Topology, Chaos, and Fractals

Topology is a important field of mathematics which studies shapes and spaces. There are plenty of awesome images created by chaos and fractals phenomenon.

References

- Malessky, Perciante and Yunker, Fractals for the Classoom, 1992, Springer.
- Daniel, Fractals and Chaos in Biology (http://www.govhs.org/vhsweb/Gallery.nsf/Files//fractals_and_chaos_in_biology.pdf)

In [1]:
```
1 from IPython.display import Image
```

People

Housdorff, Koch, Sierpinski, and Mandelbrot, Persons who concered

In [4]:
```
1 Image("images/housedorff.jpg",width=200)
```

Out[4]:

In [6]:
```
1 Image("images/koch.jpg",width=200)
```

Out[6]:

In [7]:
```
1 Image("images/Sierpinski.jpg",width=200)
```

Out[7]:

In [8]:
```
1 Image("images/Mandelbrot.jpg",width=200)
```

Out[8]:

In [ ]:
```
1
```

In [1]:
```
1 from IPython.display import YouTubeVideo
2
3 YouTubeVideo('L0MK7qz13bU')
```

Out[1]:

Shape

Möbius strip and Klein bottle

```
One-sided surface
```

In [5]:

```
import requests
from IPython.display import Image

url="http://www.yankodesign.com/images/design_news/2022/09/mobius-shaped-continuous-coffee-table-takes-inspi
Image(requests.get(url).content)

```

Out[5]:

Dimension:

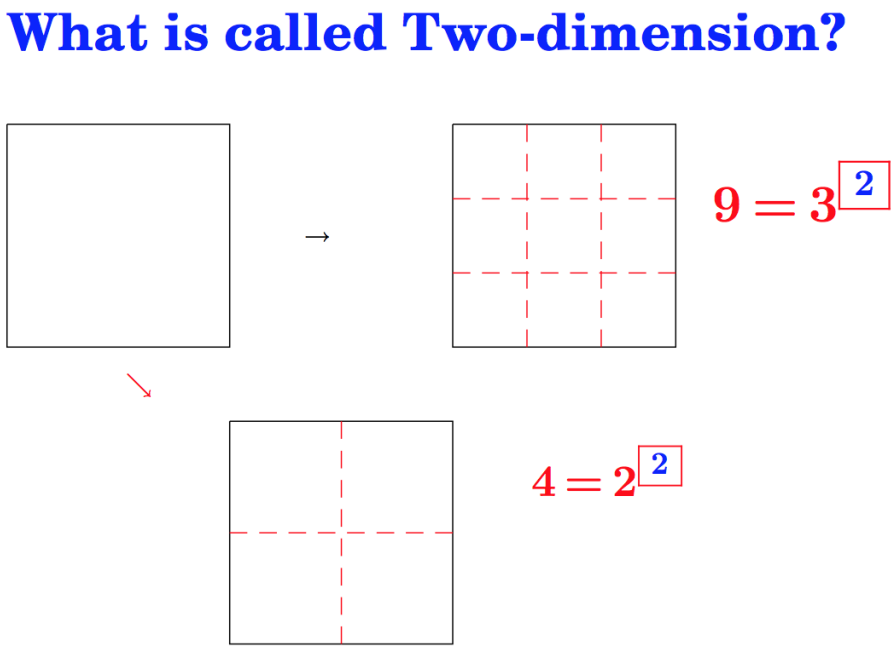
As well-known, the value of dimension is always in interger type, for instance: 1D, 2D, 3D, 4D, i.e. coordinates in (x, y, z, t).

```
    Whether does geometric objects with non-integer dimension exist or not?
```

2 Dimensions

In [10]:
```
1 Image("images/2-dim.png",width=500)
```

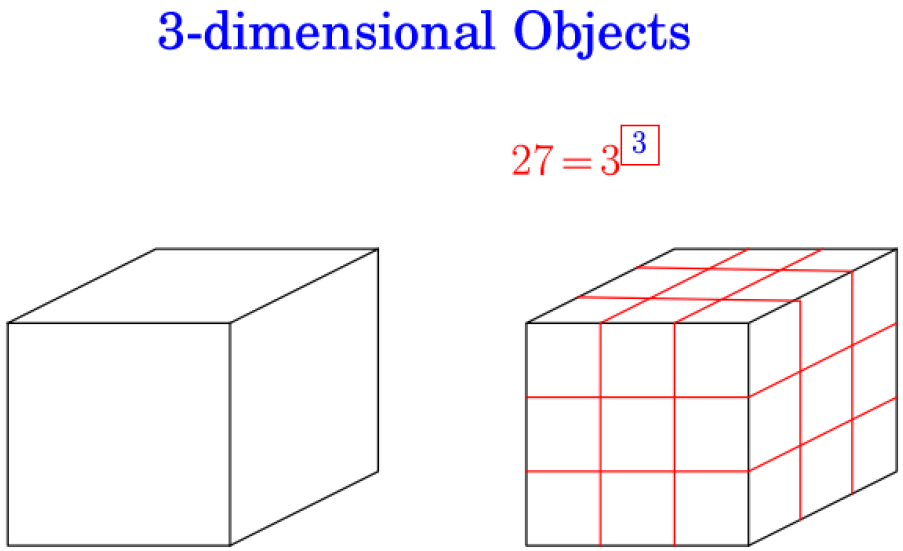
Out[10]:



Three Dimensions

In [11]:
```
1 Image("images/3-dim.png",width=500)
```

Out[11]:



Definition of "Dimension"

So, what is dimesion defined? While one unit is divided into $n(=1/\varepsilon)$ sections and there are $N = n^d$ duplicates created, the dimension is defined as:

$$N = n^d = (1/\varepsilon)^d \rightarrow d = \frac{\ln N}{\ln n} = \frac{\ln N}{\ln(1/\varepsilon)}$$

Suppose that each side of object is subdivided into n partitions and N_n is the total number of objects created. Then the dimension, d, is difined as follows:

$$d = \lim_{n\to\infty} \frac{\log N_n}{\log n}$$

Cantor set (Dimension < 1):

In [12]:
```
1 Image("images/Cantor.png")
```

Out[12]:

Dimension of Cantor's set is $\log 2/\log 3 \sim 0.631 (< 1)$ since 2 duplicates created by three divisions.

In [ ]:
```
1
```

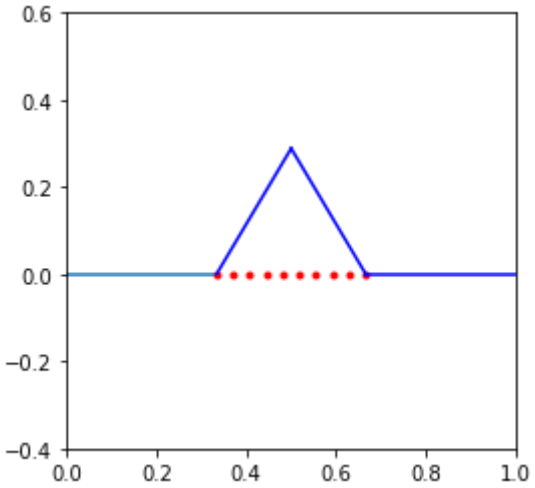
Koch's SnowFlake

The middle sub-division is removed after three divisions on each sides starting from equilateral triangle:

In [3]:

```
%matplotlib inline

import pylab as plt
from numpy import *
n=50
m=10
r=1/3.
x0=linspace(0,r,n)
x00=linspace(0,r,m)
x1=x00+r*ones(m)
x2=x0+2*r*ones(n)
x11=x0/2+r*ones(n)
x12=-x0/2+2*r*ones(n)

plt.figure(figsize=(4,4))
plt.plot(x0,0*x0)+plt.plot(x1,0*x00,'r.')+plt.plot(x2,0*x0,'b-') \
    +plt.plot(x11,sqrt(3)/2*x0,'b-')+plt.plot(x12,sqrt(3)/2*x0,'b-')
plt.xlim(0,1)
plt.ylim(-0.4,0.6)
```

Out[3]: (-0.4, 0.6)

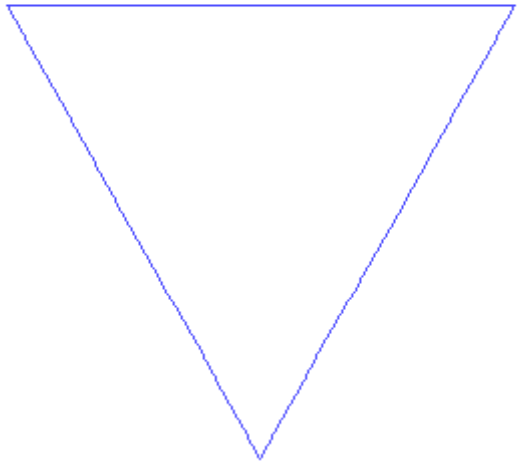


In [2]:
```
1 log(4)/log(3)
```

Out[2]: 1.2618595071429148

In [13]:
```
1 Image("images/Koch.gif")
```

Out[13]:



The dimension of koch snowflake is **1.2618595071429148** since got 4 after 3 divisions.

Circumference Length of Koch Snowflake

Trisecting each side and getting rid of the middle section gets two sub-sides left:

number of steps	number of sides	length of side	total length
0	3	1	1×3
1	3×4	$\frac{1}{3}$	$3 \times \frac{4}{3}$
2	3×4^2	$\frac{1}{3^2}$	$3 \times \frac{4^2}{3^2}$

3	3×4^2	$\frac{1}{3^3}$	$3 \times \frac{4^3}{3^3}$

- Infinite length of circumference measured in 1-dimension since

$$\lim_{n\to\infty} 3\frac{4^n}{3^n} = \infty$$

Does the last result break our knowledge about:

```
Length of enclosed curve is finite
```

In [4]:

```
import svgwrite
from svgwrite import cm, mm
```

In [5]:

```
import math

def koch_snowflake():
    # Koch Snowflake and Sierpinski Triangle combination fractal using recursion
    # ActiveState Recipe 577156
    # Created by FB36 on Sat, 27 Mar 2010 (MIT)
    # http://code.activestate.com/recipes/577156-koch-snowflake-and-sierpinski-triangle-combination/

    def tf (x0, y0, x1, y1, x2, y2):
        a = math.sqrt((x1 - x0) ** 2 + (y1 - y0) ** 2)
        b = math.sqrt((x2 - x1) ** 2 + (y2 - y1) ** 2)
        c = math.sqrt((x0 - x2) ** 2 + (y0 - y2) ** 2)

        if (a < stop_val) or (b < stop_val) or (c < stop_val):
            return

        x3 = (x0 + x1) / 2
        y3 = (y0 + y1) / 2
        x4 = (x1 + x2) / 2
        y4 = (y1 + y2) / 2
        x5 = (x2 + x0) / 2
        y5 = (y2 + y0) / 2
        points = [(x3, y3), (x4, y4), (x5, y5)]

        # append new polygon to snowflake element
        snowflake.add(dwg.polygon(points))
        tf(x0, y0, x3, y3, x5, y5)
        tf(x3, y3, x1, y1, x4, y4)
        tf(x5, y5, x4, y4, x2, y2)

    def sf (ax, ay, bx, by):
        f = math.sqrt((bx - ax) ** 2 + (by - ay) ** 2)

        if f < 1.:
            return

        f3 = f / 3
        cs = (bx - ax) / f
        sn = (by - ay) / f
        cx = ax + cs * f3
        cy = ay + sn * f3
```

```
        h = f3 * math.sqrt(3) / 2
        dx = (ax + bx) / 2 + sn * h
        dy = (ay + by) / 2 - cs * h
        ex = bx - cs * f3
        ey = by - sn * f3
        tf(cx, cy, dx, dy, ex, ey)
        sf(ax, ay, cx, cy)
        sf(cx, cy, dx, dy)
        sf(dx, dy, ex, ey)
        sf(ex, ey, bx, by)

    # const values
    stop_val = 8.
    imgx = 512
    imgy = 512

    # create a new drawing
    dwg = svgwrite.Drawing("images/test.svg", (imgx, imgy))

    # create a new <g /> element, we will insert the snowflake by the <use /> element
    # here we set stroke, fill and stroke-width for all subelements
    # attention: 'stroke-width' is not a valid Python identifier, so use 'stroke_witdth'
    #   underlines '_' will be converted to dashes '-', this is true for all svg-keyword-attributs
    # if no 'id' is given ( like dwg.g(id="sflake") ), an automatic generated 'id' will be generated
    snowflake = dwg.g(stroke="blue", fill="rgb(90%,90%,100%)", stroke_width=0.25)

    # add the <g /> element to the <defs /> element of the drawing
    dwg.defs.add(snowflake)

    mx2 = imgx / 2
    my2 = imgy / 2
    r = my2
    a = 2 * math.pi / 3
    for k in range(3):
        x0 = mx2 + r * math.cos(a * k)
        y0 = my2 + r * math.sin(a * k)
        x1 = mx2 + r * math.cos(a * (k + 1))
        y1 = my2 + r * math.sin(a * (k + 1))
        sf(x0, y0, x1, y1)

    x2 = mx2 + r * math.cos(a)
    y2 = my2 + r * math.sin(a)
```

```
    tf(x0, y0, x1, y1, x2, y2)

    # create an <use /> element
    use_snowflake = dwg.use(snowflake)

    # you can transform each <use /> element
    # use_snowflake.rotate(15, center=(imgx/2, imgy/2))

    # insert snowflake by the <use /> element
    dwg.add(use_snowflake)
    dwg.save()
    return dwg.tostring()
```

