```r
set.seed(911)

#1.

  #Beta(2,2)
    #What does Beta(2,2) equal?
      gamma(2)
```

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

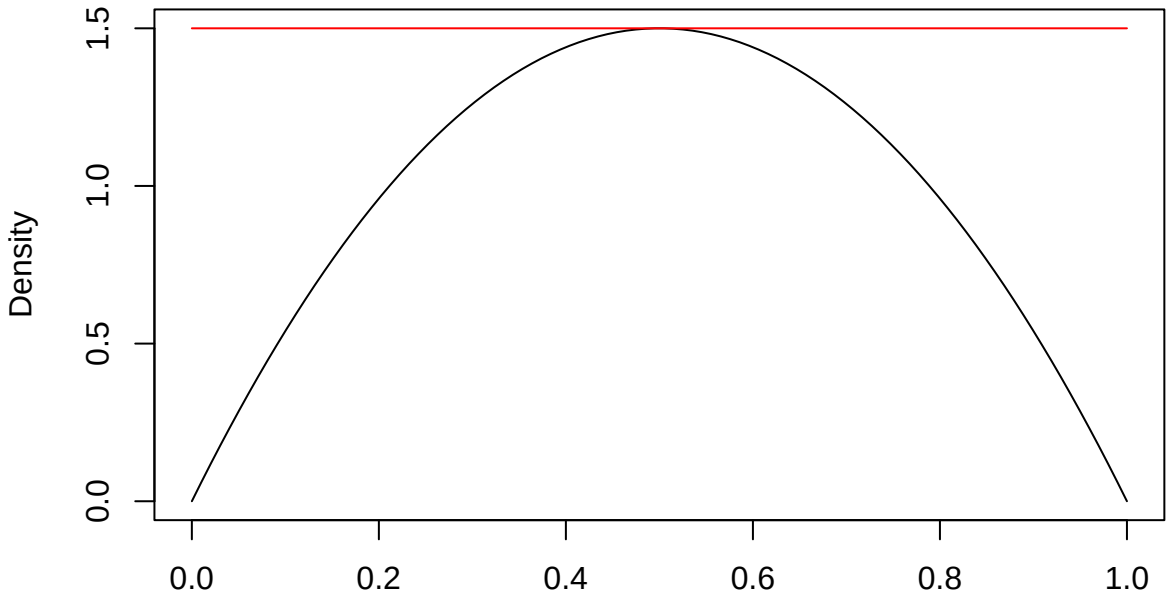
```r
      gamma(4)
```

```
## [1] 6
```

```r
      #Beta(2,2) = 6x - 6x^2
      fX <- function(x) {
        return(6*x - 6*x^2)
      }
    #Plotting Beta(2,2)
      grid <- seq(0, 1, length.out=100)
      plot(fX(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
    #Given that Beta(2,2) takes it's maximum at x=0.5, what is this maximum?
