Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Теория функций комплексного переменного

Лабораторная работа №1

Выполнили: студенты группы P3208, Васильев Н. А., Елисеев К. И.,

Есоян В. С.

Преподаватель: Бойцев А. А.

Санкт-Петербург 2024

# Текст задания

1. Докажите свойства 1 и 2 для множества Мандельброта.
2. Напишите программу, которая будет строить визуализацию множества Мандельброта. Выберете разумные ограничения, поварьируйте максимальное количество итераций. Попробуйте приблизить отдельные части множества, чтобы увидеть фрактальную структуру.
3. Напишите программу, которая по заданному *c* строит заполненное множество Жюлиа. Поварьируйте максимальное количество итераций, попробуйте пронаблюдать фрактальную структуру, рассмотрите множество при разных *с*.
4. Найдите какой-нибудь неразобранный фрактал. Опишите его структуру, построение. Нарисуйте визуализации.

# Доказательство

Свойства множества Мандельброта:

1. Множество Мандельброта переходит само в себя при сопряжении.

► Множество Мандельброта переходит само в себя при сопряжении. Иными словами, оно симметрично относительно вещественной оси.

Пусть

Тогда

Следовательно,

При сопряжении значение 𝑥 останется прежним, то есть координата 𝑥 не поменяется, а значение 𝑦 заменяется на −𝑦, поэтому точки симметричны относительно оси O𝑥

◄

1. Если , то *c* не принадлежит множеству Мандельброта.

► Пусть

Пусть , где

С помощью индукции нужно показать, что для

База индукции

Предположение индукции

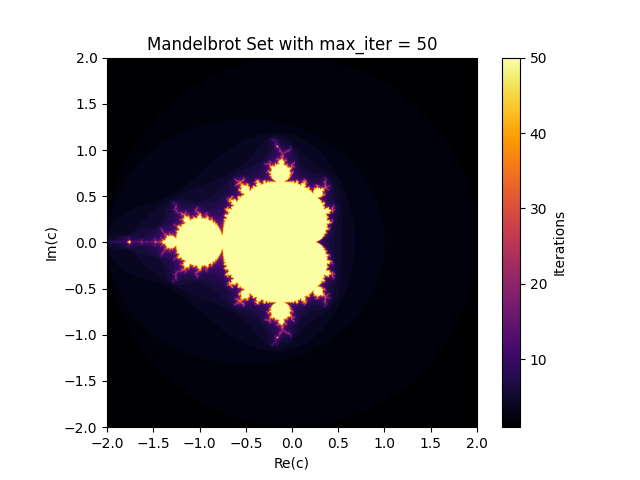
Переход индукции :

Это означает, что при , когда , и поэтому не может быть ограниченным для всех

◄

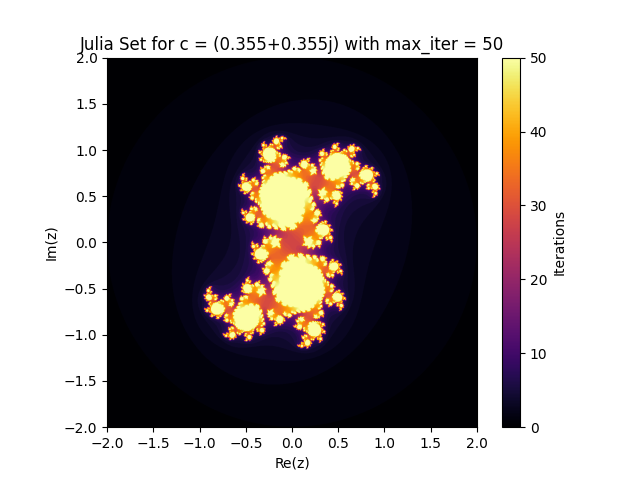
# Исходный код программы визуализации множества Мандельброта на Python

