task1

April 9, 2020

Уравнение для анализа

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

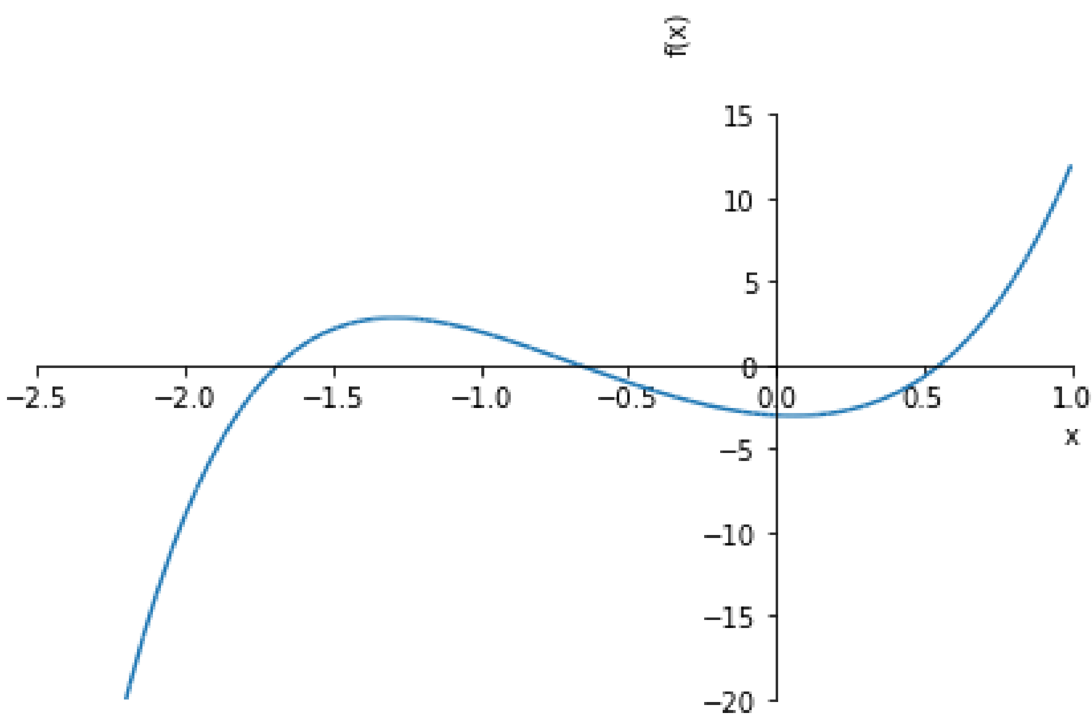
$x^5 + 2x^4 + 5x^3 + 8x^2 - x - 3$

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 относительняа 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 18, 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)`

[2]:
```python
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.ticker as ticker
import math
import random
```

[3]:
```python
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
```

[4]:
```python
def simple_iterations(epsilon, r, iter=1000, draw=False, left=0, right=0):
    (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:
        print('График зависимости членов итерационной последовательности от␣
↪номера')
        plt.figure(figsize=(15,5))
        plt.plot([float(i) / 10 for i in range(1, len(x) + 1)], x, 'r.', ms=3,␣
↪ls='-', lw=0.4)
        plt.show()
        print(f'Траектория сходимости в плоскости x - phi(x) на отрезке ({left},␣
↪{right})')
        plt.figure(figsize=(15,5))
        plt.plot([phi(i, r) for i in x], x, 'r.', ms=3, ls='-', lw=0.4)
        plt.show()

    return (next_x, iterations, wave)
```

[5]:
```
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,␣
↪left=left, right=right)
        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
```

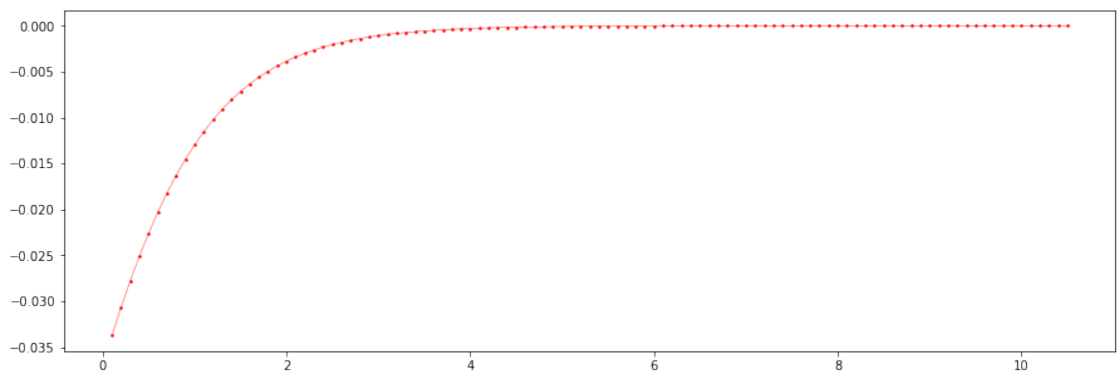
```
            roots[2][1].append(cur_r)

    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]))
```

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

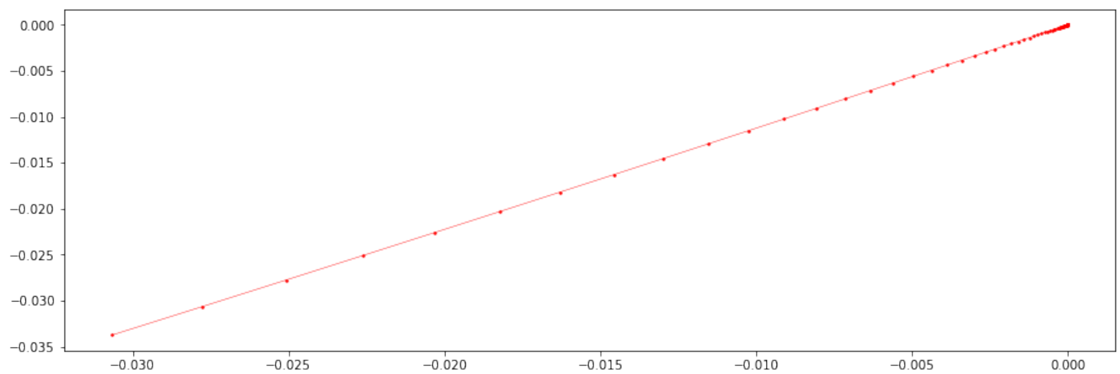
```
0.01 to 0.99 with eps = 1e-08 and iter = 10000:

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 =            []
```

[7]:
```
check_conv(0.01, 0.99, 2, 1e-8, 1e-6, draw=True)
```

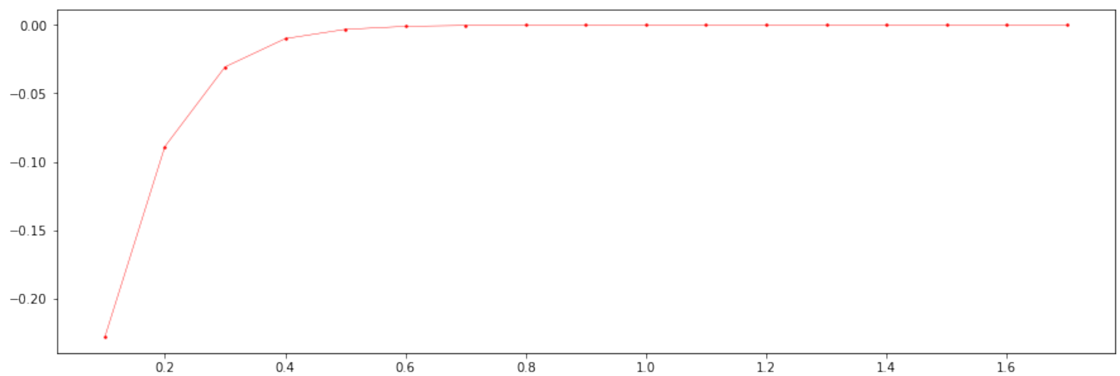
График зависимости членов итерационной последовательности от номера



