Tutorial on R Software

MUMS Undergraduate Workshop

Pulong Ma and Wenjia Wang, SAMSI

2/25/2019

What is R?

R is a free software environment for statistical computing and graphics:

- is a different implementation of S language;
- provides a wide variety of statistical and graphical techniques, and is highly extensible;
- is available as Free Software under the terms of the Free Sofware Foundations' GNU General Public License in source code form;
- runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS;
- has powerful IDE (integrated development environment) such as RStudio.

Install R

1. Download the most recent version of R at https://cran.r-project.org.
2. Start the R program:
   - on Windows and MacOS, this will usually mean double-click on the R application
   - on UNIX-like systems, type "R" at a shell prompt.
3. As a first step with R, start with the R help brower by typing **help.start()** in the R command window.

Install RStudio

1. Go to RStudio's website https://www.rstudio.com/products/rstudio/download/.
2. Click on "Download RStudio Desktop"
3. Click on versions recommended for your system.

R Session

launch R

launch RStudio

Operators in R

```
## addition: +
3 + 2
```

```
## [1] 5
```

```
## subtraction: -
3 - 2
```

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

```
## multiplication: *
3 * 2
```

```
## [1] 6
```

Operators in R

```
## division: /
3 / 2
```

```
## [1] 1.5
```

```
## exponentiation: ^ or **
3^2
```

```
## [1] 9
```

```
## greater than: >
3 > 2
```

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

Operators in R

```
## greater than or equal to: >=
3 >= 2
```

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

```
## exactly equal to: ==
3 == 2
```

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

```
## not equal to: !=
3 != 2
```

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

Creating New Variables

Use the assignment operator <- or = to create new variables.

```
x <- 1
print(x)
```

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

```
x = 2
print(x)
```

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

Data Types

R has a wide variety of data types including

- scalars,
- vectors (numerical, character, logical),
- matrices,
- data frames,
- and lists.

Scalar

```
num = 3
print(num)
```

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

```
print(typeof(num))
```

```
## [1] "double"
```

Scalar

```
num = 3.14
num.int = as.integer(num)
print(num.int)
```

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

```
print(typeof(num.int))
```

```
## [1] "integer"
```

Vector

```
x = 1:3
print(x)
```

```
## [1] 1 2 3
```

```
y = c(4, 5, 6, 7)
y[1]  # subsetting
```

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

```
y[-1] # subsetting
```

```
## [1] 5 6 7
```

Vector

```
y[c(1,4)]  # subsetting
```

```
## [1] 4 7
```

```
y[-c(1,4)] # subsetting
```

```
## [1] 5 6
```

```
z = c(y[c(1,4)], y[-c(1,4)])
print(z)
```

```
## [1] 4 7 5 6
```

Matrix

```
a = seq(1, 9, length.out=9)
A = matrix(a, nrow=3, ncol=3)
print(A)
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
print(typeof(A))
```

```
## [1] "double"
```

```
print(class(A))
```

```
## [1] "matrix"
```

Matrix

```
# subsetting the first two elements in the first column
A[1:2, 1]
```

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

```
A[1:2, c(1,2)]
```

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

```
A[1:2, ]  # subsetting the first two rows
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
```

Array

```
b = seq(1, 8, by=1)
B = array(data=b, dim=c(2,2,2))
print(B)
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## , , 2
##
##      [,1] [,2]
## [1,]    5    7
## [2,]    6    8
```

Array

```
class(B)
```

```
## [1] "array"
```

```
B1 = B[ , , 1]

B2 = B[ , , 2]

C = cbind(B1, B2)
print(C)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    5    7
## [2,]    2    4    6    8
```

Data Frame

```r
df = data.frame(x = c(1, 5, 6), y = c(2, 4, 3))
print(df)
```

```
##   x y
## 1 1 2
## 2 5 4
## 3 6 3
```

```r
print(class(df))
```

```
## [1] "data.frame"
```

Data Frame

```
print(df$x)
```

```
## [1] 1 5 6
```

```
print(df$y)
```

```
## [1] 2 4 3
```

List

```
l1 = list(1, 2, 3)
print(l1)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
```

```
class(l1)
```

```
## [1] "list"
```

List

```r
names(l1) <- c("a", "b", "c")
print(l1)
```

```
## $a
## [1] 1
##
## $b
## [1] 2
##
## $c
## [1] 3
```

List

```
l1[[1]]  ## subsetting the first element
```

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

```
l1$a  ## subsetting the element named a
```

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

```
l1$a = 4
print(l1$a)
```

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

Linear Algebra

```
A = matrix(c(2, 3, 1, 5), nrow=2, ncol=2)

## transpose
t(A)
```

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

```
## matrix addition
B = matrix(c(2, 2, 3, 5), nrow=2, ncol=2)
A + B
```

```
##      [,1] [,2]
## [1,]    4    4
## [2,]    5   10
```

Linear Algebra

```
## matrix multiplication
A %*% B
```

```
##      [,1] [,2]
## [1,]    6   11
## [2,]   16   34
```

```
### elementwise multiplication
A * B
```

```
##      [,1] [,2]
## [1,]    4    3
## [2,]    6   25
```

