## Q1:

The following distribution has the density function:

$$f(x;\theta) = \frac{1-\cos(x-\theta)}{2\pi}, \quad 0 \le x \le 2\pi, \quad -\pi < \theta < \pi$$

For an observed random sample $x_1, x_2, \dots, x_n$ from this distribution, the log likelihood is seen to be:

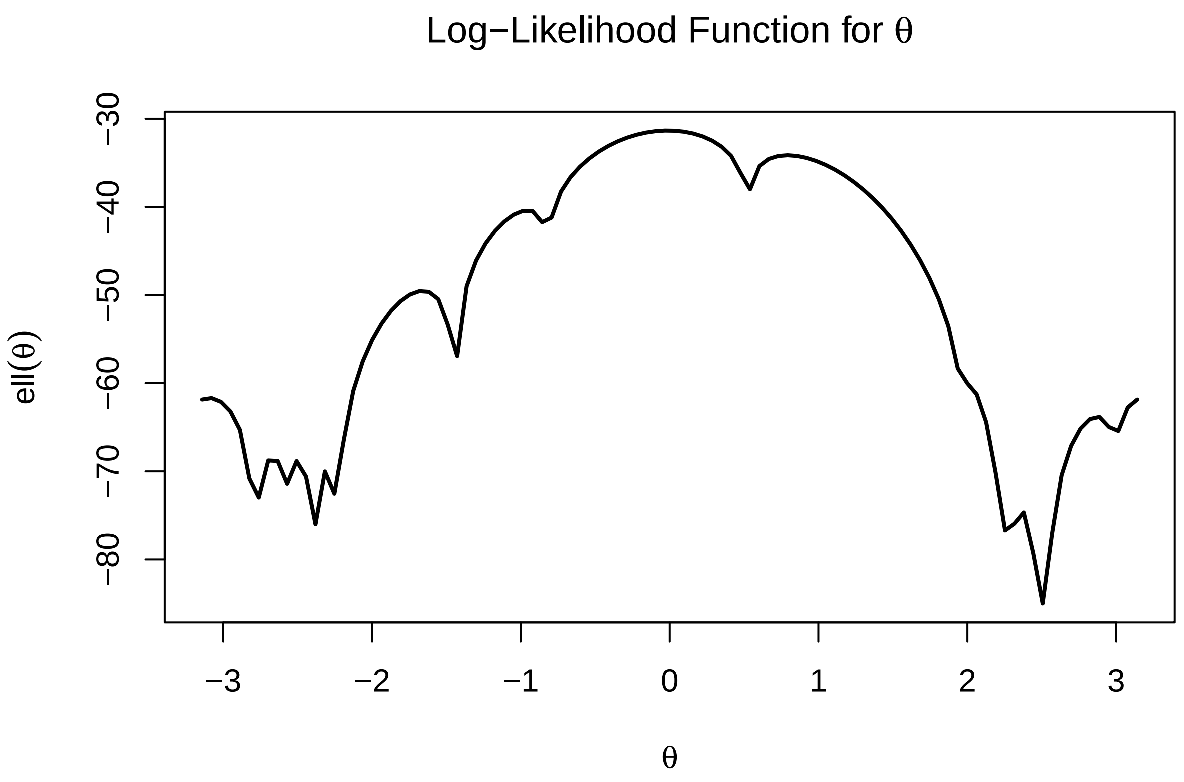
$$\ell(\theta) = -n\log 2\pi + \sum_{i=1}^{n} \log\{1-\cos(x_i-\theta)\}$$

Suppose that $(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96, 2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)$ is an observed random sample from the above distribution.

### (a)

Plot the log likelihood $\ell(\theta)$ in the range $-\pi < \theta < \pi$.

```r
x <- c(3.91, 4.85, 2.28, 4.06, 3.70, 4.04, 5.46, 3.53, 2.28, 1.96,
       2.53, 3.88, 2.22, 3.47, 4.82, 2.46, 2.99, 2.54, 0.52, 2.50)

logLike <- function(theta, x) {
  n <- length(x)
  -n * log(2 * pi) + sum(log(1 - cos(x - theta)))
}

thetaVal <- seq(from = -pi,
                    to = pi,
                    length.out = 100)

logLikeVal <- sapply(X = thetaVal,
                       FUN = logLike,
                       x = x)

plot(x = thetaVal,
     y = logLikeVal,
     type = "l",
     lwd = 2,
     col = "black",
     main = expression("Log-Likelihood Function for " * theta),
     xlab = expression(theta),
     ylab = expression(ell(theta))
     )
```



**(b)**

Use the R function `optimize()` to find the maximum likelihood estimate of $\theta$.

```r
mleEst <- optimize(f = logLike,
                   interval = c(-pi, pi),
                   x = x,
                   maximum = TRUE)

theta_mle <- mleEst$maximum
log_lik_mle <- mleEst$objective

cat("The maximum likelihood estimate of theta is:", theta_mle, "\n")
```

```
## The maximum likelihood estimate of theta is: -0.0119724
```

```r
cat("The maximum log-likelihood value is:", log_lik_mle, "\n")
```

```
## The maximum log-likelihood value is: -31.34291
```

**(c)**

Use function `newton()` in the class handout to find the maximum likelihood estimate of $\theta$ by solving $\ell'(\theta) = 0$ using the starting value of $\theta_{(0)} = 0$.

```r
logLikePrime <- function(theta) {
  -sum(sin(x - theta) / (1 - cos(x - theta)))
}

logLikeDP <- function(theta) {
  -sum(1 / (1 - cos(x - theta)))
}

init <- 0
tolerance <- 1e-6

newton <- function(fun, derf, x0, eps) {
  iter <- 0
  repeat {
    iter <- iter + 1
    x1 <- x0 - fun(x0) / derf(x0)
    if (abs(x0 - x1) < eps || abs(fun(x1)) < 1e-10)
      break
    x0 <- x1
    cat("****** Iter. No:", iter, " Current Iterate =", x1, "\n", fill = TRUE)
  }
  return(x1)
}

theta_mle <- newton(fun = logLikePrime,
                    derf = logLikeDP,
                    x0 = init,
                    eps = tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -0.01191193
##
## ****** Iter. No: 2  Current Iterate = -0.011972
```

```r
cat(paste0("The maximum likelihood estimate is: ", theta_mle))
```

```
## The maximum likelihood estimate is: -0.0119720022874401
```

