lecture 07

mcmc

Metropolis algorithm; HMC and Stan; {brms}

Paper and Workshop

review

example exam questions

See handout

Algorithm: Rejection Sampling[a]

Inputs:

- The unnormalized posterior $f(\pi \mid y)$ on $[0, 1]$.
- Desired number of draws S.
- Envelope constant M such that $M > f(\pi)\ \forall\pi$.

Algorithm:

1. **Initialize:** Sets=1.
2. **Repeat until** $s = S$:
   (*a*) $z \sim \text{Uniform}(0, 1)$.
   (*b*) $u \sim \text{Uniform}(0, 1)$.
   (*c*) **Accept–reject step:**
   If $u \leq f(z)/M$, accept: set $\pi^{(s)} = z$, $s \leftarrow s + 1$.
   Otherwise, reject z and return to Step 2a.

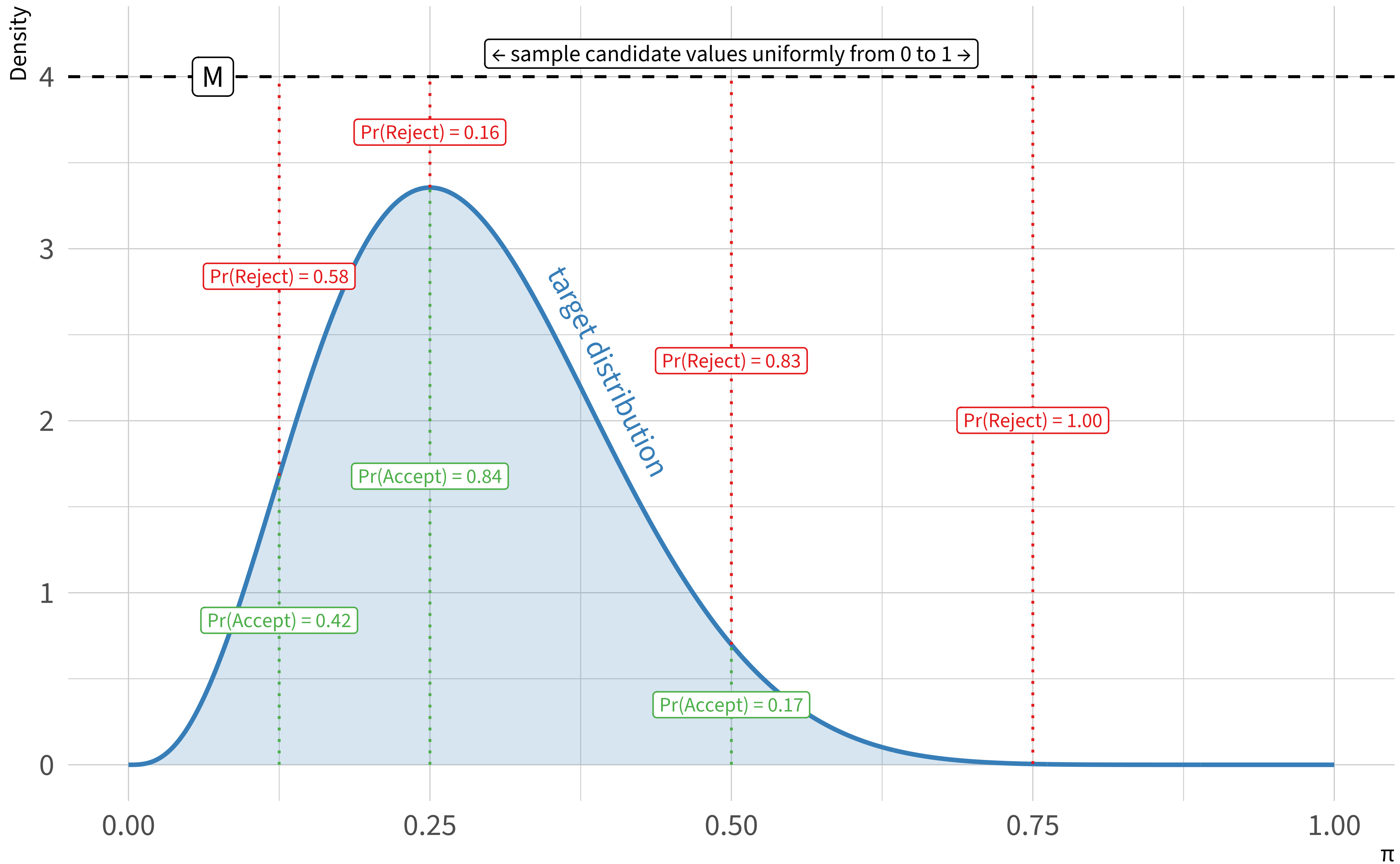
Output: $\pi^{(1)}, \pi^{(2)}, \ldots, \pi^{(S)} \sim f(\pi \mid y)$.

