Lab_Instructions

August 11, 2020

1 Instructions for the Lab Exercises:

Libraries: These are the libraries that we will be using for this laboratory Exercises.

Numpy

NumPy is the fundamental package for scientific computing with Python.

Scipy

The SciPy library is a collection of numerical algorithms and domain-specific toolboxes, including signal processing, optimization, statistics and much more.

matplotlib

matplotlib is a python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms.

About Each numpy and matplotlib are both very systematic, in the sense that you're usually not calling one-off functions from them, but rather using them throughout your code, almost like a sort of sub-programming language inside Python. By the end of this semester, you'll be fairly well-seasoned in both. Scipy, on the other hand, is more of a collection of various useful functions

2 Importing:

To import a specific library x, simply type import x.

To access the library function f, type x.f.

If you want to change the library name to y, type import x as y.

[55]:
```
import numpy as np
x=np.array([1,2,1])
print(x)
```

```
[1 2 1]
```

3 Data Types

Floats and Integers Unlike MATLAB, there is a difference between the int and float types in Python 2. Mainly, integer division returns the floor in Python 2. However, in Python 3 there is no floor, but it is always good to check this when debugging!

[56]:
```
1 / 4
1 / 4.0
```

[56]: 0.25

4 Strings

Unlike MATLAB/C, double quotes ("I am double quoted!") and single quotes ('I, however, am single quoted') are the same thing. Both represent strings. '+' concatenates strings

[57]:
```
"EE " + '120' # we can mix and match single and double quotes and Python won't␣
↪care
```

[57]: 'EE 120'

5 Lists

A list is a mutable array of data. That is we can change it after we create it. They can be created using square brackets []

Important functions:

'+' appends lists

len(x) to get length

[58]:
```
x = ["EE"] + [1, 2, 0]
print(x)
```

```
['EE', 1, 2, 0]
```

[59]:
```
print(len(x))
```

```
4
```

6 Numpy Array

The numpy array, aka an "ndarray", is like a list with multidimensional support and more functions.

Arithmetic operations on NumPy arrays correspond to elementwise operations.

Important NumPy Array functions:

.shape returns the dimensions of the array.

.ndim returns the number of dimensions.

.size returns the number of entries in the array.

len() returns the first dimension.

To use functions in NumPy, we have to import NumPy to our workspace. This is done by the command import numpy. By convention, we rename numpy as np for convenience.

7 Creating a Numpy Array

[60]:
```
x = np.array([[1, 2], [3 , 4]])
print(x)
```

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

7.0.1 Getting the shape of a Numpy Array

[61]:
```
x.shape # returns the dimensions of the numpy array
```

[61]:
```
(2, 2)
```

[62]:
```
np.shape(x) # equivalent to x.shape
```

[62]:
```
(2, 2)
```

8 Elementwise operations

One major advantage of using numpy arrays is that arithmetic operations on numpy arrays correspond to elementwise operations. This makes numpy amenable to vectorized implementations of algorithms, a common technique used for speeding up computer simulations than can be parallelized.

[63]:
```
print(x)
print()
print(x + 2) # numpy is smart and assumes you want this to be done to all␣
 ↪elements!
```

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

[[3 4]
 [5 6]]
```

9 Slicing numpy arrays

Numpy uses pass-by-reference semantics so it creates views into the existing array, without implicit copying. This is particularly helpful with very large arrays because copying can be slow.

[64]:
```
x = np.array([1,2,3,4,5,6])
print(x)
```

```
[1 2 3 4 5 6]
```

[65]:
```
##We slice an array from a to b-1 with [a:b].
y = x[0:4]
print(y)
```

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

[66]:
```
#Because slicing does not copy the array, changing y changes x.

y[0] = 7
print(x)
print(y)
```

```
[7 2 3 4 5 6]
[7 2 3 4]
```

[67]:
```
#To actually copy x, we should use .copy().

x = np.array([1,2,3,4,5,6])
y = x.copy()
y[0] = 7
print(x)
print(y)
```

```
[1 2 3 4 5 6]
[7 2 3 4 5 6]
```

10 Useful Numpy function: arange

We use arange to create integer sequences in numpy arrays. It's exactly like the normal range function in Python, except that it automatically returns the result as a numpy array.

arange(0,N) creates an array listing every integer from 0 to N-1.

arange(0,N,m) creates an array listing every m th integer from 0 to N-1 .

[68]:
```
print(np.arange(-5,5)) # every integer from -5 ... 4
```

