MatPlotLib

Computational Physics

Matplotlib

Outline

- Using Matplotlib and PyPlot
  - Matplotlib and PyPlot
  - Interactive Plotting
  - Plot method
  - Labels
  - Multiple Figures and Curves
- First Steps with Programming
  - Goals
  - Structure
  - Comments and Documentation

Matplotlib and PyPlot

- Matplotlib is a library for 2D plotting.
  - Can be used in scripts or interactively
  - Uses NumPy arrays
- PyPlot is a collection of methods within Matplotlib which allow user to construct 2D plots easily and interactively
  - PyPlot essentially reproduces plotting functions and behavior of MATLAB.
- To use matplotlib with ipython on our computers:

```
ipython --matplotlib qt
```

Importing PyPlot

- We import PyPlot as we do other packages:

```
import matplotlib.pyplot as pl
```

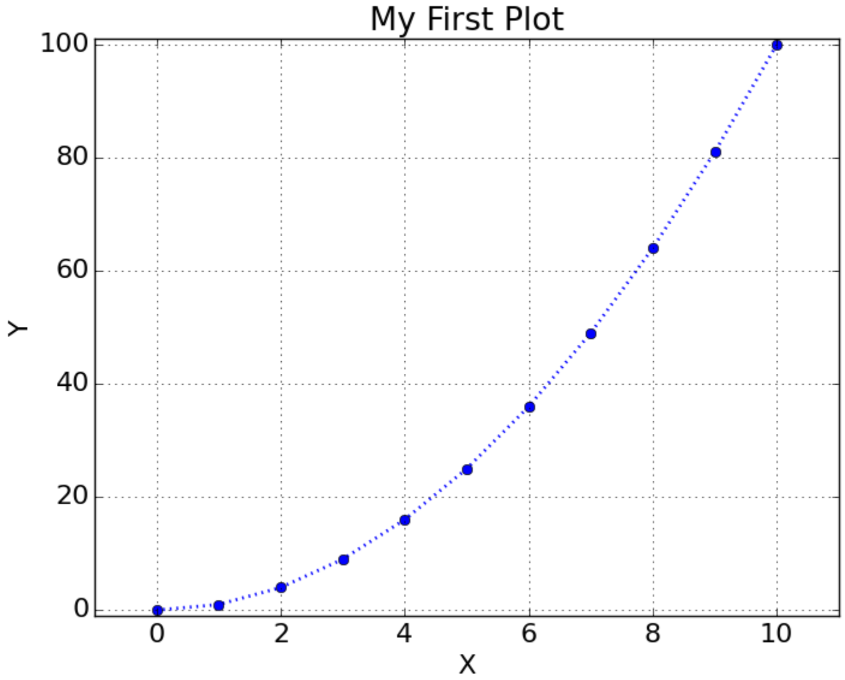
- Remember that pl above is just a shorthand for matplotlib.pyplot. This way, we can invoke PyPlot's methods easily:

```
pl.plot(X,Y)
```

- In the following slides I will show PyPlot methods with the *pl* shorthand....

Make your First PLot

```
import numpy as np
import matplotlib.pyplot as pl

# make a numpy array
X = np.linspace(0.,10.,11)

Y = X*X  # Y array is X squared

pl.ion()  # turns on interactive plotting

pl.plot(X,Y,'bo:') # plots large blue dots
                   # connected by dotted lines
pl.xlabel('X')
pl.ylabel('Y')
pl.title('My First Plot')

pl.axis([-1,11,-1,101])      # sets the dimensions

pl.grid()   # draws dotted lines on major "ticks"
```



Interactive Mode Plotting

- Interactive mode updates a plot each time a new command is issued.
  - Turn on interactive mode with method:

    pl.ion()
  - Turn off interactive mode with method:

    pl.ioff()
- When interactive mode is not on, you enter all pyplot commands and then use the method *pl.show()* to see the figure.
  - ***NB***: *pl.show()* waits for you to close the plot figure window before you can proceed.

