

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, Р3231, 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*).

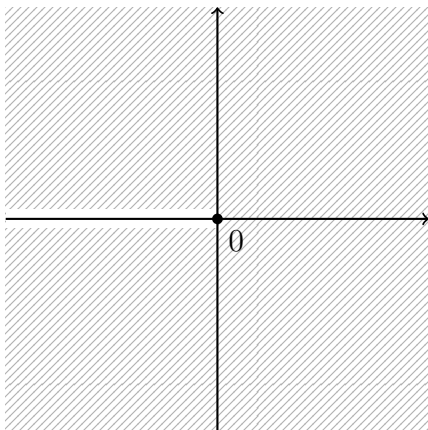


Рисунок 4

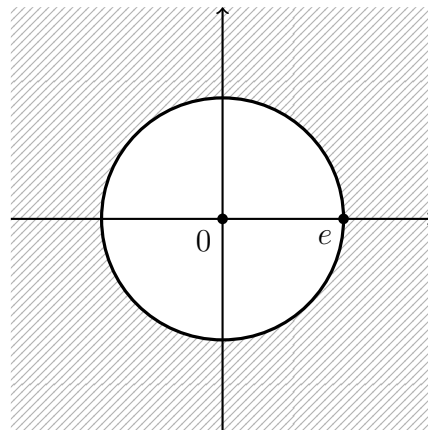
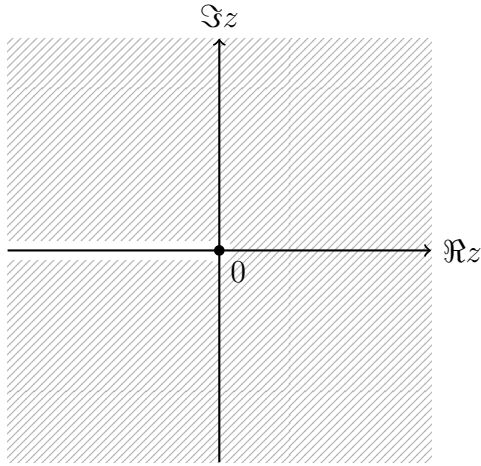


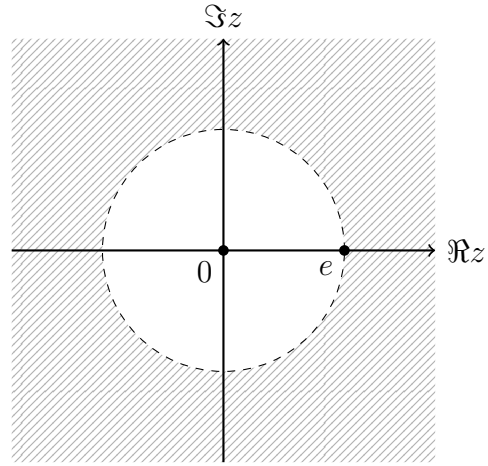
Рисунок 9

## 2 Аналитическое описание множеств

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



Множество  $P' \subset \mathbb{C}$



Множество  $Q' \subset \mathbb{C}$

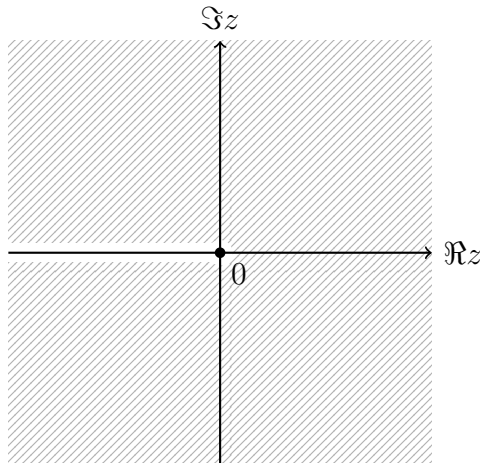
Видно, что множество  $P$  — комплексная плоскость с разрезом по действительному лучу  $(-\infty, 0]$ . Множество  $Q$ , в свою очередь, — внешняя часть комплексной окружности радиуса  $e$ :

