ECE285-Assignment-1

October 5, 2018

1 1 Getting started – Python, Platform and Jupyter

```
In [1]: print("HelloWorld!")
```

```
HelloWorld!
```

2 2 Numpy

2.0.1 2.1 Arrays

1. Run the following:

```
In [2]: import numpy as np
        a = np.array([1, 2, 3])
        print(type(a))
        print(a.shape)
        print(a[0], a[1], a[2])
        a[0] = 5
```

```
<class 'numpy.ndarray'>
(3,)
1 2 3
```

```
In [3]: print(a[0], a[1], a[2])
```

```
5 2 3
```

What are the rank, shape of a, and the current values of a[0], a[1], a[2]?
Answer: 1. The rank of a is 1. 2. The shape of a is (3,). 3. a[0], a[1], a[2] = 5, 2, 3

2. Run the following:

```
In [4]: b = np.array([[1, 2, 3], [4, 5, 6]])
```

```
In [5]: print(b[0, 0], b[0, 1], b[1, 0])
```

```
1 2 4
```

What are the rank, shape of b, and the current values of b[0, 0], b[0, 1], b[1, 0]?
Answer: 1. The rank of a is 2. 2. The shape of a is (2,3). 3. b[0, 0], b[0, 1], b[1, 0] = 1, 2, 4

3. Numpy also provides many functions to create arrays. Assign the correct comment to each of the following instructions:

```
In [6]: a = np.zeros((2,2)) # an array of all zeros
        b = np.ones((1,2)) # an array of all ones
        c = np.full((2,2), 7) # a constant array
        d = np.eye(2) # a 2x2 identity matrix
        e = np.random.random((2,2)) # an array filled with random values
```

2.0.2 2.2 Array Indexing

4. Run the following

```
In [7]: a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
        b = a[:2, 1:3]
```

The shape of a and b are:

```
In [8]: print(a.shape)
        print(b.shape)

(3, 4)
(2, 2)
```

The values in b are:

```
In [9]: b

Out[9]: array([[2, 3],
               [6, 7]])
```

5. A slice of an array is a view into the same data. Follow the comments in the following snippet

```
In [10]: print(a[0, 1]) # Prints "2"
         b[0, 0] = 77
         print(a[0, 1])

2
77
```

Last line prints 77, modifying b also modifies the original array.

6. You can also mix integer indexing with slice indexing. However, doing so will yield an array of lower rank than the original array. Note that this is quite different from the way that MATLAB handles array slicing. Create the following rank 2 array with shape (3, 4):

```
In [11]: a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

```
In [12]: row_r1 = a[1, :] # Rank 1 view of the second row of a
         row_r2 = a[1:2, :] # Rank 2 view of the second row of a
         print(row_r1, row_r1.shape) # Prints "[5 6 7 8] (4,)"
         print(row_r2, row_r2.shape) # Prints "[[5 6 7 8]] (1, 4)"
         col_r1 = a[:, 1]
         col_r2 = a[:, 1:2]
```

```
[5 6 7 8] (4,)
[[5 6 7 8]] (1, 4)
```

What are the values and shapes of col r1 and col r2?

```
In [13]: print(col_r1, col_r1.shape)
         print(col_r2, col_r2.shape)
```

```
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
```

7. Run the following:

```
In [14]: a = np.array([[1,2], [3, 4], [5, 6]])
         print(a[[0, 1, 2], [0, 1, 0]])
         print(np.array([a[0, 0], a[1, 1], a[2, 0]]))
```

```
[1 4 5]
[1 4 5]
```

They are equivalent.

8. When using integer array indexing, you can duplicate several times the same element from the source array. Compare the following instructions:

```
In [15]: b = a[[0, 0], [1, 1]]
         c = np.array([a[0, 1], a[0, 1]])
```

```
In [16]: print(b)
         print(c)
```

```
[2 2]
[2 2]
```

They are equivalent.

9. One useful trick with integer array indexing is selecting or mutating one element from each row of a matrix. Run the following code

```
In [17]: a = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
         b = np.array([0, 2, 0, 1])
         print(a[np.arange(4), b])
         a[np.arange(4), b] += 10
         print(a)
```

