event probability**: $\Pr[A \mid y]$
  - $f(\theta) = \mathrm{I}(\theta \in A)$

Expectations via Monte Carlo

- Generate draws $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$ drawn from $p(\theta)$
- Monte Carlo Estimator **plugs in average** for expectation:

$$\mathbb{E}[f(\theta)|y] \approx \frac{1}{M} \sum_{m=1}^{M} f(\theta^{(m)})$$

- Can be made **as accurate as desired**, because

$$\mathbb{E}[f(\theta)] = \lim_{M \to \infty} \frac{1}{M} \sum_{m=1}^{M} f(\theta^{(m)})$$

- *Reminder*: By CLT, error goes down as $1 \,/\, \sqrt{M}$

Part VI
Markov Chain Monte Carlo

Markov Chain Monte Carlo

- Standard Monte Carlo draws i.i.d. draws

  $$\theta^{(1)}, \dots, \theta^{(M)}$$

  according to a probability function $p(\theta)$

- Drawing an i.i.d. sample is often impossible when dealing with complex densities like Bayesian posteriors $p(\theta|y)$

- So we use Markov chain Monte Carlo (MCMC) in these cases and draw $\theta^{(1)}, \dots, \theta^{(M)}$ from a Markov chain

Markov Chains

- A Markov Chain is a sequence of random variables

$$\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(M)}$$

such that $\theta^{(m)}$ only depends on $\theta^{(m-1)}$, i.e.,

$$p(\theta^{(m)}|y, \theta^{(1)}, \ldots, \theta^{(m-1)}) \ = \ p(\theta^{(m)}|y, \theta^{(m-1)})$$

- Drawing $\theta^{(1)}, \ldots, \theta^{(M)}$ from a Markov chain according to $p(\theta^{(m)} \mid \theta^{(m-1)}, y)$ is more tractable
- Require marginal of each draw, $p(\theta^{(m)}|y)$, to be equal to true posterior

Applying MCMC

- Plug in just like ordinary (non-Markov chain) Monte Carlo
- Adjust standard errors for dependence in Markov chain

MCMC for Posterior Mean

- Standard Bayesian estimator is posterior mean

$$\hat{\theta} \;=\; \int_{\Theta} \theta \, p(\theta|y)\, d\theta$$

  - Posterior mean minimizes expected square error

- Estimate is a conditional expectation

$$\hat{\theta} = \mathbb{E}[\theta|y]$$

- Compute by averaging

$$\hat{\theta} \;\approx\; \frac{1}{M} \sum_{m=1}^{M} \theta$$

MCMC for Posterior Variance

- Posterior variance works the same way,

$$\begin{aligned}\mathbb{E}[(\theta - \mathbb{E}[\theta \mid y])^2 \mid y] &= \mathbb{E}[(\theta - \hat{\theta})^2] \\ &\approx \frac{1}{M} \sum_{m=1}^{M} (\theta^{(m)} - \hat{\theta})^2\end{aligned}$$

MCMC for Event Probability

- Event probabilities are also expectations, e.g.,

$$\mathsf{Pr}[\theta_1 > \theta_2] \ = \ \mathbb{E}[\mathsf{I}[\theta_1 > \theta_2]] \ = \ \int_\Theta \mathsf{I}[\theta_1 > \theta_2]\, p(\theta|y) d\theta.$$

- Estimation via MCMC just another plug-in:

$$\mathsf{Pr}[\theta_1 > \theta_2] \approx \frac{1}{M} \sum_{m=1}^{M} \mathsf{I}[\theta_1^{(m)} > \theta_2^{(m)}]$$

- Again, can be made as accurate as necessary

MCMC for Quantiles (incl. median)

- These are not expectations, but still plug in
- Alternative Bayesian estimator is posterior median
  - Posterior median minimizes expected absolute error
- Estimate as median draw of $\theta^{(1)}, \ldots, \theta^{(M)}$
  - just sort and take halfway value
  - e.g., Stan shows 50% point (or other quantiles)
- Other quantiles including interval bounds similar
  - estimate with quantile of draws
  - estimation error goes up in tail (based on fewer draws)

Part VII

MCMC Algorithms

Random-Walk Metropolis

- Draw random initial parameter vector $\theta^{(1)}$ (in support)
- For $m \in 2{:}M$
  - Sample proposal from a (symmetric) jumping distribution, e.g.,
    $$\theta^* \sim \text{MultiNormal}(\theta^{(m-1)}, \sigma \mathbf{I})$$
    where $\mathbf{I}$ is the identity matrix
  - Draw $u^{(m)} \sim \text{Uniform}(0,1)$ and set
    $$\theta^{(m)} = \begin{cases} \theta^* & \text{if } u^{(m)} < \dfrac{p(\theta^*|y)}{p(\theta^{(m-1)}|y)} \\ \theta^{(m-1)} & \text{otherwise} \end{cases}$$

Metropolis and Normalization

- Metropolis only uses posterior in a ratio:

$$\frac{p(\theta^* \mid y)}{p(\theta^{(m)} \mid y)}$$

- This **allows** the use of **unnormalized densities**
- Recall Bayes's rule:

$$p(\theta|y) \propto p(y|\theta)\, p(\theta)$$

- Thus we only need to evaluate sampling (likelihood) and prior
  - i.e., no need to compute normalizing integral for $p(y)$,

$$\int_{\Theta} p(y|\theta)\, p(\theta) d\theta$$

Metropolis-Hastings

- Generalizes Metropolis to asymmetric proposals
- Acceptance ratio is

$$\frac{J(\theta^{(m)}|\theta^{*}) \ \times \ p(\theta^{*}|y)}{J(\theta^{*}|\theta^{(m-1)}) \ \times \ p(\theta^{(m)}|y)}$$

  where J is the (potentially asymmetric) proposal density

- i.e.,

$$\frac{\text{probabilty of being at } \theta^{*} \text{ and jumping to } \theta^{(m-1)}}{\text{probability of being at } \theta^{(m-1)} \text{ and jumping to } \theta^{*}}$$

Metropolis-Hastings (cont.)

