Introduction to R

Mwaura Patrick

2023-09-13

Introduction

This document provides an introduction to the R programming language. It covers various features of the R environment and demonstrates how to use them.

Sample Session

In this section, we will explore some basic features of the R environment through sample sessions. These examples will help you become familiar with R's capabilities.

```
# Help Start
# help.start()
```

Start the HTML interface to online help using your web browser. Explore the features of this facility to get acquainted with it.

Generate Random Vectors

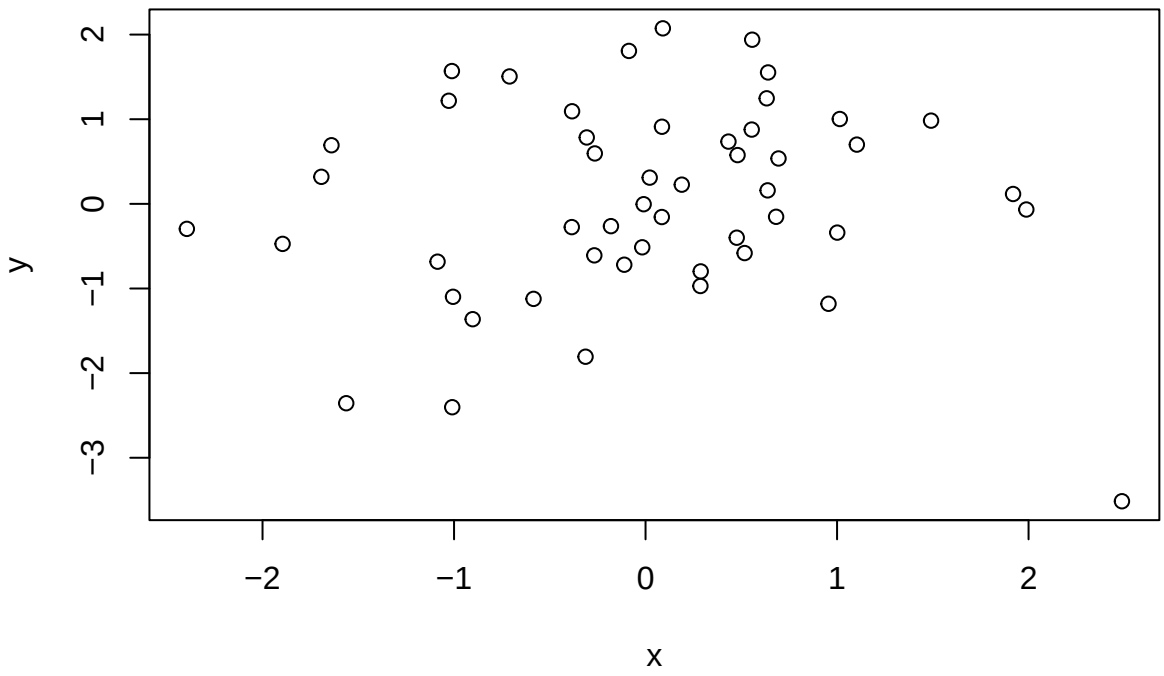
Let's generate two pseudo-random normal vectors for x- and y-coordinates.

```
x <- rnorm(50)
y <- rnorm(x)
```

Plot Data

Plot the points in the plane, and a graphics window will appear automatically.

```
plot(x, y)
```



R Workspace

Check which R objects are currently in the R workspace.

```
ls()
```

```
## [1] "x" "y"
```

Data Cleanup

Remove objects that are no longer needed to clean up the workspace.

```
rm(x, y)
```

Creating Data

Make x a sequence from 1 to 20.

```
x <- 1:20
```

Weight Vector

Create a 'weight' vector of standard deviations.

```
w <- 1 + sqrt(x) / 2
```

Data Frame

Create a data frame of two columns, x and y, and display it.

```
dummy <- data.frame(x = x, y = x + rnorm(x) * w)
dummy
```

```
##     x         y
## 1   1  3.190100
## 2   2  2.538004
## 3   3  1.891214
## 4   4  6.192257
## 5   5  4.893769
## 6   6  5.640920
## 7   7  4.065206
## 8   8  7.004427
## 9   9 10.694629
## 10 10 13.502619
## 11 11  9.423287
## 12 12 14.405571
## 13 13 15.217048
## 14 14 16.021178
## 15 15 12.198528
## 16 16 20.058743
## 17 17 21.800171
## 18 18 12.804916
## 19 19 18.587329
## 20 20 13.678728
```

Linear Regression

Fit a simple linear regression model and view the analysis.

```
fm <- lm(y ~ x, data = dummy)
summary(fm)
```

```
##
## Call:
## lm(formula = y ~ x, data = dummy)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -5.5572 -1.8009 -0.1278  2.2062  5.2628
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.2454     1.3864   0.898    0.381
## x             0.8995     0.1157   7.773 3.69e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.984 on 18 degrees of freedom
## Multiple R-squared:  0.7704, Adjusted R-squared:  0.7577
## F-statistic: 60.41 on 1 and 18 DF,  p-value: 3.686e-07
```

Weighted Regression

Perform a weighted regression.

```
fm1 <- lm(y ~ x, data = dummy, weight = 1 / w^2)
summary(fm1)
```

```
##
## Call:
## lm(formula = y ~ x, data = dummy, weights = 1/w^2)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -1.76433 -0.70994 -0.09207  0.79521  1.68467
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.08324    0.96550   1.122    0.277
## x            0.91525    0.09845   9.297 2.71e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.067 on 18 degrees of freedom
## Multiple R-squared:  0.8276, Adjusted R-squared:  0.8181
## F-statistic: 86.43 on 1 and 18 DF,  p-value: 2.71e-08
```

Variables from Data Frame

Make the columns in the data frame visible as variables and can be accessed by simply giving their names.

