task1

April 9, 2020

Уравнение для анализа

[6]:
```
display(y)
```

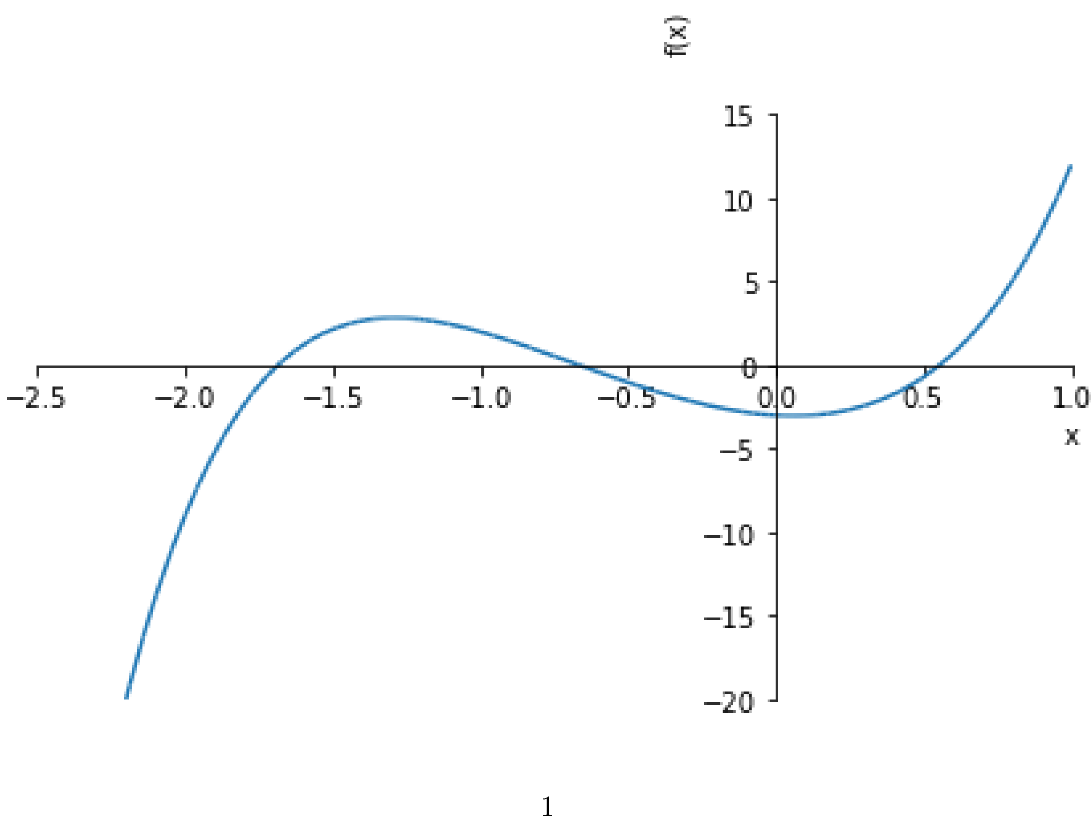
$x^5 + 2x^4 + 5x^3 + 8x^2 - x - 3$

Для локализации мы смотрим промежутки возрастания и убывания (найденные с помощью производной) и проверяем на каждом наличие корня. Для этого достаточно того, чтобы либо значение функции в левой границе было нулём, либо значения были разных знаков. Мы знаем, что будет единственный корень на этом промежутке, так как он монотонный. Тогда корень можно найти методом Ньютона.

В качестве бесконечности взяли 10^5. Так как $y(10^5) > 0$, корни производной примерно $-1,29$ и $0,06$, и $y'(10^5) > 0$. Наша функция непрерывна, поэтому после "бесконечности" корней точно не будет. Аналогично и отрицательная бесконечность -10^5. $y(-10^5) < 0$ и $y'(-10^5) > 0$.

График уравнения

[8]:
```
display(plot(y, (x, -2.5, 1.0), ylim=[-20, 20]))
```



```
<sympy.plotting.plot.Plot at 0xd195cb0>
```

Корни уравнения

[9]:
```
display(roots)
```

```
[-1.68605466239665, -0.666021897044982, 0.554554429143766]
```

Локализация корней

[10]:
```
display([{'l': seg[0], 'r': seg[1]} for seg in sol[2]])
```

```
[{'l': -100000.0, 'r': -1.29022829728533},
 {'l': -1.29022829728533, 'r': 0.0591165490773438},
 {'l': 0.0591165490773438, 'r': 100000.0}]
```

Полученные корни методом Ньютона

[11]:
```
display(sol[0])
```

```
[-1.68605466250080, -0.666021902922307, 0.554554429497462]
```

Максимальная относительньяа погрешность

[25]:
```
display(max([abs((sol[0][i] - roots[i]) / roots[i]) for i in range(len(roots))]))
```

$8.82452228244335 \cdot 10^{-9}$

[ ]:
```

```

[1]:
```
from sympy import Symbol
from sympy.solvers import solveset
from sympy.plotting import plot
```

[2]:
```
x = Symbol('x')
a, b, c, d, e, f = 1, 2, 5, 8, -1, -3
y = a * x**5 + b * x**4 + c * x**3 + d * x**2 + e * x + f
roots = sorted(list(map(lambda i: i.evalf(), filter(lambda i: i.is_real,␣
 ↪solveset(y, x)))))
```

[23]:
```
def newton_method(y, l, r, eps=10**(-4)):
    m = (l + r) / 2.0
    m_num = y.subs(x, m)
    cnt = 0
```

```
    while (abs(m_num) > eps):
        m -= m_num / y.diff().subs(x, m)
        m_num = y.subs(x, m)
        cnt += 1
    return (m, cnt)
```

[14]:
```
def find_solutions(y, p, inf=1e5):
    if p <= 2:
        return (sorted(list(map(lambda i: i.evalf(), filter(lambda i: i.is_real,␣
↪solveset(y, x))))), 0, [])
    sign = lambda x: 1 if x >= 0 else -1
    fs = find_solutions(y.diff(), p - 1)
    xs = [-inf] + fs[0] + [inf]
    segments, roots, cnt = [], [], fs[1]
    for i in range(len(xs) - 1):
        l, r = xs[i:i+2]
        yl, yr = y.subs(x, l), y.subs(x, r)
        if yr != 0 and (yl == 0 or sign(yl) != sign(yr)):
            ans = newton_method(y, l, r)
            segments += [(l, r)]
            roots += [ans[0]]
            cnt += ans[1]
    return (roots, cnt, segments)
```

[24]:
```
sol = find_solutions(y, 5)
```

task2

April 5, 2020

```
x = r * x * (1 - x)
phi(x) = r * x * (1 - x)
phi'(x) = r - 2rx
```

для сходимости к корню необходимо, чтобы первое выбранное нами значение лежало в такой окрестности корня, что:
`|phi'(x)| <= 1`

