Introduction to Scientific Computing
Lecture 6

Professor Hanno Rein

Last updated: October 15, 2016

6.4 Cubic spline interpolation

Instead of going to higher and higher order, there is another way of creating a smooth function that interpolates data-points. A cubic spline is a piecewise continuous curve that passes through all of the values of a given dataset. This works particularly well for smooth datasets with no noise. Each of the piecewise curves is a cubic polynomial with coefficients a_i, b_i, c_i and d_i:

$$S_i(x) = a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + d_i \qquad \text{for } x \in [x_i, x_{i+1}]$$

If we have N data-points, there are $N-1$ intervals, hence $(N-1)\cdot 4$ coefficients that we need to find. Two conditions in each interval arise because we have to match the two data-points at each end.

$$S_i(x_i) = y_i, \quad S_i(x_{i+1}) = y_{i+1}$$

How about the other two parameters? We want to have a smooth function! Thus, we require that the derivatives of the piecewise functions match at the interval boundaries:

$$S'_{i-1}(x_i) = S'_i(x_i), \quad S''_{i+1}(x_i) = S''_i(x_i)$$

We now have almost as many conditions as we have free parameters, except at the boundaries.What could we possibly do there? There are multiple choices and it depends on the problem. We use one called *natural* boundary condition which says that we set the second derivatives to zero at the boundary.

As you can probably guess, this set of equations that we are generating will become a matrix equation. Let's go through the individual steps. Finding the value for the d_is is simple. Our requirement gives us

$$d_i = S_i(x_i) = y_i$$

The condition to match the point at $i+1$ gives

$$S_i(x_{i+1}) = a_i(x_{i+1}-x_i)^3 + b_i(x_{i+1}-x_i)^2 + c_i(x_{i+1}-x_i) + d_i = y_{i+1}$$

Let's call the derivatives at point i, D_i, i.e.

$$S'_i(x_i) = D_i = c_i$$

and therefore

$$S'_i(x_{i+1}) = D_{i+1} = 3a_i(x_{i+1}-x_i)^2 + 2b_i(x_{i+1}-x_i) + D_i$$

We can now setup an equation system for a, b, c and d:

$$\begin{aligned}
a_i &= (D_{i+1} + D_i)(x_{i+1} - x_i)^{-2} - 2(y_{i+1} - y_i)(x_{i+1} - x_i)^{-3} \\
b_i &= (-D_{i+1} - 2D_i)(x_{i+1} - x_i)^{-1} + 3(y_{i+1} - y_i)(x_{i+1} - x_i)^{-2} \\
c_i &= D_i \\
d_i &= y_i
\end{aligned}$$

We have one requirement left to play with, to match second derivatives at each interval. This gives us:

$$\begin{aligned}
S''_{i-1}(x_i) &= S''_i(x_i) \\
S''_i(x_{x+i}) &= S''_{i+1}(x_{i+1})
\end{aligned}$$

which equates to

$$6a_i(x_{i+1} - x_i) + 2b_i = 2b_{i+1}$$

Let us now combine this with the earlier equation to get

$$\begin{aligned}
3(y_i - y_{i-1})(x_i - x_{i-1})^{-1}(x_{i+1} - x_i) + 3(y_{i+1} - y_i)(x_{i+1} - x_i)^{-1}(x_i - x_{i-1}) \\
= D_{i-1}(x_{i+1} - x_i) \\
+D_i(3(x_{i+1} - x_i) + (x_i - x_{i-1})) \\
+D_{i+1}(x_i - x_{i-1})
\end{aligned}$$

Let us define

$$Y_i \equiv 3\frac{y_i - y_{i-1}}{x_i - x_{i-1}}(x_{i+1} - x_i) + 3\frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x_i - x_{i-1})$$

We can write this as a matrix equation

$$\begin{pmatrix}
\ddots & & & \\
& (x_{i+1} - x_i) & (3x_{i+1} - 2x_i - x_{i-1}) & (x_i - x_{i-1}) & \\
& & & & \ddots
\end{pmatrix} \cdot \begin{pmatrix} \vdots \\ D_i \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ Y_i \\ \vdots \end{pmatrix}$$

At the end points, we have not enough information to fully determine all variables. We therefore come up with new requirements there, let the second derivatives be zero.

$$\begin{aligned}
S''_0(x_0) &= 2b_0 = 0 \\
S''_{N-2}(x_{N-1}) &= 6a_{N-2}(x_{N-1} - x_{N-2}) + 2b_{N-2} = 0
\end{aligned}$$

This gives in terms of the D coefficients:

$$\begin{aligned}
2D_0 + D_1 &= 3\frac{y_1 - y_0}{x_1 - x_0} \equiv Y_0 \\
D_{N-2} + 2D_{N-1} &= 3\frac{y_{N-1} - y_{N-2}}{x_{N-1} - x_{N-2}} \equiv Y_{N-1}
\end{aligned}$$