```
[ 1  6  7 11]
[[11  2  3]
 [ 4  5 16]
 [17  8  9]
 [10 21 12]]
```

Observation:

np.arange(4) = [0, 1, 2, 3] and b = [0, 2, 0, 1] So a[np.arange(4), b]) will be [a[0,0], a[1,2], a[2,0], a[3,1]]. That is [1, 6, 7, 11]

Modifying this diretly via a[np.arange(4), b] += 10 will also modify the original values in a

10. Run the following

```
In [18]: a = np.array([[1,2], [3, 4], [5, 6]])
         bool_idx = (a > 2)
```

What is the result stored in bool idx? Run the following:

What are the values printed? What is their shape?

```
In [19]: print(a[bool_idx])
         print(a[a > 2])
         print(a[bool_idx].shape)
         print(a[a > 2].shape)
```

```
[3 4 5 6]
[3 4 5 6]
(4,)
(4,)
```

Their shapes are the same, (4,)

2.0.3 2.3 Datatypes

11. Here is an example What are the datatypes of x, y, z?

```
In [20]: x = np.array([1, 2]) # Let numpy choose the datatype
         print(x.dtype)
         y = np.array([1.0, 2.0]) # Let numpy choose the datatype
         print(y.dtype)
         z = np.array([1, 2], dtype=np.int32) # Force a particular datatype
         print(z.dtype)
```

```
int64
float64
int32
```

2.0.4 2.4 Array math

12. Give the output for each print statement in the following code snippet

```
In [21]: x = np.array([[1,2],[3,4]], dtype=np.float64)
         y = np.array([[5,6],[7,8]], dtype=np.float64)
         # Elementwise sum; both produce the array
         print(x + y)
         print(np.add(x, y))
         # Elementwise difference; both produce the array
         print(x - y)
         print(np.subtract(x, y))
         # Elementwise product; both produce the array
         print(x * y)
         print(np.multiply(x, y))
         # Elementwise division; both produce the array
         print(x / y)
         print(np.divide(x, y))
         # Elementwise square root; produces the array
         print(np.sqrt(x))
```

```
[[ 6.  8.]
 [10. 12.]]
[[ 6.  8.]
 [10. 12.]]
[[-4. -4.]
 [-4. -4.]]
[[-4. -4.]
 [-4. -4.]]
[[ 5. 12.]
 [21. 32.]]
[[ 5. 12.]
 [21. 32.]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[0.2        0.33333333]
 [0.42857143 0.5       ]]
[[1.         1.41421356]
 [1.73205081 2.        ]]
```

13. What are the mathematical operations performed by the last two instructions?

```
In [22]: v = np.array([9,10])
         w = np.array([11, 12])
         print(v.dot(w))
         print(np.dot(v, w))
```

```
219
219
```

It performs the inner product of vectors. 9*11 + 10*12 = 219

14. What are the mathematical operations performed in the snippet below?

```
In [23]: x = np.array([[1,2],[3,4]])
         print(x.dot(v))
         print(np.dot(x, v))
```

```
[29 67]
[29 67]
```

It performs the matrix product.

15. Write a code to compute the product of the matrices x with the following matrix y

```
In [24]: y = np.array([[5,6,7],[7,8,9]])
         print(np.dot(x,y))
```

```
[[19 22 25]
 [43 50 57]]
```

```
In [25]: print(x.shape)
         print(y.shape)
```

```
(2, 2)
(2, 3)
```

```
In [26]: print(np.dot(x,y))
```

```
[[19 22 25]
 [43 50 57]]
```

```
In [27]: print(np.dot(y.T,x))
```

```
[[26 38]
 [30 44]
 [34 50]]
```

```
In [28]: x = np.array([[1,2],[3,4]])
         print(np.sum(x)) # Compute sum of all elements; prints "10"
         print(np.sum(x, axis=0)) # Compute sum of each column; prints "[4 6]"
         print(np.sum(x, axis=1)) # Compute sum of each row; prints "[3 7]"
```

```
10
[4 6]
[3 7]
```

16. In the following

```
In [29]: x = np.array([[1,2,3], [3,4,5]])
         print(x)
         print(x.T)
```