In [6]:
```
l=koch_snowflake()
from IPython.display import SVG
SVG(filename='images/test.svg')
```

Out[6]: <IPython.core.display.SVG object>

In [7]:
```
%matplotlib inline

import numpy as np
import time
from IPython.display import clear_output,Image, HTML
import matplotlib.pylab as plt
import os
from numpy import random
from tempfile import NamedTemporaryFile

```

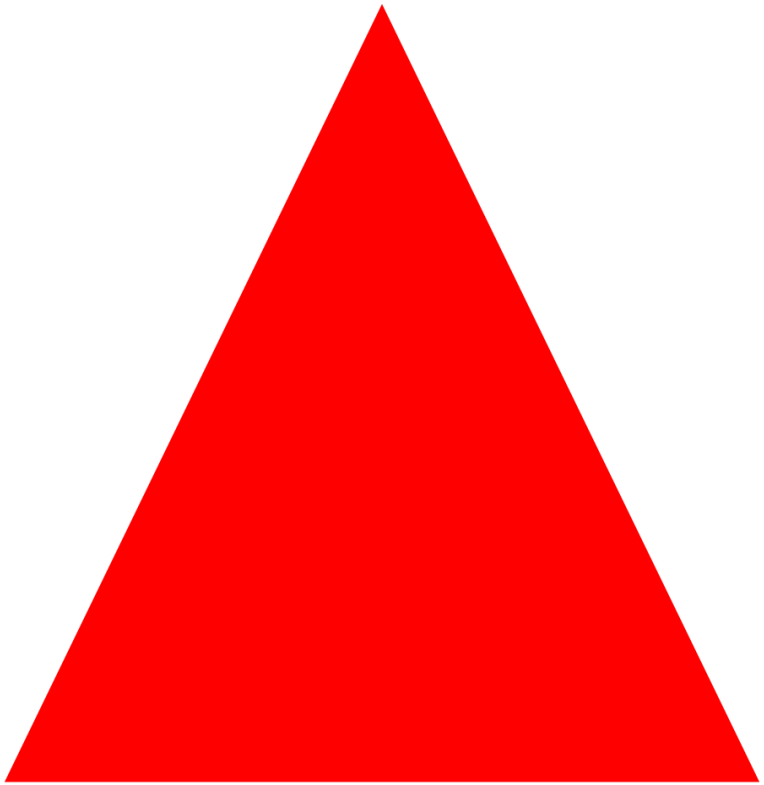
Questions:

- What is total length of Cantor set?
- What is the limit of total lenght of koch snowflake? And find its total length in right dimension.

Example Sierpinski Triangle

In [15]:
```
1 Image("images/Sierpinski.gif",width=400)
```

Out[15]:

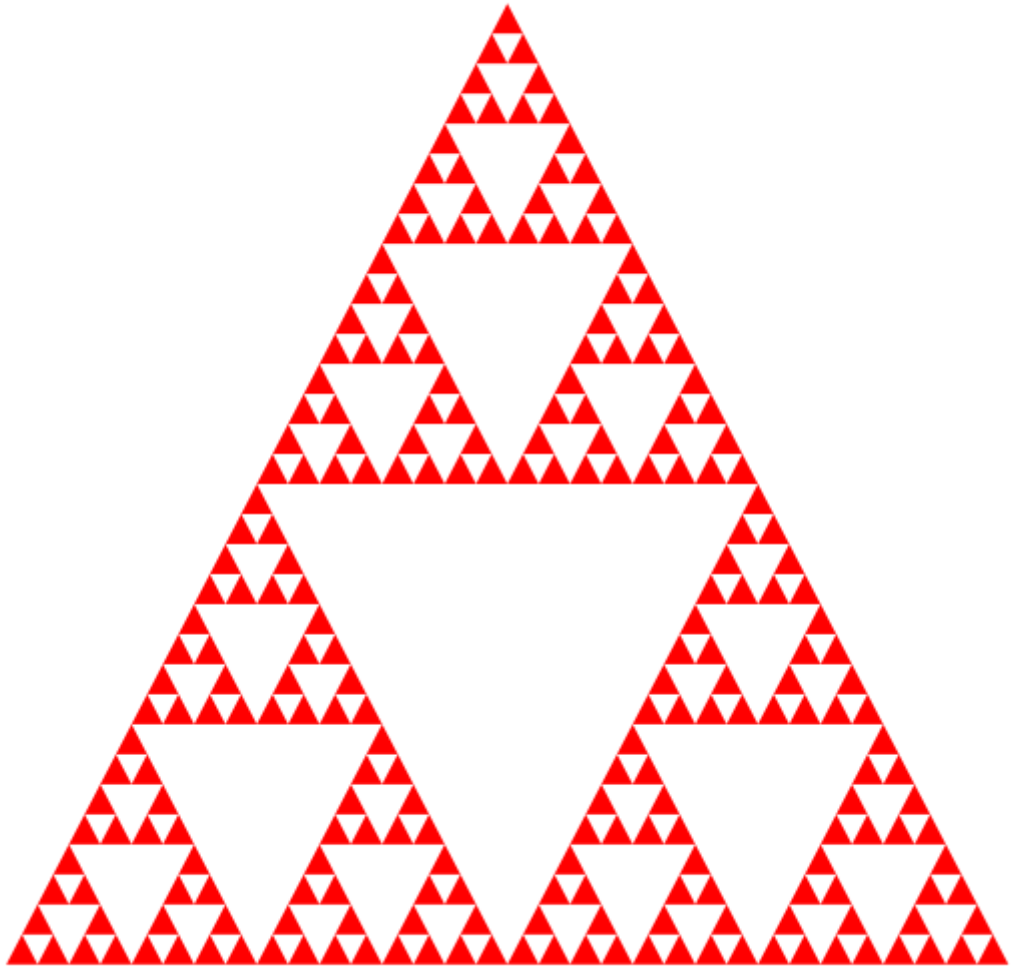


In [2]:

```
%matplotlib inline
import random
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import math

def triangle(vertices,ax):
    """
    Take the arguments of vertices (np.array) and axis object and gives out triangle
    formed with midpoints and returns vertices of midpoints
    """
    l=[]
    l.append((vertices[0]+vertices[1])/2)
    l.append((vertices[1]+vertices[2])/2)
    l.append((vertices[0]+vertices[2])/2)
    ax.add_patch(patches.Polygon(l,fill=True,facecolor='w'))
    return l

def sierper(verts,ax,count=0):
    """
    Gives out a serper with the vertices given
    """
    if count==5:
        return
    count=count+1
    vertices=triangle(verts,ax)
    list1=np.array([[0,0,2],[1,0,1],[2,1,2]])
    for i in range(3):
        temp_array=np.array([verts[list1[i,0]],vertices[list1[i,1]],vertices[list1[i,2]]])
        sierper(temp_array,ax,count)
```

In [6]:

```
fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(111)

verts = np.array([[0.0, 1.0], [-math.sqrt(3)/2, -0.5], [math.sqrt(3)/2, -0.5]])
poly = patches.Polygon(verts,facecolor='r',fill=True)
ax.add_patch(poly)
ax.set_xlim(verts[:,0].min()-0.1,verts[:,0].max()+0.1)
ax.set_ylim(verts[:,1].min()-0.1,verts[:,1].max()+0.1)
sierper(verts,ax)
plt.axis('off')
plt.show()
```



In [7]:
```
1 plt.axes?
```

Dimension

Got 3 left after two divisions at each side implies:

$$N = 2^d \to 3 = 2^d$$
$$d = \frac{\log 3}{\log 2} \sim 1.585 < 2$$

Area of Sierpinski Triangle

Bisecting each side of triangle gets three sub-triangles:

number of steps	number of sub-triangles	area of sub-triangle	total Area
0	1	$\frac{\sqrt{3}}{4}$	$1 \times \frac{\sqrt{3}}{4}$
1	3	$\frac{1}{4} \times \frac{\sqrt{3}}{4}$	$3 \times \frac{\sqrt{3}}{4^2}$
2	3^2	$\frac{1}{4^2} \times \frac{\sqrt{3}}{4}$	$3^2 \times \frac{\sqrt{3}}{4^3}$
3	3^3	$\frac{1}{4^3} \times \frac{\sqrt{3}}{4}$	$3^3 \times \frac{\sqrt{3}}{4^4}$

Area

$A(n)$: the total area left at step$= n, n = 0, 1, 2, \cdots$. Then

$$A(n+1) = \frac{3}{4}A(n)$$
$$A(n) = (\frac{3}{4})^{n-1}\frac{\sqrt{3}}{4}$$

This means

$$\lim_{n\to\infty} A(n) = 0$$

NO AREA left if measured by two dimensional unit!

Problem

What is the total length of coastline of Britian?

- Benoît Mandelbrot (1967). "How Long Is the Coast of Britain? Statistical Self-Similarity and Fractional Dimension", Science, New Series, Vol. 156, No. 3775. (May 5, 1967), pp. 636-638.

In [16]:
```
1 Image("images/britain.png",width=400)
```

Out[16]:

Note

- Decide the length of unit, and measure whole the British land; different scale, different measurement!
- Get the data by the HTML tool from [WebPlotDigitizer (http://arohatgi.info/WebPlotDigitizer)](http://arohatgi.info/WebPlotDigitizer).

Theory and Dicussions

Around 1921, the British mathematician **Lewis Fry Richardson** started thinking about the length of coastlines of Britain. Richardson's idea was to measure a coastline, such as the west coast of Britain as above. by choosing a "size of measurement" x and then seeing how many units it was measured.

```
However, a large step size misses a lot of the detail of the coast.
```

By lots of in's and out's, Richardson used successively smaller measurement x and each time recorded the corresponding total length y.

