Assn2_Parth_Dethaliya_24-27-29

October 5, 2024

[1]:
```
# Name: Parth Jilubhai Dethaliya
# Reg No: 24-27-29
# Programme: M.Tech. Data Science
# Assignment Number: 2
```

[2]:
```
import numpy as np
def print_var_info(var):
    print(f"values    : \n{var}")
    print(f"Shape     : {var.shape}")
    print(f"Dimension : {var.ndim}")
```

[3]:
```
## Q1
# a
var1 = np.arange(31) # including 0 and 30 as well, total 31 values

print_var_info(var1)
```

```
values    :
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
Shape     : (31,)
Dimension : 1
```

[4]:
```
# b
# Since 31 values are present in var1 (5,6) or (6,5) shapes may not work so,
var2 = var1.reshape(31,1)

print_var_info(var2)
```

```
values    :
[[ 0]
 [ 1]
 [ 2]
 [ 3]
 [ 4]
 [ 5]
 [ 6]
 [ 7]
 [ 8]
 [ 9]
 [10]
 [11]
 [12]
 [13]
 [14]
 [15]
 [16]
 [17]
 [18]
 [19]
 [20]
 [21]
 [22]
 [23]
 [24]
 [25]
 [26]
 [27]
 [28]
 [29]
 [30]]
Shape     : (31, 1)
Dimension : 2
```

[5]:
```
# c
# Similarly 3d shape
var3 = var2.reshape(1,31,1) # or var1.reshape() both would result in same

print_var_info(var3)
```

```
values    :
[[[ 0]
  [ 1]
  [ 2]
  [ 3]
  [ 4]
  [ 5]
  [ 6]
  [ 7]
  [ 8]
  [ 9]
  [10]
  [11]
  [12]
  [13]
  [14]
  [15]
  [16]
  [17]
  [18]
  [19]
  [20]
  [21]
  [22]
  [23]
  [24]
  [25]
  [26]
  [27]
  [28]
  [29]
  [30]]]
Shape     : (1, 31, 1)
Dimension : 3
```

[6]:

```
# d

var2[1,0] = -1

print(var1)
print(var3)
```

```
[ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
[[[ 0]
  [-1]
  [ 2]
  [ 3]
  [ 4]
  [ 5]
  [ 6]
  [ 7]
  [ 8]
  [ 9]
  [10]
  [11]
  [12]
  [13]
  [14]
  [15]
  [16]
  [17]
  [18]
  [19]
  [20]
  [21]
  [22]
  [23]
  [24]
  [25]
  [26]
  [27]
  [28]
  [29]
  [30]]]
```

The values in var1 as well as var3 changes because reshapes doesn't returns copy rather it returns view (in most of the cases).

[7]:
```
# e i) Sum var3 over its second dimension
print(f"Shape      : {var3.shape}")

sum_var3_ax1 = np.sum(var3, axis=1)
print("Sum over second dimension:", sum_var3_ax1) # Second dimension contains␣
 ↪all the 31 values so it sums over it

# e ii) Sum var3 over its 3rd dimension

sum_var3_ax2 = np.sum(var3, axis=2)
print("Sum over second dimension:", sum_var3_ax2) # it will sum between columns␣
 ↪but since we only have 31 rows and
# single column so it will be same thing but with dimension 2 (as it has␣
 ↪reduced that 2nd axis by summin over it)

# e iii) Sum var3 over its 1st & 3rd dimension



sum_var3_ax1_3 = np.sum(var3, axis=(0,2))
print("Sum over second dimension:", sum_var3_ax1_3) # it willfirst sum over␣
 ↪axis 0 which is between the depth, but since
# there exists only one depth so the output information may not change, only␣
 ↪axis 0 will be reduces
# then it sums over axis 2, again same thing will happen as mention in "e (ii)"␣
 ↪and the dimension will be 1 now
```

```
Shape      : (1, 31, 1)
Sum over second dimension: [[463]]
Sum over second dimension: [[ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19 20 21 22 23
  24 25 26 27 28 29 30]]
Sum over second dimension: [ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
17 18 19 20 21 22 23
 24 25 26 27 28 29 30]
```

Conclusion regarding output shape: if an array is of shape (a,b,c), any_operation(axis=0) returns (b,c) (eliminates the given axis) example: i) any_operation(axis=1) returns (a,c) ii) any_operation(axis=(0,1) returns (b) likewise...