The PyPlot Plot Method

- pl.plot(X,Y,'CLM')
  - X is X array for plot
  - Y is Y array for plot
  - X and Y must have same number of points
  - String 'clm' tells how to make the plot:
    - C indicates the color
    - L indicates the line style:

c m y r g b w k

- -- : -. omit symbol for no line

    - M indicates marker style

. + 0 * x s d ^ v > < p h

None = no symbol

Labelling the plot

- *pl.xlabel('name of x axis')* - prints a label along the x-axis
- *pl.ylabel('name of y axis')* - prints a label along the y-axis
- *pl.title('title for plot')* - writes a title across the top of the graph
- *pl.axis([xmin, xmax, ymin, ymax])* - sets limits for plot with array shown
- *pl.grid('on')* – turn on grid lines

PyPlot Figures

- Matplotlib allows you to use one or more “figures” for making graphs.
- To start plotting in a figure, we use the figure method e.g.:

  ```
  pl.figure(1)
  ```

  - In interactive mode, this opens figure 1 and shows window on screen. Ready to start accepting plot commands.
  - The figure number can be any integer > 1
- “Close” a figure when done:

  ```
  pl.close(1)  # closes figure 1
  ```

  ```
  pl.close('all') # closes ALL open figures
  ```

Multiple Figure Example

```
X = np.linspace(0.,10.,11)
Y = X*X          # Y array is X squared
Z = X*X*X       # Z array is X cubed
pl.ion()  # turns on interactive plotting

pl.figure(1)
pl.plot(X,Y,'bo:') # plots large blue dots
                   # connected by dotted lines
pl.xlabel('X')
pl.ylabel('Y')
pl.title('First Plot')

pl.figure(2)
pl.plot(X,Z,'rs-') # plots large red squares
                   # connected by solid lines
pl.xlabel('X')
pl.ylabel('Z')
pl.title('Second Plot')
```

FIGURE 1 WINDOW:

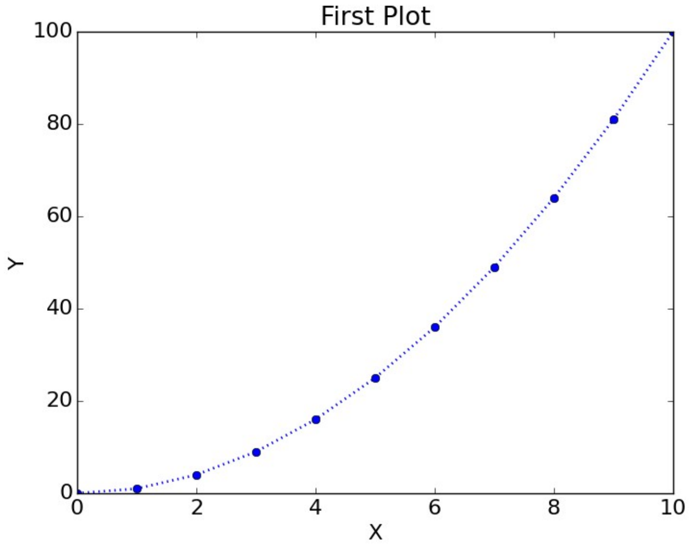
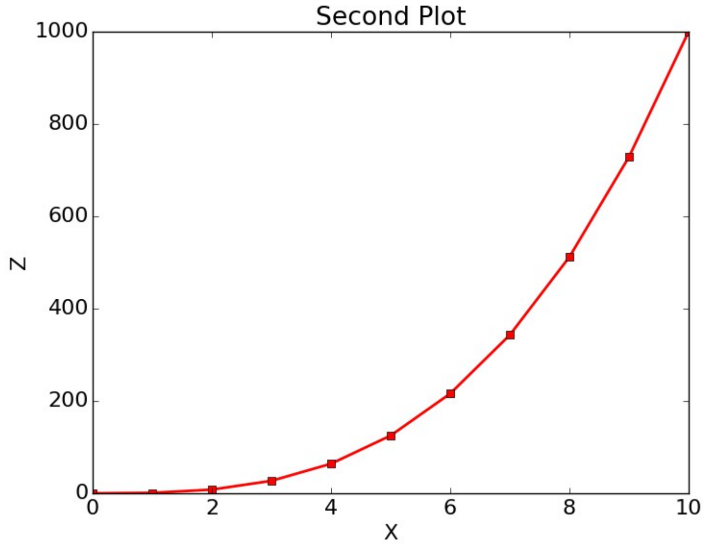