### (d)

What happens if you use $\theta_{(0)} = -2.0$ and $-2.7$, respectively, as starting values? Explain why.

```r
# x0 = -2.0
theta_mle_2_0 <- newton(fun = logLikePrime,
                        derf = logLikeDP,
                        x0 = -2.0,
                        eps = tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -1.756154
##
## ****** Iter. No: 2  Current Iterate = -1.641367
##
```

```
## ****** Iter. No: 3  Current Iterate = -1.657301
##
## ****** Iter. No: 4  Current Iterate = -1.65828
##
## ****** Iter. No: 5  Current Iterate = -1.658283
```

```r
# Cheeky lil line break, eh?
cat(paste0("The MLE of theta with initial value -2.0 is: ", theta_mle_2_0, "\n"))
```

```
## The MLE of theta with initial value -2.0 is: -1.65828322990256
```

```r
# x0= -2.7
theta_mle_2_7 <- newton(fun = logLikePrime,
                        derf = logLikeDP,
                        x0 = -2.7,
                        eps = tolerance)
```

```
## ****** Iter. No: 1  Current Iterate = -2.674114
##
## ****** Iter. No: 2  Current Iterate = -2.666794
##
## ****** Iter. No: 3  Current Iterate = -2.6667
```

```r
cat(paste0("The MLE of theta with initial value -2.7 is:", theta_mle_2_7))
```

```
## The MLE of theta with initial value -2.7 is:-2.66669992610095
```

In both instances we achieve convergence to a local maximums. This happens because the method used doesn't look over a large enough range of values, i.e. it finds a sufficient area for searching and either peaks or flat areas of a curve can cause the procedure to converge not to the global maximum. This is because the `newton` function as-written is sensitive to local maximums, such that we are able to satisfy the exit criteria of the optimization process without converging to the global maximum.

Turn in the plot, any functions you write, function calls, and the results.

**P.S.** To help you out with the necessary derivatives, I derived them below, but you need to check them out!

$$\frac{\partial \ell}{\partial \theta} = -\sum_{i=1}^{n} \frac{\sin(x_i - \theta)}{1 - \cos(x_i - \theta)}$$

$$\frac{\partial^2 \ell}{\partial \theta^2} = -\sum_{i=1}^{n} \frac{1}{1 - \cos(x_i - \theta)}$$

## Q2:

**Regression to the mean.** Consider the following very simple genetic model in which a population consists of equal numbers of two sexes: male and female. At each generation, men and women are paired at random, and each pair produces exactly two offspring, one male and one female. We are interested in the distribution of height from one generation to the next. Supposing that the height of both children is just the average of the heights of their parents, how will the distribution of height change across generations?

### (a)

Represent the heights of the current generation as a dataframe with two variables, $M$ and $F$ for the two sexes. Randomly generate a population at generation 1, as for males with $X_1, X_2, \ldots, X_{100} \sim N(125, 25^2)$, and for females $X_1, X_2, \ldots, X_{100} \sim N(125, 15^2)$.

```r
set.seed(43)

maleHeights <- rnorm(n = 100,
                     mean = 125,
                     sd = 25)
femaleHeights <- rnorm(n = 100,
                       mean = 125,
                       sd = 15)

gen1 <- data.frame(M = maleHeights,
                   F = femaleHeights)
```

### (b)

Take the dataframe from (a) and randomly permute the ordering of men. Men and women are then paired according to rows, and heights for the next generation are calculated by taking the mean of each row. The function should return a dataframe with the same structure, giving the heights of the next generation. You will need to use the `sample(x, size = n)` function to return a random sample of size $n$ from the vector $x$. You will also need to use the `apply()` function.

```r
genNextGen <- function(df) {
  permM <- sample(df$M, size = nrow(df))

  nextGen <- data.frame(M = permM, F = df$F)
  # get values for update
  nextGen$Mean_Height <- apply(X = nextGen,
                                MARGIN = 1,
                                FUN = function(row) {mean(row)}
                                )
  # update dataset
  nextGen <- data.frame(M = nextGen$Mean_Height, F = nextGen$Mean_Height)
  nextGen
}

nextGen <- genNextGen(df = gen1)
head(nextGen)
```

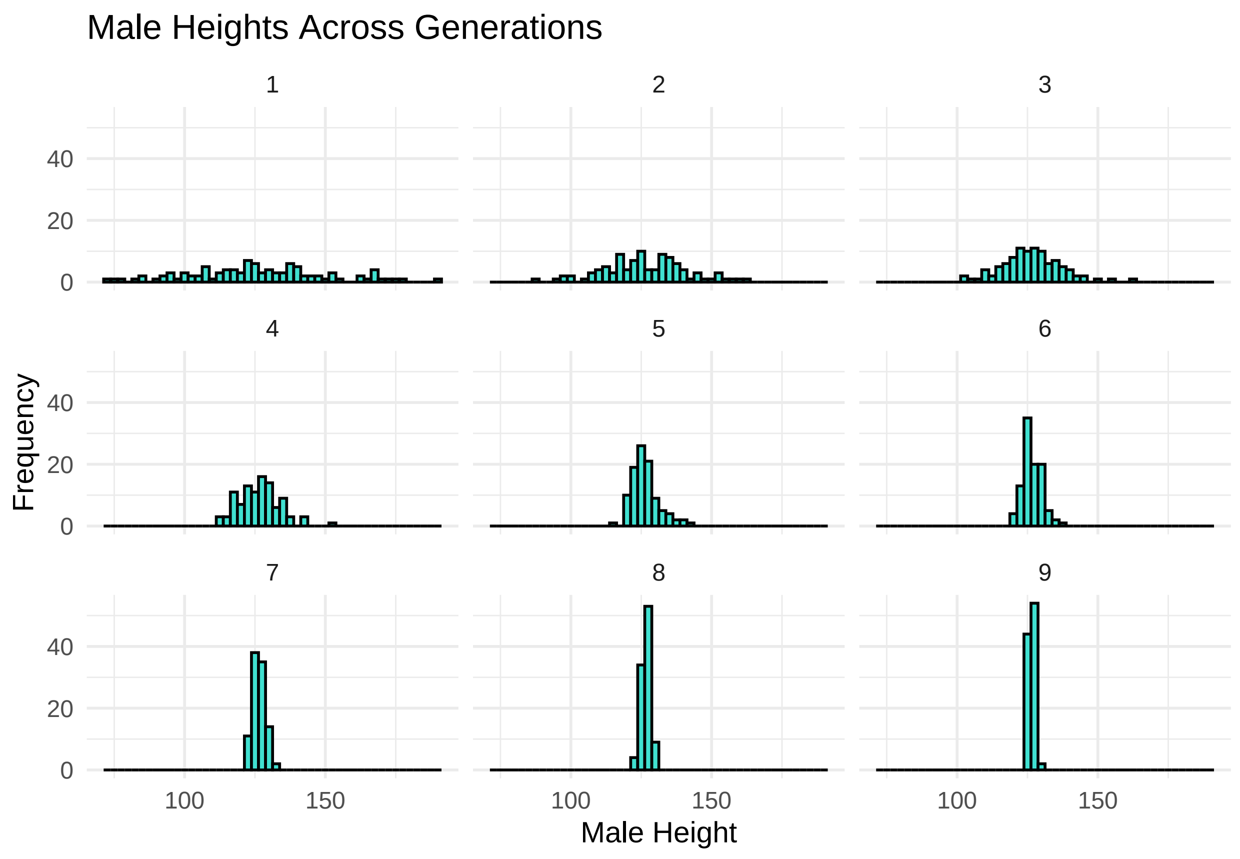
```
##          M        F
## 1 139.5480 139.5480
## 2 126.8168 126.8168
## 3 128.4576 128.4576
## 4 120.1116 120.1116
## 5 142.6801 142.6801
## 6 128.3991 128.3991
```