We can now complete the matrix from above to

$$\begin{pmatrix}
2 & 1 & 0 & \cdots & & & \\
(x_2-x_1) & (3x_2-2x_1-x_0) & (x_1-x_0) & 0 & \cdots & & \\
0 & (x_3-x_2) & (3x_3-2x_2-x_1) & (x_2-x_1) & 0 & \cdots & \\
 & & & \ddots & & & \\
 & & (x_{i+1}-x_i) & (3x_{i+1}-2x_i-x_{i-1}) & (x_i-x_{i-1}) & & \\
 & & & \ddots & & & \\
 & & & & 0 & 1 & 2
\end{pmatrix}$$

$$\cdot \begin{pmatrix} D_0 \\ \vdots \\ D_i \\ \vdots \\ D_{N-1} \end{pmatrix} = \begin{pmatrix} Y_0 \\ \vdots \\ Y_i \\ \vdots \\ Y_{N-1} \end{pmatrix}$$

This is a tridiagonal system and can easily be solved. Here, we just use the Gaussian eliminiation that we already know. It is easy to find more efficient ways to solve it, but we don't bother.

Once solved for the D_is, we can solve for the a_i, b_i, c_i and d_is. This then defined all parameters for the piecewise cubic function.

In the following figure, we apply this method to the average temperature in Toronto. As you can see, it gives a very smooth and reasonable fit.

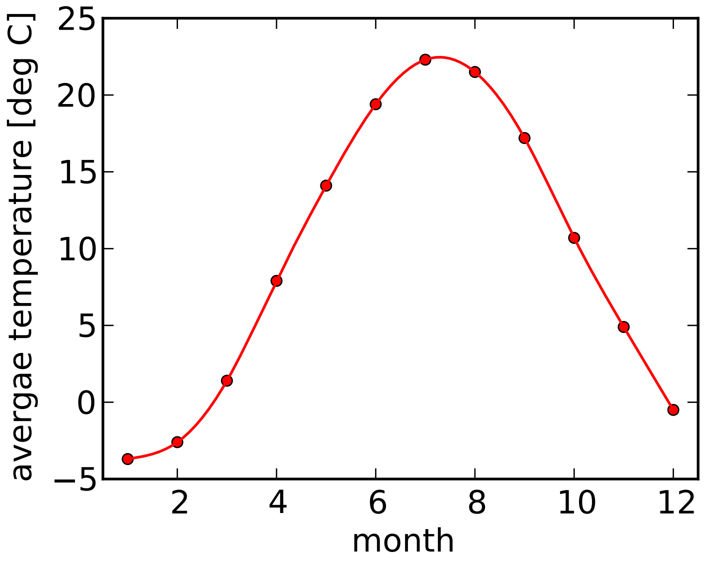

Figure 1: Temperature in Toronto. Cubic spline interpolation.

One way to further improve this spline is to take advantage of the fact that the temperature is periodic. This removes the extra criteria at the boundaries. The matrix will get some extra components (i.e. is not tridiagonal anymore).

Now, we have this great spline interpolation method. Can you use it for any problem? No. Here's a word of caution. Look at the following dataset.

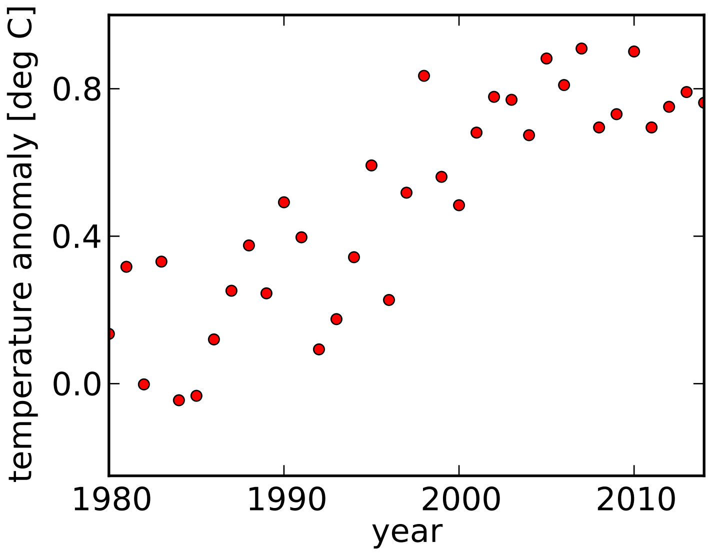

Figure 2: Global temperature anomaly. Source: Climatic Research Unit (University of East Anglia).