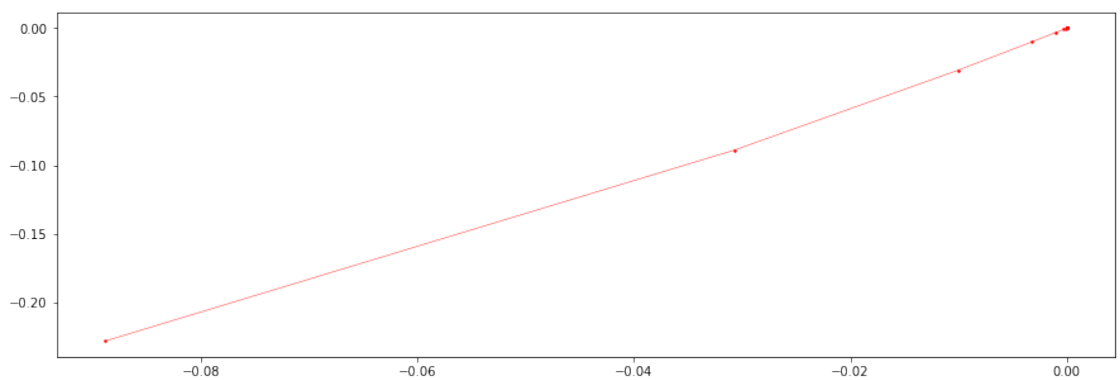
Траектория сходимости в плоскости x - phi(x) на отрезке (0.01, 0.99)



`График зависимости членов итерационной последовательности от номера`



`Траектория сходимости в плоскости x - phi(x) на отрезке (0.01, 0.99)`



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

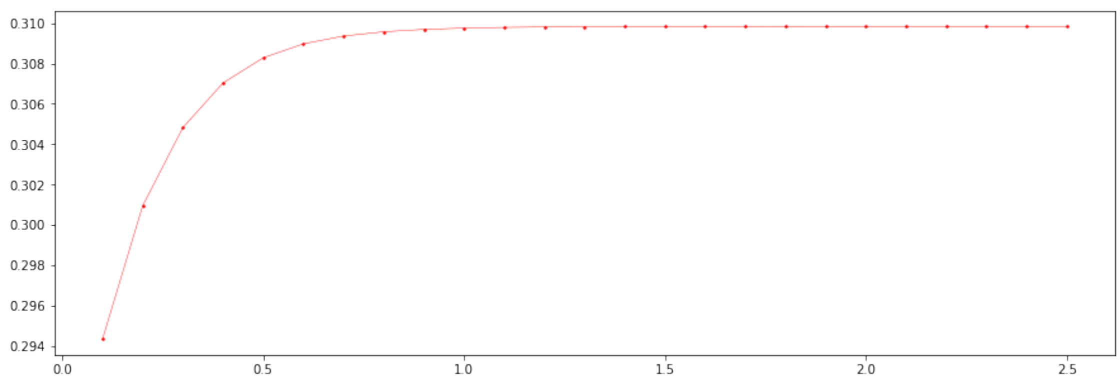
[8]:
```python
check_conv(1.01, 2.99, 10**4, 1e-8, 1e-6)
print('\n' * 3)
check_conv(1.01, 1.99, 10**4, 1e-8, 1e-6)
print('\n' * 3)
check_conv(2.01, 2.99, 10**4, 1e-8, 1e-6)
```

```
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 = 4952
root 1 - (1 / r), wave =     5024
not conv =                   24
not conv points =            [2.9892349550382793, 2.9845431350773706,
2.986365700667192, 2.985286764745876, 2.989432606371902, 2.987416387403414,
2.9877140806574776, 2.989492832782317, 2.986090572588725, 2.9865694559909226,
2.9856805322178643, 2.9885698117480555, 2.9872050991342274, 2.9853009295306943,
2.9885920705689135, 2.9852830597167834, 2.9899355767279827, 2.9867268642330727,
2.98629996898916, 2.9862933582892666, 2.9890135412729224, 2.9854290538687955,
2.987130080155495, 2.98477702869347]




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 =     9940
not conv =                   60
not conv points =            [2.9862714249309716, 2.989222617444504,
2.9856741404675207, 2.9889537774427635, 2.986040990395459, 2.9854267802815206,
2.9888789974099357, 2.9863523897144666, 2.985269168437876, 2.9898253800813572,
2.988245187465537, 2.9884576373842178, 2.985015949866633, 2.989826752108873,
```

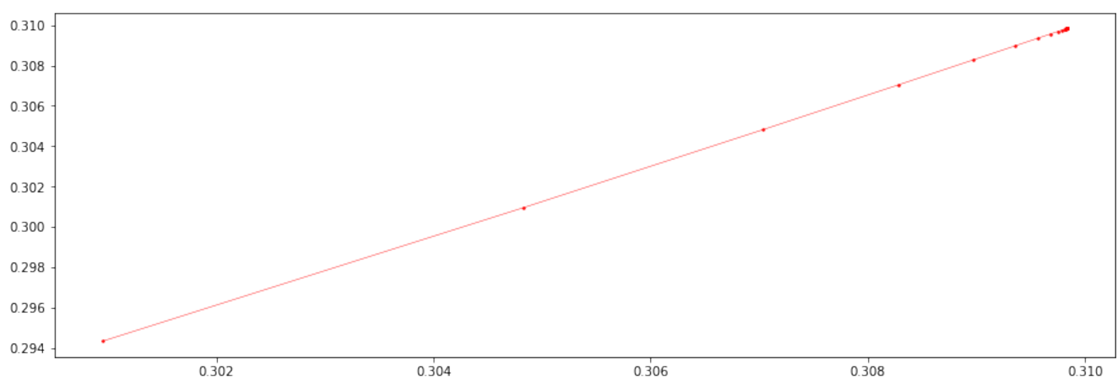
```
2.984763646690693, 2.9856906784269968, 2.9884289800966877, 2.9890485077534574,
2.9888389452249857, 2.9894803832509074, 2.9873829124364857, 2.9885820227448283,
2.987133512697204, 2.9895486979692496, 2.9871358088499687, 2.989256236992371,
2.9873703019751794, 2.986544481192915, 2.987818394433347, 2.9848090668799037,
2.9863769736705597, 2.987431146996556, 2.984946683874562, 2.988958840751083,
2.987333899338654, 2.9857916851518898, 2.984660199187437, 2.987273893166856,
2.98848191798856, 2.984794017449434, 2.9880088297468776, 2.9856325910563224,
2.9870102006634665, 2.9885419116294782, 2.9869363452561344, 2.9845174606050384,
2.9878425959165797, 2.986768078574401, 2.984588352969049, 2.9884038183181465,
2.989706153415825, 2.9854506248850115, 2.98930933630881, 2.985032814037498,
2.9853620044646902, 2.985831763102135, 2.9862976658190736, 2.989656010915273,
2.985133625994884, 2.989571614660689]
```

