`CDynamics`: A Package to Plot Approximate Filled-In Julia Sets

Dr. Vikram Aithal[1] and Kanak Dhotre[1]

[1]Institute of Chemical Technology, Mumbai, India

March 4, 2023

Abstract

This paper presents the `CDynamics` program which allows users to explore numerous rudimentary concepts in the field of holomorphic dynamics and can be used for teaching as well as research purposes. The package is written fully in Python and is built using `SymPy`[Meu+17], `NumPy`[Har+20], and `MatPlotLib`[Hun07]. It is free, open-source and licensed under the terms of GNU GPL v3. The source-code, installation guide and usage examples are available over at https://github.com/kanakdhotre99/CDynamics. The star-feature of the package is the option to plot approximate filled-in Julia sets of arbitrary rational maps defined on the Riemann sphere. Other functionalities, however, include the options to compute fixed points, critical points, and forward orbits of the aforementioned maps.

1 Introduction

A significant part of complex dynamics is concerned with the study of the system defined by collecting n-fold iterates of a holomorphic self-map defined on a compact Riemann surface. The most natural technique to understand these dynamics is to identify a dichotomy on the Riemann surface which characterizes predictable versus chaotic dynamics. In modern literature[Mil90] these sets, namely, the normality set and it's complement, are referred to as the Fatou set and the Julia set respectively. Plots corresponding to filled-in Julia sets help us visualize the aforementioned dichotomy comprehensibly. One must take note of the fact that we can infer the behaviour of our dynamical system using these approximations because filled-in Julia sets of rational maps are indeed computable[BY08]. Given a holomorphic self-map f of $\hat{\mathbb{C}}$ which must necessarily be rational[Kum21] with a finite degree d, and a discrete grid $[a,b]\times[c,d]\in\mathbb{R}^2$ with $u\times v$ number of points it in, our goal is to generate an image of appropriate resolution in which the RGB-value of the (r,s)-th pixel corresponds to one, the number of iterations the orbit of $r+is$ took to either converge to some attracting fixed point of f or reach a user-dictated bail-out value, and two, which one of the $k\leq d+1$ attracting fixed points is closest to $f^{\circ n}(r+is)$. It follows that if we limit the choice of colors with which we are coloring the initialized image to two, and only base the coloring on the number of iterations taken, the boundary of the resultant binary image would respresent the Julia set of f. Over the past decade, multiple softwares and libraries have been developed[Jan21][Gar22][Inc] which let users viusalize Julia sets but, to the best of our knowledge, are either proprietary, are availiable only on Windows/MacOS, only generate plots for polynomial maps (as opposed to more general rational maps) or don't provide functionalities to tune relevant hyperparameters such as the depth of the orbit, bail-out value, color-map, etc. This motivated us to develop `CDynamics` which helps researchers, teachers and students to explore rudimentary concepts in the field of complex dynamics.

2 Algorithm

Computing fixed and critical points of f

Given a complex-valued function f, `CDynamics.Julia.fixed_points()` computes the fixed points of f symbolically using `SymPy`'s `solve()` function. Critical points of f are computed similarly.

Finding the orbit of a seed under f

Given a seed z, depth d and, a bailout value b, `CDynamics.Julia.orbit()` recursively computes and stores the images of z under n-folds of f while n is less than d, the penultimate fold hasn't exceed b and the orbit hasn't encountered a fixed point of f.

Plotting The Julia Set of f

Let $a_1,\cdots,a_k$ be the fixed points of f. `CDynamics.Julia.plot()`[1] first computes the distance between a_i and $f^{\circ n}(z)$ for each i and then returns the argument which minimizes the distance, say $\hat{a}$. If the computed distance isn't sufficiently small and f fixes ∞, it is safe to assume that the seed escapes to ∞ instead. Let $C=\{(cr_1,cg_1,cb_1),\cdots,(cr_k,cg_k,cb_k)\}$ be an array of unique RGB-tuples. For each z in a user-dictated discrte grid $[a,b]\times[c,d]$ `CDynamics.Julia.plot()` picks

[1]Refer to the subroutine named `find_nearest()`

a color from C based on which one of the k fixed points is closest to the penultimate value in it's orbit. Then, based on how many iterates z takes before it's orbit terminates, the lightness value of the RGB tuple is increased by multiplying it into a non-negative value g. This determines the visibility of the escape potential in the final plot. At first it might seem unnecessary to let the length of the orbit have an affect on the RGB-value of the corresponding pixel. However, the caveat here is that not all rational functions have attracting behaviour, i.e, it is possible that none of the fixed points of f are attracting. Lastly, if we pass this (possibly multi-colored) image through a binary color map, we get an image whose boundary will correspond to the Julia set of f.