Отсюда следует, что:
`(r - 1) / (2 * r) < x < (r + 1) / (2 * r)`

[48]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker
import math
import random
```

[29]:
```
def phi(x, r):
    return r * x * (1 - x)

def get_window(r):
    return (r - 1) / (2 * r), (r + 1) / (2 * r)

def is_wave(x, x_0, x_1):
    return x_0 < x < x_1 or x_0 > x > x_1
```

[132]:
```
def simple_iterations(epsilon, r, iter=1000, draw=False):
    (left_border, right_border) = get_window(r)
    iterations = 0

    cur_x = random.uniform(left_border, right_border)
    next_x = phi(cur_x, r)

    wave_cur = cur_x
    wave_next = next_x

    x = [next_x]
    wave = False

```

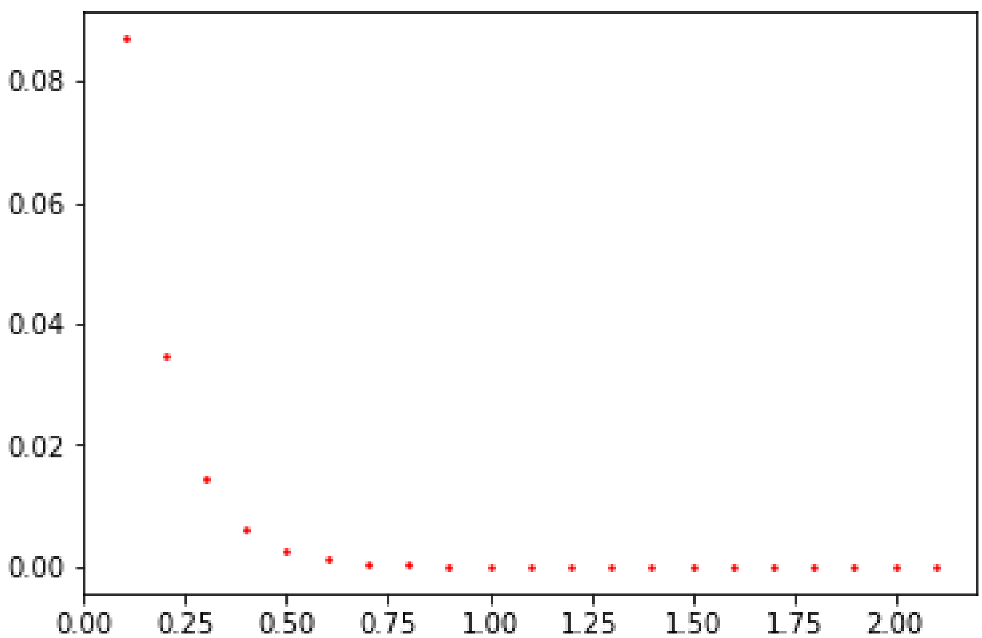
```
    while (abs(cur_x - next_x) > epsilon):
        cur_x = next_x
        next_x = phi(cur_x, r)
        x.append(next_x)
        iterations += 1
        if (iterations > iter):
            return (None, iter, wave)

    for i in range(1, len(x)):
        wave = is_wave(next_x, x[i - 1], x[i])
        if wave:
            break

    if draw:
        plt.plot([float(i) / 10 for i in range(1, len(x) + 1)], x, 'r.', ms=3)
        plt.show()

    return (next_x, iterations, wave)
```

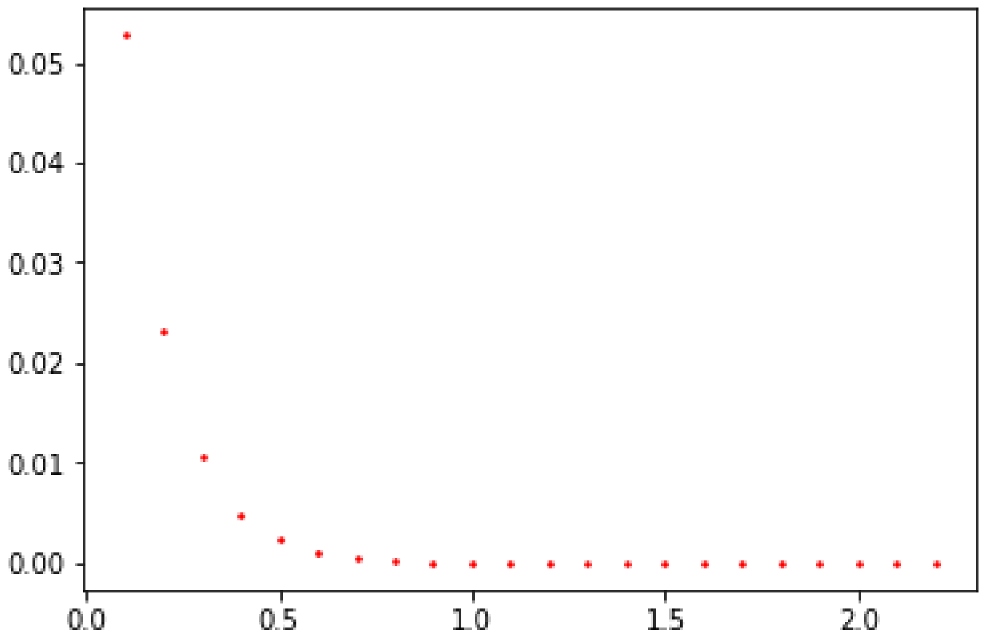
[133]:
```
def check_conv(left, right, iter, eps, check_eps, draw=False):
    roots = [[0, 0], [0, 0], [0, []]]
    for i in range(0, iter):
        cur_r = random.uniform(left, right)
        (answer, iters, wave) = simple_iterations(eps, cur_r, draw=draw)
        x1 = 0.0
        x2 = 1 - (1 / cur_r)
        if (answer is not None):
            if (abs(answer - x1) < abs(answer - x2) and abs(answer - x1) <␣
 ↪check_eps):
                roots[0][wave] += 1
            elif (abs(answer - x2) < check_eps):
                roots[1][wave] += 1
            else:
                roots[2][0] += 1
                roots[2][1].append(cur_r)
        else:
            roots[2][0] += 1
            roots[2][1].append(cur_r)

    if not draw:
        print(f"{left} to {right} with eps = {eps} and iter = {iter}:\n")
        print("root 0.0, not wave =          " + str(roots[0][0]))
        print("root 0.0, wave =              " + str(roots[0][1]))
        print("root 1 - (1 / r), not wave = " + str(roots[1][0]))
        print("root 1 - (1 / r), wave =     " + str(roots[1][1]))
        print("not conv =                    " + str(roots[2][0]))
        print("not conv points =             " + str(roots[2][1]))
```

[134]:
```
check_conv(0.01, 0.99, 10**4, 1e-8, 1e-6)
```

