

# MAS212 Scientific Computing and Simulation

Dr. Sam Dolan

School of Mathematics and Statistics,  
University of Sheffield

Autumn 2017

<http://sam-dolan.staff.shef.ac.uk/mas212/>

G18 Hicks Building  
s.dolan@sheffield.ac.uk

# Today's lecture

- Scientific computing modules:
  - numpy
  - matplotlib
  - scipy
- Basic plotting
- Example #0: Lissajous curves
- Example #1: Monte Carlo integration
- Example #2: The logistic map
- Example #3: Quadratic maps and fractals

$$z \rightarrow z^2 + c$$

# matplotlib

- matplotlib is a 2D plotting library for Python
- Home page: <http://matplotlib.org/>
- matplotlib.pyplot is a collection of command-style functions that make matplotlib work like MATLAB
- Tutorial:  
[http://matplotlib.org/users/pyplot\\_tutorial.html](http://matplotlib.org/users/pyplot_tutorial.html)

# Plotting

## Convention:

Import pyplot as follows:

```
>>> import matplotlib.pyplot as plt
```

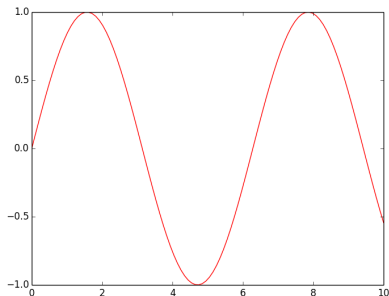
## Jupyter Notebook

To include plots in your Jupyter Notebook, use:

```
%matplotlib inline
```

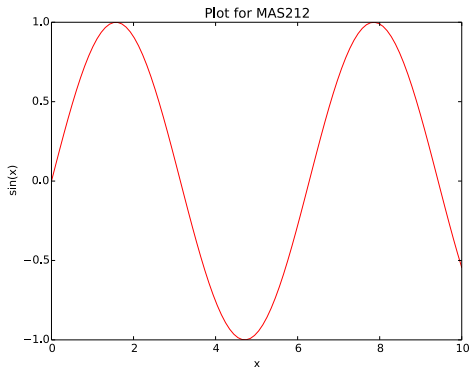
# Plotting: Simple Example

```
>>> import numpy as np
>>> import matplotlib.pyplot as plt
>>> xs = np.linspace(0.0, 10.0, 100) # 100 points
>>> ys = np.sin(xs)
>>> plt.plot(xs, ys, 'r-') # red line
>>> plt.savefig('sin.png') # save to file
>>> plt.show()
```



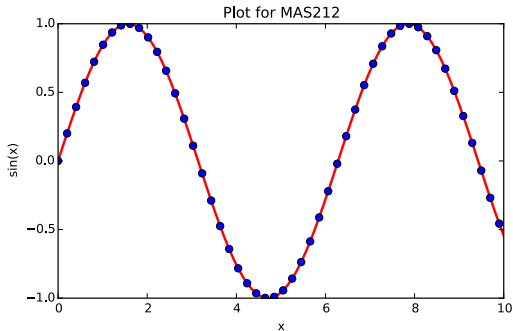
# Plotting: Axis Labels

```
>>> plt.xlabel("x")  
>>> plt.ylabel("sin(x)")  
>>> plt.title("Plot for MAS212")
```



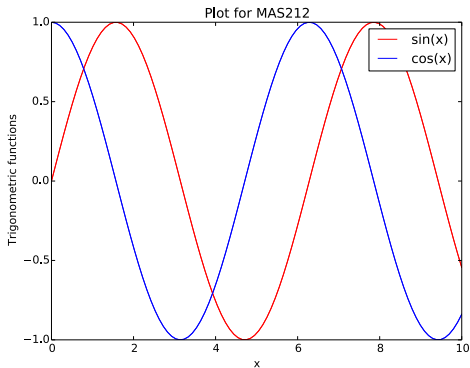
## Plotting: Thicker lines. Points.

```
>>> plt.plot(xs, ys, 'r-', linewidth=2.0) # thicker red line  
>>> plt.plot(xs[::2], ys[::2], 'bo')  
# plot every other point as a blue filled circle
```



