Lab 6: Simulating functions of continuous random variables

Professor Scott

Spring 2025

Task 1: Simulating a triangle

(Dobrow 6.57/6.8) An isosceles right triangle has side length uniformly distributed on the unit interval $(0,\ 1)$. Simulate 500 hypotenuses for such an isosceles right triangle.

Code set-up

`runif(500,0,1)` will generate 500 variates from a $uniform(0,\ 1)$ distribution. Recall that for an isosceles right triangle, the hypotenuse is $\sqrt{2}$ times the length of one of the sides. You can use this formula to simulate the hypotenuse length from a simulated side legnth.

To plot the triangles, you need merely to plot the hypotenuse on a line using the `lines` function. If we stored our uniform random variates in a vector `x`, the following code chunk will plot the ith hypotenuse generated. This single hypotenuse is not very exciting. But if you wrap this code in a for-loop over the 500 simulation experiments, it will fill in the "triangle space".

```
# set up the plot by outlining the border from 0 to 1 on each of the triangle side 1 and side 2
#plot(c(0,1,0),c(1,0,0), type="p", xlab="Triangle side 1", ylab="Triangle side 2")
#i = 17 # plot the 17th hypotenuse; remove this line once you set-up the for-loop over i
# draw the hypotenuse
#lines(c(0,x[i]),c(x[i],0))
```

The problem: Simulate 500 hypotenuses for the isosceles right triangle

```
# Dobrow Exercise 6.57, based on 6.8
# True expected value and variance are sqrt(2)/2 and 1/6
n = 500
sides = runif(n,0,1)
hyp = numeric(n)

for (i in 1:n) {
  hyp[i] = sides[i]*sqrt(2)
}

mean(hyp)
```

```
## [1] 0.7170213
```

```r
var(hyp)
```

```
## [1] 0.1618542
```

Report the following:

- The empirical (simulated) expected length and variance of the length of the hypotenuse.

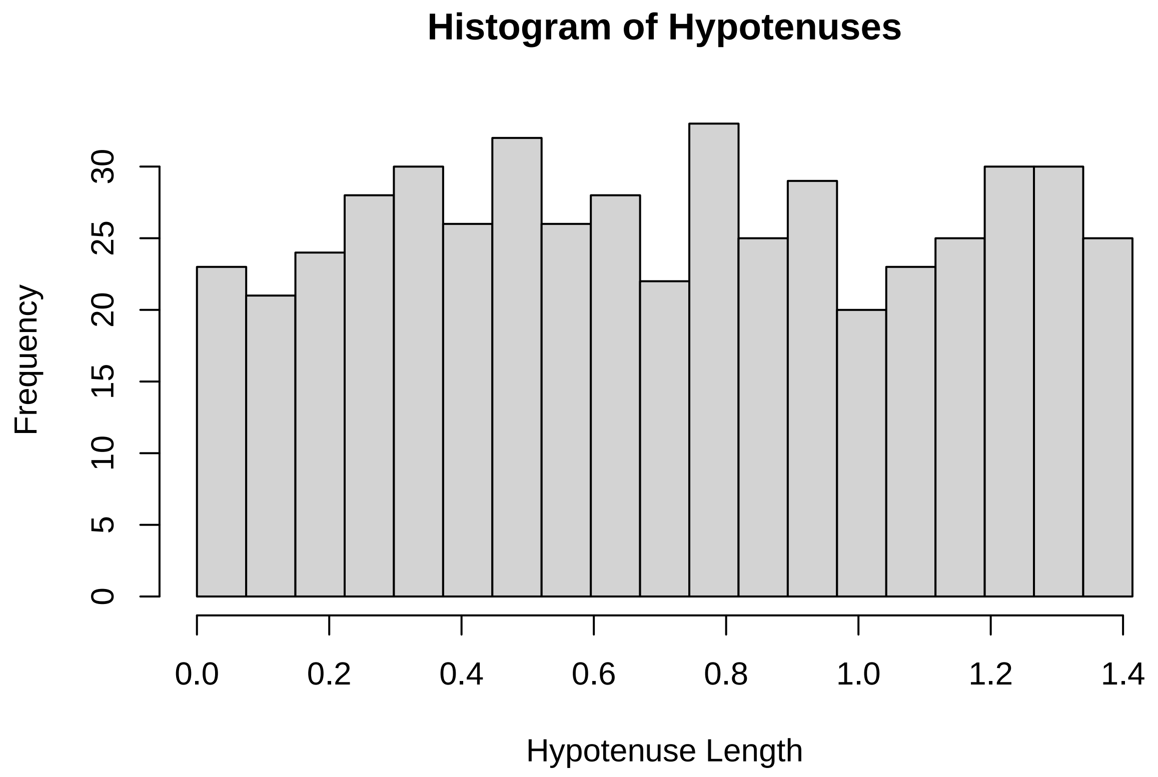
The simulated mean is ~0.7 and the simulated variance is ~0.16

