Assignment 1 Submission

Gao Haochun A0194525Y Ge Siqi Wang Pei A0194486M
Wei Yifei A0203451W

Group 37: Gao Haochun

Q1

```r
library(Matrix) # Matrix operations
library(MASS) # For Moore-Penrose pseudo-inverse ginv()
rankMatrix(matrix(c(1,1,2,2),nrow=2))
```

```
## [1] 1
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 4.440892e-16
```

```r
rankMatrix(matrix(c(1,1,2,2,1,2),nrow=3))
```

```
## [1] 2
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 6.661338e-16
```

```r
rankMatrix(matrix(c(1,1,2,0),nrow=2))
```

```
## [1] 2
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 4.440892e-16
```

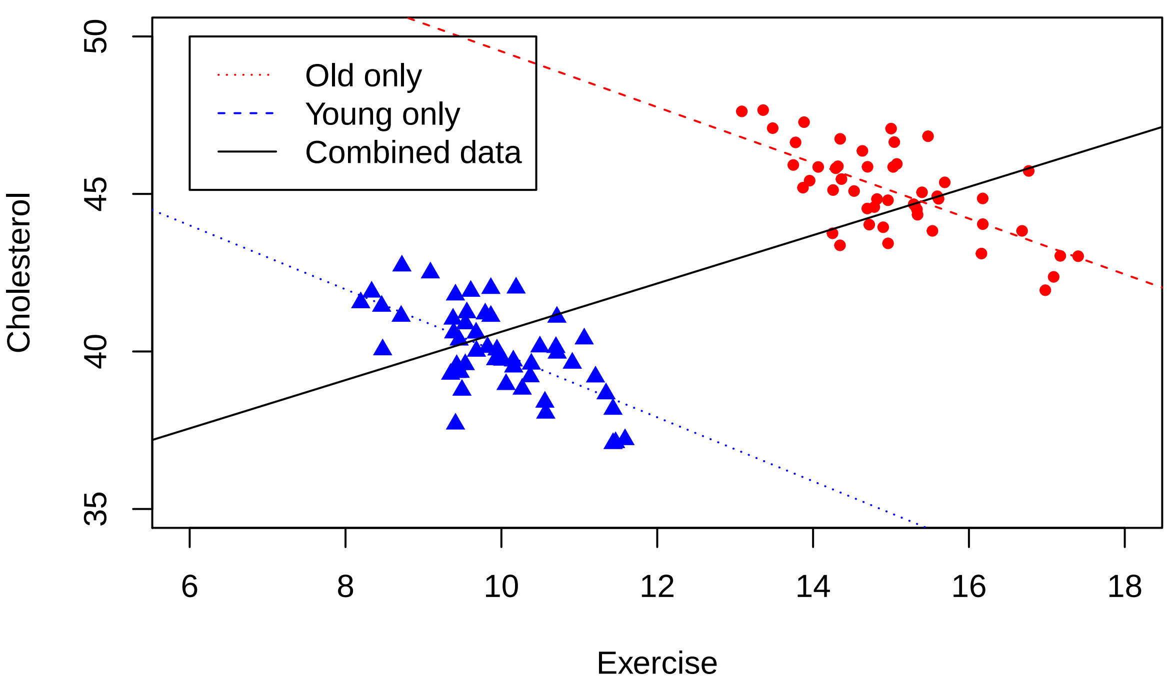
```r
rankMatrix(matrix(c(1,2,3,2,0,2),nrow=3))
```

```
## [1] 2
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 6.661338e-16
```

Q2

I. Reproducing the plot

```r
df = read.csv("exercise_and_cholesterol.csv")
old_df = df[df$Age=="Old",]
young_df = df[df$Age=="Young",]

attach(df)
plot(Cholesterol ~ Exercise,col=ifelse(Age=="Old","red","blue"),
     pch=ifelse(Age=="Old",20,17),ylim=c(35,50),xlim=c(6,18))

all_model = lm(Cholesterol ~ Exercise, data=df)
young_model = lm(Cholesterol ~ Exercise, data=young_df)
old_model = lm(Cholesterol ~ Exercise, data=old_df)
abline(all_model$coefficients[1],all_model$coefficients[2], lty="solid")
abline(young_model$coefficients[1],young_model$coefficients[2], lty="dotted",col="blue")
abline(old_model$coefficients[1],old_model$coefficients[2], lty="dashed",col="red")
legend(6, 50, legend=c("Old only", "Young only","Combined data"),
       col=c("red", "blue","black"), lty=c("dotted","dashed","solid"), cex=1)
```



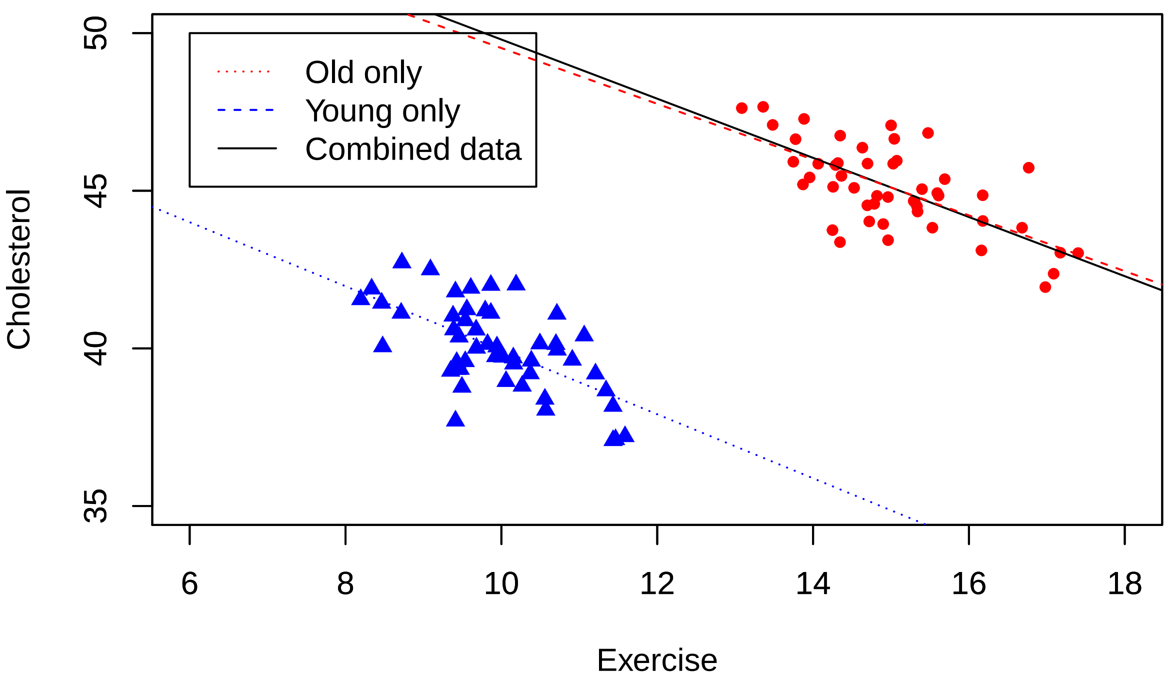
II. Are the three lines giving consistent or conflicting insights? Explain why.

The 3 lines are giving conflicting insights. The young and old only lines suggest that Cholesterol and Exercise has negative association while the combined data suggesting positive association. The inconsistency occurs because Cholesterol is positively associated with Age so that the two groups are different. Furthermore, the old only group's exercise data is in the range between 13 - 18 while the young only group's data ranging from 7 - 12.

III. Can you propose an alternative estimation using all the data in order to obtain the same insight as the two separate regressions using only the old or the young people?