      fX(0.5)
```

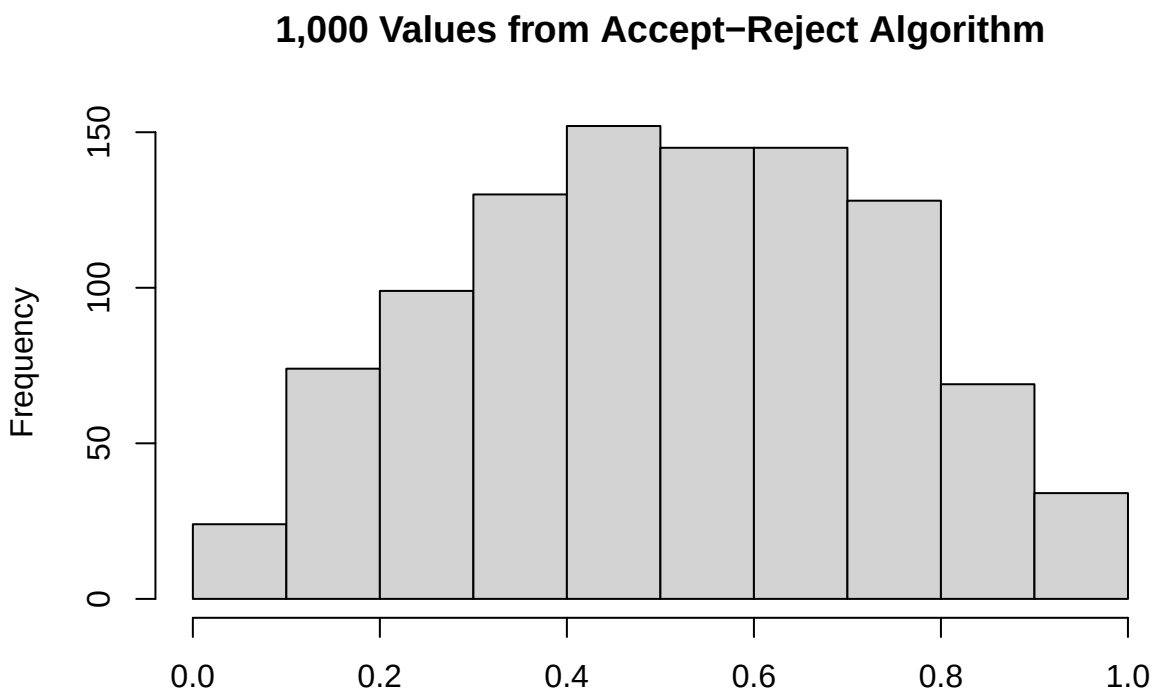
```
## [1] 1.5
```

```r
    #Let g(x) = 1
      g <- function(x) {
        return(rep(1,length(x)))
      }
    #Let c = 1.5
      c <- 1.5
    #Is f(x) <= c*g(x) for all x >= 0?
      plot(fX(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
      points(c * g(grid) ~ grid, type = 'l', col = 'red')
```




```r
#Algorithm
  accept_reject_beta_2_2 <- function(n) {
    fX <- function(x) {
      return(6*x - 6*x^2)
    }
    g <- function(x) {
      return(rep(1,length(x)))
    }
    c <- 1.5
    for (i in 1:n) {
      if (i == 1) {
        result <- list(NA, NA)
      }
    U <- runif(1)
    Z <- runif(1)
    n_failures <- c(0)
      while(U > fX(Z) / (c * g(Z))) {
        U <- runif(1)
        Z <- runif(1)
        n_failures <- n_failures + 1
      }
    result[[1]][i] <- Z
    result[[2]][i] <- n_failures
    }
    return(result)
  }
```

```
    #Histogram (1,000 draws)
      hist(accept_reject_beta_2_2(1000)[[1]], ylab="Frequency", xlab=NA, main="1,000 Values from Accept
```




```
    #Average number of failed draws
      library(magrittr)
      accept_reject_beta_2_2(1000)[[2]] %>% mean()
```

```
## [1] 0.475
```

```
  #Beta(10,10)
    #What does Beta(10,10) equal?
