Markov Chain Monte Carlo for Bayesian Inference

Ben Goodrich

March 07, 2022

Motivation for MCMC

- We generally cannot work out the denominator of Bayes' Rule

$$f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right) = \frac{f\left(\boldsymbol{\theta} \mid \ldots\right) L\left(\boldsymbol{\theta}; \mathbf{y}\right)}{\int_{-\infty}^{\infty} \ldots \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f\left(\boldsymbol{\theta} \mid \ldots\right) L\left(\boldsymbol{\theta}; \mathbf{y}\right) d\theta_1 d\theta_2 \ldots d\theta_J}$$

- We want to draw from the (posterior) distribution whose PDF is $f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right)$ without having to evaluate (the denominator of) that PDF
- Drawing is quite possible if we give up on the idea of INDEPENDENT draws, which leads to Markov Chain Monte Carlo (MCMC)

Three Popular Schemes for MCMC

1. Metropolis-Hastings: Draw a proposal, $\boldsymbol{\theta}'$, from an easy distribution, whose PDF is q and probabilistically accept it based on ratios $\frac{f(\boldsymbol{\theta}'|\mathbf{y},\ldots)}{f(\boldsymbol{\theta}|\mathbf{y},\ldots)} \times \frac{q(\boldsymbol{\theta})}{q(\boldsymbol{\theta}')}$. Since the constants cancel, you actually only need the kernels to evaluate this.
2. Gibbs: Factor the posterior kernel as $k\left(\theta_j \mid \boldsymbol{\theta}_{-j}, \mathbf{y}, \ldots\right) \times k\left(\boldsymbol{\theta}_{-j} \mid \mathbf{y}, \ldots\right)$ and for each $j = \{1, 2, \ldots, J\}$ draw from the known univariate distribution whose PDF is proportional to is $k\left(\theta_j \mid \boldsymbol{\theta}_{-j}, \mathbf{y}, \ldots\right)$
3. Stan: Take the logarithm of the posterior kernel and then differentiate that with respect to each θ_j. Move through the parameter space using Hamiltonian ODEs and random Gaussian wind gusts. Where the parameters are at any point in time is a draw from the posterior distribution.

- For the past decade, people (should) have been using (3) rather than (1) or (2)
- You don't need to know the details of (3) because you aren't physicsts and you don't need to know the details of (1) or (2) because they should be obsolete

Bivariate Normal Distribution

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

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

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

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

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

```
rbinorm <- function(n, mu_X, sigma_X, mu_Y, sigma_Y, rho) {
  x <- rnorm(n, mean = mu_X, sd = sigma_X)
  y <- rnorm(n, mean = mu_Y + rho * sigma_Y / sigma_X * (x - mu_X),
             sd = sigma_Y * sqrt((1 + rho) * (1 - rho)))
  return(cbind(x, y))
}
mu_X <- 0; mu_Y <- 0; sigma_X <- 1; sigma_Y <- 1; rho <- 0.75
indep <- replicate(26, colMeans(rbinorm(100, mu_X, sigma_X, mu_Y, sigma_Y, rho)))
rownames(indep) <- c("x", "y"); colnames(indep) <- letters
round(indep, digits = 3)
```

```
##        a      b      c      d     e     f      g      h     i      j      k     l     m
## x -0.096 -0.027 -0.044 -0.101 0.089 0.056 -0.091 -0.140 0.009 -0.013 -0.062 0.073 0.179
## y -0.075  0.032 -0.097 -0.085 0.064 0.135 -0.035 -0.176 0.015 -0.095  0.012 0.074 0.104
##        n     o     p      q      r     s    t      u      v      w      x     y     z
## x -0.057 0.245 0.080 -0.013 -0.106 0.099 0.18 -0.024 -0.112 -0.146 -0.051 0.018 0.164
## y -0.090 0.010 0.035  0.050 -0.149 0.086 0.21 -0.048 -0.156 -0.119 -0.098 0.008 0.239
```