3 Using `CDynamics`

3.1 Numerical Computations

`CDynamics` can be used as a handy tool for numerical computations that come up frequently in the study of complex dynamics. As an example, consider $f(z) = (4z^4 - 4z^6 + z^8)/(1 - 4z^2 + 2z^4)$

```
#pip install CDynamics
import CDynamics
z = CDynamics.sp.symbols('z')

#Define f(z)
f = CDynamics.Julia((4*z**4 - 4*z**6 + z**8)/(1-4*z**2+2*z**4))

#Finding the fixed points of f(z)
f.fixed_points

#Finding the critical points of f(z)=p/q(z)
f.critical_points

#Computing the orbit of a seed(=0.1) under f(z) with depth(=10) and bail_out(=1000)
f.orbit(0.1,10,1000)
```

3.2 Plotting Approximate Julia Sets

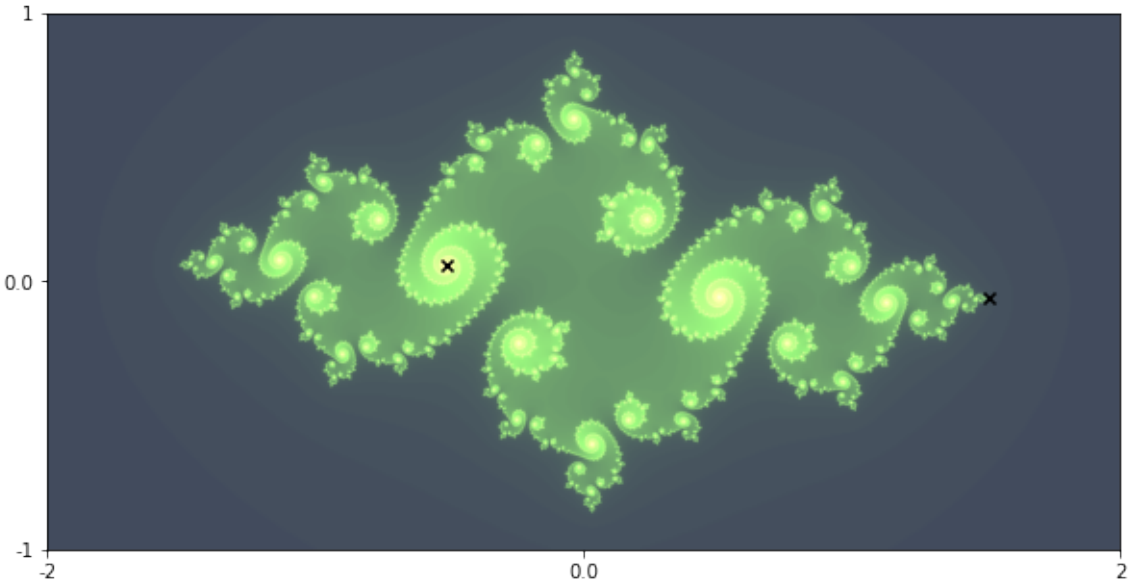
As discussed earlier, a key feature of our library is the option to plot Julia sets corresponding to arbitrary rational maps defined on the Riemann sphere. As an example, let's plot the Julia set admitted by $f(z) = z^2 - 0.765 + 0.12i$

```
#Initialize f(z)
f = CDynamics.Julia(z**2-0.765+(0.21*CDynamics.sp.I))

#Computing the fixed points of f(z)=p/q(z)
f.fixed_points
```