      gamma(10)
```

```
## [1] 362880
```

```
      gamma(20)
```

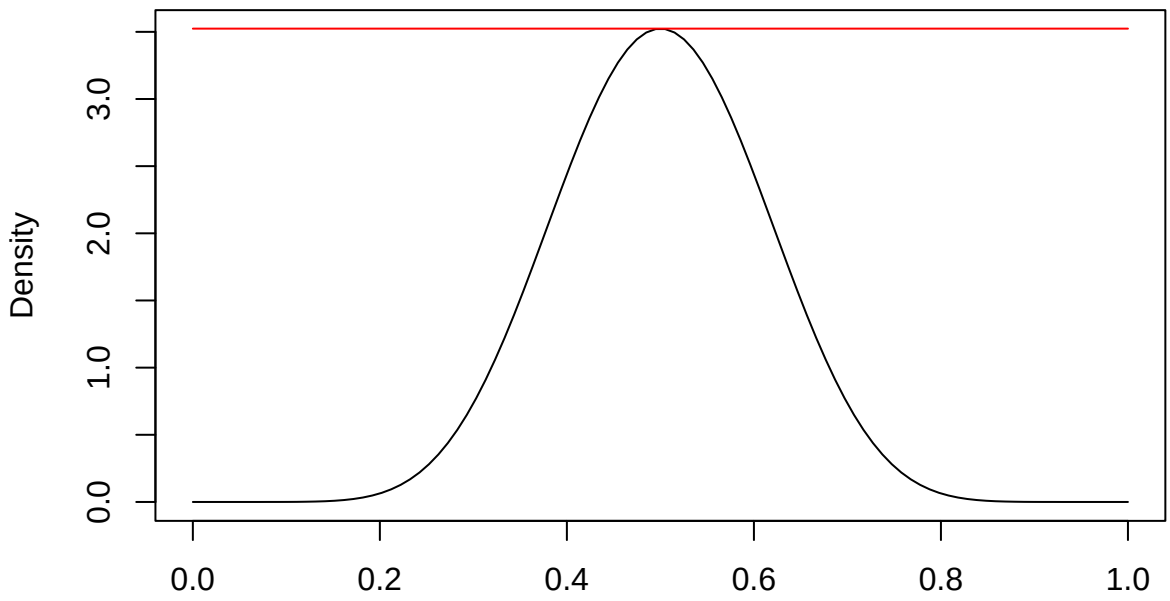
```
## [1] 1.216451e+17
```

```
      #Beta(10,10) = (x^9*(1-x)^9)/(gamma(10)*gamma(10)/gamma(20))
      fX <- function(x) {
        (x^9*(1-x)^9)/(gamma(10)*gamma(10)/gamma(20)) %>% return()
```

```
  }
#Plotting Beta(10,10)
  grid <- seq(0, 1, length.out=100)
  plot(fX(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
#Given that Beta(10,10) takes it's maximum at x=0.5, what is this maximum?
  fX(0.5)
```

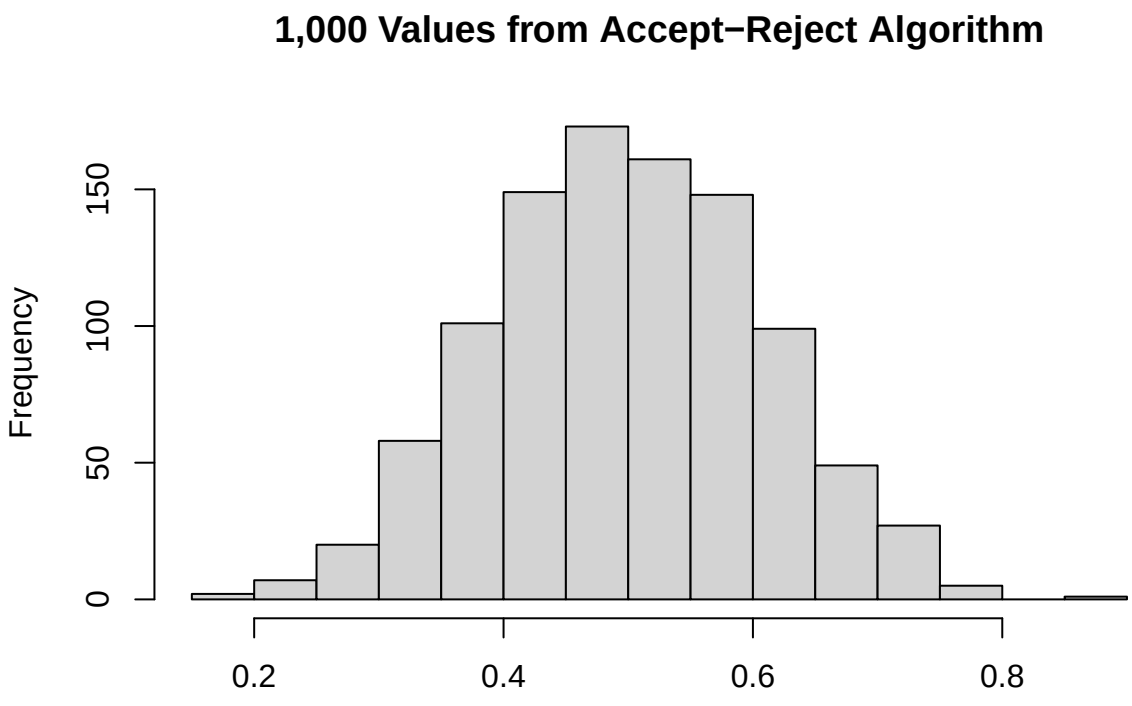
```
## [1] 3.523941
```

```
#Let g(x) = 1
  g <- function(x) {
    return(rep(1,length(x)))
  }
#Let c = 3.523941
  c <- 3.523941
#Is f(x) <= c*g(x) for all x >= 0?
