EE2703 - Assignment 8

Ata Karim Subhani <EE21B025>

April 16, 2023

1 Instructions

- Apart from this report file I have given two notebook one for assignment 8 and assignment 2 (only circuit solving part) to test/compare outputs.
- In this report I have briefly explained the working of **Cython**.
- Also I have given one .netlist file on which I have compared the output and compare speed of the Cython with Python.

2 AIM:

To implement Assignment 2 using **Cython** to speed up the code.

3 Factorial Computation

[]:
```
def factorial(N):
    fact = []
    fact.append(1)
    fact.append(1)
    for i in range(2, N+1):
        fact.append(i*fact[i-1])
    return fact[N]
```

[2]:
```
%timeit factorial(40)
```

```
3.5 µs ± 76.7 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

[3]:
```
%load_ext Cython
```

[4]:
```
%%cython
import cython
import numpy as np
cimport numpy as np
@cython.boundscheck(False)
@cython.cdivision(True)
def c_factorial(int N):
    cdef int[100000] fact
```

```
    cdef int i
    fact[0] = 1
    fact[1] = 1
    for i in range(2, N+1):
        fact[i] = i*fact[i-1]
    return fact[N]
```

[5]:
```
%timeit c_factorial(40)
%timeit np.math.factorial(40)
```

```
63.7 ns ± 2.54 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
313 ns ± 7.33 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

3.1 Inferences:

- Time taken by **Python** implementation is **3.5 μs**
- Time taken by **Cython** implementation is **63.7 ns**
- Time taken by **numpy** implementation is **313 ns**

I have been able to speed up my python code by almost 50 times by using cython.

4 Optimizing "System of linear equation" solver

Procedure

- I have declared all the variables to its corresponding data types so while converting the python file to C file, they don't need to check data type of any variable. It saved time, while in python we need to keep track of last assigned values to all variables which make it slower.
- Also I make"@cython.cdivision" decorator true, so it would not check zero division error as I have taken care of that while writing the code. So it also saved running time of code.
- Also I make"@cython.boundscheck" decorator false, so it would not check if the index of array is out of bound or not. So it also saving running time.
- Since we have to do AC analysis of circuit too, hence all the matrices are complex in nature. As there is no builtin data type like complex in C language so Cython is converting the complex data type using some inbuilt function or library into C language code.
- The Cython code could have been more faster, if I would use only float data type for matrices.

[6]:
```
%load_ext Cython
```

[13]:
```
%%cython
import cython
@cython.cdivision(True)
@cython.boundscheck(False)
cdef C_Partial_Pivot(list A,list b,int k):
    cdef complex big = A[k][k]
    cdef int index = k
```

```
    cdef int i, j, m, n
    cdef complex temp
    m = len(A)
    n = len(A[0])
    for i in range(k+1, m):
        if (abs(A[i][k]) > abs(big)):
            big =  A[i][k]
            index = i
    if (index != k):
        for j in range(k, n):
            temp= A[k][j]
            A[k][j] = A[index][j]
            A[index][j] = temp
        # swap element of 'b' vector
        temp = b[k]
        b[k] = b[index]
        b[index] = temp


@cython.cdivision(True)
@cython.boundscheck(False)
cdef C_REF(list A, list b):
    cdef int m = len(A)  # m is no. of rows.
    cdef int n = len(A[0])  # n is no. of columns.
    cdef int i, j, k
    cdef complex factor
    for k in range(n):
        # Pivoting the element.
        C_Partial_Pivot(A, b, k)
        for i in range(k+1, m):
            factor = A[i][k]/A[k][k]
            b[i] = b[i] - factor*b[k]
            for j in range(k, n):
                A[i][j] = A[i][j] - factor*A[k][j]


@cython.cdivision(True)
@cython.boundscheck(False)
def C_solve(list A, list b):
    cdef int m = len(A)  # m is no. of rows (equivalent to no. of equation)
    cdef int n = len(A[0])  # n is no. of columns (equivalent to no. of variable.
 ↪)
    cdef int NumOfNonZerosRow = 0
    cdef int i, j, cnt
    cdef int row, col
    # cdef complex add
    if (m < n):
```

```
        # We have less no. of equations than number of variable.
        return "Insufficient number of linearly independent equations, So there␣
↪are infinite solutions"
    # Convert coefficient matrix to Row Echelon Form
    C_REF(A,b)
    # now count the number rows in Row echelon form of coefficient matrix with␣
↪all zero's element.
    for i in range(m):
        for j in range(n):
            cnt = 0  # no. of non zeros element in a row.
            if (A[i][j] != 0):
                cnt = 1
                break
        if (cnt == 1):  # there is atleast one non zero element in that row
            NumOfNonZerosRow += 1
            continue
        if (cnt == 0):  # all element is zero
            if (b[i] != 0):  # It means that
                return "Inconsistent system, Hence No Solution"
    # Num of non zeros row is the number of linearly independent equations in␣
↪given system of equations
    if (NumOfNonZerosRow < n):
        return "Insufficient number of linearly independent equations, So there␣
↪are infinite solutions"

    # Note: Number of non zeros row can't be less than n provided(m>=n)

    # Back Substitution.
    if (NumOfNonZerosRow == n):
        x = []  # x is final solution vector
        # x.clear()
        for i in range(n):
            x.append(0)
        for row in range(n-1, -1, -1):
            add = 0
            for col in range(row+1, n):
                add += A[row][col]*x[col]
            x[row] = (b[row]-add)/A[row][row]
        return x
```

[8]:
```
import numpy as np
n = 10
A = np.random.rand(n, n)
b = np.random.rand(n)
```

[14]:
```
%timeit C_solve(A.tolist(),b.tolist())
```

```
34 µs ± 2.01 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

[15]:
```
%timeit np.linalg.solve(A,b)
```

```
17.6 µs ± 320 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

4.1 Inferences

I have solve randomly generated 10x10 matrix and compared the result.

- Time taken with **cython** is **34 µs**
- Time taken with **numpy** is **17.6 µs**
- Time taken with **Python** is **242 µs** (refer assignment2.ipynb file uploaded in this week zip folder.)

Hence I have speed up the code 7 times.

5 SPICE

Here I could have used data type assignment to all the variable and then solved the matrix, but to avoid any kind of mess with the code I did not do that, since we are not doing any computation in below block. Only purpose of below block of code is to read the netlist file and make the conductance matrix and then call 'C_solve' function to solve the circuit.

Since all kind of computation is being done by 'C_solve' function, so most of the running time would be decided by that only. Hence I avoided here using any kind of cython syntax.

[33]:
```
from math import pi, sin, cos
import cmath
import time
def AC_MNA(content_of_netlist, start, end):

    freq = set()
    f=0
    for line in content_of_netlist[end+1:]:
        if(line.split()[0]=='.ac'):
            f = float(line.split()[2])
            freq.add(line.split()[2]) #putting frequencies in set "freq"
        if(len(freq)>1):
            return "Circiut Contains more than one frequency source."
    w = 2*pi*f
    # Counting number of nodes and Voltage source.
    st = set()
    n = 0
    Nv = 0
    for line in content_of_netlist[start+1:end]:
        if(len(line.split())==0): # when line is blank
            continue
```

```
        elif(line[0]=='#'): # when line is comment
            continue
        else:
            st.add(line.split()[1])
            st.add(line.split()[2])
            if(line[0]=='V'):
                Nv+=1
    n = len(st)-1 # no. of nodes
    size = n + Nv # size of conductance matrix.
    A = [] # conductance matrix
    B = []
    #initialising conductance matrix (A) and B vector.
    for row in range(size):
        lt = []
        for col in range(size):
            lt.append(0)
        A.append(lt)
        B.append(0)
    a, b = 0, 0
    for line in content_of_netlist[start+1:end]:
        if(len(line.split())==0):
            continue
        discription = [word for word in line.split('#')[0].split()]
        # Operations to store node number in a and b.
        if(discription[1]=='GND'):
            discription[1]='n0'
        elif(discription[2]=='GND'):
            discription[2]='n0'
        a, b = discription[1], discription[2] # a and b store nodes information
        if(a[0]=='n'): # when nodes are in form of n1, n2, .....
            a = int(a[1:])
        if(b[0]=='n'):
             b = int(b[1:])
        a, b = int(a), int(b)  # here finally a and b stores node number.

        if(len(line.split())==0): # when line is blank
            continue
        elif(line[0]=='#'): # when line start with comment
            continue

        elif(line[0]=='R'):
            if(a==0 and b!=0):
                A[b-1][b-1] += 1/float(discription[-1])
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/float(discription[-1])
            else:
                A[b-1][b-1] += 1/float(discription[-1])
```

```
                A[a-1][a-1] += 1/float(discription[-1])
                A[a-1][b-1] -= 1/float(discription[-1])
                A[b-1][a-1] -= 1/float(discription[-1])
            continue

        elif(line[0]=="L"):
            Z = complex(0, float(discription[-1])*w)

            if(a==0 and b!=0):
                A[b-1][b-1] += 1/Z
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/Z
            else:
                A[b-1][b-1] += 1/Z
                A[a-1][a-1] += 1/Z
                A[a-1][b-1] -= 1/Z
                A[b-1][a-1] -= 1/Z
            continue
        elif(line[0]=="C"):
            Z = complex(0, 1/(float(discription[-1])*w))

            if(a==0 and b!=0):
                A[b-1][b-1] += 1/Z
            elif(b==0 and a!=0):
                A[a-1][a-1] += 1/Z
            else:
                A[b-1][b-1] += 1/Z
                A[a-1][a-1] += 1/Z
                A[a-1][b-1] -= 1/Z
                A[b-1][a-1] -= 1/Z
            continue
        elif(line[0]=='V'):
            x = n + int(discription[0][1:])-1
            V_mag = float(discription[-2])
            phase = float(discription[-1])
            B[x] = complex(V_mag*cos(phase*(pi/180)), V_mag*sin(phase*(pi/180)))␣
↪# updating B vector with value of Voltage of source.
            if(a==0 and b!=0):
                A[x][b-1] = -1
                A[b-1][x] = -1
            elif(b==0 and a!=0):
                A[x][a-1] = 1
                A[a-1][x] = 1
            else:
                A[x][b-1] = -1
                A[x][a-1] = 1
                A[a-1][x] = 1
```

```
                A[b-1][x] = -1
            continue
        elif(line[0]=='I'):
            I_mag = float(discription[-2])
            phase = float(discription[-1])
            I =  complex(I_mag*cos(phase*(pi/180)), I_mag*sin(phase*(pi/180)))
            if(a==0 and b!=0):
                B[b-1] += I
            elif(b==0 and a!=0):
                B[a-1] -=I
            else:
                B[a-1] -=I
                B[b-1] += I
    x = C_solve(A, B)
    for i in range(n):
        print(f"Node {i+1} Voltage : magnitude = {abs(x[i])} V, phase = {cmath.
↪phase(x[i])*(180/pi)} degrees")
    for i in range(Nv):
        print(f"Current through Voltage source V{i+1} : magnitude = {␣
↪abs(x[n+i])} A, phase =  {cmath.phase(x[n+i])*(180/pi)} degrees")
    # print(f"\n\n\nTime to solve the circuit {t2-t1}")
    return ""
```

