CV-ps0

cavidano3

August 2020

1 Problem 1

(a)
```
>>> x = np.random.permutation(1000)
```

When only handed an integer permutation first creates and array as if np.arrange(1000) was called which creates and array from 0 to 999 with every integer in between. Then creates a permutation of that array eg shuffles all values so this would return an array with every value between 0 and 999 in a random order.

(b)
```
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> b = a[2,:]
```

Assigns to the variable a a 3x3 matrix (still just called arrays in numpy sorry I do a lot more eigen nowadays and thats totally going to mess with me). The matrix representation is given below.

```
   1  2  3
4  5  6
7  8  9
```

Then it assigns b to be all of the values of the 3rd row. So b equals an np.array of 7,8,9

(c)
```
>>> a = np.array([[1,2,3],[4,5,6],[7,8,9]])
>>> b = a.reshape(-1)
```

A is the same as in part b. Then when assigned to b the array is flattend eg the numpy array is just a 1d array of the values from 1 to 9

(d)
```
>>> x = np.random.randn(5,1)
>>> y = x[x>0]
```

The args to randn are the dimensions of the output array. So x is equal to a np.array of shape (5,1) with each entry being pull from a normal distribution around zero. The y = x[x¿0] is using conditional indexing to only add the entries of x that are positive

(e)
```
>>> x = np.zeros(10)+0.5
>>> y = 0.5*np.ones(len(x))
>>> z = x + y
```

Using np.zeros or np.ones with 1 argument (shape argument) will create an array of that length consisting of only zeros or ones, respectively. These are both then manipulated by broadcasting operations with scalars which applies the operation to each element in the array individually. So each element of x is simply 0.5 and each element of y is also 0.5. Then z is a broadcasting operation between 2 equal size arrays which means each element in the arrays are added to the value at the same index in the other array and the output consists of an array of size len(x) with each index being the result of element wise additions. So just a convoluted way to make an array of ones

```
(f) >>> a = np.arange(1,100)
    >>> b = a[::-1]
```

Assigns a to an array consisting of all the integers from 1 and 99 (exclusive of 100).

`a[::-1]`

means that no starting and stopping index are given and thus all values are going to be indexed in a flat array manner but the last element of this indexing expression is the increment value -1, which means the are going to be indexed in reverse order

2 Problem 2

(a) Write a function random_dice(N) that returns the roll of a six-sided die over N trials using np.random.rand.

```
def random_dice(N):
    return ((np.random.rand(N) * 6) + 1).astype(int)
```

(b) Write a function reshape_vector(y) that takes a six element vector (e.g. y = np.array([1, 2, 3, 4, 5, 6])) and converts it into a two column matrix (e.g. np.array([[1,2],[3,4],[5,6]])) using np.reshape.

```
def reshape_vector(y):
  return y.reshape((3,2))
```

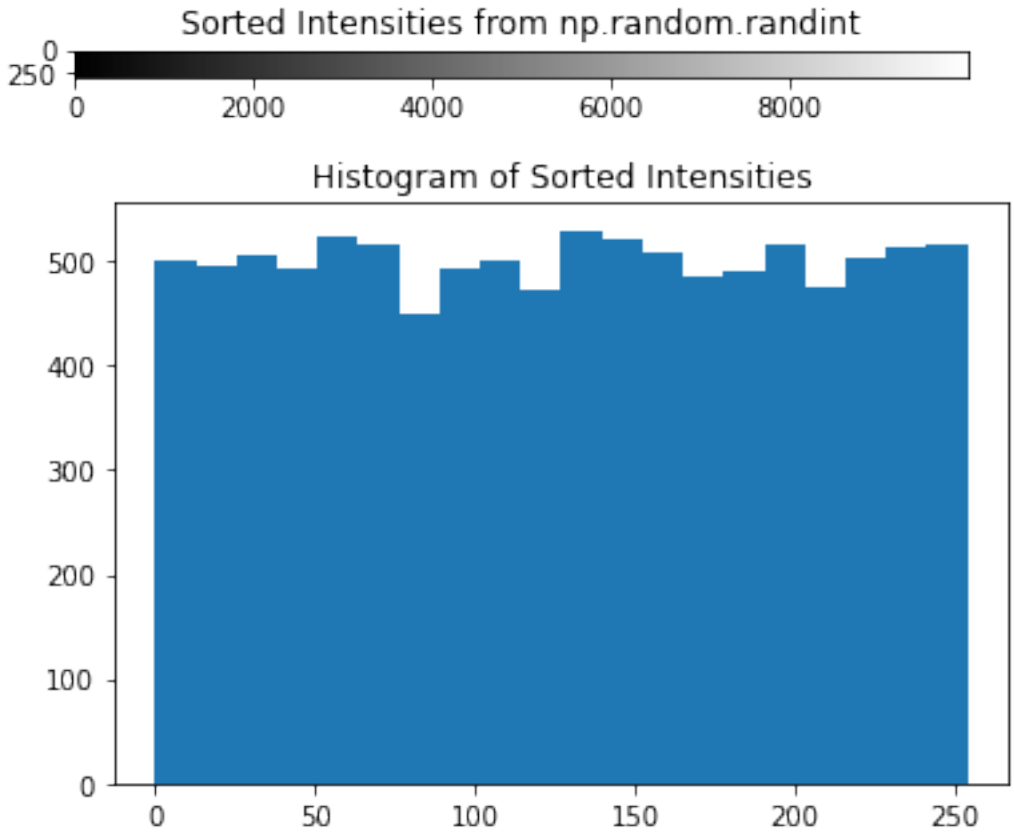
(c) Write a function max_value(z) that takes a 2D matrix (e.g. z = np.array([[1,2],[3,4],[5,6]])) and returns the row and column of the first occurrence of the maximum value using np.max and np.where.

```
def max_value(z):
  mx = np.max(z)
  return np.where(z == mx)
```

