Monte Carlo Methods

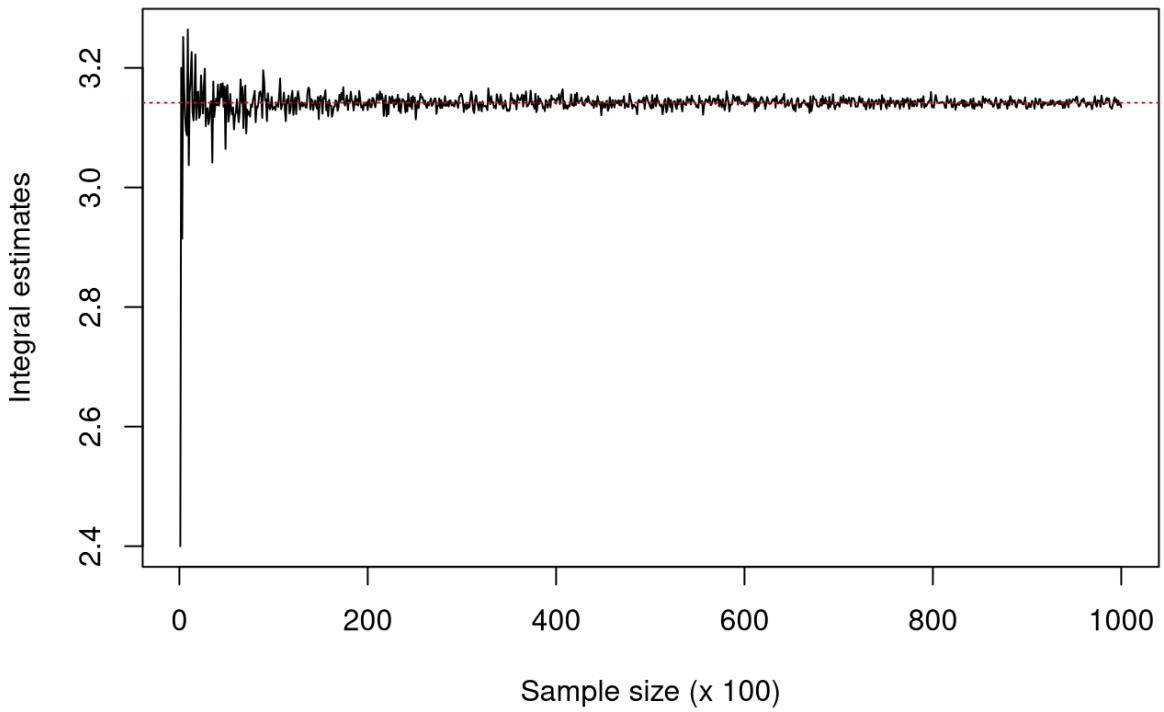
STAT5003

Justin Wishart

Monte Carlo methods

Monte Carlo estimation of π

```
# First create a function that we can simulate random points inside the square on the [-1,1]
 unit,
# the value of Pi = 4(# random pts inside circle / # random pts in square)
est.pi <- function(sample.size) {
    # Generate two vectors for random points in unit circle
    x.pos <- runif(sample.size, min = -1, max = 1)
    y.pos <- runif(sample.size, min = -1, max = 1)

    # Test if points are inside the unit circle
    is.point.inside.circle <- x.pos^2 + y.pos^2 <= 1
    number.of.points.inside.circle <- sum(is.point.inside.circle)
    # Estimate Pi
    return(4*(number.of.points.inside.circle/sample.size))
}

# estimate pi and plot convergence of Monte Carlo estimate
estimates <- c()
for(N in seq(from = 10, to = 100000, by = 100)){
  estimates <- c(estimates, est.pi(N))
}

# plot Monte Carlo estimates with different sampling size
plot(estimates, type="l", xlab="Sample size (x 100)", ylab="Integral estimates")
abline(h = pi, col="red", lty = "dotted")
```




```
pi.estimation.mc <- function(N)
{
    x <- runif(N, min = -1, max = 1)
    y <- runif(N, min = -1, max = 1)
    R <- sum(x^2 + y^2 <= 1)
    4 * R / N
}
```

Consider the quarter circle instead and simulate on $X \sim U(0,1)$ $f(x) = 1$ if $x \in (0,1) = A$ and $f(x) = 0$ otherwise

$$\text{Area of a quarter unit circle} = \pi/4 = \int_0^1 \sqrt{1-x^2}\, dx = \int_A g(x)f(x)\, dx \Rightarrow \pi = 4\int_A g(x)f(x)\, dx \approx 4 \times \frac{1}{N}\sum_{i=1}^{n} g(X_i)$$

