Markov Chain Monte Carlo for Bayesian Inference

Ben Goodrich

April 01, 2019

Obligatory Disclosure

- Ben is an employee of Columbia University, which has received several research grants to develop Stan
- Ben is also a manager of GG Statistics LLC, which uses Stan for business purposes
- According to Columbia University policy, any such employee who has any equity stake in, a title (such as officer or director) with, or is expected to earn at least $\$5,000.00$ per year from a private company is required to disclose these facts in presentations

Probability

- The analogue of “basic physics” for us is probability: a shared language for communicating about uncertain (sometimes future) propositions
- To be Bayesian, you have to know / learn probability theory
- To be Bayesian, you have to know a good bit about computer programming
- To be Bayesian, you have to know a good bit about the substance
- To be Bayesian, you have to be prepared for everyone to disagree with you
- Key idea is the Multiplication Rule:

$$\Pr(A\&B) = \Pr(A)\Pr(B \mid A) = \Pr(B)\Pr(A \mid B)$$

from which Bayes’ Rule follows if $\Pr(A) > 0$

$$\Pr(B \mid A) = \frac{\Pr(B)\Pr(A \mid B)}{\Pr(A)}$$

Four or Five Sources of Uncertainty

1. Uncertainty about parameters in models
2. Uncertainty about which model is best
3. Uncertainty about what to do with the output of the (best) model(s)
4. Uncertainty about whether the software works as intended
5. Uncertainty about extrapolating from the data you have to the data you want

- Bayesians use probability to describe their uncertainty in (1) and (2)
- The Bayesian approach links with decision theory, which prescribes (3)
- The Stan software does as much as we can to mitigate (4)
- By implication, other approaches / software may refer to probability but fail to handle one or more of the above four items
- These include randomization inference, frequentist inference, supervised learning, and others

A Very, Very Frequentist Example

- Suppose you plan to collect N independent observations on a count outcome (Y) that are generated according to a Poisson distribution with expectation μ
- What probability distribution does the the sample mean follow?
- Let $S = \sum_{n=1}^{N} y_n$ and $\overline{y} = \frac{S}{N}$. The probability that Y takes the value y is

$$\Pr\left(Y = y \mid \mu\right) = \frac{\mu^y e^{-\mu}}{y!}$$

- The probability of observing the entire sample of size N is

$$\Pr\left(y_1, y_2, \ldots y_N \mid \mu\right) = \prod_{n=1}^{N} \frac{\mu^{y_n} e^{-\mu}}{y_n!} = e^{-N\mu} \frac{\mu^{\sum_{n=1}^{N} y_n}}{\prod_{n=1}^{N} y_n!} = \frac{\mu^S e^{-N\mu}}{?}$$

- ? must be S! to make this a PMF, namely Poisson with expectation $N\mu$

A Special Case of the Central Limit Theorem

- If S is distributed Poisson with expectation $N\mu$, then $\overline{y} = \frac{S}{N}$ has expectation μ and $\overline{y} = \frac{S}{N}$ has variance $\frac{N\mu}{N^2} = \frac{\mu}{N}$
- As $N \uparrow \infty$, then the skewness of S, which is $\frac{1}{\sqrt{N\mu}}$, approaches 0 and the excess kurtosis of S, which is $\frac{1}{N\mu}$, approaches 0
- Therefore, $\overline{y} = \frac{S}{N}$ has no skewness or excess kurtosis as $N \uparrow \infty$
- The normal is the only distribution with no skewness or excess kurtosis
- Thus, as $N \uparrow \infty$, $\overline{y} = \frac{S}{N}$ is distributed normal with expectation μ and standard deviation $\sqrt{\frac{\mu}{N}}$
- And as $N \uparrow \infty$, $\frac{\overline{y} - \mu}{\sqrt{\frac{\mu}{N}}}$ is distributed standard normal

A First Look at the Stan Language

```
functions { /* saved as poisson_mean_rng.stan in R's working directory */
  real poisson_mean_rng(int N, real mu) {
    real S = 0;
    if (N < 0)  reject("N must be non-negative");
    if (mu <= 0) reject("mu must be positive");
    for (n in 1:N) {
      int y = poisson_rng(mu);
      S += y;
    }
    return S / N; // avoid "integer division"
  } // equivalent to `mean(rpois(N, mu))` in R
}
```

```
rstan::expose_stan_functions("poisson_mean_rng.stan")
args(poisson_mean_rng) # this is now a function in R
```

```
## function (N, mu, base_rng__ = <pointer: 0x555baac483b0>, pstream__ = <pointer: 0x7f0b4a1e5b
## NULL
```

Frequentist Perspective on Probability

- Probability is necessitated by deliberate randomization, such as sampling from a population
- Probability of X is interpreted as the proportion of times X happens in the limit as the number of random trials approaches infinity
- The probability statements pertain to estimators (or functions thereof)
- Sample mean is distributed normally across datasets (iff variance exists)
- Maximum likelihood estimates are distributed normally across datasets (under some assumptions)
- The probability statements are always pre-data
- The probability statements are conditional on the parameters being estimated
- What use is this Frequentist perspective on probability for applied research?

