# IPython and Matplotlib

May 13, 2017

## 1 IPython and Matplotlib

### 1.1 Starting an IPython notebook

```
$ ipython notebook
```

After starting the notebook import `numpy` and `matplotlib` modules automagically.

```
In [1]: %pylab
```

```
Using matplotlib backend: MacOSX
Populating the interactive namespace from numpy and matplotlib
```

Or if you want to have the plots show inside the notebook use:

```
In [2]: %pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

### 1.2 Plotting with Pylab

Let us now plot a a sinusoid. For this we need to prepare an array (or vector) of function arguments.

```
In [3]: x = arange(0, 20, 0.01)
```

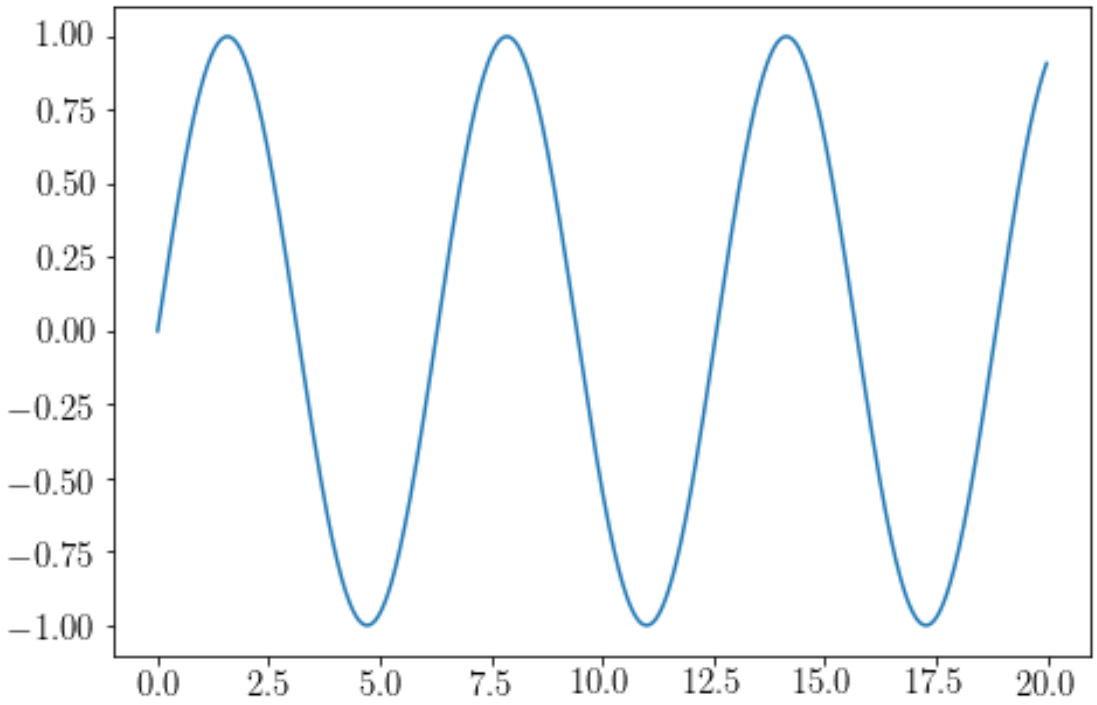
Now let us calculate the function values using numpy's integrate `sin()` function.

```
In [4]: y = sin(x)
```

Plotting is very simple now. Just write:

```
In [5]: plot(x, y)
```

```
Out[5]: [<matplotlib.lines.Line2D at 0x1126b4e80>]
```

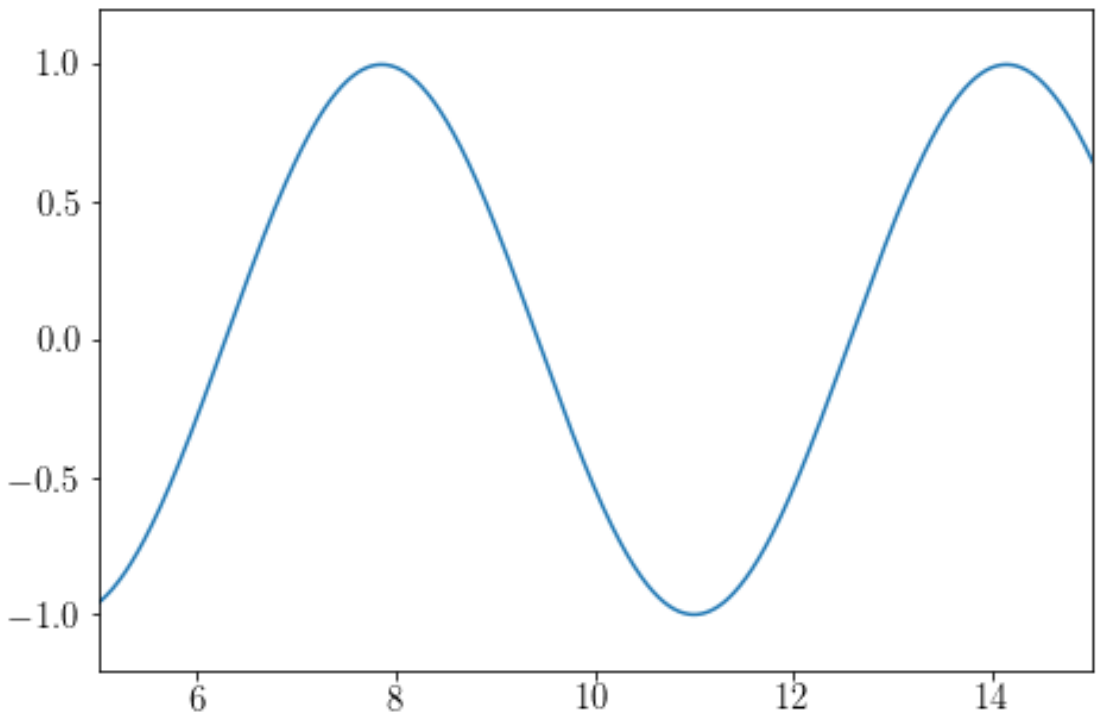



### 1.2.1 Axes limits

Now we want to change the axis limits using the `xlim()` and `ylim()` commands.