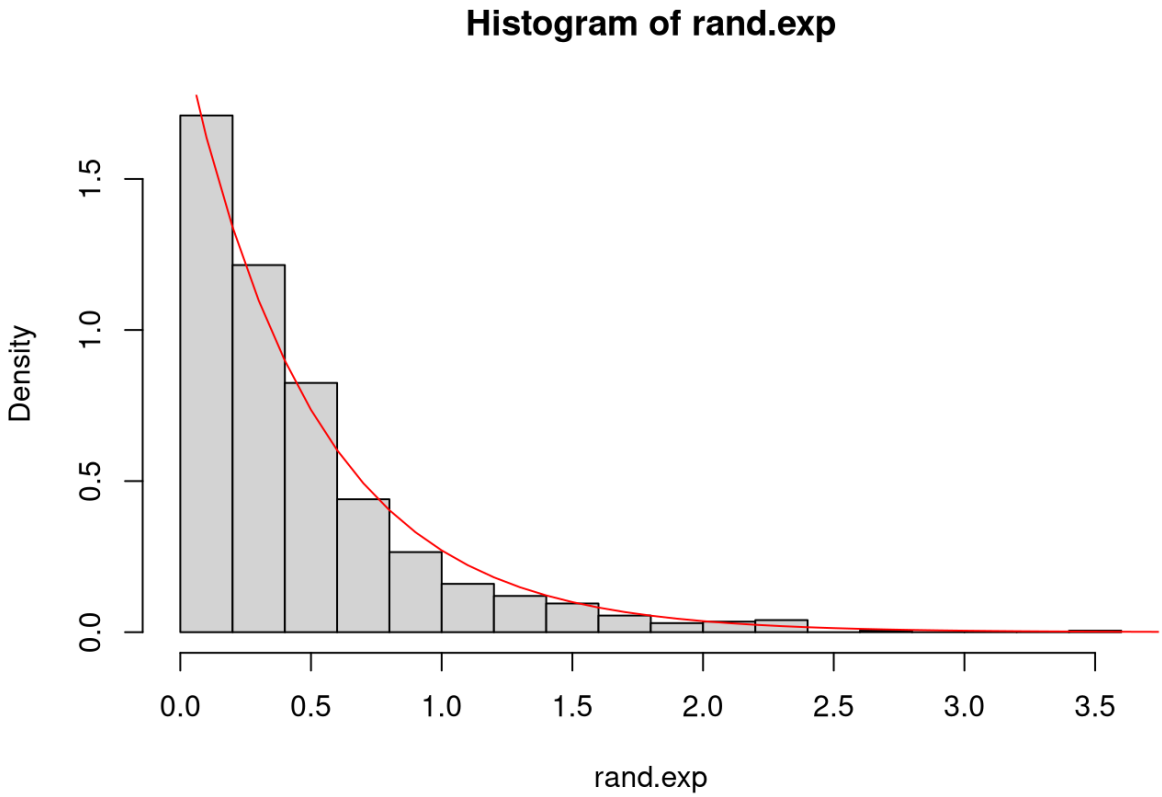
```
single.pi.mc <- function(N)
{
    x <- runif(N, min = 0, max = 1)
    mean(4 * sqrt(1 - x^2))
}
```

Simulating random variables

In the lecture, we learnt how we can use the inverse transform method to simulate random variables from any distribution, as long as we know the function of the inverse CDF.

Let's simulate the exponential distribution and compare with the pdf of the exponential distribution.

```
lambda = 2

# Sample from uniform random distribution
randu <- runif(1000)
rand.exp <- -log(1-randu)/lambda
hist(rand.exp, n=20, freq = FALSE)

y = lambda*exp(-lambda*seq(0,5,0.1))
points(seq(0,5,0.1), y, type = "l", col = "red")
```