[Replace this with answer here]

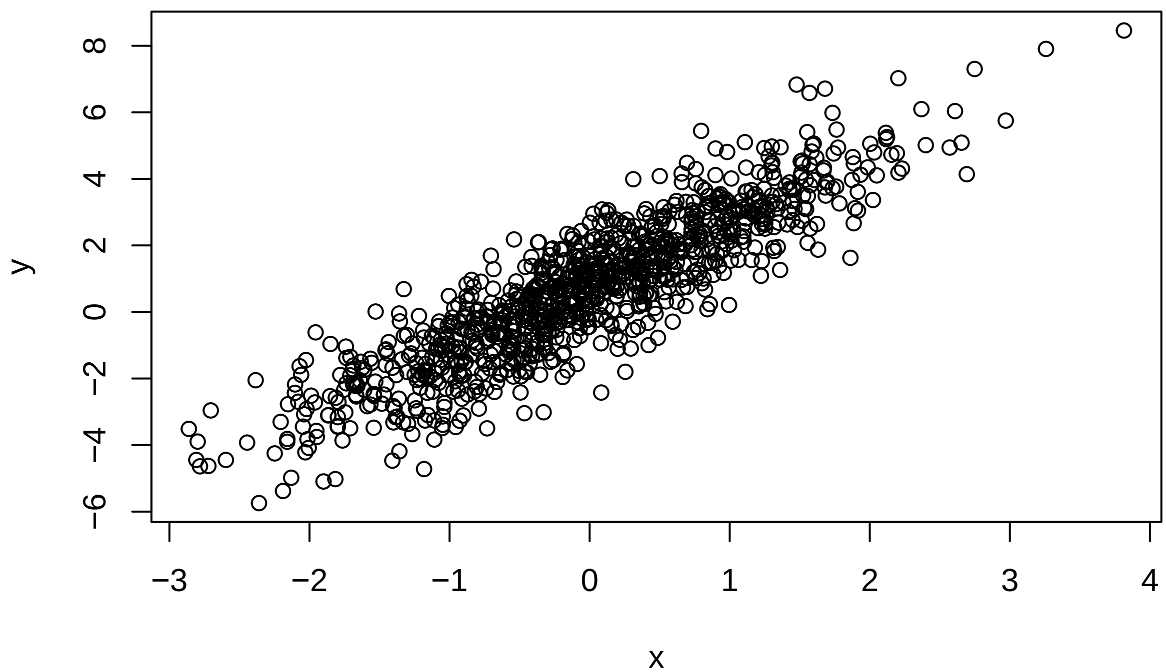
```
plot(young_df$Cholesterol ~ young_df$Exercise,col="blue",pch=17,ylim=c(35,50),xlim=c(6,18),xlab="",ylab
par(new=TRUE)
plot(old_df$Cholesterol ~ old_df$Exercise,col="red",pch=20,ylim=c(35,50),xlim=c(6,18),xlab="Exercise",yl

all_model = lm(Cholesterol ~ Exercise + Age, data=df)
young_model = lm(Cholesterol ~ Exercise, data=young_df)
old_model = lm(Cholesterol ~ Exercise, data=old_df)
abline(all_model$coefficients[1],all_model$coefficients[2], lty="solid")
abline(young_model$coefficients[1],young_model$coefficients[2], lty="dotted",col="blue")
abline(old_model$coefficients[1],old_model$coefficients[2], lty="dashed",col="red")
legend(6, 50, legend=c("Old only", "Young only","Combined data"),
       col=c("red", "blue","black"), lty=c("dotted","dashed","solid"), cex=1)
```



Q3

Data generation

```r
set.seed(37)
# Data generation
x = rnorm(1000) # Sample 1000 points from N(0, 1)
e = rnorm(1000)
y = 0.7 + 2*x + e
plot(y ~ x)
```



###

Use R's lm() function:

$$\hat{y} = 0.734 + 1.954x$$

```r
# Use lm()
model = lm(y ~ x)
summary(model)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.3179 -0.6318  0.0383  0.6764  3.2130
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.73407    0.03216   22.82   <2e-16 ***
## x            1.95394    0.03165   61.74   <2e-16 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.017 on 998 degrees of freedom
## Multiple R-squared:  0.7925, Adjusted R-squared:  0.7923
## F-statistic:  3812 on 1 and 998 DF,  p-value: < 2.2e-16
```

```
# y = 0.734 + 1.954 * x
```

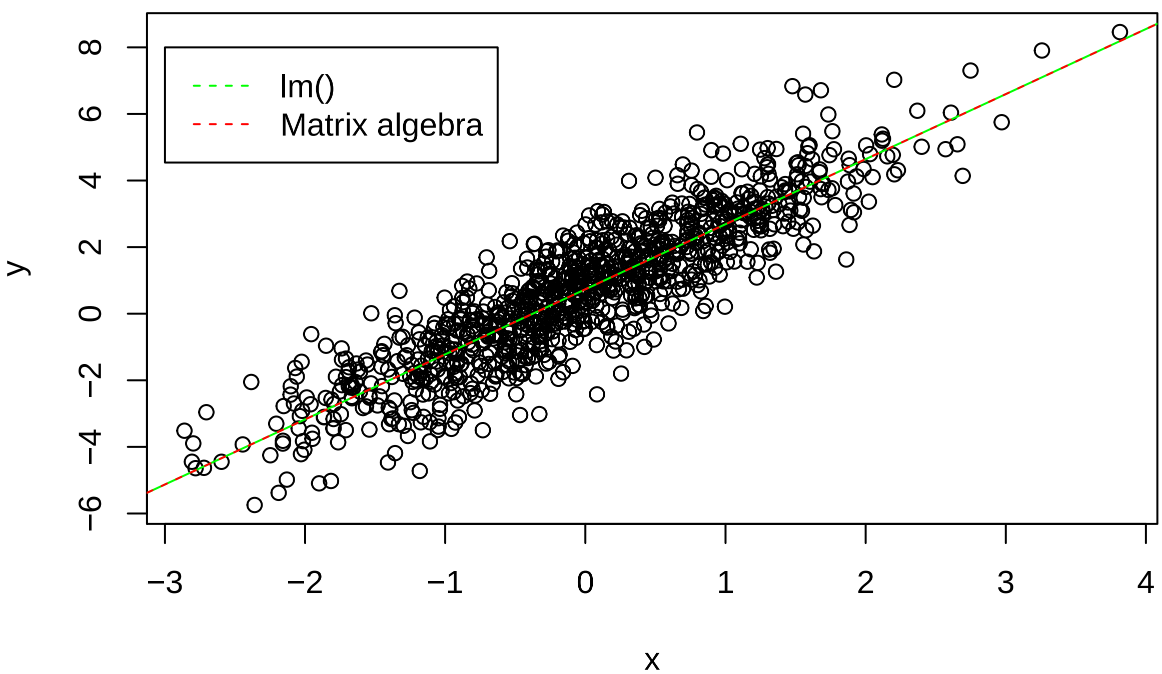
Use matrix algebra:

$$\hat{\beta} = (X^T X)^{-1} X^T Y \implies \hat{y} = 0.734 + 1.954x$$

```
# Use matrix algebra
X = cbind(rep(1,1000),x) # Add bias(constant for intercept) to data matrix
Y = y
beta_h = ginv(t(X)%*%X)%*%t(X)%*%Y
beta_h # \beta0 = 0.734, \beta1 = 1.954
```

```
##           [,1]
## [1,] 0.7340735
## [2,] 1.9539367
```

```
plot(y ~ x)
abline(model$coefficients[1],model$coefficients[2],col="green")
abline(beta_h[1],beta_h[2],col="red",lty="dashed")
legend(-3,8,legend=c("lm()", "Matrix algebra"),col=c("green","red"),lty=c("dashed","dashed"))
```



###

Use R's optim():

$$\hat{\beta} = argmin_{\beta} \sum_{i=1,\ldots,n} (y_i - \beta_0 - \beta_1 \times x_i)^2 \implies \hat{y} = 0.734 + 1.954x$$

```r
X = cbind(rep(1,1000), x)
Y = y
# Objective function
fun = function(beta) {
  sum((Y - X%*%beta)^2)
}
# Start optimization with random initialization
optim(runif(2),fun) # \beta0 = 0.734, \beta1 = 1.954
```

```
## $par
## [1] 0.734061 1.953818
##
## $value
## [1] 1031.934
##
## $counts
## function gradient
##       57       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

Use formula for 1 variable regression:

$$\hat{\beta}_1 = \frac{Cov(x_i, y_i)}{Var(x_i)} \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x} \implies \hat{y} = 0.734 + 1.954$$