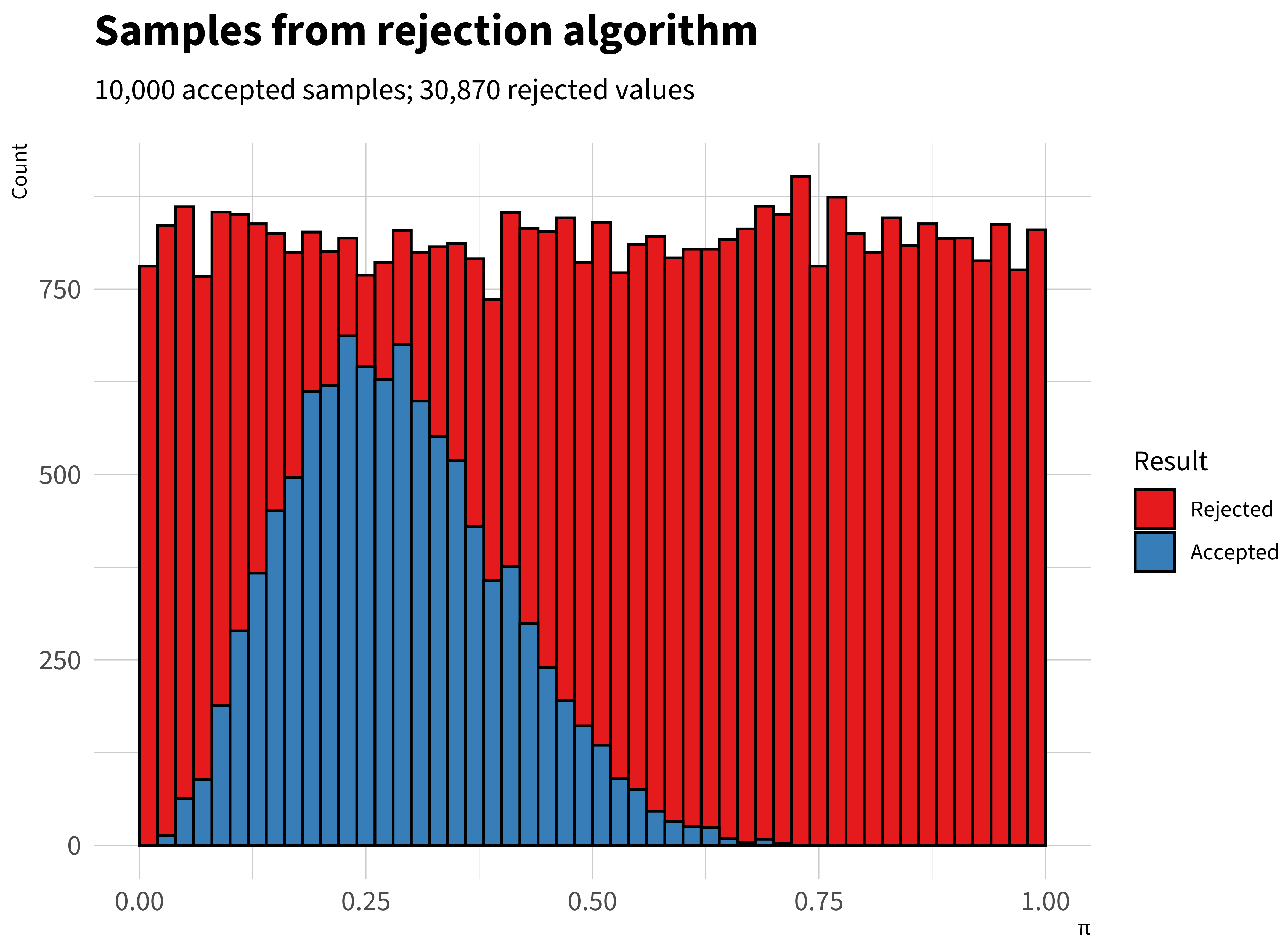
---

[a] To make the rejection algorithm simple, I've written it to apply specifically to the posterior for the Bernoulli model, which has support $[0, 1]$. The target density doesn't need to be a posterior and can have support other than $[0, 1]$. The proposal distribution doesn't have to be uniform. The key is that M is larger than the maximum of the target distribution and draws are accepted with probability $f(z)/M$.

