Markov Chain Monte Carlo for Bayesian Inference

Ben Goodrich

April 06, 2020

Standard Normal to General Normal

- PDF of the standard normal distribution is $f\left(z\right)=\frac{1}{\sqrt{2\pi}}e^{-\frac{1}{2}z^{2}}$
- If Z is distributed standard normal and $\sigma>0$, what is the distribution of $X\left(Z\right)=\mu+\sigma Z$?
- $\Pr\left(Z\leq z\right)=\Pr\left(Z\leq z\left(x\right)\right)$
- $z\left(x\right)=\frac{x-\mu}{\sigma}$ whose derivative is $\frac{\partial}{\partial x}z\left(x\right)=\frac{1}{\sigma}$
- $f\left(x\mid\mu,\sigma\right)=\frac{\partial}{\partial x}\Pr\left(Z\leq z\left(x\right)\right)=\boldsymbol{f\left(z\left(x\right)\right)\times\frac{\partial}{\partial x}z\left(x\right)}=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^{2}}$ which is the PDF for a general normal distribution
- $\mathbb{E}X=\mu+\sigma\mathbb{E}Z=\mu$
- $\mathbb{E}(X-\mu)^{2}=\mathbb{E}(\sigma Z)^{2}=\sigma^{2}\mathbb{E}Z^{2}=\sigma^{2}$

General Normal to Lognormal

- If X is distributed normal with expectation μ and standard deviation $\sigma > 0$, what is the PDF of $Y(X) = e^X$?

Poisson Likelihood with Lognormal Prior

- 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 lognormal distribution is again

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

- Poisson PMF for N observations with sum s is $f(y_1, \ldots, y_n \mid \lambda) = \frac{\lambda^s e^{-N\lambda}}{s!}$
- Bayes' Rule is $f(\lambda \mid \mu, \sigma, y_1, \ldots, y_n) \propto k(\lambda) = \lambda^{s-1} e^{-N\lambda - \frac{1}{2}\left(\frac{\ln \lambda - \mu}{\sigma}\right)^2}$
- The denominator of Bayes' Rule is $\int_0^\infty k(\lambda)\, d\lambda$ but is not elementary

