COMP 364: Computer Tools for Life Sciences

Using libraries: NumPy & Data visualization with MatPlotLib

Christopher J.F. Cameron and Carlos G. Oliver

Key course information

Midterm

- how was it? too hard? too easy?
- if you could not write the midterm
  - please contact Chris or Carlos ASAP

Quiz #5

- will be available October 30th, 2017
- access closes at 11:59:59 pm on the same day
- MC questions will cover topics from the last two weeks

Assignment #3

- coming soon!

Whitespace cheatsheet

Common whitespace characters:

- '_' or ' '- space
- '\t' - a tab
- '\n' - newline character ('\r\n' in Window environments)
- '\r' - carriage return (move to the beginning of the line)
- '\f' - form feed (advance to next page)
- '\s' - any whitespace

Data

Let's think back to last Friday...

- randomly chose 10 genes and two time points
- learned how to label genes based on a similarity measure
- in our case, gene expression across two chosen time points
- then by cluster analysis, we grouped genes together

```
start,end = [170 ,240]
gene_names = ["YMR274C", "YOR263C", "YLR371W",
              "YDL120W", "YKL096W", "YHR192W",
              "YMR046C", "YDR038C", "YMR172W",
              "YDL045C"]
obs = [(-0.24, -0.03), (-0.22, 0.25),
       (0.06, 0.1), (-0.02, -0.57),
       (0.66, -0.73), (0.24, 0.34),
       (0.17, -0.75), (0.59, 0.27),
       (-0.08, 0.18), (0.07, 0.23)]
cluster_labels = [0, 0, 1, 2, 2, 1, 2, 1, 0, 1]
```

Cluster analysis - *k-means*

In our last lecture, we ran the *k-means* algorithm

- using an implementation from Python's SciPy module
  - **scipy.cluster.vq.kmeans2()**

k-means algorithm:

Step 1 - randomly choose k centroids

Step 2 - for each data point, assign it to the nearest centroid

Step 3 - recalculate each cluster's new centroid

Step 4 - repeat step 2

Step 5 - if data points have changed clusters, go to step 3
if no data points have been reassigned, stop

Example - `https://www.youtube.com/watch?v=5I3Ei69I40s`

Matplotlib

Having numerical data is useful, but...

- sometimes it is more practical to visualize data in a plot

To do this we are going to use Python's **Matplotlib** module

- allows for 2D and 3D plotting
- provides useful functionality for Python to work like MATLAB
  - another useful programming language

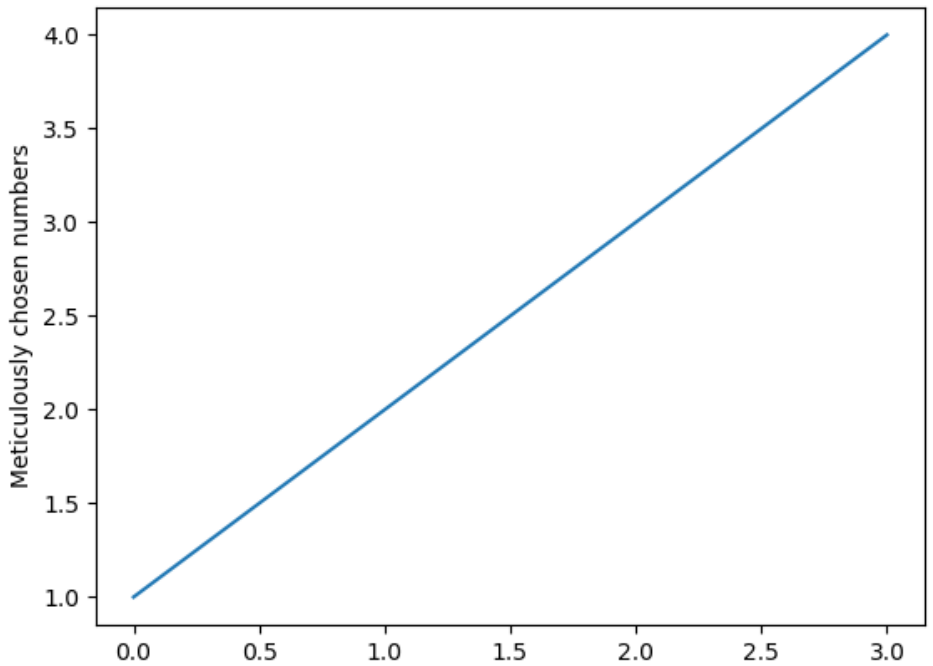
Let's start by importing the module:

```
1 import matplotlib.pyplot as plt
```

Matplotlib #2

Let's create a simple plot:

```
1 import matplotlib.pyplot as plt
2 plt.plot([1,2,3,4])
3 plt.ylabel("Meticulously chosen numbers")
4 plt.show()  # displays figure
```



Matplotlib #3

Why does the x-axis ranges from 0-3 and the y-axis from 1-4?

