HW8

2024-11-06

Q1

Write a function which takes 2 arguments n and k which are positive integers. It should return the nXn matrix:

$$\begin{pmatrix} k & 1 & 0 & 0 & \dots & 0 & 0 \\ 1 & k & 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & k & 1 & \dots & 0 & 0 \\ 0 & 0 & 1 & k & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & k & 1 \\ 0 & 0 & 0 & 0 & \dots & 1 & k \end{pmatrix}$$

Call the function defined above for n = 6 and k = 5, and provide the matrix you obtain.

```r
create_matrix <- function(n, k) {
  mat <- matrix(0, n, n)

  diag(mat) <- k

  for (i in 1:(n-1)) {
    mat[i, i+1] <- 1
    mat[i+1, i] <- 1
  }

  return(mat)
}

matrix_result <- create_matrix(6, 5)
matrix_result
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    5    1    0    0    0    0
## [2,]    1    5    1    0    0    0
## [3,]    0    1    5    1    0    0
## [4,]    0    0    1    5    1    0
## [5,]    0    0    0    1    5    1
## [6,]    0    0    0    0    1    5
```

```r
create_matrix <- function(n, k) {
  # Create an n x n matrix filled with zeros
  mat <- matrix(0, n, n)
  # Fill the diagonal with k
  diag(mat) <- k

  # Fill the diagonals above and below the main diagonal with 1
  mat[row(mat) == col(mat) + 1] <- 1
  mat[row(mat) == col(mat) - 1] <- 1

  return(mat)
}

# Call the function for n = 6 and k = 5
matrix_result <- create_matrix(6, 5)
matrix_result
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    5    1    0    0    0    0
## [2,]    1    5    1    0    0    0
## [3,]    0    1    5    1    0    0
## [4,]    0    0    1    5    1    0
## [5,]    0    0    0    1    5    1
## [6,]    0    0    0    0    1    5
```

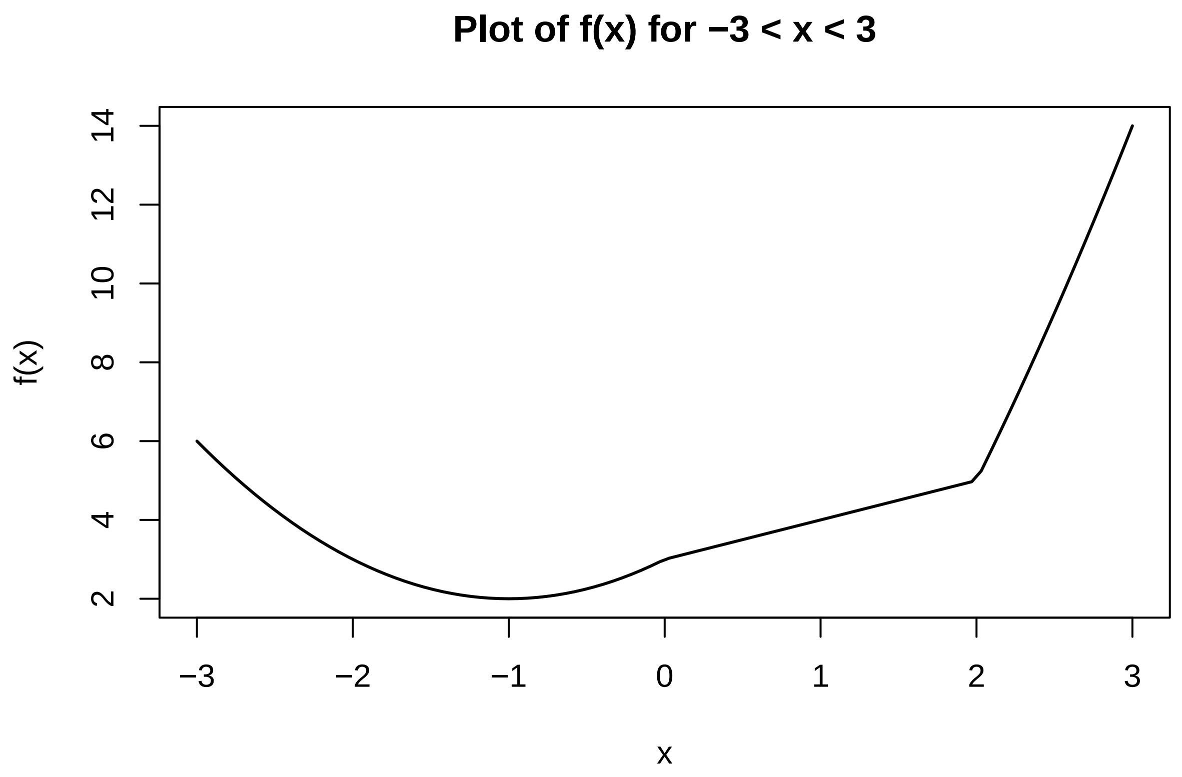
Q2

Consider the continuous function

$$f(x) = \begin{cases} x^2 + 2x + 3 & \text{if } x < 0 \\ x + 3 & \text{if } 0 \leq x < 2 \\ x^2 + 4x - 7 & \text{if } 2 \leq x \end{cases}$$

Write a function tmpFn which takes a single argument xVec. The function should return the vector of values of the function f(x) evaluated at the values xVec. Plot the function f(x) for -3 < x < 3.

```r
tmpFn <- function(xVec) {
  result <- numeric(length(xVec))

  result[xVec < 0] <- xVec[xVec < 0]^2 + 2 * xVec[xVec < 0] + 3
  result[xVec >= 0 & xVec < 2] <- xVec[xVec >= 0 & xVec < 2] + 3
  result[xVec >= 2] <- xVec[xVec >= 2]^2 + 4 * xVec[xVec >= 2] - 7

  return(result)
}


xValues <- seq(-3, 3, length.out = 100)
yValues <- tmpFn(xValues)

plot(x = xValues, y = yValues,
     type = "l",
     col = "black",
     lwd = 1.5,
     xlab = "x",
     ylab = "f(x)",
     main = "Plot of f(x) for -3 < x < 3")
```


Plot of f(x) for −3 < x < 3
f(x)
x
2
4
6
8
10
12
14
−3
−2
−1
0
1
2
3

Q3

Greatest common divisor of two integers The greatest common divisor (gcd) of two integers m and n can be calculated using Euclid's Algorithm: Divide m by n. If the remainder is zero, the gcd is n. If not, divide n by the remainder. If the remainder is zero, then the previous remainder is the gcd. If not, continue dividing the remainder into previous remainder until a remainder of zero is obtained. The gcd is the value of the last nonzero remainder. Write a function gcd(m,n) using a while loop to find the gcd of two integers m and n.

```r
gcd <- function(m, n) {
  m <- abs(m)
  n <- abs(n)

  while (n != 0) {
    remainder <- m %% n
    m <- n
    n <- remainder
  }

  return(m)
}
```

```r
gcd(4*7, 4*4*4*7)
```

```
## [1] 28
```

```r
gcd(47, 93)
```

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

Q4

eQTL mapping. (The following problem was suggested by Professor Dan Nettleton.) Write a function order.matrix which takes in a matrix x and returns a matrix containing the row and column indices of the sorted values of x. Test this function on a 4X3 matrix of independent χ_1^2 pseudo-random deviates.

```
order.matrix <- function(x) {
  sorted_indices <- order(x)

  row_indices <- (sorted_indices - 1) %% nrow(x) + 1
  col_indices <- (sorted_indices - 1) %/% nrow(x) + 1

  result <- cbind(row_indices, col_indices)
  return(result)
}

set.seed(37)
test_matrix <- matrix(rchisq(4 * 3, df = 1), nrow = 4, ncol = 3)
dim(test_matrix)
```

```
## [1] 4 3
```

```
test_matrix
```

```
##            [,1]       [,2]       [,3]
## [1,] 0.84692934 0.05563524 2.57108739
## [2,] 1.18006741 1.86200588 0.01183871
## [3,] 0.34755917 2.01569730 0.07067748
## [4,] 0.06079836 2.13410150 0.01434852
```

```
indices <- order.matrix(test_matrix)
indices
```

```
##       row_indices col_indices
##  [1,]           2           3
##  [2,]           4           3
##  [3,]           1           2
##  [4,]           4           1
##  [5,]           3           3
##  [6,]           3           1
##  [7,]           1           1
##  [8,]           2           1
##  [9,]           2           2
## [10,]           3           2
## [11,]           4           2
## [12,]           1           3
```

```
dim(indices)
```

```
## [1] 12  2
```

Q5

Polar representation of a number. Let $x \in \mathbb{R}^p$. The polar respresentation of $\mathbf{x} = (x_1, x_2, ..., x_p)$ is given by $(R, \theta_1, \theta_2, ..., \theta_{p-1})$, where:

$$\begin{aligned}
x_1 &= R\cos\theta_1 \\
x_2 &= R\sin\theta_1\cos\theta_2 \\
x_3 &= R\sin\theta_1\sin\theta_2\cos\theta_3 \\
\dots &= \dots \\
x_{p-1} &= R\prod_{i=1}^{p-2}\sin\theta_i\cos\theta_{p-1} \\
x_p &= R\prod_{i=1}^{p-1}\sin\theta_i,
\end{aligned}$$

where $0 \le R < \infty, 0 \le \theta_1 < 2\pi$ and $0 \le \theta_i < \pi$ i = 2, 3, . . . , p - 1.

(a)

Write a function polaroid which takes in an arbitrary p-dimensional vector $\mathbf{x}$ and provides its polar representation as a vector, with the first element as R and the remainder being $\theta_1, \theta_2, \dots, \theta_{p-1}$.

```r
polaroid <- function(x) {
  R <- sqrt(sum(x^2))

  p <- length(x)
  theta <- numeric(p - 1)

  if (R > 0) {
    theta[1] <- atan2(x[2], x[1])
    if (theta[1] < 0) {
      theta[1] <- theta[1] + 2 * pi
    }

    for (i in 2:(p - 1)) {
      numerator <- sqrt(sum(x[(i + 1):p]^2))
      denominator <- sqrt(sum(x[i:p]^2))
      theta[i] <- acos(numerator / denominator)
    }
  }

  return(c(R, theta))
}

x <- c(1, 2, 3)
polar_representation <- polaroid(x)
polar_representation
```

```
## [1] 3.7416574 1.1071487 0.5880026
```

```r
polaroid <- function(x) {
  R <- sqrt(sum(x^2))

  if (R == 0) return(c(R))

  theta1 <- atan2(x[2], x[1])
  if (theta1 < 0) {
    theta1 <- theta1 + 2 * pi
  }

  p <- length(x)
  if (p > 2) {
    numerators <- sqrt(rev(cumsum(rev(x[-c(1, 2)]^2))))
    denominators <- sqrt(cumsum(rev(x[-1]^2)))
    theta_rest <- acos(numerators / denominators)
  } else {
    theta_rest <- numeric(0)
  }

  return(c(R, theta1, theta_rest))
}

x <- c(1, 2, 3)
polar_representation <- polaroid(x)
print(polar_representation)
```

```
## [1] 3.7416574 1.1071487 0.0000000 0.5880026
```

(b)

Write a function normalize which takes in a matrix and returns its normalized form: i.e., the matrix with rows scaled such that the sum of squares of each row is equal to 1.

```r
normalize <- function(x) {
  row_norms <- sqrt(rowSums(x^2))

  normalized_matrix <- x / row_norms

  normalized_matrix[is.nan(normalized_matrix)] <- 0

  return(normalized_matrix)
}
```

```r
set.seed(37)
test_matrix <- matrix(rnorm(12), nrow = 4, ncol = 3)
test_matrix
```

```
##            [,1]       [,2]        [,3]
## [1,]  0.1247540 -0.8283492  0.85595441
## [2,]  0.3820746 -0.3327136  0.21599549
## [3,]  0.5792428 -0.1921595 -0.37770210
## [4,] -0.2937481  1.3629827  0.03869354
```

```r
rowSums(test_matrix^2)
```

```
## [1] 1.4343838 0.3033334 0.5151063 1.9455071
```

```r
normalized_matrix <- normalize(test_matrix)
normalized_matrix
```

```
##            [,1]       [,2]       [,3]
## [1,]  0.1041650 -0.6916410  0.7146904
## [2,]  0.6937261 -0.6041022  0.3921792
## [3,]  0.8070718 -0.2677401 -0.5262607
## [4,] -0.2106002  0.9771786  0.0277410
```

```r
rowSums(normalized_matrix^2)
```

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

(c)

Obtain a 1000X5 matrix **y** of $N(0,1)$ pseudo-random deviates. Use apply and normalize to obtain the normalized values. Call this matrix **z**. We test whether the columns of **z** are uniform on $U(-1,1)$. One may test whether a sample $x \sim U(-1,1)$ using ks.test(x, "punif", min=-1, max=1) where punif represents the cumulative distribution function of the uniform over range $(-1,1)$. Summarize your results.

```r
set.seed(37)

y <- matrix(rnorm(1000 * 5), nrow = 1000, ncol = 5)

normalize <- function(x) {
  row_norms <- sqrt(rowSums(x^2))
  normalized_matrix <- sweep(x, 1, row_norms, FUN = "/")
  normalized_matrix[is.nan(normalized_matrix)] <- 0
  return(normalized_matrix)
}

z <- apply(X = y,
           MARGIN = 2,
           FUN = function(col) normalize(matrix(col, ncol = 1))
           )

ks_tests <- apply(X = z,
                  MARGIN = 2,
                  FUN = function(column) ks.test(column, "punif", min = -1, max = 1)
                  )
```

```
## Warning in ks.test.default(column, "punif", min = -1, max = 1): ties should not
## be present for the one-sample Kolmogorov-Smirnov test
## Warning in ks.test.default(column, "punif", min = -1, max = 1): ties should not
## be present for the one-sample Kolmogorov-Smirnov test
## Warning in ks.test.default(column, "punif", min = -1, max = 1): ties should not
## be present for the one-sample Kolmogorov-Smirnov test
## Warning in ks.test.default(column, "punif", min = -1, max = 1): ties should not
## be present for the one-sample Kolmogorov-Smirnov test
## Warning in ks.test.default(column, "punif", min = -1, max = 1): ties should not
## be present for the one-sample Kolmogorov-Smirnov test
```

```
ks_tests
```

```
## [[1]]
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  column
## D = 0.515, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## [[2]]
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  column
## D = 0.531, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## [[3]]
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  column
## D = 0.525, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## [[4]]
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  column
## D = 0.504, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## [[5]]
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  column
## D = 0.51, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

The KS test results strongly suggest that the columns of the normalized matrix z are not uniformly distributed on (-1,1). The deviations are significant, implying that the normalization did not transform the data into a uniform distribution over this range. Further investigation or a different transformation might be needed if the goal is to achieve a uniform distribution.

(d)

Obtain polar representations of **y** using your function polaroid and test whether $R^2 \sim \chi^2_5$ distribution. Provide a page of histograms or boxplots of $\theta_1, \theta_2, \theta_3, \theta_4$. Test whether these are from the uniform distributions on their respective ranges, i.e., $[0, 2\pi)$ for θ_1 and $[0, \pi)$ for $\theta_2, \theta_3, \theta_4$.

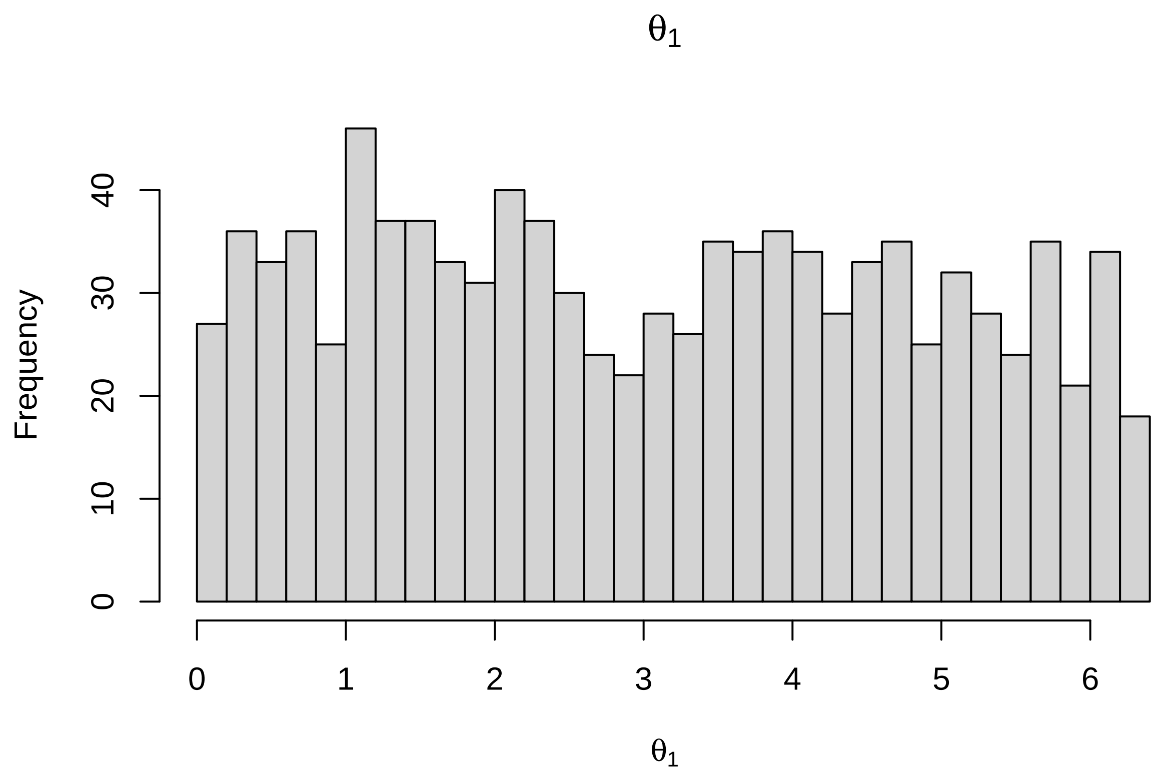
```r
set.seed(37)

polaroid <- function(x) {
  # Calculate R
  R <- sqrt(sum(x^2))

  # If R is zero, return R with no angles (undefined case)
  if (R == 0) return(c(R))

  # Calculate the first angle theta1 using atan2
  theta1 <- atan2(x[2], x[1])
  if (theta1 < 0) theta1 <- theta1 + 2 * pi

  p <- length(x)

  # Calculate the remaining angles, only if p > 2
  if (p > 2) {
    numerators <- sqrt(rev(cumsum(rev(x[-c(1, 2)]^2))))
    denominators <- sqrt(cumsum(rev(x[-1]^2))[-1])

    # Ensure lengths match for division
    if (length(numerators) != length(denominators)) {
      stop("Numerators and denominators lengths do not match.")
    }

    # Calculate theta_rest and clamp values to be in [-1, 1]
    cos_values <- numerators / denominators
    cos_values <- pmin(pmax(cos_values, -1), 1)  # Clamp values to [-1, 1]
    theta_rest <- acos(cos_values)
  } else {
    theta_rest <- numeric(0)  # No additional angles if p <= 2
  }

  # Return the polar representation
  return(c(R, theta1, theta_rest))
}

# Test the function on a matrix y
set.seed(123)
y <- matrix(rnorm(1000 * 5), nrow = 1000, ncol = 5)
polar_representation <- t(apply(y, 1, polaroid))

# Extract R values and theta values
R_values <- polar_representation[, 1]
```

```
theta_values <- if (ncol(polar_representation) > 1) polar_representation[, -1] else NULL
```

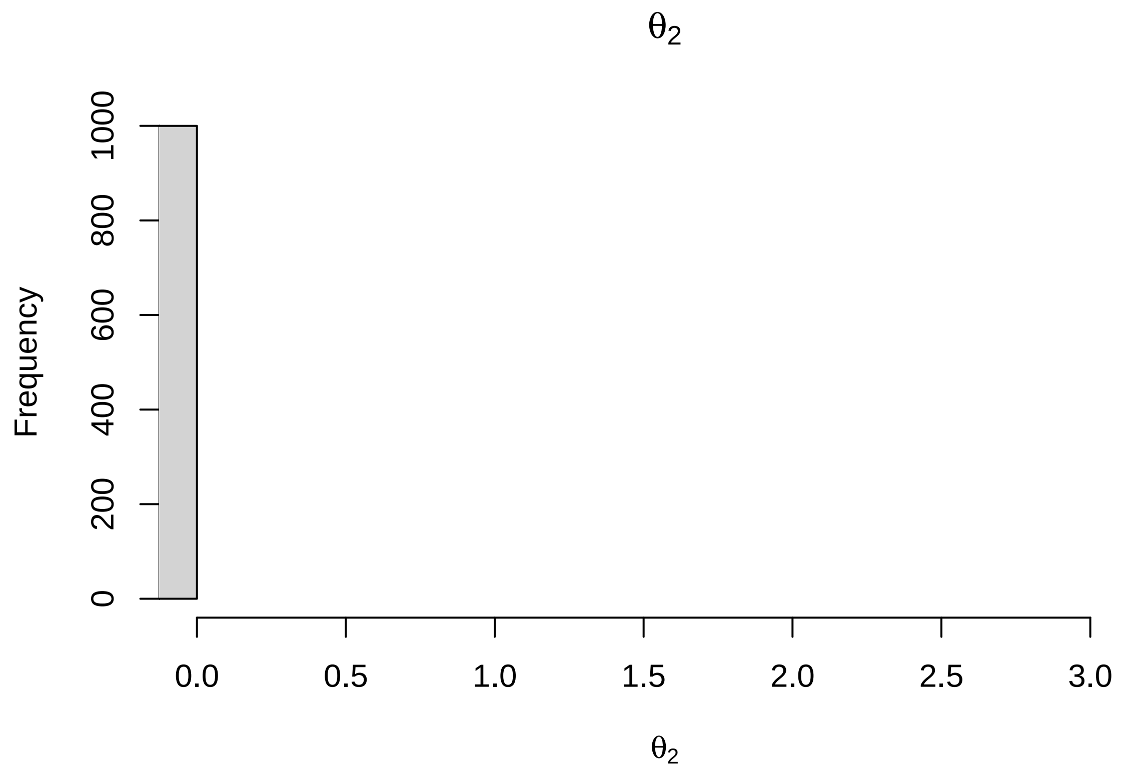
```
R_squared <- R_values^2

chisq_test <- ks.test(R_squared, "pchisq", df = 5)
chisq_test
```

```
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  R_squared
## D = 0.020785, p-value = 0.7807
## alternative hypothesis: two-sided
```

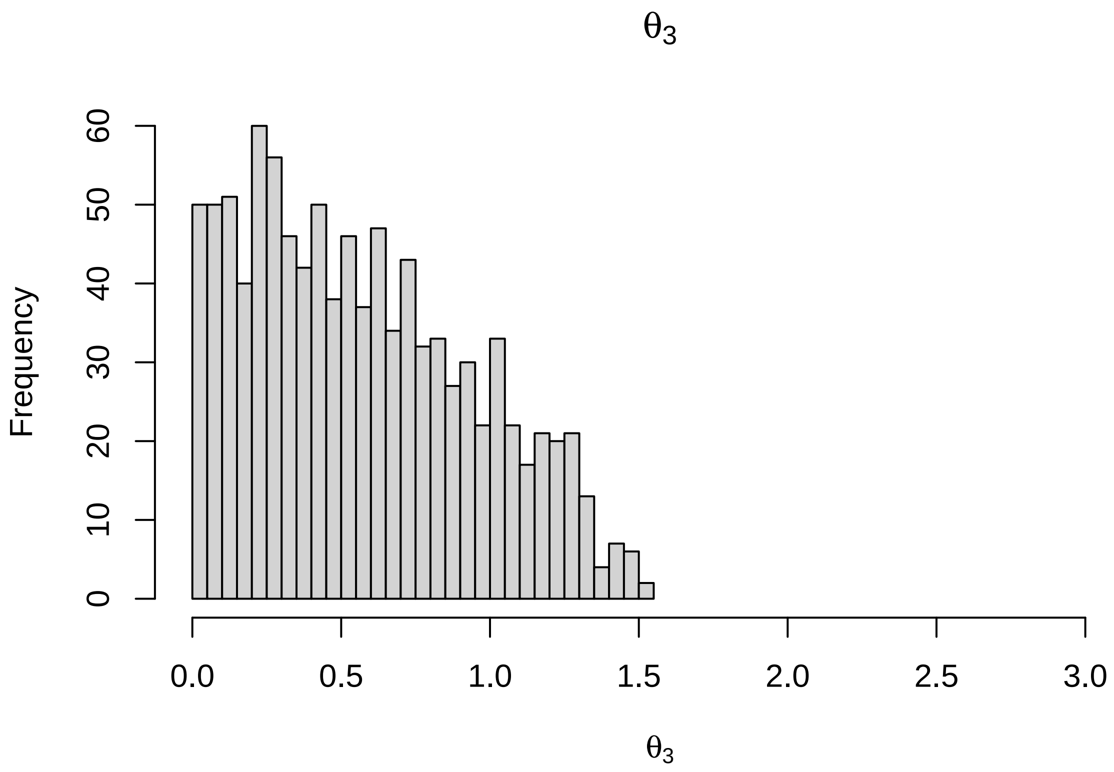
```
hist(theta_values[, 1], breaks = 30, main = expression(theta[1]), xlab = expression(theta[1]), xlim = c
```



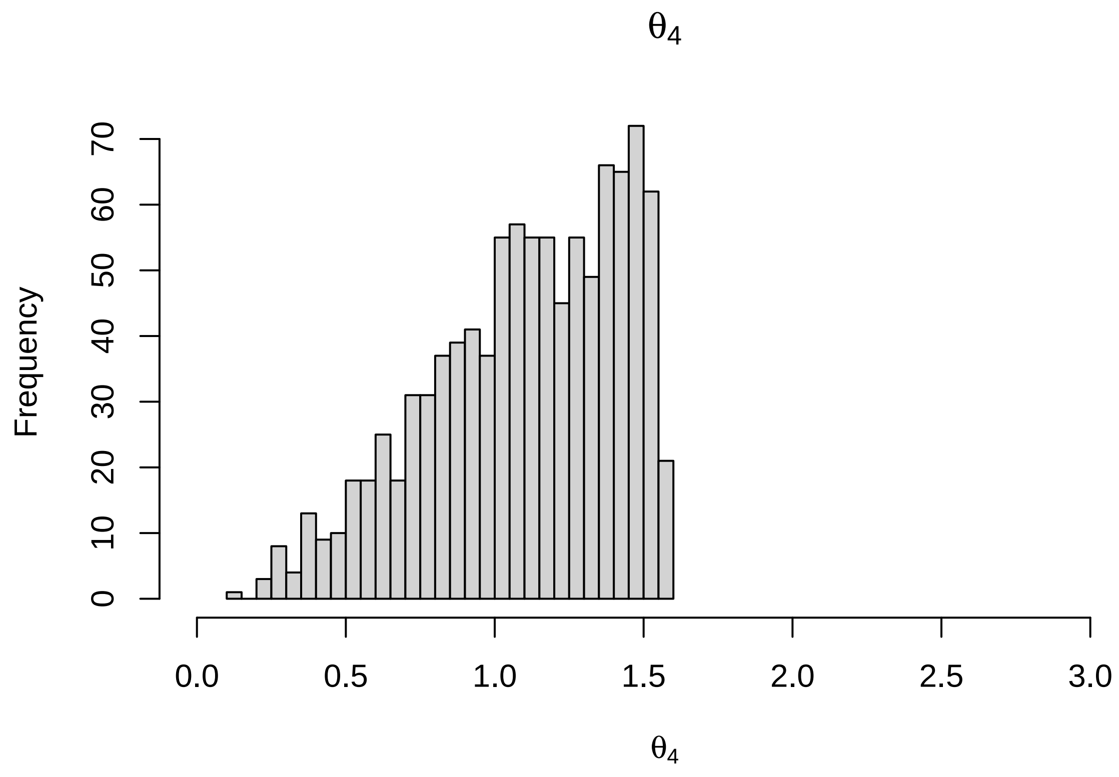
```
hist(theta_values[, 2], breaks = 30, main = expression(theta[2]), xlab = expression(theta[2]), xlim = c
```



```
hist(theta_values[, 3], breaks = 30, main = expression(theta[3]), xlab = expression(theta[3]), xlim = c
```



```
hist(theta_values[, 4], breaks = 30, main = expression(theta[4]), xlab = expression(theta[4]), xlim = c
```



```r
theta1_test <- ks.test(theta_values[, 1] / (2 * pi), "punif", min = 0, max = 1)

theta2_test <- ks.test(theta_values[, 2] / pi, "punif", min = 0, max = 1)
```

```
## Warning in ks.test.default(theta_values[, 2]/pi, "punif", min = 0, max = 1):
## ties should not be present for the one-sample Kolmogorov-Smirnov test
```

```r
theta3_test <- ks.test(theta_values[, 3] / pi, "punif", min = 0, max = 1)
theta4_test <- ks.test(theta_values[, 4] / pi, "punif", min = 0, max = 1)

list(theta1_test = theta1_test, theta2_test = theta2_test,
     theta3_test = theta3_test, theta4_test = theta4_test)
```

```
## $theta1_test
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  theta_values[, 1]/(2 * pi)
## D = 0.041297, p-value = 0.06602
## alternative hypothesis: two-sided
##
##
## $theta2_test
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
```

```
##
## data:  theta_values[, 2]/pi
## D = 1, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## $theta3_test
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  theta_values[, 3]/pi
## D = 0.5565, p-value < 2.2e-16
## alternative hypothesis: two-sided
##
##
## $theta4_test
##
##  Asymptotic one-sample Kolmogorov-Smirnov test
##
## data:  theta_values[, 4]/pi
## D = 0.50011, p-value < 2.2e-16
## alternative hypothesis: two-sided
```