Lecture4

February 7, 2019

1 Lecture 4, Developing a physical model

Overview: * Drag in one dimension. * Running python code. * More object oriented programing in Python, developing our "Particle.py" class. * Moving code out of the Jupyter Notebook to make it reusable. The `run` statement and simple imports.

Next Lecture: Non-linear oscillations and chaos.

1.1 Tasks

1.1.1 Importing and running code

1. In your current working directory, create a new text file called "Test.py". Open "Test.py" in a text editor, define a simple string inside the file, and type the command to print that string i.e.

```
word = "hello"
print(word)
```

save the file.

2. Now in this Jupyter Notebook, enter the command `run Test.py`. In another cell, check if you have access to the variable named `word`? *Note that you must have the Jupyter notebook open in the same directory where "Test.py" is saved. To check, type* `ls` *in a notebook code cell, you should see an output of the current working directory listing and "Test.py" should be listed here.*

3. Now restart the Kernel and try `import Test` in another cell of the notebook. Do you have access to the variable `word` from this import? Try the 'dot' notation i.e. type `Test.` and then use the Tab key to see available completions.

4. Finally, open "Test.py" in your text editor and change the string in the variable `word` to something different. Try `run Test.py` and `import Test` again. What do you notice?

5. Write a simple python function in the file "test.py" and save the file. Example function:

```
def factorial(x):
   f = x
   for ii in reversed(range(x)[1:]):
       f = f*ii
   return f
```

6. From the Jupyter notebook menu, click 'Kernel' -> 'Restart & Clear Output'.
7. In the Jupyter notebook, run `Import Test`, again. Can you access and use your function? (Use the dot notation)
8. Try the command `run Test.py` again. Can you access and use your function?
9. Think about when you might want to use `import` and when you might want to use `run` for your own code.
10. Import the `sys` module and print the attribute `sys.path`. What do you think is the meaning of this variable ? For detailed explanations, read the help file.

1.1.2 Particle Object

1. Extend particle class and all subclasses to 3-Dimensions (use numpy arrays).
2. Try writing a class for an electric charge in $\vec{E}$ and $\vec{B}$ fields. $\vec{F} = q(\vec{E} + \vec{v} \times \vec{B})$.
3. Import your class into this work book and use it to explore the dynamics of an electric charge $\vec{E}$ and $\vec{B}$ fields.

```
In [1]: run Test.py
```

```
In [2]: import Test

        Test.factorial
```

```
Out[2]: <function Test.factorial(x)>
```

```
In [3]: import sys

        print(sys.path)
```

```
['', '/Users/apple/Desktop/phys1600/Lectures', '/Library/Frameworks/Python.framework/Versions/3
```

```
In [4]: import numpy as np
        x = np.array([1,2,3])
        y=[np.array([2,3,4])]
        y.append(x)
```