- General form ensures equilibrium by maintaining *detailed balance*
- Like Metropolis, only requires ratios
- Many algorithms involve a Metropolis-Hastings “correction”
    - Including vanilla HMC and RHMC and ensemble samplers

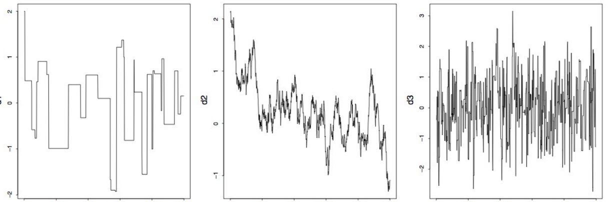
Detailed Balance & Reversibility

- Definition is measure theoretic, but applies to densities
  - just like Bayes's rule
- Assume Markov chain has stationary density $p(a)$
- Suppose $\pi(a|b)$ is density of transitioning from b to a
  - use of π to indicates different measure on Θ than p
- Detailed balance is a reversibility equilibrium condition

$$p(a)\,\pi(b|a) \;=\; p(b)\,\pi(a|b)$$

Optimal Proposal Scale?

- Proposal scale σ is a free; too low or high is inefficient



(a) Proposal variance too large (b) Proposal variance too small (c) Proposal variance approximately optimised

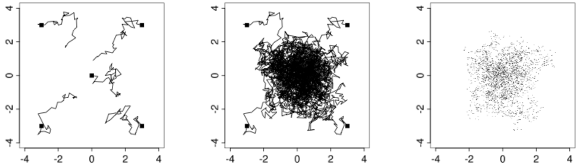
- *Traceplots* show parameter value on y axis, iterations on x
- Empirical tuning problem; theoretical optima exist for some cases

Roberts and Rosenthal (2001) Optimal Scaling for Various Metropolis-Hastings Algorithms. *Statistical Science*.

Convergence

- Imagine releasing a hive of bees in a sealed house
  - they disperse, but eventually reach equilibrium where the same number of bees leave a room as enter it (on average)
- May take many iterations for Markov chain to reach equilibrium

Convergence: Example



- Four chains with different starting points
  - *Left*: 50 iterations
  - *Center*: 1000 iterations
  - *Right*: Draws from second half of each chain

Gelman et al., *Bayesian Data Analysis*

Potential Scale Reduction ($\hat{R}$)

- Gelman & Rubin recommend M chains of N draws with **diffuse initializations**
- Measure that each chain has same posterior mean and variance
- If not, may be stuck in multiple modes or just not converged yet
- Define statistic $\hat{R}$ of chains s.t. **at convergence**, $\hat{R} \to 1$
  - $\hat{R} >> 1$ implies non-convergence
  - $\hat{R} \approx 1$ **does not guarantee convergence**
  - Only measures marginals

Split $\hat{R}$

- Vanilla $\hat{R}$ may not diagnose non-stationarity
  - e.g., a sequence of chains with an increasing parameter
- **Split $\hat{R}$**: Stan splits each chain into first and second half
  - start with M Markov chains of N draws each
  - split each in half to creates $2M$ chains of $N/2$ draws
  - then apply $\hat{R}$ to the $2M$ chains

Calculating $\hat{R}$ Statistic: Between

- M chains of N draws each
- **Between-sample variance** estimate

$$B = \frac{N}{M-1} \sum_{m=1}^{M} (\bar{\theta}_m^{(\bullet)} - \bar{\theta}_{\bullet}^{(\bullet)})^2,$$

where

$$\bar{\theta}_m^{(\bullet)} = \frac{1}{N} \sum_{n=1}^{N} \theta_m^{(n)} \quad \text{and} \quad \bar{\theta}_{\bullet}^{(\bullet)} = \frac{1}{M} \sum_{m=1}^{M} \bar{\theta}_m^{(\bullet)}.$$

Calculating $\hat{R}$ (cont.)

- M chains of N draws each
- **Within-sample variance** estimate:

$$W = \frac{1}{M} \sum_{m=1}^{M} s_m^2,$$

where

$$s_m^2 = \frac{1}{N-1} \sum_{n=1}^{N} (\theta_m^{(n)} - \bar{\theta}_m^{(\bullet)})^2.$$

Calculating $\hat{R}$ Statistic (cont.)

