Assn2_Parth_Dethaliya_24-27-29

October 8, 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]:
```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]:
```
# 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]:
```python
# 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]:
```python
# 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]:
```python
# b 10x10 matrix
column_vector = arr2.reshape(10,1)
arr3 = column_vector + arr2
```

[11]:
```python
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]:
```python
# c

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

```
[[0.42091235 1.81489674 1.2949665  0.58422979 2.38532519]
 [0.61188755 0.87601509 1.64796024 1.64762384 2.18702232]
 [0.33631897 0.69952875 0.83422619 0.51233697 0.59033491]
 [0.52212359 2.00099907 0.61464681 1.14010647 0.75440642]
 [2.46580777 1.04810113 1.31750982 0.41296156 0.22908517]
 [0.15316116 1.12689728 1.90409151 0.39081889 1.22615262]
 [1.327881   3.58721671 0.2343564  1.57593254 0.22931869]
 [1.27292078 0.35139919 2.41428525 0.57429689 1.34286326]
 [0.8927045  0.28857607 1.36392418 1.00525404 0.20494896]
 [0.21936135 0.42803847 1.19286316 0.36689174 1.80378151]
 [0.49875248 0.75926858 1.20437877 1.70210154 1.19371655]
 [0.34186869 0.9234617  3.1882551  5.83969796 1.05176531]
 [2.26950066 2.25961792 6.92050537 0.93378732 0.23326291]
 [4.19662721 0.25796419 0.81361865 3.55083127 1.8957381 ]
 [3.77087997 9.74062167 1.61211074 1.69601021 1.08735038]
```

```
 [0.35303374 2.71934653 1.51480657 0.85993871 4.16258598]
 [1.6023097  2.81675301 0.4551519  0.55495611 1.26641427]
 [0.68729947 1.17787661 0.3874774  1.0668792  0.12699152]
 [0.38451703 1.29694607 3.05392167 4.48615237 3.09838709]
 [0.19579052 0.43897576 0.33713382 1.14525335 0.99551986]
 [3.01026952 1.03755039 3.80318382 7.62871862 3.20092983]
 [0.47207691 2.47544979 0.94739815 1.15578457 0.95687394]
 [3.9254264  1.30836942 0.69063721 1.28162544 0.93707013]
 [0.77278342 1.15420957 1.44249057 1.45725718 0.23187154]
 [1.17028233 7.60871501 0.92776427 3.97459831 1.14967504]
 [1.18317739 0.06685034 0.58067847 0.91948932 2.56007873]
 [0.26090828 0.27392161 0.63837938 0.72388201 0.43173133]
 [0.60456112 0.19076568 6.26034424 4.02307158 0.36005   ]
 [4.68000551 0.64653731 0.43333346 2.55122262 0.335259  ]
 [4.65988911 4.56541545 0.31655695 2.41463032 2.98149856]
 [1.54400665 3.29375977 0.60260571 4.02791774 0.65478625]
 [0.67765205 0.52881606 0.495344   0.21218031 1.03136661]
 [0.39715672 1.66253289 0.4273032  1.21164467 0.90588554]
 [1.75600092 5.08004171 1.77893088 0.32118667 0.66727983]
 [0.22915325 3.07957824 0.77967188 1.39811072 0.1613056 ]
 [0.52959561 0.93140499 0.27303201 0.4277564  1.12831673]
 [4.58957364 1.46960084 1.98922323 0.2830903  0.0883134 ]
 [4.37604175 0.89960079 0.39855598 2.22630584 0.55176236]
 [0.36958557 0.90848496 2.20429381 0.68811048 3.12624859]
 [0.43176793 1.43469277 1.91701947 1.99636105 1.58885266]
 [1.32049741 1.25574995 1.7315332  1.23882423 5.74420062]
 [2.18711746 0.86557516 1.04966168 1.70370909 0.89403645]
 [0.96797475 1.10828932 0.52764718 2.2731851  0.86323126]
 [0.61447571 0.23135699 2.7875905  2.71122411 1.87639708]
 [0.62374627 0.81194585 1.06122462 1.32569027 1.09096522]
 [0.94855639 0.64871491 2.04571813 0.10269383 1.80556829]
 [1.20292213 1.3547615  0.64283624 0.2042067  0.18870633]
 [0.47721508 0.15987833 1.046163   0.41480417 0.44217532]
 [2.32501727 0.25149159 0.197126   2.86407459 0.39751699]
 [0.97200968 0.23312634 0.6376289  0.14782383 0.4919565 ]]
```

[13]:

```python
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)
```

```
[1.34713926 1.83227701 1.34099051 1.53082909 1.13968739]
[1.39606209 1.60299376 1.41880132 1.63910482 1.25817761]
```