```r
beta1 = cov(x,y)/var(x)
beta0 = mean(y) - beta1*mean(x)
beta0 # 0.734
```

```
## [1] 0.7340735
```

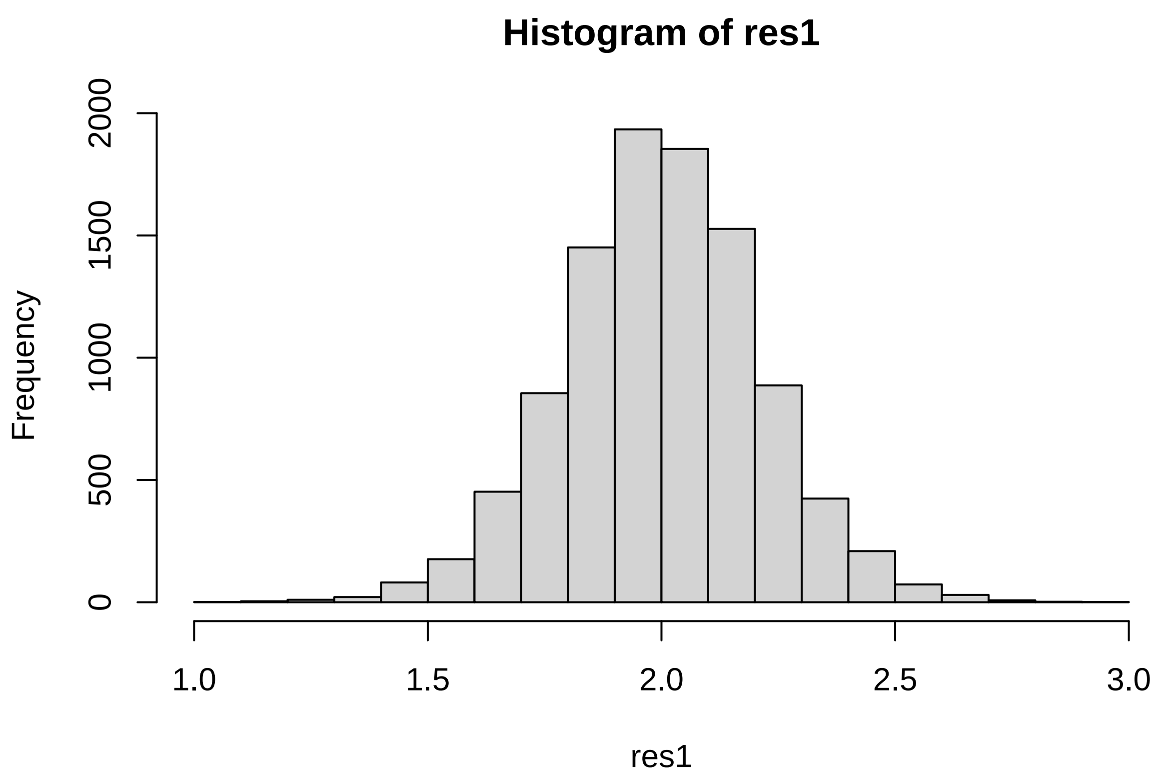
```r
beta1 # 1.954
```

```
## [1] 1.953937
```

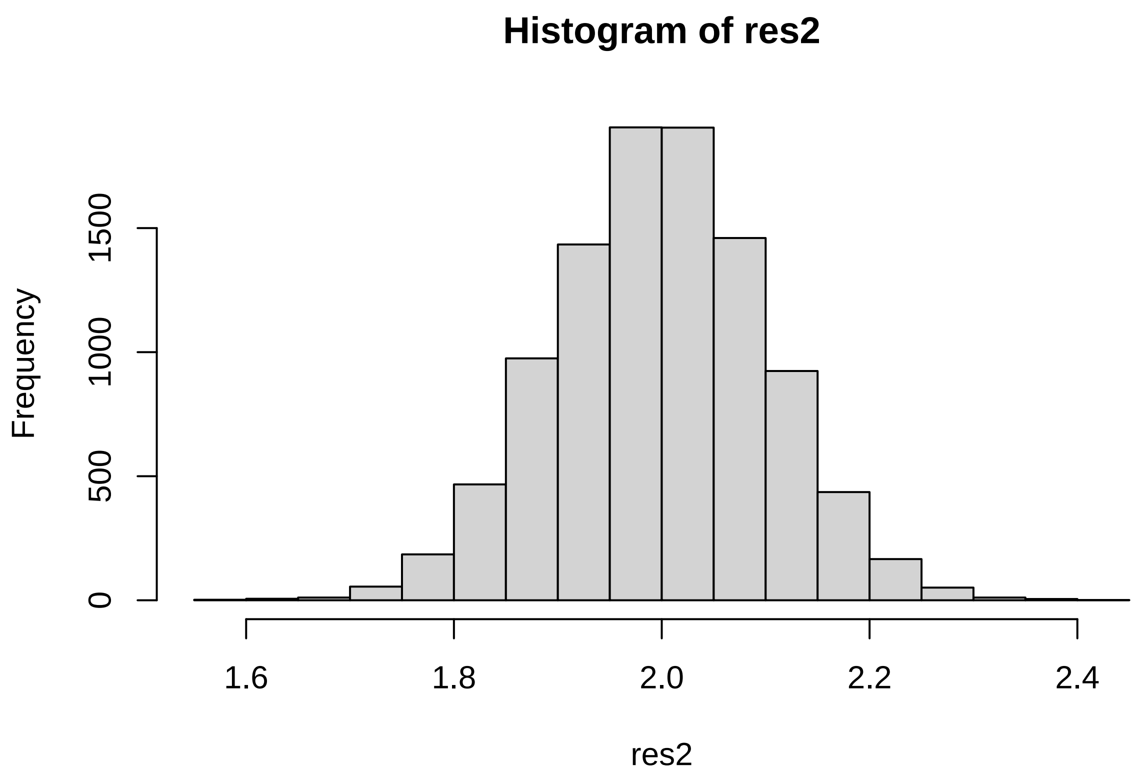
Check distribution of $\hat{\beta}_1$

```r
set.seed(37)
S = 10000
single_loop = function(N) {
  x = rnorm(N)
  e = rnorm(N)
  Y = 0.7 + 2*x + e
  X = cbind(rep(1,N),x)
  beta_h = ginv(t(X)%*%X)%*%t(X)%*%Y
  beta1 = beta_h[2]
```

```r
}
vec_loop = Vectorize(single_loop)


res1 = vec_loop(rep(25,S))
hist(res1)
```




```r
res2 = vec_loop(rep(100,S))
hist(res2)
```




```r
mean(res1) # 2.002271
```

```
## [1] 2.002271
```

```r
mean(res2) # 1.998513
```

```
## [1] 1.998513
```

```r
var(res1) # 0.04465469
```

```
## [1] 0.04465469
```

```r
var(res2) # 0.01046778
```

```
## [1] 0.01046778
```

```r
t.test(res1, res2, alternative = "two.sided", var.equal = FALSE) # p-value = 0.1095
```

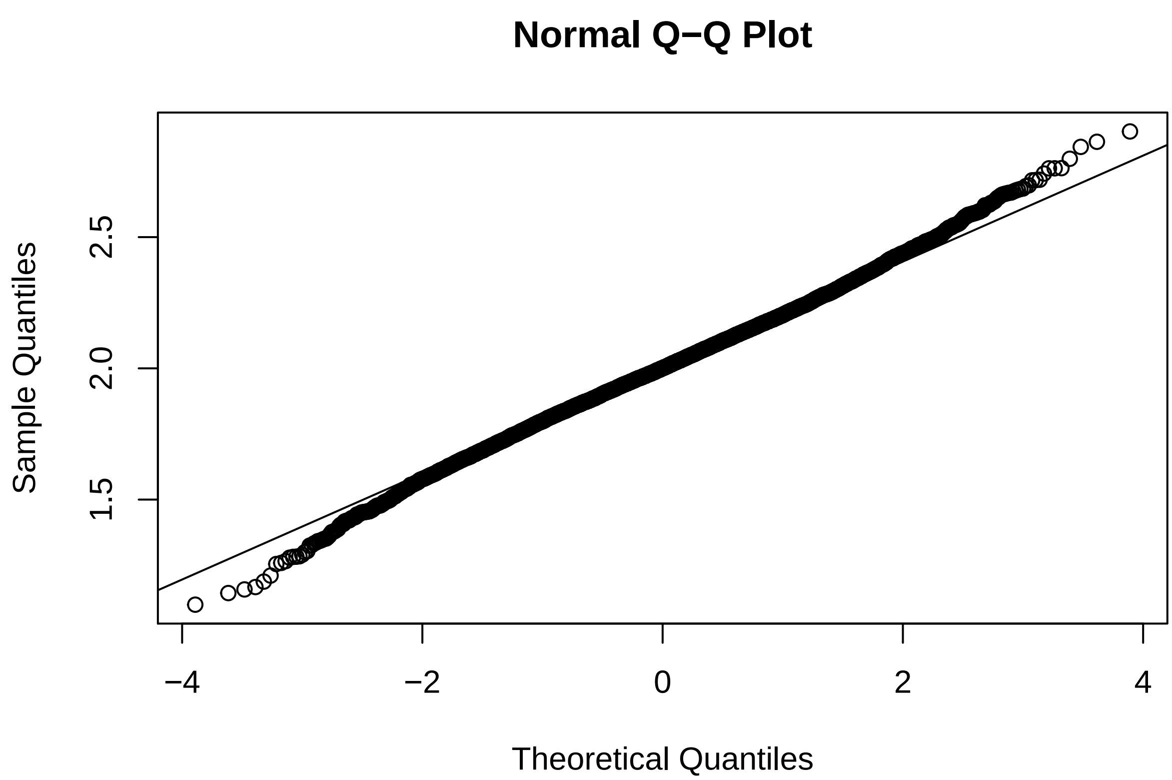
```
##
##  Welch Two Sample t-test
##
## data:  res1 and res2
## t = 1.6006, df = 14443, p-value = 0.1095
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.0008440232  0.0083600243
```