  plot(fX(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
  points(c * g(grid) ~ grid, type = 'l', col = 'red')
```




```
#Algorithm
  accept_reject_beta_10_10 <- function(n) {
    fX <- function(x) {
    (x^9*(1-x)^9)/(gamma(10)*gamma(10)/gamma(20)) %>% return()
    }
```

```
    g <- function(x) {
    return(rep(1,length(x)))
    }
    c <- 3.523941
    for (i in 1:n) {
      if (i == 1) {
        result <- list(NA, NA)
      }
    U <- runif(1)
    Z <- runif(1)
    n_failures <- c(0)
      while(U > fX(Z) / (c * g(Z))) {
        U <- runif(1)
        Z <- runif(1)
        n_failures <- n_failures + 1
      }
    result[[1]][i] <- Z
    result[[2]][i] <- n_failures
    }
    return(result)
  }
#Histogram (1,000 draws)
  hist(accept_reject_beta_10_10(1000)[[1]], ylab="Frequency", xlab=NA, main="1,000 Values from Accep
```



```
    #Average number of failed draws
      accept_reject_beta_10_10(1000)[[2]] %>% mean()
```

```
## [1] 2.637
```

```
#Write a small description of what accounts for the difference in the number of accepted proposals betw
```

Answer: The average number of failed proposals before an accepted proposal in a) is 0.475 proposals. The average number of failed proposals before an accepted proposal in b) is 2.637 proposals. There are more failed proposals in part b because fX(Z) / (c * g(Z)) is more likely to be a a smaller value (which U will be more likely to be greater than) in part b) that in part a).

```
#2

  #a)
    #Monte Carlo estimate:
    monte_carlo <- function(n=1) {
      #f(x) = cos((pi*x)/2)
      f <- function(x) {
        cos((pi*x)/2) %>% return()
      }
      for (i in 1:n) {
        if (i == 1) {
          result <- c()
        }
        #g(x) = U ~ [0,1]
        X <- replicate(1000, runif(1))
        result[i] <- sum(f(X)) / length(X)
      }
      return(result)
    }
    monte_carlo()
```

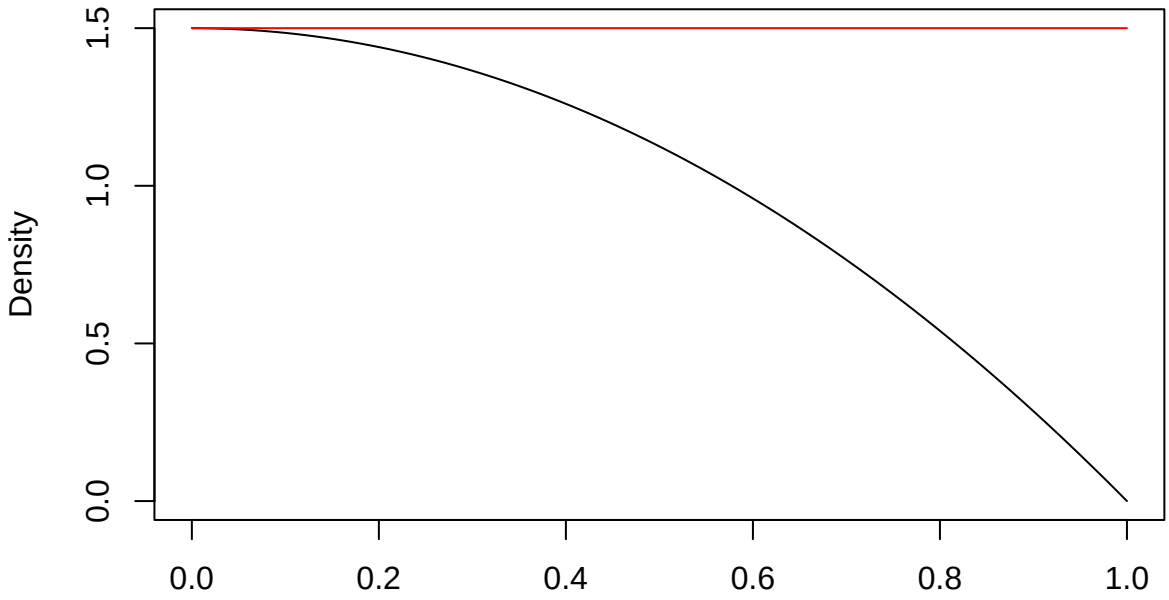
```
## [1] 0.6391589
```

```
  #b)
    #h(x) = 3(1 - x^2)/2
    #3(1 - x^2)/2 is not a standard pdf, so we need to implement an accept-reject algorithm to generate

    #Accept-reject algorithm
      #Let f(x) = h(x)
      fx <- function(x) {
        3*(1 - x^2)/2 %>% return()
      }
      #Plotting f(x)
      grid <- seq(0, 1, length.out=100)
      plot(fx(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