```
[[1 2 3]
 [3 4 5]]
[[1 3]
 [2 4]
 [3 5]]
```

What is the transpose of x? How is it different from x?
Exchange the element based on the symmetry for diagonal.

2.0.5 2.5 Broadcasting

17. Complete the following code in order to add the vector v to each row of a matrix x:

```
In [30]: x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
         v = np.array([1,0,1])
         y = np.zeros(x.shape)
         for i in range(x.shape[0]):
             for j in range(x.shape[1]):
                 y[i,j] = x[i, j] + v[j]
         print(y)
```

```
[[ 2.  2.  4.]
 [ 5.  5.  7.]
 [ 8.  8. 10.]
 [11. 11. 13.]]
```

18. The previous solution works well; however when the matrix x is very large, computing an explicit loop in Python could be slow. An alternative is to use tile. Run the following and interpret the result.

```
In [31]: vv = np.tile(v, (4, 1))
         print(vv)
```

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

Next, complete the code to compute y from x and vv without explicit loops.

```
In [32]: x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
         v = np.array([1,0,1])
         vv = np.tile(v, (4, 1))
         y = x + vv
         print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

19. Numpy broadcasting allows us to perform this computation without actually creating multiple copies of v. This is simply obtained as follows

```
In [33]: x = np.array([[1,2,3], [4,5,6], [7,8,9], [10, 11, 12]])
         v = np.array([1, 0, 1])
         y = x + v # Add v to each row of x using broadcasting
         print(y)
```

```
[[ 2  2  4]
 [ 5  5  7]
 [ 8  8 10]
 [11 11 13]]
```

Compute the outer product of v and w using broadcasting

```
In [34]: v = np.array([1,2,3]) # v has shape (3,)
         w = np.array([4,5]) # w has shape (2,)

In [35]: print(np.reshape(v, (3, 1)) * w)
```

```
[[ 4  5]
 [ 8 10]
 [12 15]]
```

20. Add v to each row of x using broadcasting

```
In [36]: x = np.array([[1,2,3], [4,5,6]])

In [37]: print(x + v)

[[2 4 6]
 [5 7 9]]
```

21. Add w to each column of x using broadcasting.

```
In [38]: print(x + np.reshape(w, (2, 1)))

[[ 5  6  7]
 [ 9 10 11]]
```

22. Numpy treats scalars as arrays of shape (). Write a code to multiply x with 2.

```
In [39]: print(x * 2)

[[ 2  4  6]
 [ 8 10 12]]
```