```
## sample estimates:
## mean of x mean of y
##  2.002271  1.998513
```

```r
# Test normality
library(nortest)
#Anderson-Darling normality test
ad.test(res1) # p-value = 2.064e-08
```

```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 3.3628, p-value = 2.064e-08
```

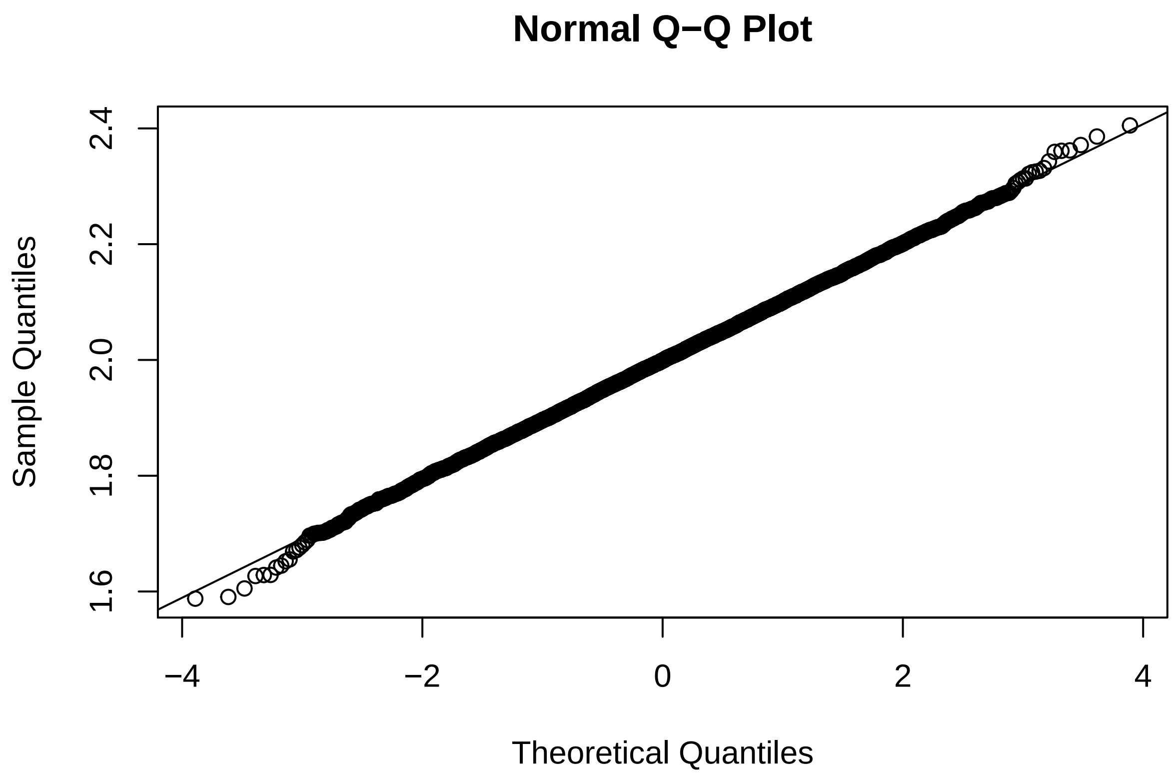
```r
qqnorm(res1)
qqline(res1)
```




```r
ad.test(res1) # p-value = 2.064e-08
```

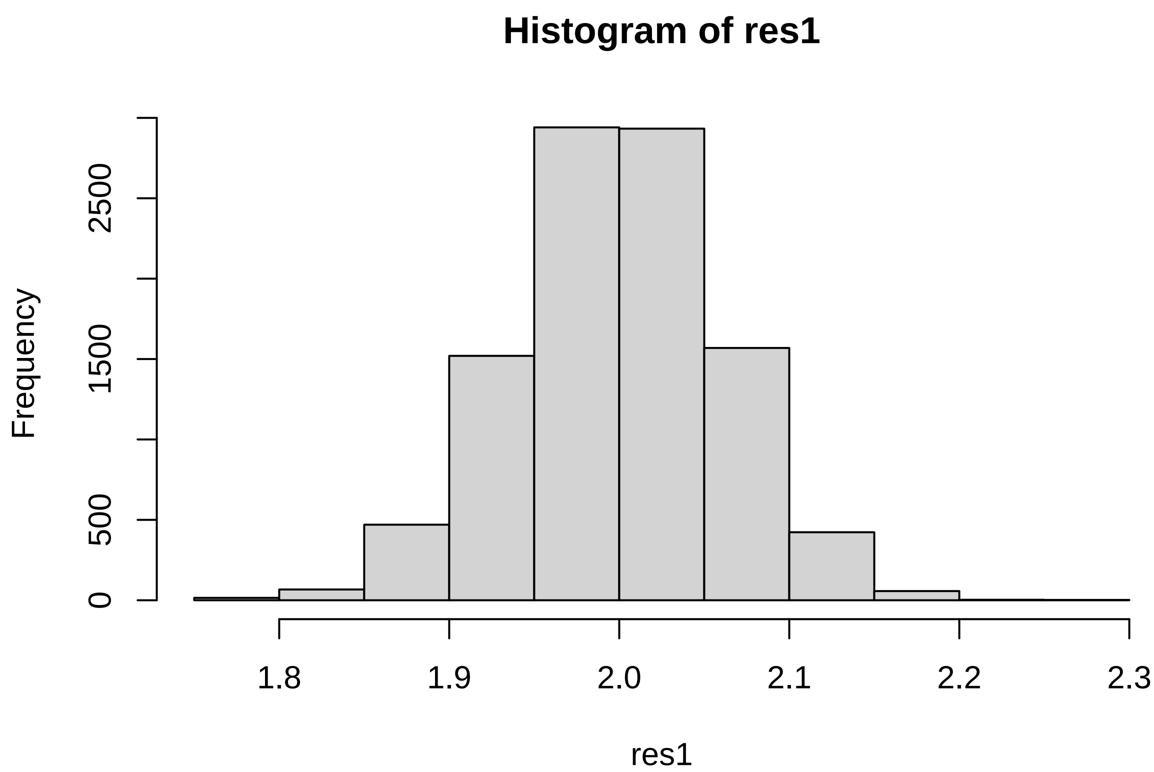
```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 3.3628, p-value = 2.064e-08
```

```r
qqnorm(res2)
qqline(res2)
```

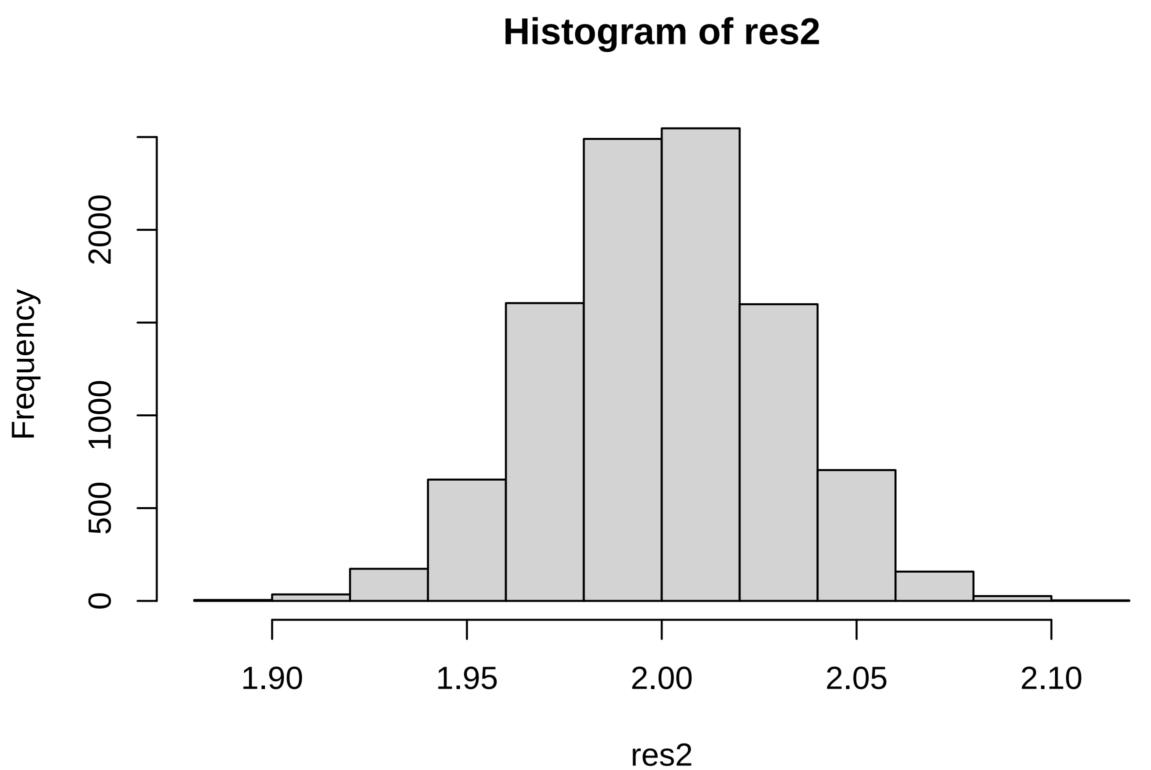


Use uniformly distributed error