Illustrating the logic of the rejection algorithm

For a beta(4, 10) target distribution.




Samples from rejection algorithm
10,000 accepted samples; 30,870 rejected values
Count
750
500
250
0
0.00
0.25
0.50
0.75
1.00
π
Result
Rejected
Accepted

Metropolis

Algorithm: Metropolis Sampling

Inputs:

- The unnormalized log-posterior $\log f(\theta \mid y)$.
- Desired number of iterations S.
- Tuning parameter τ controlling the width of the uniform proposal.

Algorithm:

1. **Initialize:** Choose $\theta^{(1)}$ and set $s = 1$.
2. **Repeat until $s = S$:**
   (a) **Propose:** Draw $z \sim \text{Uniform}(\theta^{(s)} - \tau,\ \theta^{(s)} + \tau)$.
   (b) **Compute log-acceptance:** $\Delta = \log f(z) - \log f(\theta^{(s)})$.
   (c) **Accept–reject step:**
      - Draw $u \sim \text{Uniform}(0, 1)$.
      - If $\log u \leq \Delta$, accept: set $\theta^{(s+1)} = z$; otherwise set $\theta^{(s+1)} = \theta^{(s)}$.[a]

   (d) Increment $s \leftarrow s + 1$.

Output: A Markov chain $\{\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(S)}\}$ with stationary distribution $f(\theta \mid y)$. Consecutive draws are dependent; discard a burn-in and tune τ to maintain an acceptance rate of roughly 0.2–0.5.

