STAT 321

Charles Hwang

Professor Matthews

STAT 321-001

4 October 2019

Exercise 9.4

```r
rm(list=ls())
# "Truncating to n terms, the error is no greater in magnitude than the last term in the sum."
x <- 1.5-1
l <- log(1+x)
c(x^46/46,x^47/47,10^-16,x^48/48,x^49/49)
```

```
## [1] 3.089316e-16 1.511793e-16 1.000000e-16 7.401487e-17 3.625218e-17
```

```r
# There are 48 terms required to calculate log(1.5) with an error of 10^-16 or less.
x <- 2-1
c(x^9999999999999999/9999999999999999,10^-16,x^10000000000000000/10000000000000000) # (1)^1000000000000
```

```
## [1] 1e-16 1e-16 1e-16
```

```r
# There are 10,000,000,000,000,000 terms required to calculate log(2) with an error of 10^-16 or less.
x <- sqrt(2)-1
c(x^36/36,x^37/37,10^-16,x^38/38,x^39/39)
```

```
## [1] 4.610763e-16 1.858223e-16 1.000000e-16 7.494460e-17 3.024709e-17
```

```r
# By calculating log(sqrt(2)) then multiplying by 2, it significantly reduces the number of terms neede
```

Exercise 9.5

```r
rm(list=ls())
# First formula:                              Second formula:
# S(xi-xbar)^2/(n-1) =
# S(xi^2-2*xi*xbar+xbar^2)/(n-1) =
# (S(xi^2-2*xi*xbar)+n*xbar^2)/(n-1) =
# (S(xi^2)-2*xbar*S(xi)+n*xbar^2)/(n-1) =
# (S(xi^2)-2*xbar(n*xbar)+n*xbar^2)/(n-1) =
# (S(xi^2)-2*n*xbar^2+n*xbar^2)/(n-1) =
# (S(xi^2)-n*xbar^2)/(n-1) =                  (S(xi^2)-n*xbar^2)/(n-1) =
# ... =                                       ... =
# s^2/(n-1) =                                 s^2/(n-1) =
# S^2                                         S^2
# The first formula takes six operations to get to the same point that the second formula begins at.
# The second formula suffers from catastrophic cancellation because the xi's (and xbar) may have a diff
x <- c(.123456789,.123456789123456789) # Choosing two numbers with different decimal lengths
```

```r
n <- length(x)
xbar <- mean(x)
f1 <- sum((x[1]-xbar)^2,(x[n]-xbar)^2)
f2 <- sum(x[1]^2,x[n]^2)-n*xbar^2
c(f1,f2)
```

```
## [1] 7.620789e-21 3.469447e-18
```

```r
# There is a clear difference in the answers for the case where n = 2 due to catastrophic cancellation
```

Exercise 9.9

```r
rm(list=ls())
x <- c(11,12,13,14,15,16,17,18,19,20)
y <- c(21,22,23,24,25,26,27,28,29,30)
z <- rep(0,length(x)+length(y))
sum <- z
n <- mean(length(x),length(y))
for(k in 1:2*n){
  i <- 1
  base <- x[1]*y[k+1]
  while(i < k){
    sum[i] <- x[i]*y[k-i]
    i <- i+1
  }
  z[k] <- base+sum(sum)
}
z
```