[9]:
```python
check_conv(1.01, 1.99, 2, 1e-8, 1e-6, draw=True)
```

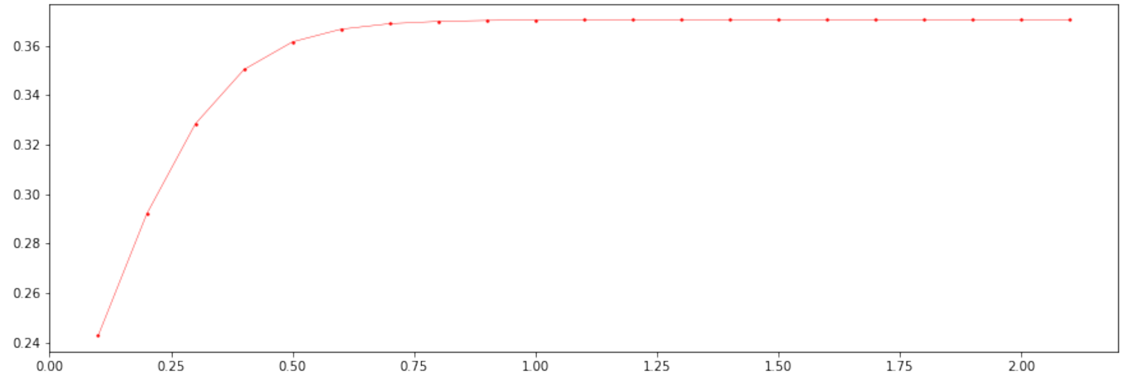
```
График зависимости членов итерационной последовательности от номера
```



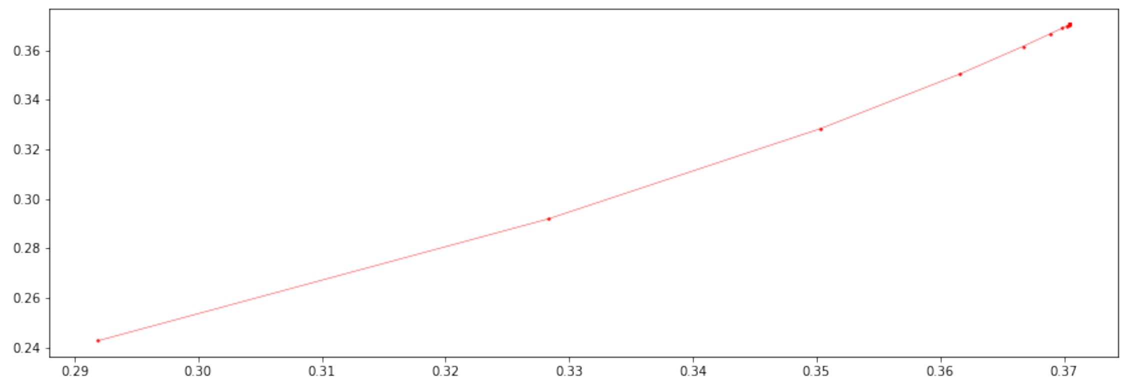

```
Траектория сходимости в плоскости x - phi(x) на отрезке (1.01, 1.99)
```




```
График зависимости членов итерационной последовательности от номера
```

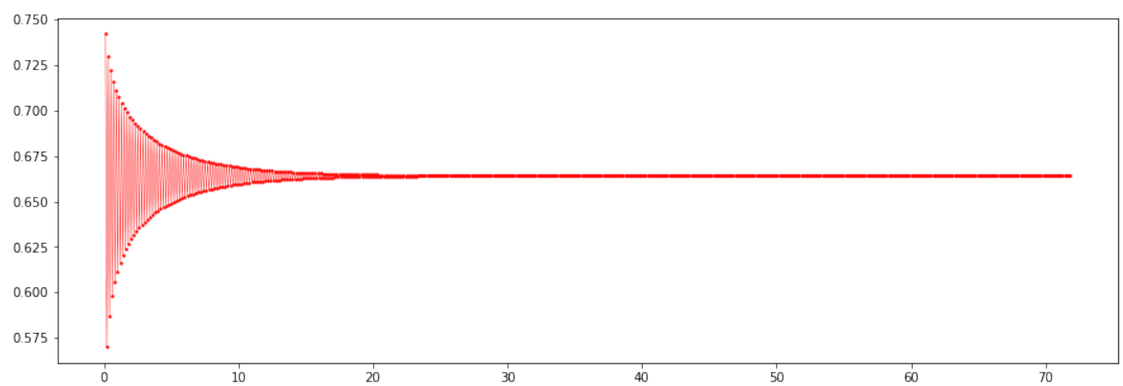


Траектория сходимости в плоскости x - phi(x) на отрезке (1.01, 1.99)



[10]:
```
check_conv(2.01, 2.99, 2, 1e-8, 1e-6, draw=True)
```

График зависимости членов итерационной последовательности от номера



Траектория сходимости в плоскости x - phi(x) на отрезке (2.01, 2.99)

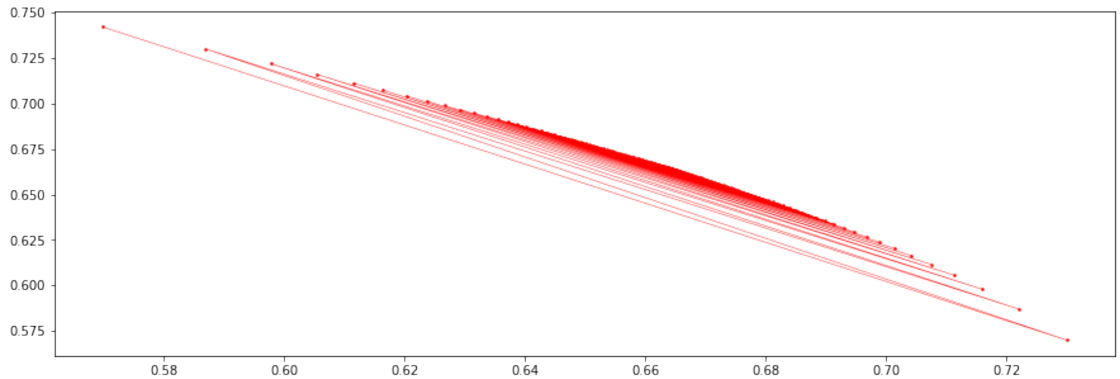
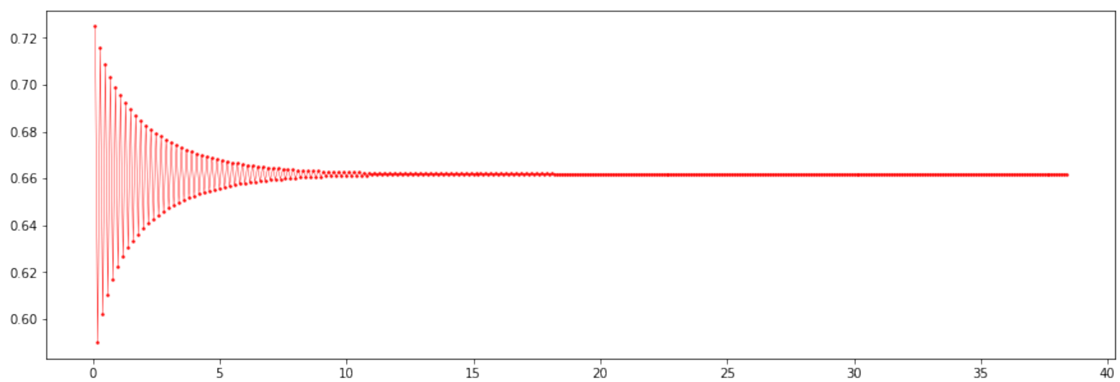
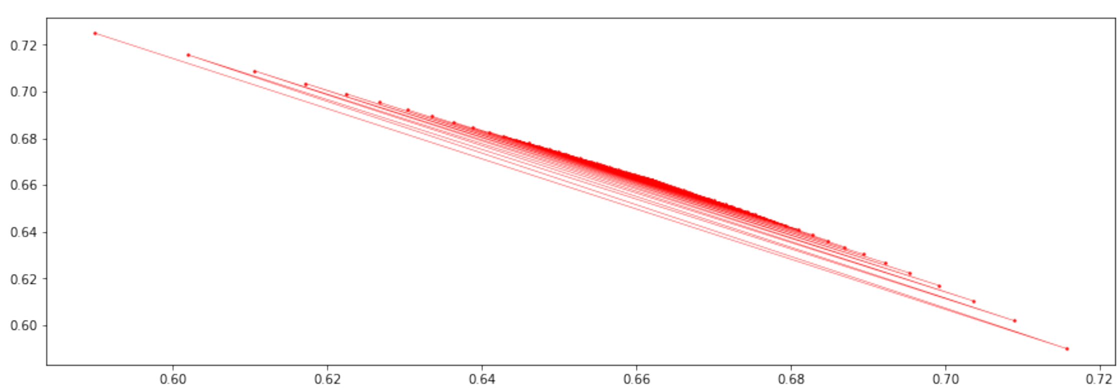


