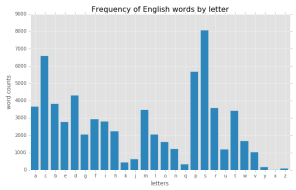
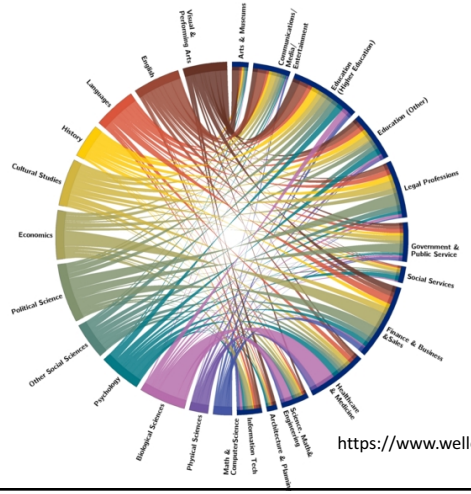
Data Visualization with matplotlib



CS111 Computer Programming

Department of Computer Science
Wellesley College

Visualizing data is important!



What careers do Wellesley graduates choose based on their major?

Can we generate this visualization with cs1graphics? NO. We need a tool that works with **data**.

https://www.wellesley.edu/admission/why/after

Plotting with `matplotlib`

A plot is a graphical technique for representing a **data set**, usually as a graph showing the relationship between two or more variables.

In CS111, we'll be using Python's **matplotlib** library to make plots/graphs/charts.

```
import matplotlib.pyplot as plt
import numpy as np
```

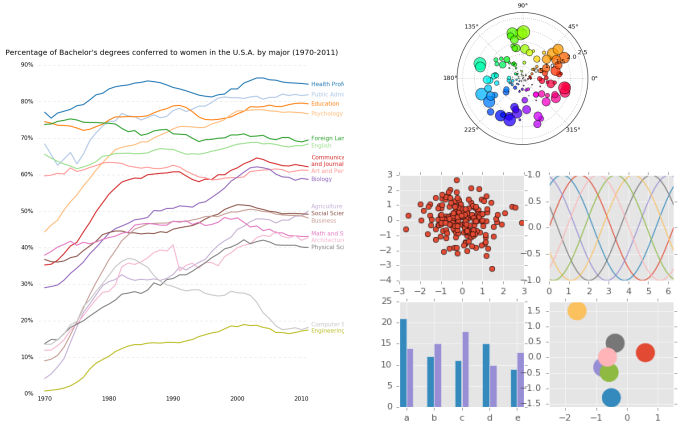
In all our examples, we will need to import the **numpy** and **matplotlib.pyplot** modules.
These come with Canopy – no need to download anything.

Resources

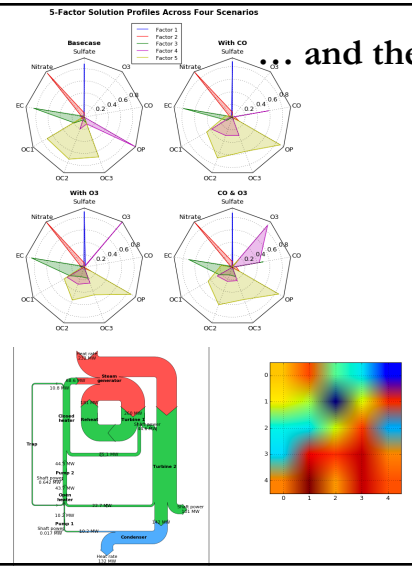
matplotlib examples: http://matplotlib.org/examples

