ECE 285 – IVR – Assignment #0 Python, Numpy and Matplotlib

Adapted by Sneha Gupta, Shobhit Trehan and Charles Deledalle from CS228, itself adapted by Volodymyr Kuleshov and Isaac Caswell from the CS231n Python tutorial by Justin Johnson (`http://cs231n.github.io/python-numpy-tutorial/`).

1 Getting started – Python, Platform and Jupyter

Python We are going to use `python` 3. For a quick brush-up on `python` refer this website: `https://docs.python.org/3/tutorial/`. For some help on how to install and configure python/conda/jupyter on your computer, please refer to the document "Setting up Python and Jupyter with Conda".

Jupyter Start a *Jupyter Notebook*. Please refer to `http://jupyter.org/` for documentation. To start, go to `File` → `New Notebook` → `Python 3`. Replace `Untitled` (top left corner, next to *jupyter*) by `HelloWorld`. Next type in the cell

```
print('HelloWorld!')
```

followed by `Ctrl+Enter`. Use `Ctrl+S` to save your notebook (regularly).

Tips:

- Learn more about tips, tricks and shortcuts.
- Backup regularly your notebooks and work with your teammates using `git`.
- Avoid conflicts by clearing all outputs before committing: `Cell` → `All Output` → `Clear`.

For the following questions, please write your code and answers directly in your notebook. Organize your notebook with headings, markdown and code cells (following the numbering of the questions).

2 Numpy

Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays. If you are already familiar with MATLAB, you might find this tutorial useful to get started with Numpy.

2.1 Arrays

A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers. The number of dimensions is the *rank* of the array; the *shape* of an array is a tuple of integers giving the size of the array along each dimension. We can initialize numpy arrays from nested Python lists, and access elements using square brackets.

1. Run the following:

```python
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
```

What are the rank, shape of `a`, and the current values of `a[0]`, `a[1]`, `a[2]`?

2. Run the following:

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

What are the rank, shape of `b`, and the current values of `b[0, 0]`, `b[0, 1]`, `b[1, 0]`?

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

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

Hint: use the `print` command to check your answer.

You can read about other methods of array creation in the documentation.

2.2 Array indexing

Slicing: Numpy offers several ways to index into arrays. Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array:

4. Run the following

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

What are the shape of a and b? What are the values in b?

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

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

What does the last line print? Does modifying b modify 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):

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

There are two ways of accessing the data in the middle row (or column) of the array. Mixing integer indexing with slices yields an array of lower rank, while using only slices yields an array of the same rank as the original array. Run the following:

```
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]
```

What are the values and shapes of col_r1 and col_r2?

Integer array indexing: When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing create new arrays using the data from another array.

7. Run the following:

```
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]]))
```

Are the two printed arrays equivalent?

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

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

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

```
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)
```

What do you observe?

Boolean array indexing: Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

10. Run the following

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

    What is the result stored in `bool_idx`?
    Run the following:

    ```
    print(a[bool_idx])
    print(a[a > 2])
    ```

    What are the values printed? What is their shape?

For brevity we have left out a lot of details about numpy array indexing; if you want to know more you should read the documentation.

2.3 Datatypes

Every numpy array is a grid of elements of the same type. Numpy provides a large set of numeric datatypes that you can use to construct arrays. Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

11. Here is an example

    ```
    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)
    ```

    What are the datatypes of `x`, `y`, `z`?

You can read all about numpy datatypes in the documentation.

2.4 Array math

Basic mathematical functions operate elementwise on arrays, and are available both as operator overloads and as functions in the numpy module:

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

    ```
    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))
```

Note that unlike MATLAB, `*` is elementwise multiplication, not matrix multiplication. We instead use the `dot` function to compute inner products of vectors, to multiply a vector by a matrix, and to multiply matrices. `dot` is available both as a function in the numpy module and as an instance method of array objects.

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

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

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

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

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

```
y = np.array([[5,6,7],[7,8,9]])
```

Can you write the same code for the product of `y` with `x`?

Numpy provides many useful functions for performing computations on arrays; a useful one is `sum`:

```
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]"
```

You can find the full list of mathematical functions provided by numpy in the documentation. Apart from computing mathematical functions using arrays, we frequently need to reshape or otherwise manipulate data in arrays. The simplest example of this type of operation is transposing a matrix; to transpose a matrix, simply use the `T` attribute of an array object.

16. In the following

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

    What is the transpose of `x`? How is it different from `x`?

Numpy provides many more functions for manipulating arrays; you can see the full list in the documentation.

2.5 Broadcasting

Broadcasting is a powerful mechanism that allows numpy to work with arrays of different shapes when performing arithmetic operations. Frequently we have a smaller array and a larger array, and we want to use the smaller array multiple times to perform some operation on the larger array.

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

    ```python
    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] = # complete
    ```

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

    ```python
    vv = np.tile(v, (4, 1))
    print(vv)
    ```

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

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

    ```python
    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
    ```

    The line `y = x + v` works even though `x` has shape `(4, 3)` and `v` has shape `(3,)` due to broadcasting; this line works as if `v` actually had shape `(4, 3)`, where each row was a copy of `v`, and the sum was performed elementwise. Broadcasting two arrays together follows these rules:

(a) If the arrays do not have the same rank, prepend the shape of the lower rank array with 1s until both shapes have the same length.

(b) The two arrays are said to be *compatible* in a dimension if they have the same size in the dimension, or if one of the arrays has size 1 in that dimension.

(c) The arrays can be broadcast together if they are compatible in all dimensions.

(d) After broadcasting, each array behaves as if it had shape equal to the elementwise maximum of shapes of the two input arrays.

(e) In any dimension where one array had size 1 and the other array had size greater than 1, the first array behaves as if it were copied along that dimension.

For more details about broadcasting please refer to the documentation or this explanation. Functions that support broadcasting are known as *universal functions*. You can find the list of all universal functions in the documentation.

Compute the outer product of `v` and `w` using broadcasting

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

Hint: use `np.reshape(v, (3, 1))`.

20. Add `v` to each row of `x` using broadcasting

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

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

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

Broadcasting typically makes your code more concise and faster, so you should strive to use it where possible.

2.6 Numpy Documentation

This brief overview has touched on many of the important things that you need to know about numpy, but is far from complete. Check out the numpy reference to find out much more about numpy.

3 Matplotlib

Matplotlib is a plotting library. In this section, we give a brief introduction to the `matplotlib.pyplot` module, which provides a plotting system similar to that of MATLAB.

3.1 Plotting

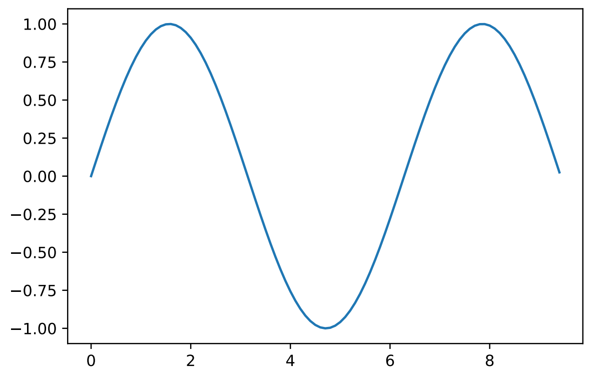
An important function in matplotlib is `plot`, which allows you to plot 2D graphs. Here is a simple example:

```
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.
```

23. Run the above example and check that it produces the following plot:



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

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

    Interpret each of these instructions.

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

You can read much more about the `plot` function in the documentation.

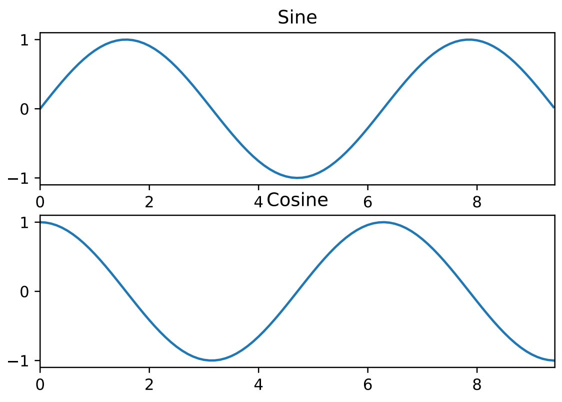
3.2 Subplots

You can display different plots in the same figure using the `subplots` function. Here is an example:

```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)

fig, axes = plt.subplots(nbrows=2)
axes[0].plot(x, y_sin)
axes[0].set_xlim(0, 3 * np.pi)
axes[0].set_title('Sine')
fig.show()
```

26. Complete the above code to create a second subplot in the same grid representing the cosine function. This is how your result should look like.



You can read much more about the `subplot` function in the documentation.

3.3 Images

27. Download `ece285_IVR_assignments.zip` and extract the file in the location of your Jupyter Notebook:

    - `assets/cat.jpg`

28. Run the following code and report your result.

```python
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
fig, axes = plt.subplots(nbcols=2)
axes[0].imshow(img)

# Show the tinted image

# 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.
axes[1].imshow(np.uint8(img_tinted))

fig.show()
```