[8]:
```
# f i)
print("Second row")
print(var2[2,:])

# f ii)
print("\nLast column")
print(var2[:,-1]) # Only 1 column exists so the output will with full data but␣
  ↪1d

# f iii) Top right 2x2
# Since it's not possible with shape (31,1) so creating new variable with (5,6)␣
  ↪shape to do this
var4 = np.arange(30).reshape(5,6)
print("\nvar4: ")
print_var_info(var4)

print("\n")
print("Top right 2x2: \n")

print(var4[:4,-4:])
```

```
Second row
[2]

Last column
[ 0 -1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30]

var4:
values    :
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]]
Shape     : (5, 6)
Dimension : 2


Top right 2x2:

[[ 2  3  4  5]
 [ 8  9 10 11]
 [14 15 16 17]
 [20 21 22 23]]
```

[9]:
```
# Q2
# a
arr = np.arange(10)
print("original array  : ", arr)
arr2 = arr + 1
print("Broadcasted + 1 : ",arr2)
```

```
original array  :  [0 1 2 3 4 5 6 7 8 9]
Broadcasted + 1 :  [ 1  2  3  4  5  6  7  8  9 10]
```

[10]:
```
# b 10x10 matrix
column_vector = arr2.reshape(10,1)
arr3 = column_vector + arr2
```

[11]:
```
arr3
```

[11]:
```
array([[ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11],
       [ 3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
       [ 4,  5,  6,  7,  8,  9, 10, 11, 12, 13],
       [ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14],
       [ 6,  7,  8,  9, 10, 11, 12, 13, 14, 15],
       [ 7,  8,  9, 10, 11, 12, 13, 14, 15, 16],
       [ 8,  9, 10, 11, 12, 13, 14, 15, 16, 17],
       [ 9, 10, 11, 12, 13, 14, 15, 16, 17, 18],
       [10, 11, 12, 13, 14, 15, 16, 17, 18, 19],
       [11, 12, 13, 14, 15, 16, 17, 18, 19, 20]])
```

[12]:
```
# c

import numpy.random as npr
data = np.exp(npr.randn ( 50 , 5 ) )
print(data)
```

```
[[ 0.38235034  0.47052197  0.53214629  0.40205941  2.91523004]
 [ 2.09759357  6.7715061   1.87417993  2.23864297  1.32071546]
 [ 0.87450436  0.32739146  1.67193682  1.66378008  0.62403148]
 [ 0.24493919  0.70024222  0.58580026  0.85574078  1.92754272]
 [ 0.21456058  0.60270527  1.0972436   0.79434142  1.4114013 ]
 [ 0.48448822  0.72102741  0.87042439  0.57519546  0.86214031]
 [ 1.7380417   1.36251009  3.05740094  7.82942831  1.01136941]
 [ 1.09660995  0.82763883  3.91180426  0.89560132  1.19201264]
 [ 0.46463446  0.75688305  2.30421174  1.37665969  0.57275142]
 [ 0.17485048  1.049277    0.27547992  1.1294214   1.25206785]
 [ 0.35353941  1.10328982  0.68705863  0.4821      4.88034987]
 [ 1.27380803  0.40903791  0.30665223  2.33216675  0.61671796]
 [ 2.46778942  0.86358348  0.2519135   0.49389942  0.47557763]
 [ 0.59847432  1.56789963  0.61186137  0.46275371  3.87413232]
 [ 2.43903977  5.22751147  1.06642947  0.33476891  0.67491127]
 [ 0.06963168  1.26421882  1.1115445   0.06367931  2.01671067]
 [ 9.02691775  0.58573989  1.1673564   5.89537922  1.54407345]
 [ 5.04280364  1.08837524  1.15437486  0.72243696  0.40697481]
 [ 0.84990804  0.31230545  1.92914941  0.98035944  2.30270128]
 [ 1.68134421  1.61808019  1.22204211  0.59249168  0.86292201]
 [ 1.10458064  1.60106534  5.73051277  4.40731247  2.21583517]
 [ 0.48413142  0.84176457  0.8149363   1.80916436  0.34600285]
 [ 0.35747069  0.7483486   0.7001333   4.80071015  0.25918724]
 [ 0.27706548  0.61372494  0.16333313  1.48792037  0.41081579]
 [ 1.3685866   4.05336671  1.23184701  0.23848722  3.42932578]
 [ 0.45740037  0.89822703  0.25182078  2.30581421  0.34269687]
 [ 0.46983396  8.21617173  4.55730463  0.19743147  0.86416487]
 [ 0.21656583  1.41821823  0.90924882  2.44278065  0.95401947]
 [12.98801133  1.66756708  1.02498594  0.62199627  0.12466774]
 [ 0.23352828  1.49689326  1.21756798 13.63536932  0.23006328]
 [ 9.65718561  0.13178202  1.60640208  2.70827242  0.14678533]
 [ 1.47256576  0.27787278  0.15183198  8.08129085  1.40715872]
 [ 4.64508427  1.42864191  0.19686782  0.16315877  0.21599943]
 [ 1.11124272  2.27035587  2.60345896  1.03521854  0.59453139]
 [ 0.69408769 10.44493568  0.27796471  1.55845384  0.7779147 ]
 [ 1.34840899  0.73887632  0.98823567  1.00099782  1.7892939 ]
 [ 1.39881014  0.54458281  0.38039964  1.15377251  1.09535485]
 [ 2.59785481  1.12508868  0.15934267  0.72477957  0.15854823]
 [ 0.34142663  1.33605563  0.82568839  0.15709303  0.32715635]
 [ 0.35862749 14.02454596  1.66611477  0.65457015  1.2356376 ]
 [ 0.54583687  4.603758    0.65128755  0.59922415  3.24879359]
 [ 4.1059682   1.10274585  0.88537284  0.27981202  2.01558802]
 [ 0.43738869  4.38361422  3.39925361  0.71041841  4.06540601]
 [ 0.5176052   1.06473146  0.82508289  1.69731185  1.95034458]
 [ 0.6003816   3.33345797  0.37293915  1.803347    0.26232508]
 [ 0.20605182  0.30892283  0.19303384  2.04201804  0.43045054]
 [ 0.49921889  1.19922506  0.98001505  0.1823939   0.99863417]
 [ 1.60878768  2.37476806  0.32081522  1.13670549  0.35315203]
 [ 2.96776773  0.29138487  4.40887413  1.85732645  1.08200108]
 [ 0.61845036  0.20797488  0.59749474  0.75161403  0.77548499]]
```

