Chapter 7

Fatou Sets and π

7.1 Finding π in a Fatou set

Since the Mandelbrot set is a map of Julia sets (see section 4.1), are the Julia sets along paths into a pinch point in the Mandelbrot set (such as the line $c = \frac{1}{4} + \varepsilon$) related to π too? What about looking at how many points each possible number of iterations has? In figure 7.1 this would be the area of each colour. If we multiply the number of iterations

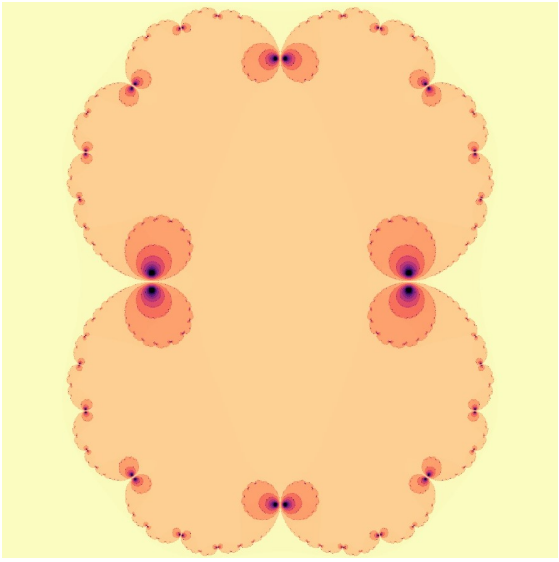

Figure 7.1: The Fatou set with $c = 0.251$.

before escaping by a power of ε we find that these number are close to multiples of π as $\varepsilon \to 0$. So looking at Julia sets with $c = \frac{1}{4} + \varepsilon$ and plotting the array of the number of iterations against its index we see that in figure 7.2 this staircase pattern emerges as $\varepsilon \to 0$. See appendix A.6 for the whole code.

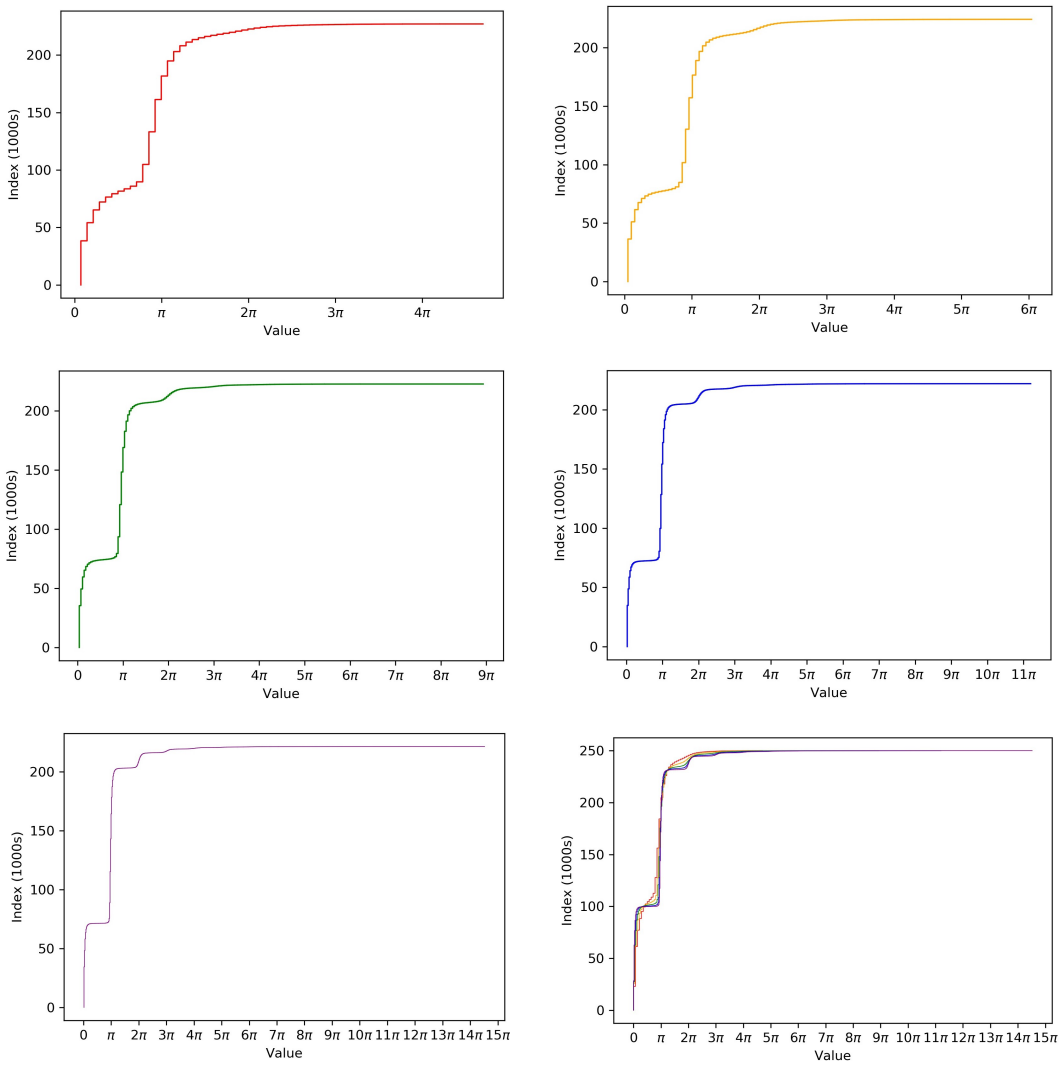

Figure 7.2: From right to left and top to bottom: the $\sqrt{\varepsilon}N$ lists for $c = \frac{1}{4} + \varepsilon$ Julia sets with $\varepsilon = 0.05, 0.025, 0.0125, 0.00625, 0.003125$, and a graph of overlaid lists.