The above figure shows the global temperature anomaly. This is an indication of climate change. Clearly, there is a lot of noise in the data. Nevertheless, one can see a very dominant trend towards higher temperatures. We could just fit a spline to this curve. The result is shown in the following figure.

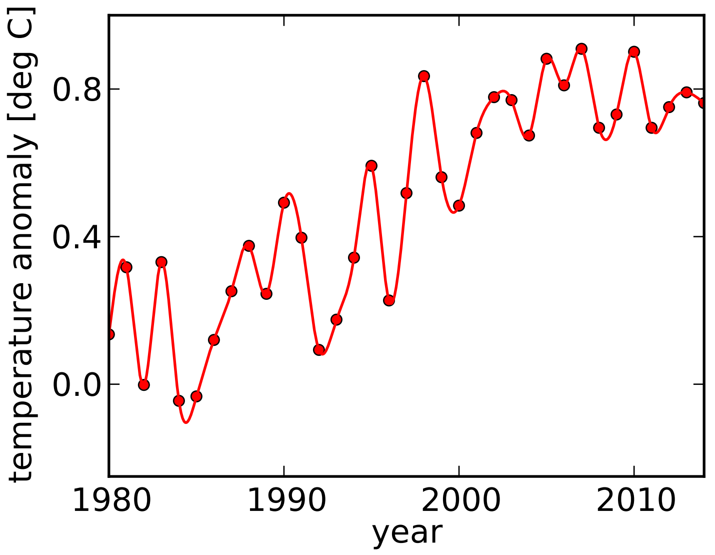

Figure 3: Global temperature anomaly fitted with a cubic spline.

Although the spline is clearly going through all data-points and the curve is smooth at every point in the interval, it is not a good indication of the trend. Why is that? We have fitted a smooth function to noisy data. It's the same issue that we encountered earlier and is sometimes called *ringing* or *Runge's phenomenon*.

Thus, interpolating this data with a constant or piece-wise linear function or even a cubic spline does not make much sense. We would effectively try to interpolate the noise, not the data. So how can we interpolate this data in a meaningful way. We can use a least square fit!

For example, a straight line fit will give us an indiction of how fast the temperature anomaly has risen over time. We can even use it to extrapolate how much temperatures will be rising in the future. The idea of a straight line fit is to find a linear function

$$f(x) = a_1 + a_2 x$$

which is the *best* fit to the data (see our earlier discussion on least square fits). Figure 4 shows the straight line for for the climate data discussed above.

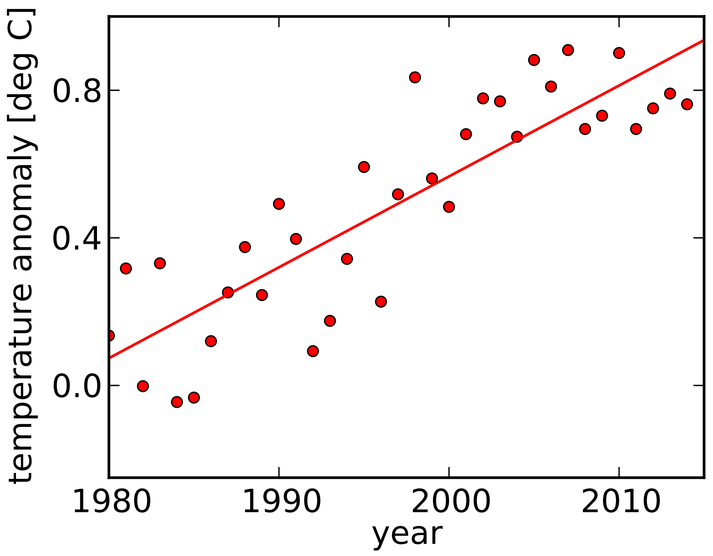

Figure 4: Global temperature anomaly with a straight line fit.

7 Plotting

In this lecture we'll talk about how to make plots in python. To start, here is a quote from wikipedia that describes what we mean by a *plot*:

> A plot is a graphical technique for representing a data set, usually as a graph showing the relationship between two or more variables. [...] Graphs are a visual representation of the relationship between variables, very useful for humans who can quickly derive an understanding which would not come from lists of values.

This lecture is not only about how to actually make a plot in python. It is also about what the practical and sometimes even philosophical choices you make when creating a plot.

7.1 Matplotlib

In this course do all the plotting with matplotlib. Matplotlib is a python module. There are many ways to achieve similar results with different packages.

A module is basically a big piece of code that we can use to do a certain task (here: plotting) without having to write everything ourselves. There are many completely free modules available for python. This is one reasons why python is so popular. In this course we won't use many modules because one of the goals of the course is to teach you numerical algorithms, not so much how to use packages. However, if you want to solve a problem as quickly as possible, there is a good chance that you'll be using a lot of modules. Rarely, you'll have to start completely from scratch.