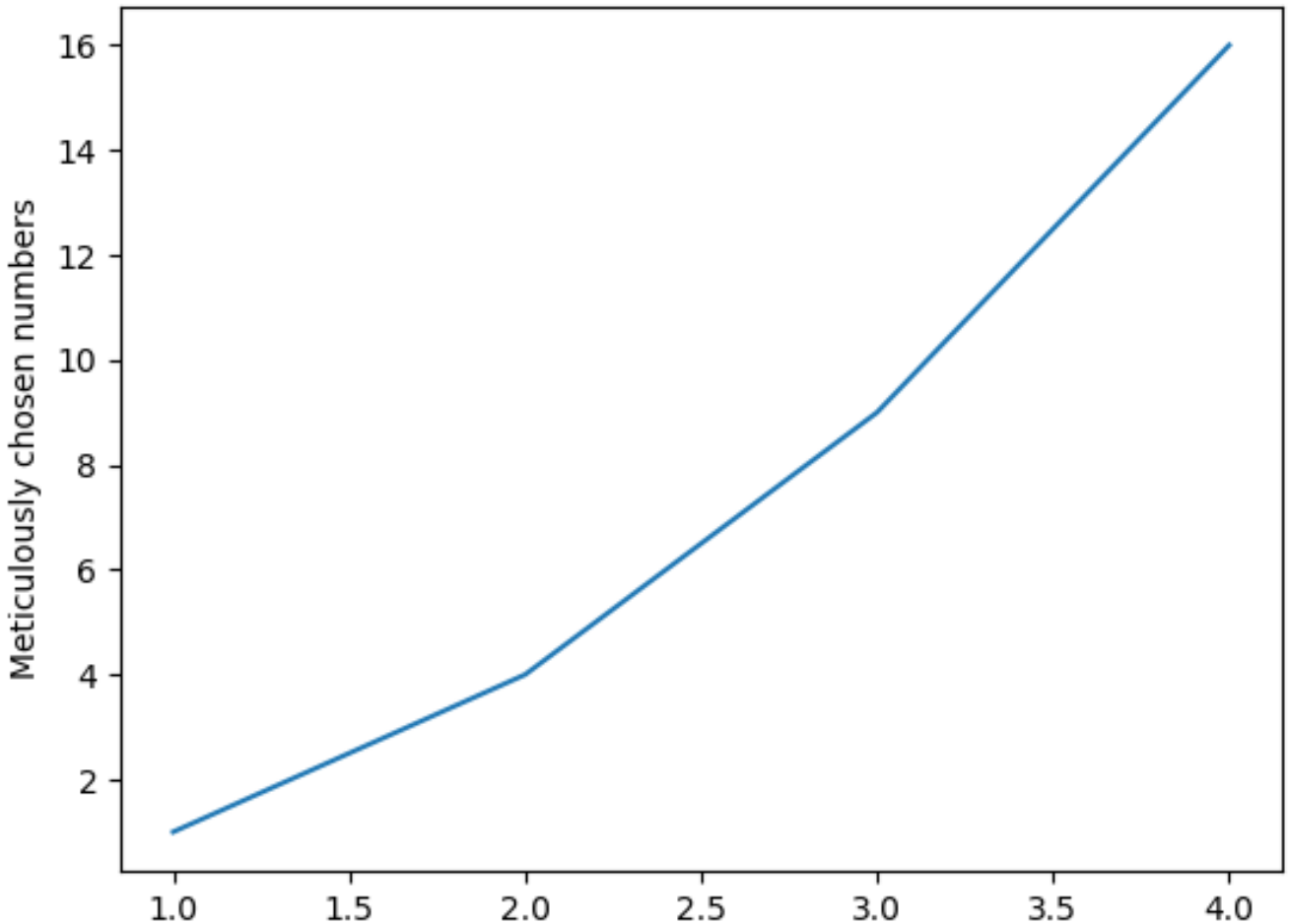
If you provide a single list or array to the **.plot()** command