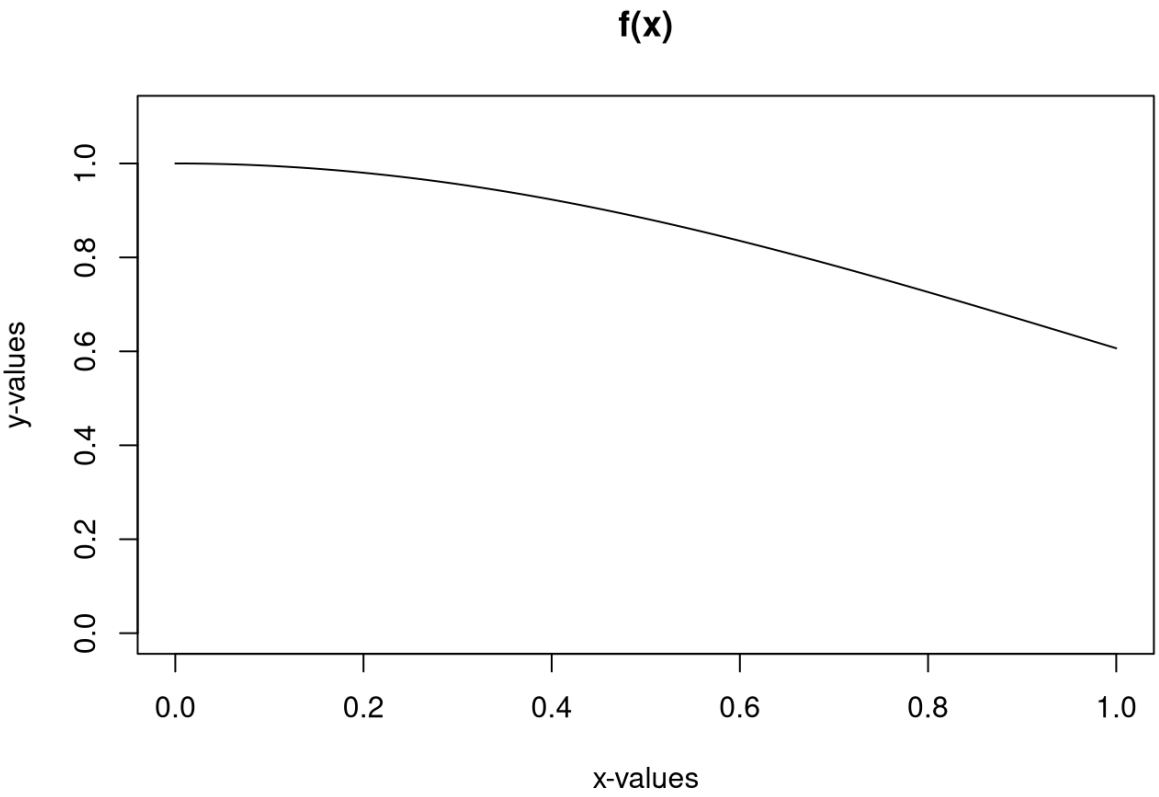


Acceptance rejection method

```
N <- 1e3
sampled <- NULL
f <- wavethresh::dclaw
g <- dnorm
M <- 2.5
i <- 0
while (length(sampled) < N)
{
    x <- rnorm(1) # 1. simulate from g (use rnorm, not dnorm)
    accept <- min(f(x)/(M * dnorm(x)), 1) # 2. Compute acceptance probability.
    if (runif(1) < accept)
        sampled <- c(sampled, x)
    i <- i + 1
}
```

Monte Carlo integration

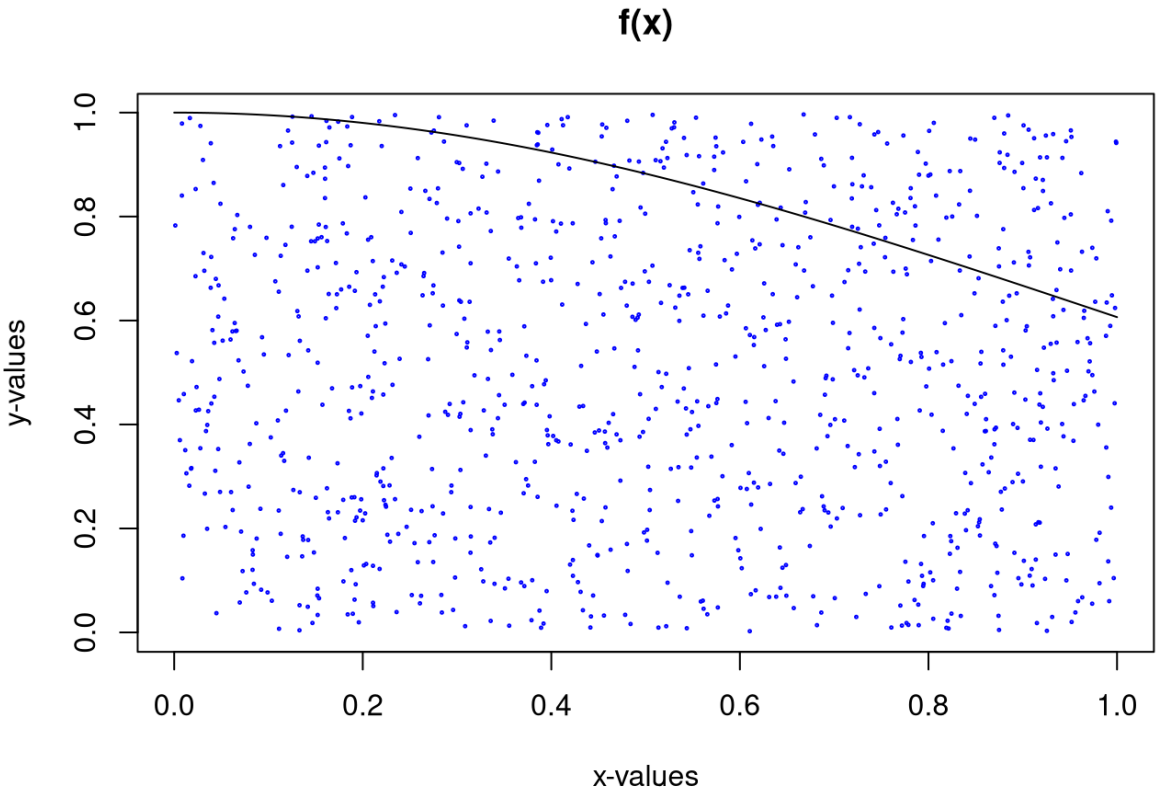
```
# function to be integrated
s <- function(x){(exp(-x^2/2))}

# plot data and create the true relationship line
x.plot <- seq(0, 1, length.out=1000)
y.plot <- s(x.plot)
plot(x.plot, y.plot, xlab="x-values", ylab="y-values", main="f(x)", type="l", ylim=c(0, 1.1))
```




```
# Monte Carlo approach to identify maximum point of the function in a range of interest say 0 to 3
mc.x <- runif(10000, min=0, max=1)
max.y <- max(s(mc.x))
max.y
```