- **Variance** estimate:

$$\widehat{\text{var}}^{+}(\theta|y) = \frac{N-1}{N} W + \frac{1}{N} B.$$

  recall that W is within-chain variance and B between-chain

- **Potential scale reduction** statistic (“R hat”)

$$\hat{R} = \sqrt{\frac{\widehat{\text{var}}^{+}(\theta|y)}{W}}.$$

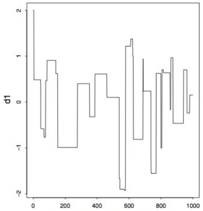
Correlations in Posterior Draws

- Markov chains typically display autocorrelation in the series of draws $\theta^{(1)}, \ldots, \theta^{(m)}$
- Without i.i.d. draws, central limit theorem *does not apply*
- Effective sample size N_{eff} divides out autocorrelation
- N_{eff} must be estimated from sample
  - Fast Fourier transform computes correlations at all lags
- Estimation accuracy proportional to

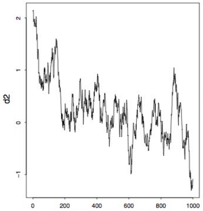
$$\frac{1}{\sqrt{N_{\text{eff}}}}$$

Reducing Posterior Correlation

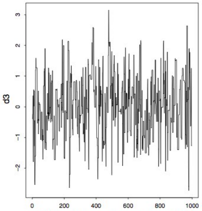
- Tuning algorithm parameters to ensure good mixing
- Recall Metropolis traceplots of Roberts and Rosenthal:



(a) Proposal variance too large



(b) Proposal variance too small



(c) Proposal variance approximately optimised

- Good jump scale σ produces good mixing and high N_{eff}

Effective Sample Size

- Autocorrelation at lag t is correlation between subseqs
  - $(\theta^{(1)}, \ldots, \theta^{(N-t)})$ and $(\theta^{(1+t)}, \ldots, \theta^{(N)})$
- Suppose chain has density $p(\theta)$ with
  - $\mathbb{E}[\theta] = \mu$ and $\mathrm{Var}[\theta] = \sigma^2$
- Autocorrelation ρ_t at lag $t \geq 0$:

$$\rho_t = \frac{1}{\sigma^2} \int_\Theta (\theta^{(n)} - \mu)(\theta^{(n+t)} - \mu)\, p(\theta)\, d\theta$$

- Because $p(\theta^{(n)}) = p(\theta^{(n+t)}) = p(\theta)$ at convergence,

$$\rho_t = \frac{1}{\sigma^2} \int_\Theta \theta^{(n)}\, \theta^{(n+t)}\, p(\theta)\, d\theta$$

Estimating Autocorrelations

- Effective sample size (N draws in chain) is defined by

$$N_{\text{eff}} \;=\; \frac{N}{\sum_{t=-\infty}^{\infty} \rho_t} \;=\; \frac{N}{1 + 2\sum_{t=1}^{\infty} \rho_t}$$

- Estimate in terms of variograms (M chains) at lag t
  - Calculate with fast Fourier transform (FFT)

$$V_t = \frac{1}{M} \sum_{m=1}^{M} \left(\frac{1}{N_m - t} \sum_{n=t+1}^{N_m} \left(\theta_m^{(n)} - \theta_m^{(n-t)} \right)^2 \right)$$

- Adjust autocorrelation at lag t using cross-chain variance as

$$\hat{\rho}_t = 1 - \frac{V_t}{2\,\widehat{\text{var}}^{+}}$$

- If not converged, $\widehat{\text{var}}^{+}$ overestimates variance

Estimating N_{eff}

- Let T' be first lag s.t. $\rho_{T'+1} < 0$,
- Estimate autocorrelation by

$$\hat{N}_{\text{eff}} = \frac{MN}{1 + \sum_{t=1}^{T'} \hat{\rho}_t}.$$

- NUTS avoids negative autocorrelations, so first negative autocorrelation estimate is reasonable
- For basics (not our estimates), see

  Charles Geyer (2013) Introduction to MCMC. In *Handbook of MCMC*.

  (free online at `http://www.mcmchandbook.net/index.html`)

Gibbs Sampling

- Draw random initial parameter vector $\theta^{(1)}$ (in support)
- For $m \in 2{:}M$
  - For $n \in 1{:}N$:
    - draw $\theta_n^{(m)}$ according to conditional

      $$p(\theta_n | \theta_1^{(m)}, \ldots, \theta_{n-1}^{(m)}, \theta_{n+1}^{(m-1)}, \ldots, \theta_N^{(m-1)}, y).$$
- e.g, with $\theta = (\theta_1, \theta_2, \theta_3)$:
  - draw $\theta_1^{(m)}$ according to $p(\theta_1 | \theta_2^{(m-1)}, \theta_3^{(m-1)}, y)$
  - draw $\theta_2^{(m)}$ according to $p(\theta_2 | \theta_1^{(m)}, \theta_3^{(m-1)}, y)$
  - draw $\theta_3^{(m)}$ according to $p(\theta_3 | \theta_1^{(m)}, \theta_2^{(m)}, y)$

Generalized Gibbs

- "Proper" Gibbs requires conditional Monte Carlo draws
  - typically works only for conjugate priors
- In general case, may need to use less efficient conditional draws
  - Slice sampling is a popular general technique that works for discrete or continuous θ_n (JAGS)
  - Adaptive rejection sampling is another alternative (BUGS)
  - Very difficult in more than one or two dimensions

Sampling Efficiency

- We care only about N_{eff} per second
- Decompose into
  1. Iterations per second
  2. Effective sample size per iteration
- Gibbs and Metropolis have high iterations per second (especially Metropolis)
- But they have low effective sample size per iteration (especially Metropolis)
- Both are particular weak when there is high correlation among the parameters in the posterior