```
0.01 to 0.99 with eps = 1e-08 and iter = 10000:

root 0.0, not wave =         10000
root 0.0, wave =             0
root 1 - (1 / r), not wave = 0
root 1 - (1 / r), wave =     0
not conv =                   0
not conv points =            []
```

[129]:
```
check_conv(0.01, 0.99, 2, 1e-8, 1e-6, draw=True)
```





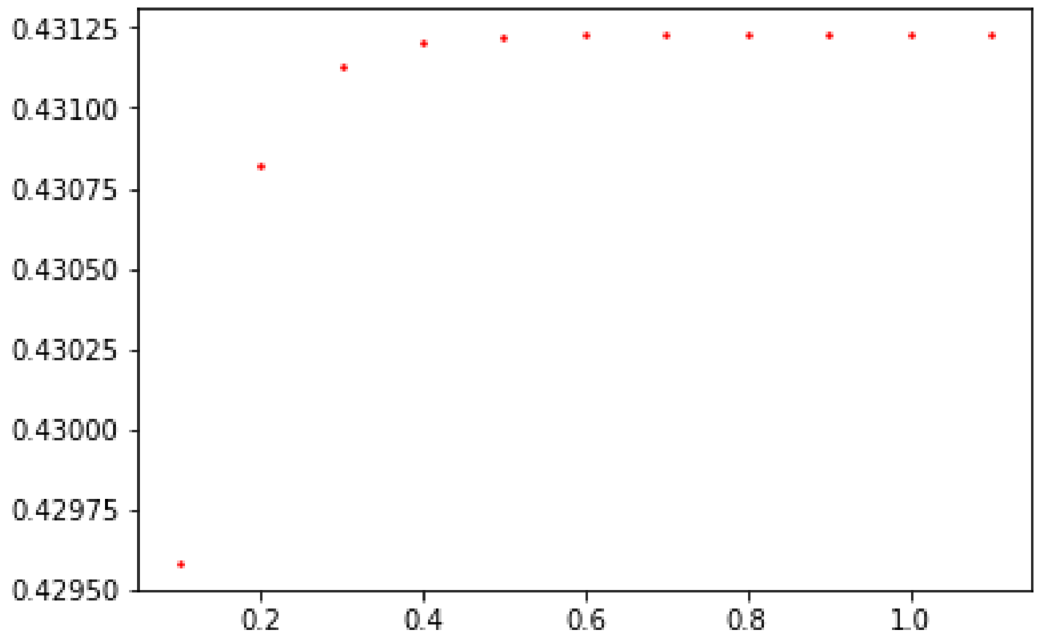
Как видно из приведенных нами вычислений, при выборе `r` из промежутка (0, 1) данная нам функция `phi` сходится монотонно к корню x0 = 0

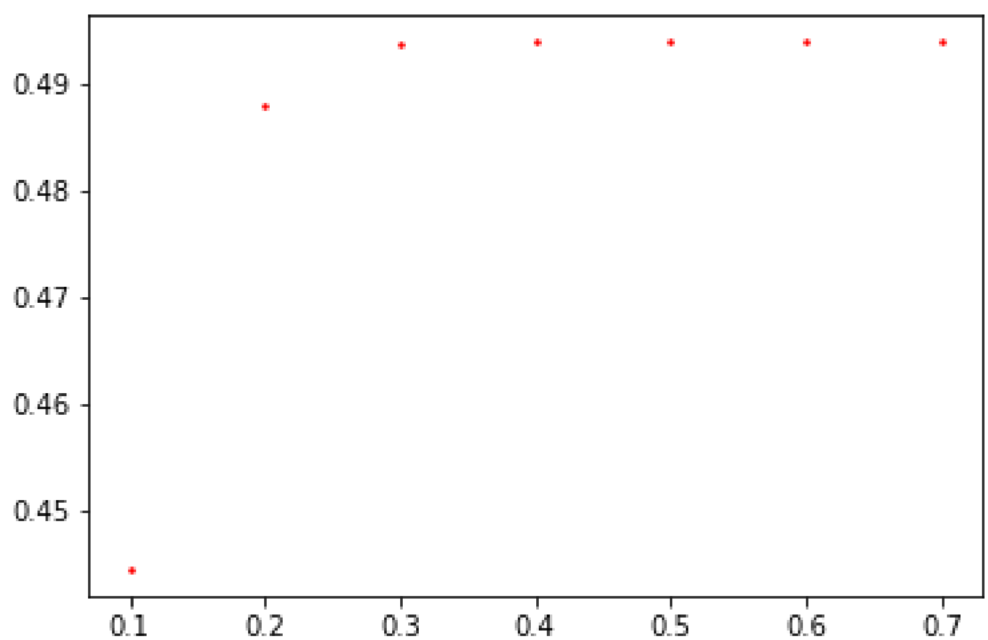
[135]:
```
check_conv(1.01, 2.99, 10**4, 1e-8, 1e-6)
print('\n' * 3)
check_conv(1.01, 1.99, 10**4, 1e-8, 1e-6)
print('\n' * 3)
check_conv(2.01, 2.99, 10**4, 1e-8, 1e-6)
```

```
1.01 to 2.99 with eps = 1e-08 and iter = 10000:

root 0.0, not wave =         0
root 0.0, wave =             0
root 1 - (1 / r), not wave = 5098
root 1 - (1 / r), wave =     4872
not conv =                   30
not conv points =            [2.9858038669655578, 2.988689492104717,
2.987032161988262, 2.9889769114316906, 2.984842810451674, 2.988109999205744,
2.985546500007847, 2.985134288337461, 2.989347870039911, 2.9864070727424803,
2.9895908534333895, 2.98996697403847, 2.9889086605972253, 2.987603358911911,
2.9848534335154273, 2.98670274574726, 2.986520256690107, 2.987471538222497,
2.9898378514452006, 2.989719814761925, 2.9861084744984407, 2.9887080621797004,
2.987268533044896, 2.984467117990767, 2.988331238131094, 2.9877642520824494,
2.9847280878253715, 2.9849018199709456, 2.9886294576590284, 2.988028072998878]
```

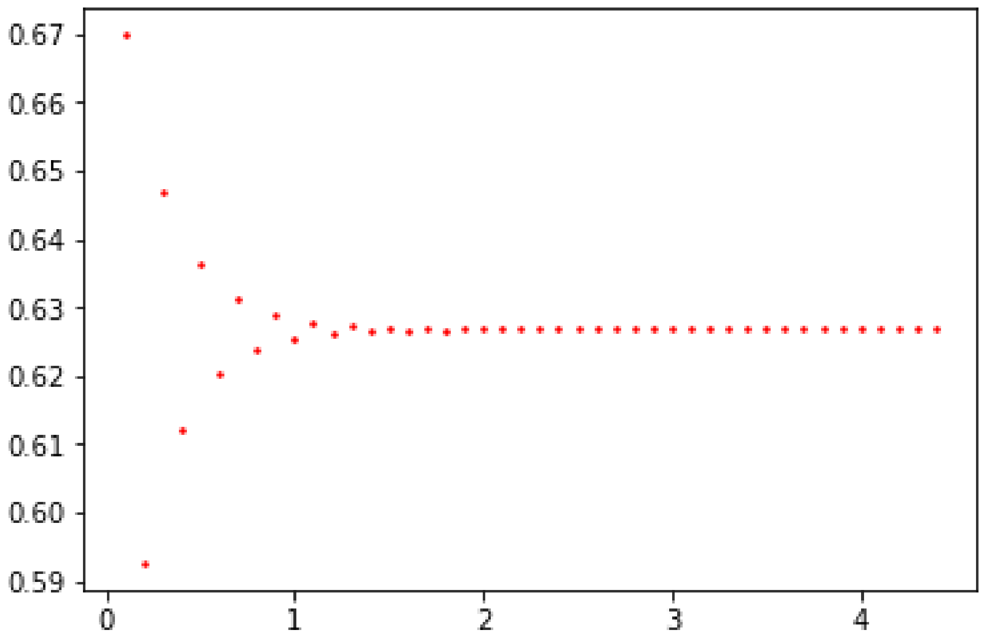
```
1.01 to 1.99 with eps = 1e-08 and iter = 10000:

root 0.0, not wave =         0
root 0.0, wave =             0
root 1 - (1 / r), not wave = 10000
root 1 - (1 / r), wave =     0
not conv =                   0
not conv points =            []




2.01 to 2.99 with eps = 1e-08 and iter = 10000:

root 0.0, not wave =         0
root 0.0, wave =             0
root 1 - (1 / r), not wave = 0
root 1 - (1 / r), wave =     9944
not conv =                   56
not conv points =            [2.9875714871108254, 2.9845666405340787,
2.98489289615924, 2.9853595508393393, 2.9875785512259014, 2.989862720093173,
2.9889073386554017, 2.988515458111989, 2.98870190686446, 2.988570917631945,
2.9865224388667047, 2.9845866057455313, 2.988979871734747, 2.9864870980627356,
2.988747315846073, 2.9862920734501923, 2.988128077941778, 2.988666542306466,
2.9889969425398, 2.9897234956485224, 2.9861955530427498, 2.9896210565818766,
2.989652342130404, 2.987720675850038, 2.9898028493627495, 2.987526064469785,
2.9863854013063422, 2.9892356208718702, 2.985475908220258, 2.98809220565674,
2.9860433701777627, 2.9882228753436806, 2.989406746686404, 2.985454562749208,
2.9867878871654923, 2.9882010881085495, 2.987343426040013, 2.984987218527074,
2.9868177872056627, 2.987584187096588, 2.988355062735282, 2.986780541156848,
2.9863228507897492, 2.989846969939312, 2.9884571574330883, 2.984972554317339,
2.988637886809634, 2.9865990010773116, 2.9856890805790792, 2.988476085286938,
2.9868496395560555, 2.987476282601809, 2.98708044804947, 2.985716936926943,
2.9854467592530893, 2.9867793311069795]
```

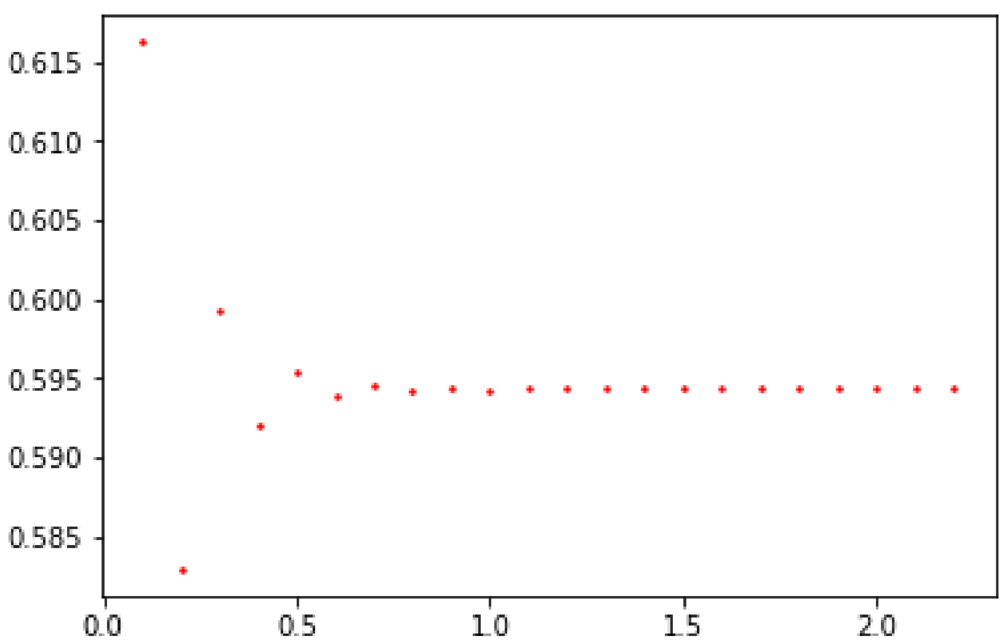
[131]:
```python
check_conv(1.01, 1.99, 2, 1e-8, 1e-6, draw=True)
```





[118]:
```
check_conv(2.01, 2.99, 2, 1e-8, 1e-6, draw=True)
```



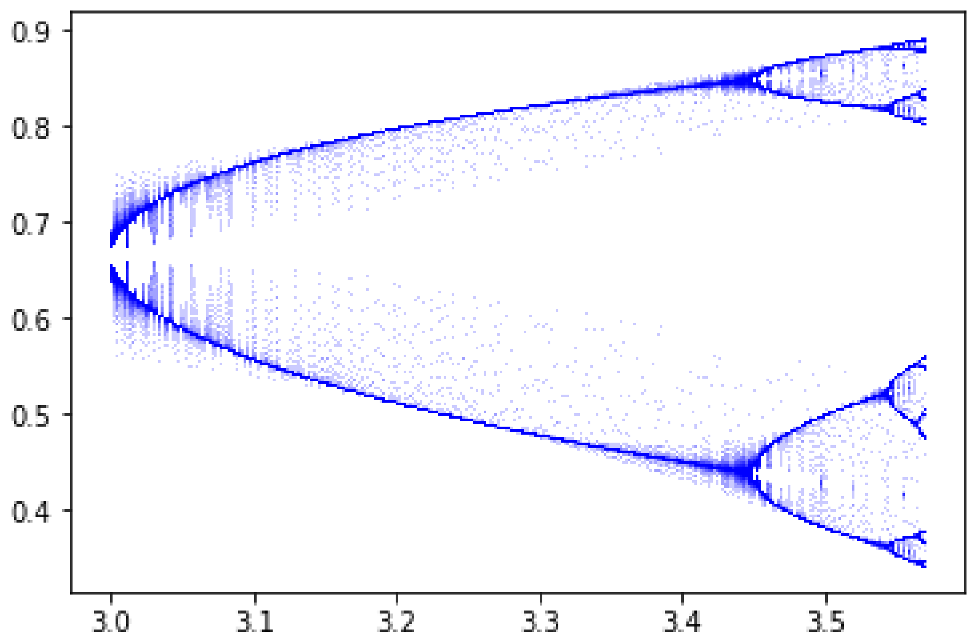


Как видно из приведенных нами вычислений, при выборе `r` из промежутка (1, 3) данная нам функция `phi` сходится к корню x1 = 1 - (1 / r)

При этом если выбирать `r` из промежутка (1, 2), то функция `phi` сходится к корню x1 монотонно, а при выборе `r` из промежутка (2, 3) — колебательно

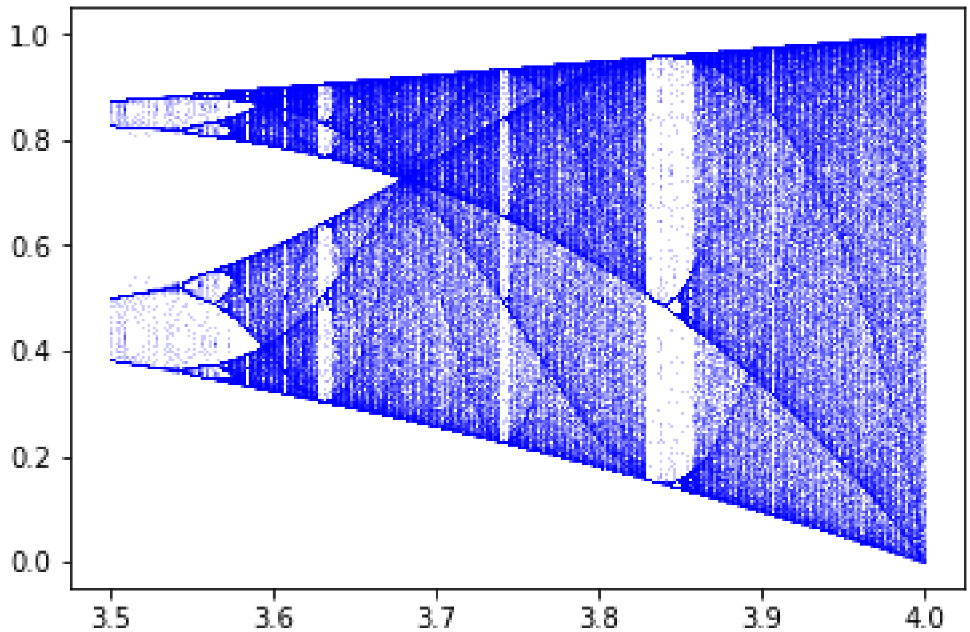
[136]:
```
def show_graphic(r, rang, eps):
    xVals = []
    yVals = []

