Markov Chain Monte Carlo

Ben Goodrich

February 19, 2019

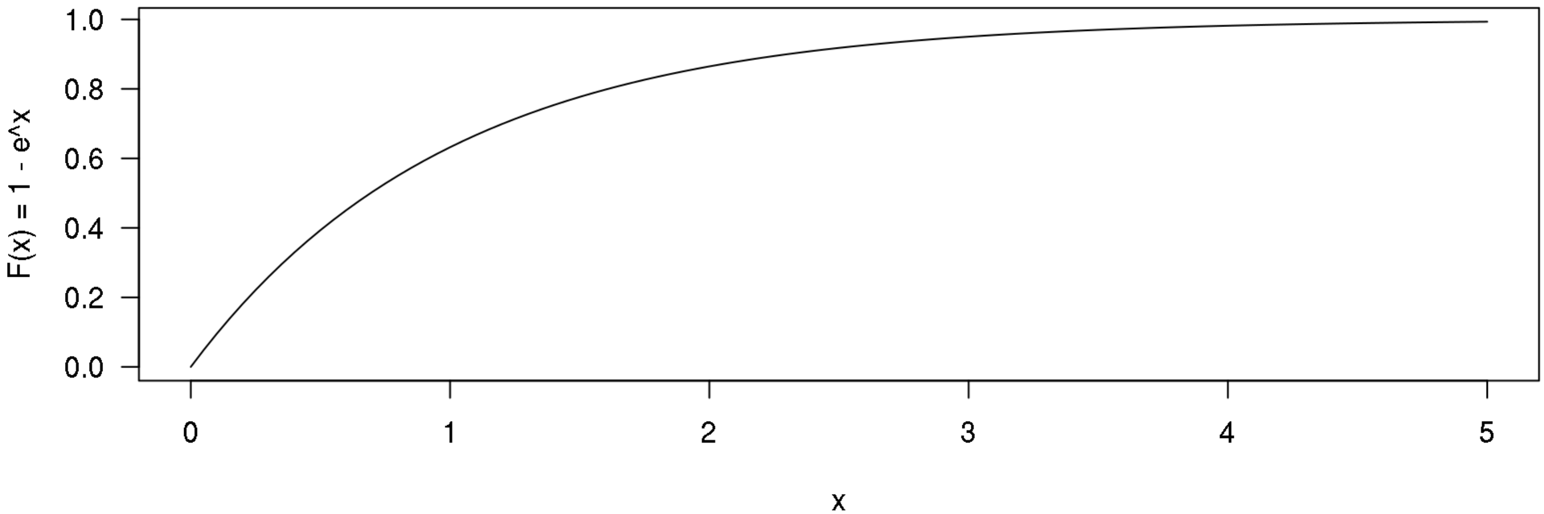
Obligatory Disclosure

- 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

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 $K = -1 + 2^{64} = 18,446,744,073,709,551,615$ 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 $\Omega_Y = \{0,1,\dots,K\}$ with each probability $\frac{1.0}{K}$
  2. Letting $\tilde{u} = \frac{\tilde{y}}{1.0+K}$, which casts to a double-precision denominator
- The CDF of the uniform distribution on (a,b) is $F\left(u|\, a,b\right) = \frac{u-a}{b-a}$ and the PDF is $f\left(u|\, a,b\right) = \frac{1}{b-a}$. 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
- For example, if $u(x) = 1 - e^x$, then $x(u) = \ln(1 - u) = F^{-1}(u)$
- If U is distributed standard uniform and $X = F^{-1}(U)$ what is the PDF of X?
- Since $u = F(x)$ has a constant density of 1 and $\frac{\partial}{\partial x} u(x) = \frac{\partial}{\partial x} F(x) = f(u(x))$, the PDF of X is whatever $f(x)$ is
- If $F(x)$ does not have an explicit form, you may have to numerically solve for $\tilde{x}$ such that $F(\tilde{x}) = \tilde{u}$