```r
set.seed(37)
S = 10000
single_loop = function(N) {
  x = rnorm(N)
  e = runif(N)
  Y = 0.7 + 2*x + e
  X = cbind(rep(1,N),x)
  beta_h = ginv(t(X)%*%X)%*%t(X)%*%Y
  beta1 = beta_h[2]
}
vec_loop = Vectorize(single_loop)

res1 = vec_loop(rep(25,S))
hist(res1)
```




```r
res2 = vec_loop(rep(100,S))
hist(res2)
```



```r
mean(res1) # 1.999372
```

```
## [1] 1.999372
```

```r
mean(res2) # 2.000016
```

```
## [1] 2.000016
```

```r
var(res1) # 0.003772373
```

```
## [1] 0.003772373
```

```r
var(res2) # 0.0008630136
```

```
## [1] 0.0008630136
```

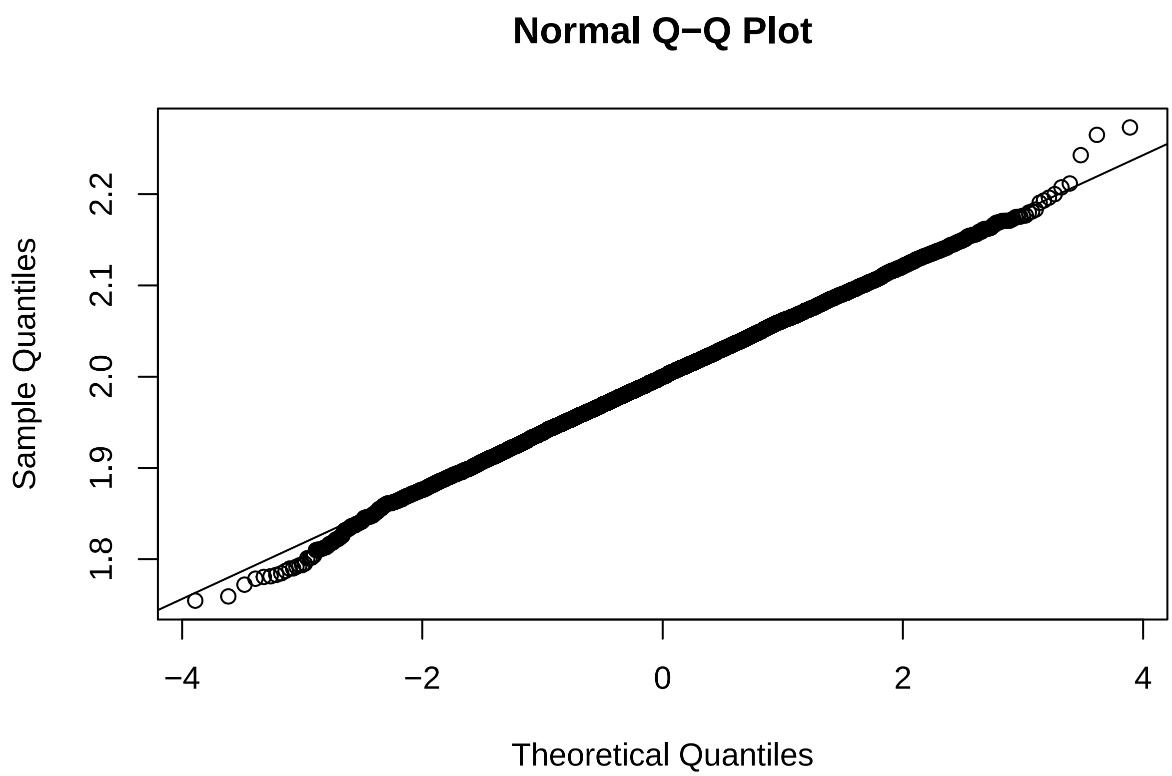
```r
t.test(res1, res2, alternative = "two.sided", var.equal = FALSE) # p-value = 0.3446
```

```
##
##  Welch Two Sample t-test
##
## data:  res1 and res2
## t = -0.94514, df = 14346, p-value = 0.3446
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.001978011  0.000691045
## sample estimates:
## mean of x mean of y
##  1.999372  2.000016
```

```r
#Anderson-Darling normality test
ad.test(res1) # p-value = 0.4207
```

```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 0.37211, p-value = 0.4207
```

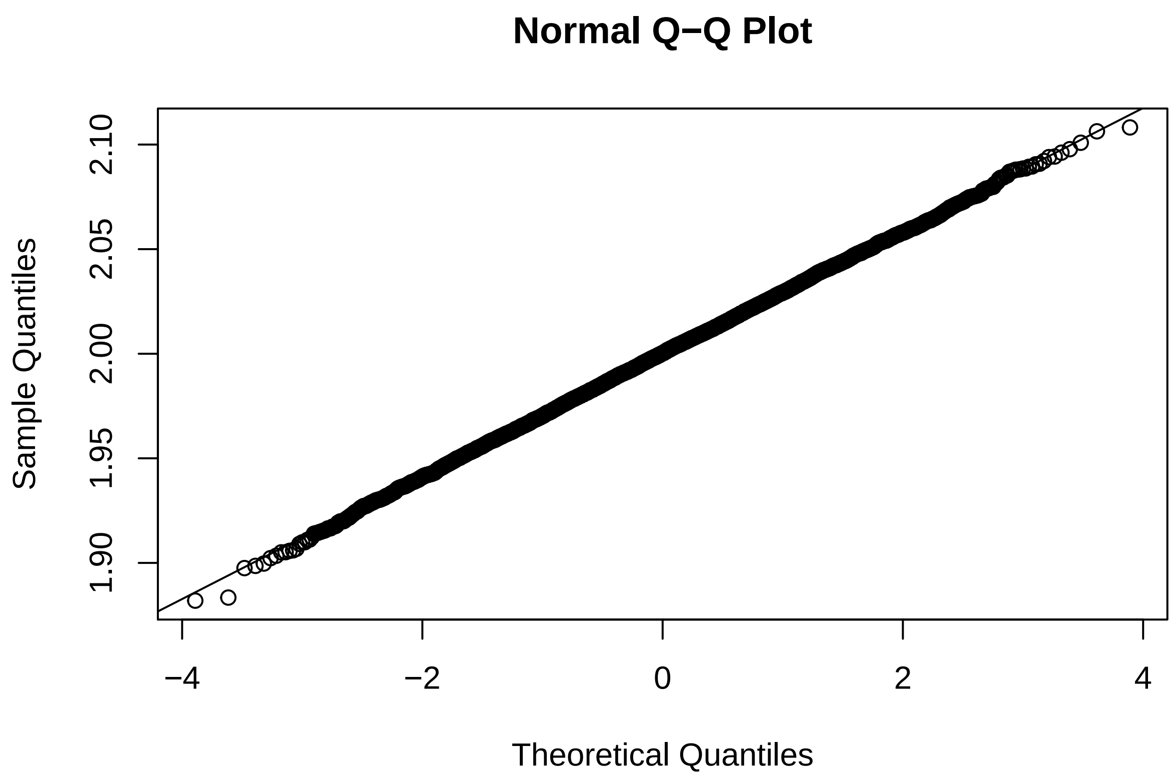
```r
qqnorm(res1)
qqline(res1)
```