This shows us that for small ε these values are all approximately multiples of π.

The reason why for larger ε the graph has those little steps everywhere is because the maximum array value is not very high (since $\frac{1}{4} + \varepsilon$ is further away from the Mandelbrot set) and since the array is made up of integers there are not many different values for N.

We knew that the origin would be π since that is the point with $z_0 = 0$ and therefore

one of the points we used in chapter 3; but why are all other points multiples of π too?

7.2 Explaining how the π multiples appear

We can plot the arrays for values of c with small ε and see that the space is split into distinct areas corresponding to multiples of π, as in figure 7.3. You can see that this

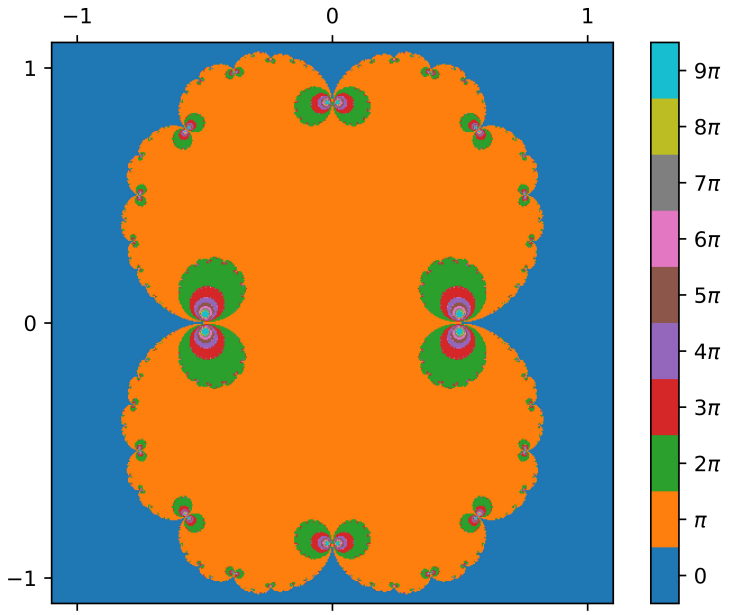

Figure 7.3: $\sqrt{\varepsilon}N$ array for the $c = 0.25 + 0.001$ Fatou set.

creates a fractal pattern with these concentric circles around the edge of the shape centred on particular points (I will hereon refer to them as clams due to their resemblance to an open clam). Note that the scale stops at 9π due to the limiting effect of the iteration maximum value ('imax'), in reality this pattern goes on forever since it is a Fatou set. The points at the centre of each clam must be fixed points of the iterative equation (4.1). We can find the fixed points by iterating the equation a number of times then solving for it being equal to z, for example doing this four times we would need to solve:

$$z = (((z^2 + c)^2 + c)^2 + c)^2 + c,$$

which gives points that are periodic with the period being a factor of 4 (1, 2, or 4).

The two points in the Fatou set near the centre of the open clam (like pearls in a clam) to the right of the origin are both fixed points (of period 1) and thus the solutions to:

$$\begin{aligned} z &= z^2 + c, \\ z &= \frac{1}{2} \pm \frac{\sqrt{10}}{100} i. \end{aligned} \tag{7.1}$$

The points on the other side of the set are not fixed points, but they do go to the points (7.1) after one iteration of (4.1), and points in the left clam go to the right clam after one iteration. In fact this happens for all points in all of the other clams, they move in a sequence going to clams of larger size (but staying in regions of the same colour) one by one, see figure 7.4 for an example.

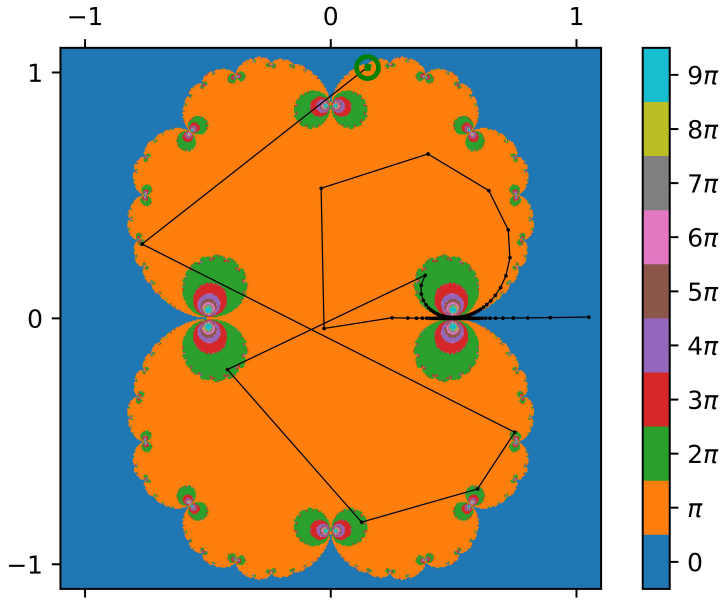

Figure 7.4: The (4.1) sequence for $z_0 = 0.148 + 1.020i$.

