Numerical Analysis Intro

How is your numerical computing intuition? Assume each of the following claims is referring to a computation being done on a computer with binary number representations (more on that later). Each may be true, false, or somewhere in-between. Discuss!

1. Every rational number can be stored exactly.

   Remark. The answer is no - at the minimum, given any computer, there is a maximum number that it can hold in memory. Adding one gives a number that cannot be stored. But in fact, there are lots of rational numbers whose binary representation are't finite and are not stored exactly. See one later in this handout!

2. The distance between any two adjacent numbers is the same and is called machine epsilon.

   Remark. This is definitely false. We will see that the distance between adjacent numbers is the same only for numbers whose representations have the same exponent - for more on this, and the definition of machine epsilon, see the floating point representations of real numbers handout.

3. All computed answers are wrong.

   Remark. This statement is, in my opinion, almost true. There are computations on integers that are completely accurate, as long as those integers aren't too big (all your computers will give you the right answer for $3 + 9$). But in general, all computations come with round-off errors, making the answers not quite "right."

   Part of the problem here is - what does "wrong" mean? We'll be discussing and defining different types of errors (absolute, relative, forward, backward) and properties of algorithms (stability) to be more accurate about this discussion.

4. Accurate solutions are more important than fast solutions.

   Remark. Sometimes. The main things that makes this true or false are, one, what are you doing with the solution, and two, how slow is a not-fast solution? For instance, if it's a medical image processing result, such that someone's life depends on the accuracy, and you can get the result in a day or two, this is probably a true statement. If, however, you're modeling a wildfire response and a really accurate solution would take a year on a supercomputer to find (completely realistic), that accurate solution would then do you no good. Hopefully the wildfire is already out!

   I bring this up because, often, this is the trade-off algorithm designers make: giving up accuracy for speed. It's a rare and beautiful thing when we can find an algorithm that is both faster and more accurate!

5. It is more difficult to find a solution of a problem when the solution is 0 than when the solution is 100.

   Remark. It's a bit of a generalization, but mostly true. Small numbers (and small errors, the kind that result from truncation for instance) are kind of the bane of a

computer scientist's existence. We'll particularly see this when we talk about systems of equations, but for now, just think about what happens when you divide by a small number. What's $100 \div 10^{-16}$, and how does the result compare to the original number, 100?

You'll have seen this idea before in the definitions of absolute and relative error. Let a be your approximation and x be the exact answer. Then $|x - a|$ is the absolute error, and $\frac{|x-a|}{|x|}$ is the relative error. If x is close to zero...

6. Let $\mathbf{b}$ be a real vector size $n \times 1$ and A be a real $n \times n$ matrix. As long as A is invertible, the system $A\mathbf{x} = \mathbf{b}$ is solvable quickly and accurately.

   Remark. This is a similar issue. A matrix like $\begin{bmatrix} 0 & 1 \\ 0 & 300 \end{bmatrix}$ is not invertible, and a computer will recognize that. But if it's instead $\begin{bmatrix} 10^{-16} & 1 \\ 0 & 300 \end{bmatrix}$, it's now invertible - but nearly singular. In fact, inverting matrices at all is not a good numerical technique; matrices are instead factored. But even factoring has problems with this situation, and good computing programs will give you warnings like "This matrix is close to singular."

7. Solving a problem 100 times (like $A\mathbf{x} = \mathbf{b}$ for the same A, different $\mathbf{b}$) takes 100 times as much work as solving the same problem once.

   Remark. Hopefully not! There's always some extra work involved when solving a problem lots of times, but another beneficial property of many algorithms is the ability to re-use parts of the computation. There are definitely techniques (include LU factorization, which you should have heard of in linear algebra) that reduce the work in this particular example.

8. Storing every $m \times m$ matrix A requires storing m^2 numbers.

   Remark. Not every matrix has to be stored this way, though most ("dense") are. In particular, think about the structure of the identity matrix (an example of a "sparse" matrix). How else could you store it?

9. A calculator computes $\sin(\frac{\pi}{124})$ by looking it up in a table in memory.

   Remark. It definitely does not. In early mathematics, these tables did exist for quick reference, much like the normal tables in statistics. But even then - where did the numbers in the table come from? The answer is a polynomial approximation to the trig function. We'll be talking about how to come up with those polynomials for sets of data points ("interpolation").

Binary Number System

Developing an understanding of numerical techniques must begin with an understanding of the numbers on which the techniques operate. It is not difficult to imagine that storing an irrational number exactly is impossible on a computer - at some point we must stop storing additional digits. This is a source of error called *truncation error*. But in fact, the

finite storage space per number affects far more than just irrational numbers, which we now explore.

Modern arithmetic models are based on binary representations of numbers. A base 10 number, such as $(321)_{10}$, means $3 \cdot 10^2 + 2 \cdot 10^1 + 1 \cdot 10^0$. In *binary* or *base 2*, the powers of 10's are replaced with powers of 2. So $(101)_2 = 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = (5)_{10}$. Binary digits are called *bits*, and are always 0 or 1.

Example 1. Convert each of the following binary numbers to their decimal representations.

- $(10110)_2 = 2^4 + 2^2 + 2 = 16 + 4 + 2 = 22$
- $(1100.01)_2 = 2^3 + 2^2 + \frac{1}{2^2} = 12.25$
- $(.111)_2 = \frac{1}{2} + \frac{1}{4} + \frac{1}{8} = \frac{7}{8}$

Remark. Just for fun, here's the infinitely repeating extension of that last example: $(0.\overline{1})_2 = \sum_{i=1}^{\infty} \frac{1}{2^i} = (\sum_{i=0}^{\infty} \frac{1}{2^i}) - \frac{1}{2^0} = \frac{1}{1-\frac{1}{2}} - \frac{1}{1} = 1$, just like $0.\bar{9}$! This is a geometric series, which you should (at least vaguely) recall from calculus. We don't do a lot with series in this class (though a quick review of Taylor polynomials wouldn't be a bad thing), but ideas like this are the sort of thing that could turn up in homework. I fully expect you to look them up!

Converting decimal to binary is a bit more complicated. For the integer part, repeatedly divide by 2, looking for the remainder, then list the remainders in the reverse order they're found. For the fractional part, repeatedly multiply by 2, and check if the result is more or less than 1, now listing in the same order they're found.

Example 2. Convert $(11.3125)_{10}$ to binary.

We start with the integer part:

$$\begin{aligned} 11 \div 2 &= 5R1 \to 1 \\ 5 \div 2 &= 2R1 \to 1 \\ 2 \div 2 &= 1R0 \to 0 \\ 1 \div 2 &= 1R0 \to 1 \end{aligned}$$

So $(11)_{10} = (1011)_2$. Now the fractional part.

$$\begin{aligned} .3125 \cdot 2 &= 0 + .625 \to 0 \\ .625 \cdot 2 &= 1 + .25 \to 1 \\ .25 \cdot 2 &= 0 + .5 \to 0 \\ .5 \cdot 2 &= 1 \to 1 \end{aligned}$$

Thus $(.3125)_{10} = (.0101)_2$, and combining gives $(11.3125)_{10} = (1011.0101)_2$.

Example 3. Convert each of the following to binary representation.

- $0.5625 = (0.1001)_2$

- $3.125 = (11.001)_2$
- $\frac{7}{10} = (0.1\overline{0110})_2$

  Remark. On this last one, when you're doing your multiplying, you want to watch for a pattern. If you ever multiply and get a result that you already had (in this one, 0.4), stop! The pattern repeats from there.