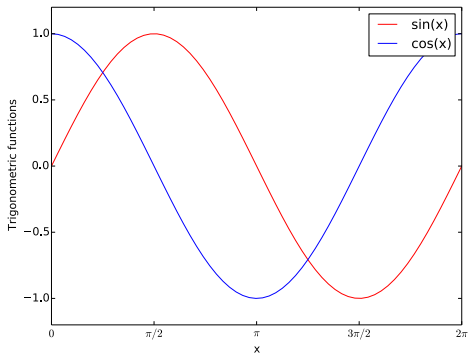
## Plotting: Add a legend

```
>>> plt.plot(xs, np.sin(xs), 'r-', label="sin(x)") # red line
>>> plt.plot(xs, np.cos(xs), 'b-', label="cos(x)") # blue line
>>> plt.ylabel("Trigonometric functions")
>>> plt.legend() # add a legend to the plot
```





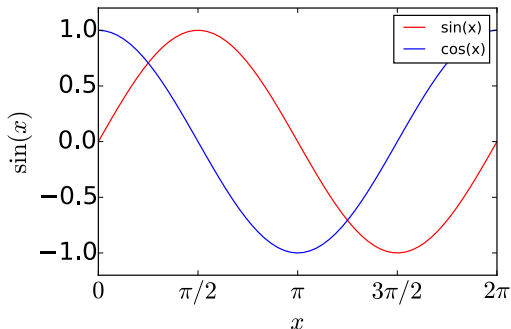
```
>>> plt.axis([0, 2.0*np.pi, -1.2, 1.2])
>>> # now set the ticks and labels on the x-axis.
>>> xticks = [0, np.pi/2.0, np.pi, 3*np.pi/2.0, 2*np.pi]
>>> xlabel = ['$0$', '$\pi/2$', '$\pi$', '$3\pi/2$', '$2\pi$']
>>> plt.xticks(xticks, xlabel)
```



# How to improve readability?

## 1. Enlarge text elements using fontsize optional argument:

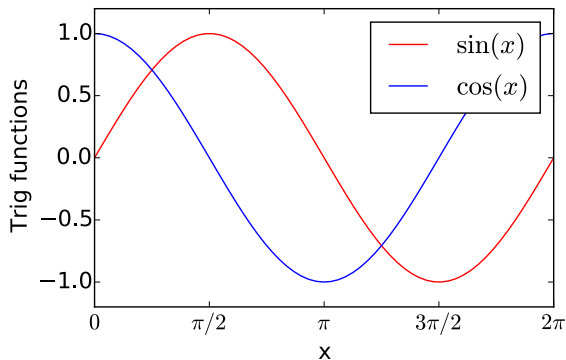
```
>>> plt.xticks(fontsize=20)
>>> plt.yticks(fontsize=20)
>>> plt.xlabel("$x$", fontsize=20)
>>> plt.ylabel("$\sin(x)$", fontsize=20)
```



# How to improve readability?

## 2. Or: change default font size:

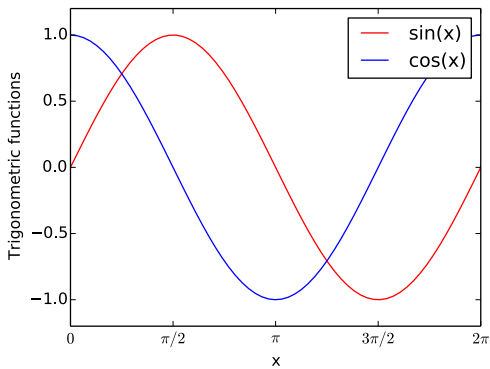
```
>>> plt.rcParams.update({'font.size': 16})
```



## How to improve readability?

3. Or: Make all labels and lines bigger by making the plot smaller!

```
>>> F = plt.gcf() # get current figure
>>> size = F.get_size_inches()
>>> fac = 0.7
>>> F.set_size_inches(size[0]*fac, size[1]*fac)
```



## Example #0: Lissajous curves

- A Lissajous curve is the graph of a system of parametric equations:

$$x = A \sin(at + \delta), \quad y = B \sin(bt)$$

where  $a, b, A, B, \delta$  are constants and  $t$  is the parametric variable

- The appearance depends chiefly on the ratio  $a/b$ 
  - $a/b = 1 \quad \Rightarrow$  ellipses, circles, lines
  - $a/b = 2, \delta = \pi/4 \quad \Rightarrow$  parabola
  - $a/b \in \mathbb{Q} \quad \Rightarrow$  closed curve
- If  $a/b$  is irrational, the curve does not close.