- Since a side has length $X \sim U(0,\ 1)$, the hypotenuse $H = X \cdot \sqrt{2}$ has expected value $E(H) = \sqrt{2} \cdot E(X) = \sqrt{2}/2$. $VAR(H) = 2VAR(X) = 1/6$. Compare the truth with your empirical values in the previous bullet.

E(X)=(0+1)/2=0.5, E(H)=sqrt(2).E(X)=0.707, VAR(X)=(1-0)^2/12=0.833, VAR(H)=2VAR(X)=0.167 The empirical values and the true values are incredibly close to each other. The simulated values only tend to be off by +- 0.1 when ran with an n of 500.

- Present a histogram of the simulated hypotenuse lengths. What distribution does it seem to follow and why?

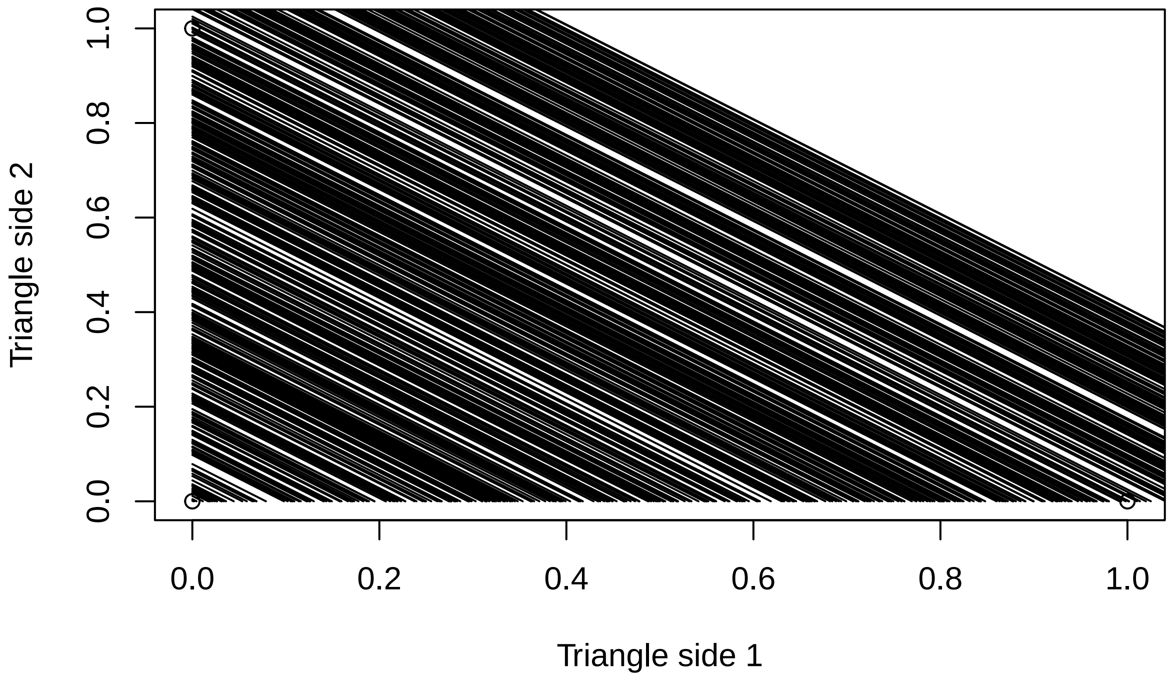
```r
hist(hyp,breaks=seq(0, sqrt(2), length.out=20),main="Histogram of Hypotenuses", xlab="Hypotenuse Length
```



Given the expected values and variance were so close, it is no surprise that the histogram for the hypotenuses resembles a uniform distribution