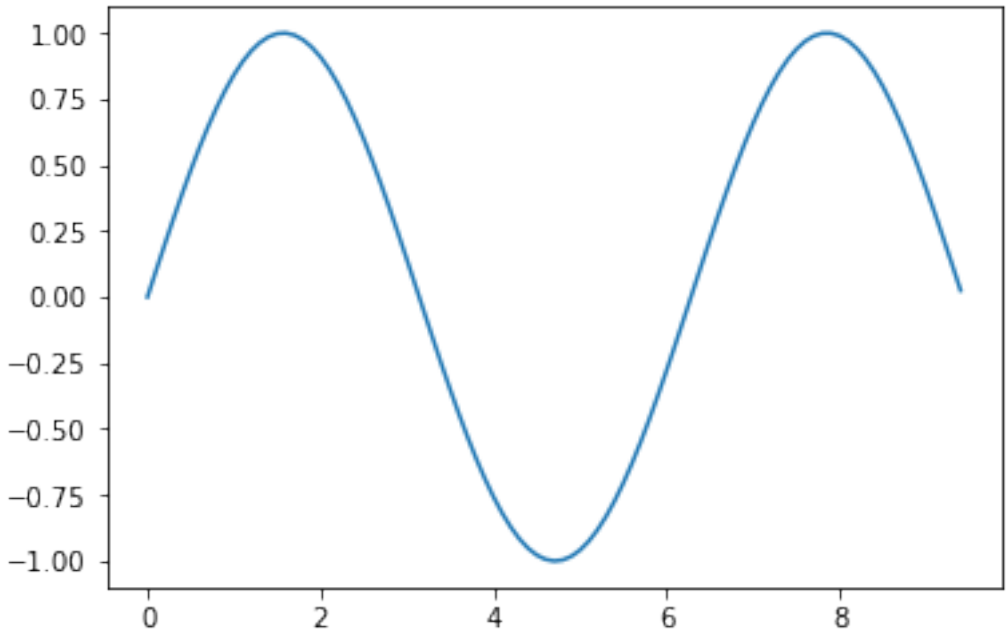
2.0.6 2.6 Numpy Documentation

3 3 Matplotlib

3.0.1 3.1 Plotting

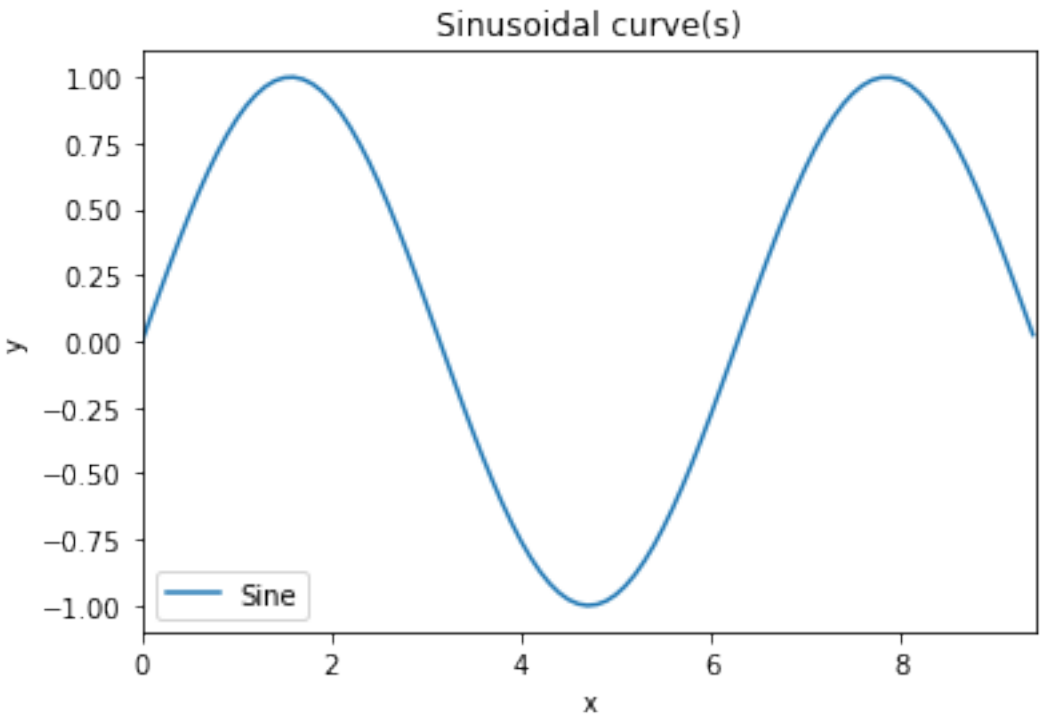
An important function in matplotlib is plot, which allows you to plot 2D graphs. Here is a simple example:

```
In [41]: import numpy as np
         import matplotlib.pyplot as plt
         # Compute the x and y coordinates for points on a sine curve
         x = np.arange(0, 3 * np.pi, 0.1)
         y = np.sin(x)
         # Plot the points using matplotlib
         plt.plot(x, y)
         plt.show() # You must call plt.show() to make graphics appear.
```



