Info 371 PS1

1 Outliers in Different Distributions

1.1 Normal Distribution

Let's compute the variance of means: we pick n random numbers r times, and for each one we will calculate their mean. Afterwards, we'll see how the variance of means will change when we change n. For this example, we will have *n = 10* and *r = 1000*.

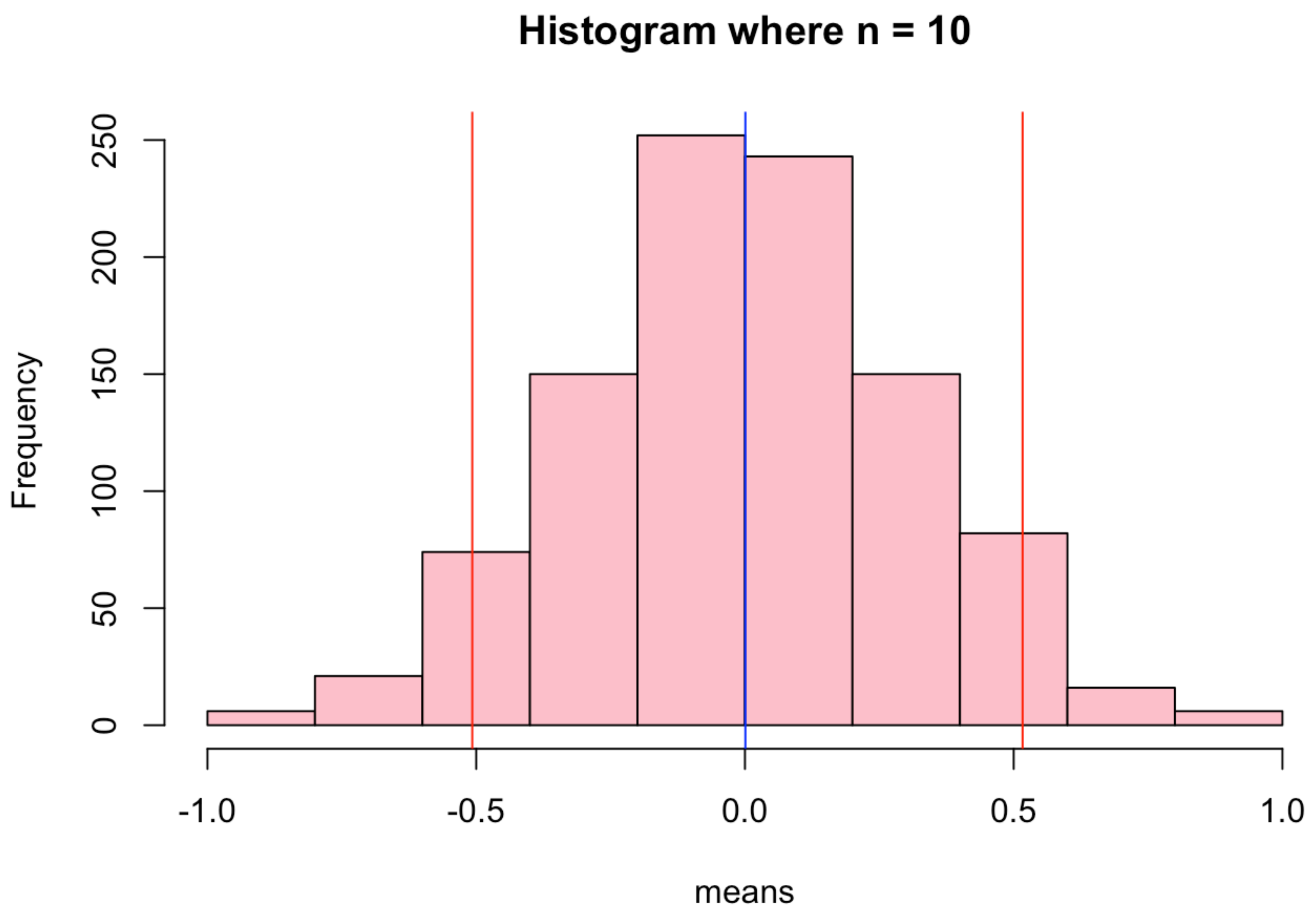
```
n <- 10
r <- 1000

means <- sapply(replicate(r, sapply(n, rnorm, simplify = FALSE)), mean)
```

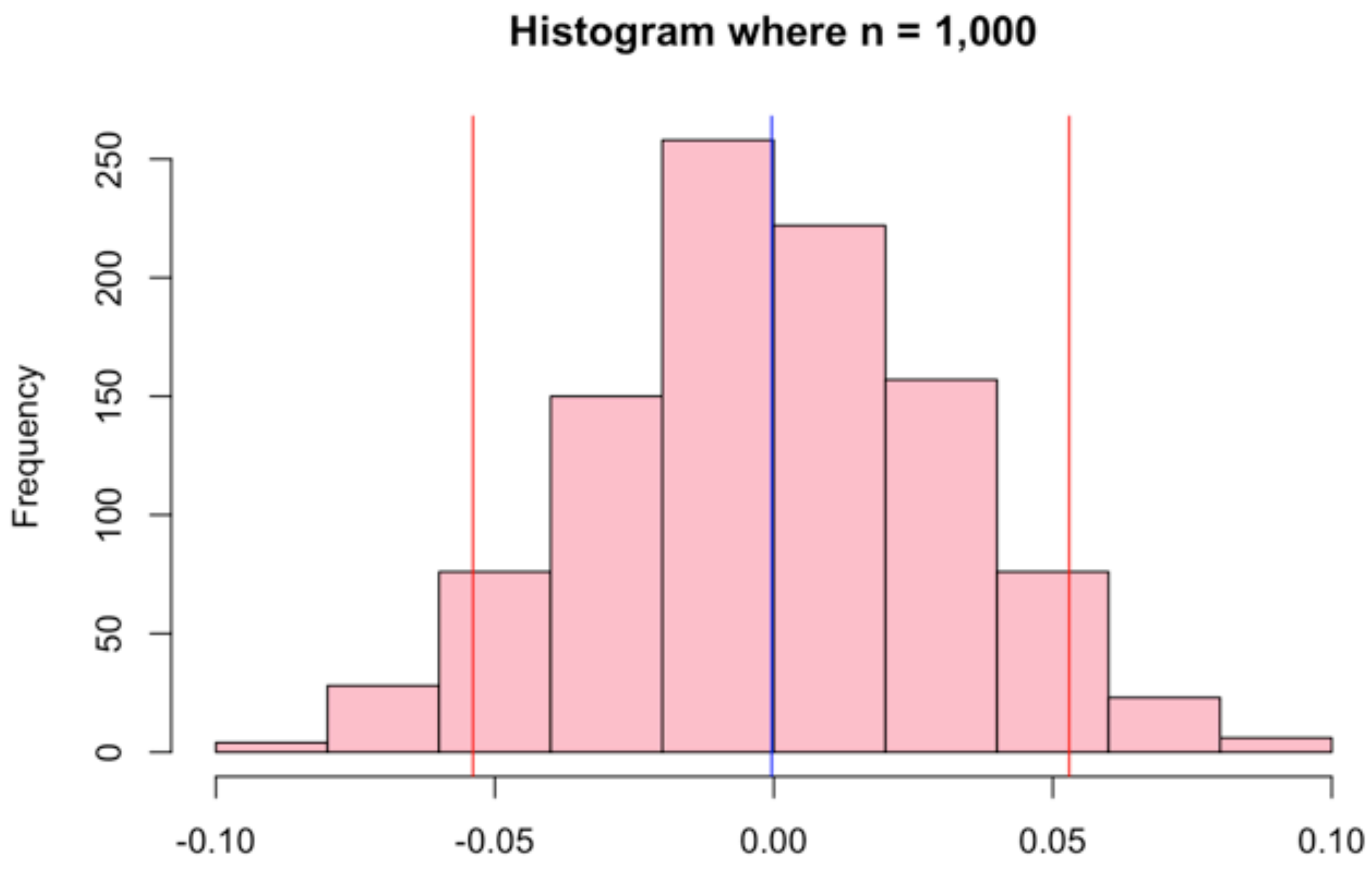
For *n = 10* and *r = 1000*, the mean of our sample of means is **7.610^{-4}**. We have a standard deviation of **0.30924** and a 95% confidence region of **[-0.50735, 0.51655]**.

Let's see the differences when we increase n. For *n = 1,000*, we have a mean of **-3.810^{-4}**, a standard deviation of **0.03167** and a 95% confidence region of **[-0.05393, 0.51655]**. We can start making comparisons by observing that the new mean is closer to 0. The new standard deviation is 9.76445 times smaller than the first standard deviation, approximately 10x smaller (and the variance is also much lower). Finally, our 95% confidence interval is narrower because our mean values are closer to 0.

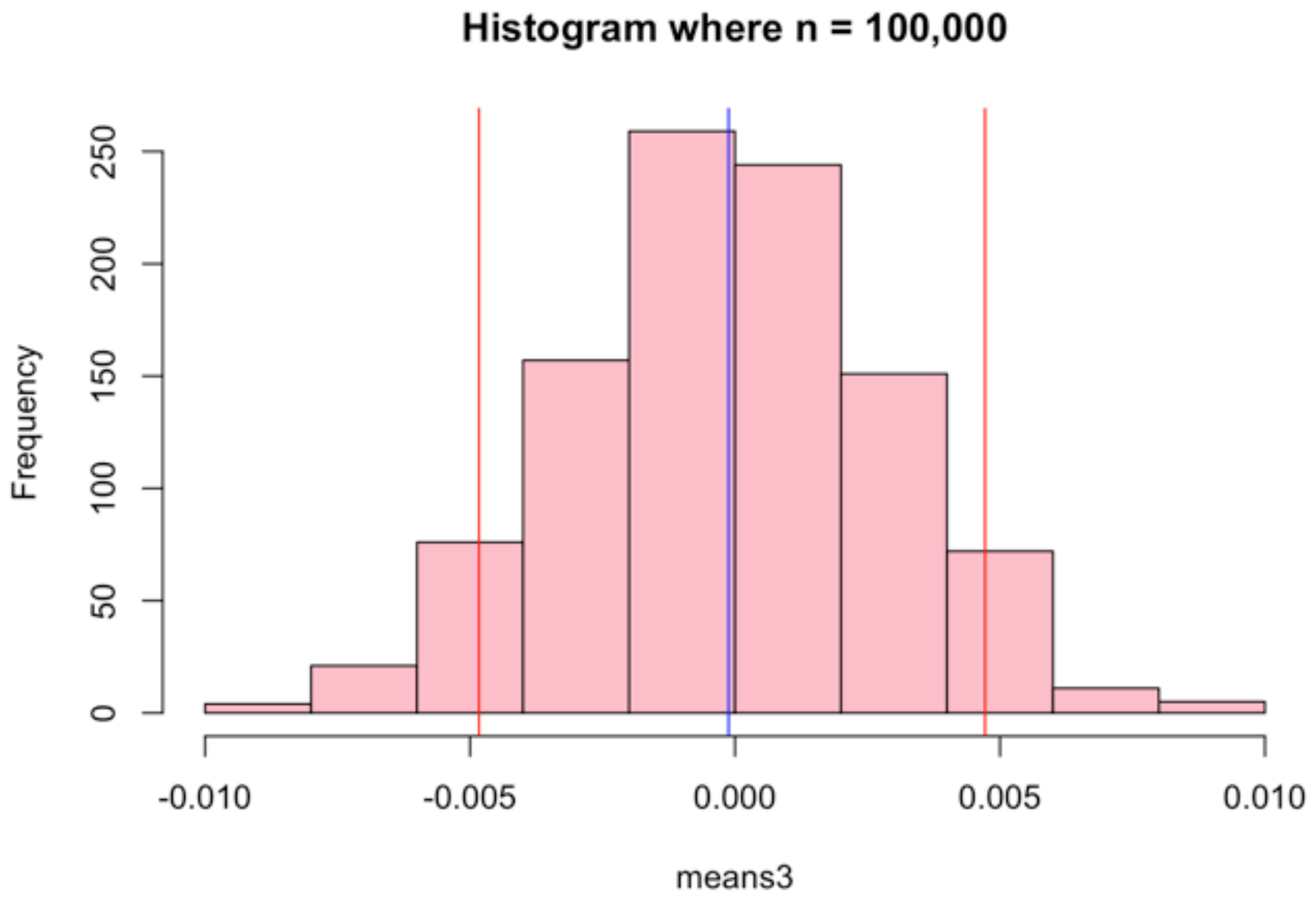
For *n = 100,000*, we have a mean of **-1.210^{-4}**, a standard deviation of **0.00297** and a 95% confidence region of **[-0.00483, 0.00472]**. Unsurprisingly, we see a decrease in the mean of means, standard deviation, and 95% confidence interval.

