Week8Prob

GVV Praneeth Reddy <EE21B048>

April 17, 2023

[1]:
```
import math
import numpy as np
```

[2]:
```
def lin_eqn_sol(A, b):
    try:
        aug=A
        r = len(A)
        c = r
        for i in range(r):
            aug[i].append(b[i])
        for i in range(r):
            if aug[i][i] ==0:
                a=0
                for j in range(i+1,r):
                    if a==1:
                        break
                    if aug[j][j]!= 0:
                        for k in range(r+1):
                            t=aug[i][k]
                            aug[i][k]=aug[j][k]
                            aug[j][k]= t
                            a=1
                            break
        for i in range(r):
            aug[i] = [aug[i][j]/aug[i][i] for j in range(c+1)]
            for k in range(i+1,r):
                aug[k] = [aug[k][p] - aug[i][p]*(aug[k][i]/aug[i][i]) for p in␣
 ↪range(c+1)]

        x = [0 for q in range(c)]
        for z in range(r-1,-1,-1):
            u = aug[z][c] - sum([aug[z][j]*x[j] for j in range(z+1,r)])
            if u==0 and aug[z][z]==0:
                return "The given system of equations has infinite solutions"
            elif u!=0 and aug[z][z]==0:
                return "The given system of linear equations is inconsistent"
            else:
```

```
                x[z] = u/aug[z][z]
            return x
    except ValueError:
        print("The input matrix contains elements other than numbers")
```

This the linear equation solver function written using python from the 2nd assignment

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

[4]:
```
%%cython --annotate
import cython
@cython.cdivision(True)
@cython.boundscheck(False)
@cython.wraparound(False)
cpdef c_lin_eqn_sol(A,b):
    cdef int r = len(A)
    cdef int c = r
    cdef list x = [0]*r
    cdef list aug = [[0.0]* (c+1) for i in range(r)]
    cdef int i1,j1,i, j, k, p, z
    cdef complex t, u
    try:
        for i1 in range(r):
            for j1 in range(c):
                aug[i1][j1] = A[i1][j1]
            aug[i1][c] = b[i1]
        for i in range(r):
            if aug[i][i] ==0:
                a=0
                for j in range(i+1,r):
                    if a==1:
                        break
                    if aug[j][j]!= 0:
                        for k in range(r+1):
                            t=aug[i][k]
                            aug[i][k]=aug[j][k]
                            aug[j][k]= t
                            a=1
                            break
        for i in range(r):
            aug[i] = [aug[i][j]/aug[i][i] for j in range(c+1)]
            for k in range(i+1,r):
                aug[k] = [aug[k][p] - aug[i][p]*(aug[k][i]/aug[i][i]) for p in␣
 ↪range(c+1)]

        x = [0 for q in range(c)]
        for z in range(r-1,-1,-1):
```

```
            u = aug[z][c] - sum([aug[z][j]*x[j] for j in range(z+1,r)])
            if u==0 and aug[z][z]==0:
                return "The given system of equations has infinite solutions"
            elif u!=0 and aug[z][z]==0:
                return "The given system of linear equations is inconsistent"
            else:
                x[z] = u/aug[z][z]
        return x
    except ValueError:
        print("The input matrix contains elements other than numbers")
```