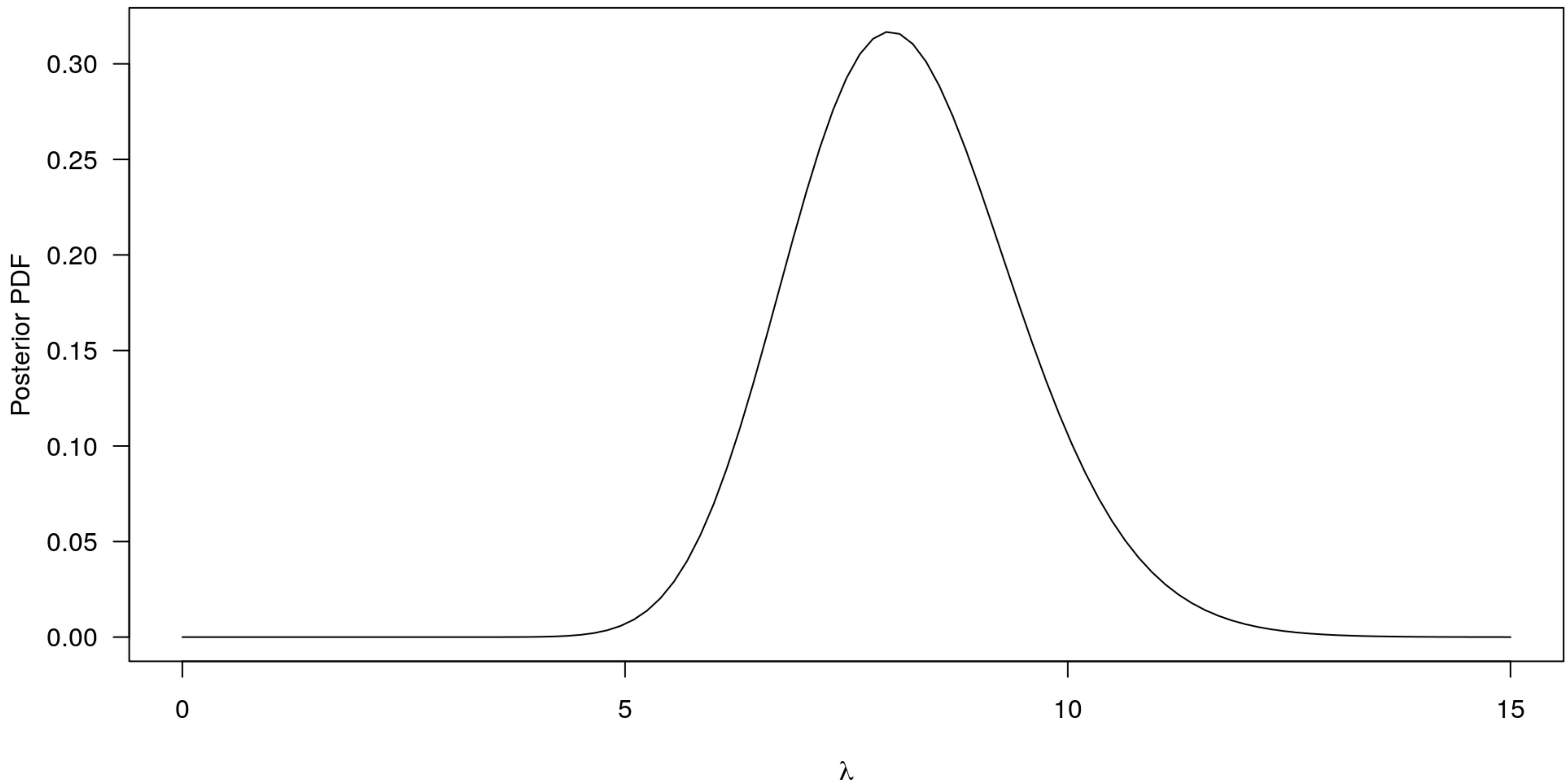
Posterior PDF

In breakout rooms, one person screenshare and the rest help to write a R function that evaluates the above posterior PDF:

1. Choose arbitrary real values of μ and $\sigma > 0$ integers $s \geq 0$ and $N > 0$
2. Write / wrap a function of λ that evaluates the lognormal prior PDF
3. Write / wrap a function of λ that evaluates the Poisson likelihood at $N\lambda$
4. Write a function of λ that multiplies the prior and likelihood together
5. Call the `integrate` function on the function from (4) to compute the denominator of Bayes' Rule
6. Write a function of λ that calls the function from (4) and divides by the constant from (5)

