1 Introduction to Plotting

Python has a small number of built-in functions. Many of the functions we will be using are in a package. The most common packages we will use are numpy, which contains many useful functions for scientific computing, and pylab, which contains functions for plotting. You can also use scipy, which contains numpy.

Type the following into a file, read the remarks explaining each line, and then copy each line into Python.

```python
import numpy

import pylab

The exponential function exp(t) is not built into Python, but it is part of numpy. Since you are using a function in the numpy package, you call it as numpy.exp(t).

```python
def myfunction(xvalue):
    f_value = xvalue**4 + 5*xvalue**2 - 3*xvalue
    return f_value

def f(x):
    return x**2 * numpy.exp(-1*x/3)
```

`linspace(start, end, num_steps)` makes a list that starts with ‘start’ and ends with ‘end’, and has num_steps equally-spaces steps.

```python
xaxis = numpy.linspace(-2,10,100)
```

applying a function to a list applies it to every element in the list.

```python
yvals = f(xaxis)
```

to make a plot you need a list of x values and a list of y values.

```python
pylab.plot(xaxis,yvals)
```

the plot now exists, so you can show it or save it

```python
pylab.show()
```

OR

```python
pylab.savefig('test1.png')
```

Find that file and view it in a way that doesn’t involve Python.
Parametric plots. Now try:

```python
liss1 = numpy.sin(2 * xaxis)
liss2 = numpy.sin(5 * xaxis)

pylab.plot(liss1, liss2)
pylab.show()
```

**Task:** If it doesn’t look good, figure out how to make it look better.

**Task:** In Problem Set 2, French’s problem 3–19, you considered a mass resting on a frictionless horizontal table and connected to rigid supports via two identical springs, each of relaxed length \( \ell_0 \) and spring constant \( k \). In this arrangement, the mass can move in two independent directions, \( x \) and \( y \), which you examined. You found the separate motion to be

\[
x(t) = A_0 \cos \left( \sqrt{\frac{2k}{m}} t \right)
\]

\[
y(t) = A_0 \cos \left( \sqrt{\frac{2k(\ell - \ell_0)}{\ell}} t \right).
\]

In that problem, you were asked to plot the resulting path of the mass \( m \) after releasing the mass from rest at the point \( x = y = A_0 \) when \( \ell = 9\ell_0/5 \). Assume that \( k/m = 2.0 \) units. Write a Python program that plots the trajectory of the mass under these conditions.

## 2 Making and manipulating lists

Python uses a powerful method for making lists called "list comprehension". It is similar to the way sets are described in mathematics. Suppose we wanted the set \{9, 16, 25, 36, 49, 64, 81, 100\}, which you recognize as the squares of the natural numbers from 3 to 10. A mathematical notation for that set might be

\[
\{ n^2 : 3 \leq n \leq 10 \}
\]

In Python, you can make a list of those numbers with this command:

```python
my_squares = [ n**2 for n in range(3,11) ]
```

Note that this is a list, not a set, because the order of the elements matters. Also recall that `range(3,11)` is the list of whole numbers starting at 3 and not including 11.

To add another element to the end of a list, use `append`:

```python
my_squares.append(121)
```
Try:

def my_squares:
    return [1, 2, 3, 4, 5, 6, 7, 8, 9]

    print(my_squares)
    pylab.plot(my_squares)
    pylab.show(my_squares)

2.1 Slicing a list

Often you need to extract part of a list. Maybe you need the 3rd element, or the last element, or the 2nd through the 5th elements.

Try all these commands and see what they do:

my_squares[3]
my_squares[0]
my_squares[-1]
my_squares[2:5]
my_squares[:5]
my_squares[-2:]
my_squares[:-2]
my_squares[4:4]

The notation [n:m] is called a ‘slice.’ You can also slice a string. Try this:

"Is today especially fun?"[7:11]

There is a big difference between strings and lists: append() does not work on strings because strings cannot be changed. The official Python way to say it is “strings are immutable.” If you want to put the letter X on the end of my_string, you can do:

    my_string = my_string + "X"

or more succinctly:

    my_string += "X"

3 Euler’s Method: an exercise in plots and lists

We will now implement Euler’s method for finding numerical approximations to differential equations. Recall that this involves starting with some initial conditions for the position, velocity, and acceleration of a particle, and choosing a time step. (This time step is used to incrementally build your approximation for the kinematic variables.) At each step, we use the previous acceleration, velocity, and position to determine the new acceleration, velocity, and position. This is perfectly suited to programming in Python.

We will use three arrays: one for each of position, velocity, and acceleration. It is also a good idea to use a fourth array for time, as this will make plotting easier. At the start, each array will contain only one entry: the initial value of that variable. We will set up a loop, where each pass through the loop is one time step. In the loop, we will compute the new values of position, velocity, and acceleration, and then we will append each of those new values to the appropriate list.
Strive to make your program readable. If you compute the new values of position, velocity, and acceleration on separate lines, and then append each of those to the appropriate list, it will be easy for other people to read your code, and it will be easy for you to check that it is correct. If you combine several steps then it may be difficult for you to debug your code, and it definitely will demand more effort for other people to read it.

In your program, you will repeatedly need to use the previous values of position, velocity, and acceleration to compute the new value. Remember that if \( \text{arry} \) is an array, then \( \text{arry}[-1] \) is the last element of the list.

**Task:** For our initial example, we will consider the familiar case of the simple harmonic oscillator arising from a mass attached to a horizontal spring. For a mass \( m \) displaced \( x \) from its equilibrium position, we can show: \( \ddot{x} = -\frac{k}{m}x \), where \( k \) is the spring constant.

Let \( x_o, v_o, a_o \), and \( x_n, v_n, a_n \), be respectively the *old* and *new* values of position, velocity, and acceleration in Euler’s method.

1. If \( \Delta t \) is the time step, write down the equations for \( x_n, v_n, \) and \( a_n \) in terms of \( x_o, v_o, \) and \( a_o \).

2. Suppose \( k = 32 \) N/m and \( m = 0.5 \) kg, and suppose the mass on the spring is initially displaced a distance 2.0 m and then released from rest. Write down the initial values of \( x, v, \) and \( a \), and then begin writing a Python program that contains three lines, creating three arrays which contain the initial values of \( x, v, \) and \( a \). 

3. In this particular case, use a time step of 0.01 sec and write down the equations for \( x_n, v_n, \) and \( a_n \) in terms of \( x_o, v_o, \) and \( a_o \).

4. Implement Euler’s method using the equations you wrote down. Set up a \( \text{for} \) loop with 300 steps. Translate your equations into Python, giving meaningful names to \( t_n, x_n, v_n, \) and \( a_n \), and using the last elements of your time, position, velocity, and acceleration lists in place of \( t_o, x_o, v_o, \) and \( a_o \). Then append your new values \( t_n, x_n, v_n, \) and \( a_n \) to those lists.

5. Run your code and then make plots of position vs time, position vs velocity, and position vs acceleration. Do the plots look as you expect? Comment. Show your plots to your instructor.

### 4 Assignment:

Modify your program for dropping a ball off a tower. Assume a drag force due to air resistance given by \( bv^2 \) where \( v \) is the speed of the ball and the drag coefficient \( b \) depends on the physical properties of the air and the ball. Prompt the user for the value of \( b \), and use Euler’s method to find the position of the ball as a function of time. Submit your program along with a plot of velocity vs time for an interesting value of tower height and drag coefficient. Explain why your plot is interesting.

### Final report:

Submit your program for the ball dropping assignment. Place it in the ‘koutsits’ drop_box before 4:30 p.m. on Monday. Be sure to identify your code appropriately; BOTH in your choice of filename and also in the introductory comments of your code.