FYS3150-Project 1
Vector and matrix operations

Chenghao Liu

September 9, 2018

1. Introduction

The aim of this project is to get familiar with various vector and matrix operations, from dynamic memory allocation to the usage of programs in the library package of the course. The problem of this project is to solve the problem of matrix simplification, that is, to simplify the symmetric matrix into a diagonal matrix and determine the operation speed of different matrix order programs.

2. Method

I am using Python for programming here. I used python's numpy to generate the matrix. It provides a method for generating a diagonal matrix. I obtained the symmetric matrix by adding the three matrices, and I can also change the addition coefficient to change a,b,c.

Taking a 5th order matrix, a=1, b=-2, c=1 as an example

The simplified matrix is as shown:

```
choose n for matrix:5
time: 0.0
[[-2.          1.          0.          0.          0.        ]
 [ 0.         -1.5         1.          0.          0.        ]
 [ 0.          0.         -1.33333333  1.          0.        ]
 [ 0.          0.          0.         -1.25        1.        ]
 [ 0.          0.          0.          0.         -1.2       ]]
>>>
```

In the program I used a for loop, multiplying a line by a certain coefficient to eliminate the next line. After the entire for loop, all the lower part of the element will be eliminated, leaving only the diagonal element and the upper part of the element. To generate the desired matrix. Use the time function to get the time before and after the for loop, so that you can get the calculation speed, which is equivalent to the number of floating-point operations. Because I can't find a way to get the number of floating-point operations in Python, I use time instead.

The code for the entire program is as shown:

```
import numpy as np
import time

n = int(input('choose n for matrix:'))
a1 = np.eye(n,k=0)
a2 = np.eye(n,k=1)
a3 = np.eye(n,k=-1)

a = (-2)*a1+1*a2+1*a3
matrixA = a

m = len(matrixA)
i = 0

t1 = time.time()
while i < n-1:
    x = matrixA[i][i]
    y = matrixA[i+1][i]
    mtp= y/x
    matrixA[i+1] = matrixA[i+1] - mtp*matrixA[i]
    i = i+1
    pass

t2 = time.time()
t = t2-t1
print('time:',t)
```

3. Implementaion

Substituting n=10, 100, 1000, 10000, 100000, I found that Python will have MemoryError when n is greater than 10,000. There are certain restrictions.

Firstly, input the n for matrix, we can choose n under 100000, and then we can get the data of the simplified matrix that we want, and also we can get the calculate time for simplify the matrix.

```
Python 3.6.3 (v3.6.3:2c5fed8, Oct  3 2017, 18:11:49) [MSC v.1900 64 bit (AMD64)]
 on win32
Type "copyright", "credits" or "license()" for more information.
>>>
========== RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:100
time: 0.0009996891021728516
>>>
========== RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:1000
time: 0.006010532379150391
>>>
========== RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:10000
time: 0.32295680046081543
>>>
========== RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:10
time: 0.0
>>>
========== RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:100000
Traceback (most recent call last):
  File "D:/课件/物理学/计算物理/Project1/Project 1test.py", line 5, in <module>
    a1 = np.eye(n,k=0)
  File "C:\Users\asus\AppData\Local\Programs\Python\Python36\lib\site-packages\n
umpy\lib\twodim_base.py", line 186, in eye
    m = zeros((N, M), dtype=dtype, order=order)
MemoryError
>>>
```

Record the calculation time corresponding to different n and draw through python's matplotlib library.. First, take the logarithm of lg for both n and time, and then perform a linear fit, and find that the slope of the corresponding line is close to 1.4. The result is y =1.431 x -6.225

The code and image of the drawing are as shown:

I use python's matplotlib to build the figure, and use numpy to linear fit the line that lg(n) and lg(time) combined together. In the end the result shows a good linear relationship between lg(n) and lg(time), but the difference is that the teacher said the slope should be 2 in the figure, but however I get 1.431. This might due to the difference between time function and floating count times, because I can't find the function of counting floating count time, so I replace it with the cpu time. And I can't get the data above n =10000, this is the limitation of Python, if the n>10000, memory error will happen in the program.

```
import matplotlib.pyplot as plt
import math
import numpy as np

x = [10, 100, 1000, 10000]
y = [0.00001,  0.0009996891021728516, 0.006010532379150391, 0.32295680046081543]

for i in range(0, 4):
    x[i] = math.log10(x[i])
    y[i] = math.log10(y[i])
    pass
z1 = np.polyfit(x, y, 1)
p1 = np.poly1d(z1)
print(z1)
print(p1)
plt.figure('time-n fig')
ax = plt.gca()
ax.set_xlabel('lg(n)')
ax.set_ylabel('lg(time)')

ax.plot(x, y)
plt.show()
```

4. Results

```
============= RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:100
time: 0.0009996891021728516
>>>
============= RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:1000
time: 0.006010532379150391
>>>
============= RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:10000
time: 0.32295680046081543
>>>
============= RESTART: D:/课件/物理学/计算物理/Project1/Project 1test.py =======
======
choose n for matrix:10
time: 0.0
```

Code results

For n = 10, the time is nearly zero, and I use 0.00001 to replace zero(cause lg(0) doesn't exist)

For n = 100, the time is 0.0009997s

For n = 1000, the time is 0.0060105s

For n = 10000, the time is 0.3229568s

For n = 100000, memory error will happen so I can't get the data and the time for n>100000

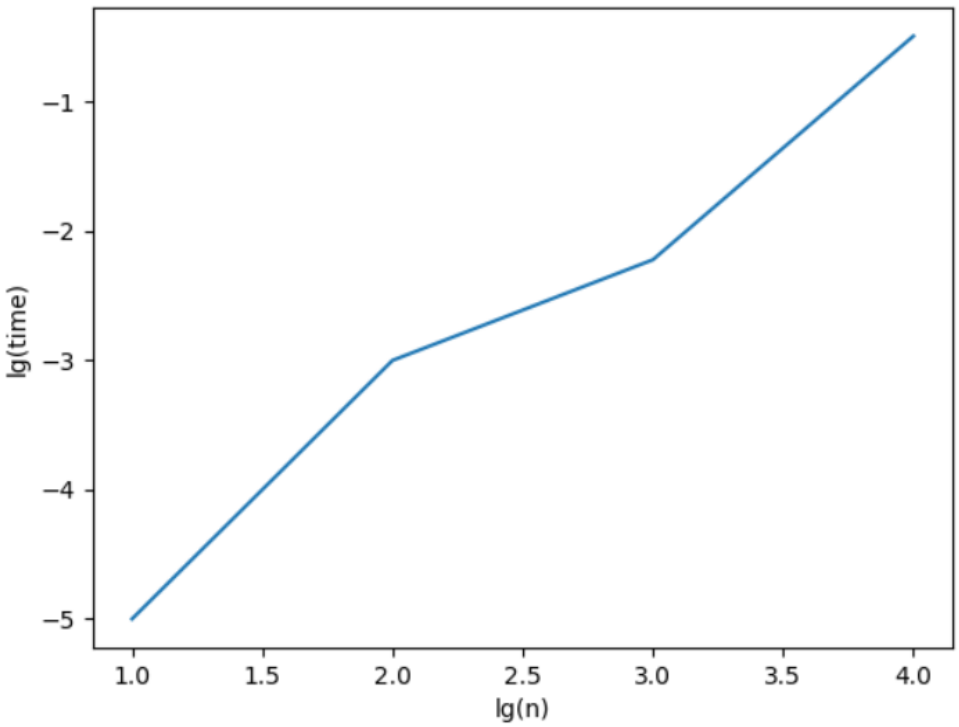

Image figure of lg(n) and lg(time)

For linear fit of this figure, the result is y = 1.431 – 6.225

The result is a good proof of the previously expected of linear relationship, and also reflects the number of operations required to simplify the matrix.

5. Concluding Remarks

With importance sampling and exponential distribution, are good and accurate. Using these methods we get away with a lot less evaluations of the integrand. This "effect" gets stronger as the dimensions of the integral n increased. Using this algorithm is also pretty satisfactory. However, if n > 10000, the memory is very limited, which will cause error.

6. References

Hjort-Jensen,M., 2015. Computational physics.

The program can be found in https://github.com/lch172061365/Computational-Physics.git