```
In [6]: xlim(5, 15)
        ylim(-1.2, 1.2)
        plot(x, y)
```

```
Out[6]: [<matplotlib.lines.Line2D at 0x10abe3dd8>]
```

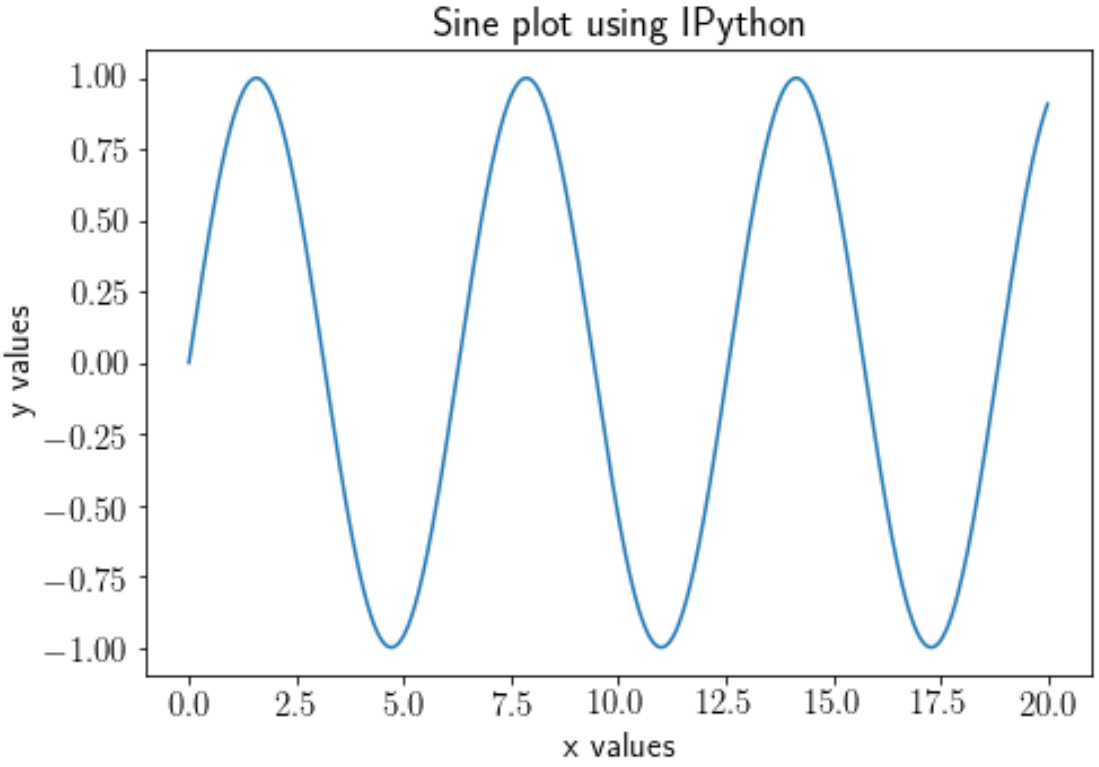



### 1.2.2 Labels and Title

Every proper plot needs axes labels and a title.

