Lecture 8: Distributions and Simulations

STAT GR5206

Statistical Computing & Introduction to Data Science

Cynthia Rush
Columbia University

October 27, 2017

Course Notes

- I'll leave the last 20 minutes to of class for questions about the exam.
- Homework due Monday.
- Lab next week is a review session – nothing planned, bring questions.

Simulating Random Variables from Uncommon Probability Distributions

Simulating from Uncommon Probability Distributions

We study methods for transforming i.i.d. uniform $[0, 1]-$distributed random variables into a prescribed target distribution.

- Useful when we want to sample from a distribution that's not one with built-in `R` functions.
- Helpful in understanding how `R`'s simulation functions build off of PRNGs.

Discuss different methods, applicable to different classes of target distributions.

Target Distribution

Want X uniformly distributed on the set $\{0, 1, \ldots, n-1\}$, i.e.

$$P(X = k) = \frac{1}{n} \text{ for } k = 0, 1, \ldots, n-1.$$

- Maybe use a PRNG with a state space $0, 1, \ldots, n-1$? Not a good idea since we want a long period.
- Instead: first generate a rv $U \sim$ uniform $[0, 1]$, then transform with $X = \lfloor nU \rfloor$.
- Can show rigorously that X has the desired distribution.

```
n <- 10
samp <- 100
table(floor(n*runif(samp)))/samp
samp <- 10000
table(floor(n*runif(samp)))/samp
```

Target Distribution

Want X to take values in $\{1, 2, 3\}$ with probabilities $(1/2, 1/4, 1/4)$.

- Use the fact that for a rv $U \sim$ uniform $[0, 1]$ and an event $E = \{U \leq p\}$ then $P(E) = p$.
- Define X as follows:

$$X = \begin{cases} 1 & \text{if } 0 \leq U < 1/2 \\ 1 & \text{if } 1/2 \leq U < 3/4 \\ 1 & \text{if } 3/4 \leq U \leq 1 \end{cases}$$

Inverse Transform Method

- The Inverse Transform Method is a general method that can be used when the target distribution in one-dimensional.
- Uses the inverse cdf denoted F^{-1} of the target distribution:

$$F^{-1}(u) = \inf\{x \in \mathbb{R} | F(x) \geq u\}$$

  for all $u \in (0, 1)$.
- If F is bijective (i.e. F is strictly monotonically increasing with no jumps), then F^{-1} is the usual inverse and can be found by solving $F(x) = u$ for x.

Method

Whenever F^{-1} can be determined, then $X = F^{-1}(U)$ is such that $X \sim F$.

Why does this work?

$$\begin{aligned} P(X \leq x) &= P(F^{-1}(U) \leq x) \\ &\overset{(a)}{=} P(F(F^{-1}(U)) \leq F(x)) \\ &= P(U \leq F(x)) \\ &= F(x), \end{aligned}$$

where (a) follows by monotonicity of F.

Inverse Transform Method

Algorithm

1. Derive the inverse function F^{-1}. To do this:
   - Solve $F(x) = u$ for x to find $x = F^{-1}(u)$.
2. Write a function to compute $x = F^{-1}(u)$.
3. For each realization:
   - Generate a random value u from Uniform(0,1).
   - Compute $x = F^{-1}(u)$ as a realization from the target distribution.

Example: Target distribution is exponential with $\lambda = 2$.

The pdf of the exponential distribution is $f(x) = \lambda e^{-\lambda t}$, so the cdf is

$$F(x) = \int_0^x f(t)dt = \int_0^x \lambda e^{-\lambda t} dt = 1 - e^{-\lambda x}.$$

Now we invert the cdf.

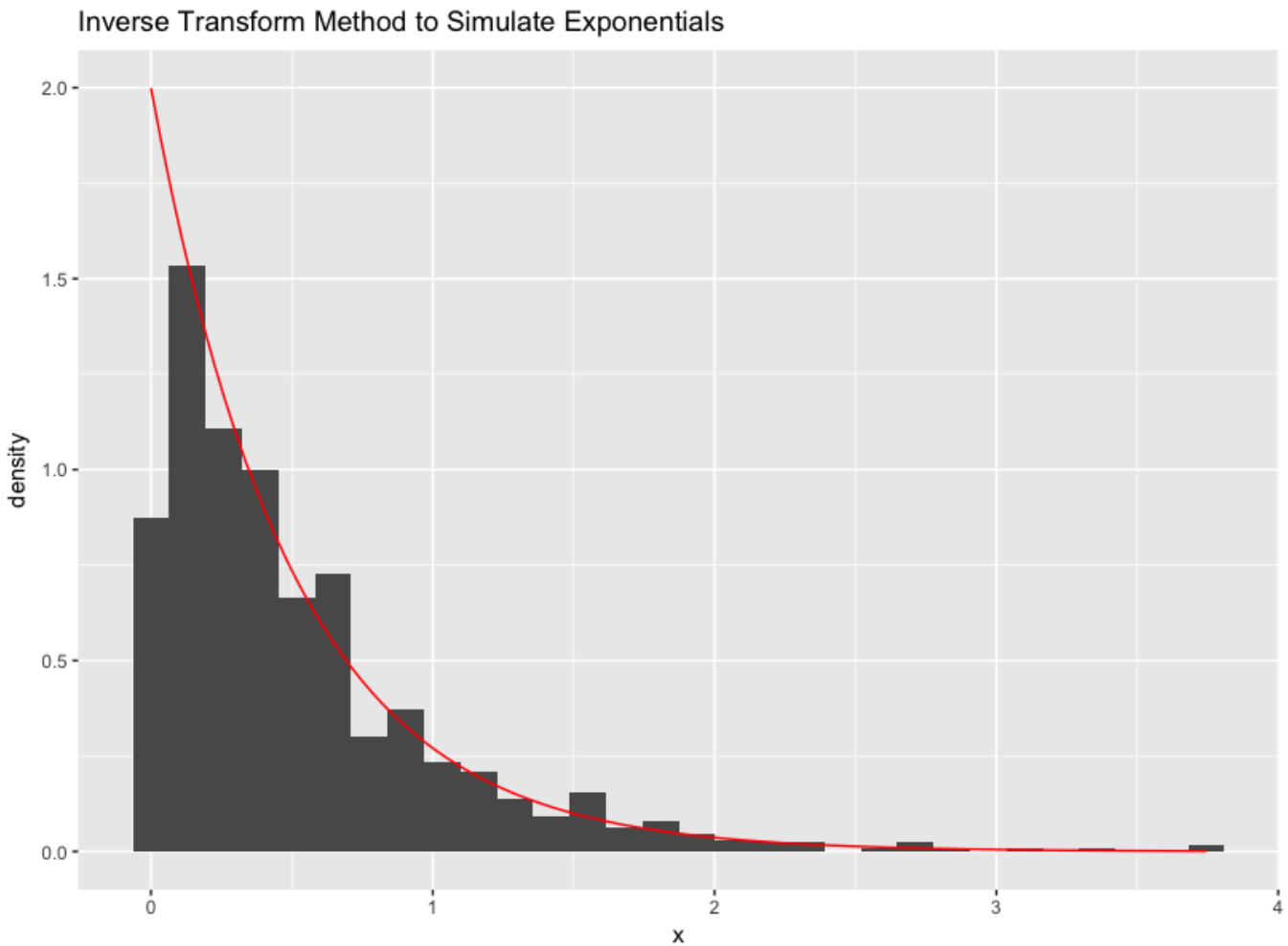
$$u = 1 - e^{-\lambda x} \quad \rightarrow \quad x = -\frac{1}{\lambda}\log(1-u)$$

Inverse Transform Method

Example: Target distribution is exponential with $\lambda = 2$.

```
lambda <- 2
n      <- 1000
u      <- runif(n) # Simulating uniform rvs

Finverse <- function(u, lambda) {
  # Function for the inverse transform
  stopifnot(u > 0 & u < 1)
  return(-(1/lambda)*log(1-u))
}
x <- Finverse(u, lambda)

ggplot(data = data.frame(x)) +
  geom_histogram(aes(x = x, y = ..density..)) +
  stat_function(mapping = aes(x = x), fun = dexp,
      args = list(rate = 2), color = "red") +
  labs(title = "Inverse Transform Method to Simulate Exponential
```

Inverse Transform Method: Exponential



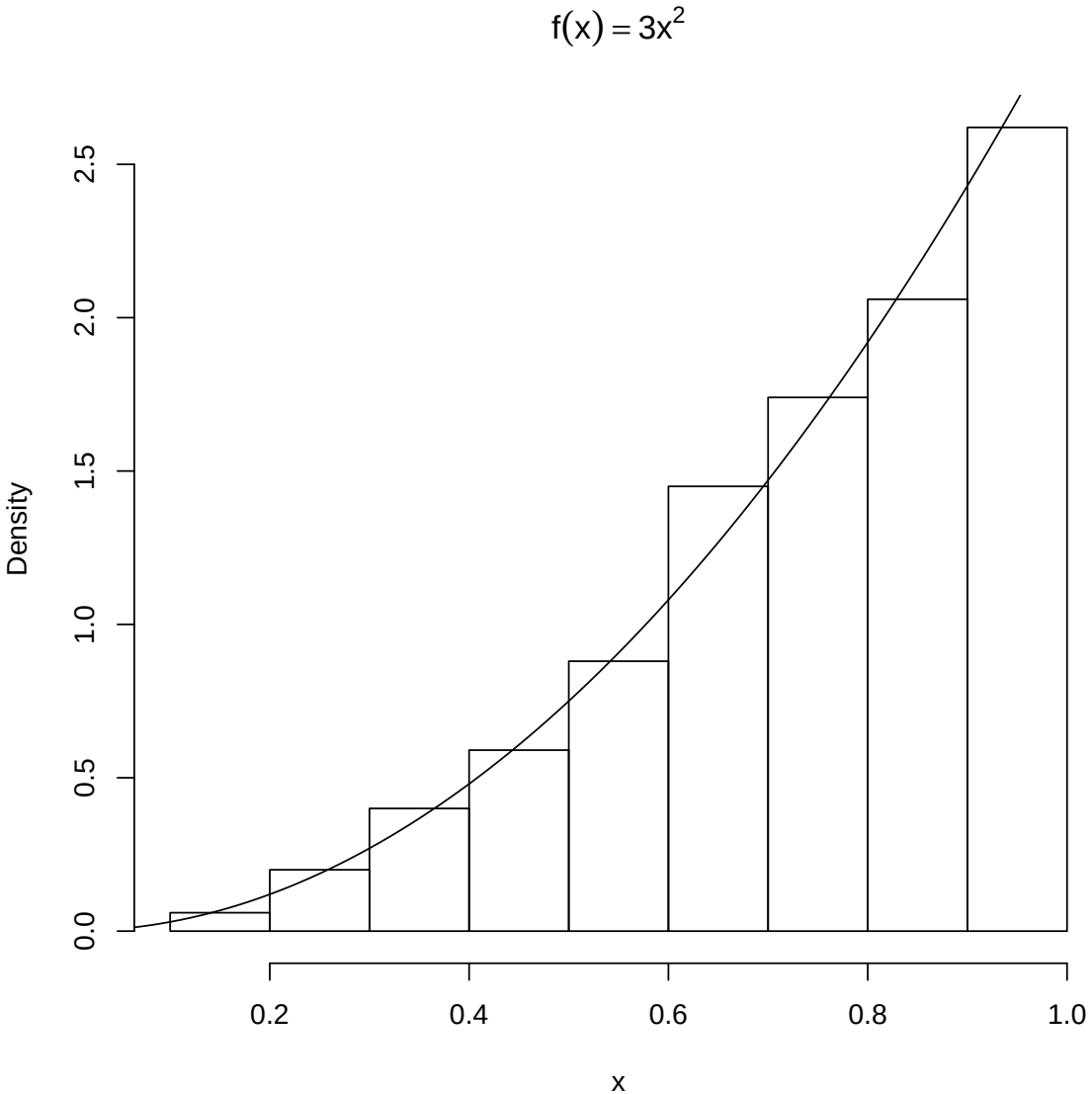
Check Yourself

Task

Simulate a random sample of size 1000 from the pdf $f_X(x) = 3x^2$, $0 \leq x \leq 1$.

- Find F and then F^{-1}.
- Plot the empirical distribution (histogram) with the correct density also shown on the plot.

Inverse Transform Method



Acceptance-Rejection Algorithm

The inverse transform method can be applied when F^{-1} is easy to evaluate. Sometimes this isn't the case (e.g. the normal distribution).

- **Rejection sampling** is a more advanced and very popular method for random number generation.
- How does it work? By sampling candidates from an easier to generate distribution then correcting the sampling probability by randomly rejecting some candidates.

Pros and Cons

- Not restricted to Uniform[0,1] input samples. Idea: generate samples of approximately the correct distribution, then reject some to hit target distribution exactly.
- Can be generalized to work on very general spaces beyond $\mathbb{R}^d$.
- Requires a random and potentially large number of input samples to generate one output, so efficiency can be a concern.

Basic Algorithm

- Input:
  - A probability density g (the **proposal** density)
  - A function p with values in [0,1] (the **acceptance probability**)
- Random Values:
  - X_n i.i.d. with density g (the **proposals**)
  - U_n i.i.d. Uniform[0,1]
- Output: A sequence of i.i.d. random variables with density

$$f(x) = \frac{1}{Z} p(x) g(x) \text{ where } Z = \int p(x) g(x) \, dx.$$

- Algorithm: For $n = 1, 2, 3, ...$
  - Generate $X_n \sim g$ and $U_n \sim$ Uniform[0,1]
  - if $U_n \leq p(X_n)$ then output X_n.

The Rejection Method

Notes

- U_n values randomly decide whether to output or ignore X_n.
- Value X_n output with probability $p(X_n)$.
- If the X_n value is chosen, say it is **accepted**. Otherwise it is **rejected**.
- Idea: we can sample from g easily and then use this to sample from f.

Theory

- The elements of the output are i.i.d. with density f:

$$f(x) = \frac{1}{Z} p(x) g(x) \text{ where } Z = \int p(x) g(x) dx.$$

- Each proposal is accepted with probability Z so the number of proposals required to generate each output is geometrically distributed with mean Z.

Example

- Let the proposal density g be Uniform[-1, +1] and the acceptance probability $p(x) = \sqrt{1 - x^2}$.
- Accepted samples have density

$$f(x) = \frac{1}{Z} p(x) g(x) = \frac{4}{\pi} \sqrt{1 - x^2} \times \frac{1}{2} \mathbb{I}\{-1 \leq x \leq +1\}.$$

- This is the 'semicircle distribution'.

The Rejection Method

```
p <- function(x) {sqrt(1 - x^2)}

generate_one <- function(p_func) {
  val <- NULL
  while (is.null(val)) {
    x <- runif(1, min = -1, max = +1)
    u <- runif(1)
    if (u <= p_func(x)) {val <- x}
  }
  return(val)
}

num <- 1000
samp <- rep(NA, num)
for (i in 1:num) {samp[i] <- generate_one(p)}
hist(samp)
```

The Envelope Rejection Method

- Usually run basic algorithm by choosing the acceptance probabilities p such that the output density

$$f(x) = \frac{1}{Z} p(x) g(x)$$

  coincides with the target density.
- We can write the algorithm more directly with this in mind.

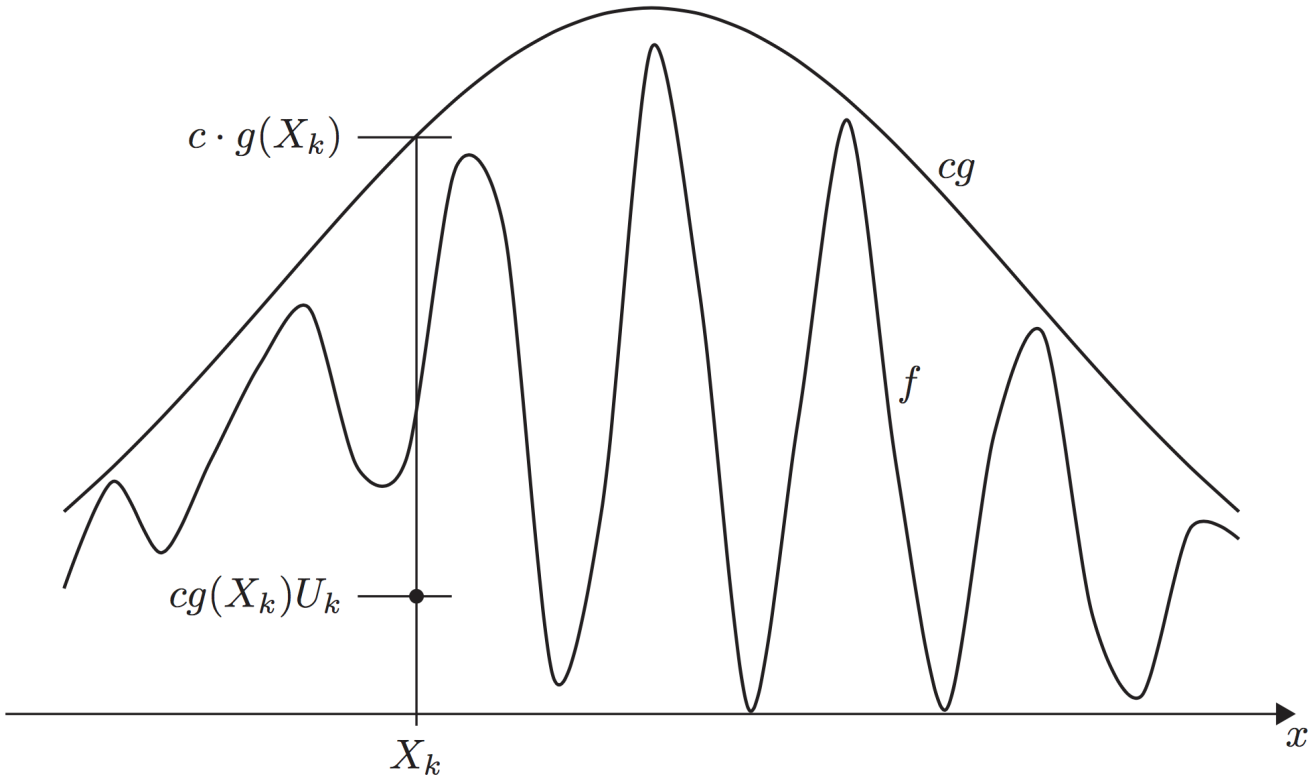
Algorithm

- Input:
  - A probability density f (the **target density**)
  - A probability density g (the **proposal** density)
  - A constant $c > 0$ such that $f(x) \leq cg(x)$ for all x
- Random Values:
  - X_n i.i.d. with density g (the **proposals**)
  - U_n i.i.d. Uniform[0,1]
- Output: A sequence of i.i.d. random variables with density f
- Algorithm: For $n = 1, 2, 3, ...$
  - Generate $X_n \sim g$ and $U_n \sim$ Uniform[0,1]
  - if $cg(X_n)\, U_n \leq f(X_n)$ then output X_n.

Notes

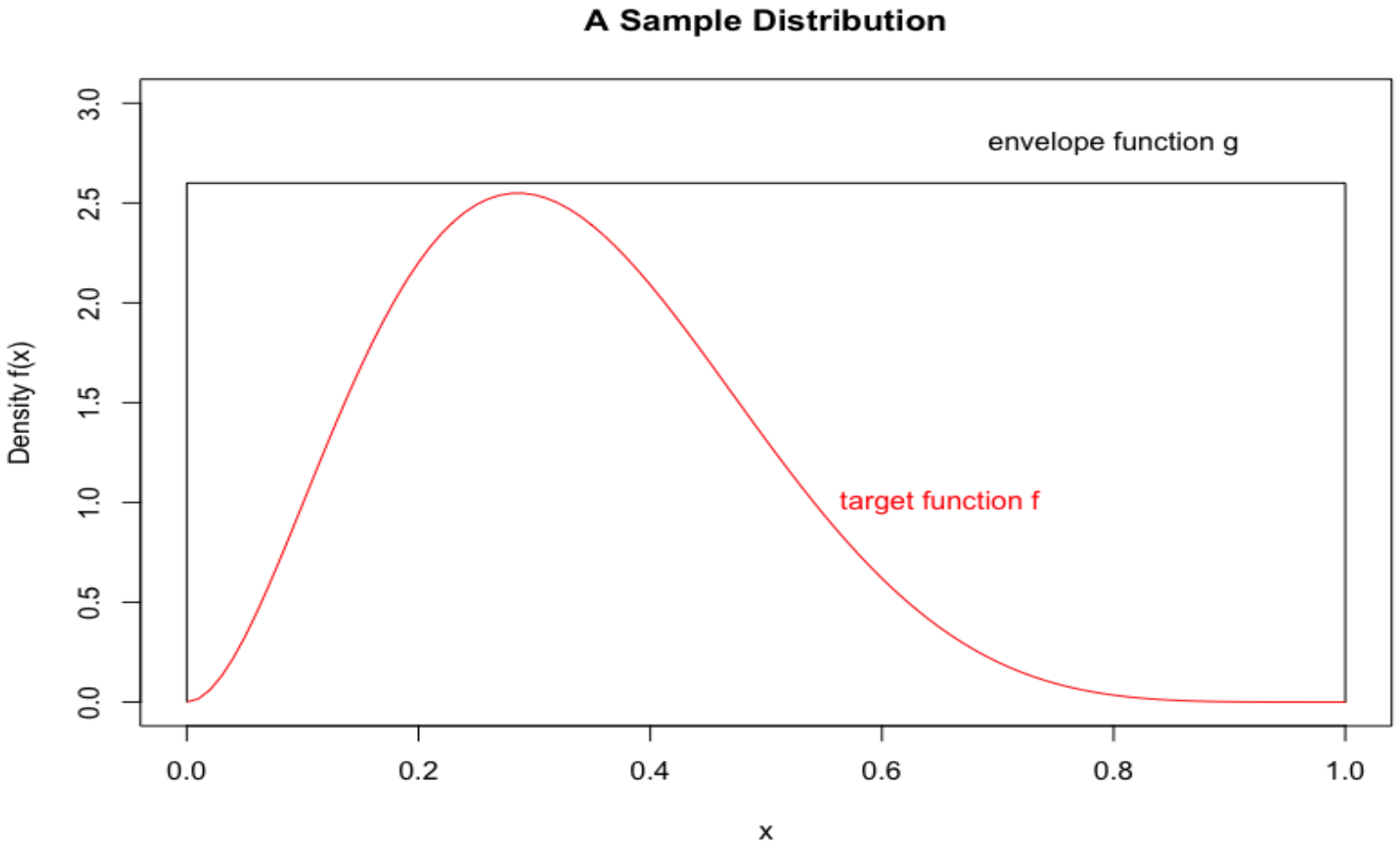
- This is the basic algorithm with acceptance probability $p(x) = \frac{f(x)}{cg(x)}$ if $g(x) > 0$ and $p(x) = 1$ otherwise.
- The elements of the output are i.i.d. with density f.
- Each proposal is accepted with probability $1/c$ so the number of proposals required to generate each output is geometrically distributed with mean $1/c$.
- Function cg sometimes called the 'envelope' for target density f.
- Actually could just use input: a function f (the non-normalized **target density**). Don't need to be able to calculate $Z = \int f(x)dx$ for the algorithm to work!

The Envelope Rejection Method



An Easy Example

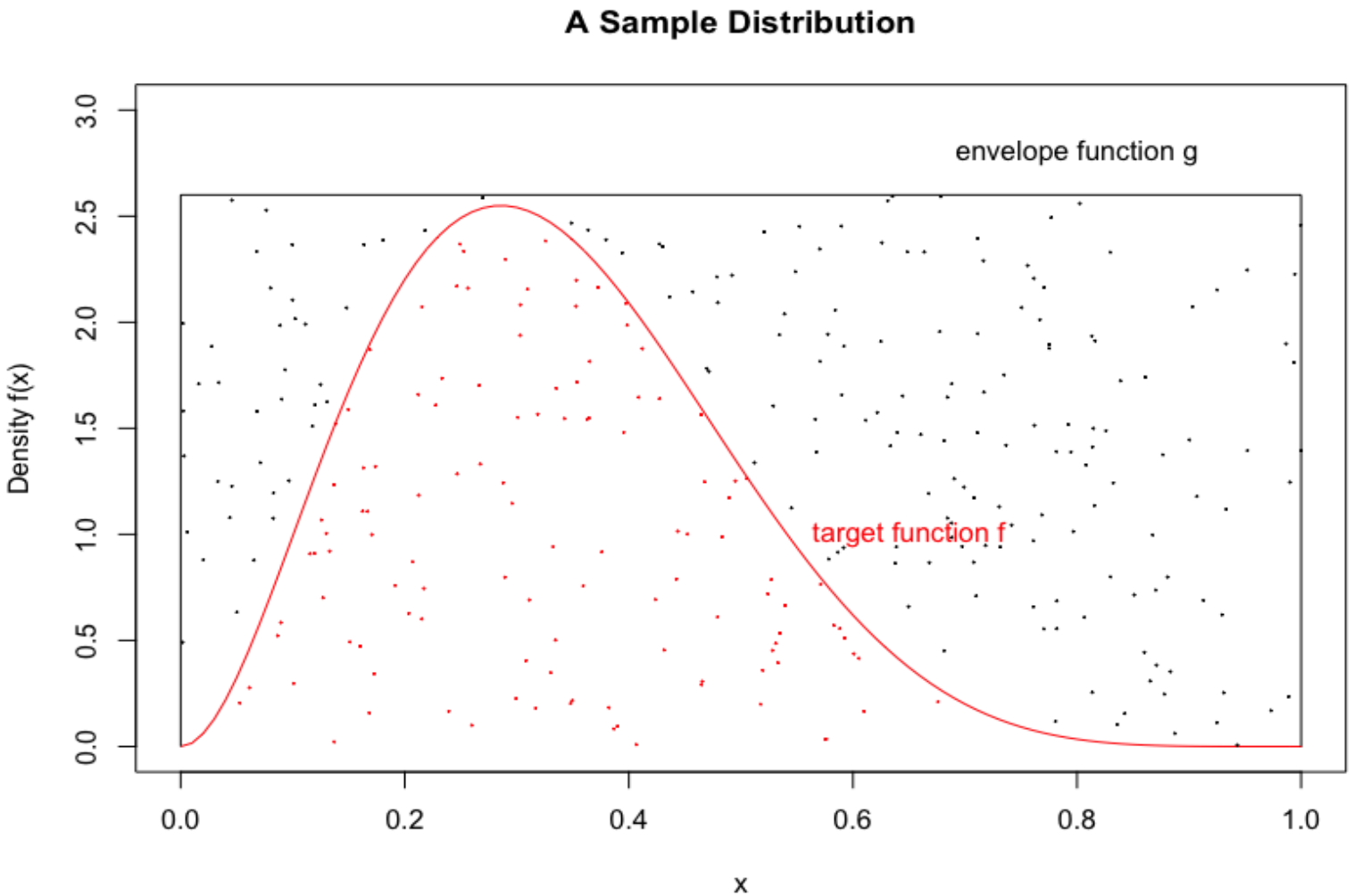
Suppose the pdf f is zero outside an interval $[c, d]$ (in the image [0,1]), and $\leq M$ on the interval (in the image $M \approx 2.5$). Can then choose proposal density g to be uniform.



