Untitled

G.Sai Krishna Reddy <ee21b049>

February 9, 2023

1 Factorial:

1.1 Iterative Algorithm:

[1]:
```
def fact(n):
  if n==0 or n==1:
     return 1
  f=n
  while n>1:
     f=f*(n-1)
     n=n-1
  return f
n=int(input('Enter a positive integer:'))
print(fact(n))
%timeit fact(n)
```

```
Enter a positive integer: 5

120
319 ns ± 6.74 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

In this code I used iteration to find the factorial of a number.

1.2 Recursive Algorithm

[2]:
```
def fact(n):
    if n==0:
     return 1
    else:
       return fact(n-1)*n
n=int(input('Enter a positive integer:'))
print(fact(n))
%timeit fact(n)
```

```
Enter a positive integer: 5

120
448 ns ± 15.6 ns per loop (mean ± std. dev. of 7 runs, 1,000,000 loops each)
```

In this code I used recursion to find the factorial of a number.

Here the second code takes more time than first one because second one uses recursion and every time it calls the function it has to implement from the first line of the code. So it takes more time than the first code.

2 Gaussian Elimination:

[5]:
```
import sys
def swap_rows(ag,x,y):
    for p in range(n+1):
        temp=ag[x][p]
        ag[x][p]=ag[y][p]
        ag[y][p]=temp
def linear_eq_solver(A, b):
    n = len(A)
    if n != len(b):
        return "Error: coefficient matrix and constant vector have different␣
↪number of rows."

    ag = []
    for i in range(n):
        ag.append(A[i] + [b[i]])
        #print(ag)

    for k in range(n):
        if ag[k][k] == 0:
           for t in range(0,n):
               if ag[t][k]!=0:
                   swap_rows(ag,k,t)
                   break
        d= ag[k][k]
        for j in range(k, n+1):
            ag[k][j]=ag[k][j]/d
        for i in range(n):
            if i == k:
                continue
            factor = ag[i][k]
            for j in range(k, n+1):
                ag[i][j]=ag[i][j]-factor * ag[k][j]
    #print(ag)
    for u in range(n):
        if ag[u][u]==0 and ag[u][n]==0:
            return "The given system of linear equations has infinite solutions.
↪"
        elif ag[u][u]==0 and ag[u][n]!=0:
            return "The given system of linear equations has no␣
↪solution(Inconsistent)."
    #print(ag)
    x = [0 for i in range(n)]
    for i in range(n):
        x[i] = ag[i][n]
    return x

eq = int(input("Number of equations:"))
var = int(input("Number of variables:"))
if eq!=var:
    print("Matrix A won't be a square matrix with the given inputs.please give␣
 ↪same number of equations and variables")
    sys.exit()
A = []
print("Enter the elements of the coefficient matrix:")
#since A is a square matrix, number of equations equals to number of rows and␣
 ↪coloumns
j=1
try:
    while eq>0:
        print(f"Enter the elements of row {j},separate each element with a␣
 ↪space:")
        row = list(map(complex, input().split()))
        A.append(row)
        eq=eq-1
        j=j+1
    b=[]
    print("Enter the elements of the constant vector, separate each element␣
 ↪with a space:")
    b = list(map(complex, input().split()))
except ValueError:
    print("please give valid input elements(a float number)")
    sys.exit()
n=len(A)
print(A,b)
print("Answer:")
print(linear_eq_solver(A, b))
%timeit linear_eq_solver(A, b)
```

```
Number of equations: 10
Number of variables: 10

Enter the elements of the coefficient matrix:
Enter the elements of row 1,separate each element with a space:

 1 2 3 4 5 6 7 8 9 10

Enter the elements of row 2,separate each element with a space:

 11 10 12 36 25 98 65 32 14 12

Enter the elements of row 3,separate each element with a space:
 2 21 3 36 39 56 42 58 51 53

Enter the elements of row 4,separate each element with a space:

 10 11 23 15 14 18 19 16 32 25

Enter the elements of row 5,separate each element with a space:

 74 75 96 58 42 16 35 25 69 96

Enter the elements of row 6,separate each element with a space:

 38 39 36 54 52 50 15 14 17 18

Enter the elements of row 7,separate each element with a space:

 1 2 6 55 42 32 95 47 49 2

Enter the elements of row 8,separate each element with a space:

 88 75 94 86 53 50 12 18 19 20

Enter the elements of row 9,separate each element with a space:

 15 75 82 83 81 10 12 19 17 57

Enter the elements of row 10,separate each element with a space:

 24 26 51 57 58 49 48 20 23 24

Enter the elements of the constant vector, separate each element with a space:

 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)]
Answer:
[(-12.186839273517311+0j), (-277.9486378443716+0j), (-99.55969289225702+0j),
(496.1019926987126+0j), (-231.0160736010484+0j), (19.479560717572546-0j),
(-101.230989325415+0j), (-186.88277737895635+0j), (33.49044217633475-0j),
(192.2823391266532-0j)]
85.7 µs ± 556 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Here I iterated through every diagonal elemnet and checked if it's zero.If it is zero then I swapped that row with a row which has non zero element at that position.After finding that non zero diagonal element, by some operations I made every element in that coloumn is zero except that diagonal element.After iterating through every diagonal element the final augmented matrix will look like identity matrix plu b matrix which is the solution matrix.And then I checked for the conditions like inconsistency and infinite solutions.