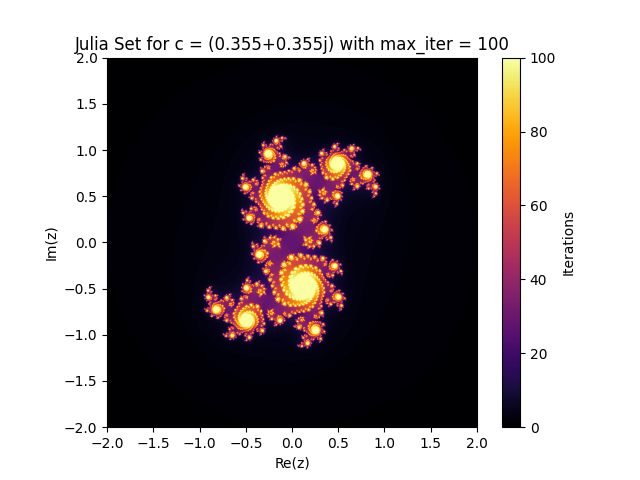
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def main():  
 center = (0.0, 0.0)  
 scale = 2.0  
 resolution = (800, 800)  
 max\_iters = [50, 100, 150]  
  
 plot\_mandelbrot\_set(center, scale, resolution, max\_iters)  
  
  
def mandelbrot\_set(center, scale, resolution, max\_iter):  
 width, height = resolution  
 x\_min = center[0] - scale  
 x\_max = center[0] + scale  
 y\_min = center[1] - scale  
 y\_max = center[1] + scale  
  
 x\_vals = np.linspace(x\_min, x\_max, width)  
 y\_vals = np.linspace(y\_min, y\_max, height)  
  
 image = np.zeros((height, width))  
  
 for x\_idx, x in enumerate(x\_vals):  
 for y\_idx, y in enumerate(y\_vals):  
 c = x + 1j \* y  
 z = 0  
 iteration = 0  
 while abs(z) <= 2 and iteration < max\_iter:  
 z = z \*\* 2 + c  
 iteration += 1  
 image[y\_idx, x\_idx] = iteration  
  
 return image  
  
  
def plot\_mandelbrot\_set(center, scale, resolution, max\_iters):  
 for max\_iter in max\_iters:  
 mandelbrot\_image = mandelbrot\_set(center, scale, resolution, max\_iter)  
 plt.imshow(mandelbrot\_image, cmap="inferno", extent=(  
 center[0] - scale, center[0] + scale, center[1] - scale, center[1] + scale))  
 plt.colorbar(label='Iterations')  
 plt.title(f"Mandelbrot Set with max\_iter = {max\_iter}")  
 plt.xlabel("Re(c)")  
 plt.ylabel("Im(c)")  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

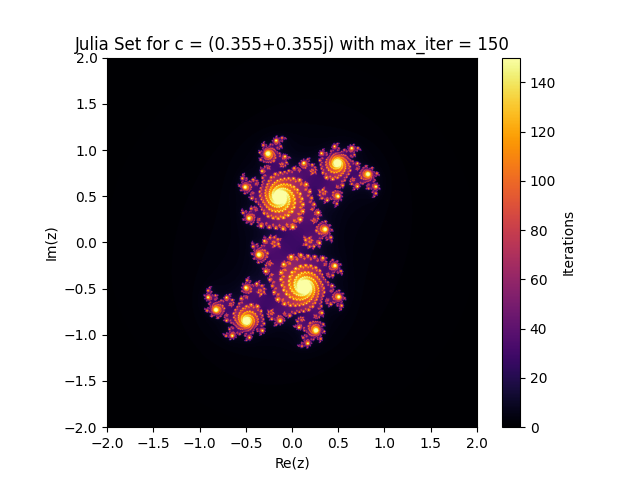


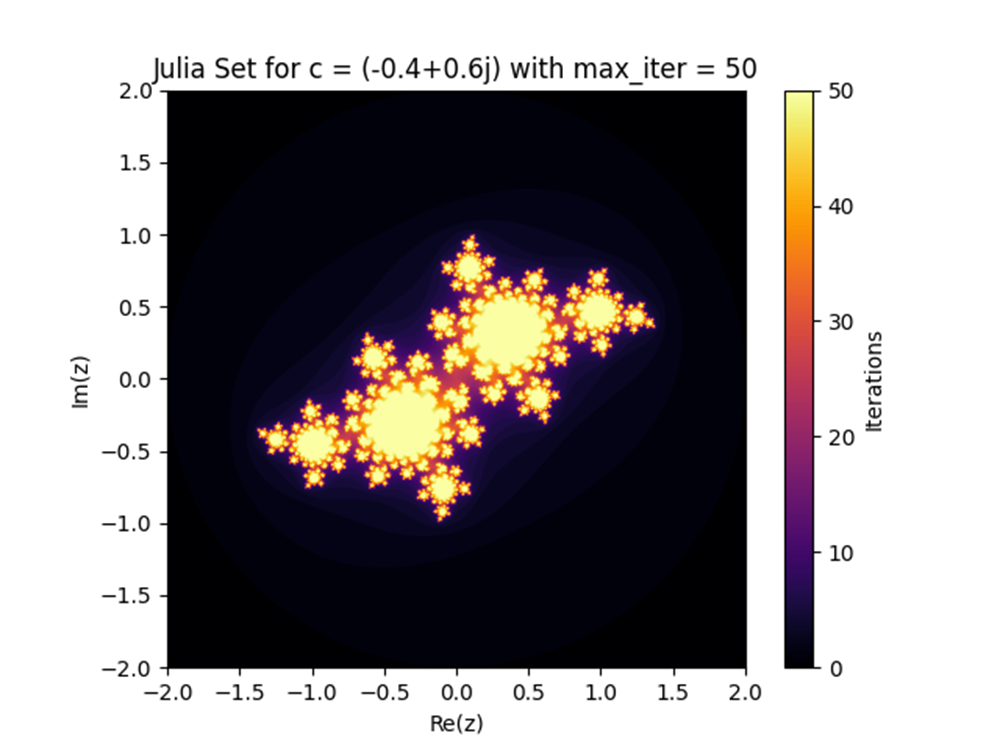
# Исходный код программы визуализации множества Жюлиа на Python