Autoregressive 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 where $x_t = \mu \left(1 - \rho\right) \rho x_{t-1} + \epsilon_t$
- As $T \uparrow \infty$, the T-th realization is distributed normal with expectation μ and standard deviation $\frac{\sigma}{\sqrt{1-\rho^2}}$, where σ is the standard deviation of ϵ_t, as in

```
T <- 500; S <- 1000; x_T <- replicate(S, {
  x <- rpois(n = 1, 1)
  for (t in 1:T) x <- mu_X * (1 - rho) + rho * x + rnorm(n = 1, mean = 0, sd = sigma_X)
  return(x)
})
c(mean_diff = mean(x_T) - mu_X, sd_diff = sd(x_T) - sigma_X / sqrt(1 - rho^2))
```

```
##   mean_diff     sd_diff
##  0.01207659 -0.07252255
```

General Markov Processes

- Let X_t have conditional PDF $f_t\left(X_t \middle| X_{t-1}\right)$. Their joint PDF is

$$f\left(X_0, X_1, \ldots, X_{T-1}, X_T\right) = f_0\left(X_0\right) \prod_{t=1}^{T} f_t\left(X_t \middle| X_{t-1}\right),$$

  but we usually consider a special case where $f_t\left(X_t \middle| X_{t-1}\right) = f\left(X_t \middle| X_{t-1}\right)$
- Can we construct a (homogeneous) Markov process such that the marginal distribution of X_T has a sought after distribution as $T \uparrow \infty$?
- Yes, although generally with a nonlinear, homogeneous Markov process
- If so, they you can get a random draw — or a set of S 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, Gibbs, Stan, and others all satisfy this as $T \uparrow \infty$

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

```
means <- replicate(26, colMeans(Metropolis(S, 2.75, mu_X, sigma_X, mu_Y, 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 -0.105 -0.099 0.095 -0.093 0.319 0.125 -0.154 -0.006 0.089 -0.021 0.072 -0.122 -0.001
## y -0.101 -0.133 0.109 -0.136 0.257 0.061 -0.068 -0.037 0.106 -0.060 0.041 -0.186  0.060
##        N     O     P     Q      R      S      T      U     V      W      X     Y     Z
## x -0.080 0.125 0.016 0.114 -0.251 -0.101  0.003 -0.127 0.136 -0.013 -0.048 0.136 0.346
## y -0.096 0.112 0.021 0.000 -0.219 -0.111 -0.038 -0.142 0.169  0.060 -0.073 0.177 0.280
```

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

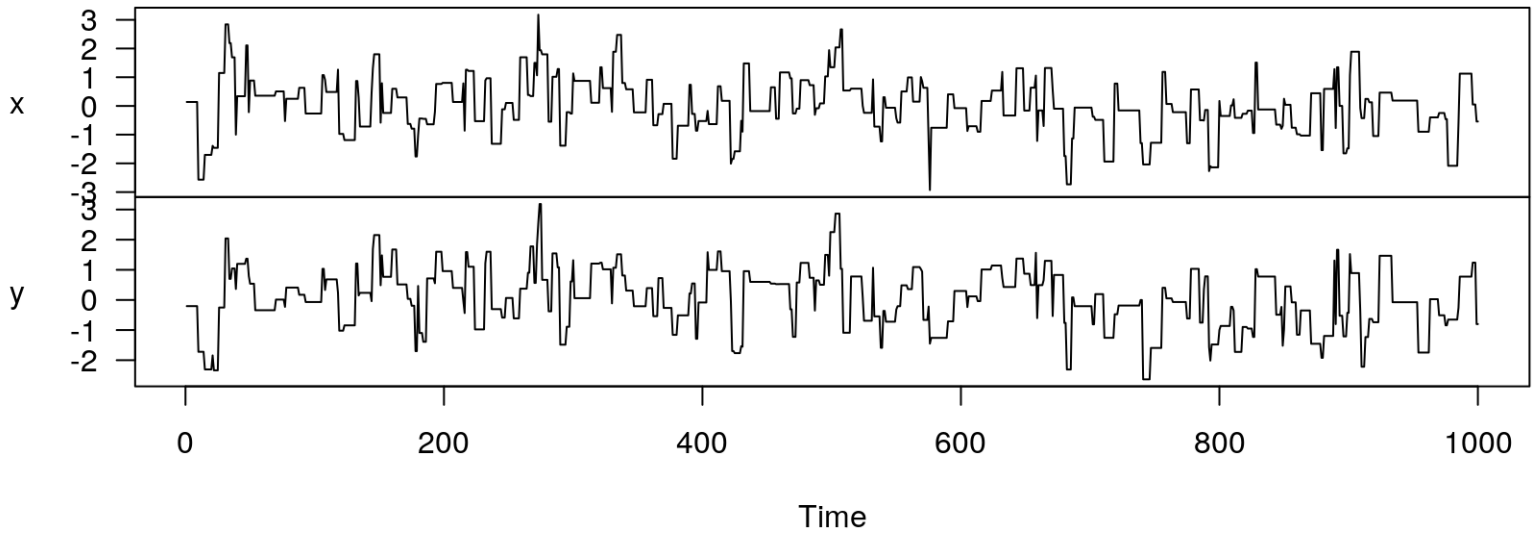
```
##        a      b      c      d     e     f      g      h     i      j      k     l     m
## x -0.096 -0.027 -0.044 -0.101 0.089 0.056 -0.091 -0.140 0.009 -0.013 -0.062 0.073 0.179
## y -0.075  0.032 -0.097 -0.085 0.064 0.135 -0.035 -0.176 0.015 -0.095  0.012 0.074 0.104
##        n     o     p      q      r     s    t      u      v      w      x     y     z
## x -0.057 0.245 0.080 -0.013 -0.106 0.099 0.18 -0.024 -0.112 -0.146 -0.051 0.018 0.164
## y -0.090 0.010 0.035  0.050 -0.149 0.086 0.21 -0.048 -0.156 -0.119 -0.098 0.008 0.239
```