[14]:

```python
normalized = (data-mean)/std
print(normalized)
```

```
[[-0.72386707  0.11565008 -0.09234579 -0.68908739  0.98899714]
```

```
[-0.58210355 -0.39676243  0.1708878   0.00556497  0.81499954]
[-0.7866619  -0.4930832  -0.43592787 -0.73605071 -0.58598763]
[-0.64873657  0.21721896 -0.59967204 -0.32596607 -0.44202577]
[ 0.79408693 -0.3028432  -0.07553484 -0.80096678 -0.90296028]
[-0.92262246 -0.2598387   0.36188936 -0.81543128 -0.0280998 ]
[-0.05061176  1.08292739 -0.88326122 -0.04126671 -0.90275538]
[-0.09140949 -0.68308152  0.74234972 -0.69557597  0.07430603]
[-0.37364927 -0.71736844 -0.04092284 -0.41405718 -0.9241382 ]
[-0.87348115 -0.64125418 -0.16848602 -0.83106147  0.47873118]
[-0.66608527 -0.46047906 -0.15989864  0.04115203 -0.05656031]
[-0.78254227 -0.37086754  1.31951253  2.74399876 -0.18111309]
[ 0.64836546  0.35836511  4.10271662 -0.46074216 -0.89929459]
[ 2.07889801 -0.73407545 -0.45129527  1.24881769  0.55941698]
[ 1.76285997  4.44126509  0.1441542   0.03717292 -0.14988956]
[-0.77425429  0.60927074  0.07159279 -0.50898308  2.54842546]
[ 0.15310044  0.66243218 -0.71861017 -0.70821015  0.00722712]
[-0.52612424 -0.23201577 -0.76907623 -0.37380111 -0.99254068]
[-0.75088381 -0.16703134  1.21933775  1.85980758  1.61466161]
[-0.8909781  -0.63528495 -0.80661831 -0.32260392 -0.23046473]
[ 1.19824837 -0.30860146  1.7780756   3.91266003  1.70463605]
[-0.68588691  0.47615946 -0.35153356 -0.3157245  -0.26437396]
[ 1.87758191 -0.16079683 -0.54300468 -0.23352011 -0.2817505 ]
[-0.46266833 -0.2449325   0.01766548 -0.11879029 -0.90051542]
[-0.16759942  3.27773652 -0.36617489  1.5256396  -0.0952038 ]
[-0.15802725 -0.83837947 -0.62500282 -0.47008219  1.14233177]
[-0.84264029 -0.72536639 -0.58197425 -0.59786086 -0.72515173]
[-0.58754206 -0.77075031  3.61042295  1.55730432 -0.78804734]
[ 2.43771637 -0.52200429 -0.73488057  0.59583255 -0.8097998 ]
[ 2.42278369  1.61679794 -0.82196285  0.50660489  1.51209969]
[ 0.10982128  0.92276768 -0.6086513   1.56047003 -0.52943585]
[-0.53328566 -0.5862529  -0.68863823 -0.93212528 -0.1990116 ]
[-0.7415012   0.03249461 -0.73937744 -0.2792344  -0.3091129 ]
[ 0.26718754  1.897665    0.26855489 -0.86091789 -0.51847357]
[-0.86621249  0.80587404 -0.47660997 -0.15742717 -0.96243235]
[-0.64318999 -0.36653234 -0.85442015 -0.79130219 -0.1139443 ]
[ 2.37058754 -0.07280172  0.42537356 -0.88580399 -1.02647816]
[ 2.21207989 -0.38389009 -0.76081474  0.38358366 -0.61983247]
[-0.76196764 -0.37904138  0.58575544 -0.62122829  1.63910822]
[-0.71580882 -0.09185346  0.37152996  0.23337434  0.2901454 ]
[-0.0560927  -0.18951491  0.23320961 -0.26147961  3.93618728]
[ 0.58721128 -0.40246022 -0.27527387  0.04220215 -0.31950969]
[-0.31777512 -0.26999435 -0.66454918  0.4142071  -0.34653919]
[-0.58018232 -0.74859684  1.02072995  0.70035205  0.54244652]
[-0.57330066 -0.43172943 -0.26665118 -0.20473516 -0.14671778]
[-0.33218964 -0.52081582  0.46750279 -1.00364632  0.48029897]
[-0.14337045 -0.13547748 -0.57865069 -0.93733397 -0.93839003]
[-0.68207278 -0.78760768 -0.2778829  -0.79976312 -0.71598782]
[ 0.6895762  -0.737608   -0.91102459  0.80020022 -0.75517254]
```

[-0.31477994 -0.74763118 -0.58253389 -0.97416556 -0.67230815]]

[15]:
```
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               :  [ 7.54951657e-17 -9.54791801e-17 -1.11022302e-17
-1.35447209e-16
  2.22044605e-16]
```

[16]:
```
# 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
```

[ ]:

```

```