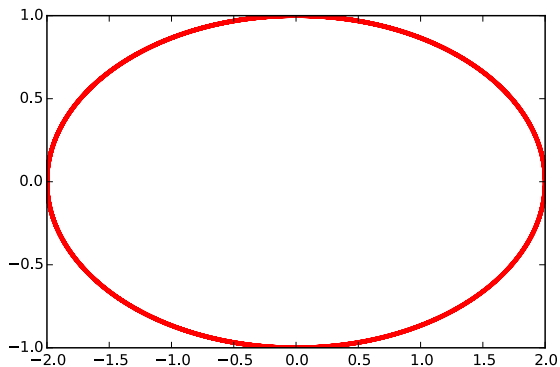
## Example #0: Lissajous curves

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

def plot_lissajous(b=1, a=1, A=2, B=1, d=np.pi/2):
    ts = np.linspace(0,30,600)
    plt.plot(A*np.sin(a*ts+d), B*np.sin(b*ts), 'r-', linewidth=3)
```

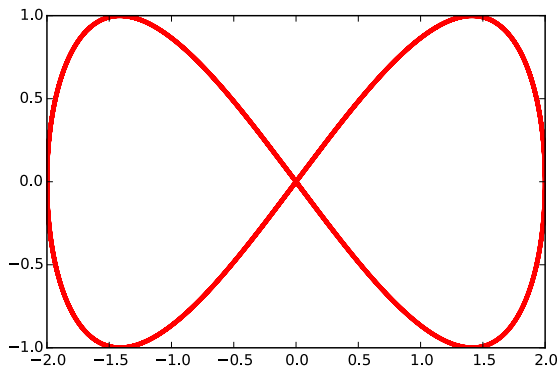
## Example #0: Lissajous curves

```
plot_lissajous(b=1)
```



## Example #0: Lissajous curves

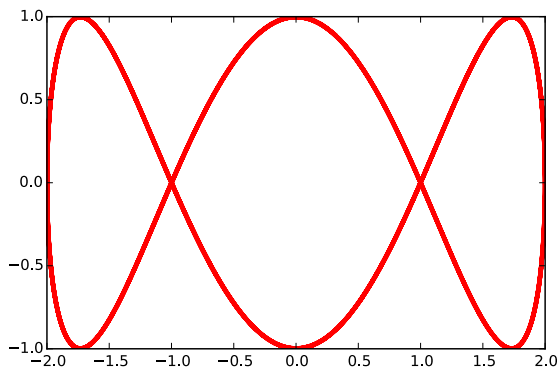
```
plot_lissajous(b=2)
```





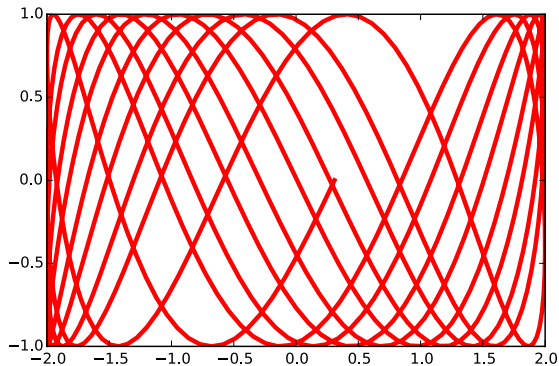
## Example #0: Lissajous curves

```
plot_lissajous(b=3)
```



## Example #0: Lissajous curves

```
plot_lissajous(b=np.pi)
```



- $b = \pi$  is **irrational**  $\Rightarrow$  curve does not close.

## Figures and subplots

- Plots are contained within a `Figure` object.
- Create a new figure with

```
fig = plt.figure()
```

- To add a new subplot to a figure:

```
ax1 = fig.add_subplot(2,2,1)
```

(This will add the first subplot to a  $2 \times 2$  array of subplots).

- `ax1` is an `AxesSubplot` object
- `plot` function belongs to `AxesSubplot` object rather than `Figure`

# Figures and subplots

- Useful functions:

`plt.gcf()` Get current figure

`plt.gca()` Get current axes

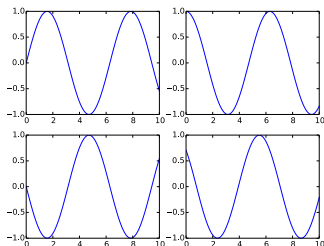
`plt.clf()` Clear figure

`plt.cla()` Clear axes

`plt.close()` Close plotting window

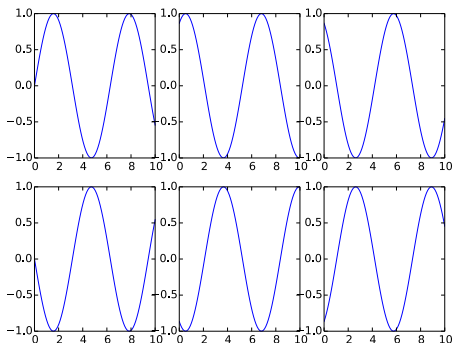
# Multiple plots: method #1

```
>>> xs = np.linspace(0,10,100)
>>> f = plt.figure()
>>> ax = f.add_subplot(2,2,1)
>>> ax.plot(xs, np.sin(xs))
>>> ax = f.add_subplot(2,2,2)
>>> ax.plot(xs, np.sin(xs+np.pi/2.0))
>>> ax = f.add_subplot(2,2,3)
>>> ax.plot(xs, np.sin(xs+np.pi))
>>> ax = f.add_subplot(2,2,4)
>>> ax.plot(xs, np.sin(xs+3.0*np.pi/4.0))
```



## Multiple plots: method #2

```
>>> fig, axes = plt.subplots(2,3) # create 2 x 3 array of subplots
>>> axs = axes.flatten() # turn into 1D array.
>>> for i in range(len(axs)):
...     axs[i].plot(xs, np.sin(xs + 2.0*np.pi*i/len(axs)))
... 
```



## Multiple plots: method #2

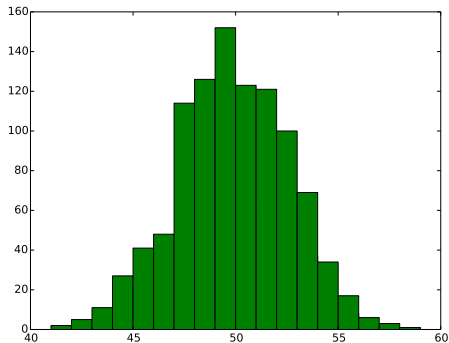
- I prefer method #2

```
>>> fig, axes = plt.subplots(2,3)
```

- This line returns a reference to the Figure, and to a  $2 \times 3$  array of AxesSubplot objects
- Now refer to subplots by position in array, e.g. `axes[0,1]`.
- In the example above I looped over the array to quickly generate the subplots

# Histograms

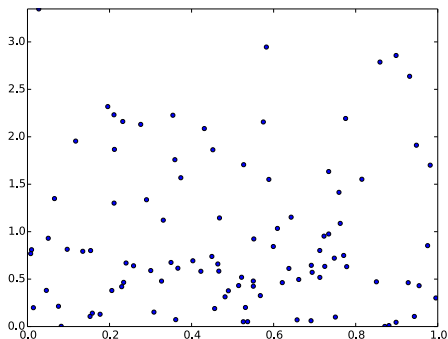
```
>>> samplesize = 100  
>>> Npts = 1000  
>>> data = [sum(np.random.rand(samplesize)) for k in range(Npts)]  
>>> plt.hist(np.array(data), bins=20, range=(40,60))
```





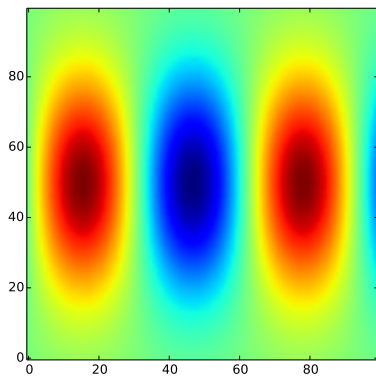
# Scatter plots

```
>>> N = 100    # number of points
>>> xs = np.random.rand(N)    # uniform distribution [0,1]
>>> ys = -np.log(np.random.rand(N))    # Exponential distribution
>>> plt.axis([0,1,0,max(ys)])
[0, 1, 0, 3.3473478211703664]
>>> plt.scatter(xs, ys)
```



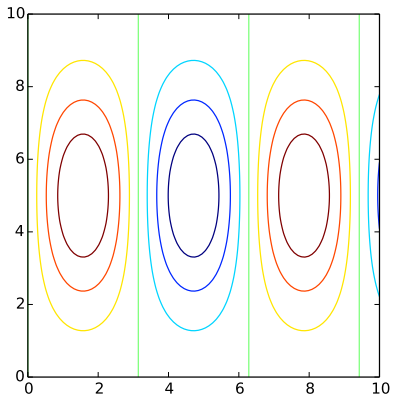
## Colour map plots

```
>>> xs = np.linspace(0, 10, 100)
>>> y1 = np.sin(xs)
>>> y2 = np.exp(-(xs - 5.0)**2 / 10.0)
>>> zs = y1.reshape(len(y1), 1) * y2.reshape(1, len(y2))
>>> plt.gcf().set_size_inches(6,6) # square
>>> plt.imshow(zs.T, origin='lower')
```



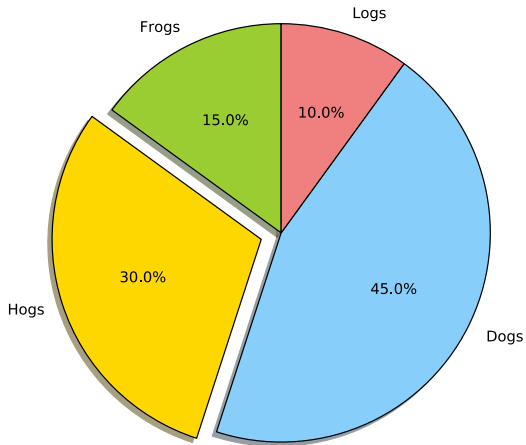
# Contour plots

```
>>> plt.contour(xs, xs, z.T, origin='lower')
```

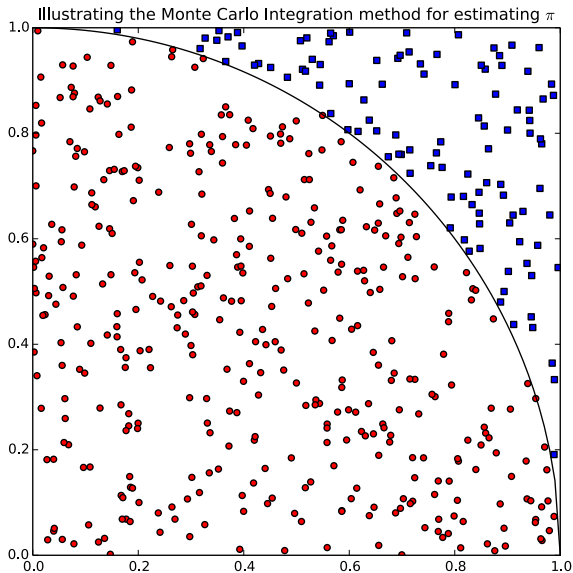


## Pie charts

More example code at <http://matplotlib.org/examples/>



# Example #1: Monte Carlo integration



## Example #1: Monte Carlo integration

### Algorithm:

- Generate a point in the unit square by drawing  $x, y$  from uniform distribution  $[0, 1]$
- If  $x^2 + y^2 \leq 1$ , the point is inside the circle; else, outside.
- Let  $n =$  num of points inside;  $N =$  total number of points;
- Area of unit circle,

$$A = \pi \approx \frac{4n}{N}$$

- Hence we may use random numbers to estimate  $\pi$ .

```
import numpy as np
def estimate_pi(N):
    a = np.random.rand(2*N)
    b = a.reshape(2,N)
    c = b[0]**2 + b[1]**2
    return 4.0*c[c < 1.0].size / float(N)
```

```
>>> estimate_pi(10000)
3.1584
>>> estimate_pi(100000)
3.14348
>>> estimate_pi(1000000)
3.144024
>>> estimate_pi(10000000)
3.1419672
```

## Example #1: Monte Carlo integration

- Suppose we use  $N$  points to estimate  $\pi$
- What is the error  $\sigma$  in the estimate?

$$\sigma \propto \frac{1}{\sqrt{N}}.$$

as expected from **central limit theorem**.

- Inefficient method for low-dimensional integrals
- May be useful for  $d$ -dimensional integration when  $d \gg 1$ :

$$\sigma \sim N^{-1/2} \quad \text{vs.} \quad \sigma \sim N^{-k/d}$$

where  $k$  is the order of the standard integration method.



## Example #2: The Logistic Map

A simple **non-linear** recurrence relation:

$$x \rightarrow \mu x(1 - x)$$

where  $\mu$  is a parameter.

Originally used to model population dynamics.

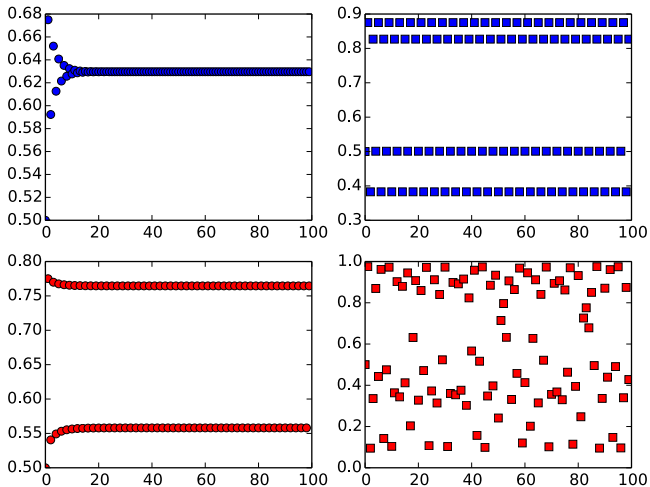
## Example #2: The Logistic Map

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

def logisticmap(mu=2.0, N=100):
    xs = 0.5*np.ones(N)
    for i in np.arange(N-1):
        xs[i+1] = mu*xs[i]*(1.0-xs[i])
    return xs

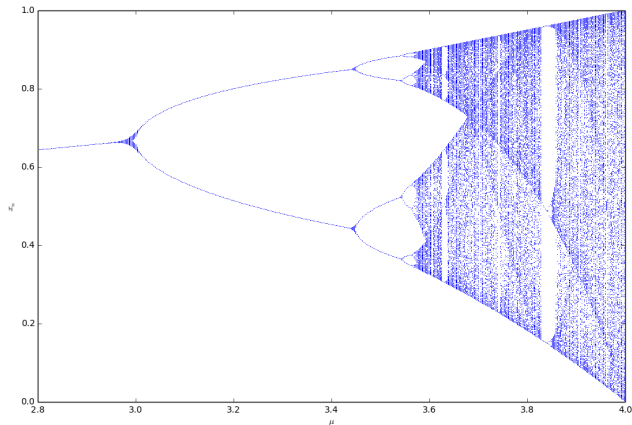
fig, ax = plt.subplots(2,2)
ax[0,0].plot(logisticmap(2.7), 'bo')
ax[1,0].plot(logisticmap(3.1), 'ro')
ax[0,1].plot(logisticmap(3.5), 'bs')
ax[1,1].plot(logisticmap(3.9), 'rs')
plt.show()
```

## Example #2: The Logistic Map



- To plot pixels:

```
plt.plot(data, ',') # comma
```



This is the bifurcation diagram for the logistic map.

## Example #3: Quadratic maps $z \rightarrow z^2 + c$

- Consider an iterative map  $z_{k+1} = f(z_k)$  generating a sequence  $z_0, z_1, z_2, \dots$  where

$$f(z_k) = z_k^2 + c, \quad z_k, c \in \mathbb{C},$$

- The behaviour of the sequence depends on the starting value  $z_0$  and the complex constant  $c$
- Some possibilities: the sequence may
  - converge on an attractive **fixed point**,  $f(z) = z$
  - repeat a sequence of period  $p$ , such that  $f^p(z) = z$
  - **diverge** towards infinity, or
  - neither converge nor diverge but wander around

### Definition: Julia set

The **Julia set** is the closure of the set of repelling periodic points.

## Example #3: Quadratic maps $z \rightarrow z^2 + c$

Consider the special case  $c = 0$ :

- Three possibilities:
  - 1  $|z_0| < 1$  : the sequence converges towards the **fixed point** at the origin
  - 2  $|z_0| > 1$  : the sequence diverges,  $|z_k| \rightarrow \infty$
  - 3  $|z_0| = 1$  : the sequence remains on the unit circle.
- For  $|z_0| = 1$ , if  $\arg(z_0) \in \mathbb{Q}$  the sequence is periodic, otherwise it is aperiodic.
- The Julia set is the unit circle.
- What happens when  $c \neq 0$ ?

## Example #3: Quadratic maps $z \rightarrow z^2 + c$

### Exercise

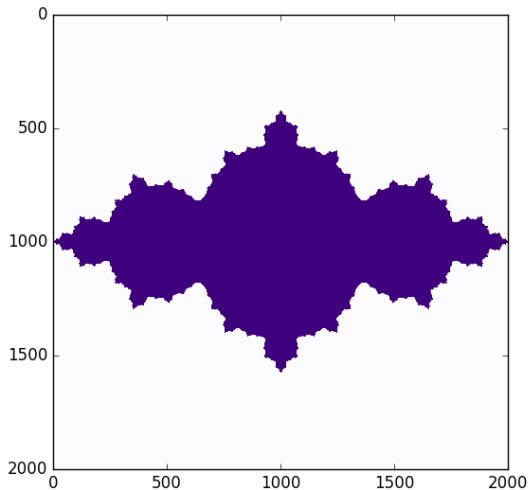
- Make an  $N \times N$  array of  $z_0$  values, linearly-spaced across the complex domain  $-d \leq \text{Re}(z_0) \leq d$ ,  $-d \leq \text{Im}(z_0) \leq d$  for  $d = 1.5$  (say)
- Apply the quadratic map  $N$  times to every value in the array.
- Construct the set of points  $S$  satisfying the condition  $|z_N| < \alpha$ , for  $\alpha = 2$  (say).
- Colour the region  $S$  on the Argand diagram, using `plt.imshow()`

## Example #3: Quadratic maps $z \rightarrow z^2 + c$

```
N = 2000
d = 1.5
xs = np.linspace(-d, d, N)
ys = np.linspace(-d, d, N)*1j
# use broadcasting to make an NxN array from rows/columns
zs = xs.reshape((1, N)) + ys.reshape((N, 1))
c = -0.75
for k in range(10): # iterate
    zs = zs**2 + c # ufunc acting on whole array
arr = (abs(zs) < 2.0)*1 # make an array of 1s (in S) and 0s
plt.imshow(arr, cmap='Purples', origin='lower')
plt.savefig("quadraticmap.png") # save to file
```

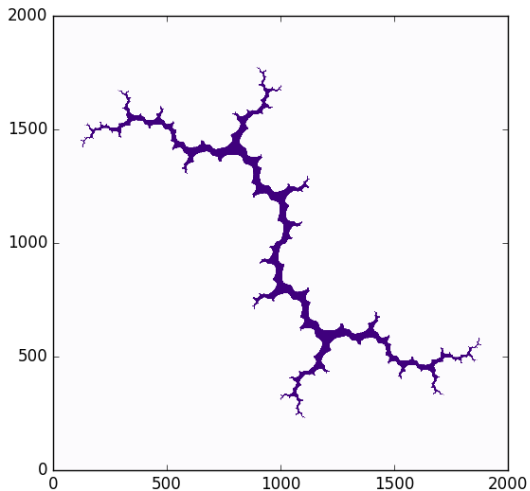


### Example #3: Quadratic maps $z \rightarrow z^2 + c$



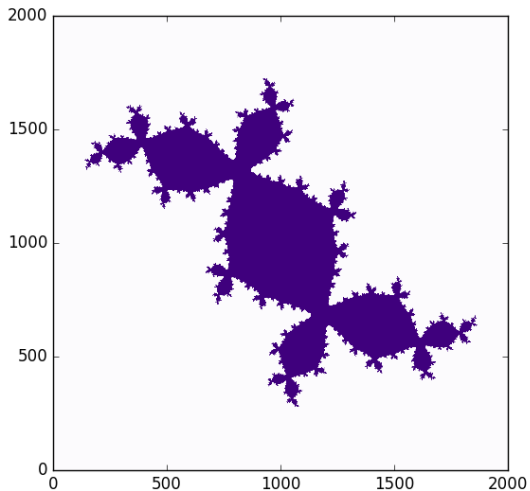
$$c = -0.75$$

### Example #3: Quadratic maps $z \rightarrow z^2 + c$



$$c = i$$

### Example #3: Quadratic maps $z \rightarrow z^2 + c$



$$c = -0.123 + 0.745i$$