Definimos el vector aleatorio

```
Z <- matrix(nrow = 4, ncol = 10)
Z[,1] <- c(0.2, 2.4, -1.61, 5.7)
Z[,2] <- c(0, 3.2, -3.04, 6.4)
Z[,3] <- c(0.8, -0.8, -3.71, 7.5)
Z[,4] <- c(-1.3, 1.2, 1.64, 4)
Z[,5] <- c(-0.4, 0.6, 4.2, 2.6)
Z[,6] <- c(2.1, -2.2, 0.28, 5.4)
Z[,7] <- c(-0.8, -1.4, 1.06, 4.8)
Z[,8] <- c(1.6, 0.8, 2.1, 3.8)
Z[,9] <- c(2.2, 1, 1.36, 4.2)
Z[,10] <- c(0.6, -1.8, -1.2, 6.2)
Z <- data.frame(t(Z))
names(Z) <- c("Y", "X1", "X2", "X3")
t(Z)
```

```
##     [,1]  [,2]  [,3]  [,4] [,5]  [,6]  [,7] [,8] [,9] [,10]
## Y   0.20  0.00  0.80 -1.30 -0.4  2.10 -0.80  1.6 2.20   0.6
## X1  2.40  3.20 -0.80  1.20  0.6 -2.20 -1.40  0.8 1.00  -1.8
## X2 -1.61 -3.04 -3.71  1.64  4.2  0.28  1.06  2.1 1.36  -1.2
## X3  5.70  6.40  7.50  4.00  2.6  5.40  4.80  3.8 4.20   6.2
```

Definimos las matrices X^T y X.

Veremos que X^T es generada por la matriz `X_transpose`, y que efectivamente es de dimensiones 4x10

```
X_transpose <- rbind(rep(1,10), t(Z[,2:4]))
X_transpose
```

```
##     [,1]  [,2]  [,3] [,4] [,5]  [,6]  [,7] [,8] [,9] [,10]
##     1.00  1.00  1.00 1.00  1.0  1.00  1.00  1.0 1.00   1.0
## X1  2.40  3.20 -0.80 1.20  0.6 -2.20 -1.40  0.8 1.00  -1.8
## X2 -1.61 -3.04 -3.71 1.64  4.2  0.28  1.06  2.1 1.36  -1.2
## X3  5.70  6.40  7.50 4.00  2.6  5.40  4.80  3.8 4.20   6.2
```

Y en cambio, X, generado por la matriz `X`, es de dimensiones 10x4

```
X <- t(X_transpose)
X
```

```
##           X1    X2  X3
##  [1,] 1  2.4 -1.61 5.7
##  [2,] 1  3.2 -3.04 6.4
##  [3,] 1 -0.8 -3.71 7.5
##  [4,] 1  1.2  1.64 4.0
##  [5,] 1  0.6  4.20 2.6
##  [6,] 1 -2.2  0.28 5.4
##  [7,] 1 -1.4  1.06 4.8
##  [8,] 1  0.8  2.10 3.8
##  [9,] 1  1.0  1.36 4.2
## [10,] 1 -1.8 -1.20 6.2
```

La matriz $X^T X$ de dimensiones 4x4 se genera con el siguiente código:

```
matrix_XX <- X_transpose%*%X
matrix_XX
```

```
##              X1      X2      X3
##    10.00  3.000   1.080  50.600
## X1  3.00 30.120  -3.036  12.000
## X2  1.08 -3.036  54.829 -26.126
## X3 50.60 12.000 -26.126 275.180
```

Y su determinante es:

```
round(det(matrix_XX))
```

```
## [1] 0
```

Para hallar los valores de α_1, α_2 y α_3 podemos resolver el sistema sobredeterminado del tipo $\mathbf{A}\boldsymbol{\alpha} = \mathbf{b}$ utilizando el método de Gauss; no obstante, el siguiente código permite realizar los cálculos mediante la matriz inversa de Moore-Penrose de forma más rapida.

```
library(MASS)
library(matlib)

# A es la matriz 10x3 de las columnas 1s, X2 y X3
A <- as.matrix(X[,-2])
# b es el vector de dimensión 10 de la columna X1
b <- as.matrix(X[,2])

Ag <- ginv(A)
alphas_X1 <- Ag %*% b
```

Los coeficientes del vector $\boldsymbol{\alpha}$ que resuelven el sistema son

```
alphas_X1
```

```
##           [,1]
## [1,] 29.333333
## [2,] -3.333333
## [3,] -5.666667
```