```
[-5 -4 -3 -2 -1  0  1  2  3  4]
```

[69]:
```
print(np.arange(0,5,3)) # every other integer from 0 ... 4
```

```
[0 3]
```

11 numpy.linspace

numpy.linspace(start, stop, num=50, endpoint=True, retstep=False, dtype=None, axis=0)[source]
Return evenly spaced numbers over a specified interval.

Returns num evenly spaced samples, calculated over the interval [start, stop].

11.1 Plotting

We will use matplotlib.pyplot to plot signals and images.

By convention, we import matplotlib.pyplot as plt.

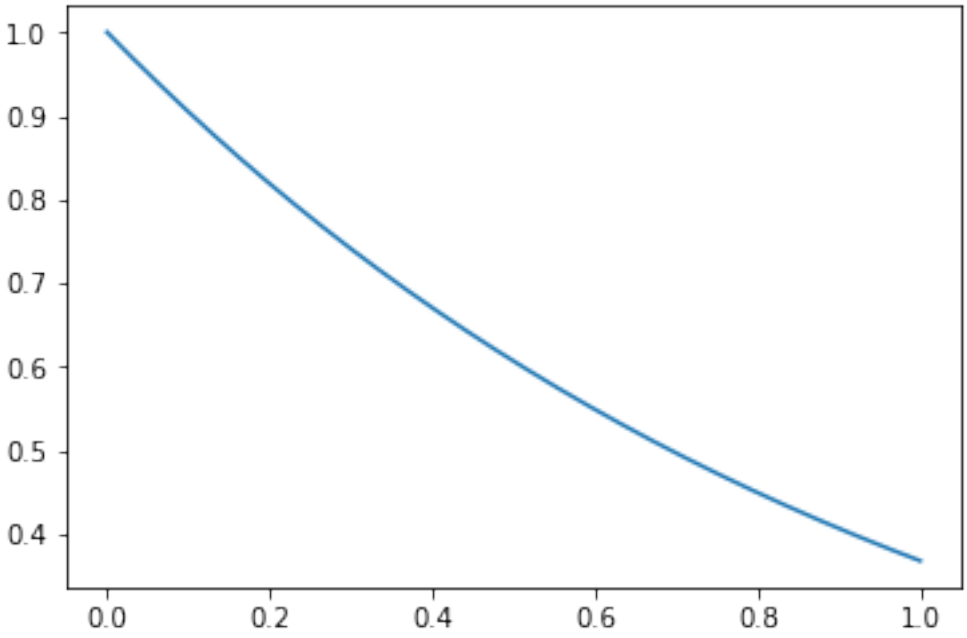
To display the plots inside the browser, we use the command %matplotlib inline - do not forget this line whenever you start a new notebook. We'll always include it for you, but in case your plots aren't showing up in the notebook when you're playing around on your own, it's probably because you forgot this. If you don't include it, Python will default to displaying it in another window on your computer, which normally is fine, but here we need the plots in the notebook so they show up in your submission PDF.

