Assn2_Parth_Dethaliya_24-27-29

October 8, 2024

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

[2]:
```python
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]:
```python
## 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]:
```python
# 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]:
```python
# 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]:
```python
# 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]:
```python
# 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 third 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 first,third 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 third 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 first,third 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[1,:])

# 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[:2,-2:])
```

```
Second row
[-1]

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:

[[ 4  5]
 [10 11]]
```

[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)
```

```
[[ 0.13229182  0.39721069  8.3678117   0.88091519  0.78534795]
 [ 1.1196919   1.47960031  1.69209521  0.25459477  0.5030388 ]
 [ 2.4487117   2.33291236  0.60274116  3.25405418  8.15856902]
 [ 1.04600821  3.45910292  1.20869025  0.52069193  1.95885277]
 [ 1.43256151  0.60955491  2.47809694  1.99690595  0.49137185]
 [ 0.53065645  2.73139153  1.11948245  0.4632505   4.51046351]
 [ 0.91598094  0.39871461  0.44057713  0.90685407  1.25208093]
 [ 5.6727857   2.24761558  0.5714233   0.15712224  2.42640908]
 [ 8.06325256  1.15928064  0.31349123  1.88657776  1.2082491 ]
 [ 1.55886364  1.29158243  1.01839507  0.23227627  1.27863051]
 [ 0.49530623  0.29778728  0.52356885  0.42079815  7.99480334]
 [ 0.48306008  0.74142276  0.98438274  0.51874428  2.32025173]
 [ 0.98226191  2.58905789  0.66329924  0.11886209  0.56154774]
 [ 2.74573058  0.86371696  0.91360364  0.18878987  1.10784177]
 [ 2.8059662   0.36201195  1.41655453  0.11922446  4.38202385]
```

```
 [ 0.85204655  2.12333447  0.28906948  0.7529258   1.29092123]
 [ 0.29265948  0.7753107   0.16264586  1.67239303  0.13733294]
 [ 0.36638882  0.69623991  0.24693859  0.55146543  7.78035216]
 [ 0.22266936  1.02379112 11.96526992  2.44759177  2.3850377 ]
 [ 8.35267772  1.16655125  0.62134689  6.19670375  4.04417533]
 [27.44357743  0.31247982  0.81454733  0.15266395  1.31835454]
 [ 0.84686158  0.55745581  2.83680902  0.74399516  0.66352456]
 [ 0.44012025  2.24633017  0.7029408   0.22626505  0.58068104]
 [ 1.01884628  0.33523397  0.62453713  1.84481719  1.86398793]
 [ 0.41026182  2.50439289  0.24908101  0.95095174  2.53443318]
 [ 0.58272083  0.5895959   0.70301295  0.38228042  0.90553543]
 [ 3.34962561  5.7408649   0.5831893   0.33659583  1.89905087]
 [ 2.93818512  0.33643166  0.89982054  0.64358729  0.30207579]
 [ 1.05546278  2.03189243  0.74591055  1.25600447  0.56667275]
 [ 2.33202607  0.1729393   5.49859471  9.01744217  2.54580476]
 [ 1.10312158  1.18132481  1.52844569  1.18225154  4.89123294]
 [ 0.52451822  1.45098809  0.9188544   0.84564303  0.38161375]
 [ 0.4155313   0.21977817  0.65223202  0.4130988   6.51447277]
 [ 0.87154053  2.44160611  0.85575953  2.20387769  3.40729262]
 [ 0.14688363  0.56069502  0.77558375 10.82876143  1.30979827]
 [ 0.40901795  0.33704482  0.31477298  2.51190632  1.02359118]
 [ 1.38988314  2.27221406  1.07926439  1.02963811  2.86029646]
 [ 0.92445086  0.65752098  0.88948291  0.70300029  0.04598369]
 [ 0.70950143  0.65942511  0.20892291  1.6108211   0.12547167]
 [ 0.51210691  2.76010361  0.2756458   0.75028278  0.22903389]
 [ 0.75180298  1.30699762  2.45793774  2.72082362  1.33900009]
 [ 2.25838547  0.34063229  0.18487967  0.20314186  2.13930735]
 [ 1.41296851  0.09692474  2.09958656  0.34618667  0.35905557]
 [ 0.56638609  3.69980926  7.54261557  2.67675264  0.34610111]
 [ 1.09382417  1.20078371  0.59713914  0.74868553  8.05346451]
 [ 3.64050935  0.59130831  0.34719952  0.47317292  2.26084248]
 [ 0.55844141  0.62052054  0.06156567  2.88865532  1.62031351]
 [ 2.87954124  0.55451959  0.56316166  2.31115225  1.20441798]
 [ 1.30395797  4.57793524  2.27537411  0.84132928  0.83165199]
 [ 0.1489721   2.06007317  0.18535706  2.46888701  0.75611431]]
```