Autocorrelation of Metropolis MCMC

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

```
## [1] 228
```

```
colnames(xy) <- c("x", "y"); plot(as.ts(xy), main = "")
```



Effective Sample Size of Markov Chain Output

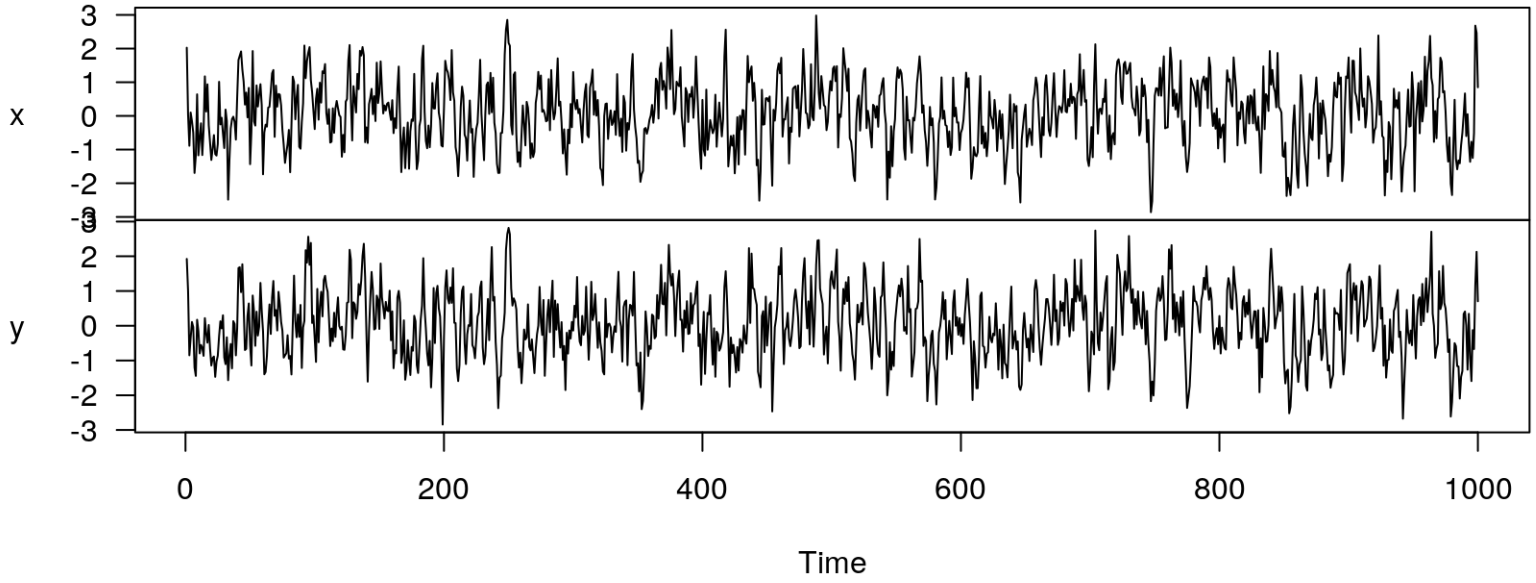
- If a Markov Chain mixes fast enough for the MCMC CLT to hold, then
  - The Effective Sample Size is $n_{eff} = \frac{S}{1+2\sum_{n=1}^{\infty}\rho_n}$, where ρ_n is the ex ante autocorrelation between two draws that are n iterations apart
  - The MCMC Standard Error of the mean of the S draws is $\frac{\sigma}{\sqrt{n_{eff}}}$ where σ is the true posterior standard deviation
- If $\rho_n = 0 \forall n$, 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 Sampling from the Bivariate Normal

```
Gibbs <- function(S, mu_X, sigma_X, mu_Y, sigma_Y, rho) {
  draws <- matrix(NA_real_, nrow = S, ncol = 2)
  x <- rpois(n = 1, 1) # arbitrary starting value
  beta   <- rho * sigma_Y / sigma_X
  lambda <- rho * sigma_X / sigma_Y
  sqrt1mrho2 <- sqrt( (1 + rho) * (1 - rho) )
  sigma_XY <- sigma_X * sqrt1mrho2 # these are both smaller than the
  sigma_YX <- sigma_Y * sqrt1mrho2 # marginal standard deviations!!!
  for (s in 1:S) {
    y <- rnorm(n = 1, mean = mu_Y + beta *   (x - mu_X), sd = sigma_YX)
    x <- rnorm(n = 1, mean = mu_X + lambda * (y - mu_Y), sd = sigma_XY)
    draws[s, ] <- c(x, y)
  }
  return(draws)
}
```