From figure 7.4 we can see that the vast majority of terms in the sequence occur between the two fixed points (7.1), the sequence goes in-between these two points twice and it started in the green area which has value 2π. This pattern happens again in figure 7.5 where z_0 is in a 7π (grey) area and the sequence goes between (7.1) seven times. This means that each journey between (7.1) contributes approximately π to the $\sqrt{\varepsilon}N$ value and hence we can estimate how many points each journey contributes to the sequence by simply dividing π by $\sqrt{\varepsilon}$. For this value of ε $(= 0.001)$

$$\frac{\pi}{\sqrt{\varepsilon}} = \frac{\pi}{\sqrt{10^{-3}}} = 10\sqrt{10}\pi \approx 99.346$$

should be how many terms are contributed to the sequence for every journey between the (7.1) fixed points. In fact figure 7.5 shows exactly 696 terms and $10\sqrt{10}\pi \times 7 \approx 695.42$ (5 s.f.), so this formula is evidently a good estimator.

You may have noticed that each clam has smaller clams at the edge of each region, and these smaller clams start with the region of the π-multiple 1 higher than the region they

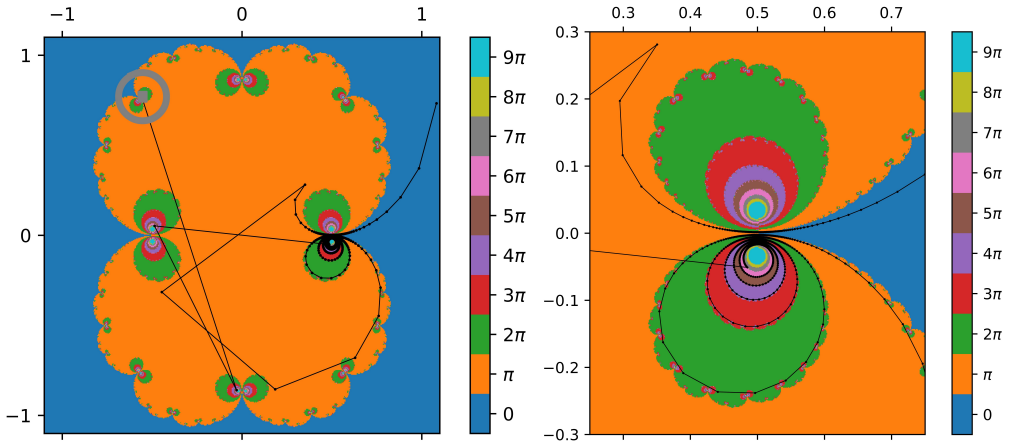

Figure 7.5: The sequence for $z_0 = -0.558 + 0.770i$.

are in. This pattern repeats on each ring with a fractal nature of course. If we start a sequence with z_0 in one of these smaller clams the sequence will move anti-clockwise along to the other small clams eventually make its way into the large clams, but not before traveling between the fixed points and thus contributing π to $\sqrt{\varepsilon}N$ in the process. See figure 7.6 for an example.

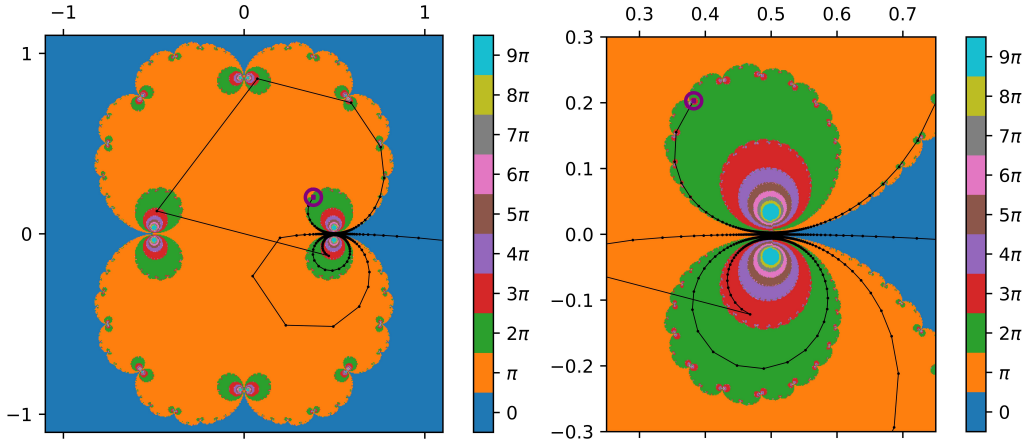

Figure 7.6: The sequence for $z_0 = 0.382 + 0.203i$.

7.3 Fatou sets near $-\frac{3}{4}$

For other paths into point-bridges such as $-0.75 + \varepsilon i$ we also get the same staircase property at multiples of π, although this time we multiply the array by ε rather than

$\sqrt{\varepsilon}$, like in chapter 3. See figure 7.7.

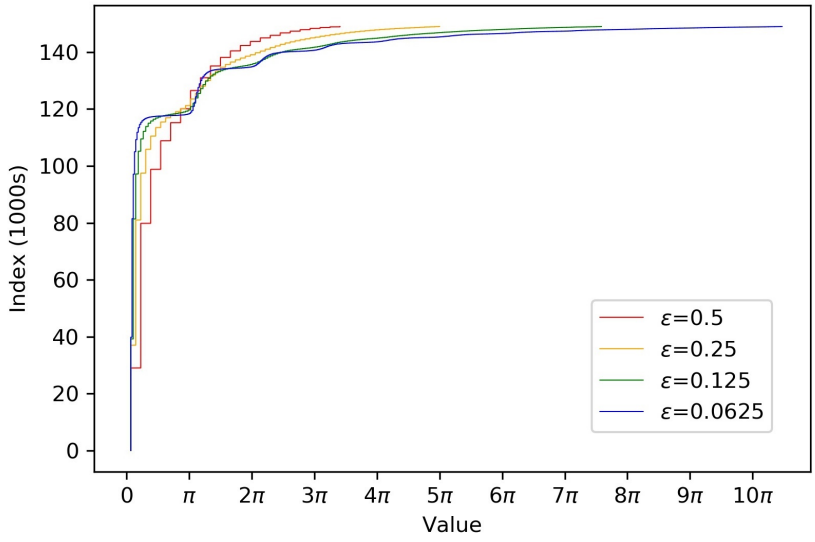

Figure 7.7: The εN lists for $c = -\frac{3}{4} + \varepsilon i$ Julia sets with decreasing ε.