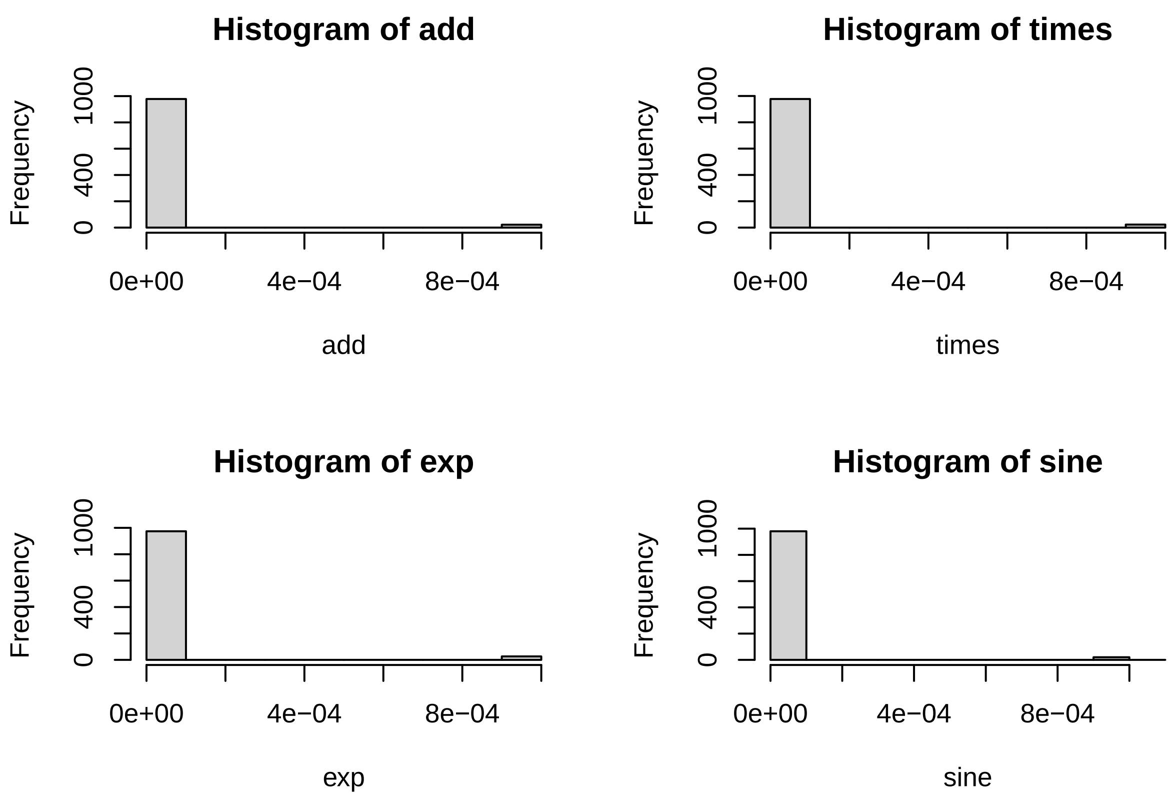
```
##  [1]  0  0  0  0  0  0  0  0  0 NA  0  0  0  0  0  0  0  0  0 NA
```

```r
# z <- c(x[1]y[1],x[1]y[2]+x[2]y[1],x[1]y[3]+x[2]y[2]+x[3]y[1],...,x[1]y[n]+x[2]y[n-1]+...+x[n]y[1],x[2
# z[1] <- x[1]y[1]
# z[2] <- x[1]y[2]+x[2]y[1]
# z[3] <- x[1]y[3]+x[2]y[2]+x[3]y[1]
# z[...]
# z[n] <- x[1]y[n]+x[2]y[n-1]+...+x[n]y[1]
# z[n+1] <- x[2]y[n]+x[3]y[n-1]+...+x[n]y[2]
# z[n+2] <- x[3]y[n]+x[4]y[n-1]+...+x[n]y[3]
# z[...]
# z[2n-1] <- x[n]y[n]
# z[2n] <- -(x[n]y[n+1]+x[n+1]y[n])
```

Exercise 9.10

```r
rm(list=ls())
i <- 1 # Addition
add <- rep(0,1000)
while(i <= 1000) {
rand <- rnorm(2)
add[i] <- system.time(sum(rand))[3]
i = i+1
}
i <- 1 # Multiplication
```

```r
times <- rep(0,1000)
while(i <= 1000) {
rand <- rnorm(2)
times[i] <- system.time(prod(rand))[3]
i = i+1
}
i <- 1 # Exponential
exp <- rep(0,1000)
while(i <= 1000) {
rand <- rnorm(1)
exp[i] <- system.time(exp(rand))[3]
i = i+1
}
i <- 1 # Sinusoidal
sine <- rep(0,1000)
while(i <= 1000) {
rand <- rnorm(1)
sine[i] <- system.time(sin(rand))[3]
i = i+1
}
par(mfrow=c(2,2))
hist(add)
hist(times)
hist(exp)
hist(sine)
```




```r
# The distributions are all nearly identical with one another. As expected, the time it takes to run ea
```

Exercise 10.3

```
rm(list=ls())
library(spuRs)
```

```
## Loading required package: MASS
## Loading required package: lattice
```

```
fixedpoint(cos,0)
```

```
## At iteration 1 value of x is: 1
## At iteration 2 value of x is: 0.5403023
## At iteration 3 value of x is: 0.8575532
## At iteration 4 value of x is: 0.6542898
## At iteration 5 value of x is: 0.7934804
## At iteration 6 value of x is: 0.7013688
## At iteration 7 value of x is: 0.7639597
## At iteration 8 value of x is: 0.7221024
## At iteration 9 value of x is: 0.7504178
## At iteration 10 value of x is: 0.731404
## At iteration 11 value of x is: 0.7442374
## At iteration 12 value of x is: 0.7356047
## At iteration 13 value of x is: 0.7414251
## At iteration 14 value of x is: 0.7375069
## At iteration 15 value of x is: 0.7401473
## At iteration 16 value of x is: 0.7383692
## At iteration 17 value of x is: 0.7395672
## At iteration 18 value of x is: 0.7387603
## At iteration 19 value of x is: 0.7393039
## At iteration 20 value of x is: 0.7389378
## At iteration 21 value of x is: 0.7391844
## At iteration 22 value of x is: 0.7390183
## At iteration 23 value of x is: 0.7391302
## At iteration 24 value of x is: 0.7390548
## At iteration 25 value of x is: 0.7391056
## At iteration 26 value of x is: 0.7390714
## At iteration 27 value of x is: 0.7390944
## At iteration 28 value of x is: 0.7390789
## At iteration 29 value of x is: 0.7390893
## At iteration 30 value of x is: 0.7390823
## At iteration 31 value of x is: 0.739087
## At iteration 32 value of x is: 0.7390838
## At iteration 33 value of x is: 0.739086
## At iteration 34 value of x is: 0.7390845
## At iteration 35 value of x is: 0.7390855
## At iteration 36 value of x is: 0.7390849
## At iteration 37 value of x is: 0.7390853
## At iteration 38 value of x is: 0.739085
## At iteration 39 value of x is: 0.7390852
## At iteration 40 value of x is: 0.7390851
## At iteration 41 value of x is: 0.7390852
## At iteration 42 value of x is: 0.7390851
## At iteration 43 value of x is: 0.7390851
## At iteration 44 value of x is: 0.7390851
```

```
## At iteration 45 value of x is: 0.7390851
## At iteration 46 value of x is: 0.7390851
## At iteration 47 value of x is: 0.7390851
## At iteration 48 value of x is: 0.7390851
## At iteration 49 value of x is: 0.7390851
## At iteration 50 value of x is: 0.7390851
## At iteration 51 value of x is: 0.7390851
## At iteration 52 value of x is: 0.7390851
## At iteration 53 value of x is: 0.7390851
## Algorithm converged
```

```
## [1] 0.7390851
```

```
cosx <- function(x){
  c(cos(x)-x,-sin(x)-1)
}
newtonraphson(cosx,0)
```

```
## At iteration 1 value of x is: 1
## At iteration 2 value of x is: 0.7503639
## At iteration 3 value of x is: 0.7391129
## At iteration 4 value of x is: 0.7390851
## Algorithm converged
```

```
## [1] 0.7390851
```

```
# Yes, the Newton-Raphson method is faster than the fixed-point method.
```

Exercise 10.10a

```
rm(list=ls())
library(spuRs)
sine <- function(x){
  c(sin(x),cos(x))
}
nr <- newtonraphson(sine,3)
```

```
## At iteration 1 value of x is: 3.142547
## At iteration 2 value of x is: 3.141593
## Algorithm converged
```

```
cat("The Newton-Raphson method was able to get within",abs(nr-pi),"of pi.")
```