[70]:
```
import matplotlib.pyplot as plt # by convention, we import matplotlib.pyplot as␣
↪plt
```

[71]:
```
# Generate signals
x = np.arange(0, 1, 0.001)
y1 = np.exp(-x)                              # decaying exponential
y2 = np.sin(2 * np.pi * 10.0 * x)/4.0 + 0.5  # 10 Hz sine wave
```

11.1.1 Plotting One Signal

[72]:
```
plt.figure()
plt.plot(x, y1)
plt.show()
```



12 You can also add a title and legend with plt.title(), plt.legend() to make your plots look professional!

13 Using dollar signs you can add math symbols (like Latex)!

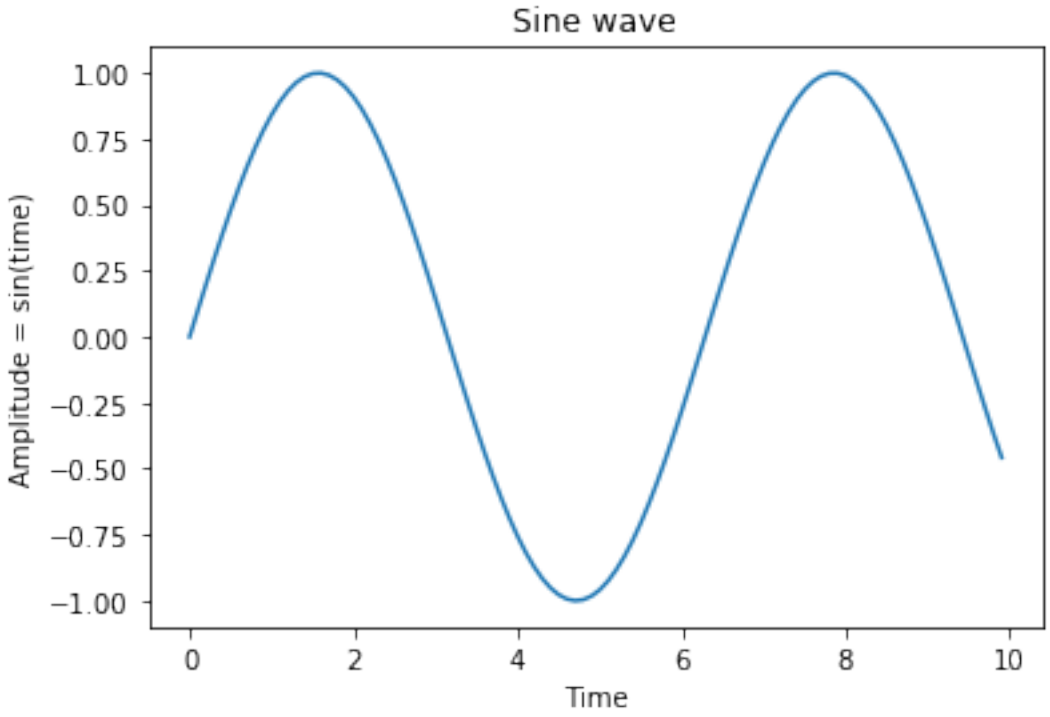
14 Examples Generate continuous time and discrete time signal for the following

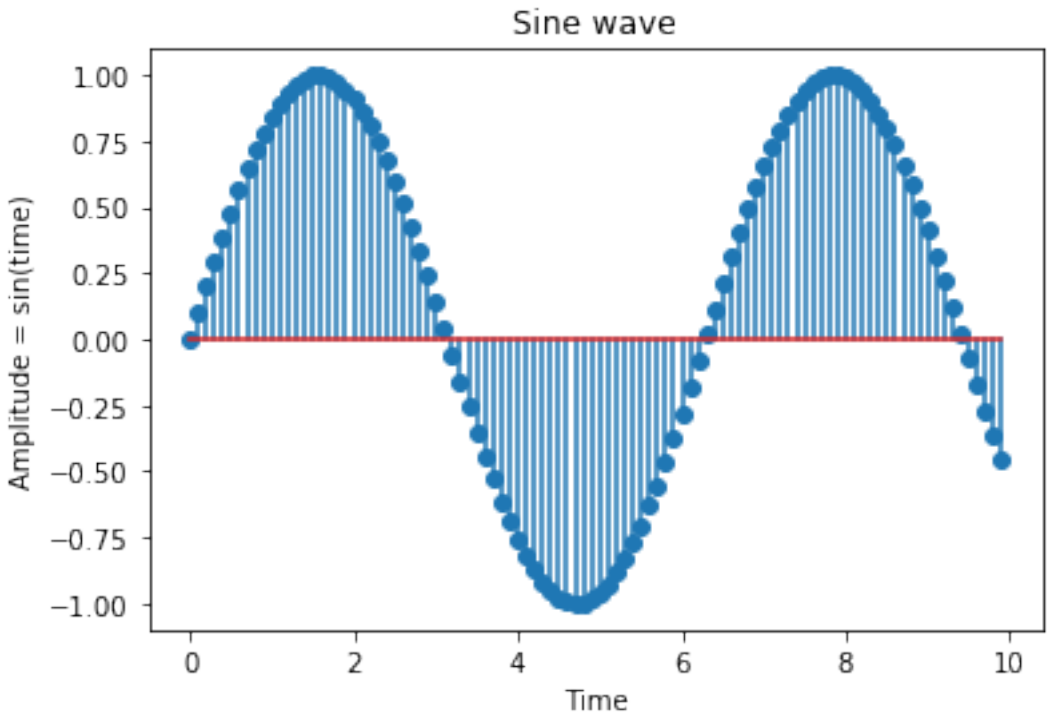
(a) y = sin(t)

[73]:
```
import matplotlib.pyplot as plt
import numpy as np
```

[74]:
```