```
In [7]: xlabel('x values')
        ylabel('y values')
        title('Sine plot using IPython')
        plot(x, y)
```

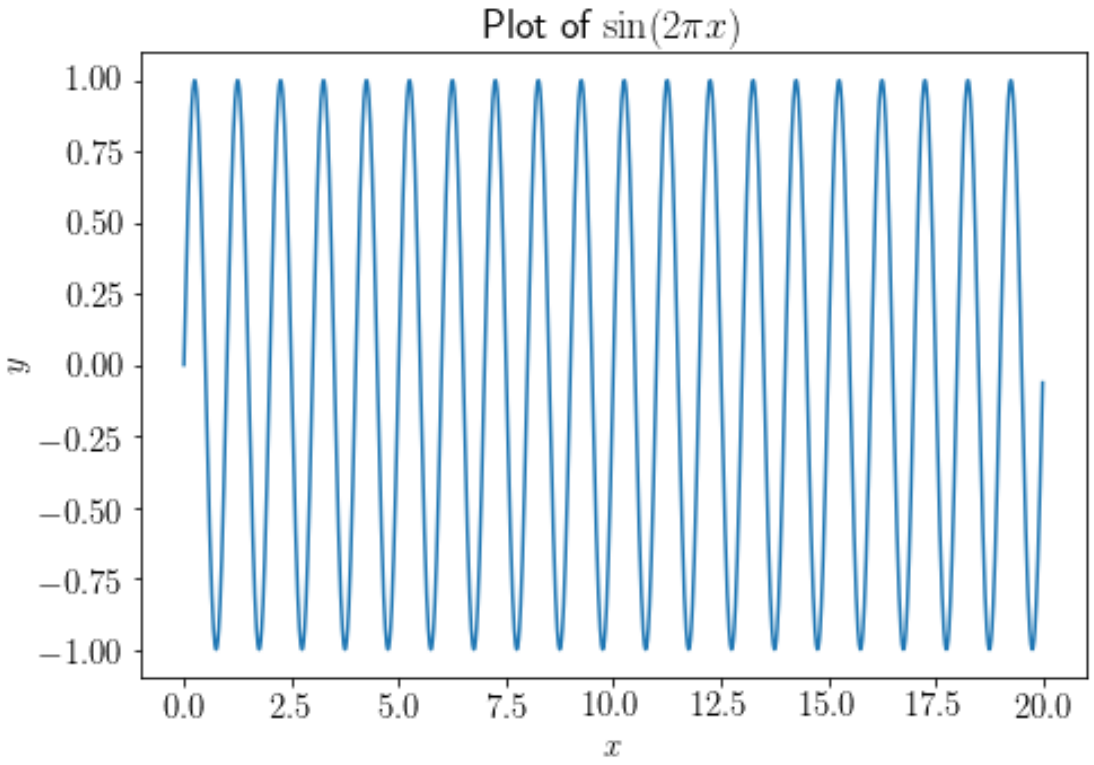
```
Out[7]: [<matplotlib.lines.Line2D at 0x11282d940>]
```




Labels can of course also use LaTeX labels. (More LaTeX later this semester...)

```
In [8]: y = sin(2 * pi * x)
        xlabel(r'$x$') # the 'r' before the string indicates a "raw string"
        ylabel(r'$y$')
        title(r'Plot of $\sin(2 \pi x)$')
        plot(x, y)
```

```
Out[8]: [<matplotlib.lines.Line2D at 0x112a2b320>]
```

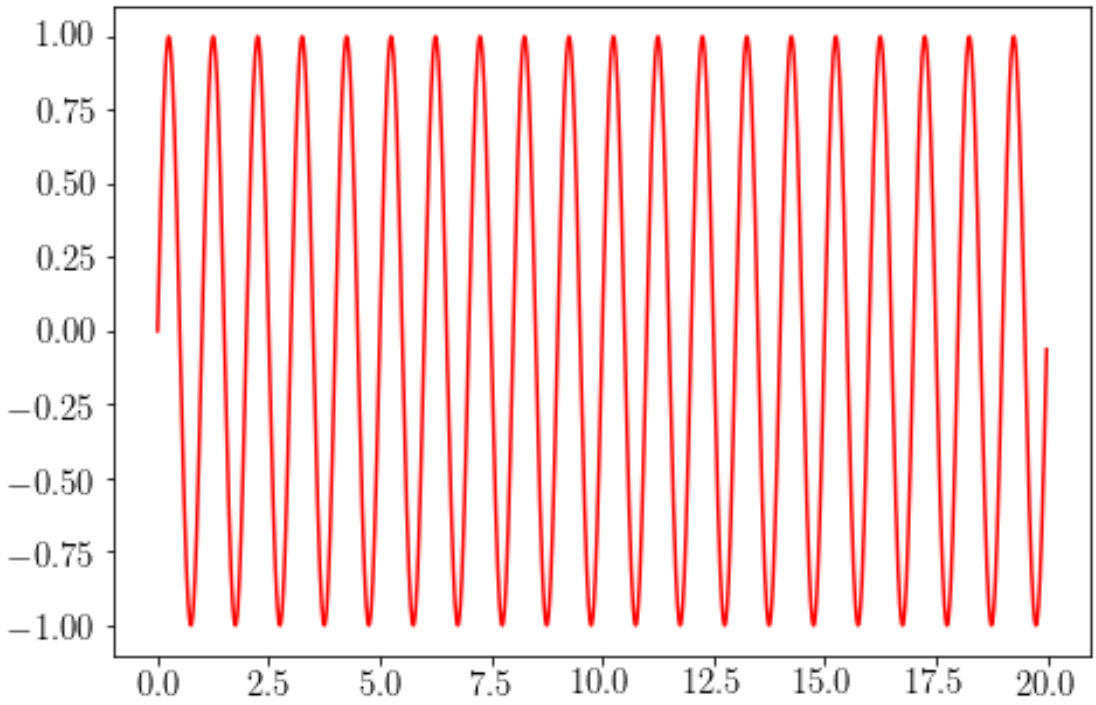



### 1.2.3 Different Line Styles

To change the line color, we can use the keyword argument `color` when calling the `plot` function.