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

```
# now use the maximum estimated by Monte Carlo sampling to samply y with in the range of 1 to max.y
mc.y <- runif(10000, min=0, max=max.y)
plot(mc.x[1:1000], mc.y[1:1000], col="blue", xlab="x-values", ylab="y-values", main="f(x)", cex = 0.2)
#
lines(x.plot, y.plot, type="l")
```




```
#
# count how many points fall below the curve
area.ratio <- sum(mc.y < s(mc.x)) / 10000
auc <- area.ratio * max.y
auc
```

```
## [1] 0.854
```

```
##
# A simpler (more straightforward) approach.
# This is the more 'standard' approach for Monte Carlo integration

n = 10000
x <- runif(n, min=0, max=1)
sx <- s(x)
ans.mc <- mean(sx)
ans.mc
```

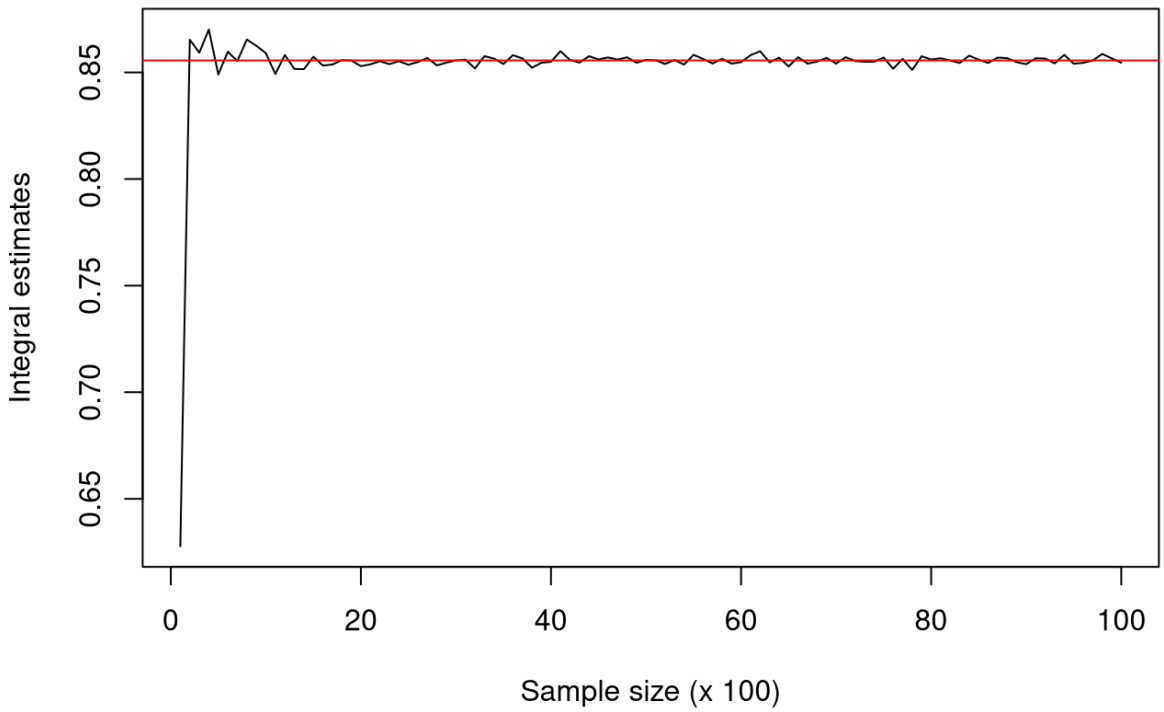
```
## [1] 0.8564566
```

```
##
# Compare our Monte Carlo solution to 'adaptive quadrature', which is a
# deterministic method for doing numerical itegrations. For well-behaved
# 1D functions, quadrature methods give very accurate estimates of the
# integral.
#
# Note: 'integrate' is a standard R function for doing quadratures.
##
ans.quad <- integrate(s, 0, 1)
ans.quad
```