График зависимости членов итерационной последовательности от номера



Траектория сходимости в плоскости x - phi(x) на отрезке (2.01, 2.99)

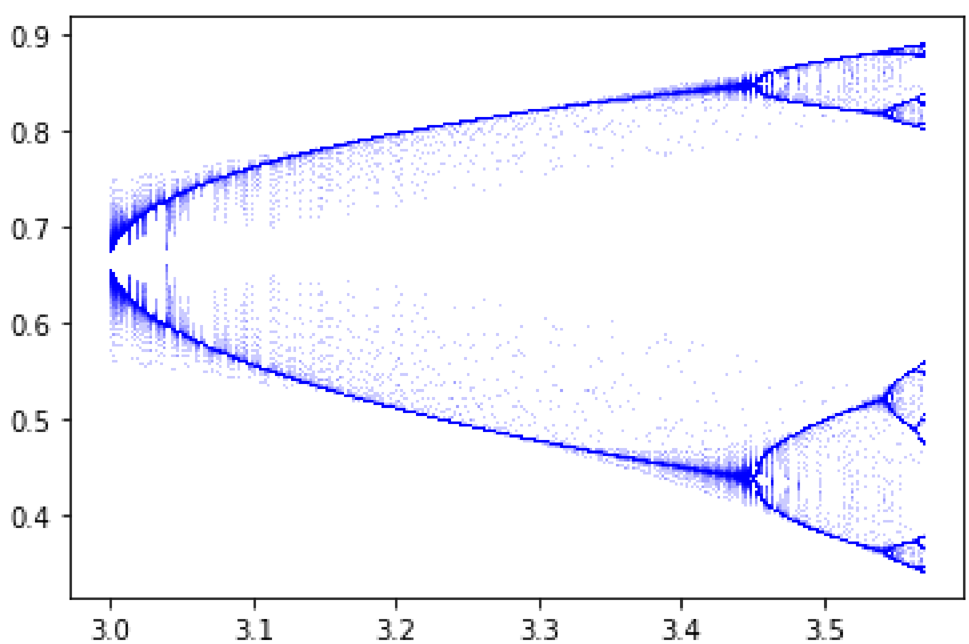


Как видно из приведенных нами вычислений, при выборе **r** из промежутка (1, 3) данная нам функция **phi** сходится к корню x1 = 1 - (1 / r)

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

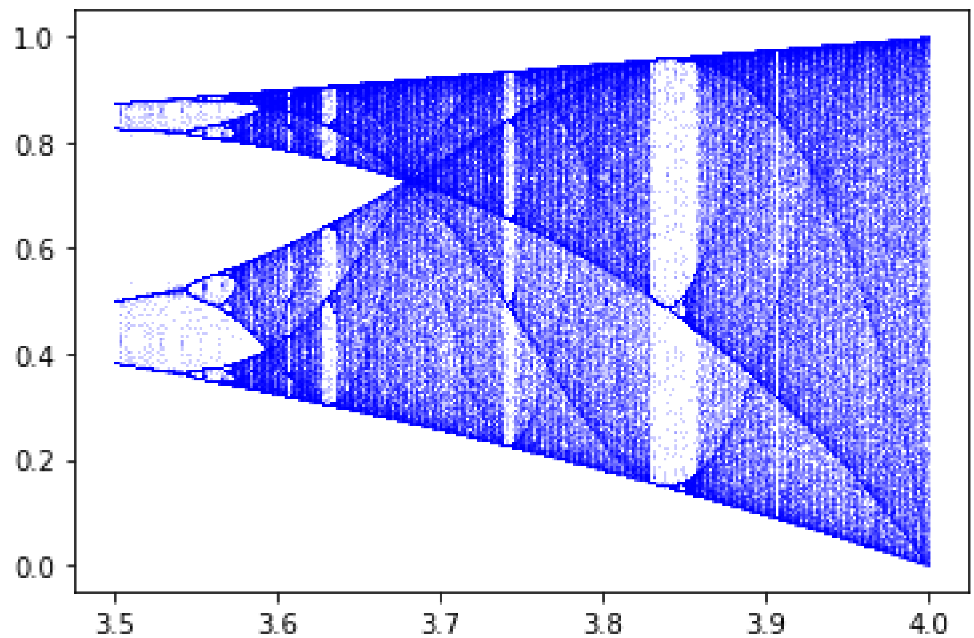
[88]:
```
def show_graphic(r, rang, eps):
    xVals = []
    yVals = []
    yVals1 = []

    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()
```

[89]:
```
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.

[13]:
```
def show_chaos():
    show_graphic(3.50, 500, 0.001)

show_chaos()
```



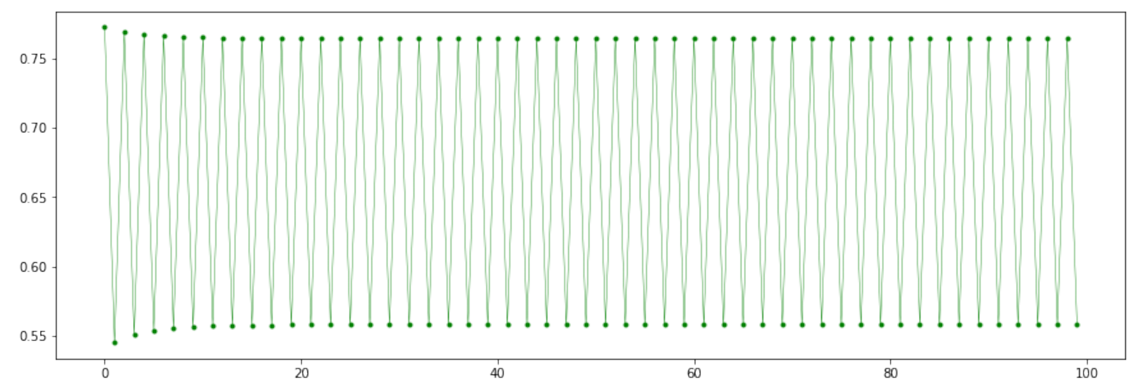
По теории, между числами R(inf) и 4, поведение последовательности должно представлять из себя детерменированный хаос. График ведет себя в соответствии с теорией. На нем присутствуют зоны таких R, при которых наблюдаются сгущения и разрежения итерационной последовательности. По теории в окрестности R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

[118]:
```
def show_iteration1(r, rang, eps):
    xVals = []
    yVals1 = []