```
In [9]: plot(x, y, color='red')
```

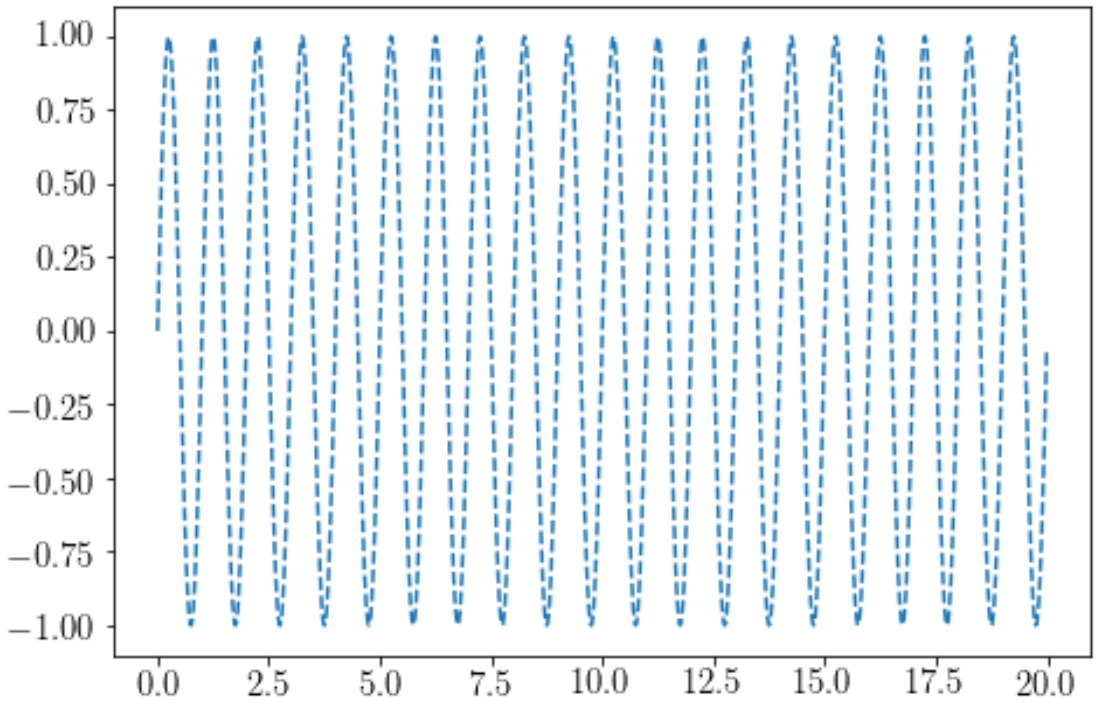
```
Out[9]: [<matplotlib.lines.Line2D at 0x112cd4c50>]
```




Or we can change the line style from a solid line to a dashed one using the `linestyle` keyword argument.

```
In [10]: plot(x, y, linestyle='dashed')
```

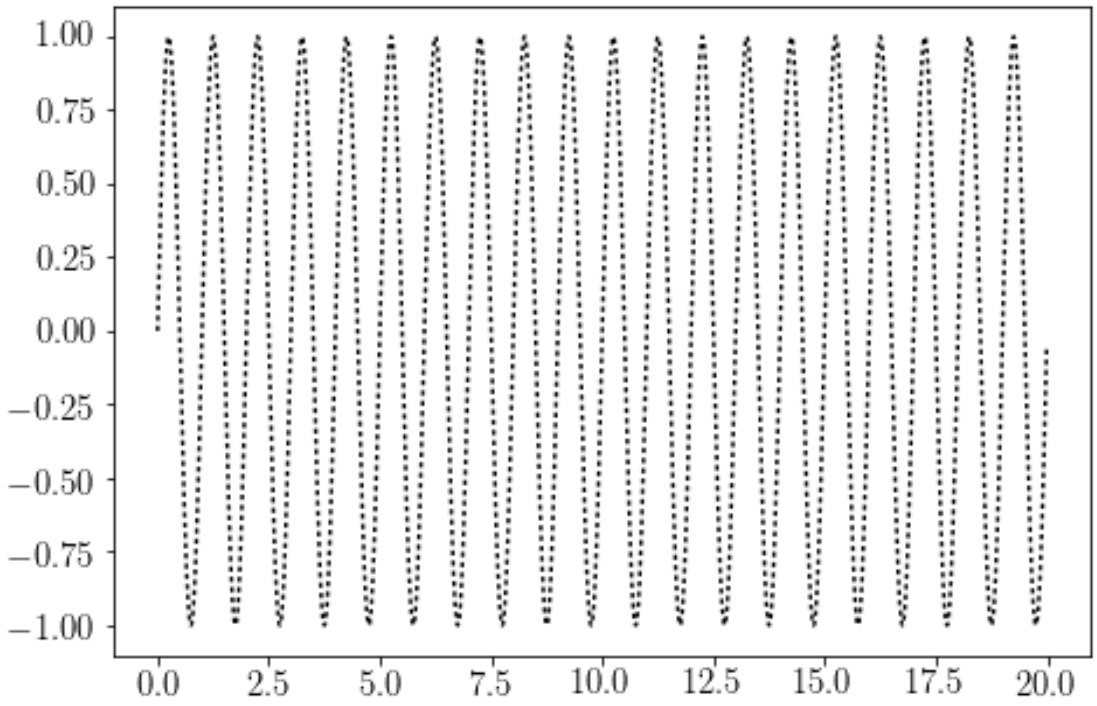
```
Out[10]: [<matplotlib.lines.Line2D at 0x112e38208>]
```




There is a very convenient way two combine these two in a very short way.

```
In [11]: plot(x, y, ':k')
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x112f9b1d0>]
```



Here are a few examples of colors and their abbreviations.

```
'r' = red
'g' = green
'b' = blue
'c' = cyan
'm' = magenta
'y' = yellow
'k' = black
'w' = white
```

Some examples for linestyles are:

```
'-' = solid
'--' = dashed
':' = dotted
'-.' = dot-dashed
'.' = points
'o' = filled circles
'^' = filled triangles
```

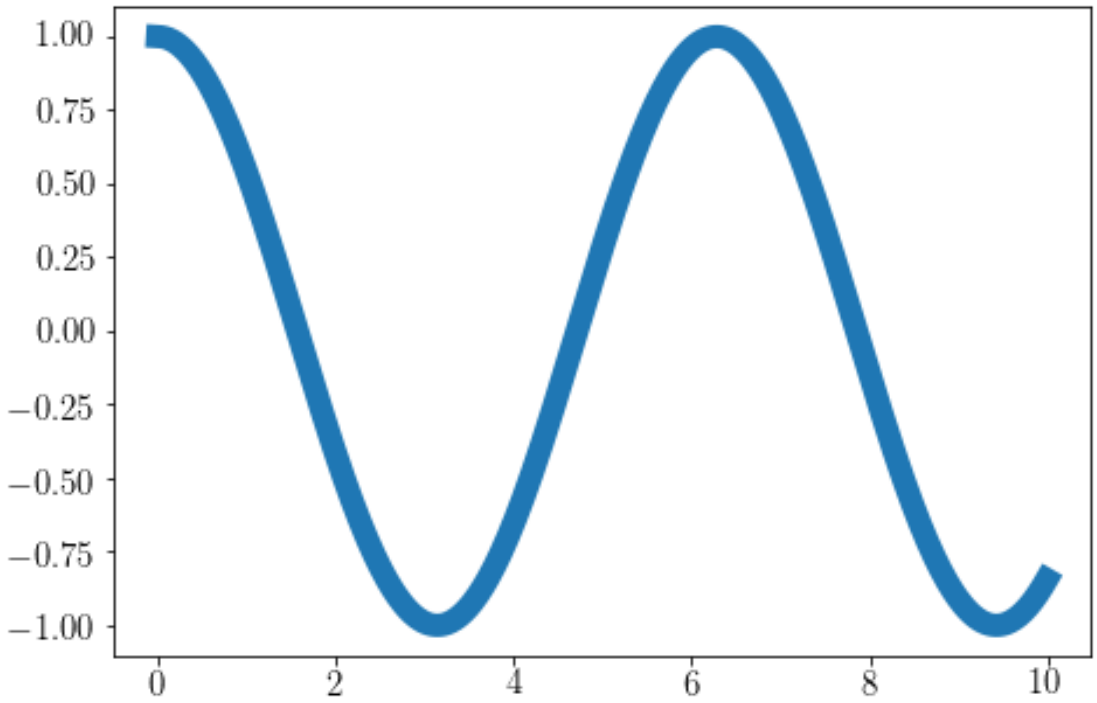
Speaking of abbreviations!
Do you remember Python's `help()` function?
In IPython you can just use `?` to get information on functions.
Let's try it on the plot function, right now.

```
In [12]: plot?
```

You cannot only change the style of the line, but also its width.

```
In [13]: x = arange(0, 10, 0.01)
         y = cos(x)
         plot(x, y, linewidth=10.0)

