# Problem Set 5

github: stephaniesookim

October 22, 2014

/raggedright

## Number 1

```
load("~/stat243-fall-2014/ps/ps5prob1.Rda")

log <- X%*%beta
p <- exp(log)/(1+exp(log))
likelihood<-prod(dbinom(y,n,p))
likelihood

## [1] 0

beta <- c(1,2,3,4)
log <- X%*%beta
p <- exp(log)/(1+exp(log))
likelihood<-prod(dbinom(y,n,p))
likelihood

## [1] 0
```

From the above, we can observe that the likelihood is 0 regardless of the value of beta.
This is because of the underflow. Since the number is less than 1e-308, it returns 0.

```
load("~/stat243-fall-2014/ps/ps5prob1.Rda")
log <- X%*%beta
p <- exp(log)/(1+exp(log))
logsum <- sum(dbinom(y,n,p,log=T))
logsum

## [1] -1862
```

To avoid this, we can take a log and then sum them up. The result is -1862.331.

## Number 2 (a)

```
options(digits=22)
1.000000000001

## [1] 1.000000000001000088901
```

It prints 1.000000000001000088901.
Thus we have 16 decimal places of accuracy here.

## Number 2 (b)

```
options(digits=22)
x <- c(1, rep(1e-16, 10000))
sum(x)

## [1] 1.000000000000999644811
```

It prints 1.000000000000999644811.
The sum() does not give the right answer up to the accuracy expected from part (a).
We have 12 decimal places of accuracy.

## Number 2 (c) : Python

```
import numpy as np
from decimal import Decimal
vec=np.array([1e-16]*(10001))
vec[0]=1
print (np.sum(vec))

## 1.0
```

It prints 1.0.
Summation in Python is even less accurate than in R.

## Number 2 (d)

```
options(digits=22)
x <- c(1, rep(1e-16, 10000))

# first for loop
fl1 <- 0
for (i in 1:length(x)) {
  fl1 <- fl1 + x[i]
}
fl1

## [1] 1

# second for loop: do the summation with 1 as the last value in the vector
fl2 <- 0
for (i in 2:length(x)) {
  fl2 <- fl2 + x[i]
}
fl2 <- fl2 + x[1]
fl2

## [1] 1.000000000001000088901
```

The first for loop returns a wrong value (1).
On the other hand, the second for loop returns a value that has 16 decimal places of accuracy.

## Number 2 (d) : Python

```
vec = np.array([1e-16]*(10001))
vec[0] = 1.0

#first for loop
fl1 = 0
for i in range(len(vec)):
  fl1 += vec[i]
print(Decimal(fl1))

# second for loop: do the summation with 1 as the last value value in the vector
fl2 = 0
for i in range(1, len(vec)):
  fl2 += vec[i]
fl2 = fl2 + vec[0]
print(Decimal(fl2))

## 1
## 1.000000000001000088900582341011613607406616210937 5
```

The first for loop returns a wrong value (1) like in R.
On the other hand, the second for loop returns the right value.

## Number 2 (e)

Apparently the results suggest that sum() does not simply sum the numbers from left to right.
The result should have been the same in two for loops if it worked that way.
The difference comes from the fact that we add 1 from the beginning or at last.
Thus we can notice that sum() is more accurate when we sum up numbers with same magnitudes.
It tries to be accurate up to 16 decimal places.

## Number 2 (f)