Bivariate Normal Distribution

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

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

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

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

where X is MARGINALLY normal and $Y|X$ 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 such 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
indep <- replicate(26, colMeans(binormal_rng(S = 100, mu_X, mu_Y, sigma_X, sigma_Y, rho)))
rownames(indep) <- c("x", "y"); colnames(indep) <- letters
```

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 $f_s\left(X_s \mid X_{s-1}\right)$. Their joint PDF is

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

- 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

Metropolis-Hastings Markov Chain Monte Carlo

- Suppose you want to draw from some distribution whose PDF is $f\left(\boldsymbol{\theta}|\ldots\right)$ but do not have a customized algorithm to do so.
- Initialize $\boldsymbol{\theta}$ to some value in Θ and then repeat S times:

  1. Draw a proposal for $\boldsymbol{\theta}$, say $\boldsymbol{\theta}'$, from a distribution whose PDF is $q\left(\boldsymbol{\theta}'|\ldots\right)$
  2. Let $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\ldots)}{f(\boldsymbol{\theta}|\ldots)} \frac{q(\boldsymbol{\theta}|\ldots)}{q(\boldsymbol{\theta}'|\ldots)}\}$. N.B.: Constants cancel so not needed!
  3. If α^* is greater than a standard uniform variate, set $\boldsymbol{\theta} = \boldsymbol{\theta}'$
  4. Store $\boldsymbol{\theta}$ as the s-th draw

- The S draws of $\boldsymbol{\theta}$ have PDF $f\left(\boldsymbol{\theta}|\ldots\right)$ but are NOT independent
- If $\frac{q(\boldsymbol{\theta}|\ldots)}{q(\boldsymbol{\theta}'|\ldots)} = 1$, called Metropolis MCMC

Metropolis Sampling from a Bivariate Normal

```
functions { /* saved as Metropolis_rng.stan in R's working directory */

  real binormal_lpdf(row_vector xy, real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    real beta = rho * sigma_Y / sigma_X; real sigma = sigma_Y * sqrt(1 - square(rho));
    if (is_inf(xy[1]) || is_inf(xy[2])) return negative_infinity();
    return normal_lpdf(xy[1] | mu_X, sigma_X) + // normal_lpdf is the logarithm of the normal PDF
           normal_lpdf(xy[2] | mu_Y + beta * (xy[1] - mu_X), sigma);
  }

  matrix Metropolis_rng(int S, real half_width, real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    matrix[S, 2] draws; real x = 0; real y = 0; // must initialize these before the loop so they persist
    for (s in 1:S) {
      real x_ = uniform_rng(x - half_width, x + half_width);
      real y_ = uniform_rng(y - half_width, y + half_width); // vvv can call previously-declared functions
      real alpha_star = exp(binormal_lpdf([x_, y_] | mu_X, mu_Y, sigma_X, sigma_Y, rho) -
                            binormal_lpdf([x , y ] | mu_X, mu_Y, sigma_X, sigma_Y, rho));
      if (alpha_star > uniform_rng(0, 1)) { // Q([x, y]) / Q[x_, y_] = 1 in this case
        x = x_; y = y_;
      } // otherwise leave x and y the same as they were on iteration s - 1
      draws[s, 1] = x;  draws[s, 2] = y;
    } // x_, y_, and alpha_star all get deleted here but x and y do not
    return draws;
  }
}
```

```
rstan::expose_stan_functions("Metropolis_rng.stan")
```

Efficiency in Estimating $\mathbb{E}X$ & $\mathbb{E}Y$ w/ Metropolis

```
means <- replicate(26, colMeans(Metropolis_rng(S, 2.75, mu_X, mu_Y, sigma_X, sigma_Y, rho)))
rownames(means) <- c("x", "y"); colnames(means) <- LETTERS; round(means, digits = 3)
```

```
##       A      B      C      D     E      F      G      H     I      J      K      L     M
## x 0.142 -0.147 -0.095 -0.072 0.082 -0.050 -0.194 -0.175 0.005  0.076 -0.130 -0.033 0.057
## y 0.167 -0.122 -0.013 -0.113 0.074 -0.001 -0.215 -0.163 0.014 -0.003  0.006 -0.013 0.036
##        N      O      P      Q     R      S     T      U      V     W      X      Y     Z
## x -0.074 -0.021 -0.057 -0.032 0.031  0.037 0.081 -0.034 -0.087 0.032 -0.113 -0.059 0.155
## y -0.043 -0.081 -0.113  0.050 0.076 -0.012 0.085 -0.088 -0.124 0.014 -0.003 -0.045 0.095
```