[13]:
```
std = np.std(data,axis=0)# for each column that means we have to compute␣
↪between rows
mean = np.mean(data,axis=0)
print(std)
print(mean)
```

```
[2.53622086 2.69123172 1.24231533 2.4336724  1.10454843]
[1.7053151  2.00756827 1.27562354 1.80731343 1.25703347]
```

[14]:
```
normalized = (data-mean)/std
print(normalized)
```

```
[[-5.21628371e-01 -5.71131163e-01 -5.98460980e-01 -5.77421193e-01
   1.50124388e+00]
 [ 1.54670470e-01  1.77017006e+00  4.81807135e-01  1.77234017e-01
   5.76543211e-02]
 [-3.27578227e-01 -6.24315178e-01  3.19011821e-01 -5.89780885e-02
  -5.73086685e-01]
 [-5.75807860e-01 -4.85772383e-01 -5.55272289e-01 -3.91002769e-01
   6.07043779e-01]
 [-5.87785764e-01 -5.22014878e-01 -1.43586681e-01 -4.16231870e-01
   1.39756510e-01]
 [-4.81356689e-01 -4.78049085e-01 -3.26164493e-01 -5.06279307e-01
  -3.57515479e-01]
 [ 1.29036863e-02 -2.39688828e-01  1.43423924e+00  2.47449693e+00
  -2.22411307e-01]
 [-2.40004788e-01 -4.38434724e-01  2.12199001e+00 -3.74624010e-01
  -5.88664352e-02]
 [-4.89184778e-01 -4.64725952e-01  8.27960638e-01 -1.76956331e-01
  -6.19512949e-01]
 [-6.03442956e-01 -3.56079065e-01 -8.05064218e-01 -2.78546953e-01
  -4.49561139e-03]
 [-5.32988160e-01 -3.36009139e-01 -4.73764503e-01 -5.44532382e-01
   3.28035993e+00]
 [-1.70137811e-01 -5.93977228e-01 -7.79972108e-01  2.15663095e-01
  -5.79707956e-01]
 [ 3.00634038e-01 -4.25078517e-01 -8.24033971e-01 -5.39683983e-01
  -7.07488978e-01]
 [-4.36413402e-01 -1.63370787e-01 -5.34294433e-01 -5.52481805e-01
   2.36938353e+00]
 [ 2.89298415e-01  1.19645706e+00 -1.68390472e-01 -6.05070973e-01
  -5.27022797e-01]
 [-6.44929405e-01 -2.76211612e-01 -1.32075198e-01 -7.16462136e-01
   6.87771746e-01]
 [ 2.88681587e+00 -5.28318827e-01 -8.71494807e-02  1.67979297e+00
   2.59870891e-01]
 [ 1.31592977e+00 -3.41551056e-01 -9.75989600e-02 -4.45777530e-01
  -7.69598362e-01]
 [-3.37276250e-01 -6.29920792e-01  5.26054740e-01 -3.39796759e-01
   9.46692587e-01]
 [-9.45142031e-03 -1.44724840e-01 -4.31302997e-02 -4.99172261e-01
  -3.56807774e-01]
 [-2.36862046e-01 -1.51047170e-01  3.58595689e+00  1.06834389e+00
   8.68048589e-01]
 [-4.81497369e-01 -4.33185926e-01 -3.70829552e-01  7.60548254e-04
  -8.24799165e-01]
 [-5.31438104e-01 -4.67897158e-01 -4.63240067e-01  1.22999164e+00