Linear Algebra

```
A = matrix(c(2, 1, 1, 5), nrow=2, ncol=2)
b = c(1,2)
## solve the system Ax = b
solve(A, b)
```

```
## [1] 0.3333333 0.3333333
```

```
## compute cholesky decomposition
R = chol(A)
## use triangular solvers
backsolve(R, backsolve(R, b, transpose=TRUE))
```

```
## [1] 0.3333333 0.3333333
```

Importing Data

Let's load the buit-in R data "Orange" using **data()**.

```
data("Orange")
head(Orange)
```

```
##   Tree  age circumference
## 1    1  118            30
## 2    1  484            58
## 3    1  664            87
## 4    1 1004           115
## 5    1 1231           120
## 6    1 1372           142
```

```
names(Orange)
```

```
## [1] "Tree"          "age"           "circumference"
```

Importing Data

Let's save the data into local disk.

```
Tree = Orange$Tree
age = Orange$age
circ = Orange$circumference

# save three vectors into the .RData format
save(Tree, age, circ, file="mydata.RData")

# save circumference into the .csv format
write.csv(circ, file="mydata.csv")

# load the csv file again
mycirc = read.csv(file="mydata.csv")
```

Writing Your Own Function

```r
mysquare <- function(x){
  y = x^2     ## main body of the function
  return(y)   ## return variable y
}

y1=mysquare(3)
print(y1)
```

```
## [1] 9
```

```r
## load functions stored in your local disk
source("mysquare.R")
y2=mysquare(4)
print(y2)
```

```
## [1] 16
```

Packages

```
## Install from repository

install.packages(c("mvtnorm", "plotrix"))
```

```
##
## The downloaded binary packages are in
##  /var/folders/_k/ckfbbmb51nz6bptcpb68qftm0000gn/T//Rtmpt
```

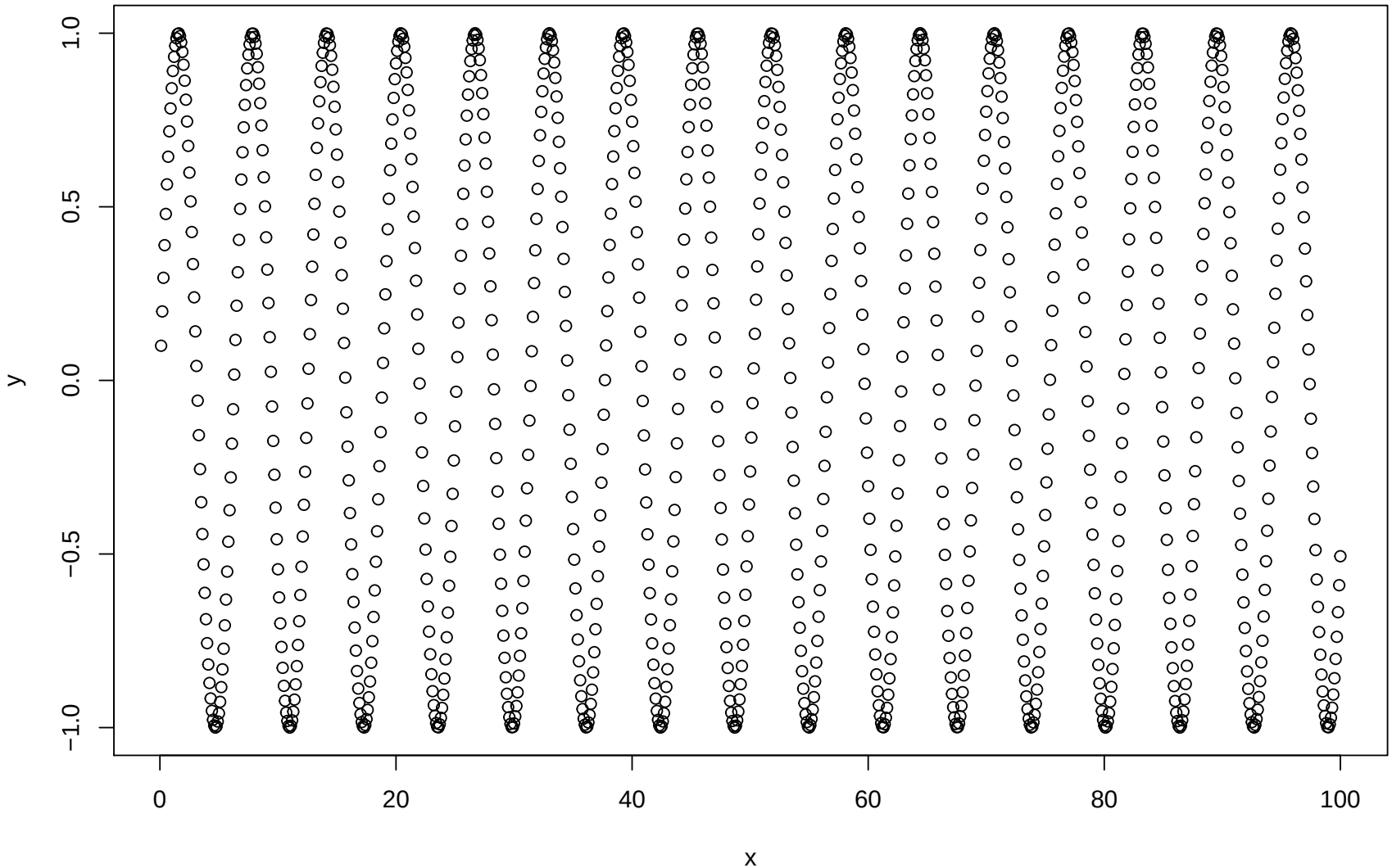
```
## load packages
library(mvtnorm)
library(plotrix)
```