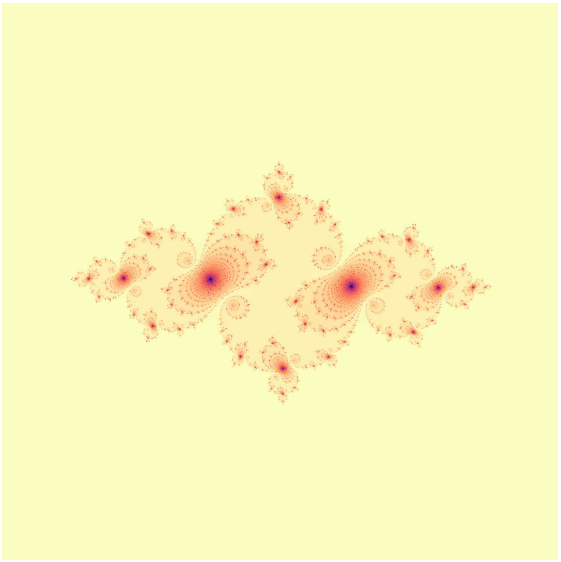

Figure 7.8: The Fatou set with $c = -0.75 + 0.05i$.

The fixed points with period 1 are at the large 'black hole' to the left and to the very right tip of the set. Similarly to the Fatou set of $c = 0.251$ most of each sequences terms

occur around the first of these fixed points, as illustrated in figure 7.9.

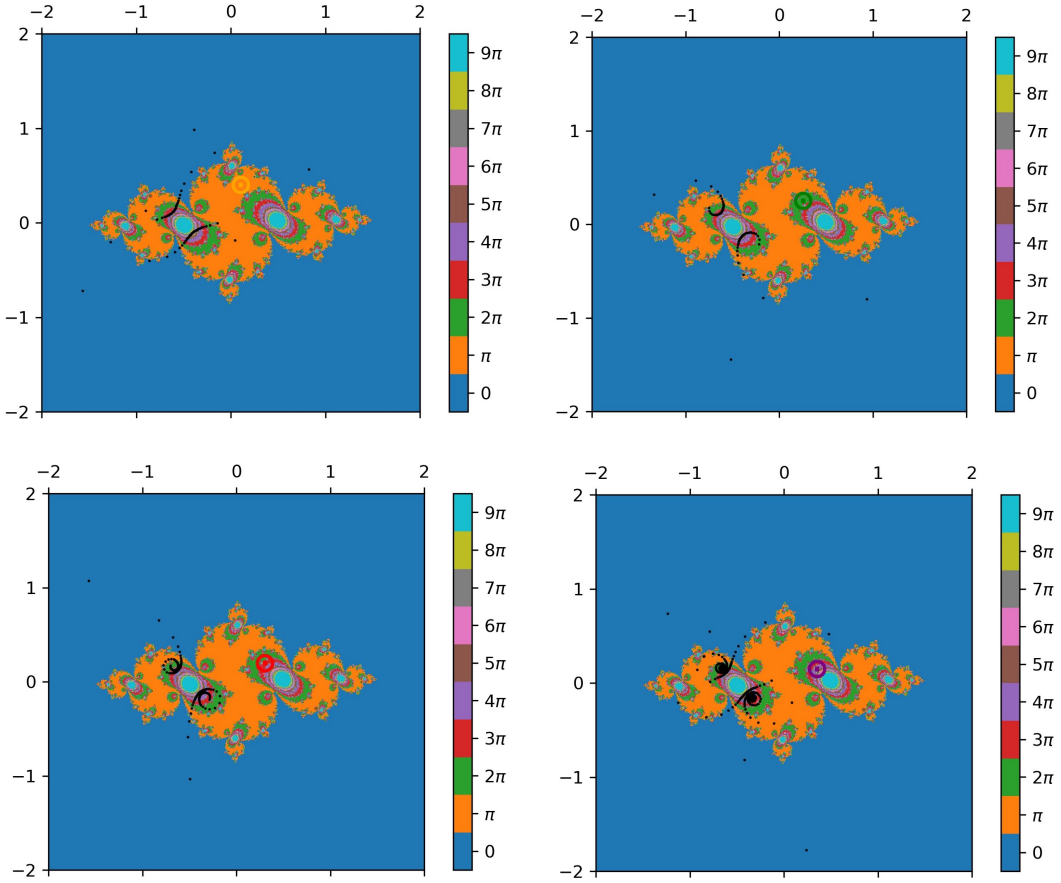

Figure 7.9: The εN arrays with overlayed diverging sequence terms for initial z_0 in different regions.

It appears as though half the terms occur to the top-left of the fixed point and the other half to the bottom-right.

Chapter 8

Conclusion

We have shown how and why π occurs in the billiard problem, essentially due to the conservation of energy being an ellipse. We also showed how and why π appears near $\frac{1}{4}$ in the Mandelbrot set, it was because the sequence could be approximated by a function with the tangent function in, and the tangent function is fundamentally related to π. Later we looked breifly at Julia and Fatou sets. Then we studied the structure of the Mandelbrot set and its relation to periodicity of orbits. We also looked at the Buddhabrot and investigated subsets of the Buddhabrot. Finally we found a relation between π and particular Fatou sets, then explored how it occurs.