R Code for Previous Example

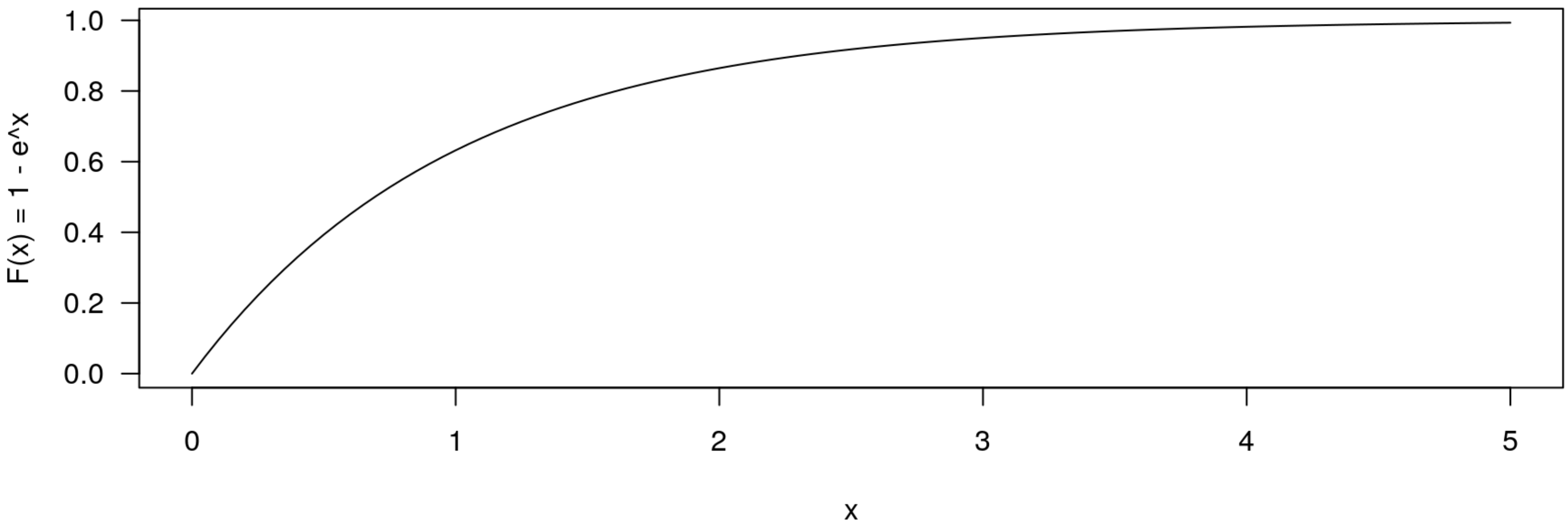
Plot from Previous Slide



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 $\Omega_Y = \{0, 1, \dots, M\}$ 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 (a la `rexp`), 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 $\mu > 0$ to make it exponential with rate $\frac{1}{\mu}$

Inverse CDF Sampling of Continuous RVs

- In principle, the previous implies an algorithm to draw from ANY univariate continuous distribution
- If U is distributed standard uniform, what is the PDF of $X = F^{-1}\left(U\right)$?
- $\Pr\left(U \leq u\right) = u = \Pr\left(U \leq u\left(x\right)\right)$
- $u\left(x\right) = F\left(x \mid \boldsymbol{\theta}\right)$ with derivative $f\left(x \mid \boldsymbol{\theta}\right)$
- So the PDF of X is $1 \times f\left(x \mid \boldsymbol{\theta}\right)$
- `rnorm(1, mu, sigma)` is implemented by `qnorm(runif(1), mu, sigma)`

Generalized λ Distribution (GLD)

- GLD is a four parameter (i.e. very flexible) continuous distribution defined by its inverse CDF

$$F^{-1}\left(u \mid m, r, a, s\right) = m + r \times F^{-1}\left(u \mid a, s\right) = m + r \times \frac{S\left(u \mid a, s\right) - S\left(\frac{1}{2} \mid a, s\right)}{S\left(\frac{3}{4} \mid a, s\right) - S\left(\frac{1}{4} \mid a, s\right)}$$

where m is the median, r is the inter-quartile range, $a \in (-1, 1)$ is an asymmetry parameter, $s \in (0, 1)$ is a tail steepness parameter, and $S\left(u \mid a, s\right)$ is a complicated increasing function

- The CDF and PDF of the GLD do not have explicit forms, which is not a problem for us

```
rstan::expose_stan_functions("quantile_functions.stan") # defines GLD_icdf() and GLD_rng() in R
source("GLD_helpers.R")                                 # brings into R GLD_solver() and GLD_solver_bounded()
args(GLD_solver); args(GLD_solver_bounded)              # these two functions solve for a and s
```

```
## function (lower_quartile, median, upper_quartile, other_quantile,
##     alpha, check = TRUE)
## NULL
```

```
## function (bounds, median, IQR, check = TRUE, ...)
## NULL
```