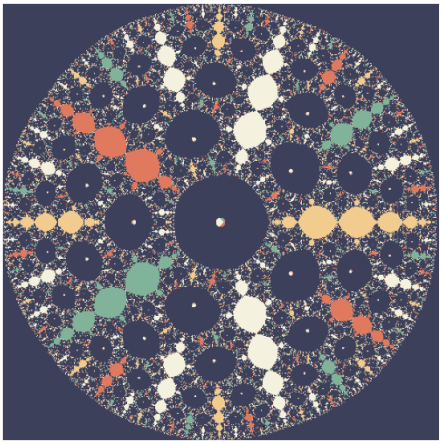
Since both the finite-valued fixed points of f are repelling, f does not have attracting behaviour. Hence, it is essential that we let the coloring of our grid depend continuously on the length of the orbit, we shall do so by setting the glow parameter to a sufficiently large value.

```
#Plotting the Julia set corresponding to f(z) computed on [-2,2]x[-1,1] with 1080^2 points and a
    depth of 100 iterations per point.
f.plot(-2,2,-1,1,resx=1080,resy=1080,depth=100,glow=5)
```

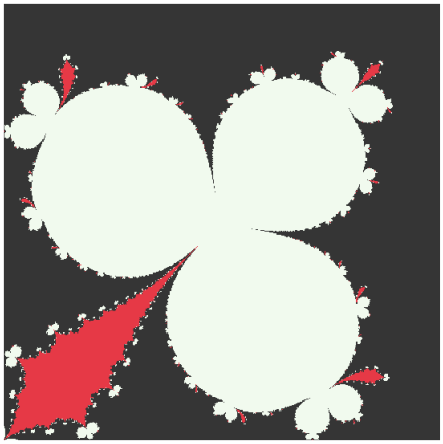
Notice how the overlayed `xs` mark the fixed-points of f. One can chose not to plot these by setting the `fixed_points` to `False`. Users can also change the default color-palette by passing a list of sufficient (greater than number of fixed-points) number of RGB-tuples to the `cols` parameter.



