ps5

Ramon Crespo

10/18/2018

Comments about the homework I worked on the homework by myself

Problem 1 For a symmetric matrix A, we have the following matrix decomposition (the eigendecomposition): A = VQVT, where V is an orthogonal matrix of (column) eigenvectors, and Q is a diagonal matrix of eigenvalues. Use the properties discussed in Section 1 to briefly show that |A| is the product of the eigenvalues.

Answer: A = VQVT. This by definition means that A is a squared psd matrix with positive eigenvalues. The eigendecomposition is defined as decomposing a matrix into two components, the eigenvalues (representd as the diagonal components of Q) and the orthogonal eigenvectors (represented by the columns of V). This means that if we obtain the eigenvectors and the corresponding eigenvalues of A we can build a matrix multiplication VQVT to represent A. Since A is a psd matrix, it is correct to state that |A| is a product of its eigenvalues and correspongin eigenvectors. If A is not psd, then this answer would not apply as it would have negative eigenvalues and |A| would not be the product of its eigenvalues.

Problem 2. 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.

Answer: From ps4 we noticed how the exponent of a function could problematic. The reason for this is that elevating the exponent value to a large number leads to large computation values that could lead to problematic computations. Recognizing that this problem is a binomial logistic regression problem we can just rewrite the function as: 1/1+exp(-z). When the value of z is large, the equation will approach 1 which is a stable solution.

Problem 3 Explain why these two estimates agree to only 5 digits when mathematically the variance of z and the variance of x are exactly the same.

Answer: The formatC variable is taking two different inputs. To understand the difference we must analyze what is happening in the var function. The number of values that the var function has is limited. Thus when we add values to the front of the variable, we are dumping the last values of the float number. Meaning that as we add more values to the front of the number, we forget the last values of the number (this is because memory per number is limited to 16 bites). Then the variance will be different as the values that are forgotten might have different impact in the variance.

```r
set.seed(1)
z <- rnorm(10, 0, 1)
x <- z + 1e1
formatC(var(z), 20, format = 'f')
```

```
## [1] "0.60931443706111987346"
```

```r
formatC(var(x), 20, format = 'f')
```

```
## [1] "0.60931443706112009551"
```

Problem 4 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.

Problem 4.A (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.

Answer With the information given and the posibilities of accessing p cores, we might want to break Y into p blocks so that we can parallelize the computation and use all the cores at our disposal. If we fixed the number of computations and n != p, then we will have computational capacity that is not used.

Problem 4.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 and so forth. Pass X and the jth submatrix of Y as the p jth task out of p total tasks.

Answer -> Total computational complexity = O(n3) -> Total Memory including original matrix and results -> i) original matrix size = 2n2 ii) at the split = p (n2 + 2nm) + original memory iii) final results = n2 + original memory -> Communication cost: i) to send data = p(n2 + nm) = n3/m + n2 ii) to return results = p(nm) = n2

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 p2 different tasks that we have to iterate through using our p cores. -> Total computational complexity = O(n3) -> Total Memory including original matrix and results -> i) original matrix size = 2n2 ii) at the split = 4n(n+m) iii) final results = n2 + original memory

-> Communication cost: i) to send data = p2(2mn) = n2/m2(2mn) = 2n3/m ii) to return results = p2(m2) = n2

Answer From my results, it seems like the first approach has an advantage in the amount of data that needs to be sent to the cores nontheless it requires less memory space. Depends of the size of the matrics I would choose between the computations. If memory is an issue then I would use approach two, even if it is a bit slower due to data transfer. If memory is not an issue, I would use approach number one as it probably is faster due to less data transfer.