STAT 243: Problem Set 5

Jin Rou New [jrnew]

October 21, 2014

1 Problem 1

There was an overflow, since the result is smaller than the smallest number that can be represented, i.e. e^{-1022}. They should instead calculate the log likelihood by taking the sum of the logarithms of the probabilities.

```
options(digits = 10)
load("ps5prob1.Rda")
p <- c(1/(1 + exp(-X %*% beta)))
likelihood <- prod(dbinom(y, n, p)) # Value of 0 obtained
likelihood

## [1] 0

log_likelihood <- sum(log(dbinom(y, n, p)))
log_likelihood

## [1] -1862.33088
```

We can then calculate the likelihood by hand as either $e^{-1862.3309}$ or $10^{-1862.3309}/ln(10) = 10^{-1862.3309}/2.3026$.

2 Problem 2

(a) We can expect 16 decimal places of accuracy at most for our result.

```
options(digits = 20)
num <- 1.000000000001
num

## [1] 1.0000000000010000889
```

(b), (c) and (d) Yes, `sum()` give the right accuracy. In Python, it gives 0 decimal places whether I use the base `sum()` or from the numpy package. In both R and Python, using a `for` loop to do the summation with 1 as the first value in the vector do not give the right answer, but only a result with 0 decimal places of accuracy. With 1 as the last value in the vector, the right answer with 16 decimal places of accuracy is obtained.

```
# In R
options(digits = 20)
x <- c(1, rep(1e-16, 10000))
identical(num, sum(x))

## [1] FALSE
```

```
sum(x) # Right answer

## [1] 1.0000000000009996448

sum_x <- 0
for (i in 1:(length(x)))
  sum_x <- sum_x + x[i]
identical(num, sum_x)

## [1] FALSE

sum_x # 0 dp of accuracy

## [1] 1

y <- c(rep(1e-16, 10000), 1)
sum_y <- 0
for (i in 1:(length(y)))
  sum_y <- sum_y + y[i]
identical(num, sum_y)

## [1] TRUE

sum_y # 16 dp of accuracy, right answer

## [1] 1.0000000000010000889
```

```
# In Python
import numpy as np
import decimal
num = 1.000000000001
x = np.array([1e-16]*(10001))
x[0] = 1
print(np.equal(num, sum(x)))
print(decimal.Decimal(sum(x))) # 0 dp of accuracy

print(np.equal(num, np.sum(x)))
print(decimal.Decimal(np.sum(x))) # 0 dp of accuracy

sum_x = 0
for x_select in x:
  sum_x += x_select
print(np.equal(num, sum_x))
print(decimal.Decimal(sum_x)) # 0 dp of accuracy

y = np.array([1e-16]*(10001))
y[len(y)-1] = 1
sum_y = 0
for y_select in y:
  sum_y += y_select
print(np.equal(num, sum_y))
print(decimal.Decimal(sum_y)) # 16 dp of accuracy, right answer

## False
## 1
## False
## 1
## False
## 1
## True
## 1.0000000000010000889005823410116136074066162109375
```

(e) It suggests that R's `sum()` function does not simply sum from left to right.
(f) The `sum` function in R calls a sum function written in C. The C source code is available at: `https://github.com/wch/r-source/blob/trunk/src/main/summary.c`. In C, the `do_summary` function calls `rsum` to carry out summation of doubles. A long double (`LDOUBLE`) with higher precision than a double is used for the sum, hence the precision is kept.

```
static Rboolean rsum(double *x, R_xlen_t n, double *value, Rboolean narm)
{
    LDOUBLE s = 0.0;
    Rboolean updated = FALSE;

    for (R_xlen_t i = 0; i < n; i++) {
  if (!narm || !ISNAN(x[i])) {
    if(!updated) updated = TRUE;
    s += x[i];
}
    }
    if(s > DBL_MAX) *value = R_PosInf;
    else if (s < -DBL_MAX) *value = R_NegInf;
    else *value = (double) s;

    return updated;
}
```