FIGURE 2 WINDOW:



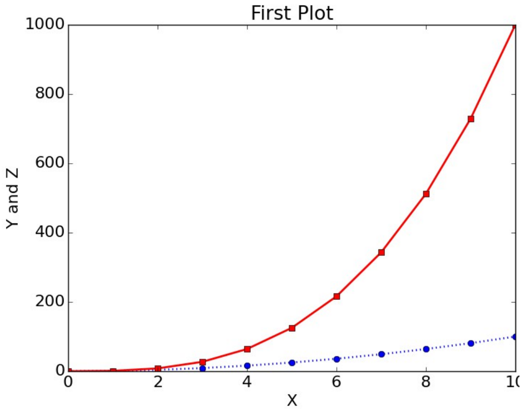
Multiple Curves on the same Plot

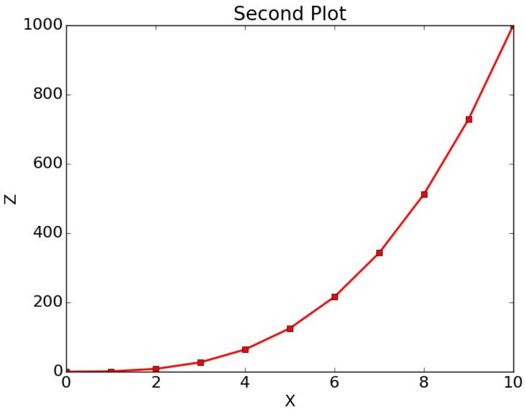
hold method

```
X = np.linspace(0.,10.,11)
Y = X*X          # Y array is X squared
Z = X*X*X        # Z array is X cubed
pl.ion()  # turns on interactive plotting

pl.figure(1)
pl.plot(X,Y,'bo:')
pl.plot(X,Z,'rs-')  # hold is 'on' by default
                    # so this line is added
pl.xlabel('X')
pl.ylabel('Y and Z')
pl.title('First Plot')

pl.hold('off')  # turn off hold for
                # second figure
pl.figure(2)
pl.plot(X,Y,'bo:')
pl.plot(X,Z,'rs-')  # this redraws without
                    # the first graph
pl.xlabel('X')
pl.ylabel('Z')
pl.title('Second Plot')
```

The hold method allows us to control whether a call to the plot method will redraw the graph.

By default hold is 'on' when we start up, so additional lines will be added to the graph.





Some Plotting Guidelines

- Keep it simple and neat
  - first priority is to convey information.
  - Results count -- not by fancy fonts and colors.
- Be Honest
  - Show all the data
  - Show the errors
- Always Label Axes
  - Remember units!
  - Include Legends
  - Include Titles
- Fewer curves is better.
- Think Big:
  - Big Labels
  - Big Points
  - Big Lines