- Present, on a single set of axes, the simulated hypotenuses using the triangle_plot code chunk above. That is, plot a diagonal line representing the hypotenuse for each simulated value. What do you find?

```r
plot(c(0,1,0),c(1,0,0), type="p", xlab="Triangle side 1", ylab="Triangle side 2")
for (i in 1:n) {
  lines(c(0,hyp[i]),c(hyp[i],0))
}
```



The plot is almost a completely blacked-out plot of lines. The diagonal lines seem to represent the hypotenuse, so drawing a line until the x and y axis of the hypotenuse endpoints intersect would create the triangle

Task 2: Simulating Gaussian probabilities

Finding areas under the normal curve (integrating with respect to the normal pdf) is analytically quite difficult due to the form of the pdf. However, Monte Carlo estimates via simulation are quick and accurate.

The Box-Muller algorithm is a popular Gaussian simulator, applying a transformation method to convert uniform variates into normal variates. The algorithm for simulating two random variates Z_1 and Z_2 from a $N(0,\ 1)$ distribution is as follow.

Box-Muller Gaussian simulator

```
i.  Generate $U_1, U_2 \sim U(0,1)$, two uniform random variates.
ii. Compute $Z_1 = \sqrt{-2\ln(U_1)}\cos(2\pi U_2)$ and $Z_2 = \sqrt{-2\ln(U_1)}\sin(2\pi U_2)$.
```

Recall that if we want a variate $X \sim N(\mu, \sigma^2)$, we can compute $X = \sigma Z + \mu$ for both Z_1 and Z_2.

Code set-up

For those who want to try, the Box-Muller Gaussian simulator can be simulated without for-loops by using a matrix of generated uniform variates. Otherwise, code each of the items in the algorithm to generate two variates. Then wrap that in a for-loop to repeat and generate the desired number of Gaussian variates.

The following code will draw a histogram of your variates, x, along with a density plot for evaluation.

```
hist(x, xlab="N(2,2) variates", main="", freq=FALSE)
```

```
lines(seq(-4,8,0.05), dnorm(seq(-4,8,0.05),2,sqrt(2)), col="blue")
```

The problem

Use the Box-Muller algorithm to simulate 1000 random variates of $X \sim N(2,\ 2)$.

```r
N = 1000
Z = matrix(ncol=2, nrow=N)
X = numeric(N)
Y = numeric(N)

for (i in 1:N) {
  U1 = runif(1,0,1)
  U2 = runif(1,0,1)
  Z[i,1]=sqrt(-2*log(U1))*cos(2*pi*U2)
  Z[i,2]=sqrt(-2*log(U1))*sin(2*pi*U2)
  X[i]=sqrt(2)*Z[i,1]+2
  Y[i]=sqrt(2)*Z[i,2]+2
}
```

Report the following:

a) Mean, standard deviation, and median of the variates.

```r
mean(X)
```

```
## [1] 2.015037
```

```r
mean(Y)
```

```
## [1] 2.020547
```

```r
sd(X)
```

```
## [1] 1.448403
```

```r
sd(Y)
```

```
## [1] 1.441995
```

```r
median(X)
```

