ФГАОУ ВО «НИУ ИТМО»
Факультет программной инженерии
и компьютерной техники

# Лабораторная работа №2

## Построение конформных отображений

(по дисциплине «Теория функций комплексного переменного»)

**Выполнил:**

Лабин Макар Андреевич,
ТФКП 22.4, P3231, 466449.

**Проверил:**

Поздняков Семён Сергеевич

г. Санкт-Петербург, Россия
2025

# 1 Постановка задач

В ходе лабораторной работы *по варианту №15* будут выполнены следующие задачи:

1. Аналитически описать заданные множества *(множества на рисунке закрашены штриховкой, а все граничные точки подразумеваются не принадлежащими им)*.

2. Воспользовавшись композицией классических преобразований, составить конформное отображение, которое переводит первую область во вторую.

3. Составить обратное отображение, переводящее второе множество в первое.

4. На любом удобном языке программирования написать программу, которая изобразит первое множество и все этапы его преобразования во второе. Достаточно наглядным будет взять набор точек множества, передающий его форму *(может понадобится сделать набор «более плотным» в какой-то части множества)*.

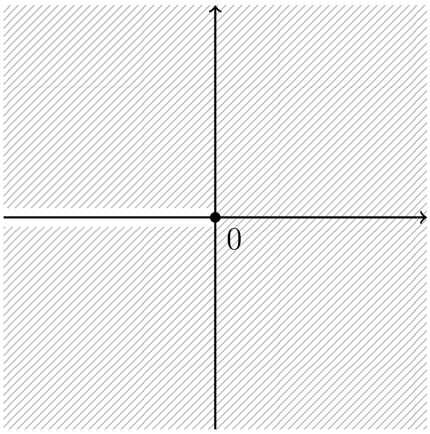



Рисунок 4

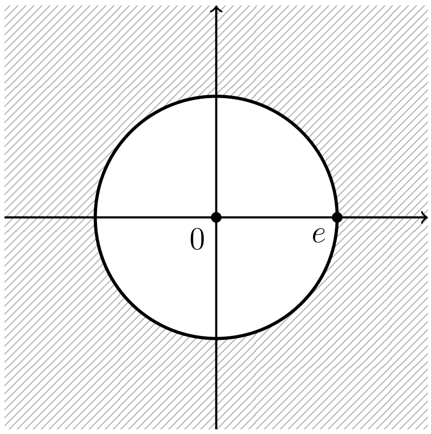
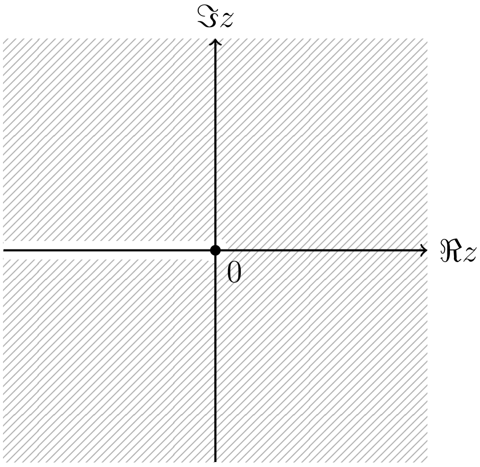


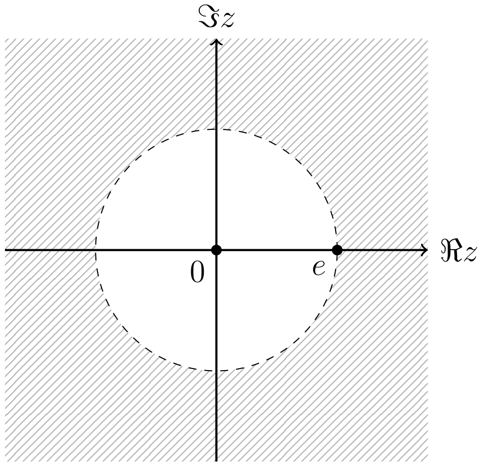

Рисунок 9

# 2 Аналитическое описание множеств

Даны изображения множеств $P$ и $Q$ на комплексной плоскости $\mathbb{C}$:




Множество $P' \subset \mathbb{C}$




Множество $Q' \subset \mathbb{C}$

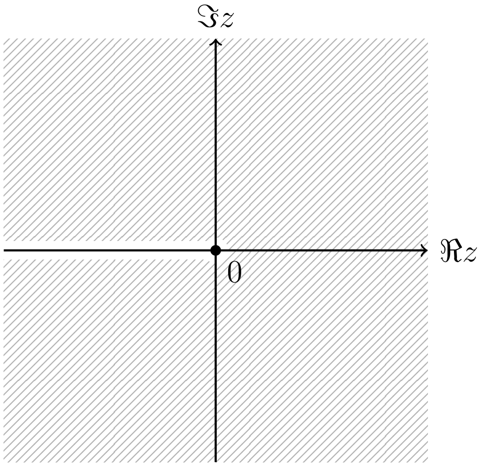
Видно, что множество $P$ — комплексная плоскость с разрезом по действительному лучу $(-\infty, 0]$. Множество $Q$, в свою очередь, — внешняя часть комплексной окружности радиуса $e$:

$$P = \{z \in \mathbb{C} \,|\, \Re z > 0 \vee \Im z \neq 0\} \qquad Q = \{z \in \mathbb{C} \,|\, |z| > e\}$$

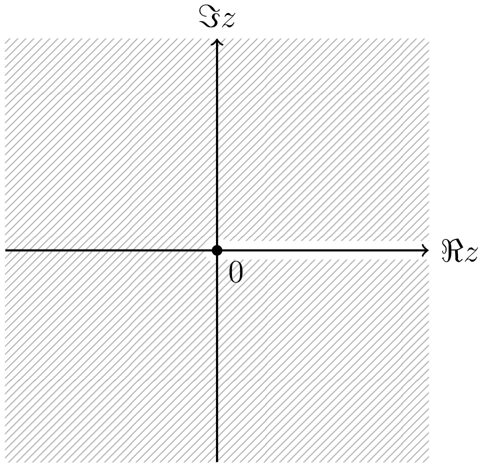
# 3 Конформные отображения множеств

Составим конформное отображение $\omega(z)$, переводящее множество $P$ в $Q$:

1. Применим к $P$ конформное отображение $\omega_1(z) = -z$:

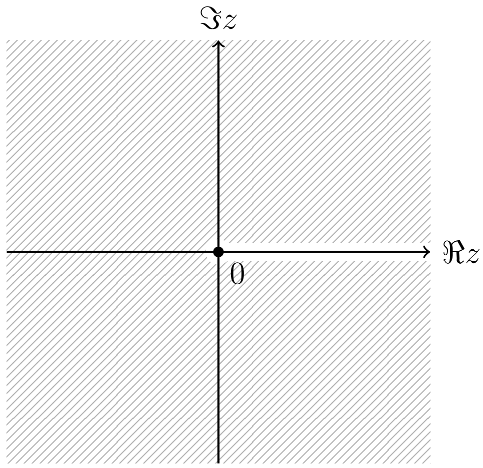



Множество $P$

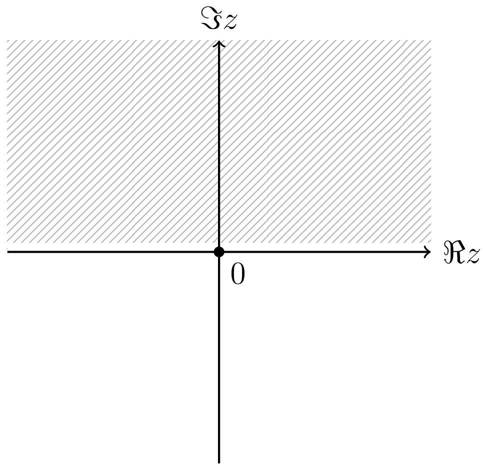



Множество $P_1 = \omega_1(P)$

2. Применим к $P_1$ конформное отображение $\omega_2(z) = \sqrt{z}$:

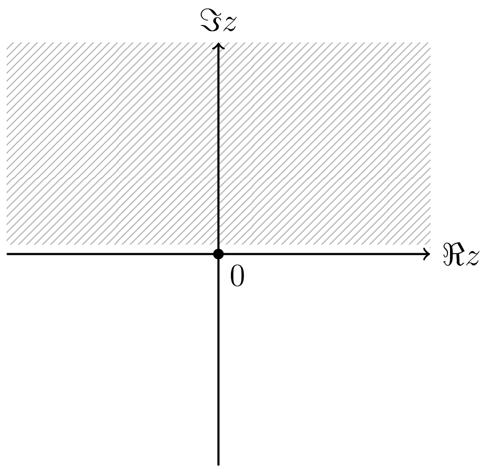



Множество $P_1$

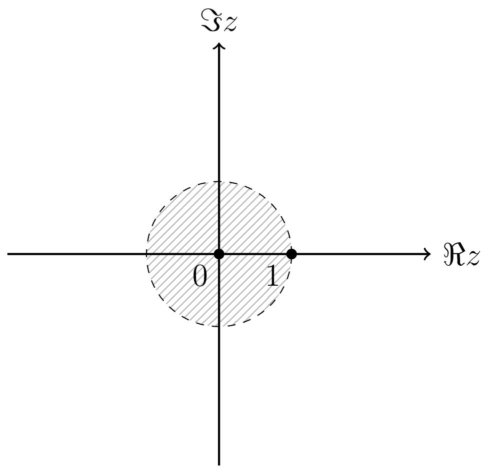



Множество $P_2 = \omega_2(P_1)$

3. Применим к $P_2$ конформное отображение $\omega_3(z) = \frac{z-i}{z+i}$:

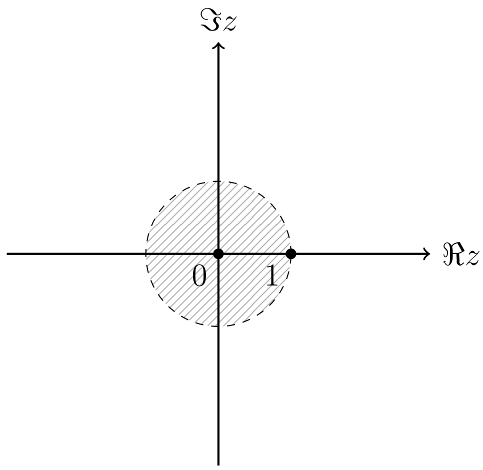



Множество $P_2$

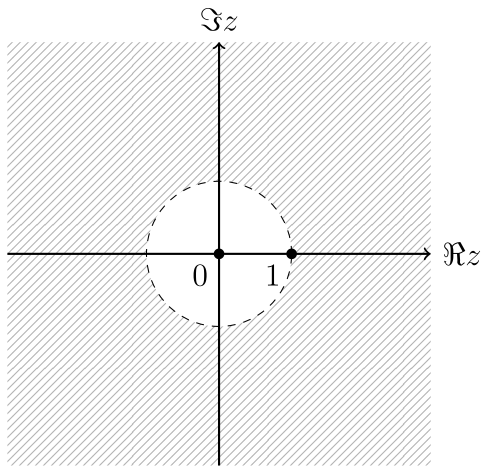



Множество $P_3 = \omega_3(P_2)$

4. Применим к $P_3$ конформное отображение $\omega_4(z) = \frac{1}{z}$:

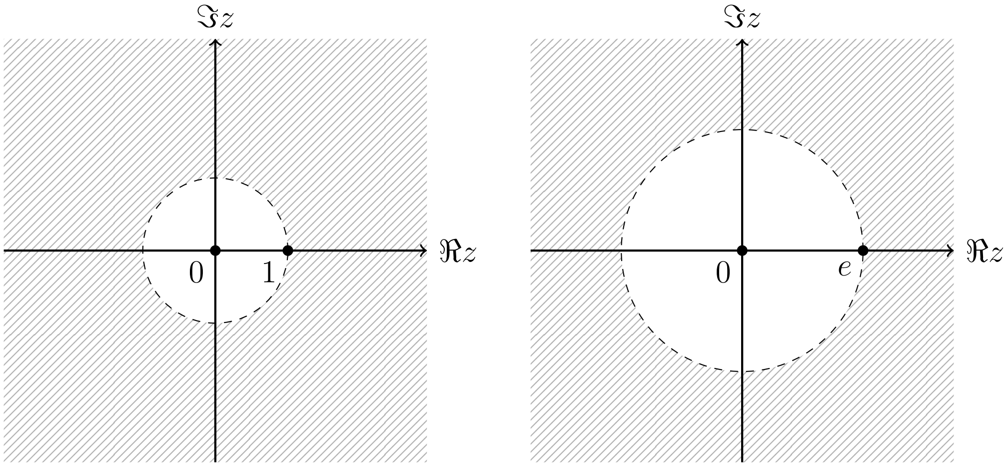



Множество $P_3$




Множество $P_4 = \omega_4(P_3)$

5. Применим к $P_4$ конформное отображение $\omega_5(z) = ez$:



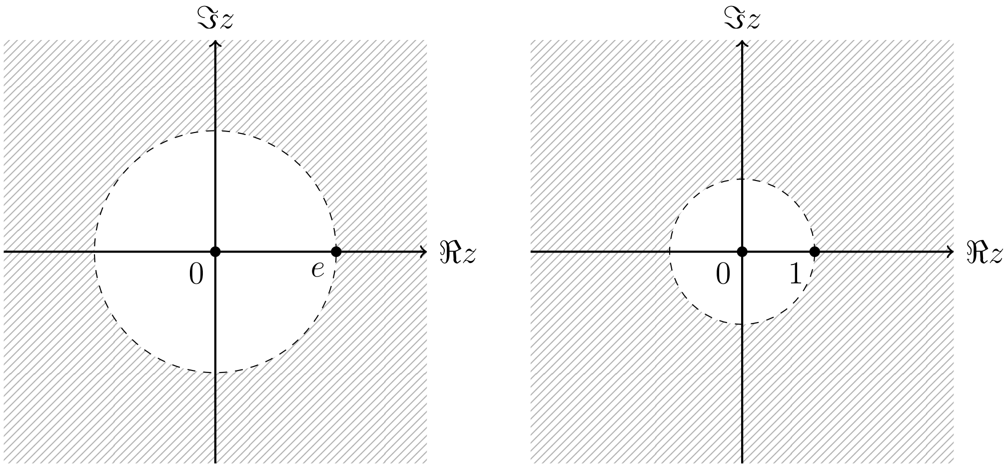

Множество $P_4$ Множество $P_5 = \omega_5(P_4) = Q$

Итого $\omega(z)$ имеет вид:

$$\omega = \omega_5 \circ \omega_4 \circ \omega_3 \circ \omega_2 \circ \omega_1 \implies \omega(z) = e \cdot \frac{\sqrt{-z} + i}{\sqrt{-z} - i}$$

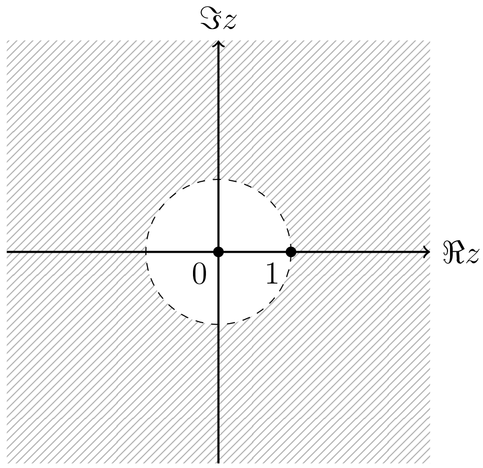
Составим обратное конформное отображение $z(\omega)$, которое переводит множество $Q$ в $P$:

1. Применим к $Q$ конформное отображение $z_1(\omega) = \frac{\omega}{e}$:

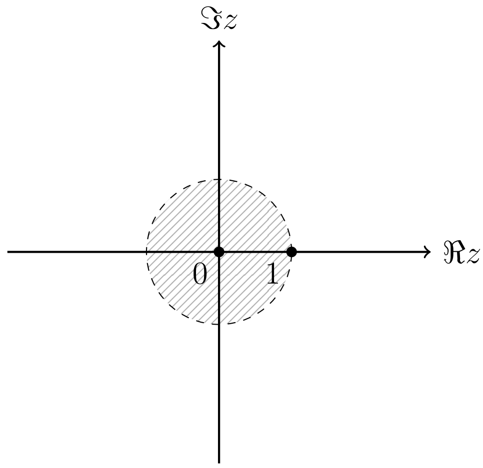



Множество $Q$ Множество $Q_1 = z_1(Q)$

2. Применим к $Q_1$ конформное отображение $z_2(\omega) = \frac{1}{\omega}$:

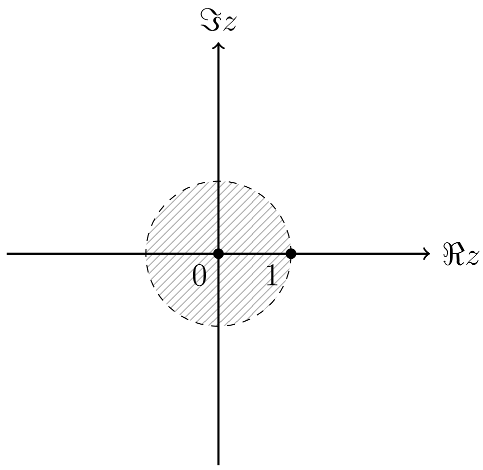



Множество $Q_1$

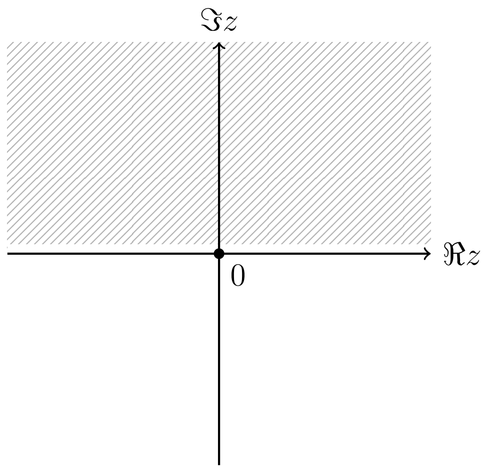



Множество $Q_2 = z_2(Q_1)$

3. Применим к $Q_2$ конформное отображение $z_3(\omega) = i \cdot \frac{1+\omega}{1-\omega}$:

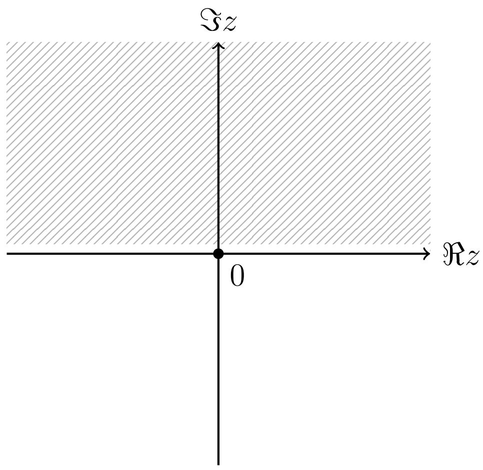



Множество $Q_2$

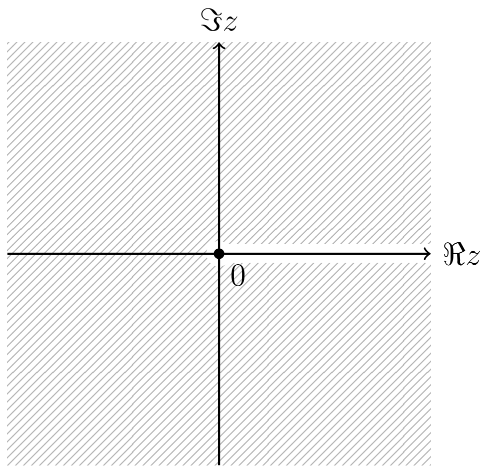



Множество $Q_3 = z_3(Q_2)$

4. Применим к $Q_3$ конформное отображение $z_4(\omega) = \omega^2$:

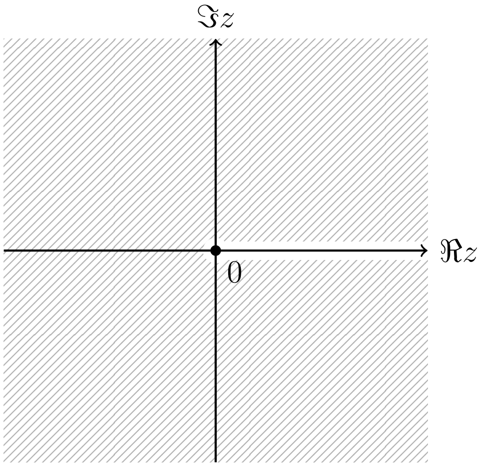



Множество $Q_3$

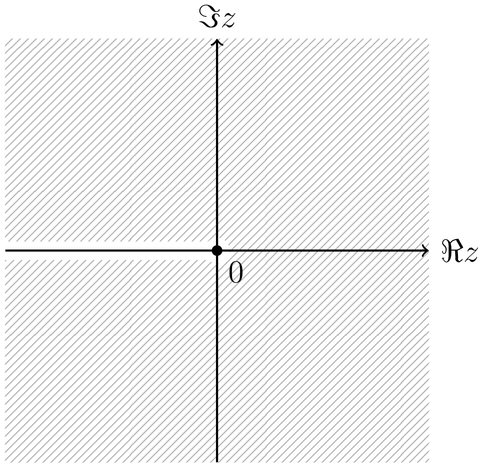



Множество $Q_4 = z_4(Q_3)$

5. Применим к $Q_4$ конформное отображение $z_5(\omega) = -\omega$:




Множество $Q_4$




Множество $Q_5 = z_5(Q_4) = P$

Итого $z(\omega)$ имеет вид:

$$z = z_5 \circ z_4 \circ z_3 \circ z_2 \circ z_1 \implies z(\omega) = -\left(i \cdot \frac{1 + \frac{e}{\omega}}{1 - \frac{e}{\omega}}\right)^2$$

# 4 Скрипты для создания графики

Чтобы изобразить множество $P$ и все его промежуточные образы, воспользуемся графическим пакетом *TikZ*. Для генерации необходимых файлов напишем скрипты на языке *Python*.

Для генерации точек, описывающих множество $P$, ограничим $|z| \leq 3$. Ещё для наглядности разделим комплексную плоскость на подмножества и окрасим их градиентами, которые меняются от близости к началу координат. Более того, используем логарифмическое шкалирование радиусов:

| | **Arg** $z$ | **Градиент** |
|---|---|---|
| 1. | $\left(-\pi, -\dfrac{3\pi}{4}\right)$ |  |
| 2. | $\left[-\dfrac{3\pi}{4}, 0\right)$ |  |
| 3. | $\left[0, \dfrac{3\pi}{4}\right]$ |  |
| 4. | $\left(\dfrac{3\pi}{4}, \pi\right)$ |  |

Листинг 1: Скрипт `generate_dots.py`

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

```python

# Parameters for point generation
epsilon = 0.02  # Small number to exclude boundary values
density = 50    # Parameter affecting point density

def complex_positive(density):
    """
    Generates points in the sector (3pi/4, pi) of the complex plane.
    Points are distributed in logarithmic scale by radius.
    """
    angles_far = np.linspace(np.pi - np.pi / 4 + epsilon, np.pi - epsilon, density // 9)
    radius_far = np.exp(np.linspace(np.log(0.005), np.log(3), density // 2))
    r_grid, theta_grid = np.meshgrid(radius_far, angles_far)
    z = r_grid * np.exp(1j * theta_grid)
    points = z.flatten()
    distances = np.abs(points)
    colors = distances / np.max(distances)  # Normalized distances for coloring
    return points, colors

def complex_negative(density):
    """
    Generates points in the sector (-pi, -3pi/4) of the complex plane.
    """
    angles_center = np.linspace(-np.pi + epsilon, -np.pi + np.pi / 4 - epsilon, density // 9)
    radius_center = np.exp(np.linspace(np.log(0.005), np.log(3), density // 2))
    r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
    z = r_grid * np.exp(1j * theta_grid)
    points = z.flatten()
    distances = np.abs(points)
    colors = distances / np.max(distances)
    return points, colors

def complex_far_positive(density):
    """
    Generates points in the sector [0, 3pi/4] of the complex plane.
    """
    angles_center = np.linspace(0, np.pi - np.pi/4, density // 4)
    radius_center = np.exp(np.linspace(np.log(0.005), np.log(3), density // 2))
    r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
    z = r_grid * np.exp(1j * theta_grid)
    points = z.flatten()
    distances = np.abs(points)
```

```
    colors = distances / np.max(distances)
    return points, colors

def complex_far_negative(density):
    """
    Generates points in the sector [-3pi/4, 0) of the complex
       plane.
    """
    angles_center = np.linspace(-np.pi + np.pi/4, -epsilon,
       density // 4)
    radius_center = np.exp(np.linspace(np.log(0.005), np.log(3),
       density // 2))
    r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
    z = r_grid * np.exp(1j * theta_grid)
    points = z.flatten()
    distances = np.abs(points)
    colors = distances / np.max(distances)
    return points, colors

def get_P(plane):
    """
    Filters points: keeps only those with Re(z) > 0
    or Im(z) != 0 (excludes non-positive real axis).
    """
    mask = np.logical_or(plane.real > 0, plane.imag != 0)
    return plane[mask]

# Definition of mapping functions omega_1, ..., omega_5
def omega_1(z):
    """omega_1(z) = -z (central symmetry about origin)"""
    return -z

def omega_2(z):
    """omega_2(z) = sqrt(z) (square root extraction)"""
    r = np.abs(z)
    theta = np.angle(z)
    sqrt_r = np.sqrt(r)
    theta_pos = np.mod(theta, 2*np.pi)  # Bring angle to [0, 2*pi)
    new_theta = theta_pos / 2
    return sqrt_r * (np.cos(new_theta) + 1j * np.sin(new_theta))

def omega_3(z):
    """omega_3(z) = (z - i)/(z + i) (fractional-linear
       transformation)"""
    return (z-1j)/(z+1j)

def omega_4(z):
    """omega_4(z) = 1/z (inversion)"""
    return 1/z

def omega_5(z):
```

```python
    """omega_5(z) = z * e (multiplication by Euler's number e)"""
    return z * np.e

def get_all_Ps(complex_region):
    """
    Applies all five mappings sequentially to input set.
    """
    P = get_P(complex_region)
    P1 = omega_1(P)
    P2 = omega_2(P1)
    P3 = omega_3(P2)
    P4 = omega_4(P3)
    P5 = omega_5(P4)
    return [P, P1, P2, P3, P4, P5]

def scatter_regions(regions, axes, colors, cmap):
    """
    Displays 6 sets on 3x2 grid of plots.

    Parameters:
    regions: list of 6 arrays of complex numbers
    axes: 3x2 matrix of axes
    colors: array of values for colormap
    cmap: name of colormap
    """
    if len(regions) < 6:
      print("Error: lack of regions")
      pass
    for y in range(3):
      for x in range(2):
        axes[y, x].scatter(regions[y * 2 + x].real, regions[y * 2
            + x].imag,
                            s=1, alpha=0.6, c=colors, cmap=cmap)

if __name__ == "__main__":
    fig, ax = plt.subplots(3, 2)

    # Generate and display points from four sectors with different
        colormaps
    reg1, col1 = complex_far_positive(density)
    scatter_regions(get_all_Ps(reg1), ax, col1, 'winter')

    reg2, col2 = complex_far_negative(density)
    scatter_regions(get_all_Ps(reg2), ax, col2, 'copper')

    reg3, col3 = complex_positive(density)
    scatter_regions(get_all_Ps(reg3), ax, col3, 'cool')

    reg4, col4 = complex_negative(density)
    scatter_regions(get_all_Ps(reg4), ax, col4, 'autumn')

```

```python
    # Save data from each plot to separate file
    for i in range(6):
        row = i // 2
        col = i % 2

        with open(f'step_{i+1}_data.txt', 'w') as f:
            # Get all point collections from the plot
            for collection_idx, collection in enumerate(ax[row, col].collections):
                data = collection.get_offsets()  # Coordinates (x, y)
                colors = collection.get_array()   # Values for colormap
                cmap_name = collection.get_cmap().name  # Name of colormap

                # Write data to file: x, y, color value, colormap name
                for j in range(len(data)):
                    x, y = data[j]
                    color_val = colors[j] if j < len(colors) else 0.0
                    f.write(f'{x:.6f} {y:.6f} {color_val:.6f} {cmap_name}\n')

        print(f"Step {i+1}: data saved")
```

Для транспиляции окрашенных наборов точек в графики *TikZ* воспользуемся скриптом:

Листинг 2: Скрипт `tikzify.py`

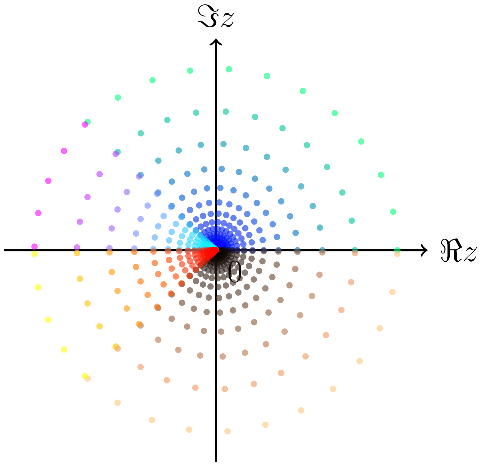
```python
import numpy as np
import matplotlib as plt
import os

def get_tikz_color_with_alpha(color_val, colormap, alpha=1.0):
    """
    Convert matplotlib colormap value to TikZ color with alpha transparency.

    Parameters:
    color_val: normalized color value [0, 1]
    colormap: name of matplotlib colormap
    alpha: opacity value [0, 1]
    """
    cmap = plt.colormaps[colormap]
    rgba = cmap(color_val)

    # Convert rgba values (0-1) to rgb (0-255) for TikZ
    r = int(rgba[0] * 255)
    g = int(rgba[1] * 255)
    b = int(rgba[2] * 255)
    a = rgba[3] * alpha  # Use alpha from colormap if available

    return f"{{rgb,255:red,{r}; green,{g}; blue,{b}}}, opacity={a:.2f}"
```

```python

def convert_to_tikz(input_file, output_file, step_num):
    """
    Convert data file with point coordinates to TikZ code.

    Parameters:
    input_file: text file with columns: x, y, color_value, colormap_name
    output_file: output .tex file with TikZ code
    step_num: ordinal numeral of current step
    """
    data = np.loadtxt(input_file, dtype=object)

    # Validate data format
    if data.shape[1] != 4:
        raise ValueError("File must have 4 columns: x, y, color, colormap")

    # Extract and convert data columns
    points = data[:, :2].astype(float)
    colors = data[:, 2].astype(float)
    cmaps = data[:, 3]

    # Normalize color values to [0, 1] range
    colors_min = colors.min()
    colors_max = colors.max()
    colors_norm = (colors - colors_min) / (colors_max - colors_min)

    # Initialize TikZ code with axes and origin point
    tikz_code = r"""\begin{tikzpicture}[scale=0.8]

\draw[->, thick] (-3.5,0) -- (3.5,0) node[right] {$\Re z$};
\draw[->, thick] (0,-3.5) -- (0,3.5) node[above] {$\Im z$};

\fill (0,0) circle (2.5pt) node[below right] {$0$};
"""
    match step_num:
        case '4' | '5':
            tikz_code += "\\fill (1,0) circle (2.5pt) node[below left] {$1$};\n"
        case '6':
            tikz_code += f"\\fill ({np.e},0) circle (2.5pt) node[below left] {{$e$}};\n"
        case _:
            tikz_code += "\n"

    count = 0
    for (x, y), color_val, cmap in zip(points, colors_norm, cmaps):
        # Skip points outside the plotting area
```

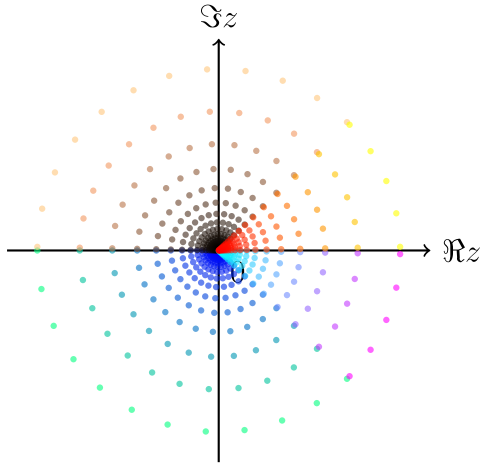
```
        if abs(x) > 3.5 or abs(y) > 3.5:
            continue

        count += 1
        tikz_color = get_tikz_color_with_alpha(color_val, cmap, alpha=0.6)
        tikz_code += f"\\fill[color={tikz_color}] ({x:.4f},{y:.4f}) circle (1.5pt);\n"

    tikz_code += "\n\\end{tikzpicture}"

    with open(output_file, 'w') as f:
        f.write(tikz_code)

    print(f"Saved: {output_file} ({count} points)")

if __name__ == "__main__":
    # Find all data files starting with 'step_' and ending with '_data.txt'
    data_files = [f for f in os.listdir() if f.startswith('step_') and f.endswith('_data.txt')]

    for data_file in data_files:
        # Extract step number from filename (e.g., 'step_1_data.txt' -> '1')
        step_num = data_file.split('_')[1]
        output_file = f'step_{step_num}_tikz.tex'

        convert_to_tikz(data_file, output_file, step_num)
```

# 5 Результаты отображений множества

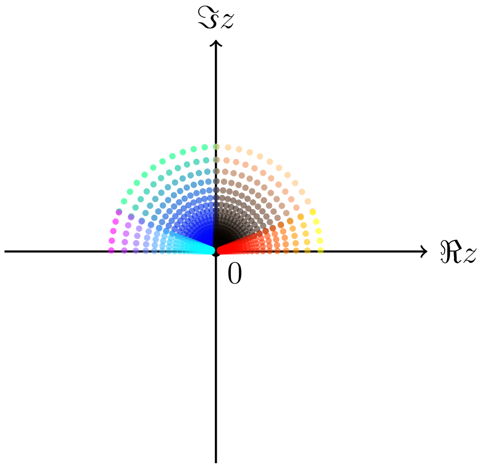
В результате работы вышеуказанных скриптов были получены следующие множества:



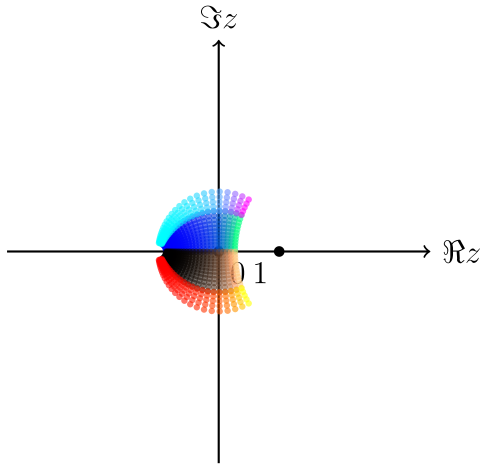

Множество $P$



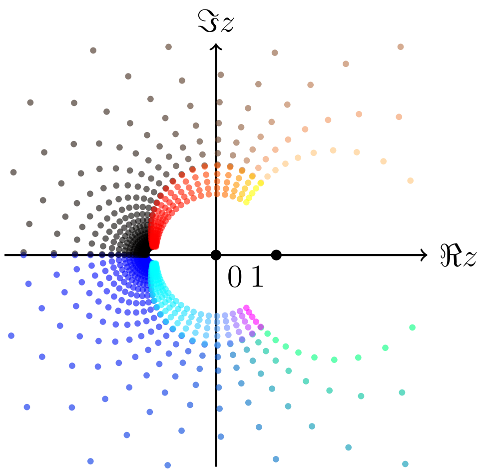

Множество $P_1 = \omega_1(P)$



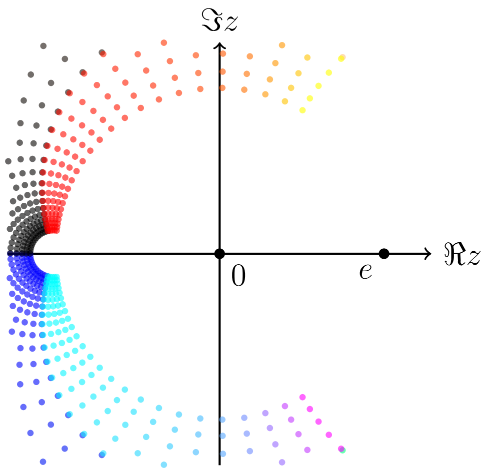

Множество $P_2 = \omega_2(P_1)$




Множество $P_3 = \omega_3(P_2)$




Множество $P_4 = \omega_4(P_3)$




Множество $P_5 = \omega_5(P_4) = Q$

# 6 Заключение

В ходе выполнения лабораторной работы была исследована задача конформного отображения заданных областей комплексной плоскости:

1. Было дано аналитическое описание исходного множества $P$ *(комплексная плоскость с разрезом по действительному лучу* $(-\infty, 0]$*)* и целевого множества $Q$ *(внешность окружности радиуса* $e$*)*.

2. Построено конформное отображение

$$\omega(z) = e \cdot \frac{\sqrt{-z} + i}{\sqrt{-z} - i},$$

   представляющее собой композицию пяти элементарных преобразований:

   - центральная симметрия,
   - взятие квадратного корня,
   - дробно-линейное преобразование,
   - инверсия,
   - растяжение на константу $e$.

3. Построено обратное отображение

$$z(\omega) = -\left(i \cdot \frac{1 + \frac{e}{\omega}}{1 - \frac{e}{\omega}}\right)^2,$$

   переводящее множество $Q$ обратно в $P$.

4. Реализован набор скриптов на Python, включающий:

   - генерацию точек исходного множества с учётом его структуры и окраской в зависимости от аргумента и модуля,
   - последовательное применение всех пяти отображений,
   - визуализацию каждого этапа преобразования,
   - экспорт данных в форматы, пригодные для построения графиков с помощью TikZ.

5. Получены наглядные графики всех промежуточных образов, подтверждающие корректность построенных отображений и сохранение конформности на каждом шаге.

Работа демонстрирует практическое применение теории конформных отображений для преобразования областей комплексной плоскости, а также возможность автоматизации процесса визуализации с помощью скриптов.