Basic Plots

```
## Use of "plot" function
x = 1:1000/10
y = sin(x)
print(head(x))
plot(x, y)
```

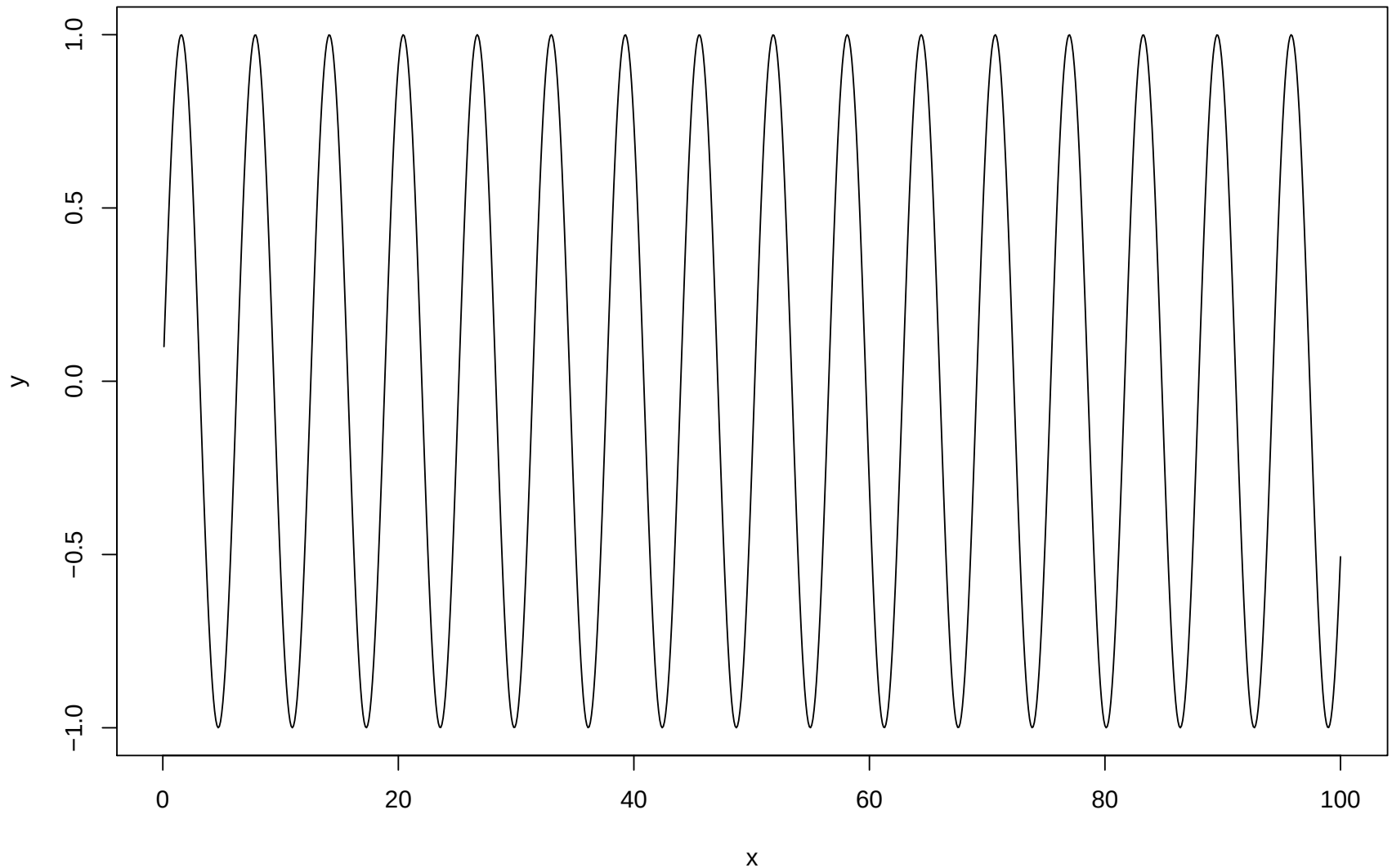
Basic Plots

```
## [1] 0.1 0.2 0.3 0.4 0.5 0.6
```