But there were many things we did not conclusively prove, but still merit further investigation. Why is there a difference in the separate π-paths at -1.25, -0.75, and 0.25? Can we adjust the paths so that they give a better result? It seems that we can because adjusting the parabola into -1.25, so that it was further from the Mandelbrot set, gave a result closer to π. How would a parabola into 0.25 change the results? How can you prove the π result at paths in the complex plane? Remember we only proved the case for $0.25 + \varepsilon$ which is just on the real axis.

We could also explore whether there is a way to explain the strands and bulb sizes in the Mandelbrot set, or is that just a property of the fractal? Are there anymore interesting properties in subsets of the Buddhabrot?

It would also be interesting to explore more into the fixed/periodic points in Julia and Fatou sets. Such as why they occur and where they occur in relation to the sets structure. In the Julia set with $c = -1.11235 + 0.25315i$ (see figure 5.9) the periodic points are located at areas with significant structure, see figure 8.1. Notice how one red point is at the first left junction between the largest fractal pieces and the other red point is at the rightmost point in the set. The blue points are both at junctions with three branches, and the green points are all at points on perpendicular offshoots to the direction of the smallest fractal pieces (all on the edges of the set).

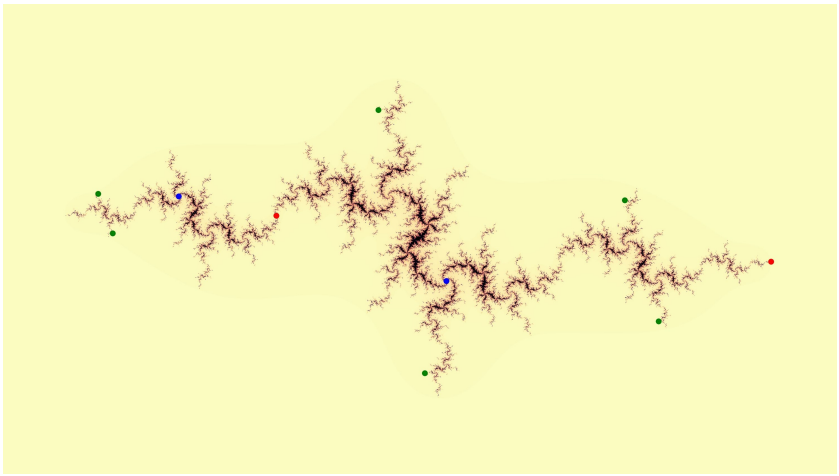

Figure 8.1: The Julia set at $c = -1.11235 + 0.25315i$ with periodic points marked: points with a period of 1 (completely fixed points) are red, with a period of exactly 2 are blue, and with a period of exactly 3 are green.

I also made several plots of fixed and periodic points on the $c = 0.251$ Fatou set, see figure 8.2. In the figure we see that the first seven images have periodic points along the boundary of the region where $\sqrt{\varepsilon}N > \frac{\pi}{2}$, the other images no longer have this, and the last two images do not even have the solutions with period 1 (7.1). This is likely due to a floating point error in the numerical methods that were used to calculate the points, but what about the eighth and ninth images?

The task of finding π in an unexpected place is more difficult than I first anticipated; after all there are many places to look for an unexpected occurrence of π. It would perhaps be easier to work backwards like Gregory Galperin seemingly did in his billiard problem, rather than stumbling upon π as David Boll did in the Mandelbrot set.