Out[13]: [<matplotlib.lines.Line2D at 0x113111dd8>]
```




### 1.2.4 Adding a legend to your plot

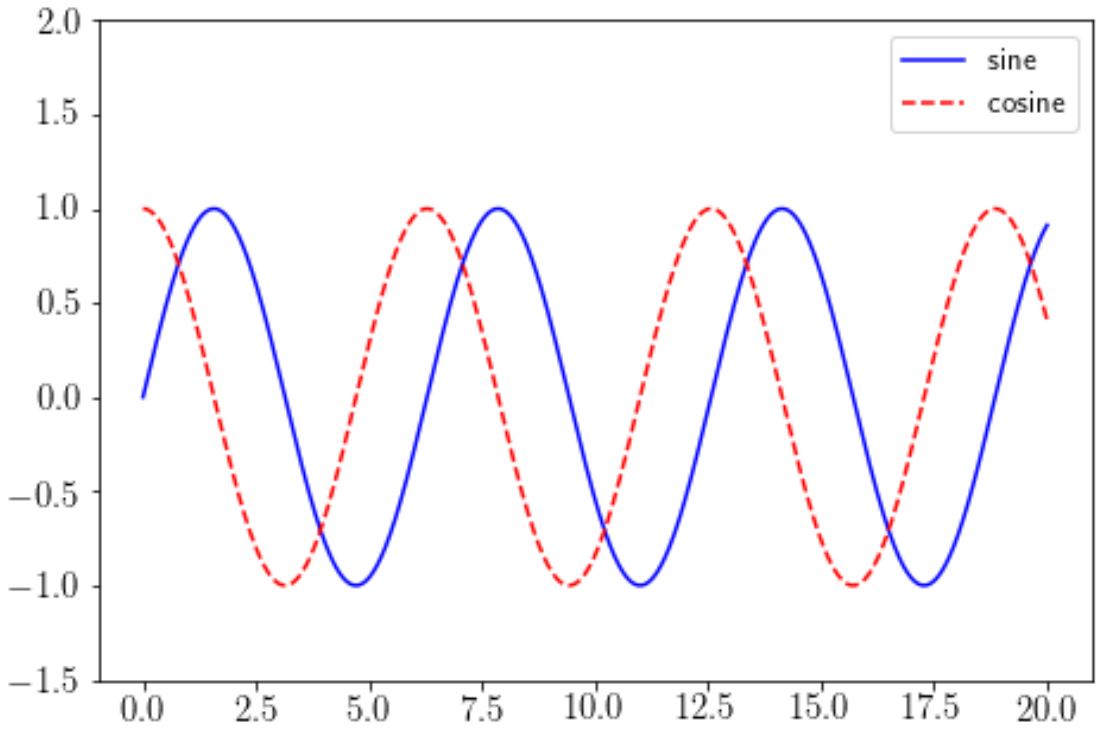
Naturally it is possible to have multiple lines or functions in the same plot.
In this case, it is very useful and recommended to include a legend to allow the reader to see which line refers to which function.

```
In [14]: x = arange(0, 20, 0.001)

In [15]: y1 = sin(x)
         y2 = cos(x)

In [16]: plot(x, y1, '-b', label='sine')
         plot(x, y2, '--r', label='cosine')
         legend(loc='upper right')
         ylim(-1.5, 2.0)

Out[16]: (-1.5, 2.0)
```



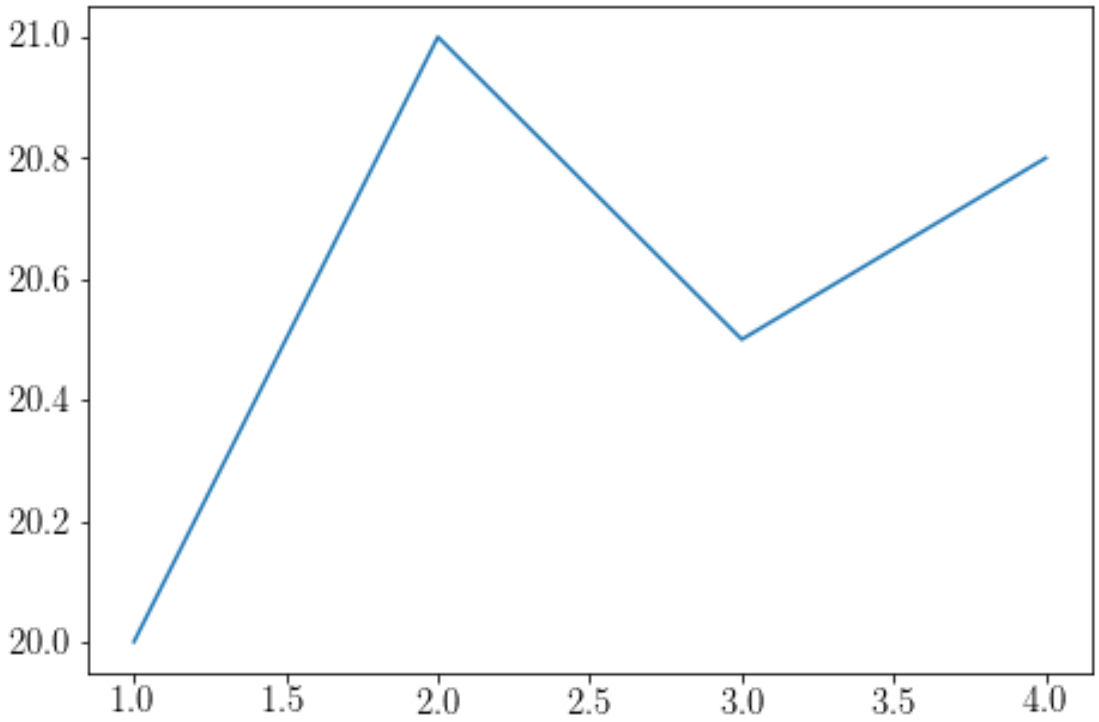

### 1.2.5 Plotting discrete values

Now instead of plotting a continous function let us consider an example of discrete arguments and function values.