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

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

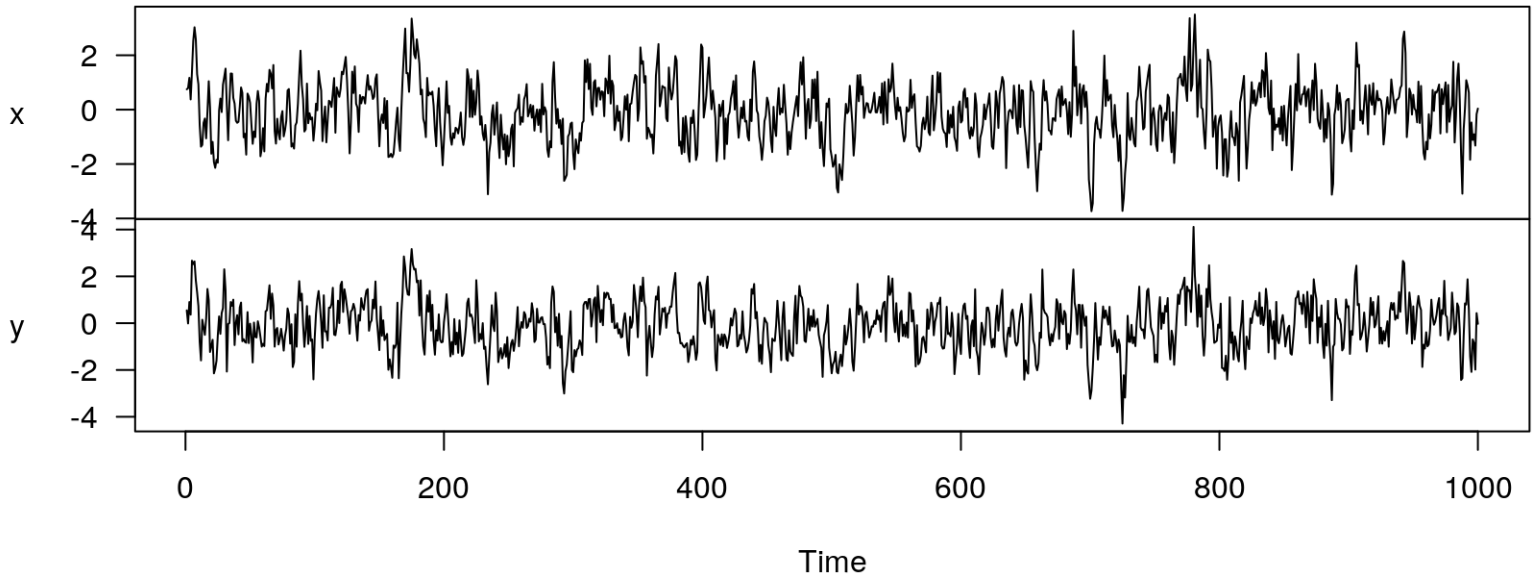


What the BUGS Software Family Essentially Does

```
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, J), UB = rep(Inf, J), # optional bounds on theta
                    S = 1000) { # number of posterior draws to obtain
  J <- length(theta); draws <- matrix(NA_real_, nrow = S, ncol = J)
  for(s in 1:S) { # these loops are slow, as is approximating the ICDF of theta[-k]
    for (j in 1:J) {
      full_conditional <- function(theta_j)
        return(log_kernel(c(head(theta, j - 1), theta_j, tail(theta, J - j)), ...))
      theta[j] <- ur(pinv.new(full_conditional, lb = LB[j], ub = UB[j], islog = TRUE,
                              uresolution = 1e-8, smooth = TRUE, center = theta[j]))
    }
    draws[s, ] <- theta
  }
  return(draws)
}
```

Gibbs Sampling a la BUGS: $n_{eff} \approx 200$

```
xy <- BUGSish(log_kernel = dbinorm, theta = c(x = -1, y = 1), mu_X, sigma_X,
              mu_Y, sigma_Y, rho, log = TRUE)
colnames(xy) <- c("x", "y")
plot(as.ts(xy), main = "")
```



Comparing Stan to Archaic MCMC Samplers

- Stan only requires user to specify kernel 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 $n_{eff} = \frac{S}{1+2\sum_{n=1}^{\infty}\rho_n}$
- Unlike Gibbs sampling, Stan produces warning messages when things are not going swimmingly. Do not ignore these (although we did on HW3)!
- Unlike BUGS, Stan does not permit discrete unknowns but even BUGS has difficulty drawing discrete unknowns with a sufficient amount of efficiency

Differentiating the Log Posterior Kernel

- Stan always works with log-PDFs or really log-kernels (in $\boldsymbol{\theta}$)

$$\ln f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right) = \ln f\left(\boldsymbol{\theta} \mid \ldots\right) + \ln L\left(\boldsymbol{\theta}; \mathbf{y}\right) - \ln f\left(\mathbf{y} \mid \ldots\right)$$

- The gradient of the log posterior PDF is the gradient of the log-kernel

$$\boldsymbol{\nabla} \ln f\left(\boldsymbol{\theta} \mid \mathbf{y}, \ldots\right) = \boldsymbol{\nabla} \ln f\left(\boldsymbol{\theta} \mid \ldots\right) + \boldsymbol{\nabla} \ln L\left(\boldsymbol{\theta}; \mathbf{y}\right) + \mathbf{0}$$

