8 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.

8.1 Copying files

Before we get started on plotting, we need to talk about moving files between different computers as we want to download the pdf file we generate from the linux machine to your own computer. There are a couple of ways to do that, depending on the operating system and terminal program you decided to use. The easiest way to copy files to a computer running Windows is by using the MobaXterm application. On the left hand side, you can see a tree of the remote directory structure. Simply go to the subdirectory where your file is in and drag it to any folder on your windows machine to copy the file.

If you are using MacOSX, Linux or another unix-like operating system, you can use the command line on your computer to copy files with the scp program. The basic syntax is as follows

```
scp username@rein001.utsc.utoronto.ca:~/PSCB57/
   assignment_06/plot.pdf .
```

where you would of course replace `username` with your own username and you would adjust the path to the file according to what you want to transfer. Don't forget the dot at the end. This tells the scp program to copy the file to the current directory you're in on your local machine. When you execute the scp program it will ask for your password.

8.2 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.

8.3 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.

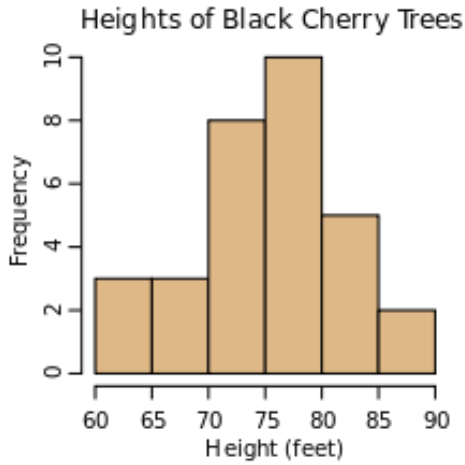

Figure 8: A simple histogram (image from wikipedia).

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 44: 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:

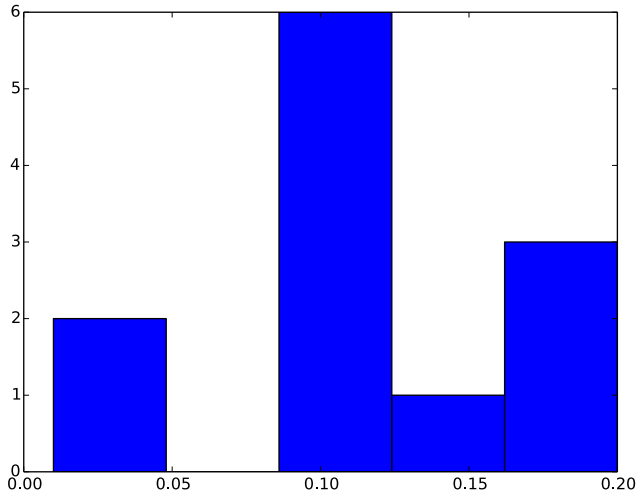

Figure 9: Histogram example using matplotlib.

8.4 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 45: 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.

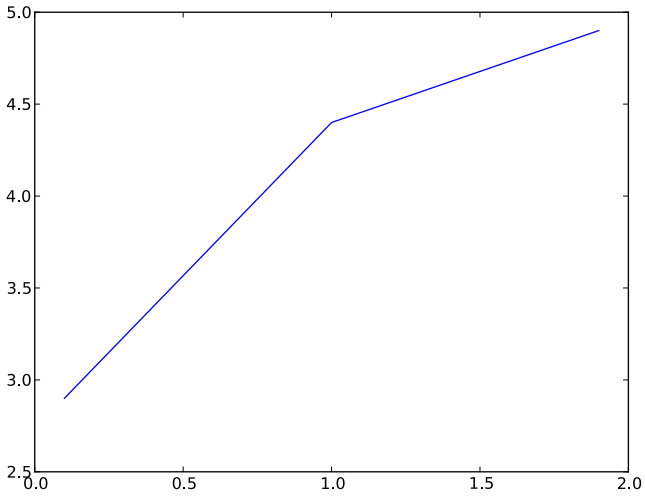

Figure 10: Plot from matplotlib example 1.