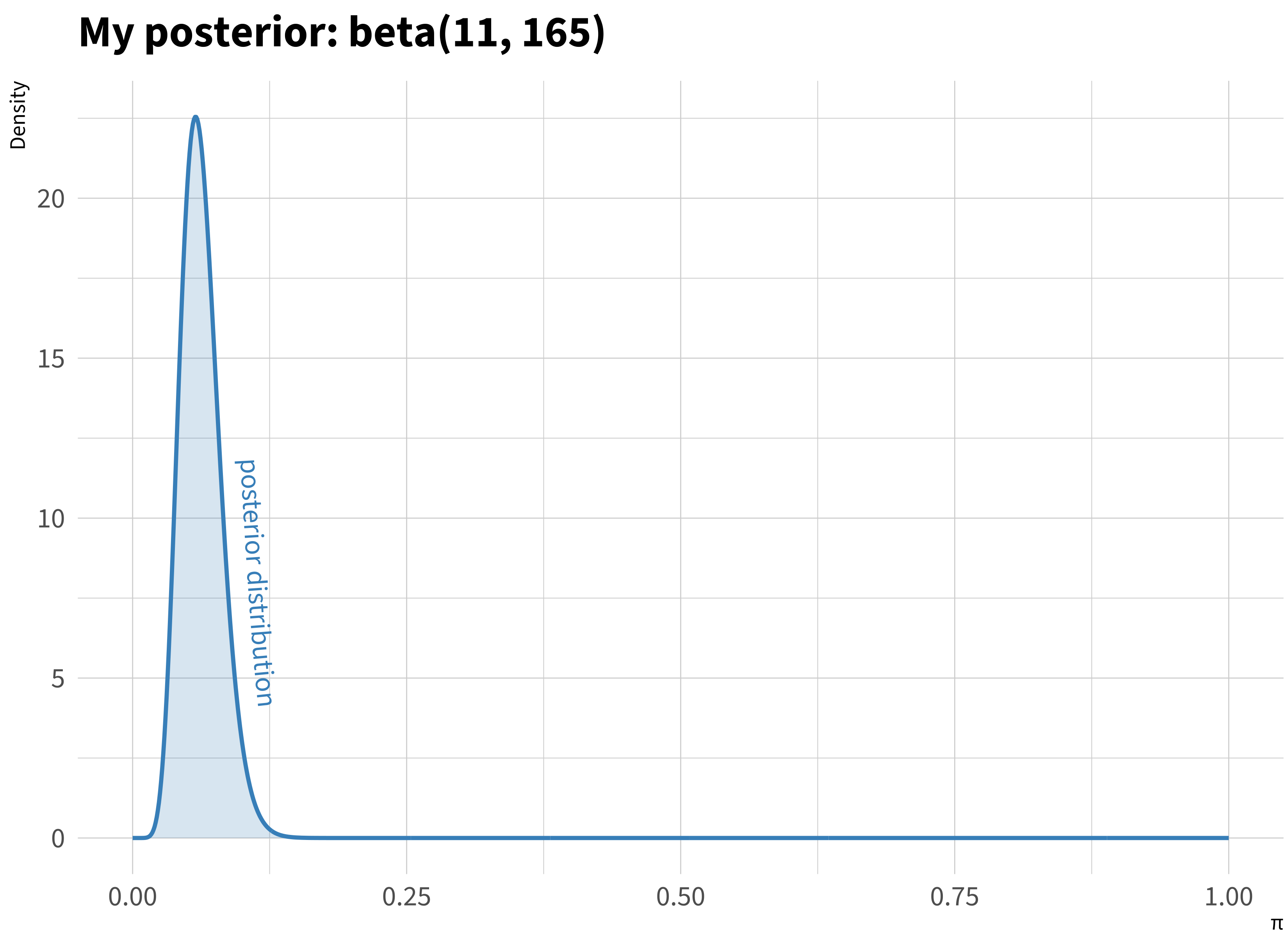
---

[a] This is equivalent to accepting with probability $\min\{1,\ f(z)/f(\theta^{(s)})\}$, meaning we always accept moves toward higher density and sometimes accept moves toward lower density.

```r
metrop <- function(logf, theta_start, S = 10000, tau = 0.1, ...) {
  # initialize matrix of samples with starting values
  k <- length(theta_start)
  samples <- matrix(NA_real_, nrow = S, ncol = k)
  samples[1, ] <- theta_start

  # proceed with algorithm
  for (s in 2:S) {
    # extract current location
    current <- samples[s - 1, ]

    # generate symmetric random-walk proposal
    proposed_move <- runif(k, -tau, tau)
    proposal <- current + proposed_move

    # acceptance step; equivalent to ratio on original scale
    delta <- logf(proposal, ...) - logf(current, ...)
    if (delta > 0) {
      accept <- TRUE
    } else {
      accept <- (log(runif(1)) <= delta)
    }

    # update samples
    if (accept) {
      samples[s, ] <- proposal
    } else {
      samples[s, ] <- current
    }
  }

  # return
  samples
}
```

Toothpaste Cap

Beta(11, 165)



https://gist.github.com/carlislerainey/37521a4fed2ca7d3c74b6408c35ccf24

Example

Logistic Regression

```
glm(formula = f, family = binomial, data = turnout)
            coef.est coef.se
(Intercept) 1.0578   0.1396
rs(age)     0.9934   0.1212
rs(educate) 1.1837   0.1370
rs(income)  1.0013   0.1535
racewhite   0.2508   0.1465
---
  n = 2000, k = 5
  residual deviance = 2024.0, null deviance = 2266.7 (difference = 242.8)
```

https://gist.github.com/carlislerainey/3147faeecf88fa92a897cf4a8f1ec3e2

MCMC

MCMC offers a powerful and general way to sample from an unnormalized (log-)posterior.

The Metropolis algorithm is one such method.

Hamiltonian Monte Carlo (HMC) via Stan is even better.

R-hat and burn-in

The first MCMC samples highly dependent on the starting values.

Because of this, you need to:

1. Discard the samples from a burn-in period.
2. Run multiple chains and check that R-hat is less than 1.01.

ESS and large samples

The MCMC samples are dependent on the previous samples.

Because of this, you need to:

1. Generate more samples that you would need if they were independent.
2. Use ESS to understand your effective sample size.

Why the Metropolis Algorithm Works

Symmetric proposals give equal opportunity.