[4]: <IPython.core.display.HTML object>

This the linear equation solver function written using cython

[5]:
```
r = int(input("Enter the number of rows of the coefficient matrix: "))
c = int(input("Enter the number of columns of the coefficient matrix: "))

if r!=c:
    print("The coefficient matrix should be a square matrix")
else:

    input_A=input("Enter the coefficient matrix elements separated by spaces:")
    elements = input_A.split()
    A = [list(map(complex, elements[i:i+c])) for i in range(0, r*c, c)]
    print("Enter the constant matrix with a single space between the elements:")
    b=list(map(complex, input().split()))
    A_c=[]
    b_c=[]
    A_c.extend(A)
    b_c.extend(b)
    print(A,b)
```

```
Enter the number of rows of the coefficient matrix:  10
Enter the number of columns of the coefficient matrix:  10
Enter the coefficient matrix elements separated by spaces: 1 2 3 4 5 6 7 8 9 10
11 10 12 36 25 98 65 32 14 12 2 21 3 36 39 56 42 58 51 53 10 11 23 15 14 18 19
16 32 25 74 75 96 58 42 16 35 25 69 96 38 39 36 54 52 50 15 14 17 18 1 2 6 55 42
32 95 47 49 2 88 75 94 86 53 50 12 18 19 20 15 75 82 83 81 10 12 19 17 57 24 26
51 57 58 49 48 20 23 24

Enter the constant matrix with a single space between the elements:

 100 101 589 54 632 759 666 21 230 25

[[(1+0j), (2+0j), (3+0j), (4+0j), (5+0j), (6+0j), (7+0j), (8+0j), (9+0j),
(10+0j)], [(11+0j), (10+0j), (12+0j), (36+0j), (25+0j), (98+0j), (65+0j),
(32+0j), (14+0j), (12+0j)], [(2+0j), (21+0j), (3+0j), (36+0j), (39+0j), (56+0j),
(42+0j), (58+0j), (51+0j), (53+0j)], [(10+0j), (11+0j), (23+0j), (15+0j),
(14+0j), (18+0j), (19+0j), (16+0j), (32+0j), (25+0j)], [(74+0j), (75+0j),
```

```
(96+0j), (58+0j), (42+0j), (16+0j), (35+0j), (25+0j), (69+0j), (96+0j)],
[(38+0j), (39+0j), (36+0j), (54+0j), (52+0j), (50+0j), (15+0j), (14+0j),
(17+0j), (18+0j)], [(1+0j), (2+0j), (6+0j), (55+0j), (42+0j), (32+0j), (95+0j),
(47+0j), (49+0j), (2+0j)], [(88+0j), (75+0j), (94+0j), (86+0j), (53+0j),
(50+0j), (12+0j), (18+0j), (19+0j), (20+0j)], [(15+0j), (75+0j), (82+0j),
(83+0j), (81+0j), (10+0j), (12+0j), (19+0j), (17+0j), (57+0j)], [(24+0j),
(26+0j), (51+0j), (57+0j), (58+0j), (49+0j), (48+0j), (20+0j), (23+0j),
(24+0j)]] [(100+0j), (101+0j), (589+0j), (54+0j), (632+0j), (759+0j), (666+0j),
(21+0j), (230+0j), (25+0j)]
```

This cell takes two matrices as input. Here i have given a 10x10 matrix as A and 10x1 matrix as B.

[6]:
```
print(lin_eqn_sol(A, b))
%timeit lin_eqn_sol(A, b)
```

```
[(-12.186839273516398+0j), (-277.9486378443721-0j), (-99.559692892257-0j),
(496.10199269871254+0j), (-231.01607360104833+0j), (19.479560717572447+0j),
(-101.230989325415+0j), (-186.88277737895635+0j), (33.49044217633475+0j),
(192.2823391266532+0j)]
126 µs ± 3.28 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Here i am calling the function written using python and it takes 126 μ s

[7]:
```
print(c_lin_eqn_sol(A_c, b_c))
%timeit c_lin_eqn_sol(A_c, b_c)
```

```
[(-12.186839273516398+0j), (-277.9486378443721-0j), (-99.559692892257-0j),
(496.10199269871254+0j), (-231.01607360104833+0j), (19.479560717572447+0j),
(-101.230989325415+0j), (-186.88277737895635+0j), (33.49044217633475+0j),
(192.2823391266532+0j)]
47.3 µs ± 1.55 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

The function written using cython takes 47.3 μ s

In the cython function i have used cdivision and set bounds check and wrap arround to false because i am not giving any negative indices of out of bound indicies. I have also declared all the variable types using cdef. All these will help the code to run faster than a normal python code where the variables are not declared initially.

