Problem Set #5

Todd Faulkenberry

10/15/2018

Question 2

Expit doesn't work in this scenario because of how R evaluates numbers. Numbers larger than 2^1024 cause overflow, leading whatever number you put in to be converted to an infinite ('Inf.'). Expit in this scenario actually returns a "1" instead of an Inf, however. This is because R is dividing Inf by Inf + 1, which is effectively Inf/Inf, which is 1.

```
expit_bad <- function(n) {
  # Doesn't convert n before taking exp(), meaning the function
  # will return 1 instead of desired ratio.

  return(exp(n) / (1 + exp(n)))
}

exp(5) / (1 + exp(5))
```

```
## [1] 0.9933071
```

```
expit_bad(5)
```

```
## [1] 0.9933071
```

```
expit_bad(100)
```

```
## [1] 1
```

In order to get the value back that we want, we need to adjust the calculation by taking the natural log of the exponential function divided by the natural log of 10. This allows us to isolate our n and bring it to a scale where we will be able to get the answer we're looking for without overflowing R. The numbers below in my expit_good function are a little off because I had trouble converting the 1 to the proper number to add to the logarithm, but logically this new process should work if executed faithfully.

```
expit_good <- function(n) {
  # Converts number to log10 before running expit, which
  # will return the wanted ratio.

  n_log10 = n / log(10)
  return(n_log10 / (1 + n_log10))
}

exp(5) / (1 + exp(5))
```

```
## [1] 0.9933071
```

```
expit_good(5)
```

```
## [1] 0.684689
```

$$\begin{aligned} \det(A - \lambda I) = p(\lambda) &= (-1)^n(\lambda - \lambda_1)(\lambda - \lambda_2)\cdots(\lambda - \lambda_n) \\ &= (-1)(\lambda - \lambda_1)(-1)(\lambda - \lambda_2)\cdots(-1)(\lambda - \lambda_n) \\ &= (\lambda_1 - \lambda)(\lambda_2 - \lambda)\cdots(\lambda_n - \lambda) \end{aligned}$$

Figure 1: Factoring the characteristic polynomial to show that determinant equals the product of the eigenvalues. See explanation below.

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

Figure 2: Reduced form. See explanation below.

Question 1

I switch the order of question 1 and question 2 because of difficulties I was having with the display of my problem set in Knitr. I received significant help on this question from this math StackExchange post (https://math.stackexchange.com/questions/507641/show-that-the-determinant-of-a-is-equal-to-the-product-of-its-eigenvalues). I had trouble understanding this question - I'm teaching myself linear algebra on the fly and this is still above my head - but I spent time thinking about this question and looking for answers. As the Stack Exchange post shows, we know a determinant is a product of its eigenvalues because eigenvalues also serve as the root of the characteristic polynomial. When we factor this polynomial we get the results above. If we set the eigenvalue equal to zero, we then get a reduced form that shows the determinant is equal to the products of the individuals eigenvalues.

Question 3

The reason we only have a few digits of accuracy when taking the variance of x and comparing it to the variance of z is because we change the numbers in the x vector by adding 1e12 (aka 13 significant digits) to all numbers in the vector. Thus, while all numbers in n are centered around 0, all numbers in x are centered around 1e12. While this change is uniform in x and thus doesn't actually impact the variance, it interferes with R's accuracy in calculating it because R must consider those additional 13 digits. Thus, we would expect the variance here to only be accurate to three decimal places (because 13 + 3 = 16, and we only expect R's first 16 digits to be accurate.)

This question identifies a limitation with R. Even when calculating descriptive statistics or other calculations that may have way fewer than 16 digits, R will still produce inaccurate calculations if the underlying values are very large or very small. When dealing with very large or very small numbers in R, we can get better estimates of these statistics if we take the logarithm of these numbers and then do our calculations because these resulting statistics will be accurate for more digits.

Question 4

Part A

In this scenario, grouping by column means you may not be maximizing the use of your cores. This is because the computation necessary for each column may vary significantly, meaning some cores will finish much earlier

than others. This creates an inefficient process. Instead, if you split the task into p blocks with m = n / p columns, each core will be computing the same size block, meaning you will both maximize your use of cores and have all computations finish at roughly the same time.

Part B

Approach A will be better for both minimizing memory use and limiting communication. Because Approach B isn't as well-threaded as Approach A, it has many more calculations. Thus, while Approach A does have a lot of overhead that will drive up its total processing time, the order of growth of Approach B is more and thus will use a large amount of memory for bigger calculations. Additionally, the large number of calculations for Approach B means that it will have to send more calculations back and forth between workers because it has more information that has to go the core and then come back. This extra communication will also drive up processing time. Approach B is the less efficient process in both instances.