```
ad.test(res1) # p-value = 0.4207
```

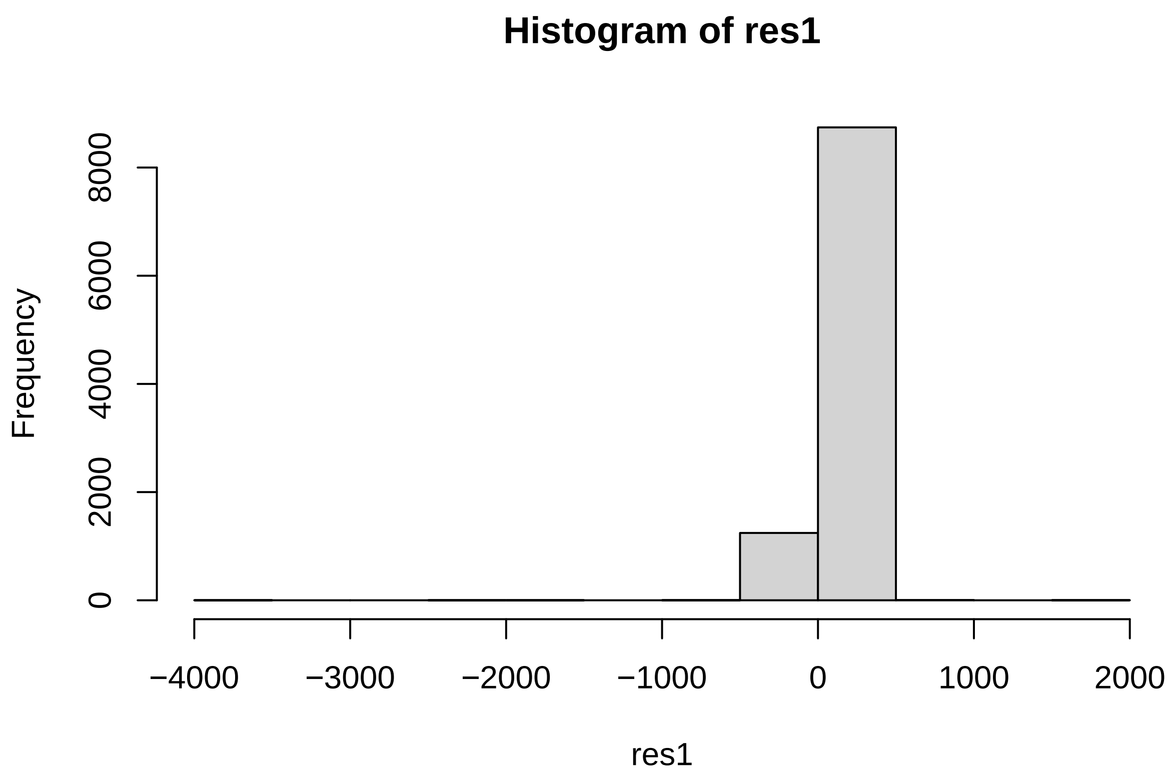
```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 0.37211, p-value = 0.4207
```

```
qqnorm(res2)
qqline(res2)
```

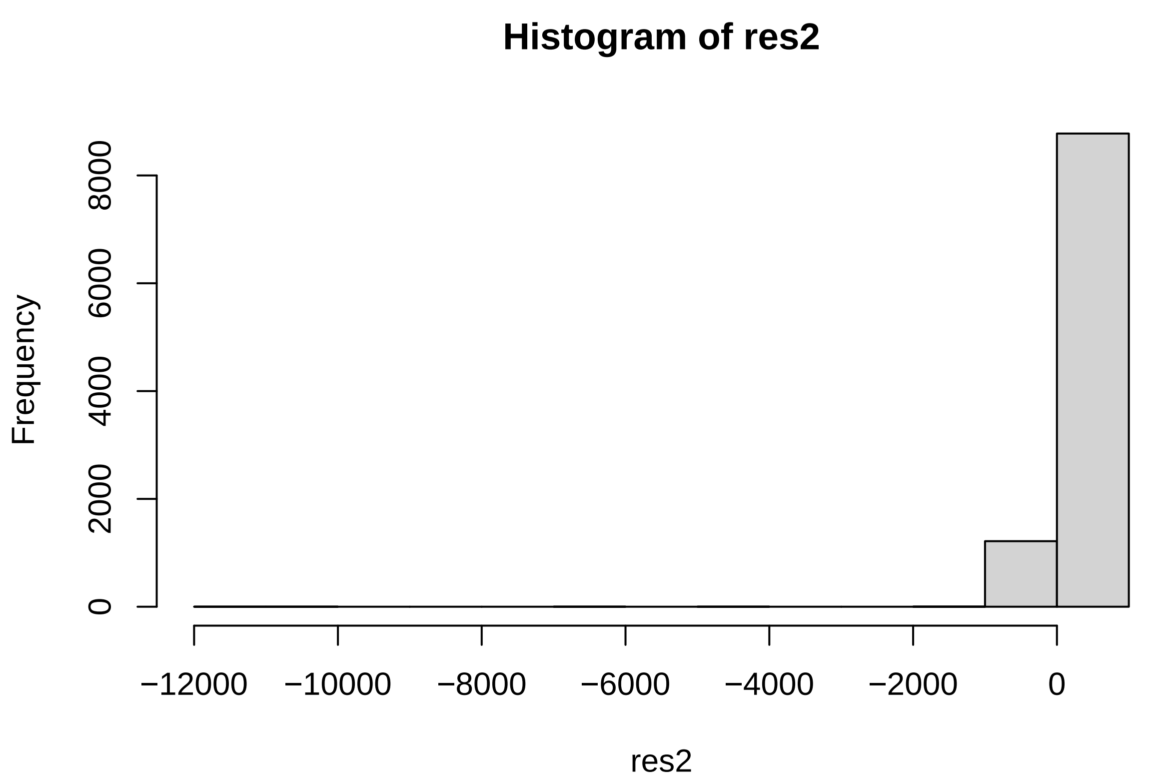


Use Cauchy distributed error

```
set.seed(37)
S = 10000
single_loop = function(N) {
  x = rnorm(N)
  e = rcauchy(N)
  Y = 0.7 + 2*x + e
  X = cbind(rep(1,N),x)
  beta_h = ginv(t(X)%*%X)%*%t(X)%*%Y
  beta1 = beta_h[2]
}
vec_loop = Vectorize(single_loop)

res1 = vec_loop(rep(25,S))
hist(res1)
```




```r
res2 = vec_loop(rep(100,S))
hist(res2)
```



```r
mean(res1) # 1.907382x
```

```
## [1] 1.907382
```

```r
mean(res2) # -1.599096
```

```
## [1] -1.599096
```

```r
var(res1) # 3427.213
```

```
## [1] 3427.213
```

```r
var(res2) # 29851.75
```

```
## [1] 29851.75
```

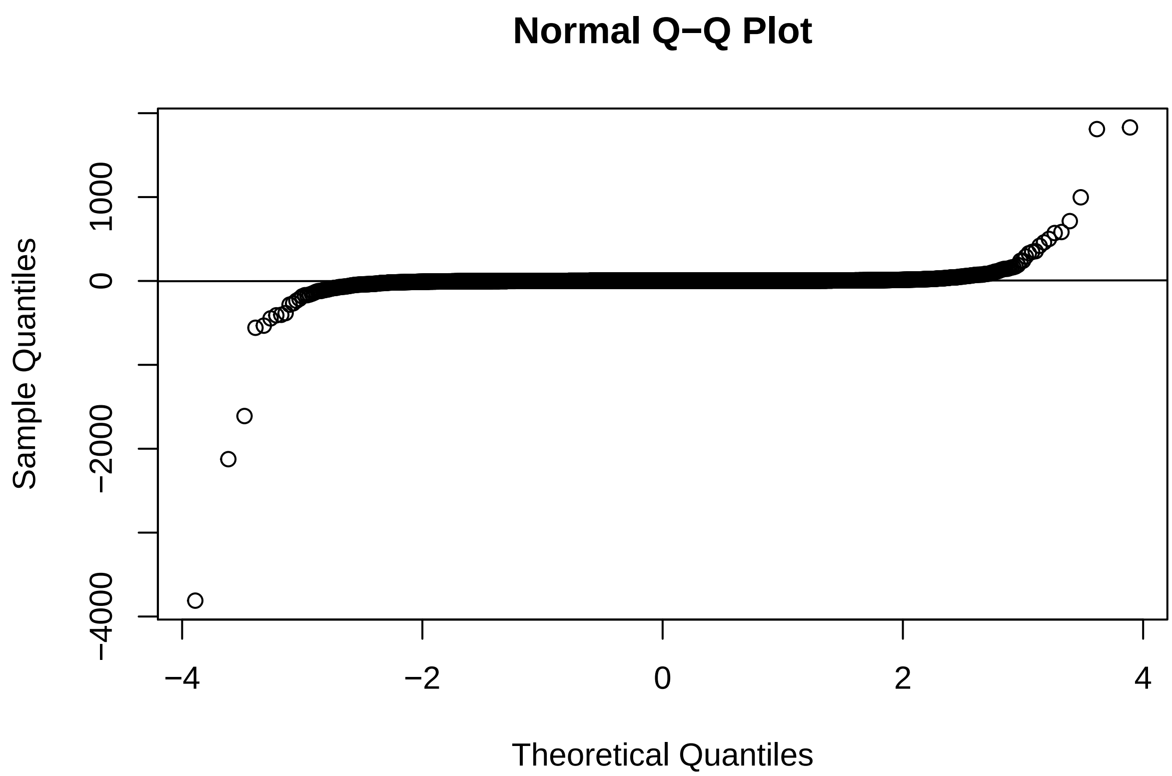
```r
t.test(res1, res2, alternative = "two.sided", var.equal = FALSE) # p-value = 0.05461
```