7.2 1D Data

The simplest dataset only contains a one dimensional series of numbers. To plot one dimensional data on a two dimensional surface in a meaningfull way, you can make use of histograms. In a histogram, the area in the rectangle is proportional to the occurance rate of the variable. The width of each rectangle is the class interval. Let us start with a simple example.

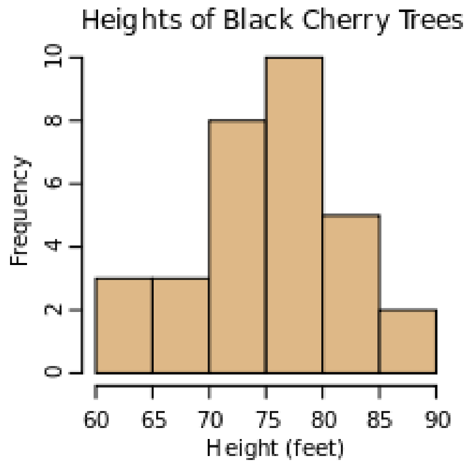

Figure 5: A simple histogram (image from wikipedia).

The above figure shows the distribution of heights of cherry trees. Note that the only meassured quantity is "height". The y axis is labeled frequency and lists the number of trees in a given "bin". Note that this allows the viewer to very quickly get an idea of the average height of a tree (75m) as well as the spread around the average (10m).

One can choose the width of the bins arbitrarily. In the above example, the width of a bin is 5 meter. However, the limiting case where we have very few, say one, bin is clearly not very useful. The other limiting case, where we have so many bins that each bin only contains one datapoint (here, one tree) is also not desirable.

Let us now finally produce our first plot in python. The code is listed in full below. First, we import the matplotlib module. This is done with the import statement. Then, we specify the output format that we want. There are many options (png, jpg, LaTeX). We will use the pdf format. Then, we import yet another part of matplotlib, pyplot. This module includes the functions that allow us to actually plot something.

We then create a figure and add a subplot it it. The data is hard coded for our first example, just a series of numbers in an array (you already know how to read in data from a file). We can create the plot with the `hist` command. Note that we give it the number of bins as an argument. We then save the plot to a file. You could also display it on the screen instead, by using the `show()` command and ommitting the line with `use('pdf')` further up.

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
x = [0.1,0.2,0.12,0.02,0.14,0.2,0.18,0.09,0.11,0.1,0.01,0.1]
numBins = 5
ax.hist(x,numBins)
plt.savefig("plot.pdf")
```

Code 1: Matplotlib example 1.

There are many other options to pass to the `hist` function. We won't discuss them here but you can find them easily online.

The result is the following plot:

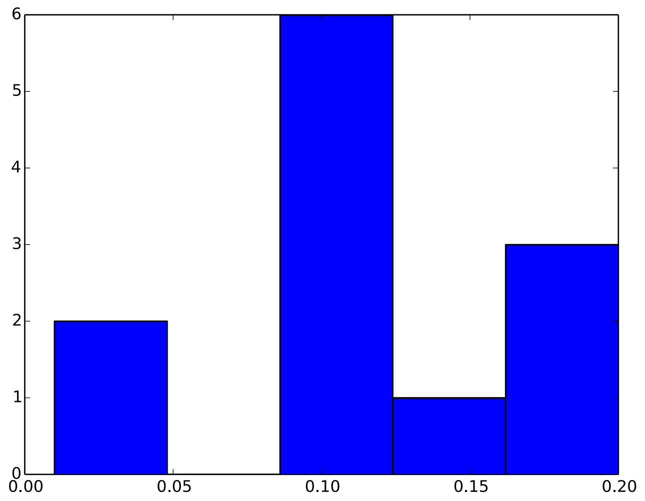

Figure 6: Histogram example using matplotlib.

7.3 2D Data

In the first example of a plot containing two dimensional data, we simply plot a curve whose joining three points. The points are given as two arrays, x and y. Both are of length 3. The complete script is only five lines long and looks like this:

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
plt.plot([0.1,1,1.9],[2.9,4.4,4.9])
plt.savefig("plot.pdf")
```

Code 2: Matplotlib example 1.

The above script produces a pdf file that is shown in the next figure. Note that there are many properties of the plot which are now set to default, but which you might want to change. For example, the colour of the line, the size of the plot, the labels on the axis, etc.

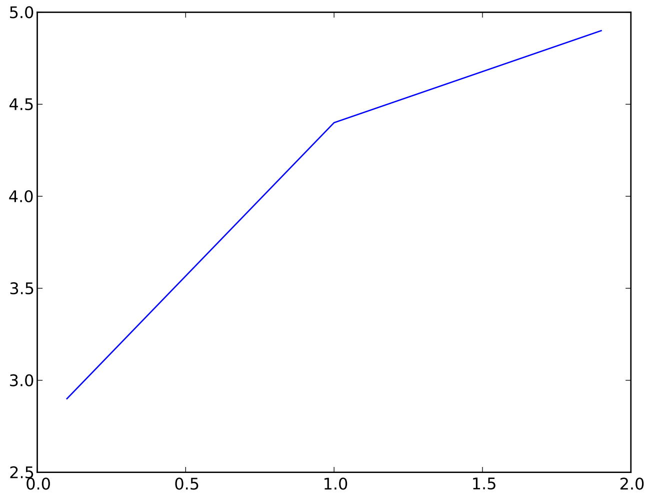

Figure 7: Plot from matplotlib example 1.

Our first example joined the data-points with a line. We can also plot only the actual data-points without joining them by adding an additional option to the plot command.

```python
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
plt.plot([0.1,1,1.9],[2.9,4.4,4.9],'ro')
plt.savefig("plot.pdf")
```

Code 3: Matplotlib example 2.

The output now looks like this:

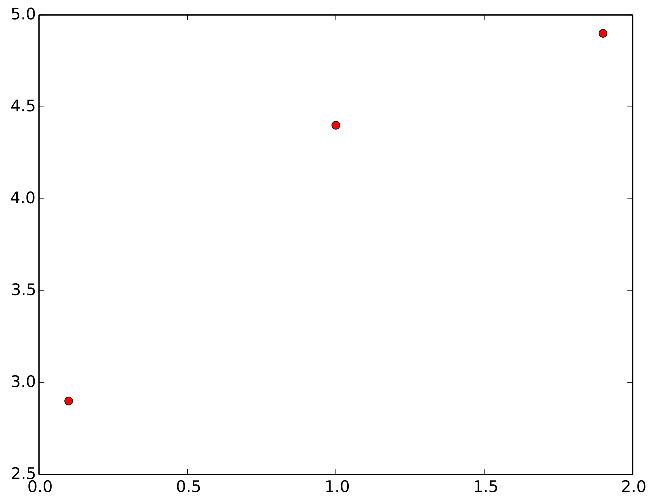

Figure 8: Plot from matplotlib example 2.

Note that you can plot multiple curves on one figure by calling the plot command multiples times and only then saving the figure.

So far we only plotted data-points. If we want to plot a function, the easiest way is to create a set of data-points ourselves. How many data-points we use depends on how accurate we want to plot to be. Have a look at the following code, which plots the sine function.

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
x = []
y = []
import math
N=1000
for i in xrange(N):
    xp = 2.*math.pi * i/N
    x.append( xp )
    y.append( math.sin(xp) )
plt.plot(x,y)
plt.savefig("plot.pdf")
```

Code 4: Matplotlib example 3.

The output of the above code produces a sine curve sampled at $N = 1000$ data-points. It looks like this:

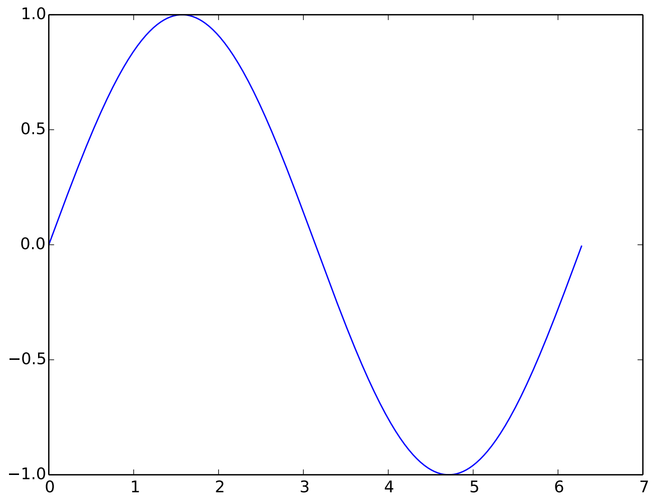

Figure 9: Plot from matplotlib example 3.

7.4 3D Data

There are multiple concepts that allow you to plot three dimensional data on a two dimensional surface. First, you could simply project your data down to two dimensions. Second, you can make use of a series of plots, each being a cut in the dimension that you are not plotting. Third, you can use colour to represent information about the third dimensions.

Let's try out some of these ideas. We start by creating some data. We need three arrays now, one for each dimension.

```
n = 100
xs = []
ys = []
zs = []
for i in range(n):
    xs.append(i)
    ys.append(i)
    zs.append(i*i)
```

Code 5: Python code generating data for plotting examples.

To plot that data with a simple projection into the xy plane, we can use the same syntax as before:

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
ax.scatter(xs, ys)
plt.savefig("plot.pdf")
```

Code 6: Matplotlib example 4.

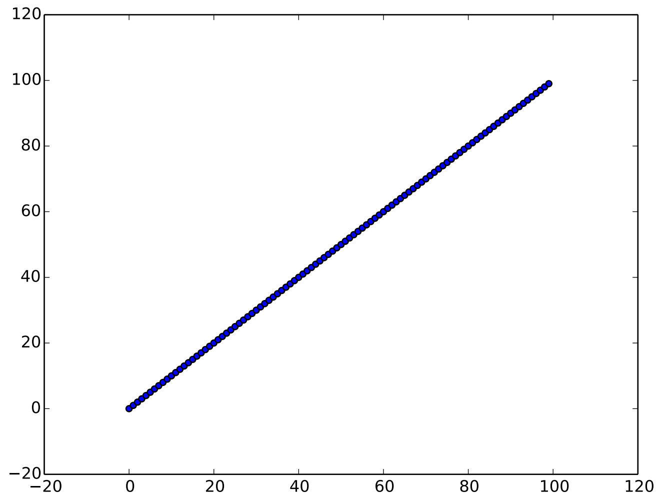

Figure 10: Plot from matplotlib example 4.

Of course, in the above plot, we've lost all the information about the third dimension.

Next, let's try plotting the same data with matplotlib 3d projection function. Here is the sample code:

```
import matplotlib
matplotlib.use('pdf')
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(xs, ys, zs)

plt.savefig("plot.pdf")
```

Code 7: Matplotlib example 5.

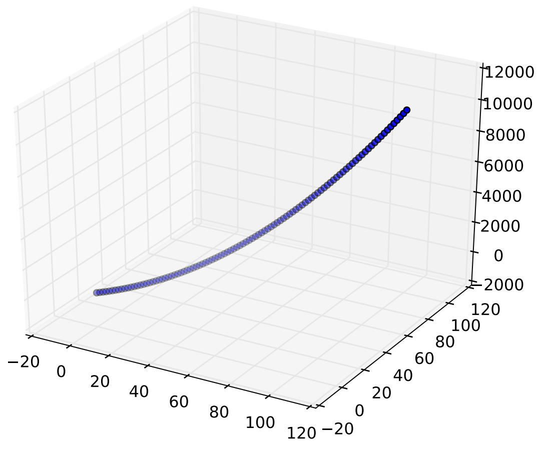

Figure 11: Plot from matplotlib example 5.

The result is better, we can now estimate a bit better how the data looks like. However, we can still not really see the what the function looks like.

Next, let's plot the same data once again and use colour as the third dimension. With matplotlib you can do this with the following snippet:

```
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111)
sc = ax.scatter(xs, ys, c=zs, cmap='gray')
plt.colorbar(sc)
plt.savefig("plot.pdf")
```

Code 8: Matplotlib example 6.

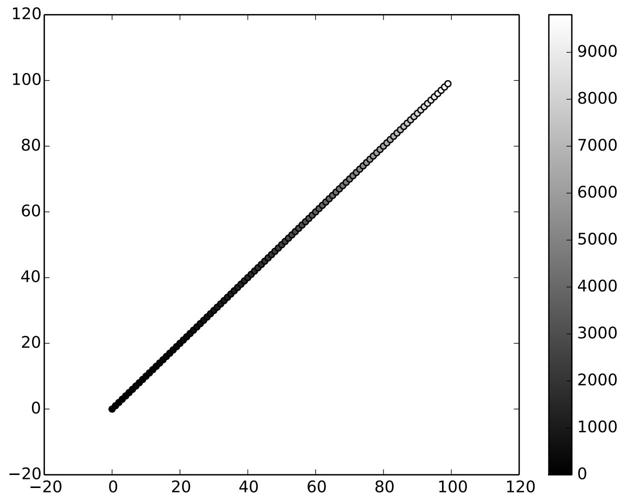

Figure 12: Plot from matplotlib example 6.

The above plot finally contains all the information in one plot. This kind of plot is particularly useful if you have a grid of datapoints. You can then use the `imshow` function as is shown in the next snipped and plot.

```python
import matplotlib
matplotlib.use('pdf')
import matplotlib.pyplot as plt
import math

fig = plt.figure()
ax = fig.add_subplot(111)
n = 100
data = []
for i in range(n):
    row = []
    for j in range(n):
        row.append(math.sin(i/5.)*math.cos(j/10.))
    data.append(row)
sc = ax.imshow(data, cmap='gray')
plt.colorbar(sc)
plt.savefig("plot4.pdf")
```

Code 9: Matplotlib example 7.

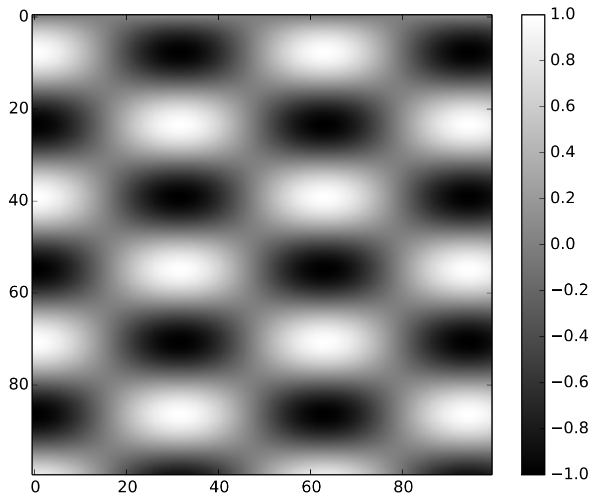

Figure 13: Plot from matplotlib example 7, plotting the function $f(x, y) = \sin(x/5)\cos(x/10)$.

7.5 A comment on colourbars

In recent years. there has been a considerable amount of discussion about how to choose a good colourmap.

There is a concept of a *perceptually uniform* colormap. If satisfied, then the colormap has the property that if your data goes from 0.1 to 0.2, this should create about the same perceptual change as if your data goes from 0.8 to 0.9.

The difficult task is now to find a colour map that is nice to look at, perceptually uniform, perceptually uniform when printed in black and white and perceptually uniform when seen by people with colourblindness. Colourblindness is much wider spread then you might estimate (approximately 8% of men an 1% of women are at least partially colorblind).

Let's look at three examples (taken from `http://bids.github.io/colormap/`).

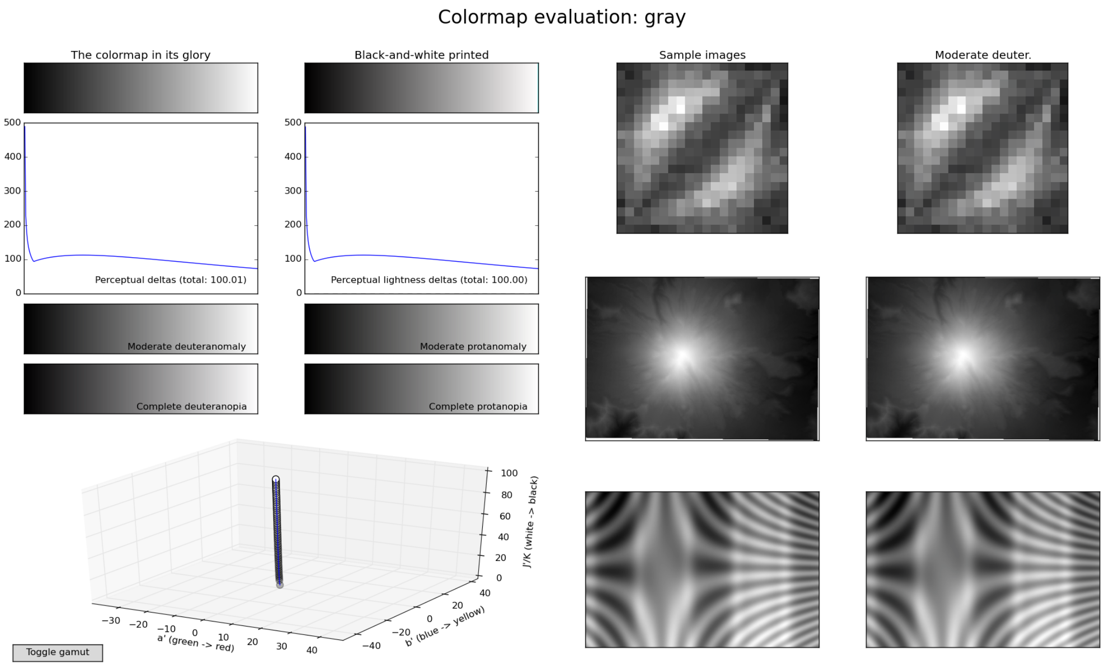

Figure 14: Properties of the gray color map.

