Week2Pres

February 8, 2023

1 Assignment

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.

[21]:
```python
import time

# Function to find the factorial of a number using recursion
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1) # function calling another instance of itself␣
 ↪in recurring manner

N = int(input("Enter the value of N: "))
start = time.time()
result = factorial(N)
end = time.time()

# Timing and printing the result of the function
%timeit factorial(N)
print("The factorial of", N, "is", result)
print("Time taken to compute the factorial:", end - start, "seconds")
```

```
2.25 µs ± 281 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
The factorial of 12 is 479001600
Time taken to compute the factorial: 0.00017833709716796875 seconds
```

This program is excecuted with a recurring function , that repeated call itself until the fucntion is completed.

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.

[23]:
```
import numpy as np
from timeit import default_timer as timer

#defining a function to solve a set of consistent linear Equations
def Linear_solver(A,b):
    A = np.array(A,dtype=float)
    b = np.array(b,dtype=float)
    size = len(b) - 1

    # Exception for wrong format of matrix passed
    if A.shape[0] != b.shape[0]:
        raise Exception("The number of rows in A and b must be the same.")

    if np.linalg.det(A) == 0:
            raise Exception("The matrix A is singular, and cannot be inverted.")

    #Pivoting the matrix to avoid zero errors
    for j in range(size):
        Max = j
        for k in range(j,size):
            if np.abs(A[k][j]) > np.abs(A[Max][j]) :
                Max = k
        A[[j, Max]] = A[[Max, j]]

        for i in range(size-j):
            fact = A[j][j]/A[i+j+1][j]
            A[i+j+1] = fact*A[i+j+1] - A[j]
            b[i+j+1] = fact*b[i+j+1] - b[j]

    #Finding the diagonal matrix
    for j in range(size,0,-1):
        for i in range(size):
            fact = A[j-i-1][j]/A[j][j]
            A[j-i-1] = A[j-i-1] - fact*A[j]
            b[j-i-1] = b[j-i-1] - fact*b[j]

    for i in range(size):
        b[i] = b[i]/A[i][i]
    return b

# Generating randon 10 X 10 matrices to test the solver
A = np.random.random_sample(size = (10,10))*100
b = np.random.random_sample(size = (10))*10

#printing and timing the function againt np.linalg.solve()
print(Linear_solver(A,b))
%timeit Linear_solver(A,b)
```

```
print(np.linalg.solve(A,b))
%timeit np.linalg.solve(A,b)
```

```
[-2.42284778e+01 -1.37911807e+01 -6.21988263e+00  8.31099362e+00
  1.15057068e+01  7.57463093e+00 -4.77432099e+00  1.58531366e+01
  1.41622549e+01  5.20172085e+06]
1.55 ms ± 58.4 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
[ 2.9686826   1.61032448  0.63256917 -1.00627507 -1.4185851  -0.91404035
  0.65840464 -1.76023232 -1.76045683  0.95132066]
9.82 µs ± 999 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

Major part of the cell is a function defentition to solve a set of linear equations - It starts by converting the input list into array for computational ease - Then it checks if the dimensions of the arrays are consistent for a solution,and also if the determinant of the matrix is zero. - Then the matrix is pivoted to avoid the zero coefficient errors and to give more consistent answers - Then the matrix is converted into a diagonal matrix - In the last step the final solution to the linear system is found out by eqating the diagonal terms in the A matrix and the terms of the b matrix

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.

The thrid problem is solved in another file uploade with this **netlist_solver.py**

1.1 Explanation for the third problem

- The program starts by defining class for all the components.
- It checks if the file has been passed to it throught the command line properly
- File is read along with registering the values of the start and end of the file as well as the ac check data. Errors int his part are also rooted out with a try except block
- All the data given in through the netlist file is now transfered to a list l for easy access, this is done simultaniously while addressing variety in the data recieved for each component
- Now the number of nodes is decided along with the information about them
- Penultimate step of the program is to make the MNA matrix to be solved, this is done by taking different scenarious for various components and depending on the kind of voltage they might encounter.

Note: The program must be run by using the following command in the terminal while ensuring that the python and netlist file are both in the current working directory

```
python netlist_solver.py ckt1.netlist
```