week2new

Varsha S P, EE21B154 <ee21b154@smail.iitm.ac.in>

February 8, 2023

1 Assignment

The following are the problems you need to solve for this assignment. You need to submit your code (either as standalone Python script or a Python notebook), a PDF document explaining your solution (either generated from the notebook or a separate LaTeX document), and any supporting files you may have (such as circuit netlists you used for testing your code).

- Write a function to find the factorial of N (N being an input) and find the time taken to compute it. This will obviously depend on where you run the code and which approach you use to implement the factorial. Explain your observations briefly.
- Write a linear equation solver that will take in matrices A and b as inputs, and return the vector x that solves the equation $Ax = b$. Your function should catch errors in the inputs and return suitable error messages for different possible problems.
  - Time your solver to solve a random 10×10 system of equations. Compare the time taken against the `numpy.linalg.solve` function for the same inputs.
- Given a circuit netlist in the form described above, read it in from a file, construct the appropriate matrices, and use the solver you have written above to obtain the voltages and currents in the circuit. If you find AC circuits hard to handle, first do this for pure DC circuits, but you should be able to handle both voltage and current sources.

1.1 Problem statement 1

Write a function to find the factorial of N (N being an input) and find the time taken to compute it. This will obviously depend on where you run the code and which approach you use to implement the factorial. Explain your observations briefly.

[16]:
```python
import numpy as np
x=int(input("Enter a number"))
def fact(x):
    return np.math.factorial(x)
if type(x)==int and x>0:
    print(fact(x))
else:
    print("Enter a positive integer")

'''print(fact(x))'''
%timeit fact(x)
```

```
Enter a number 4
```

```
24
142 ns ± 4.84 ns per loop (mean ± std. dev. of 7 runs, 10,000,000 loops each)
```

[17]:
```
import numpy as np
x=float(input("Enter a number"))

def fact(x):
    a=1
    for i in range(1,int (x)+1):
        a=a*(i)

    return a

if x.is_integer()==True and x>0:
    print(fact(x))
else:
    print("Enter a positive integer")
%timeit fact(x)
```

```
Enter a number 4

24
337 ns ± 12 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

1.1.1 Explanation

The function written for factorial using recursion takes more time and lesser loops compared to numpy function math.factorial().

1.1.2 Explanation

We know that if determinant of a square matrix is zero, it has no solution or has infinite solutions. Hence, we are first checking for the det of the entered matrix. Then for non-zero elements lambada is taken and then subtracted from every row. After getting zeroes in lower traingle, we do back substitution.

1.1.3 Timing the defined function vs in.built fuction

Time your solver to solve a random 10×10 system of equations. Compare the time taken against the numpy.linalg.solve function for the same inputs.

[21]:
```
A= np.array([[1 ,3 ,1 ,2 ,6 ,6 ,0 ,1 ,3 ,5 ],
[7 ,0 ,2 ,0 ,5 ,5 ,6 ,3 ,3 ,3 ],
[6 ,0 ,6 ,0 ,0 ,8 ,4 ,5 ,3 ,7 ],
[8 ,5 ,4 ,9 ,3 ,5 ,3 ,5 ,8 ,7 ],
[7 ,6 ,3 ,8 ,9 ,2 ,3 ,8 ,7 ,8 ],
[9 ,5 ,7 ,0 ,7 ,7 ,0 ,1 ,8 ,6 ],
[3 ,9 ,7 ,9 ,2 ,1 ,7 ,6 ,7 ,1 ],
[8 ,5 ,6 ,4 ,4 ,0 ,3 ,7 ,2 ,5 ],
[1 ,2 ,7 ,6 ,1 ,5 ,2 ,0 ,8 ,1 ],
[6 ,4 ,4 ,3 ,6 ,2 ,7 ,8 ,5 ,2 ]] ,float)
B= np.array( [[2],
[3],
[4],
[1],
[2],
[2],
[2],
[9],
[7],
[5]],float)

print(gaussElim(A,B))

%timeit gaussElim(A,B)
```

```
[[ 0.57592865]
 [-1.1434718 ]
 [ 1.700912  ]
 [ 1.56273285]
 [ 1.1733649 ]
 [ 1.36712171]
 [-1.35775437]
 [ 1.04556496]
 [-1.97475077]
 [-2.06722465]]
358 µs ± 15.6 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