```
round(indep, digits = 3) # note S was 100, rather than 1000
```

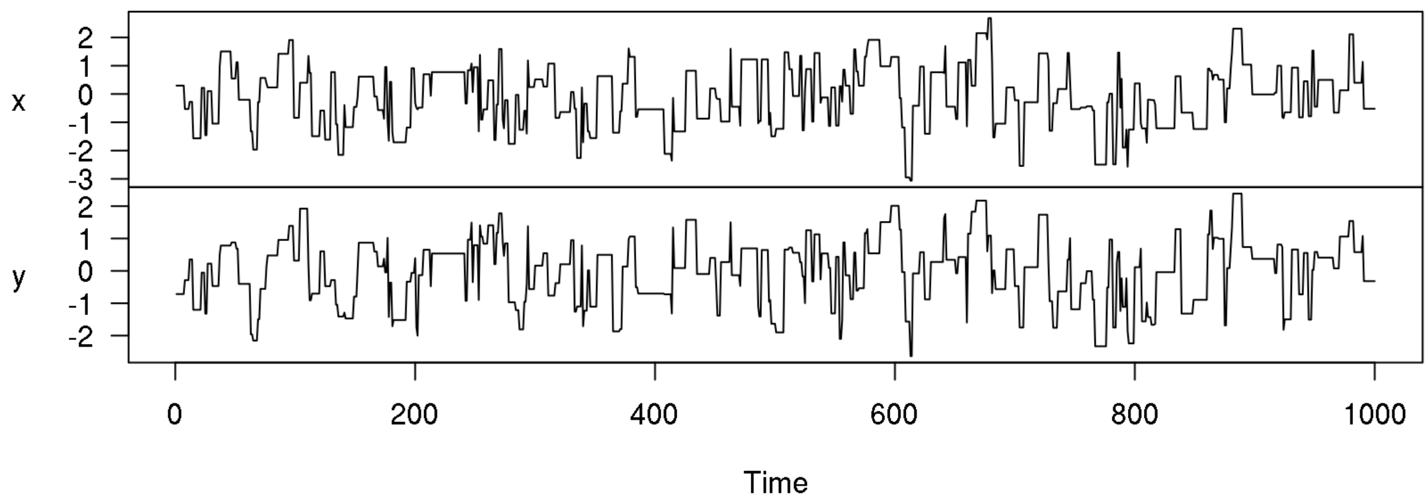
```
##       a      b      c     d     e      f      g      h      i      j      k      l     m
## x 0.146 -0.146 -0.148 0.106 0.059  0.010 -0.029 -0.135  0.033 -0.107 -0.115  0.029 0.034
## y 0.111 -0.053 -0.155 0.045 0.096 -0.026 -0.081 -0.054 -0.001 -0.083 -0.119 -0.027 0.115
##       n      o      p      q      r      s      t     u      v     w      x      y     z
## x 0.065 -0.067 -0.005 -0.135 -0.130 -0.325 -0.130 0.093 -0.117 0.248  0.023 -0.012 0.124
## y 0.013 -0.125  0.035 -0.104 -0.169 -0.180 -0.188 0.136 -0.076 0.145 -0.031  0.025 0.074
```

Autocorrelation of Metropolis MCMC

```
xy <- Metropolis_rng(S, 2.75, mu_X, mu_Y, sigma_X, sigma_Y, rho); nrow(unique(xy))
```

```
## [1] 236
```

```
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 Metropolis sampler, n_{eff} is estimated to be ≈ 100 for both margins

Gibbs Samplers

- Metropolis-Hastings where $q\left(\theta'_k \middle| \ldots\right) = f\left(\theta'_k \middle| \boldsymbol{\theta}_{-k} \ldots\right)$ and $\boldsymbol{\theta}_{-k}$ consists of all elements of $\boldsymbol{\theta}$ except the k-th
- $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\ldots)}{f(\boldsymbol{\theta}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f\left(\theta'_k|\boldsymbol{\theta}_{-k}\ldots\right)}\} = \min\{1, \frac{f\left(\theta'_k|\boldsymbol{\theta}_{-k}\ldots\right) f(\boldsymbol{\theta}_{-k}|\ldots)}{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots) f(\boldsymbol{\theta}_{-k}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f\left(\theta'_k|\boldsymbol{\theta}_{-k}\ldots\right)}\} = 1$ so θ'_k is ALWAYS accepted by construction. But θ'_k may be very close to θ_k when the variance of the "full-conditional" distribution of θ'_k given $\boldsymbol{\theta}_{-k}$ is small
- Can loop over k to draw sequentially from each full-conditional distribution
- Presumes that there is an algorithm to draw from the full-conditional distribution for each k. Most times have to fall back to something else.