To visualize the differences in the three n sample sizes:


Histogram where n = 10
Frequency
0
50
100
150
200
250
-1.0
-0.5
0.0
0.5
1.0
means


Histogram where n = 1,000
Frequency
0
50
100
150
200
250
-0.10
-0.05
0.00
0.05
0.10

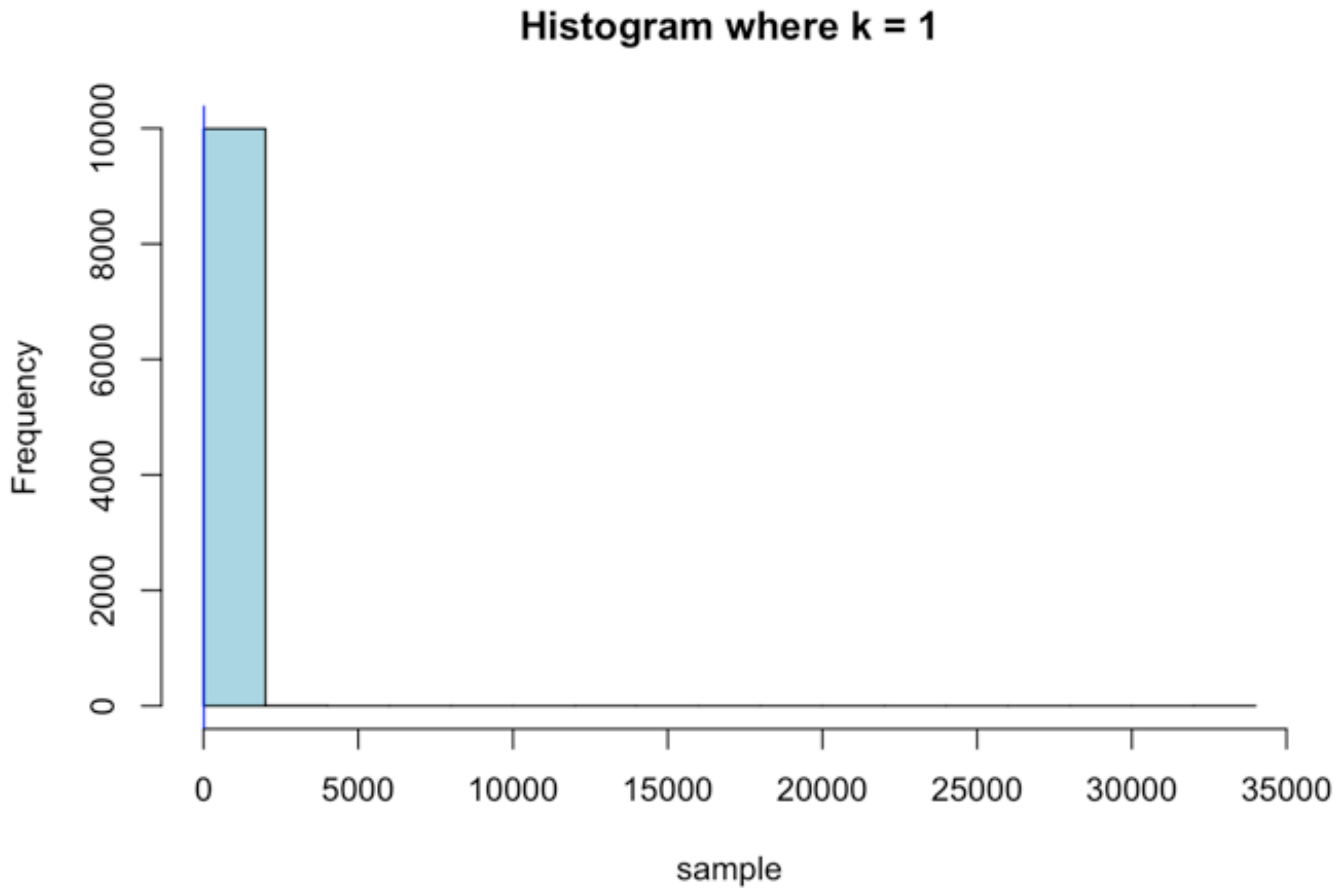
means2



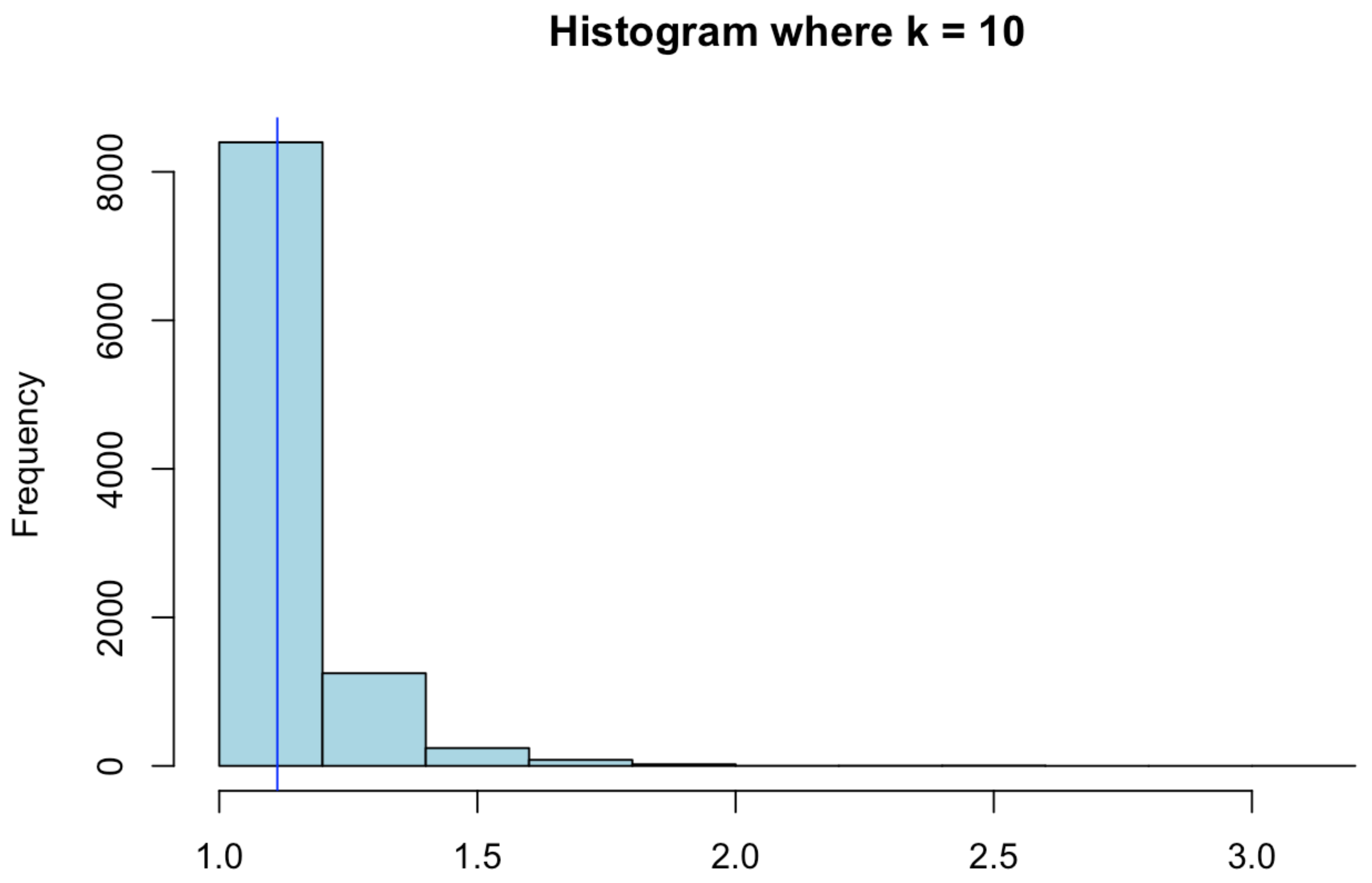
1.2 Pareto Distribution

Normal distribution has very nice properties, despite it's density function looking a little bit… nasty. But there are many things in our world that are not well approximated by normal distribution. Examples include human income, size of cities, links to webpages, popularity of actors, number of citations of researchers, size of wildfires… All of these are very inequal; there are cities that are enourmous, but most towns are small. Some researchers have hundreds of thousands of citations, but the majority has only a few. These type of outcomes can be described by Pareto distribution.

First, let's explore the Pareto Distribution looking at different values of k (shape). We can notice that the shape of the distributions are right-skewed.