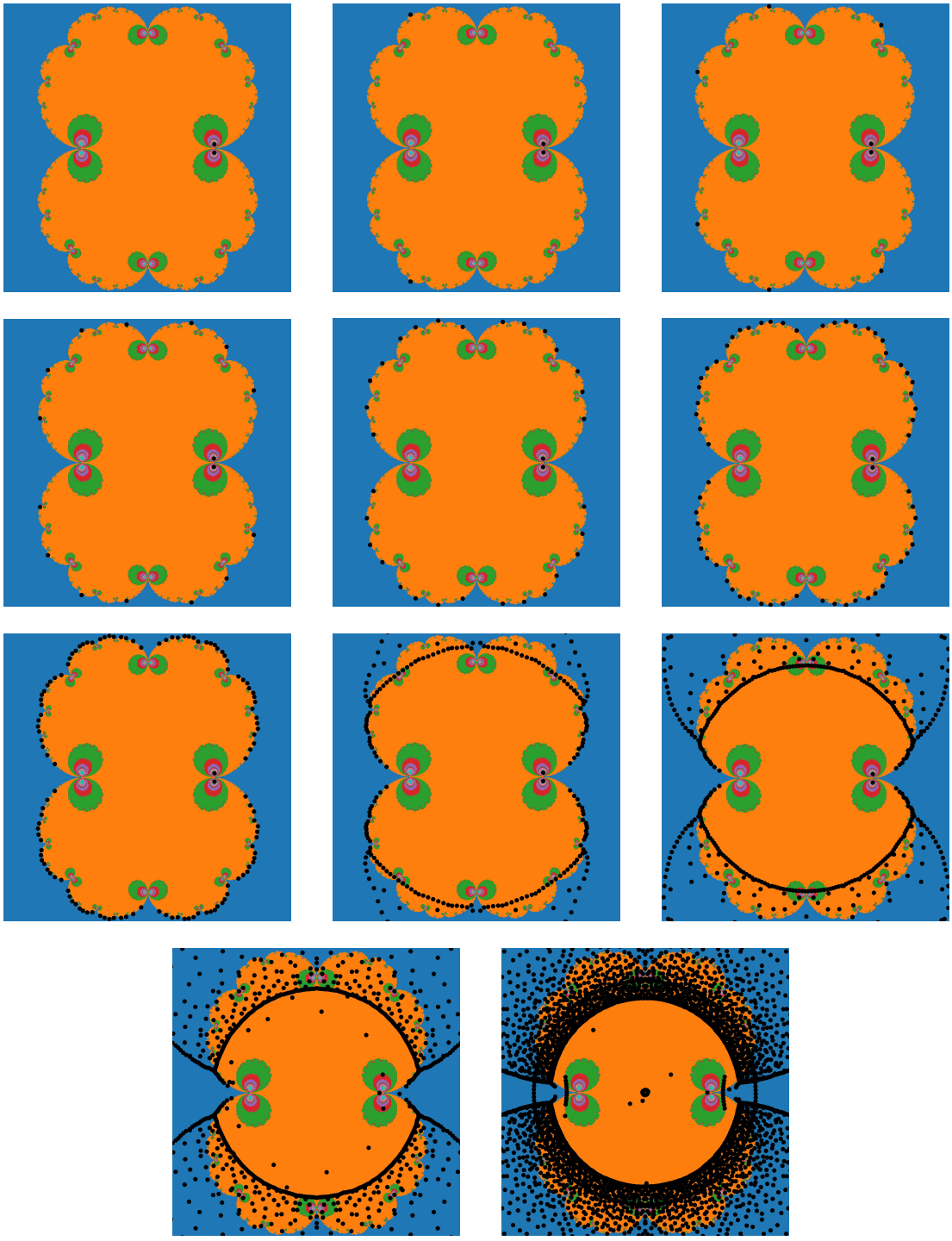

Figure 8.2: The Fatou set with $c = 0.251$ with periodic points overlayed where the periods are (from left to right and top to bottom) factors of: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, and 11.

Bibliography

[1] Keith Conrad, "The Gaussian Integral", 25 May 2019. https://kconrad.math.uconn.edu/blurbs/analysis/gaussianintegral.pdf

[2] Henry Ricardo, "A Modern Introduction to Differential Equations", Page 428, 23 March 2016.

[3] Raymond Ayoub, "Euler and the zeta function", The American Mathematical Monthly, Vol. 81, Pages 1067-1086, 1974.

[4] Gregory Galperin, "Playing Pool with π (the number π from a billiard point of view)", Regular and Chaotic Dynamics, Vol. 8, No. 4, Pages 375-394, 2003.

[5] Benoit Mandelbrot, "The fractal geometry of nature", 1983.

[6] Kenneth Falconer, "Fractal geometry: mathematical foundations and applications", Page 120, 1990.

[7] Elibarzilay, Roserra Code, "File:RacketSierpinski.png", 25 April 2013. https://rosettacode.org/wiki/File:RacketSierpinski.png.

[8] David Boll, Google Groups, "Pi & the Mandelbrot set, also DiffEQ question" 24 April 1991. https://groups.google.com/forum/?hl=en#!topic/sci.math/jHYDf-Tm0-8.

[9] Xah Lee, "Xah's Top 10 Math Wonders", 18 October 2010. http://xahlee.info/math/top_math_wonders.html.

[10] Aaron Klebanoff, "π in the Mandelbrot Set", Fractals, Vol. 9, No. 4, Pages 393-402, 2001.

[11] David Petry, Newsgroups: alt.fractals, "Pi and the Mandelbrot set", 27 March 1992. https://people.math.osu.edu/edgar.2/piand.html.

[12] Gaston Julia, "Mémoire sur l'iteration des fonctions rationnelles", Journal de Mathématiques Pures et Appliquées, Vol. 8, Pages 47–245, 1918.

[13] Pierre Fatou, "Sur les substitutions rationnelles", Comptes Rendus de l'Académie des Sciences de Paris, Vol. 164, Pages 806–808 and Vol. 165, Pages 992–995, 1917.

[14] Heinz-Otto Peitgen, Peter Richter. “Julia Sets and the Mandelbrot Set.” In The Beauty of Fractals: Images of Complex Dynamical Systems. Berlin: Springer-Verlag, Page 161, 1986.

[15] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. Chaos and Fractals: New Frontiers of Science. Springer-Verlag, Page 794, 1992.

[16] Adrien Douady, John Hubbard, “Etude dynamique des polynomes complexes”, Prepublications mathematiques d’Orsay 2/4 (1984/1985).

[17] Jeremy Kahn, “The Mandelbrot Set is Connected: a Topological Proof”, 8 August 2001.

[18] Hoehue, Frank Klemm, Wikimedia Commons, “File:Mandelbrot Set - Periodicitires coloured.png”, originally upload 31 October 2005, last updated 12 August 2017. `https://commons.wikimedia.org/wiki/File:Mandelbrot_Set_%E2%80%93_Periodicities_coloured.png`

[19] Georg Cantor, ”Ueber eine Eigenschaft des Inbegriffes aller reellen algebraischen Zahlen”, Journal für die Reine und Angewandte Mathematik, Vol. 7, Pages 258–262, 1874.

[20] Dominic Ford, “Online Mandelbrot Set Plotter”, ScienceDemos.org.uk, last updated: 4 April 2020. `https://sciencedemos.org.uk/mandelbrot.php`

[21] Melinda Green, superliminal.com, “The Buddhabrot Technique”, last updated: 13 April 2019. `http://superliminal.com/fractals/bbrot/bbrot.htm`

Appendix A

Python Code

Below is some of the Python code I have written to produce many of the various images in the report. I felt it was not necessary to include the code in its entirety to the main body of the report and hence put it here in the appendix. Although where I felt the code was useful to explain and more complex than translating a method into code, it was done so such as in sections 3.4, 4.2, and 6.2.