[23]:
```
A= np.array([[1 ,3 ,1 ,2 ,6 ,6 ,0 ,1 ,3 ,5 ],
[7 ,0 ,2 ,0 ,5 ,5 ,6 ,3 ,3 ,3 ],
[6 ,0 ,6 ,0 ,0 ,8 ,4 ,5 ,3 ,7 ],
[8 ,5 ,4 ,9 ,3 ,5 ,3 ,5 ,8 ,7 ],
[7 ,6 ,3 ,8 ,9 ,2 ,3 ,8 ,7 ,8 ],
[9 ,5 ,7 ,0 ,7 ,7 ,0 ,1 ,8 ,6 ],
[3 ,9 ,7 ,9 ,2 ,1 ,7 ,6 ,7 ,1 ],
[8 ,5 ,6 ,4 ,4 ,0 ,3 ,7 ,2 ,5 ],
[1 ,2 ,7 ,6 ,1 ,5 ,2 ,0 ,8 ,1 ],
[6 ,4 ,4 ,3 ,6 ,2 ,7 ,8 ,5 ,2 ]] )
B= np.array( [[2],
[3],
[4],
[1],
[2],
[2],
[2],
[9],
[7],
[5]])
print(np.linalg.solve(A,B))
%timeit np.linalg.solve(A, B)
```

```
[[ 0.57592865]
 [-1.1434718 ]
 [ 1.700912  ]
 [ 1.56273285]
 [ 1.1733649 ]
 [ 1.36712171]
 [-1.35775437]
 [ 1.04556496]
 [-1.97475077]
 [-2.06722465]]
19.1 µs ± 754 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

1.1.4 Explanation

After using %timeit we see that our solver is slower than the in.built numpy fucntiom

1.2 Gaussian Elimination

Write a linear equation solver that will take in matrices   and   as inputs, and return the vector   that solves the equation $=$. Your function should catch errors in the inputs and return suitable error messages for different possible problems.

[32]:
```
def dot(v1, v2):
    return sum(x*y for x, y in zip(v1, v2))

import numpy as np
def gaussElim(a,b):
    n = len(b)

    if np.linalg.det(abs(a))!=0:
        for k in range(0,n-1):
            for i in range(k+1,n):
                if a[i,k] != 0.0:
                    lamb = a [i,k]/a[k,k]

                    a[i,k+1:n] = a[i,k+1:n] - lamb*a[k,k+1:n]
                    b[i] = b[i] - lamb*b[k]
        for k in range(n-1,-1,-1):
            b[k] = (b[k] - dot(a[k,k+1:n],b[k+1:n]))/a[k,k]
        return b
    else:
        return "No solution or infinite solution for the given matrix"
```

1.3 SPICE simulator

Given a circuit netlist in the form described above, read it in from a file, construct the appropriate matrices, and use the solver you have written above to obtain the voltages and currents in the circuit. If you find AC circuits hard to handle, first do this for pure DC circuits, but you should be able to handle both voltage and current sources.

[54]:
```
import numpy as np
import sys
import cmath
def ckt(file):
    try:
        with open(file,'r') as f:
            data=f.read().split("\n")
    except Exception as ex:
        print (ex)
        return
    fre=0

    for l in data:
        if l=='':
            continue
        i=l.split()

        if i[0]=='.circuit':
            start=data.index(l)
        if i[0]=='.end':
            end=data.index(l)
        if i[0]=='.ac':
            fre=float(i[2])

        #ac
    for l in data:
        if l=='':#ac dc
            continue
        i=l.split()
        if i[0][0]=='V' and i[3]=='dc' and fre!=0:
            print("ac and dc found")
            sys.exit()

    return data[start+1:end],fre
```

1.4 Explanation

We have defined a fuction called ckt(), to read a .netlist and return values. ‘freq’ returns the value of frequency of the circuit is at all there is an AC source. Start, end is used to determine the number of nodes in a given circuit. We also check if both AC and DC source if detected the same circuit and print an error message.