A Very, Very Bayesian Example

- Taking limits, we can express Bayes' Rule for continuous random variables with Probability Density Functions (PDFs)

$$f(B \mid A) = \frac{f(B)\, f(A \mid B)}{f(A)}$$

- The PDF of the Gamma distribution (shape-rate parameterization) is

$$f(\mu \mid a, b) = \frac{b^a}{\Gamma(a)} \mu^{a-1} e^{-b\mu}$$

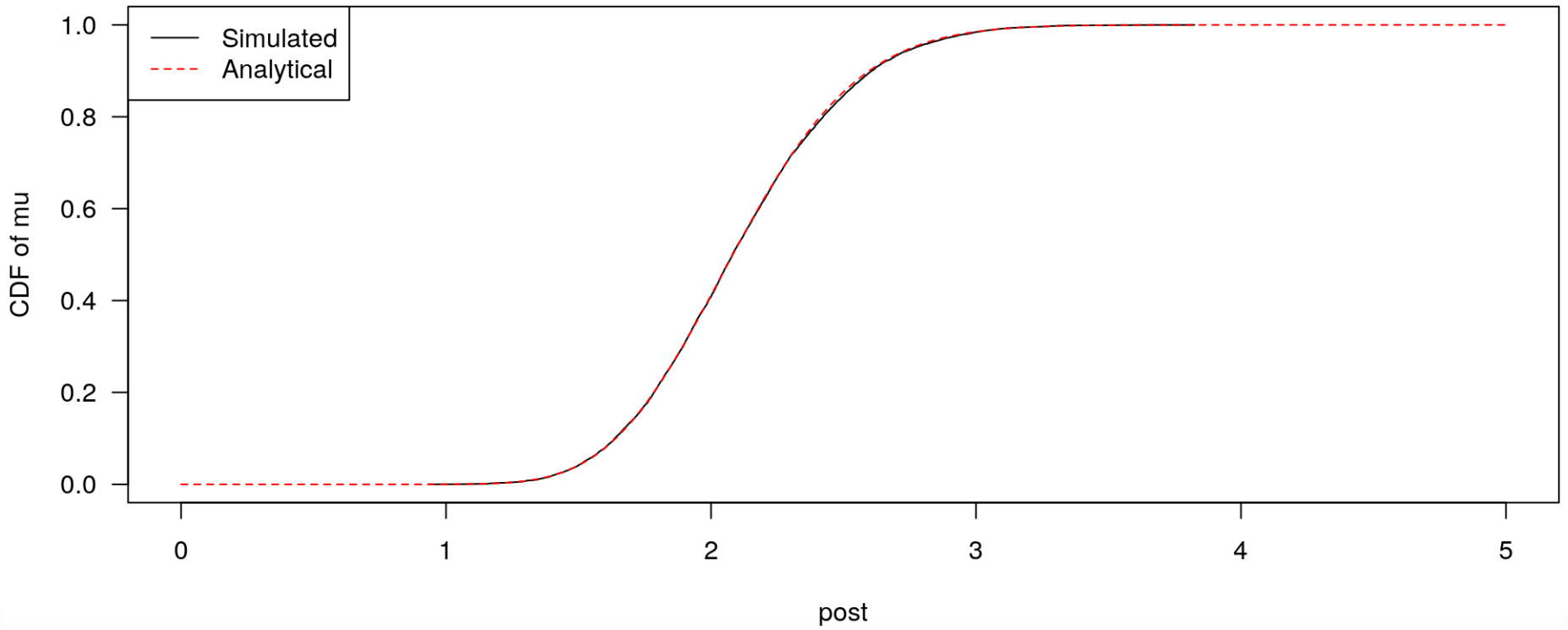
- Poisson PMF for N observations is again $f(y_1, \ldots, y_n \mid \mu) = \frac{\mu^S e^{-N\mu}}{S!}$
- Bayes' Rule is $f(\mu \mid a, b, y_1, \ldots, y_n) = \frac{\mu^{a-1} e^{-b\mu} \mu^S e^{-N\mu}}{?} = \frac{\mu^{a+S-1} e^{-(b+N)\mu}}{?}$
- ? must be $\frac{\Gamma(a^*)}{(b^*)^{a^*}}$ where $a^* = a + S$ and $b^* = b + N$ so posterior is Gamma

Drawing from the Conditional Distribution in Stan

```
functions { /* saved as conditional_rng in R's working directory */
  vector conditional_rng(int S, real a, real b, int[] y) { // y is a 1D integer array
    int sum_y = sum(y);
    vector[S] post; // holds draws from the conditional distribution
    int s = 1;
    if (a <= 0 || b <= 0) reject("a and b must be positive");
    while (s <= S) {
      real mu_tilde = gamma_rng(a, b);
      int y_tilde[size(y)]; // prior predictive distribution for entire sample
      for (n in 1:size(y)) y_tilde[n] = poisson_rng(mu_tilde);
      if (sum_y == sum(y_tilde)) {
        post[s] = mu_tilde;
        s += 1;
      }
    }
    return sort_asc(post);  // sorting just makes it easier to plot the ECDF
  }
}
```