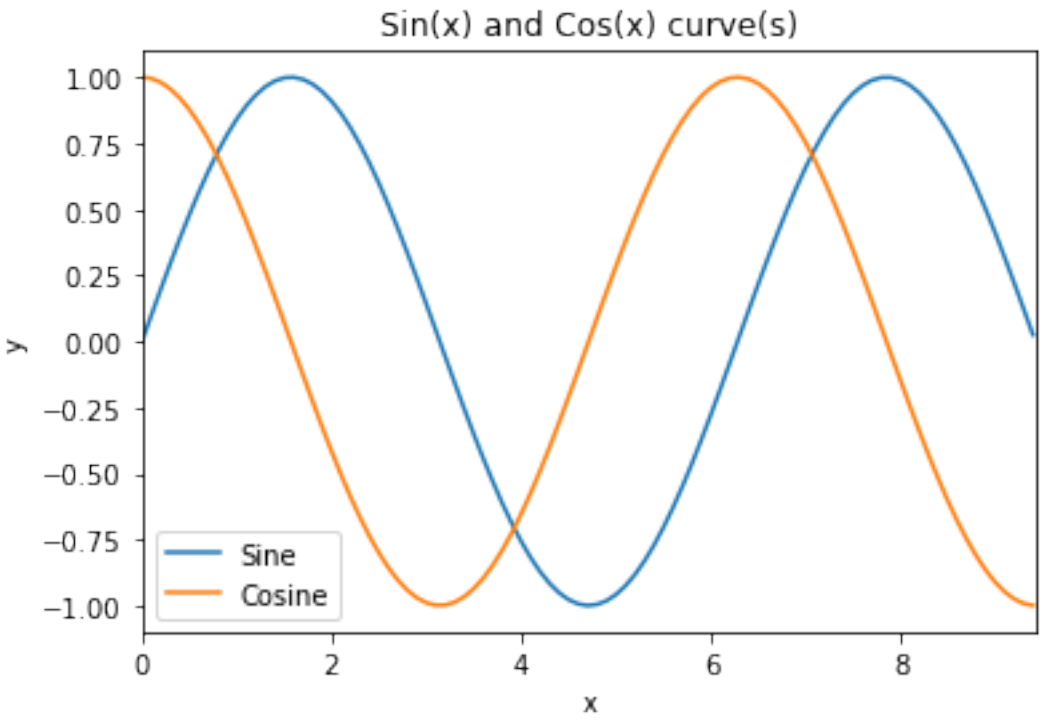
24. Modify the above example by adding these extra instructions before plt.show()

```
In [42]: # Compute the x and y coordinates for points on a sine curve
         x = np.arange(0, 3 * np.pi, 0.1)
         y = np.sin(x)
         # Plot the points using matplotlib
         plt.plot(x, y)

         plt.xlim(0, 3 * np.pi)
         plt.xlabel('x')
         plt.ylabel('y')
         plt.title('Sinusoidal curve(s)')
         plt.legend(['Sine'])

         plt.show() # You must call plt.show() to make graphics appear.
```



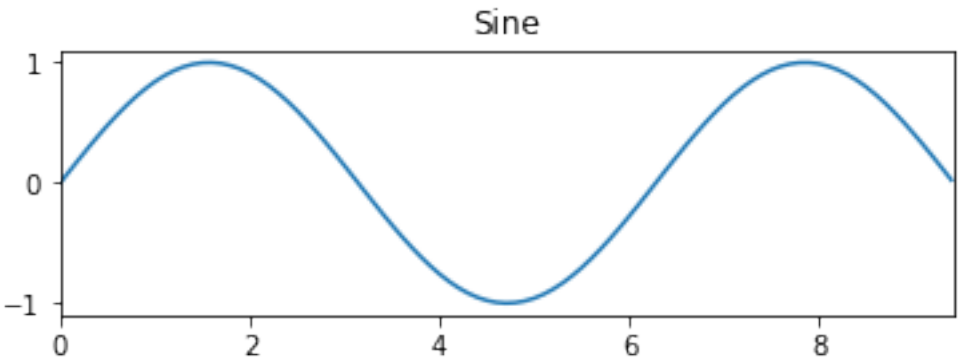
25. Write a code to plot both sine and cosine curves in the same plot with proper legend.

```
In [43]: # Compute the x and y coordinates for points on a sine curve
         x = np.arange(0, 3 * np.pi, 0.1)
         y1 = np.sin(x)
         y2 = np.cos(x)
         # Plot the points using matplotlib
         plt.plot(x, y1)
         plt.plot(x, y2)

         plt.xlim(0, 3 * np.pi)
         plt.xlabel('x')
         plt.ylabel('y')
         plt.title('Sin(x) and Cos(x) curve(s)')
         plt.legend(['Sine', 'Cosine'])

         plt.show() # You must call plt.show() to make graphics appear.
```