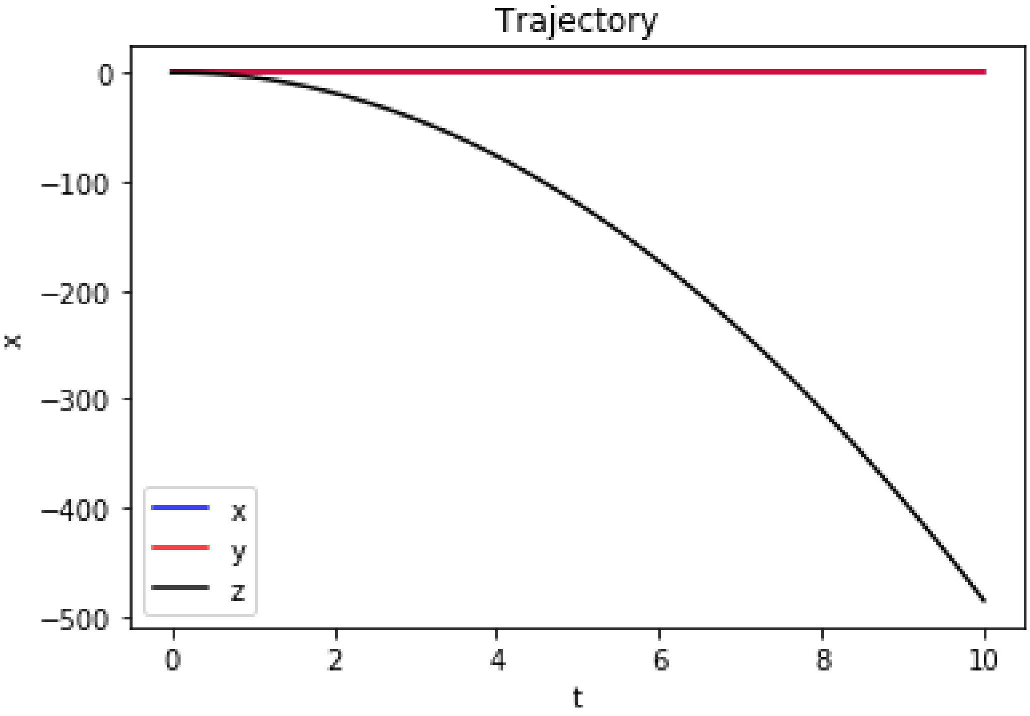
```
In [5]: y
```

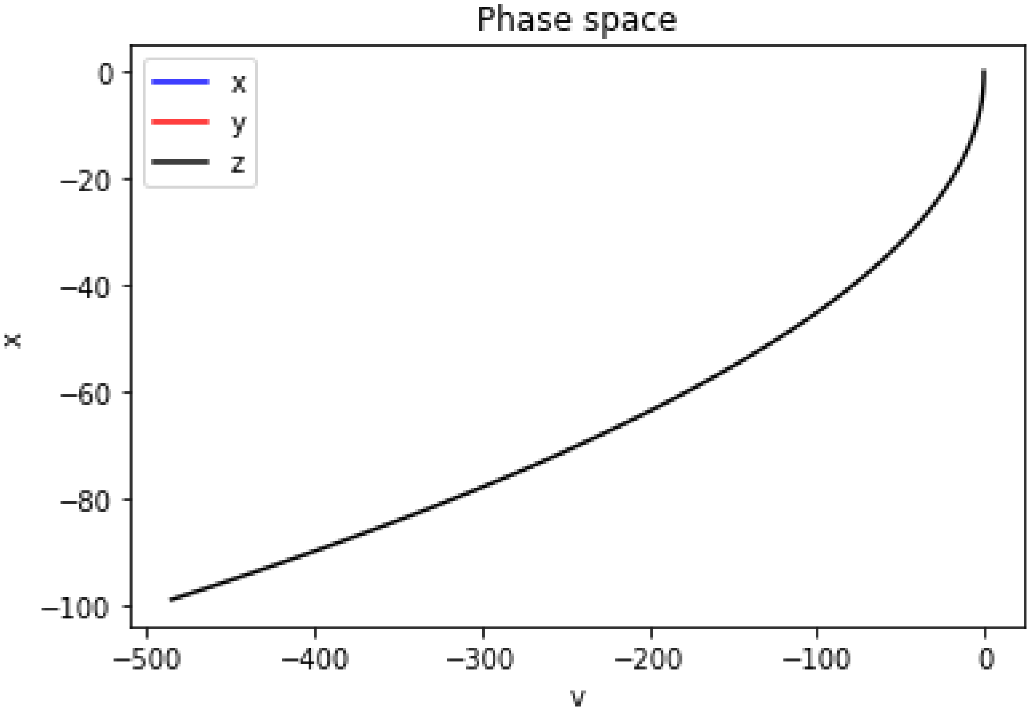
```
Out[5]: [array([2, 3, 4]), array([1, 2, 3])]
```

```
In [6]: import Particle
        fall = Particle.FallingParticle()
```

```
A new particle has been init'd
```

```
In [7]: fall.Euler_trajectory()
        fall.plot()
```