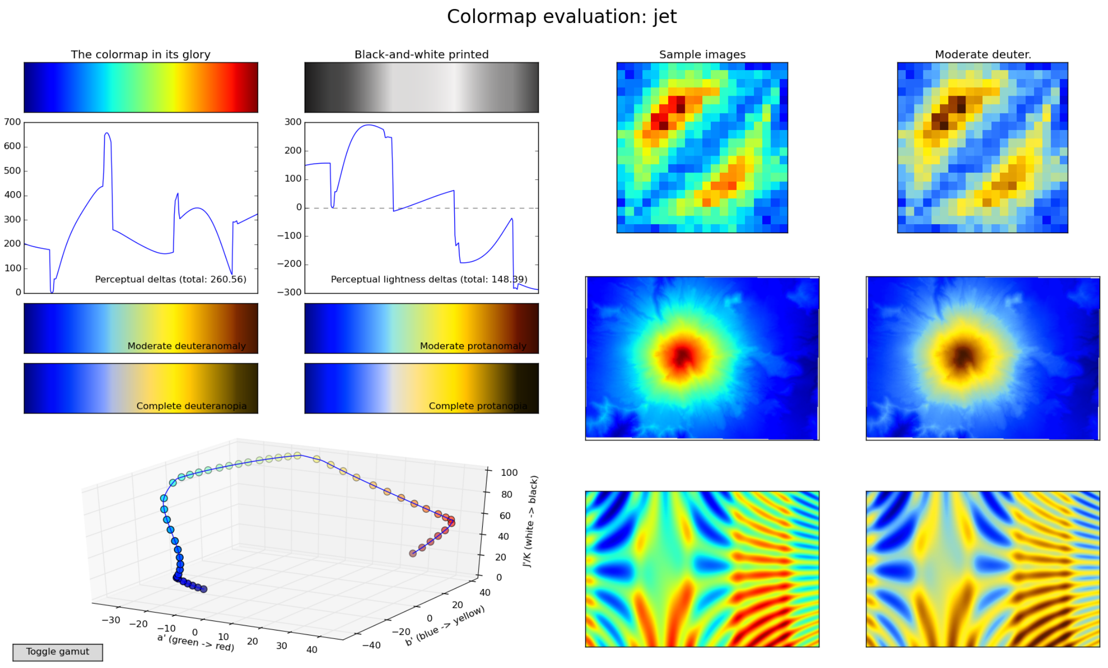

Figure 15: Properties of the jet color map.

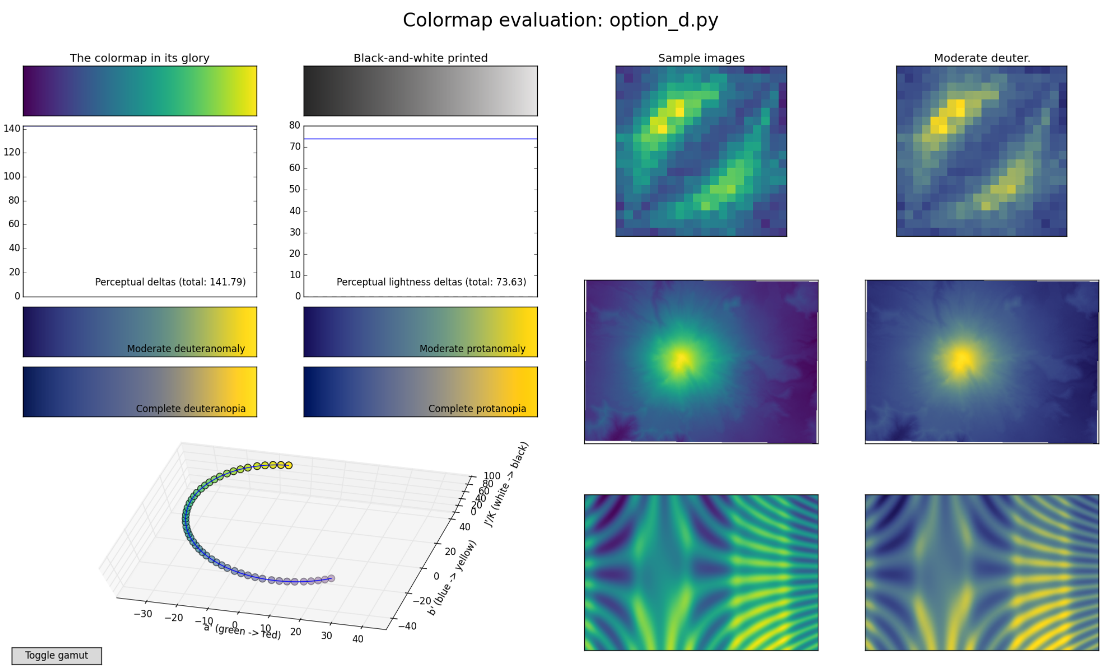

Figure 16: Properties of the viridis color map (the new default in matplotlib).

In summary: do not use jet. Black and white is great. But if you want a more colorful map, choose one such as viridis, the new default in matplotlib.

7.6 Final comments, and a checklist for a good plot

This is all that we'll need for plotting in this course. Keep in mind that there is an almost infinite number of options for each plotting style within matplotlib. And if there isn't the right option available for your task, there is likely another package out there that does it for you.

Here is a checklist for what is generaly considered a good plot.

- Axes have labels and units
- All dimensions on the plot (x, y, color, shape, thickness, opacity) are used if needed, but only if needed.
- Colours, if used, are suitable for colourblind people
- Colormap has a constant contrast gradient

End of lecture 6.