CS 111: Homework 7: Due by 11:59 pm Sunday, March 5, 2023

Submit your paper as one PDF file, and tell GradeScope which page(s) each problem is on. If you worked with a partner, you must each turn in your own homework paper, and report the name and perm number of your partner. No groups of more than two allowed.

1. (Compare NCM problem 1.38.) The moral of this problem is that, with floating-point arithmetic, sometimes two algorithms look equivalent but one is better than the other at getting an accurate answer. Suppose you have a number x that is an approximation to another number $x_{\text{exact}} \neq 0$. Define the *relative error* in x as $|(x_{\text{exact}} - x)/x_{\text{exact}}|$.

1.1. The classic quadratic formula says that the two roots of the quadratic equation

$$ax^2 + bx + c = 0$$

are

$$x_0, x_1 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}.$$

Use this formula in `numpy` (show your input and output) to compute both roots for

$$a = 1, \quad b = 10{,}000{,}000{,}000, \quad c = 1.$$

Also compute the roots two other ways: first with `numpy`'s `np.roots()`, and then by hand. To at least one significant digit, what is the relative error of the approximation computed using the quadratic formula to x_0? To x_1? What are the relative errors of the approximations computed using `np.roots()`?

1.2. You should have found in (1.1) that the classic formula is good for computing one root but not the other. Explain in a sentence why in this case one root isn't computed accurately. Hint: The answer involves IEEE floating-point arithmetic!

1.3. Use the classic formula to compute one root accurately, and then use the fact that

$$x_0 x_1 = \frac{c}{a}$$

to compute the other. What are the relative errors now?

2. Recall from lecture (or NCM section 2.9) the definition of the condition number $\kappa_2(A)$ of a matrix. Let

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

2.1. What IEEE 64-bit floating-point number represents $\kappa_2(A)$? Give your answer both as an ordinary base-10 number, and also as a 64-bit word (16 hexadecimal digits). You may obtain the latter from `cs111.print_float64()`.

2.2. What IEEE 64-bit floating-point number represents $\kappa_2(B)$? Give your answer both as an ordinary base-10 number, and also as a 64-bit word (16 hexadecimal digits).

3. 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 `cs111.print_float64()` to see the actual bits that represent any number. Recall that one hexadecimal digit stands for 4 bits.

3.1. *Machine epsilon*, or just ϵ for short, is defined as the smallest floating-point number x such that $1 + x > 1$ 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? (Give your answer as an exact arithmetic expression involving powers of 2, not a decimal expansion.) What is the 16-digit hex representation of ϵ in the IEEE standard? (Use `cs111.print_float64()`.) Approximately how close is ϵ to 10^{-16} in relative terms? (Answer this by computing $|\epsilon - 10^{-16}|/\epsilon$.)

3.2. What is the 16-digit hex representation of $1/\epsilon$? What is its approximate value in base 10? (Here you should give a decimal expansion.)

3.3. 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 involving powers of 2; 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".)

3.4. 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 (as a decimal expansion)? 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.

3.5. 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. For this problem, you may want to read the article "Tearing apart Google's TPU 3.0 AI coprocessor" (under Readings on the course GitHub site), especially the section "TPU Chips" that starts on page 9.

The 1985 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 before about 2010, 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.

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

4.2. What is machine epsilon for IEEE 16-bit floating-point? For bfloat? (Show your answer both as an exact mathematical expression and as an ordinary base-10 decimal approximation. You don't need to show the 16-bit hex representation unless you want to.)

4.2. 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.

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