- This gradient is basically exact, and the chain rule can be executed by a C++ compiler without the user having to compute any derivatives

Hamiltonian Monte Carlo

- Stan pairs the J "position" variables $\boldsymbol{\theta}$ with J "momentum" variables $\boldsymbol{\phi}$ and draws from the joint posterior distribution of $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$
- Since the likelihood is NOT a function of ϕ_j, the posterior distribution of ϕ_j 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 $\widetilde{\phi}_j$ from its normal distribution.
- Using physics, the realizations of each $\widetilde{\phi}_j$ at iteration s "push" $\boldsymbol{\theta}$ from iteration $s-1$ for a random amount of time through the parameter space whose topology is defined by the (negated) log-kernel of the posterior distribution
- Although the ODEs must be solved numerically, the integral in "time" is one-dimensional and there are very good customized numerical integrators

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 and R code

Warnings You Should Be Aware Of

1. Divergent Transitions: This means the tuned stepsize ended up too big relative to the curvature of the log-kernel. Increase `adapt_delta` above its default value (usually 0.8) and / or use more informative priors
2. Hitting the maximum treedepth: This means the tuned stepsize ended up so small that it could not get all the way around the parameter space in one iteration. Increase `max_treedepth` beyond its default value of 10 but each increment will double the wall time, so only do so if you hit the max a lot
3. Bulk / Tail Effective Sample Size too low: This means the tuned stepsize ended up so small that adjacent draws have too much dependence. Increase the number of iterations or chains
4. $\widehat{R} > 1.01$: This means the chains have not converged. You could try running the chains longer, but there is probably a deeper problem.
5. Low Bayesian Fraction of Information: This means that you posterior distribution has really extreme tails. You could try running the chains longer, but there is probably a deeper problem.