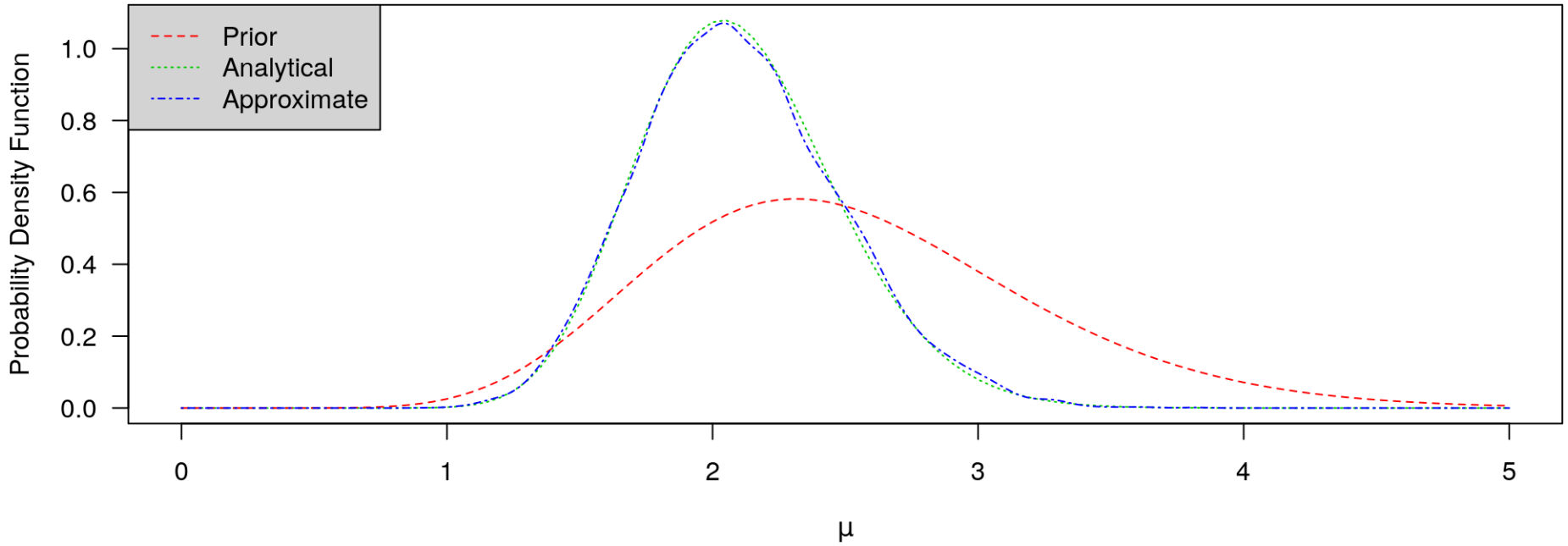
Posterior vs. Posterior

```
rstan::expose_stan_functions("conditional_rng.stan")
a <- 4 * pi; b <- 5; y <- c(3, 1, 0, 2, 2, 4, 1, 3, 2, 1); a_star <- a + sum(y);
b_star <- b + length(y); S <- 10000; post <- conditional_rng(S, a, b, y)
plot(post, (1:S) / S, type = "l", xlim = c(0, 5), ylab = "CDF of mu")
curve(pgamma(mu, a_star, b_star), lty = 2, col = 2, add = TRUE, xname = "mu")
legend("topleft", legend = c("Simulated", "Analytical"), lty = 1:2, col = 1:2)
```



Prior vs Posterior

```
curve(dgamma(mu, a_star, b_star), from = 0, to = 5, lty = 3, col = 3, xname = "mu",
      xlab = expression(mu), ylab = "Probability Density Function")
curve(dgamma(mu, a, b), lty = 2, col = 2, add = TRUE, xname = "mu")
lines(density(post, from = 0, to = 5), lty = 4, col = 4)
legend("topleft", legend = c("Prior", "Analytical", "Approximate"),
       lty = 2:4, col = 2:4, bg = "lightgrey")
```



Bayesian Perspective on Probability

- Probability is necessitated by incomplete information and used to describe your degree of belief that something is true
- The probability statements pertain to beliefs about unknowns
- The probability statements are conditional on the data actually observed
- You have beliefs about how much the S&P500 will grow by the end of 2019
- You express your beliefs with a probability distribution, such as a normal distribution with a mean of $+4\%$ and a standard deviation of 5%
- As more data comes during 2019, you update your beliefs about where the S&P500 will be at the end of 2019 to some new probability distribution
- Note the data are not, and need not be, a sample or an experiment for you to use probability distributions to describe your beliefs in a rigorous way

(Dis)Advantages of Bayesian Inference

- Bayesian inference remains useful in situations other paradigms specialize:
  - Experiments: What are your beliefs about the ATE after seeing the data?
  - Repeated designs: Bayesian estimates have correct frequentist properties
  - Predictive modeling: If you only care about predictions, use the posterior predictive distribution
- Bayesian inference is very useful when you are using the results to make a decision or take an action; other paradigms are not
- Bayesian inference is orders of magnitude more difficult for your computer because it is attempting to answer a more ambitious question
- The Bayesian approach is better suited for convincing yourself of something than convincing other people

Difficulty of Analytical Bayesian Inference

- Bayes Rule for an unknown parameter (vector) $\boldsymbol{\theta}$ conditional on known data (vector) $\mathbf{y}$ can be written as