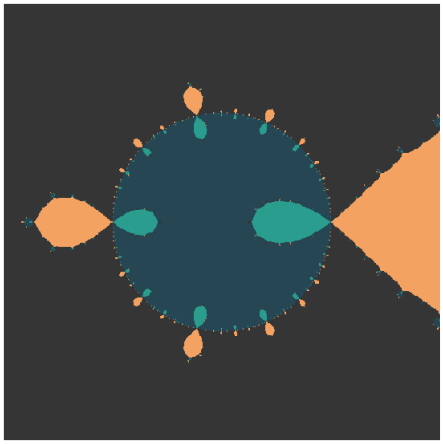
(a) $f(z) = z^2 - 0.765 + 0.12i$



(a) $f(z) = (0.01 + z^5)/(z^3)$

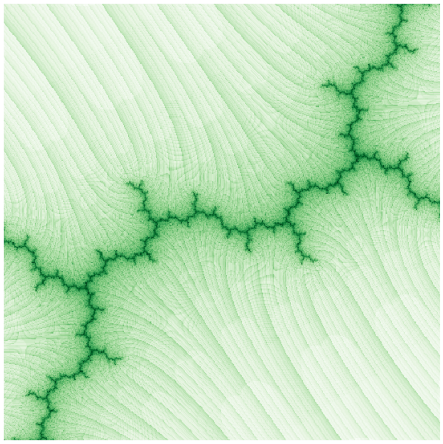


(b) $f(z) = z^5 + (0.8 + 0.8i)z^4 + z$

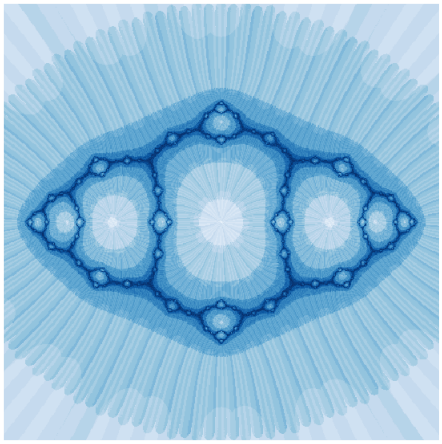


(c) $f(z) = (4z^3 - z^4)/(4z - 1)$

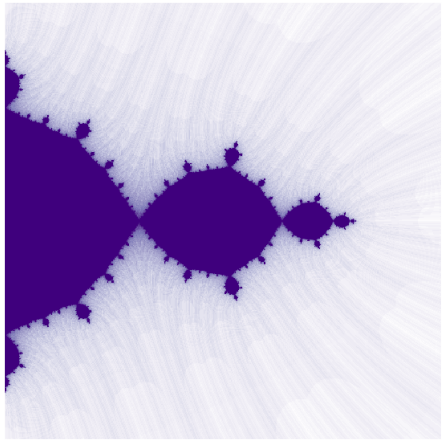
Figure 1: A Few More Examples Of Filled-In Julia Sets



(a) $f(z) = z^2 + i$



(b) $f(z) = 1/(z^2 - 1)$



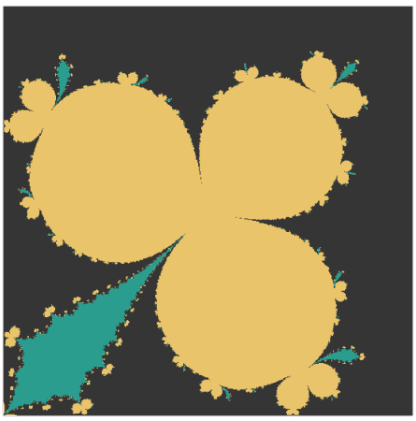
(c) $f(z) = z^2 - 1$

Figure 2: Examples Of Julia Sets Plotted By Setting `filled=False,cmap='Greens'`

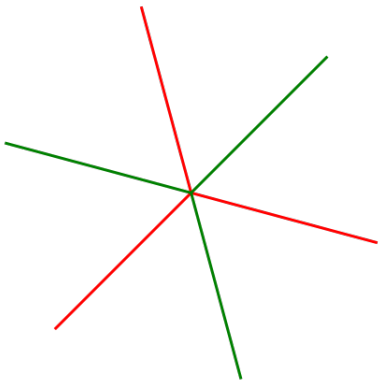
3.3 Visualizing The Leau-Fatou Flower

In the case where a parabolic fixed point admits a basin of attraction, the dynamics in question are extremely slow. The Leau-Fatou flower theorem[Mil90] grants a method to construct "petals" using attraction and repulsion vectors which can trap these dynamics. For demonstration purposes we will restrict ourselves to the case where the rational map $f = p/q$ can be expressed locally as $f(z) = z + az^{n+1} + O(z^{n+2})$. With f as assumed and $n \geq 2$, $\mathbf{v} \in \mathbb{C}$ is called a repulsion vector for f at the origin if $na\mathbf{v}^{n-1} = +1$ and an attraction vector if $na\mathbf{v}^{n-1} = -1$. It follows that there are n equally spaced attraction vectors at 0, seperated by n equally spaced repulsion vectors, say $\mathbf{v}_0$ (which is necessarily repelling), $\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_{2n-1}$ where $\mathbf{v}_j = \exp \pi ij/n\mathbf{v}_0$ and $na\mathbf{v}_j^n = (-1)^j$. Let's plot these vectors over the filled-in Julia plot of $f(z) = z^5 + (0.8 + 0.8i)z^4 + z$.

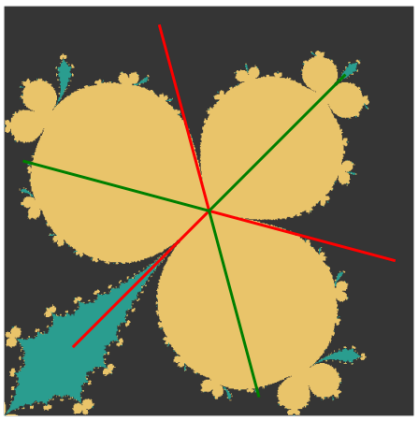
```
import numpy as np
import matplotlib.pyplot as plt

#Defining n, a and f as above
n = 3
a = complex(0.8,0.8)
f = CDynamics.Julia(z**5+(0.8+0.8*CDynamics.sp.I)*z**4+z)

#Plotting the Julia set
f.plot(depth=100)

#Plotting the attraction and repulsion vectors
def v(j):
    if j==0:
        return((1/(n*a))**(1/n))
    else:
        return(np.exp((np.pi*complex(0,1)*j)/n)*v(0))

rep,att = [v(i) for i in range(0,len(p),2)], [v(i) for i in range(1,len(p),2)]

[plt.plot([0,rep[i].real],[0,rep[i].imag],color='red',linewidth=3) for i in range(len(rep))]
[plt.plot([0,att[i].real],[0,att[i].imag],color='green',linewidth=3) for i in range(len(att))]
```



(a) $J(f(z) = z^5 + (0.8 + 0.8i)z^4 + z)$



(b) Corresponding v_i's



(c) v_i's overlapped on $J(f)$

Notice how the immediate basins for the three attraction vectors (colored in green) resemble balloons pulled together at the origin and seperated by the three repulsion vectors (colored in red).

3.4 Studying Singular Perturbations Of Polynomials

Consider a polynomial $P : \hat{\mathbb{C}} \circlearrowleft$. We are now interested in the dynamics of maps defined by perturbing $P(z)$ by the inclusion of a pole. Namely, maps of the form

$$F_\lambda(z) = P(z) + \frac{\lambda}{(z-a)^d}$$

where $\lambda, a \in \mathbb{C}$. For the sake of our discussion, we will restrict our discussion to the case where $P(z) = z^n, a = 0, d = n$. Notice how the $2n$ critical points of F_λ are now given by $z = \lambda^{1/2n}$, however the critical values v_λ, i.e the images of the critical points, are only two, namely $\pm 2\sqrt{\lambda}$. Further, we let B_λ be the immediate basin of attraction at ∞ and let T_λ be the open set containing 0 which is mapped to B_λ under F_λ. In [Dev13] Robert L. Devaney states the following trichotomy which we shall verify for cases $n = 2, 3$ and 4.

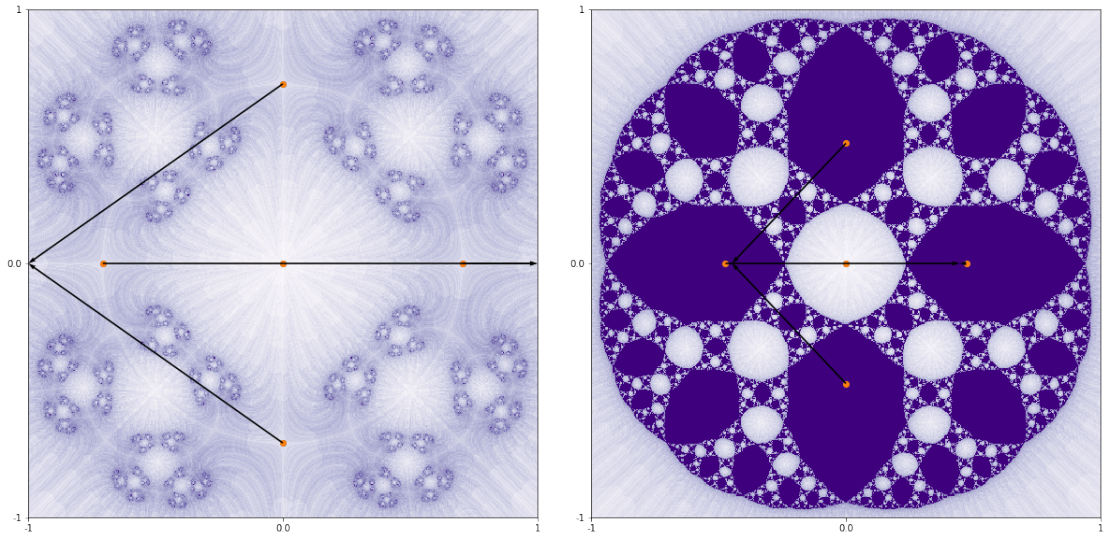
Theorem 3.1. (Escape Trichotomy): If the orbits of the free critical points (excluding 0 and ∞) tend to ∞ then,

1. If v_λ lies in B_λ (if one does, the other must due to the $z \to -z$ symmetry), $J(F_\lambda)$ is a Cantor set.
2. If v_λ lies in T_λ, then $J(F_\lambda)$ is a Cantor set of concentric simple closed curves, each one of which surrounds the origin.

3. In all other cases, $J(F_\lambda)$ is a connected set. More precisely, if $F_\lambda^k(v\lambda) \in T_\lambda$ where $k \geq 1$, then $J(F_\lambda)$ is a Sierpiński curve.

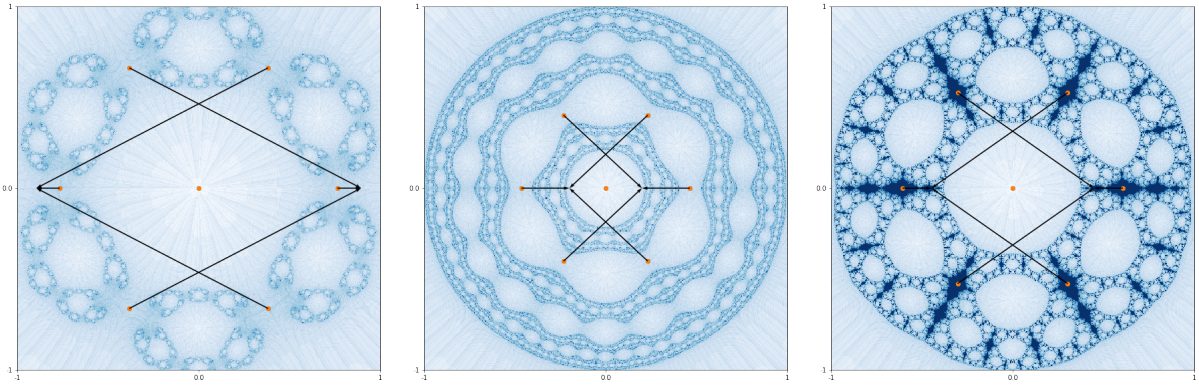
To visualize how the critical orbits determine the structure of the Julia set in accordance with the above stated result, let's overlay these orbits on the plot of the corresponding Julia set by setting the `critical_orbit_depth` to 2.

```
#Defining f(z) as above for n=2
f1 = CDynamics.Julia(z**2 + (0.25/z**2))
f2 = CDynamics.Julia(z**2 + (0.05/z**2))

#Plotting the Julia set
f1.plot(filled=False,cmap='Purples')
f2.plot(filled=False,cmap='Purples')
```

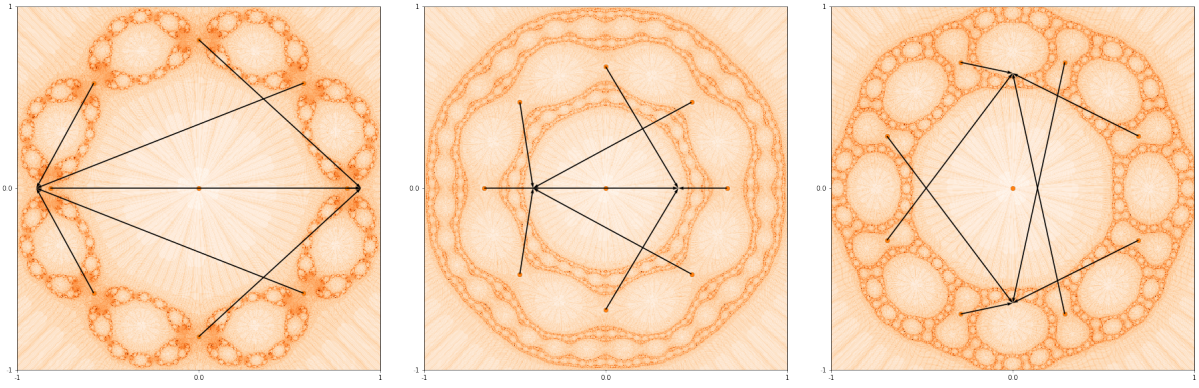


(a) $\lambda = 0.25, v_\lambda \in B_\lambda$ and $J(F_\lambda)$ is a Cantor set (b) $\lambda = 0.05, v_\lambda \notin B_\lambda, T_\lambda$ and $J(F_\lambda)$ is a Sierpiński curve

Figure 3: $F_\lambda(z) = z^2 + \lambda/z^2$



(a) $n = 2, \lambda = 0.2, v_\lambda \in B_\lambda$ (b) $n = 2, \lambda = 0.01, v_\lambda \in T_\lambda$ (c) $n = 2, \lambda = 0.05, v_\lambda \notin B_\lambda, T_\lambda$



(d) $n = 3, \lambda = 0.2, v_\lambda \in B_\lambda$ (e) $n = 3, \lambda = 0.04, v_\lambda \in T_\lambda$ (f) $n = 3, \lambda = -0.1, v_\lambda \notin B_\lambda, T_\lambda$

Figure 4: $F_\lambda(z) = z^n + \lambda/z^n$ for $n = 3, 4$

3.5 Parameter Spaces

We've seen how `CDynamics` can be used identify ideas for studying the behaviour of an iterated function system in the dynamical plane. In the spirit of exploring how it can also be used to identify characterizations in the parameter plane as well, consider the family $Q_c = \{z^2 + c : c \in \mathbb{C}\}$. Notice how the critical point of each map in Q_c is 0. This motivates use to define a natural bifurcation based on the whether or not the critical orbits tends to ∞. Let's plot $\mathcal{M} = \{c \in \mathbb{C} : Q_c^n(0) \not\to \infty\}$.

```
#Varying c in [-2,0.6]x[-1.3,1.3]
cvals = CDynamics.Julia.grid(-2,0.6,-1.3,1.3,1080,1080)

#Collecting the length of orbits for varying maps in Qc
mandelbrot = []
for c in cvals:
    mandelbrot.append(len(Julia(z**2 + c).orbit(0,1000,1000)))

#Plotting the Mandelbrot set
mandelbrot = np.asarray(mandelbrot).reshape(1080,1080)
plt.imshow(mandelbrot,cmap='Reds')
```