Hamiltonian Monte Carlo & NUTS

- Slower iterations per second than Gibbs or Metropolis
- Much higher effective sample size per iteration for complex posteriors (i.e., high curvature and correlation)
- Overall, much higher $N_{1\text{ff}}$ per second

- Details in the next talk . . .
- Along with details of how Stan implements HMC and NUTS

Part VIII

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
- Adaptation **during warmup**
    - step size adapted to target total acceptance rate
    - mass matrix estimated with regularization
- Adaptation **during sampling**
    - simulate forward and backward in time until U-turn
- **Slice sample** along path

(Hoffman and Gelman 2011, 2014)

Posterior Inference

- Generated quantities block for **inference**
  (predictions, decisions, and event probabilities)
- **Extractors** for draws in sample 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

Penalized MLE

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

Posterior Analysis: Estimates

- For each parameter (and `lp__`)
  - Posterior mean
  - Posterior standard deviation
  - Posterior MCMC error esimate: sd$/N_{\text{eff}}$
  - Posterior quantiles
  - Number of effective samples
  - $\hat{R}$ convergence statistic

- . . . and much much more in ShinyStan

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** (soon: Hessian)
  - data I/O and parameter initialization
  - model provides variable names and dimensionalities
  - transforms to and from constrained representation (with or without Jacobian)

Part IX
How Stan Works

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)

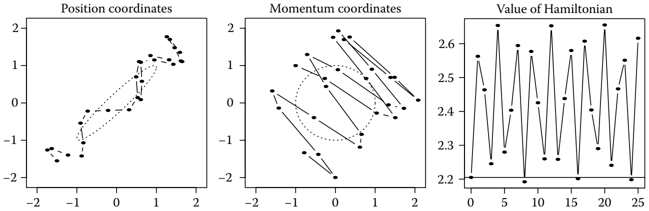
Optimize: L-BFGS

- Initialize unconstrained parameters and Hessian
  - Random values on unconstrained scale uniform in $(-2, 2)$
    - or user specified on constrained scale, transformed
  - Hessian approximation initialized to unit matrix
- While not converged
  - Move unconstrained parameters toward optimum based on Hessian approximation and step size (Newton step)
  - If diverged (arithmetic, support), reduce step size, continue
  - else if converged (parameter change, log density change, gradient value), return value
  - else update Hessian approx. based on calculated gradient

Sample: Hamiltonian Flow

- Generate random **kinetic energy**
  - random Normal$(0, 1)$ in each parameter
- Use negative log posterior as **potential energy**
- Hamiltonian is kinetic plus potential energy
- **Leapfrog Integration**: for *fixed* stepsize (time discretization), number of steps (total time), and mass matrix,
  - update momentum half-step based on potential (gradient)
  - update position full step based on momentum
  - update momentum half-step based on potential
- Numerical solution of Hamilton's first-order version of Newton's second-order diff-eqs of motion (force $=$ mass $\times$ acceleration)

Sample: Leapfrog Example



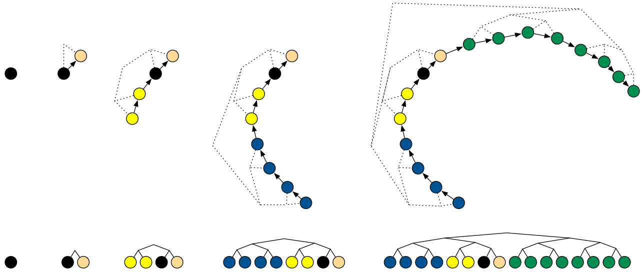
- Trajectory of 25 leapfrog steps for correlated 2D normal (ellipses at 1 sd from mean), stepsize of 0.25, initial state of $(-1, 1)$, and initial momentum of $(-1.5, -1.55)$.

Radford Neal (2013) MCMC using Hamiltonian Dynamics. In *Handbook of MCMC*. (free online at http://www.mcmchandbook.net/index.html)

Sample: No-U-Turn Sampler (NUTS)

- Adapts Hamiltonian simulation time
  - goal to maximize mixing, maintaining detailed balance
  - too short devolves to random walk
  - too long does extra work (i.e., orbits)
- For exponentially increasing number of steps up to max
  - Randomly choose to extend forward or backward in time
  - Move forward or backward in time number of steps
    - stop if any subtree (size 2, 4, 8, ...) makes U-turn
    - remove all current steps if subtree U-turns (not ends)
- Randomly select param with density above slice (or reject)

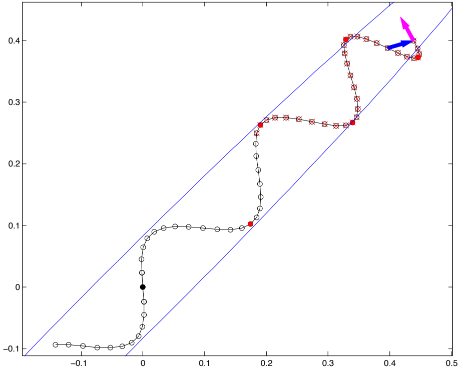
Sample: NUTS Binary Tree



- Example of repeated doubling building binary tree forward and backward in time until U-turn.