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


N = 0.5

# setting up the graph and axis
fig,ax = plt.subplots(1,1)
ax.set_aspect('equal')

ax.spines['left'].set_position('center')
ax.spines['bottom'].set_position('center')

ax.spines['right'].set_color('none')
ax.spines['top'].set_color('none')

plt.xticks([])
plt.yticks([])

# plotting the conservation of energy circle
energyx = np.concatenate((np.linspace(-1.0,-0.9,30),np.linspace(-0.9,0.9,40),np.linspace(0.9,1.0,30)))
energyy = np.array([(1-energyx[i]**2)**0.5 for i in range(len(energyx))])
plt.plot(energyx, energyy,'-b')
plt.plot(energyx, -energyy,'-b')

# calculating the velocities
u = np.array([0])   # y coords
v = np.array([-1])  # x coords

c = u[-1] + 10**N*v[-1]
vv = (10**N*c+(-c**2+100**N+1)**0.5)/(1+100**N)
uu = -10**N*vv + c
u = np.concatenate((u,np.array([uu,-uu])))
v = np.concatenate((v,np.array([vv,vv])))

while u[-1]>0:   #(0<=u[-1]<=v[-1]) == False
    c = u[-1] + 10**N*v[-1]
    vv = (10**N*c+(-c**2+100**N+1)**0.5)/(1+100**N)
    uu = -10**N*vv + c
    u = np.concatenate((u,np.array([uu,-uu])))
    v = np.concatenate((v,np.array([vv,vv])))

v = v[:-1]          # last u was +ve so couldn't have hit wall --> remove last pair
u = u[:-1]

print(len(v)-1)     # counting lines not points, hence length -1

# plotting the coordinates and paths
plt.plot(v,u,'deeppink')
plt.plot(v,u,'r.')

# plotting the end condition
xl = 10**N/(1+100**N)**0.5
plt.plot([0,xl],[0,xl*10**-N],'g-')

plt.show()
```

Figure A.1: Creating a phase diagram of the colliding balls.

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


imax = 1000

# define the function to iterate the Mbrot sequence
def inSet(c):
    iterate = 0
    for i in range(imax):
        if abs(iterate) > 2:
            return i
        else:
            iterate = iterate**2 + c
    return imax

# set-up the array representing the complex plane
dim = 1001
xmin,xmax = -2,1
ymin,ymax = -1.5,1.5
xList = np.linspace(xmin,xmax,dim)
yList = np.linspace(ymin,ymax,dim)
points = np.zeros((len(yList),len(xList)))
for i in range(len(xList)-1,-1,-1):
    for j in range(len(yList)):
        pointIn = inSet(complex(xList[i],yList[j]))
        points[j,i] = pointIn

plt.ion()
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlim(xmin,xmax)
ax.set_ylim(ymin,ymax)
ax.set_aspect('equal')

power = 0.5
cax = ax.matshow(points**power,cmap=cm.magma_r,origin='lower',extent=(xmin,xmax,ymin,ymax))
plt.savefig('Mbrot.jpg',format='jpg',dpi=300)
plt.show()
```

Figure A.2: Generating a picture of the Mandelbrot set.

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



c = complex(-0.75,0.05)
R = abs((-1 - (1+4*abs(c))**0.5)/2)      # escape radius and quadratic formula

imax = 100

# julia sequence length function
def inSet(x):
    iterate = x
    for i in range(imax):
        if abs(iterate) > R:
            return i
        else:
            iterate = iterate**2 + c
    return imax

# values into the array
dim = 401
xList = np.linspace(-2,2,dim)
yList = np.linspace(-2,2,dim)
julia = np.zeros((len(yList),len(xList)))
for i in range(len(xList)-1,-1,-1):
    for j in range(len(yList)):
        pointIn = inSet(complex(xList[i],yList[j]))
        julia[j,i] = pointIn

# plot the array
plt.ion()
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)
ax.set_aspect('equal')

cax = ax.matshow(julia,cmap=cm.magma_r,origin='lower',extent=(-2,2,-2,2))

# print the maximum value in the array if <imax Fatou set
print(np.amax(julia))

plt.savefig('Julia.jpg',format='jpg',dpi=300)
plt.show()
```

Figure A.3: Generating a picture of a Julia or Fatou set.

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

# array set-up
dim = 401
xList = np.linspace(-2,2,dim)
yList = np.linspace(-2,2,dim)
julia = np.zeros((len(yList),len(xList)))
diff = xList[1]-xList[0]

imax = 100

# mbrot function which returns N and the sequence
def inSet(c):
    zn = [0]
    for i in range(imax):
        if abs(zn[-1]) > 2:
            return i,zn
        else:
            zn += [zn[-1]**2 + c]
    return imax,zn

# the function to incriment the array for each term in the sequence for c
def path(c):
    i,zn = inSet(c)
    if i != imax:
        for z in zn[1:-1]:  # no z_0=0 since it is in every sequence and no last term since it has modulus >2
            x = z.real
            y = z.imag
            julia[int(round(y/diff)+(dim-1)/2)][int(round(x/diff)+(dim-1)/2)] += 1.0

# the function for the anti-buddhabrot
def antipath(c):
    i,zn = inSet(c)
    if i == imax:
        for z in zn[1:-1]:
            x = z.real
            y = z.imag
            julia[int(round(y/diff)+(dim-1)/2)][int(round(x/diff)+(dim-1)/2)] += 1.0

# generating the paths for random c
cdim = 4000
crands = [complex(4*np.random.random()-2,4*np.random.random()-2) for i in range(cdim**2)]
for i in crands:
    path(i)         # change for antipath(i) to make the anti-buddhabrot

plt.imshow(julia,cmap=cm.gray)  # need julia**0.5 for anti to exaggerate image

plt.savefig('Buddhabrot.jpg',format='jpg',dpi=300)
plt.show()
```

Figure A.4: Generating a picture of the Buddhabrot or Anti-Buddhabrot.

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

# function to find N for a Julia sequence
def inSet(z,c):
    iterate = z
    for i in range(100):
        if abs(iterate) > 2:
            return i
        else:
            iterate = iterate**2 + c
    return i

# make the matrix for the background image
dim = 1001
xList = np.linspace(-2,2,dim)
yList = np.linspace(-2,2,dim)
julia = np.zeros((len(yList),len(xList)))
for i in range(len(xList)-1,-1,-1):
    for j in range(len(yList)):
        julia[j,i] = inSet(complex(xList[i],yList[j]),complex(-1.25-0.001**2,0.001))

# plot the Julia/Fatou set in the background
plt.ion()
fig=plt.figure()
ax=fig.add_subplot(111)
ax.set_xlim(-2,2)
ax.set_ylim(-2,2)
ax.set_aspect('equal')

cax = ax.matshow(julia,cmap=cm.magma_r,origin='lower',extent=(-2,2,-2,2))


# function to find the next term in the sequence
def f(z,c):
    return z**2 + c

# e is the list of epsilons and p1 the first sequence with z_0 at the start of the parabola
e = np.linspace(0.001,0.021,400)
p1 = [complex(-1.25-e[0]**2,e[0])]

n = 400    # 2000 includes all points, last lists are NAN (not sure why)

# p is the list of lists of z_ns i.e. p[0] is the list of z_1s (z_1=c) p[1] the list of z_2s etc.
p = [p1]
while abs(p[-1][0]) <= 2:
    p += [[f(p[len(p)-1][0],p[0][0])]]

# add elements to the p[i]s for the sequences with c in e
while (abs(p[0][-1])<=2) and (len(p[0])<len(e)):
    p[0] += [complex(-1.25-e[len(p[0])]**2,e[len(p[0])])]
for i in range(len(p)-1):
    while (abs(p[i+1][-1])<=2) and (len(p[i+1])<len(p[i])):
        p[i+1] += [f(p[i][len(p[i+1])],p[0][len(p[i+1])])]

# plot the sequences
px = []
py = []
for i in range(n):
    px += [[pp.real for pp in p[i]]]
    py += [[pp.imag for pp in p[i]]]

colours = ['r','orange','g','b']
for i in range(n):
    plt.plot(px[i],py[i],color=colours[i%4],linestyle='None',marker='.',markersize=1)

plt.savefig('Julia.jpg',format='jpg',dpi=1200)
plt.show()
```

Figure A.5: Plotting sequences on-top of a Julia or Fatou set.

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

# the function to convert the matrix into a sorted list
def stairs(mat):
    x = np.sort(mat,axis=None)
    return x[100000:-1000]  # removing the excess of points at the ends

# the loop to make different staircases
for i in range(4):
    imageversion = i+1
    eps = 2**(-i-1)
    c = complex(-0.75,eps)
    R = abs((1 + (1+4*abs(c))**0.5)/2)

    # function to find N
    def inSet(x):
        iterate = x
        for i in range(1000):
            if abs(iterate) > R:
                return i
            else:
                iterate = iterate**2 + c
        return (i+1)

    # generating the matrix
    xmin, xmax = -2,2
    ymin, ymax = -2,2
    xdim,ydim = 500,500
    xList = np.linspace(xmin,xmax,xdim)
    yList = np.linspace(ymin,ymax,ydim)
    julia = np.zeros((len(yList),len(xList)))
    for i in range(len(xList)-1,-1,-1):
        for j in range(len(yList)):
            pointIn = eps*inSet(complex(xList[i],yList[j]))     # multiply N by power of eps
            julia[len(yList)-j-1,i] = pointIn

    # convert the matrix into a list
    points = stairs(julia)
    print(len(points))

    colours = ['r','orange','g','b','purple']
    ci = imageversion-1

    # plot the sorted list as index against element
    fig=plt.figure()
    ax=fig.add_subplot(111)
    ax.set_xlabel('Value')
    ax.set_ylabel('Index (1000s)',rotation=90)
    ax.set_xticks(np.arange(0,18*np.pi,step=np.pi))
    ax.set_xticklabels(['0','$\pi$','2$\pi$','3$\pi$','4$\pi$','5$\pi$','6$\pi$','7$\pi$','8$\pi$','9
    plt.plot(points,np.linspace(0,(len(points)-1)/1000,len(points)),color=colours[ci%5],linestyle='-')
    plt.legend()
plt.savefig('stairs{}.jpg'.format(eps),format='jpg',dpi=300)
```

Figure A.6: Plotting the εN list against its index.

COVID19 IMPACT SHEET
Project 3/4
Department of Mathematical Sciences

Student Name: Ethan Tribe

Year group (3/4): 3

Project Topic: Finding pi in unexpected places

Project supervisor(s): Prof. P. E. Dorey

Did Covid19 prevent you from completing part of your project report (Yes/No): No

If 'Yes', please indicate what it prevented you from doing (max 100 words):

Please summarise the action taken in response (max 100 words):