CMPS 3500 Programming Languages
Activity 4, Part 4

Title: Analyzing the accuracy of Floating Point Numbers
Name: Mihail Chitorog
Date: 10/27/2024

In this report I am analyzing the difference in determinant calculation between our Ada program and the online calculator for the this 7x7 matrix:

```
-12  38  86  92 -29 -64  60
 65 -47  25 -43 -63  56 -48
-44 -15 -93  55  21 -80  -3
 37 -20  12  47  38 -57 -43
-72  -9   32  -4 -97 -46  27
-72 -46 -50  88 -94 -91 -94
 91   -1   71 -90  86   7  64
```

Our Ada program (printed as Float): 5604422189060.0
Our Ada program (printed as Long_Float): 5604422274528.00
Online calculator determinant is: 5604422274528.021

1. The problem is the print statement conversion. In our Ada code, the determinant is correctly calculated using Long_Float (64-bit precision). However, when printing, we convert it to a regular Float (32-bit) using:
   Put(Item=>Float(detm3),Aft => 0, Exp => 0).
   This conversion causes us to lose half of our significant figures.

2. If we look at the original matdet.adb file, we see that it used:
   Put(Long_Float'Image(detm3))
   This method preserved the full 64-bit precision when printing. Using this method would give us the correct answer: 5604422274528.00

3. The calculation itself was always correct. The precision "problem" was just a display issue. We were losing precision at the final step when converting from Long_Float to Float for output.

4. The difference between our Long_Float result (5604422274528.00) and the online calculator (5604422274528.021) is minimal and might be due to the online calculator using different rounding modes, extended precision calculation or a computer algebra system.

5. There are limitations of floating-point math. Even though 64-bit floating-point math we're using is pretty precise, it can't perfectly represent every number, especially in big calculations with lots of steps. When we calculate a matrix's

determinant, we have to multiply and add numbers many times. Each operation has tiny rounding errors, and these errors add up, which can affect the final answer.

6. It could be that the online calculators are using 128-bit floating point quadruple precision which are much more precise than a 64-bits precision.

7. Another reason is that they might be using a computer algebra system like SymPy where they may avoid floating-point arithmetic altogether, working instead with symbolic or rational numbers to eliminate rounding errors. Apparently arbitrary-precision libraries allow to set an exact precision level as needed. This is very useful for calculations that need a high degree of accuracy, like determinants of large matrices or complex algorithms that tend to have cumulative rounding errors.