  -9.03397447e-01]
 [-5.63140867e-01 -5.17920222e-01 -8.95336615e-01 -1.31239137e-01
  -7.66120941e-01]
```

```
[-1.32767812e-01  7.60171792e-01 -3.52378563e-02 -6.44633274e-01
  1.96667910e+00]
[-4.92037089e-01 -4.12205770e-01 -8.24108611e-01  2.04834792e-01
 -8.27792222e-01]
[-4.87134681e-01  2.30697468e+00  2.64158463e+00 -6.61503151e-01
 -3.55682551e-01]
[-5.86995117e-01 -2.18988963e-01 -2.94912818e-01  2.61114529e-01
 -2.74332927e-01]
[ 4.44862527e+00 -1.26336649e-01 -2.01750388e-01 -4.87048774e-01
 -1.02518432e+00]
[-5.80307039e-01 -1.89755127e-01 -4.67317396e-02  4.86016766e+00
 -9.29764751e-01]
[ 3.13532257e+00 -6.96999161e-01  2.66259722e-01  3.70205534e-01
 -1.00516022e+00]
[-9.17701385e-02 -6.42715183e-01 -9.04594452e-01  2.57798766e+00
  1.35915500e-01]
[ 1.15911402e+00 -2.15115764e-01 -8.68342918e-01 -6.75585860e-01
 -9.42497423e-01]
[-2.34235269e-01  9.76458441e-02  1.06883928e+00 -3.17255062e-01
 -5.99794512e-01]
[-3.98714254e-01  3.13513226e+00 -8.03064088e-01 -1.02256817e-01
 -4.33768912e-01]
[-1.40723591e-01 -4.71416838e-01 -2.31332464e-01 -3.31316414e-01
  4.81880572e-01]
[-1.20851050e-01 -5.43611851e-01 -7.20609236e-01 -2.68541043e-01
 -1.46375309e-01]
[ 3.51917189e-01 -3.27909181e-01 -8.98548735e-01 -4.44814949e-01
 -9.94510716e-01]
[-5.37764075e-01 -2.49518700e-01 -3.62174674e-01 -6.78078283e-01
 -8.41861791e-01]
[-5.30981991e-01  4.46523336e+00  3.14325375e-01 -4.73664115e-01
 -1.93706973e-02]
[-4.57167687e-01  9.64684574e-01 -5.02558384e-01 -4.96405875e-01
  1.80323476e+00]
[ 9.46547337e-01 -3.36211266e-01 -3.14131757e-01 -6.27652847e-01
  6.86755356e-01]
[-4.99927442e-01  8.82884194e-01  1.70941307e+00 -4.50715974e-01
  2.54255266e+00]
[-4.68299078e-01 -3.50336539e-01 -3.62662072e-01 -4.51998328e-02
  6.27687383e-01]
[-4.35661387e-01  4.92670210e-01 -7.26614546e-01 -1.62981265e-03
 -9.00556612e-01]
[-5.91140659e-01 -6.31177697e-01 -8.71429077e-01  9.64405116e-02
 -7.48344678e-01]
[-4.75548572e-01 -3.00361804e-01 -2.37949646e-01 -6.67682112e-01
 -2.33941125e-01]
[-3.80595469e-02  1.36443021e-01 -7.68571629e-01 -2.75553909e-01
 -8.18326680e-01]
[ 4.97769202e-01 -6.37694402e-01  2.52210570e+00  2.05504329e-02
 -1.58465113e-01]
[-4.28537099e-01 -6.68687642e-01 -5.45858837e-01 -4.33788626e-01
 -4.35968644e-01]]
```