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

Convergence of Monte Carlo estimate

```
estimates <- c()
for(n in seq(1, 10000, by=100)){
  x <- runif(n, min=0, max=1)
  sx <- s(x)
  ans <- mean(sx)
  estimates <- c(estimates, ans)
}

# plot Monte Carlo estimates with different sampling size
plot(estimates, type="l", xlab="Sample size (x 100)", ylab="Integral estimates")
abline(h = ans.quad[[1]], col = "red")
```



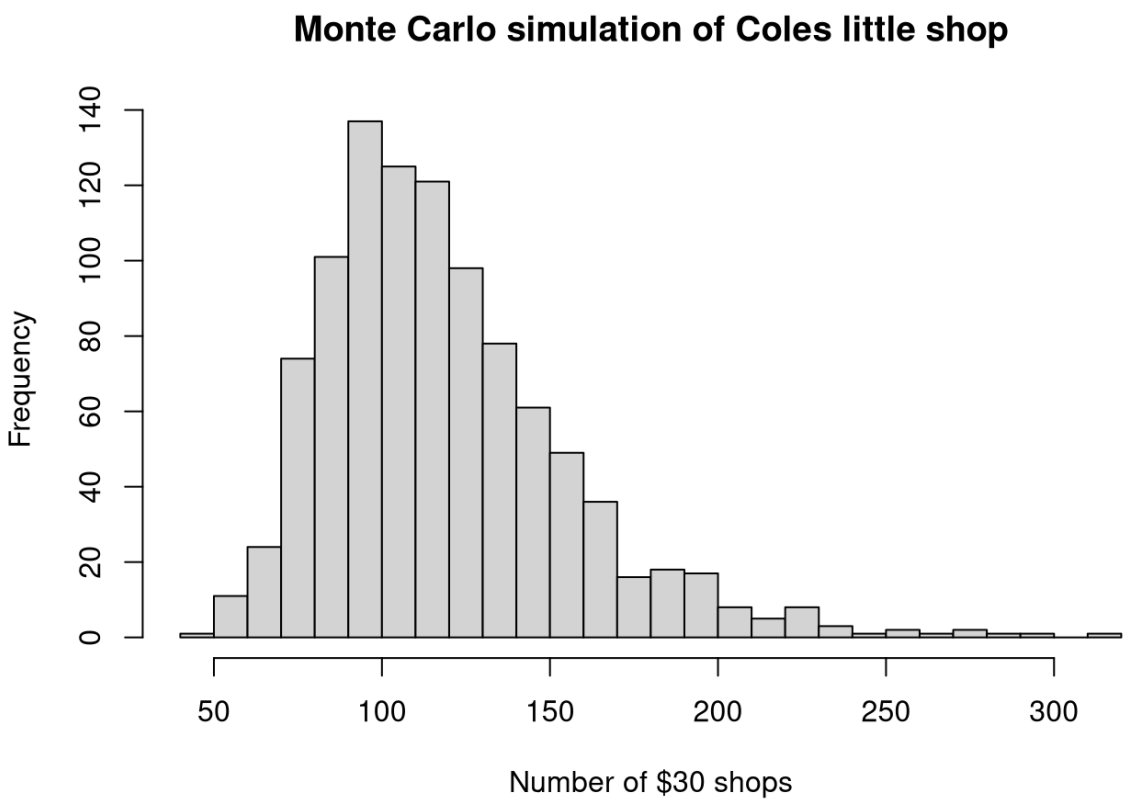
Monte Carlo simulation examples

Cole's Little Shop

```
n.shops.sim <- c()
for(i in 1:1000) {
   collected <- c()
   n.shops <- 0
   while(length(collected) < 30) {
      newitem <- sample(30,1)
      collected <- union(collected, newitem)
      n.shops <- n.shops + 1
   }
   n.shops.sim <- c(n.shops.sim, n.shops)
}

# Expected number of shops you need to do
mean(n.shops.sim)
```

```
## [1] 119.167
```