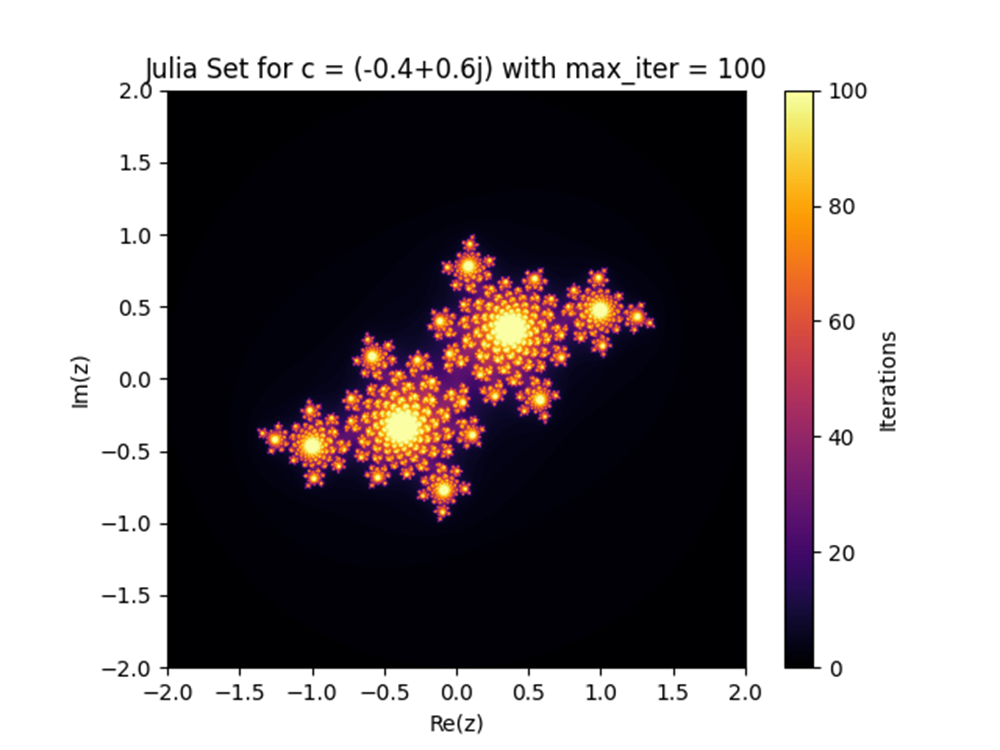
import numpy as np  
import matplotlib.pyplot as plt  
  
  
def main():  
 center = (0.0, 0.0)  
 scale = 2.0  
 resolution = (1600, 1600)  
 c\_values = [0.355 + 0.355j, -0.4 + 0.6j, -0.70176 - 0.3842j, -0.5251993 + 0.5251993j]  
 max\_iters = [50, 100, 150]  
  
 plot\_julia\_set(center, scale, resolution, c\_values, max\_iters)  
  
  
def julia\_set(center, scale, resolution, c, max\_iter):  
 width, height = resolution  
 x\_min = center[0] - scale  
 x\_max = center[0] + scale  
 y\_min = center[1] - scale  
 y\_max = center[1] + scale  
  