      #f(x) takes it's maximum at 0, what is this maximum?
        fx(0)
```

```
## [1] 1.5
```

```r
#Let g(x) = 1
  g <- function(x) {
    return(rep(1,length(x)))
  }
#Let c = 1.5
  c <- 1.5
#Is h(x) <= c*g(x) for all x >= 0?
  plot(fx(grid) ~ grid, type = 'l', ylab="Density", xlab=NA)
  points(c * g(grid) ~ grid, type = 'l', col = 'red')
```




```r
#Algorithm
  accept_reject_hx <- function(n) {
    fx <- function(x) {
    3*(1 - x^2)/2 %>% return()
    }
    g <- function(x) {
    return(rep(1,length(x)))
    }
    c <- 1.5
    for (i in 1:n) {
      if (i == 1) {
        pseudo_random_number <- c()
      }
    U <- runif(1)
    Z <- runif(1)
```

```
          while(U > fx(Z) / (c * g(Z))) {
            U <- runif(1)
            Z <- runif(1)
          }
          pseudo_random_number[i] <- Z
          }
          return(pseudo_random_number)
        }

      #Importance Sampling
      importance_sampling <- function(n=1) {
        h <- function(x) {
        3*(1 - x^2)/2 %>% return()
        }
        f <- function(x) {
          cos((pi*x)/2) %>% return()
        }
        #g = dunif(x)
        for (i in 1:n) {
          if (i == 1) {
            result <- c()
          }
          Y <- accept_reject_hx(1000)
          result[i] <- sum(f(Y)*dunif(Y)/h(Y)) / length(Y)
          }
        return(result)
        }
      importance_sampling()
```

```
## [1] 0.636512
```

```
  #c)
  cat("The variance of the Monte Carlo estimate after 1,000 draws is", monte_carlo(1000) %>% var(), fil
```

```
## The variance of the Monte Carlo estimate after 1,000 draws is 8.820884e-05
```

```
  cat("The variance of the importance sampling estimate after 1,000 draws is", importance_sampling(1000
```

```
## The variance of the importance sampling estimate after 1,000 draws is
## 9.806054e-07
```

Answer: The variance of the importance sampling estimates is smaller than the variance of the Monte Carlo estimates. This is because if we choose h(x) = |f(x)|g(x) / integral(|f(z)|g(z)dz), then an application of Cauchy Schwarz tells us that the condition for the importance sampling estimator to have a smaller variance than the naive estimator is satisfied. It should be noted that the variance of the naive Monte Carlo estimator is already very small. This is because the variance is equal to sqrt(var(f(X))/n) and n in this case is 1,000.