- matplotlib assumes it is a sequence of y values
- and automatically generates the x values for you
- python ranges start with 0
- the default x vector has the same length as y but starts with 0

To plot x versus y:

```
1 plt.plot([1, 2, 3, 4], [1, 4, 9, 16])
```

Matplotlib #4



Jupyter notebooks

To use matplotlib in the Jupyter notebooks

- add '**%matplotlib inline**'
- before the import statement of the module

```
1 %matplotlib inline
2
3 import matplotlib
4 import matplotlib.pyplot as plt
```

You could also change the IPython kernels

- but this is outside the scope of the course
- `https://stackoverflow.com/questions/19410042/how-to-make-ipython-notebook-matplotlib-plot-inline`

Useful matplotlib functions

To save a figure to file use **.savefig()**

- `https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.savefig.html`

When producing multiple plots

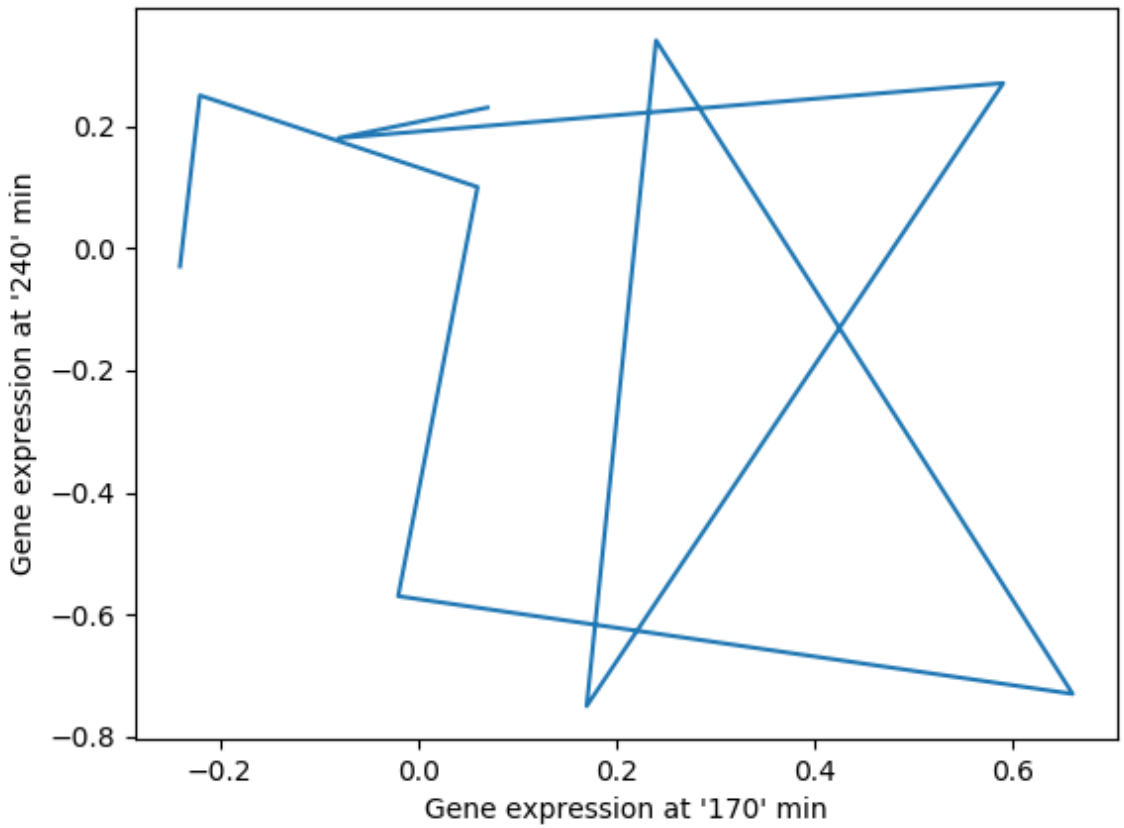
- make sure to close the previous one using **.close()**
- `https://matplotlib.org/devdocs/api/_as_gen/matplotlib.pyplot.close.html`