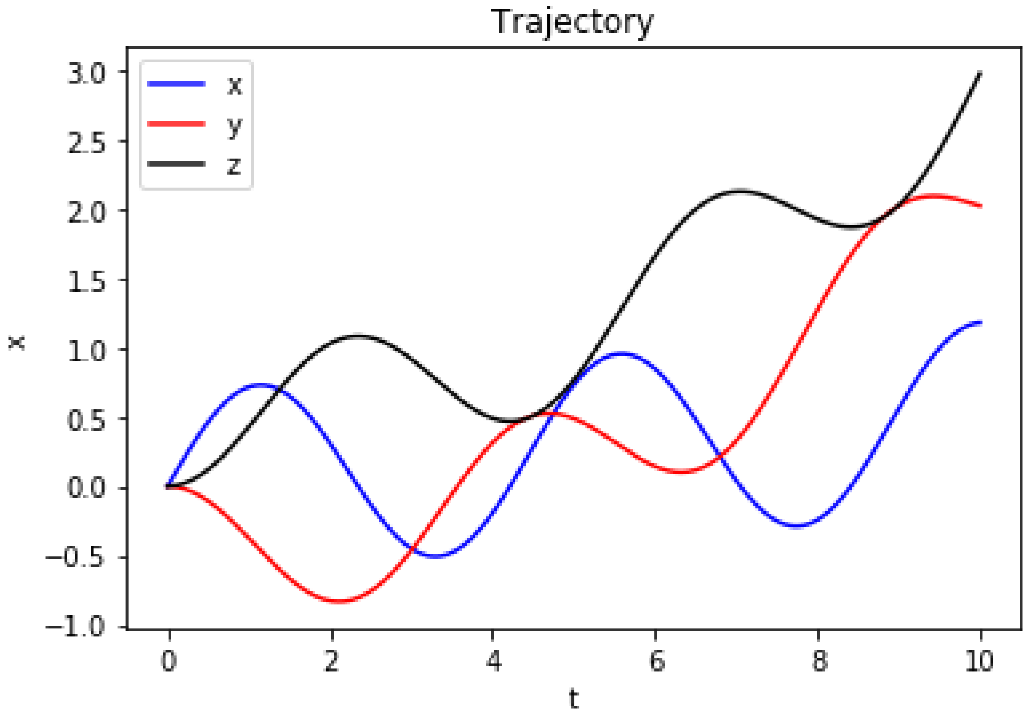


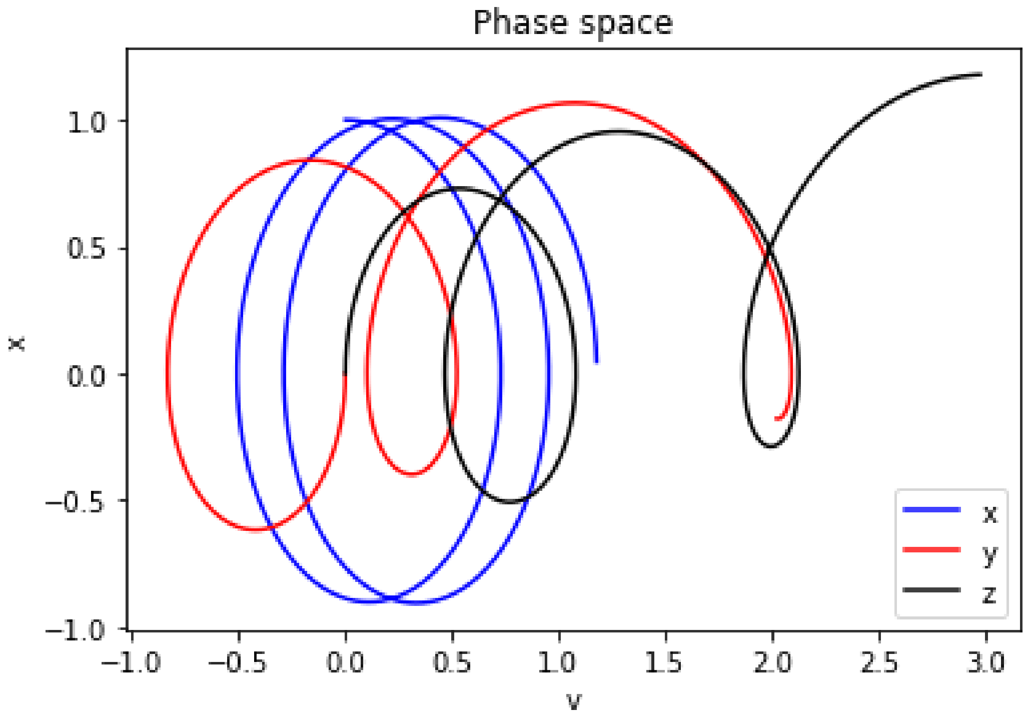
```
In [20]: electron = Particle.ElectricCharge (xv0 = np.array([0,0,0,1.0,0,0]), E = np.array([0,
```

```
A new particle has been init'd
```

```
In [21]: electron.Euler_trajectory()
```

```
In [22]: electron.plot()
```





```
In [ ]:
```

```
In [ ]:
```