Computational Physics/ Structured Programming

Lecture IV: Graphics & Visualization

Dr. Ian Kaniu

Department of Physics
University of Nairobi

May 2024

Outline of the Lecture

Outline of the Lecture

1. Graphs
2. Scatter Plots
3. Density Plots
4. Vpython

Outline of the Lecture

1. Graphs
2. Scatter Plots
3. Density Plots
4. Vpython

Outline of the Lecture

1. Graphs
2. Scatter Plots
3. Density Plots
4. Vpython

Graphics & Visualization

- So far we have created programs that print out words and numbers, but often we will also want our programs to produce graphics, meaning pictures of some sort.
- There are two main types of computer graphics used in physics.
  1) Most common of scientific visualizations, the graph: a depiction of numerical data displayed on calibrated axes.
  2) Scientific diagrams and animations: depictions of the arrangement or motion of the parts of a physical system, which can be useful in understanding the structure or behavior of the system.

Graphs

Graphs

- A number of Python packages include features for making graphs.
- One such package which is powerful, easy-to-use, and popular is pylab.
- `Pylab` contains features for generating graphs of many different types. We will concentrate of three types that are especially useful in physics: ordinary line graphs, scatter plots, and density (or heat) plots.

Graphs

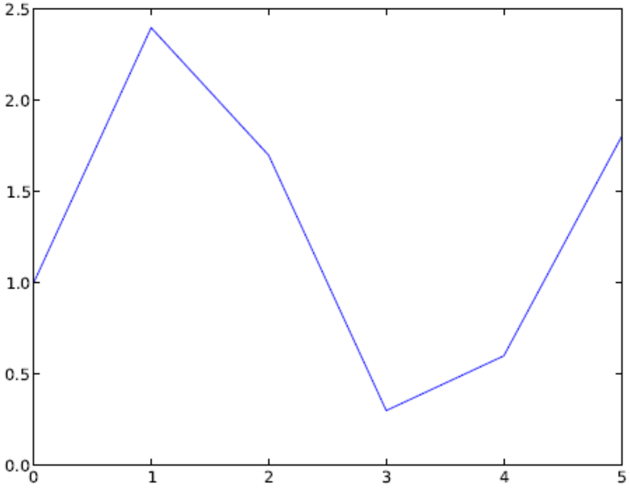
- To create an ordinary graph in Python we use the functions `plot` and `show` from the `pylab` package.

```
from pylab import plot,show
y = [ 1.0, 2.4, 1.7, 0.3, 0.6, 1.8 ]
plot(y)
show()
```

- After importing the two functions from pylab, we create the list of values to be plotted, create a graph of those values with `plot(y)`, then display that graph on the screen with `show()`.

Graphs

If we run the program above, it produces a new window on the screen with a graph in it like this:



Graphs

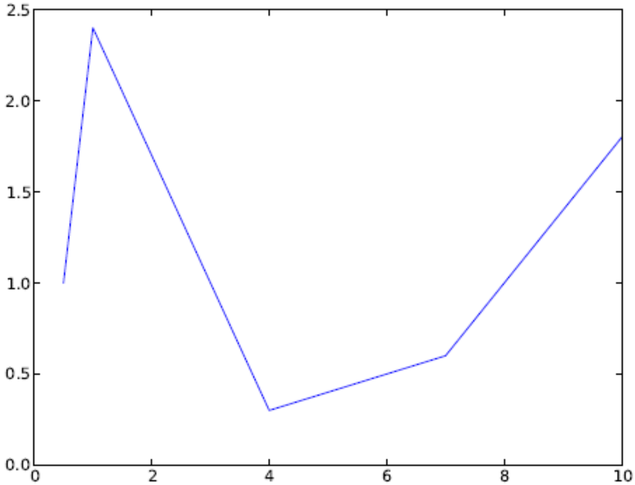
- When we want to specify both the x- and y-coordinates for the points in the graph, we use a plot statement with two list arguments, thus:

```
from pylab import plot,show
x = [ 0.5, 1.0, 2.0, 4.0, 7.0, 10.0 ]
y = [ 1.0, 2.4, 1.7, 0.3, 0.6, 1.8 ]
plot(x,y)
show()
```

- The two lists must have the same number of entries, as here. If they do not, you'll get an error message and no graph.
- Once the graph is displayed on the screen you can do other things with it i.e. zoom in on portions of the graph, move your view around the graph, or save the graph as an image file on your computer.

Graphs

If we run the program above, it produces a new window on the screen with a graph in it like this:



Graphs

- Let us apply the plot and show functions to the creation of a slightly more interesting graph, a graph of the function *sinx* from $x = 0$ to $x = 10$.
- To do this we first create an array of the x values, then we take the sines of those values to get the y-coordinates of the points:

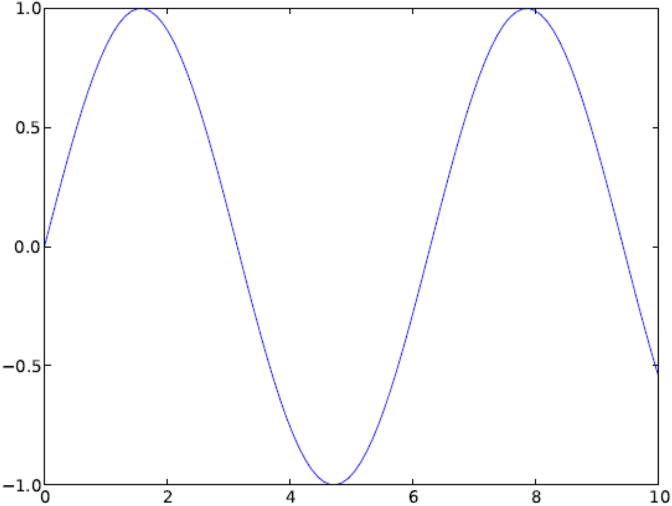
```
from pylab import plot,show
from numpy import linspace,sin

x = linspace(0,10,100)
y = sin(x)
plot(x,y)
show()
```

- The `linspace` function from numpy (see Handout I, section 2.5!) is used to generate the array of x-values.
- The `sin` function from `numpy` , which is a special version of sine that works with arrays—it just takes the sine of every element in the array (Alternatively we could have used the ordinary `sin` function from the `math` package).

Graphs

- If we run the program above, it produces a new window on the screen with a graph in it like this:



Graphs

- Notice that our plot consists of a finite set of points - a hundred of them in this case - and the computer draws straight lines joining these points.
- So the end result is not actually curved; it's a set of straight-line segments.
- This is a useful and widely used trick for making curves in computer graphics: choose a set of points spaced close enough together that when joined with straight lines the result looks like a curve even though it really isn't.

Graphs

- Suppose we have some experimental data in a computer file `values.txt`, stored in two columns, like this:

0	12121.71
1	12136.44
2	12226.73
3	12221.93
4	12194.13
5	12283.85
6	12331.6
7	12309.25
...	

Graphs

- We can make a graph of these data as follows:

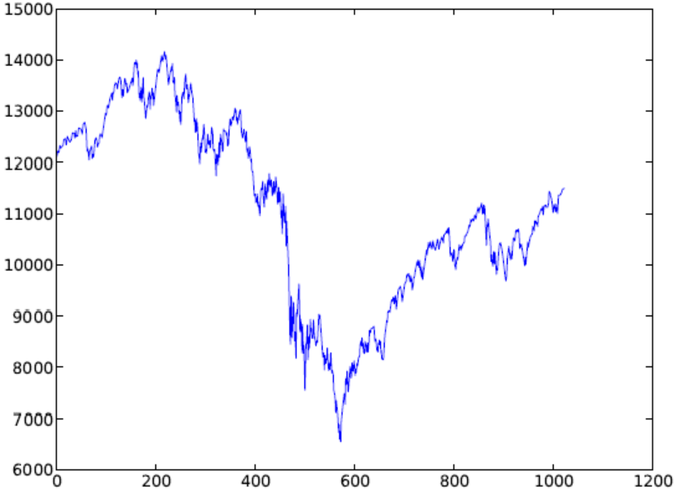
```
from numpy import loadtxt
from pylab import plot,show

data = loadtxt("values.txt",float)
x = data[:,0]
y = data[:,1]
plot(x,y)
show()
```

- The `loadtxt` function from `numpy` (see Handout I, section 2.4.3) is used to read the values in the file and put them in an array.
- The Python's array slicing facilities is used to extract the first and second columns of the array and put them in separate arrays x and y for plotting.

Graphs

If we run the program above, it produces a new window on the screen with a graph in it like this:



Graphs

- You can also vary the style in which the computer draws the curve on the graph. To do this a third argument is added to the plot function:

```
plot(x,y,"g--")
```

- The first letter of the string tells the computer what color to draw the curve with.
- Allowed letters are *r*, *g*, *b*, *c*, *m*, *y*, *k*, and *w*, for red, green, blue, cyan, magenta, yellow, black, and white, respectively.
- The remainder of the string says what style to use for the line.
- There are many options, i.e "-" for a solid line (like the ones we've seen so far), "--" for a dashed line, "o" to mark points with a circle, and "s" to mark points with a square.

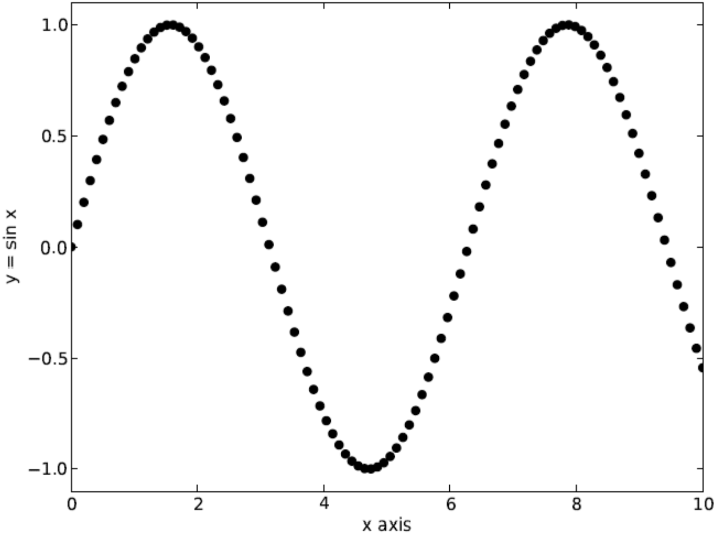
Graphs

- For example, the modification below plots our sine wave as a set of black circular points:

```
plot(x,y,"ko")
ylim(-1.1,1.1)
xlabel("x axis")
ylabel("y = sin x")
show()
```

Graphs

If we run the program above, it produces a new window on the screen with a graph in it like this:



Graphs

- We will often need to plot more than one curve or set of points on the same graph. This can be achieved by using the `plot` function repeatedly.

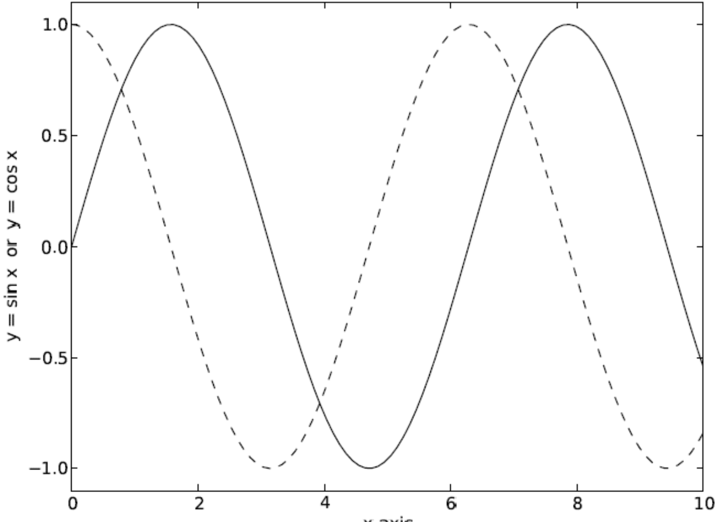
```
from pylab import plot,ylim,xlabel,ylabel,show
from numpy import linspace,sin,cos

x = linspace(0,10,100)
y1 = sin(x)
y2 = cos(x)
plot(x,y1,"k-")
plot(x,y2,"k-")
ylim(-1.1,1.1)
xlabel("x axis")
ylabel("y = sin x or y = cos x")
show()
```

- The program plots both the sine function and the cosine function on the same graph, one as a solid curve, the other as a dashed curve.

Graphs

If we run the program above, it produces a new window on the screen with a graph in it like this:



Class Activity 5: 20 Mins

Group Work (Plotting Experimental Data)

:

You have been given a file called `sunspots.txt`, which contains the observed number of sunspots on the Sun for each month since January 1749. The file contains two columns of numbers, the first being the month and the second being the sunspot number.

a) Write a program that reads in the data and makes a graph of sunspots as a function of time.

b) Modify your program to display only the first 1000 data points on the graph.

c) Modify your program further to calculate and plot the running average of the data, defined by Y_k

$$Y_k = \frac{1}{2r} \sum_{m=-r}^{r} y_{k+m}$$

where $r = 5$ in this case (and the y_k are the sunspot numbers). Have the program plot both the original data and the running average on the same graph, again over the range covered by the first 1000 data points.

Scatter Plots

Scatter Plots

- In an ordinary graph, such as those of the previous section, there is one independent variable, usually placed on the horizontal axis, and one dependent variable, on the vertical axis.
- The graph is a visual representation of the variation of the dependent variable as a function of the independent one.
- In other cases, however, we measure or calculate two dependent variables.
- A classic example in physics is the temperature and brightness - also called the magnitude of stars.
- Typically we might measure temperature and magnitude for each star in a given set and we would like some way to visualize how the two quantities are related.

Scatter Plots

- A standard approach is to use a*scatter plot*, a graph in which the two quantities are placed along the axes and we make a dot on the plot for each pair of measurements, i.e., for each star in this case.

Scatter Plots

- `Pylab` provides the function `scatter`, which is designed specifically for making scatter plots.
- It works in a similar fashion to the `plot` function: you give it two lists or arrays one containing the x-coordinates of the points and the other containing the y-coordinates, and it creates the corresponding scatter plot:
  ```
  scatter(x,y)
  ```
- You do not have to give a third argument telling scatter to plot the data as dots - all scatter plots use dots automatically but to display it you need to use the function `show`.

Scatter Plots

- Suppose, for example, that we have the temperatures and magnitudes of a set of stars in a file called `stars.txt` on our computer, like this:

4849.4	5.97
5337.8	5.54
4576.1	7.72
4792.4	7.18
5141.7	5.92
6202.5	4.13
...	

Scatter Plots

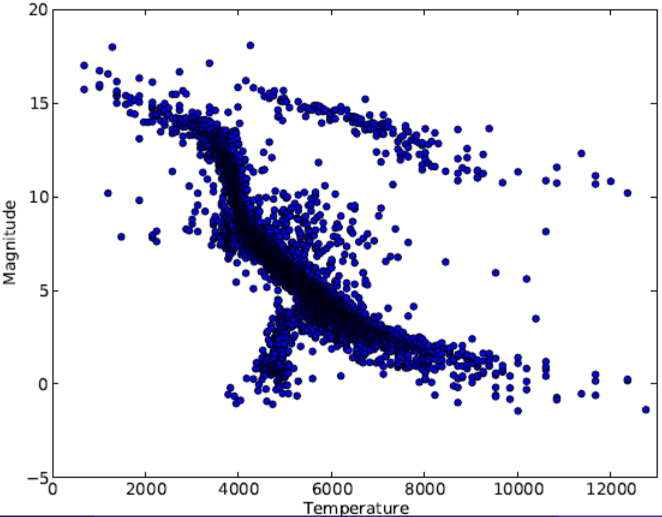
- The first column is the temperature and the second is the magnitude.
- Here's a Python program to make a scatter plot of these data:

```
# Plot of the Hertzsprung-Russell diagram
from pylab import scatter,xlabel,ylabel,xlim,ylim,show
from numpy import loadtxt

data = loadtxt("stars.txt",float)
x = data[:,0]
y = data[:,1]
scatter(x,y)
xlabel("Temperature")
ylabel("Magnitude")
xlim(0,13000)
ylim(-5,20)
show()
```

Scatter Plots

If we run the program above, it produces a new window on the screen with a scatter plot in it like this:



Scatter Plots

- `xlabel` and `ylabel` were used to label the temperature and magnitude axes, and `xlim` and `ylim` to set the ranges of the axes.
- You can also change the size and style of the dots and many other things, as well as use scatter two or more times in succession to plot two or more sets of data on the same graph, etc.
- See the on-line manual at `matplotlib.org` for more details.

Density Plots

Density Plots

- Two-dimensional data are harder to visualize on a computer screen than the one-dimensional lists of values that go into an ordinary graph.
- One tool that is helpful in many cases is the *density plot*, a two-dimensional plot where color or brightness is used to indicate data values.
- In Python density plots are produced by the function `imshow` from `pylab`.

Density Plots

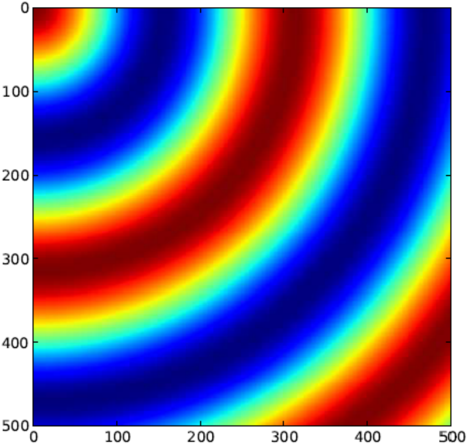
- Here's the program that produces a density plot:

```
from pylab import imshow,show
from numpy import loadtxt

data = loadtxt("circular.txt",float)
imshow(data)
show()
```

Density Plots

If we run the program above, it produces a new window on the screen with a density plot in it like this:



Density Plots

- The file circular.txt contains a simple array of values, like this:

```
0.0050 0.0233 0.0515 0.0795 0.1075 ...
0.0233 0.0516 0.0798 0.1078 0.1358 ...
0.0515 0.0798 0.1080 0.1360 0.1639 ...
0.0795 0.1078 0.1360 0.1640 0.1918 ...
0.1075 0.1358 0.1639 0.1918 0.2195 ...
  ...    ...    ...    ...    ...
```

- The program reads the values in the file and puts them in the two-dimensional array data using the `loadtxt` function, then creates the density plot with the `imshow` function and displays it with show.
- The computer automatically adjusts the color-scale so that the picture uses the full range of available shades.

Density Plots

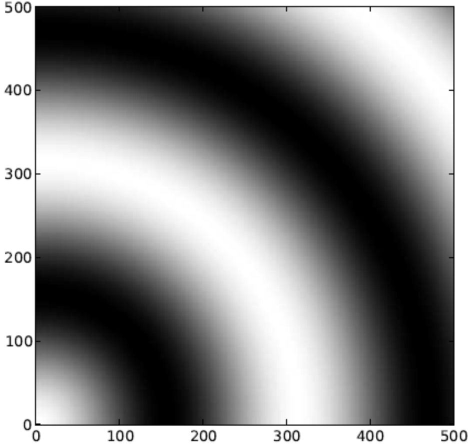
- Density plots with this particular choice of colors from blue to red (or similar) are sometimes called *heat maps*, because the same color scheme is often used to denote temperature, with blue being the coldest temperature and red being the hottest.
- To change to gray-scale, for instance, you use the function gray, which takes no arguments:

```
from pylab import imshow,gray,show
from numpy import loadtxt

data = loadtxt("circular.txt",float)
imshow(data,origin="lower")
gray()
show()
```

Density Plots

The gray color scheme, which runs from black for the lowest values to white for the highest:



Density Plots

- `Pylab` provides many other color schemes, which you may find useful occasionally.
- A complete list, with illustrations, is given in the on-line documentation at `matplotlib.org`, but here are a few that might find use in physics:

`jet`	The default heat-map color scheme
`gray`	Gray-scale running from black to white
`hot`	An alternative heat map that goes black-red-yellow-white
`spectral`	A spectrum with 7 clearly defined colors, plus black and white
`bone`	An alternative gray-scale with a hint of blue
`hsv`	A rainbow scheme that starts and ends with red

Reading Material: "3D Graphics" - Handout will be uploaded together with the lecture

Vpython

Visualization Using `vpython`

- *VPython*, the nickname for Python plus the `Visual` package, is particularly useful for creating 3-D solids, 2-D plots, and animations.

Visualization Using vpython

- Here's a code that produces two plots using vpython:

```
from vpython import *            # Import Vpython
graph1=graph(align='left',width=400, height=400,
  background=color.white,foreground=color.black)
   Plot1=gcurve(color=color.red)  # gcurve method
for x in arange(0,8.1,0.1):    # x range
      Plot1.plot(pos=(x,5*cos(2*x)*exp(-0.4*x)))
 graph2=graph(align='right',width=400, height=400,
  background=color.white,foreground=color.black,
title='2-D Plot', xtitle='x',ytitle='f(x)')
   Plot2=gdots(color=color.black)  # Dots
for x in arange(-5,5,0.1):
Plot2.plot(pos=(x,cos(x)))  # plot dots
```

Visualization Using `vpython`

- Notice that the plotting technique with VPython is to create first a plot object, and then to add the points one at a time to the object.
- In contrast, `matplotlib.org` creates a vector of points and plots the entire vector in one fell swoop.
- Here's a link where you can find more vpython examples with animations: https://www.glowscript.org/#/user/GlowScriptDemos/folder/Examples/

Visualization Using vpython

Please go through the tutorial videos presented in the following links

- https://www.youtube.com/watch?v=vEMCiugDnKI&list=PLdCdV2GBGyXOnMaPS1BgO7IOU_OOApuMo
- https://www.youtube.com/watch?v=aPnZ4TIUnO8&list=PLdCdV2GBGyXOnMaPS1BgO7IOU_OOApuMo&index=2
- https://www.youtube.com/watch?v=IbRo-_sTUvQ&list=PLdCdV2GBGyXOnMaPS1BgO7IOU_OOApuMo&index=3
- https://www.youtube.com/watch?v=Y2snMyfgDuo&list=PLdCdV2GBGyXOnMaPS1BgO7IOU_OOApuMo&index=4

Summary of Lecture

Questions and Class Discussions:

- Graphs
- Scatter Plots
- Density Plots
- 3D Plots & vPython

These lecture notes will be uploaded in SOMAS

- Post your questions, comments and suggestions in the **Forum Discussion Page** in SOMAS.