    for j in range(rang):
        (lBorder, rBorder) = get_window(r)
        x = random.uniform(lBorder, rBorder)
        for i in range(500):
            x = phi(x, r)
            xVals.append(x)
            yVals.append(r)
        r += eps

    plt.plot(yVals, xVals, 'b+', ms=0.1)
    plt.show()
```

[137]:
```
def show_bifurcation():
    show_graphic(3.0, 570, 0.001)

show_bifurcation()
```



В теории график должен распасться в диапазонах 3-3.35..., 3.35...-3.52..., 3.52...-3.56... и т.д. Мы можем заметить, что на нашем графике это выполняется. Мы получили каскад бифуркаций удвоения периода.

[138]:
```
def show_chaos():
    show_graphic(3.50, 500, 0.001)

show_chaos()
```



По теории, между числами R(inf) и 4, поведение последовательности должно представлять из себя детерменированный хаос. График ведет себя в соответствии с теорией. На нем присутствуют зоны таких R, при которых наблюдаются сгущения и разрежения итерационной последовательности. По теории в окрестности R = 4 должен наблюдаться белый шум. Наш график подходит под теорию и в этом случае.

[ ]:
```
```

task3

April 5, 2020

1 Исследовать поведение итерационной последовательности при решении уравнения в комплексной плоскости методом Ньютона.

$$z^3 - 1 = 0$$

$$z_{n+1} = z_n - \frac{f(z)}{f'(z)}$$

$$f(z) = z^3 - 1$$

$$f'(z) = 3z^2$$

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
STEPS = 100
EPS = 1e-10

def f(z):
    return z**3 - 1

def f_der(z):
    return 3*z**2

def step(z):
    return z - f(z)/f_der(z)

def find_root_vect(initial):
    with np.errstate(divide='ignore', invalid='ignore'):
        z = initial
        for _ in range(STEPS):
            z = step(z)
        return z
```

[3]:

```
ROOTS = [1, np.exp(2j * np.pi / 3), np.exp(-2j * np.pi / 3)]

def get_color_by_root(root):
    for i, true_root in enumerate(ROOTS):
            if np.abs(true_root - root) < EPS:
                return i
    return 3

get_colors_by_roots = np.vectorize(get_color_by_root)
```

[4]:

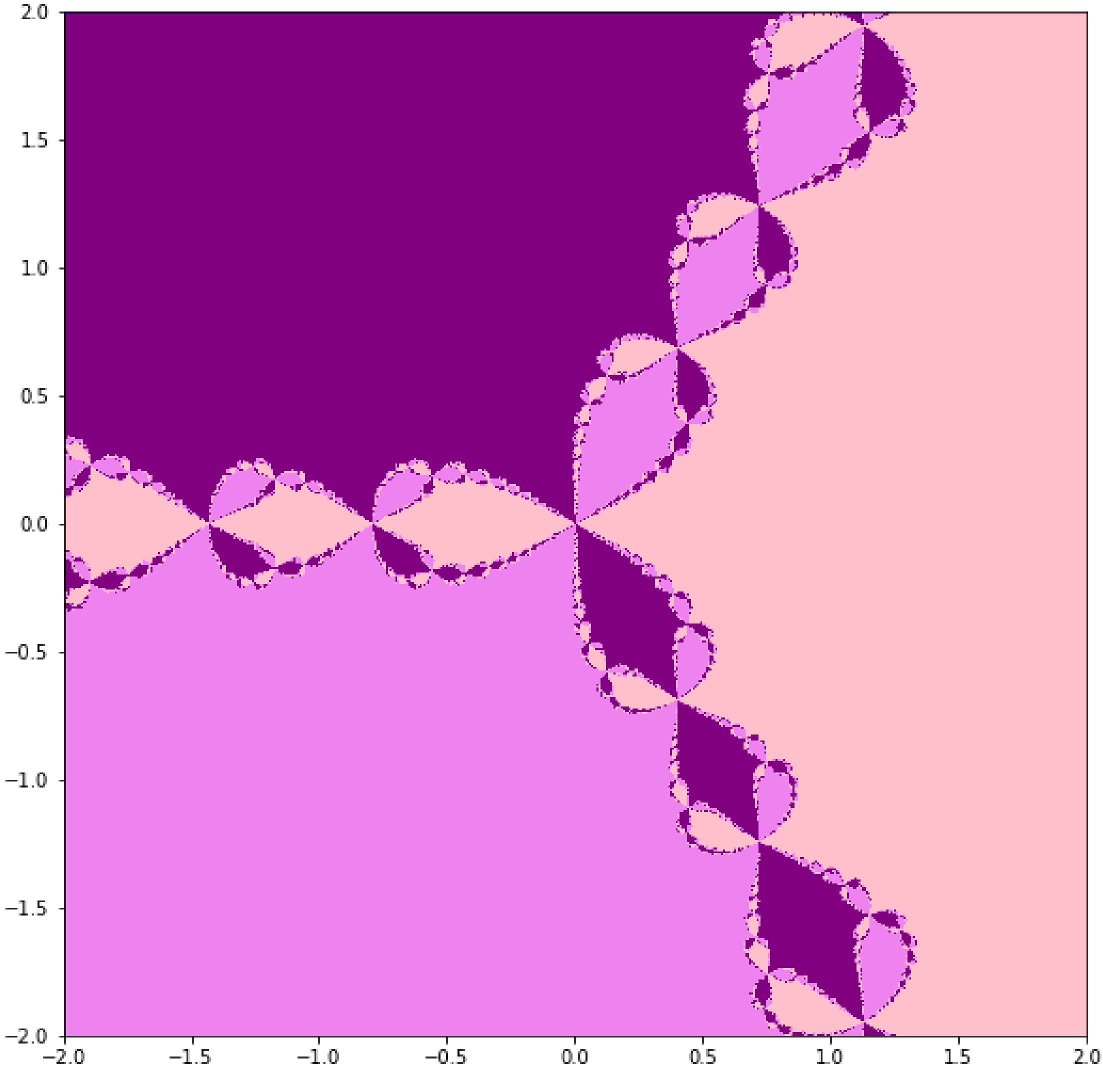
```
from multiprocessing import Pool

def get_colors(initials):
    with Pool() as pool:
        roots = pool.map(find_root_vect, initials)
        return pool.map(get_colors_by_roots, roots)
```

[65]:

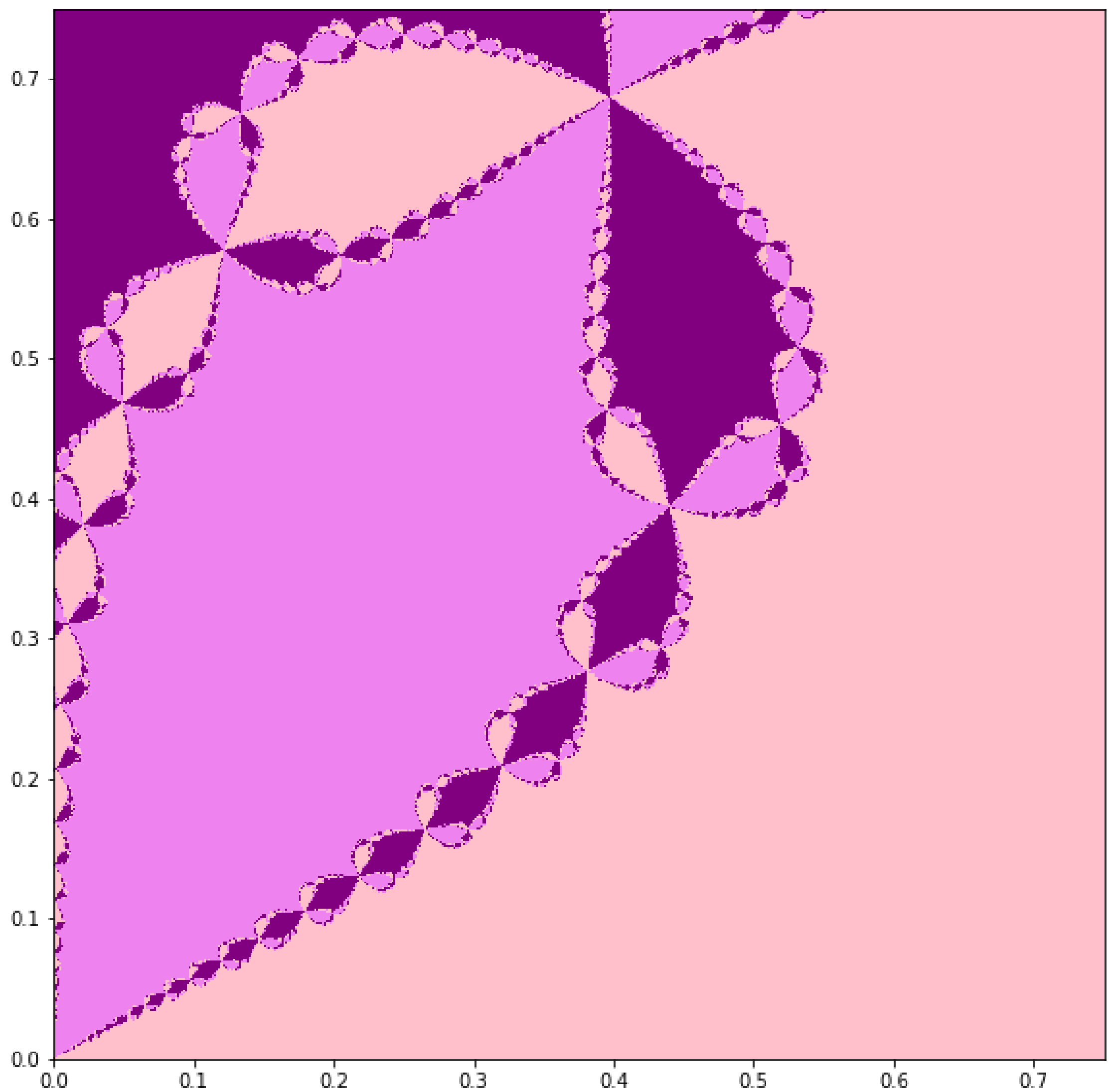
```
from matplotlib.colors import ListedColormap, Normalize

def draw(x0, y0, x1, y1, resolution):
    x = np.linspace(x0, x1, resolution)
    y = np.linspace(-y1, -y0, resolution)
    xv, yv = np.meshgrid(x, y)
    res = get_colors(xv + 1j*yv)
    color_list = ['pink', 'violet', 'purple', 'red']
    plt.figure(figsize=(10, 10))
    plt.imshow(res, extent=(x0, x1, y0, y1), cmap=ListedColormap(color_list),␣
 ↪norm=Normalize(0, 4))
    return plt
```