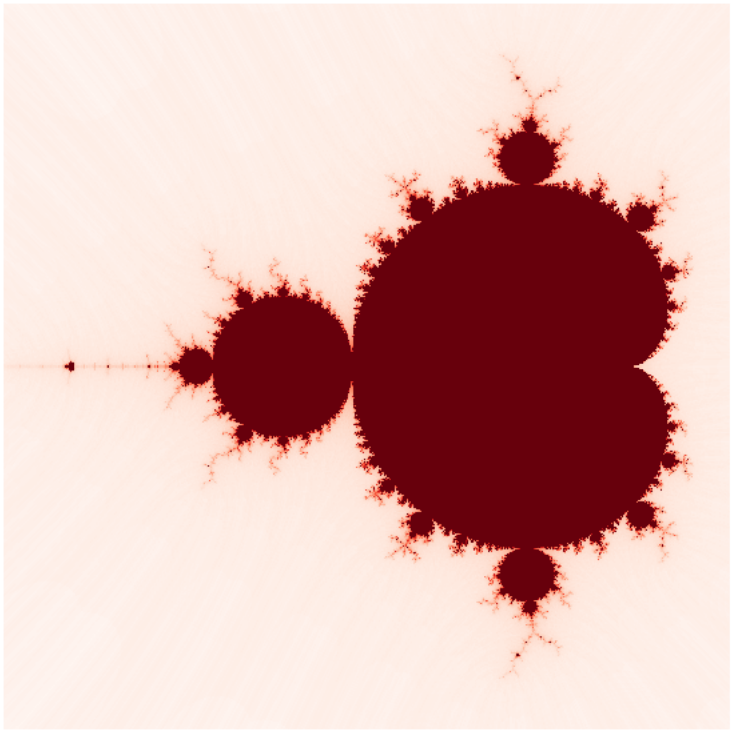


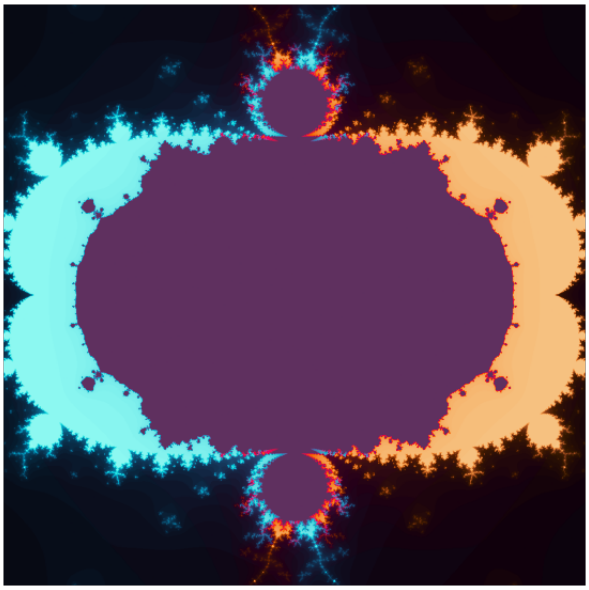
Figure 5: A bifurcation of the c-plane famously known as the Mandelbrot set

Since the parameter space in the above example was 2-dimensional, it was possible for us to plot it on a single plane. However, this might not always be the case. Consider cubic functions of the form $f(z) = z^3 - 3a^2z + b$. The parameter space is now four-dimensional and depends continuously on the two complex parameters a and b.For the sake of visualizing a "slice" of the parameter space we can fix one parameter and vary the other.

```
#Fixing a = 0.5 and varying b over [-1,1]x[-1,1]
a = 0.5
bvals = CDynamics.Julia.grid(-1,1,-1,1,1080,1080)
```

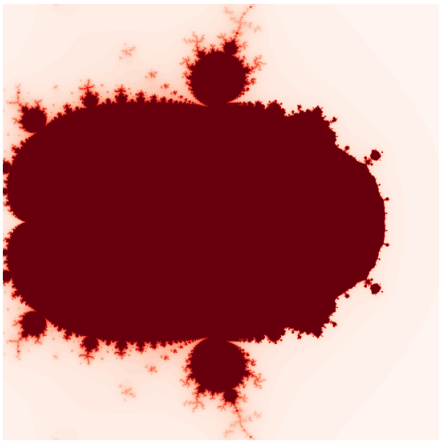
Since critical points determine the structure of the Julia set. For each value of b, we compute the orbits of $\pm a$.

```
#Computing the orbit of +a at each map
critical1 = []
for b in bvals:
    critical1.append(len(Julia(z**3-3*a**2*z+b).orbit(a,25,1000)))

#Computing the orbit of -a at each map
critical2 = []
for b in bvals:
    critical1.append(len(Julia(z**3-3*a**2*z+b).orbit(-a,25,1000)))

#Plotting the slices
plt.imshow(critical1,cmap='Blues')
plt.imshow(critical2,cmap='Reds')
```



(a) A slice of the parameter space in $\mathbb{C}^2$

(b) Coloring based only on the orbit of $+a$



(c) Coloring based only on the orbit of $-a$

References

[Mil90] John W Milnor. "Dynamics in one complex variable: Introductory lectures". In: *arXiv preprint math/9201272* (1990).

[Hun07] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[BY08] Mark Braverman and Michael Yampolsky. *Computability of Julia sets*. Springer, 2008.

[Dev13] Robert Devaney. "Singular perturbations of complex polynomials". In: *Bulletin (New Series) of the American Mathematical Society* 50 (July 2013). DOI: 10.1090/S0273-0979-2013-01410-1.

[Meu+17] Aaron Meurer et al. "SymPy: symbolic computing in Python". In: *PeerJ Computer Science* 3 (Jan. 2017), e103. ISSN: 2376-5992. DOI: 10.7717/peerj-cs.103. URL: https://doi.org/10.7717/peerj-cs.103.

[Har+20] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[Jan21] Thomas Marsh Jan Hubička. *XaoS*. 2021. URL: https://github.com/xaos-project/XaoS.

[Kum21] S Kumaresan. *A Pathway to Complex Analysis*. Techno World, 2021.

[Gar22] Virginia Sterling McCabe Garth Thornton. *Jux*. 2022. URL: https://www.xenodream.com/jux.htm.

[Inc] Wolfram Research Inc. *Mathematica, Version 13.0.0*. Champaign, IL, 2021. URL: https://www.wolfram.com/mathematica.