Stat243: Problem Set 5, Due Monday Oct. 28

October 18, 2019

Comments:

- This covers Unit 6 and the initial part of Unit 9.
- It's due at 2 pm on Monday October 28, **both submitted as a PDF to bCourses as well as committed to your Github repository**.
- Please note my comments in the syllabus about when to ask for help and about working together. In particular, **please give the names of any other students that you worked with on the problem set and indicate in comments any ideas or code you borrowed from another student.**
- The formatting requirements are the same as previous problem sets.

Problems

1. In class and in the Unit 6 notes, I mentioned that integers as large as 2^{53} can be stored exactly in the double precision floating point representation. Demonstrate how the integers 1, 2, 3, ...,$2^{53}-2$, $2^{53}-1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format where d is represented as 52 bits. I'm not expecting anything particularly formal - just write out for a few numbers and show the pattern. Then show that 2^{53} and $2^{53}+2$ can be represented exactly but $2^{53}+1$ cannot, so the spacing of numbers of this magnitude is 2. Finally show that for numbers starting with 2^{54} that the spacing between integers that can be represented exactly is 4. (Note you don't need to write out e in base 2; you can use base 10). Then confirm that what you've shown is consistent with the result of executing $2^{53}-1$, 2^{53}, and $2^{53}+1$ in R.

2. Here we'll consider the effects of adding together numbers of very different sizes. Let's consider adding the number 1 to 10000 copies of the number 1×10^{-16}:. Mathematically the answer is obviously $1 + 1 \times 10^{-12} = 1.000000000001$ by multiplication, but we want to use this as an example of the accuracy of summation with numbers of very different magnitudes, so consider the sum $1 + 1 \times 10^{-16} + \cdots + 1 \times 10^{-16}$.

   (a) How many digits of accuracy are the most we can expect of our result (i.e., assuming we don't carry out the calculations in a dumb way). In other words, if we store 1.000000000001 on a computer, how many digits of accuracy do we have? This is not a trick question.

   (b) In R, create the vector $x = c(1, 1 \times 10^{-16}, \ldots, 1 \times 10^{-16})$. Does the use of *sum()* give the right answer up to the accuracy expected from part (a)?

   (c) Do the same as in (b) in Python. For Python the *Decimal()* function from the decimal package is useful for printing additional digits. Also, this code will help you get started in creating the needed vector:

```
import numpy as np
vec = np.array([1e-16]*(10001))
```

(d) Use a *for* loop to do the summation, $((x_1 + x_2) + x_3) + ...$. Does this give the right answer? Now use a *for* loop to do the summation with the 1 as the last value in the vector instead of the first value. Right answer? If either of these don't give the right answer, how many decimal places of accuracy does the answer have? Do the same in Python.

Your results from (b) and (d) should suggest that R's *sum()* function is not simply summing the numbers from left to right.

3. Please read Section 1 of Unit 9 (Numerical Linear Algebra) and answer the following question. For a symmetric matrix A, we have the following matrix decomposition (the eigendecomposition): $A = \Gamma\Lambda\Gamma^\top$, where Γ is an orthogonal matrix of (column) eigenvectors, and Λ is a diagonal matrix of eigenvalues. Use the properties discussed in Section 1 to briefly show that $|A|$ is the product of the eigenvalues.

4. Extra credit: This problem explores the smallest positive number that R can represent and how R represents numbers just larger than the smallest positive number that can be represented. (Note: if you did this in Python you'd get the same results.)

   (a) By experimentation in R, find the base 10 representation of the smallest positive number that can be represented in R. Hint: it's rather smaller than 1×10^{-308}.

   (b) Explain how it can be that we can store a number smaller than 1×2^{-1022}, which is the value of the smallest positive number that we discussed in class. Start by looking at the bit-wise representation of 1×2^{-1022}. What happens if you then figure out the natural representation of 1×2^{-1023}? You should see that what you get is actually a well-known number that is not equal to 1×2^{-1023}. Given the actual bit-wise representation of 1×2^{-1023}, show the progression of numbers smaller than that that can be represented exactly and show the smallest number that can be represented in R written in both base 2 and base 10.

   Hint: you'll be working with numbers that are not normalized (i.e., denormalized; numbers that do not have 1 as the fixed number before the decimal point in the representation at the bottom of page 7 of Unit 6).