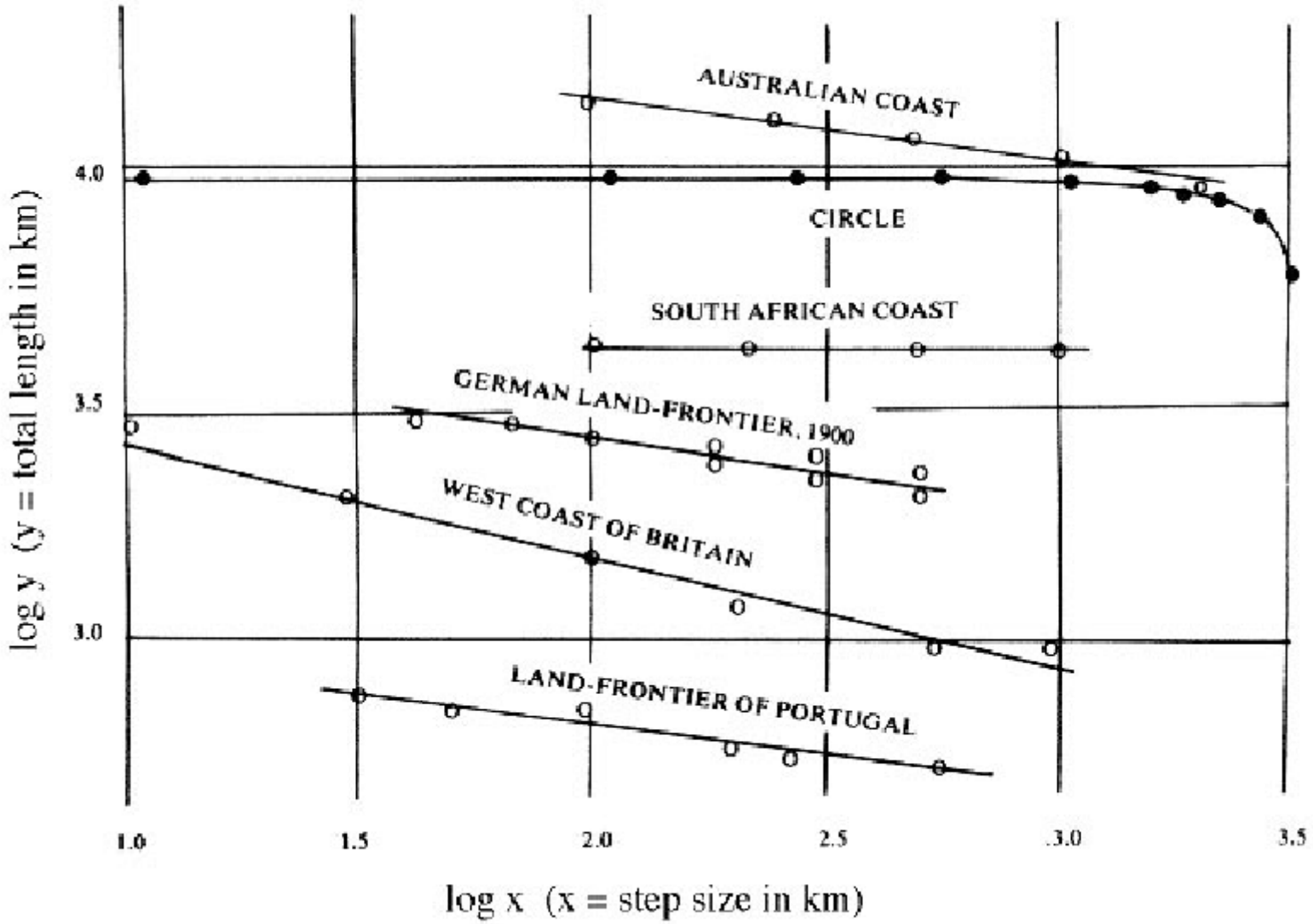
Finally, instead of plotting y versus x, Richardson had the clever idea of plotting the logarithm of y versus the logarithm of x, and found that they are in linear relation, i.e.

$$\log y = m \log x + b \ (\text{ or } y = Bx^m)$$

Here the data of several coastlines come from different locations:

In [17]:
```
1 Image("images/coastline.png")
```

Out[17]:



Computing Result

$$m = \frac{2.987 - 3.32}{2.726 - 1.4647} \sim -0.264$$
$$b = \log(y) - m\log(x) = 2.987 - (-0.264) * (2.726) \sim 3.7$$

This gives the the solution:

$$\log y = -0.264 \log x + 3.7$$
$$y = 10^{3.7} 10^{-0.264 \log x} \sim 5012 \times x^{-0.264} = \frac{5012}{x^{0.264}}$$

In [1]:
```
(2.987-3.32)/(2.726-1.4647)
```

Out[1]:
```
-0.2640133195908981
```

In [ ]:
```
2.987-(-0.264)*(2.726)
```

In [7]:
```
10**(3.7)
```

Out[7]:
```
5011.872336272725
```

$$y = \frac{5012}{x^{0.264}}$$

As x is becomes smaller and smaller, then we get the larger and larger y, For example, $x = 0.0001$ km, $y \sim 16904$ km which is more than one third of earth circumference. Actually, y would approach infinity while x approaches 0.

```
Is it reasonable to measure a geometric object and get infinite quantity?
```

Now we know this paradox come from the concept of **DIMEMSION**.

In [8]:
```
5012/(0.0001)**(0.264)
```

Out[8]:
```
57017.879565470364
```

Conclusion

Theoretically, the measurement of physical quantity of object is

$$\texttt{(number of steps)(step size) = (quantity of measurement)},$$

where $N(x)$ is number of steps (measurement), using any step size x.

In this case, the measurement, $y = M(x)$, of Britain Coastline with unit measurement, x, is:

$$M(x) \cdot x^1 = 5012 \times x^{-0.264} \rightarrow M(x) = 5012 \times x^{-1.264}$$

This is false that the measurement, $M(x)$ approaches ∞ if the dimension is one!

If the dimension of coastline is d, total length should satisfy:

$$\begin{aligned} \text{Total length} = & \quad 5012 \times x^{-1.264} \times x^d \\ = & \quad 5012 \times x^{d-1.264} \end{aligned}$$

This implies the dimension of coastline more reasonably being equal to $d = 1.264$ (but not 1) to confirm the total lentgh of coastline to be finite!

Question

Estimate the dimension of Taiwan coastline.

Chaos and Fractal

Complicated but ordered

Definition

The first term a_0 in $\{a_n\}_{n=0}^{\infty}$, with $a_{n+1} = f(a_n)$, is called periodic point with cycle of period m if there exists $m \in \mathbb{N}$ such that

$$f(a_m) = a_0$$

a_0 is called fixed point if $f(a_0) = a_0$.

Li-Yorks

Periodic 3 implies Infinite periodic

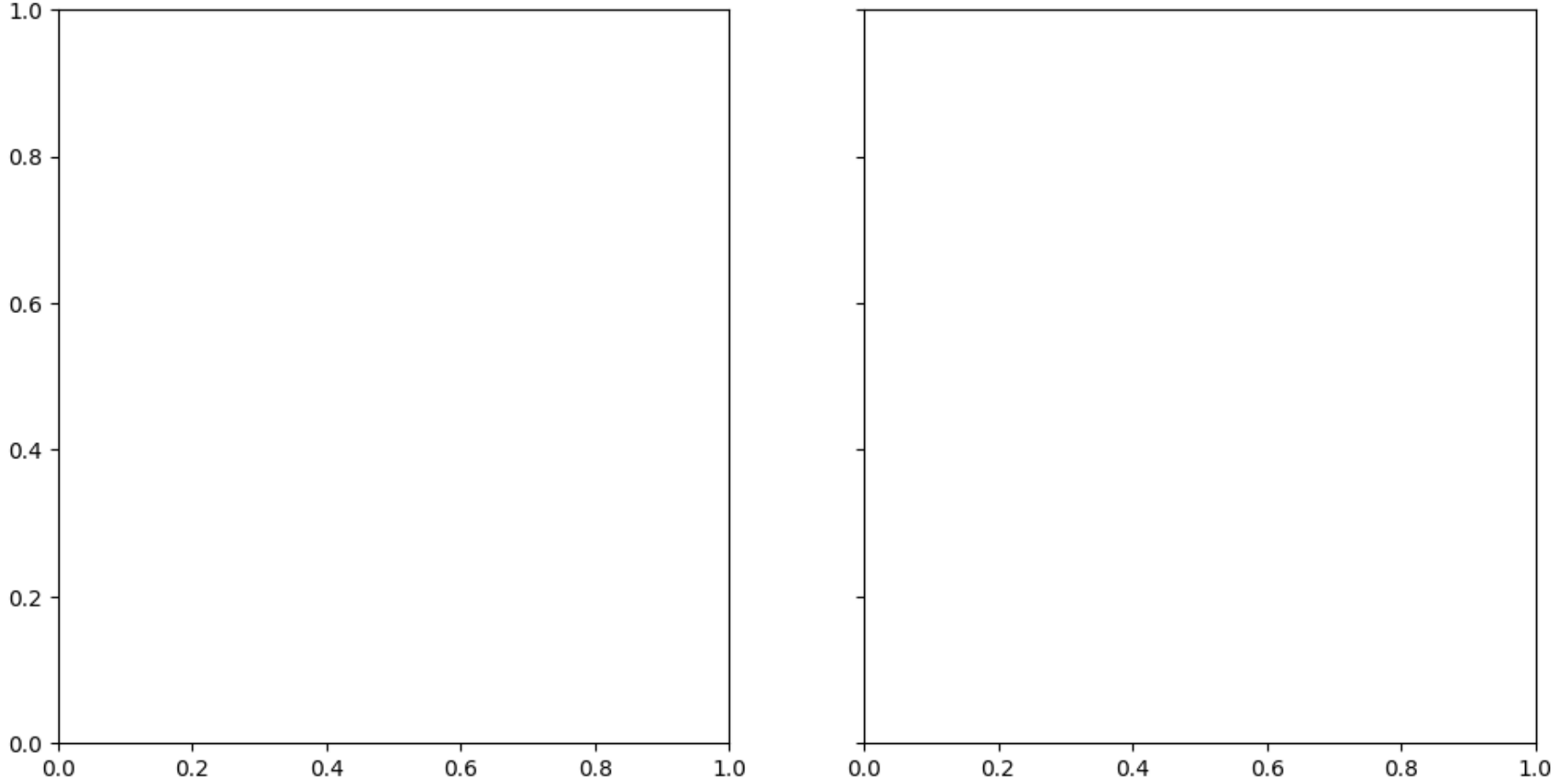
$$a_{a+1} = f(a_n) = ra_n(1 - a_n)$$

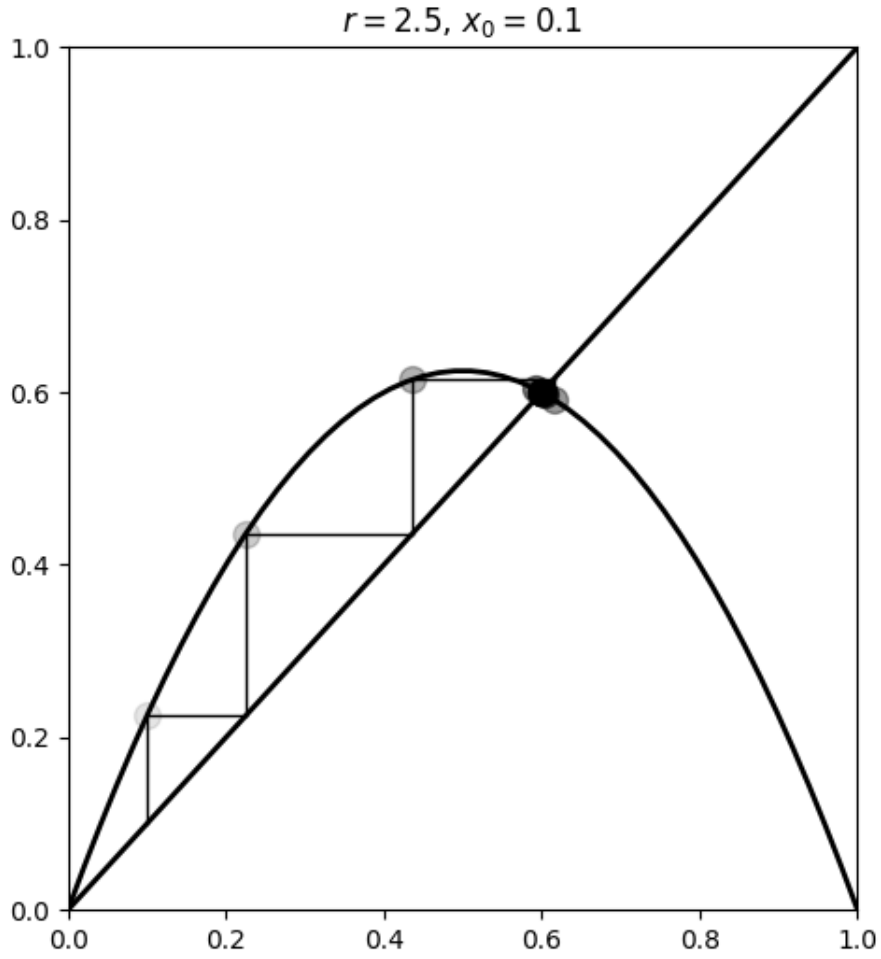
In [56]:

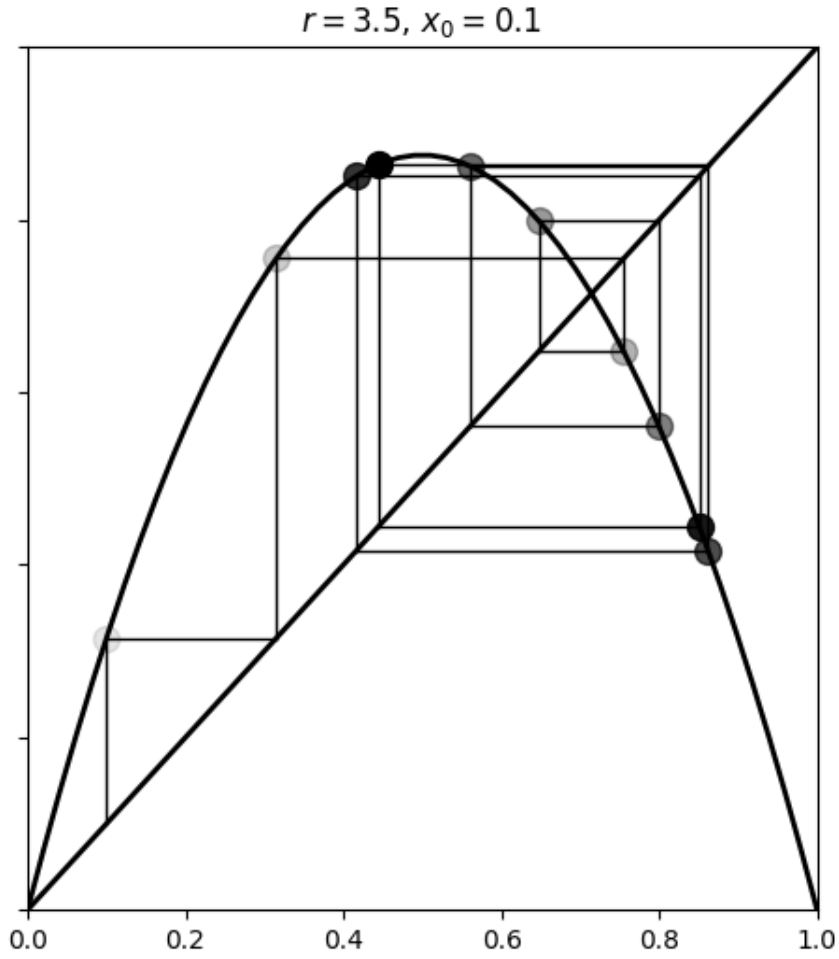
```
def logistic(r, x):
    return r * x * (1 - x)
```

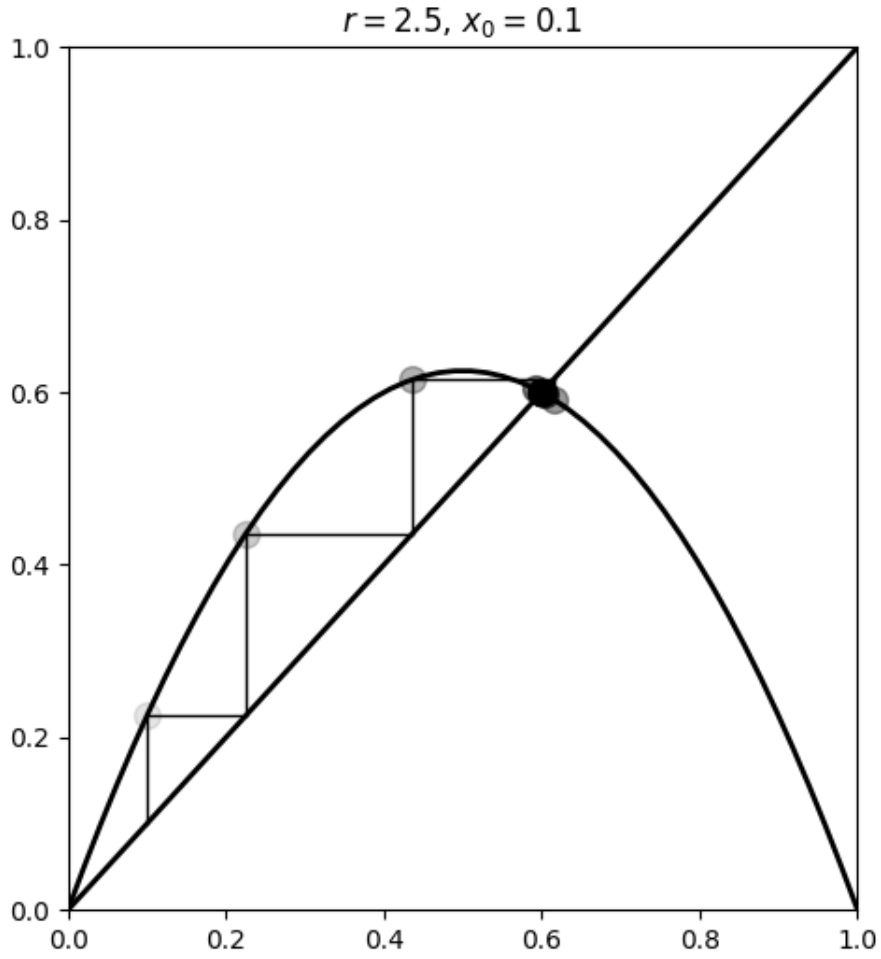
In [59]:

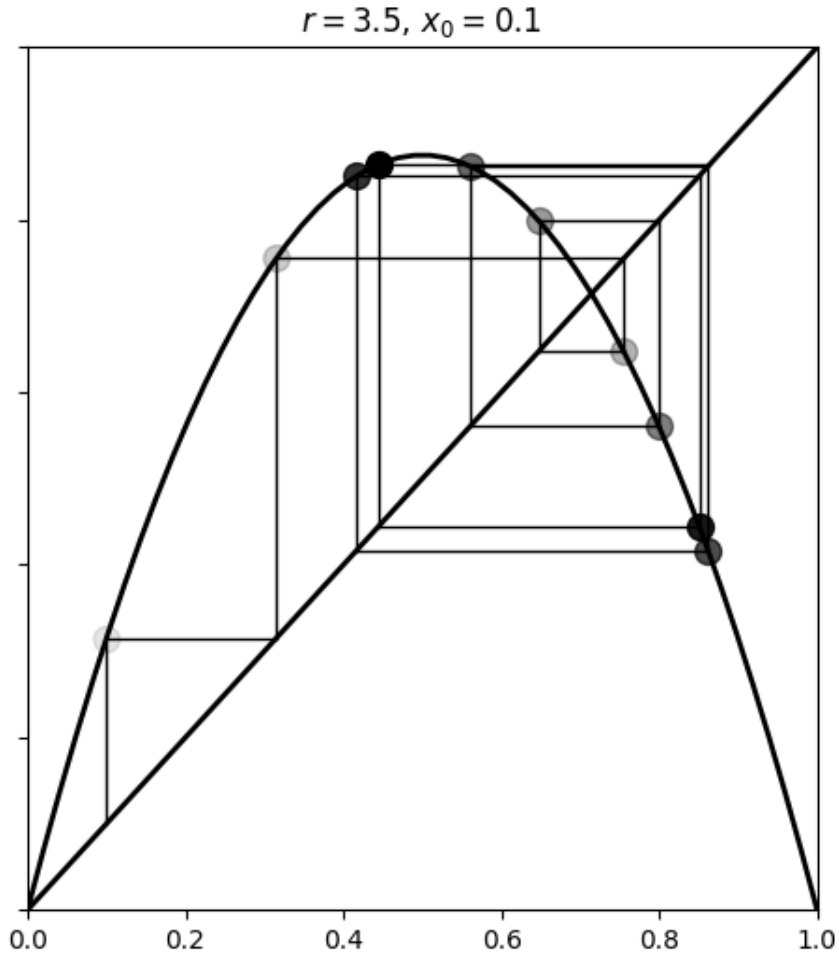
```
def plot_system(r, x0, n, ax=None):
    # Plot the function and the
    # y=x diagonal line.
    t = np.linspace(0, 1)
    ax.plot(t, logistic(r, t), 'k', lw=2)
    ax.plot([0, 1], [0, 1], 'k', lw=2)

    # Recursively apply y=f(x) and plot two lines:
    # (x, x) -> (x, y)
    # (x, y) -> (y, y)
    x = x0
    for i in range(n):
        y = logistic(r, x)
        # Plot the two lines.
        ax.plot([x, x], [x, y], 'k', lw=1)
        ax.plot([x, y], [y, y], 'k', lw=1)
        # Plot the positions with increasing
        # opacity.
        ax.plot([x], [y], 'ok', ms=10,
                alpha=(i + 1) / n)
        x = y

    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.set_title(f"$r={r:.1f}, \, x_0={x0:.1f}$")


fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6),
                               sharey=True)
plot_system(2.5, .1, 10, ax=ax1)
plot_system(3.5, .1, 10, ax=ax2)
plt.show()
```