Let's try plotting our gene expression data from before

Plotting gene expression data

```
# split obs by x and y vals
x_vals,y_vals = zip(*obs)
plt.plot(x_vals,y_vals)
# set x and y labels
plt.xlabel("Gene expression at '"+str(start)+"' min")
plt.ylabel("Gene expression at '"+str(end)+"' min")
# write figure to file
plt.savefig("./gene_figure.png")
# close matplotlib figure
plt.close()
```

Plotting gene expression data #2



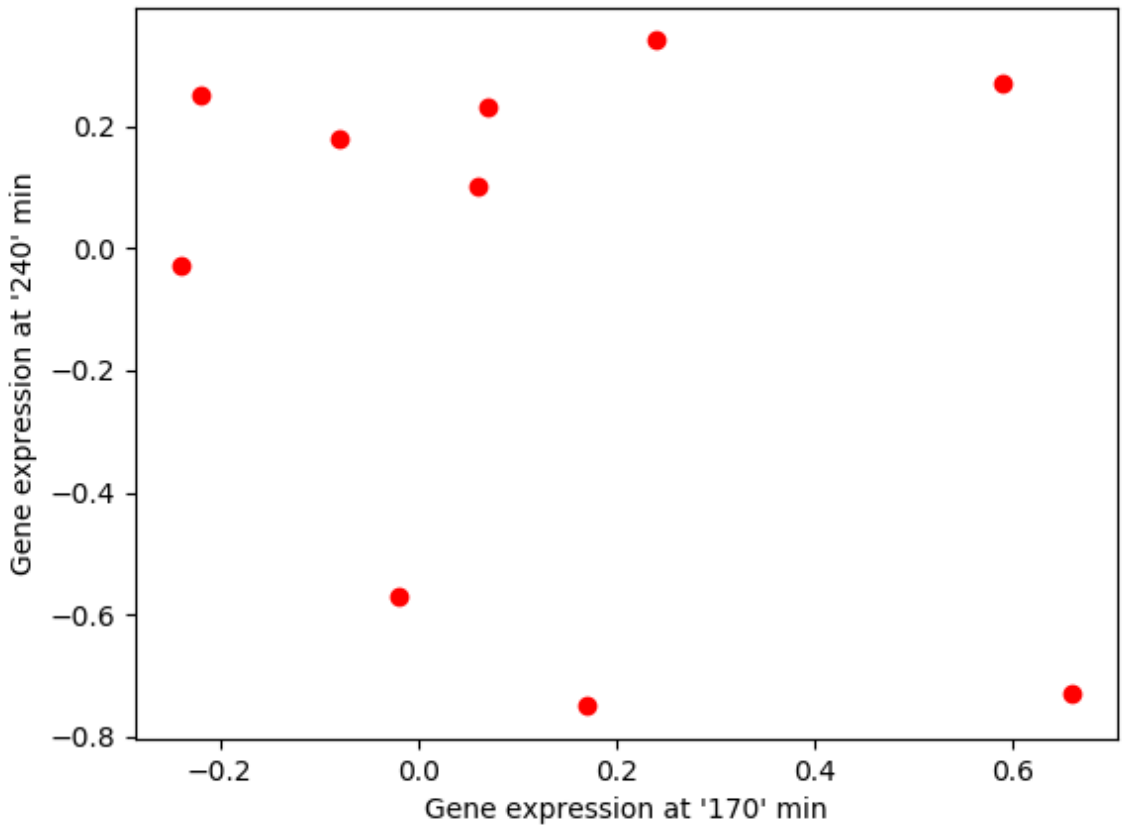
That doesn't quite look right

- let's look at the matplotlib API

Plotting gene expression data #3

```
# split obs by x and y vals
x_vals,y_vals = zip(*obs)
plt.plot(x_vals,y_vals,"ro")
# set x and y labels
plt.xlabel("Gene expression at '"+str(start)+"' min")
plt.ylabel("Gene expression at '"+str(end)+"' min")
# write figure to file
plt.savefig("./gene_figure_2.png")
# close matplotlib figure
plt.close()
```

Plotting gene expression data #4



That's better, but how can we colour data points by cluster?

- let's look at the matplotlib API

Colours in matplotlib

Matplotlib functions can handle many different colour codes:

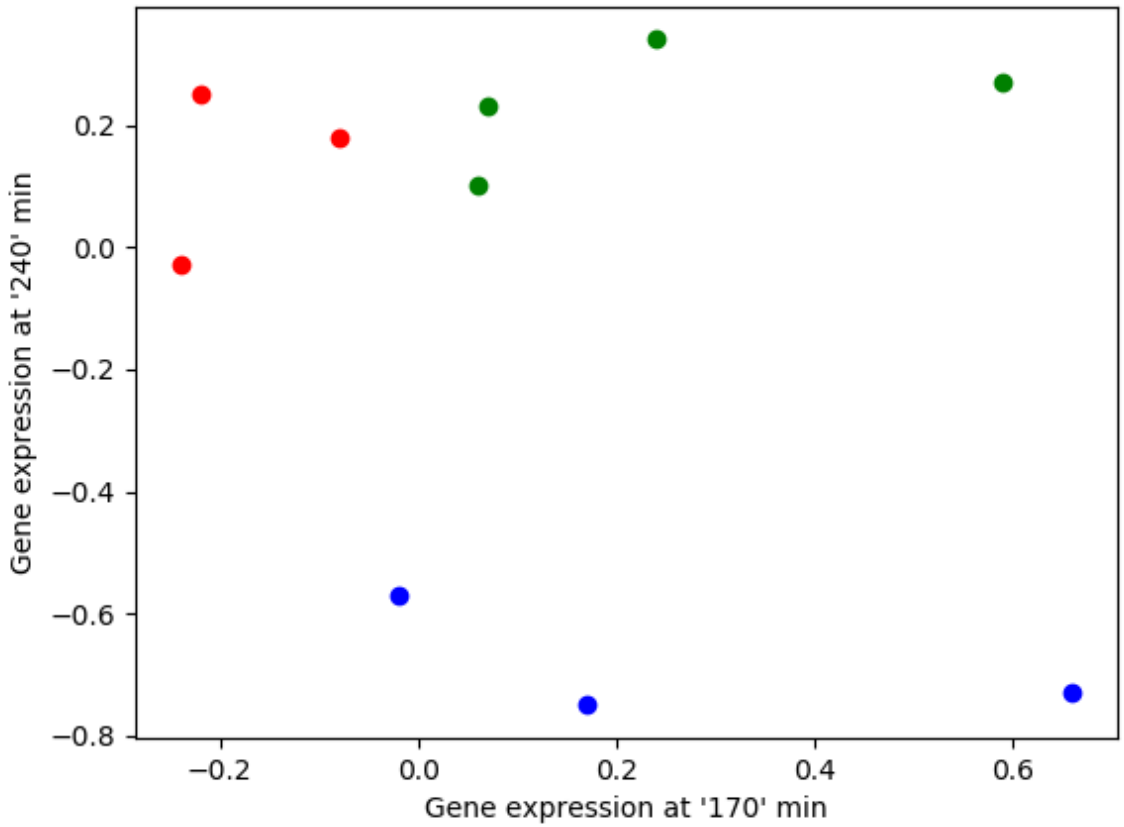
1. RGB
   - '(0, 0, 0)' - black
2. hex RGM
   - '#0F0F0F' - black
3. character
   - 'b' - blue
   - 'k' - black
   - 'r' - red
4. many more...

For more information, see:
`https://matplotlib.org/api/colors_api.html`

Plotting gene expression data #5

```
# define set of colors to use
colors = ["r", "g", "b"]
# get set of labels used
labels = set(cluster_labels)
# iterate over labels and plot subsets
for label in labels:
    color = colors[label]
    vals = []
    for i,cluster in enumerate(cluster_labels):
        if(cluster == label):
            # index obs for cluster label
            vals.append(obs[i])
    x_vals,y_vals = zip(*vals)
    # plot current cluster, based on label
    plt.plot(x_vals,y_vals,"o",color=color)
```