```
##
##  Welch Two Sample t-test
##
## data:  res1 and res2
## t = 1.9221, df = 12265, p-value = 0.05461
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.06934377  7.08229918
## sample estimates:
## mean of x mean of y
##  1.907382 -1.599096
```

```r
#Anderson-Darling normality test
ad.test(res1) # p-value = 2.2e-16
```

```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 3277, p-value < 2.2e-16
```

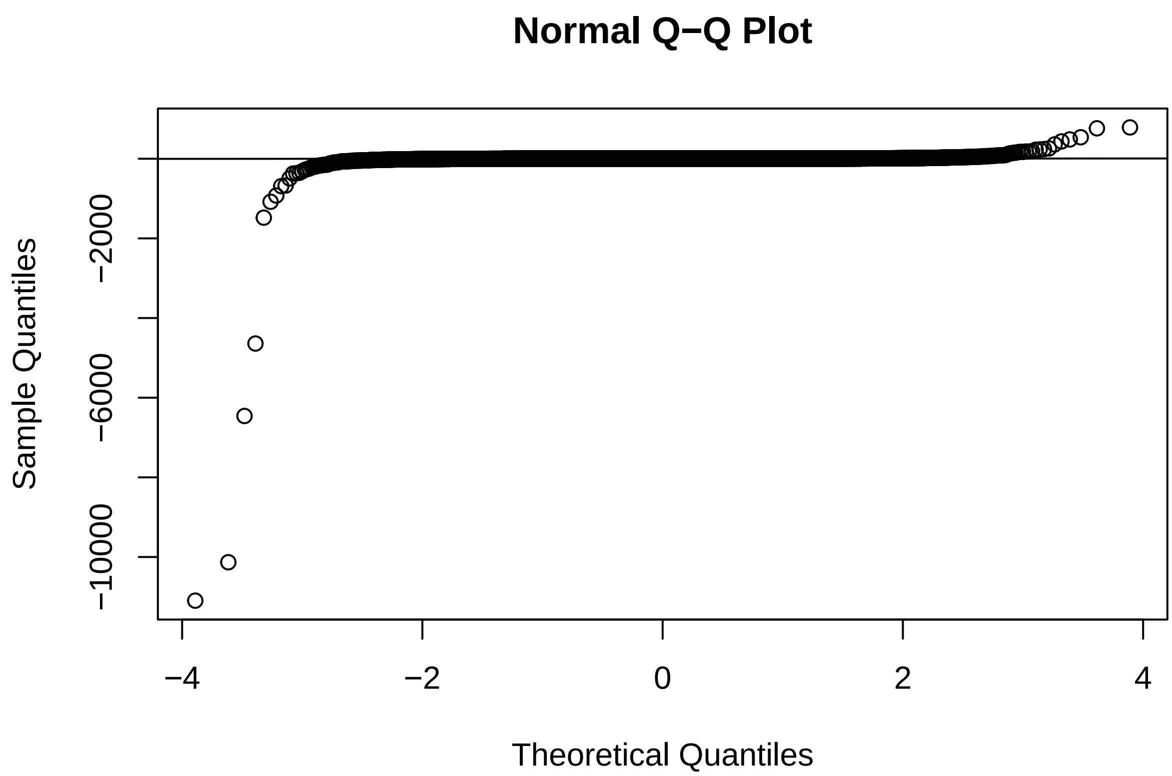
```r
qqnorm(res1)
qqline(res1)
```




```r
ad.test(res1) # p-value = 2.2e-16
```

```
##
##  Anderson-Darling normality test
##
## data:  res1
## A = 3277, p-value < 2.2e-16
```

```r
qqnorm(res2)
qqline(res2)
```



Q4

Recurrent formula of compound interest rate:

$$x_{n+1} = x_n(1+r) - P \implies x_n = x_0(1+r)^n + P\frac{1-(1+r)^n}{r}$$

Given:

$$x_0 = 10000, x_{60} = 0, P = 250 \implies r = 0.01439478$$

```
library(numDeriv)

obj_func = function(r, x0=10000,p=250, n=60) {
  # Iteration method to calculate xn
  x = x0
  for(i in 1:n) {
    x = x*(1+r) - p
  }
  res = x
}

analytical_func = function(r, x0=10000,p=250, n=60) {
  # Close form of xn
  xn = x0*(1+r)^n + p*(1-(1+r)^n)/r
}

bisection_method = function(f,left,right,tol=1e-10,n=1000) {
  if(sign(f(left))==sign(f(right))) {
```

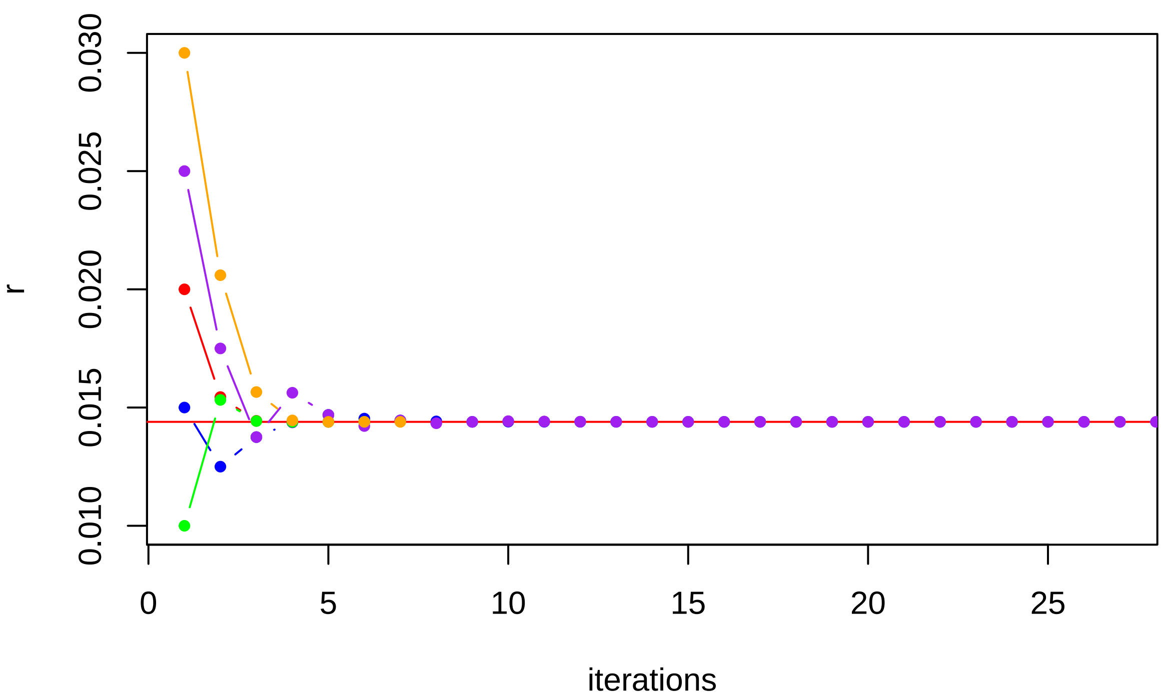
```
    print("Bad Initial Points!")
    stop()
  }

  history = right
  for (i in 1:n) {
    mid = (left+right)/2
    if (f(mid)==0) {
      history = c(history,mid)
      res = list('optim r' = mid, 'iterations' = i,'history'=history)
      return(res)
    } else if (sign(f(mid))==sign(f(right))) {
      right = mid
      history = c(history,right)
    } else {
      left = mid
      history = c(history,left)
    }

    if(abs(left-right) < tol) {
      res = list('optim r' = mid, 'iterations' = i,'history'=history)
      return(res)
    }
  }
  print("Maximum number of iteration reached")
  res = list('optim r' = mid, 'iterations' = i,'history'=history)
  return(res)
}

newton_method = function(f, r0, tol=1e-8,n=1000) {
  history = r0
  for(i in 1:n) {
    deriv = genD(func = f, x = r0)$D
    r1 = r0 - f(r0)/deriv[1]
    history = c(history,r1)
    if(abs(r1-r0) < tol) {
      res = list('optim r' = r1, 'iterations' = i,'history'=history)
      return(res)
    }
    r0 = r1
  }
  print("Maximum number of iteration reached")
  res = list('optim r' = r1, 'iterations' = i,'history'=history)
  return(res)
}

sol1 = newton_method(obj_func, 0.01)
sol1$`optim r`
```

```
## [1] 0.01439478
```

```r
sol2 = newton_method(analytical_func, 0.02)
sol3 = newton_method(analytical_func, 0.03)
sol4 = bisection_method(obj_func, 0.01, 0.025)
sol5 = bisection_method(analytical_func, 0.01, 0.015)

mat = cbind(sol1$history,sol2$history,sol3$history)
```

```
## Warning in cbind(sol1$history, sol2$history, sol3$history): number of rows of
## result is not a multiple of vector length (arg 1)
```