[6]:
```
import numpy as np
import sys
eq = int(input("Number of equations:"))
var = int(input("Number of variables:"))
if eq!=var:
    print("Matrix A won't be a square matrix with the given inputs.please give␣
 ↪same number of equations and variables")
    sys.exit()
A = []
print("Enter the elements of the coefficient matrix:")
#since A is a square matrix, number of equations equals to number of rows and␣
 ↪coloumns
j=1
try:
    while eq>0:
        print(f"Enter the elements of row {j},separate each element with a␣
 ↪space:")
        row = list(map(complex, input().split()))
        A.append(row)
        eq=eq-1
        j=j+1
    b=[]
    print("Enter the elements of the constant vector, separate each element␣
 ↪with a space:")
    b = list(map(complex, input().split()))
except ValueError:
    print("please give valid input elements(a float number)")
    sys.exit()
n=len(A)
print(A,b)
A1=np.array(A)
b1=np.array(b)
print("Answer:")
try:
    print(np.linalg.solve(A1, b1))
except LinAlgError:
    print("Please give a valid input")
%timeit np.linalg.solve(A1, b1)
```

```
Number of equations: 10
Number of variables: 10

Enter the elements of the coefficient matrix:
```

```
Enter the elements of row 1,separate each element with a space:

 1 2 3 4 5 6 7 8 9 10

Enter the elements of row 2,separate each element with a space:

 11 10 12 36 25 98 65 32 14 12

Enter the elements of row 3,separate each element with a space:

 2 21 3 36 39 56 42 58 51 53

Enter the elements of row 4,separate each element with a space:

 10 11 23 15 14 18 19 16 32 25

Enter the elements of row 5,separate each element with a space:

 74 75 96 58 42 16 35 25 69 96

Enter the elements of row 6,separate each element with a space:

 38 39 36 54 52 50 15 14 17 18

Enter the elements of row 7,separate each element with a space:

 1 2 6 55 42 32 95 47 49 2

Enter the elements of row 8,separate each element with a space:

 88 75 94 86 53 50 12 18 19 20

Enter the elements of row 9,separate each element with a space:

 15 75 82 83 81 10 12 19 17 57

Enter the elements of row 10,separate each element with a space:

 24 26 51 57 58 49 48 20 23 24

Enter the elements of the constant vector, separate each element with a space:

 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)]
Answer:
```

```
[ -12.18683927+0.j -277.94863784+0.j  -99.55969289-0.j  496.1019927 +0.j
 -231.0160736 +0.j   19.47956072+0.j -101.23098933+0.j -186.88277738-0.j
   33.49044218+0.j  192.28233913+0.j]
24.8 µs ± 659 ns per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

For small number of equations My function and linalg.solve are taking similar time to complete but for larger number of equations like 10*10, linalg.solve is about 3-4 times faster than mine.

3 Circuit Solver:

[11]:
```python
def NetlistConvert(file_path):
    with open(file_path, 'r') as file:
            netlist = file.readlines()
    net =[]
    t = 0
    for line in netlist:
        if line[0:2] == ".a": t = 1
        if line[0:2] == ".e": t = 0
        if t == 1:
             linesplit = line.split("#")[0].split('\n')[0].split('  ')
             net.append(linesplit[0])
        if line[0:2] == ".c": t = 1
    return net
net = NetlistConvert('ckt1.netlist')
print(net)
def MatrixSizeInc(MNA, b):
    l = MNA.shape[0]
    MNA_temp = np.zeros((l+1, l+1))
    b_temp = np.zeros((l+1,1))
    MNA_temp[:l, :l] = MNA
    b_temp[:l, :] = b
    return MNA_temp, b_temp
def addRes(MNAdc,value,i,j):
    MNAdc[i][i] += 1/float(value)
    MNAdc[j][j] += 1/float(value)
    MNAdc[i][j] -= 1/float(value)
    MNAdc[j][i] -= 1/float(value)
    return MNAdc
def create_MNA_matrix(netlist):
    nodes = set()
    components = []
    v_type = set()
    k = 0
    t = complex(0,1)
    freq = 0
    if netlist[-1].startswith(".ac"):
      split = netlist[-1].split()
    freq = float(split[2])*2*math.pi
  for line in netlist:
    split_line = line.split()
    print(split_line)
    if len(split_line) == 3:
      continue
    component_type = split_line[0]
    nodes.update([split_line[1], split_line[2]])
    try:
      components.append((component_type, split_line[1],␣
↪split_line[2],split_line[3],split_line[4]))
    except IndexError:
      try:
        components.append((component_type, split_line[1],␣
↪split_line[2],split_line[3]))
      except IndexError:
        components.append((component_type, split_line[1],␣
↪split_line[2],split_line[3],split_line[4],split_line[5]))
  node_dict = {node: i for i, node in enumerate(nodes)}
  num_nodes = len(nodes)
  num_components = len(components)
  MNA = np.zeros((num_nodes, num_nodes))
  b = np.zeros((num_nodes,1))
  if freq !=0:
      MNA = np.zeros((num_nodes, num_nodes))*t
      b = np.zeros((num_nodes,1))*t
  for component in components:
      try:
       component_type, node1, node2, acdc ,value = component
      except ValueError:
           try:
            component_type, node1, node2,value = component
           except ValueError:
            component_type, node1, node2, acdc ,value, phase = component
      i = node_dict[node1]
      j = node_dict[node2]
      k = node_dict["GND"]
      if component_type[0] == 'R':
          MNAdc = addRes(MNA,value,i,j)
      elif component_type[0] == 'I':
          v_type.update([acdc])
          b[i] -= float(value)
          b[j] += float(value)