    (lBorder, rBorder) = get_window(r)
    x = random.uniform(lBorder, rBorder)
    for i in range(rang):
        x = phi(x, r)
        xVals.append(x)
        yVals1.append(i)
```

```
    plt.figure(figsize=(15,5))
    plt.plot(yVals1, xVals, 'g.', ms=6, ls='-', lw = 0.4)
    plt.show()
```

[119]:
```
print("График зависимости членов итерационной последовательности от номера␣
 ↪итерации в области каскада бифуркаций"
      " удвоения периода r = 3.1.")
show_iteration1(3.1, 100, 0.01)
print("График зависимости членов итерационной последовательности от номера␣
 ↪итерации в области каскада бифуркаций"
      " удвоения периода r = 3.53.")
show_iteration1(3.53, 100, 0.01)
print("График зависимости членов итерационной последовательности от номера␣
 ↪итерации в области хаоса r = 3.7.")
show_iteration1(3.7, 100, 0.01)
```

```
График зависимости членов итерационной последовательности от номера итерации в
области каскада бифуркаций удвоения периода r = 3.1.
```




```
График зависимости членов итерационной последовательности от номера итерации в
области каскада бифуркаций удвоения периода r = 3.53.
```

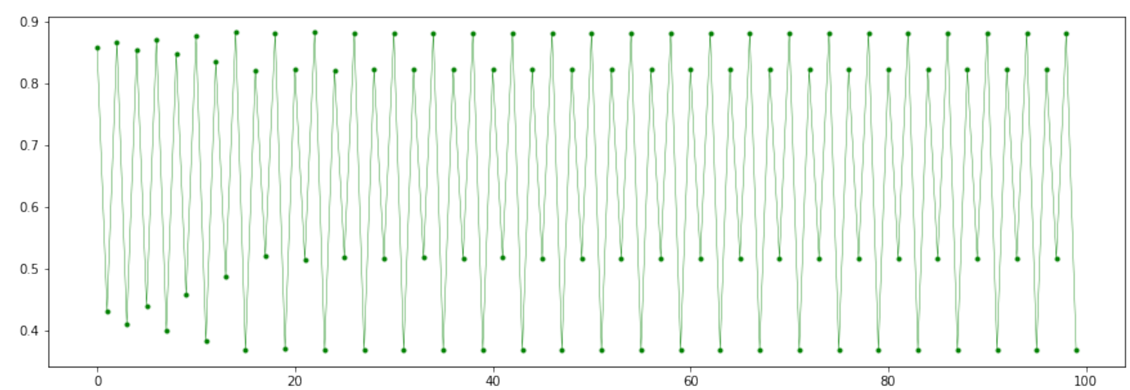
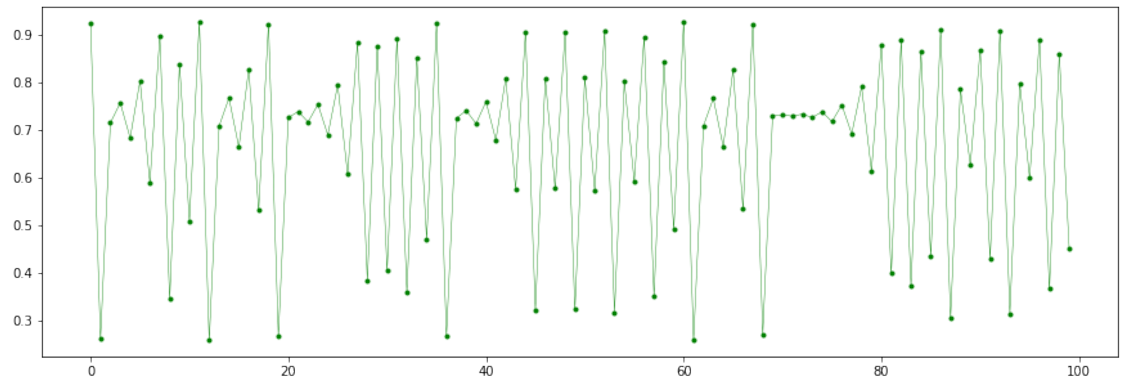


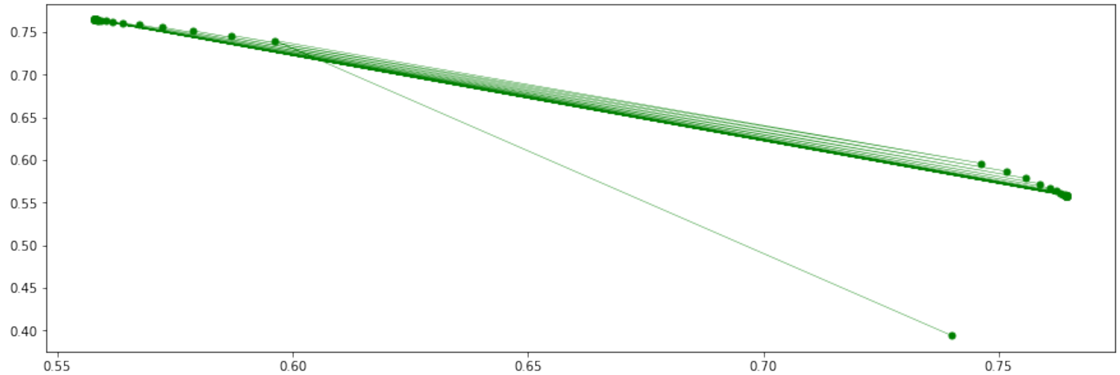
График зависимости членов итерационной последовательности от номера итерации в области хаоса r = 3.7.



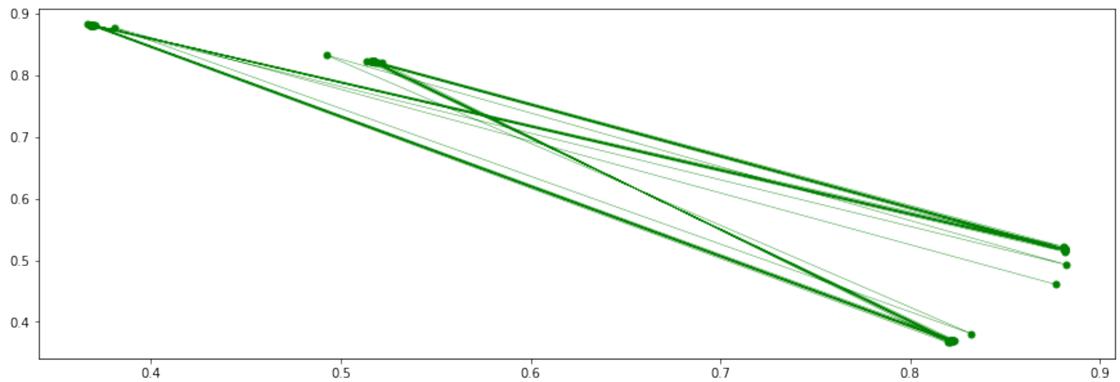

```
[122]: def show_iteration2(r, rang, eps):
           xVals = []
           yVals1 = []

           (lBorder, rBorder) = get_window(r)
           x = random.uniform(lBorder, rBorder)
           for i in range(rang):
               xVals.append(x)
               x = phi(x, r)
               yVals1.append(x)

           plt.figure(figsize=(15,5))
           plt.plot(yVals1, xVals, 'g.', ms=10, ls='-', lw = 0.4)
           plt.show()
```