(d) Write a function count_ones(v) that returns the number of 1's that occur in the vector v.

```
def count_ones(v):
  return len(v[v == 1])
```

3 Problem 3


Sorted Intensities from np.random.randint
0
250
0
2000
4000
6000
8000
Histogram of Sorted Intensities
500
400
300
200
100
0
0
50
100
150
200
250

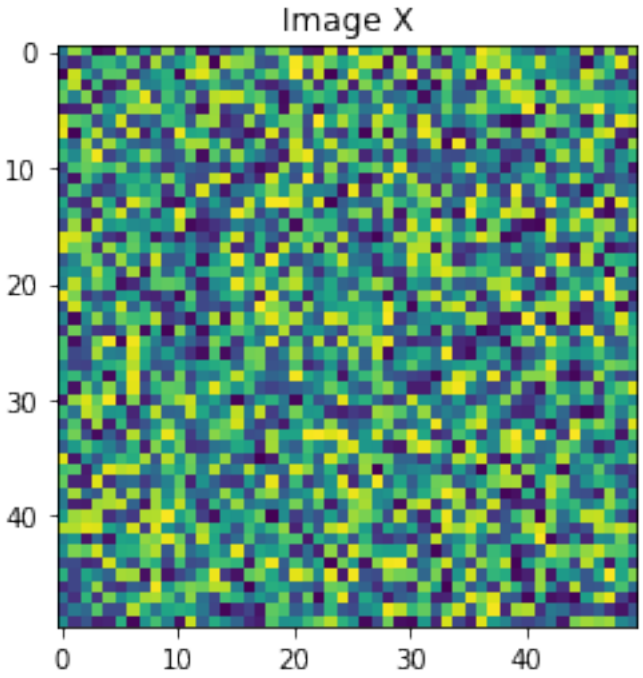


Image X
0
10
20
30
40
0
10
20
30
40

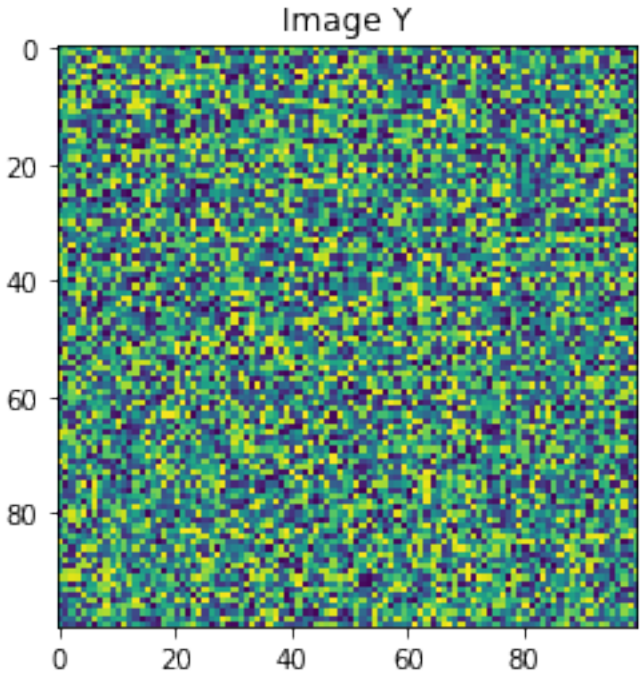


Image Y
0
20
40
60
80
0
20
40
60
80

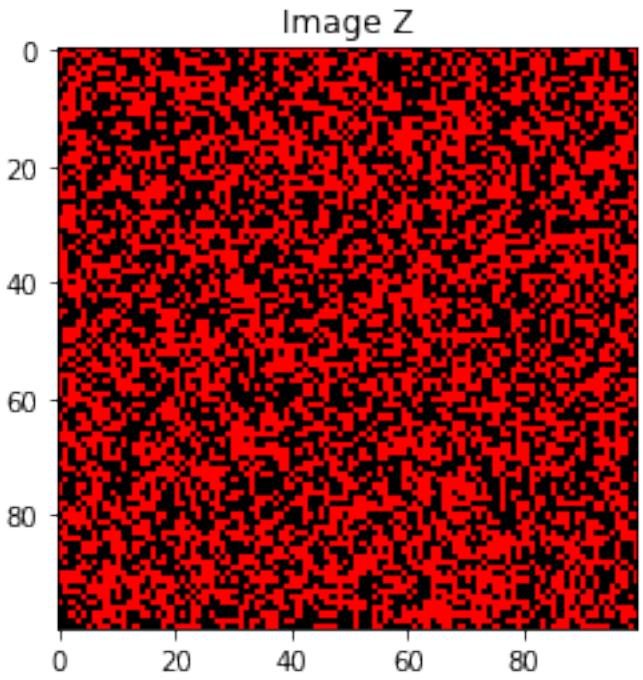
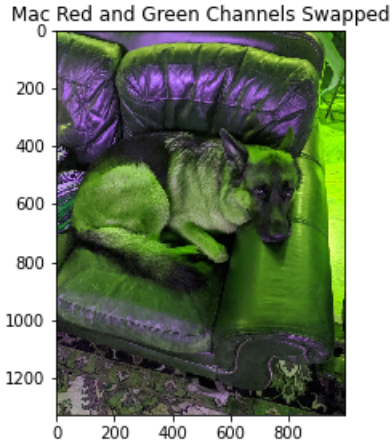
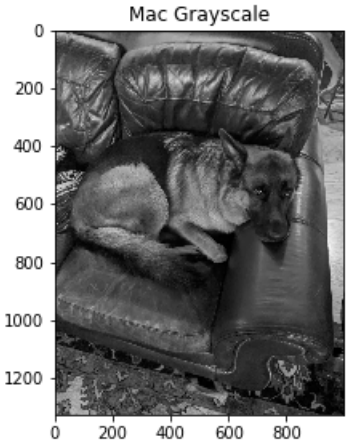