- From any point x, the chance of proposing z is the same as proposing x from z.
- The proposal distribution itself doesn't favor any direction.

Asymmetric acceptance adds the right bias.

- Moves to higher density are always accepted.
- Moves to lower density are accepted with probability $f(z)/f(x)$.
- This rule makes the chain linger in high-density regions.

Balanced flows keep the target intact.

- Although proposals are symmetric, the acceptance rule ensures that the expected number of transitions $A \to B$ equals those from $B \to A$.
- This condition guarantees that the samples are stationary.

Long-run behavior reflects probability mass.

- Because the chain moves through the space according to these balanced transition rules, it spends time in each region in proportion to its probability under the target.
- In the long run, the sample frequencies mirror the target distribution.

Stan

Stan offers and *extremely efficient* alternative to Metropolis and other MCMC algorithms.

It uses a *hyper-optimized* version of Hamiltonian Monte Carlo.

Stan and it’s universe of supporting software is extremely well-documented and widely used.

```
data {
  int<lower=0> N;
  int<lower=1> K;
  array[N] int<lower=0, upper=1> y;
  matrix[N, K] X;
}
parameters {
  vector[K] beta;
}
model {
  beta ~ normal(0, 5);                // weakly informative prior
  y ~ bernoulli_logit(X * beta);      // logistic regression likelihood
}
```

https://gist.github.com/carlislerainey/3f59c3eb5cadfd35fa064083b92d7dab

{brms}

High-level interface to Stan

{brms} lets you specify Bayesian models using R's familiar formula syntax `y ~ x1 + x2`, then translates them automatically into efficient Stan code for sampling.

This is worth emphasizing—{brms} writes *efficient* Stan code.

Broad model support

It can fit all kinds of models.

Much more general than any particular fitting function we've seen so far.

Seamless post-processing and visualization

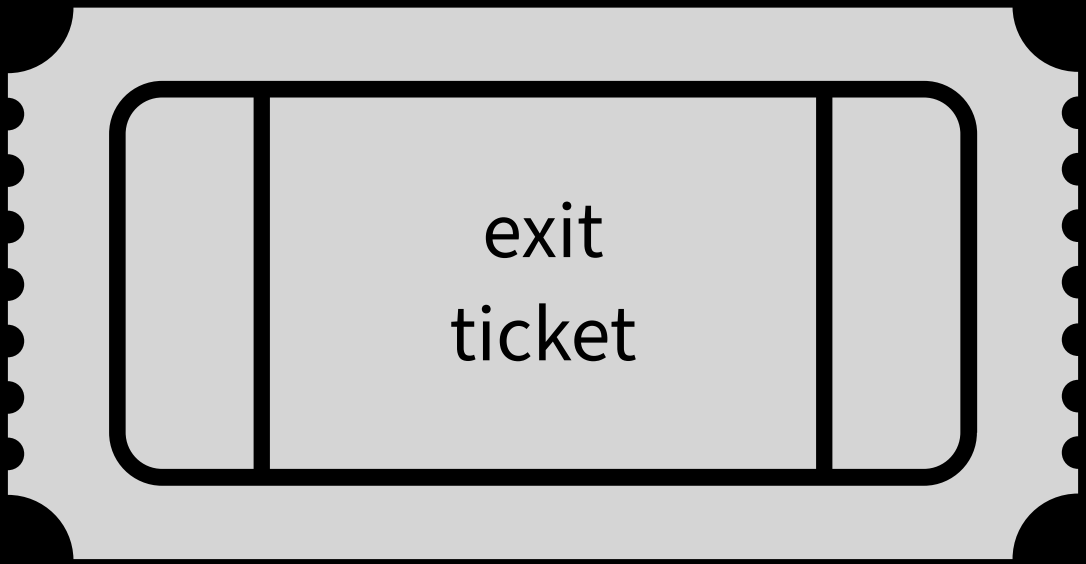
Built-in tools integrate with {bayesplot}, {posterior}, and {tidybayes} to summarize, diagnose, and visualize posterior draws without writing any Stan code directly.

Also {marginaleffects}!

https://gist.github.com/carlislerainey/a0b5cb6e92ded08d6f70bfc426e6610f

Why MCMC?

Exam



List three important ideas from today's class. For each, briefly connect it to one or more ideas from last week.