  Hoffman and Gelman. 2014. The No-U-Turn Sampler. *JMLR*. (free online at http://jmlr.org/papers/v15/hoffman14a.html)

Sample: NUTS U-Turn



- Example of trajectory from one iteration of NUTS.
- Blue ellipse is contour of 2D normal.
- Black circles are leapfrog steps.
- Solid red circles excluded below slice
- U-turn made with blue and magenta arrows
- Red crossed circles excluded for detailed balance

Sample: HMC/NUTS Warmup

- Estimate stepsize
  - too small requires too many leapfrog steps
  - too large induces numerical inaccuracy
  - need to balance
- Estimate mass matrix
  - Diagonal accounts for parameter scales
  - Dense optionally accounts for rotation

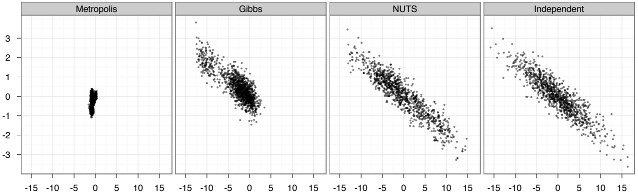
Sample: Warmup (cont.)

- Initialize unconstrained parameters as for optimization
- For exponentially increasing block sizes
  - for each iteration in block
    - generate random kinetic energy
    - simulate Hamiltonian flow (HMC fixed time, NUTS adapts)
    - choose next state (Metroplis for HMC, slice for NUTS)
  - update regularized point estimate of mass matrix
    - use parameter draws from current block
    - shrink diagonal toward unit; dense toward diagonal
  - tune stepsize (line search) for target acceptance rate

Sample: HMC/NUTS Sampling

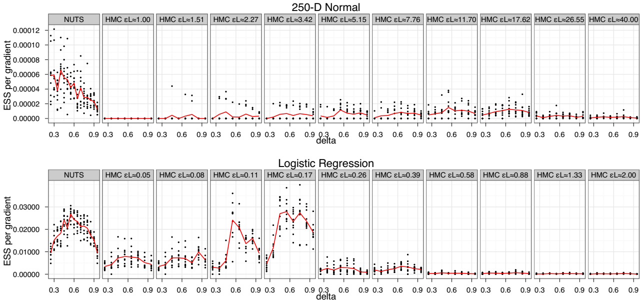
- Fix stepsize and and mass matrix
- For sampling iterations
  - generate random kinetic energy
  - simulate Hamiltonian flow
  - apply Metropolis accept/reject (HMC) or slice (NUTS)

NUTS vs. Gibbs and Metropolis



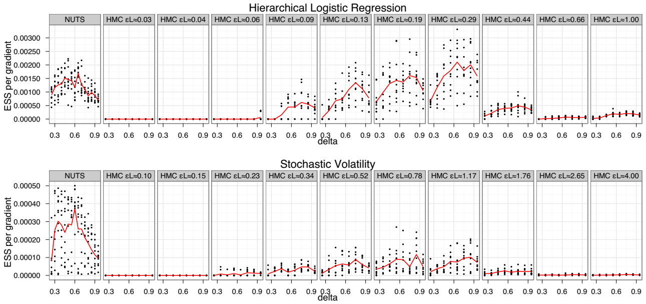
- Two dimensions of highly correlated 250-dim normal
- **1,000,000 draws** from Metropolis and Gibbs (thin to 1000)
- **1000 draws** from NUTS; 1000 independent draws

NUTS vs. Basic HMC



- 250-D normal and logistic regression models
- Vertical axis is effective sample size per sample (bigger better)
- Left) NUTS; Right) HMC with increasing $t = \epsilon L$

NUTS vs. Basic HMC II



- Hierarchical logistic regression and stochastic volatility
- Simulation time t is $\epsilon\, L$, step size (ϵ) times number of steps (L)
- NUTS can beat optimally tuned HMC (latter very expensive)

Part X
Under Stan’s Hood

Euclidean Hamiltonian

- **Phase space**: q position (parameters); p momentum
- **Posterior density**: $\pi(q)$
- **Mass matrix**: M
- **Potential energy**: $V(q) = -\log \pi(q)$
- **Kinetic energy**: $T(p) = \frac{1}{2} p^\top M^{-1} p$
- **Hamiltonian**: $H(p, q) = V(q) + T(p)$
- **Diff eqs**:

$$\frac{dq}{dt} = +\frac{\partial H}{\partial p} \qquad \frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

Leapfrog Integrator Steps

- Solves Hamilton's equations by **simulating dynamics**
  (symplectic [volume preserving]; ϵ^3 error per step, ϵ^2 total error)
- Given: **step size** ϵ, **mass matrix** M, **parameters** q
- **Initialize kinetic** energy, $p \sim \text{Normal}(0, \mathbf{I})$
- **Repeat** for L leapfrog steps:

$$p \leftarrow p - \frac{\epsilon}{2} \frac{\partial V(q)}{\partial q} \qquad \text{[half step in momentum]}$$

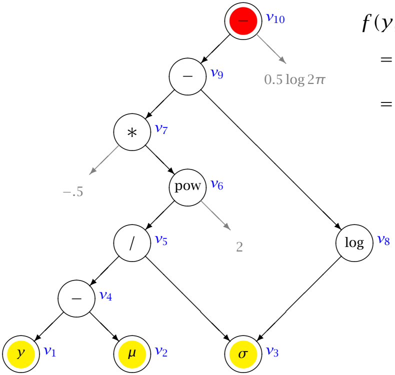
$$q \leftarrow q + \epsilon M^{-1} p \qquad \text{[full step in position]}$$

$$p \leftarrow p - \frac{\epsilon}{2} \frac{\partial V(q)}{\partial q} \qquad \text{[half step in momentum]}$$