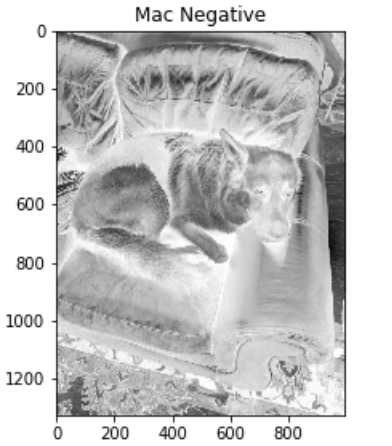


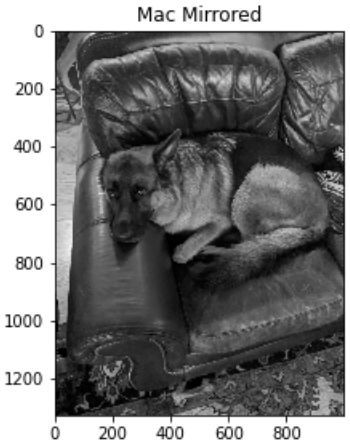
Image Z
0
20
40
60
80
0
20
40
60
80

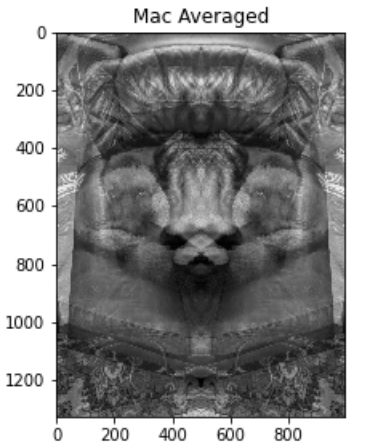
4 Problem 4

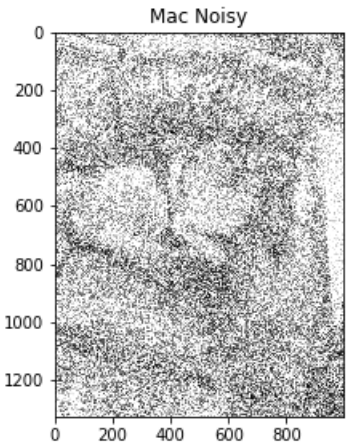

Mac Red and Green Channels Swapped
0
200
400
600
800
1000
1200
0
200
400
600
800


Mac Grayscale
0
200
400
600
800
1000
1200
0
200
400
600
800


Mac Negative
0
200
400
600
800
1000
1200
0
200
400
600
800


Mac Mirrored
0
200
400
600
800
1000
1200
0
200
400
600
800


Mac Averaged
0
200
400
600
800
1000
1200
0
200
400
600
800


Mac Noisy
0
200
400
600
800
1000
1200
0
200
400
600
800