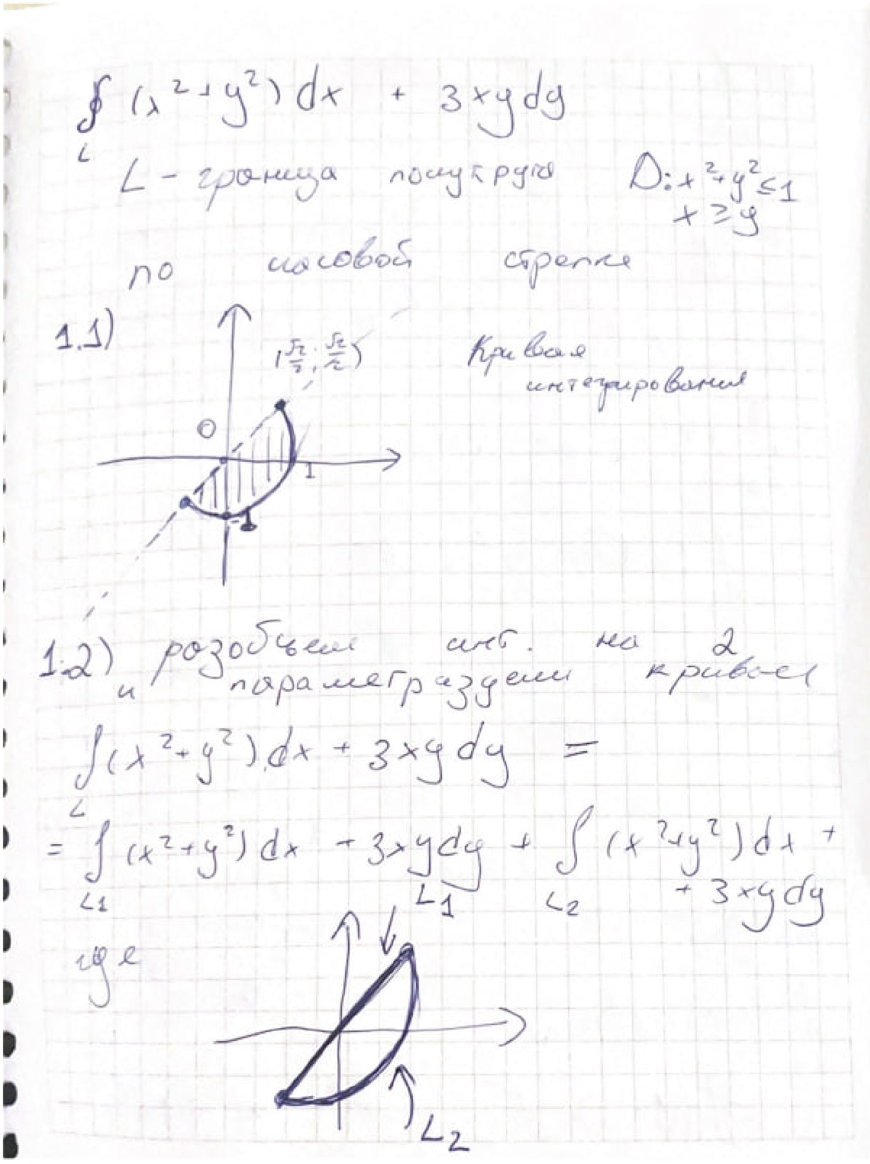
Лабораторная работа по математическому анализу №2

Веселкова Варвара M3234

Тема: Двойной и криволинейный интегралы. Формула Грина.

1 Часть 1. Аналитический метод



$\oint_L (x^2+y^2)\,dx + 3xy\,dy$

L – граница полукруга $D: x^2+y^2 \le 1,\ x \ge y$

по часовой стрелке

1.1) $\left(\frac{\sqrt{2}}{2}; \frac{\sqrt{2}}{2}\right)$ Кривая интегрирования

1.2) разобьем инт. на 2 и параметризуем кривые

$\int_L (x^2+y^2)\,dx + 3xy\,dy =$

$= \int_{L_1} (x^2+y^2)\,dx + 3xy\,dy + \int_{L_2} (x^2+y^2)\,dx + 3xy\,dy$

где

1) L_1 можно задать параметр

как: $\begin{cases} x = t \\ y = t \end{cases}$

тогда знак: $\int\limits_{L_0} f_1(x;y)\,dx + f_2(x;y)\,dy =$

$= \int\limits_a (f_1(x(t); y(t))\,x'(t) +$ -//- но по y $)\,dt$

$\int\limits_{L_1} (x^2 - y^2)\,dx + 3+y\,dy =$

$= \int\limits_{-\frac{\sqrt2}{2}}^{\frac{\sqrt2}{2}} (2t^2 \cdot 1 + 3t^2 \cdot 1)\,dt = \frac{5\sqrt2}{6}$

2) L_2 можно задать как

~~xxx~~ $\begin{cases} x = \cos t \\ y = \sin t \end{cases}$

(где $t \in [-\frac{3}{4}\pi; \frac{1}{4}\pi]$

↑

помним про порядок обхода!