```
[57]: def matrix():
          lines, fre=ckt('ckt3.netlist')
          nodes=[]
          nodes_var={}
          count=0
          curr=0

          for l in lines:
              if l=='':
                  continue
              i=l.split()
              for k in range(1,3):
                  if i[k] not in nodes:
                      nodes.append(i[k])
                      nodes_var[i[k]]=count
                      count+=1

          curr=0
          curr_var={}
          for l in lines:
              if l=='':
                  continue
              i=l.split()
              if i[0][0]=='V':
                  curr_var[i[0]]=(i[1],i[2],curr+count)
                  curr+=1


          mb=[0]*(curr+count)
          ma=[]
          for i in range(curr+count):
              ma.append([])
              for j in range(curr+count):
                  ma[i].append(0)

          for k in nodes_var.keys():
              for l in lines:
                  if l=='':
                      continue
                  i=l.split()
                  for m in range(1,3):
                      if i[m]==k:
                          if i[0][0]!='V' and i[0][0]!='I':
                              if i[0][0]=='R':
                                  add=float(i[3])
                                  if m==1:
                                      ma[nodes_var[k]][nodes_var[i[1]]]+=1/add
```

```
                                ma[nodes_var[k]][nodes_var[i[2]]]-=1/add
                            elif m==2:
                                ma[nodes_var[k]][nodes_var[i[2]]]+=1/add
                                ma[nodes_var[k]][nodes_var[i[1]]]-=1/add
                        if i[0][0]=='C':
                            add=(complex(0,-1))*(1/(fre*2*np.pi*float(i[3])))
                            if m==1:
                                ma[nodes_var[k]][nodes_var[i[1]]]+=1/add
                                ma[nodes_var[k]][nodes_var[i[2]]]-=1/add
                            elif m==2:
                                ma[nodes_var[k]][nodes_var[i[2]]]+=1/add
                                ma[nodes_var[k]][nodes_var[i[1]]]-=1/add
                        if i[0][0]=='L':
                            add = (float(i[3])*fre*2*np.pi*(complex(0,1)))
                            if m==1:
                                ma[nodes_var[k]][nodes_var[i[1]]]+=1/add
                                ma[nodes_var[k]][nodes_var[i[2]]]-=1/add
                            elif m==2:
                                ma[nodes_var[k]][nodes_var[i[2]]]+=1/add
                                ma[nodes_var[k]][nodes_var[i[1]]]-=1/add
                    elif i[0][0]=='I':
                        if m==1:
                            mb[nodes_var[k]]= -float(i[4])
                        elif m==2:
                            mb[nodes_var[k]]= float(i[4])
                    elif i[0][0]=='V':
                        if m==1:
                            ma[nodes_var[k]][curr_var[i[0]][2]]=1
                        if m==2:
                            ma[nodes_var[k]][curr_var[i[0]][2]]= -1
                if i[0][0]=='V' :
                    ma[curr_var[i[0]][2]][nodes_var[i[1]]]=1
                    ma[curr_var[i[0]][2]][nodes_var[i[2]]]=-1
                    mb[curr_var[i[0]][2]]=float(i[4])

    return nodes,nodes_var,curr,curr_var,ma,mb,add,count


nodes,nodes_var,curr,curr_var,ma,mb,add,count = matrix()

reference = nodes_var['GND']
del mb[reference]
for i in range(len(ma)):
    del ma[i][reference]
del ma[reference]
#print(ma, mb)
```

```
a=np.array(ma, dtype = complex)
b=np.array(mb, dtype = complex)


x = gaussElim(a ,b)

for i in range(count-1):
    print("V",i+1,"=",b[i],"V")
for i in range(curr):
    print("I",i+1,"=",b[i+count-1],"A")
```

```
V 1 = (-10+0j) V
V 2 = (-5.029239766081871+0j) V
V 3 = (-2.5730994152046778+0j) V
V 4 = (-1.4035087719298247+0j) V
V 5 = (-0.9356725146198832+0j) V
I 1 = (-0.004970760233918129-0j) A
```

1.5 Explanation

The file read using ckt() function is now called in the function matrix(), and the 2 values it returns are assigned to lines(list of lines in the netlist) and frequency. Now each line in the lines is read through and it's nodes are stored in 'nodes' and a dictionary is defined 'nodes_var' which stores all the nodes and a key is assigned to each one. Auxillary current is also stored in the curr_var. The sum of curr and count gives the total number of rows or columns in the matrix that needs to be solved.

Now, we define two matrices ma, mb and each element is defined to be zero initially. For each node, we read through the entire data and append to the appropriate matrix it's resistive, inductive or capacitive values. If voltage source is detected auxillary current is added to the matrix and also verified if it's AC or DC. From the acquired matric column and row of 'GND' is deleted as it's unecessary. The matrix generated is in list type, but our Gaussian solver takes in array, we use np.array. Now the gaussElim() is used to solve the two arguments passed ma,mb(a,b as array).