Computational Physics ps-2 Report

Tongzhou Wang, GitHub account: TZW56203, repository: phys-ga2000

September 17, 2024

1 Problem 1

```
NumPy's 32-bit floating point representation of 100.98763 is
100.98763 decimal ->
        bitlist = [0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1,
           1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1]
        sign = 0
        exponent = [1, 0, 0, 0, 0, 1, 0, 1]
        mantissa = [1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1,
           0, 1, 0, 1, 1]

This 32-bit representation in fact corresponds to
100.98763275146484
We can check that: 13236651/131072 =  100.98763275146484

The difference is
2.7514648479609605e-06
```

2 Problem 2

```
In np.float32 precision, the smallest number you can add to 1 and get an
   answer that is different than 1 is approximately
2^-23 or anything larger than 2^-24

For example
1 + 2^-23 = 1.0000001
1 + 1.01 * 2^-24 = 1.0000001
1 + 2^-24 = 1.0

In np.float64 precision, the smallest number you can add to 1 and get an
   answer that is different than 1 is approximately
2^-52 or anything larger than 2^-53

For example
1 + 2^-52 = 1.0000000000000002
```

```
1 + 1.01 * 2^-53 = 1.0000000000000002
1 + 2^-53 = 1.0

In np.float32, the minimum (positive) number is approximately
1e-45
In np.float32, the maximum (positive) number is approximately
3.4028235e+38

In np.float64, the minimum (positive) number is approximately
5e-324
In np.float64, the maximum (positive) number is approximately
1.7976931348623157e+308
```

The following codes were used to compute the minimum and maximum (positive) numbers in `np.float32` and `np.float64`.

```
min32 = np.float32((0 + np.exp2(-23)) * np.exp2(-126))
max32 = np.float32((1 + (1 - np.exp2(-23))) * np.exp2(127))
min64 = np.float64((0 + np.exp2(-52)) * np.exp2(-1022))
max64 = np.float64((1 + (1 - np.exp2(-52))) * np.exp2(1023))
```

3 Problem 3

```
The Madelung constant computed using for loop is
-1.7365049
The Madelung constant computed without using for loop is
-1.736073


The time using for loop is
5.275421608006582 second
The time without using for loop is
0.017490418045781553 second


The function without using for loop is faster
```

According to wikipedia, the Madelung constant for a sodium ion in sodium chloride is -1.747565. Here, I only considered L = 50 atoms in each of the three directions. For larger L, the result should approach the true value.

4 Problem 4

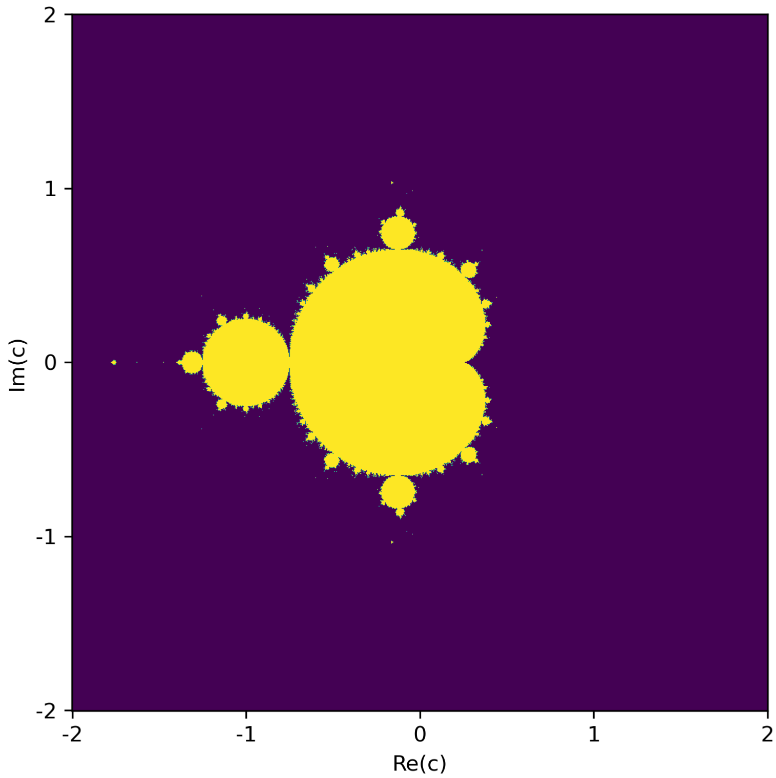

Figure 1: Mandelbrot set.

5 Problem 5

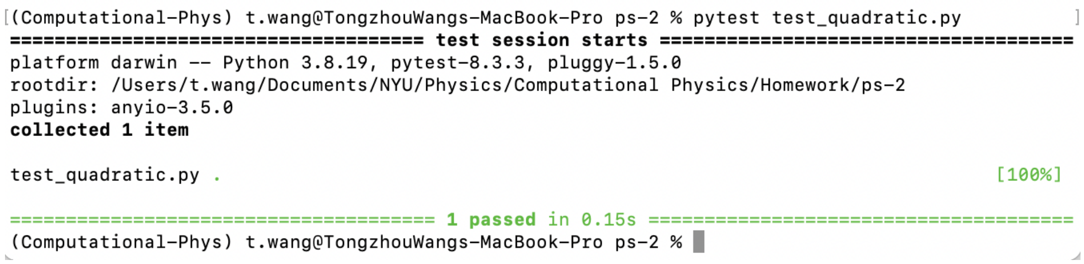

Figure 2: Unit test result.

The function I wrote is shown below. I think the key is to avoid subtractive cancellation errors.

```python
def quadratic(a, b, c):
    v1 = np.abs(np.float64((- b + np.sqrt(np.square(b) - 4 * a * c))))
    v2 = np.abs(np.float64((- b - np.sqrt(np.square(b) - 4 * a * c))))

    if v1 > (np.abs(b) / 10):
        x1 = np.float64((- b + np.sqrt(np.square(b) - 4 * a * c)) / (2 * a))
    else:
        x1 = np.float64((2 * c) / (- b - np.sqrt(np.square(b) - 4 * a * c)))

    if v2 > (np.abs(b) / 10):
        x2 = np.float64((- b - np.sqrt(np.square(b) - 4 * a * c)) / (2 * a))
    else:
        x2 = np.float64((2 * c) / (- b + np.sqrt(np.square(b) - 4 * a * c)))

    return x1, x2
```