Plotting gene expression data #6



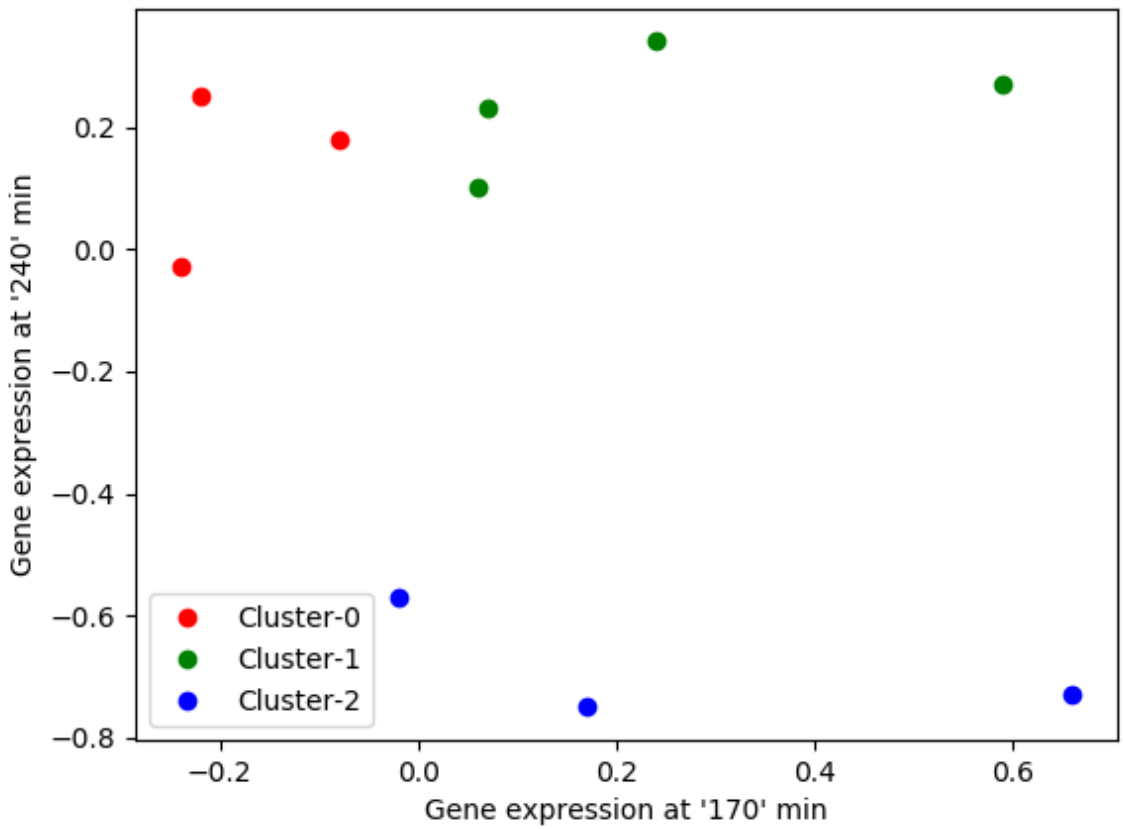
Even better, let's add a legend

- let's look at the matplotlib API

Plotting gene expression data #7

```
# iterate over labels and plot subsets
for label in labels:
    color = colors[label]
    vals = []
    for i,cluster in enumerate(cluster_labels):
        if(cluster == label):
            # index obs for cluster label
            vals.append(obs[i])
    x_vals,y_vals = zip(*vals)
    # plot current cluster, based on label
    plt.plot(x_vals,y_vals,"o",
             color=color,
             label="Cluster-"+str(label))
plt.legend(loc="best")
```

Plotting gene expression data #8



Gene expression across a time course

Let's try something a little more difficult

- begin by reading in all expression data for chosen genes

```
data_dict = {}
with open("./Spellman.csv","r") as f:
    header = f.readline().rstrip().split(",")
    time_points = [int(val) for val in header[1:]]
    for line in f:
        line = line.rstrip().split(",")
        gene_name = line[0]
        # if gene was randomly chosen
        if(gene_name in gene_names):
            exps = [float(val) for val in line[1:]]
            data_dict[gene_name] = exps
```

Gene expression across a time course #2

Okay, now we have a Python dictionary containing:

- keys that represent gene names
- values containing a gene's expression across the time course