```
[123]: print("Траектория сходимости в плоскости x phi(x). r = 3.1.")
       show_iteration2(3.1, 200, 0.01)
       print("Траектория сходимости в плоскости x phi(x). r = 3.53.")
       show_iteration2(3.53,200, 0.01)
       print("Траектория сходимости в плоскости x phi(x). r = 3.7.")
       show_iteration2(3.7, 100, 0.01)
```

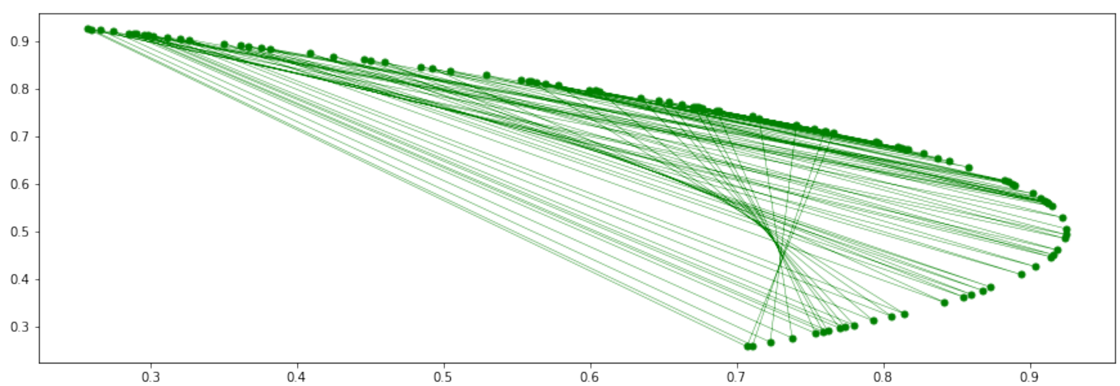
Траектория сходимости в плоскости x phi(x). r = 3.1.



Траектория сходимости в плоскости x phi(x). r = 3.53.



Траектория сходимости в плоскости x phi(x). r = 3.7.




```
[ ]:
```

```
[ ]:
```

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
```

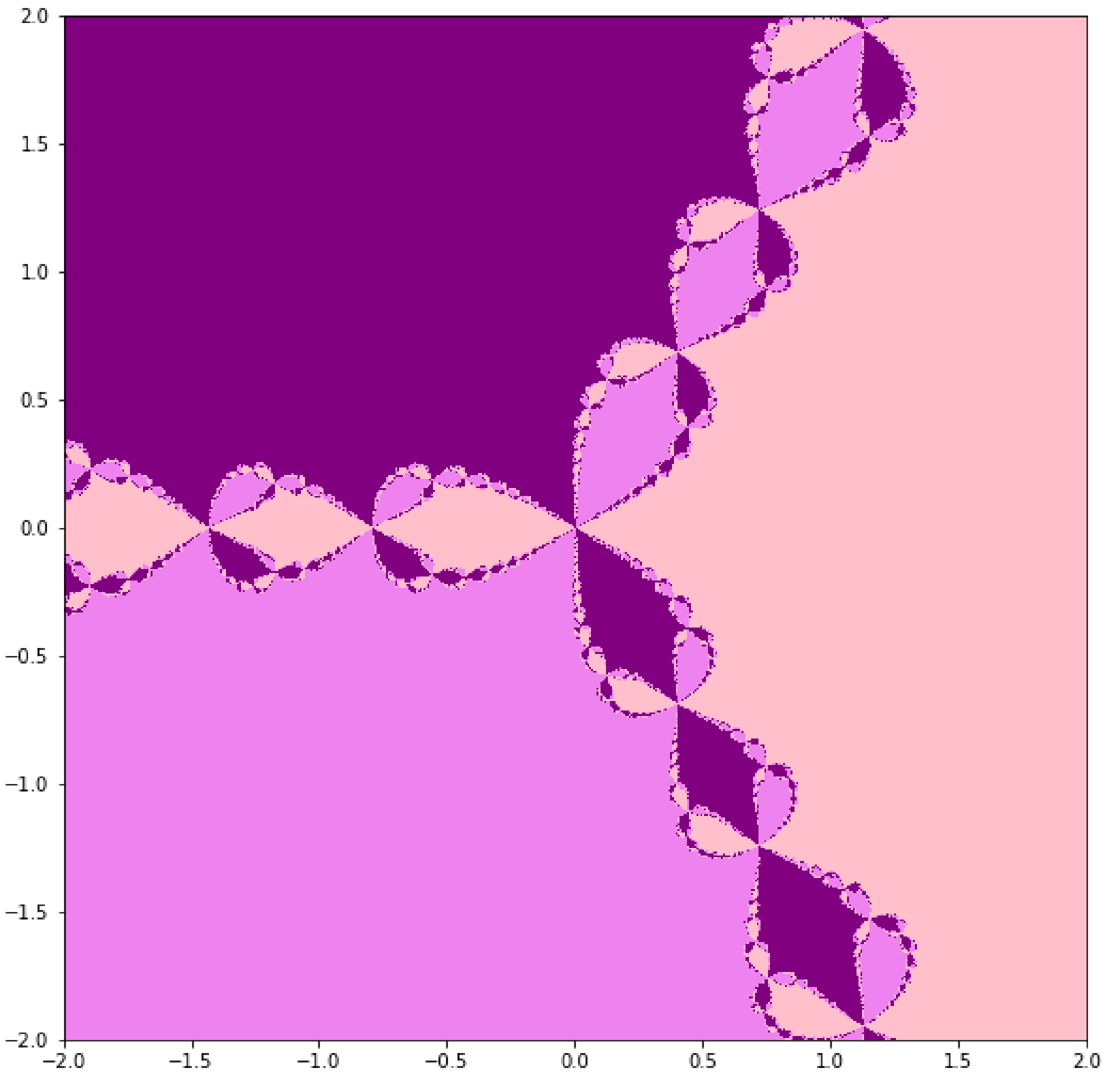
[2]:
```
STEPS = 100
EPS = 1e-10

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)
```