[32]:

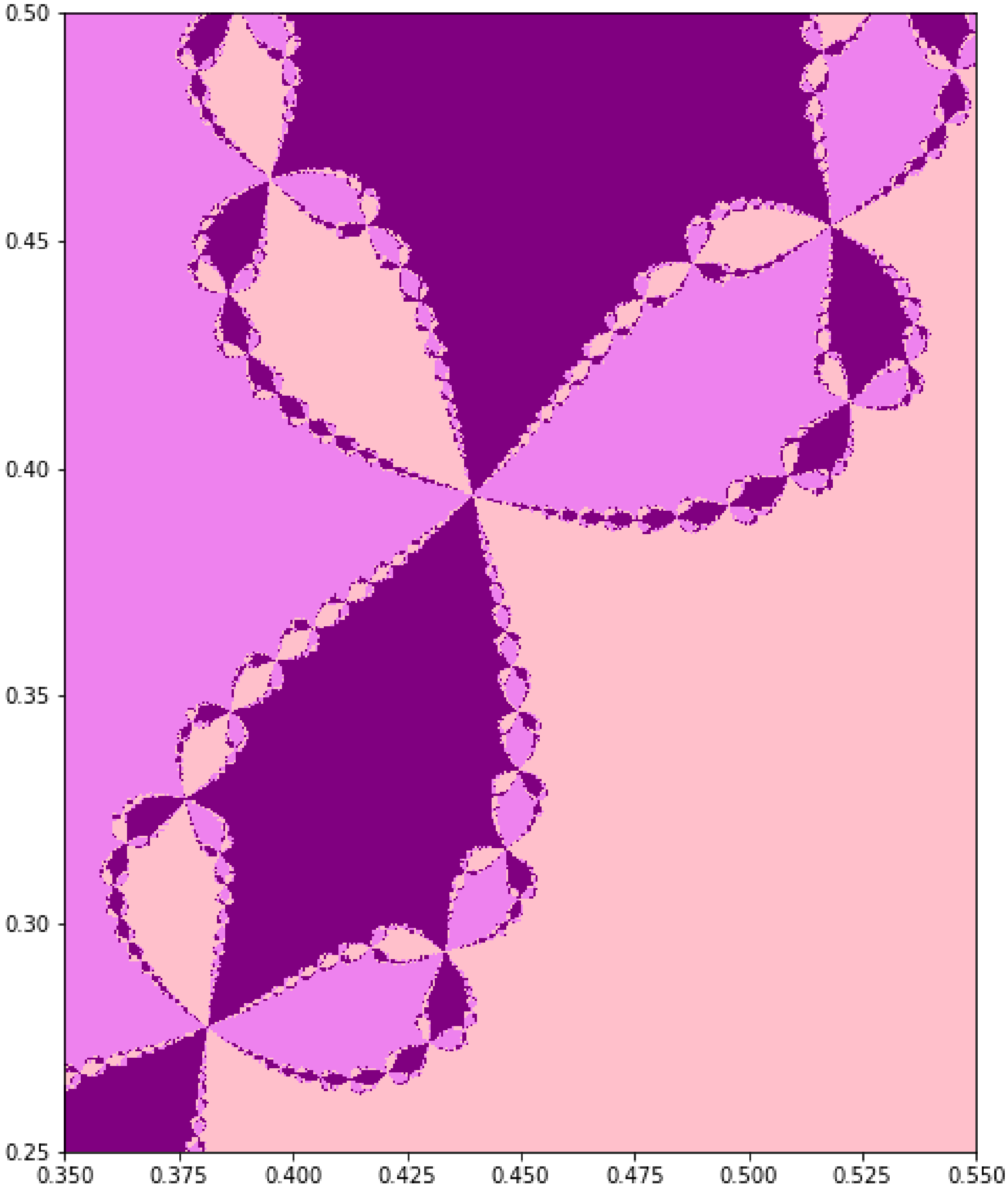
```
draw(-2, -2, 2, 2, 1000)
```



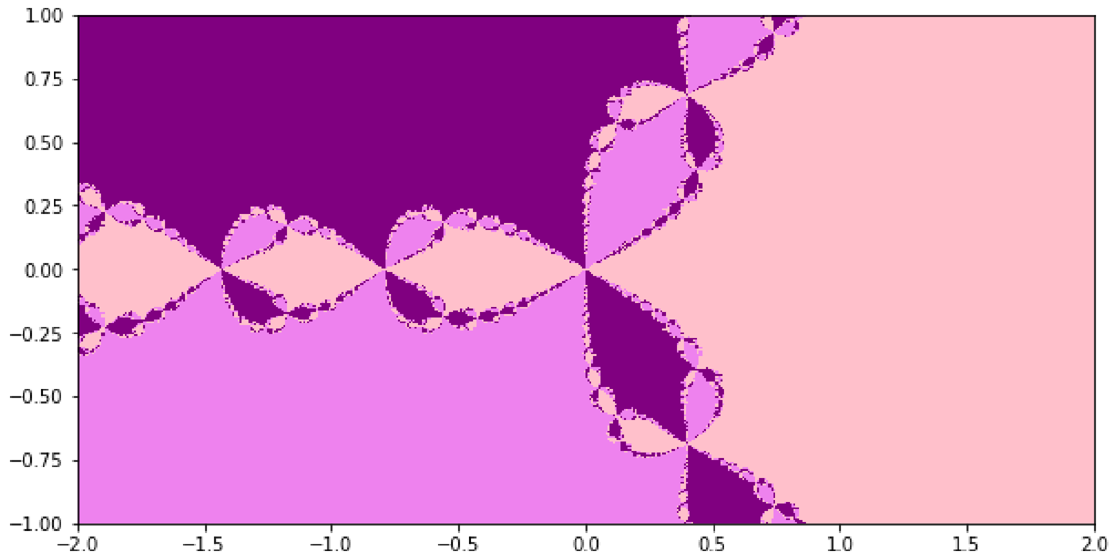
[34]:
```
draw(0, 0, 0.75, 0.75, 1000)
```