Geometric Idea

```
plot(c(0,1), c(0,3), ty = "n", main = "A Sample Distribution",
     ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 2.8, "envelope function g")
lines(c(0,0,1,1), c(0,2.6,2.6,0))

x1       <- runif(300, min = 0, max = 1);
y1       <- runif(300, min = 0, max = 2.6)
selected <- y1 < dbeta(x1, 3, 6)

points (x1, y1, col = 1+selected, cex = 0.1)
```

Geometric Idea



Geometric Idea

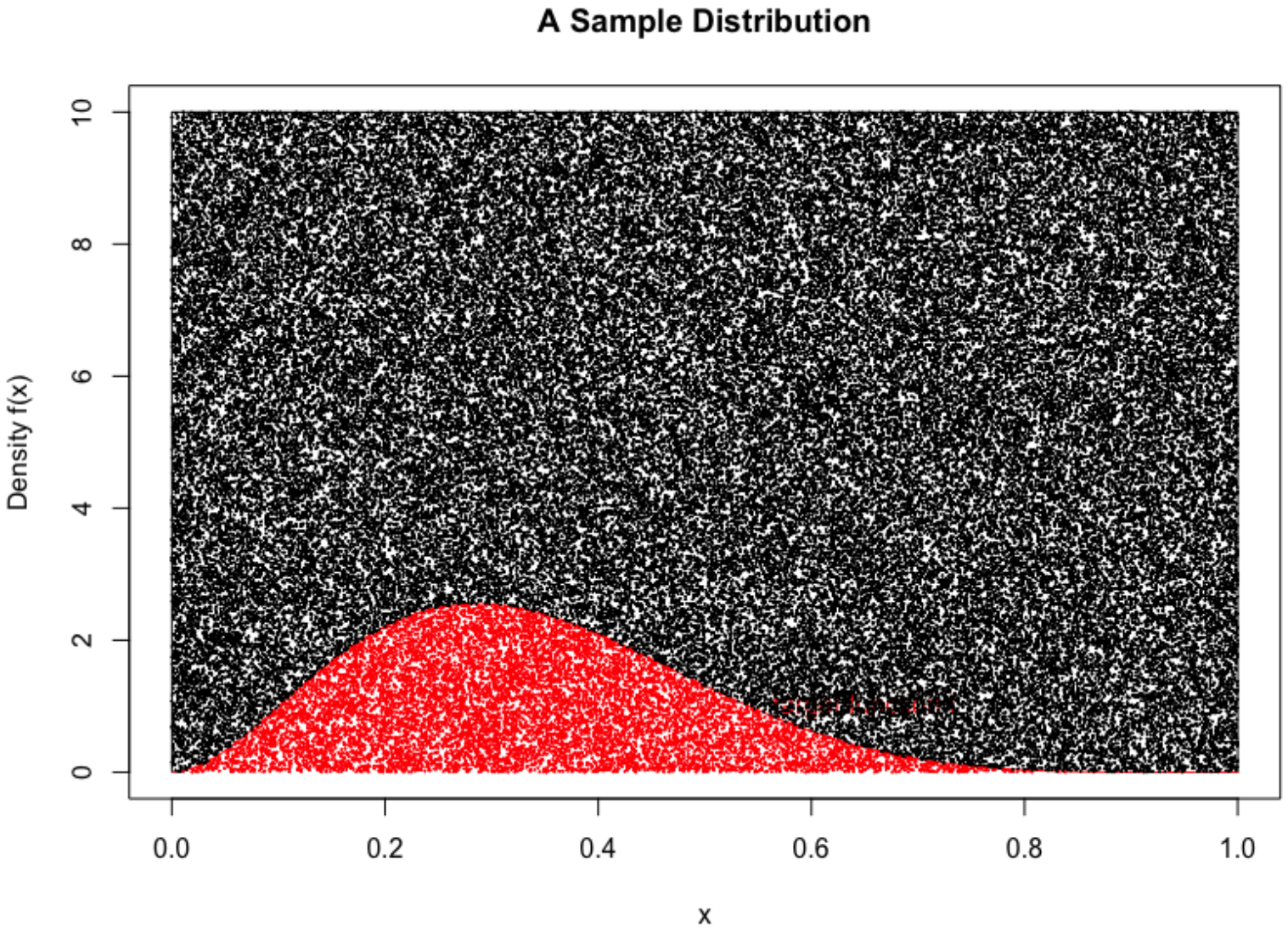
```
mean(selected)  # Proportion selected
accepted.points <- x1[selected]

# Proportion of sample points less than 0.5.
mean(accepted.points < 0.5)

# The true distribution.
pbeta(0.5, 3, 6)
```

Geometric Idea

For this to work efficiently, we have to cover the target distribution with one that sits close to it.

```
plot(c(0,1), c(0,10), ty = "n", main = "A Sample Distribution",
     ylab = "Density f(x)", xlab = "x")
curve (dbeta(x, 3, 6), add = TRUE, col = "red")
text(.65, 1, "target function f", col = "red")
text(.8, 9.7, "envelope function g")
lines(c(0,0,1,1), c(0,10,10,0))

x2       <- runif(100000, 0, 1)
y2       <- runif(100000, 0, 10)
selected <- y2 < dbeta(x2, 3, 6)

mean(selected) # Proportion selected

points (x2, y2, col = 1+selected, cex = 0.1)
```

Geometric Idea



Why does it work?

$$P(X_n \leq x) = P\left(X_n \leq x \middle| U \leq \frac{f(X)}{cg(X)} \right) = \cdots = \int_{-\infty}^{x} f(z)\,dz$$

Exercise: Fill in the missing pieces with Bayes' Rule.

The Envelope Rejection Method

Good methods have the following properties:

1. Constant $c > 0$ chosen such that $cg(x) > f(x)$ for all x.
2. Easy to sample from g.
3. Generate few rejected draws, i.e. $1/c$ is big or c is small.

A simple approach to finding the envelope:

Determine $\max_x\{f(x)\}$, then use a uniform distribution as g, and $c = \max_x\{f(x)\}$.

Example: Beta distribution

Beta(4,3) distribution

Target density: $f(x) = 60x^3(1-x)^2$ for $0 \leq x \leq 1$.

- Can't invert f(x) analytically, so can't use inverse transform method.
- We'll take g to be the uniform distribution on $[0, 1]$. Then, $g(x) = 1$.
- Let $f.max = \max_{x \in [0,1]} f(x)$, then we form envelope with $c = f.max$, so that

$$cg(x) = f.max \geq f(x).$$

- Take the derivative to find the maximum:

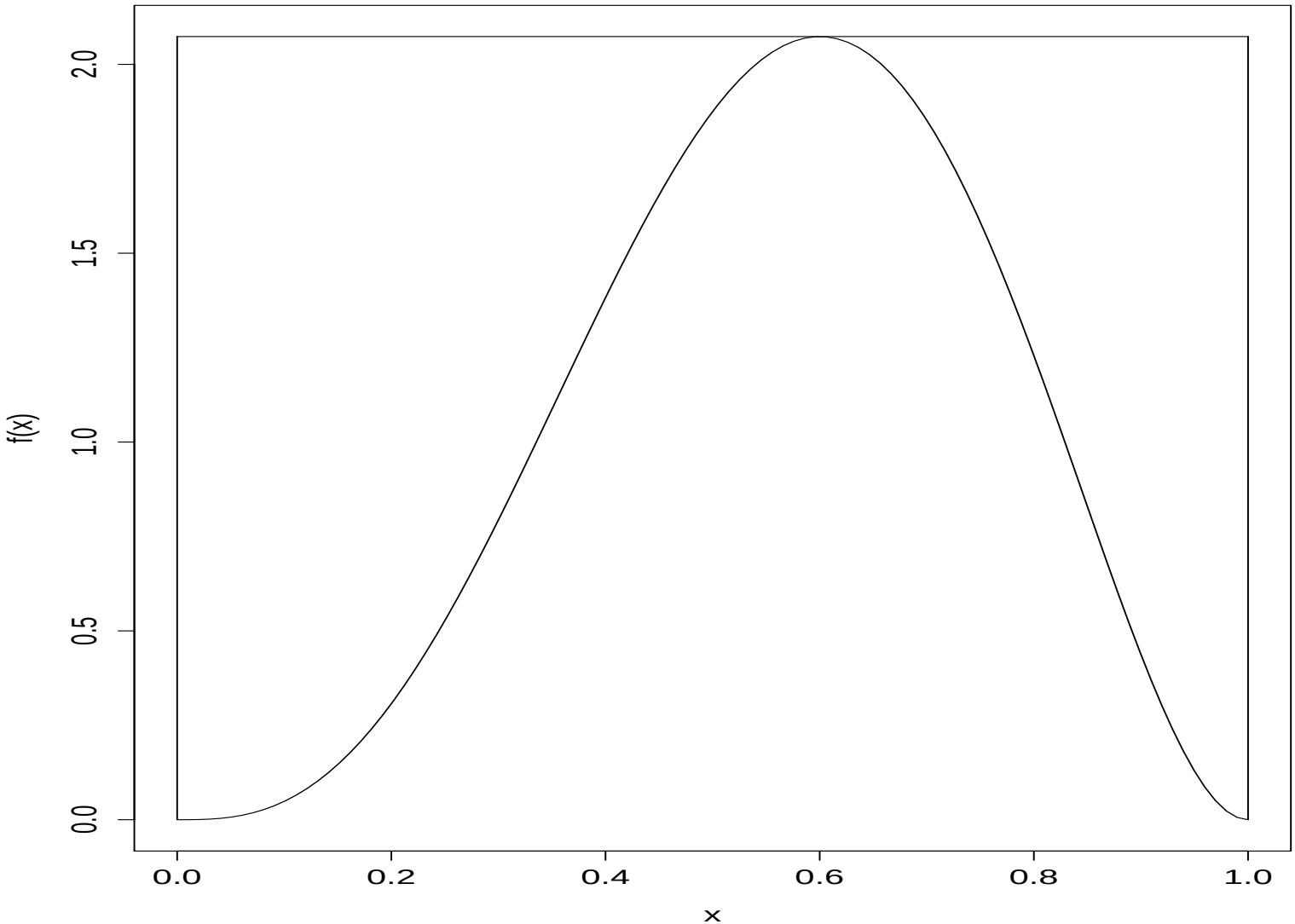
$$f'(x) = 180x^2(1-x)^2 - 120x^3(1-x) = 0 \quad \rightarrow \quad x.max = 0.6.$$

Example: Beta pdf and envelope

Solution Part I

```
f <- function(x) {
  stopifnot(x >= 0 & x <= 1)
  return(60*x^3*(1-x)^2)
}

x <- seq(0, 1, length = 100)
plot(x, f(x), type="l", ylab="f(x)", col = "red")

xmax  <- 0.6
f.max <- 60*xmax^3*(1-xmax)^2

lines(c(0, 0), c(0, f.max), lty = 1)
lines(c(0, 1), c(f.max, f.max), lty = 1)
lines(c(1, 1), c(f.max, 0), lty = 1)
```

Example: Beta pdf and Envelope

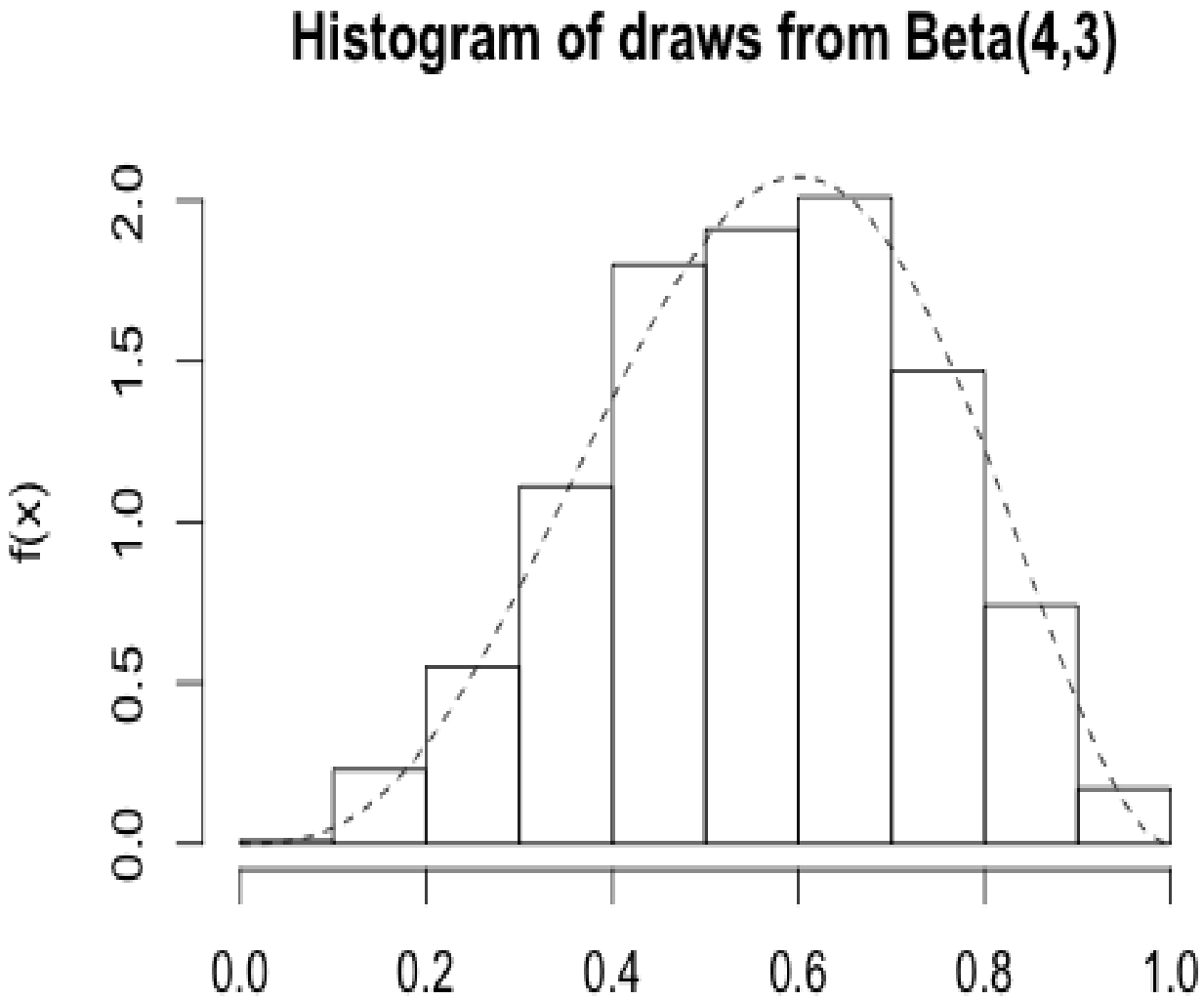


Example: Accept-Reject Algorithm for Beta distribution

Solution Part II

```
n.samps <- 1000         # number of samples desired
n       <- 0            # counter for number samples accepted
samps   <- numeric(n.samps) # initialize the vector of output

while (n < n.samps) {
  x <- runif(1)    #random draw from g
  u <- runif(1)
  if (f.max*u < f(x)) {
    n         <- n + 1
    samps[n] <- x
  }
}

x <- seq(0, 1, length = 100)
hist(samps, prob = T, ylab = "f(x)", xlab = "x",
     main = "Histogram of draws from Beta(4,3)")
lines(x, dbeta(x, 4, 3), lty = 2)
```

Example: Accept-Reject Algorithm for Beta distribution



Monte Carlo Methods

Monte Carlo Methods

- From what we've studied up until now, we have the tools to simulate statistical models.
- Now we'll briefly discuss how these simulations can be used to study properties of the underlying models.
- Approach: generate a large number of samples from a model and learn about the behavior of the model by studying the statistical properties of the samples.

Examples

- Expected value of a random variable can be approximated by the average of a large number of samples (LLN).
- The probability of an event can be approximated by the proportion of occurrences in a large number of samples.
- The quality of an inference method can be assessed by repeatedly generating synthetic data and testing how well the method recovers (known) properties of the synthetic data.

Today

- Focus on computing expectations of the form $\mathbb{E}[f(X)]$ since many interesting questions can be reduced to this kind of problem.
- Approaches:
  - Analytic: sometimes can compute directly by integration – if X has density $\phi(\cdot)$ then

$$\mathbb{E}[f(X)] = \int f(x)\phi(x)\,dx.$$

  - Numerical integration: when the analytic approach fails, we can try to approximate the integral.
  - We study Monte Carlo Integration – use the idea that if $X_1, X_2, \ldots$ are i.i.d. with the same distribution as X then with probability 1,

$$\mathbb{E}[f(X)] = \lim_{N\to\infty} \frac{1}{N}\sum_{j=1}^{N} f(X_j).$$

Monte Carlo Integration

To estimate $\mathbb{E}[f(X)]$ we use the approximation $\mathbb{E}[f(X)] \approx \frac{1}{N}\sum_{j=1}^{N} f(X_j)$ for large N when $X_1, X_2, \ldots$ are i.i.d. with the same distribution as X. This also allows us to estimate:

- Probabilities:
$$\mathbb{P}(X \leq a) = \mathbb{E}[\mathbb{I}\{X \leq a\}].$$
- Definite Integrals:
$$\int_a^b f(x)\,dx = (b-a)\int_a^b f(x)\frac{1}{b-1}\,dx = (b-a)\mathbb{E}[f(X)],$$
where $X \sim$ Uniform[a, b].
- Indefinite Integrals: using a known density $p(\cdot)$,
$$\int g(x)\,dx = \int \frac{g(x)}{p(x)}p(x)\,dx = \mathbb{E}\left[\frac{g(X)}{p(X)}\right],$$
where $X \sim p$.

Example

Suppose $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$. Then computing the expectation $\mathbb{E}[\sin(X)^2]$ or the probability $\mathbb{P}(X \leq 1.36)$ are analytically difficult but easily done via simulation.

```
n     <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est <- mean(sin(norms)^2)
est
est2 <- mean(norms <= 1.36)
est2
pnorm(1.36, sd = 1/sqrt(2))
```

Example

Estimate the integral

$$\int_{-\infty}^{\infty} \sin(x)^2 e^{-x^2}\, dx,$$

using MC techniques.

Solution

We'll use standard normal random variables and a standard normal density denoted $p(x)$:

$$\int \sin(x)^2 e^{-x^2}\, dx = \int \frac{\sin(x)^2 e^{-x^2}}{\frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}x^2}}\, p(x)\, dx = \sqrt{2\pi}\, \mathbb{E}\left[\sin(X)^2 e^{-\frac{1}{2}X^2}\right]$$

```
n     <- 10000000
norms <- rnorm(n)
est   <- sqrt(2*pi) * mean(sin(norms)^2 * exp(-(1/2)*norms^2))
est
```

There's an easier way...

Looking again:

$$\int \sin(x)^2 e^{-x^2}\, dx = \sqrt{\pi} \int \sin(x)^2 \left(\frac{1}{\sqrt{\pi}} e^{-x^2} \right) dx = \sqrt{\pi}\, \mathbb{E}[\sin(X)^2],$$

where $X \sim \mathcal{N}(0, \frac{1}{\sqrt{2}})$. From work earlier:

```
n     <- 10000000
norms <- rnorm(n, sd = 1/sqrt(2))
est   <- sqrt(pi)*mean(sin(norms)^2)
est
```

Monte Carlo Integration

By the Central Limit Theorem,

$$\frac{1}{N}\sum_{i=1}^{N} f(X_i) \xrightarrow{d} \mathcal{N}\left(\int f(x)dx, \frac{1}{\sqrt{N}} sd(f(X))\right).$$

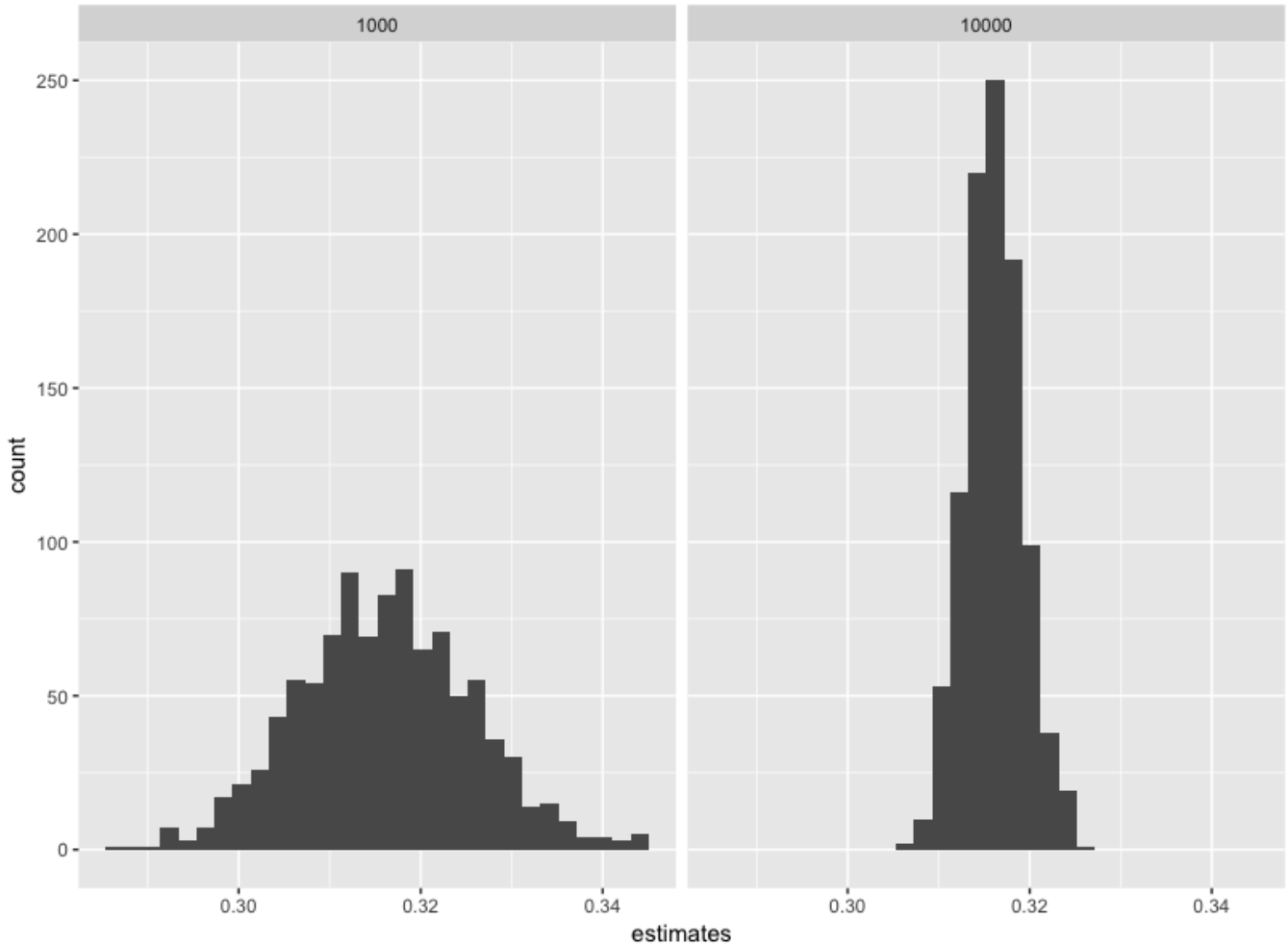
- The Monte Carlo approximation is unbiased.
- Note that by increasing N the error can get as small as you'd like, even if f or X are very complicated.
- On the other hand, larger N means more computational time.

Example

Computing the expectation $\mathbb{E}[\sin(X)^2]$ for $X \sim \mathcal{N}\left(0, \frac{1}{\sqrt{2}}\right)$.

```
n1 <- 10000; n2 <- 1000
estvec1 <- rep(NA, 1000); estvec2 <- rep(NA, 1000)

for (i in 1:1000) {
  norms1     <- rnorm(n1, sd = 1/sqrt(2))
  estvec1[i] <- mean(sin(norms1)^2)
}
for (i in 1:1000) {
  norms2     <- rnorm(n2, sd = 1/sqrt(2))
  estvec2[i] <- mean(sin(norms2)^2)
}

df <- data.frame(estimates = c(estvec1, estvec2),
                 n = c(rep(n1, 1000), rep(n2, 1000)))
ggplot(df) +
  geom_histogram(aes(x = estimates)) + facet_wrap(~ n, ncol = 2)
```

Example



Check Yourself

Tasks

- Estimate $P(X < 3)$ where X is an exponentially distributed random variable with $rate = 1/3$.
- Use built-in `R` functions to find the exact probability.

More Simulations

Recall,

We can simulate random numbers in R from various distributions:

- `rnorm()`: generate normal random variables
- `pnorm()`: normal distribution function, $\Phi(x) = P(Z < x)$
- `dnorm()`: normal density function, $\phi(x) = \Phi'(x)$
- `qnorm()`: normal quantile function, $\Phi^{-1}(u)$

Replace `norm` with other distribution names, all the same functions apply.

Why do we Simulate?

Recall,

`R` gives us great simulation tools (unlike many other languages). Why should we simulate, though?

- Often simulations are easier than hand calculations.
- Often simulations can be made more realistic than hand calculations.

Today We'll do a FUN Simulation!

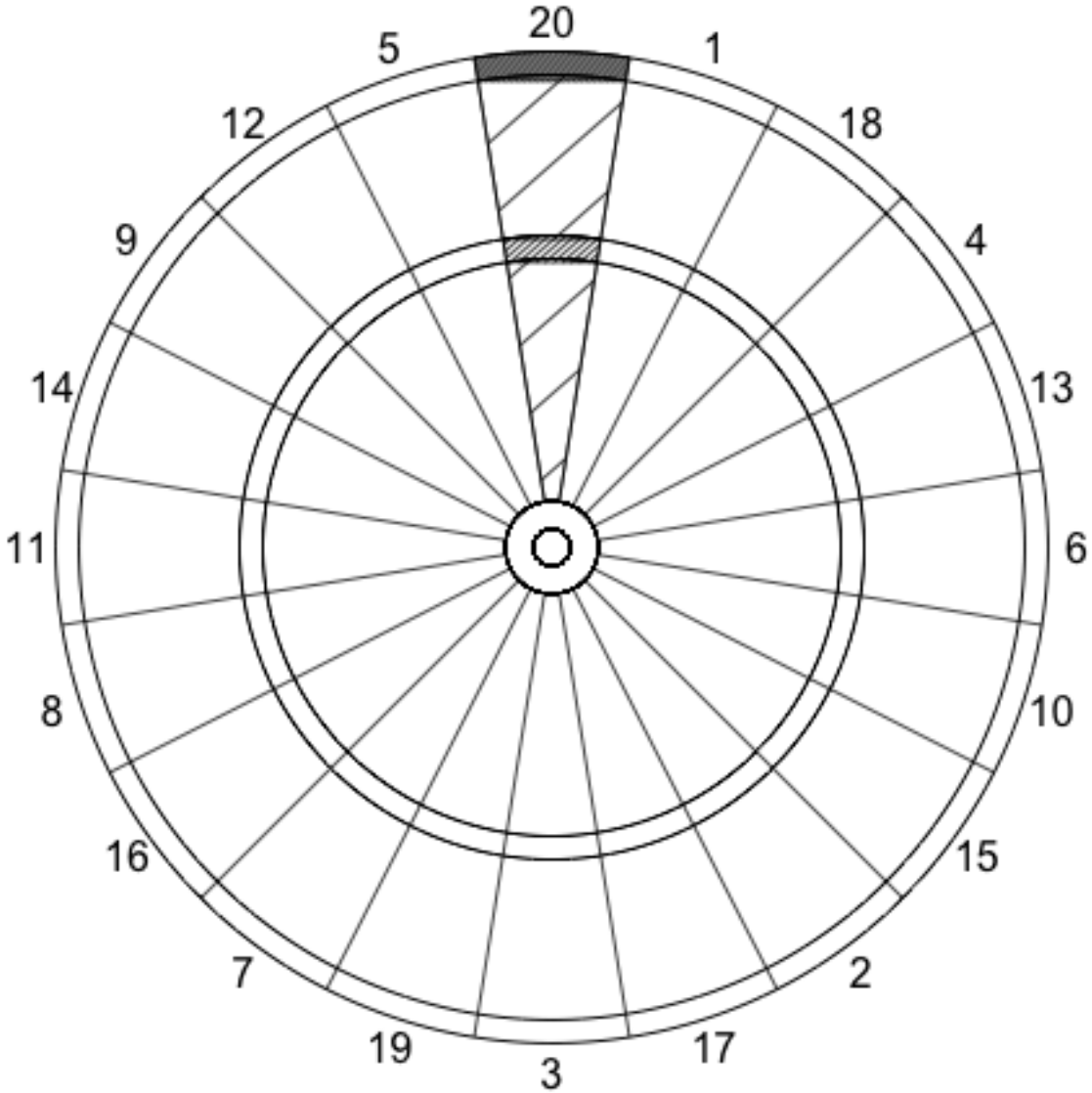
Darts[1]

Darts is a game where you throw metal darts at a dartboard and receive different scores for landing the dart in different regions.

- Land in a slice, obtain the score of the corresponding number.
- Land in the "double ring", get double the score.
- Land in the "triple ring", get triple the score.
- Land in the "double bullseye", get 50 points.
- Land in the "single bullseye", get 25 points.

[1] Darts simulation notes adapted from 'A Statistician Plays Darts'

Today We'll do a FUN Simulation!



Some Questions to Answer with Simulation

- What strategy provides the better score? Throwing such that darts land uniformly at random on the board, or aiming at the center, with your throws being normally distributed with some variance.
- If you're aiming at a spot and your throws are normally distributed with some variance, what spot should you aim for?
- Can I estimate the variance of my throws, if I aim at the center and receive a certain score?