 x\_vals = np.linspace(x\_min, x\_max, width)  
 y\_vals = np.linspace(y\_min, y\_max, height)  
  
 image = np.zeros((height, width))  
  
 for x\_idx, x in enumerate(x\_vals):  
 for y\_idx, y in enumerate(y\_vals):  
 z = x + 1j \* y  
 iteration = 0  
 while abs(z) <= 2 and iteration < max\_iter:  
 z = z \*\* 2 + c  
 iteration += 1  
 image[y\_idx, x\_idx] = iteration  
  
 return image  
  
  
def plot\_julia\_set(center, scale, resolution, c\_values, max\_iters):  
 for c in c\_values:  
 for max\_iter in max\_iters:  
 julia\_image = julia\_set(center, scale, resolution, c, max\_iter)  
 plt.imshow(julia\_image, cmap="inferno", extent=(  
 center[0] - scale, center[0] + scale, center[1] - scale, center[1] + scale))  
 plt.colorbar(label='Iterations')  
 plt.title(f"Julia Set for c = {c} with max\_iter = {max\_iter}")  
 plt.xlabel("Re(z)")  
 plt.ylabel("Im(z)")  
 plt.show()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

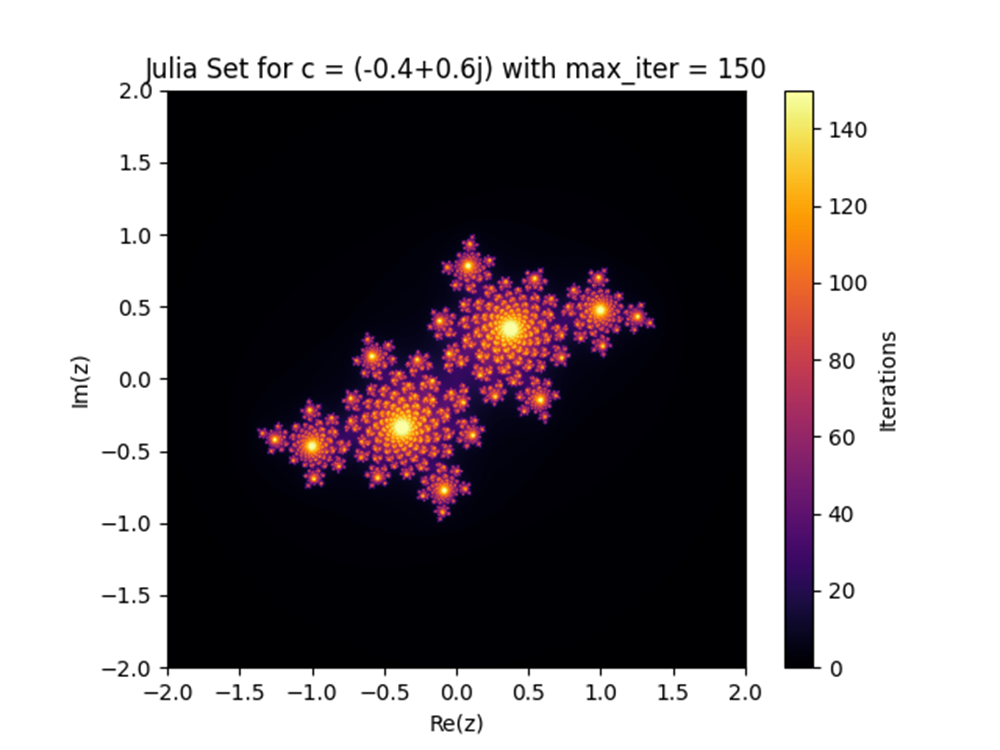