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.

```
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 46: Matplotlib example 2.

The output now looks like this:

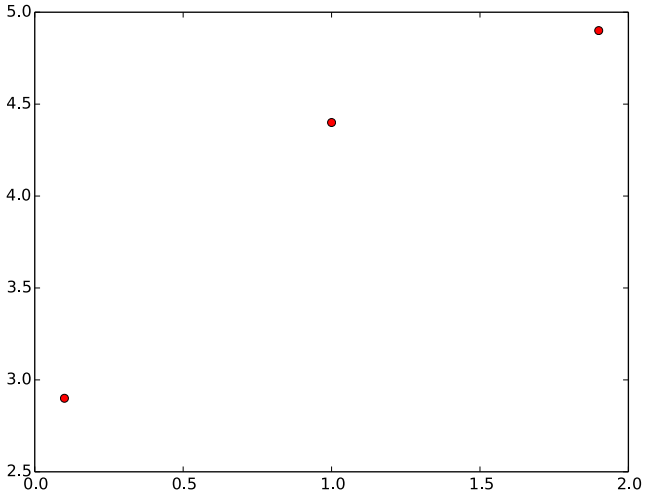

Figure 11: Plot from matplotlib example 2.

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 47: Matplotlib example 3.

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

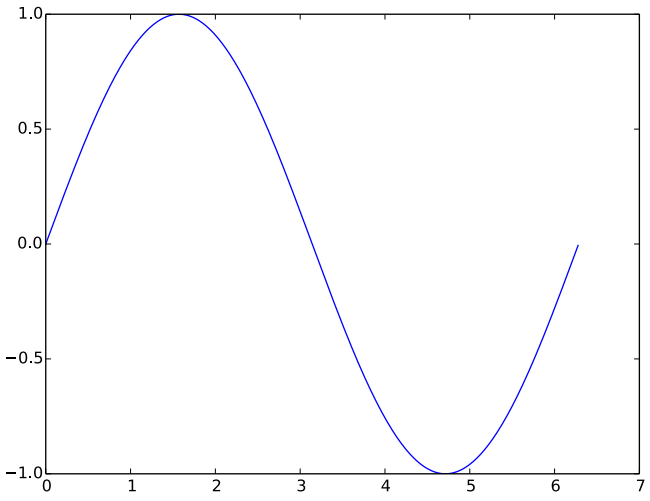

Figure 12: Plot from matplotlib example 3.

8.5 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 48: 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 49: Matplotlib example 4.

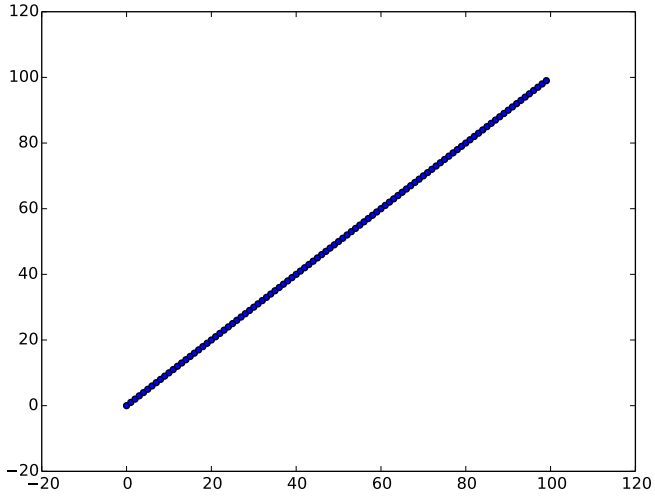

Figure 13: Plot from matplotlib example 4.

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 50: Matplotlib example 5.

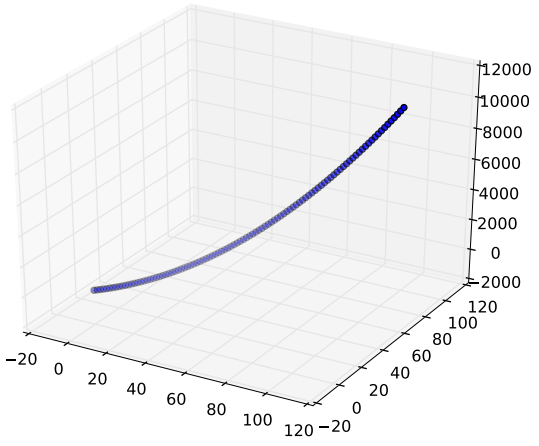

Figure 14: Plot from matplotlib example 5.

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 51: Matplotlib example 6.

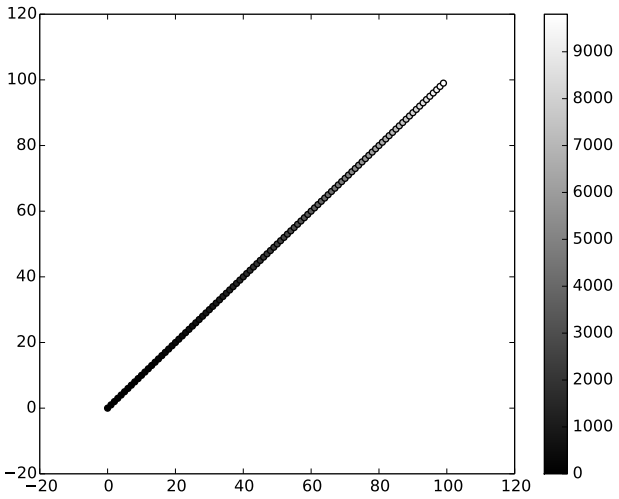

Figure 15: Plot from matplotlib example 6.

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.