[15]:
```python
std = np.std(normalized,axis=0)
mean = np.mean(normalized,axis=0)
print("Standard Deviation : ",std)
print("Mean               : ",mean)
# Mean 0, Std = 1
```

```
Standard Deviation :  [1. 1. 1. 1. 1.]
Mean               :  [ 1.23234756e-16 -1.06581410e-16 -6.66133815e-18
-4.55191440e-17
  3.36397576e-16]
```

[16]:
```python
# 3 Vandermonde matrix

N = 12
def vandermonde(N):
    vec = np.arange (N) +1
    vector = np.arange(N) + 1
    v2 = np.arange(N)  # Power Vector
    output = vector.reshape(N,1)**v2
    return output
def printMat(Mat):
    for row in Mat:
        print(" ".join(f"{elem}" for elem in row))

Vector = vandermonde(N)
printMat(Vector)
```

```
1 1 1 1 1 1 1 1 1 1 1 1
1 2 4 8 16 32 64 128 256 512 1024 2048
1 3 9 27 81 243 729 2187 6561 19683 59049 177147
1 4 16 64 256 1024 4096 16384 65536 262144 1048576 4194304
1 5 25 125 625 3125 15625 78125 390625 1953125 9765625 48828125
1 6 36 216 1296 7776 46656 279936 1679616 10077696 60466176 362797056
1 7 49 343 2401 16807 117649 823543 5764801 40353607 282475249 1977326743
1 8 64 512 4096 32768 262144 2097152 16777216 134217728 1073741824 0
1 9 81 729 6561 59049 531441 4782969 43046721 387420489 -808182895 1316288537
1 10 100 1000 10000 100000 1000000 10000000 100000000 1000000000 1410065408
1215752192
1 11 121 1331 14641 161051 1771561 19487171 214358881 -1937019605 167620825
1843829075
1 12 144 1728 20736 248832 2985984 35831808 429981696 864813056 1787822080
-20971520
```

[17]:
```
# b

x = np.ones(12)

b = np.dot(Vector,x)
print(b)
```

```
[1.20000000e+01 4.09500000e+03 2.65720000e+05 5.59240500e+06
 6.10351560e+07 4.35356467e+08 2.30688120e+09 1.22713351e+09
 9.43953692e+08 3.73692871e+09 3.10225064e+08 3.10073456e+09]
```

[18]:
```
# c naive solution

Vector_inv = np.linalg.inv(Vector)
Sol = np.dot(Vector_inv,b)
print(Sol)
# The result is almost ones(12) with slight variation maybe due to floating␣
 ↪points instability while inversing Vector
```

```
[1.00158882 0.99722672 1.00127029 0.99991608 1.00000095 0.99999905
 1.00000018 0.99999999 1.         1.         1.         1.        ]
```

[19]:
```
# d solve using numpy

Sol_inbuilt = np.linalg.solve(Vector,b)

print(Sol_inbuilt)
```

```
[1.0000114  0.99997033 1.00002961 0.99998507 1.00000423 0.9999993
 1.00000007 1.         1.         1.         1.         1.        ]
```

[20]:
```
# The solution using .solve() seems more accurate but let's verify that using␣
 ↪some statistics

Diff_solve = x - Sol
Diff_solve_inbuilt = x - Sol_inbuilt
print(np.std(Diff_solve) , np.std(Diff_solve_inbuilt) )
# clearly the inbuilt function method's solution is more closer to ones(12)
```

```
0.0009931496457600455 1.3316958977450673e-05
```

[21]:
```
#https://github.com/PARTH1D/Parth_24-27-29/blob/main/
 ↪Assn2_Parth_Dethaliya_24-27-29.ipynb
```

[ ]:
```

```