Usage

```
attach(dummy)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     x
```

Nonparametric Local Regression

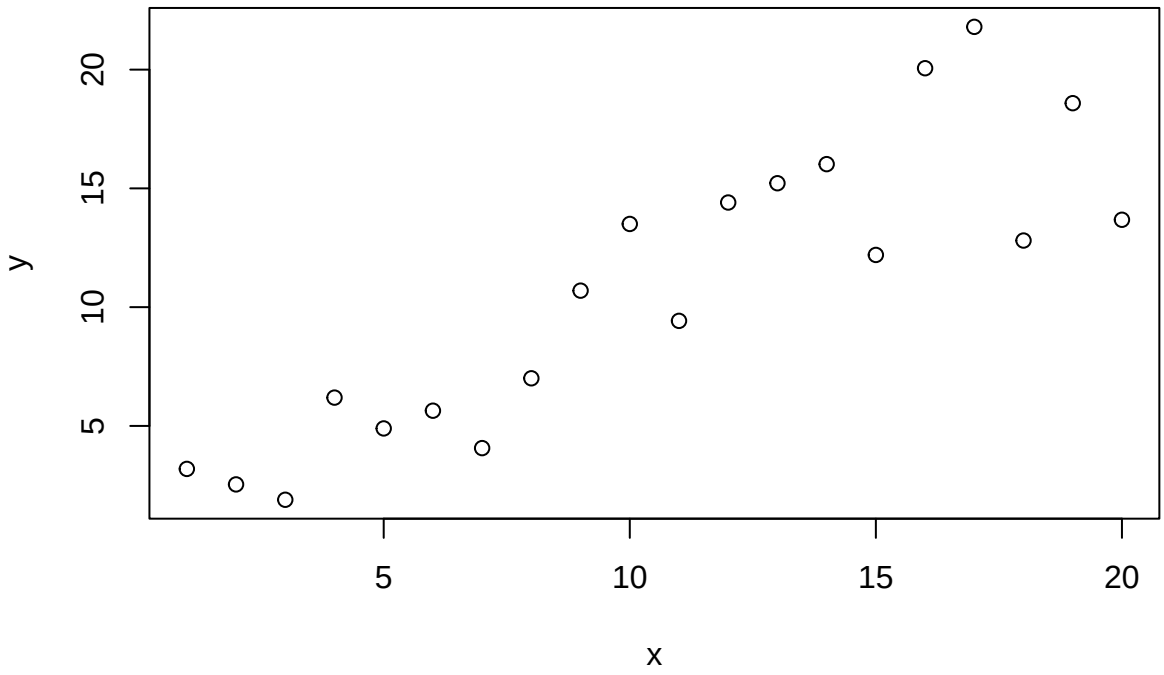
Create a nonparametric local regression function.

```
lrf <- lowess(x, y)
```

Plotting

Generate standard point plots.

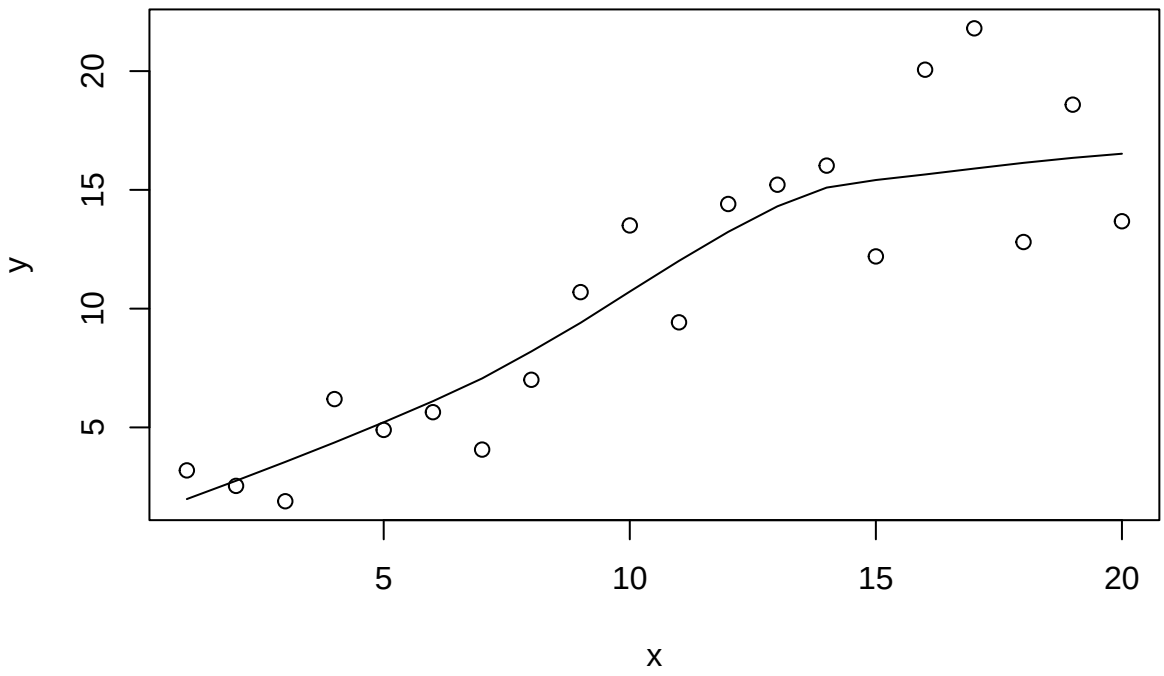
```
plot(x, y)
```



Adding Local Regression

Add the local regression to the plot.