Using the Generalized λ Distribution

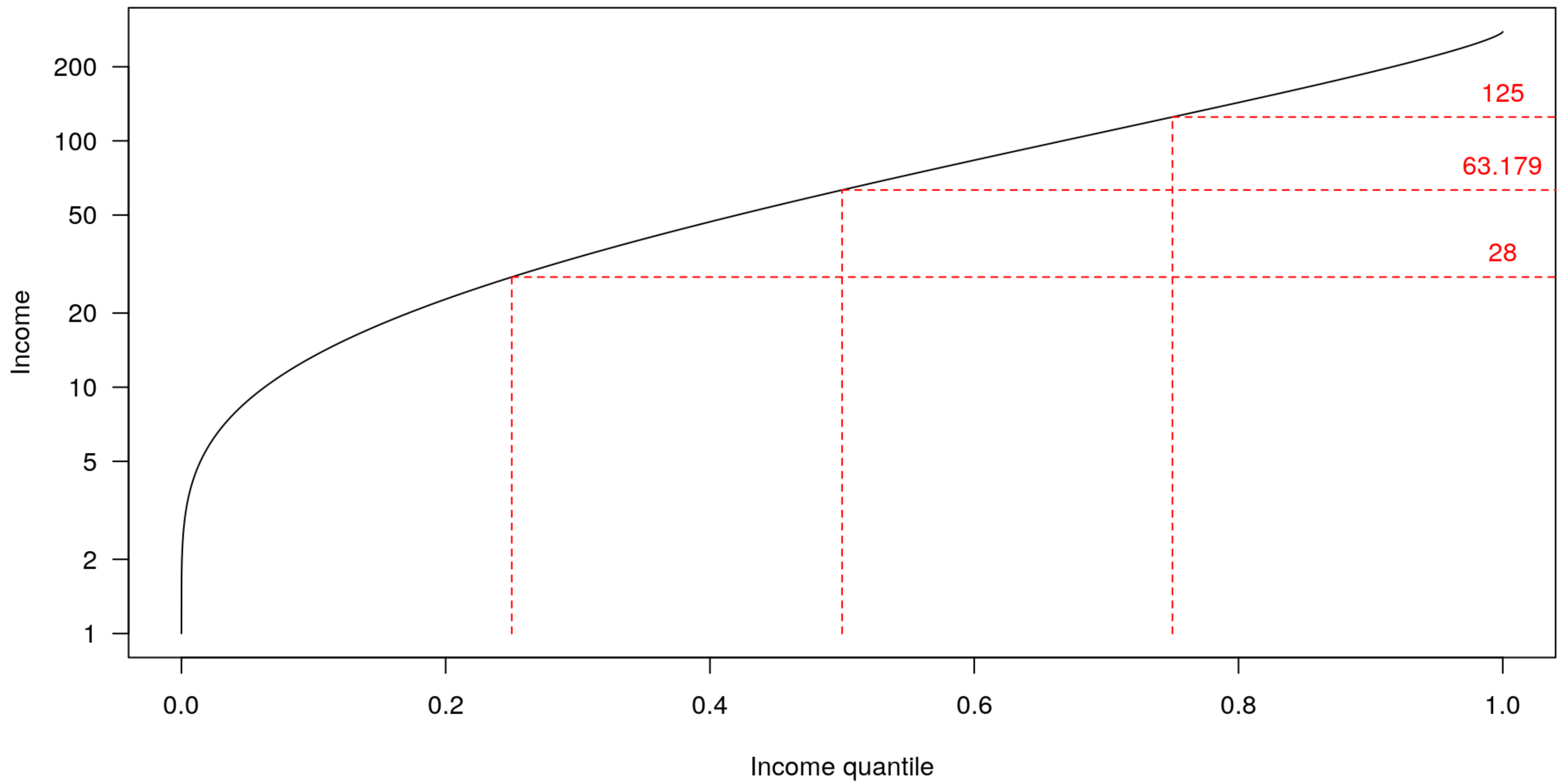
- In 2018, the 20% percentile of household income was \$25,600. The median was \$63,179, and the 80% percentile was \$130,000.

```
a_s <- GLD_solver(lower_quartile = log(28), median = log(63.179), upper_quartile = log(125),
                  other_quantile = 0, alpha = 0) # note warning
```

```
## Warning in GLD_solver(lower_quartile = log(28), median = log(63.179), upper_quartile =
## log(125), : solution implies a bounded upper tail at 5.62493307014712
```

```
Q <- Vectorize(GLD_icdf, vectorize.args = "p")
curve(exp(Q(u, median = log(63.179), IQR = log(125) - log(28),
            asymmetry = a_s[1], steepness = a_s[2])),
      from = 0, to = 1, xname = "u", xlab = "Income quantile", ylab = "Income", log = "y")
```