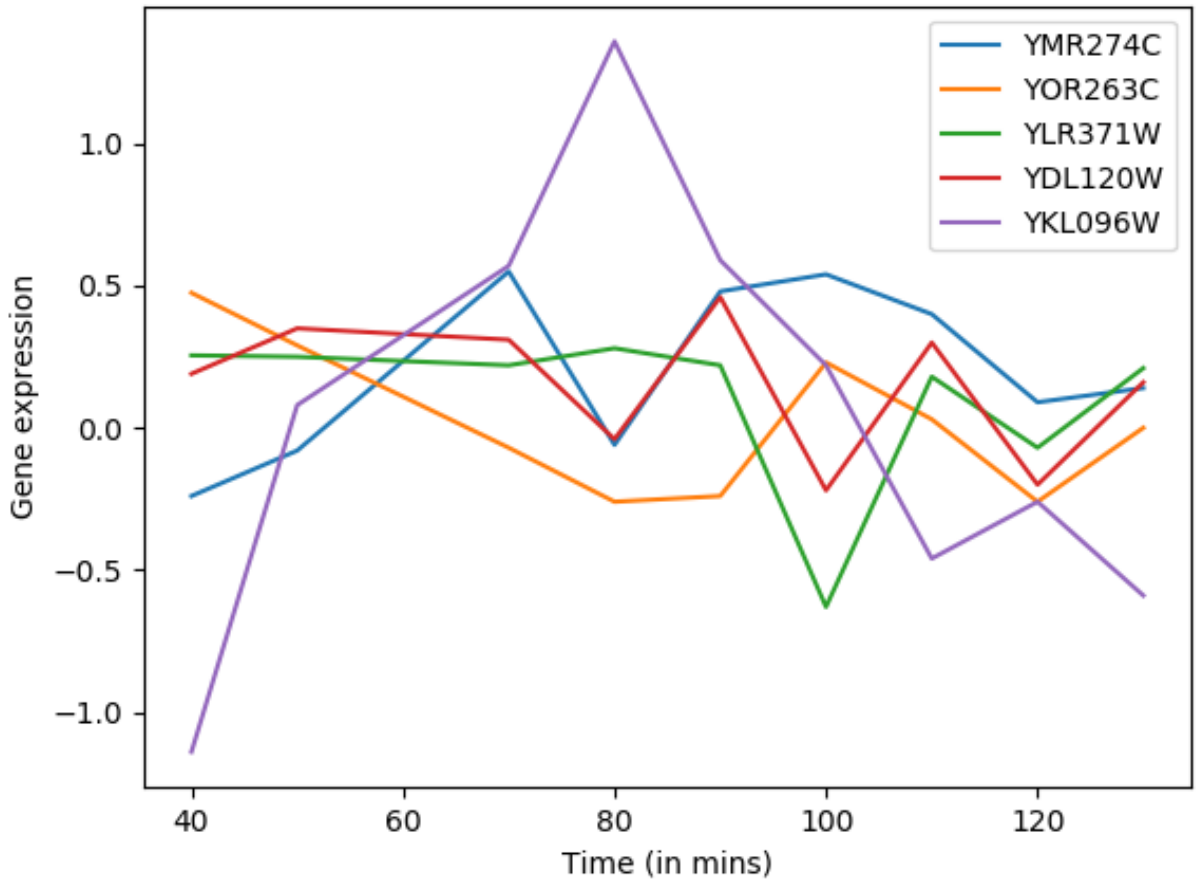
```
gene_name = "YMR046C"
print(data_dict[gene_name])
#prints:
#[0.0, -0.09, 0.145, 0.38, 0.1, 0.02, 0.03, 0.43,
# -0.56, 0.38, -0.32, 0.23, -0.5, 0.17, 0.12, 0.21,
# -0.09, 0.09, -0.17, 0.11, -0.75, -0.11, 0.045]
```

How can we display the expression of these genes as a line plot?

Gene expression across a time course #3

```
# iterate over genes
for gene_name in gene_names[:5]:
    # plot corresponding time points and
    # gene expression
    plt.plot(time_points[:10],data_dict[gene_name][:10],
             label=gene_name)
plt.ylabel("Gene expression")
plt.xlabel("Time (in mins)")
plt.legend(loc="best")
plt.savefig("./gene_figure.png")
plt.close()
```