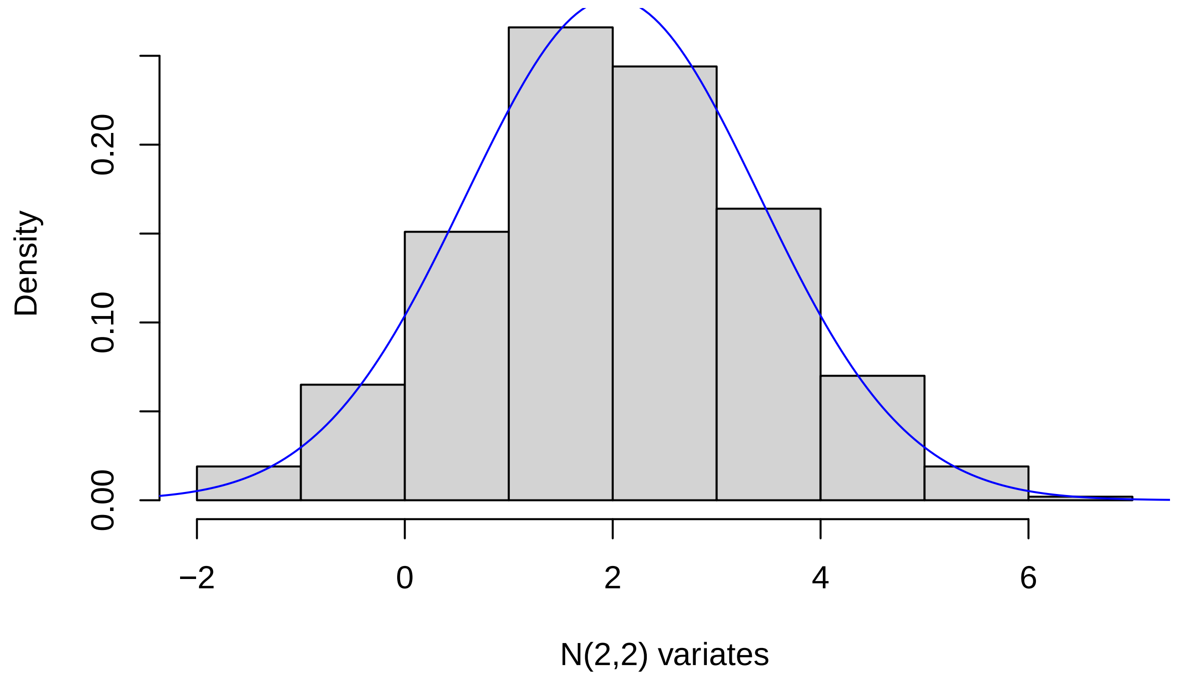
```
## [1] 1.998208
```

```r
median(Y)
```

```
## [1] 2.008994
```

b) Histogram of the variates and an overlay with the true density.

```r
hist(X, xlab="N(2,2) variates", main="", freq=FALSE)
lines(seq(-4,8,0.05), dnorm(seq(-4,8,0.05),2,sqrt(2)),col="blue")
```



c) The proportion of simulated values out of the 1000 that lie one, two, and three standard deviations from the mean (that is, within $\mu \pm \sigma$, $\mu \pm 2\sigma$, and $\mu \pm 3\sigma$). These percentages are estimates of $P(\mu - i\sigma < X < \mu + i\sigma)$ for $i = 1, 2, 3$.

```
print(sprintf("One std dev above: %f", pnorm(mean(X)+sd(X),2,2,lower.tail=FALSE)))
```

```
## [1] "One std dev above: 0.232170"
```

```
print(sprintf("One std dev below: %f", pnorm(mean(X)-sd(X),2,2)))
```

```
## [1] "One std dev below: 0.236785"
```

```
print(sprintf("Within one std dev: %f", pnorm(mean(X)+sd(X),2,2)-pnorm(mean(X)-sd(X),2,2)))
```

```
## [1] "Within one std dev: 0.531045"
```

```
print(sprintf("Two std dev above: %f", pnorm(mean(X)+2*sd(X),2,2,lower.tail=FALSE)))
```

```
## [1] "Two std dev above: 0.072707"
```

```
print(sprintf("Two std dev below: %f", pnorm(mean(X)-2*sd(X),2,2)))
```

```
## [1] "Two std dev below: 0.074809"
```

```
print(sprintf("Within two std devs: %f", pnorm(mean(X)+2*sd(X),2,2)-pnorm(mean(X)-2*sd(X),2,2)))
```

```
## [1] "Within two std devs: 0.852484"
```

```
print(sprintf("Three std dev above: %f", pnorm(mean(X)+3*sd(X),2,2,lower.tail=FALSE)))
```

```
## [1] "Three std dev above: 0.014624"
```

```
print(sprintf("Three std dev below: %f", pnorm(mean(X)-3*sd(X),2,2)))
```

```
## [1] "Three std dev below: 0.015191"
```

```
print(sprintf("Within three std devs: %f", pnorm(mean(X)+3*sd(X),2,2)-pnorm(mean(X)-3*sd(X),2,2)))
```

```
## [1] "Within three std devs: 0.970185"
```

d) The true values of the three probabilities in (c) are 0.68, 0.95, and 0.997 respectively. Compare your estimates from (c) with the true values. This is called the "68-95-99.7" rule for the Gaussian/Normal distribution.

The probabilities are somewhat close, but definitely not exact. It seems like, for an approximation, it would be close enough to get ballpark guesses. As the standard deviations increase, it seems like the estimation gets better.