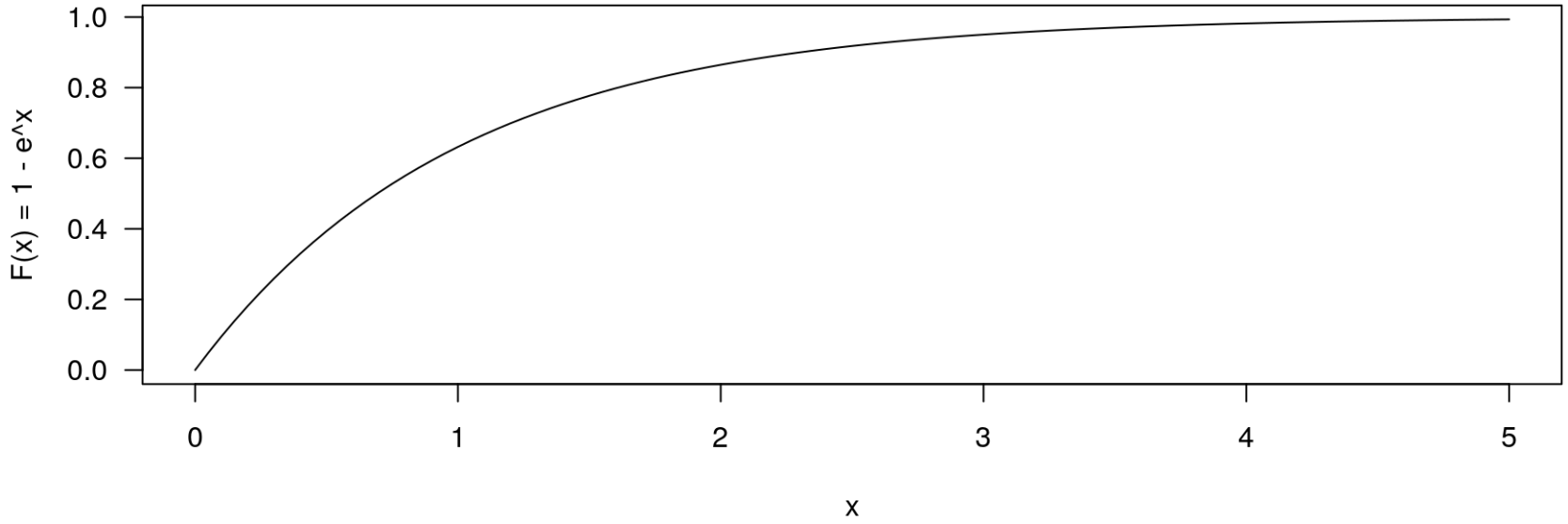
$$f(\boldsymbol{\theta} \mid \mathbf{y}) = \frac{f(\boldsymbol{\theta}) f(\mathbf{y} \mid \boldsymbol{\theta})}{f(\mathbf{y})} = \frac{f(\boldsymbol{\theta}) f(\mathbf{y} \mid \boldsymbol{\theta})}{\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f(\boldsymbol{\theta}) f(\mathbf{y} \mid \boldsymbol{\theta}) \, d\theta_1 d\theta_2 \ldots d\theta_K}$$

- To obtain the denominator of Bayes Rule, you would need to do an integral
- The Risch Algorithm tells you if an integral has an elementary form (rare)
- In most cases, we can't write the denominator of Bayes Rule in a useful form
- But we can draw from a distribution whose PDF is characterized by the numerator of Bayes Rule without knowing the denominator

Drawing from a Uniform Distribution

- Randomness can be harvested from physical sources, but it is expensive
- Modern Intel processors have a (possibly) true random-number generator
- In practice, software emulates a true random-number generator for speed
- Let $M = -1 + 2^{64} = 18,446,744,073,709,551,615$ be the largest unsigned integer that a 64-bit computer can represent. You can essentially draw uniformally from $\Omega_U = [0, 1)$ by
  1. Drawing $\tilde{y}$ from $\Omega_Y = \{0, 1, \ldots, M\}$ with each probability $\frac{1.0}{M}$
  2. Letting $\tilde{u} = \frac{\tilde{y}}{1.0+M}$, which casts to a double-precision denominator
- The CDF of the uniform distribution on (a, b) is $F(u|\, a, b) = \frac{u-a}{b-a}$ and the PDF is $f(u|\, a, b) = \frac{1}{b-a}$. Standard is a special case with $a = 0$ and $b = 1$.

Drawing from an Exponential Distribution



- To draw from this (standard exponential) distribution, you could
  1. Draw $\tilde{u}$ from a standard uniform distribution
  2. Find the point on the curve with height $\tilde{u}$
  3. Drop to the horizontal axis at $\tilde{x}$ to get a standard exponential realization
  4. Optionally scale $\tilde{x}$ by a given μ to make it non-standard

Inverse CDF Sampling of Continuous RVs

- In principle, the previous implies an algorithm to draw from ANY univariate continuous distribution
- But to draw efficiently from it, it is best to work out (if possible) $F^{-1}(u) = x(u)$, which is known as the inverse CDF from $(0, 1)$ to Ω_X
- If $F(x)$ does not have an explicit form, you may have to numerically solve for $\tilde{x}$ such that $F(\tilde{x}) = \tilde{u}$
- For example, this is how R draws from a normal distribution, which has PDF

$$f(x \mid \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Bivariate Normal Distribution

The PDF of the bivariate normal distribution over $\Omega = \mathbb{R}^2$ is

$$f\left(x, y \middle| \mu_X, \mu_Y, \sigma_X, \sigma_Y, \rho\right) =$$

$$\frac{1}{2\pi\sigma_X\sigma_Y\sqrt{1-\rho^2}} e^{-\frac{1}{2\left(1-\rho^2\right)}\left(\left(\frac{x-\mu_X}{\sigma_X}\right)^2+\left(\frac{y-\mu_Y}{\sigma_Y}\right)^2-2\rho\frac{x-\mu_X}{\sigma_X}\frac{y-\mu_Y}{\sigma_Y}\right)} =$$

$$\frac{1}{\sigma_X\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu_X}{\sigma_X}\right)^2} \times \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y-\left(\mu_y+\beta(x-\mu_X)\right)}{\sigma}\right)^2},$$

where X is MARGINALLY normal and $Y \middle| X$ is CONDITIONALLY normal with expectation $\mu_Y + \beta\left(x - \mu_X\right)$ and standard deviation $\sigma = \sigma_Y\sqrt{1-\rho^2}$, where $\beta = \rho\frac{\sigma_Y}{\sigma_X}$ is the OLS coefficient when Y is regressed on X and σ is the error standard deviation. We can thus draw $\tilde{x}$ and then condition on it to draw $\tilde{y}$.

Drawing from the Bivariate Normal Distribution

```
functions { /* saved as binormal_rng.stan in R's working directory */
  matrix binormal_rng(int S, real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    matrix[S, 2] draws; real beta = rho * sigma_Y / sigma_X; // calculate constants once ...
    real sigma = sigma_Y * sqrt(1 - square(rho));            // ... before the loop begins
    for (s in 1:S) {
      real x = normal_rng(mu_X, sigma_X);
      real y = normal_rng(mu_Y + beta * (x - mu_X), sigma);
      draws[s, 1] = x; draws[s, 2] = y;
    }
    return draws;
  }
}
```

```
rstan::expose_stan_functions("binormal_rng.stan")
S <- 1000; mu_X <- 0; mu_Y <- 0; sigma_X <- 1; sigma_Y <- 1; rho <- 0.75
colMeans(binormal_rng(S = 100, mu_X, mu_Y, sigma_X, sigma_Y, rho))
```

```
## [1] 0.1456386 0.1106789
```

Markov Processes

- A Markov process is a sequence of random variables with a particular dependence structure where the future is conditionally independent of the past given the present, but nothing is marginally independent of anything else
- An AR1 model is a linear Markov process
- Let X_s have conditional PDF $f_s\left(X_s \middle| X_{s-1}\right)$. Their joint PDF is

$$f\left(X_0, X_1, \ldots, X_{S-1}, X_S\right) = f_0\left(X_0\right) \prod_{s=1}^{S} f_s\left(X_s \middle| X_{s-1}\right)$$

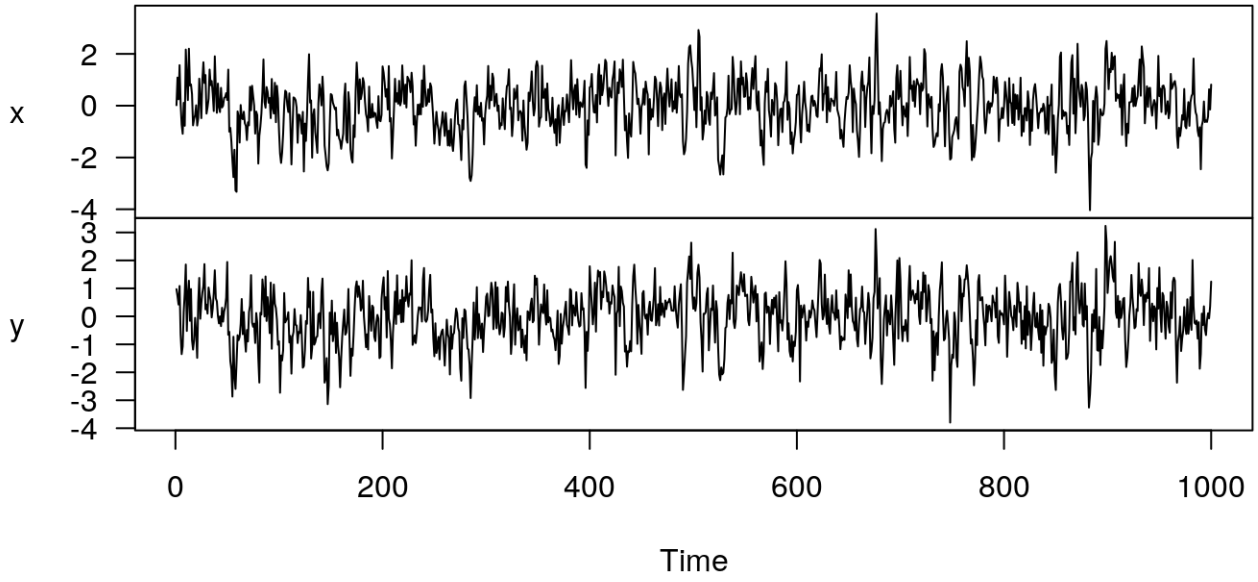
- Can we construct a Markov process such that the marginal distribution of X_S is a given target distribution as $S \uparrow \infty$?
- If so, they you can get a random draw — or a set of dependent draws — from the target distribution by letting that Markov process run for a long time
- Basic idea is that you can marginalize by going through a lot of conditionals

What the BUGS Software Family Essentially Does

```
library(Runuran) # defines ur() which draws from the approximate ICDF via pinv.new()
BUGSish <- function(log_kernel, # function of theta outputting posterior log-kernel
                    theta,      # starting values for all the parameters
                    ...,        # additional arguments passed to log_kernel
                    LB = rep(-Inf, K), UB = rep(Inf, K), # optional bounds on theta
                    S = 1000) { # number of posterior draws to obtain
  K <- length(theta); draws <- matrix(NA, nrow = S, ncol = K)
  for(s in 1:S) { # these loops are slow, as is approximating the ICDF | theta[-k]
    for (k in 1:K) {
      full_conditional <- function(theta_k)
        return(log_kernel(c(head(theta, k - 1), theta_k, tail(theta, K - k)), ...))
      theta[k] <- ur(pinv.new(full_conditional, lb = LB[k], ub = UB[k], islog = TRUE,
                              uresolution = 1e-8, smooth = TRUE, center = theta[k]))
    }
    draws[s, ] <- theta
  }
  return(draws)
}
```

Gibbs Sampling a la BUGS

```
xy <- BUGSish(binormal_lpdf, theta = c(0,0),
              mu_X, mu_Y, sigma_X, sigma_Y, rho)
colnames(xy) <- c("x", "y")
plot(as.ts(xy), main = "")
```



Effective Sample Size of Markov Chain Output

- If a Markov Chain mixes fast enough for the MCMC CLT to hold, then
  - The Effective Sample Size is $n_{eff} = \frac{S}{1+2\sum_{k=1}^{\infty}\rho_k}$, where ρ_k is the ex ante autocorrelation between two draws that are k iterations apart
  - The MCMC Standard Error of the mean of the S draws is $\frac{\sigma}{\sqrt{n_{eff}}}$ where σ is the true posterio standard deviation
- If $\rho_k = 0 \forall k$, then $n_{eff} = S$ and the MCMC-SE is $\frac{\sigma}{\sqrt{S}}$, so the Effective Sample Size is the number of INDEPENDENT draws that would be expected to estimate the posterior mean of some function with the same accuracy as the S DEPENDENT draws that you have from the posterior distribution
- Both have to be estimated and unfortunately, the estimator is not that reliable when the true Effective Sample Size is low (~5% of S)
- For this `BUGSish` sampler, n_{eff} is estimated to be ≈ 200 for both margins

Comparing Stan to Historical MCMC Samplers

- Only requires user to specify numerator of Bayes Rule
- Unlike Gibbs sampling, proposals are joint
- Like Gibbs sampling, proposals always accepted
- Like Gibbs sampling, tuning of proposals is (often) not required
- Unlike Gibbs sampling, the effective sample size is typically 25% to 125% of the nominal number of draws from the posterior distribution because ρ_1 can be negative in $n_{eff} = \frac{S}{1+2\sum_{k=1}^{\infty}\rho_k}$
- Unlike Gibbs sampling, Stan produces warning messages when things are not going swimmingly. Do not ignore these!
- Unlike BUGS, Stan does not permit discrete unknowns but even BUGS has difficulty drawing discrete unknowns with a sufficient amount of efficiency
- Metropolis-Hastings is another historical MCMC sampler that you may have heard about and Stan is always better than M-H

Hamiltonian Monte Carlo

- Instead of simply drawing from the posterior distribution whose PDF is $f\left(\boldsymbol{\theta}|\,\mathbf{y}\ldots\right) \propto f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right)$ Stan augments the "position" variables $\boldsymbol{\theta}$ with an equivalent number of "momentum" variables $\boldsymbol{\phi}$ and draws from

$$f\left(\boldsymbol{\theta}|\,\mathbf{y}\ldots\right) \propto \int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \prod_{k=1}^{K} \frac{1}{\sigma_k\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{\phi_k}{\sigma_k}\right)^2} f\left(\boldsymbol{\theta}\right) L\left(\boldsymbol{\theta};\mathbf{y}\right) d\phi_1 \ldots d\phi_K$$

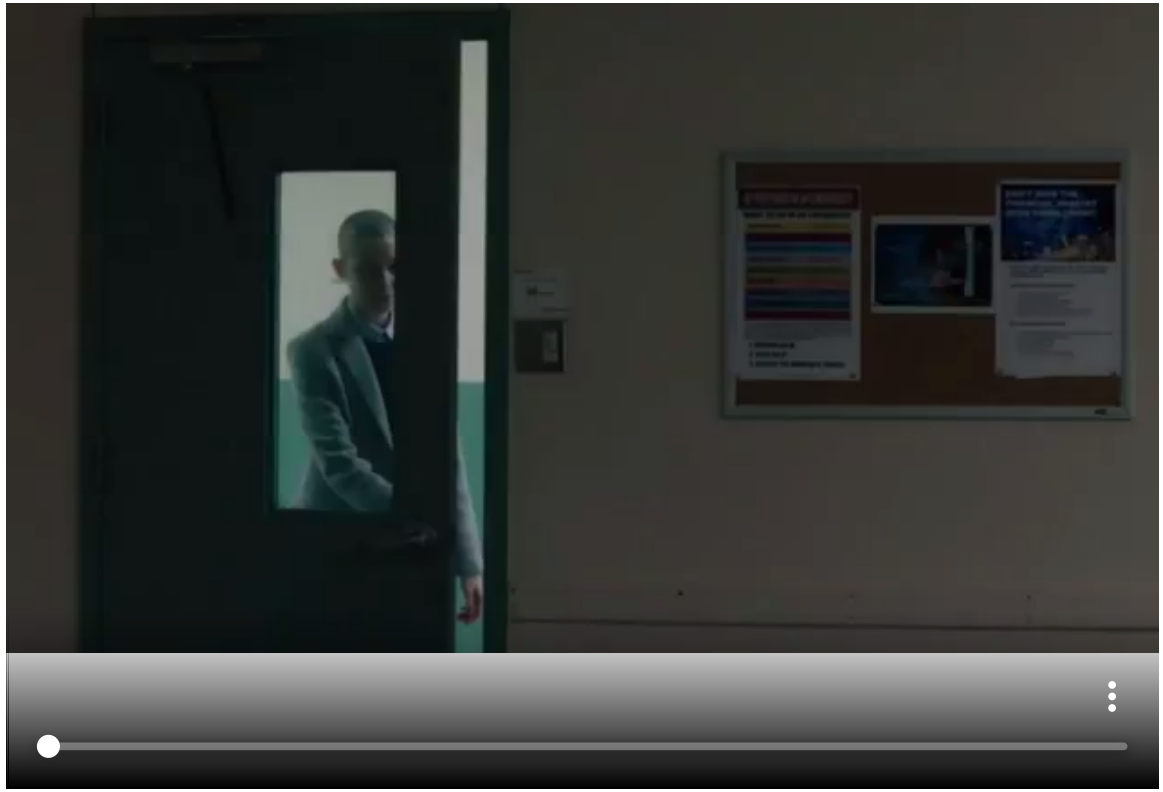
- Since the likelihood is NOT a function of ϕ_k, the posterior distribution of ϕ_k is the same as its prior, which is normal with a "tuned" standard deviation. So, at the s-th MCMC iteration, we just draw each $\widetilde{\phi}_k$ from its normal distribution.
- Using physics, the realizations of each $\widetilde{\phi}_k$ at iteration s "push" $\boldsymbol{\theta}$ from iteration $s-1$ through the parameter space whose topology is defined by the negated log-kernel of the posterior distribution: $-\ln f\left(\boldsymbol{\theta}\right) - \ln L\left(\boldsymbol{\theta};\mathbf{y}\right)$
- See HMC.R demo on Canvas

Demo of Hamiltonian Monte Carlo

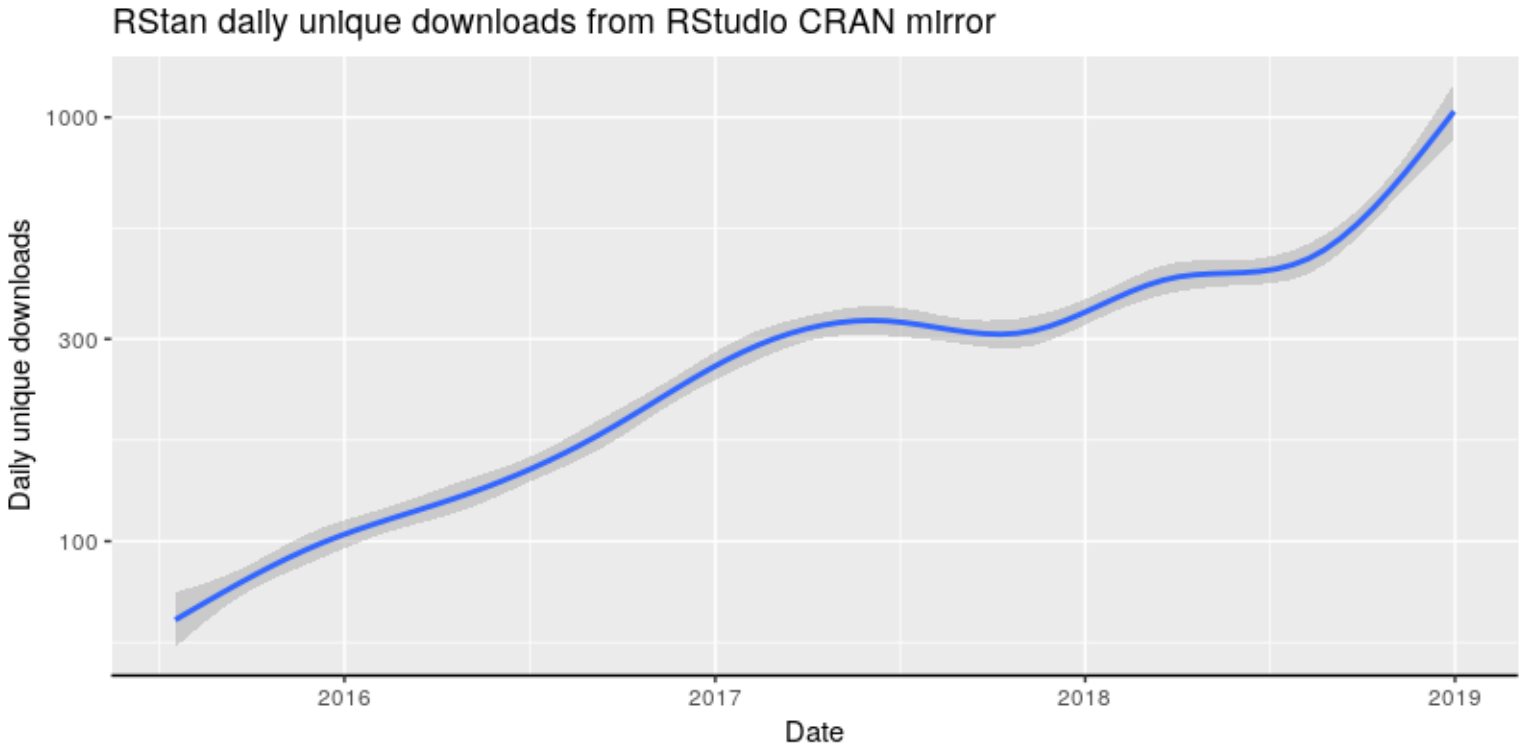
No U-Turn Sampling (NUTS)

- The location of $\boldsymbol{\theta}$ moving according to Hamiltonian physics at any instant would be a valid draw from the posterior distribution
- But (in the absence of friction) $\boldsymbol{\theta}$ moves indefinitely so when do you stop?
- Hoffman and Gelman (2014) proposed stopping when there is a "U-turn" in the sense the footprints turn around and start to head in the direction they just came from. Hence, the name No U-Turn Sampling.
- After the U-Turn, one footprint is selected with probability proportional to the posterior kernel to be the realization of $\boldsymbol{\theta}$ on iteration s and the process repeates itself
- NUTS discretizes a continuous-time Hamiltonian process in order to solve a system of Ordinary Differential Equations (ODEs), which requires a stepsize that is also tuned during the warmup phase

Season 3, Episode 9 of Billions



Stan Is Trending



What is Stan?