      elif component_type[0] == 'V':
          v_type.update([acdc])
          if acdc.startswith("dc"):
                MNA,b = MatrixSizeInc(MNA,b)
                l = MNA.shape[0]-1
                MNA[l][i] += 1
                MNA[l][j] -= 1
                MNA[i][l] += 1
                MNA[j][l] -= 1
                b[l] -= float(value)

        elif component_type[0] == 'C':
            if freq == 0:
             print("This function cannot compute DC circuit with a capacitance")
            return None
            MNA[i][i] += t*float(value)*freq
            MNA[j][j] += t*float(value)*freq
            MNA[i][j] -= t*float(value)*freq
            MNA[j][i] -= t*float(value)*freq
        elif component_type[0] == 'L':
            if freq == 0:
             print("This function cannot compute DC circuit with a capacitance")
            return None
            MNA[i][i] += 1/t*float(value)*freq
            MNA[j][j] += 1/t*float(value)*freq
            MNA[i][j] -= 1/t*float(value)*freq
            MNA[j][i] -= 1/t*float(value)*freq
    b = np.squeeze(b)
    MNA = np.delete(MNA, k, axis=0)
    MNA = np.delete(MNA, k, axis=1)
    b = np.delete(b, k)
    v_type = list(v_type)
    if len(v_type) == 1 and v_type[0] == 'dc':
        return MNA, b, node_dict,0
    elif len(v_type) == 1 and v_type[0] == 'ac':
        return MNA, b, node_dict,1
    else:
        return("The code doesn't work for DC+AC supply")
net = NetlistConvert('ckt1.netlist')
try:
    V,b,nodes,check = create_MNA_matrix(net)
    V = V.tolist()
    b = b.tolist()
    a = linear_eq_solver(V,b)
    if check == 0:
      print(f"DC :{a}\nNodes:{nodes}")
    if check == 1:
      print(f"AC :{a}\nNodes:{nodes}")
except ValueError:
    a = create_MNA_matrix(net)
```

```
pass
```

```
['R1 GND 1 1e3', 'R2 1 2 4e3', 'R3 2 GND 20e3', 'R4 2 3 8e3', 'R5 GND 4 10e3',
'V1 GND 4 dc 5']
['R1', 'GND', '1', '1e3']
['R2', '1', '2', '4e3']
['R3', '2', 'GND', '20e3']
['R4', '2', '3', '8e3']
['R5', 'GND', '4', '10e3']
['V1', 'GND', '4', 'dc', '5']
DC :[5.0, 0.0, 0.0, 0.0, 0.0005]
Nodes:{'4': 0, '1': 1, '2': 2, 'GND': 3, '3': 4}
```

The above code first reads the inputs from a netlist, then splits the lines to find components of the circuit.After finding the components, I tried to find if the given circuit contains AC voltage or DC voltage(if its AC the voltage line contains more elements)and I seperated voltages and current sources with resistors,capacitors and inductors.Then I created two matrices(MNA and b),MNA matrix contains components like Resistors or Capacitors or Inductors and b matrix contains the variable like node voltages and currents.Then I gave these two matrices as inputs to the linear_eq_solver funtion which gives us the values of node voltages and currents.The above does not work if there are sources with different frequencies or if a circuit contains both AC and DC.