```r
maxLen = nrow(mat)
plot(1:length(sol5$history),sol5$history,type="b",pch=20,col="blue",ylim=c(0.01,0.03),xlab="iterations"
abline(sol1$`optim r`,0,col="red")
lines(1:length(sol2$history),sol2$history,type="b",pch=20,col="red")
lines(1:length(sol1$history),sol1$history,type="b",pch=20,col="green")
lines(1:length(sol4$history),sol4$history,type="b",pch=20,col="purple")
lines(1:length(sol3$history),sol3$history,type="b",pch=20,col="orange")
```



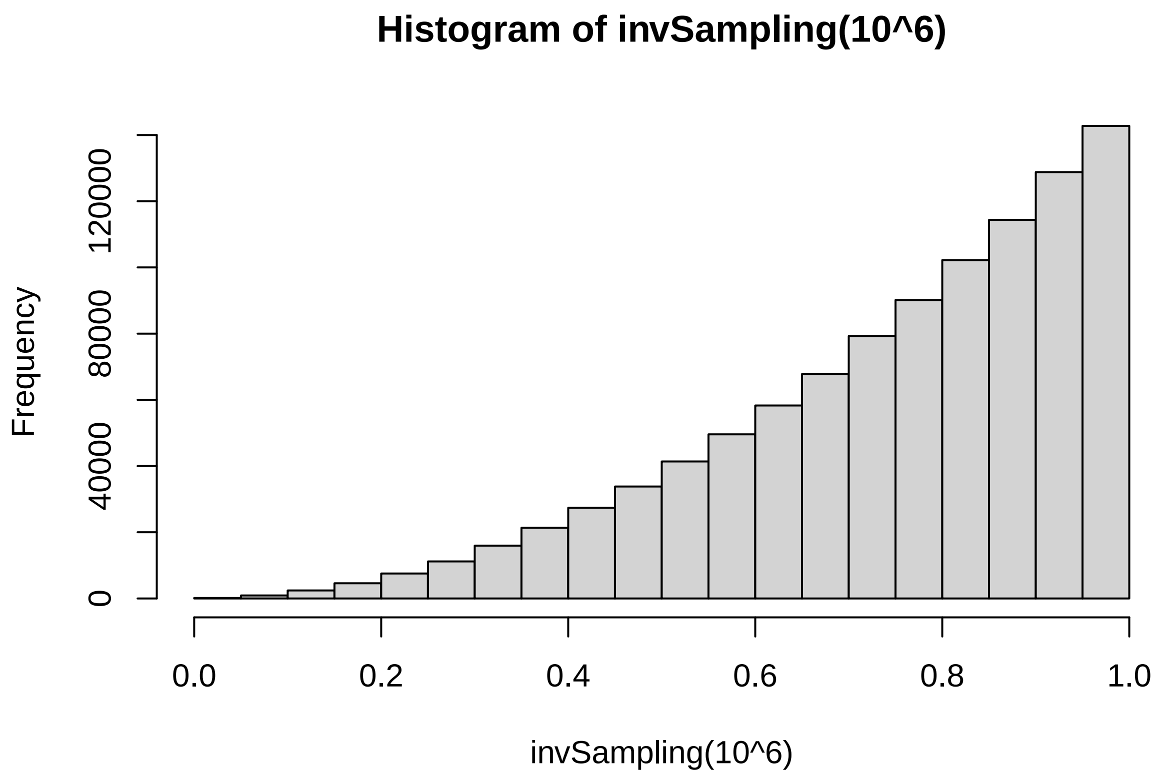
Q5

Find the inverse function of X

$$f(x) = 3x^2 \cdot I(0,1) \implies F(x) = \int_0^x f(x)dx = x^3, x \in [0,1] \implies x = F^{-1}(u) = u^{1/3}$$

```r
set.seed(37)

invSampling = function(N) {
  # Returns a vector of N elements sampled by inversion.
```

```
  return(runif(N)^(1/3))
}

hist(invSampling(10^6))
```



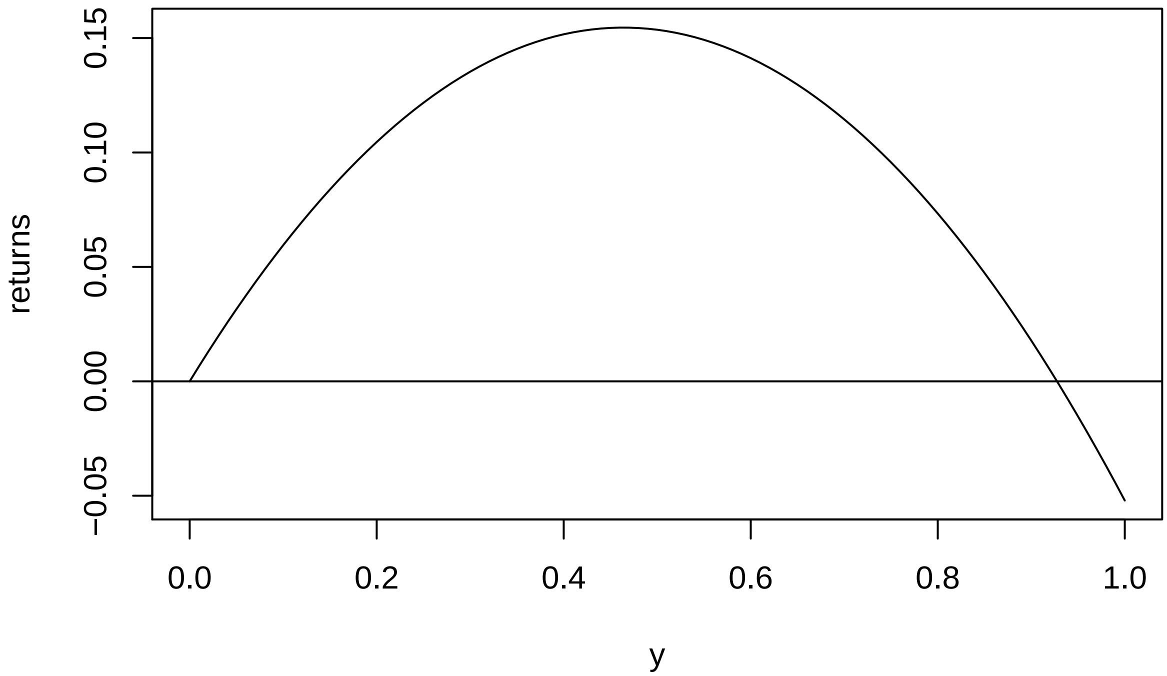
Expected return given y:

$$E[p(x,y)|y] = \int_0^1 p(x,y)f(x)dx$$

```
set.seed(37)

revenue = function(x,y) {
  return(y*(log(x)+1)-y^2*sqrt(1-x^4))
}

X = invSampling(1000000)
expectedReturn = function(y) {
  # Compute expected return given y
  N = 1000000
  #X = invSampling(N)
  Y = rep(y,N)

  # Take mean of 10^6 sampled return to approximate expectation
  res = sum(mapply(revenue, X, Y))/N
}
```

```
# take y from 0 to 1 with step 0.01
y = seq(0,1,0.01)
returns = sapply(y,expectedReturn)
```

```r
plot(returns ~ y, type="l")
abline(0,0)
```



Use R's optim() to minimize objective function:

```r
optim(0, function(y){-1*expectedReturn(y)})
```

```
## Warning in optim(0, function(y) {: one-dimensional optimization by Nelder-Mead is unreliable:
## use "Brent" or optimize() directly
```

```
## $par
## [1] 0.4637619
##
## $value
## [1] -0.1545742
##
## $counts
## function gradient
##       54       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

```r
#0.4639308
```