

**Computational Physics / PHYS-GA 2000 / Problem Set #2**  
**Due September 17, 2024**

You *must* label all axes of any plots, including giving the *units*!!

1. Figure out how NumPy's 32-bit floating point representation (which is the IEEE standard) represents the number 100.98763 in bits. This 32-bit representation in fact corresponds to a slightly different number. Calculate what that 32-bit number is and how much the actual number differs from its 32-bit floating point representation
2. Demonstrate using examples that 32-bit is less precise and smaller dynamic range than 64-bit. First, find approximately the smallest value that you can add to 1, and get an answer that is different than 1, in both 32-bit (`np.float32`) and 64-bit (`np.float64`) precision. Second, approximately find the minimum and maximum (positive) numbers that these two types can represent without an underflow or overflow.
3. Exercise 2.9 of Newman. Note that the physical constants drop out so you do not need to worry about them (whenever possible you should seek to remove physical constants from the innards of your computations!). Write two versions of the code, one which uses a `for` loop and one which does not. Use the `timeit` module to determine which is faster.
4. Exercise 3.7 of Newman. Note that you can use a NumPy array to perform the iterations for each value of  $c$  all at once, which will be much faster than using a `for` loop over  $c$ .
5. Exercise 4.2 of Newman. For Part (c), put your solver into a *module* that can be imported. Name the module `quadratic`, and name your function within it `quadratic`. Create a file called `test_quadratic.py` with the contents shown below, and ensure that the call `pytest test_quadratic.py` returns no errors. This is a simple example of a unit test.

```
import pytest

import numpy as np
import quadratic

def test_quadratic():
    # Check the case from the problem
    x1, x2 = quadratic.quadratic(a=0.001, b=1000., c=0.001)
    assert (np.abs(x1 - (- 1.e-6)) < 1.e-10)
    assert (np.abs(x2 - (- 0.9999999999999999e+6)) < 1.e-10)

    # Check a related case to the problem
    x1, x2 = quadratic.quadratic(a=0.001, b=-1000., c=0.001)
```

- 2 -

```
assert (np.abs(x1 - (0.999999999999e+6)) < 1.e-10)
assert (np.abs(x2 - (1.e-6)) < 1.e-10)

# Check a simpler case (note it requires the + solution first)
x1, x2 = quadratic.quadratic(a=1., b=8., c=12.)
assert (np.abs(x1 - (- 2.)) < 1.e-10)
assert (np.abs(x2 - (- 6)) < 1.e-10)
```