ORIE 5270 HW4

Ko-Cheng Wang (kw582)

Repo url: https://github.coecis.cornell.edu/kw582/ORIE5270/tree/master/HW4

Files Description:

1. complexity_plot.pdf: The complexity plot for Problem 4
2. detectSubnormalNumber.py: The Python solution to detect subnormal number required by Problem 2
3. hw4.py: A modified NameHeap class from HW3
4. MatrixVectorProductComplexity.py: The Python code for Problem 4
5. NameHeapTests.py: The Python code to test the correctness of hw4.py
6. qesolution.py: The Python code to find roots with quadratic formula, Problem 3-(1)
7. qesolutionRevise.py: The Python code to find roots with quadratic formula and conjugate method, Problem 3-(3)
8. solution: Contain the detailed solution to HW4
9. TestDetectSubnormalNumber.py: The python code to test the correctness of detectSubnormalNumber.py, Problem 2
10. TestQesolution.py: The python code to test the correctness of qesolution.py and qesolutionRevise.py, Problem 3

Problem 1)

To run the file NameHeapTests.py, run the following command in the HW4 directory:

```
python3 -m unittest NameHeapTests.py
```

The purpose of each testing case is explained as comment in the NameHeapTests.py

Problem 2)

** Has to install "bitstring" by "pip install bitstring"

** To run TestDetectSubnormalNumber.py, run the command:

```
python3 -m unittest TestDetectSubnormalNumber.py
```

(1) Subnormal number is a way to represent numbers that are smaller in magnitude that 2^-1022. In a subnormal problem, the leading digit of the significand is assumed to be a zero and the exponent is its smallest value (-1022 in this case).

An example: $2^{-1022-49}$

(2) For a subnormal number, the exponent must be its smallest value (-1022) and the mantissa is nonzero. These two criteria are tested by the "if statement" at line 14 of detectSubnormalNumber.py. If both conditions are met, then the algorithm should output True. Output False otherwise.

A test file TestDetectSubnormalNumber.py is created to test the correctness of detectSubnormalNumber.py. The test cases include the two smallest subnormal numbers ($2^{-1022-52}$, $2^{-1022-51}$), the one mentioned in the class ($2^{-1022-50}$)) and the one that I provided as the answer to Problem 2-(1) ($2^{-1022-49}$). These will be tested to output "True".

Some other crucial cases such as 2**(-1022) and 0, float('inf') are also tested as well, expected to output "False".

Problem 3)

** To run TestQesolution.py, run the command:

python3 -m unittest TestQesolution.py

(1) qesolution.py contains the code to calculate two roots with the quadratic formula for different input b and c

Testqesolution.py contains two test cases (normal cases) to test the accuracy of the root calculated from qesolution.py. To try different b and c, simply change the parameters to other values

Test cases:

i. test_qeSol uses b =2**10+0.4543653446748 and c = 3**5, which satisfies the accuracy test

ii. test_qeSol1 uses b =2**20 and c = 3**15 , which satisfies the accuracy test

iii. test_qeSol3 uses b = 245435345 and c = 1234327, which satisfies the accuracy test

(2) Several test cases in Testqesolution.py are used to test some extreme values so that they fail the function qesolution.py (with higher relative error) For some extreme values, the relative error increases, since floating point only contains a finite amount of precision so there might be different magnitude of small perturbations in the input of a numerical function. When values are especially large, the perturbations are more significant.

Test cases:

i. test_qeSol2 uses b = 2**25 + 7 ** 4 and c = 1.9848237598435923649562934, results in the product not accurate enough (relative error > 10^-12)

(3) qesolutionRevise.py calculates two sets of roots by quadratic formula and conjugate trick. Because we have to selectively chooses the more numerically stable formula depending on the right situation, I compare the relative error associated with two sets of roots and return the set with smaller relative error. In this way, qesolutionRevise.py can always pass the previous tests in TestQesolution.py.

Test cases:

i. test_qeSol3_revise uses b = 2**25 + 7 ** 4 and c = 1.9848237598435923649562934. It satisfies the accuracy test despite it does not pass with qesolution.py previously.

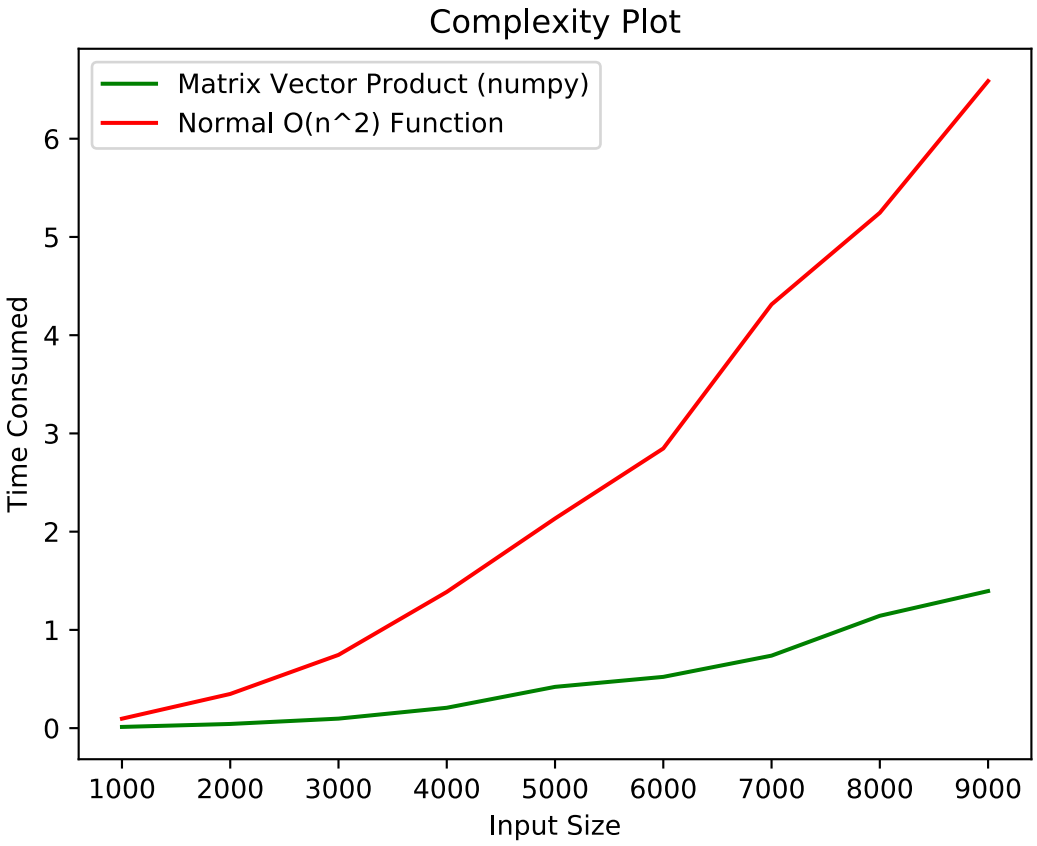
ii. test_qeSol4_revise uses b =2**20 and c = 3**15, which still satisfies the accuracy test

Problem 4)

** To run MatrixVectorProductComplexity.py, run the command:

python MatrixVectorProductComplexity.py

The rationale is that I should compare the time consumed by the numpy matrix-vector product function and by some other function that runs in O(n^2) with the same input size. The following is the plot:



For the O(n^2) function I use, please refer to MatrixVectorProductComplexity.py. The x-axis (Input Size) denotes the size of "n" ($\mathbb{R}^{n\times n}$) used in matrix vector multiplication and the other function I use. y-axis (Time Consumed) is the time used to run the given function.

From the plot, we can see that the time required by numpy matrix-vector product function is strictly less than the simple O(n^2) function I use, and the difference in the time consumed increases with the increase of input size. Hence, we can argue that the numpy matrix-vector product function takes O(n^2) time.