[8]:
```
def circuit_solver(filename):
    with open(filename,"r") as ckt:
        circuit=ckt.readlines()

        nodes=set()
        VS_count=0
        c=0
        e=0

        for l in circuit:
```

```
        l=l.split()
        if l[0] == ".circuit":
            c = 1
            continue
        if l[0]==".ac":
            freq=float(l[2])*2*(math.pi)
        elif l[0] == ".end":
            e = 1
            continue
        if c==1 and e==1:
            break
        if c ==1 and e==0:
            if l[0][0]=='V':
                VS_count+=1
            if l[1]!='GND':
                nodes.add(int(l[1]))
            if l[2]!='GND':
                nodes.add(int(l[2]))

    n=max(nodes)
    N=n+VS_count

    A=[[complex(0) for _ in range(N)]for _ in range(N)]
    B=[complex(0) for _ in range(N)]
    c=0
    e=0
    r=n

    for l in circuit:
        l=l.split()
        if l[0] == ".circuit":
            c = 1
            continue
        if l[0]==".ac":
            w=float(l[2])
        elif l[0] == ".end":
            e = 1
            continue
        if c==1 and e==1:
            break
        if c ==1 and e==0:
            if l[0][0]=='R':
                value=float(l[3])
                if l[1]!='GND' and l[2]!='GND':
                    n_1=int(l[1])-1
                    n_2=int(l[2])-1
                    A[n_1][n_1]+=1/value
```

```
                A[n_2][n_2]+=1/value
                A[n_1][n_2]-=1/value
                A[n_2][n_1]-=1/value
            elif l[1]!='GND' and l[2]=='GND':
                n_1=int(l[1])-1
                A[n_1][n_1]+=1/value
            elif l[1]=='GND' and l[2]!='GND':
                n_2=int(l[2])-1
                A[n_2][n_2]+=1/value

        if l[0][0]=='C':
            value=float(l[3])
            if l[1]!='GND' and l[2]!='GND':
                n_1=int(l[1])-1
                n_2=int(l[2])-1
                A[n_1][n_1]+=value*freq*1j
                A[n_2][n_2]+=value*freq*1j
                A[n_1][n_2]-=value*freq*1j
                A[n_2][n_1]-=value*freq*1j
            elif l[1]!='GND' and l[2]=='GND':
                n_1=int(l[1])-1
                A[n_1][n_1]+=value*freq*j
            elif l[1]=='GND' and l[2]!='GND':
                n_2=int(l[2])-1
                A[n_2][n_2]+=value*freq*1j

        if l[0][0]=='L':
            value=float(l[3])
            if l[1]!='GND' and l[2]!='GND':
                n_1=int(l[1])-1
                n_2=int(l[2])-1
                A[n_1][n_1]+=1/(value*freq*1j)
                A[n_2][n_2]+=1/(value*freq*1j)
                A[n_1][n_2]-=1/(value*freq*1j)
                A[n_2][n_1]-=1/(value*freq*1j)
            elif l[1]!='GND' and l[2]=='GND':
                n_1=int(l[1])-1
                A[n_1][n_1]+=1/(value*freq*1j)
            elif l[1]=='GND' and l[2]!='GND':
                n_2=int(l[2])-1
                A[n_2][n_2]+=1/(value*freq*1j)

        elif l[0][0]=='V':
            type = l[3]
            value = float(l[4])
            if type=='ac':
                phase = float(l[5])
```

```python
                    B[r]=value
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        A[r][n_1]+=1
                        A[r][n_2]-=1
                        A[n_1][r]+=1
                        A[n_2][r]-=1
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        A[r][n_1]+=1
                        A[n_1][r]+=1
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        A[r][n_2]-=1
                        A[n_2][r]-=1
                    r+=1

                elif l[0][0]=='I':
                    type = l[3]
                    value = float(l[4])
                    if type=='ac':
                        phase = float(l[5])
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        B[n_1]-=value
                        B[n_2]+=value
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        B[n_1]-=value
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        B[n_2]+=value
    # print(A)
    # print(B)
    return lin_eqn_sol(A, B)
```