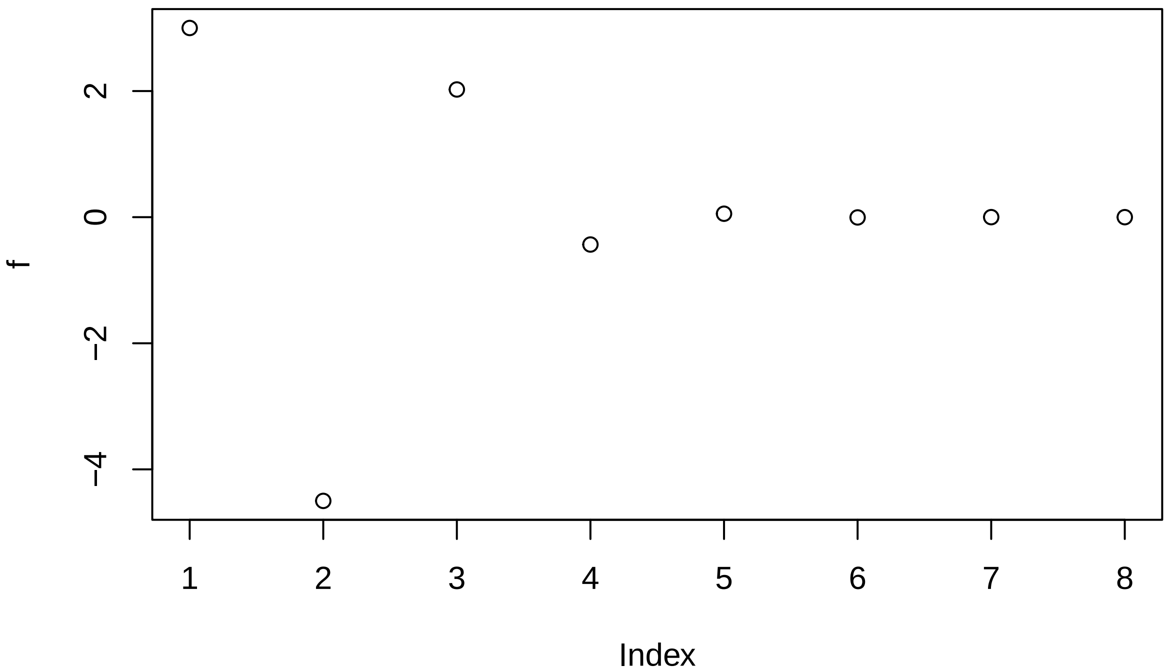
```
## The Newton-Raphson method was able to get within 2.893161e-10 of pi.
```

Exercise 10.10b

```
rm(list=ls())
estpi <- function(x = 3,n = 7){
  f <- rep(0,n)
  i <- 0
  while(i <= n) {
    f[i+1] <- (-1)^i*(x^(2*i+1))/factorial(2*i+1) # f[i+1] defined to avoid calling f[0]
    i <- i+1
  }
  print(x+sum(f))
```

```
  plot(f) # Plot is from 1 to 8, corresponding to the entries in f
}
estpi()
```

```
## [1] 3.14112
```



Exercise 10.10c

```
rm(list=ls())
estpi <- function(x = 3,n = 7){
  f <- rep(0,n)
  i <- 0
  while(i <= n) {
    f[i+1] <- (-1)^i*(x^(2*i+1))/factorial(2*i+1) # f[i+1] defined to avoid calling f[0]
    i <- i+1
  }
  print(c(f[n+1],10^-6)) # Prints last term of f (which is the smallest)
}
estpi()
```

```
## [1] -1.097284e-05  1.000000e-06
```

```
estpi(n=8) # Testing values of n, increasing by 1
```

```
## [1] 3.63072e-07 1.00000e-06
```

```
# An approximation correct up to six decimal places is obtained when n >= 8.
# A better way to calculate pi would be to copy and paste the number of digits of pi desired from an on
pi
```

```
## [1] 3.141593
```