Based on:

# An Introduction to R

## Notes on R: A Programming Environment for Data Analysis and Graphics

### Version 3.6.0 (2019-04-26)

#### W. N. Venables, D. M. Smith and the R Core Team

# Generate two pseudo-random normal vectors of x- and y-coordinates.

x <- rnorm(20)

y <- rnorm(x)

# Plot the points in the plane. A graphics window will appear automatically.

plot(x, y)



# See which R objects are now in the R workspace.

ls()

# Remove objects no longer needed. (Clean up).

rm(x)

# Make x = (1, 2, . . . , 20).

x <- 1:20

# A 'weight' vector of standard deviations.

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

# Make a data frame of two columns, x and y, and look at it.

df <- data.frame(x=x, y=y + rnorm(x)\*w)

df

# Fit a simple linear regression and look at the analysis.

fm <- lm(y ~ x, data=df)

# Summary data

summary(fm)

# Raw data

fm

round(anova(fm), 3)

Index Df Sum Sq Mean Sq F value Pr(>F)

x 1 3.261 3.261 0.856 0.36

Residuals 48 182.944 3.811

# With y to the left of the tilde, we are modelling y dependent on x.

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

# Since we know the standard deviations, we can do a weighted regression.

summary(fm1)

# Make the columns in the data frame visible as variables.

attach(df)

x

y

# Make a nonparametric local regression function.

lrf <- lowess(x, y)

lrf$x

# Standard point plot.

plot(x, y)



# Add in the local regression.

lines(x, lrf$y)

# The true regression line: (intercept 0, slope 1).

abline(0, 1, lty=3)

# Unweighted regression line.

abline(coef(fm))

# Weighted regression line.

abline(coef(fm1), col = "red")



# Remove data frame from the search path.

detach()

# A standard regression diagnostic plot to check for heteroscedasticity. Can you see it?

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



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

$'x'

[1] -1.43953147093846

[2] -1.15034938037601

[3] -0.453762190169879

[4] -0.93458929107348

[5] 0.93458929107348

[6] 1.95996398454005

[7] 1.15034938037601

[8] 0.597760126042478

[9] -0.755415026360469

[10] -0.318639363964375

[11] 0.755415026360469

[12] 0.45376219016988

[13] 1.43953147093846

[14] 0.0627067779432138

[15] 0.189118426272792

[16] -1.95996398454005

[17] -0.0627067779432138

[18] 0.318639363964375

[19] -0.189118426272792

[20] -0.597760126042478

$'y'

[1] -2.9527431701389

[2] -2.49760213384058

[3] -1.47810422609728

[4] -2.37651516452467

[5] 2.71524947253966

[6] 4.21727762541659

[7] 3.15928638266353

[8] 1.88406894368372

[9] -2.31239277966191

[10] -1.23390059730553

[11] 2.13954410649769

[12] 0.679579603490292

[13] 3.81831156734968

[14] 0.108789902466678

[15] 0.213620358778393

[16] -3.539699910075

[17] -0.0425741069869814

[18] 0.296089226341137

[19] -0.911091933566069

[20] -1.88719316703045

# A normal scores plot to check for skewness, kurtosis and outliers. (Not very useful here.)



# Clean up again.

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

# Get the path to the data file.

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

filepath

# Optional. Look at the file.

file.show(filepath)

# Read in the Michelson data as a data frame, and look at it. There are five experiments (column Expt) and each has 20 runs (column Run) and sl is the recorded speed of light, suitably coded.

mm <- read.table(filepath)

mm

# Change Expt and Run into factors.

mm$Expt <- factor(mm$Expt)

mm$Expt

mm$Run <- factor(mm$Run)

# Make the data frame visible at position 3 (the default).

attach(mm)

# Compare the five experiments with simple boxplots.

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



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

summary(fm)

[[1]]

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Index | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
| Run | 19 | 113344 | 5965.47368421052 | 1.10534759097536 | 0.36320934066802 |
| Expt | 4 | 94514.0000000002 | 23628.5 | 4.37814445858507 | 0.00307058926287645 |
| Residuals | 76 | 410166 | 5396.92105263158 |  |  |

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

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

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

anova(fm0, fm)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Index | Res.Df | RSS | Df | Sum of Sq | F | Pr(>F) |
| 1 | 95 | 523510 |  |  |  |  |
| 2 | 76 | 410166 | 19 | 113344 | 1.10534759097536 | 0.363209340668021 |

detach()

# Clean up before moving on.

rm(fm, fm0)

#

# We now look at some more graphical features: contour and image plots.

# x is a vector of 50 equally spaced values in ????? ??? x ??? ??. y is the same.

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

y <- x

# f is a square matrix, with rows and columns indexed by x and y respectively, of values of the function cos(y)/(1 + x2).

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

oldpar <- par(no.readonly = TRUE)

# Save the plotting parameters and set the plotting region to "square".

par(pty = "s")

# Make a contour map of f; add in more lines for more detail.

contour(x, y, f)

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



# fa is the "asymmetric part" of f. (t() is transpose).

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

# Make a contour plot, . . .

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



# . . . and restore the old graphics parameters.

par(oldpar)

# Make some high density image plots, (of which you can get hardcopies if you wish), . . .

image(x, y, f)



image(x, y, fa)

#objects();

# . . . and clean up before moving on.

rm(x, y, f, fa)

# R can do complex arithmetic, also.

th <- seq(-pi, pi, len=100)

z <- exp(1i\*th)

# 1i is used for the complex number i.

par(pty="s")

# Plotting complex arguments means plot imaginary versus real parts. This should be a circle.

plot(z, type="l")



# Suppose we want to sample points within the unit circle. One method would be to take complex numbers with standard normal real and imaginary parts...

w <- rnorm(100) + rnorm(100)\*1i

w <- ifelse(Mod(w) > 1, 1/w, w)

# . . . and to map any outside the circle onto their reciprocal.

# All points are inside the unit circle, but the distribution is not uniform.

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

lines(z)

# The second method uses the uniform distribution. The points should now look more evenly spaced over the disc.

w <- sqrt(runif(100))\*exp(2\*pi\*runif(100)\*1i)

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

lines(z)

# Clean up again.

rm(th, w, z)