CS 111: Homework 5: Due by 11:59 pm Monday, February 17, 2020

Submit your paper as one PDF file, and tell GradeScope which page(s) each problem is on.

1. If Q is a 2-by-2 orthogonal matrix such that

$$Q \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \alpha \\ 0 \end{pmatrix},$$

what is one possible value of α? Explain (in English, in LaTeX) why your answer is correct. You don't have to say what Q is. Hint: There are two possible values of α; we'll accept either one.

2. Suppose you are given n-by-n matrices A, B, and C, with B and C nonsingular, and an n-vector b. Let I be the n-by-n identity matrix. Write a few lines of Python code that use `npla.solve()` to compute

$$x = B^{-1}(2A + I)(C^{-1} + A)b$$

without computing any new matrices or any matrix inverses.

3. Recall from lecture (or NCM section 2.9) the definition of the condition number $\kappa(A)$ of a matrix. Let

$$A = \begin{pmatrix} 1 & 0 \\ 0 & 1000 \end{pmatrix} \quad \text{and} \quad B = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix}.$$

3.1. What IEEE 64-bit floating-point number represents $\kappa(A)$?

3.2. What IEEE 64-bit floating-point number represents $\kappa(B)$?

4. These questions are all about IEEE standard 64-bit floating-point arithmetic, which is behind both numpy's `float64` type and C's `double` type. You can use `print_float64()` from the Feb 13 class to see the actual bits that represent any number. Recall that one hexadecimal digit stands for 4 bits.

4.1. *Machine epsilon*, or just ϵ for short, is defined as the largest floating-point number x such that $x + 1 = 1$ in floating-point arithmetic. The experiment we did in class showed that ϵ is approximately 10^{-16}, which is why we say that IEEE floating-point can be accurate to about 16 decimal digits.

What is the exact value of ϵ? (Your answer should be an exact arithmetic expression, not a decimal expansion.) What is the 16-digit hex representation of ϵ in the IEEE standard? How close is ϵ to 10^{-16}?

4.2. What is the 16-digit hex representation of $1/\epsilon$? What is its approximate value in base 10?

4.3. What is the largest non-infinite positive number that can be represented exactly in IEEE floating-point? Give your answer three ways: As an exact arithmetic expression, as the 16-hex-digit IEEE representation, and as an approximate value in base 10. (Hint: Consider the largest possible value of the 11-bit exponent field in the IEEE standard, but remember that value is reserved to represent "infinity".)

4.4. A common mistake some people make (but not you!) is to think that ϵ is the smallest positive floating-point number. It's not, by a long shot. Consider for example $x = \epsilon^{10}$. What is the approximate value of x in base 10? Is x an exact floating-point number? If so, give its IEEE 16-hex-digit representation; if not, give an IEEE 16-hex-digit representation of a floating-point number as close to x as you can.

4.5. How many different floating-point numbers x are there with $1 < x < 2$? How many with $4096 < x < 8192$? How many with $1/64 < x < 1/32$?

4.6. Standard 64-bit integer arithmetic (such as `int64_t` in C) uses twos complement to represent integers from roughly -2^{63} to 2^{63}. In this system, the number of negative integers is different from the number of positive integers. Why? Is the number of negative IEEE floating-point numbers different from the number of positive IEEE floating-point numbers? Why or why not?

5. For this problem, you may want to read the article "Tearing apart Google's TPU 3.0 AI coprocessor" (linked from the Feb 13 class notes), especially the section "TPU Chips" that starts on page 8.

The IEEE floating-point standard specifies a 16-bit version and a 32-bit version as well as the ubiquitous 64-bit version. The 16-bit version was never used much, because most scientific modeling requires higher precision. Recently, though, 16-bit floating-point has become popular in machine learning applications, because the weights in a neural network don't need to be determined very precisely.

IEEE standard 16-bit floating-point uses 1 bit for the sign, 5 bits for the exponent, and 10 bits for the mantissa. However, when Google developed its TPU ("Tensor Processing Unit") chip for machine learning, it used a different 16-bit format that it calls "bfloat", with 1 bit for the sign, 8 bits for the exponent, and 7 bits for the mantissa.

5.1. How many different floating-point numbers x are there with $1 < x < 2$ in IEEE 16-bit floating-point? In bfloat?

5.2. Assume that both IEEE 16-bit and bfloat treat exponents the same way IEEE 64-bit does, with the largest exponent reserved for infinity and all the exponents shifted to represent about the same number of negative exponents as positive exponents. (I don't actually know whether this is true for bfloat.) What is the largest non-infinite positive number that can be represented exactly in IEEE 16-bit floating-point? In bfloat? Give your answers both as exact arithmetic expressions and as approximate base-10 numbers.

5.3. Name one advantage of bfloat over IEEE-16, and one advantage of IEEE-16 over bfloat.

6. Consider each of the following python loops. For each loop, answer: How many iterations does it do before halting? What are the last two values of x it prints (both as decimals printed by python, and as IEEE standard 16-hex-digit representations)? Explain in one sentence what property of the floating-point system the loop's behavior demonstrates.

6.1.

```
x = 1.0
while 1.0 + x > 1.0:
    x = x / 2.0
    print(x)
```

6.2.

```
x = 1.0
while x + x > x:
    x = 2.0 * x
    print(x)
```

6.3.

```
x = 1.0
while x + x > x:
    x = x / 2.0
    print(x)
```