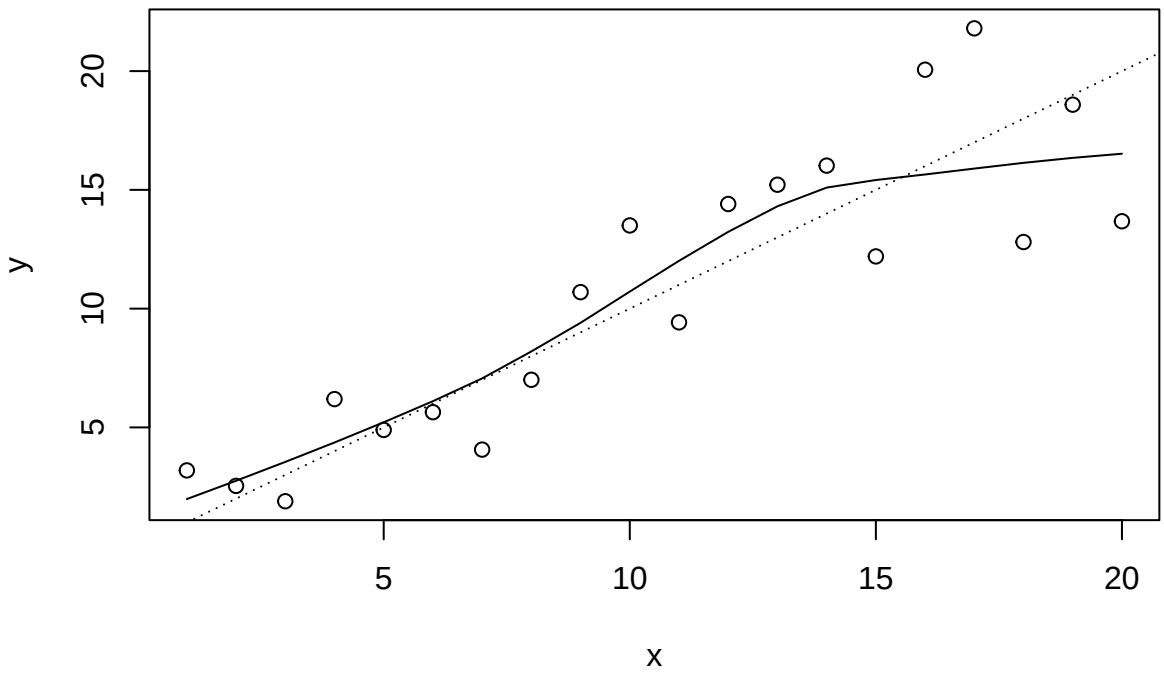
```
plot(x, y)
lines(x, lrf$y)
```



True Regression Line

Include the true regression line.

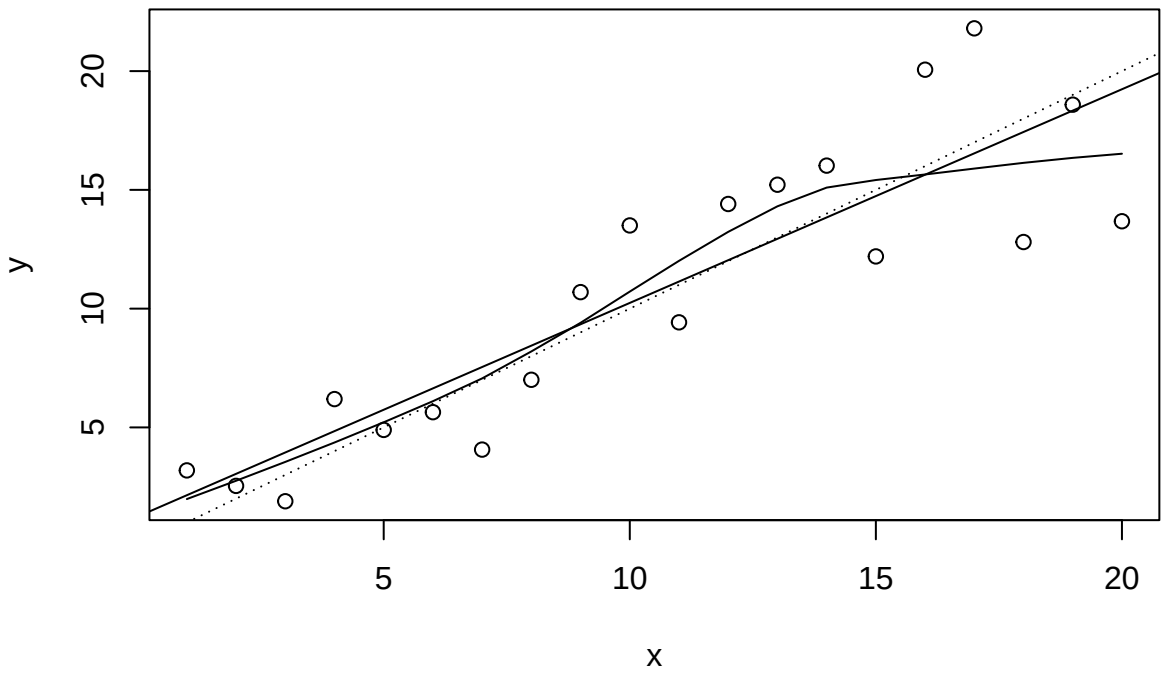
```
plot(x, y)
lines(x, lrf$y)
abline(0, 1, lty = 3)
```



Unweighted Regression Line

Plot the unweighted regression line.

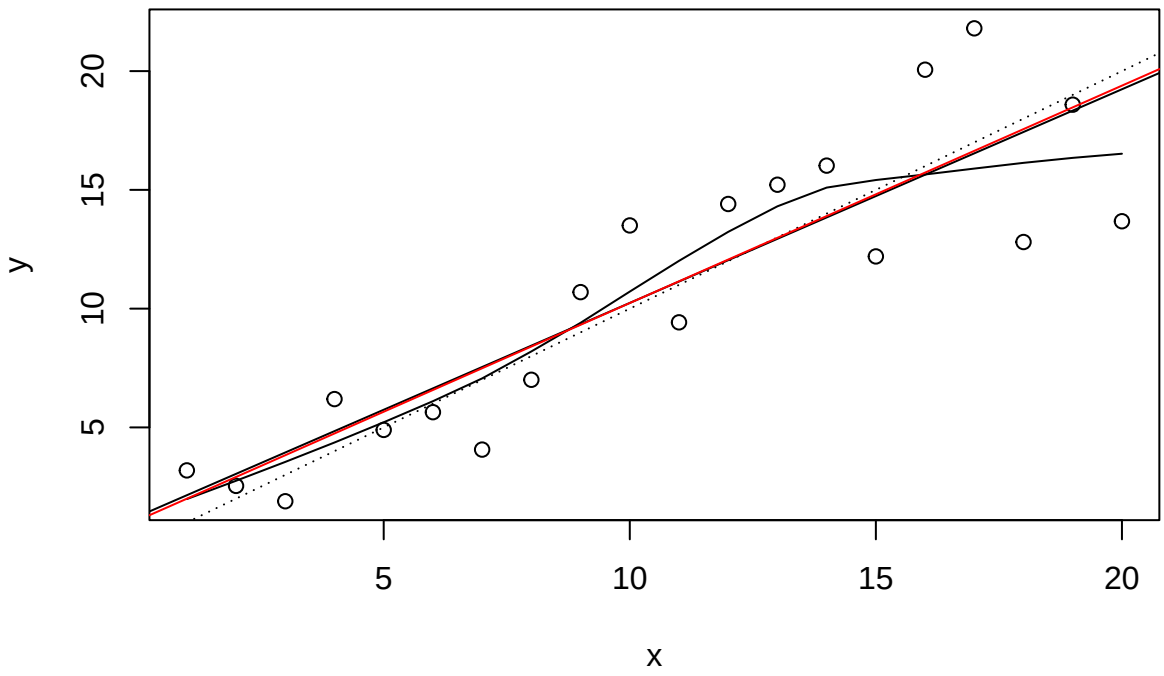
```
plot(x, y)
lines(x, lrf$y)
abline(0, 1, lty = 3)
abline(coef(fm))
```



Weighted Regression Line

Plot the weighted regression line.

```
plot(x, y)
lines(x, lrf$y)
abline(0, 1, lty = 3)
abline(coef(fm))
abline(coef(fm1), col = "red")
```



Data Frame Cleanup

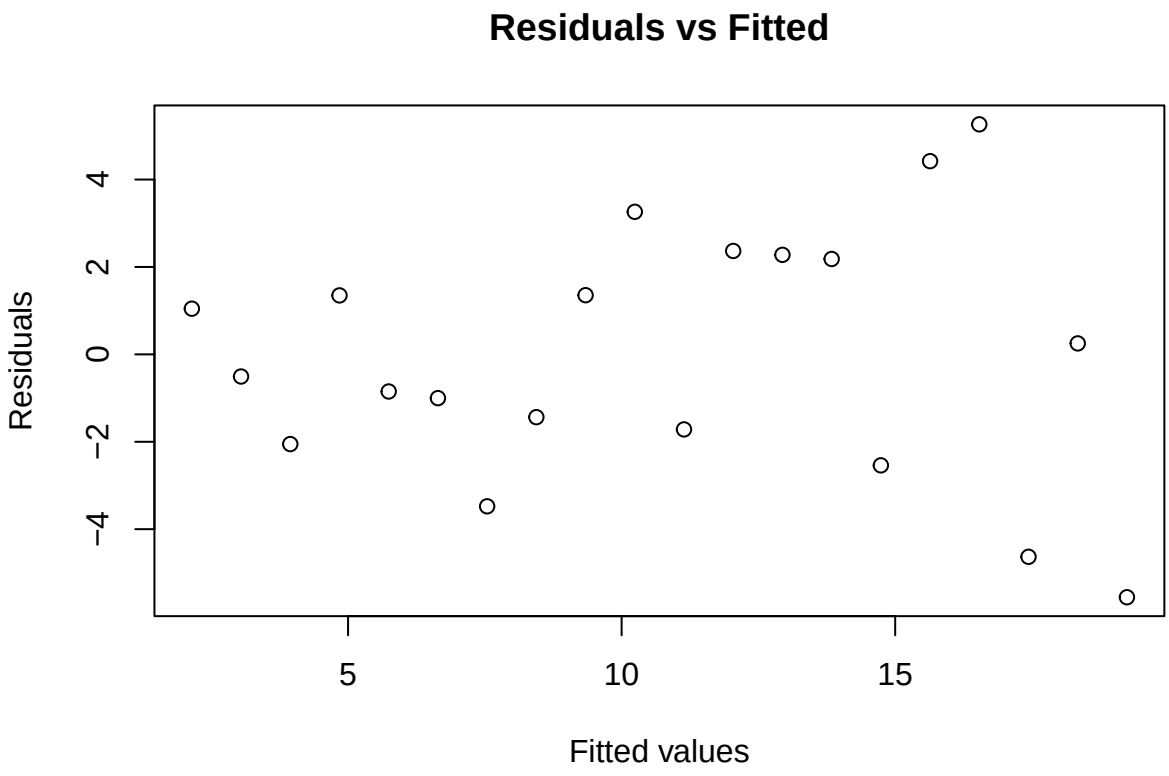
Remove the data frame from the search path.

```
detach()
```

Regression Diagnostic Plot

Create a standard regression diagnostic plot to check for heteroscedasticity.

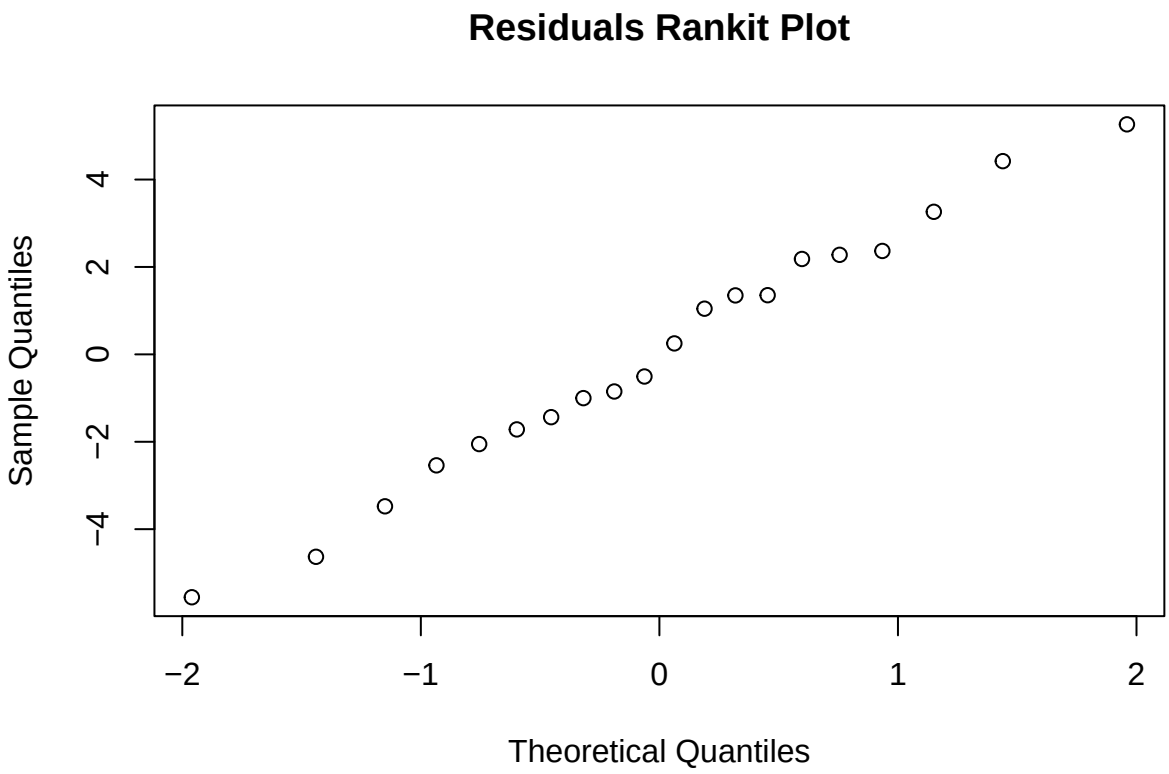
```
plot(fitted(fm), resid(fm),
     xlab = "Fitted values",
     ylab = "Residuals",
     main = "Residuals vs Fitted")
```



Normal Scores Plot

Generate a normal scores plot to check for skewness, kurtosis, and outliers.

```
qqnorm(resid(fm), main = "Residuals Rankit Plot")
```



Cleanup

Clean up the workspace.

```
rm(fm, fm1, lrf, x, dummy)
```

Michelson Classical Experiment

The next section explores data from the classical experiment of Michelson to measure the speed of light.

Data File Path

Get the path to the data file.

```
filepath <- system.file("data", "morley.tab", package = "datasets")
filepath
```

```
## [1] "C:/PROGRA~1/R/R-43~1.1/library/datasets/data/morley.tab"
```

Optional File Review

Optionally, you can review the data file.

```
# file.show(filepath)
```

Read Data

Read in the Michelson data as a data frame and examine it.

```
mm <- read.table(filepath)
head(mm)
```

```
##     Expt Run Speed
## 001    1   1   850
## 002    1   2   740
## 003    1   3   900
## 004    1   4  1070
## 005    1   5   930
## 006    1   6   850
```

```
tail(mm)
```

```
##     Expt Run Speed
## 095    5  15   810
## 096    5  16   940
## 097    5  17   950
## 098    5  18   800
## 099    5  19   810
## 100    5  20   870
```

Change Data Types

Change Expt and Run into factors.

```
mm$Expt <- factor(mm$Expt)
mm$Run <- factor(mm$Run)
```

Attach Data Frame

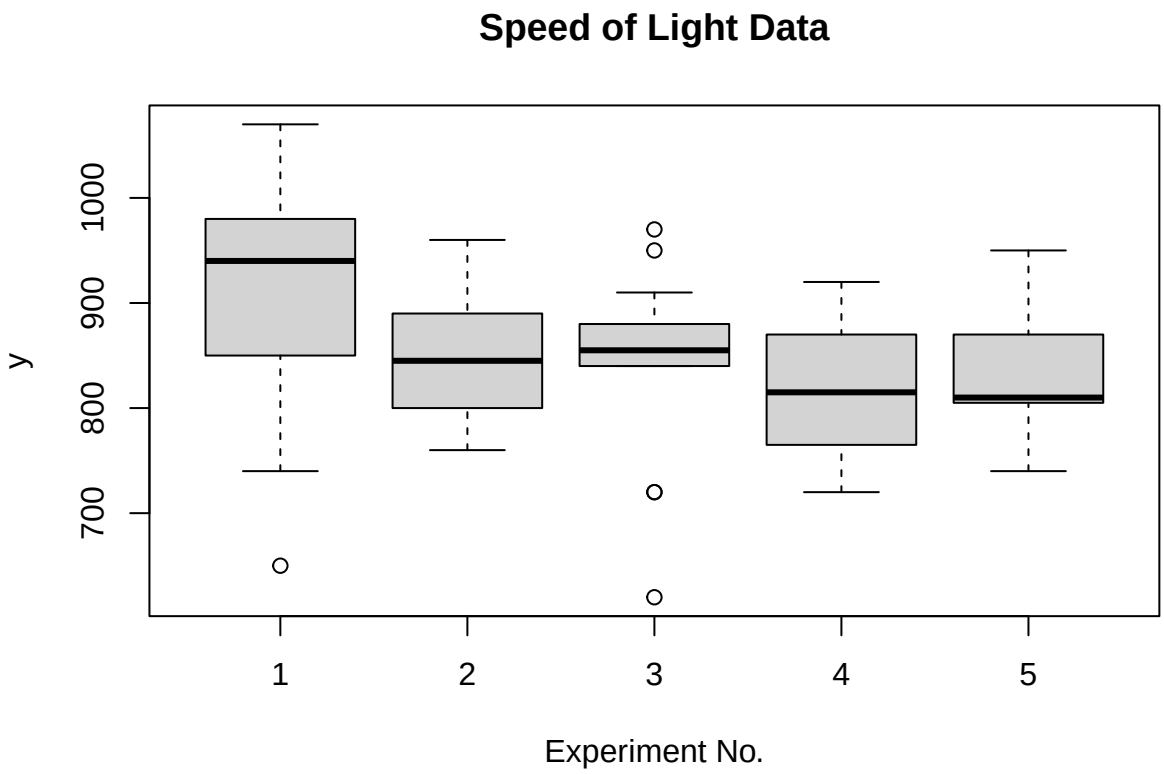
Make the data frame visible at position 2 (the default).

```
attach(mm)
```

Boxplots

Compare the five experiments with simple boxplots.

```
plot(Expt, Speed, main = "Speed of Light Data", xlab = "Experiment No.")
```



Analyze Data

Analyze the data as a randomized block, with 'runs' and 'experiments' as factors.

```
fm <- aov(Speed ~ Run + Expt, data = mm)
summary(fm)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## Run         19 113344    5965   1.105 0.36321
## Expt         4  94514   23629   4.378 0.00307 **
## Residuals   76 410166    5397
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Sub-Model Analysis

Fit a sub-model omitting 'runs' and compare using a formal analysis of variance.

```
fm0 <- update(fm, . ~ . - Run)
anova(fm0, fm)
```

```
## Analysis of Variance Table
##
## Model 1: Speed ~ Expt
## Model 2: Speed ~ Run + Expt
##   Res.Df    RSS Df Sum of Sq      F Pr(>F)
## 1     95 523510
## 2     76 410166 19    113344 1.1053 0.3632
```

Clean Up

Clean up before moving on.

```
detach()
rm(fm, fm0)
```

Graphical Features

Contour and Image Plots

In this section, we explore contour and image plots.

Data Preparation

Prepare data for plotting.

```
x <- seq(-pi, pi, len = 50)
y <- x
```

Create a Matrix

Create a square matrix of values for the function cos(y) / (1 + x^2).

```
f <- outer(x, y, function(x, y) cos(y) / (1 + x^2))
```

Set Plotting Parameters

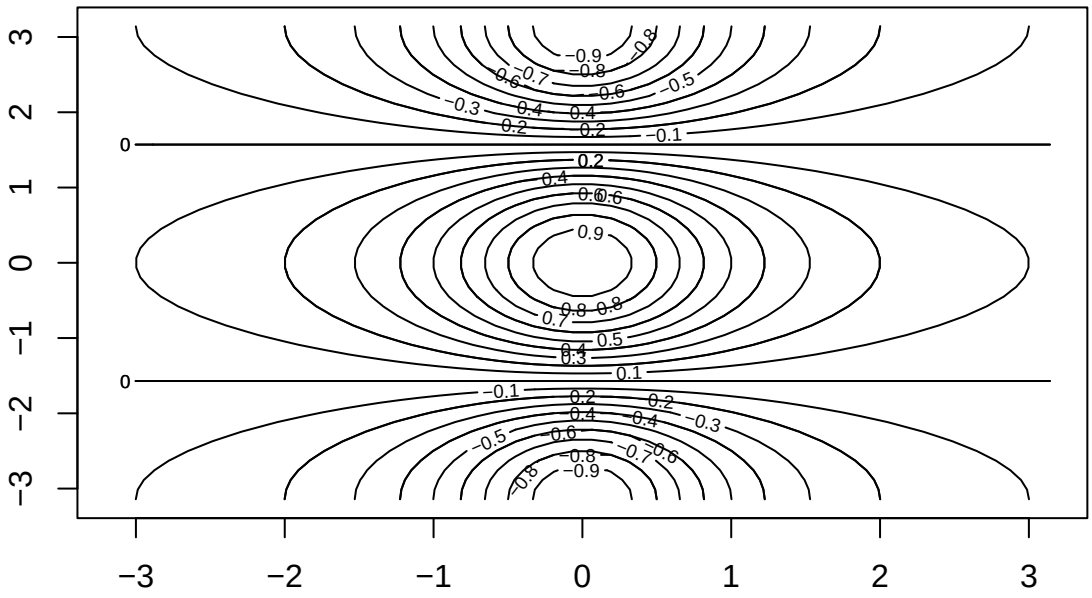
Set the plotting parameters and the plotting region to “square.”

```
oldpar <- par(no.readonly = TRUE)
par(pty = "s")
```

Contour Map

Make a contour map of f and add more lines for detail.

```
contour(x, y, f)
contour(x, y, f, nlevels = 15, add = TRUE)
```



Asymmetric Part

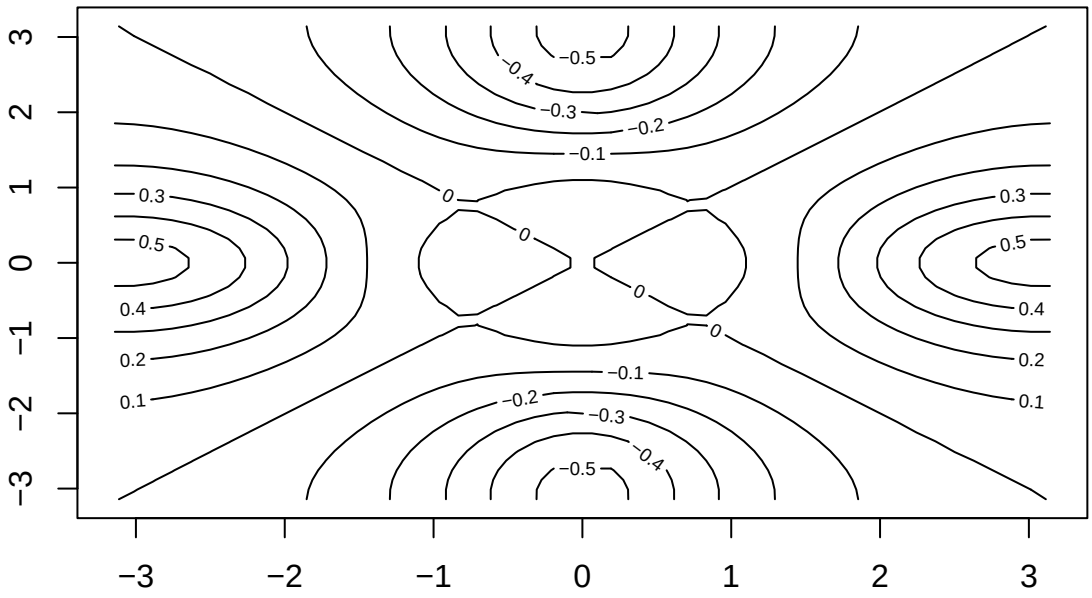
Calculate the "asymmetric part" of f.

```
fa <- (f - t(f)) / 2
```

Contour Plot

Make a contour plot.

```
contour(x, y, fa, nlevels = 15)
```



Restore Graphics Parameters

Restore the old graphics parameters.

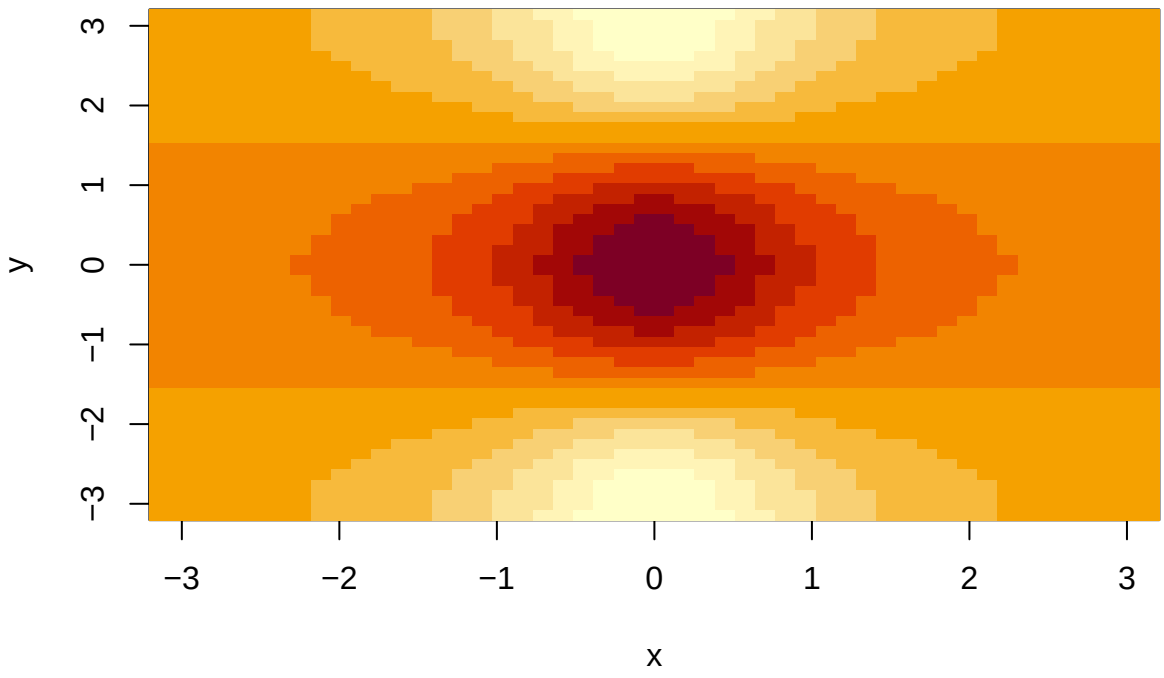
```
par(oldpar)
```

Image Plots

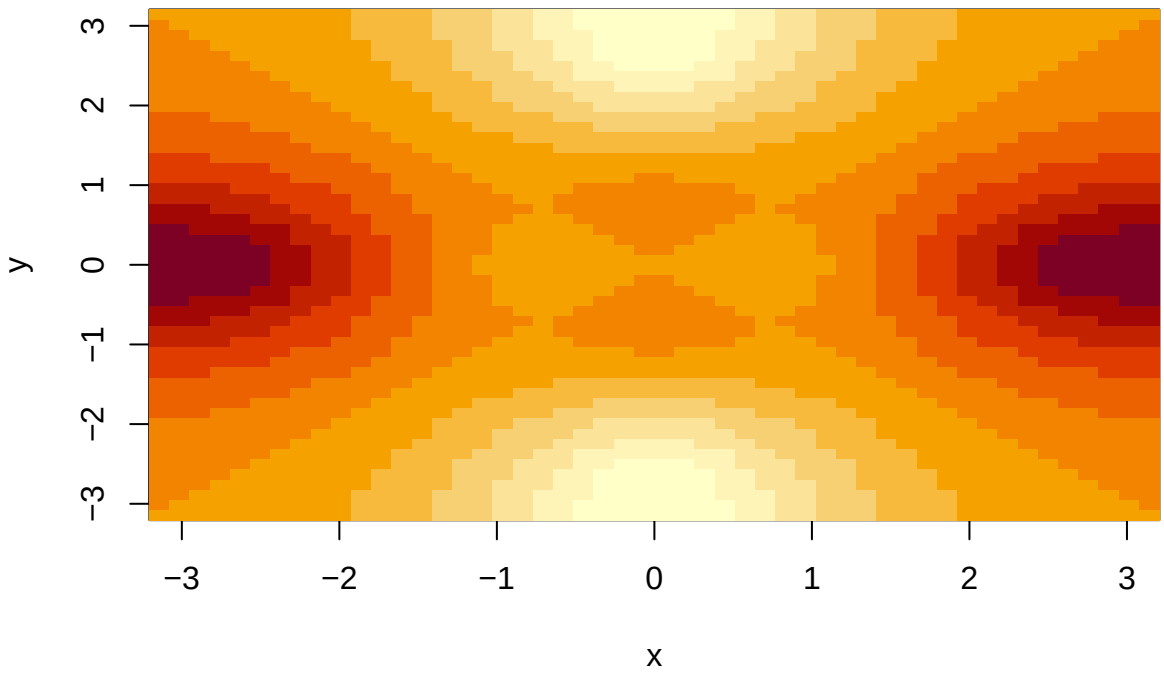
Create high-density image plots.