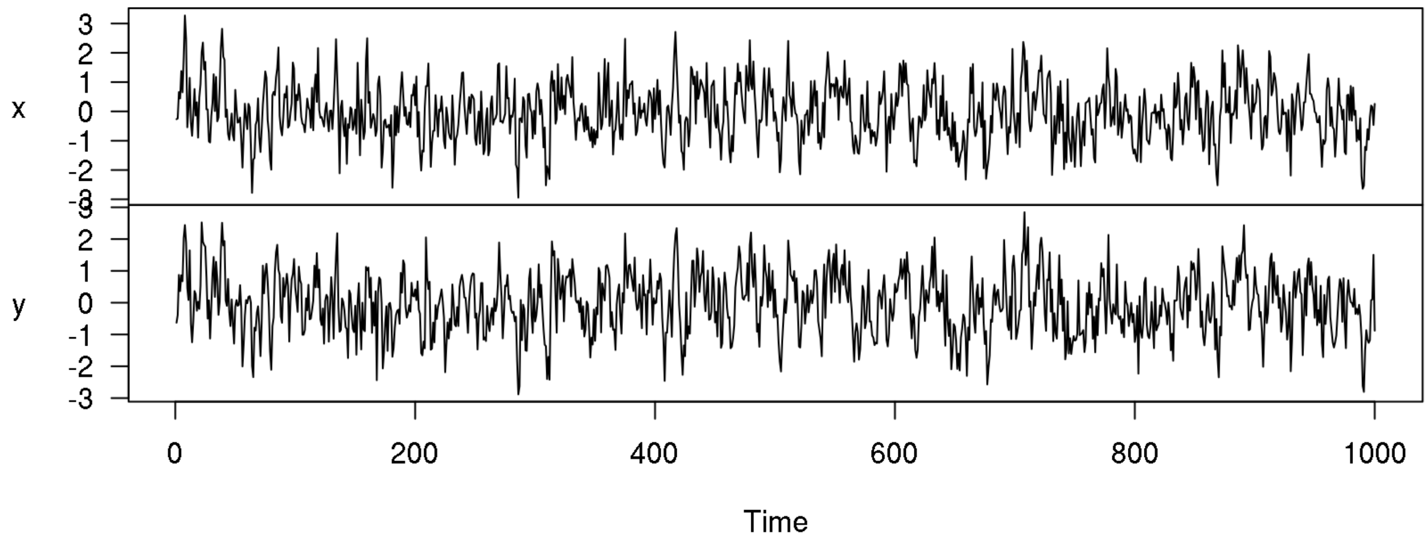
Gibbs Sampling from the Bivariate Normal

```
functions { /* saved as Gibbs_rng.stan in R's working directory */
  matrix Gibbs_rng(int S, real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    matrix[S, 2] draws; real x = 0; // must initialize before loop so that it persists
    real beta = rho * sigma_Y / sigma_X;
    real lambda = rho * sigma_X / sigma_Y;
    real sqrt1mrho2 = sqrt(1 - square(rho));
    real sigma_YX = sigma_Y * sqrt1mrho2;
    real sigma_XY = sigma_X * sqrt1mrho2; // this is smaller than in binormal_rng.stan !
    for (s in 1:S) {
      real y = normal_rng(mu_Y + beta * (x - mu_X), sigma_YX); // y needs a persistent x
      x = normal_rng(mu_X + lambda * (y - mu_Y), sigma_XY); // overwritten not redeclared
      draws[s, 1] = x; draws[s, 2] = y;
    } // y gets deleted here but x does not
    return draws;
  }
}
```

```
rstan::expose_stan_functions("Gibbs_rng.stan")
```

Autocorrelation of Gibbs Sampling: $n_{eff} \approx 300$

```
xy <- Gibbs_rng(S, mu_X, mu_Y, sigma_X, sigma_Y, rho)
colnames(xy) <- c("x", "y")
plot(as.ts(xy), main = "")
```



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: $n_{eff} \approx 200$

```
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 = "")
```