## (c)

Use the above function to generate nine generations, then use `ggplot2` to facet histograms to plot the distribution of male heights in each generation. This is called regression to the mean. *(Hint: Instead of using `facet_grid`, you will need to use `facet_wrap(, nrow = 3)`, in order to create $3 \times 3$ histograms.)*

```r
# me-ow
library(purrr)

genMGen <- function(df, nGen) {
  # Create a list of generations by iterating over 1:nGen
  generations <- map(1:nGen, function(i) {
    if (i > 1) {
      df <<- genNextGen(df)  # Update df for each generation
    }
    df$M
  })

  # Combine into a data frame
  genDat <- data.frame(
    Generation = rep(1:nGen, each = nrow(df)),
    Male_Height = unlist(generations)
  )

  genDat
}

# Call the function with your parameters
nGenData <- genMGen(df = gen1, nGen = 9)
```

```r
library(ggplot2)

ggplot(data = nGenData,
     aes(x = Male_Height)) +
  # you wouldn't believe how to spell turquoise, would you?
  # I got it wrong like twice before checking with the ol' Google
  geom_histogram(binwidth = 2.5, color = "black", fill = "turquoise") +
  facet_wrap(~ Generation, nrow = 3) +
  labs(
    title = "Male Heights Across Generations",
    x = "Male Height",
    y = "Frequency"
  ) +
  theme_minimal()
```



## Q3:

Clustering Output Parsing

The `mmclustering` program available at http://math.univ-lille1.fr/~wicker/softwares.html (but *note*: you do not need to download or run any such program) provides a partitioning of observations into different groups. However, the output is provided in a form which makes it difficult to easily do further analysis in R. In this exercise, we will write a function that will take the output (from a given file) and write out the classification for each observation as a vector.

The files provided in `Iris1.out` and `Iris2.out` contain results from grouping the iris dataset into a certain number of categories. The file has the following lines:

- The first line in the file contains the number of clusters, indicated by the phrase **Number of clusters** followed by a space, a colon `:`, a space, and then an integer (indicating the number of clusters). For example:

  `Number of clusters : 4`

- The second line is a blank.

- The third line contains the id of the first cluster (always designated by 0) and starts with **Cluster** followed by a space, then `0`, followed by a space, semi-colon `;`, the word **size** followed by an equality sign `=` and an integer which denotes the size of the cluster. For example:

  `Cluster 0; size=37`

- The next number of lines (which should match the cluster size given in the third line) contain the observation indices that belong to that group. This group of indices ends with a blank line.

- The following lines repeat the same format for the second cluster (indicated by **Cluster 1**), and this process continues until all cluster memberships are listed.

## (a)

The objective here is to write a function which will read one of the files above and provide a vector of length equal to the sum of the cluster sizes, containing the group indicators of the observations in the total population.

To do this, we can use the `readLines` function to read the file line-by-line (each line will be read in as a character string). Then, we can use multiple string-matching methods to parse the first line to obtain the number of clusters. The second line is a blank line. The first line after each blank line contains the size of the cluster (with memberships following in the next lines). Again, we will use string-splitting and matching techniques (e.g., `strsplit`) to obtain the cluster size. The next lines are converted from character strings to integers. Write the above function.

```r
parseClusterFile <- function(file) {
  allLines <- readLines(file)
  groups <- cumsum(allLines == "")
  splitVec <- split(allLines[allLines != ""], groups[allLines != ""])

  # Initialize cluster assignments as NA
  clusterAssignment <<- rep(NA, length(allLines))

  # Process each cluster
  lapply(seq_along(splitVec), function(i) {
    elements <- splitVec[[i]]
    elements <- elements[!grepl("[A-Za-z]", elements)]
    elements <- as.numeric(elements)

    clusterAssignment[elements] <<- i - 1
  })

  return(as.vector(na.omit(clusterAssignment)))
}
```

```r
string_list <- c(
  "Gautham", "I", "beg", "of", "you,",
  "have", "mercy", "upon", "me", "for", "committing", "the", "grave", "sin", "of", "writing", "a", "for
  "forgive", "me", "my", "sins", "of", "not", "believing", "us", "to", "be", "rational,",
  "and", "forgive", "me", "most", "of", "all", "for", "not", "having", "yet", "learned", "how", "to", "
)

for (word in string_list) {
  print(word)
}
```

```
## [1] "Gautham"
## [1] "I"
## [1] "beg"
## [1] "of"
## [1] "you,"
## [1] "have"
## [1] "mercy"
## [1] "upon"
## [1] "me"
## [1] "for"
## [1] "committing"
## [1] "the"
## [1] "grave"
## [1] "sin"
## [1] "of"
## [1] "writing"
## [1] "a"
## [1] "for"
## [1] "loop,"
## [1] "forgive"
## [1] "me"
## [1] "my"
## [1] "sins"
## [1] "of"
## [1] "not"
## [1] "believing"
## [1] "us"
## [1] "to"
## [1] "be"
## [1] "rational,"
## [1] "and"
## [1] "forgive"
## [1] "me"
## [1] "most"
## [1] "of"
## [1] "all"
## [1] "for"
## [1] "not"
## [1] "having"
## [1] "yet"
## [1] "learned"
## [1] "how"
## [1] "to"
## [1] "play"
## [1] "Bridge."
```

### (b)

Cross-tabulate the results of a call to the above function on the file `Iris1.out` and the file `Iris2.out`.

```
Iris1 <- parseClusterFile("Iris1.out")
Iris1
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
##  [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 1 3 1 1 1 1 3 1 1 1 1 1 1
## [112] 1 3 1 1 1 1 1 3 1 3 1 3 1 1 3 3 1 1 1 1 1 3 1 1 1 1 3 1 1 1 3 1 1 1 3 1 1
## [149] 3
```

```
Iris2 <- parseClusterFile("Iris2.out")
Iris2
```

```
##   [1] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
##  [38] 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1
```

```
irisTable <- table(Iris1, Iris2)
irisTable
```

```
##      Iris2
## Iris1  1  2
##     1 37  0
##     2  0 49
##     3 63  0
```

## Q4:

Multivariate Normal Sampling and Standardization

Let $X \sim N_3(\mu, \Sigma)$, where $\mu = \left(\frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}, \frac{1}{\sqrt{3}}\right)'$ and $\Sigma$ is a $3 \times 3$ diagonal matrix with diagonal elements 1 and off-diagonal elements $\rho$. Consider standardizing $X$ to get $Y = \frac{X}{\|X\|}$.