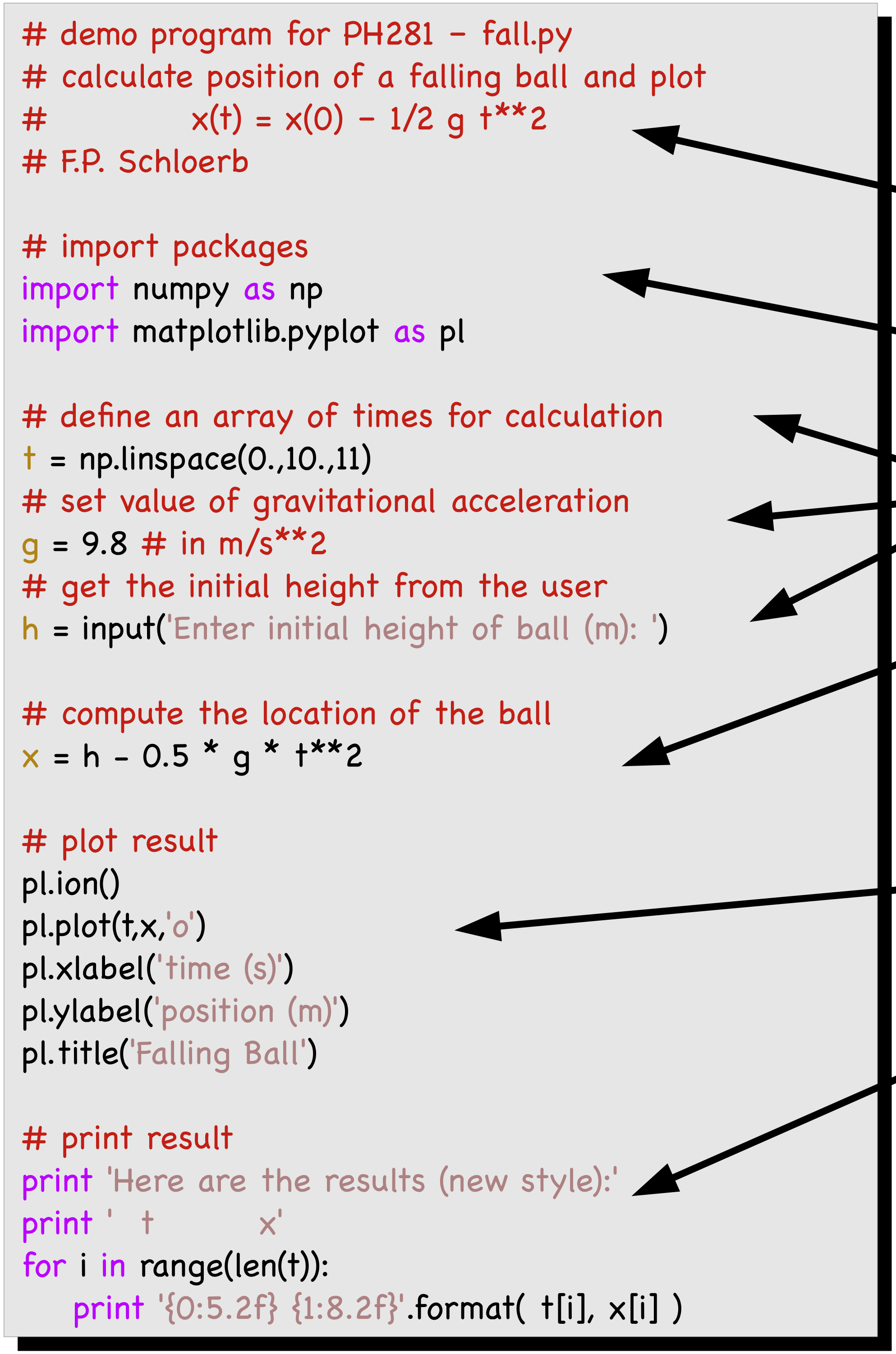
First Steps in Programming

Good programs will...

- Give correct answers.
- Be clear and easy to read. Action of each part should be easy to analyze.
- Document itself for the sake of readers and programmers.
- Be easy to use.
- Be built up out of small programs that can be independently verified.
- Be easy to modify and robust enough to keep giving correct answers after modification.
- Document the data formats used.
- Use trusted libraries.
- Be published or passed on to others to use or develop.

From Landau et al “Computational Physics: Problem Solving with Python”

Example



```
# demo program for PH281 - fall.py
# calculate position of a falling ball and plot
#             x(t) = x(0) - 1/2 g t**2
# F.P. Schloerb

# import packages
import numpy as np
import matplotlib.pyplot as pl

# define an array of times for calculation
t = np.linspace(0.,10.,11)
# set value of gravitational acceleration
g = 9.8 # in m/s**2
# get the initial height from the user
h = input('Enter initial height of ball (m): ')

# compute the location of the ball
x = h - 0.5 * g * t**2

# plot result
pl.ion()
pl.plot(t,x,'o')
pl.xlabel('time (s)')
pl.ylabel('position (m)')
pl.title('Falling Ball')

# print result
print 'Here are the results (new style):'
print '  t          x'
for i in range(len(t)):
    print '{0:5.2f} {1:8.2f}'.format( t[i], x[i] )
```

Comments about Script

Import Pacakges

Initialize Parameters

Calculate

Display Results

Plot Graph

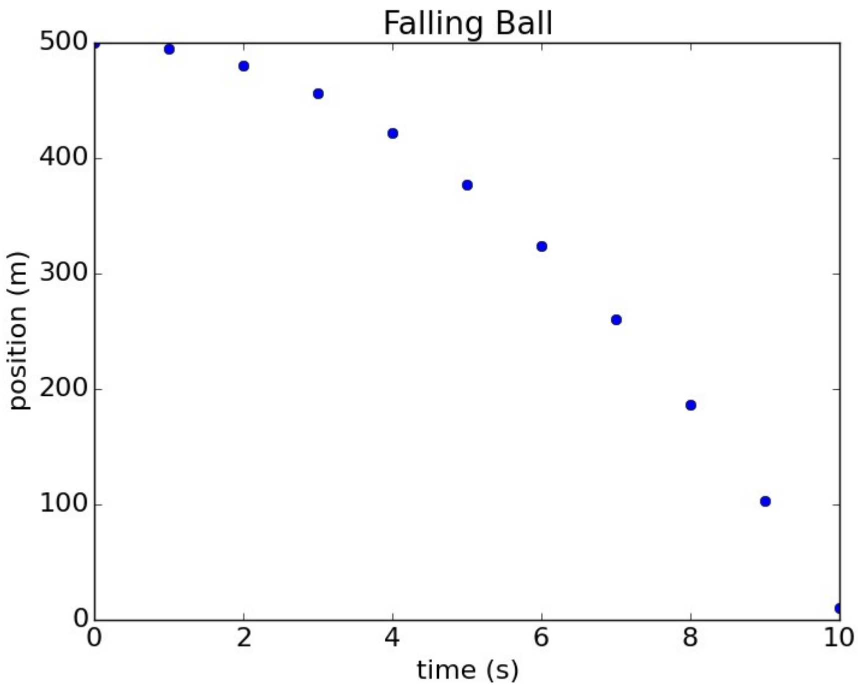
Print Table

NB: When you run this script, ipython will then have *np* and *pl* loaded and t, g, h, and x will all be defined!

Running *fall.py*

```
In [1]: run fall.py
Enter initial height of ball (m):
500
Here are the results (new style):
  t         x
 0.00   500.00
 1.00   495.10
 2.00   480.40
 3.00   455.90
 4.00   421.60
 5.00   377.50
 6.00   323.60
 7.00   259.90
 8.00   186.40
 9.00   103.10
10.00    10.00

In [2]:
```

User entered this number



Comments

- Comments are useful for explaining what your program is doing, both to yourself and to others.
- In Python, comments follow the **#** character.
- Examples:

```
# this is a comment line

x = y * 2 # this comment follows a statement
```

Comments about Comments

- Use comments liberally
  - Others (e.g. graders) won't know what you are doing without comments.
  - You won't remember what you did at some point in the future.
- Comments must be useful. Consider...

```
# initialize t

t = np.linspace(0.,10.,101)
```

Versus

```
# initialize array of times for calculation; time in s

t = np.linspace(0.,10.,101)
```

Python Docstrings

- Docstrings provide a way to document your modules and scripts so that they can use “help” command easily.
- A standard way to document things.
- Function Example

```
def myfunct(x):
    “””This is a demo function docstring
    “””
    print x # do something in the function
```

Docstrings Example

Module itest.py

```
"""itest module contains test functions for PH281 Demonstrations

"""
import numpy as np

def testfn(x):
    """prints argument
    """
    print x

def testfn2(x,n):
    """multiplies numpy array by a factor of n

    Args:
        x : numpy array
        n : factor for multiplication
    Returns:
        numpy array with x multiplied by n
    """
    return(x*n)
```

Result of running help(itest):

```
Help on module itest:

NAME
    itest - itest module contains test functions for PH281 Demonstrations

FILE
    /Users/schloerb/PH281/PH281F15/Programs/itest.py

FUNCTIONS
    testfn(x)
        prints argument

    testfn2(x, n)
        multiplies numpy array by a factor of n

        Args:
            x : numpy array
            n : factor for multiplication
        Returns:
            numpy array with x multiplied by n
```