time = np.arange(0, 10, 0.1);
amp  = np.sin(time)
plt.plot(time, amp)
plt.title('Sine wave')
plt.xlabel('Time')
plt.ylabel('Amplitude = sin(time)')
plt.show()
plt.stem(time, amp)
plt.title('Sine wave')
```

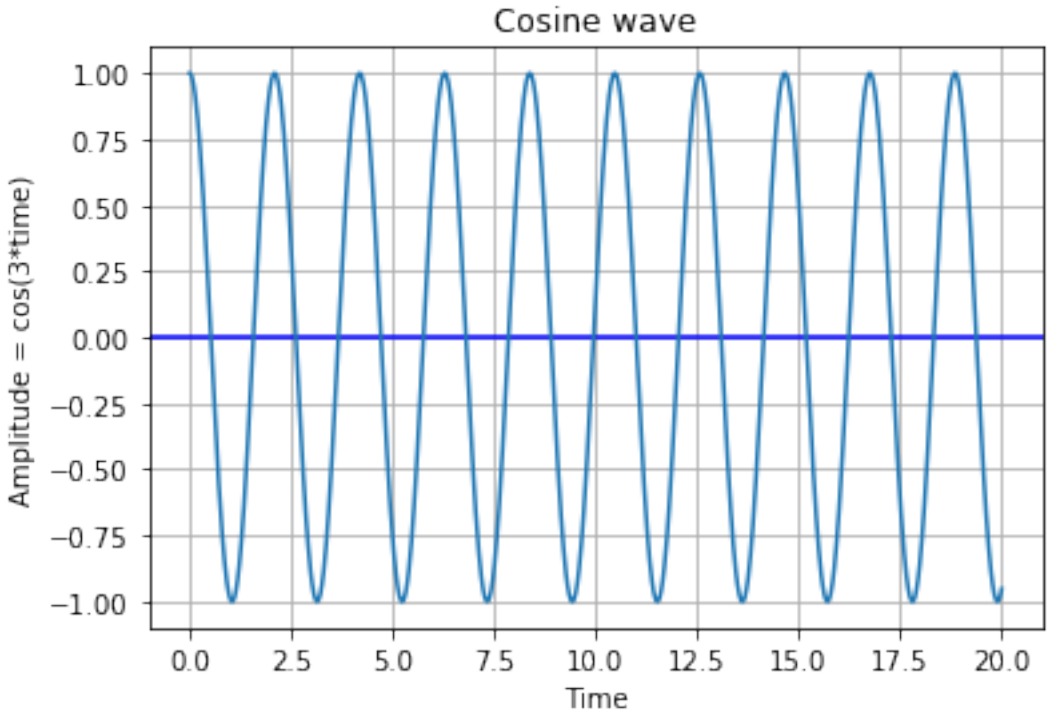
```
plt.xlabel('Time')
plt.ylabel('Amplitude = sin(time)')
plt.show()
```





(b) y= cos(3t)

[75]:
```
#time = np.arange(0, 10, 0.1);
time = np.linspace(0, 20, 1000)
amp  = np.cos(3*time)
plt.grid(True, which='both')
plt.axhline(y=0, color='b')
plt.plot(time, amp)
plt.title('Cosine wave')
plt.xlabel('Time')
plt.ylabel('Amplitude = cos(3*time)')
plt.show()
```



15 References: For complete details on the functions you can use the following links.

1. The official Python 3 language documentation. https://docs.python.org/3/.
2. The official numpy and scipy documentation. https://docs.scipy.org/doc/.
3. The official matplotlib documentation. https://matplotlib.org/contents.html