(Estimating parameters in the general scenario, for large $p$, and factorial structure of $\Sigma$ where we specify $\Sigma = \Lambda\Lambda' + \Psi$, with the basics first introduced in Stat 5010, was part of the Fall 2020 dissertation of Fan Dai from Iowa State University.)

### (a)

Write a function in R which generates a sample of $Y$ of size $n$ as outlined above.

```
library(MASS)

generate_Y_sample <- function(n, rho) {
  mu <- rep(1 / sqrt(3), 3)
  sigma <- matrix(rho, nrow = 3, ncol = 3)
  diag(sigma) <- 1

  X <- mvrnorm(n,
               mu = mu,
               Sigma = sigma)
  # Y <- t(apply(X = X,
  #              MARGIN = 1,
  #              FUN = function(x) {x / sqrt(sum(x^2))}
  #              ))

  # Calculate the norms (sqrt of sum of squares) for each row
  row_norms <- sqrt(rowSums(X^2))
  # Use sweep to divide each row by its corresponding norm
  Y <- sweep(X, MARGIN = 1, STATS = row_norms, FUN = "/")

  Y
}

set.seed(43)
Y_sample <- generate_Y_sample(n = 100,
                              rho = 0.5)
head(Y_sample)
```

```
##              [,1]       [,2]        [,3]
## [1,] -0.065044313  0.9312034  0.35864962
## [2,]  0.640310585  0.5401334  0.54613022
## [3,]  0.002601037  0.9947306  0.10249053
## [4,]  0.864541069 -0.4962718 -0.07926585
## [5,]  0.213152542  0.3773388  0.90121111
## [6,]  0.169734847  0.3859364  0.90677625
```

## (b)

For different $\rho \in \{-0.5, -0.25, 0, 0.25, 0.5\}$, display the observations in three dimensions. You may use $n = 100$ as the sample size in each case. Comment.

```
library(rgl)
```

```
## Warning: package 'rgl' was built under R version 4.4.2
```

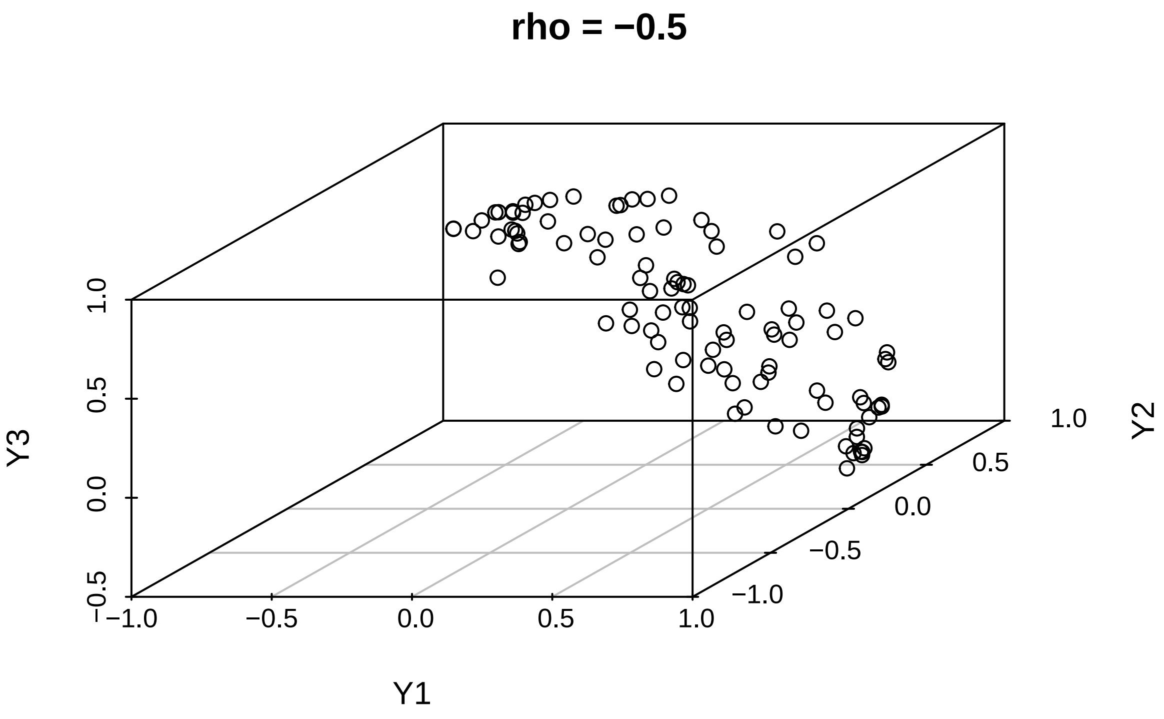
```
library(knitr)
library(rmarkdown)

# This makes pngs, but doesn't include them inline/indocument

plot_Y_for_single_rho <- function(n, rho) {
  Y <- generate_Y_sample(n, rho)
  open3d()
  plot3d(x = Y[, 1],
         y = Y[, 2],
```

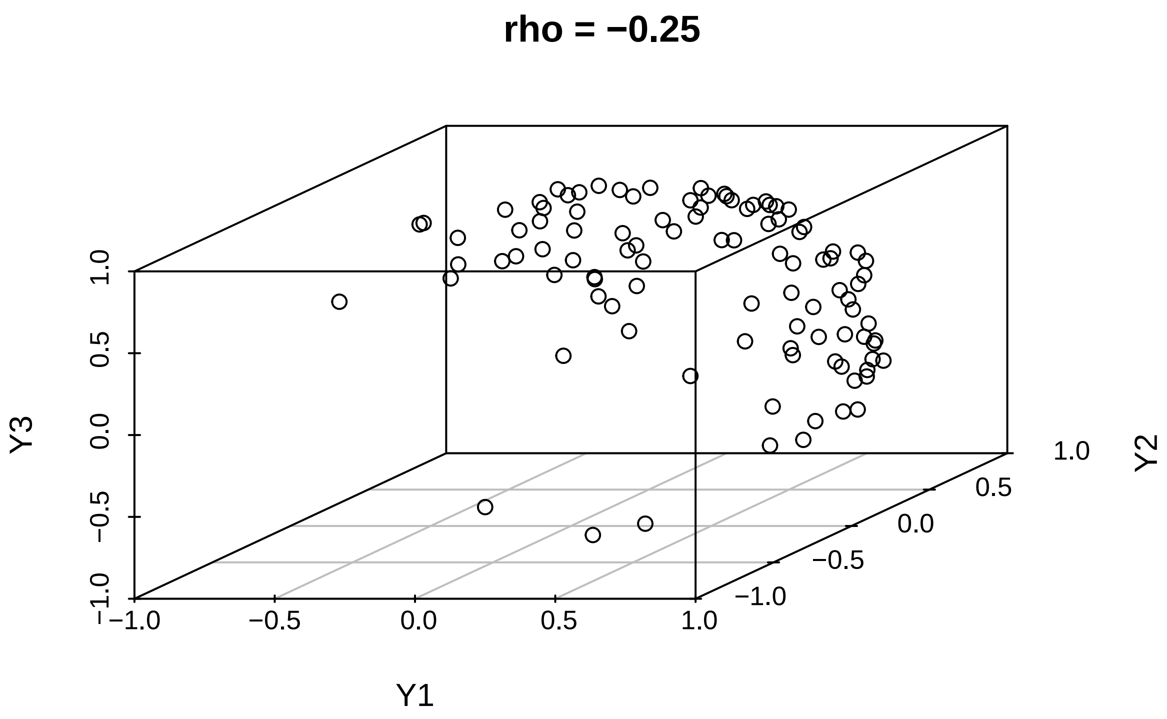
```r
         z = Y[, 3],
         col = "black",
         type = "s",
         size = 0.5,
         xlab = "Y1",
         ylab = "Y2",
         zlab = "Y3",
         main = paste("rho =", rho))

  rglwidget()
}

rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
n <- 100

for (rho in rho_values) {
  plot_Y_for_single_rho(n, rho)
  # Sys.sleep(0.5)
}
```

```
## Warning in snapshot3d(scene = x, width = width, height = height): webshot =
## TRUE requires the webshot2 package and Chrome browser; using rgl.snapshot()
## instead

## Warning in snapshot3d(scene = x, width = width, height = height): webshot =
## TRUE requires the webshot2 package and Chrome browser; using rgl.snapshot()
## instead
## Warning in snapshot3d(scene = x, width = width, height = height): webshot =
## TRUE requires the webshot2 package and Chrome browser; using rgl.snapshot()
## instead
## Warning in snapshot3d(scene = x, width = width, height = height): webshot =
## TRUE requires the webshot2 package and Chrome browser; using rgl.snapshot()
## instead
## Warning in snapshot3d(scene = x, width = width, height = height): webshot =
## TRUE requires the webshot2 package and Chrome browser; using rgl.snapshot()
## instead
```

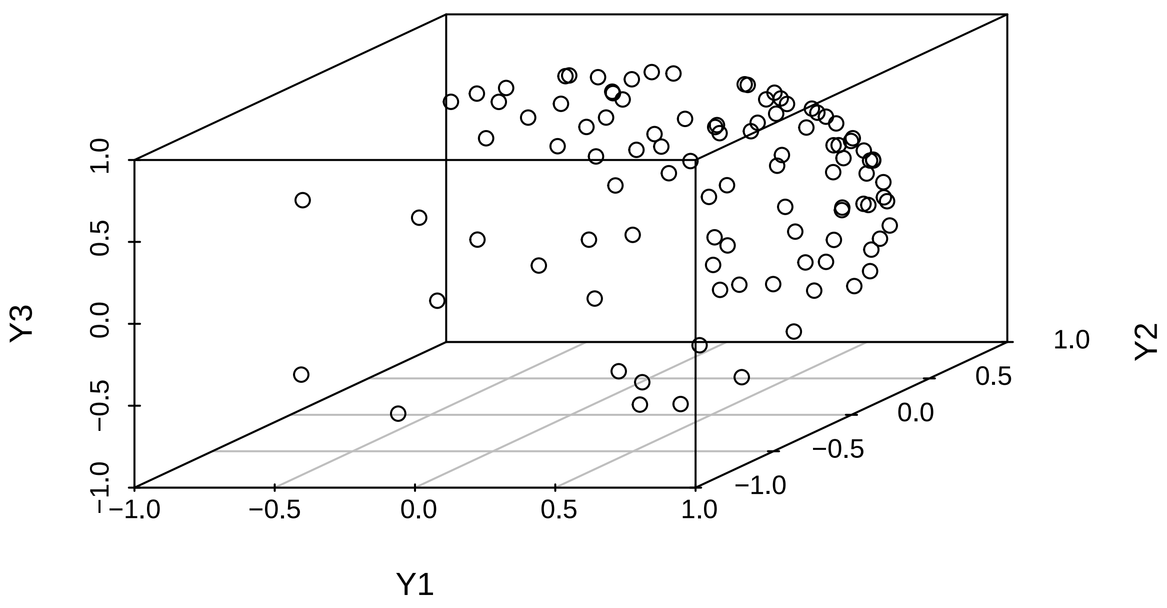
```r
library(scatterplot3d)

plot_Y_for_single_rho <- function(n, rho) {
  Y <- generate_Y_sample(n, rho)

  scatterplot3d(x = Y[,1],
                y = Y[,2],
                z = Y[,3],
                main = paste("rho =", rho),
                xlab = "Y1",
                ylab = "Y2",
                zlab = "Y3")
}

rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
n <- 100
```

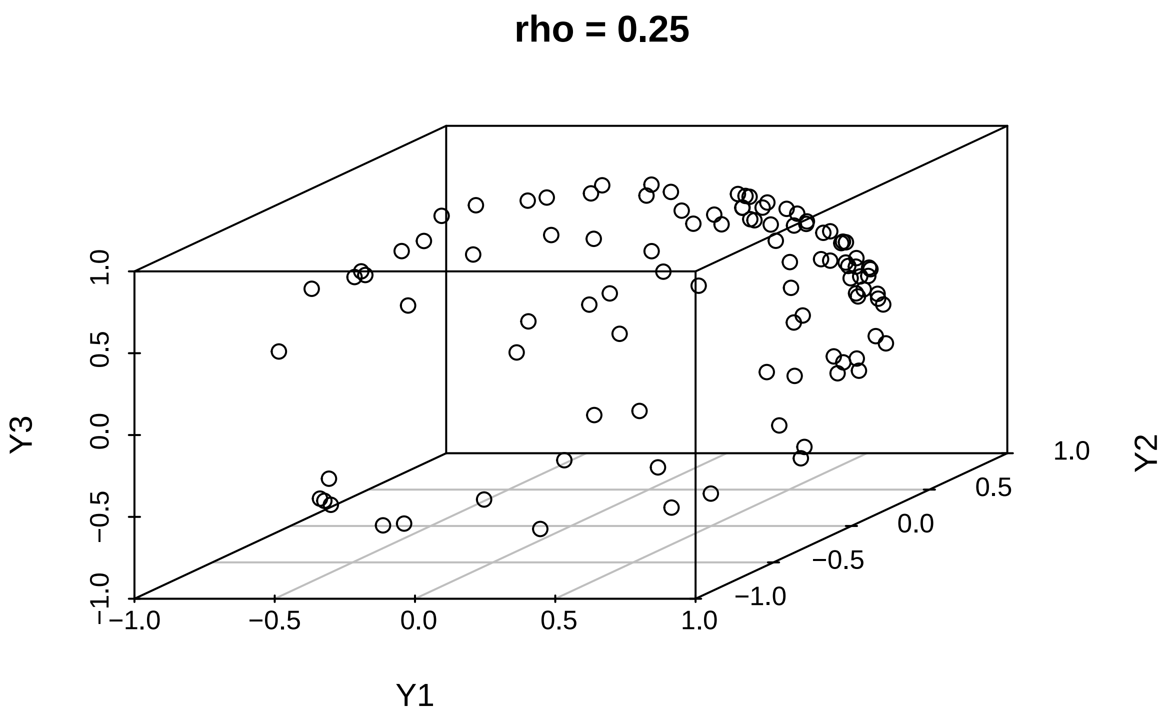
```r
for (rho in rho_values) {
  plot_Y_for_single_rho(n, rho)
  # Sys.sleep(0.5)
}
```

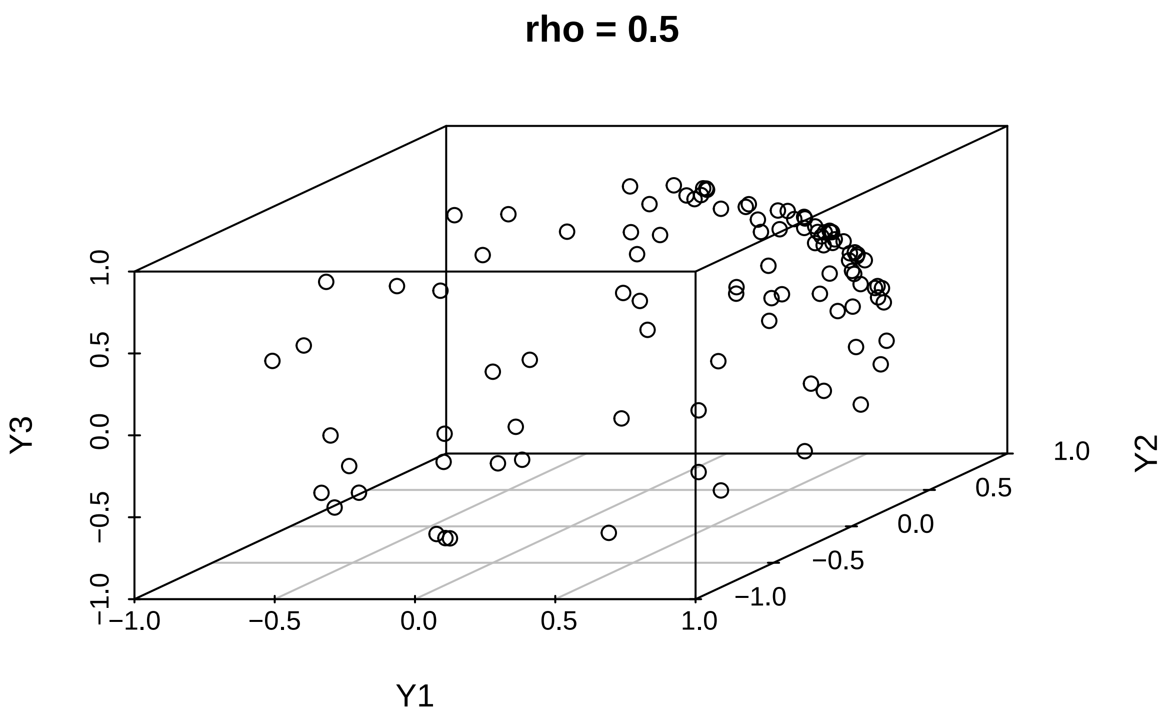



rho = −0.25
Y3
Y1
Y2
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.0
0.5
0.0
−0.5
−1.0

rho = 0




rho = 0.25
Y1
Y2
Y3
−1.0
−0.5
0.0
0.5
1.0



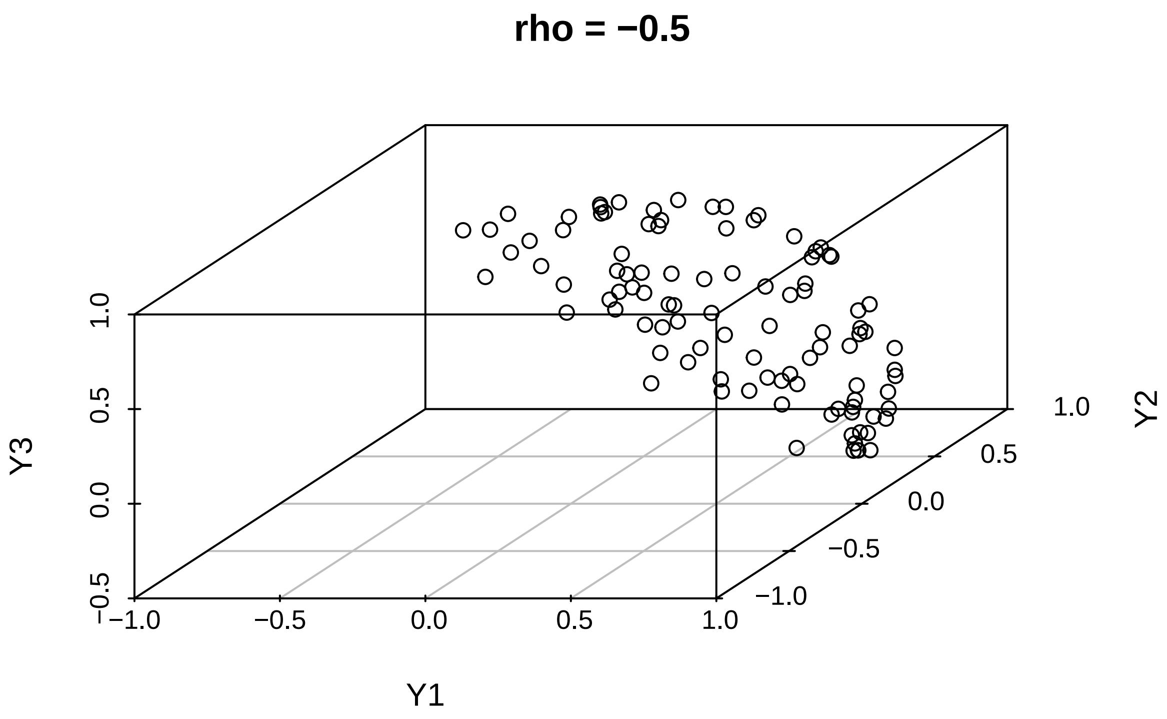
Comment: Boy, sure is a bit difficult to compare these plots, amiright?

But for serious, I'm replotting again for some perspective.

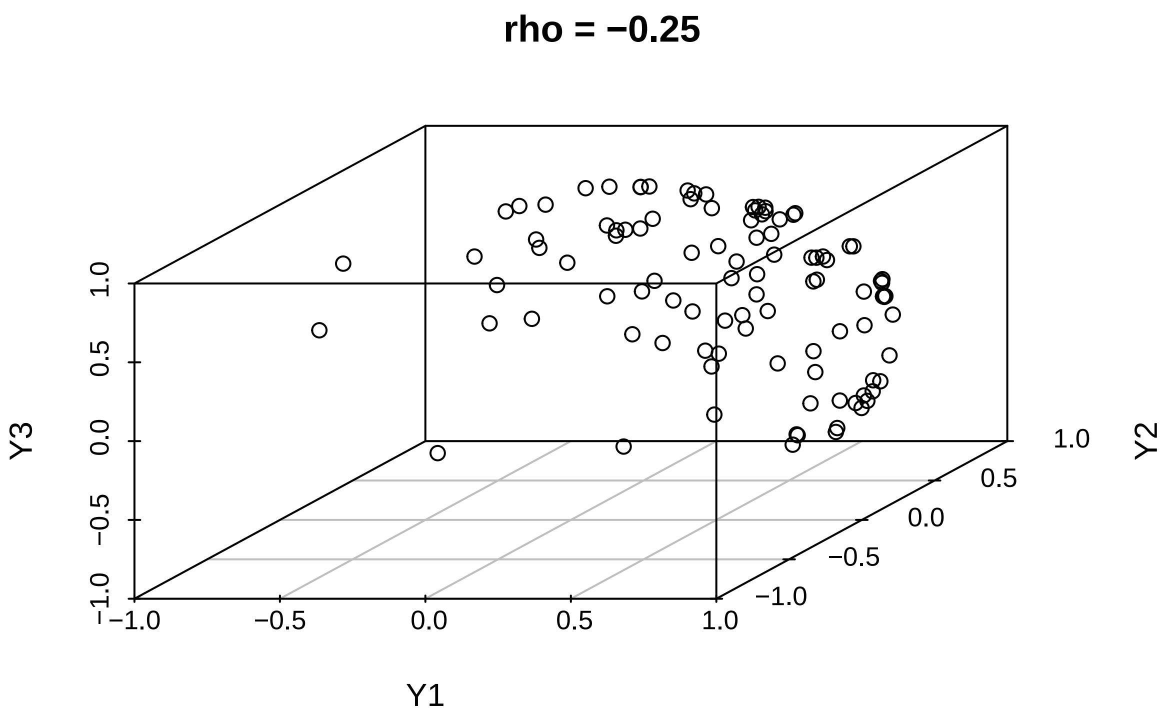
```r
library(scatterplot3d)

plot_Y_for_single_rho <- function(n, rho, angle = 45, phi = 20) {
  Y <- generate_Y_sample(n, rho)

  scatterplot3d(
    x = Y[,1],
    y = Y[,2],
    z = Y[,3],
    main = paste("rho =", rho),
    xlab = "Y1",
    ylab = "Y2",
    zlab = "Y3",
    angle = angle,    # horizontal angle
    phi = phi         # vertical angle
  )
}

rho_values <- c(-0.5, -0.25, 0, 0.25, 0.5)
n <- 100

# Plot with a fixed perspective different than default
for (rho in rho_values) {
  plot_Y_for_single_rho(n, rho, angle = 45, phi = 20)
  # Sys.sleep(0.5)
}
```

```
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
```

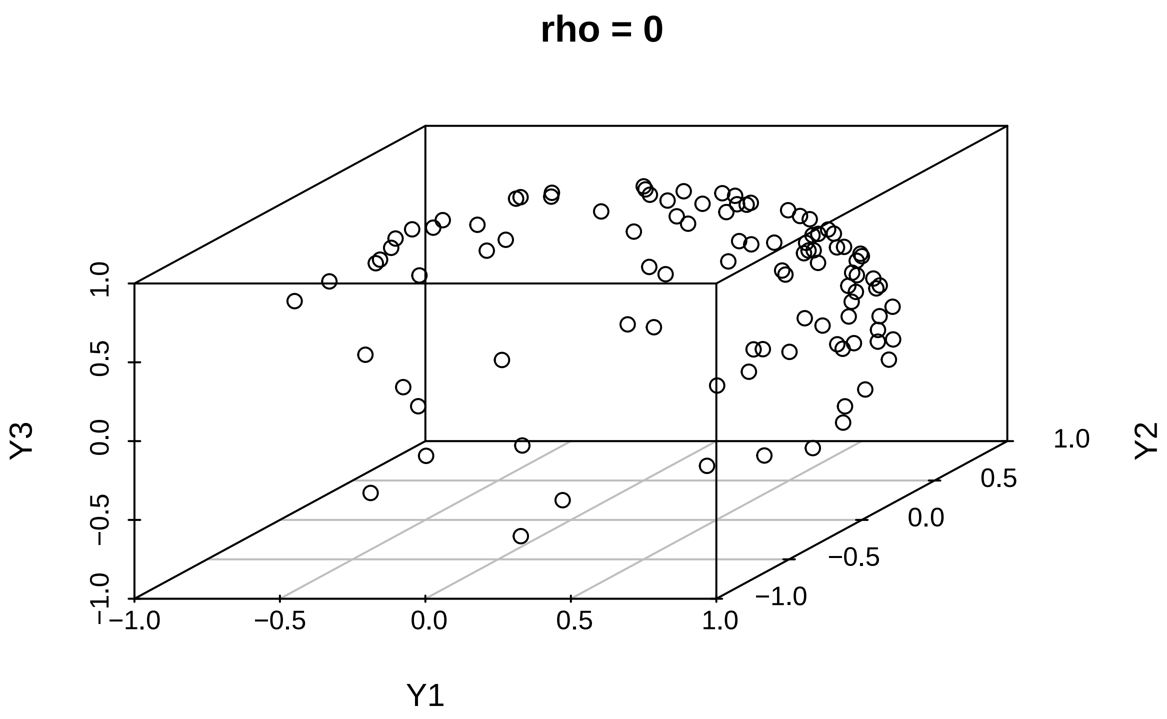
```
## Warning in plot.xy(xy.coords(x, y), type = type, ...): "phi" is not a graphical
## parameter
```



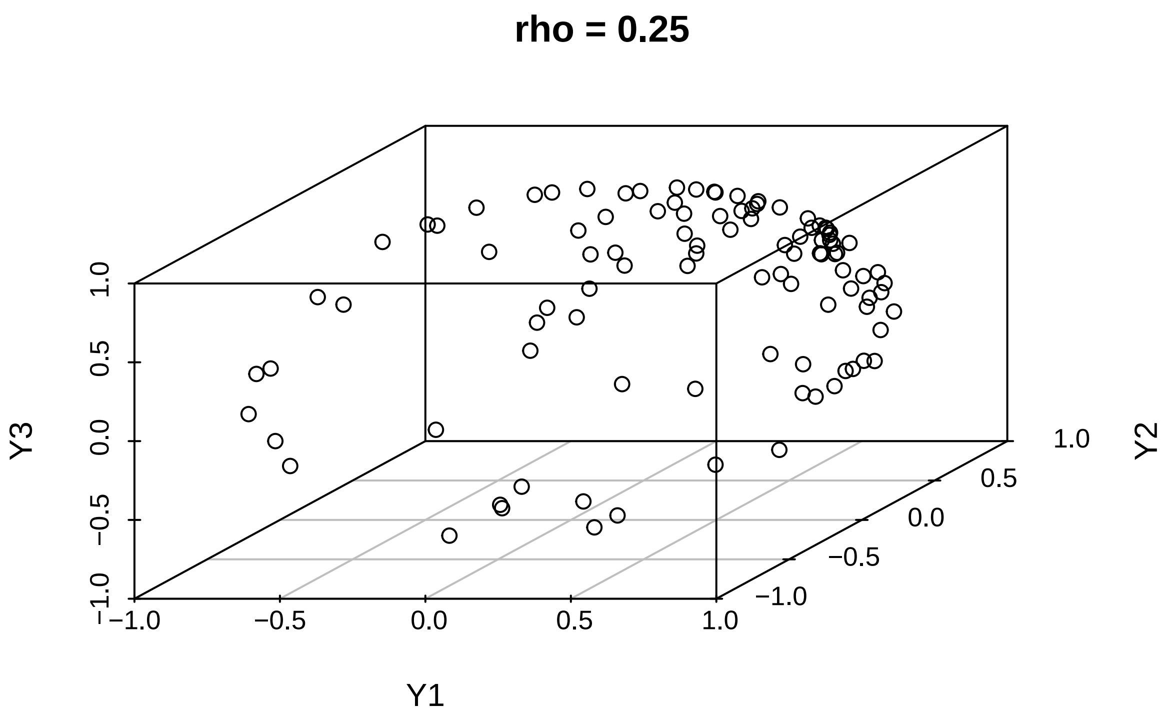
```
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
```



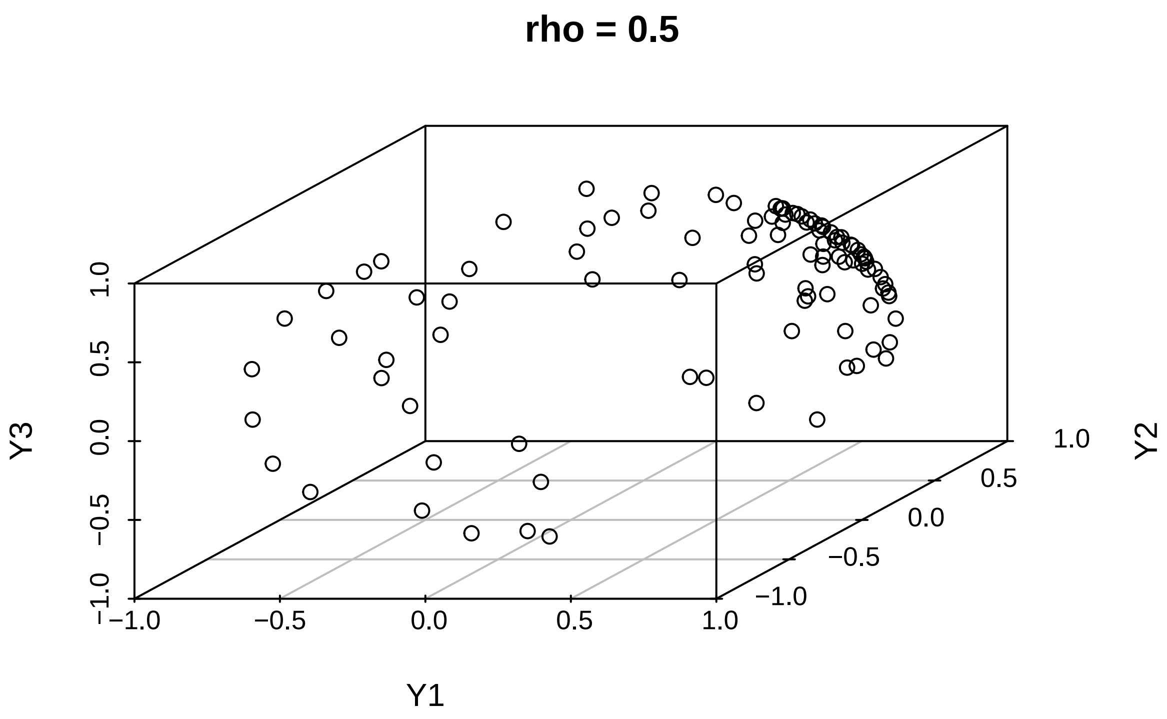
```
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
```



```
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
```



```
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
## Warning in title(main, sub, ...): "phi" is not a graphical parameter
```



Lower values of Rho (correlation) tend to exhibit a greater spread/dispersion of observations/values. By contrast we see something more akin to a 3-dimensional diagonal line. Positive rho values tend to cluster about positive values (x, y, z), whereas negative rho values tend to cluster about opposite ends of the quadrants, i.e. exhibiting positive-negative-positive, negative-positive-negative, etc. combinations of pairs for observations.