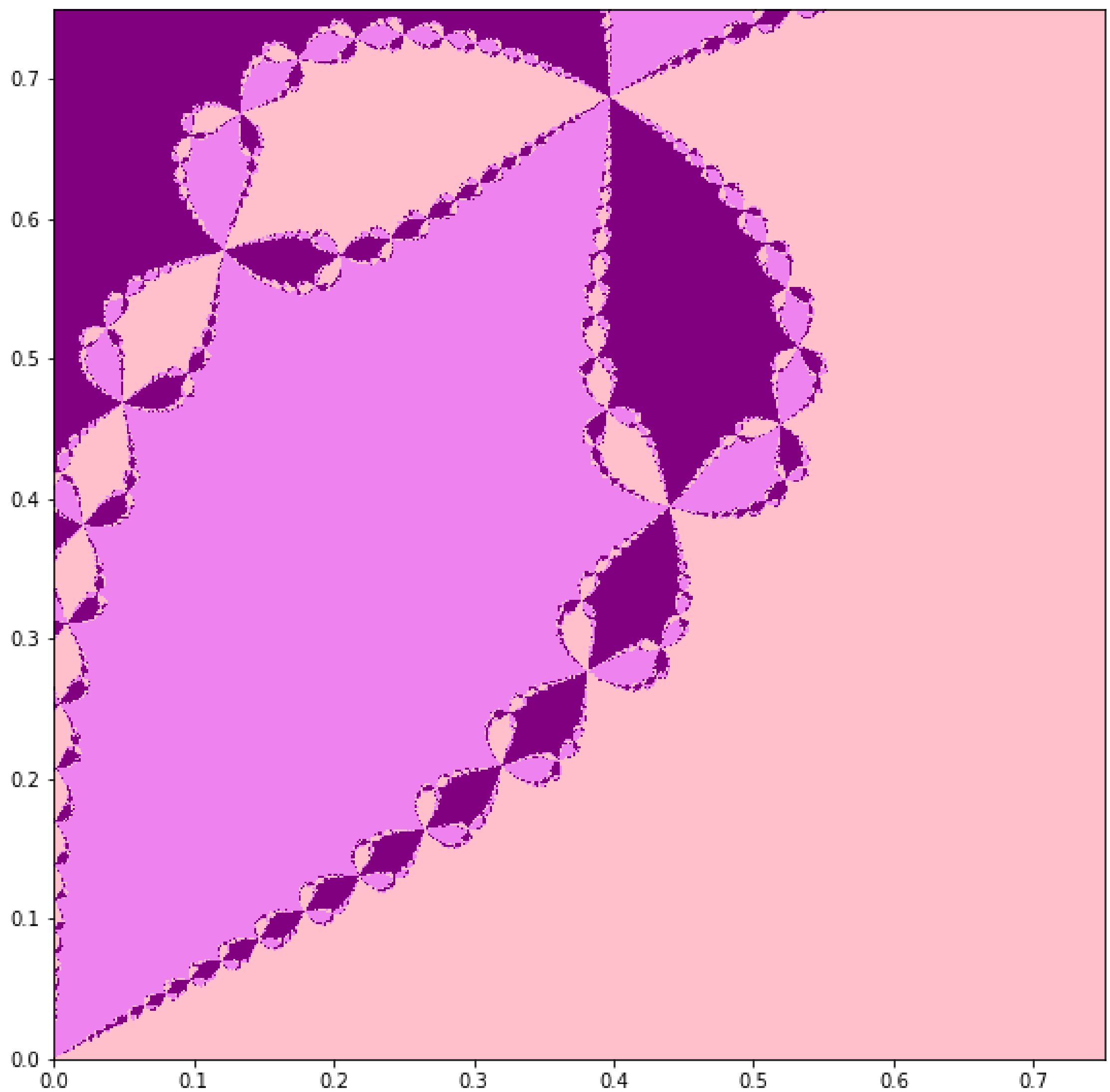
[4]:
```
from multiprocessing import Pool

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
```

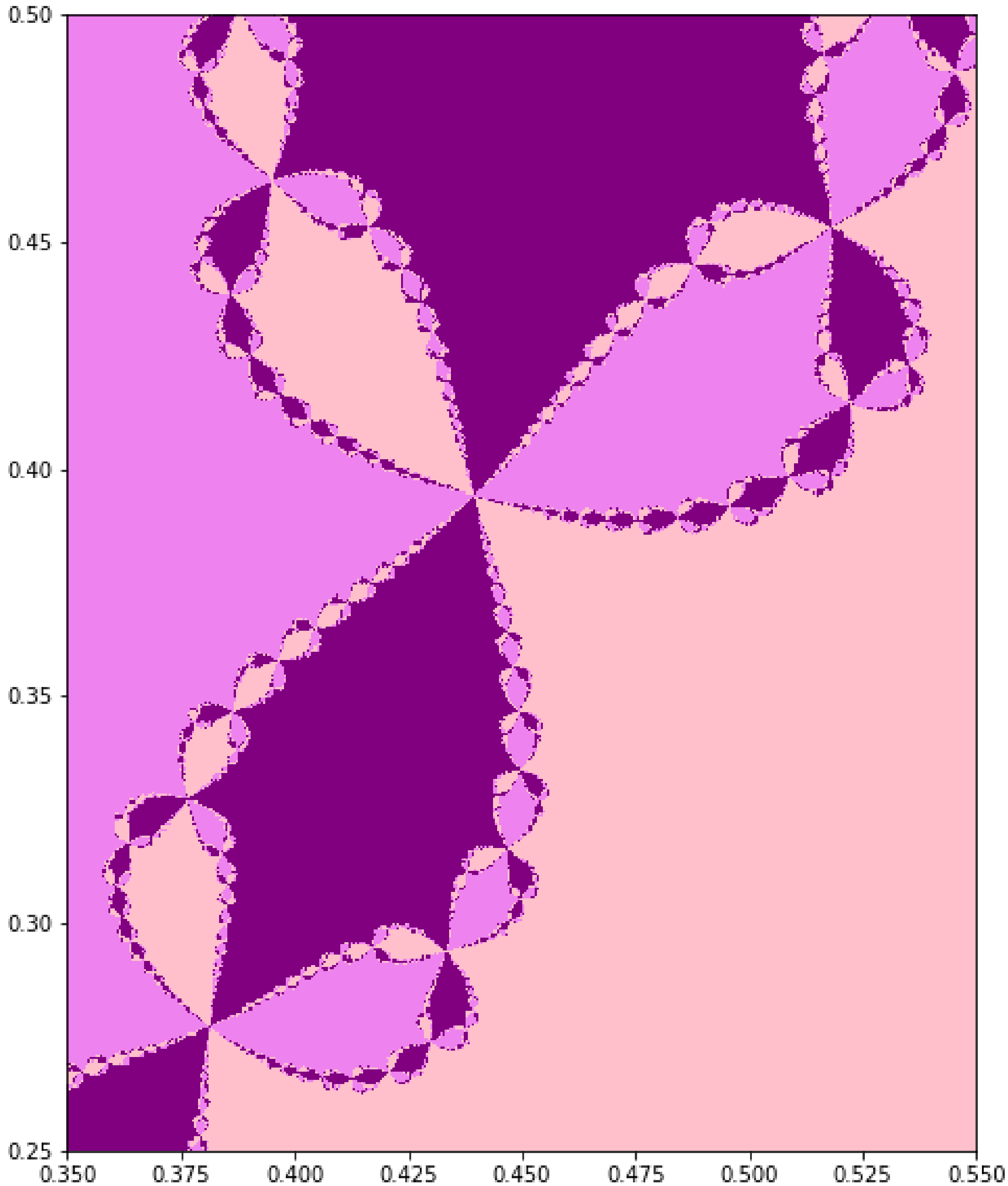
[32]:
```
draw(-2, -2, 2, 2, 1000)
```



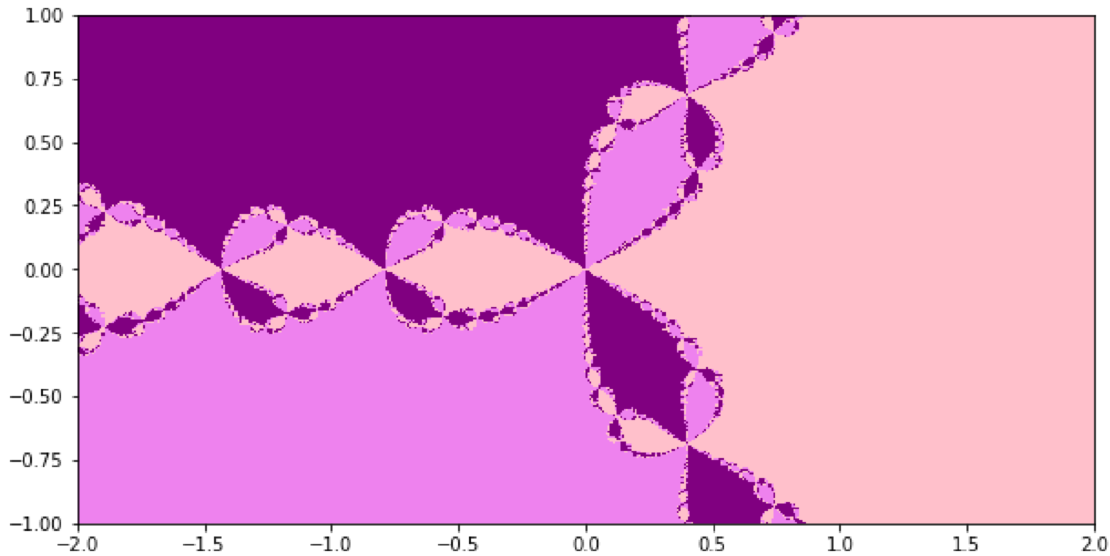
[34]:
```
draw(0, 0, 0.75, 0.75, 1000)
```



