STAT5003

Week 10 : Monte Carlo

Jaslene Lin
The University of Sydney

Readings

Important

- Easily accessible text in Explorations in Monte Carlo Methods

Monte Carlo Methods

What are Monte Carlo methods?

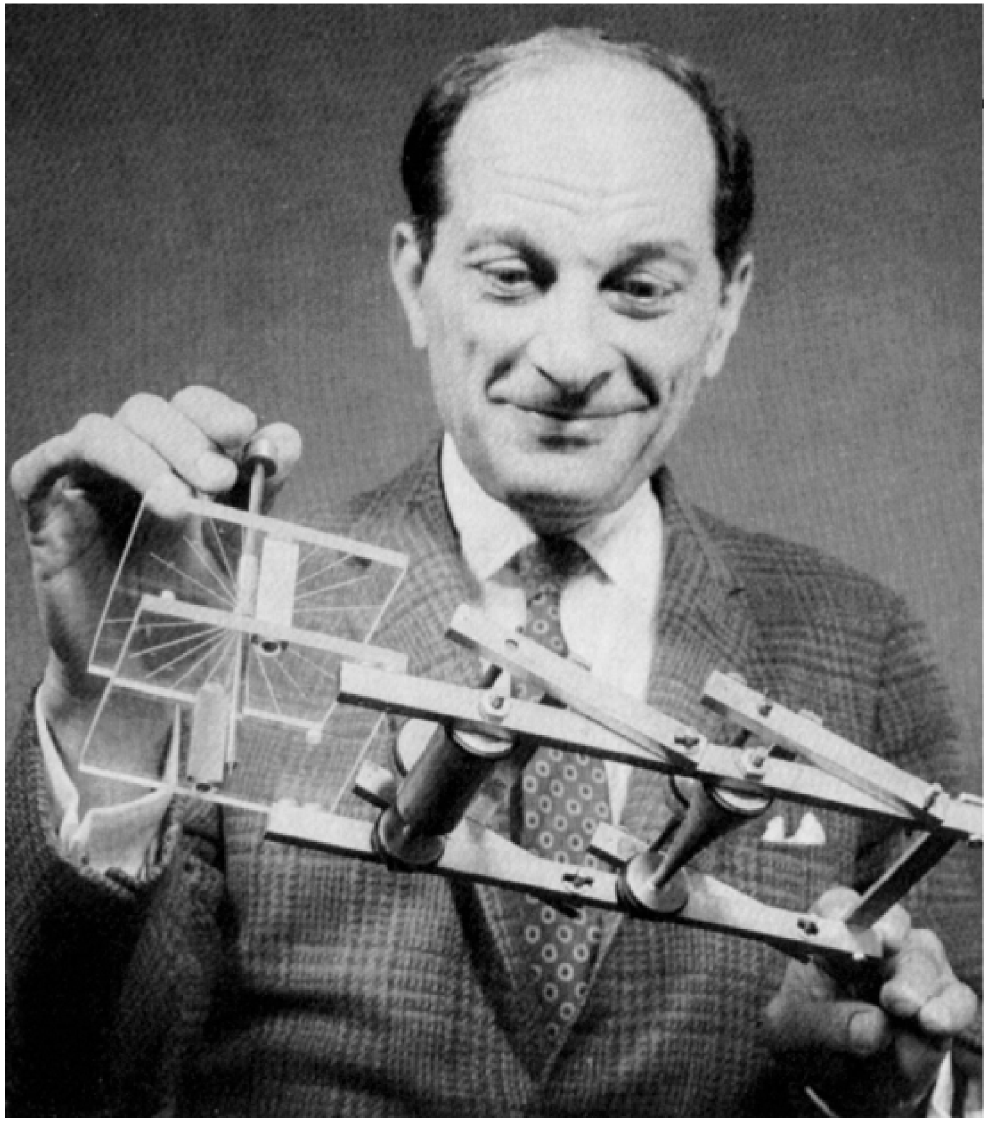
- Monte Carlo are a class of computational methods that can be applied to a wide range of problems
- Key aspect of Monte Carlo methods are that they rely on **random sampling**
- Generally provide approximate solutions
    - Not exact solutions
- Used in cases where analytic solutions don't exist or are too difficult to implement
- Monte Carlo methods are sometimes referred to as stochastic simulation

History of the Monte Carlo method

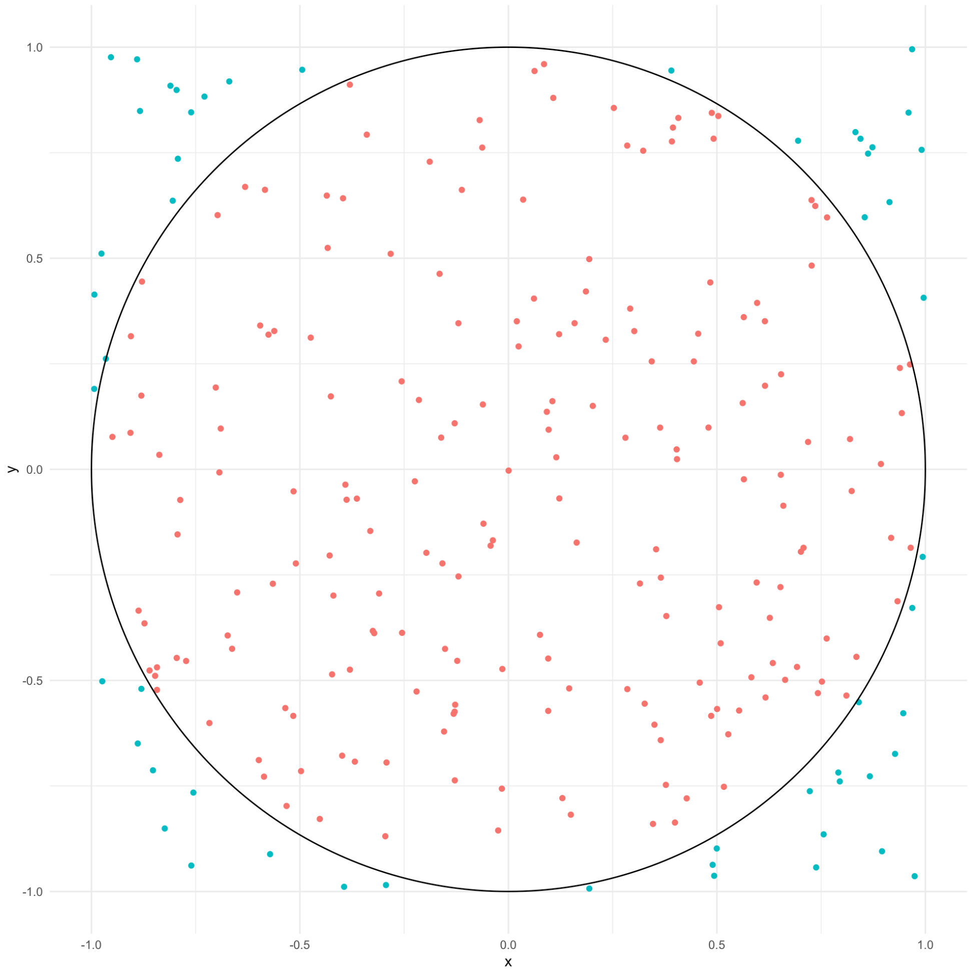
- Invented by a Polish mathematician called Stanislaw Ulam in the late 1940s
- Inspiration during recovery from illness with a thought experiment:

> The question was what are the chances that a Canfield Solitaire laid out with 52 cards will come out successfully? After spending a lot of time trying to estimate them by pure combinatorial calculations, I wondered whether a more practical method than "abstract thinking" might not be to lay it out say one hundred times and simply observe and count the number of successful plays.

- Monte Carlo was the code name given for the project
  - ➟ Inspired by the Monte Carlo casino in Monaco where Ulam's uncle would borrow family money to gamble


By Originally uploaded by Deer*lake (Transferred by Deer*lake) - Originally uploaded on en.wikipedia, Public Domain, https://commons.wikimedia.org/w/index.php?curid=18403886

Monte Carlo estimation of π



- Area of circle: πr^2; area of square: $4r^2$
  - ➡ r is the radius
- A Monte Carlo method to estimate π
  1. Randomly draw N points within unit square at random
  2. Count the R points which are inside the unit circle
  3. Compute the ratio R/N and estimate π as $4R/N$

Using Monte Carlo to estimate π

In this example, the mathematical statistics of the procedure is:

- Write the parameter we want to estimate, i.e., π, as an expectation
- Represent the parameter as a sample approximation
- Let $f(\cdot)$ be the probability density function (pdf) of X and A be the support of X

$$\mathbb{E}[X] = \int_A t f(t)\, dt \approx \frac{1}{N} \sum_{i=1}^{N} X_i$$

- The law of large numbers guarantees that as the number of samples we draw increase, the estimator converges to the true parameter value almost surely

Expectation of a random variable

From last slide, we can write the expected value of a random variable X as a sum:

$$\mathbb{E}[X] = \int_A t \cdot f(t)\, dt \approx \frac{1}{N} \sum_{i=1}^{N} X_i$$

We can replace X with a function $g(X)$. Then the previous equation becomes:

$$\mathbb{E}[g(X)] = \int_A g(t) \cdot f(t)\, dt \approx \frac{1}{N} \sum_{i=1}^{N} g(X_i)$$

Here, we assume to draw $X_i \sim f$

Monte Carlo Integration example

Goal is to evaluate the following definite integral

$$\int_0^1 e^{-x^2/2}\,dx \approx \frac{1}{N}\sum_{i=1}^{N} g(X_i).$$

Find a random variable such that the above holds. Need to be careful such that:

- Domain of the random variable and g agree with the above definition

- One example is:
  - $g(x) = e^{-x^2/2}$
  - X_i is a uniform random variable on $(0,1)$
  - Then $\mathbb{E}[g(X)] = \int_0^1 e^{-x^2/2}\,dx \approx \frac{1}{N}\sum_{i=1}^{n} g(X_i)$