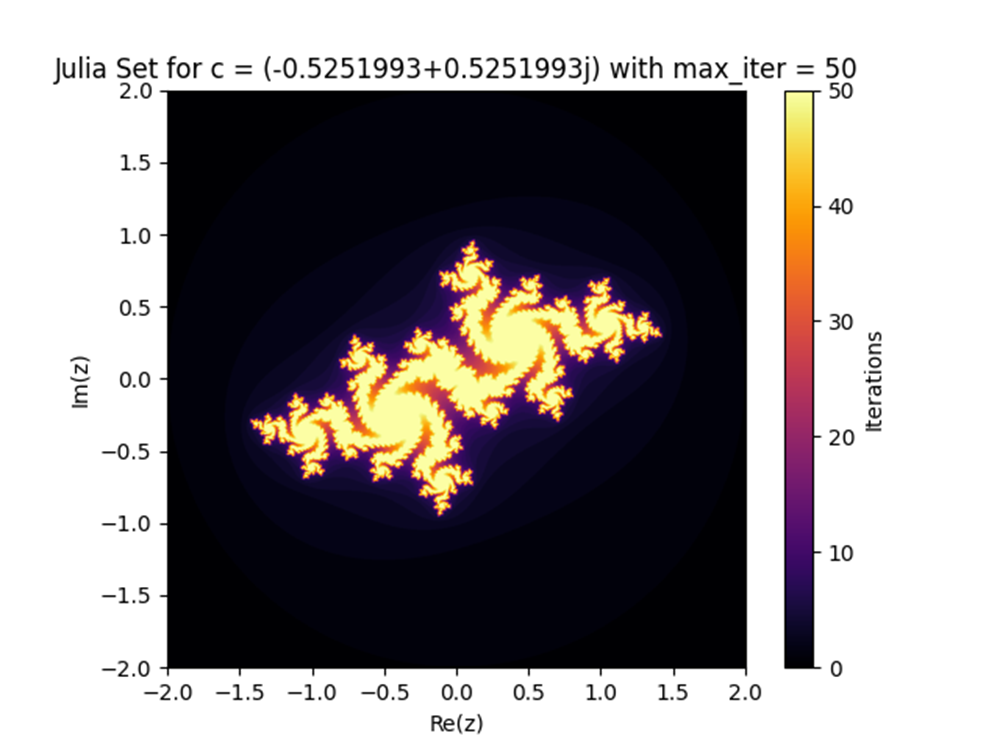


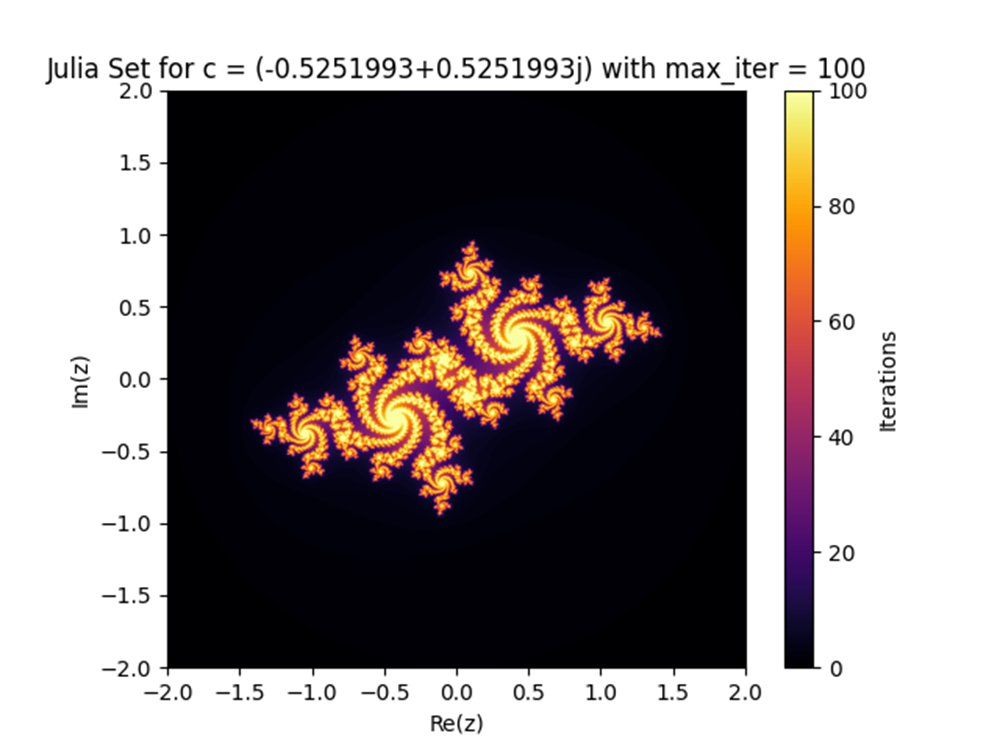


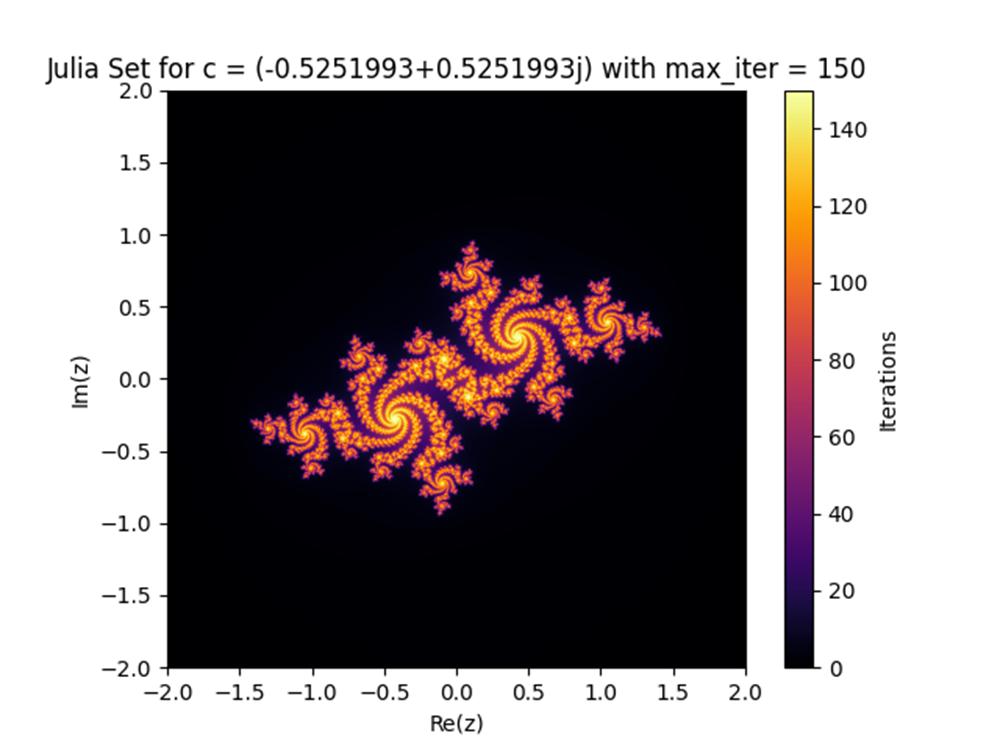












# Фрактал

**Фрактал Burning Ship** был открыт Бренданом Боуэрсом в 1992 году. Он отличается от множества Мандельброта тем, что в процессе итерации используются абсолютные значения действительной и мнимой частей z, что придаёт фракталу форму, напоминающую горящее судно.

Формула итерации для фрактала Burning Ship выглядит следующим образом:

# Исходный код программы визуализации фрактала Burning Ship на Python

import numpy as np  
import matplotlib.pyplot as plt  
from numba import jit  
  
  
@jit(nopython=True)  
def burning\_ship(xmin, xmax, ymin, ymax, width, height, max\_iter):  
 real = np.linspace(xmin, xmax, width)  
 imag = np.linspace(ymin, ymax, height)  
 div\_time = np.zeros((height, width), dtype=np.float64)  
  
 for i in range(height):  
 for j in range(width):  
 c = complex(real[j], imag[i])  
 z = complex(0, 0)  
 n = 0  
 while abs(z) <= 2 and n < max\_iter:  
 z = complex(abs(z.real), abs(z.imag))  
 z = z \* z + c  
 n += 1  
 if n < max\_iter:  
 div\_time[i, j] = n + 1 - np.log(np.log2(abs(z)))  
 else:  
 div\_time[i, j] = n  
 div\_time = div\_time / div\_time.max()  
 return div\_time  
  
  
def plot\_burning\_ship(fractal, xmin, xmax, ymin, ymax, max\_iter, cmap='inferno'):  
 plt.figure(figsize=(10, 10))  
 plt.imshow(fractal, extent=[xmin, xmax, ymin, ymax], cmap=cmap, interpolation='bilinear')  
 plt.colorbar(label='Iterations')  
 plt.title(f"Burning ship with max\_iter = {max\_iter}")  
 plt.xlabel("Re(z)")  
 plt.ylabel("Im(z)")  
 plt.show()  
  
  
def main():  
 xmin, xmax = -2.0, -1.7  
 ymin, ymax = -0.08, 0.025  
 width, height = 1000, 1000  
 max\_iter = 1000  
  
 fractal = burning\_ship(xmin, xmax, ymin, ymax, width, height, max\_iter)  
 plot\_burning\_ship(fractal, xmin, xmax, ymin, ymax, max\_iter)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

