Week 5

We have now done basic variables, functions, lists, arrays and loops. We can now do some plotting.

Script

In previous labs we looked at generating and manipulating lots of mathematical operations and expressions. We can use this to generate a lot of data that we then want to analyse. One crucial tool for this analysis is the plot.

One example is the graphical method for solving problems like

$$\begin{aligned} \max x_1 + x_2 &= z \\ 2x_1 + x_2 &\leq 4 \\ x_1 + 2x_2 &\leq 3 \end{aligned}$$

where $x_1, x_2 \geq 0$.

We use the constraints (the last two equations combined with $x_1, x_2 \geq 0$) to plot the *feasible region*. We then check the value of the cost function at all corners of the feasible region.

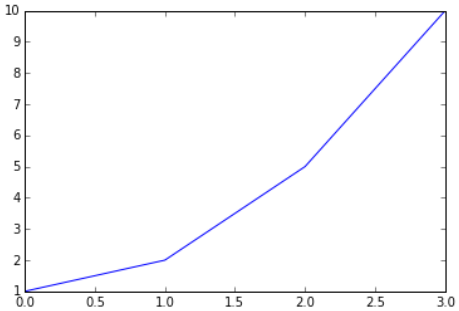
The standard Python plotting library is called `matplotlib`. We'll start by showing its simplest interface, `pyplot`. First `import` the library:

```
%matplotlib inline
```

```
from matplotlib import pyplot
```

If we want to plot a data set `ys` and coordinate locations `xs`, we can use the `plot` command:

```
xs = [0, 1, 2, 3]
ys = [1, 2, 5, 10]
pyplot.plot(xs, ys)
pyplot.show()
```



- `plot` will use a line to joing the points by default;
- We don't have to use a list; a tuple, or a `numpy` `array` would work just as well;
- `pyplot.show()` at the end forces it to display the plot. This is not always needed, but it is better to always use it.

`matplotlib` is most effectively used together with `numpy`. For example, suppose we want to plot where $C_1 = 2x_1 + x_2 - 4 = 0$, which corresponds to the first constraint above. We don't know what are reasonable values of x_1 and x_2, but let's create the plot for $x_1 \in [-1, 3]$.

First generate arrays to sample x_1 over this range. We can create linearly spaced points using `numpy.linspace`:

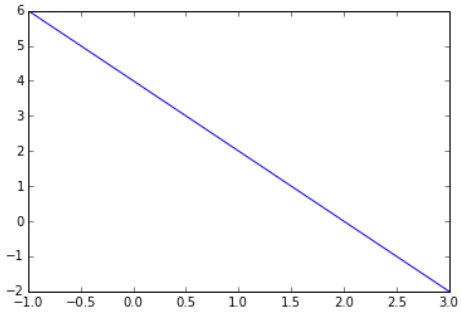
```
import numpy

x1 = numpy.linspace(-1, 3)
```

We can then construct the values of x_2 such that $C_1 = 0$ in one operation:

```
x2_C1 = 4 - 2 * x1
```

We now plot this line:

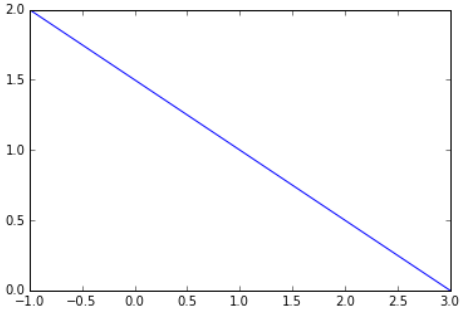
```
pyplot.plot(x1, x2_C1)
pyplot.show()
```



We can do the same for the constraint $C_2 = x_1 + 2x_2 - 3$, giving

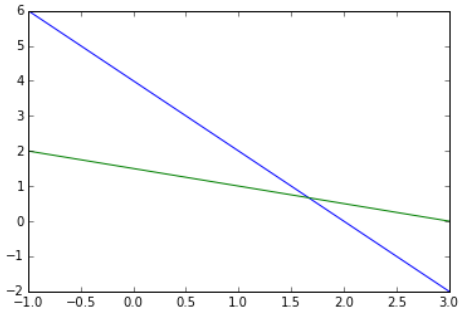
```
x2_C2 = (3 - x1) / 2
```

```
pyplot.plot(x1, x2_C2)
pyplot.show()
```



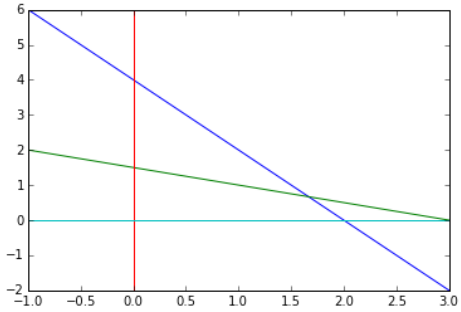
We can combine both of these in one plot:

```
pyplot.plot(x1, x2_C1)
pyplot.plot(x1, x2_C2)
pyplot.show()
```



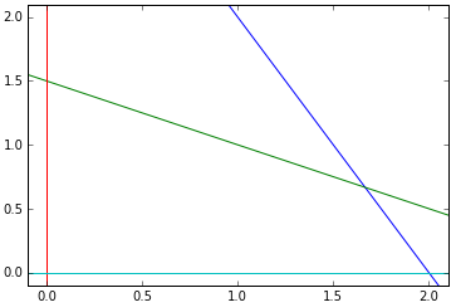
Finally we can add lines corresponding to $x_1, x_2 \geq 0$. For this we can just use two points connected together:

```
pyplot.plot(x1, x2_C1)
pyplot.plot(x1, x2_C2)
pyplot.plot([0, 0], [-2, 6])
pyplot.plot([-1, 3], [0, 0])
pyplot.show()
```



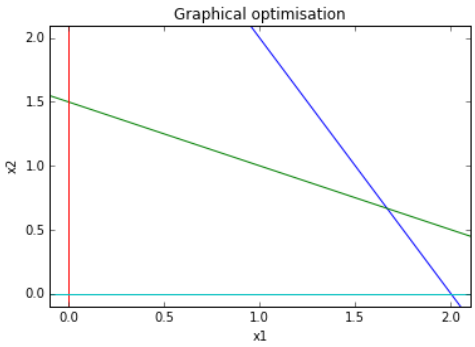
We can see that the feasible region only runs up to $x_1 \sim 2$ and $x_2 \sim 2$. We can restrict the range of the plot to concentrate on this area:

```
pyplot.plot(x1, x2_C1)
pyplot.plot(x1, x2_C2)
pyplot.plot([0, 0], [-2, 6])
pyplot.plot([-1, 3], [0, 0])
pyplot.xlim(-0.1, 2.1)
pyplot.ylim(-0.1, 2.1)
pyplot.show()
```



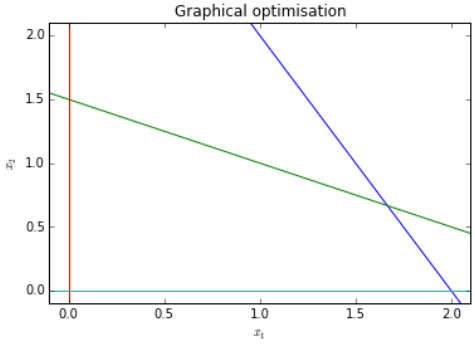
We can add axis labels and titles to the plot to make it clearer:

```
pyplot.plot(x1, x2_C1)
pyplot.plot(x1, x2_C2)
pyplot.plot([0, 0], [-2, 6])
pyplot.plot([-1, 3], [0, 0])
pyplot.xlim(-0.1, 2.1)
pyplot.ylim(-0.1, 2.1)
pyplot.xlabel('x1')
pyplot.ylabel('x2')
pyplot.title('Graphical optimisation')
pyplot.show()
```



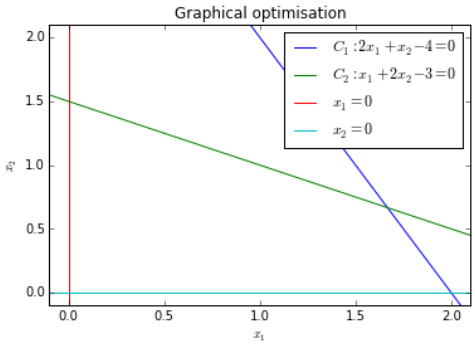
We will cover LaTeX later in the course, but we can already use the notation to make the plot look nicer. Surround any mathematical text with dollar signs `$`; add an `r` before the string to max it "raw"; use underscores for subscripts and hats for superscripts:

```
pyplot.plot(x1, x2_C1)
pyplot.plot(x1, x2_C2)
pyplot.plot([0, 0], [-2, 6])
pyplot.plot([-1, 3], [0, 0])
pyplot.xlim(-0.1, 2.1)
pyplot.ylim(-0.1, 2.1)
pyplot.xlabel(r'$x_1$')
pyplot.ylabel(r'$x_2$')
pyplot.title('Graphical optimisation')
pyplot.show()
```



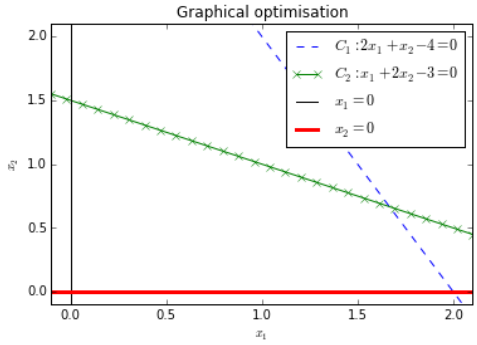
To remind us what each curve is, label them and add a legend:

```
pyplot.plot(x1, x2_C1, label=r"$C_1: 2 x_1 + x_2 - 4 = 0$")
pyplot.plot(x1, x2_C2, label=r"$C_2: x_1 + 2 x_2 - 3 = 0$")
pyplot.plot([0, 0], [-2, 6], label=r"$x_1 = 0$")
pyplot.plot([-1, 3], [0, 0], label=r"$x_2 = 0$")
pyplot.xlim(-0.1, 2.1)
pyplot.ylim(-0.1, 2.1)
pyplot.xlabel(r'$x_1$')
pyplot.ylabel(r'$x_2$')
pyplot.title('Graphical optimisation')
pyplot.legend()
pyplot.show()
```



We can change line styles and colors, and use markers instead:

```
pyplot.plot(x1, x2_C1,
            linestyle="--",
            label=r"$C_1: 2 x_1 + x_2 - 4 = 0$")
pyplot.plot(x1, x2_C2,
            marker="x",
            label=r"$C_2: x_1 + 2 x_2 - 3 = 0$")
pyplot.plot([0, 0], [-2, 6],
            color="black",
            label=r"$x_1 = 0$")
pyplot.plot([-1, 3], [0, 0],
            linewidth=3,
            label=r"$x_2 = 0$")
pyplot.xlim(-0.1, 2.1)
pyplot.ylim(-0.1, 2.1)
pyplot.xlabel(r'$x_1$')
pyplot.ylabel(r'$x_2$')
pyplot.title('Graphical optimisation')
pyplot.legend()
pyplot.show()
```



Combinations of these commands will give most things that you want.

Exercise

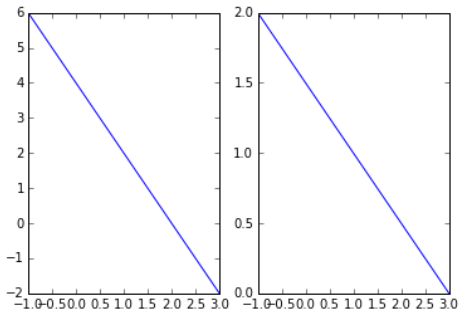
Plot x^s on $x \in [0, 1]$ for $s = 2, \ldots, 5$. Do the same for $x^{1/s}$. Add legends and axis labels.

More complex plots

A plot is made up of lots of individual pieces that can be separately manipulated to control all the fine details. At the top level is the `figure` object, which contains everything. This `axis` object is the next level down, which can be moved within a `figure` and manipulated again. The `axis` is the thing on which you can create individual plots. This has the `x` and `y` (and possibly additional) pieces and allows control of the ticks and labels, for example.

We are going to create a figure with two subplots. Each subplot has its own `axis`. We create the figure and axis objects first. The we plot the C_1 and C_2 constraints as above on each subplot:

```
fig, axes = pyplot.subplots(nrows=1, ncols=2)
axes[0].plot(x1, x2_C1)
axes[1].plot(x1, x2_C2)
pyplot.show()
```



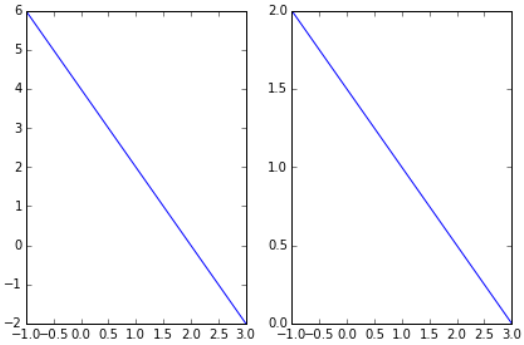
We can check that the `axes` object is a container that contains two things:

```
print(axes)
print(len(axes))
```

```
[<matplotlib.axes._subplots.AxesSubplot object at 0x10e399128>
 <matplotlib.axes._subplots.AxesSubplot object at 0x10ee069b0>]
2
```

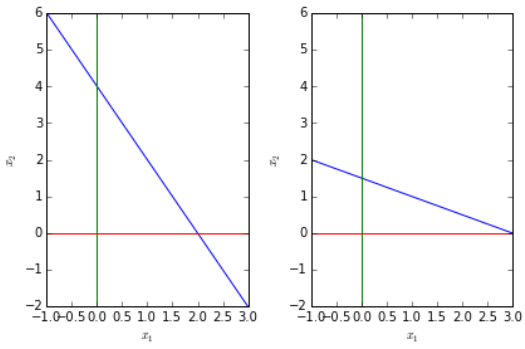
We note that the text labels look a bit too close together. To fix this, use the `tight_layout` command:

```
fig, axes = pyplot.subplots(nrows=1, ncols=2)
axes[0].plot(x1, x2_C1)
axes[1].plot(x1, x2_C2)
fig.tight_layout()
pyplot.show()
```



We will add the $x_1, x_2 \geq 0$ constraints and add axis labels. Note that the plotting commands (like `plot`) remain the same, but the setting commands (`xlim`, `xlabel` etc) sometimes add `set_` (use tab completion to check what works):

```
fig, axes = pyplot.subplots(nrows=1, ncols=2)
axes[0].plot(x1, x2_C1)
axes[1].plot(x1, x2_C2)
for axis in axes:
    axis.plot([0, 0], [-2, 6])
    axis.plot([-1, 3], [0, 0])
    axis.set_xlabel(r"$x_1$")
    axis.set_ylabel(r"$x_2$")
fig.tight_layout()
pyplot.show()
```



Exercise

Repeat the plots of x^s and $x^{1/s}$ from above. First produce a single figure with two subplots: one subplot should contain x^s and one $x^{1/s}$, for $s = 2, \ldots, 5$. Each subplot should have a legend and appropriate axis labels. Second produce a single figure with eight subplots, one for each curve. Each subplot should have appropriate axis labels, but no legend.

Exercise

Repeat the plots of x^s and $x^{1/s}$ but use logarithmic scales (and so set $x \in [10^{-5}, 1]$). Investigate the `loglog`, or `semilogx` and `semilogy` functions, or the `xscale('log')` function, for example.