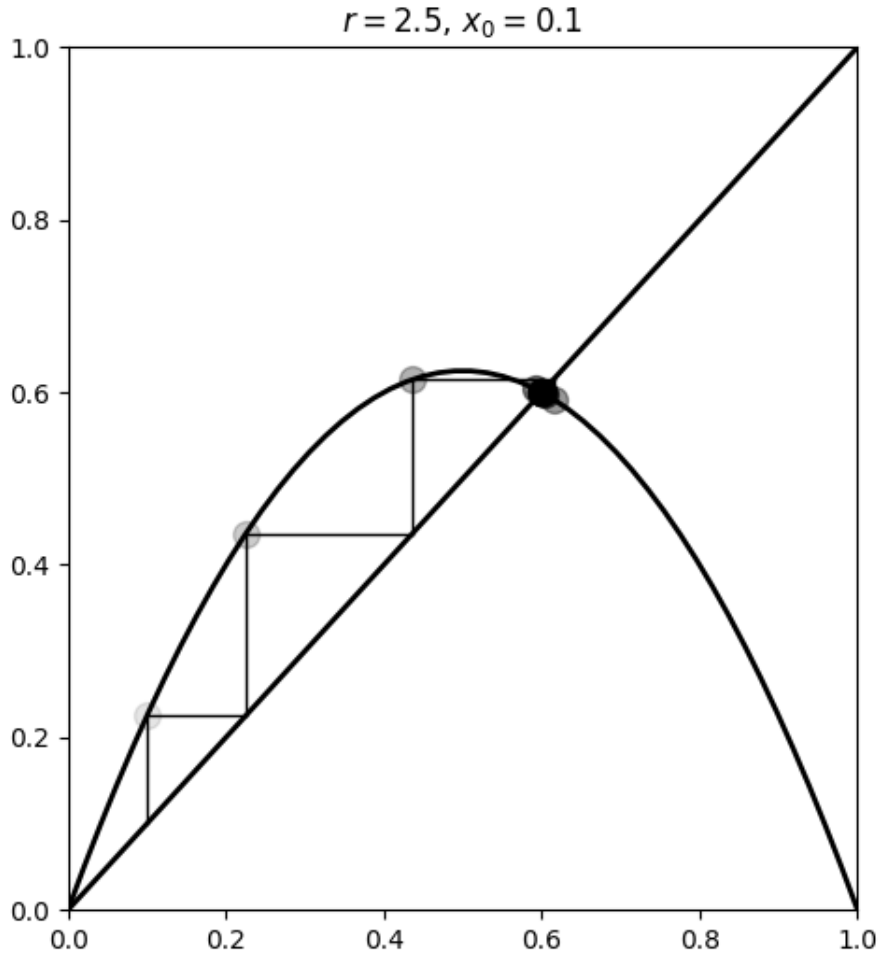

1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.2
0.4
0.6
0.8
1.0

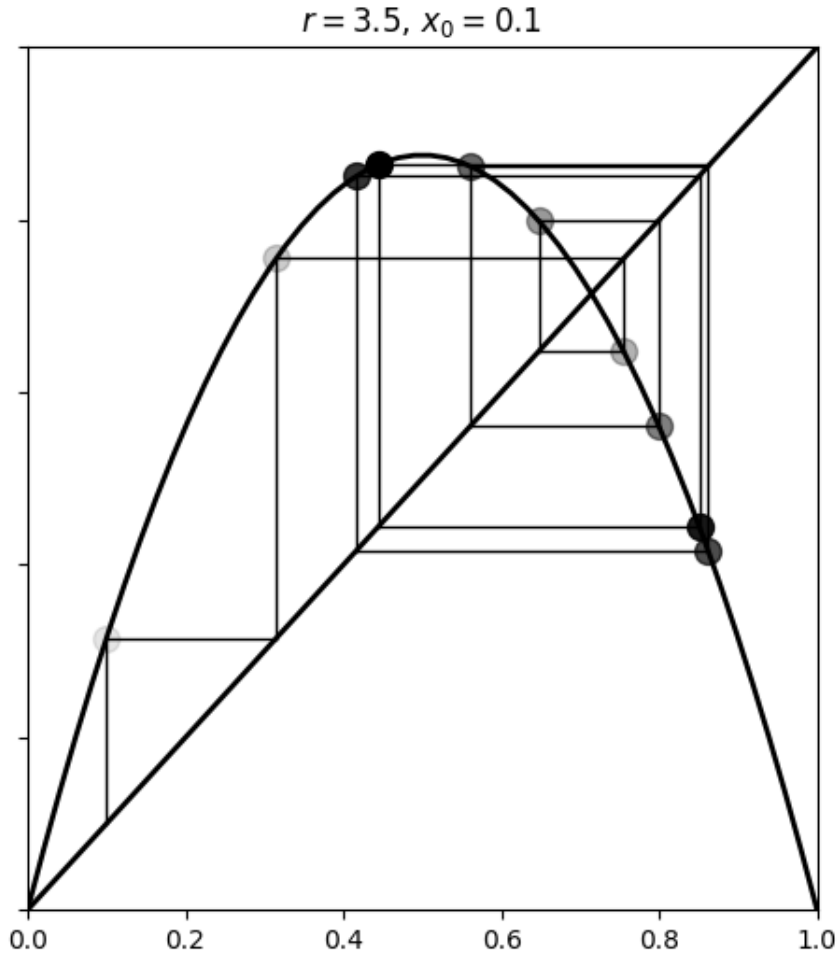

$r = 2.5$, $x_0 = 0.1$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


$r = 3.5$, $x_0 = 0.1$
0.0
0.2
0.4
0.6
0.8
1.0


$r = 2.5$, $x_0 = 0.1$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


$r = 3.5$, $x_0 = 0.1$
0.0
0.2
0.4
0.6
0.8
1.0

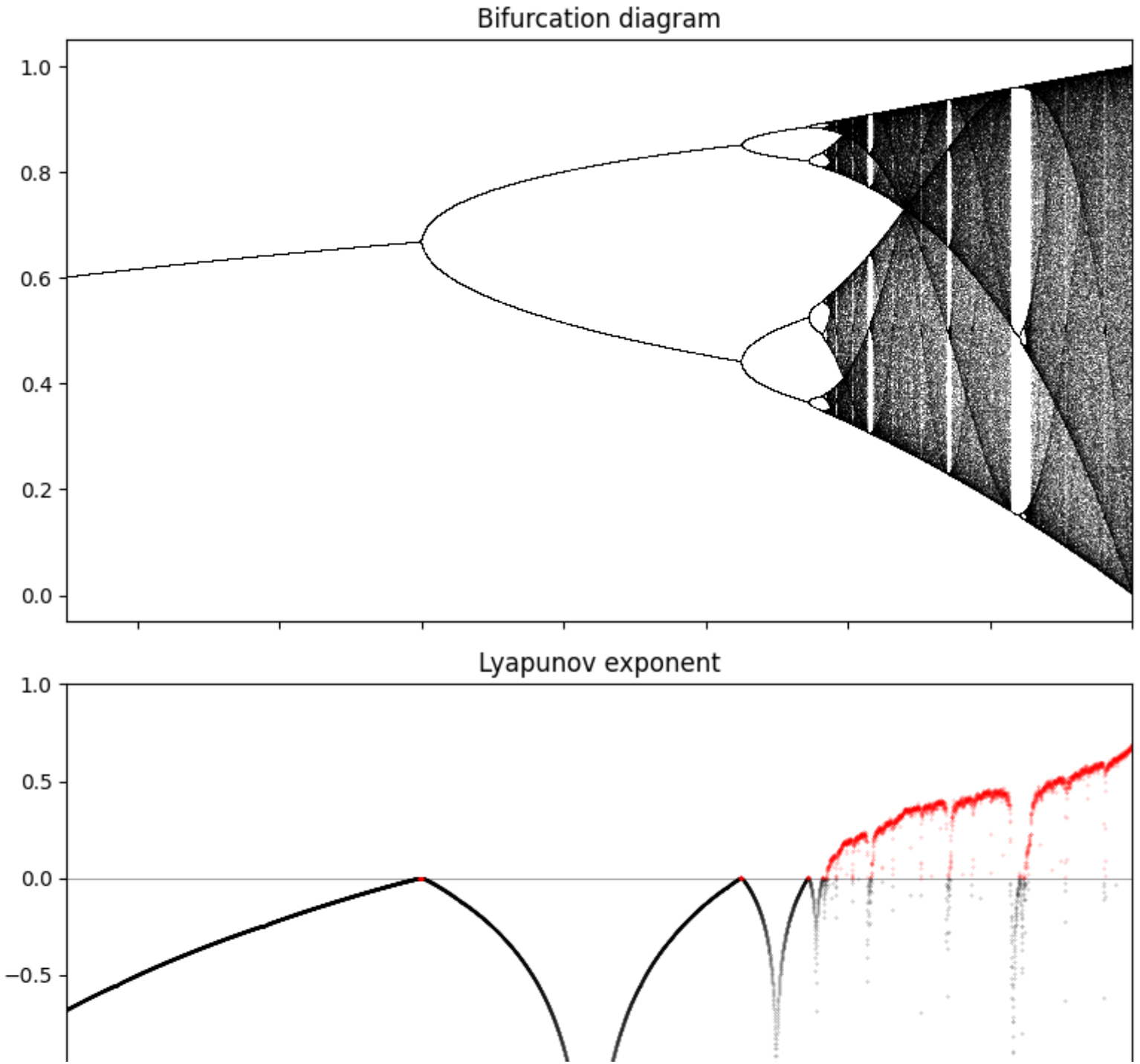

$r = 2.5$, $x_0 = 0.1$
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

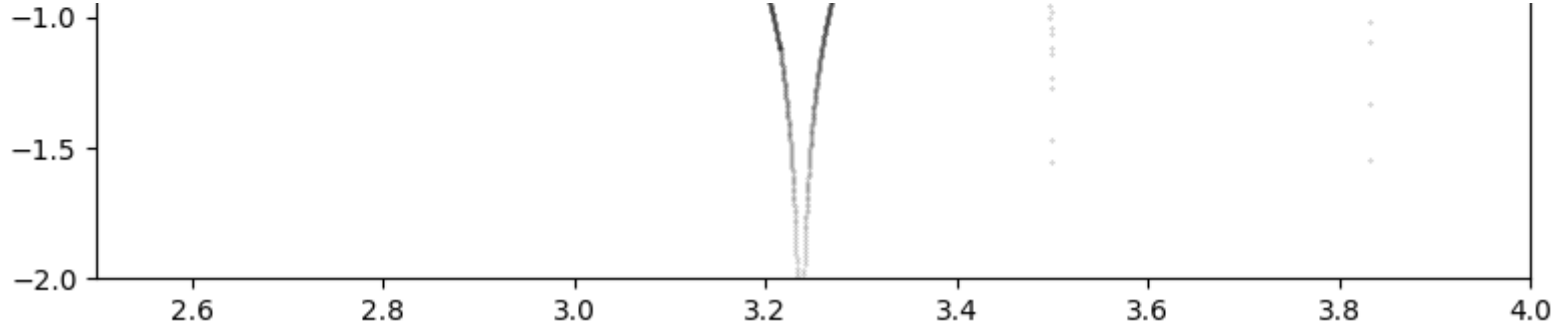

$r = 3.5$, $x_0 = 0.1$
0.0
0.2
0.4
0.6
0.8
1.0

In [66]:

```
n = 10000
r = np.linspace(2.5, 4.0, n)

iterations = 1000
last = 100

x = 1e-5 * np.ones(n)
lyapunov = np.zeros(n)

fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(8, 9),
                               sharex=True)
for i in range(iterations):
    x = logistic(r, x)
    # We compute the partial sum of the
    # Lyapunov exponent.
    lyapunov += np.log(abs(r - 2 * r * x))
    # We display the bifurcation diagram.
    if i >= (iterations - last):
        ax1.plot(r, x, ',k', alpha=.25)
ax1.set_xlim(2.5, 4)
ax1.set_title("Bifurcation diagram")

# We display the Lyapunov exponent.
# Horizontal line.
ax2.axhline(0, color='k', lw=.5, alpha=.5)
# Negative Lyapunov exponent.
ax2.plot(r[lyapunov < 0],
         lyapunov[lyapunov < 0] / iterations,
         '.k', alpha=.5, ms=.5)
# Positive Lyapunov exponent.
ax2.plot(r[lyapunov >= 0],
         lyapunov[lyapunov >= 0] / iterations,
         '.r', alpha=.5, ms=.5)
ax2.set_xlim(2.5, 4)
ax2.set_ylim(-2, 1)
ax2.set_title("Lyapunov exponent")
plt.tight_layout()

```

```
plt.show()
```


Bifurcation diagram
1.0
0.8
0.6
0.4
0.2
0.0
Lyapunov exponent
1.0
0.5
0.0
−0.5



In [64]:
```
1 plt.show()
```

Example

Consider the following function:

$$h(x) = 2 - |x - 1|$$

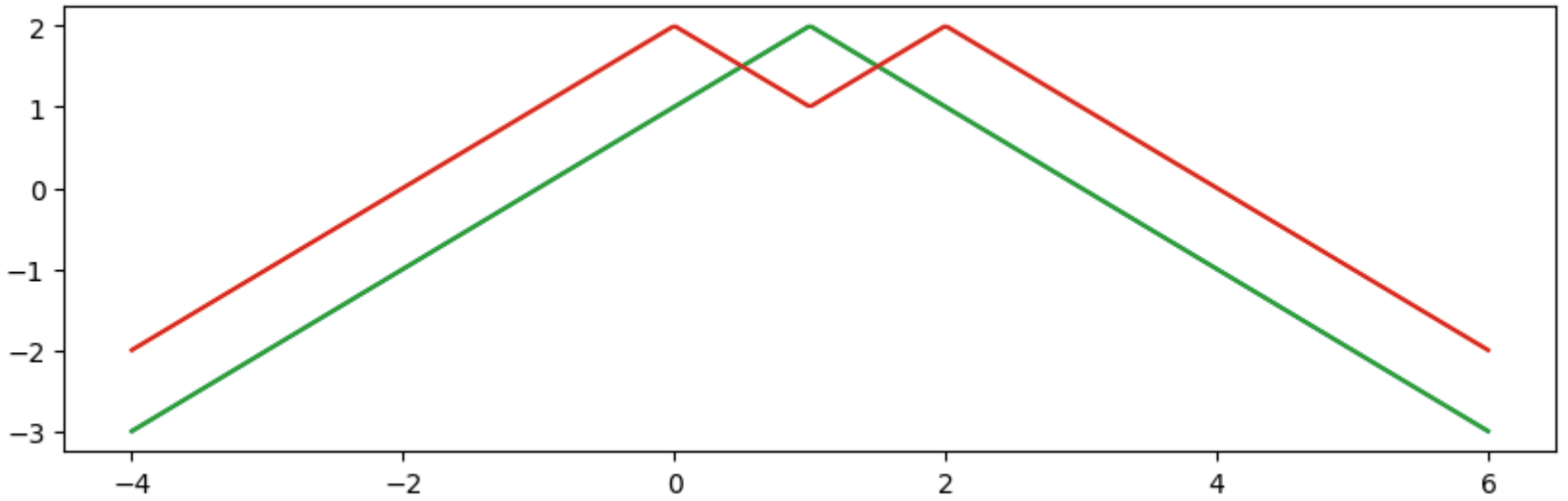
1. It is trivial:

$$h(x) = \begin{cases} 3 - x, & \text{if } x > 1; \\ 1 + x, & \text{if } x \leqslant 1. \end{cases}$$

2. The graphs about $h(x)$ and its composed functions, $g^n(x)$ are as follows pictures.

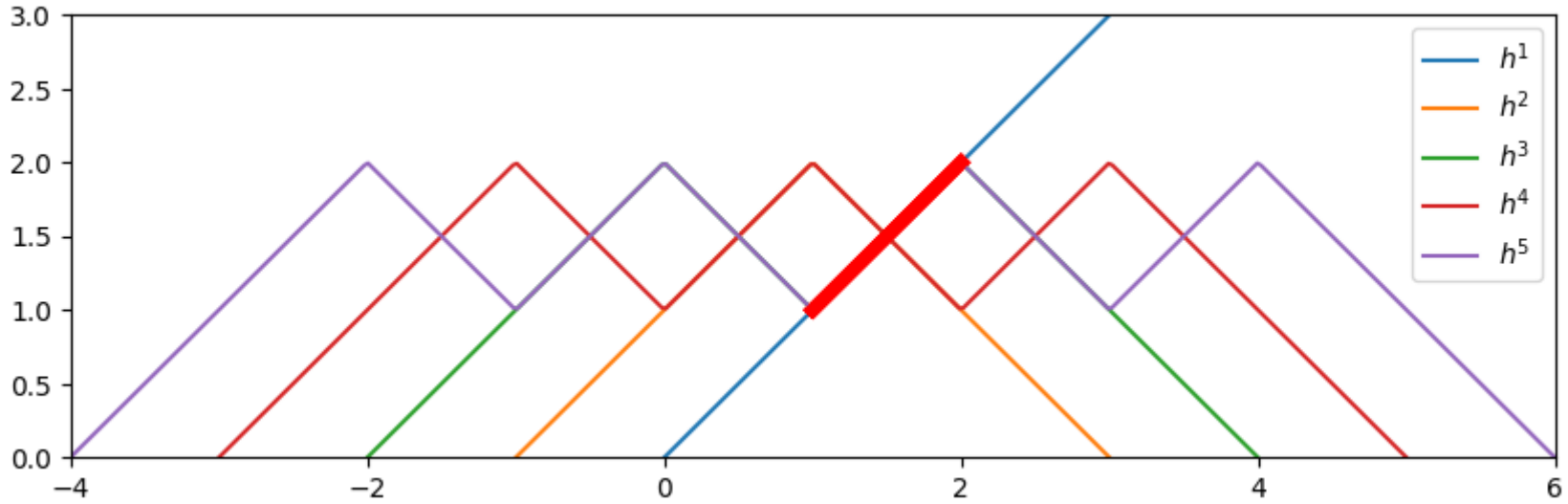
$h(x)$ is a wedge with straight lines and one vertex on the line $y = 2$, (the blue line). $h^2(x)$ is a wedge with three vertices on the line $y = 2$, (the green line). The solution of equation is the part which these two curves intersect with each other, (the line with thicker segment and in red), i.e. $x \in [1, 2]$. It is trivial to prove that any point within this range is fixed point only!

In [42]:
```
plt.rcParams['figure.figsize'] = (10,3) #wide graphs by default
def h(x):
    return 2-abs(x-1)

x=np.linspace(-4,6,400)
plt.plot(x,h(x),x,h(h(x)))
plt.show()
```



In [50]:

```
import time
from IPython.display import clear_output,display
f, ax = plt.subplots()
ax.set_xlim([-4,6]);ax.set_ylim([0,3])

x=np.linspace(-4,6,400);y=x
for i in np.arange(5):
    label=f"$h^{i+1}$"
    plt.plot(x,y,label=label,)

    plt.plot((1,2),(1,2),'r-',linewidth=5)
    plt.legend()
    y=h(y)

    time.sleep(2)
    clear_output()
    display(f)
plt.close()
```



Now consider the following mapping:

$$Z_{n+1} = Z_n^2 + C$$

where

1. $C = x + iy$ where (x, y) are points on 2-dimensional plane and $i = \sqrt{-1}$.
2. Z_0 varies within a square region and C fixed;

   a) simple plot;

   b) animation with parameter, C.

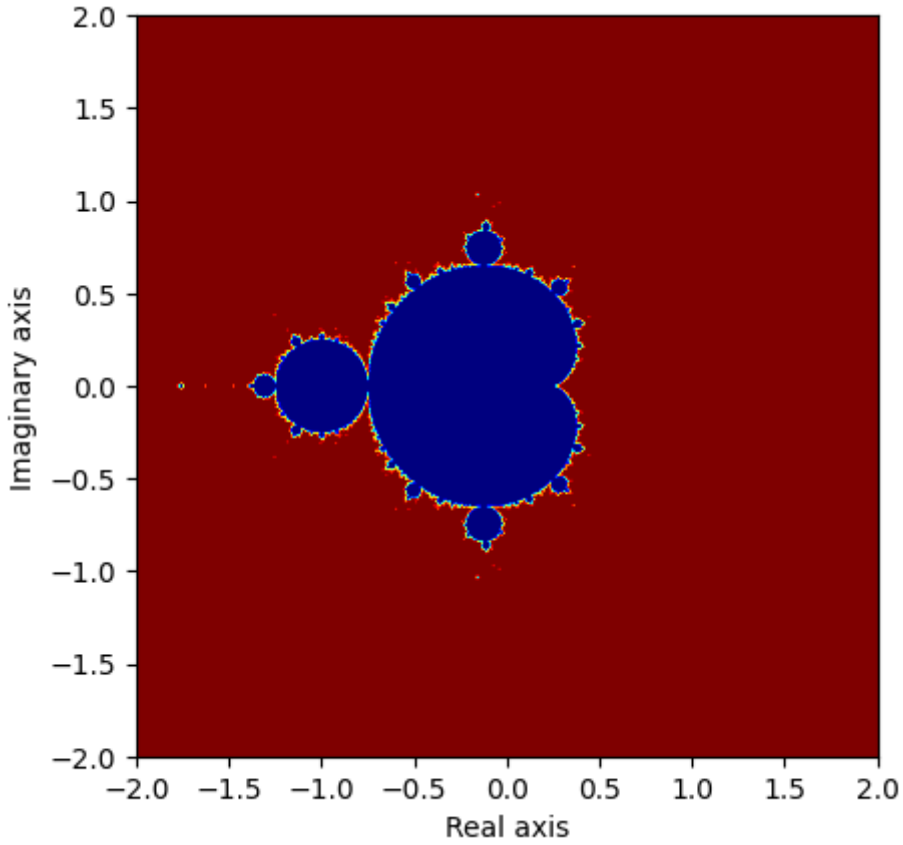
Not all points are `finite` after iterating continuously.

In [15]:

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

# Define the function that generates the fractal
def mandelbrot(c, maxiter):
    z = 0
    n = 0
    while abs(z) <= 2 and n < maxiter:
        z = z*z + c
        n += 1
    return n
def mandelbrot(c, maxiter):
    z = np.zeros_like(c)
    n = np.zeros_like(c, dtype=int)
    for i in range(maxiter):
        mask = np.abs(z) <= 2
        z[mask] = z[mask]**2 + c[mask]
        n[mask] = i
    n[np.abs(z) > 2] = maxiter
    return n


# Define the range of x and y values for the fractal
x = np.linspace(-2, 2, 1000)
y = np.linspace(-2, 2, 1000)

# Create a grid of complex numbers corresponding to each point on the plot
c = x[:,np.newaxis] + 1j*y[np.newaxis,:]

# Generate the fractal by computing the number of iterations for each complex number
fractal = mandelbrot(c, 100)


```

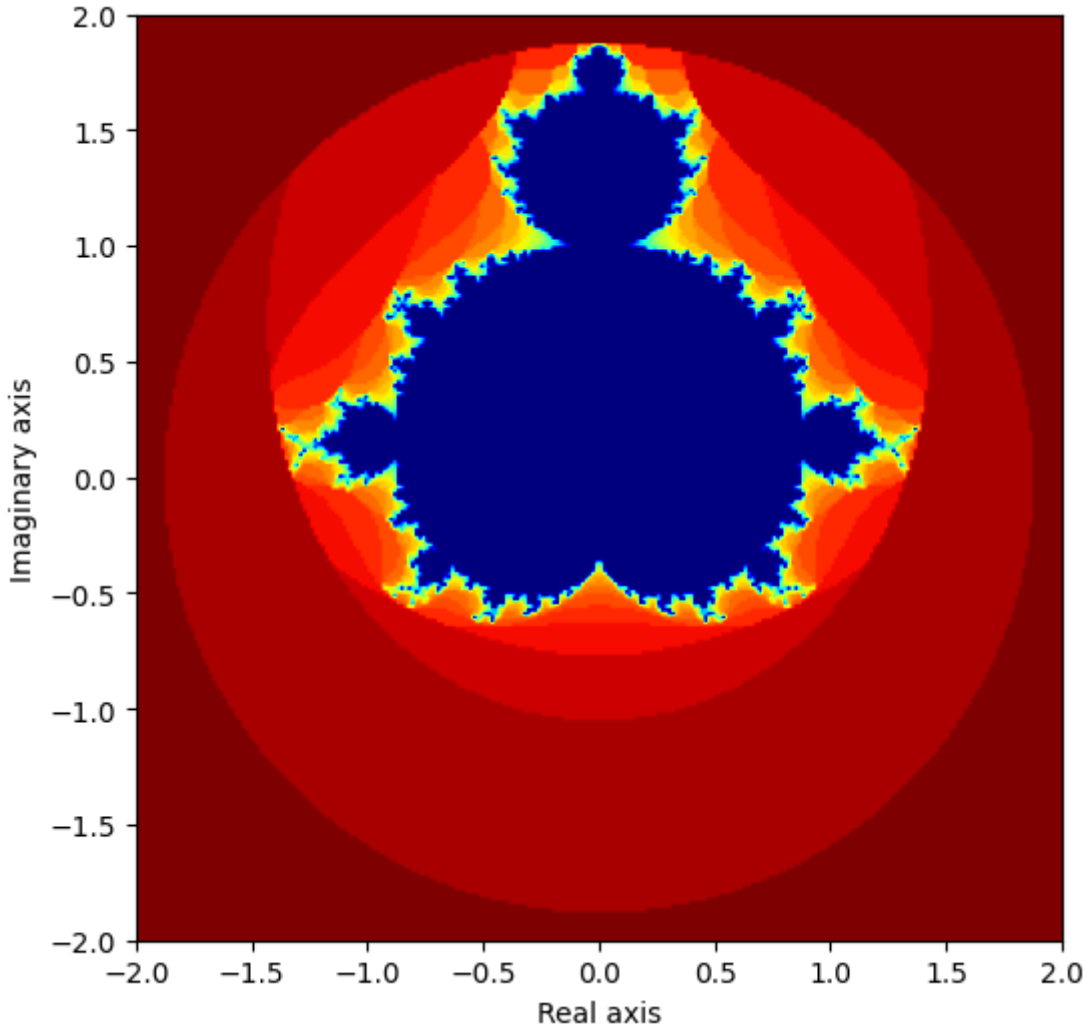
In [16]:

```
# Plot the fractal
plt.imshow(fractal.T, cmap='jet', extent=[-2, 2, -2, 2])
plt.xlabel('Real axis')
plt.ylabel('Imaginary axis')
plt.show()
```



In [25]:

```
# Plot the fractal
plt.imshow(image, cmap='jet', extent=[-2, 2, -2, 2])
plt.xlabel('Real axis')
plt.ylabel('Imaginary axis')
plt.show()
```



In [30]:
```
1 x
```

Out[30]: 2.0

In [32]:

```
# Create a Plotly figure
fig = go.Figure()

# Add a heat map trace to the figure to display the fractal
fig.add_trace(go.Heatmap(z=image,
                         x=xValue, y=yValue,
                         colorscale='Jet',
                         zmin=0, zmax=100,
                         showscale=False))

# Configure the layout of the figure
fig.update_layout(title='Mandelbrot Fractal',
                  xaxis_title='Real axis',
                  yaxis_title='Imaginary axis',
                 width=500,height=500,)

# Add a callback to handle mouse clicks on the plot
fig.update_layout(
    updatemenus=[dict(type="buttons",
                      buttons=[dict(label="Reset",
                                    method="update",
                                    args=[{"visible": [True]},
                                          {"xaxis.range": [-2, 2],
                                           "yaxis.range": [-2, 2]}])])])

def zoom_callback(trace, points, selector):
    if len(points.point_inds) == 0:
        return
    x, y = points.xs[0], points.ys[0]
    dx = (x.max() - x.min()) / 4
    dy = (y.max() - y.min()) / 4
    fig.update_layout(xaxis_range=[x.min() - dx, x.max() + dx],
                      yaxis_range=[y.min() - dy, y.max() + dy])

fig.data[0].on_click(zoom_callback)

# Display the figure
fig.show()
```

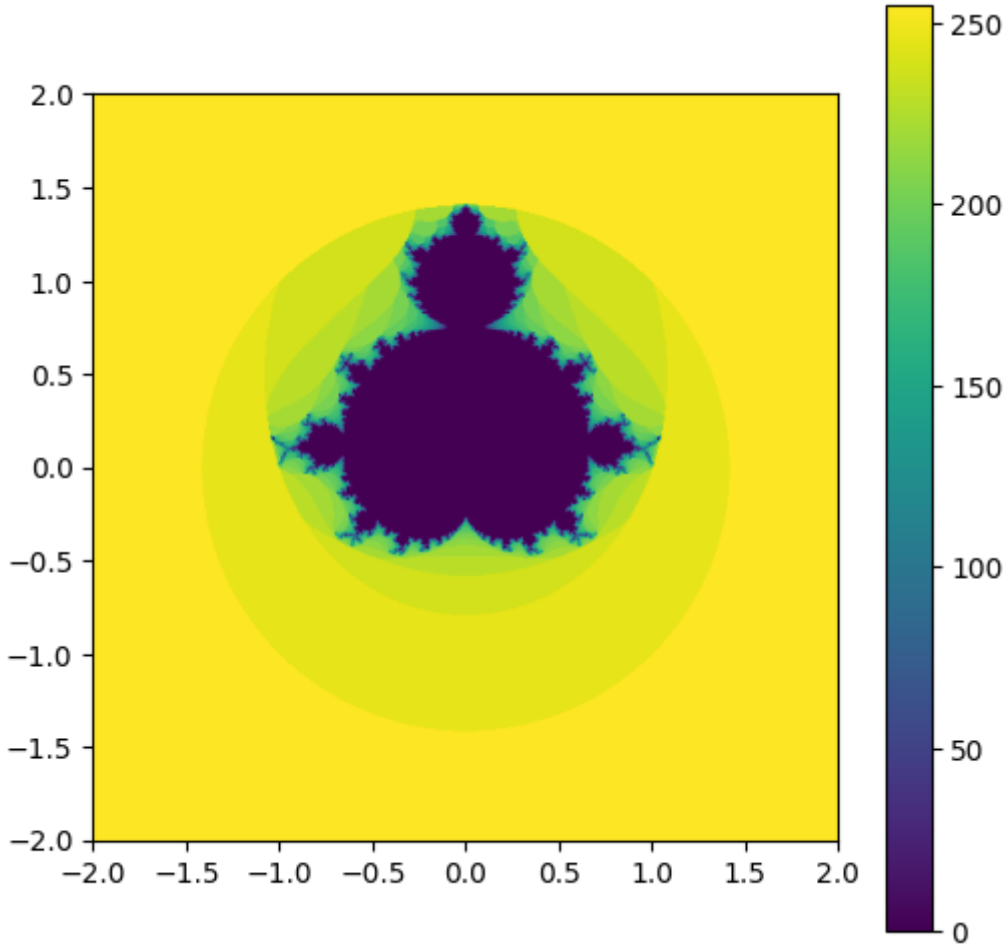
```
39
```

In [36]:

```
x = np.linspace(-2, 2, 1000)
y = np.linspace(-2, 2, 1000)

# Create a grid of complex numbers corresponding to each point on the plot
c = x[:,np.newaxis] + 1j*y[np.newaxis,:]

# Create a heatmap with the fractal as the z values
fig = go.Figure(data=go.Heatmap(z=image, x=x, y=y, zmin=0, zmax=100, colorscale='jet'))

# Set the axis limits and labels
fig.update_layout(
    xaxis_title='Real axis',
    yaxis_title='Imaginary axis',
    margin=dict(l=50, r=50, b=50, t=50),
    template='plotly_dark',width=500,height=500,
)

# Display the plot
fig.show()

```

In [35]:

```
plt.rcParams['figure.figsize'] = (6,6)

n=2

maxIteration = 30
z_min = -n-n*1j
z_max = n+n*1j

# Set the image size here
N=512

imageSize = [N,N]
image = np.zeros(imageSize,int)
xValue = np.linspace(z_min.real, z_max.real, imageSize[0])
yValue = np.linspace(z_min.imag, z_max.imag, imageSize[1])

extent=(z_min.real,z_max.real,z_min.imag,z_max.imag)

delta=2
for k in np.arange(imageSize[1]):
    for i in np.arange(imageSize[0]):
        iteration = 0
        x = xValue[i]
        y = yValue[k]
        while (x*x+y*y < delta) & (iteration < maxIteration):
           z=x+y*1j
           x= (z*z).real + xValue[i]
           y= (z*z).imag + yValue[k]
           iteration = iteration + 1
        if iteration == maxIteration:
           image[i,k] = 0.
        else:
           image[i,k] = 255.-iteration*255/maxIteration

plt.imshow(image,extent=extent)
plt.colorbar()
plt.show()
```



Another respect, consider the same mapping:

$$Z_{n+1} = Z_n^2 + C$$

but C is fixed.

Here the result, $C = 0 + 0.65i$.

In [2]:

```
import os

# Specify image width and height
w, h = 200, 200

# Specify real and imaginary range of image
re_min, re_max = -2.0, 2.0
im_min, im_max = -2.0, 2.0

z_min = -2.-2.j
z_max = 2.+2.j

# Pick a value for c
c = complex(0.0,0.65)

# Generate evenly spaced values over real and imaginary ranges
xValue = np.arange(z_min.real, z_max.real, (z_max.real - z_min.real) / w)
yValue = np.arange(z_max.imag, z_min.imag, (z_min.imag - z_max.imag) / h)

# Open output file and write PNG header info
fout = open('images/julia.pgm', 'w')
fout.write('P2\n# Julia Set image\n' + str(w) + ' ' + str(h) + '\n255\n')

for im in yValue:
    for re in xValue:
        z = complex(re,im)
        n = 255
        while abs(z) < 10 and n >= 5:
            z = z*z + c
            n -= 5
        # Write pixel to file
        fout.write(str(n) + ' ')
    # End of row
    fout.write('\n')
# Close file
fout.close()

os.system('convert images/julia.pgm images/julia.png')
```

```
39
```

Out[2]: 0

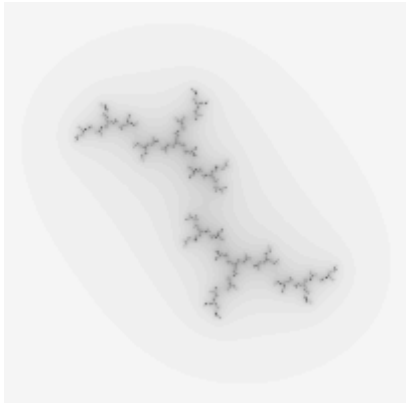
Now, make animation with $C = x + yi$ where point, (x, y), varies within a square region.

In [7]:

```
def makepic(filename,frame,c_im):
    # Open file and write PGM header info
    fout = open(filename, 'w')
    fout.write('P2\n# Julia Set image\n' + str(w) + ' ' + str(h) + '\n255\n')

    # Generate pixel values
    for im in yValue:
        for re in xValue:
            z = complex(re, im)
            c = complex(0,c_im)
            n = 255
            while abs(z) < 10 and n >= 5:
                z = z*z + c
                n = n - 5
            # Write pixel value to file
            fout.write(str(n) + ' ')
        fout.write('\n')

    # Close file
    fout.close()
    # create png file
    command = "convert tmp/%03d.pgm tmp/%03d.png" % (frame,frame)
    os.system(command)
    os.system("rm tmp/%03d.pgm" %(frame))

```

In [37]:

```
# Specify image width and height
w, h = 200, 200

# Specify real and imaginary range of image
z_min = -2.-2j
z_max = 2.+2j

# Pick a value for c
c = complex(0.0,0.65)

# Generate evenly spaced values over real and imaginary ranges
xValue = np.arange(z_min.real, z_max.real, (z_max.real - z_min.real) / w)
yValue = np.arange(z_max.imag, z_min.imag, (z_min.imag - z_max.imag) / h)

frame=0

# Iterate over a range of c values
for c_im in np.arange(0.0, 1, 0.04):
    # Increment frame counter
    frame += 1
    filename = "tmp/{0:03d}.pgm".format(frame)

    makepic(filename,frame, c_im)
    clear_output()
    i=Image("tmp/%03d.png" % (frame))
    display(i)
```



In [40]:

```
import math
import os

x, y, r = -1.5, -1.5, 3.0
p1, p2 = 1.0, 3.0
N = 100

pscale = pow(p2/p1, 1/float(N-1))
p = p1

for n in range(N):
    p = p1 * math.pow(pscale, n)

    #print "frame %d/%d: " % (n+1, N), x, y, r, p

    command = "./mandelbrot " + \
        str(x) + " " + str(y) + " " + \
        str(r) + " " + str(p) + " " + \
        "tmp/frame%03d.pgm" % (n)
    os.system(command)

    command = "convert tmp/frame%03d.pgm tmp/frame%03d.png" % (n,n)
    os.system(command)

    command = "rm tmp/frame%03d.pgm" % (n)
    os.system(command)

os.system("convert -delay 20 tmp/frame* images/ms.gif")
```

Out[40]: 0

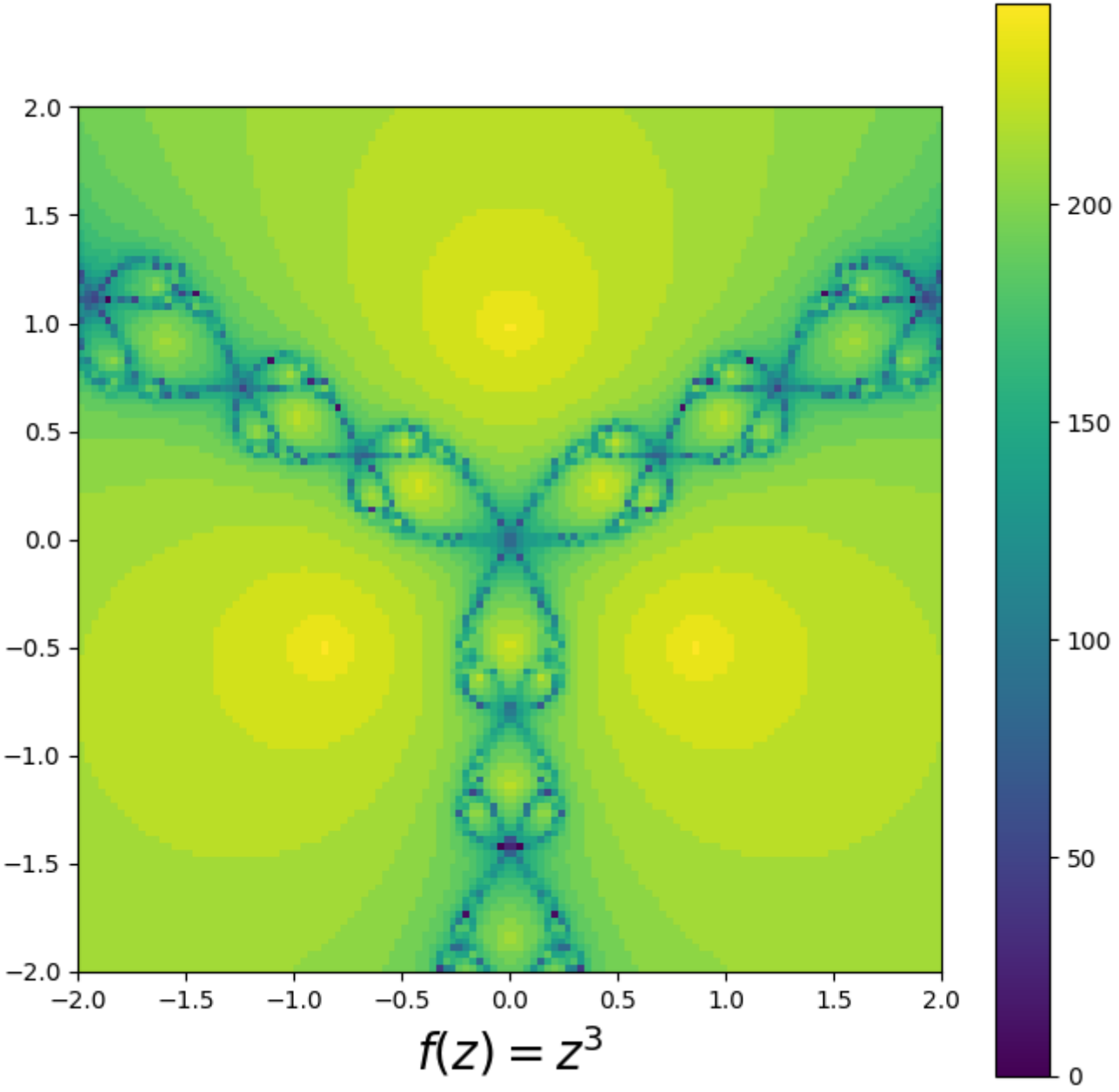
Newton Fractle

$$Z_{n+1} = Z_n - \frac{f(Z_n)}{f'(Z_n)}$$

In [39]:

```
plt.rcParams['figure.figsize'] = (8,8)
maxIteration = 30
z_min = -2-2j
z_max = 2+2j

# Set the image size here
N=128

imageSize = [N,N]
image = np.zeros(imageSize,int)
xValue = np.linspace(z_min.real, z_max.real, imageSize[0])
yValue = np.linspace(z_min.imag, z_max.imag, imageSize[1])

extent=(z_min.real,z_max.real,z_min.imag,z_max.imag)

delta=0.000001
for i in np.arange(imageSize[0]):
    for k in np.arange(imageSize[1]):
        iteration = 0
        x = xValue[i]
        y = yValue[k]
        z=x+y*1j
        while (iteration < maxIteration):
           try:
                #z -= (z**4+1)/(4*z**3)
                z -=  (z**3+1)/(3*z**2)
           except ZeroDivisionError:
                # possibly divide by 0
                continue
           #if (abs(z**4+1) < delta):
           if (abs(z**3+1) < delta):
                break
           iteration = iteration + 1

        if iteration == maxIteration:
           image[i,k] = 0
        else:
           image[i,k] = 255.-iteration*255/maxIteration

plt.imshow(image,extent=extent)
plt.xlabel(r'$f(z)=z^3$', size=20)
```

```
42 plt.colorbar()
43 plt.show()
```



In [13]:
```
def df(f):
    return
```

In [18]:
```
from sympy import diff,Symbol
Z = Symbol("Z")
```

In [19]:
```
f=Z**2+1;f
```

Out[19]:
```
Z**2 + 1
```

In [27]:
```
diff(f,Z).evalf(subs={Z:1+1j})
```

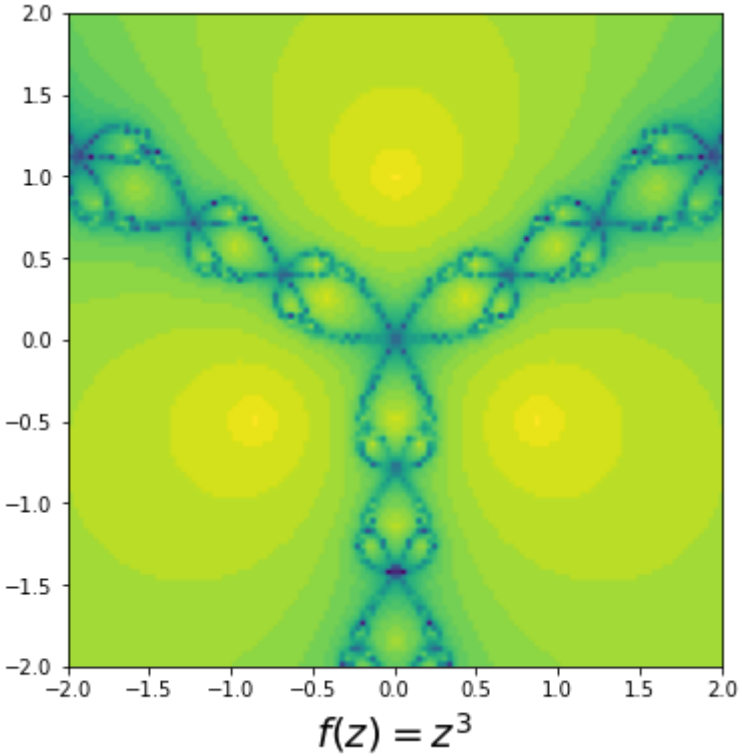
Out[27]:
```
2.0 + 2.0*I
```

In [ ]:

```
def NewtonFractle(f):
    for i in np.arange(imageSize[0]):
    for k in np.arange(imageSize[1]):
        iteration = 0
        x = xValue[i]
        y = yValue[k]
        z=x+y*1j
        while (iteration < maxIteration):
           try:
                #z -= (z**4+1)/(4*z**3)
                z -=  (z**3+1)/(3*z**2)

           except ZeroDivisionError:
                # possibly divide by 0
                continue
           #if (abs(z**4+1) < delta):
           if (abs(z**3+1) < delta):
                break
           iteration = iteration + 1

        if iteration == maxIteration:
           image[i,k] = 0
        else:
           image[i,k] = 255.-iteration*255/maxIteration

    plt.imshow(image,extent=extent)
    plt.xlabel(r'$f(z)=z^3$', size=20)
    plt.show()
```

In [19]:

```
for i in np.arange(imageSize[0]):
    for k in np.arange(imageSize[1]):
        iteration = 0
        x = xValue[i]
        y = yValue[k]
        z=x+y*1j
        while (iteration < maxIteration):
           try:
                #z -= (z**4+1)/(4*z**3)
                z -=  (z**3+1)/(3*z**2)
           except ZeroDivisionError:
                # possibly divide by 0
                continue
           #if (abs(z**4+1) < delta):
           if (abs(z**3+1) < delta):
                break
           iteration = iteration + 1

        if iteration == maxIteration:
           image[i,k] = 0
        else:
           image[i,k] = 255.-iteration*255/maxIteration

plt.imshow(image,extent=extent)
plt.xlabel(r'$f(z)=z^3$', size=20)
plt.show()
```



Probability

```
An army ex-general in Taiwan said:
    「一發飛彈攔截率 70 % 三發就是210%」
```

But what is the answer if false?

The Myth of the One:

```
    One is the my favoritest because our birthdays are same!
```

In [67]:

```
print(" The probability of what No one has same birthdays...  ")
p=1-1/365
for i in range(60-1):
    i=i+2
    q=(1- p)
    print( "%s - person group: %4.3f (%4.2f%%)" %(i,p,q*100 ))
    p=p*(1-i/365)
```

```
 The probability of what No one has same birthdays...
2 - person group: 0.997 (0.27%)
3 - person group: 0.992 (0.82%)
4 - person group: 0.984 (1.64%)
5 - person group: 0.973 (2.71%)
6 - person group: 0.960 (4.05%)
7 - person group: 0.944 (5.62%)
8 - person group: 0.926 (7.43%)
9 - person group: 0.905 (9.46%)
10 - person group: 0.883 (11.69%)
11 - person group: 0.859 (14.11%)
12 - person group: 0.833 (16.70%)
13 - person group: 0.806 (19.44%)
14 - person group: 0.777 (22.31%)
15 - person group: 0.747 (25.29%)
16 - person group: 0.716 (28.36%)
17 - person group: 0.685 (31.50%)
18 - person group: 0.653 (34.69%)
19 - person group: 0.621 (37.91%)
20 - person group: 0.589 (41.14%)
21 - person group: 0.556 (44.37%)
22 - person group: 0.524 (47.57%)
23 - person group: 0.493 (50.73%)
24 - person group: 0.462 (53.83%)
25 - person group: 0.431 (56.87%)
26 - person group: 0.402 (59.82%)
27 - person group: 0.373 (62.69%)
28 - person group: 0.346 (65.45%)
29 - person group: 0.319 (68.10%)
30 - person group: 0.294 (70.63%)
31 - person group: 0.270 (73.05%)
32 - person group: 0.247 (75.33%)
33 - person group: 0.225 (77.50%)
34 - person group: 0.205 (79.53%)
35 - person group: 0.186 (81.44%)
36 - person group: 0.168 (83.22%)
37 - person group: 0.151 (84.87%)
38 - person group: 0.136 (86.41%)
39 - person group: 0.122 (87.82%)
40 - person group: 0.109 (89.12%)
41 - person group: 0.097 (90.32%)
```

```
42 - person group: 0.086 (91.40%)
43 - person group: 0.076 (92.39%)
44 - person group: 0.067 (93.29%)
45 - person group: 0.059 (94.10%)
46 - person group: 0.052 (94.83%)
47 - person group: 0.045 (95.48%)
48 - person group: 0.039 (96.06%)
49 - person group: 0.034 (96.58%)
50 - person group: 0.030 (97.04%)
51 - person group: 0.026 (97.44%)
52 - person group: 0.022 (97.80%)
53 - person group: 0.019 (98.11%)
54 - person group: 0.016 (98.39%)
55 - person group: 0.014 (98.63%)
56 - person group: 0.012 (98.83%)
57 - person group: 0.010 (99.01%)
58 - person group: 0.008 (99.17%)
59 - person group: 0.007 (99.30%)
60 - person group: 0.006 (99.41%)
```

Check the fact of Girl team of Japan, 乃木阪46 (https://zh.wikipedia.org/wiki/%E4%B9%83%E6%9C%A8%E5%9D%8246)

8月8日

北川悠理
賀喜遙香

8月20日

秋元真夏
奥田伊呂波

Problem

1. Confirm or correct the statement of ex-army-general:

```
Suppose that the probability of accuracy of hitting target is 70% for Missile of Sky Bow type II, there
is 210% to hit the target if fire three missiles.
```

2. (Soap Comedy) Is it possible:

```
A young OL ran on the street and ate her bread-toast because she was got up too late to go to her offic
e. But she hit somebody at the corner by accident; suddenly, she felt she was so lucky because the one,
she hit, is the one she admired.
```

Using the given probabilities, we can estimate the probability of the young OL running on the street, eating her bread-toast, hitting somebody, and meeting the style she admired as follows:

- Probability of running on the street = 0.1
- Probability of eating bread-toast = 0.3
- Probability of hitting somebody = 0.5
- Probability of meeting the one she admired = 0.2
- Assuming that these events are independent, we can estimate the probability of all four events occurring as follows:

P(all events) = P(running on the street) * P(eating bread-toast) * P(hitting somebody) * P(meeting the style she admired)

$$P(\text{all events}) = 0.1 \times 0.3 \times 0.5 \times 0.2 = 0.003$$

Therefore, the estimated probability of the young OL running on the street, eating her bread-toast, hitting somebody, and meeting the style she admired is 0.003, or 0.3%. It's important to note that this estimate is based on the given probabilities and assumptions of independence, and may not reflect the actual likelihood of these events occurring in real life.

In [ ]:
```
1
```