Podemos realizar el mismo proceso para encontrar los coeficientes que hacen que el resto de combinaciones lineales existan y que hacen que las columnas sean linealmente dependientes.

```
A <- as.matrix(X[,-1])
b <- as.matrix(X[,1])
Ag <- ginv(A)
alphas_1s <- Ag %*% b

A <- as.matrix(X[,-3])
b <- as.matrix(X[,3])
Ag <- ginv(A)
alphas_X2 <- Ag %*% b

A <- as.matrix(X[,-4])
b <- as.matrix(X[,4])
Ag <- ginv(A)
alphas_X3 <- Ag %*% b
```

Concluimos que todas las columnas son combinaciones lineales del resto. Dichas combinaciones lineales son:

$$1_{10x1} = 0.03 * X_{(,1)} + 0.11 * X_{(,2)} + 0.19 * X_{(,3)}$$

$$X_{(,1)} = 29.33 * 1_{10x1} + (-3.33) * X_{(,2)} + (-5.67) * X_{(,3)}$$

$$X_{(,2)} = 8.8 * 1_{10x1} + (-0.3) * X_{(,1)} + (-1.7) * X_{(,3)}$$

$$X_{(,3)} = 5.18 * 1_{10x1} + (-0.18) * X_{(,1)} + (-0.59) * X_{(,2)}$$

```r
det(as.matrix(X)%*%X_transpose)
```

```
## [1] -1.084414e-96
```

Es decir, el determinante de XX^T es cero, por lo que los renglones de X no son linealmente independientes.

iii) Generar (usando algún paquete estadístico) una muestra i.i.d.

$$\left\{ \begin{pmatrix} Y_i \\ X_{1i} \\ X_{2i} \end{pmatrix} \right\}_{i=1}^{400} \text{ donde } \begin{pmatrix} Y_i \\ X_{1i} \\ X_{2i} \end{pmatrix} \sim N_3 \left(\begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}, \begin{pmatrix} 0.8 & 0.4 & -0.2 \\ 0.4 & 1.0 & -0.8 \\ -0.2 & -0.8 & 2.0 \end{pmatrix} \right)$$

```
library(mvtnorm)
n <- 400
mu <- c(1,0,2)
Sigma <- matrix(c(0.8, 0.4, -0.2,
                  0.4, 1, -0.8,
                  -0.2, -0.8, 2),
                nrow = 3,
                byrow = T)
Z <- data.frame(rmvnorm(n, mean = mu, sigma = Sigma))
names(Z) <- c("Y", "X1", "X2")
```

Comprobemos que las medias y la matriz de varianza covarianza se asemejan a lo indicado

```
round(colMeans(Z), 3)
```

```
##     Y    X1    X2
## 1.046 0.049 1.974
```

```
round(var(Z), 3)
```

```
##          Y     X1     X2
## Y    0.699  0.372 -0.144
## X1   0.372  1.134 -0.878
## X2 -0.144 -0.878  2.034
```

iv) Con los datos obtenidos, estimar el vector $\mathbf{b}$ usando los siguientes estimadores obtenidos por el principio de analogía:

a) $\hat{\mathbf{b}} = (\mathbf{X^T X})^{-1} \mathbf{X^T Y}$

b) $\left(\frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j \mathbf{x}_j^T \right)^{-1} \frac{1}{n} \sum_{j=1}^{n} \mathbf{x}_j Y_j$, donde $\mathbf{x_j} \equiv \begin{pmatrix} 1 \\ \mathbf{X_j} \end{pmatrix}$, $j = 1, 2, \ldots, n$

c) $\hat{\mathbf{b}} = \begin{pmatrix} \hat{\mu_y} - \hat{\Sigma}_{xy} \hat{\Sigma}_x^{-1} \hat{\mu}_x \\ \hat{\Sigma}_x^{-1} \hat{\Sigma}_{xy}^T \end{pmatrix}$, $j = 1, 2, \ldots, n$

donde

$$\hat{\mu}_y = \frac{1}{n} \sum_{j=1}^{n} Y_j \qquad \hat{\mu}_x = \frac{1}{n} \sum_{j=1}^{n} \mathbf{X}_j$$

$$\hat{\Sigma}_x = \frac{1}{n} \sum_{j=1}^{n} (\mathbf{X}_j - \hat{\mu}_x)(\mathbf{X}_j - \hat{\mu}_x)^T \qquad \hat{\Sigma}_{yx} = \frac{1}{n} \sum_{j=1}^{n} (Y_j - \hat{\mu}_y)(Y_j - \hat{\mu}_y)^T$$