Board Measurements

We'll make a list of standard dart board measurements and the numbers of the board. All measurements are in mm.

```
board = list(
  R1 = 6.35,  # center to double bullseye ring
  R2 = 15.9,  # center to single bullseye ring
  R3 = 99,    # center to inner triple ring
  R4 = 107,   # center to outer triple ring
  R5 = 162,   # center to inner double ring
  R = 170,    # center to outer double ring
  nums = c(20,1,18,4,13,6,10,15,2,17,3,19,
           7,16,8,11,14,9,12,5)) # numbers in order
```

Plotting the Board

We'll write a `drawBoard` function with the following features:

- Inputs: `board` a list containing dart board measurements.
- Outputs: None, but it will plot the board. Then we can plot on top of the board with functions like `points()` or `lines()`.

This function on Courseworks and it essentially does the following:

- Use $(0, 0)$ as the center of the board.
- Plot concentric circles centered at $(0, 0)$ to show the rings.
- Plot the 'spokes'.
- Add the numbers around the outside.

Let's Simulate Dart Throws

```
drawBoard = function(board) {
  R1 = board$R1; R2 = board$R2; R3 = board$R3; R4 = board$R4;
  R5 = board$R5; R = board$R; nums = board$nums

  mar.orig = par()$mar
  par(mar = c(0, 0, 0, 0))
  plot(c(), c(), axes = FALSE, xlim = c(-R - 15, R + 15),
       ylim = c(-R - 15, R + 15))
  t = seq(0, 2 * pi, length = 5000)
  x = cos(t); y = sin(t)
  points(R * x, R * y, type = "l")
  points(R5 * x, R5 * y, type = "l")
  points(R4 * x, R4 * y, type = "l")
  points(R3 * x, R3 * y, type = "l")
  points(R2 * x, R2 * y, type = "l")
  points(R1 * x, R1 * y, type = "l")
  t0 = pi/2 + 2 * pi/40
  points(c(R2 * cos(t0), R * cos(t0)),
         c(R2 * sin(t0), R * sin(t0)), type = "l")
  for (i in 1:19) {
```

Simulating the Scoring

We write a `scorePositions()` function with the following features.

- Inputs:
  - `x`: vector, horizontal positions of dart throws in mm, with the center of the board being 0
  - `y`: vector, vertical positions of dart throws in mm, with the center of the board being 0
  - `board`: list, containing dart board measurements
- Outputs: vector of scores, same length as x (and as y)

This function is on Courseworks and it essentially does the following:

- Calculate the radius at which the throw lands.
- Calculate the angle at which the throw lands.
- Use these to calculate the score.

Let's Simulate Dart Throws

```
scorePositions = function(x, y, board) {
  R1 = board$R1; R2 = board$R2; R3 = board$R3; R4 = board$R4
  R5 = board$R5; R = board$R; nums = board$nums

  n = length(x)
  rad = sqrt(x^2 + y^2)
  raw.angles = atan2(x,y)
  slice = 2*pi/20
  tilted.angles = (raw.angles + slice/2) %% (2*pi)
  scores = nums[floor(tilted.angles/slice) + 1]

  # Bullseyes
  scores[rad <= R1] = 50
  scores[R1 < rad & rad <= R2] = 25

  # Triples
  scores[R3 < rad & rad <= R4] = 3*scores[R3 < rad & rad <= R4]

  # Doubles
  scores[R5 < rad & rad <= R] = 2*scores[R5 < rad & rad <= R]
```

Let's Simulate Dart Throws

Our Task

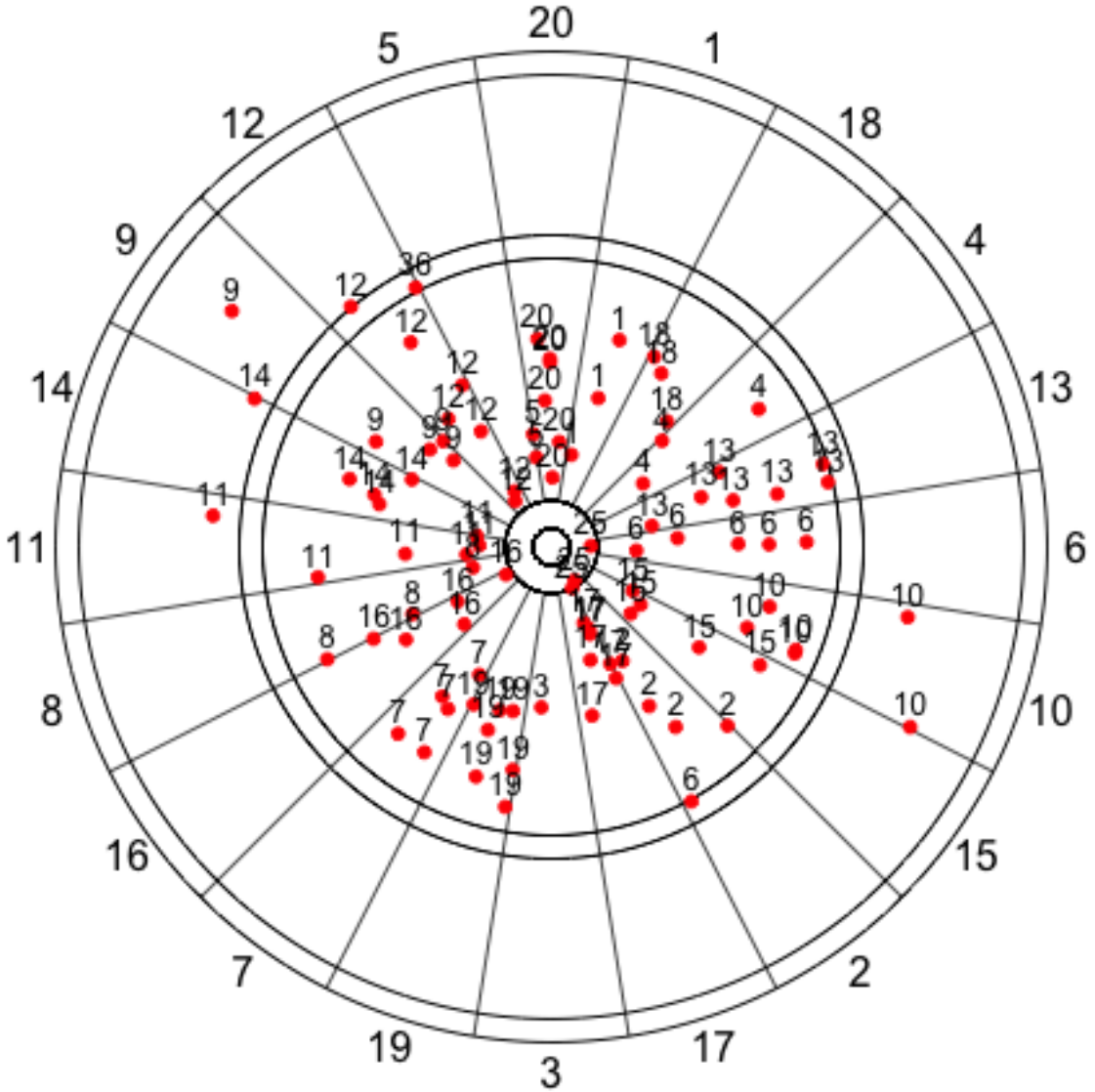
Let `x`, `y` denote the horizontal and vertical positions of the throws, measured from the center of the board which is $(0, 0)$. Let's consider the following model, `x` and `y` are both $\mathcal{N}(0, 50^2)$ (the standard deviation here is 50mm). We will:

- Generate 100 throws from this model.
- Plot the throw positions on the dart board, with `pch` = 20 (to make small solid dots).
- Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

Our Solution

```
# Simulate 100 throws with the x and y
# location modeled by N(0, 50^2).

throws  <- 100
std.dev <- 50

x <- rnorm(throws, sd = std.dev)
y <- rnorm(throws, sd = std.dev)

drawBoard(board)
points(x, y, pch = 20, col = "red")

# Score the throws and add the values to the plot

scores <- scorePositions(x, y, board)
text(x, y + 8, scores, cex = .75)
```

Let's Simulate Dart Throws



Let's Simulate Dart Throws

Our Task

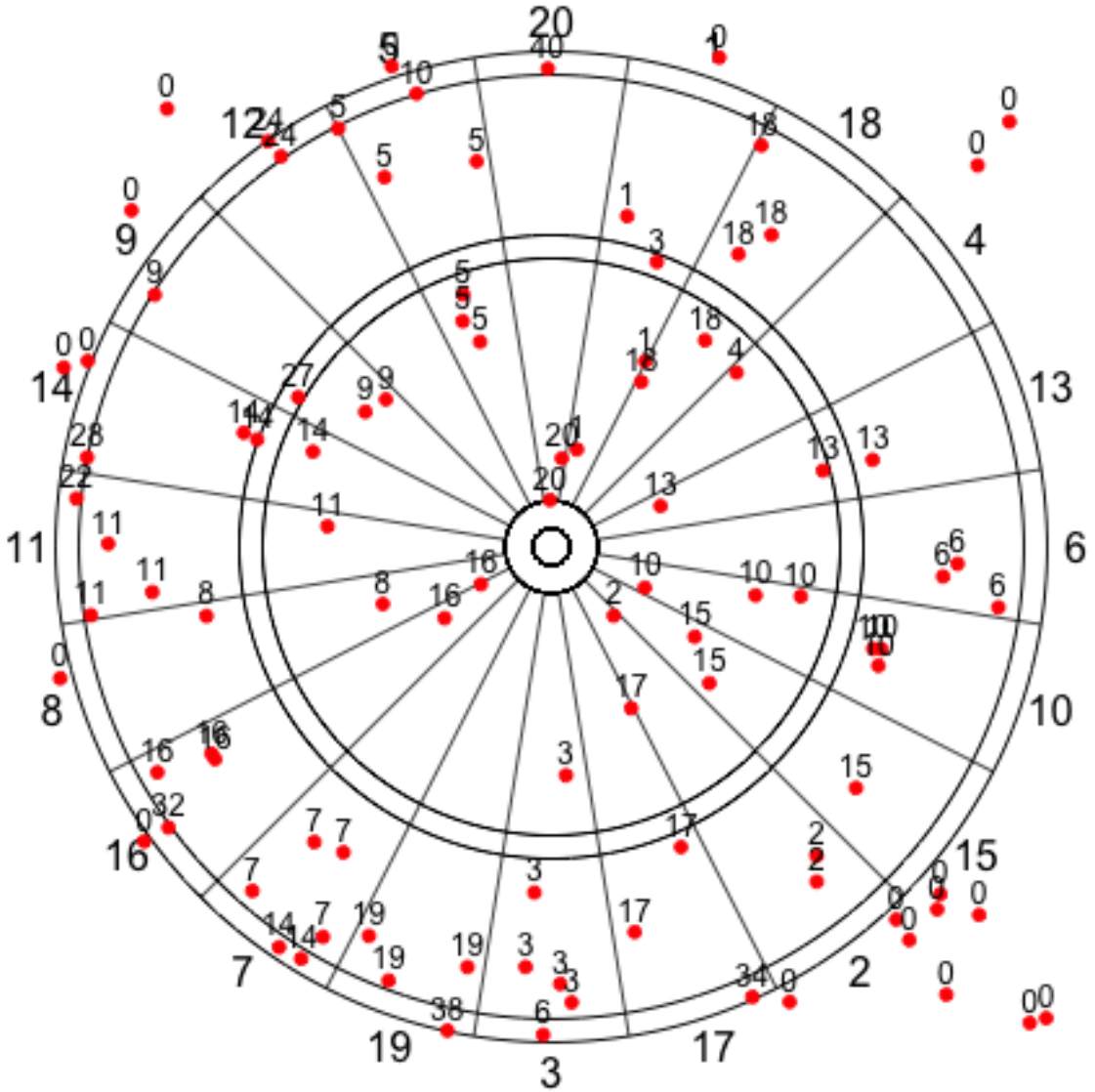
Let `x`,`y` denote the horizontal and vertical positions of the throws, measured from the center of the board which is $(0, 0)$. Consider the following model, `x` and `y` are both Uniform(-R,R) where $R = 170$ is the radius of the board. Note this is the smallest rectangle that contains the dart board. Do the following:

- Generate 100 throws from this model.
- Plot the throw positions on the dart board, with `pch` = 20 (to make small solid dots).
- Compute the scores of the 100 throws and add the scores corresponding to each throw on the plot.

Solution

```
# Simulate 100 throws with the x and y
# location modeled by Uniform(-R, R).

throws  <- 100
R       <- 170

x <- runif(throws, min = -R, max = R)
y <- runif(throws, min = -R, max = R)

drawBoard(board)
points(x, y, pch = 20, col = "red")

# We score our throws and add the values to the plot

scores <- scorePositions(x, y, board)
text(x, y + 8, scores, cex = .75)
```

Let's Simulate Dart Throws



Which Strategy is Better?

Our Task

Now let's simulate $10,000$ dart throws according to the two possible models:

1. `x` and `y` are both $\mathcal{N}(0, 50^2)$.
2. `x` and `y` are both Uniform(-R, R).

What are the average scores under each model, and which is higher?

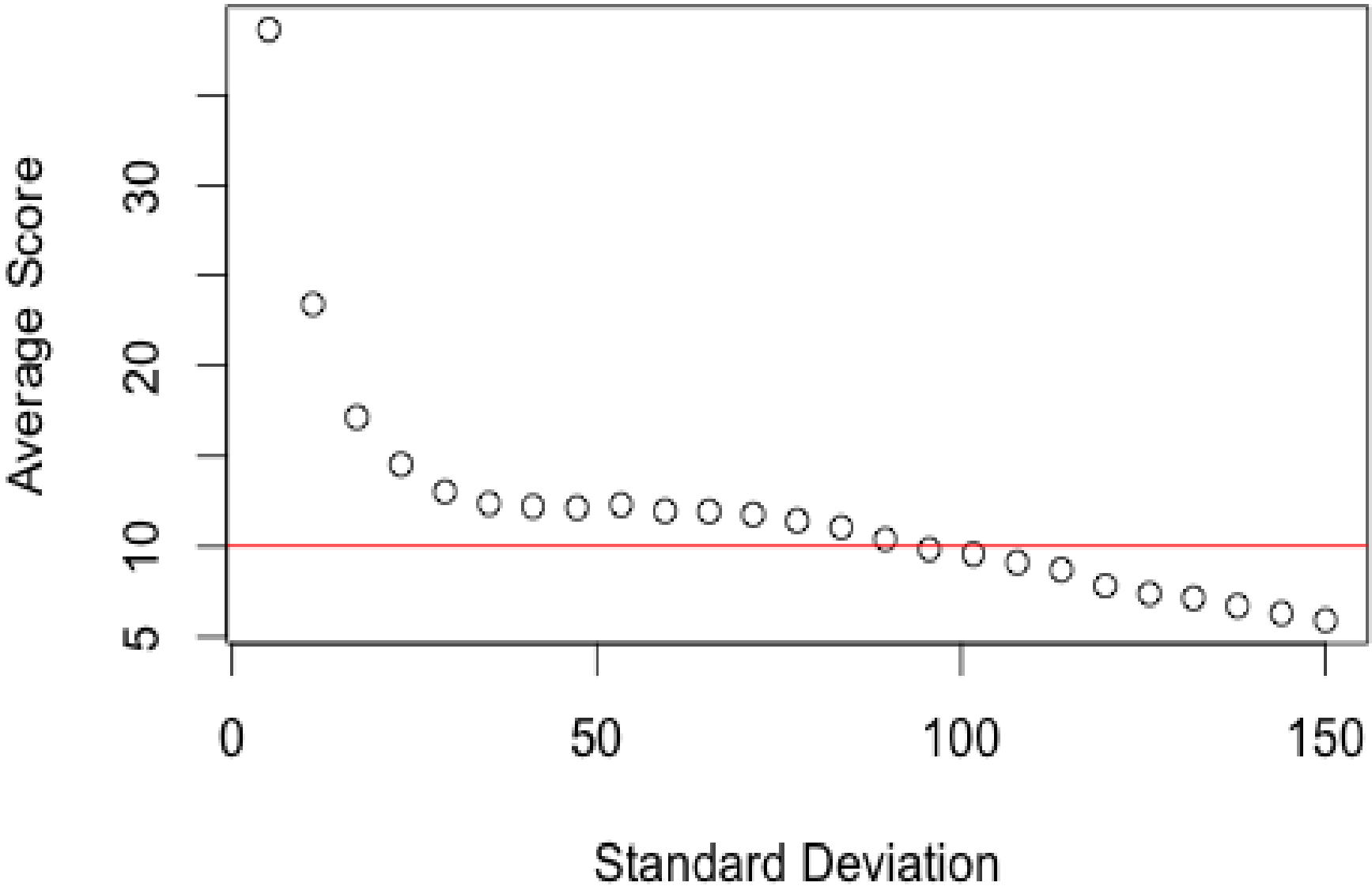
```
throws <- 10000
x1 <- rnorm(throws, sd = std.dev)
y1 <- rnorm(throws, sd = std.dev)
x2 <- runif(throws, min = -R, max = R)
y2 <- runif(throws, min = -R, max = R)

scores1 <- scorePositions(x1, y1, board)
scores2 <- scorePositions(x2, y2, board)
mean(scores1)
mean(scores2)
```

Check Yourself

Tasks

- For 25 values of standard deviation (sd) between 5 mm and 150 mm, draw $10,000$ throws from model (1). For each value of sd, compute the average score.
- Make a plot showing the average score as a function of sd. Label the axes appropriately, and draw the average score calculated for the uniform model (2) as a horizontal line, in red.
- At what value of sd does it become better to throw uniformly at the board?

Let's Simulate Dart Throws



Check Yourself

Consider fixed the value sd = 35 mm, and consider a normal model for throws where $\mathbf{x}$ is $\mathcal{N}(\text{mean.x}, 35^2)$ and $\mathbf{y}$ is $\mathcal{N}(\text{mean.y}, 35^2)$, with

1. (mean.x, mean.y) = (0, 0)
2. (mean.x, mean.y) = (0, 103)
3. (mean.x, mean.y) = (-32, -98)

Tasks

- Either by plotting or by looking at scores, determine where the throws are being aimed in each of the three models. That is, the choice of (mean.x, mean.y) = (0, 0) in model (1) means that we are aiming at the center of the board. Where are we aiming in models (2) and (3)?
- For each of the models, draw $10,000$ throws, and compute the average score. How do they compare? Can you explain the results from an intuitive perspective?

Exam Notes

- Exam is next week – on your computer. (i.e. bring a charged computer with `R` and `R`Markdown).
- You will edit a Markdown file I provide and turn it in via Courseworks before time is up.
- You can not communicate with any living, breathing person while you are taking the exam.
- You can use any class materials (have them downloaded on your machine), but not the internet.
- We will have a seating chart (some of you will be in Havemeyer 309).
- The exam will be 1.5 hours. You will not have time to look up everything in your notes.
- When the time is up, you must close your computer or we will take of points. If you haven't submitted to Courseworks yet, you will have to stay afterwards to do this one at a time with the TA.