[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)
```

```
[4.02330743 1.20473127 2.22624715 2.05294962 2.19348308]
[2.05117208 1.38328025 1.46143417 1.53706826 2.14912957]
```

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

```
[[-4.76940996e-01 -8.18497522e-01  3.10225104e+00 -3.19614794e-01
```

```
 -6.21742485e-01]
[-2.31521005e-01  7.99514880e-02  1.03609807e-01 -6.24697983e-01
 -7.50446075e-01]
[ 9.88091589e-02  7.88252235e-01 -3.85713243e-01  8.36350736e-01
  2.73967897e+00]
[-2.49835214e-01  1.72305868e+00 -1.13529141e-01 -4.95080989e-01
 -8.67464157e-02]
[-1.53756723e-01 -6.42238949e-01  4.56671116e-01  2.23988787e-01
 -7.55764992e-01]
[-3.77926782e-01  1.11901411e+00 -1.53600073e-01 -5.23060940e-01
  1.07652253e+00]
[-2.82153716e-01 -8.17249177e-01 -4.58555128e-01 -3.06979859e-01
 -4.08960821e-01]
[ 9.00158310e-01  7.17450736e-01 -3.99780805e-01 -6.72177247e-01
  1.26410603e-01]
[ 1.49431297e+00 -1.85933254e-01 -5.15640389e-01  1.70247483e-01
 -4.28943574e-01]
[-1.22364111e-01 -7.61147511e-02 -1.99007150e-01 -6.35569416e-01
 -3.96856975e-01]
[-3.86713140e-01 -9.01024981e-01 -4.21276373e-01 -5.43739653e-01
  2.66501886e+00]
[-3.89756941e-01 -5.32780634e-01 -2.14285028e-01 -4.96029699e-01
  7.80138971e-02]
[-2.65679466e-01  1.00086856e+00 -3.58511377e-01 -6.90813916e-01
 -7.23772088e-01]
[ 1.72633713e-01 -4.31269031e-01 -2.46078038e-01 -6.56751816e-01
 -4.74718863e-01]
[ 1.87605380e-01 -8.47714608e-01 -2.01593243e-02 -6.90637408e-01
  1.01796741e+00]
[-2.98044717e-01  6.14289880e-01 -5.26610308e-01 -3.81958940e-01
 -3.91253683e-01]
[-4.37081338e-01 -5.04651590e-01 -5.83398080e-01  6.59172409e-02
 -9.17169887e-01]
[-4.18755784e-01 -5.70285141e-01 -5.45534928e-01 -4.80091096e-01
  2.56725144e+00]
[-4.54477503e-01 -2.98397770e-01  4.71818044e+00  4.43519658e-01
  1.07549556e-01]
[ 1.56625009e+00 -1.79898211e-01 -3.77355803e-01  2.26972715e+00
  8.63943646e-01]
[ 6.31132613e+00 -8.88829279e-01 -2.90572790e-01 -6.74348898e-01
 -3.78746950e-01]
[-2.99333451e-01 -6.85484357e-01  6.17799713e-01 -3.86309090e-01
 -6.77281272e-01]
[-4.00429711e-01  7.16383762e-01 -3.40704926e-01 -6.38497504e-01
 -7.15049295e-01]
[-2.56586358e-01 -8.69941958e-01 -3.75922794e-01  1.49905742e-01
 -1.29994909e-01]
[-4.07851074e-01  9.30591473e-01 -5.44572584e-01 -2.85499710e-01
```