Gene expression across a time course #4



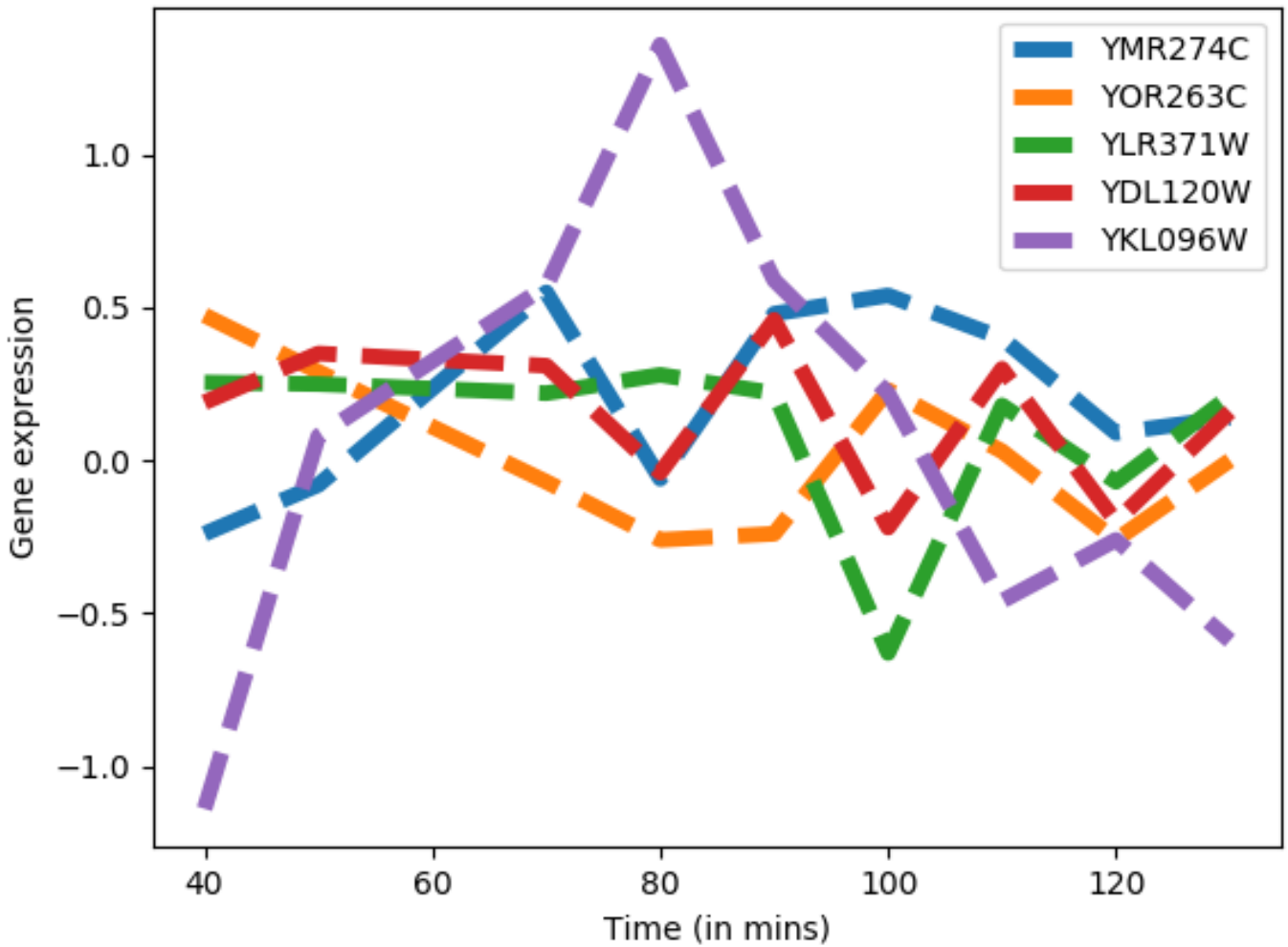
Can I make the lines thicker? Different style?

- let's look at the matplotlib API

Gene expression across a time course #5

```
# iterate over genes
for gene_name in gene_names[:5]:
    # plot corresponding time points and
    # gene expression
    plt.plot(time_points[:10],data_dict[gene_name][:10],
             label=gene_name,ls="--",lw=5)
plt.ylabel("Gene expression")
plt.xlabel("Time (in mins)")
plt.legend(loc="best")
plt.savefig("./gene_figure.png")
plt.close()
```

Gene expression across a time course #6



Next time in COMP 364

Continuing our plunge into Python's Matplotlib module

Integrating Python's **NumPy** module

- a fundamental package for scientific computing, which contains:
  - a powerful N-dimensional array object
  - useful linear algebra and random number capabilities

NumPy API:
`https://docs.scipy.org/doc/numpy-1.13.0/reference/`