StatR 101: Fall 2012

Homework 4

Rod Doe

Saturday, October 20, 2012

# News

I got the book “R in a Nutshell” this week. I had the Kindle version, and found it to be a clunky reference. The book is really good, and summarizes the various function arguments quite effectively. This helps me to remember oddities like legend is an argument to the legend function.

# The Fibonacci sequence:

## The original:

### Code:

# Get all Fibonacci numbers less than 300

Fib1 <- 1

Fib2 <- 1

Fibonacci <- c(Fib1, Fib2)

while (Fib2 < 300) {

Fibonacci <- c(Fibonacci, Fib2)

oldFib2 <- Fib2

Fib2 <- Fib1 + Fib2

Fib1 <- oldFib2

}

### Result:

> Fibonacci

[1] 1 1 1 2 3 5 8 13 21 34 55 89 144 233

## The original, sans oldFib2:

### Code:

# Get all Fibonacci numbers less than 300

Fib1 <- 1

Fib2 <- 1

Fibonacci <- c(Fib1, Fib2)

while (Fib2 < 300) {

Fibonacci <- c(Fibonacci, Fib2)

Fib2 <- Fib1 + Fib2

Fib1 <- Fibonacci[length(Fibonacci)]

}

### Result:

> Fibonacci

[1] 1 1 1 2 3 5 8 13 21 34 55 89 144 233

## Sans Fib1 and Fib:

### Code:

# Get all Fibonacci numbers less than 300

Fibonacci <- c(1,1)

while ((Fibonacci[length(Fibonacci) - 1] + Fibonacci[length(Fibonacci)]) < 300) {

Fibonacci <- c(Fibonacci, Fibonacci[length(Fibonacci) - 1] + Fibonacci[length(Fibonacci)])

}

### Result:

> Fibonacci

[1] 1 1 2 3 5 8 13 21 34 55 89 144 233

# Get all Fibonacci numbers less than 300

## Determine the count of Fibonacci numbers less than 1E6

After about ten minutes, I concluded that this was a trick. However, I found a website that reported that there are 78,489 Fibonacci numbers less than 1,000,000. When I came to work the next morning, my machine confirmed that result.

# Taylor Series

## In preparation, a factorial function:

factorial<-function(x) {

result = NA

ix = as.integer(x)

if (ix == 0) {

result = 1

}

else if (ix > 0) {

result = prod(seq(1, as.integer(x), 1))

}

return(result)

}

## The TaylorSine function:

# Sine function is infinitely differentiable, but repeats itself after 4 orders.

# We are evaluating at x= 0, so exploit that to simplify.

# sin(0) = 0, cos(0) = 1

# Any term with sine evaluates to 0 since sin(0) = 0, so 0, 2, 4 .. are all zero.

# Any term with cosine evaluates to 1 since cos(0) = 1. Sign matters...

# All we care about are the modulo 1st and 3rd terms.

TaylorSine <- function(x, kTerms) {

result = 0

for (i in 1:as.integer(kTerms)) {

modTerm <- i %% 4

if (1 == modTerm) {

result = result + (x^i / factorial(i))

}

else if (3 == modTerm) {

result = result - (x^i / factorial(i))

}

}

return(result)

}

## Third Order Result

Here is the third order approximation of sin(x) by TaylorSine(x) for range -2pi to 2pi

> # Take TaylorSine for a test drive.

> x <- seq(-2\*pi, 2\*pi, 0.01)

> y <- numeric(length(x))

> plot(x, y, xlim=c(-2\*pi, 2\*pi), ylim=c(-2,2), type="l", col=1)

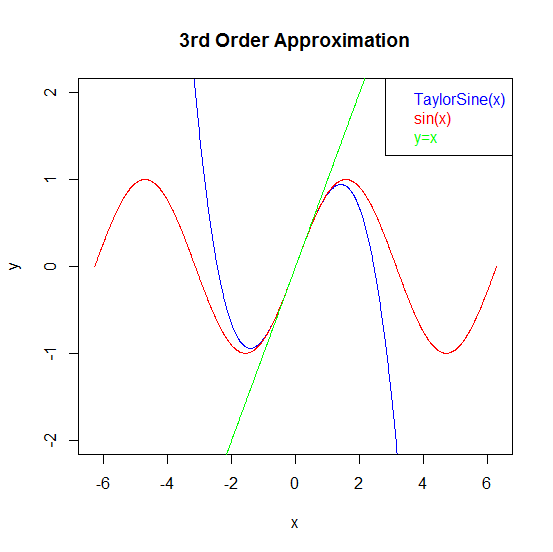
> for (i in 1:length(x)) {y[i] = TaylorSine(x[i], 3)}

> plot(x, y, xlim=c(-2\*pi, 2\*pi), ylim=c(-2,2), type="l", col=rgb(0,0,1), main="3rd Order Approximation")

> lines(x,sin(x), col=rgb(1,0,0))

> lines(x,x, col=rgb(0,1,0))

> legend("topright", legend=c("TaylorSine(x)", "sin(x)", "y=x"), text.col=c(rgb(0,0,1), rgb(1,0,0), rgb(0,1,0)))



Good approximation around 0, but diverges far from 0. This is about what I expected.

