NAME: Geonho Marco ID NUMBER: 122200
SURNAME: You

ECONOMETRICS AND DATA ANALYSIS

Home Assessment N.2 - Solution

Deadline: 9 October 2020, 11h59pm [Paris time]

Problem 1 (OLS - different flavours)

Consider the following population regression model

$$Y_i = \beta_0 + \beta_1 X_i + u_i$$

where $\beta_0 = \sqrt{2}$ and $\beta_1 = \sqrt{3}$.

Assume that X and u are distributed as Studentâs T random variables with N-1 degrees of freedom. Let the sample size be N=100 and draw a random sample from this population.

1. Write the code to compute the OLS estimators using the scalar, matrix and numeric representations discussed in class. For the numeric case use the âoptimâ R command and explicitly set the relative tolerance (using the âreltolâ option) equal to 1e-06.
2. For each of them write the corresponding R function to compute estimates.
3. Write a simple Monte Carlo simulation to study the sampling properties of the $\hat{\beta}_1^{OLS}$ in the three cases.
4. What happens when the relative tolerance of the command âoptimâ is set to 1? Is anything wrong with the OLS estimator?

The first three questions ask us to calculate first, the estimated intercept coefficient ($\hat{\beta}_0^{OLS}$) and slope coefficient ($\hat{\beta}_1^{OLS}$) using three different methods of OLS-estimation such as scalar form computation, matrix form computation, and numerical optimization. Then we have to exploit sampling properties of each $\hat{\beta}_1^{OLS}$ through Monte Carlo simulation. We are going to answer to those three questions at once since they are all related questions.

Solution and Interpretation

We first set common settings for further computations for each method. We set a sample size (which could be changed later on if we wish to), degrees of freedom, and we draw pseudo-random sample x and u following the T-distribution. Then we generate a linear model y from drawn samples and its intercept and slope coefficients as provided in the exercise.

```r
set.seed(1995) # for replicability
##====================Common Settings====================
# Sample size and Degrees of freedom
N <- 100
dof <- N - 1
# Sample drawing
x <- rt(N, dof)
u <- rt(N, dof)
# Model Generation
true.b0 <- sqrt(2)
true.b1 <- sqrt(3)
y <- true.b0 + true.b1*x + u
# Graphical environment construction
par(mfrow = c(3,2))
##========================================================
```

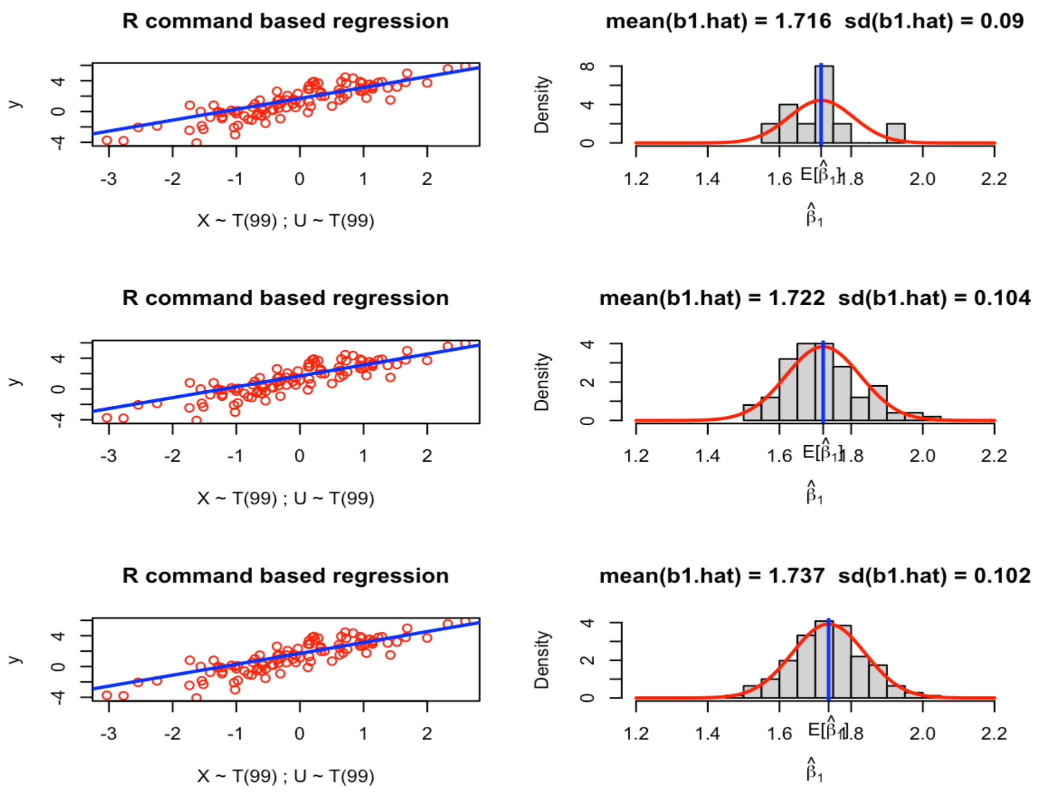
Once basic common settings are done, we can proceed to build functions to compute OLS-estimation with different methods. First, we build a function that automates regression using lm built-in function as a standard way to compute OLS estimators in R. (All the steps are explained in the comments below)

```r
Computational_Regression <- function(size, rep, intercept, slope){
  ## Regress y as response, x as predictor using lm R built-in function
  regress <- lm(formula = y ~ x)
  summary(regress)
  ## Receive OLS-estimated coefficients of the model
  b.hat.lm <- coef(regress)
  b0.hat.lm <<- as.numeric(b.hat.lm["(Intercept)"]) # make it global
  b1.hat.lm <<- as.numeric(b.hat.lm["x"])      # make it global
  ## Print regression results
  cat("==Computational Regression==",
      paste("estimated coefs :", b0.hat.lm, b1.hat.lm),
      paste("expected y :", mean(y)),
      paste("expected model :", b0.hat.lm + b1.hat.lm * mean(x)), sep = "\n")
  ## Scatter plot and fitted line with computed coefficients
  plot(x, y, col = "red", xlab = "X ~ T(99) ; U ~ T(99)", main = "R command based
      regression")
  abline(a = b1.hat.lm, b = b0.hat.lm, col = "blue", lwd = 2)
  ## Monte Carlo Simulation
  ## Set the number of replications and Initialize vectors
  R <- rep
  b0.hat <- rep(0,R)
  b1.hat <- rep(0,R)
  set.seed(1) # for replicability
  ## Draw a new sample each time in the loop and regress each time OLS estimators
  for (i in 1:R){
    X <- rt(size, (size - 1))
    U <- rt(size, (size - 1))
    Y <- true.b0 + true.b1*X + U
    b.hat <- coef(lm(Y ~ X))
    ## Store estimated intercept and slope each time in initialized vectors
    b0.hat[i] <- b.hat["(Intercept)"]
    b1.hat[i] <- b.hat["X"]
  }
  ## Print the mean of vector of estimated slopes (mean of elements of b1.hat vector)
  cat(paste("expected value of b1.hat :", mean(b1.hat)),
      paste("standard deviation of b1.hat :", sd(b1.hat)), sep = "\n")
  ## Plot histogram of all estimated slopes and indicate the position of their mean
  hist(b1.hat, freq=FALSE, xlim=c(1.2,2.2),
       main=(paste("mean(b1.hat) =",round(mean(b1.hat), digits = 3),
                   " sd(b1.hat) =", round(sd(b1.hat), digits = 3))),
       xlab=expression(hat(beta)[1]))
  abline(v = mean(b1.hat), col = "blue", lwd = 2)
  axis(1, at = mean(b1.hat), lab = expression(paste("E[", hat(beta)[1], "]")))
  ## Plot PDF of normal distribution with the mean and the standard deviation of
      estimated slopes
  xfit<-seq(1.2, 2.2, 0.01)
  lines(xfit,dnorm(xfit,mean(b1.hat),sd(b1.hat)),lwd=2,col="red")
}
## Same function with different number of replications and N fixed to 100
Computational_Regression(N, 10, true.b0, true.b1) #A
Computational_Regression(N, 100, true.b0, true.b1) #B
Computational_Regression(N, 1000, true.b0, true.b1) #C
## Same function with different sample sizes and rep fixed to 100
Computational_Regression(10, 100, true.b0, true.b1) #D
Computational_Regression(100, 100, true.b0, true.b1) #E
Computational_Regression(1000, 100, true.b0, true.b1) #F
```

The line B of the code prints the following results :

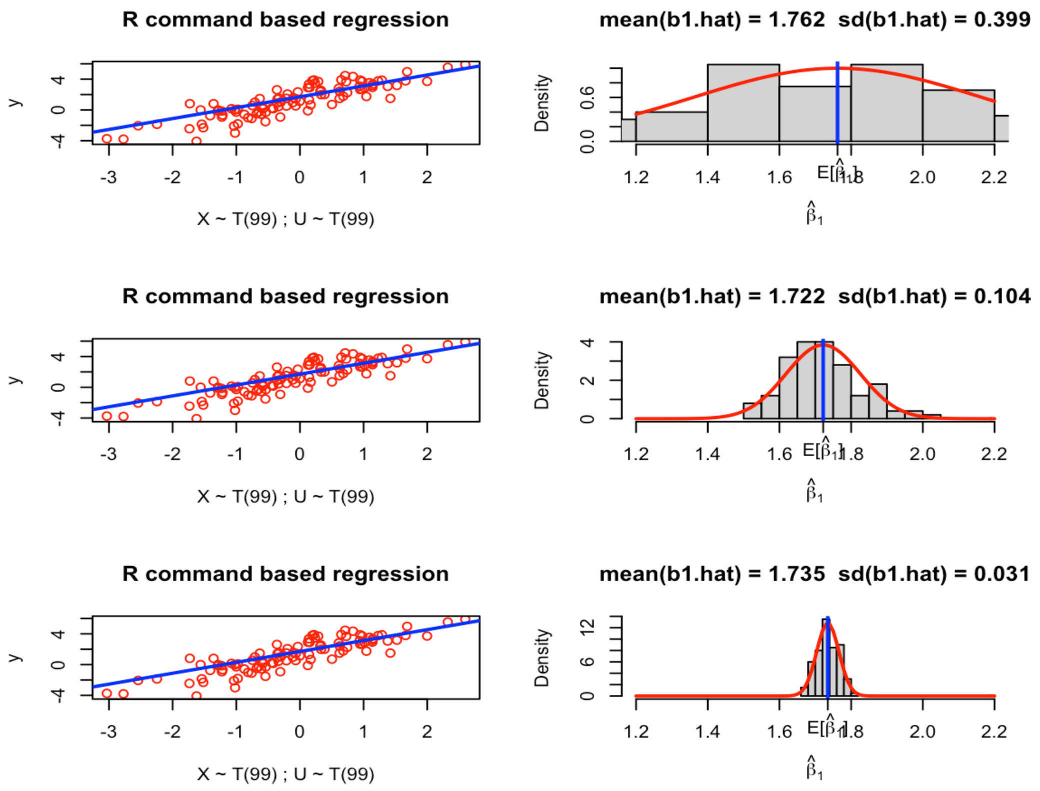
```
==Computational Regression==
estimated coefs : 1.41703267339052 1.71187490891554 #beta0.hat and beta1.hat
expected y : 1.22815368280072 # E[Y]
expected model : 1.22815368280072 # E[Y] = E[Model]
expected value of b1.hat : 1.72197079496771
standard deviation of b1.hat : 0.103944725865318
```

The lines A,B, and C of the code plot the following graphs :



The plots above show that as the number of replications (length of for loop) increases, the accuracy of the mean of b1.hat values increases as well (closer to true.b1 = 1.732). But the standard deviation of b1.hat values remains at the same level for all three cases. We can say that the Central Limit Theorem is in action in these three cases.

However in the case below where we change the sample size for each case, the results become more interesting. The lines D,E, and F of the code plot the following graphs :



As we can see in the right-hand side of the image, the increase in sample size results in increase in accuracy of the mean of estimated β_1 values and decrease in the standard deviation of estimated β_1 values obtained from Monte Carlo simulation. Here we can say that the Law of Large Number is in action.

We can replicate the code and its results above also with manual computation of OLS estimation in scalar form and in matrix form as well.

```
##============Manual Regression in Scalar Form================
Scalar_Regression <- function(size, rep, intercept, slope){
  ## Compute OLS estimators manually in scalar form
  b0.hat.scalar <<- mean(y) - (sum(x * y)/sum((x - mean(x))^2)) * mean(x)
  b1.hat.scalar <<- sum((x - mean(x))*(y - mean(y)))/sum((x - mean(x))^2)
  # COV(X,Y)/V(X)
  ## Print scalar form computation results
  cat("==Scalar Form Regression==",
      paste("estimated coefs :", b0.hat.scalar, b1.hat.scalar),
      paste("expected y :", mean(y)),
      paste("expected model :", b0.hat.scalar + b1.hat.scalar * mean(x)), sep = "\n")
  ## Scatter plot and fitted line with computed coefficients
  plot(x, y, col = "red", xlab = "x ~ T(99) ; u ~ T(99)", main = "Scalar form
      regression")
  abline(a = b1.hat.scalar, b = b0.hat.scalar, col = "blue", lwd = 2)
  ## Monte Carlo Simulation
  ## Set the number of replications and Initialize vectors
  R <- rep
  b0.hat <- rep(0,R)
  b1.hat <- rep(0,R)
  set.seed(2) # for replicability
  ## Draw a new sample each time in the loop and compute each time OLS estimators in
      scalar form
  for (i in 1:R){
    X <- rt(size, (size - 1))
    U <- rt(size, (size - 1))
    Y <- true.b0 + true.b1*X + U
    ## Store computed OLS estimators in initialized vectors
    b0.hat[i] <- mean(Y) - (sum(X * Y)/sum((X - mean(X))^2)) * mean(X)
    b1.hat[i] <- sum((X - mean(X))*(Y - mean(Y)))/sum((X - mean(X))^2)
  }
  ## Print the mean of vector of estimated slopes (mean of elements of b1.hat vector)
  cat(paste("expected value of b1.hat :", mean(b1.hat)),
      paste("standard deviation of b1.hat :", sd(b1.hat)), sep = "\n")
  ## Plot histogram of all estimated slopes and indicate the position of their mean
  hist(b1.hat, freq=FALSE, xlim=c(1.2,2.2),
       main=(paste("mean(b1.hat) =",round(mean(b1.hat), digits = 3),
                   " sd(b1.hat) =", round(sd(b1.hat), digits = 3))),
       xlab=expression(hat(beta)[1]))
  abline(v = mean(b1.hat), col = "blue", lwd = 2)
  axis(1, at = mean(b1.hat), lab = expression(paste("E[", hat(beta)[1], "]")))
  ## Plot PDF of normal distribution with the mean and the standard deviation of
      estimated slopes
  xfit <- seq(1.2,2.2,0.01)
  lines(xfit,dnorm(xfit,mean(b1.hat),sd(b1.hat)),lwd=2,col="red")
}
##============================================================
```

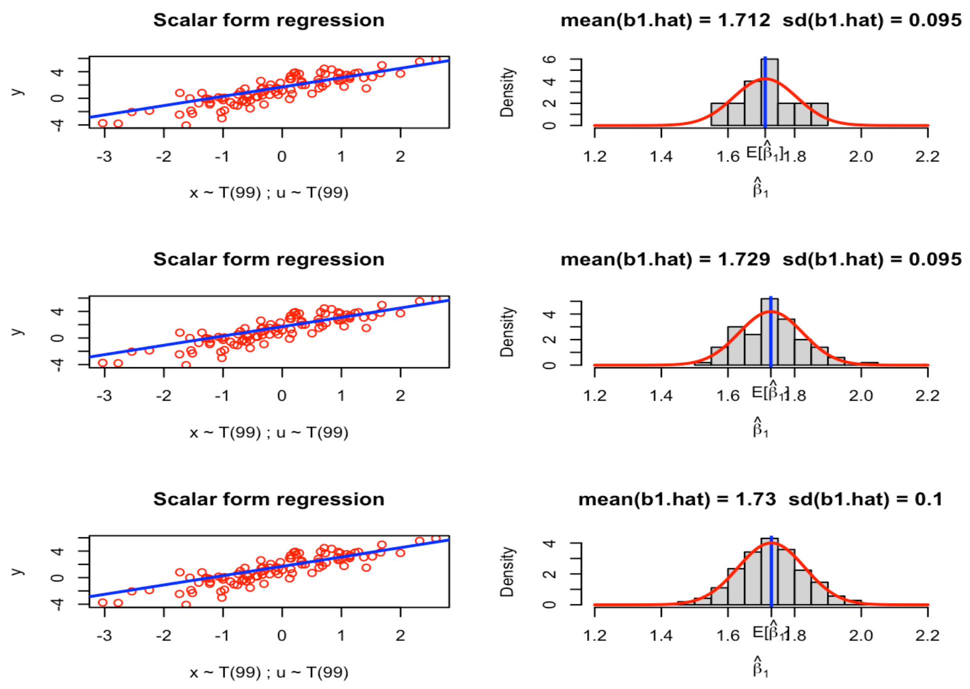
```r
##============Manual Regression in Matrix Form================
Matrix_Regression <- function(size, rep, intercept, slope){
  ## Transform x, u, and y into matrix form
  X <- as.matrix(cbind(1, x)) # X(100,2)
  U <- as.matrix(u) # U(100,1)
  Y <- as.matrix(y) # Y(100,1)
  ## Compute OLS estimators manually in matrix form
  b.hat.matrix <- solve((t(X) %*% X)) %*% t(X) %*% Y # beta_matrix = (X'X)^{-1}X'Y
  ## Separate computed Beta matrix of OLS estimators
  b0.hat.matrix <<- b.hat.matrix[1,1]
  b1.hat.matrix <<- b.hat.matrix[2,1]
  ## Print scalar form computation results
  cat("==Matrix Form Regression==",
      paste("estimated coefs :", b0.hat.matrix, b1.hat.matrix),
      paste("expected y :", mean(y)),
      paste("expected model :", b0.hat.matrix + b1.hat.matrix * mean(x)), sep = "\n")
  ## Scatter plot and fitted line with computed coefficients
  plot(x, y, col = "red", xlab = "x ~ T(99) ; u ~ T(99)", main = "Matrix form
      regression")
  abline(a = b1.hat.matrix, b = b0.hat.matrix, col = "blue", lwd = 2)
  ## Monte Carlo Simulation
  ## Set the number of replications and Initialize vectors
  R <- rep
  B <- as.matrix(c(true.b0, true.b1)) # transform true Beta vector into a matrix
  B0.hat <- rep(0,R)
  B1.hat <- rep(0,R)
  set.seed(3) # for replicability
  ## Draw a new sample each time in the loop and compute each time OLS estimators in
      matrix form
  for (i in 1:R){
    x.mat <- rt(size, (size - 1))
    X.mat <- as.matrix(cbind(1, x.mat))
    u.mat <- rt(size, (size - 1))
    U.mat <- as.matrix(u.mat)
    Y.mat <- X.mat%*%B + U.mat
    ## Store computed OLS estimators in initialized vectors
    B.hat <- solve((t(X.mat) %*% X.mat)) %*% t(X.mat) %*% Y.mat
    B0.hat[i] <- B.hat[1,1] # 1st row x 1st column element of B.hat matrix stored in
        B0.hat vector
    B1.hat[i] <- B.hat[2,1] # 2nd row x 1st column element of B.hat matrix stored in
        B1.hat vector
  }
  ## Print the mean of vector of estimated slopes (mean of elements of B1.hat vector)
  cat(paste("expected value of b1.hat :", mean(B1.hat)),
      paste("standard deviation of b1.hat :", sd(B1.hat)), sep = "\n")
  ## Plot histogram of all estimated slopes and indicate the position of their mean
  hist(B1.hat, freq=FALSE, xlim=c(1.2,2.2),
       main=(paste("mean(b1.hat) =",round(mean(B1.hat), digits = 3),
                   " sd(b1.hat) =", round(sd(B1.hat), digits = 3))),
       xlab=expression(hat(beta)[1]))
  abline(v = mean(B1.hat), col = "blue", lwd = 2)
  axis(1, at = mean(B1.hat), lab = expression(paste("E[", hat(beta)[1], "]")))
  ## Plot PDF of normal distribution with the mean and the standard deviation of
      estimated slopes
  xfit<-seq(1.2,2.2,0.01)
  lines(xfit,dnorm(xfit,mean(B1.hat),sd(B1.hat)),lwd=2,col="red")
}
##==============================================================
## Same functions with different number of reps and N fixed to 100
Scalar_Regression(N, 10, true.b0, true.b1) # A
Scalar_Regression(N, 100, true.b0, true.b1) # B
Scalar_Regression(N, 1000, true.b0, true.b1) # C
Matrix_Regression(N, 10, true.b0, true.b1) # A'
Matrix_Regression(N, 100, true.b0, true.b1) # B'
Matrix_Regression(N, 1000, true.b0, true.b1) # C'
## Same functions with different sample sizes and rep fixed to 100
Scalar_Regression(10, 100, true.b0, true.b1) # D
Scalar_Regression(100, 100, true.b0, true.b1) # E
Scalar_Regression(1000, 100, true.b0, true.b1) # F
Matrix_Regression(10, 100, true.b0, true.b1) # D'
Matrix_Regression(100, 100, true.b0, true.b1) # E'
Matrix_Regression(1000, 100, true.b0, true.b1) # F'
```

The lines B and B' print the results below :

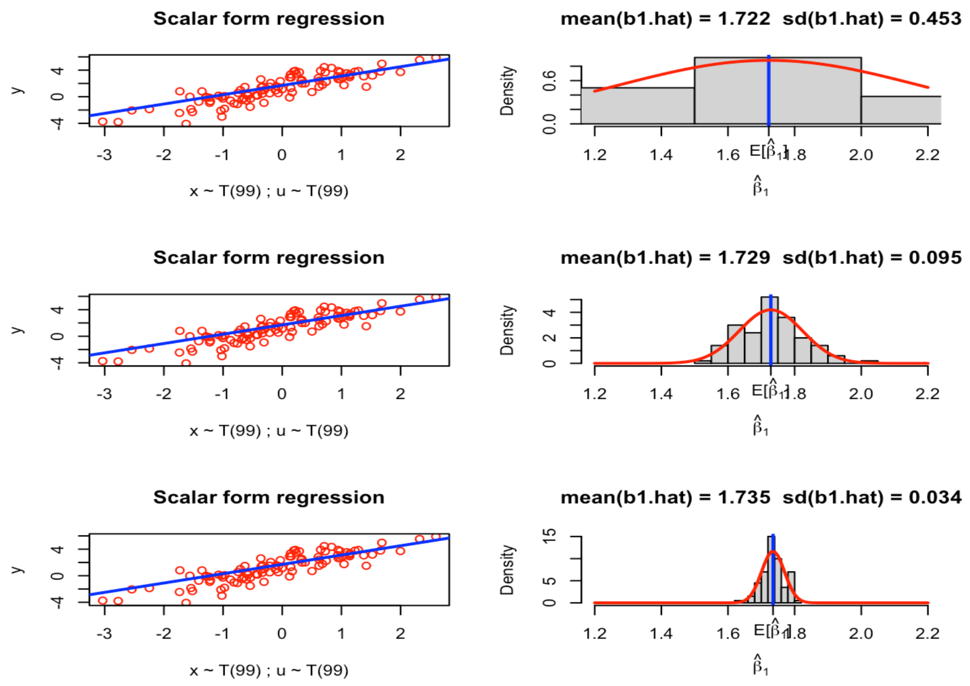
```
==Scalar Form Regression==
estimated coefs : 1.40429950324902 1.71187490891554 # b0.hat and b1.hat
expected y : 1.22815368280072
expected model : 1.21542051265922
expected value of b1.hat : 1.72864698178697
standard deviation of b1.hat : 0.0950631857094078

==Matrix Form Regression==
estimated coefs : 1.41703267339052 1.71187490891554 # b0.hat and b1.hat
expected y : 1.22815368280072
expected model : 1.22815368280072
expected value of b1.hat : 1.73660662437206
standard deviation of b1.hat : 0.0932501510911413
```

All together, the lines A,B,C and A',B',C' show the sampling properties of Central Limit Theorem, and the lines D,E,F and D',E',F' show the sampling properties of Law of Large Numbers. (The principle is the same as the first code with R built-in function **lm**).

Scalar form regression and illustration of sampling properties through Monte Carlo simulation

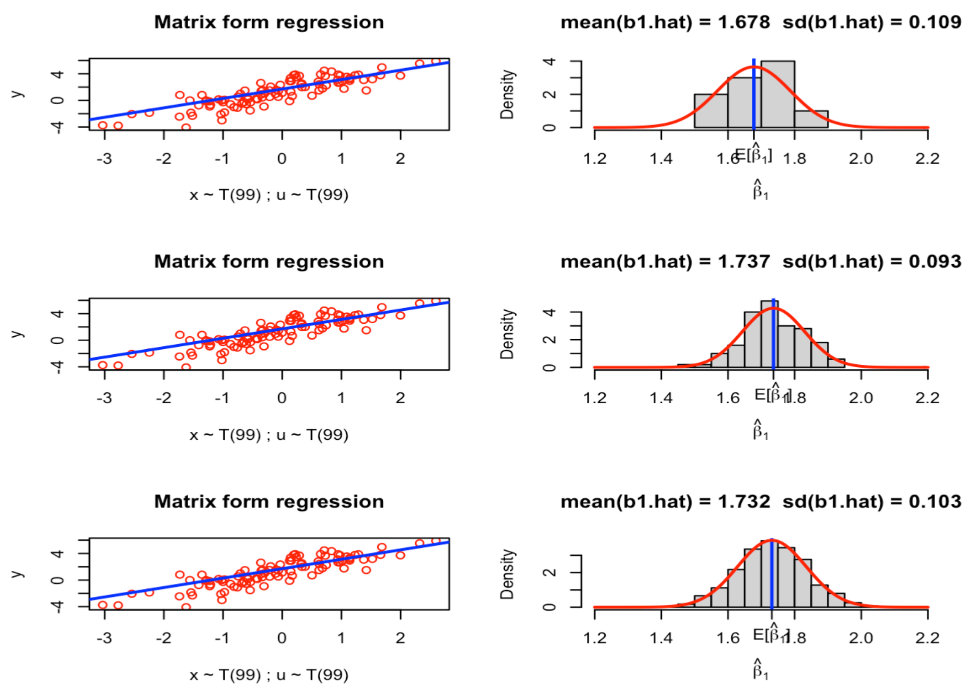


[CLT sampling properties : only the mean is affected]

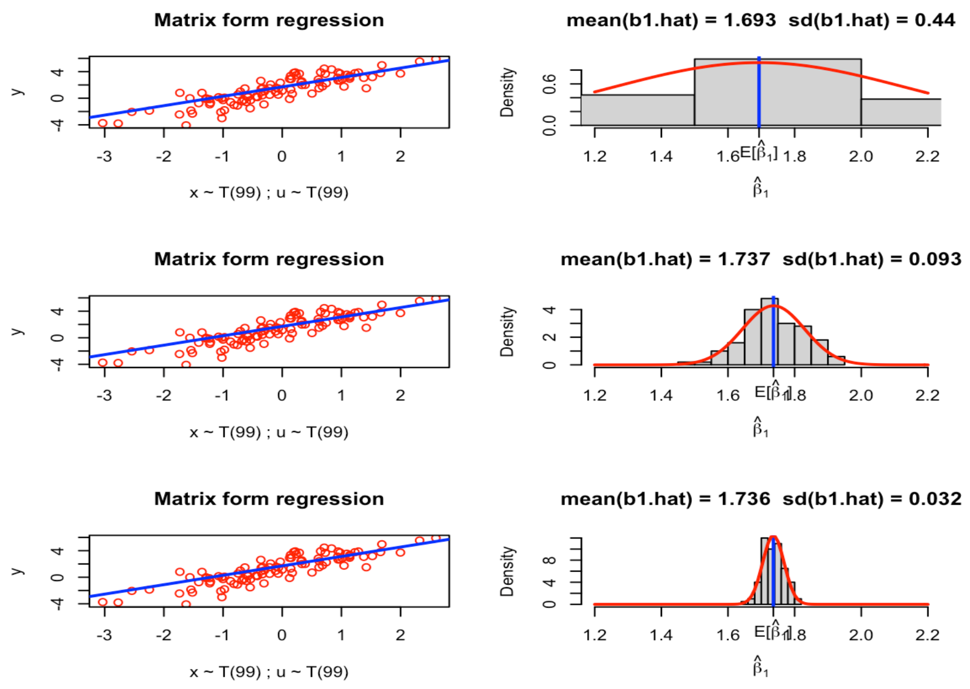


[LLN sampling properties : both mean and variance affected]

Matrix form regression and illustration of sampling properties through Monte Carlo simulation



[CLT sampling properties : only the mean is affected]



[LLN sampling properties : both mean and variance affected]

The last method, Numerical Optimization is a particular method different from all three methods described previously (the first three methods are not actually different from each other. Just the way to compute them are different). The following R code answers to the question 1 and 2 for Numerical Optimization method and the question 4 as well :

```r
## Minimization function of Residual Sum of Squares
min.RSS <- function(par){
  b0 <- par[1]
  b1 <- par[2]
  RSS <- sum((y - b0 - b1*x)^2)
  return(RSS)
}
## Numerical Optimization function
Numerical_Optimization <- function(size, rep, intercept, slope){
  ## optim() function by default execute minimization of given function with
      Nelder-Mead method
  result1 <- optim(par = c(1,1), fn = min.RSS, control = list(reltol = 1e-06)) #
      strict tolerance
  result2 <- optim(par = c(1,1), fn = min.RSS, control = list(reltol = 1)) # loose
      tolerance
  ## Get optim function results
  b0.hat.opt1 <<- result1$par[1]
  b1.hat.opt1 <<- result1$par[2]
  b0.hat.opt2 <<- result2$par[1]
  b1.hat.opt2 <<- result2$par[2]
  ## Print optimization results
  cat("==Numerical Optimization==",
      paste("estimated coefs with reltol = 1e-06 :", b0.hat.opt1, b1.hat.opt1),
      paste("number of iterations with reltol = 1e-06 :", result1$counts[1]),
      paste("estimated coefs with reltol = 1 :", b0.hat.opt2, b1.hat.opt2),
      paste("number of iterations with reltol = 1 :", result2$counts[1]),
      paste("expected y :", mean(y)),
      paste("epxected model with tolerance 1e-06 :", b0.hat.opt1 + b1.hat.opt1 *
         mean(x)),
      paste("expected model with tolerance 1 :", b0.hat.opt2 + b1.hat.opt2 *
         mean(x)), sep = "\n")
  ## Scatter plot and fitted line with computed coefficients of two different
      tolerance levels
  plot(x, y, col = "red", xlab = "x ~ T(99) ; u ~ T(99)", main = "Numerical
      Optimization")
  abline(a = b0.hat.opt1, b = b1.hat.opt1, col = "black", lwd = 2)
  abline(a = b0.hat.opt2, b = b1.hat.opt2, col = "grey", lwd = 2)
}
```

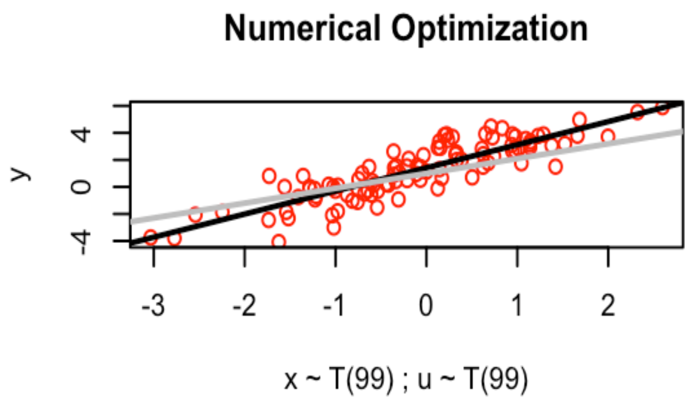
The function prints the following results :

```
==Numerical Optimization==
estimated coefs with reltol = 1e-06 : 1.41639052042738 1.71239887578413
number of iterations with reltol = 1e-06 : 45 # optimized 45 times
estimated coefs with reltol = 1 : 1 1.1
number of iterations with reltol = 1 : 3 # optimized 3 times
expected y : 1.22815368280072
epxected model with tolerance 1e-06 : 1.2274537181758
expected model with tolerance 1 : 0.878631967460522
```

The optimization result with relative tolerance equal to 1e-06 acquired approximately 1.4164 and 1.7124 as $\hat{\beta}_0$ and $\hat{\beta}_1$ values respectively after 45 times of iterations (trials). These are quite accurate.

The optimization result with relative tolerance equal to 1 acquired 1 and 1.1 as $\hat{\beta}_0$ and $\hat{\beta}_1$ values respectively after 3 times of iterations (trials).

The algorithm of **optim** function is set to stop if it's unable to reduce the value by a factor of $reltol \times (|value| + reltol)$ at a step. Therefore setting the reltol $= 1$ generated poorly estimated values for $\hat{\beta}_0$ and $\hat{\beta}_1$.

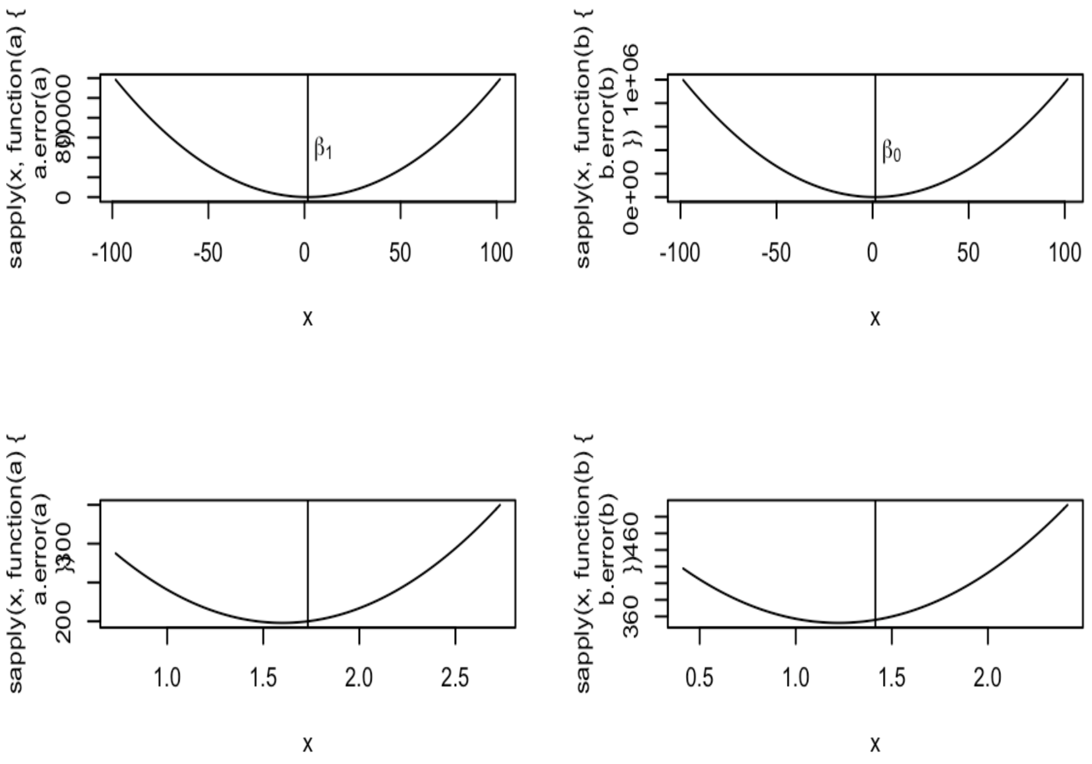


The regression line with coefficients obtained from first optimization with strict relative tolerance value (1e-06) is in black and the coefficients estimated with loose relative tolerance level is in grey. As we can see, the black line is better fitted with the data (red dots).

We can try to study the sampling properties of optimization method but it will not bring any additional interesting insights because we will get the same values over and over if the iterated values by optim function are the global minima where min.RSS formula gets minimized . Therefore there will be no interest to study the mean and the standard deviation of 100 or 1000 same values. Instead, what we can graphically illustrate what is the intuition behind optimization method and see if $\hat{\beta}_0$ and $\hat{\beta}_1$ are global minima.

```
## Transform x and y into dataframe
df <- data.frame(x, y)
## Generate Model as a function
Model <- function(x, a, b, u){
  return(b + a*x + u)
}
## Squared Errors between the model and the data
Squared_Error <- function(data, a, b, u){
  predictions <- with(data, Model(x, a, b, u))
  errors <- with(data, y - predictions)
  return(sum(errors^2))
}
## Squared error only for slope
a.error <- function(a){
  return(Squared_Error(df, a, 0, u))
}
## Squared error only for intercept
b.error <- function(b){
  return(Squared_Error(df, 0, b, u))
}
## Plot squared errors
a.curve <- curve(sapply(x, function(a) {a.error(a)}), from = true.b1 - 100, to =
    true.b1 + 100)
abline(v = true.b1) ; text(10, 500000, expression(beta[1]))
b.curve <- curve(sapply(x, function(b) {b.error(b)}), from = true.b0 - 100, to =
    true.b0 + 100)
abline(v = true.b0) ; text(10, 400000, expression(beta[0]))
a.curve <- curve(sapply(x, function(a) {a.error(a)}), from = true.b1 - 1, to =
    true.b1 + 1)
abline(v = true.b1) ; text(10, 500000, expression(beta[1]))
b.curve <- curve(sapply(x, function(b) {b.error(b)}), from = true.b0 - 1, to =
    true.b0 + 1)
abline(v = true.b0) ; text(10, 400000, expression(beta[0]))
```

The code above gives the graphs below :



The first line of graphs show that β_1 and β_0 are global minima at which the squared errors become the lowest.

The second line of graphs are zoomed version of the first line. It shows that the estimated values of β_1 and β_0 are actually slightly off the actual minima but they are close enough. They are approximated pretty well.

Conclusion

The three similar methods (R built-in function, Scalar form computation, Matrix form computation) of OLS estimation showed us that they are efficient methods. Each of them also provided statistical insights about their sampling properties through Monte Carlo simulation.

The numerical optimization is different from the other three methods but the objective remains the same. With the last graphical analysis and the result obtained from Numerical Optimization function, we can say that optimization method with strict relative tolerance provided us well approximated coefficients for the OLS estimation.