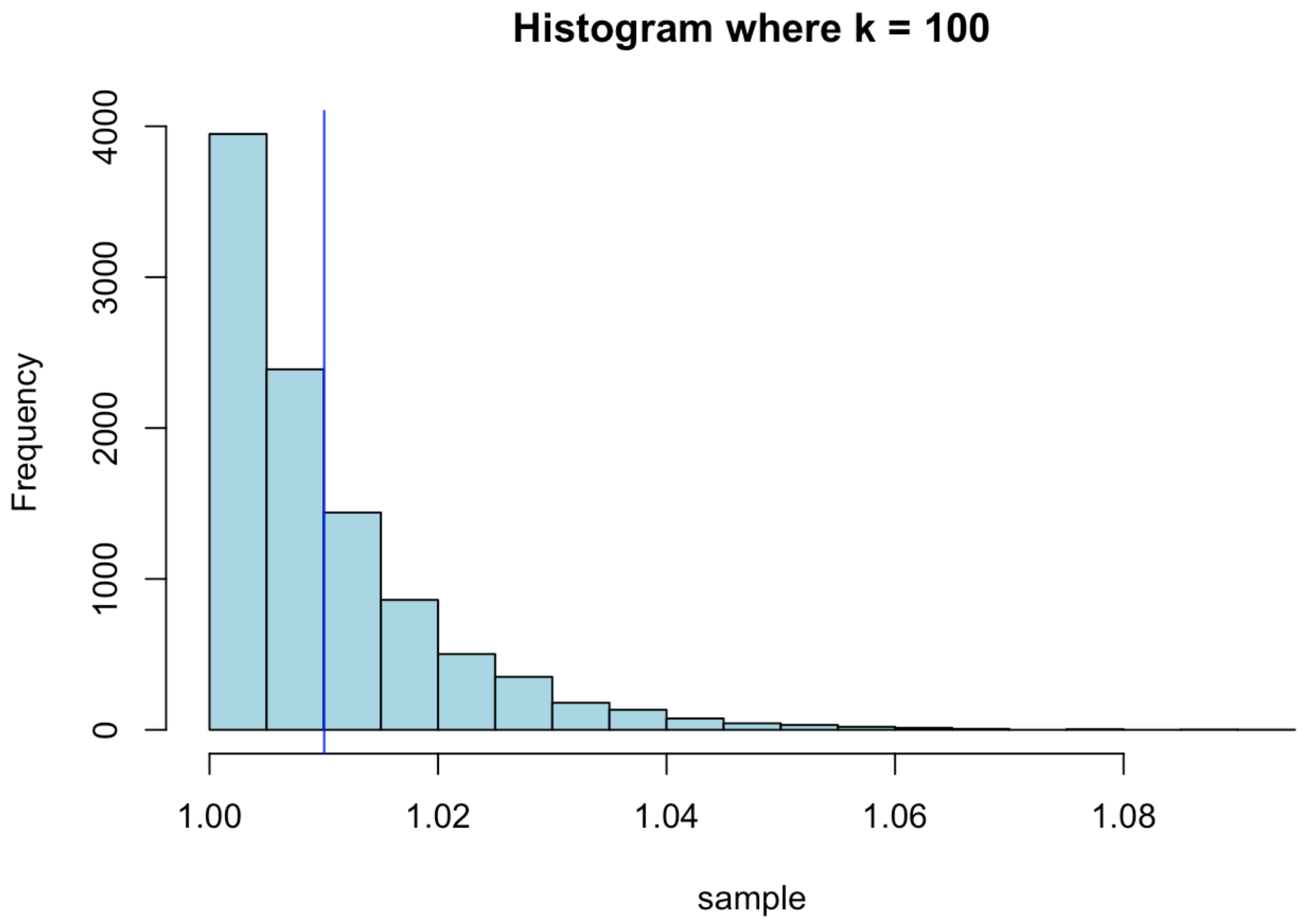


Histogram where k = 1
Frequency
0
2000
4000
6000
8000
10000
0
5000
10000
15000
20000
25000
30000
35000
sample



Histogram where k = 10
Frequency
0
2000
4000
6000
8000
1.0
1.5
2.0
2.5
3.0

sample



Now let’s explore the mean of pareto random numbers and how sample size affects this, as we did earlier. We will have repititions *r = 1,000*. Let’s first explore with *k = 0.5*, and then *k = 2*.

k = 0.5

Sample Size	Mean	Standard Deviation	95% CI
n = 10	1.018839810^{5}	2.904073310^{6}	[4.27, 4622.57]
n = 1000	4.262823310^{6}	7.597942710^{7}	[454.93, 4.216515410^{5}]
n = 100000	5.085717710^{7}	9.482335210^{8}	[4.02963510^{4}, 3.851166210^{7}]

k = 2

Sample Size	Mean	Standard Deviation	95% CI
n = 10	1.99874	0.8222	[1.34985, 3.23898]
n = 1000	1.99564	0.1014	[1.86828, 2.15359]

n = 100000	1.99992	0.01327	[1.98285, 2.01846]

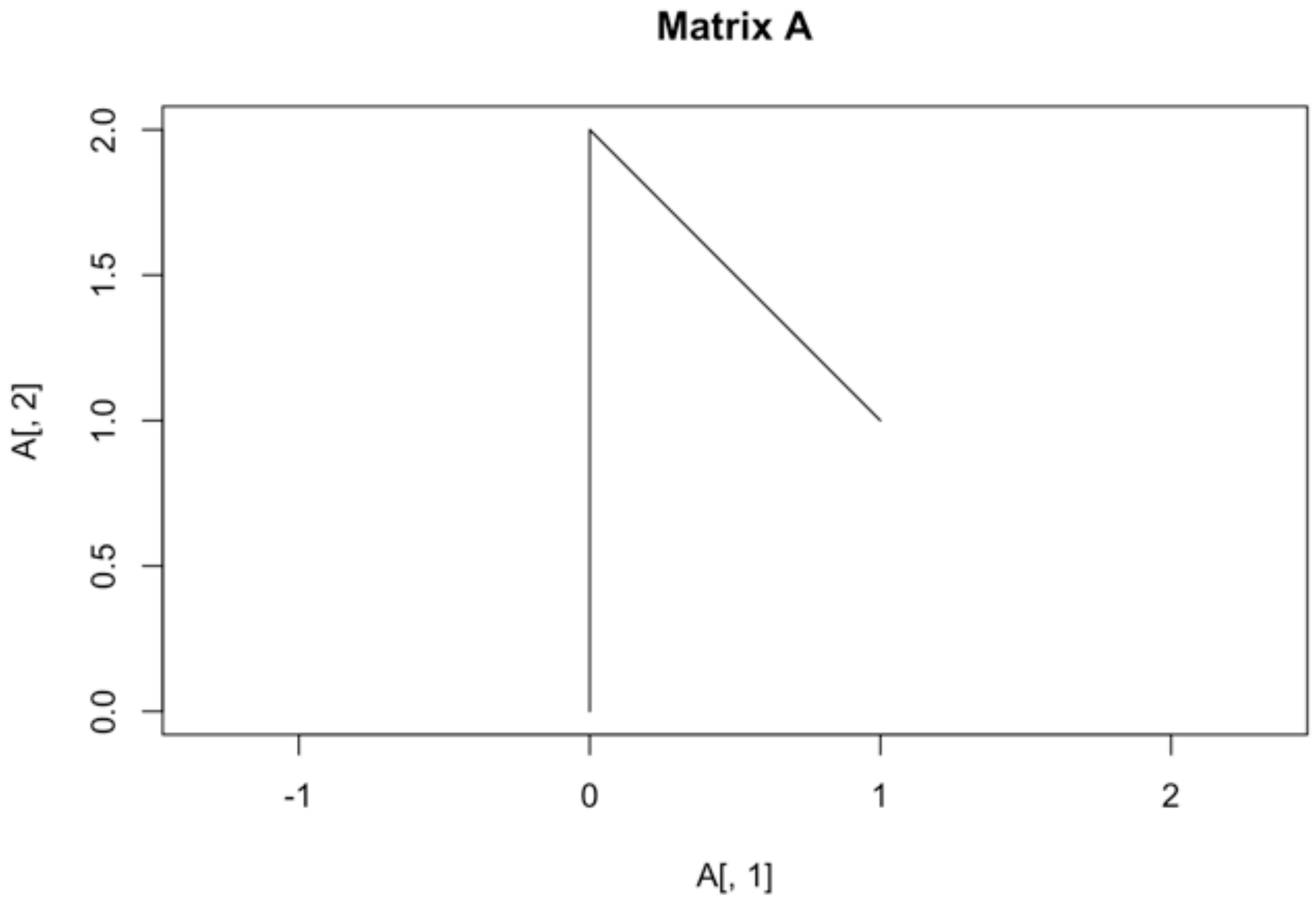
We see a big difference between pareto and normal distribution, but there's also a huge difference when comparing k<1 and k>1. In normal distribution, we noticed that as sample size increases, everything (mean of means, sd, and 95% CI) gets closer to 0. When *k < 1*, we see a pattern of these computations scaling up as n increases, very unlike our earlier patterns of changing relative to 0. But with *k > 1*, we see a pattern where the mean and standard deviation move closer to 0 just as normal distribution did. In the normal distribution and pareto distribution for *k > 1*, the CI narrows as sample size increases. However, the CI moved closer to 1 for pareto.

2 Linear Transformations of Images

2.1 Rotate Matrices

Demonstrating rotation of object matrix A:

```
##      [,1] [,2]
## [1,]    0    0
## [2,]    0    2
## [3,]    1    1
```

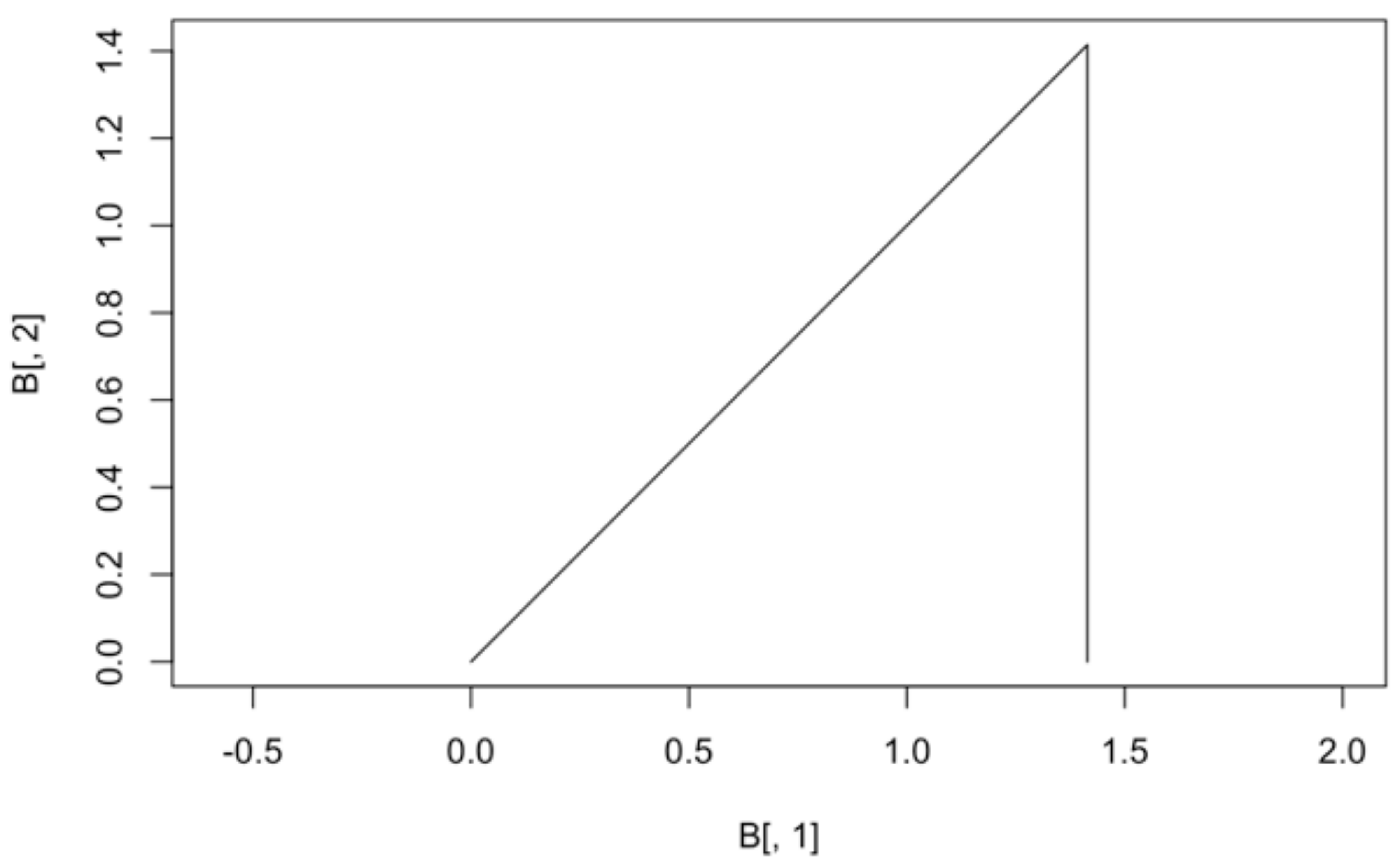


We can use the following function to create rotation matrix with a given alpha (angle) and then multiply this matrix to our object matrix for rotation

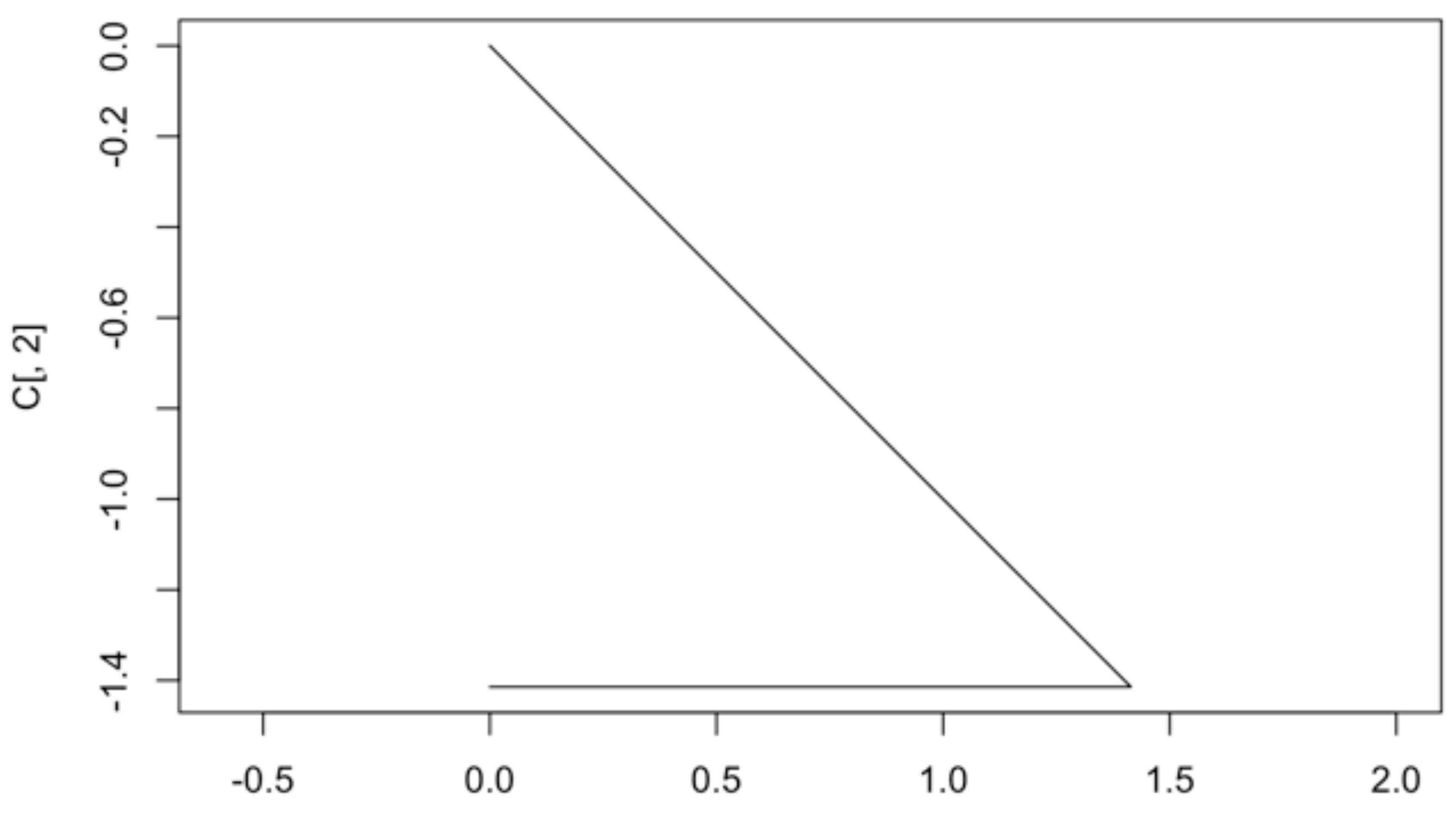
```
Rot <- function(alpha) {
  a <-alpha*pi/180
  matrix(c(cos(a), sin(a),
           -sin(a), cos(a)), 2, 2)
}
```

For example, if we want to rotate our object A 45 degrees clockwise, we can call `A %*% Rot(45)`.

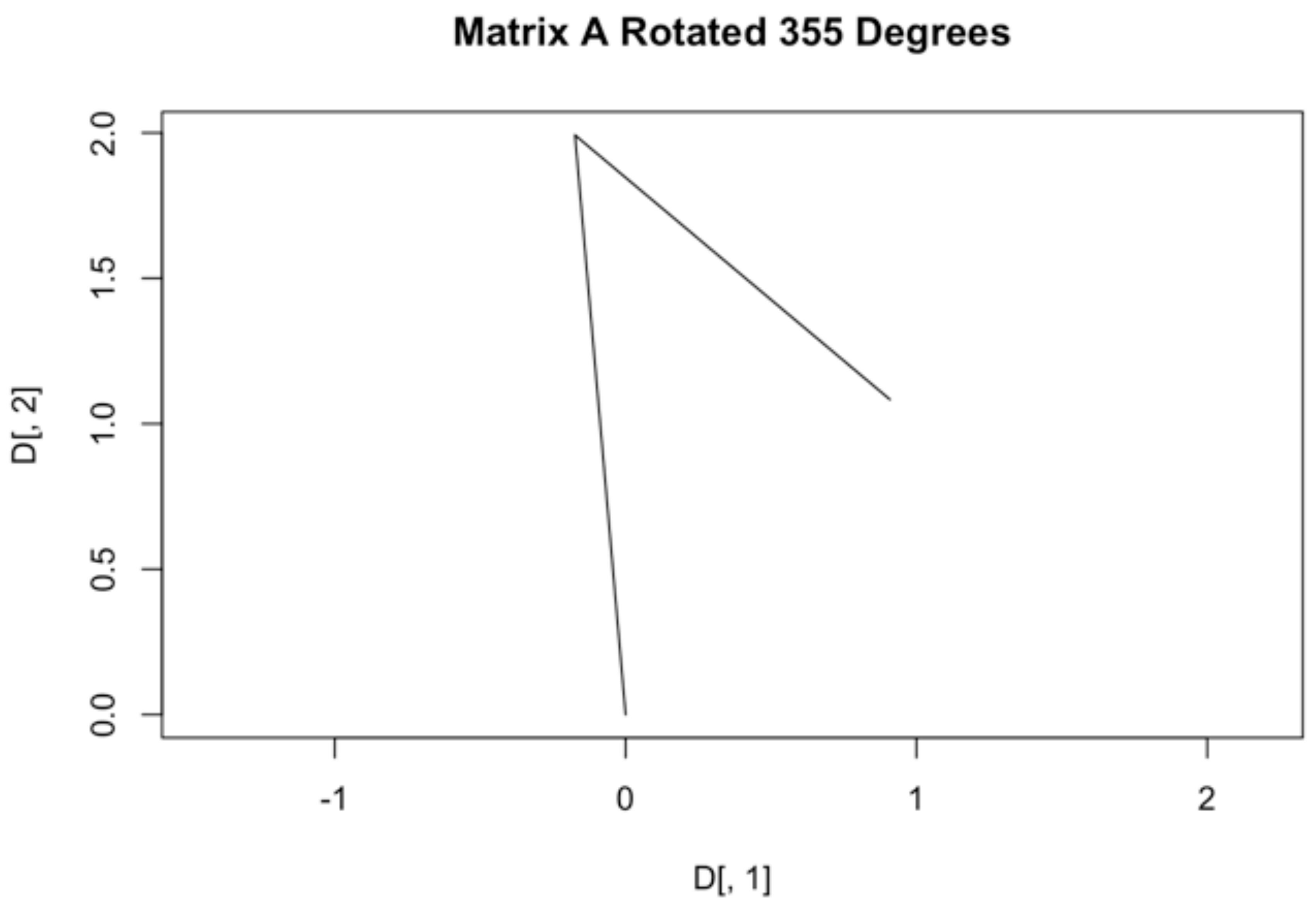
Matrix A Rotated 45 Degrees



Matrix A Rotated -225 Degrees

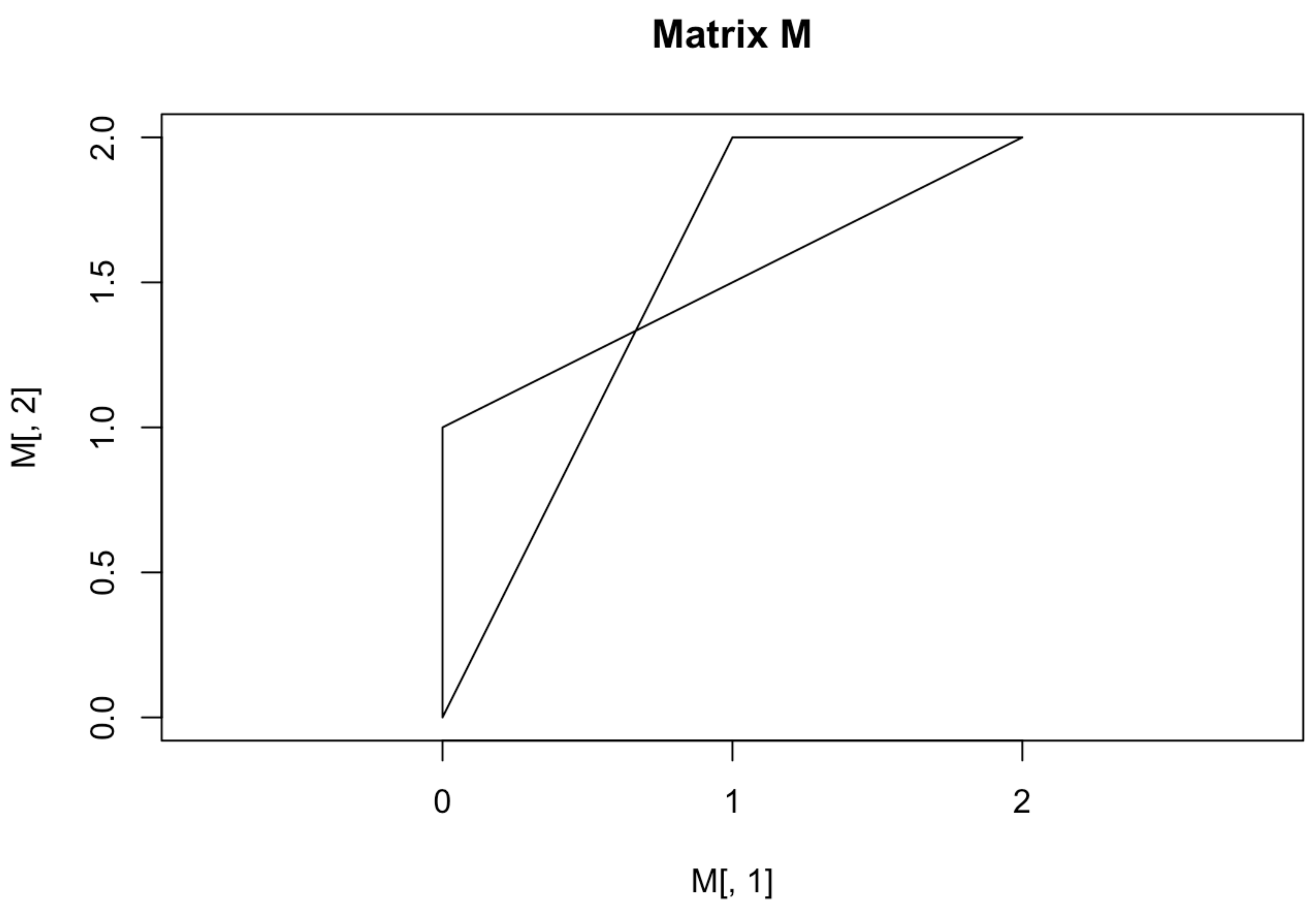


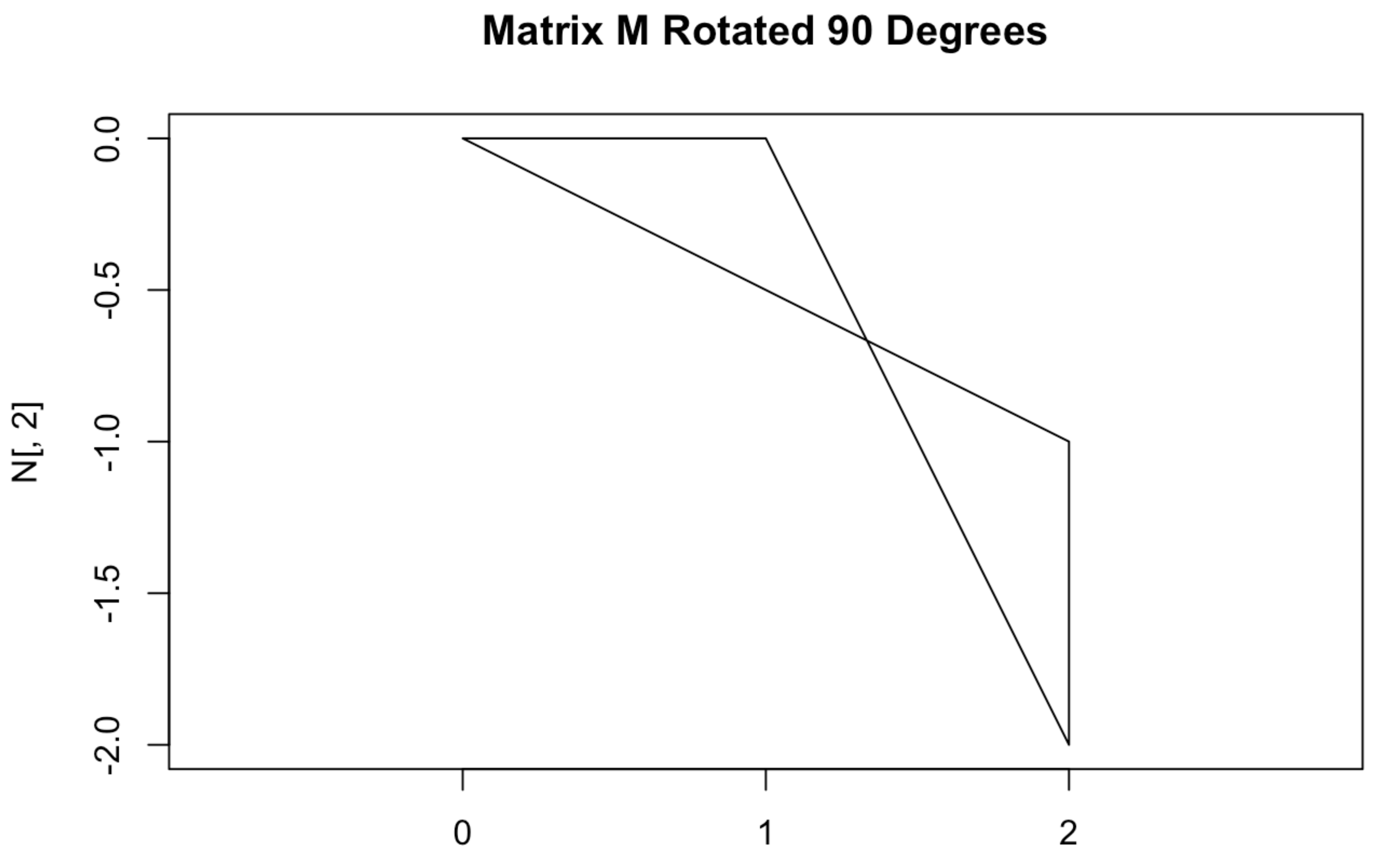
C[, 1]



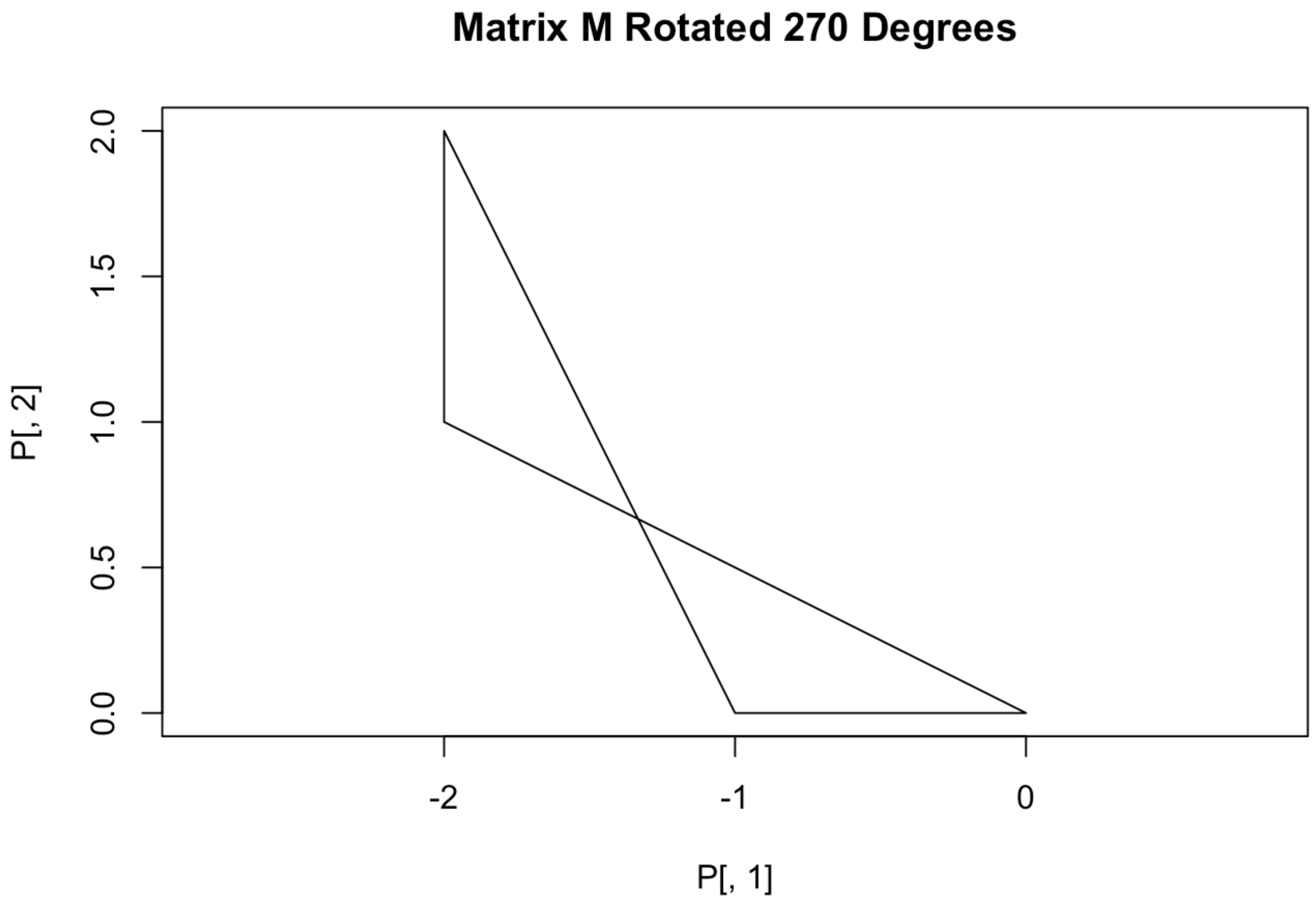
Now let’s create our own object matrix *M* and demonstrate rotation again on it.

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


Matrix M
2.0
1.5
1.0
0.5
0.0
M[, 2]
0
1
2
M[, 1]


Matrix M Rotated 90 Degrees
0.0
-0.5
-1.0
-1.5
-2.0
N[, 2]
0
1
2

N[, 1]

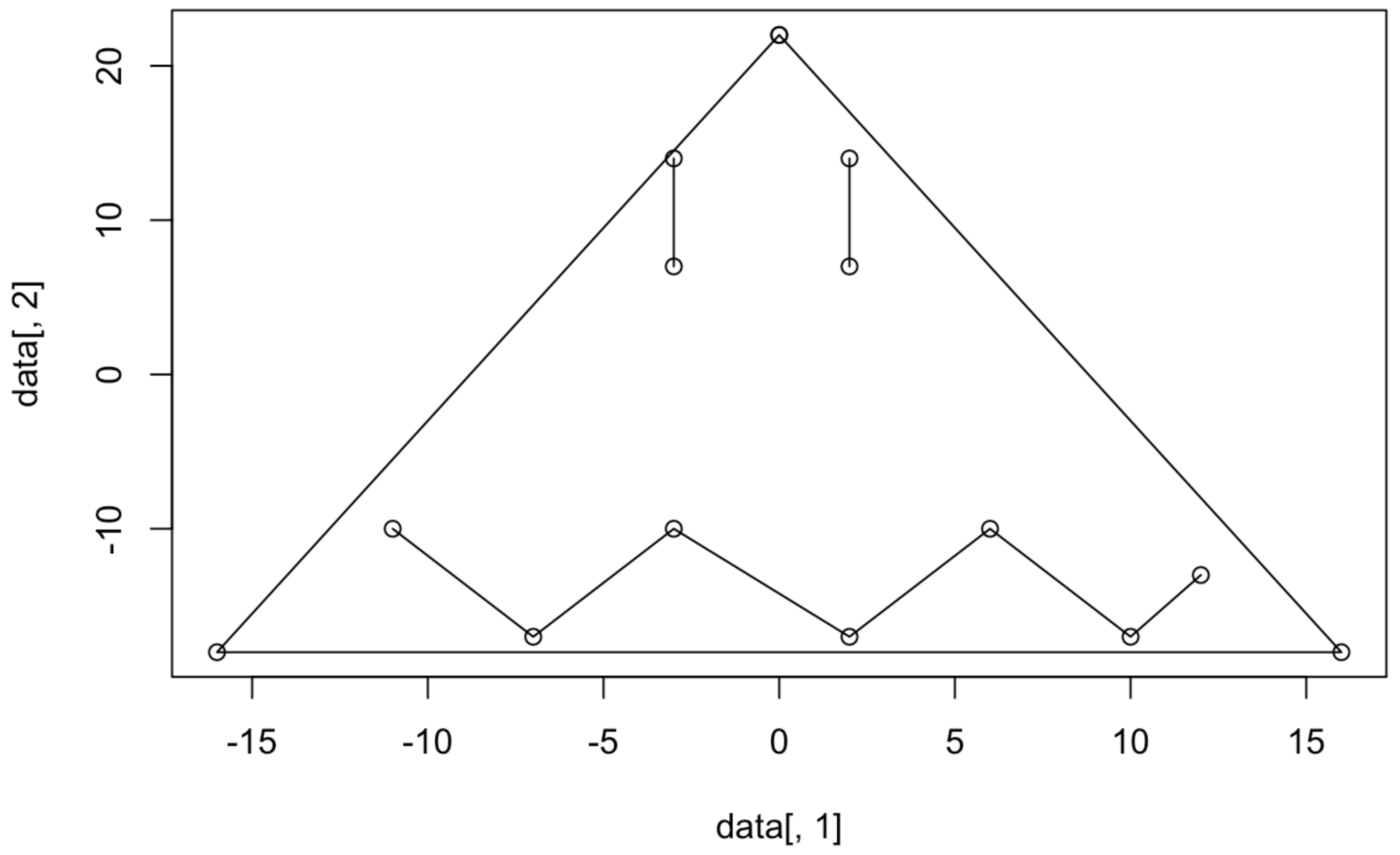


2.2 Rotate Data

Using the data from *crazy_hat.tsv*, let’s create a plot with lines connecting elements of the same group. We will use the following helper function to plot our image with any rotated crazyhat data.

```
plot_crazyhat <- function(data) {
  outline <- data[data["group"] == "outline",]
  leye <- data[data["group"] == "leye",]
  reye <- data[data["group"] == "reye",]
  mouth <- data[data["group"] == "mouth",]

  plot(data[, 1], data[, 2])
  lines(outline)
  lines(leye)
  lines(reye)
  lines(mouth)
}
```

Let’s see the what the crazyhat plot looks like!

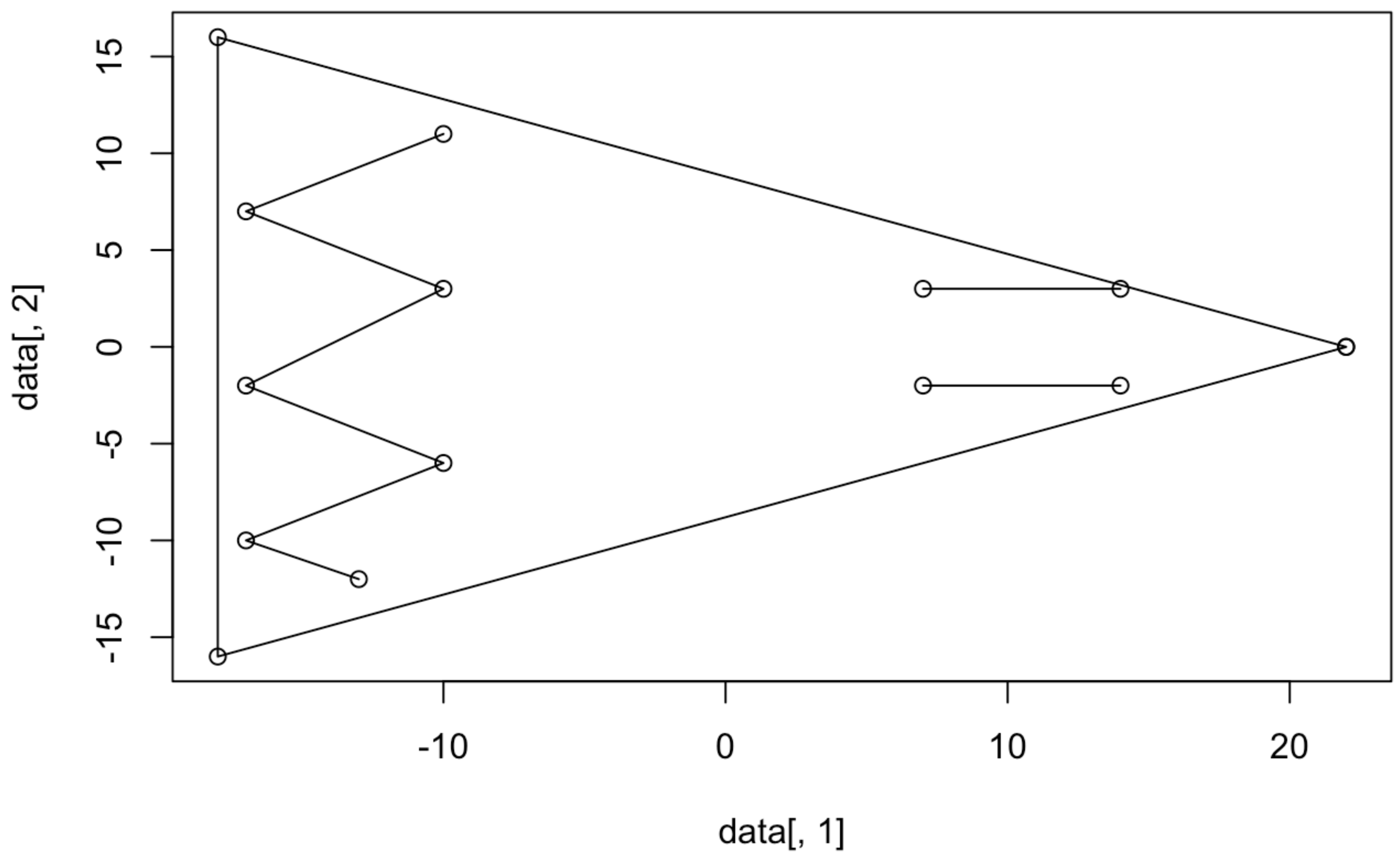


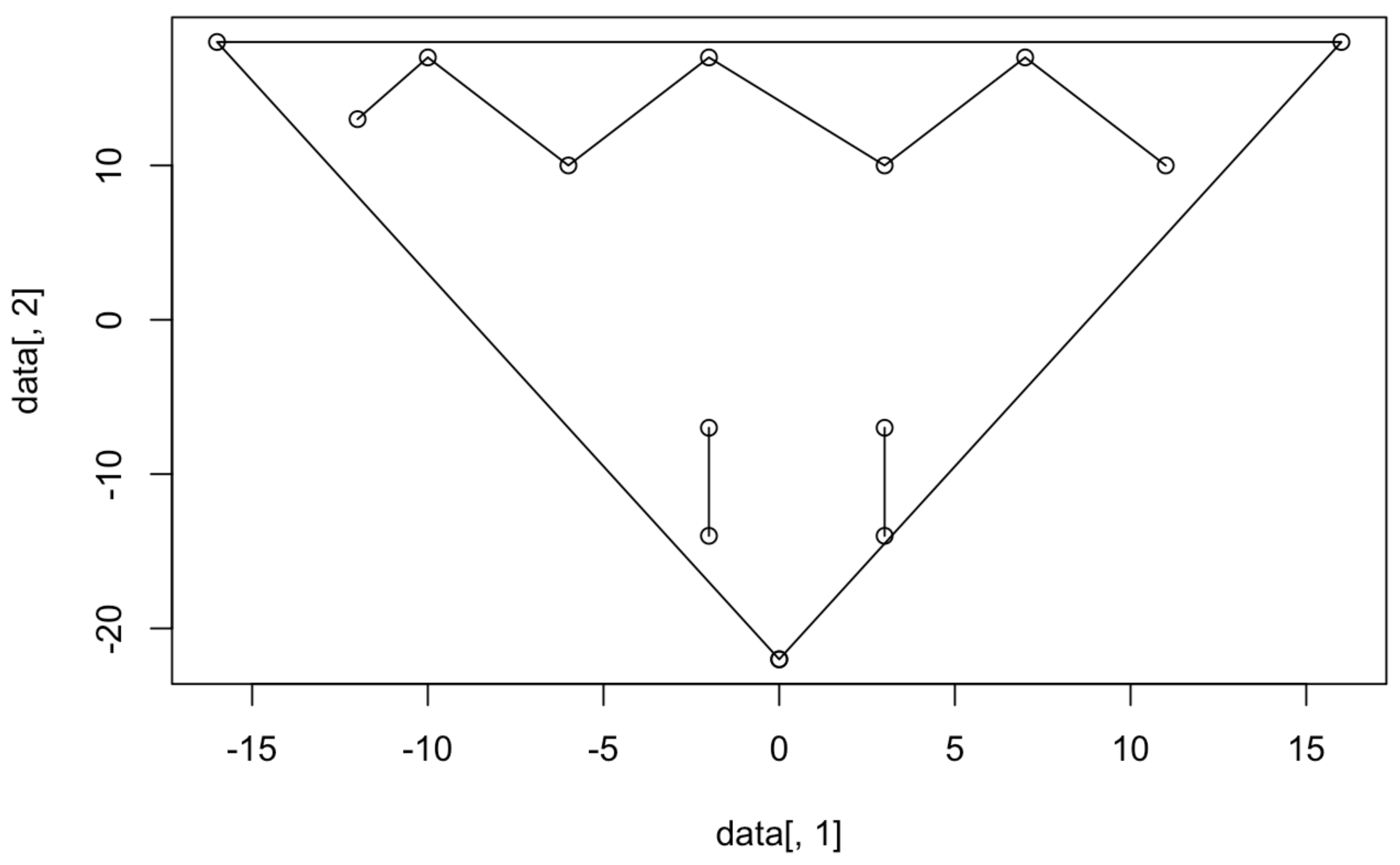
To rotate this plot, we will first make the desired columns of the data into a matrix. We will then rotate that matrix, and then convert the matrix back to a dataframe. We can then plot the dataframe and again connect elements of the same group to complete the image rotation. Here is the helper function that shows this process:

```
rotate <- function(data, x) {
  rotated <- matrix(data.matrix(data[, 1:2], rownames.force = NA), 15, 2) %*% Rot(x)
  new_data <- data.frame(rotated)
  group <- data[, 3]
  new_data$group <- group
  new_data
}
```

Let’s see some rotations below.

Image rotated by 90:





2.3 Flip and Stretch

Let’s show a few more simple image tranformations: **flipping** and **stretching**!

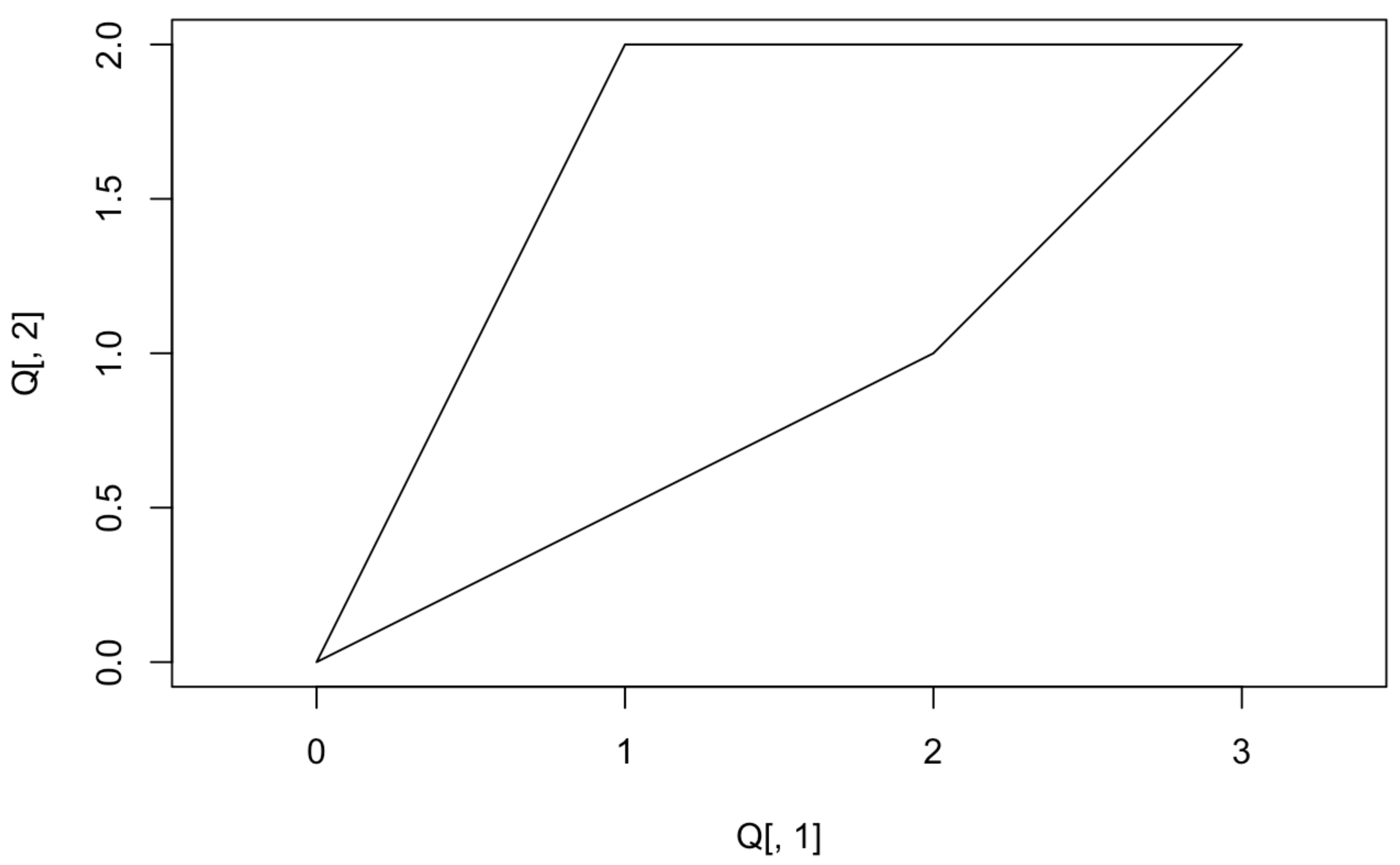
To **flip** a matrix by x (mirroring the image on y-axis), we need to multiply that matrix by the following matrix

[-1, 1 0, 1]

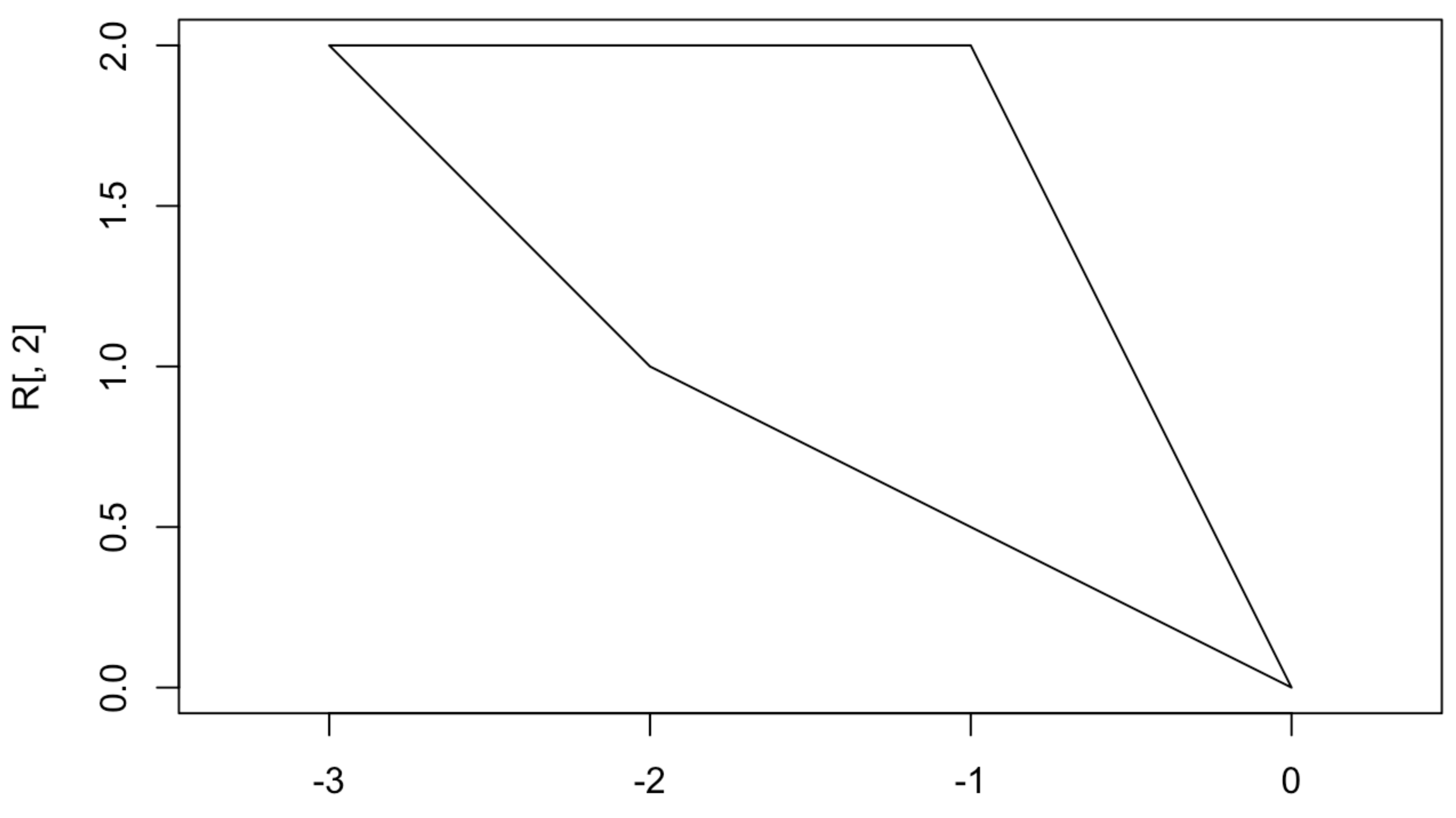
We use the helper function `flipx` to flip any given matrix *a* along the y-axis. The flipping is demonstrated below.

```
flipx <- function(a) {
  a %*% matrix(c(-1, 0,
                 0, 1), 2, 2, byrow = TRUE)
}
```

Matrix Q



Flipped Matrix Q



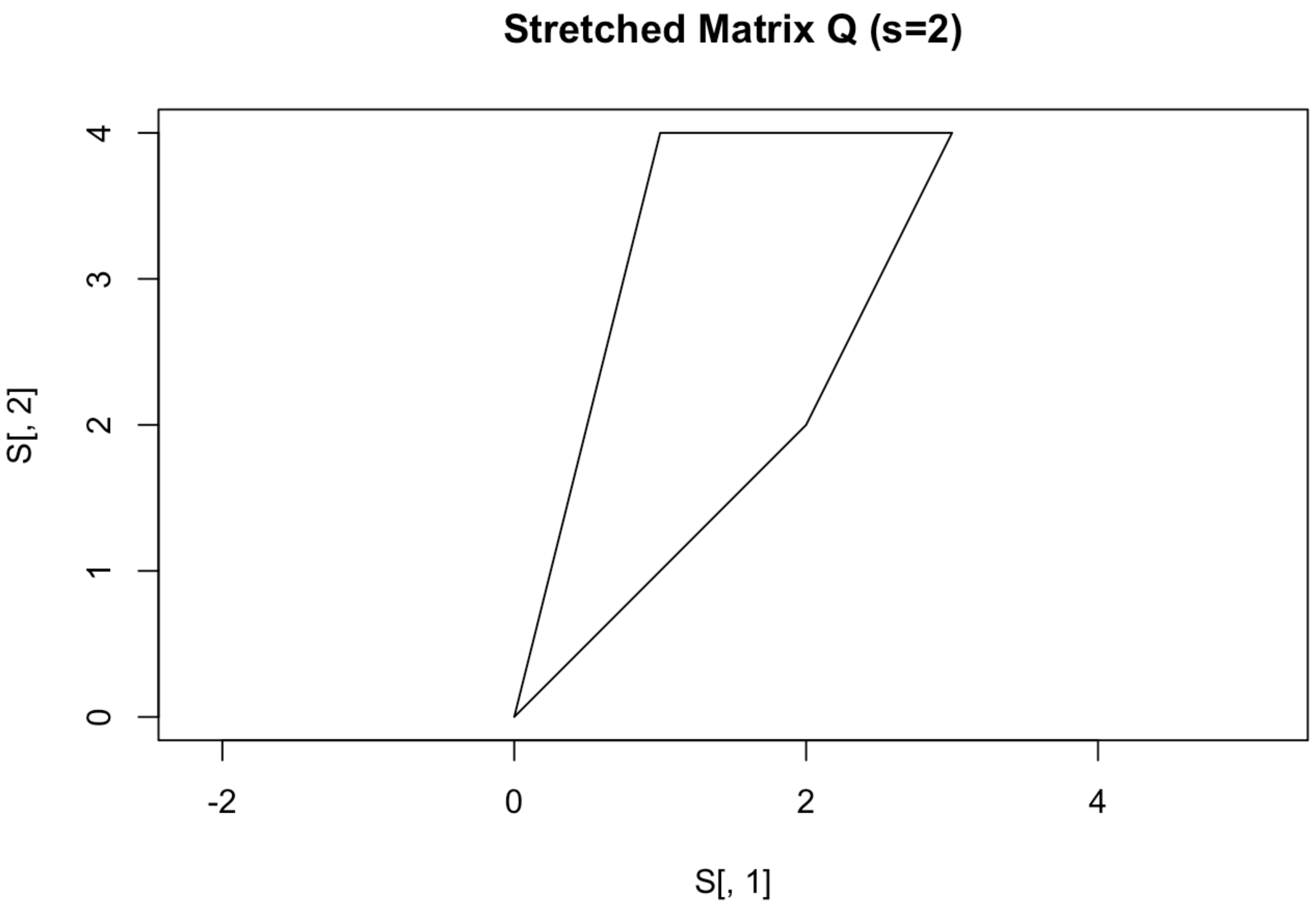
R[, 1]

To **stretch** a matrix's y elements by amount *s*, we must multiply that matrix by the following matrix:

[1, 0 0, s]

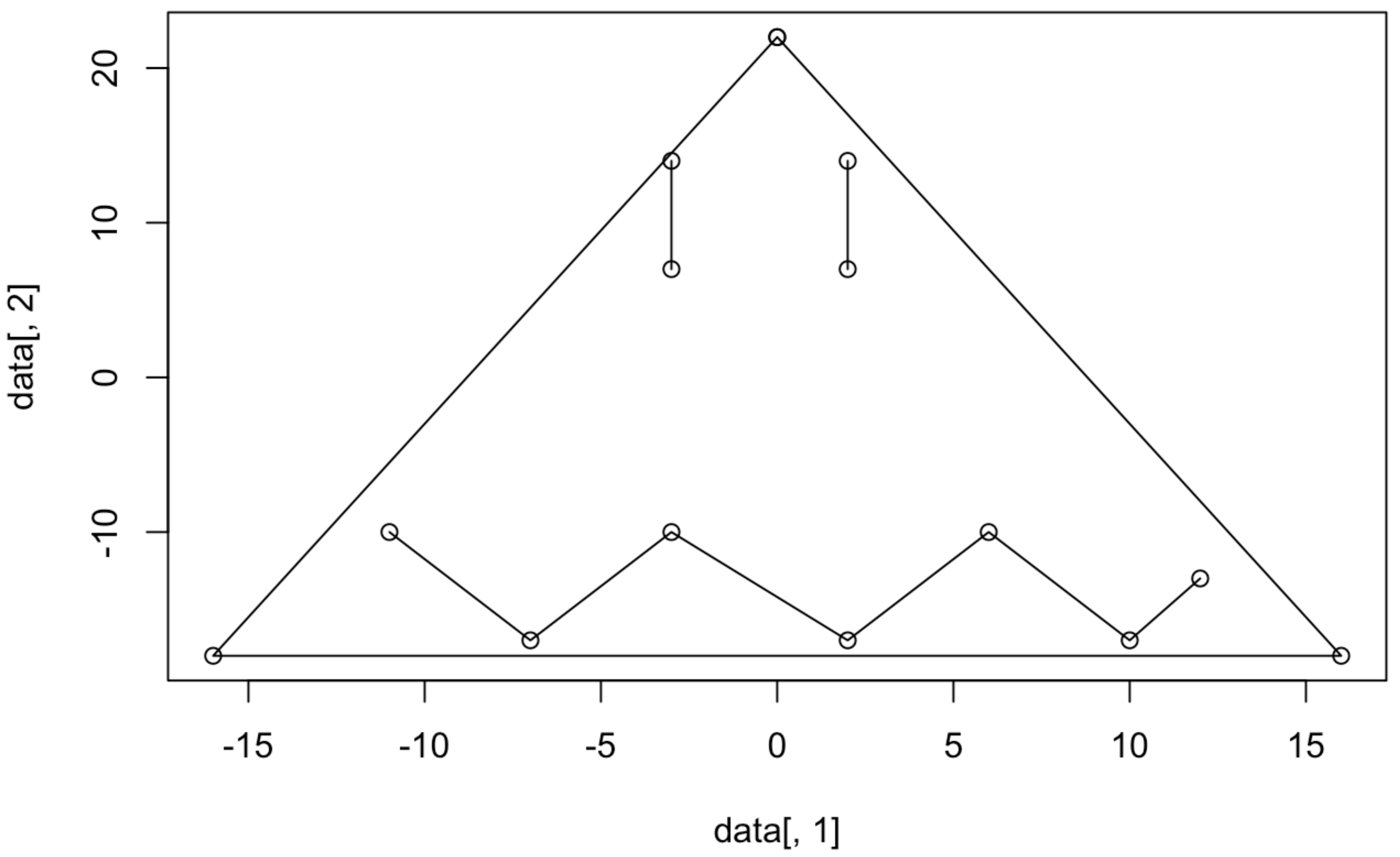
The helper function `stretchy` allows us to stretch y elements of any given matrix *a* by a given amount *s*.

```
stretchy <- function(a, s) {
  a %*% matrix(c(1, 0,
                 0, as.numeric(s)), 2, 2, byrow = TRUE)
}
```

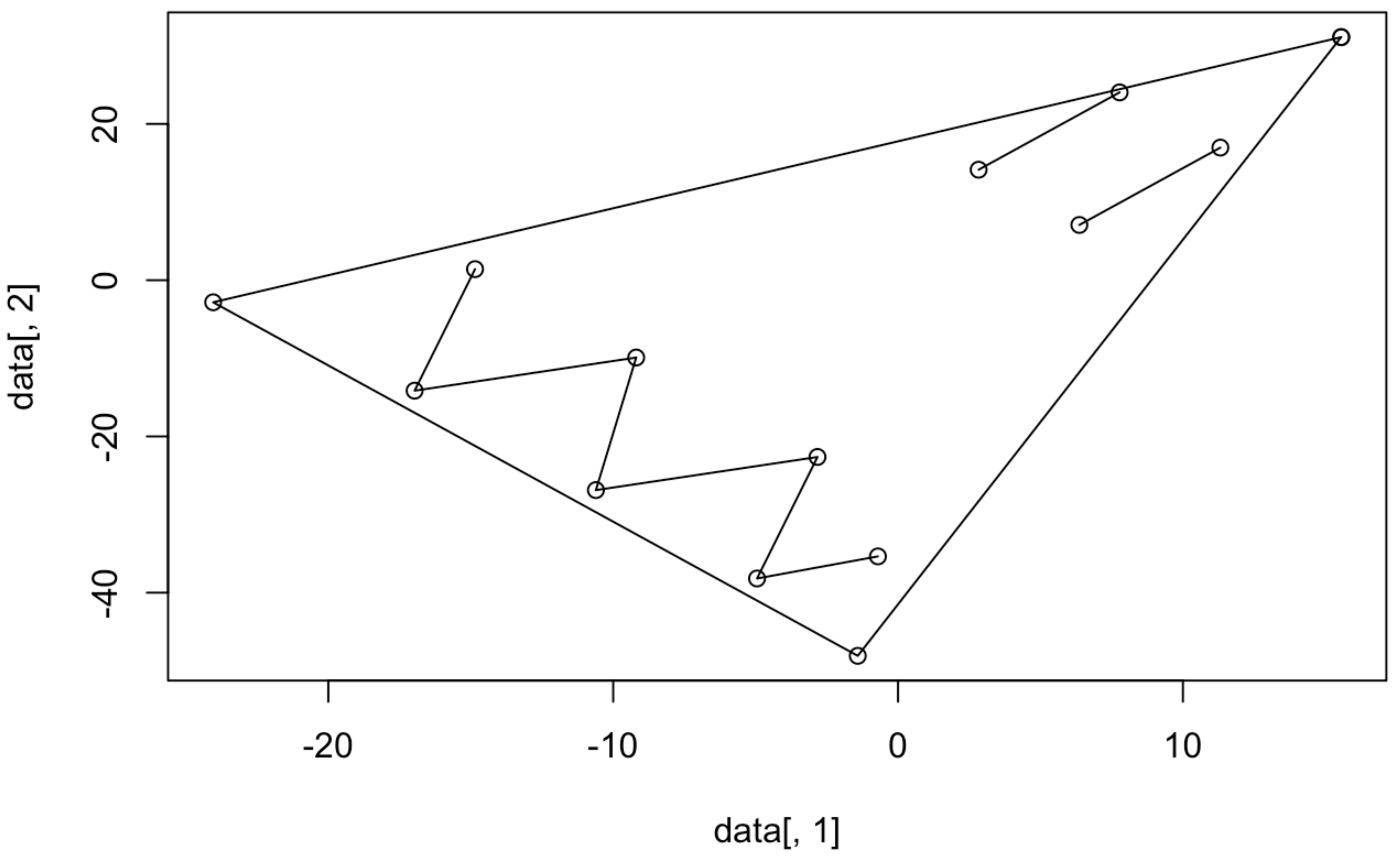


Finally, we can combine some of these simple image transformations! For example, if we want to take our *crazyhat* image again and **rotate it 45 degrees, then stretch it's y elements 2x**, we get this:

Original



Distorted



Above we discussed the transformations by right-multiplication. How do we transform the problem in a way that we can pre-multiply instead of post-multiply in order to perform an operation? Something like A^45 = R(45) * A. What will the left-hand rotation matrix R look like? We can do this through matrix transpose:

t(t(Rot(45) * t(A))