

# ps5

Baoyue Liang

10/13/2018

## Problem 1

Suppose that  $\lambda_1, \dots, \lambda_n$  are the eigenvalues of  $A$ . Then the  $\lambda$ s are also the roots of the characteristic polynomial, i.e.

$$\det(A - \lambda I) = p(\lambda) = (1)^n(\lambda - \lambda_1)(\lambda - \lambda_2)\dots(\lambda - \lambda_n) = (\lambda_1 - \lambda)(\lambda_2 - \lambda)\dots(\lambda_n - \lambda)$$

The first equality follows from the factorization of a polynomial given its roots; the leading (highest degree) coefficient  $(1)^n$  can be obtained by expanding the determinant along the diagonal.

Now, by setting  $\lambda$  to zero (simply because it is a variable) we get on the left side  $\det(A)$ , and on the right side  $\lambda_1\lambda_2\dots\lambda_n$  that is, we indeed obtain the desired result.

$$\det(A) = \lambda_1\lambda_2\dots\lambda_n$$

So the determinant of the matrix is equal to the product of its eigenvalues.

## Problem 2

When  $z$  is large,  $\exp(z)$  can be large (with more than 16 digits) and  $1 + \exp(z)$  is rounded into the nearest possible value to  $\exp(z)$  and is thus not accurate.

We could rearrange the function as follow to make it numerically stable.

$$\expit(z) = \frac{1}{1 + \exp(-z)}$$

## Problem 3

```
set.seed(1)
z <- rnorm(10, 0, 1)
x <- z + 1e12
formatC(var(z), 20, format = 'f')

## [1] "0.60931443706111987346"
formatC(var(x), 20, format = 'f')

## [1] "0.60931216345893013386"
dg <- function(x, digits = 20) formatC(x, digits, format = 'f')
dg(z[2])

## [1] "0.18364332422208223816"
```

```
dg(x[2])
```

```
## [1] "10000000000000.18359375000000000000"
```

For  $x$ , since before the dot there is already 12-13 digits, only 3-4 digits after the dot is accurate. However,  $z$  has 16 digits of precision. When calculating the variance,  $z$  has 16 digits precision, while  $X$  only has 3-4 digits precision. This explains why the two variances agree to only 5 digits.

## Problem 4

(a)

If we divide into  $n$  individual column-wise computations, there are so many tasks and each one takes little time, the communication overhead of starting and stopping the tasks will reduce efficiency. Therefore it is better to divide the matrix into  $p$  blocks to improve efficiency.

(b)

Approach 2 is better at minimizing memory use, because at any time, for Approach 2, each core is calculating only the product of between a  $1 \times n$  vector and a  $n \times 1$  vector; for Approach 1, each core is calculating the product of a  $n \times n$  matrix and a  $n \times 1$  vector.

Approach 1 is better at minimizing communication, because there are  $p^2$  tasks iterate among  $p$  cores in Approach 2 and only  $p$  tasks in Approach 1. Therefore, Approach 1 has less communication time.