```
  1.75658350e-01]
[-3.64986090e-01 -6.58806133e-01 -3.40672521e-01 -5.62501793e-01
 -5.66949502e-01]
[ 3.22732865e-01  3.61705948e+00 -3.94495675e-01 -5.84754941e-01
 -1.14009859e-01]
[ 2.20468621e-01 -8.68947805e-01 -2.52269221e-01 -4.35218165e-01
 -8.42064293e-01]
[-2.47485263e-01  5.38387443e-01 -3.21403501e-01 -1.36907301e-01
 -7.21435618e-01]
[ 6.98067441e-02 -1.00465638e+00  1.81343771e+00  3.64372016e+00
  1.80842605e-01]
[-2.35639587e-01 -1.67635259e-01  3.01006645e-02 -1.72832650e-01
  1.25011376e+00]
[-3.79452450e-01  5.62016180e-02 -2.43719467e-01 -3.36796003e-01
 -8.05803263e-01]
[-4.06541335e-01 -9.65777274e-01 -3.63482623e-01 -5.47490033e-01
  1.99014219e+00]
[-2.93199456e-01  8.78474633e-01 -2.72060828e-01  3.24805552e-01
  5.73591411e-01]
[-4.73314177e-01 -6.82795612e-01 -3.08074698e-01  4.52602104e+00
 -3.82647718e-01]
[-4.08160241e-01 -8.68438844e-01 -5.15064643e-01  4.74847532e-01
 -5.13128367e-01]
[-1.64364506e-01  7.37868963e-01 -1.71665477e-01 -2.47171262e-01
  3.24218091e-01]
[-2.80048502e-01 -6.02424199e-01 -2.56912743e-01 -4.06277856e-01
 -9.58815639e-01]
[-3.33474554e-01 -6.00843654e-01 -5.62611056e-01  3.59253046e-02
 -9.22577390e-01]
[-3.82537303e-01  1.14284688e+00 -5.32640042e-01 -3.83246367e-01
 -8.75363800e-01]
[-3.22960431e-01 -6.33192100e-02  4.47615878e-01  5.76611987e-01
 -3.69334728e-01]
[ 5.15032465e-02 -8.65461028e-01 -5.73410956e-01 -6.49760902e-01
 -4.47790823e-03]
[-1.58626598e-01 -1.06775307e+00  2.86649389e-01 -5.80083200e-01
 -8.16087444e-01]
[-3.69046117e-01  1.92285954e+00  2.73158414e+00  5.55144836e-01
 -8.21993329e-01]
[-2.37950473e-01 -1.51483188e-01 -3.88229597e-01 -3.84024393e-01
  2.69176225e+00]
[ 3.95032519e-01 -6.57384730e-01 -5.00499081e-01 -5.18227690e-01
  5.09294632e-02]
[-3.71020781e-01 -6.33136806e-01 -6.28801929e-01  6.58363482e-01
 -2.41085086e-01]
[ 2.05892584e-01 -6.87921593e-01 -4.03491819e-01  3.77059418e-01
 -4.30690166e-01]
[-1.85721356e-01  2.65175735e+00  3.65610771e-01 -3.38897248e-01
```

```
 -6.00632662e-01]
 [-4.72795086e-01  5.61779159e-01 -5.73196518e-01  4.53892654e-01
  -6.35069981e-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               :  [-3.17523785e-16 -7.32747196e-17 -1.19904087e-16
-3.88578059e-17
  4.04121181e-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
```

[ ]:
```
```