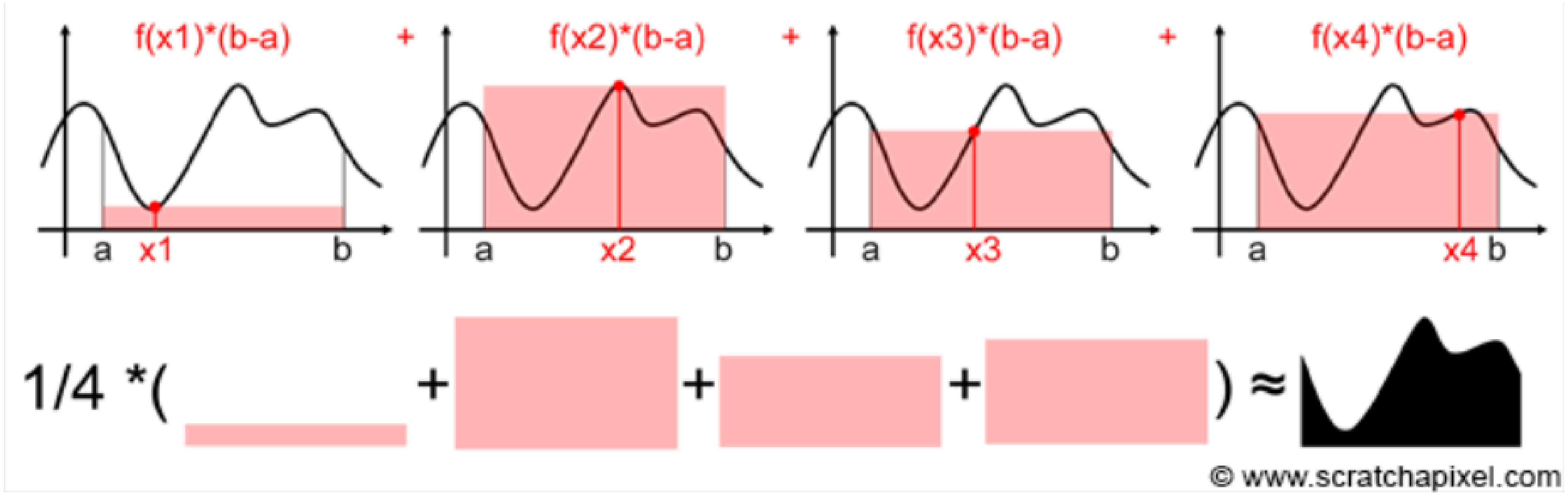
```r
## define the function
f <- function(x){
  exp(-x^2/2)
}

## exact solution
integrate(f, lower = 0, upper = 1)
```

```
0.8556244 with absolute error < 9.5e-15
```

```r
## Monte Carlo estimation
set.seed(5003)
x <- runif(1000, 0, 1)
mean(f(x))
```

```
[1] 0.8545841
```

Monte Carlo visually



Simulating random variable

- Let's say we know how to simulate random variables that has a **uniform** distribution on the interval $[0, 1]$
  - Call this uniformly distributed random variable U
  - Can do that in `R` with `runif`

> How do we simulate random variables that can have any probability distribution?

In `R`, we can draw random Gaussian variables using the function `rnorm` but how do these functions really work?

Two methods (others exist):

1. Inverse-transform method
2. Acceptance-rejection method

Inverse transform method

- Let's say X is a random variable that has a cumulative distribution function (cdf) F
- Remember that cdf is the integral of the pdf

$$F(x) = \int_{-\infty}^{x} f(t)\,dt$$

- Recall U is a uniformly distributed random variable over $[0, 1]$
- If the inverse of $F(x)$ exists, then we can generate X as:

$$X = F^{-1}(U)$$

Example: Exponential distribution

The exponential distribution has a pdf of the form:

$$f(x) = \begin{cases} \lambda e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

and a cdf of the form:

$$F(x) = \begin{cases} 1 - e^{-\lambda x} & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

The inverse cdf is:

$$F^{-1}(p) = -\frac{\log(1-p)}{\lambda}, \quad 0 < p < 1$$

To sample from the exponential distribution, we can do the following:

1. Sample $U \sim \textbf{Uniform}(0, 1)$
2. Set $X = -\frac{\log(1-U)}{\lambda}$

Acceptance-Rejection method

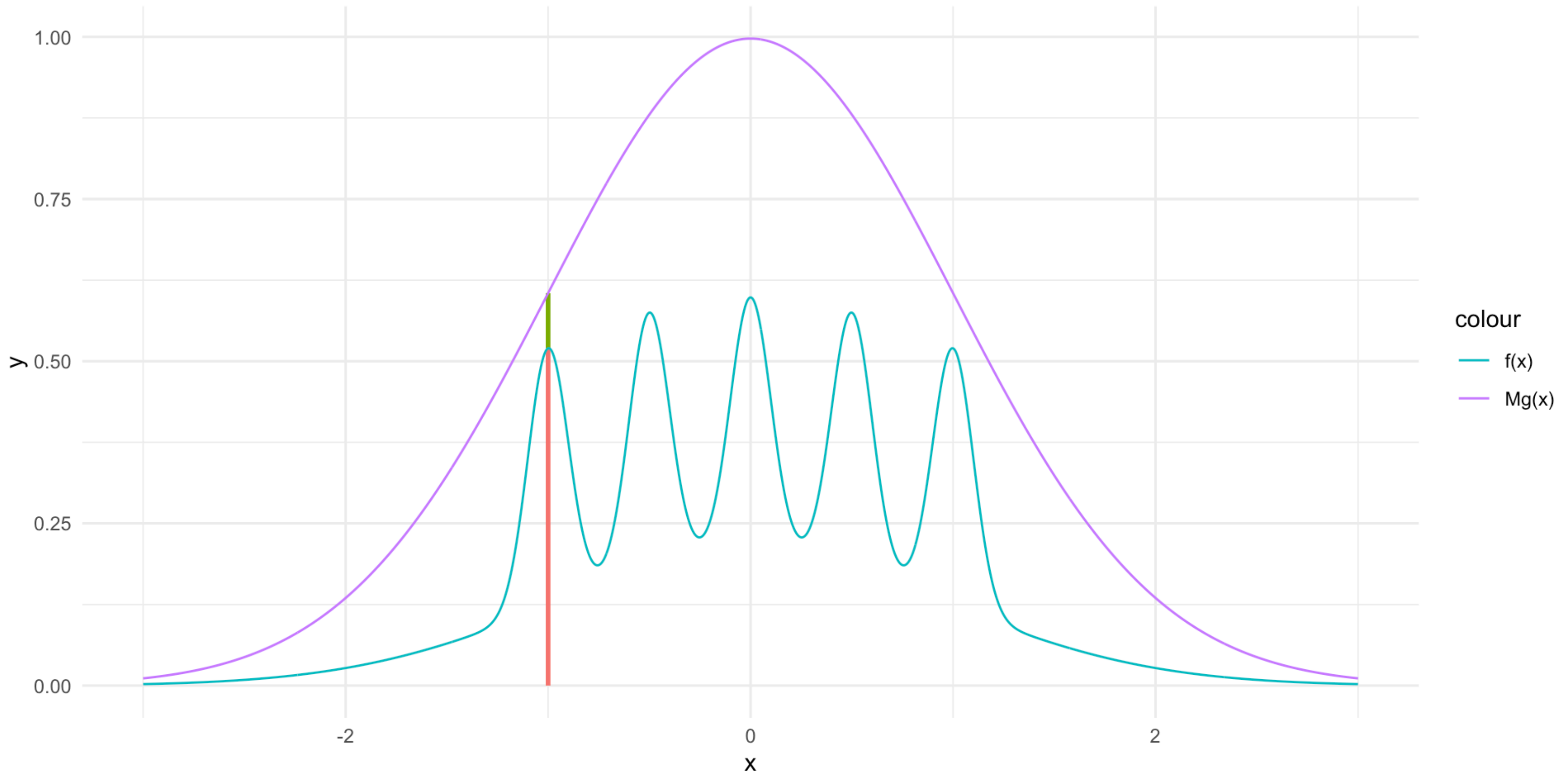
- The problem with the inverse transform method is that you need to know the functional from of the inverse of the cdf
- Acceptance-rejection method is a more general method can simulate “difficult” distributions