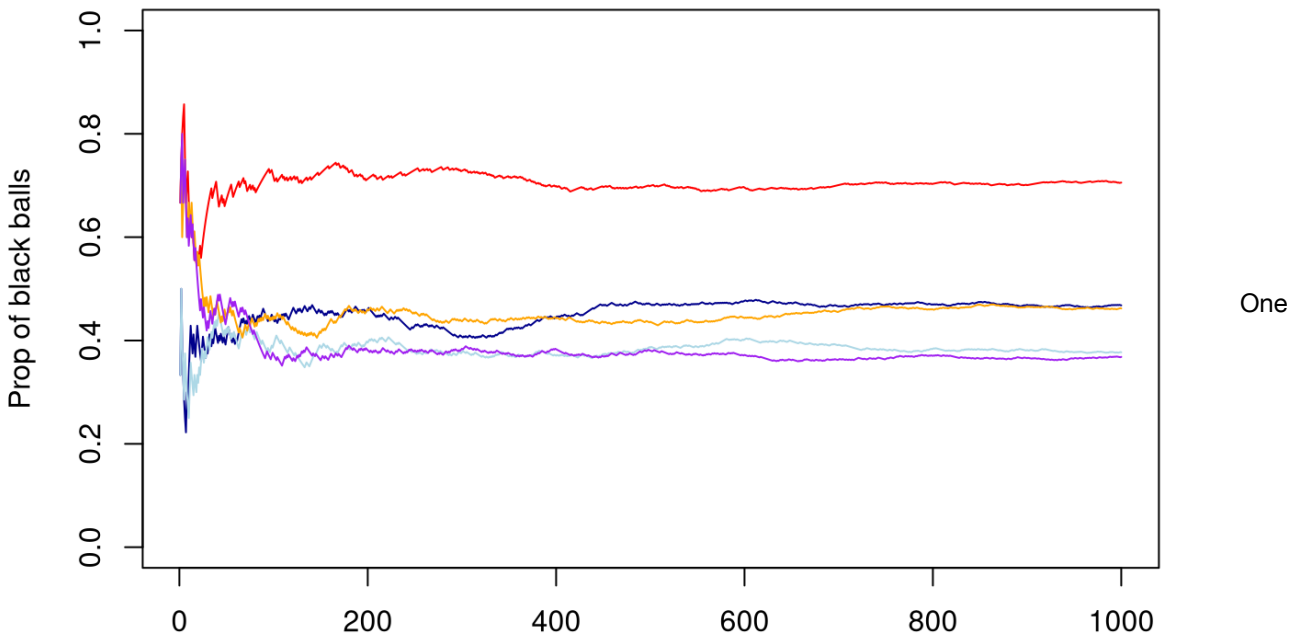
```
# Plot the histogram of number of shops required
hist(n.shops.sim, n=30, xlab = "Number of $30 shops", main =   "Monte Carlo simulation of Col
es little shop")
```



Polya urn

In the polya urn model, we start of with one black ball and one white ball in the urn. In each step, we draw a ball and put that ball plus one extra ball of the same colour back in the urn.

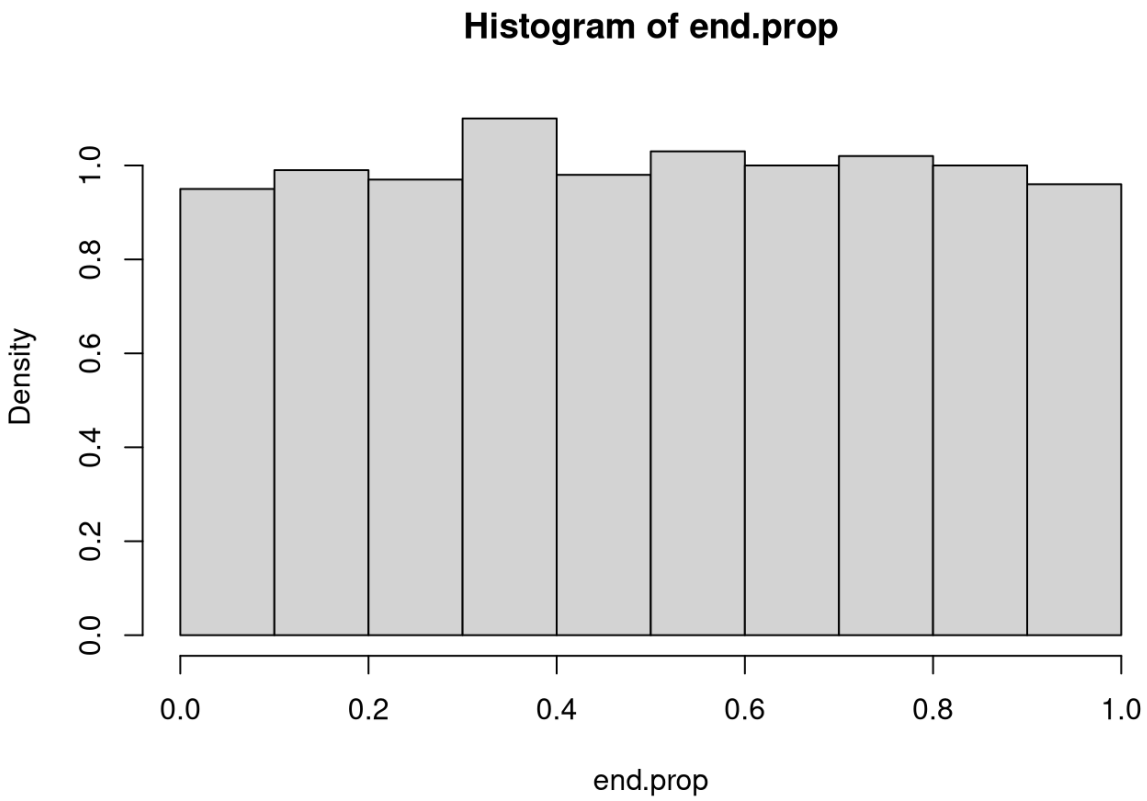
```
onerun <- function() {
  urn <- c("white", "black")
  prop.blacks <- c()
  for(step in 1:1000) {
     flip <- sample(urn, 1)
     if (flip == "white") {
        urn <- c(urn, "white")
     } else {
        urn <- c(urn, "black")
     }

     nblack <- sum(urn == "black")
     nballs <- length(urn)
     prop.blacks <- c(prop.blacks, nblack/nballs)

  }
  return(prop.blacks)
}

run1 <- onerun()
run2 <- onerun()
run3 <- onerun()
run4 <- onerun()
run5 <- onerun()

plot(run1, col = "red", type = "l", ylim = c(0,1), ylab = "Prop of black balls", xlab = "")
points(run2, col = "dark blue", type = "l")
points(run3, col = "light blue", type = "l")
points(run4, col = "orange", type = "l")
points(run5, col = "purple", type = "l")
```



One

interesting property of this polya urn model is that the proportion of black balls at time infinity converges to a uniform distribution. Let's check this is true.

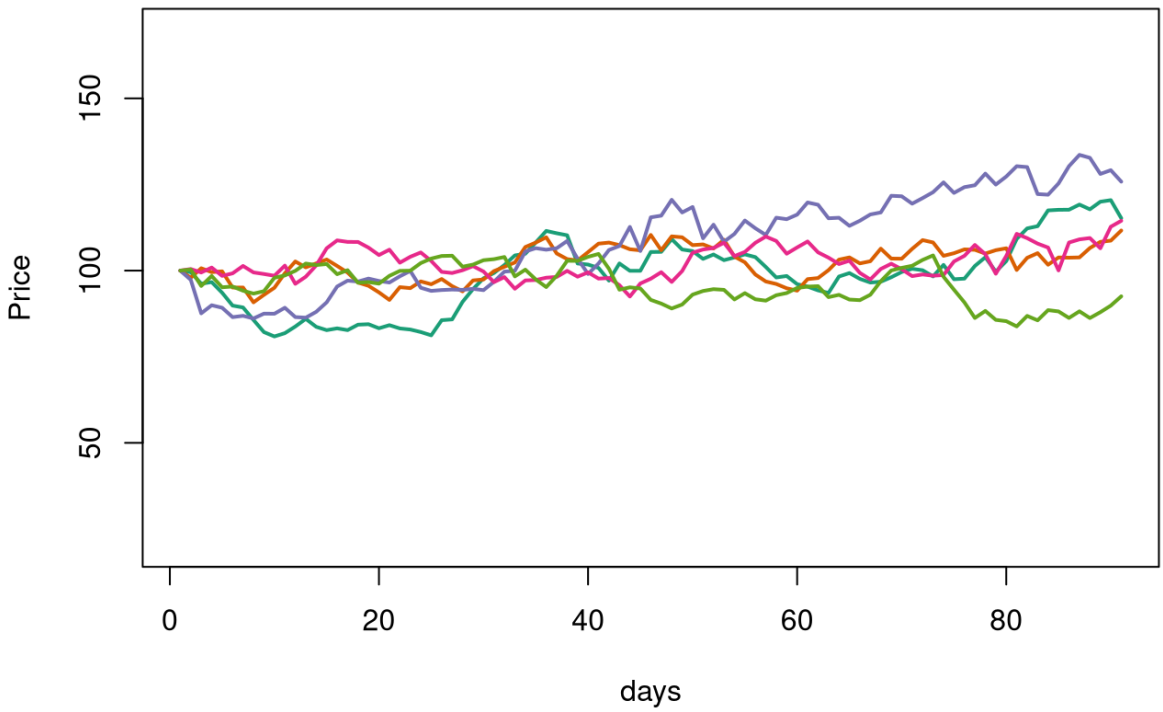
```
end.prop <- c()
for (i in 1:1000) {
  prop.blacks <- onerun()
  end.prop <- c(end.prop, prop.blacks[length(prop.blacks)])
}

hist(end.prop, n = 10, prob = TRUE)
```



Stock price and options pricing

Assuming stock price follows a geometric Brownian motion random walk, let's simulate some stock prices and use them to price a call option.