[34]:
```
def Solve_Circuit(filename):
    with open(filename, 'r') as netlist:
        content_of_netlist = netlist.readlines()
        start, end = 0, 0
        for i in range(len(content_of_netlist)):
            if(len(content_of_netlist[i].split())==0):
                continue
            if(content_of_netlist[i].split()[0] == '.circuit'):
                start = i
            elif(content_of_netlist[i].split()[0]=='.end'):
                end = i
        if(end < start):
            return "Invalid Circuit Declaration"
       #Check dc or ac source(s).
        dc = False
        ac = False
        for lines in content_of_netlist:
            if(lines[0]=='V' or lines[0]=='I'):
                if(lines.split()[3]=='dc'):
                    dc = True
                elif(lines.split()[3]=='ac'):
                    ac = True
        if(ac and dc):
```

```
            return "Given circuit has more than one frequency.(since frequency␣
↪of DC source is zero)"
        elif(dc==True):
            return DC_MNA(content_of_netlist, start, end)
        elif(ac==True):
            return AC_MNA(content_of_netlist, start, end)
```

Output

[35]:
```
t1 = time.time()
print(Solve_Circuit("AC1.netlist"))
t2 = time.time()
```

```
Node 1 Voltage : magnitude = 4.0 V, phase = 29.999999999999996 degrees
Node 2 Voltage : magnitude = 6.000000000000001 V, phase = 59.99999999999999
degrees
Node 3 Voltage : magnitude = 168.4333450781387 V, phase = 39.27476850443678
degrees
Node 4 Voltage : magnitude = 91.96228125341496 V, phase = 158.0780671886268
degrees
Node 5 Voltage : magnitude = 96.14432133702611 V, phase = 152.5410015060631
degrees
Node 6 Voltage : magnitude = 8.9239583734777 V, phase = 38.33806605740891
degrees
Node 7 Voltage : magnitude = 533.034650816758 V, phase = 21.99380938872583
degrees
Current through Voltage source V1 : magnitude = 2.090577954005548 A, phase =
153.4094702189483 degrees
Current through Voltage source V2 : magnitude = 5.1277700946880405 A, phase =
-118.8825666749316 degrees
Current through Voltage source V3 : magnitude = 2.002684775001888 A, phase =
157.20350346613174 degrees
Current through Voltage source V4 : magnitude = 20.250017332467316 A, phase =
21.54500746452838 degrees
```

Computation time

[38]:
```
print(f"Time to solve the circuit {t2-t1}")
```

```
Time to solve the circuit 0.0008935928344726562
```

5.1 Inferences:

Time taken by **cython** implementation is **0.89 ms** while, with **python** implementation it takes **1.9 ms** (see assignment2.ipynb file uploaded in this week zip file).

Note:
I have not included pure python based implementation in this Assignment8.ipynb file to avoid any type of mess in report file.