- Includes a high-level probabilistic programming language
- Includes a translator of high-level Stan syntax to somewhat low-level C++
- Includes new (and old) gradient-based algorithms for statistical inference, such as NUTS
- Includes a matrix and scalar math library that supports autodifferentiation
- Includes interfaces from R and other high-level software
- Includes R packages with pre-written Stan programs
- Includes (not Stan specific) post-estimation R functions
- Includes a large community of users and many developers

What is Autodifferentiation?

- A language like C++ supports operator overloading of `+`, `-`, etc. to do whatever
- In Stan, `c = a / b` computes `c` and both $\frac{\partial c}{\partial a} = \frac{1}{b}$ and $\frac{\partial c}{\partial b} = -\frac{a}{b^2}$
- Similarly, `d = g(c)` computes `d` and $\frac{\partial d}{\partial c} = g'(c)$
- Evaluating the chain rule is tedious for a human but easy for a computer
- Autodifferentiation allows the human to write an arbitrary (differentiable) mathematical expression and the (C++) compiler generates the code to compute the derivative automatically, even with vector / matrix expressions
- This is more accurate and / or faster than symbolic differentiation or numerical differentiation
- For Stan's purposes, Stan does autodifferentiation faster than anything else; see http://arxiv.org/abs/1509.07164

Using Stan via R

1. Write the program in a (text) .stan file w/ R-like syntax that ultimately defines a posterior log-kernel. We will not do this until May. Stan's parser, `rstan::stanc`, does two things
   - checks that program is syntactically valid & tells you if not
   - writes a conceptually equivalent C++ source file to disk
2. C++ compiler creates a binary file from the C++ source
3. Execute the binary from R (can be concurrent with 2)
4. Analyze the resulting samples from the posterior
   - Posterior predictive checks
   - Model comparison
   - Decision

Drawing from a Bivariate Normal with NUTS

```
library(rstan)
xy <- stan("binormal.stan", refresh = 0)
xy
```

```
## Inference for Stan model: binormal.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##       mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## x     0.00    0.03 1.01 -2.01 -0.66  0.00  0.67  2.01  1392    1
## y    -0.02    0.03 1.00 -2.00 -0.66 -0.03  0.63  1.95  1292    1
## lp__ -2.43    0.03 1.01 -5.10 -2.84 -2.10 -1.70 -1.45  1264    1
##
## Samples were drawn using NUTS(diag_e) at Mon Apr  1 04:25:13 2019.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Divergent Transitions

- NUTS only uses first derivatives
- First order approximations to Hamiltonian physiscs are fine for if either the second derivatives are constant or the discrete step size is sufficiently small
- When the second derviatives are very not constant across Θ, Stan can (easily) mis-tune to a step size that is not sufficiently small and θ_k gets pushed to $\pm\infty$
- When this happens there will be a warning message, suggesting to increase `adapt_delta`
- When `adapt_delta` is closer to 1, Stan will tend to take smaller steps
- Unfortunately, even as `adapt_delta` $\lim 1$, there may be no sufficiently small step size and you need to try to reparameterize your model

Exceeding Maximum Treedepth

- When the step size is small, NUTS needs many (small) steps to cross the "typical" subset of Θ and hit the U-turn point
- Sometimes, NUTS has not U-turned when it reaches its limit of 10 steps (by default)
- When this happens there will be a warning message, suggesting to increase `max_treedepth`
- There is always a sufficiently high value of `max_treedepth` to allow NUTS to reach the U-turn point, but increasing `max_treedepth` by 1 approximately doubles the wall time to obtain S draws

Low Bayesian Fraction of Missing Information

- When the tails of the posterior PDF are very light, NUTS can have difficulty moving through Θ efficiently
- This will manifest itself in a low (and possibly unreliable) estimate of n_{eff}
- When this happens there will be a warning message, saying that the Bayesian Fraction of Missing Information (BFMI) is low
- In this situation, there is not much you can do except increase S or preferably reparameterize your model to make it easier for NUTS

Runtime Exceptions

- Sometimes you will get a “informational” (not error, not warning) message saying that some parameter that should be positive is zero or some parameter that should be finite is infinite
- This means that a 64bit computer could not represent the number accurately
- If it only happens a few times and only during the warmup phase, do not worry
- Otherwise, you might try to use functions that are more numerically stable, which is discussed throughout the Stan User Manual

Bulk and Tail $\hat{R}$

- Sometimes you will get a warning message saying the bulk and / or bail $\hat{R}$ is too high
- These indicate that your Markov Chains have not converged to the same distribution
- You could simply try running them longer, but you may need to reparameterize or rethink your model
- Also, you can get a warning that the Effective Sample Size for the bulk and / or tail of the distribution is too low, in which case the Markov Chains may have converged but have not mixed well enough to obtain reliable inferences