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

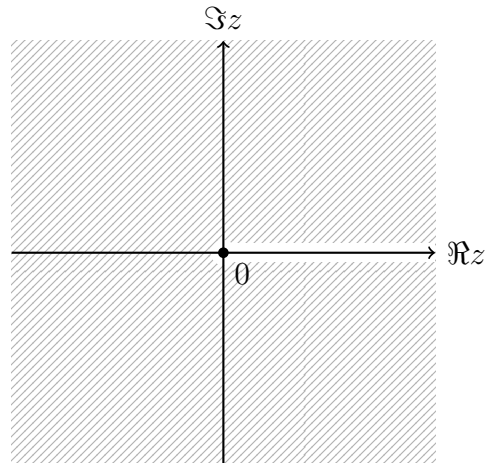
## 3 Конформные отображения множеств

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

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

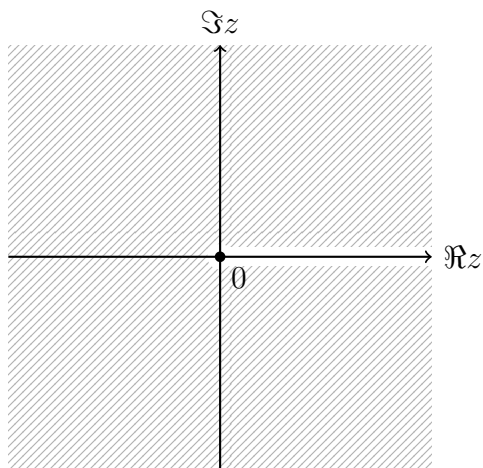


Множество  $P$

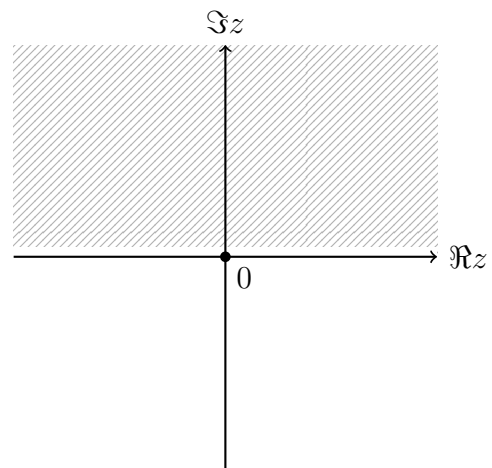


Множество  $P_1 = \omega_1(P)$

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

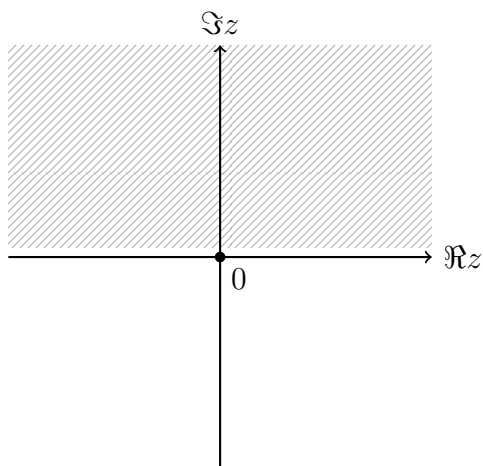


Множество  $P_1$

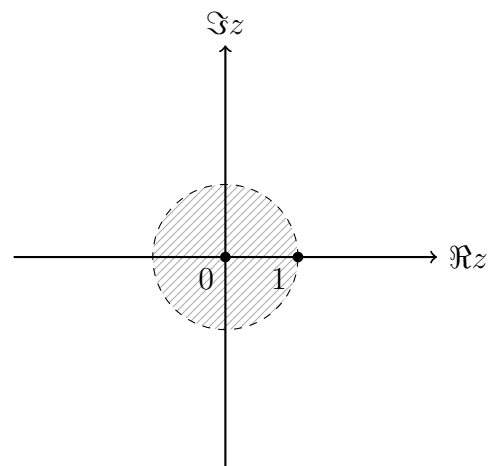


Множество  $P_2 = \omega_2(P_1)$

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

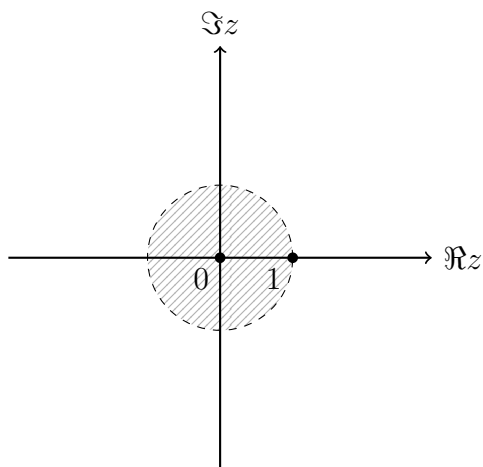


Множество  $P_2$

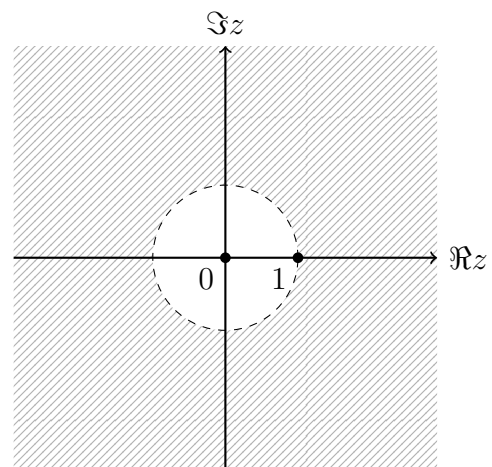


Множество  $P_3 = \omega_3(P_2)$

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

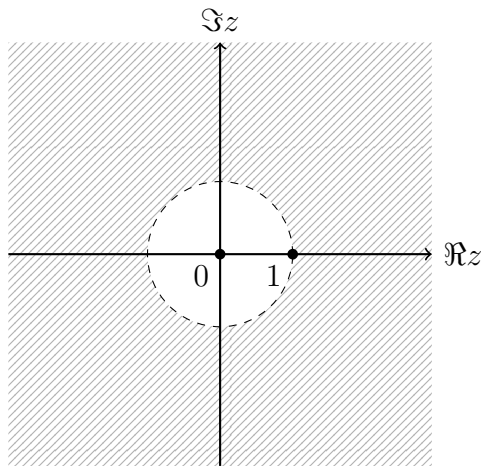


Множество  $P_3$

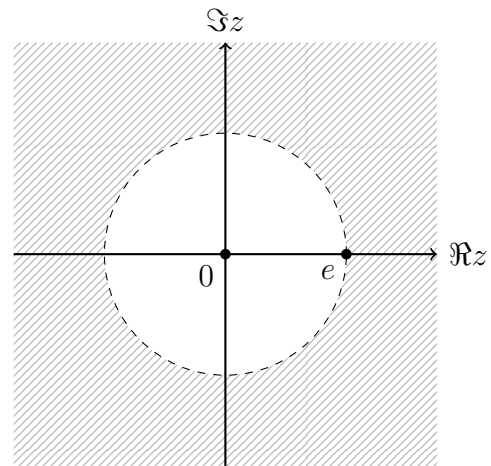


Множество  $P_4 = \omega_4(P_3)$

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



Множество  $P_4$



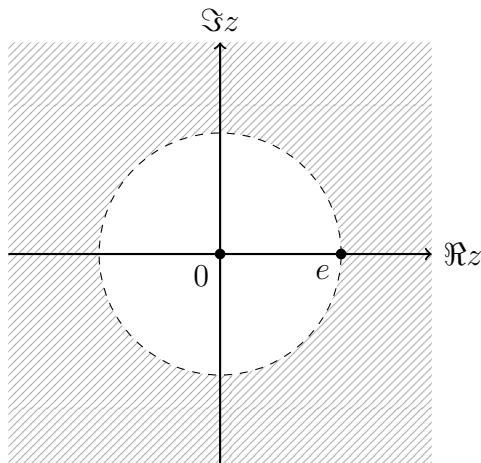
Множество  $P_5 = \omega_5(P_4) = Q$

Итого  $\omega(z)$  имеет вид:

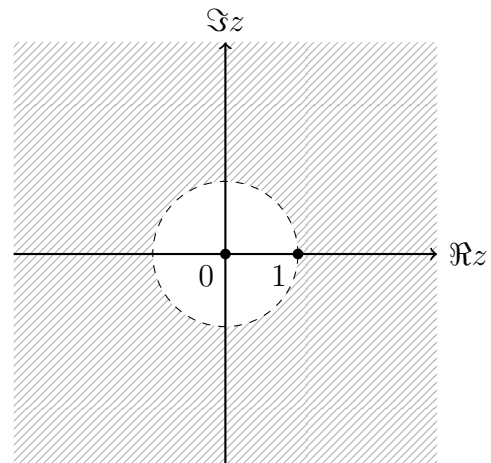
$$\omega = \omega_5 \circ \omega_4 \circ \omega_3 \circ \omega_2 \circ \omega_1 \implies \omega(z) = e \cdot \frac{\sqrt{-z} + i}{\sqrt{-z} - i}$$

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

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

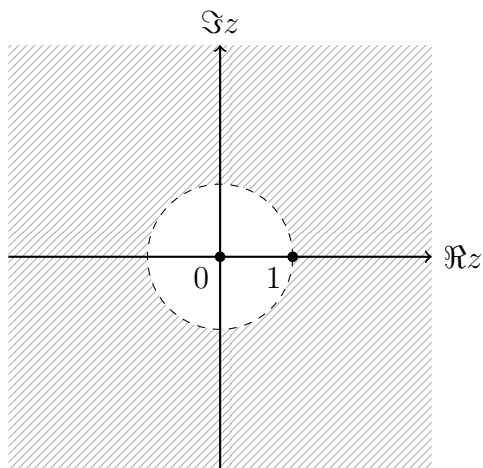


Множество  $Q$

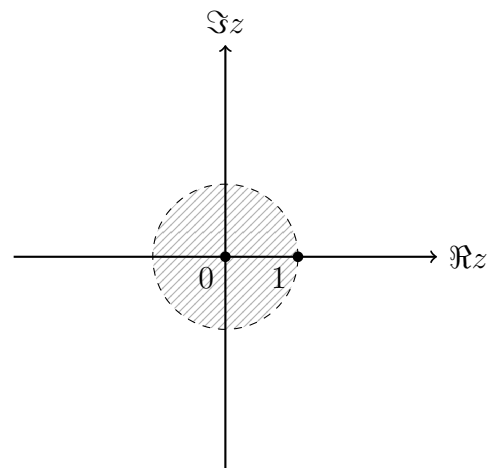


Множество  $Q_1 = z_1(Q)$

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

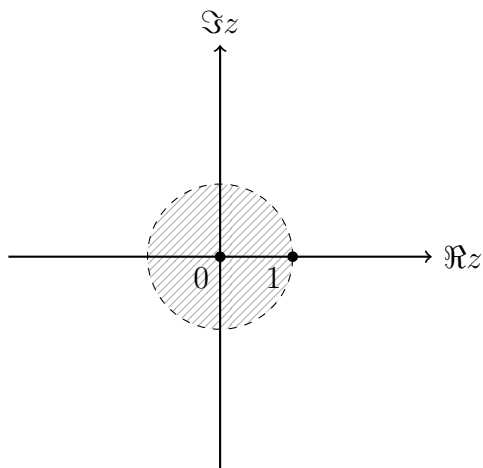


Множество  $Q_1$

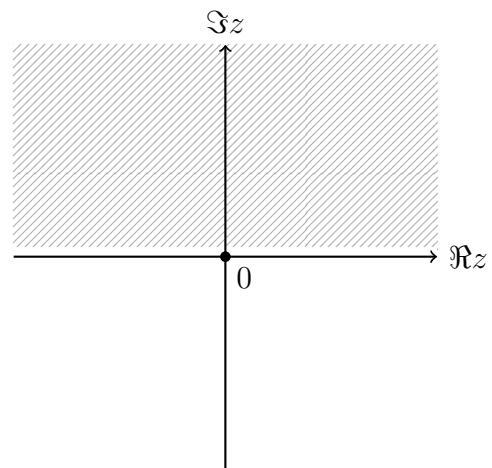


Множество  $Q_2 = z_2(Q_1)$

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

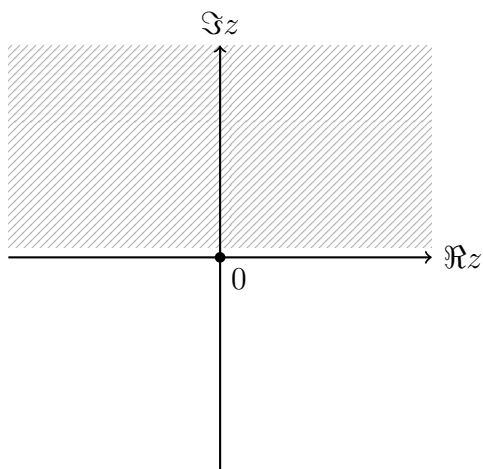


Множество  $Q_2$

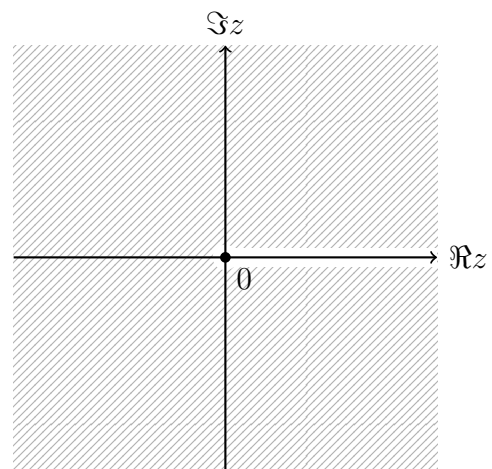


Множество  $Q_3 = z_3(Q_2)$

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

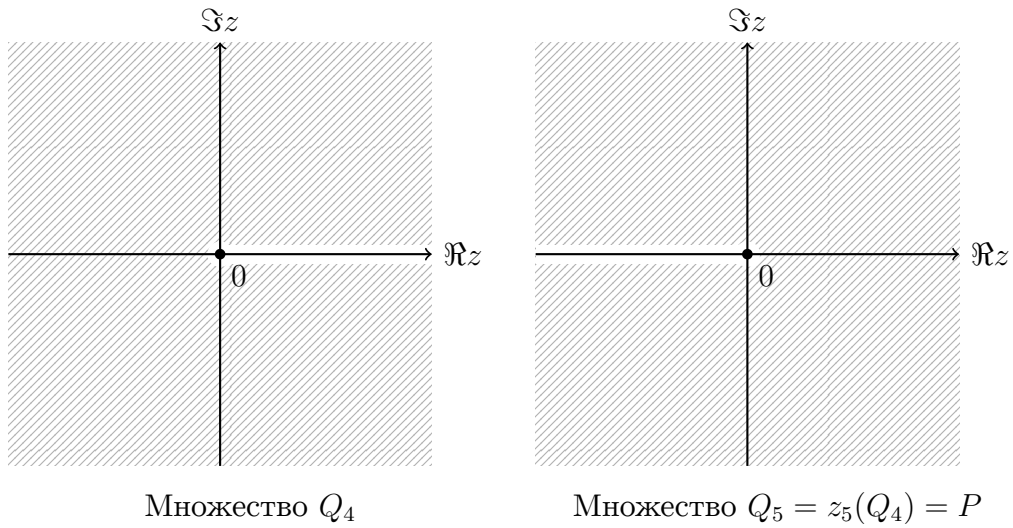


Множество  $Q_3$



Множество  $Q_4 = z_4(Q_3)$

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







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:

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

Листинг 1: Скрипт `generate_dots.py`

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3

```

```

4 # Parameters for point generation
5 epsilon = 0.02 # Small number to exclude boundary values
6 density = 50 # Parameter affecting point density
7
8 def complex_positive(density):
9     """
10     Generates points in the sector (3pi/4, pi) of the complex
11     plane.
12     Points are distributed in logarithmic scale by radius.
13     """
14     angles_far = np.linspace(np.pi - np.pi / 4 + epsilon, np.pi -
15                             epsilon, density // 9)
16     radius_far = np.exp(np.linspace(np.log(0.005), np.log(3),
17                                     density // 2))
18     r_grid, theta_grid = np.meshgrid(radius_far, angles_far)
19     z = r_grid * np.exp(1j * theta_grid)
20     points = z.flatten()
21     distances = np.abs(points)
22     colors = distances / np.max(distances) # Normalized distances
23     for coloring
24     return points, colors
25
26 def complex_negative(density):
27     """
28     Generates points in the sector (-pi, -3pi/4) of the complex
29     plane.
30     """
31     angles_center = np.linspace(-np.pi + epsilon, -np.pi + np.pi /
32                                 4 - epsilon, density // 9)
33     radius_center = np.exp(np.linspace(np.log(0.005), np.log(3),
34                                         density // 2))
35     r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
36     z = r_grid * np.exp(1j * theta_grid)
37     points = z.flatten()
38     distances = np.abs(points)
39     colors = distances / np.max(distances)
40     return points, colors
41
42 def complex_far_positive(density):
43     """
44     Generates points in the sector [0, 3pi/4] of the complex plane
45     .
46     """
47     angles_center = np.linspace(0, np.pi - np.pi/4, density // 4)
48     radius_center = np.exp(np.linspace(np.log(0.005), np.log(3),
49                                         density // 2))
50     r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
51     z = r_grid * np.exp(1j * theta_grid)
52     points = z.flatten()
53     distances = np.abs(points)
54     colors = distances / np.max(distances)

```

```

46     return points, colors
47
48 def complex_far_negative(density):
49     """
50     Generates points in the sector  $[-3\pi/4, 0)$  of the complex
51     plane.
52     """
53     angles_center = np.linspace(-np.pi + np.pi/4, -epsilon,
54                                density // 4)
55     radius_center = np.exp(np.linspace(np.log(0.005), np.log(3),
56                                       density // 2))
57     r_grid, theta_grid = np.meshgrid(radius_center, angles_center)
58     z = r_grid * np.exp(1j * theta_grid)
59     points = z.flatten()
60     distances = np.abs(points)
61     colors = distances / np.max(distances)
62     return points, colors
63
64 def get_P(plane):
65     """
66     Filters points: keeps only those with  $\text{Re}(z) > 0$ 
67     or  $\text{Im}(z) \neq 0$  (excludes non-positive real axis).
68     """
69     mask = np.logical_or(plane.real > 0, plane.imag != 0)
70     return plane[mask]
71
72 # Definition of mapping functions omega_1, ..., omega_5
73 def omega_1(z):
74     """omega_1(z) = -z (central symmetry about origin)"""
75     return -z
76
77 def omega_2(z):
78     """omega_2(z) = sqrt(z) (square root extraction)"""
79     r = np.abs(z)
80     theta = np.angle(z)
81     sqrt_r = np.sqrt(r)
82     theta_pos = np.mod(theta, 2*np.pi) # Bring angle to [0, 2*pi)
83     new_theta = theta_pos / 2
84     return sqrt_r * (np.cos(new_theta) + 1j * np.sin(new_theta))
85
86 def omega_3(z):
87     """omega_3(z) = (z - i)/(z + i) (fractional-linear
88     transformation)"""
89     return (z-1j)/(z+1j)
90
91 def omega_4(z):
92     """omega_4(z) = 1/z (inversion)"""
93     return 1/z
94
95 def omega_5(z):
96     """omega_5(z) = z * e (multiplication by Euler's number e)"""

```

```

93     return z * np.e
94
95 def get_all_Ps(complex_region):
96     """
97     Applies all five mappings sequentially to input set.
98     """
99     P = get_P(complex_region)
100     P1 = omega_1(P)
101     P2 = omega_2(P1)
102     P3 = omega_3(P2)
103     P4 = omega_4(P3)
104     P5 = omega_5(P4)
105     return [P, P1, P2, P3, P4, P5]
106
107 def scatter_regions(regions, axes, colors, cmap):
108     """
109     Displays 6 sets on 3x2 grid of plots.
110
111     Parameters:
112     regions: list of 6 arrays of complex numbers
113     axes: 3x2 matrix of axes
114     colors: array of values for colormap
115     cmap: name of colormap
116     """
117     if len(regions) < 6:
118         print("Error: lack of regions")
119         pass
120     for y in range(3):
121         for x in range(2):
122             axes[y, x].scatter(regions[y * 2 + x].real, regions[y * 2
123                                     + x].imag,
124                                 s=1, alpha=0.6, c=colors, cmap=cmap)
125
126 if __name__ == "__main__":
127     fig, ax = plt.subplots(3, 2)
128
129     # Generate and display points from four sectors with different
130     # colormaps
131     reg1, col1 = complex_far_positive(density)
132     scatter_regions(get_all_Ps(reg1), ax, col1, 'winter')
133
134     reg2, col2 = complex_far_negative(density)
135     scatter_regions(get_all_Ps(reg2), ax, col2, 'copper')
136
137     reg3, col3 = complex_positive(density)
138     scatter_regions(get_all_Ps(reg3), ax, col3, 'cool')
139
140     reg4, col4 = complex_negative(density)
141     scatter_regions(get_all_Ps(reg4), ax, col4, 'autumn')
142
143     # Save data from each plot to separate file

```

```

142 for i in range(6):
143     row = i // 2
144     col = i % 2
145
146     with open(f'step_{i+1}_data.txt', 'w') as f:
147         # Get all point collections from the plot
148         for collection_idx, collection in enumerate(ax[row, col].
            collections):
149             data = collection.get_offsets() # Coordinates (x, y)
150             colors = collection.get_array() # Values for colormap
151             cmap_name = collection.get_cmap().name # Name of
                colormap
152
153             # Write data to file: x, y, color value, colormap name
154             for j in range(len(data)):
155                 x, y = data[j]
156                 color_val = colors[j] if j < len(colors) else 0.0
157                 f.write(f'{x:.6f} {y:.6f} {color_val:.6f} {cmap_name}\
                    n')
158
159     print(f"Step {i+1}: data saved")

```

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

Листинг 2: Скрипт tikzify.py

```

1 import numpy as np
2 import matplotlib as plt
3 import os
4
5 def get_tikz_color_with_alpha(color_val, colormap, alpha=1.0):
6     """
7     Convert matplotlib colormap value to TikZ color with alpha
        transparency.
8
9     Parameters:
10     color_val: normalized color value [0, 1]
11     colormap: name of matplotlib colormap
12     alpha: opacity value [0, 1]
13     """
14     cmap = plt.colormaps[colormap]
15     rgba = cmap(color_val)
16
17     # Convert rgba values (0-1) to rgb (0-255) for TikZ
18     r = int(rgba[0] * 255)
19     g = int(rgba[1] * 255)
20     b = int(rgba[2] * 255)
21     a = rgba[3] * alpha # Use alpha from colormap if available
22
23     return f"{{rgb,255:red,{r}; green,{g}; blue,{b}}}, opacity={{a
        :.2f}}"
24

```

```

25 def convert_to_tikz(input_file, output_file, step_num):
26     """
27     Convert data file with point coordinates to TikZ code.
28
29     Parameters:
30     input_file: text file with columns: x, y, color_value,
31                 colormap_name
32     output_file: output .tex file with TikZ code
33     step_num: ordinal numeral of current step
34     """
35     data = np.loadtxt(input_file, dtype=object)
36
37     # Validate data format
38     if data.shape[1] != 4:
39         raise ValueError("File must have 4 columns: x, y, color,
40                             colormap")
41
42     # Extract and convert data columns
43     points = data[:, :2].astype(float)
44     colors = data[:, 2].astype(float)
45     cmaps = data[:, 3]
46
47     # Normalize color values to [0, 1] range
48     colors_min = colors.min()
49     colors_max = colors.max()
50     colors_norm = (colors - colors_min) / (colors_max - colors_min
51         )
52
53     # Initialize TikZ code with axes and origin point
54     tikz_code = r"""\begin{tikzpicture}[scale=0.8]
55
56     \draw[->, thick] (-3.5,0) -- (3.5,0) node[right] {$\text{Re } z$};
57     \draw[->, thick] (0,-3.5) -- (0,3.5) node[above] {$\text{Im } z$};
58
59     \fill (0,0) circle (2.5pt) node[below right] {$0$};
60     """
61
62     match step_num:
63         case '4' | '5':
64             tikz_code += "\\fill (1.2,0) circle (2.5pt) node[below
65                 left] {$1$};\n"
66         case '6':
67             tikz_code += f"\\fill ({np.e},0) circle (2.5pt) node[below
68                 left] {{$e$}};\n"
69         case _:
70             tikz_code += "\n"
71
72     count = 0
73     for (x, y), color_val, cmap in zip(points, colors_norm, cmaps)
74         :
75         # Skip points outside the plotting area
76         if abs(x) > 3.5 or abs(y) > 3.5:

```

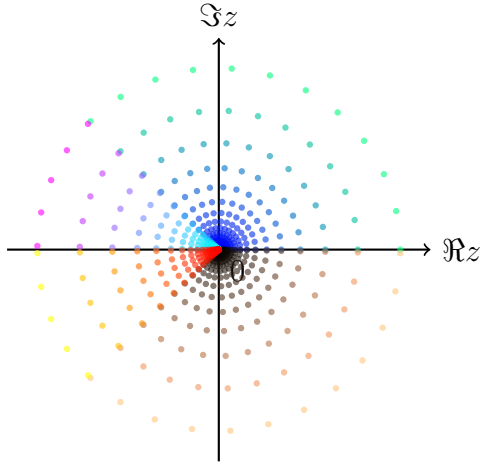
```

70         continue
71
72         count += 1
73         tikz_color = get_tikz_color_with_alpha(color_val, cmap,
74             alpha=0.6)
75         tikz_code += f"\\fill[color={tikz_color}] ({x:.4f},{y:.4f}
76             }) circle (1.5pt);\\n"
77
78         tikz_code += "\\n\\end{tikzpicture}"
79
80         with open(output_file, 'w') as f:
81             f.write(tikz_code)
82
83         print(f"Saved: {output_file} ({count} points)")
84
85     if __name__ == "__main__":
86         # Find all data files starting with 'step_' and ending with '
87         # _data.txt'
88         data_files = [f for f in os.listdir() if f.startswith('step_')
89             and f.endswith('_data.txt')]
90
91         for data_file in data_files:
92             # Extract step number from filename (e.g., 'step_1_data.txt'
93             # -> '1')
94             step_num = data_file.split('_')[1]
95             output_file = f'step_{step_num}_tikz.tex'
96
97             convert_to_tikz(data_file, output_file, step_num)

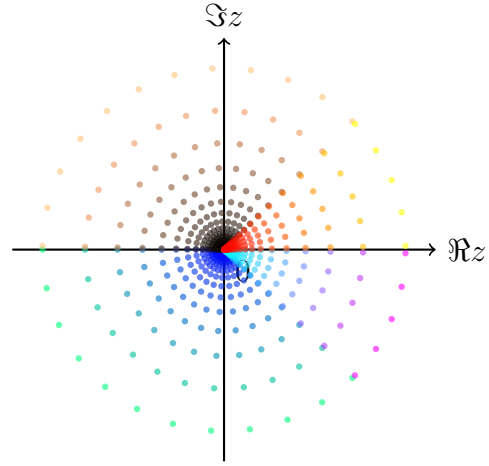
```

## 5 Результаты отображений множества

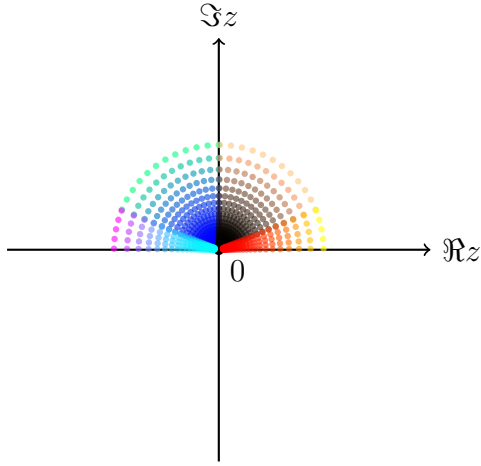
В результате работы вышеуказанных скриптов были получены следующие множества:



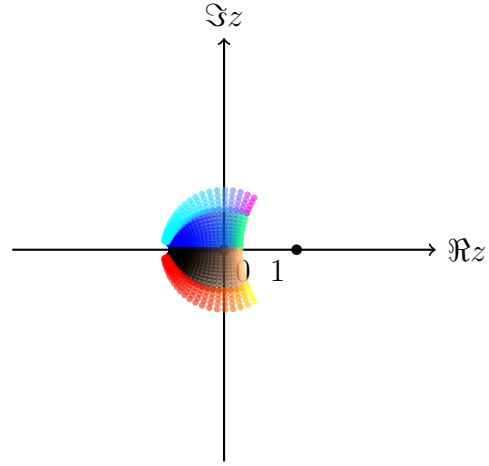
Множество  $P$



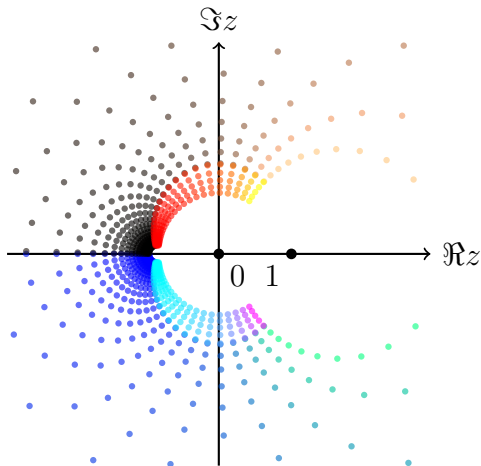
Множество  $P_1 = \omega_1(P)$



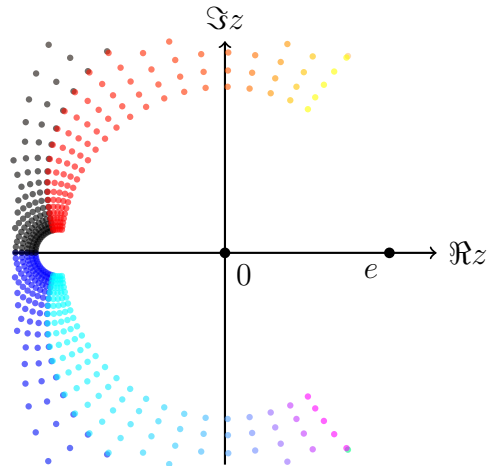
Множество  $P_2 = \omega_2(P_1)$



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.