```
image(x, y,

 f)
```




```
image(x, y, fa)
```



Clean Up

Clean up before moving on.

```
objects()
```

```
## [1] "f"        "fa"       "filepath" "mm"       "oldpar"   "w"        "x"
## [8] "y"
```

```
rm(x, y, f, fa)
```

Complex Arithmetic

In this section, we explore complex arithmetic in R.

Complex Numbers

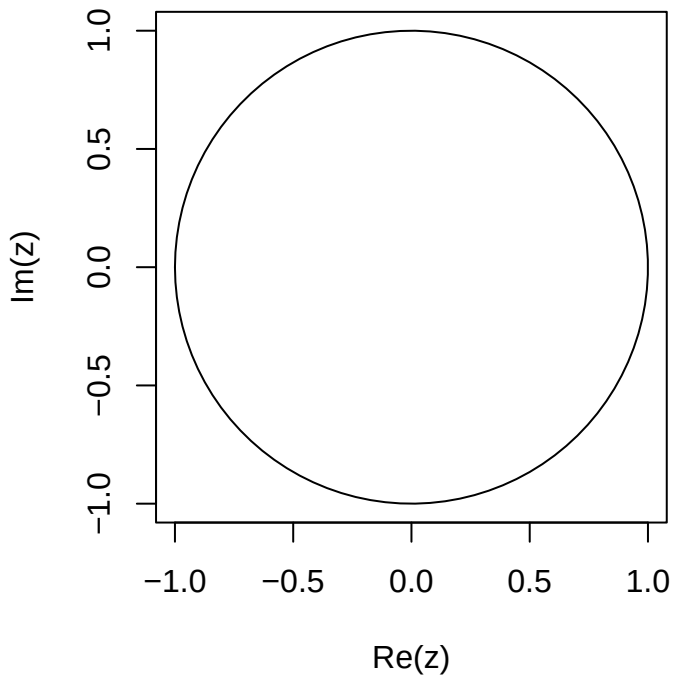
Create complex numbers.

```
th <- seq(-pi, pi, len = 100)
z <- exp(1i * th)
```

Complex Arguments Plot

Plot complex arguments.

```r
par(pty = "s")
plot(z, type = "l")
```



Sampling Points

Sample points within the unit circle using two methods.

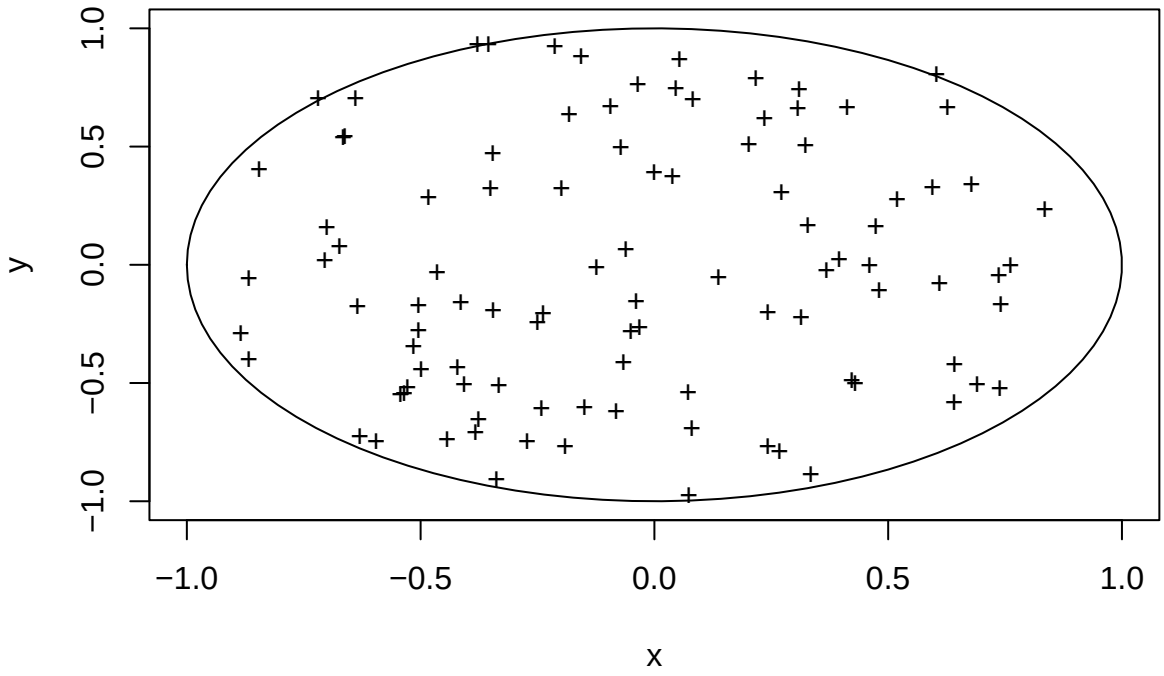
```r
# Method 1
w <- rnorm(100) + rnorm(100) * 1i
w <- ifelse(Mod(w) > 1, 1 / w, w)

# Method 2
w <- sqrt(runif(100)) * exp(2 * pi * runif(100) * 1i)
```

Plot Sampling Results

Plot the sampled points.

```r
plot(w, xlim = c(-1, 1), ylim = c(-1, 1), pch = "+", xlab = "x", ylab = "y")
lines(z)
```



Cleanup

Clean up the workspace.

```
rm(th, w, z)
```

Quit R

You will be asked if you want to save the R workspace. For an exploratory session like this, you probably do not want to save it.

```
# q()
```

This document provides a comprehensive introduction to R, covering various aspects of the language and its capabilities. “‘

This revised document is better organized, making it easier to read and follow. Each section is clearly labeled, and code blocks are appropriately formatted. Additionally, I added some headings to provide structure and improve readability.