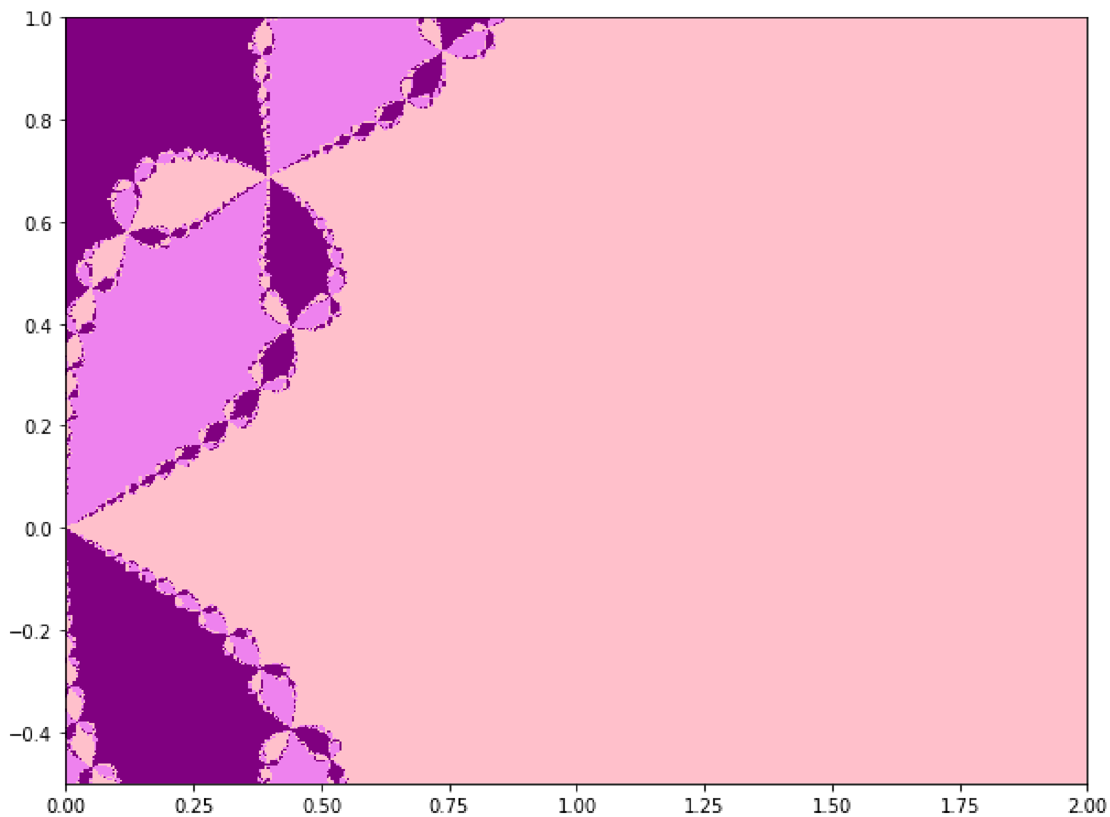
[38]:
```
draw(0.35, 0.25, 0.55, 0.5, 500)
```



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

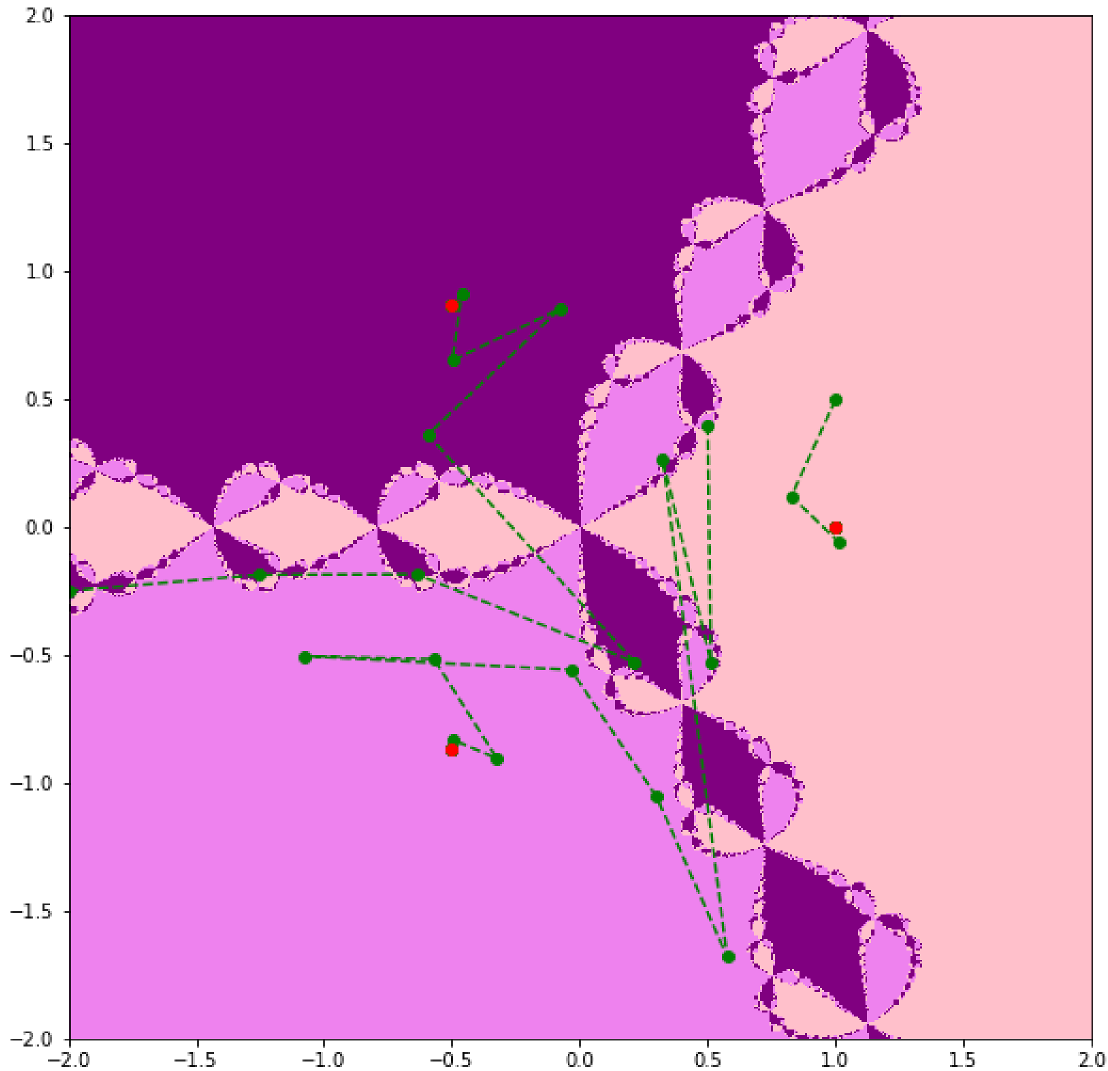


[36]:
```
draw(0, -0.5, 2, 1, 500)
```



[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)
```



[ ]: