Lec13-numpy-RyanSponzilli

October 8, 2024

1 ASTR 310 Lecture 13 - numpy

1.0.1 1. Manipulating numpy arrays

Create the following arrays and print the requested elements.

a). Create a 10x5x4 array A containing 200 values logarithmically spaced between 10^{-3} and 10^{7}. Print A[3,0,2], A[5,4,1], and the last element of A. [2 pts]

[43]:
```
import numpy as np
A = np.logspace(-3, 7, 200, base=10).reshape(10,5,4)

print(A[3,0,2])
print(A[5,4,1])
print(A[-1,-1,-1])
```

```
1.3049019780144029
757.525025877192
10000000.0
```

b). Create an array B containing

$$B = \begin{bmatrix} -2 & 8 & 10 & 1 \\ 17 & 9 & 2 & 0 \\ 1 & 6 & -4 & 10 \\ 3 & 8 & -9 & 4 \end{bmatrix}$$

and set all the elements of B larger than 4 equal to 4. Print B. [2 pts]

[48]:
```
B = np.array([
    [-2,8,10,1],
    [17,9,2,0],
    [1,6,-4,10],
    [3,8,-9,4]
])

mask = (B >= 4)
B[mask] = 4

print(B)
```

```
[[-2  4  4  1]
 [ 4  4  2  0]
 [ 1  4 -4  4]
 [ 3  4 -9  4]]
```

c). Create a 10x10 array C that contains zeros in all elements for which either index is odd and ones in all other elements. Print C. [2 pts]

[74]:
```
C = np.zeros((10,10))
C[0:10:2, 0:10:2] = 1
print(C)
```

```
[[1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [1. 0. 1. 0. 1. 0. 1. 0. 1. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
```

1.0.2 2. Computing with numpy arrays

Use NumPy arrays to do the following.

a). Create two arrays x and y containing 10 equally-spaced values between $-\pi$ and π. Then create a 2d grid of values z containing $\sin(x)\cos(y)$ at the coordinates (x,y) given by the elements of x and y. (See the reading on meshes or np.indices.) [2 pts]

[96]:
```
x = y = np.linspace(-np.pi, np.pi, 10)

xx, yy = np.meshgrid(x,y)

z = np.sin(xx)*np.cos(yy)

print(z)
```

```
[[ 1.22464680e-16  6.42787610e-01  9.84807753e-01  8.66025404e-01
   3.42020143e-01 -3.42020143e-01 -8.66025404e-01 -9.84807753e-01
  -6.42787610e-01 -1.22464680e-16]
 [ 9.38133875e-17  4.92403877e-01  7.54406507e-01  6.63413948e-01
   2.62002630e-01 -2.62002630e-01 -6.63413948e-01 -7.54406507e-01
  -4.92403877e-01 -9.38133875e-17]
 [ 2.12657685e-17  1.11618897e-01  1.71010072e-01  1.50383733e-01
   5.93911746e-02 -5.93911746e-02 -1.50383733e-01 -1.71010072e-01
  -1.11618897e-01 -2.12657685e-17]
 [-6.12323400e-17 -3.21393805e-01 -4.92403877e-01 -4.33012702e-01
  -1.71010072e-01  1.71010072e-01  4.33012702e-01  4.92403877e-01
```

```
   3.21393805e-01  6.12323400e-17]
 [-1.15079156e-16 -6.04022774e-01 -9.25416578e-01 -8.13797681e-01
  -3.21393805e-01  3.21393805e-01  8.13797681e-01  9.25416578e-01
   6.04022774e-01  1.15079156e-16]
 [-1.15079156e-16 -6.04022774e-01 -9.25416578e-01 -8.13797681e-01
  -3.21393805e-01  3.21393805e-01  8.13797681e-01  9.25416578e-01
   6.04022774e-01  1.15079156e-16]
 [-6.12323400e-17 -3.21393805e-01 -4.92403877e-01 -4.33012702e-01
  -1.71010072e-01  1.71010072e-01  4.33012702e-01  4.92403877e-01
   3.21393805e-01  6.12323400e-17]
 [ 2.12657685e-17  1.11618897e-01  1.71010072e-01  1.50383733e-01
   5.93911746e-02 -5.93911746e-02 -1.50383733e-01 -1.71010072e-01
  -1.11618897e-01 -2.12657685e-17]
 [ 9.38133875e-17  4.92403877e-01  7.54406507e-01  6.63413948e-01
   2.62002630e-01 -2.62002630e-01 -6.63413948e-01 -7.54406507e-01
  -4.92403877e-01 -9.38133875e-17]
 [ 1.22464680e-16  6.42787610e-01  9.84807753e-01  8.66025404e-01
   3.42020143e-01 -3.42020143e-01 -8.66025404e-01 -9.84807753e-01
  -6.42787610e-01 -1.22464680e-16]]
```

b). Find the sum over the innermost 4x4 grid points of z. (Your answer should be close to zero.) [1 pt]

```
[99]: print(np.sum(z[3:7, 3:7]))
```

```
-6.661338147750939e-16
```

c). Define a function f(x) that returns x^2 if its input is a scalar. If its input is an array, return an array with the same shape containing the squares of the element values. Print f(z) and f(2). [1 pt]

```
[112]: def f(x):
           if isinstance(x, np.ndarray):
               return x**2
           else:
               return x**2

       print(f(z))
       print(f(2))
```

```
[[1.49975978e-32 4.13175911e-01 9.69846310e-01 7.50000000e-01
  1.16977778e-01 1.16977778e-01 7.50000000e-01 9.69846310e-01
  4.13175911e-01 1.49975978e-32]
 [8.80095168e-33 2.42461578e-01 5.69129177e-01 4.40118067e-01
  6.86453782e-02 6.86453782e-02 4.40118067e-01 5.69129177e-01
  2.42461578e-01 8.80095168e-33]
 [4.52232910e-34 1.24587782e-02 2.92444446e-02 2.26152672e-02
  3.52731162e-03 3.52731162e-03 2.26152672e-02 2.92444446e-02
  1.24587782e-02 4.52232910e-34]
 [3.74939946e-33 1.03293978e-01 2.42461578e-01 1.87500000e-01
  2.92444446e-02 2.92444446e-02 1.87500000e-01 2.42461578e-01
  1.03293978e-01 3.74939946e-33]
 [1.32432122e-32 3.64843511e-01 8.56395844e-01 6.62266666e-01
  1.03293978e-01 1.03293978e-01 6.62266666e-01 8.56395844e-01
  3.64843511e-01 1.32432122e-32]
 [1.32432122e-32 3.64843511e-01 8.56395844e-01 6.62266666e-01
  1.03293978e-01 1.03293978e-01 6.62266666e-01 8.56395844e-01
  3.64843511e-01 1.32432122e-32]
 [3.74939946e-33 1.03293978e-01 2.42461578e-01 1.87500000e-01
  2.92444446e-02 2.92444446e-02 1.87500000e-01 2.42461578e-01
  1.03293978e-01 3.74939946e-33]
 [4.52232910e-34 1.24587782e-02 2.92444446e-02 2.26152672e-02
  3.52731162e-03 3.52731162e-03 2.26152672e-02 2.92444446e-02
  1.24587782e-02 4.52232910e-34]
 [8.80095168e-33 2.42461578e-01 5.69129177e-01 4.40118067e-01
  6.86453782e-02 6.86453782e-02 4.40118067e-01 5.69129177e-01
  2.42461578e-01 8.80095168e-33]
 [1.49975978e-32 4.13175911e-01 9.69846310e-01 7.50000000e-01
  1.16977778e-01 1.16977778e-01 7.50000000e-01 9.69846310e-01
  4.13175911e-01 1.49975978e-32]]
4
```

d). Create a 1D array containing the maximum value of z for each value of y. (Hint: see the help file on `np.max` for how to control the dimensionality of its output.) [2 pts]

[114]:
```python
print(np.max(z, axis=1))
```

```
[0.98480775 0.75440651 0.17101007 0.49240388 0.92541658 0.92541658
 0.49240388 0.17101007 0.75440651 0.98480775]
```

1.0.3 3. Working with grids and random numbers

a). Construct two 1D coordinate arrays x and y, each filled with 100 random numbers drawn from a uniform (flat) distribution ranging from 0 to 1 inclusive. Sort the arrays. [2 pts]

[124]:
```python
import numpy.random as npr

x = npr.rand(100)
y = npr.rand(100)
x = np.sort(x)
y = np.sort(y)

print(x)
print(y)
```

```
[0.00116546 0.0268371  0.03371251 0.03428371 0.03556604 0.0404675
 0.04188057 0.04649392 0.05021149 0.05676537 0.06106487 0.06954411
 0.11306618 0.12154151 0.1228342  0.13252469 0.13936843 0.14122487
 0.14145854 0.14328414 0.14772945 0.15551487 0.16479763 0.16576714
 0.17046973 0.17236418 0.17239926 0.19036268 0.19662523 0.20987727
 0.21399168 0.22370228 0.22371419 0.25582853 0.25815873 0.25984743
 0.27292658 0.30382201 0.31657567 0.32661121 0.34554826 0.35550094
 0.36248474 0.3658812  0.3887499  0.40604069 0.40733922 0.43081207
 0.43386654 0.43688904 0.44038029 0.47100837 0.50938902 0.51201593
 0.51981038 0.53135945 0.53898622 0.54065403 0.54227214 0.55830955
 0.55934744 0.57440967 0.59626999 0.59832069 0.61858516 0.63022341
 0.63401846 0.6344483  0.63709808 0.6381198  0.65064356 0.67980246
 0.70435968 0.70697095 0.70940644 0.71966153 0.72585642 0.76008612
 0.77760325 0.7814456  0.78268106 0.79762044 0.8043159  0.81011184
 0.81755522 0.8223897  0.83096716 0.85107029 0.87245164 0.88116115
 0.89114278 0.9194288  0.9298339  0.93744866 0.93894309 0.95793987
 0.98357433 0.98509089 0.99021403 0.99852295]
[0.03128755 0.03806182 0.04101909 0.05174054 0.05888604 0.06148884
 0.06566946 0.07551897 0.08107908 0.10167717 0.10412622 0.11604675
 0.14166109 0.17812391 0.17908308 0.2002479  0.21122143 0.21198916
 0.22624947 0.23925664 0.29478751 0.29488684 0.29604273 0.30362067
 0.30852751 0.31743748 0.31787636 0.32595426 0.32978292 0.33229627
 0.34634062 0.34932855 0.36453062 0.37028992 0.37601095 0.37924821
 0.39532439 0.40875008 0.41139813 0.41933494 0.42549593 0.43430087
 0.43820267 0.44845648 0.45201208 0.45454531 0.47212342 0.47310427
 0.49664644 0.52294744 0.54808351 0.55243137 0.5861467  0.61028435
 0.61510991 0.62612187 0.63685345 0.6483917  0.65847311 0.65864976
 0.6686578  0.67638497 0.68354237 0.69235471 0.70259831 0.70280557
 0.7058297  0.70653311 0.71533342 0.72232807 0.7322222  0.7557716
 0.75918066 0.78186113 0.80467323 0.81103775 0.81433026 0.83528692
 0.83660235 0.84014809 0.84247779 0.85448655 0.8626499  0.86549361
 0.86977024 0.87404677 0.88235045 0.88877935 0.89164526 0.90949224
 0.91680451 0.93028098 0.94842078 0.94992591 0.95063402 0.95074822
 0.9596028  0.96437075 0.97386773 0.97676239]
```

b). Construct 2D meshgrids from x and y. Use the meshgrids to construct the 2D uniformly sampled function

$$z(x, y) = \exp\left[-\frac{x^2 + y^2}{0.05}\right].$$

[2 pts]

[127]:
```python
X, Y = np.meshgrid(x, y)
z = np.exp(-(X**2 + Y**2)/0.05)
print(z)
```

```
[[9.80585555e-01 9.66588115e-01 9.58573650e-01 … 3.65321023e-09
  2.98384048e-09 2.14411336e-09]
 [9.71415293e-01 9.57548754e-01 9.49609240e-01 … 3.61904606e-09
  2.95593614e-09 2.12406199e-09]
 [9.66882328e-01 9.53080495e-01 9.45178029e-01 … 3.60215832e-09
  2.94214270e-09 2.11415037e-09]
 …
 [8.35633361e-09 8.23705052e-09 8.16875302e-09 … 3.11318511e-17
  2.54276298e-17 1.82716607e-17]
 [5.78270509e-09 5.70015945e-09 5.65289658e-09 … 2.15436963e-17
  1.75962918e-17 1.26442565e-17]
 [5.16519736e-09 5.09146637e-09 5.04925048e-09 … 1.92431468e-17
  1.57172670e-17 1.12940361e-17]]
```

c). NumPy provides a histogram function that we can use to find the frequency of occurrence of different values of z. The most common usage is `hist, bin_edges = np.histogram(z, bins=10)` where the bins argument supplies the number of bins (or a list of bin right-edge values), hist contains the histogram counts, and bin_edges contains the bin edges (its length is len(hist) + 1). Use the histogram function to construct a histogram of z values with bin width 0.1, and print the counts in a table like the one below.

```
1.) 0.0  0.1    9013
2.) 0.1  0.2     245
3.) 0.2  0.3     ...
```

[4 pts]

[172]:
```
hist, bin_edges = np.histogram(z, [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0])

for i in np.indices(hist.shape)[0]:
    print("{i:4} {re:4.1f} {le:4.1f} {v:5}".format(i=f"{i+1}.
 ↪)",re=bin_edges[i],le=bin_edges[i+1], v=hist[i]))
```

```
1.)    0.0  0.1  9065
2.)    0.1  0.2   272
3.)    0.2  0.3   126
4.)    0.3  0.4   126
5.)    0.4  0.5    80
6.)    0.5  0.6   100
7.)    0.6  0.7    73
8.)    0.7  0.8    48
9.)    0.8  0.9    55
10.)   0.9  1.0    55
```