STAT243 PS5

Zhaochen Gao

10/15/2018

Problem 1

Linear algebra preparation. Before class on Wednesday October 17, please read the first section of Unit 9 (Numerical Linear Algebra) and answer the following question. (Turn in your answer as part of the problem set, but please do it by Wednesday.)

Ans:

$$det(A) = det(\Gamma\Lambda\Gamma^T) = det(\Gamma) * det(\Lambda) * det(\Gamma^T)$$

$$Since\ \Gamma\ is\ an\ orthogonal\ matrix,\ so\ det(\Gamma)det(\Gamma^T) = det(\Gamma\Gamma^T) = det(I) = 1.$$

$$Therefore,\ det(A) = det(\Lambda) = \lambda_1\lambda_2..\lambda_n = product\ of\ eigenvalues.$$

Problem 2

Numerical problems in logistic regression. Consider that in logistic regression the linear predictor,Xi*beta, (where Xi is the ith row of the predictor matrix, X) is transformed into a probability using the inverse logistic (expit) transformation.This mathematical expression for the expit function doesn't work numerically on a computer for large values of z. Explain why not and re-express the function such that if implemented in computer code, it is numerically stable.

Ans:(1) If we check *.Machine$double.xmax*, we can see the maximum number computer can represent is 1.797693e+308. If z becomes really large, then it may exceed the maximum magnitude, then it will cause overflow.

(2) We can re-express the function as

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

Problem 3

Consider the following estimate of the variance of a set of large numbers with small variance:

```
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"
```

*Ans:*We have 16 digits of accuracy, since x=z+1e12, 1e12*1e-16=1e-4, so it's only accurate to 4 decimal places. Therefore, when we calculate the variance, we cannot keep the original accuracy.

Problem 4

(This problem makes use of ideas from Unit 8, to be covered at the beginning of the week of Oct. 15, so you may need to wait until then to work on this.) Let's consider parallelization of a simple linear algebra computation. In your answer, you can assume m = n/p is an integer. Assume you have p cores with which to do the computation.

(a)Consider trying to parallelize matrix multiplication, in particular the computation XY where both X and Y are n*n. There are lots of ways we can break up the computations, but let's keep it simple and consider parallelizing over the columns of Y . Given the considerations we discussed in Unit 8, when you parallelize the matrix multiplication, why might it be better to break up Y into p blocks of m = n/p columns rather than into n individual column-wise computations? Note: I'm not expecting a detailed answer here – a sentence or two is fine.

Ans: If we consider parallelizing over the columns of Y, each task should take similar amount of time, it would be better if we preschedule the tasks to each core to reduce communication.

(b)Let's consider two ways of parallelizing the computation and count (1) the amount of memory used at any single moment in time, when all p cores are doing their calculations, including memory use in storing the result and (2) the communication cost – count the total number of numbers that need to be passed to the workers as well as the numbers passed from the workers back to the master when returning the result. Which approach is better for minimizing memory use and which for minimizing communication?

- Approach A: divide Y into p blocks of equal numbers of columns, where the first block has columns 1;. . . .;m where m = n/p and so forth. Pass X and the jth submatrix of Y as the jth task out of p total tasks.
- Approach B: divide X into p blocks of rows, where the first block has rows 1;. . . ;m and Y into p blocks of columns as above. Pass pairs of a submatrix of X and submatrix of Y to the workers, resulting in pˆ2 different tasks that we have to iterate through using our p cores.

Ans:

- Approach A: At single moment in time when all p cores are doing their calculation, each core will contain nˆ2+mn original numbers. The memory used in storing the result would be nm. So, the amount of memory used should be (nˆ2+2mn)*p. So it's 8(pnˆ2 +2nˆ2) in bytes. The total number of numbers that need to be passed to the workers should be pnˆ2 +nˆ2. The total number of numbers passed from the workers back to the master when returning the result should be nm per core. We have to repeat this process p times. So it's nˆ2 in total. In general, the communication cost is pnˆ2 +2nˆ2.
- Approach B: At single moment in time when all p cores are doing their calculations, each core will contain 2mn original numbers. The memory used in storing the result would be mˆ2. So, the amount of memory used should be (2mn+mˆ2)*p, which is 2nˆ2 +nˆ2 /p. So it's 8(2nˆ2 +nˆ2 /p) in bytes. The total number of numbers that need to be passed to the workers should be 2pnˆ2. For each core, it should be 2mn every time. The total number of numbers passed from the workers back to the master should be mˆ2 per core per time. And we have repeat this communication pˆ2 times. So the communication cost in total is 2pnˆ2 +nˆ2.
- Approach A is better for minimizing communication and Approach B is better for minimizing memory use.