**ENGR 102**

**Lab #4A [100 Points]**

**Activity #1: To do in lab – as a team [50 Points]**

When performing numerical computations, one of the challenges you can run into is floating-point roundoff error. This occurs when the computer needs to represent a number that would require an infinite number of decimal digits, but rounds them off after some point. That small roundoff error can cause some significant issues.

This activity is meant to help you understand floating-point roundoff error a bit more, and learn about one way of dealing with it.

*Part 1: Identifying floating-point problems.*

We first want to illustrate some examples where floating-point roundoff can cause trouble, and see other cases where it turns out not to.

First, as a team, type in and run the following program:

a = 1/7

print(a)

b = a\*7

print(b)

Notice that the value of a is rounded off. The value of b, if we have no roundoff, should be 1. Is it?

Now try the following program:

a = 1/7

print(a)

b = 7\*a

print(b)

c = 2\*a

d = 5\*a

e = c+d

print(e)

In this case, the value of e, if we have no roundoff, should be 1. Is it?

Finally, try the following program:

from math import \*

x = sqrt(1/3)

print(x)

y = x\*x\*3

print(y)

z = x\*3\*x

print(z)

Again, the value of y and z, if we have no roundoff error, should be 1 in both cases. Is it?

Was that surprising? You should have seen from those examples that sometimes we will encounter issues due to roundoff error and sometimes we won’t. We can’t always predict when roundoff error will be obvious.

Your team should write a program to demonstrate generating roundoff error in computations. It should have the following structure:

* Generate at least 3 different examples of roundoff error from numerical computations. You can use small variations of the ones shown above. Try to come up with different types of ways that roundoff can come up in a computation.
* For each of the examples, you should output a line saying: “Is XXX the same as YYY? AAA” where XXX is one way of computing a value and YYY is another, and AAA is either True or False.
  + Note: You can output a Boolean value and it will output either “True” or “False”. For example, a line like print(1==2) would output False.
* For at least one case, try to see how large of a difference you can get between the two values (i.e. how big you can get the numerical error to become). This usually requires performing multiple operations: one where the numerical error is introduced, then subsequent operations that cause that error to grow.

*Part 2: Tolerances for Comparisons.*

In part 1, you should have seen that two different ways of computing values that “should” be identical might actually come out differently. In some cases, this is not a problem. For example, we usually don’t care if speed is incorrect in the 10th decimal place, since we usually can’t measure that precisely, anyway. But, floating point error can become a big problem if comparisons are made with floating-point values. Think about the cases from part 1. The Boolean value you output on each line was likely coming out “False” even if the values should have been the same.

A common way for dealing with floating-point error is to use **tolerances**. Tolerances let you compare two values that are close, but not identical to each other. Rather than checking whether a==b, we instead compute a-b, and see if that is within some small distance away from 0 (above or below). To do this, you take the absolute value of (a-b), and check whether it’s less than some small value, called the tolerance. If it is, the two things are considered to be equal. The tolerance is commonly abbreviated TOL or EPS (short for epsilon).

For example, using code from Part 1 and the concept of tolerances, we can compare values of the variable b and e:

# define tolerance

TOL = 1e-10

a = 1/7

print(a)

b = 7\*a

print(b)

c = 2\*a

d = 5\*a

e = c+d

print(e)

# check if b and e are equal within specified tolerance

if abs(b-e) < TOL:

print(‘b and e are equal within tolerance of’, TOL)

else:

print(‘b and e are NOT equal within tolerance of’,TOL)

Add the tolerance check on b and e to your code from Part 1 above. Then add a similar tolerance check for y and z in the example using x = sqrt(1/3).

*Note on Tolerance sizes (something to think about)*

Keep in mind that the larger the tolerance, the more numerical error you can tolerate. A large enough tolerance can catch almost any numerical errors. However, this also increases the chances of mistakenly identifying two things that should not be the same as being identical.

*Summary:*

Turn in a single program for this activity: it should incorporate all the code from parts 1-2.

As you write programs, please think about your comparisons and decide whether you need to use tolerances or not. If you are making a comparison where you are checking for exact equality and you might have some floating-point error, you probably will want to use tolerances, while if you are just checking which of two things is larger, a tolerance comparison is likely unnecessary. Tolerances are particularly helpful when checking things like whether a denominator is (nearly) 0, and thus a division is likely to create error.

Check out this link to the Python documentation: <https://docs.python.org/3/tutorial/floatingpoint.html>. This info helps explain these issues. As the document states, “. . .this is not a bug in Python, and it is not a bug in your code either.” Instead, the problems stem from the way a floating point number is represented by the hardware. Awareness of these issues may save you a lot debugging effort in the future.

Check out this link: Binary Tutorial - 5. Binary Fractions and Floating Point

<https://ryanstutorials.net/binary-tutorial/binary-floating-point.php>

Side note: Tolerances have problems of their own besides the difficulty of setting a “correct” tolerance value. Notably, with tolerances, you might find that a==b, and b==c, but a != c. Dealing with issues of roundoff error is a long-standing and ongoing area of research.

**Activity #2: To do in lab – as a team [50 points]**

The purpose of this program is to give you practice with performing Boolean logic.

Part a: Evaluating Booleans

Write a program that reads in true/false values for the variables a, b, and c and evaluates the following Boolean expressions. Your program should thus read in 3 True/False values, and should output 8 values.

1) a and b and c

2) a or b or c

3) (not (a and not b) or (not c and b)) and (not b) or (not a and b and not c) or (a and not b)

4) (not ((b or not c) and (not a or not c))) or (not (c or not (b and c))) or (a and not c) and (not a or (a and b and c) or (a and ((b and not c) or (not b))))