pyplot documentation: http://matplotlib.org/api/pyplot_summary.html

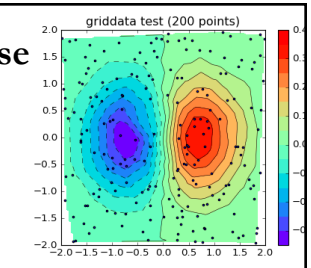
`matplotlib` can make these plots

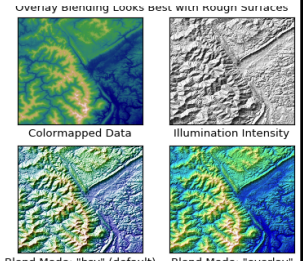


see more at http://matplotlib.org/gallery.html

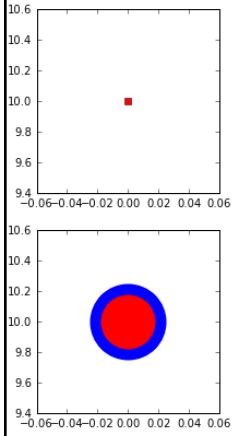


… and these





Starting simple: `plot` function



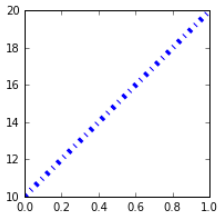
When a single value (e.g. 10) is provided, it's considered the y-value of a data point and is assigned x to 0. We can specify how the marker for the point displays.

```
plt.figure(figsize=(3, 3))
plt.plot(10, color='red', marker='s')
plt.show()
```

The function **plot** specifies many named parameters (or keyword arguments) to control how a data point is displayed.

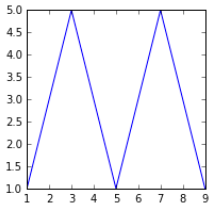
```
plt.figure(figsize=(3, 3))
plt.plot(10, color='red', marker='o',
         markersize=60,
         markeredgecolor='blue',
         markeredgewidth=10)
plt.show()
```


Plotting a line



A list of two values is displayed as line. Again, values are y-coordinates, and by default x is set to [0, 1]. Some arguments have default values, for example, line color is blue by default and the style is solid (see second plot).

```
plt.figure(figsize=(3, 3))
plt.plot([10, 20], linestyle='dashdot',
         linewidth=5)
plt.show()
```



But we can specify two lists: one with the x-coordinates and one with the y-coordinates for the data points. Then they are automatically connected by lines.

```
plt.figure(figsize=(3, 3))
plt.plot([1, 3, 5, 7, 9], [1, 5, 1, 5, 1]
plt.show()
```


The `plot` function

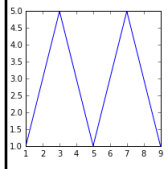
Accessing the documentation of plot via: **help(ptl.plot)** reveals its versatility:

```
In [447]: help(plt.plot)
Help on function plot in module matplotlib.pyplot:

plot(*args, **kwargs)
    Plot lines and/or markers to the
    :class:`~matplotlib.axes.Axes`.  *args* is a variable length
    argument, allowing for multiple *x*, *y* pairs with an
    optional format string.  For example, each of the following is
    legal::

        plot(x, y)        # plot x and y using default line style and color
        plot(x, y, 'bo')  # plot x and y using blue circle markers
        plot(y)           # plot y using x as index array 0..N-1
        plot(y, 'r+')     # ditto, but with red plusses
```

Notice string argument values such as 'bo' , 'r+' and 'g-' for the marker/line appearance (first letter is the color, second character the marker or line style).

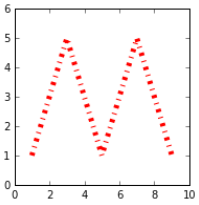
Also, many common keyword arguments have shorter names: color is c, markersize is ms, linewidth is lw, linestyle is ls, etc.

Controlling axes

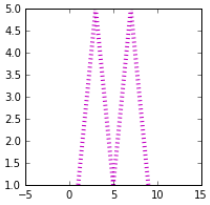


x = [1, 3, 5, 7, 9] and **y = [1, 5, 1, 5, 1]**
The axes automatically fit the provided range values. We can control them in different ways with the functions:

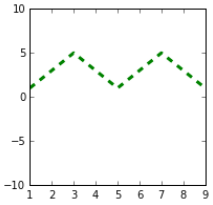
```
axis([xmin, xmax, ymin, ymax]) # also, 'off', and 'equal'
xlim((xmin, xmax))             # takes tuple or list or one or two values
ylim((ymin, ymax))
```



```
plt.plot(x, y, 'r-.', lw=5)
plt.axis([0, 10, 0, 6])
```

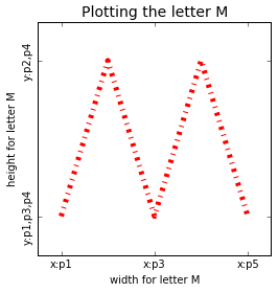


```
plt.plot(x, y, 'm:', lw=4)
plt.xlim((-5, 15))
```



```
plt.plot(x, y, 'g--', lw=3)
plt.ylim((-10, 10))
```


Decorating the Plot



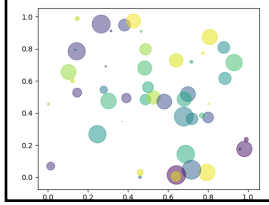
The functions **xticks** and **yticks** control the location and values of ticks on axes.
We can supply text labels for the axes with **xlabel** and **ylabel** and a **title** for the plot.

```
x = [1, 3, 5, 7, 9]
y = [1, 5, 1, 5, 1]
plt.figure(figsize=(4, 4))
plt.plot(x, y, 'r-.', lw=5)
plt.axis([0, 10, 0, 6])
plt.xticks([1, 5, 9],
           ['x:p1', 'x:p3', 'x:p5'])
plt.yticks([1, 5],
           ['y:p1,p3,p4', 'y:p2,p4'],
           rotation=90)
plt.xlabel("width for letter M")
plt.ylabel("height for letter M")
plt.title("Plotting the letter M",
          fontsize=14)
plt.show()
```


More commands and plot types

Before **plt.show**, you can use **plt.savefig("filename")** to save the plot in a file. Use **.png** as file ending. Don't put savefig after show, it saves an empty file.

This is a **scatter** plot with random points and colors. Notice **alpha** to make colors half-transparent. Uses the powerful **numpy** library.



The **plt** module specifies several dedicated functions. We'll study a few of them, for others, check examples online.

bar – bar chart (vertical)
barh – bar chart horizontal
hist – histogram
pie – pie chart
scatter – scatter splot
step – stepwise line
subplots – multiple plots in a figure

```
N = 50
x = np.random.rand(N)
y = np.random.rand(N)
colors = np.random.rand(N)
area = np.pi *(15 * np.random.rand(N))**2
plt.scatter(x, y, s=area, c=colors,
            alpha=0.5)
plt.show()
```


Plotting Overview

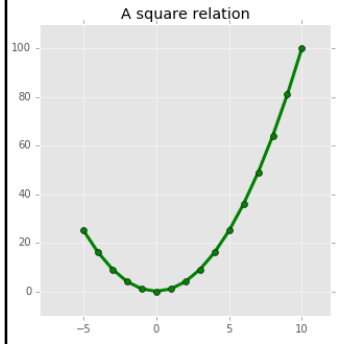
1. Plots are 2-dimensional.
2. We need to provide x and y coordinates for the points to be drawn.
3. We will create plots by using functions that expect two lists of values for x and y.
4. Usually we have a set of points x and we will apply a math function (square, sin, etc.) to get the corresponding y values. Thus, we need the **mapping** pattern.
5. These functions take keyword arguments that control the appearance of the data points: color, marker shape, line type, line width, etc.
6. The pyplot library (**plt** for short) and its functions create and manipulate a single figure object that it's updated every time we invoke a new function.
7. This object is implicit, we don't access it directly. This is why we can write several statements without an explicit mention of this object:

```
plt.plot(x, y)
plt.xlabel("some label")
plt.xlim((10, 50))
plt.show()
```

plt is not an object, it's a module. Verify it with **type(plt)**. You can see the list of all available functions and classes with **dir(plt)**.

Plotting (with list comprehensions)

```
# Preparing data for plotting
xvals = range(-5, 11)
                # [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]
yvals = [x**2 for x in xvals]
                # [25,16,9,4,1,0,1,4,9,16,25,36,49,64,81,100]
```

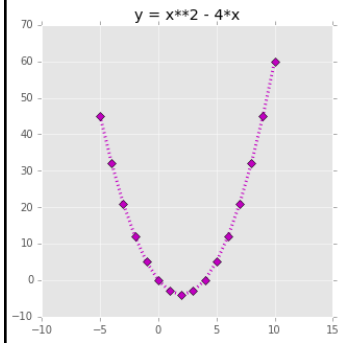



```
plt.style.use('ggplot')
plt.figure(figsize=(5, 5))

plt.plot(xvals, yvals, 'go-', lw=3)

plt.axis([-8, 12, -10, 110])
plt.title("A square relation")
plt.show()
```

The statement:
`plt.style.use('ggplot')`
controls the general style of a plot, we'll use at the start of a script file.

More plotting (with list comprehensions)


```
xvals = range(-5, 11)
                # [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,8,9,10]
```



Given the mapping function:

```
y = x**2 - 4*x
```

generate the plot that shows the relation between **x** and **y**.

To make it easier for you, we're showing you how it should look like.

The string value for the line/marker style is 'm:D' short for "magenta, dotted line, diamond marker"