```
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 52: Matplotlib example 7.

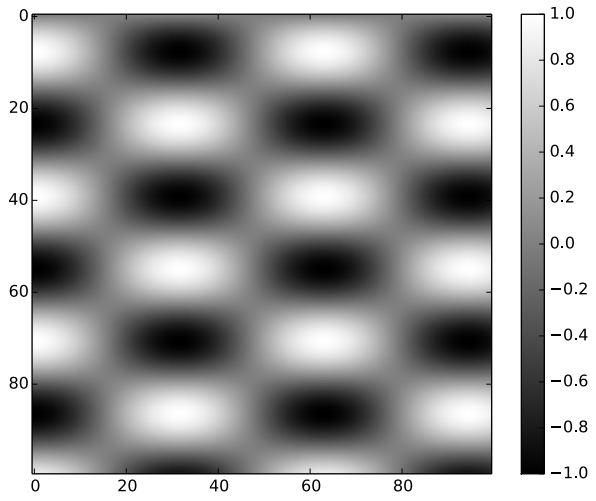

Figure 16: Plot from matplotlib example 7, plotting the function $f(x, y) = \sin(x/5)\cos(x/10)$.

8.6 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/`).

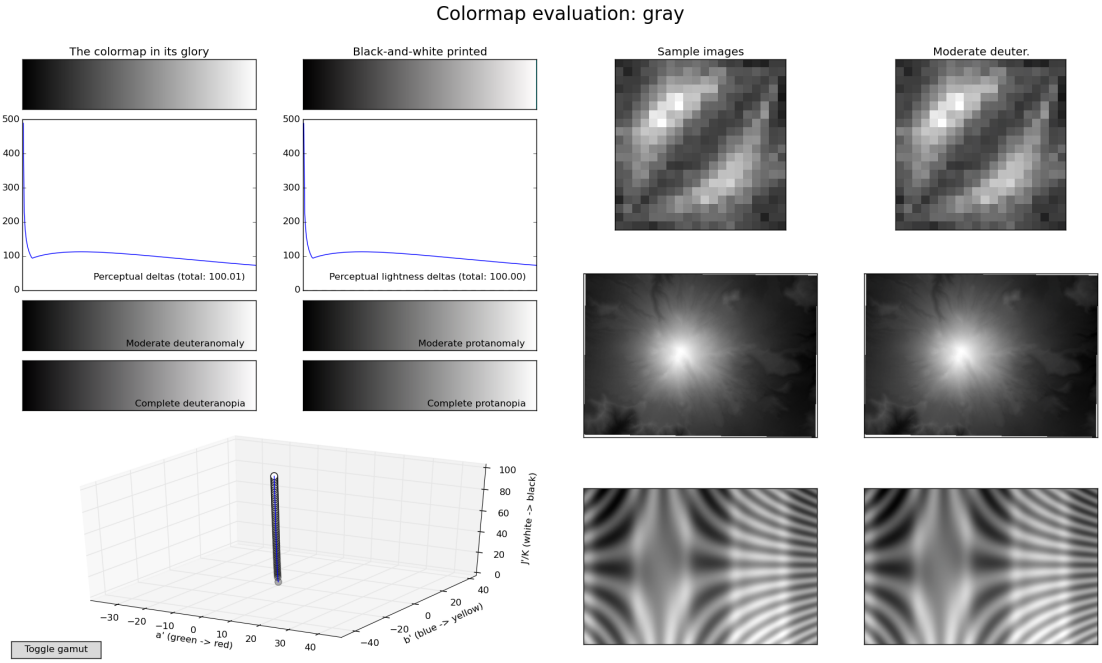

Figure 17: Properties of the gray color map.

