MA453/553 – Classwork 0.
Finite Precision Arithmetic Woes
Date: 09/04/2024

This is a sobering example (by W. Kahan) to serve as a warning of how inaccurate finite precision arithmetic can be. It is meant to scare you away from trusting blindly anything produced by a computer, and make you aware that strange things can happen easily, that what you get DOES depend on how you write the expressions (and even the constants!), and that machine arithmetic is NOT infinite precision arithmetic! Below is a Python script that performs some very simple calculations:

```
#!/usr/bin/env python3
"""
================================================================================
MA453 - Classwork 0: your name - date
Purpose: To implement W. Kahan's example of finite precision arithmetic
failings from C. Van Loan, Intro. to Computational Science, 1995; p.xxiv
--------------------------------------------------------------------------------
"""
h = 1/2
x = 2/3 - h
y = 3/5 - h
u = (x + x + x) - h
v = (y + y + y + y + y) - h
q = u/v
print('x :', x)
print('y :', y)
print('u :', u)
print('v :', v)
print('q :', q)
print()
```

1. Perform the above calculations on a piece of paper by hand exactly (as fractions) and record the values for x, y, u, v and q.

2. Now type it on the machine. Open an xterm and create a new directory ∼/MA453/Classwork/cw0, cd into it and type the code into a file "Kahan.py". Do you remember how?

```
$ mkdir MA453                    (skip this step if you had already created the MA453 directory)
$ cd MA453
$ mkdir Classwork                (skip this step if you had already created the Classwork directory)
$ cd Classwork
$ mkdir cw0                      (skip this step if you had already created the cw0 directory)
$ cd cw0
$ vi Kahan.py
```

and type it in. Save and exit from vi with ZZ, or better yet, save (:w) without exiting (since you may need to edit it again), and go to another Terminal for running it. Make sure you are in the ∼/MA453/Classwork/cw0 directory again!

3. Run the Python code "Kahan.py':

```
$ python3 kahan.py
```

You can also run the Python code "Kahan.py" like a shell script (see the first line of the code!). Set the execution permission to the file and run it.

```
$ chmod u+x Kahan.py
$ ./Kahan.py
```

4. Compare the computed values with the exact values you found by hand. Which ones agree and which ones do not? Any guess as to what may have caused the discrepancy?

5. Now let's compile and run the C and Fortran codes "Kahan.c", "Kahan.f" posted at Canvas.
C:
`$ gcc -o c.x Kahan.c` (the executable will be named c.x instead of the default a.out)
`$ ./c.x`
Fortran:
`$ gfortran -o f.x Kahan.f` (produces the executable f.x)
`$ ./f.x`

6. Did you get different values now? Which code (C, Fortran or Python) is more accurate? Any guess as to what's happening?

7. Python code by default computes in double precision arithmetic. In C and Fortran, we should declare the variable types (float/double, real/double precision etc.) in advance. The Fortran code `Kahan.f` is computing in single precision. Now let's make it work in double precision.
`$ cp Kahan.f Kahan_dp.f` (what does this do?)
Open the file "Kahan_dp.f" for editing
`$ vi Kahan_dp.f`
and make the following modifications:
replace `real` → `double precision`
replace `.0` → `.d0` in all constants (double precision constants)
`:wq` (save it and quit)
Compile and run:
`$ gfortran -o dpf.x Kahan_dp.f`
`$ ./dpf.x`

8. Did you get different values now? Is the double precision calculations in Fortran more correct than the single precision?

9. Evaluate the expression $z = |3(4/3 - 1) - 1|$ in Python. You may add the following two lines at the end of the Python code "Kahan.py" and run it.

```
z = abs(3*(4/3-1)-1)
print('z=',z)
```

Or, open an interactive Python shell from an xterm and type the expression.
`$ python`
`>>> z=abs(3*(4/3-1)-1)`
`>>> z`
What did you get? Was it expected?

10. Submit your code "Kahan.py" thru your course Canvas (with answers to 4, 6, 8 and 9 in the Comment Box).