Reverse-Mode Auto Diff

- Eval gradient in (usually small) multiple of function eval time
  - independent of dimensionality
  - time proportional to number of expressions evaluated
- Result accurate to machine precision (cf. finite diffs)
- Function evaluation builds up **expression tree**
- Dynamic program propagates **chain rule** in reverse pass
- Reverse mode computes ∇g in one pass for a function $f : \mathbb{R}^N \rightarrow \mathbb{R}$

Autodiff Expression Graph



$$f(y,\mu,\sigma)$$

$$= \log\left(\text{Normal}(y|\mu,\sigma)\right)$$

$$= -\frac{1}{2}\left(\frac{y-\mu}{\sigma}\right)^2 - \log\sigma - \frac{1}{2}\log(2\pi)$$

$$\frac{\partial}{\partial y} f(y,\mu,\sigma)$$

$$= -(y-\mu)\sigma^{-2}$$

$$\frac{\partial}{\partial \mu} f(y,\mu,\sigma)$$

$$= (y-\mu)\sigma^{-2}$$

$$\frac{\partial}{\partial \sigma} f(y,\mu,\sigma)$$

$$= (y-\mu)^2\sigma^{-3} - \sigma^{-1}$$

Autodiff Partials

var	*value*	*partials*	
v_1	y		
v_2	μ		
v_3	σ		
v_4	$v_1 - v_2$	$\partial v_4/\partial v_1 = 1$	$\partial v_4/\partial v_2 = -1$
v_5	v_4/v_3	$\partial v_5/\partial v_4 = 1/v_3$	$\partial v_5/\partial v_3 = -v_4 v_3^{-2}$
v_6	$(v_5)^2$	$\partial v_6/\partial v_5 = 2v_5$	
v_7	$(-0.5)v_6$	$\partial v_7/\partial v_6 = -0.5$	
v_8	$\log v_3$	$\partial v_8/\partial v_3 = 1/v_3$	
v_9	$v_7 - v_8$	$\partial v_9/\partial v_7 = 1$	$\partial v_9/\partial v_8 = -1$
v_{10}	$v_9 - (0.5 \log 2\pi)$	$\partial v_{10}/\partial v_9 = 1$	

Autodiff: Reverse Pass

var	*operation*	*adjoint*	*result*
$a_{1:9}$	$=$	0	$a_{1:9} = 0$
a_{10}	$=$	1	$a_{10} = 1$
a_9	$+=$	$a_{10} \times (1)$	$a_9 = 1$
a_7	$+=$	$a_9 \times (1)$	$a_7 = 1$
a_8	$+=$	$a_9 \times (-1)$	$a_8 = -1$
a_3	$+=$	$a_8 \times (1/v_3)$	$a_3 = -1/v_3$
a_6	$+=$	$a_7 \times (-0.5)$	$a_6 = -0.5$
a_5	$+=$	$a_6 \times (2v_5)$	$a_5 = -v_5$
a_4	$+=$	$a_5 \times (1/v_3)$	$a_4 = -v_5/v_3$
a_3	$+=$	$a_5 \times (-v_4 v_3^{-2})$	$a_3 = -1/v_3 + v_5 v_4 v_3^{-2}$
a_1	$+=$	$a_4 \times (1)$	$a_1 = -v_5/v_3$
a_2	$+=$	$a_4 \times (-1)$	$a_2 = v_5/v_3$

Stan’s Reverse-Mode

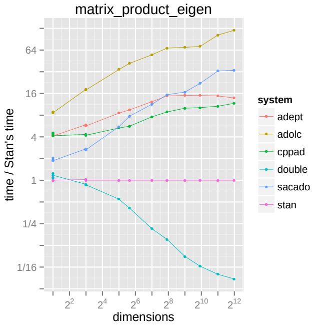
- Easily extensible **object-oriented** design
- **Code nodes** in expression graph for primitive functions
  - requires **partial derivatives**
  - built-in flexible abstract base classes
  - **lazy evaluation** of chain rule saves memory
- Autodiff through templated C++ functions
  - templating on each argument avoids excess promotion

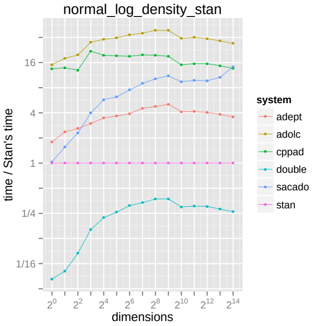
Stan’s Reverse-Mode (cont.)