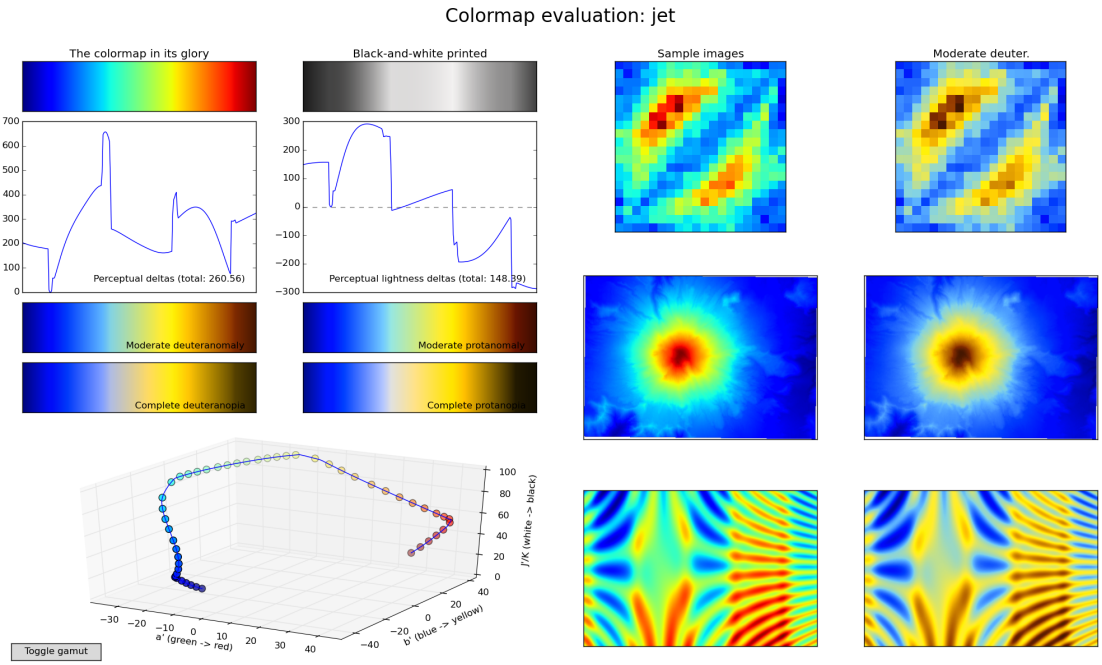

Figure 18: Properties of the jet color map.

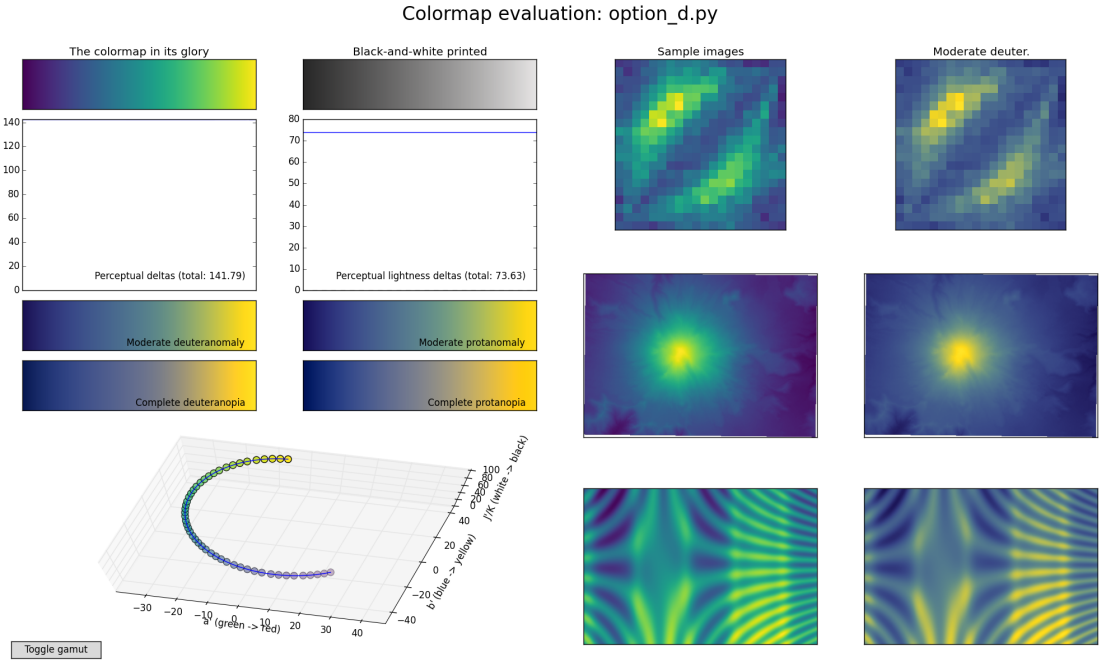

Figure 19: Properties of the viridis color map (the new default in matplotlib).

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.

8.7 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.