Acceptance-Rejection method

- Given two probability densities $f(x)$ and $g(x)$, with $f(x) < Mg(x)$ for all x
- If $g(x)$ is a distribution that can be easily sampled, then we can sample $f(x)$ using the following procedure:
    1. Draw a random variable $X \sim g(x)$
    2. Accept X with probability $\dfrac{f(x)}{Mg(x)}$
    3. Repeat steps 1 and 2 until you have the desired number of random samples

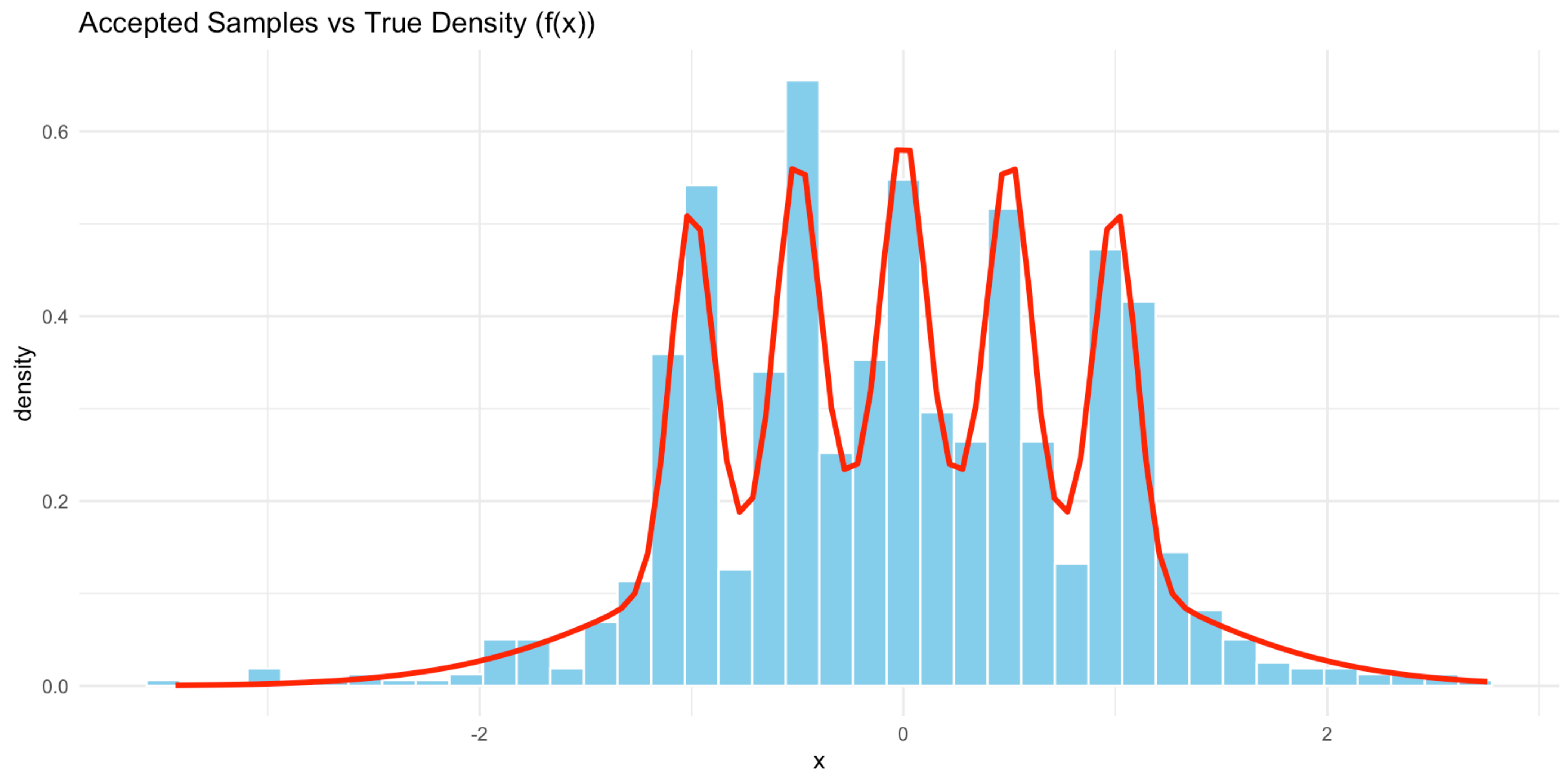
Acceptance-Rejection plot

- $f(x)$ is a normal mixture distribution, difficult to sample directly
- $M = 2.5$
- $g(x) \sim N(0,1)$



Acceptance-Rejection R Demo

▶ Code



Problems with the Acceptance-rejection method

- The envelop distribution $g(x)$ needs to closely resemble $f(x)$ for this method to work well
- If $g(x)$ does not match $f(x)$ well:
  - ➡ The method will draw a lot of unwanted samples hence is not very efficient
  - ➡ Become very problematic when the data are high-dimensional
  - ➡ Even at dimensions of ~ 10, this method becomes very inefficient, i.e., you need to draw lots of samples before one sample is accepted

Monte Carlo simulation examples

Pop Mart Blind Box

- Each Pop Mart Monster blind box costs **$22**.
- There are **10 unique figures** in the full set.
- Each box contains **one random figure**.

❓ How many boxes would you need to purchase — and how much money would you expect to spend — to collect **all 10 figures**?

Analytical Solution (before we see the Monte Carlo)

Expected number of shops you need to do:

$$\mathbb{E}[T] = n\left(\frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}\right) \approx n \log n + 0.5772n + 0.5$$

- T is the number of shops you need to do, and n is the number of items to collect

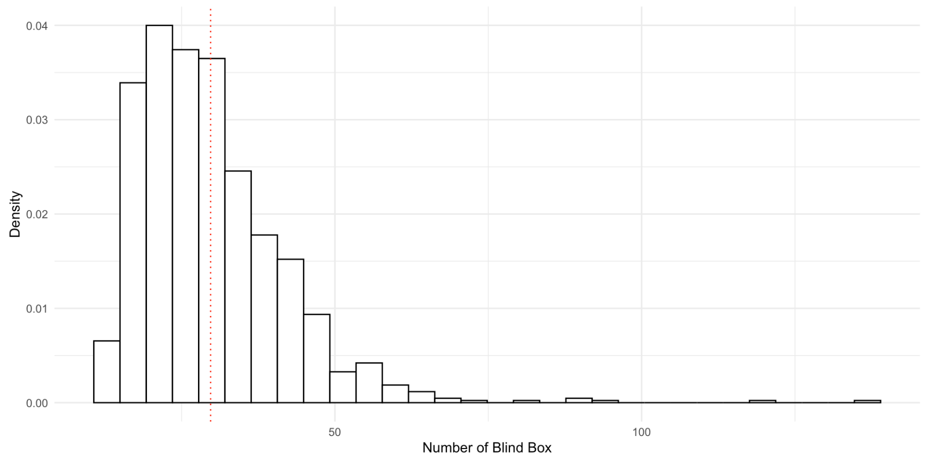
For the example, we have $n = 10$ (the number of unique figures) so $\mathbb{E}[T] \approx 30$

- On average, you need to buy about 30 boxes.
- The expected total money spent is $30 \times 22 = 660$ dollars

Monte Carlo Simulation

```r
n.shops.sim <- c()
for(i in 1:1000) {
  collected <- c(); n.shops <- 0
  while(length(collected) < 10) {
    newitem <- sample(10, 1)
    collected <- union(collected, newitem)
    n.shops <- n.shops + 1
  }
  n.shops.sim <- c(n.shops.sim, n.shops)
}
mean.sim <- mean(n.shops.sim)
mean.sim ## the average number of eligible shops.
ggplot(data.frame(x = n.shops.sim)) +
  theme_minimal() +
  geom_histogram(
    aes(x = x, y = after_stat(density)),
    colour = "black", fill = "white") +
  labs(x = "Number of Blind Box", y = "Density") +
  geom_vline(xintercept = mean.sim,
             linetype = "dotted",
             colour = "red")
```

```
[1] 29.733
```



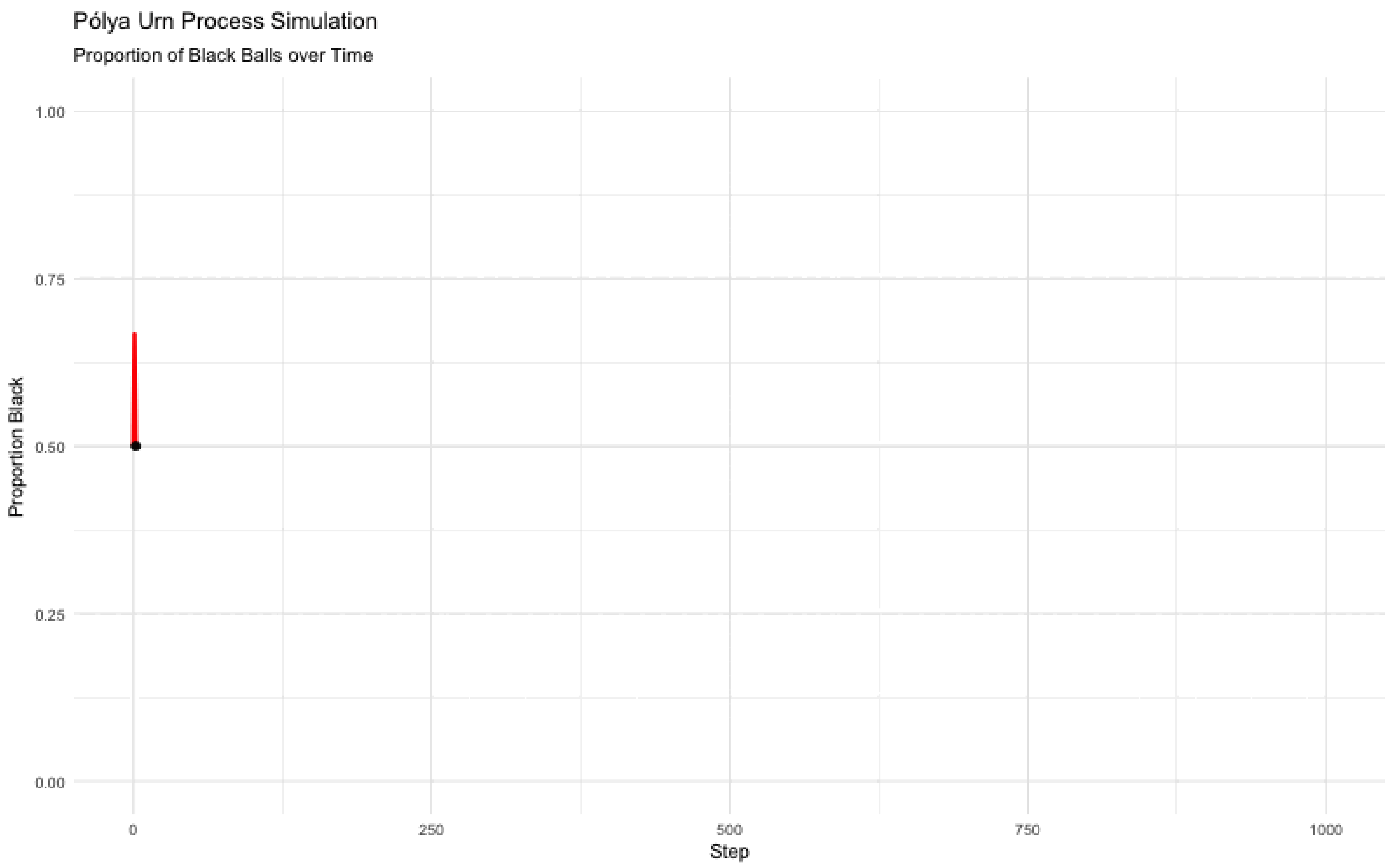
Polya urn problem

- Let an urn start with 1 black ball and 1 white ball
- Repeatedly draw a single ball from the urn
- Each time a ball is drawn:
  - You put back that ball plus one extra ball of the same colour back into the urn

> After 1000 draws, how many black balls do you expect to be in the urn?

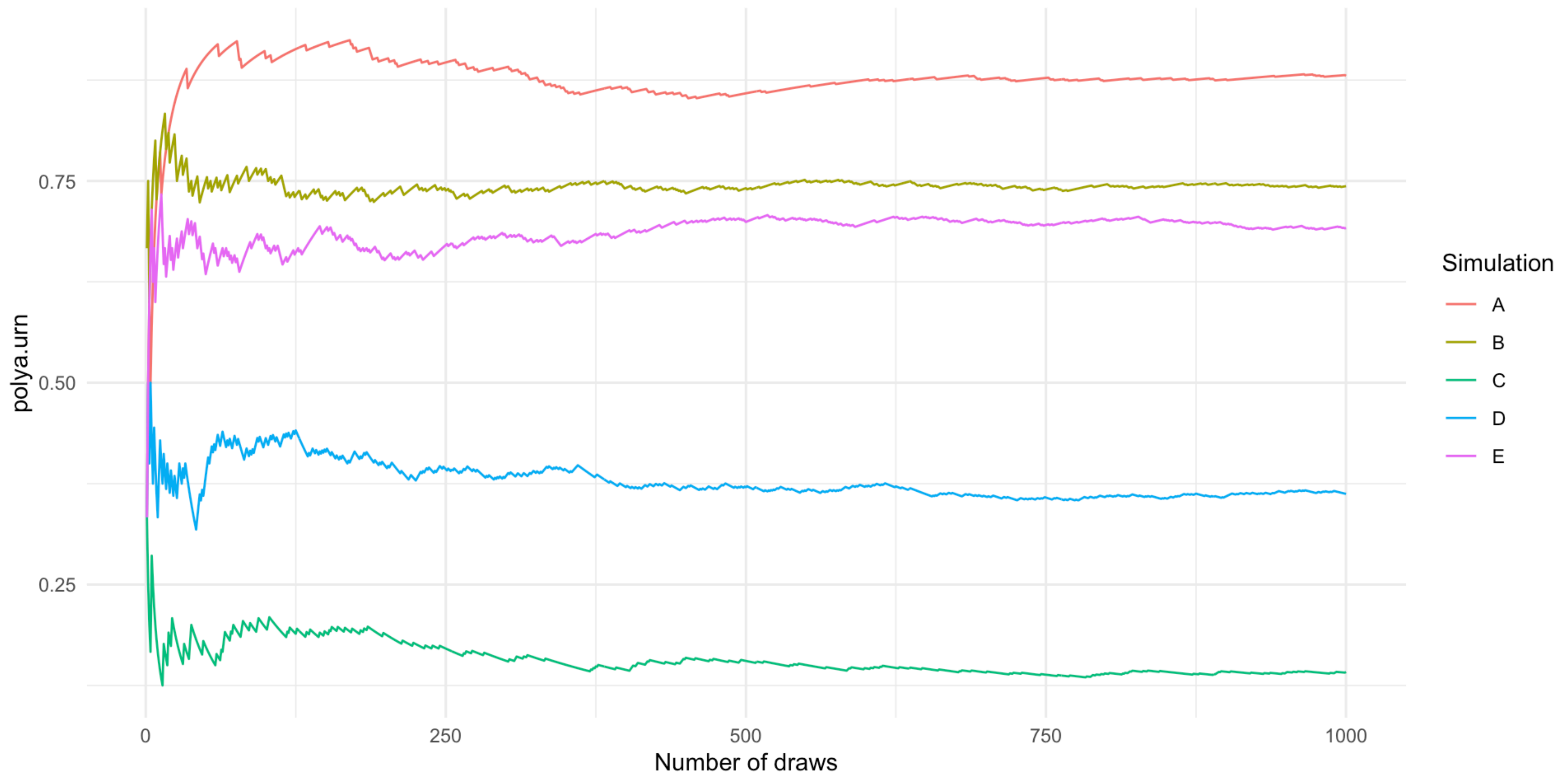
- This is an example of using Monte Carlo to simulate a process

Monte Carlo simulation of the Polya urn problem

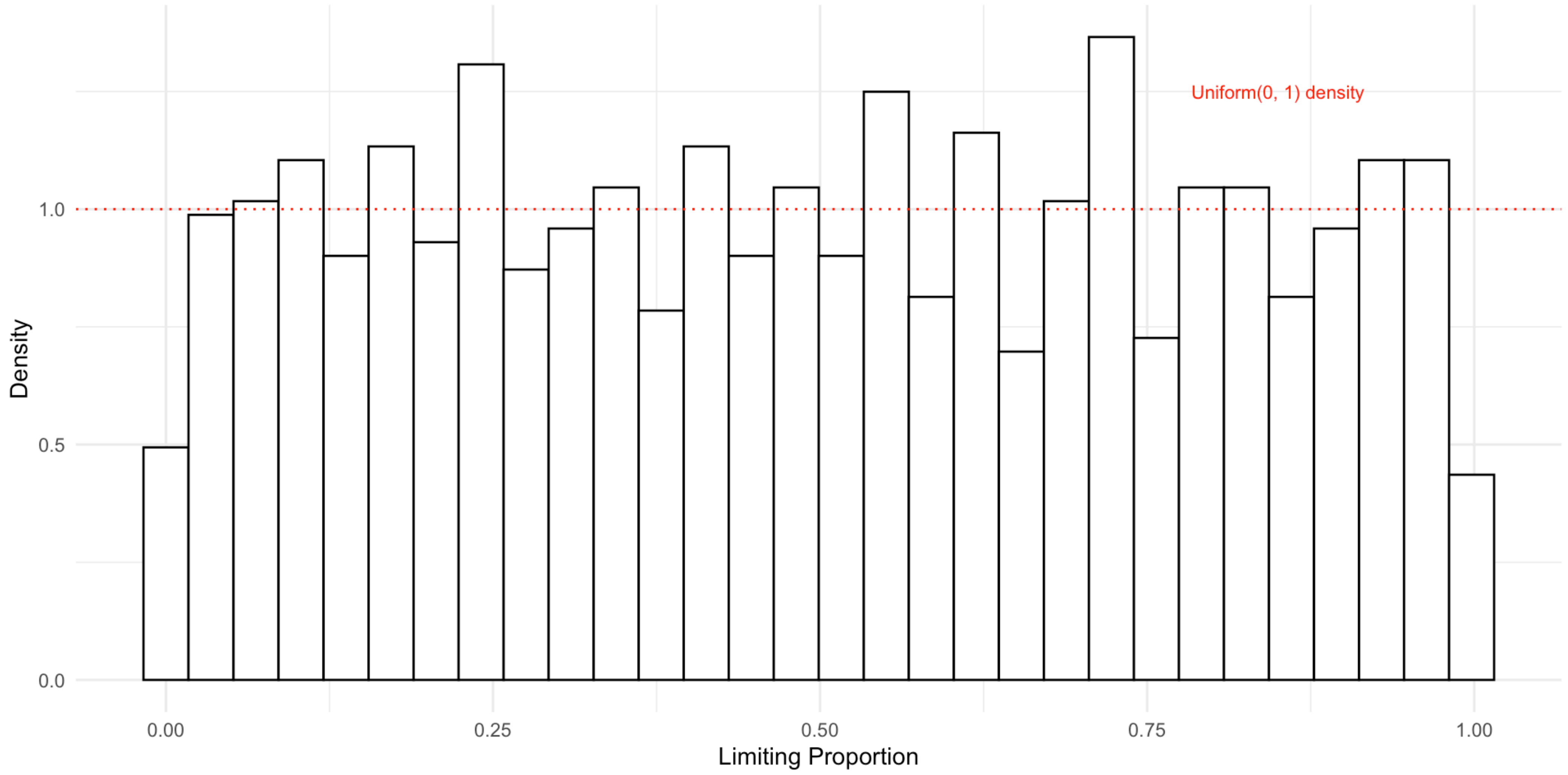


Monte Carlo simulation of the Polya urn problem

▶ Code



Asymptotic distribution of the proportion of black balls



- In the limit, the distribution is $U(0,1)$

Call Option Pricing

An European call option gives you the right to buy a stock at a particular price at expiry

> Example: 90 day call option of a stock with $105 strike. Let's say a stock is currently priced at $100. If it hits $108 at day 90, then you can exercise the option and make a $3 profit from this stock. If its price is below $105 at day 90, then you make nothing.

- Question: **How much should you pay for this option today?**

Think of this as the **insurance premium** you'd pay to guarantee the right to buy later at a known price.

A put option gives you the right to sell a stock at a particular price at expiry

Option Payoff at Expiry

- At expiration, the **payoff** of the call option is:

$$\text{Payoff} = \max(S_T - K, 0)$$

where S_T is the stock price at expiry and K is the strike price.

- There are three possible scenarios:
  - $S_T < K$: Payoff = 0 (option is worthless)
  - $S_T = K$: Payoff = 0
  - $S_T > K$: Payoff = $S_T - K$

Simulate stock price as a random (stochastic) model

Equation for the future price S of a stock:

$$dS = S(\mu dT + \sigma\sqrt{dT}\mathcal{N})$$

- μ is the growth rate of the stock price
- σ is known as the volatility
    - ➟ Think of this as the variance of the stock price
- $\mathcal{N}$ is a standard normal random variable
- dT is one unit of time

Monte Carlo Simulation

1. **Simulate** many possible future stock prices S_T based on the above equation.
2. For each simulation, **calculate payoff**.
3. The price you are willing to pay is the expected value of the payoff $E(\textbf{Payoff})$.

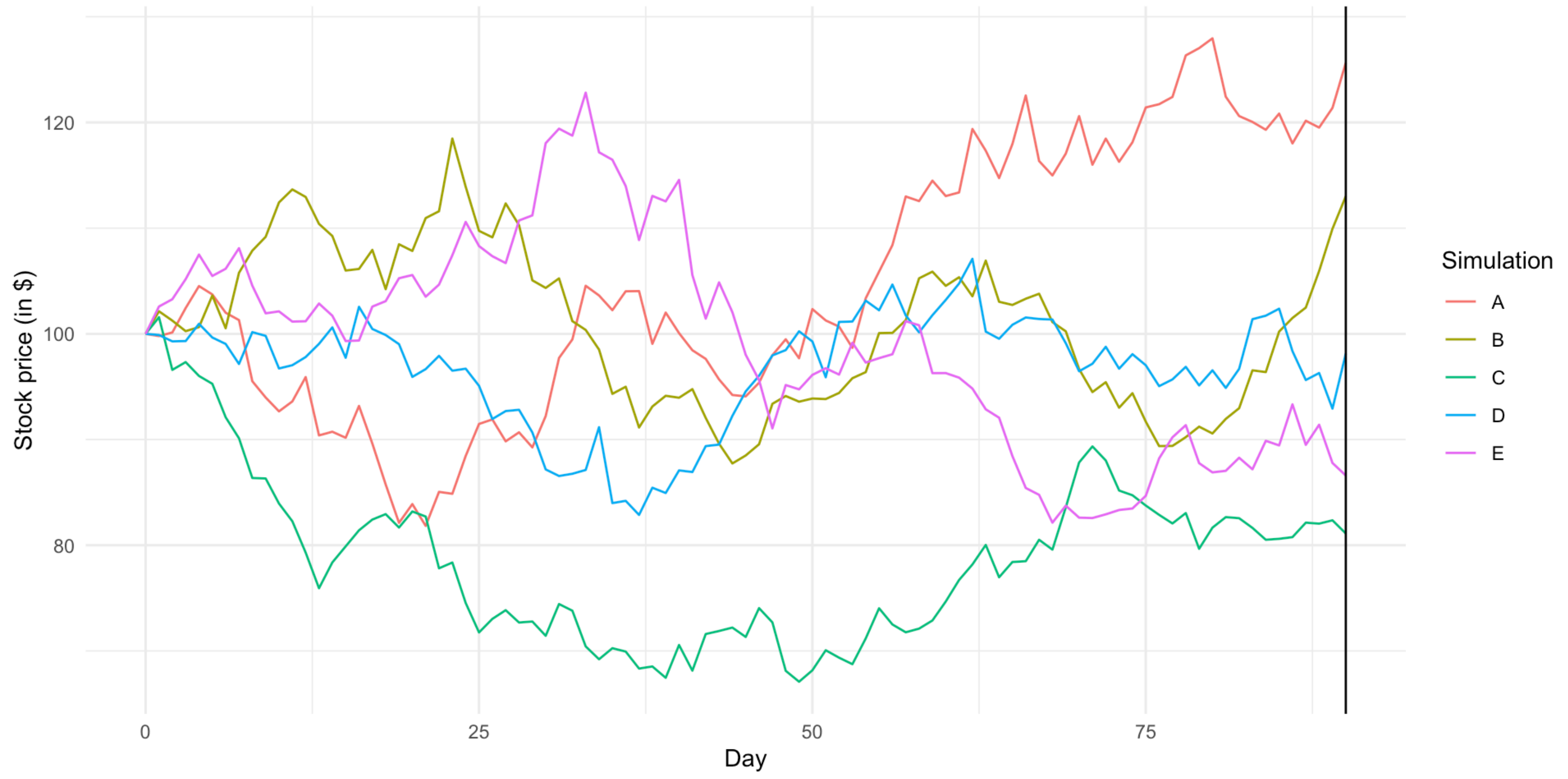
Examples of a stock price

Example: 90 day call option of a stock with $105 strike. currently priced at $100.

▶ Code

```
[1] 8.723601
```

▶ Code


Stock price (in $)
Day
Simulation
A
B
C
D
E