[10]:
```python
filename= input("Enter the file name: ")
print(circuit_solver(filename))
%timeit circuit_solver(filename)
```

```
Enter the file name:  ckt1.netlist

[0j, 0j, 0j, (-5+0j), (-0.0005-0j)]
76.1 µs ± 1.6 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

This the circuit solver function from 2nd assignment where i am using the python linear equation solver and it takes 76.1μs

[11]:
```python
def c_circuit_solver(filename):
    with open(filename,"r") as ckt:
        circuit=ckt.readlines()

        nodes=set()
        VS_count=0
        c=0
        e=0

        for l in circuit:
            l=l.split()
            if l[0] == ".circuit":
                c = 1
                continue
            if l[0]==".ac":
                freq=float(l[2])*2*(math.pi)
            elif l[0] == ".end":
                e = 1
                continue
            if c==1 and e==1:
                break
            if c ==1 and e==0:
                if l[0][0]=='V':
                    VS_count+=1
                if l[1]!='GND':
                    nodes.add(int(l[1]))
                if l[2]!='GND':
                    nodes.add(int(l[2]))

        n=max(nodes)
        N=n+VS_count

        A=[[complex(0) for _ in range(N)]for _ in range(N)]
        B=[complex(0) for _ in range(N)]
        c=0
        e=0
        r=n

        for l in circuit:
            l=l.split()
            if l[0] == ".circuit":
                c = 1
                continue
            if l[0]==".ac":
                w=float(l[2])
            elif l[0] == ".end":
                e = 1
```

```
                continue
            if c==1 and e==1:
                break
            if c ==1 and e==0:
                if l[0][0]=='R':
                    value=float(l[3])
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        A[n_1][n_1]+=1/value
                        A[n_2][n_2]+=1/value
                        A[n_1][n_2]-=1/value
                        A[n_2][n_1]-=1/value
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        A[n_1][n_1]+=1/value
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        A[n_2][n_2]+=1/value

                if l[0][0]=='C':
                    value=float(l[3])
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        A[n_1][n_1]+=value*freq*1j
                        A[n_2][n_2]+=value*freq*1j
                        A[n_1][n_2]-=value*freq*1j
                        A[n_2][n_1]-=value*freq*1j
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        A[n_1][n_1]+=value*freq*j
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        A[n_2][n_2]+=value*freq*1j

                if l[0][0]=='L':
                    value=float(l[3])
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        A[n_1][n_1]+=1/(value*freq*1j)
                        A[n_2][n_2]+=1/(value*freq*1j)
                        A[n_1][n_2]-=1/(value*freq*1j)
                        A[n_2][n_1]-=1/(value*freq*1j)
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
```

```
                        A[n_1][n_1]+=1/(value*freq*1j)
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        A[n_2][n_2]+=1/(value*freq*1j)

                elif l[0][0]=='V':
                    type = l[3]
                    value = float(l[4])
                    if type=='ac':
                        phase = float(l[5])
                    B[r]=value
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        A[r][n_1]+=1
                        A[r][n_2]-=1
                        A[n_1][r]+=1
                        A[n_2][r]-=1
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        A[r][n_1]+=1
                        A[n_1][r]+=1
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        A[r][n_2]-=1
                        A[n_2][r]-=1
                    r+=1

                elif l[0][0]=='I':
                    type = l[3]
                    value = float(l[4])
                    if type=='ac':
                        phase = float(l[5])
                    if l[1]!='GND' and l[2]!='GND':
                        n_1=int(l[1])-1
                        n_2=int(l[2])-1
                        B[n_1]-=value
                        B[n_2]+=value
                    elif l[1]!='GND' and l[2]=='GND':
                        n_1=int(l[1])-1
                        B[n_1]-=value
                    elif l[1]=='GND' and l[2]!='GND':
                        n_2=int(l[2])-1
                        B[n_2]+=value
    # print(A)
    # print(B)
    return c_lin_eqn_sol(A, B)
```

[12]:
```
filename_c= input("Enter the file name: ")
print(c_circuit_solver(filename_c))
%timeit c_circuit_solver(filename_c)
```

```
Enter the file name:  ckt1.netlist

[0j, 0j, 0j, (-5+0j), (-0.0005-0j)]
58.1 µs ± 768 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

This the circuit solver function in which i am using the cython linear eqaution solver and it takes 58.1 μ s