```
In [17]: x = [1, 2, 3, 4]
         y = [20, 21, 20.5, 20.8]

         plot(x, y)
```

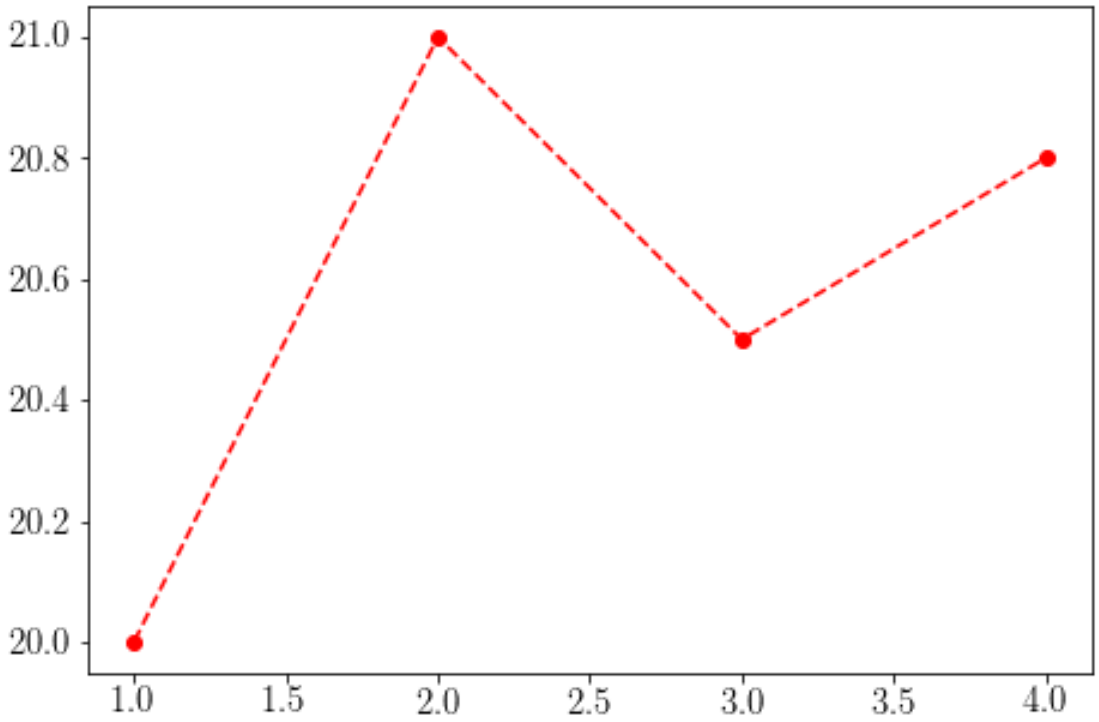
```
Out[17]: [<matplotlib.lines.Line2D at 0x1133de080>]
```




The above plot could be misleading, because we only have discrete values, but the plot suggests a continous function again.

```
In [18]: plot(x, y, linestyle='dashed', marker='o', color='red')
```

```
Out[18]: [<matplotlib.lines.Line2D at 0x113140cf8>]
```

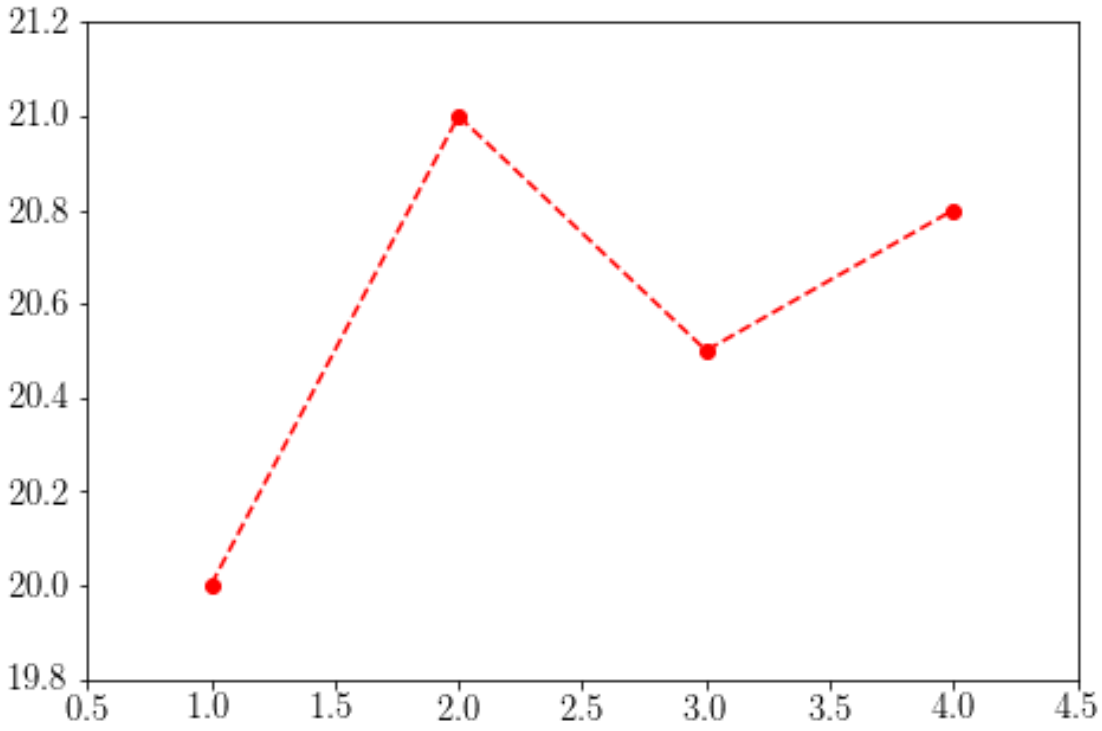



### 1.2.6 Manipulating the axes ticks

Above we learned how to set labels for the x and y axis.
But we can manipulate the axes ticks as well.

```
In [19]: plot(x, y, linestyle='dashed', marker='o', color='red')
         xlim(0.5, 4.5)
         ylim(19.8, 21.2)
```

```
Out[19]: (19.8, 21.2)
```

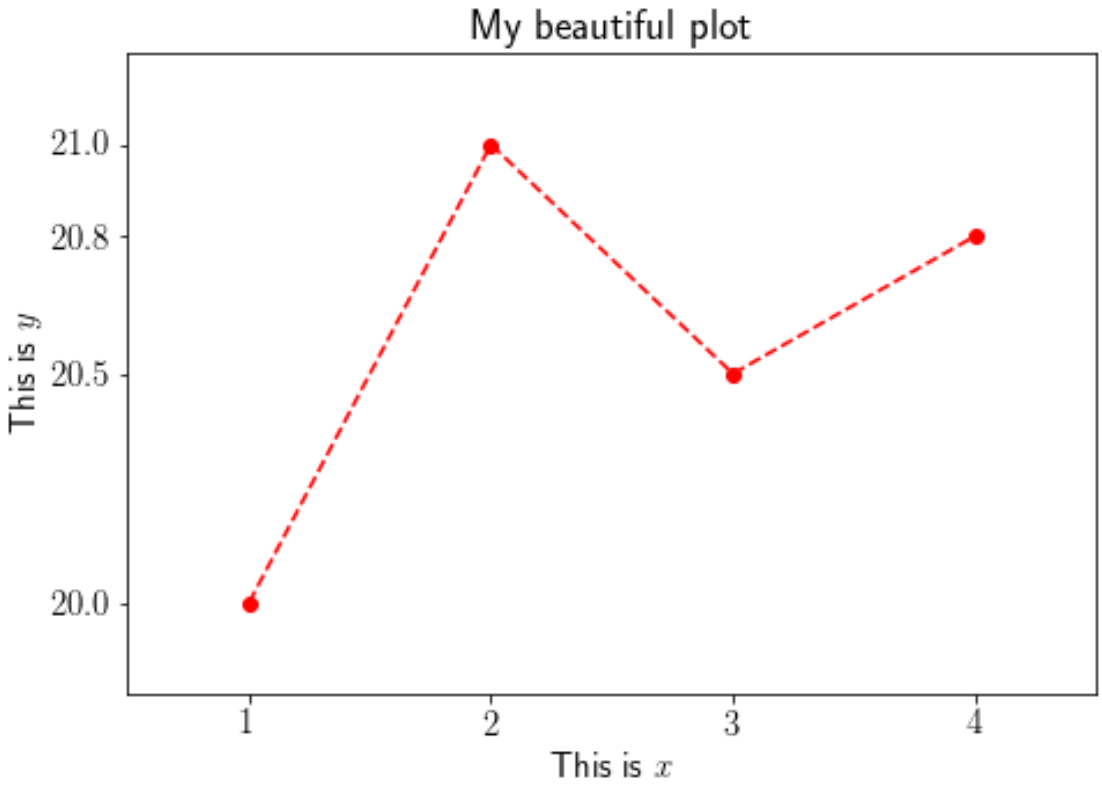



```
In [20]: plot(x, y, linestyle='dashed', marker='o', color='red')
         xlim(0.5, 4.5)
         ylim(19.8, 21.2)

         # now let us manipulate the axes ticks
         xticks([1,2,3,4])
         yticks([20, 21, 20.5, 20.8])

         # and set labels
         xlabel(r"This is $x$")
         ylabel(r"This is $y$")
         title("My beautiful plot")

Out[20]: <matplotlib.text.Text at 0x1134b5438>
```




The data in the plot above is presented very readable.
Now let's take it a step further and include errorbars in our plot.

```
In [23]: # define some data
         x = [1,2,3,4]
         y = [20, 21, 20.5, 20.8]

         # error data
         y_error = [0.12, 0.13, 0.2, 0.1]

         # just as before we plot our data
         plot(x, y, linestyle='dashed', marker='o', color='red')

         # and the we plot the errorbars separately
         errorbar(x, y, yerr=y_error, linestyle="None", marker="None", color="red")

         xlim(0.5, 4.5)
         ylim(19.8, 21.2)

         # now let us manipulate the axes ticks
         xticks([1,2,3,4])
         yticks([20, 21, 20.5, 20.8])
```

```
# and set labels
xlabel(r"This is $x$")
ylabel(r"This is $y$")
title("My beautiful plot")
```

Out[23]: <matplotlib.text.Text at 0x1138bb4a8>


In [ ]: