Санкт-Петербургский Национальный Исследовательский Университет ИТМО

Рассчетно Графическая Работа №1
По теории функций комплексной переменной

Выполнили студенты:
Кузьмина Ольга Игоревна 412986
Садовой ГРИГОРИЙ Владимирович 368748
Хромов Даниил Тимофеевич 373336

Преподаватель:
Ким Эрик Евгеньевич

Санкт-Петербург 2024г.

1 Введение

В данной работе рассматриваются фрактальные множества, такие как множества Мандельброта и Жюлиа. Основная цель - изучение их свойств и построение визуализаций для разных параметров.

2 Доказательство свойств 1 и 2

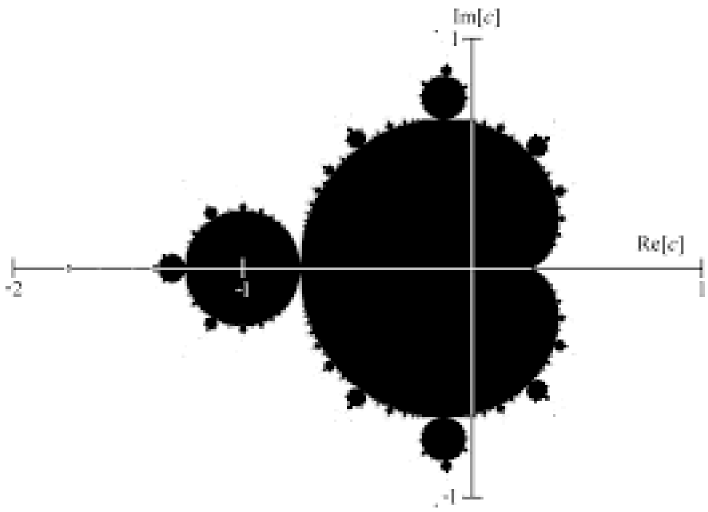
Докажите свойства 1 и 2 для множества Мандельброта.
Думаю, многие уже знакомы с множеством Мандельброта благодаря сводке для лабораторной работы. Давайте я продублирую свойства данного множества:

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

2. Если $|c| > 2$, то c не принадлежит множеству Мандельброта.

2.1 Доказательство 1:

И так начнём с определения множества Мандельброта.
Множество всех $c \in C$, при которых последовательность z_n остается ограниченной, называется множеством Мандельброта.
$z_n + 1 = z_n^2 + c, z_0 = 0$



Симметричность данного множества доказывается достаточно просто, во первых заметим:

1. Если взять точку w в комплексной плоскости и найти её комплексно-сопряжённую $\overline{w}$, то множество Мандельброта для этих точек окажется одинаковым, то есть симметричным относительно вещественной оси.

2. Орбита точки w — это последовательность значений, получающаяся при многократном применении формулы для построения множества Мандельброта. Показано, что орбита для w и орбита для $\overline{w}$ совпадают, что доказывает симметрию. Это означает, что если взять любой элемент орбиты w, его сопряжённый аналог будет элементом орбиты $\overline{w}$.

3. **Комплексное сопряжение полиномов**: Для доказательства симметрии используется свойство полиномов с вещественными коэффициентами. Комплексное сопряжение можно применять к таким полиномам без изменения их формы, что сохраняет симметрию множества Мандельброта.

4. **Ускорение вычислений**: Благодаря этой симметрии можно ускорить вычисления. Например, достаточно вычислить цвет одной точки, а цвет её симметричной копии можно просто скопировать, что упрощает работу алгоритма.

Орбита для w выглядит так:

$$\boldsymbol{O(w) = \langle 0, w, w + w^2, w + w^2 + 2w^3 + w^4, \dots \rangle}$$

А для $\overline{w}$ вот так:

$$\boldsymbol{O(\overline{w}) = \langle 0, \overline{w}, \overline{w} + \overline{w}^2, \overline{w} + \overline{w}^2 + 2\overline{w}^3 + \overline{w}^4, \dots \rangle}$$

- Что такое орбита на примере?

$$f(z) = z^2 + C$$

Для $C = 1 + 0i$:

$$\begin{aligned} f(0) &= 0^2 + 1 = 1, \\ f(1) &= 1^2 + 1 = 2, \\ f(2) &= 2^2 + 1 = 5, \\ f(5) &= 5^2 + 1 = 26 \\ &\vdots \\ &\to \infty \end{aligned}$$

Для $C = -1 + 0i$:

$$\begin{aligned} f(0) &= 0^2 + (-1) = -1, \\ f(-1) &= (-1)^2 + (-1) = 0, \\ f(0) &= 0^2 + (-1) = -1 \\ &\vdots \\ &\not\to \infty \end{aligned}$$

Поскольку все элементы орбиты $O(w)$ являются многочленами от w с вещественными коэффициентами, остаётся доказать, что комплексное сопряжение можно выносить за знак таких многочленов: $g(\overline{w}) = \overline{g(w)}$. Последний факт вытекает из двух легко проверяемых свойств комплексного сопряжения: $\overline{p \pm q} = \overline{p} \pm \overline{q}$ и $\overline{p * q} = \overline{p} * \overline{q}$ для любых комплексных чисел p и q. Итак, $O(\overline{w}) = \overline{O(w)}$, что и требовалось доказать.
Доказательство для $g(\overline{w}) = \overline{g(w)}$:
Воспользуемся двумя основными свойствами комплексного сопряжения:

1. $\overline{p \pm q} = \overline{p} \pm \overline{q}$

2. $\overline{p * q} = \overline{p} * \overline{q}$

Применим их к $\overline{g(w)}$:
$\overline{g(w)} = \overline{a_n w^n} + \overline{a(n-1)w(n-1)} + ... + \overline{a_1 w} + \overline{a_0}$
Так как $\overline{a_k w^k} = \overline{a_k}$, а $\overline{w^k} = \overline{w}^k$.

2.2 Доказательство 2:

Несложно доказать, что как только модуль z_n окажется больше 2, последовательность станет стремиться к бесконечности. При $|c| > 2$ точка c заведомо не принадлежит множеству Мандельброта, что можно вывести методом индукции, используя равенство $z_0 = 0$.
Начало итерации: Начнём с $z_0 = 0$. Тогда: $z_1 = z_0^2 + c = c$
Оценка второго члена последовательности: Рассмотрим z_2: $z_2 = z_1^2 + c = c^2 + c$
Здесь важно, что $|c| > 2$. В таком случае, $|c^2| = |c|^2 > 4$, и можно сделать предположение, что каждый последующий шаг будет увеличивать $|z_n|$.
Индуктивное утверждение
Покажем индукцией, что для всех $n \geq 1$ выполняется $|z_n| > |c|$. В частности, это будет означать, что $|z_n| \to \infty$.
Предположим, что для некоторого $n \geq 1$ выполняется $|z_n| > |c|$. Тогда:
$|z_n + 1| = |z_n^2 + c| \geq |z_n^2| - |c|$.
Так как $|z_n| > |c| > 2, |z_n^2| = |z_n|^2 > |c|^2$, и, следовательно, $|z_n + 1| > |c|^2 - |c| > |c|$.
Следовательно, $|z_n|$ неограниченно возрастает, так как с каждым шагом оно становится больше $|c|$.

3 Программная часть

С результатом программной реализации вы можете ознакомиться по следующей ссылке: Программка.
Если смотреть именно на реализацию, то можно выделить следующие моменты:

- **Множество Жюлиа**

```
function drawJuliaSet() {
    const width = canvas.width;
    const height = canvas.height;

    const imageData = ctx.createImageData(width, height);
    const data = imageData.data;

    const minRe = -2.0 / zoom + offsetX;
    const maxRe = 2.0 / zoom + offsetX;
    const minIm = -2.0 / zoom + offsetY;
    const maxIm = minIm + (maxRe - minRe) * height / width;

    const reFactor = (maxRe - minRe) / (width - 1);
    const imFactor = (maxIm - minIm) / (height - 1);
```

```
    for (let y = 0; y < height; y++) {
        const zIm = maxIm - y * imFactor;
        for (let x = 0; x < width; x++) {
            const zRe = minRe + x * reFactor;

            let Z_re = zRe, Z_im = zIm;
            let n = 0;
            let isInside = true;

            for (; n < maxIterations; n++) {
                const Z_re2 = Z_re * Z_re;
                const Z_im2 = Z_im * Z_im;

                if (Z_re2 + Z_im2 > 4) {
                    isInside = false;
                    break;
                }

                Z_im = 2 * Z_re * Z_im + cIm;
                Z_re = Z_re2 - Z_im2 + cRe;
            }

            const p = (y * width + x) * 4;
            if (isInside) {
                data[p] = 0;
                data[p + 1] = 0;
                data[p + 2] = 0;
            } else {
                const t = n / maxIterations;
                const color = getColor(t);

                data[p] = color.r;
                data[p + 1] = color.g;
                data[p + 2] = color.b;
            }
            data[p + 3] = 255;
        }
    }

    ctx.putImageData(imageData, 0, 0);
}
```

Здесь записан алгоритм по которому строиться наша визуализация множества.

- **Множество Мандельброта**

```
function drawMandelbrot() {
```

```
const width = canvas.width;
const height = canvas.height;

const imageData = ctx.createImageData(width, height);
const data = imageData.data;

const minRe = -2.0 / zoom + offsetX;
const maxRe = 1.0 / zoom + offsetX;
const minIm = -1.5 / zoom + offsetY;
const maxIm = minIm + (maxRe - minRe) * height / width;

const reFactor = (maxRe - minRe) / (width - 1);
const imFactor = (maxIm - minIm) / (height - 1);

for (let y = 0; y < height; y++) {
    const cIm = maxIm - y * imFactor;
    for (let x = 0; x < width; x++) {
        const cRe = minRe + x * reFactor;

        let Z_re = 0, Z_im = 0;
        let n = 0;
        let isInside = true;

        for (; n < maxIterations; n++) {
            const Z_re2 = Z_re * Z_re;
            const Z_im2 = Z_im * Z_im;

            if (Z_re2 + Z_im2 > 4) {
                isInside = false;
                break;
            }

            Z_im = 2 * Z_re * Z_im + cIm;
            Z_re = Z_re2 - Z_im2 + cRe;
        }

        const p = (y * width + x) * 4;
        if (isInside) {
            data[p] = 0;
            data[p + 1] = 0;
            data[p + 2] = 0;
        } else {
            const t = n / maxIterations;
            const color = getColor(t);

            data[p] = color.r;
            data[p + 1] = color.g;
```

```
                data[p + 2] = color.b;
            }
            data[p + 3] = 255;
        }
    }

    ctx.putImageData(imageData, 0, 0);
}
```

А в данном фрагменте кода, для множества Мандельброта.

4 Результаты визуализации

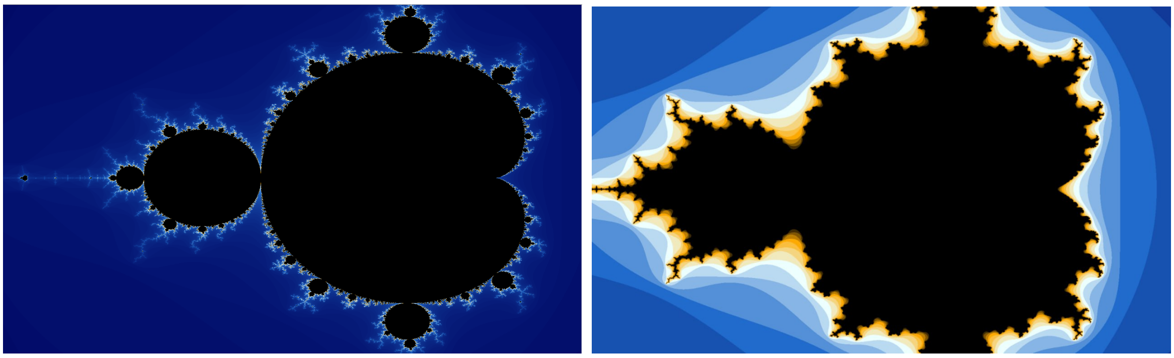


Рис. 1: Визуализации множества Мандельброта

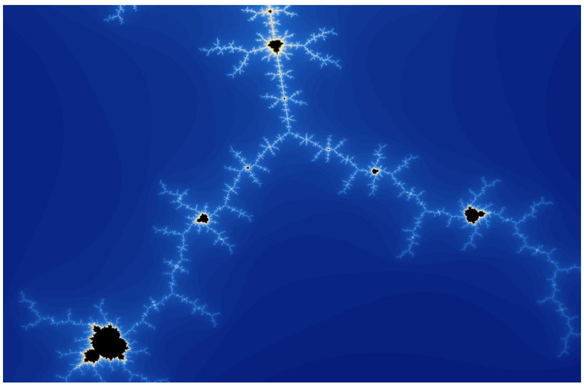


Рис. 2: Визуализации множества Мандельброта (увеличенная)

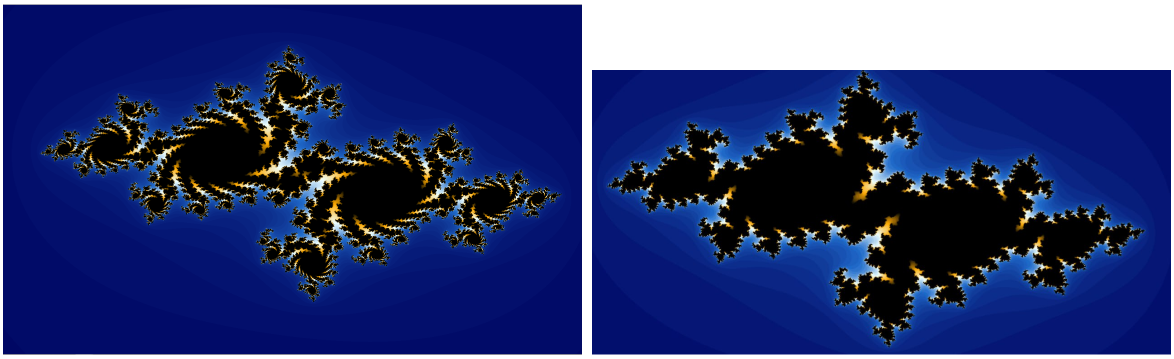


Рис. 3: Визуализации множества Жюлиа

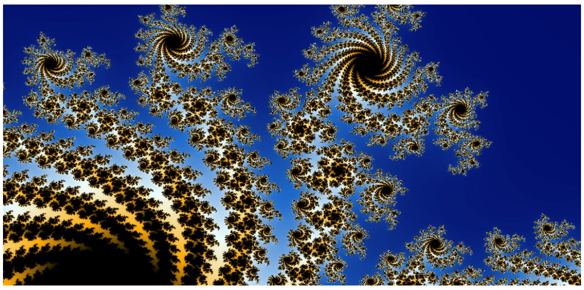


Рис. 4: Визуализации множества Жюлиа (увеличенная)

5 Изучение другого фрактала

5.1 Описание структуры фрактала Бассейны Ньютона

Метод Ньютона используется для нахождения корней функции $f(z)$. Если мы применим метод Ньютона к комплексной функции, например, $f(z) = z^3 - 1$, то корнями этого уравнения будут три корня кубического корня из единицы. Эти корни являются аттракторами, то есть точками, к которым будут сходиться точки на плоскости при итеративном применении метода Ньютона.
Пример процесса: Для функции $f(z) = z^3 - 1$:

1. **Начальное значение**: Каждая точка комплексной плоскости служит начальным значением для применения метода Ньютона.

2. **Итерация:** Метод Ньютона выполняет итерации по формуле:

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

   где $f'(z)$ — производная функции.

3. **Аттракторы**: После нескольких итераций z_n сходится к одному из корней функции $f(z) = 0$. При этом область начальных точек, которые сходятся к одному и тому же корню, формирует бассейн Ньютона.

5.2 Программная реализация

Теперь мы можем посмотреть на код для визуализации нашего фрактала:

```python
from manim import *

class NewtonBasins(Scene):
    def construct(self):
        x_min, x_max = -2, 2
        y_min, y_max = -2, 2
        max_iter = 30
        resolution = 400

        image = ImageMobject(self.newton_basins_image(x_min, x_max, y_min, y_max, max_it
        image.set_height(config.frame_height)
        self.add(image)

    def newton_basins_image(self, x_min, x_max, y_min, y_max, max_iter, resolution):
        import numpy as np
        from PIL import Image

        roots = [
            1 + 0j,
            -0.5 + 0.86602540378j,
            -0.5 - 0.86602540378j
        ]

        pixels = np.zeros((resolution, resolution, 3), dtype=np.uint8)

        for i in range(resolution):
            for j in range(resolution):
                x = x_min + (x_max - x_min) * i / (resolution - 1)
                y = y_min + (y_max - y_min) * j / (resolution - 1)
                z = complex(x, y)

                for n in range(max_iter):
                    if abs(z) > 1e-10:
                        z = z - (z**3 - 1) / (3 * z**2)
                    else:
                        break

                    for k, root in enumerate(roots):
                        if abs(z - root) < 1e-3:
                            color = [0, 0, 0]
                            color[k] = 255
                            pixels[j, i] = color
                            break
                    else:
                        continue
                    break
```

```
    return Image.fromarray(pixels)
```

Пояснение к коду:

- Корни уравнения: Мы заранее вычисляем корни для $z^3 - 1 = 0$, которые будут аттракторами.
- Метод Ньютона: Итеративно применяем формулу: $z_{n+1} = z_n - \frac{f(z_n)}{f'(z_n)}$.
- Окрашивание пикселей: Каждый пиксель окрашивается в зависимости от того, к какому корню он приближается. Например:
  - Красный цвет — первый корень.
  - Зеленый цвет — второй корень.
  - Синий цвет — третий корень.

5.3 Свойства

1. **Фрактальная структура:** Если окрасить каждую начальную точку в зависимости от того, к какому корню она стремится, получится фрактальное изображение.
2. **Зависимость от начальных условий:** Бассейны Ньютона чувствительны к начальным условиям, поэтому небольшие изменения в координатах могут привести к разному поведению, что и создает фрактальный эффект.

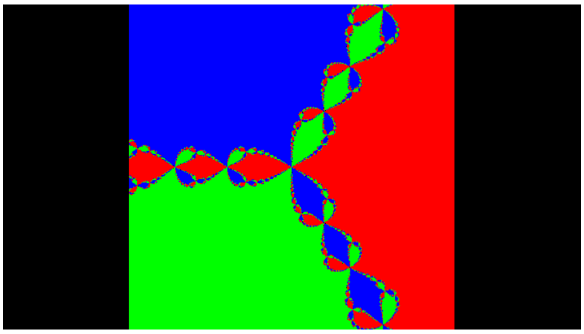


Рис. 5: Визуализация бассейнов Ньютона

Заключение

В данной работе были изучены фрактальные множества, такие как Мандельброта и Жюлиа. Программа для их визуализации реализована, а результаты представлены в виде изображений, иллюстрирующих фрактальную структуру множеств.