- Arena-based **memory management**
  - specialized C++ `operator new` for reverse-mode variables
  - custom functions inherit memory management through base
- Nested application to support ODE solver

Stan’s Autodiff vs. Alternatives

- Stan is **fastest** (and uses least memory)
  - among open-source C++ alternatives





Forward-Mode Auto Diff

- Evaluates expression graph forward from one independent variable to any number of dependent variables
- Function evaluation propagates **chain rule** forward
- In one pass, computes $\frac{\partial}{\partial x} f(x)$ for a function $f : \mathbb{R} \rightarrow \mathbb{R}^N$
  - derivative of N outputs with respect to a single input

Stan’s Forward Mode

- Templated scalar type for value and tangent
  - allows higher-order derivatives
- Primitive functions propagate derivatives
- No need to build expression graph in memory
  - much less memory intensive than reverse mode
- Autodiff through templated functions (as reverse mode)

Second-Order Derivatives

- Compute Hessian (matrix of second-order partials)

$$H_{i,j} = \frac{\partial^2}{\partial x_i \partial x_j} f(x)$$

- Required for Laplace covariance approximation (MLE)
- Required for curvature (Riemannian HMC)
- Nest reverse-mode in forward for **second order**
- N forward passes: takes gradient of derivative

Third-Order Derivatives

- Compute gradients of Hessians (tensor of third-order partials)

$$\frac{\partial^3}{\partial x_i \partial x_j \partial x_k} f(x)$$

  - Required for SoftAbs metric (Riemannian HMC)
  - N^2 forward passes: gradient of derivative of derivative

Jacobians

- Assume function $f : \mathbb{R}^N \rightarrow \mathbb{R}^M$
- Partials for multivariate function (matrix of first-order partials)

$$J_{i,j} = \frac{\partial}{\partial x_i} f_j(x)$$

- Required for stiff ordinary differential equations
  - differentiate is coupled sensitivity autodiff for ODE system
- Two execution strategies
  1. Multiple reverse passes for rows
  2. Forward pass per column (required for stiff ODE)

Autodiff Functionals

- Functionals map templated functors to derivatives
  - fully encapsulates and hides all autodiff types
- Autodiff functionals supported
  - gradients: $\mathcal{O}(1)$
  - Jacobians: $\mathcal{O}(N)$
  - gradient-vector product (i.e., directional derivative): $\mathcal{O}(1)$
  - Hessian-vector product: $\mathcal{O}(N)$
  - Hessian: $\mathcal{O}(N)$
  - gradient of trace of matrix-Hessian product: $\mathcal{O}(N^2)$
    (for SoftAbs RHMC)

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

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

Coding Probability Functions

- **Vectorized** to allow scalar or container arguments
  (containers all same shape; scalars broadcast as necessary)

- Avoid **repeated computations**, e.g. $\log \sigma$ in

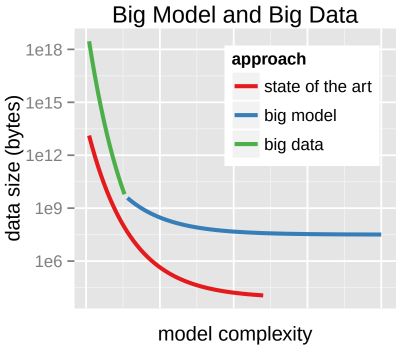
$$\begin{aligned} \log \mathsf{Normal}(y|\mu,\sigma) &= \textstyle\sum_{n=1}^{N} \log \mathsf{Normal}(y_n|\mu,\sigma) \\ &= \textstyle\sum_{n=1}^{N} -\log\sqrt{2\pi} \;-\log\sigma \;-\frac{y_n-\mu}{2\sigma^2} \end{aligned}$$

- recursive **expression templates** to broadcast and cache scalars, generalize containers (arrays, matrices, vectors)

- **traits** metaprogram to **drop constants** (e.g., $-\log\sqrt{2\pi}$ or $\log\sigma$ if constant) and calculate intermediate and return types

Part XI
Stan for Big Data

Scaling and Evaluation



- Types of Scaling: data, parameters, **models**

Riemannian Manifold HMC

- Best mixing MCMC method (fixed # of continuous params)
- Moves on Riemannian manifold rather than Euclidean
  - adapts to position-dependent curvature
- **geoNUTS** generalizes NUTS to RHMC (Betancourt *arXiv*)
- **SoftAbs** metric (Betancourt *arXiv*)
  - eigendecompose Hessian and condition
  - computationally feasible alternative to original Fisher info metric of Girolami and Calderhead (*JRSS, Series B*)
  - requires third-order derivatives and implicit integrator
- Code complete; awaiting higher-order auto-diff

Adiabatic Sampling

- Physically motivated alternative to “simulated” **annealing and tempering** (not really simulated!)
- Supplies external **heat bath**
- Operates through **contact manifold**
- System relaxes more naturally between energy levels
- Betancourt paper on *arXiv*
- Prototype complete

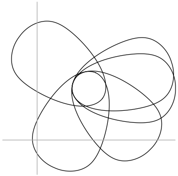
“Black Box” Variational Inference

- **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

“Black Box” EP

- Fast, approximate inference (like VB)
  - VB and EP minimize divergence in opposite directions
  - especially useful for Gaussian processes
- Asynchronous, data-parallel **expectation propagation** (EP)
- Cavity distributions control subsample variance
- Prototypte stage
- collaborating with Seth Flaxman, Aki Vehtari, Pasi Jylänki, John Cunningham, Nicholas Chopin, Christian Robert

The Cavity Distribution



- Two parameters, with data split into $y_1, \ldots, y_5$
- Contours of likelihood $p(y_k|\theta)$ for $k \in 1{:}5$
- $g_{-k}(\theta)$ is **cavity distribution** (current approx. without y_k)
- Separately computing for y_k reqs each partition to cover its area
- Combining likelihood with cavity **focuses on overlap**