Comenzemos con el inciso a. Para ello, construyamos la matriz $\mathbf{X}$ y el vector $\mathbf{Y}$.

```
X <- data.frame(rep(1, n), Z[,-1])
names(X)[1] <- "1s"
X <- as.matrix(X)
Y <- as.matrix(Z$Y)
```

Entonces, de acuerdo con la fórmula del inciso a, calculemos $\hat{\mathbf{b}}_a$

```
bhat_a <- solve(t(X)%*%X)%*%t(X)%*%Y
```

Hagamos lo propio con el inciso b, para calcular $\hat{\mathbf{b}}_b$

```
# calculemos 1/n*suma x_jx_j^'
x_jx_j <- 0
for (i in 1:n) {
  x_jx_j <- x_jx_j + 1/n * as.matrix(X[i,])%*%t(as.matrix(X[i,]))
}
x_jx_j_inv <- solve(x_jx_j)
# calculamos 1/n x_jY_j
x_jY_j <- 0
for (i in 1:n) {
  x_jY_j <- x_jY_j + 1/n * as.matrix(X[i,])%*%Y[i]
}
bhat_b <- x_jx_j_inv%*%x_jY_j
```

Y finalmente, con la fórmula de c, tenemos $\hat{\mathbf{b}}_c$

```
## c) calculemos mu_y - cov(y,x)*var(x)^1*mu_x
bhat_c <- matrix(rep(0,3))
bhat_c[1] <- mean(Y) - matrix(var(Y,X)[-1], nrow = 1)%*%solve(var(X)[-1,-1])%*%colMeans(X)[-1]
# calculemos var(X)^-1 * cov(y,x)
bhat_c[-1] <- solve(var(X)[-1,-1])%*%as.matrix(var(Y,X)[1,-1])
```

Con las tres fórmulas llegamos a los mismos resultados; es decir: $\hat{\mathbf{b}}_a = \hat{\mathbf{b}}_b = \hat{\mathbf{b}}_c$

```
beta <- data.frame(bhat_a, bhat_b, bhat_c)
rownames(beta) <- c("Bo", "B1", "B2")
beta
```

```
##       bhat_a    bhat_b    bhat_c
## Bo 0.8157886 0.8157886 0.8157886
## B1 0.4108243 0.4108243 0.4108243
## B2 0.1066048 0.1066048 0.1066048
```

v) Con los datos obtenidos, estimar $\hat{\sigma}_u^2$ usando los siguientes estimadores obtenidos por el principio de analogía:

w) $\hat{\sigma}_u^2 = \frac{1}{n}\sum_{j=1}^{n}\hat{U}_j^2 = \frac{1}{n}\sum_{j=1}^{n}(Y_j - \hat{\mathbf{b}}'\mathbf{x}_j)^2$

x) $\hat{\sigma}_u^2 = \hat{\sigma}_y^2 - \hat{\Sigma}_{yx}\hat{\Sigma}_x^{-1}\hat{\Sigma}_{yx}'$

donde $\hat{\sigma}_y^2 = \frac{1}{n}\sum_{j=1}^{n}(Y_j - \hat{\mu}_y)^2$

Calculemos primero usando la fórmula del inciso a: $\hat{\sigma}_{u,a}^2$

```
sigma_u_hat_a <- 0
for (i in 1:n) {
  sigma_u_hat_a <- sigma_u_hat_a + 1/n*(Y[i] - t(bhat_a)%*%matrix(X[i,]))^2
}
```

Ahora, con el inciso b: $\hat{\sigma}_{u,b}^2$

```
# para errores de cálculo, generemos manualmente var(Y) y cov(Y,X)
sigma_Y <- 0
for (i in 1:n) {
  sigma_Y <- sigma_Y + 1/n*(Y[i] - mean(Y))^2
}

sigma_YX <- matrix(rep(0, 2), nrow = 1)
for (i in 1:n) {
  sigma_YX <- sigma_YX + 1/n*(Y[i] - mean(Y))*t(matrix(X[i,-1]) - matrix(colMeans(X[,-1])))
}

sigma_u_hat_b <- sigma_Y - sigma_YX%*%solve(var(X)[-1,-1])%*%t(matrix(var(Y,X)[-1], nrow = 1))
```

Ambas fórmulas resultan en lo mismo, es decir: $\hat{\sigma}_{u,a}^2 = \hat{\sigma}_{u,b}^2$

```
data.frame(sigma_u_hat_a, sigma_u_hat_b)
```

```
##   sigma_u_hat_a sigma_u_hat_b
## 1     0.5600865     0.5600865
```