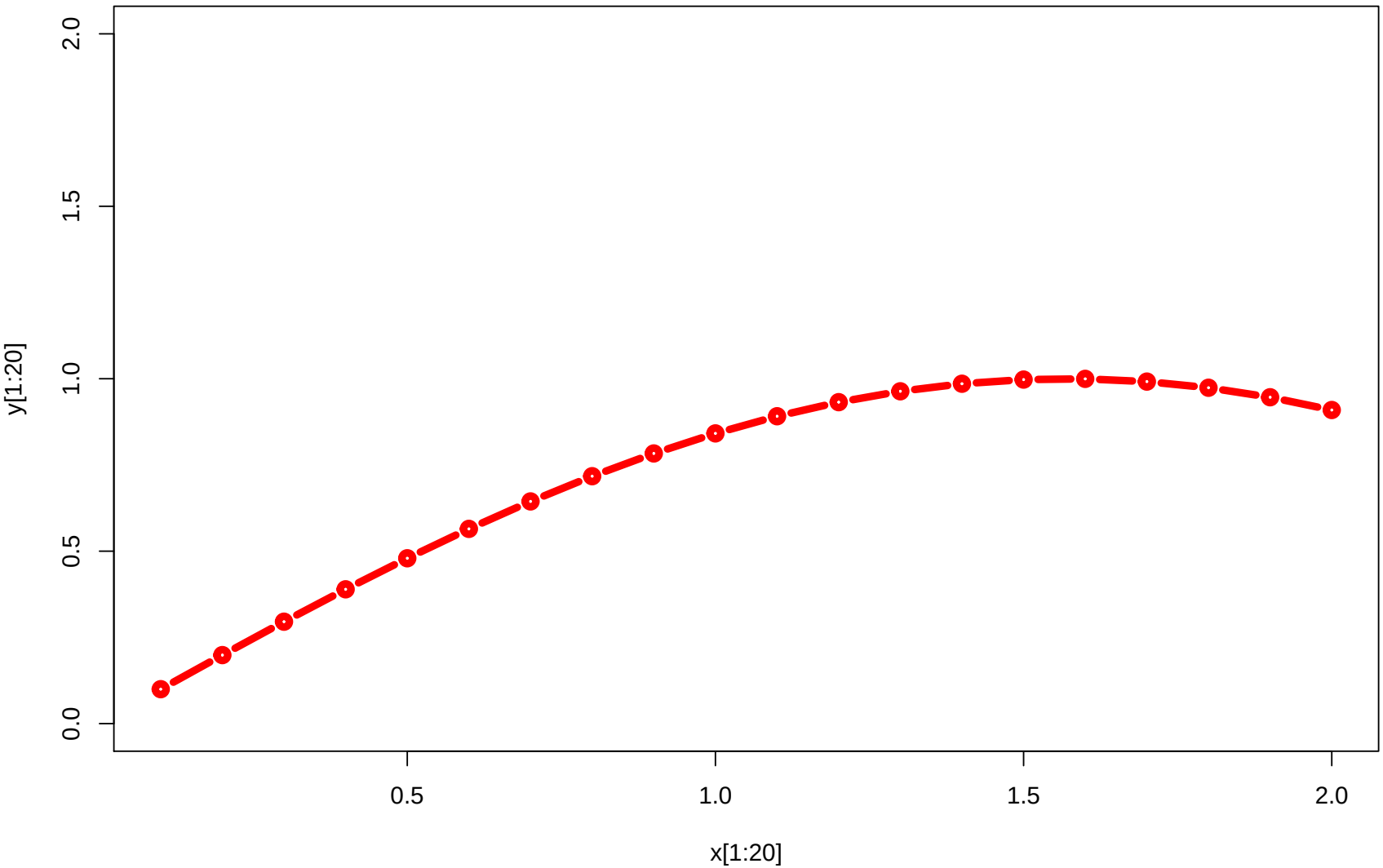
Basic Plots

```
## Use of "plot" function
plot(x, y, type = "l")
```



Basic Plots

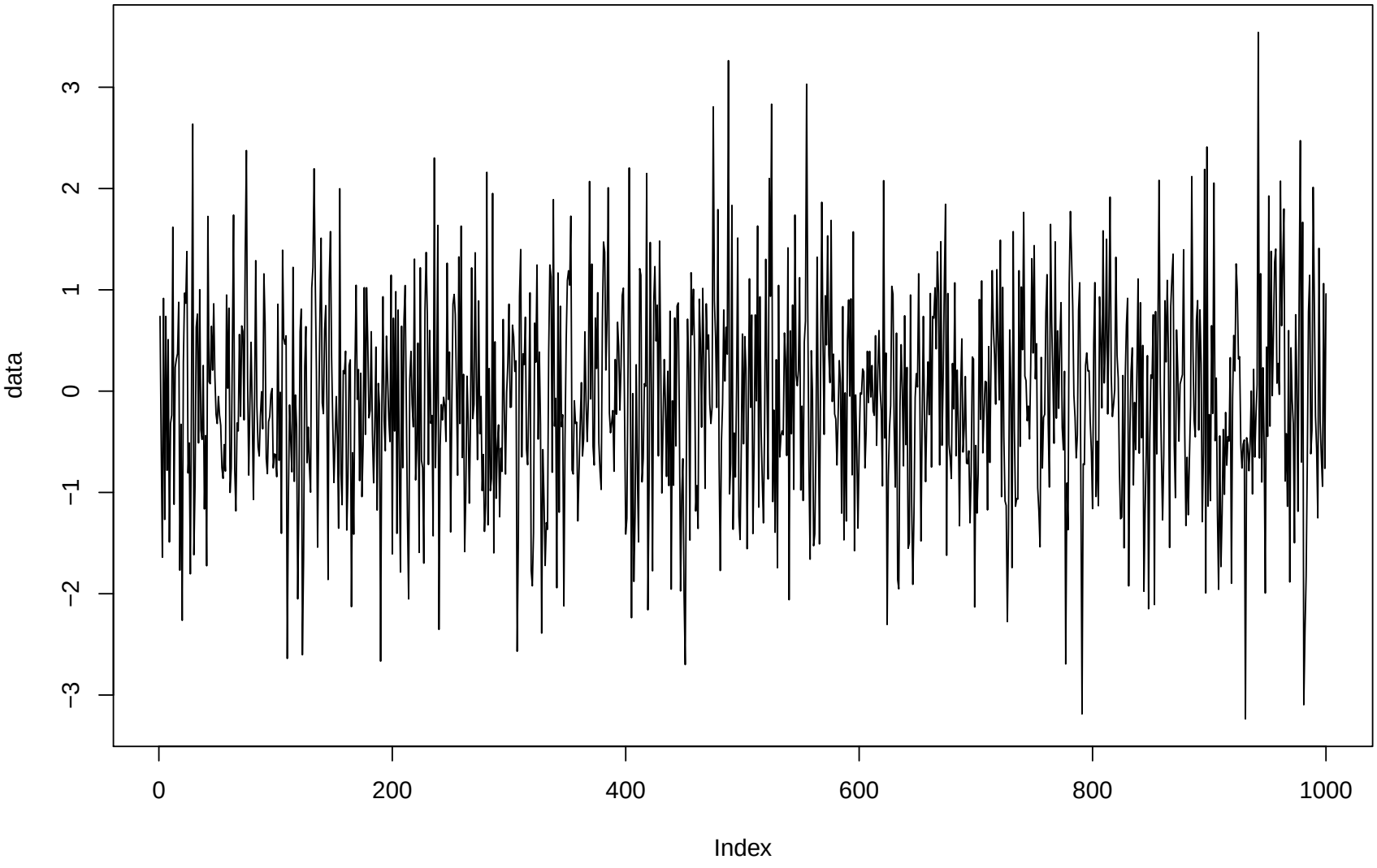
```
plot(x[1:20], y[1:20], type = 'b', col = 'red',
     lwd = 5, ylim = c(0, 2))
```



Basic Plots

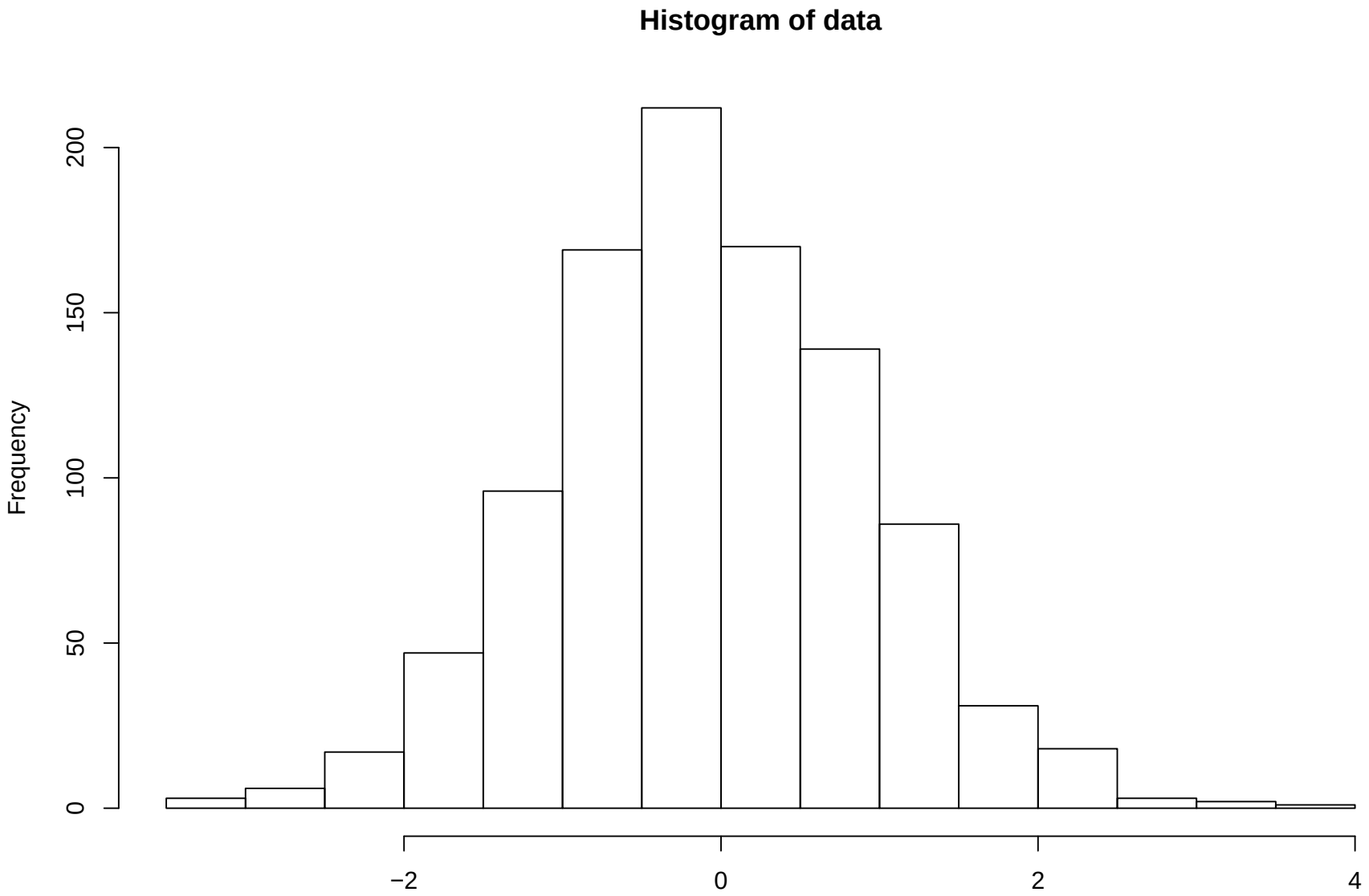
```
## Use of "hist" function
set.seed(2019)
data = rnorm(1000)
plot(data, type = 'l')
```

Basic Plots



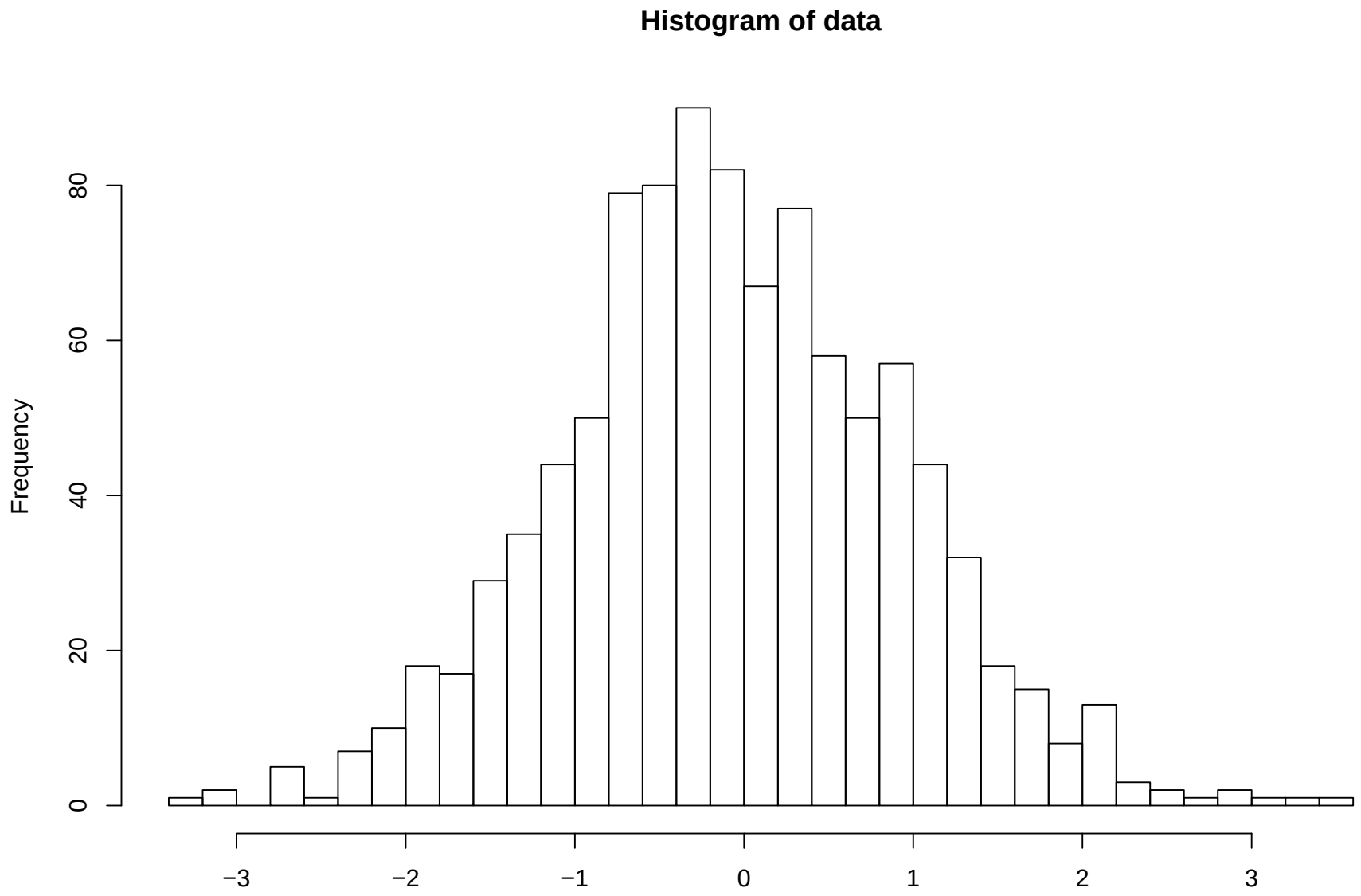
Basic Plots

```
## Use of "hist" function
hist(data)
```



Basic Plots

```
## Use of "hist" function
hist(data, breaks = 30)
```



Sampling from Basic Distributions

```
## Uniform distribution
runif(n = 2, min = 0, max = 1)
```

```
## [1] 0.948481558 0.000856857
```

```
## Normal distribution
rnorm(n = 2, mean = 0, sd = 1)
```

```
## [1]  0.4710428 -0.7306247
```

Multivariate Normal Distribution

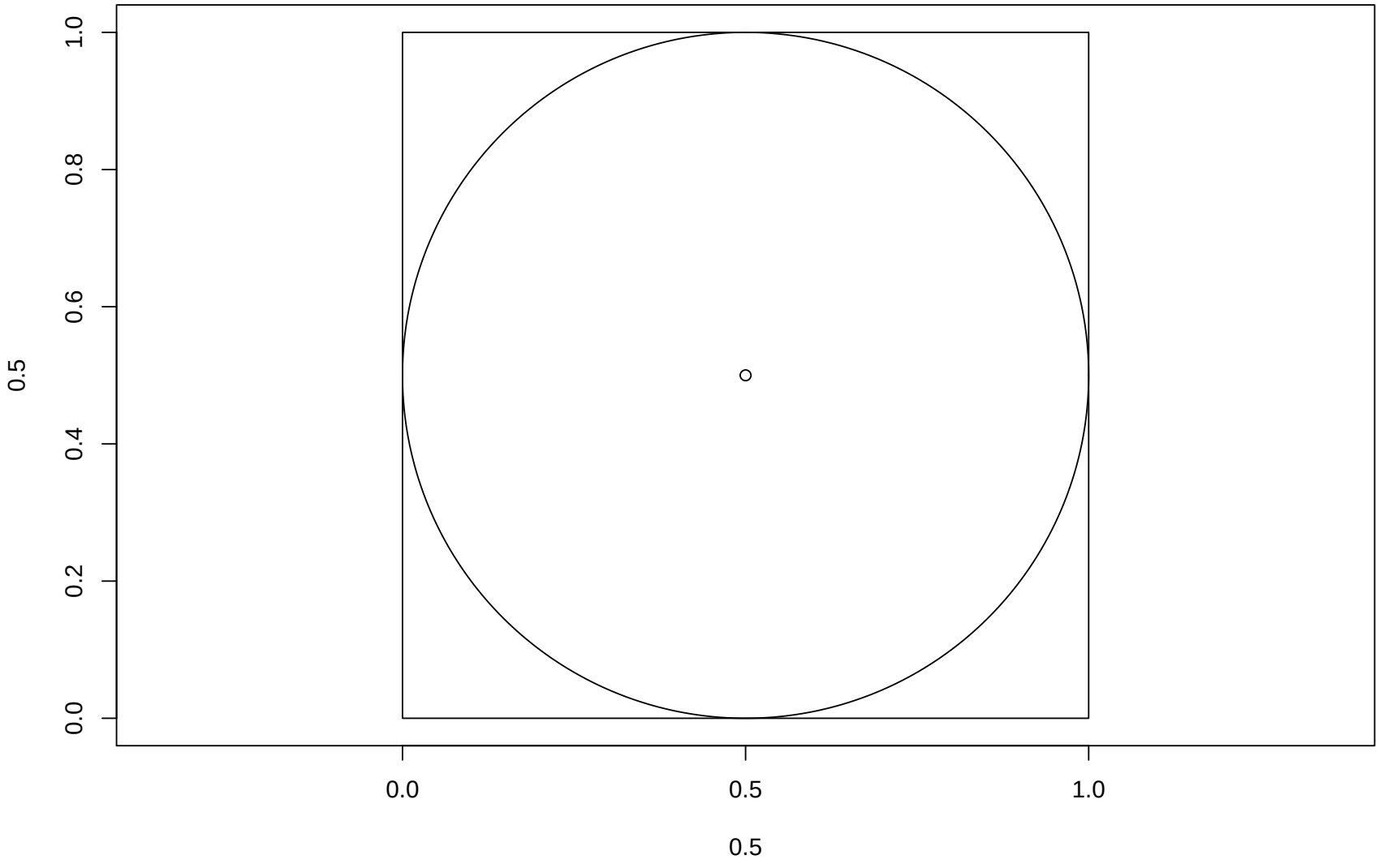
```
## Need package mvtnorm
Sigma <- matrix(c(10,3,3,2),2,2)
rmvnorm(n = 1, mean = rep(0, nrow(Sigma)),
        sigma = Sigma)
```

```
##             [,1]      [,2]
## [1,] 0.3263166 0.8458712
```

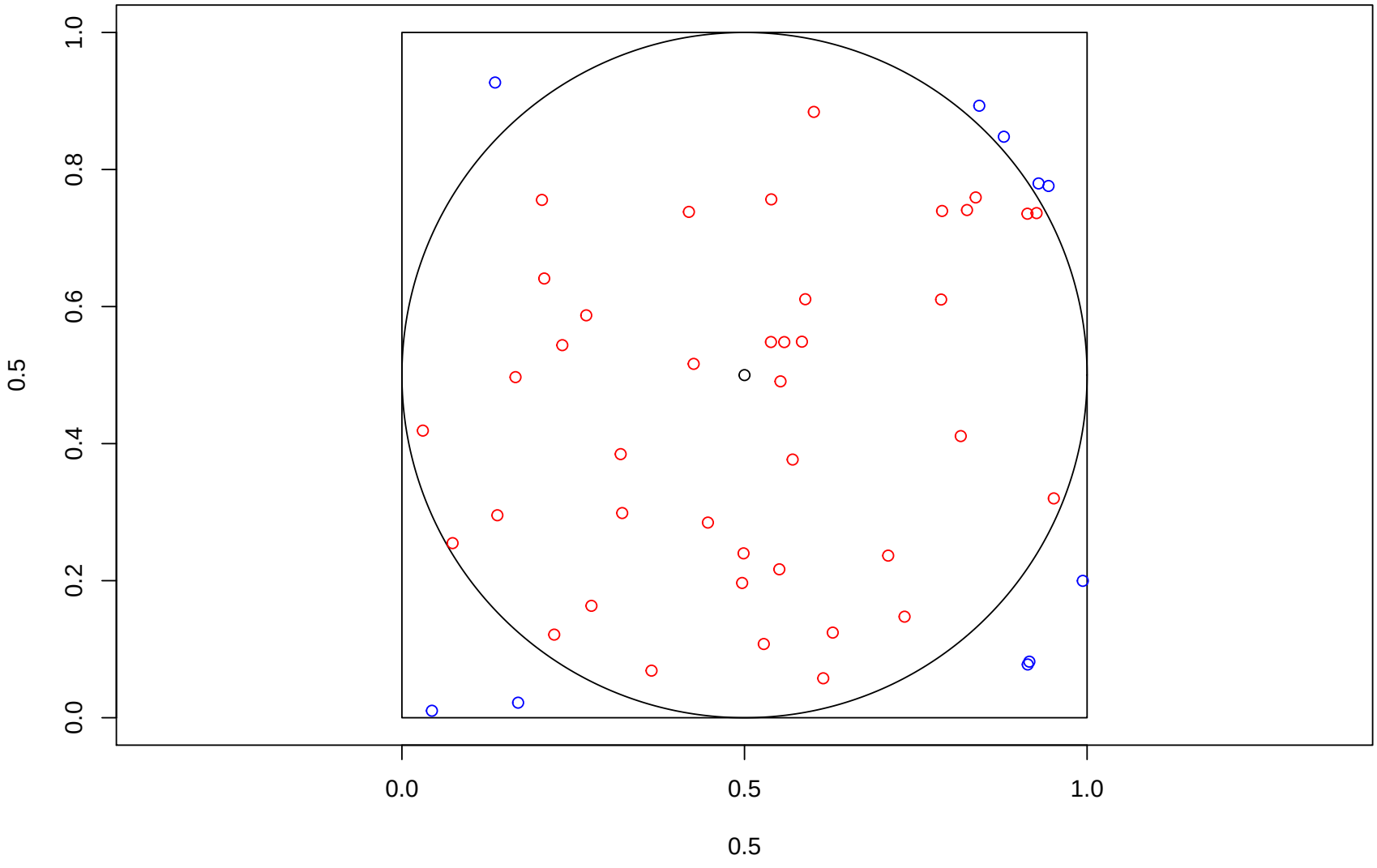
Calculating π

```
## Need package plotrix
plot(0.5,0.5,type ="p",asp = 1, xlim=c(0,1)
     ,ylim=c(0,1),color = "black")

draw.circle(0.5,0.5,0.5,nv=1000
            ,border=NULL,col=NA,lty=1,lwd=1)

rect( 0, 0, 1, 1)
```

Calculating π



Calculating π




```
## [1] 0.01859164
```

Convergence Rate