Maximum Marginal Likelihood

- Fast, approx. inference for hierarchical models: $p(\phi, \alpha)$
- Marginalize out lower-level params: $p(\phi) = \int p(\phi, \alpha) d\alpha$
- Optimize higher-level parameters ϕ^* and fix
- Optimize lower-level parameters given higher-level: $p(\phi^*, \alpha)$
- Errors estimated as in MLE
- aka "empirical Bayes"
  - but not fully Bayesian
  - and no more empirical than full Bayes
- Design complete; awaiting parameter tagging

Part XII

Posterior Modes & Laplace Approximation

Laplace Approximation

- Multivariate normal approximation to posterior
- Compute posterior mode via optimization

$$\theta^* = \text{arg max}_\theta \; p(\theta|y)$$

- Laplace approximation to the posterior is

$$p(\theta|y) \approx \text{MultiNormal}(\theta^* | -H^{-1})$$

- H is the Hessian of the log posterior

$$H_{i,j} = \frac{\partial^2}{\partial\theta_i \, \partial\theta_j} \log p(\theta|y)$$

Stan's Laplace Approximation

- Operates on unconstrained parameters
- L-BFGS to compute posterior mode $\theta*$
- Automatic differentiation to compute H
  - current R: finite differences of gradients
  - soon: second-order automatic differentiation
- Draw a sample from approximate posterior
  - transfrom back to constrained scale
  - allows Monte Carlo computation of expectations

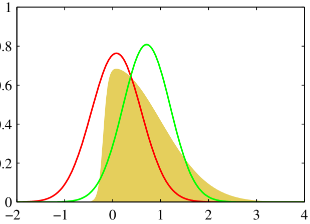
Part XIII
Variational Bayes

VB in a Nutshell

- y is observed data, θ parameters
- Goal is to approximate posterior $p(\theta|y)$
- with a convenient approximating density $g(\theta|\phi)$
  - ϕ is a vector of parameters of approximating density
- Given data y, VB computes ϕ^* minimizing KL-divergence
  - from approximation $g(\theta \mid \phi)$ to posterior $p(\theta \mid y)$

$$\begin{aligned} \phi^* &= \text{arg min}_\phi \ \text{KL}[g(\theta|\phi) \ || \ p(\theta|y)] \\ &= \text{arg max}_\phi \ -\int_\Theta \log\left(\frac{p(\theta \mid y)}{g(\theta \mid \phi)}\right) \ g(\theta|\phi) \ \mathrm{d}\theta \end{aligned}$$

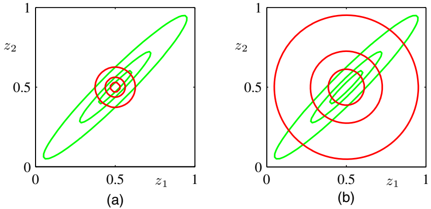
VB vs. Laplace



- *solid yellow*: target; *red*: Laplace; *green*: VB
- **Laplace** located at posterior mode
- **VB** located at approximate posterior mean

— Bishop (2006) *Pattern Recognition and Machine Learning*, fig. 10.1

KL-Divergence Example



- **Green**: true distribution p; **Red**: best approximation g
  (a) VB-like: KL$[g \,||\, p]$
  (b) EP-like: KL$[p \,||\, g]$
- VB systematically **understimates posterior variance**

— Bishop (2006) *Pattern Recognition and Machine Learning*, fig. 10.2

Stan's "Black-Box" VB

- Typically custom $g()$ per model
  - based on conjugacy and analytic updates
- Stan uses "black-box VB" with multivariate Gaussian g

$$g(\theta|\phi) = \text{MultiNormal}(\theta \mid \mu, \Sigma)$$

  for the **unconstrained posterior**
  - e.g., scales σ log-transformed with Jacobian
- Stan provides two versions
  - Mean field: Σ diagonal
  - General: Σ dense

Stan’s VB: Computation

- Use L-BFGS optimization to optimize θ^*
- Requires differentiable $\mathrm{KL}[g(\theta|\phi) \mid\mid p(\theta|y)]$
  - only up to constant (i.e., use evidence lower bound (ELBO))
- Approximate KL-divergence and gradient via Monte Carlo
  - KL divergence is an expectation w.r.t. approximation $g(\theta|\phi)$
  - Monte Carlo draws i.i.d. from approximating multi-normal
  - only need approximate gradient calculation for soundness
  - so only a few Monte Carlo iterations are enough

Stan’s VB: Computation (cont.)

- To support compatible plug-in inference
  - draw Monte Carlo sample $\theta^{(1)}, \ldots, \theta^{(M)}$ with

$$\theta^{(m)} \sim \mathsf{MultiNormal}(\theta \mid \mu^*, \Sigma^*)$$

  - inverse transfrom from unconstrained to constrained scale
  - report to user in same way as MCMC draws

- Future: reweight $\theta^{(m)}$ via importance sampling
  - with respect to true posterior
  - to improve expectation calculations

Near Future: Stochastic VB

- Data-streaming form of VB
  - Scales to billions of observations
  - Hoffman et al. (2013) Stochastic variational inference. *JMLR* 14.
- Mashup of stochastic gradient (Robbins and Monro 1951) and VB
  - subsample data (e.g., stream in minibatches)
  - upweight each minibatch to full data set size
  - use to make unbiased estimate of true gradient
  - take gradient step to minimimize KL-divergence
- Prototype code complete