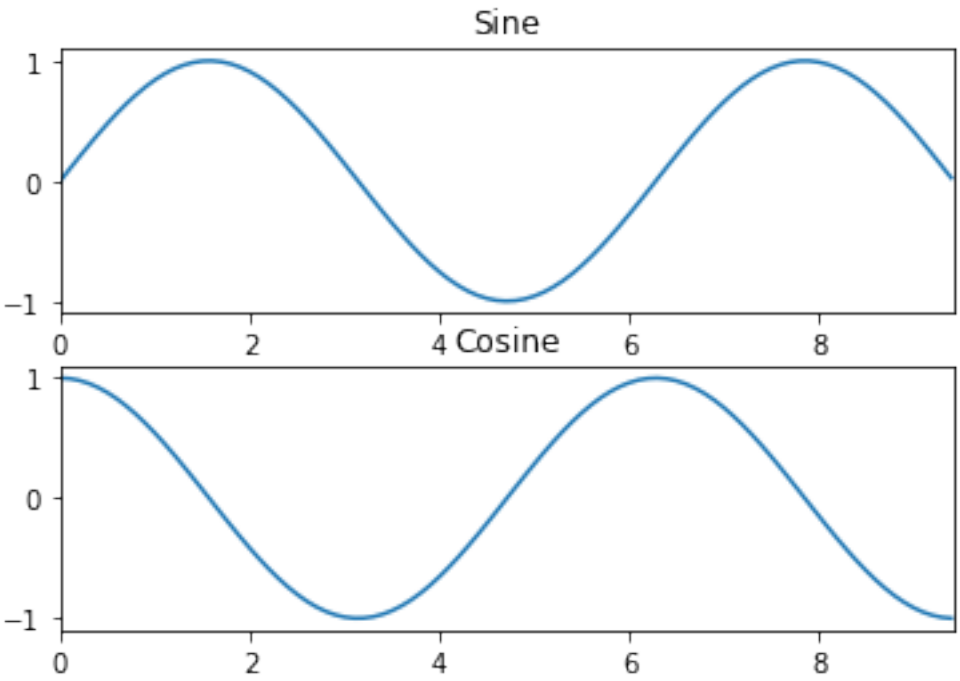
3.0.2 3.2 Subplots

```
In [44]: x = np.arange(0, 3 * np.pi, 0.1)
         y_sin = np.sin(x)
         # Set up a subplot grid that has height 2 and width 1,
         # and set the first such subplot as active.
         plt.subplot(2, 1, 1)
         # Make the first plot
         plt.plot(x, y_sin)
         plt.xlim(0, 3 * np.pi)
         plt.title('Sine')

         plt.show()
```



26. Complete the above code to create a second subplot in the same grid representing the cosine function.

```
In [45]: x = np.arange(0, 3 * np.pi, 0.1)
         y_sin = np.sin(x)
         y_cos = np.cos(x)

         # Set up a subplot grid that has height 2 and width 1,
         # and set the first such subplot as active.
         plt.subplot(2, 1, 1)
         # Make the first plot
         plt.plot(x, y_sin)
         plt.xlim(0, 3 * np.pi)
         plt.title('Sine')

         plt.subplot(2, 1, 2)
         # Make the second plot
         plt.plot(x, y_cos)
         plt.xlim(0, 3 * np.pi)
         plt.title('Cosine')

         plt.show()
```



3.0.3 3.3 Images

```
In [46]: import numpy as np
         import imageio
         import matplotlib.pyplot as plt
         img = imageio.imread('assets/cat.jpg')
         img_tinted = img * [1, 0.95, 0.9]
         # Show the original image
         plt.subplot(1, 2, 1)
         plt.imshow(img)
         # Show the tinted image
         plt.subplot(1, 2, 2)
         # A slight gotcha with imshow is that it might give strange results
         # if presented with data that is not uint8. To work around this, we
         # explicitly cast the image to uint8 before displaying it.
         plt.imshow(np.uint8(img_tinted))
         plt.show()
```


0
50
100
150
200
250
300
350
400
0
100
200
0
50
100
150
200
250
300
350
400
0
100
200