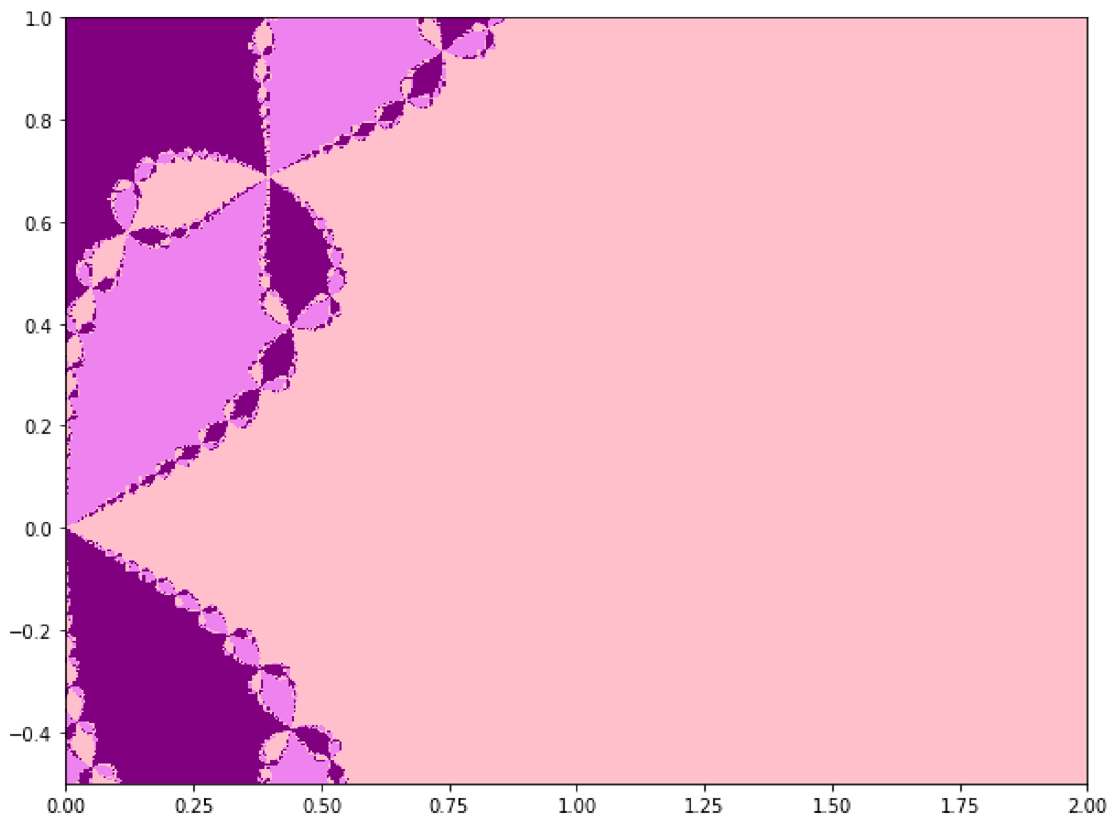
[38]:
```
draw(0.35, 0.25, 0.55, 0.5, 500)
```



[35]:
```
draw(-2, -1, 2, 1, 500)
```

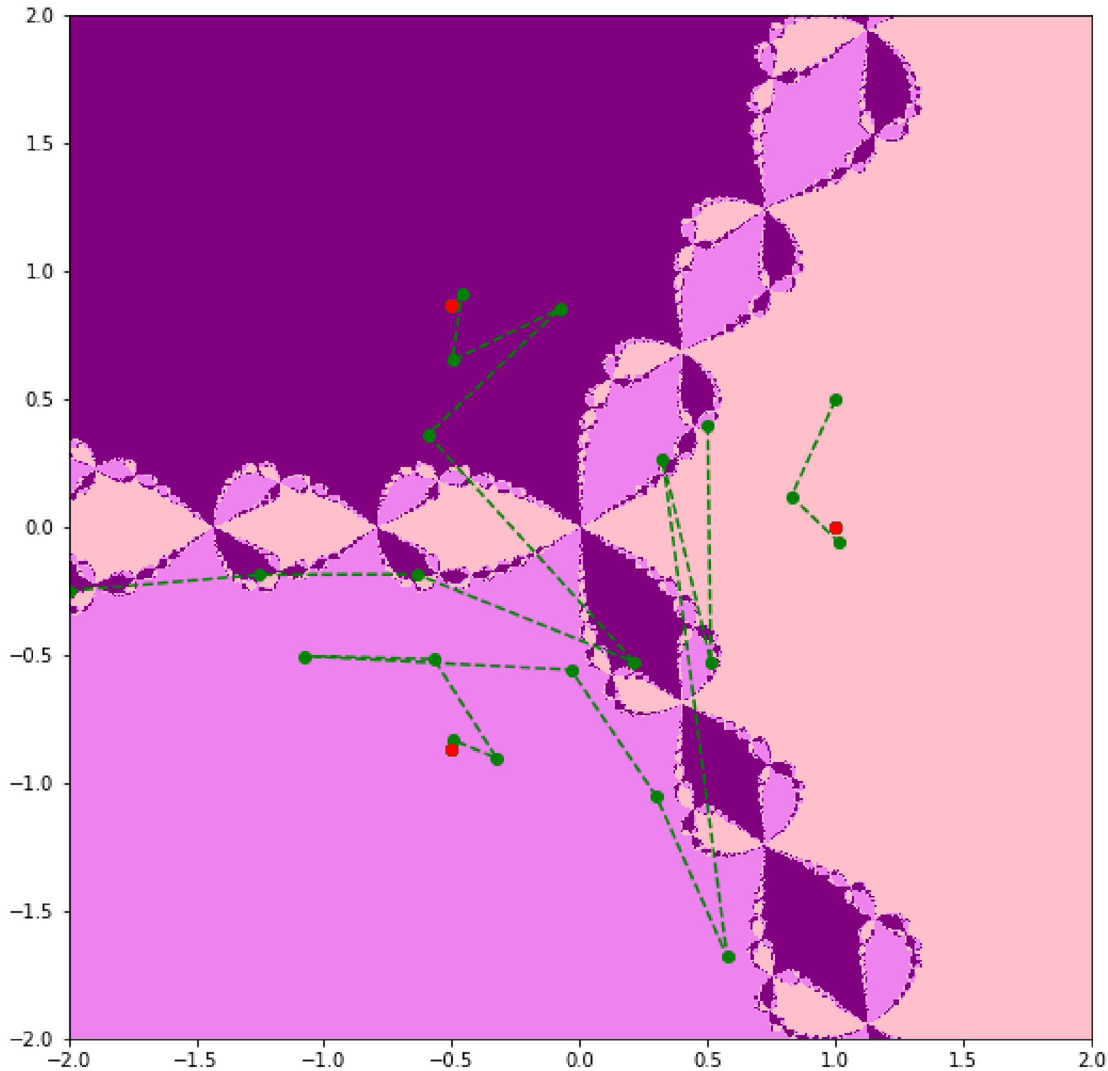


[36]:
```
draw(0, -0.5, 2, 1, 500)
```



[82]:
```
def drawSteps(z, plot):
    steps = []
    for i in range(STEPS):
        steps.append(z)
        z = step(z)
    plot.plot(np.real(steps), np.imag(steps), "go--")
    plot.plot(np.real(steps[-1]), np.imag(steps[-1]), "ro")
```

[83]:
```
plot = draw(-2, -2, 2, 2, 500)
for point in [1+0.5j, 0.5+0.4j, -2-0.25j]:
    drawSteps(point, plot)
```



[ ]:
```
```