Plot from Previous Slide



Using the Bounded Generalized λ Distribution

- What do you think the probability that someone from around NYU who is tested for coronavirus will be positive? What is your prior median and IQR?

Prior Predictive Distribution

- The prior predictive distribution, which is the marginal distribution of future data integrated over the parameters, is formed by

  1. Draw $\tilde{\theta}$ from its prior distribution
  2. Draw $\tilde{y}$ from its conditional distribution given the realization of $\tilde{\theta}$
  3. Store the realization of $\tilde{y}$

- If you prior on θ is plausible, prior predictive distribution should be plausible

Prior Predictive Distribution Matching

- When the outcome is a small-ish count, a good algorithm to draw S times from the posterior distribution is to keep the realization of $\tilde{\theta}$ if and only if the realization of $\tilde{y}$ exactly matches the observed y

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 $Y | X$ 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, ] = [x_, y_]; // a row_vector
    }
    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
```

Bivariate Normal Log-PDF

In breakout rooms, one person screenshare and collectively fill in a function like this to evaluate the logarithm of the bivariate normal PDF from two slides ago:

```
functions {
  real binormal_lpdf(row_vector xy,
                     real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    // calculate intermediate constants
    // add two calls to normal_lpdf() which is like R's dnorm(..., log = TRUE)
    // but the first argument to normal_lpdf() is separated by a | from the other two
    // return their sum
  }
}
```

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 $f\left(\boldsymbol{\theta}\middle|\dots\right)$ 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 $q\left(\boldsymbol{\theta}'\middle|\dots\right)$
  2. Let $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\dots)}{f(\boldsymbol{\theta}|\dots)}\frac{q(\boldsymbol{\theta}|\dots)}{q(\boldsymbol{\theta}'|\dots)}\}$. 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 $f\left(\boldsymbol{\theta}\middle|\dots\right)$ but are NOT independent
- If $\frac{q(\boldsymbol{\theta}|\dots)}{q(\boldsymbol{\theta}'|\dots)} = 1$, called Metropolis MCMC such as $q\left(\theta \mid a, b\right) = \frac{1}{b-a}$

Metropolis Example

In breakout rooms, utilize `binormal_lpdf` to write a Stan function to draw S realizations of x and y from a bivariate normal distribution using the Metropolis algorithm with a uniform proposal distribution whose bounds are $x, y \mp h$

```
functions {
  real binormal_lpdf(row_vector xy,
                     real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    // copy this from above
  }

  matrix Metropolis_rng(int S, real h,
                        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
    for (s in 1:S) {
      // fill in draws[s,] by calling exp(binormal_lpdf(...)) to evaluate alpha*
    }
    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
```

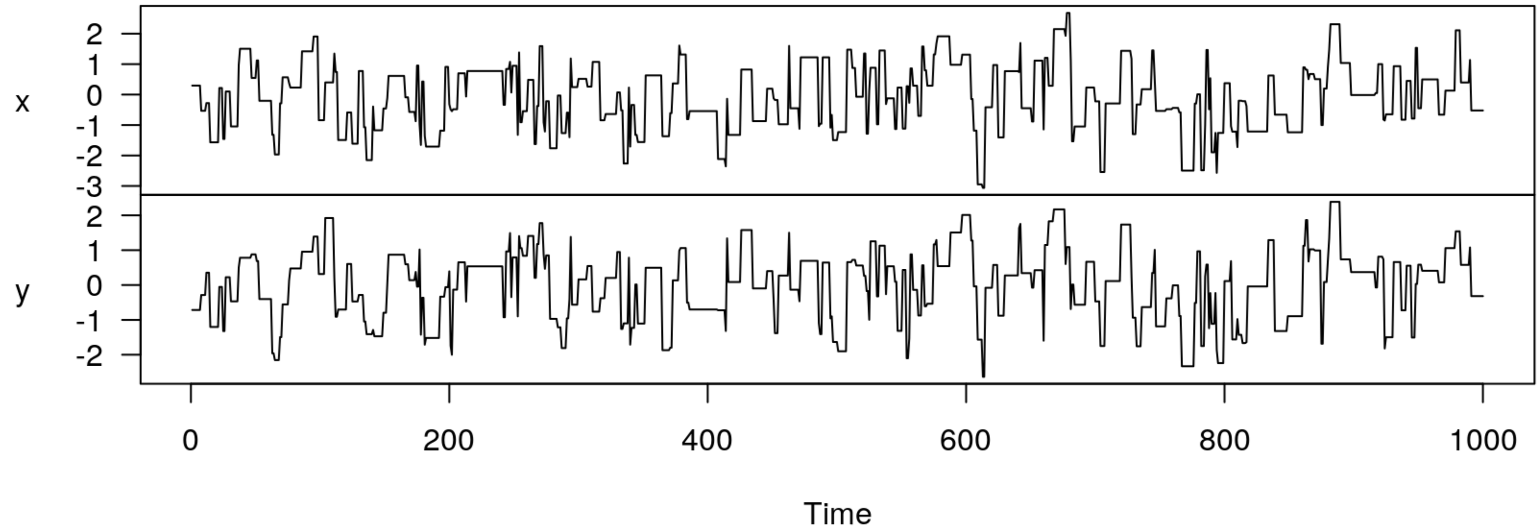
```
##        a      b      c     d     e      f      g      h      i      j      k      l     m
## x 0.146 -0.146 -0.148 0.106 0.059  0.010 -0.029 -0.135  0.033 -0.107 -0.115  0.029 0.034
## y 0.111 -0.053 -0.155 0.045 0.096 -0.026 -0.081 -0.054 -0.001 -0.083 -0.119 -0.027 0.115
##        n      o      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 the Metropolis example, 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
- $\alpha^* = \min\{1, \frac{f(\boldsymbol{\theta}'|\ldots)}{f(\boldsymbol{\theta}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)}\} = \min\{1, \frac{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)f(\boldsymbol{\theta}_{-k}|\ldots)}{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)f(\boldsymbol{\theta}_{-k}|\ldots)} \frac{f(\theta_k|\boldsymbol{\theta}_{-k}\ldots)}{f(\theta_k'|\boldsymbol{\theta}_{-k}\ldots)}\} = 1$ 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

In breakout rooms, write a `Gibbs_rng` function in the Stan language that draws S times from a bivariate normal distribution by repeatedly drawing from the normal distribution of $Y \mid X$ and then the normal distribution of $X \mid Y$

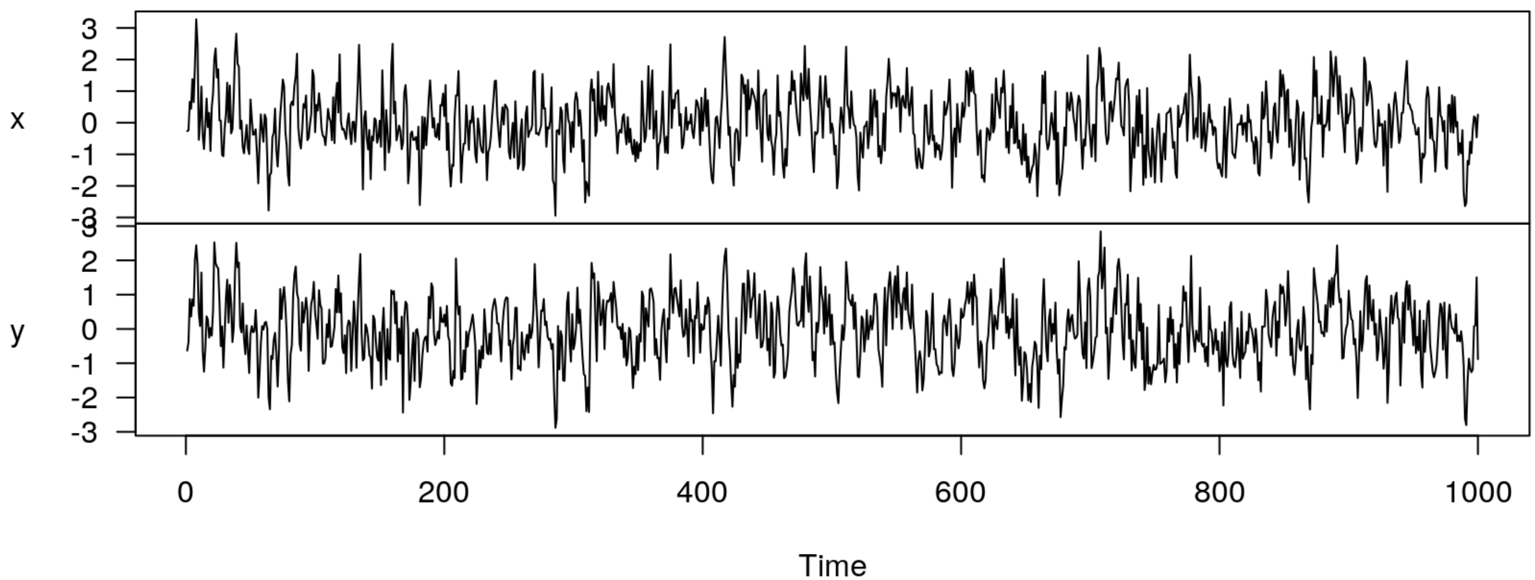
```
functions { /* saved as Gibbs_rng.stan in R's working directory */
  matrix Gibbs_rng(int S, real mu_X, real mu_Y, real sigma_X, real sigma_Y, real rho) {
    matrix[S, 2] draws; real x = 0; // must initialize before loop so that it persists
    // define many constants
    for (s in 1:S) {
      // fill in this part
    }
  }
}
```

Answer

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

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

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

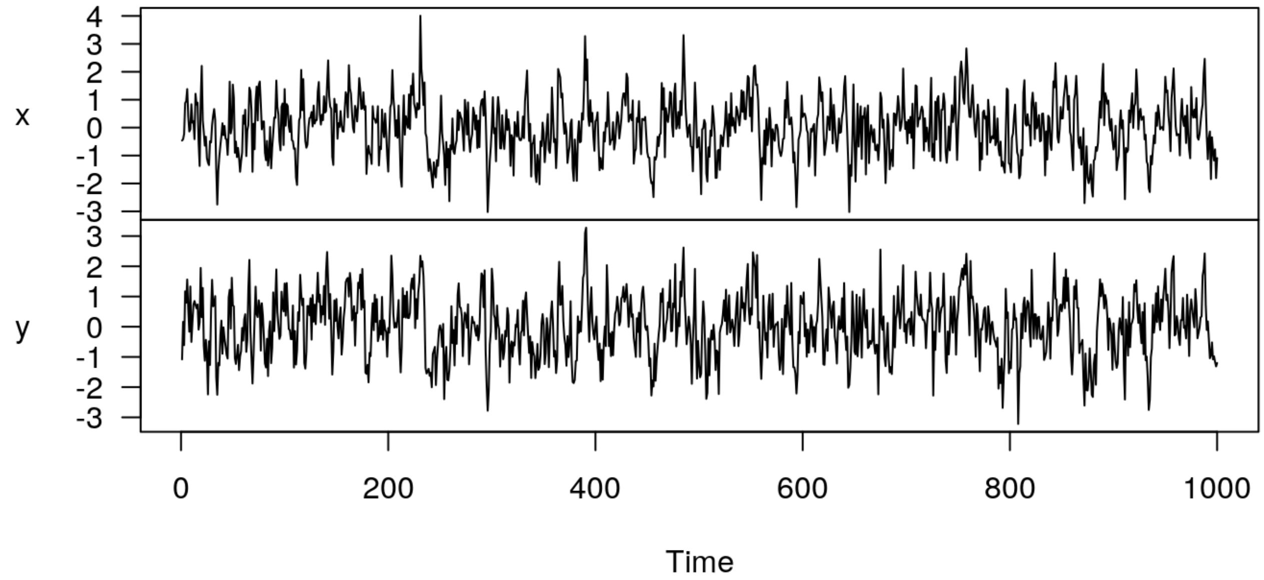


What the BUGS Software Family Essentially Does

```r
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

```r
rstan::expose_stan_functions("binormal_lpdf.stan")
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 = "")
```



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

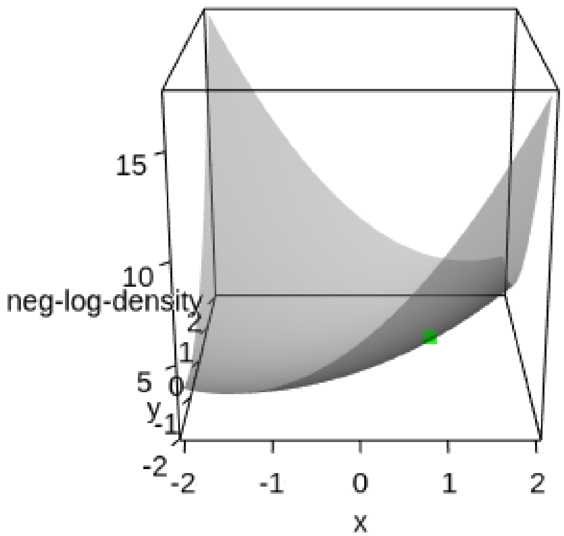
Hamiltonian Monte Carlo

- 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

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

- 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



Reverse Play Slower Faster Reset 1.00

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
- Video

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 and 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 Posterior Distribution with NUTS

```
library(rstan)
post <- stan("coronavirus.stan", refresh = 0,
             data = list(n = n, y = y, m = 0.3, IQR = 0.1,
                         asymmetry = a_s[1], steepness = a_s[2]))
post
```

```
## Inference for Stan model: coronavirus.
## 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
## p      0.79    0.00 0.10  0.56  0.73  0.81  0.87   0.94  1397    1
## theta  0.37    0.00 0.03  0.31  0.35  0.37  0.39   0.43  1286    1
## y_    82.71    0.22 9.87 63.00 76.00 82.00 89.00 103.00  2016    1
## lp__  -5.31    0.02 0.73 -7.37 -5.49 -5.03 -4.85  -4.81  1577    1
##
## Samples were drawn using NUTS(diag_e) at Mon Apr  6 02:39:50 2020.
## 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 tail $\hat{R}$ 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