## Just for fun, let’s try the 5th order approximation

x <- seq(-2\*pi, 2\*pi, 0.01)

y <- numeric(length(x))

plot(x, y, xlim=c(-2\*pi, 2\*pi), ylim=c(-2,2), type="l", col=1)

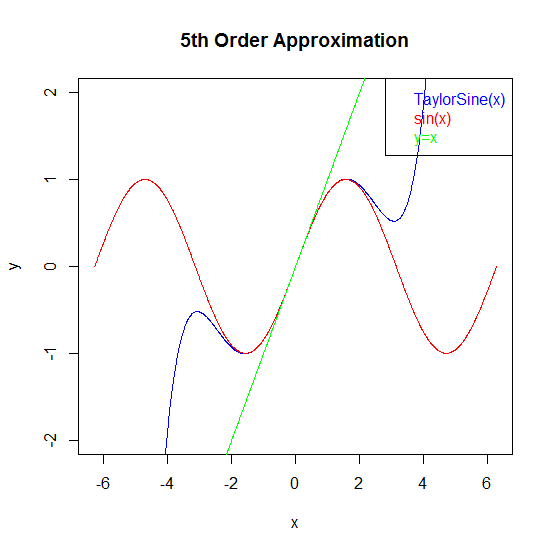
for (i in 1:length(x)) {y[i] = TaylorSine(x[i], 5)}

plot(x, y, xlim=c(-2\*pi, 2\*pi), ylim=c(-2,2), type="l", col=rgb(0,0,1), main="5th Order Approximation")

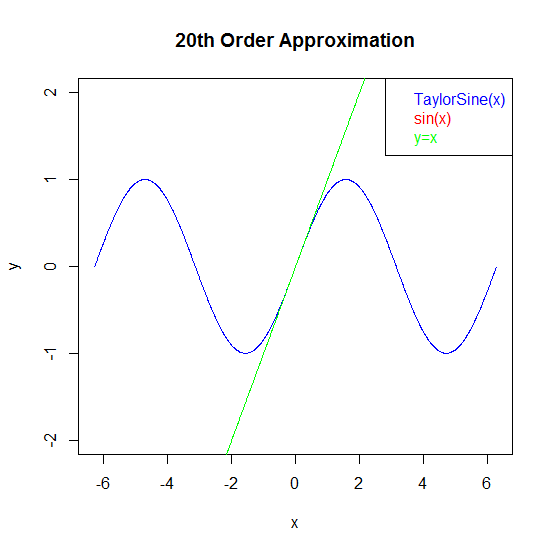
lines(x,sin(x), col=rgb(1,0,0))

lines(x,x, col=rgb(0,1,0))

legend("topright", legend=c("TaylorSine(x)", "sin(x)", "y=x"), text.col=c(rgb(0,0,1), rgb(1,0,0), rgb(0,1,0)))



That is certainly weird. Tight approximation around 0, then quite goofy farther away from 0. Just for grins, let’s try 20th order.



This approximation is quite good. Initially, the blue approximation was covered by the red sine curve. However, when I removed the red sine curve, the blue approximation appeared – they were superimposed! I expected the upper order terms to be consumed by their large factorial denominators.

## Plots of approximations in seq(2, 20,2)

x <- seq(-2\*pi, 2\*pi, 0.05)

y <- numeric(length(x))

orders <- seq(2, 20, 2)

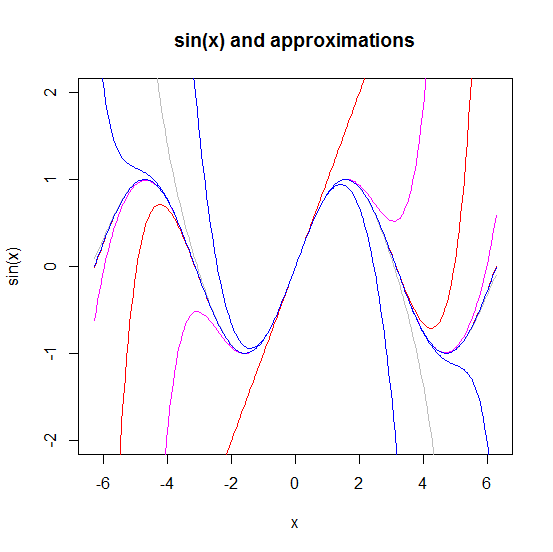
plot(x, sin(x), xlim=c(-2\*pi, 2\*pi), ylim=c(-2,2), type="l", col=1, main="sin(x) and approximations")

for (n in orders) {

for (i in 1:length(x)) {y[i] = TaylorSine(x[i], n)}

lines(x, y, col=n)

}



This might make a good Halloween decoration.

# Random variables 1

## Sample standard deviation of 1000 random samples standard uniform distribution

> n <- runif(1000, 0, 1)

> sd(n)

[1] 0.2781392

## Function to return the sd of n draws from Unif(0, b)

sdUniformDistSample <- function(n, b) {

result = NA

if (n > 0 && b > 0) {

result = sd(runif(n, 0, b))

}

return(result)

}

## Predict model of sx with upper limit b

sds <- numeric(10)

for (i in 1:10) {

+ sds[i] <- sdUniformDistSample(1000, i)

+ }

sds

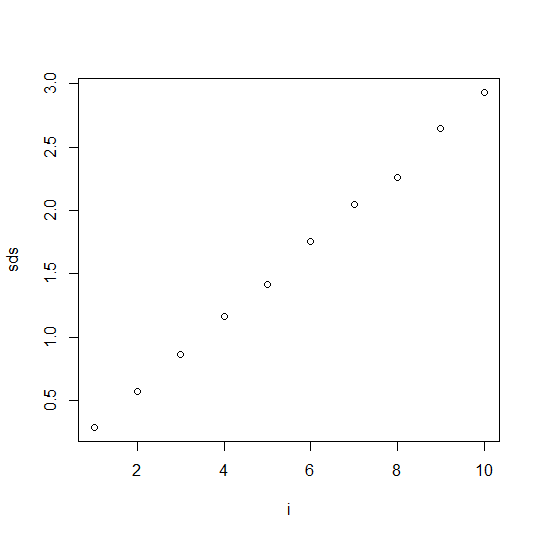
[1] 0.2855838 0.5716266 0.8632658 1.1625357 1.4126796 1.7592715 2.0488086

[8] 2.2631829 2.6503564 2.9353257

The data look vaguely linear. Let’s plot them.

> i <- seq(1:10)

> plot(i, sds)

This produces:  


Try to fit a linear model to this:

> model <- lm(sds ~ i)

> model

Call:

lm(formula = sds ~ i)

Coefficients:

(Intercept) i

-0.0182 0.2934

So, the model could be sd = 0.2934 x. In this case I ignore the intercept .

# Random variables II

## Generate data

X1 <- runif(1000, 0, 1)

X2 <- runif(1000, 0, 1)

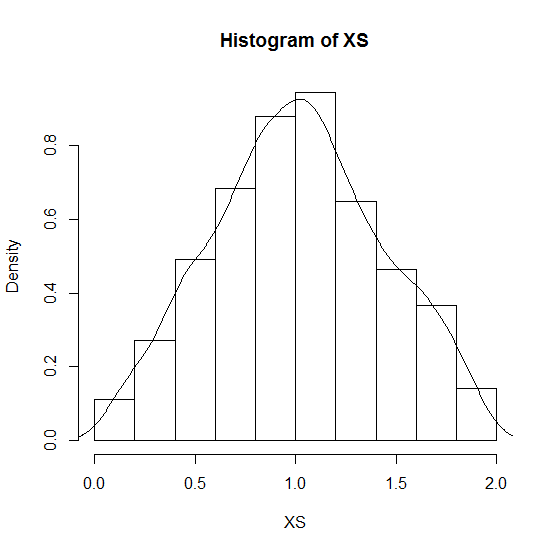
XS = X1 + X2

hist(XS, prob=T)

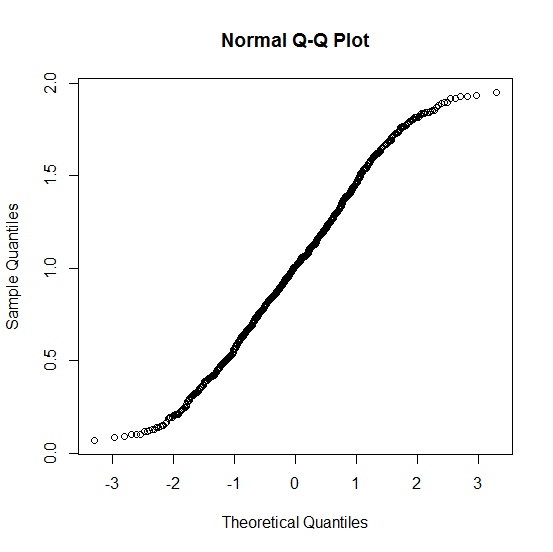
lines(density(XS))

## Guess the probability distribution function and illustrate it

Here is the histogram



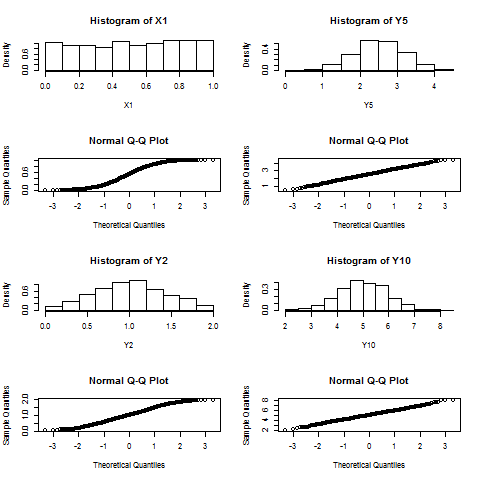
Is it normally distributed? Do a qqnorm(XS) and see what we get.



Vector XS has heavier tails than a normal distribution.

## Sum samples and observe effect on distributions.

As more data summed, the distributions become increasingly normal.



This was a good way to demonstrate how to interpret QQ plots. Way cool!