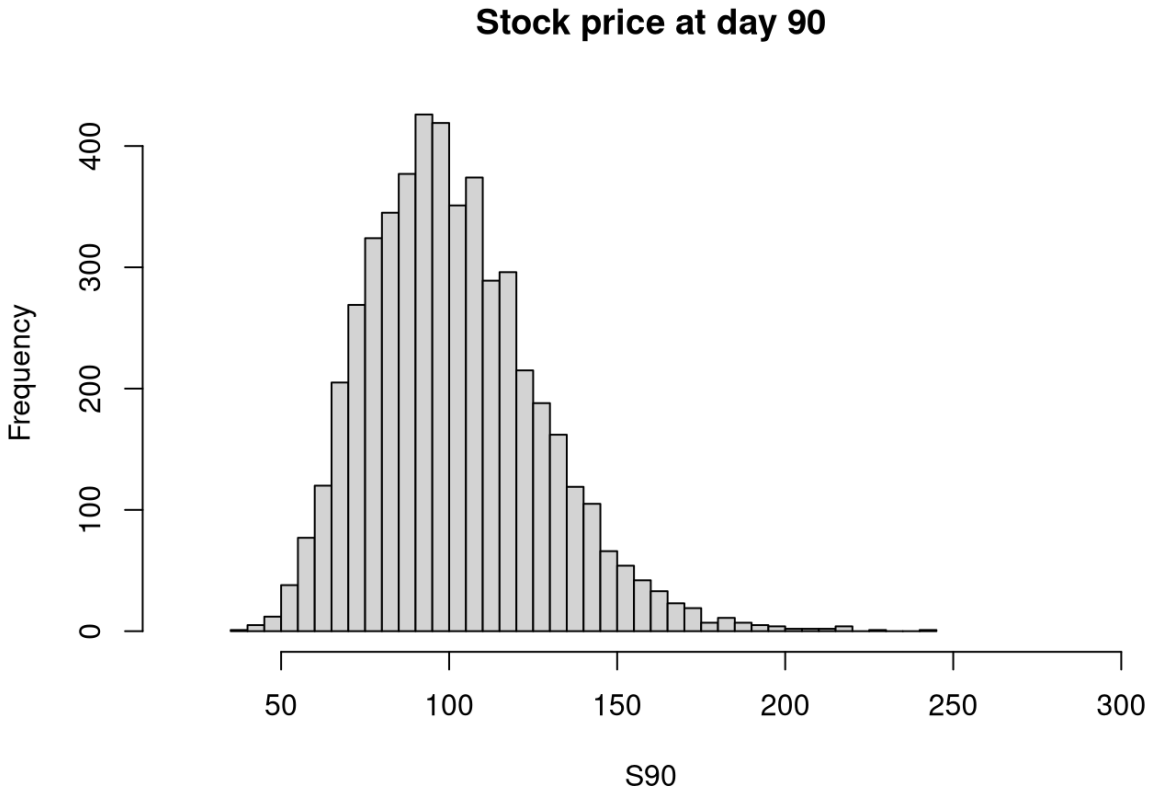
```
ndays <- 90
S0 <- 100 # Starting stock price
mu <- 0.05 # Stock drift rate per year
dT <- 1/365
sigma <- 0.5 # volatility
strike <- 105 # strike price of call option
# Draw some normal random numbers ~ N(0,1)
niters <- 5000 # Number of Monte carlo simulations to do

option.payoff.arr <- c()

stock.prices <- list()
S90 <- c()
for (j in 1:niters) {
   N <- rnorm(ndays)
   S <- S0
   S_arr <- c(S0)
   for(i in 1:ndays) {
      dS <- S*(mu*dT + sigma*sqrt(dT)*N[i])
      S <- S + dS
      S_arr <- c(S_arr, S)
   }

   S90 <- c(S90, S)
   stock.prices[[j]] <- S_arr

   # For a call option, if the stock price at expiry is
   # higher than strike price, then we profit
   # Otherwise, we don't exercise the option
   option.payoff = max(S-strike,0)
   option.payoff.arr <- c(option.payoff.arr, option.payoff)
}

library(RColorBrewer)
palette <- brewer.pal(7,"Dark2")
plot(stock.prices[[1]], type = "l", lwd = 2, col = palette[1], ylim = c(20,170), xlab = "day
s", ylab = "Price")
points(stock.prices[[2]], type = "l", lw = 2,col = palette[2])
points(stock.prices[[3]], type = "l", lw = 2,col = palette[3])
points(stock.prices[[4]], type = "l", lw = 2,col = palette[4])
points(stock.prices[[5]], type = "l", lw = 2,col = palette[5])
```




```
# Price of $105 call options is the expected value of the payoffs
print(mean(option.payoff.arr))
```

```
## [1] 8.615929
```

```
# You can see the stock price is not a normal distribution
# It is a lognormal distribution
hist(S90, n = 50, xlim = c(20,300), main = "Stock price at day 90")
```


Stock price at day 90
Frequency
0
100
200
300
400
50
100
150
200
250
300
S90