тогда:

$\int\limits_{L_2} f_1(x;y)\,dx + f_2(x;y)\,dy =$

$= -\int\limits_{-\frac{3}{4}\pi}^{\frac{1}{4}\pi} (1 \cdot -\sin t + 3\sin^4 t \cdot \cos^2 t)\,dt$

$$= -\left(\cos(t) - \cos^3(t) \Big|_{-\frac{\pi}{4}}^{\frac{3\pi}{4}}\right) =$$

$$= -\frac{\sqrt{2}}{2}$$

Тогда $\oint\limits_L f \ldots = \frac{5\sqrt{2}}{6} - \frac{\sqrt{2}}{2} =$

$$= \frac{\sqrt{2}}{3}$$

1.3) Ф-ла Грина: $\oint\limits_{\partial D} P dx + Q dy = \iint\limits_D \left(\frac{\partial Q}{\partial x} - \frac{\partial P}{\partial y}\right) dx dy$

$P = x^2 + y^2$

$Q = 3xy$

$\Rightarrow \frac{dQ}{dx} = 3y; \quad \frac{dP}{dy} = 2y$

В полярных коорд.:

$$\begin{cases} x = r\cos\varphi \\ y = r\sin\varphi \end{cases}$$

границы:

$r \in [0; 1]$

φ от $\frac{\pi}{4}$ до $-\frac{3\pi}{4}$

Тогда:

$$\iint\limits_D (3y - 2y) dx dy = \overset{\text{обход}}{-} \int\limits_{-\frac{3\pi}{4}}^{\frac{\pi}{4}} d\varphi \int\limits_0^1 r \cdot r\sin\varphi \, dr$$

$$= -\int\limits_{-\frac{3\pi}{4}}^{\frac{\pi}{4}} d\varphi \, \frac{r^3 \sin\varphi}{3} \Big|_0^1 = -\int\limits_{-\frac{3\pi}{4}}^{\frac{\pi}{4}} d\varphi \, \frac{\sin\varphi}{3} =$$

$$= -\left(-\frac{\sqrt{2}}{3}\right) = \frac{\sqrt{2}}{3}$$ сошлось!

2 Часть 2. Численный метод

Исходный код

```python
import math
from time import time
import numpy as np
```

Задаем нашу функцию

```python
def p(x, y):
    return x ** 2 + y ** 2
def q(x, y):
    return 3 * x * y
```

Поворот (для прохождения по окружности)

```python
def chord_angle(chord_length):
    return math.asin(chord_length / 2) * 2
def get_rotate_function(phi):
    def rotate(vector):
        rotation_matrix = np.array([
            [math.cos(phi), -math.sin(phi)],
            [math.sin(phi), math.cos(phi)]
        ])
        return vector.dot(rotation_matrix).round(100)

    return rotate
```

Вычисляем интеграл, используя метод трапеций

```python
def calculate_integral(dots):
    return sum([p(dots[i][0], dots[i][1]) * (dots[i][0] - dots[i - 1][0]) +
                q(dots[i][0], dots[i][1]) * (dots[i][1] - dots[i - 1][1])
                for i in range(1, len(dots))])
```

Проходим по пути (сначала по дуге, потом по прямой)

```python
def create_path(step):
    dots = []
    angle = math.pi / 4
    step_angle = chord_angle(step)
    x, y = math.sqrt(2) / 2, math.sqrt(2) / 2
    rotate = get_rotate_function(step_angle)

    while angle > -3 * math.pi / 4:
        dots.append((x, y))
        angle -= step_angle
        x, y = rotate(np.array([x, y]))

    x, y = -math.sqrt(2) / 2, -math.sqrt(2) / 2
    diagonal_step = step * math.sqrt(0.5)

    while x <= math.sqrt(2) / 2:
        dots.append((x, y))
        x += diagonal_step
        y += diagonal_step

    return dots
```

Наша функция для двойного интеграла выч. в аналитике

```python
def function_f(x, y):
    return y
```

Функции проверки для построения минимальной, максимальной и средней интегральных сумм

```python
def is_center_inside(x, y, delta):
    return x ** 2 + y ** 2 <= 1 and x >= y


def is_all_inside(x, y, delta):
    for shift_x in [delta / 2, -delta / 2]:
        for shift_y in [delta / 2, -delta / 2]:
            if (x + shift_x) ** 2 + (y + shift_y) ** 2 > 1 or (x + shift_x) < (y +
    shift_y):
```

```python
                return False
    return True


def is_corner_inside(x, y, delta):
    for shift_x in [delta / 2, -delta / 2]:
        for shift_y in [delta / 2, -delta / 2]:
            if (x + shift_x) ** 2 + (y + shift_y) ** 2 <= 1 and (x + shift_x) >= (y + shift_y):
                return True
    return False
```

В зависимости от выбранного условия делаем подсчет

```python
def calculate_integral_sum_conditionally(x, y, delta, condition):
    if condition(x, y, delta):
        return delta ** 2 * function_f(x, y)
    return 0
```

Считаем и выводим результат

```python
def main():
    integral_value = math.sqrt(2) / 3
    deltas = [0.1, 0.01, 0.001]

    for delta in deltas:
        start_time = time()
        dots = create_path(delta)
        integral_sum = calculate_integral(dots)
        elapsed_time = time() - start_time

        print(
            f"Results for delta = {delta}: Integral Sum: {integral_sum}, Error: {integral_value - integral_sum}, Time: {elapsed_time}")

    for delta in deltas:
        start_time = time()
        integral_sum, min_sum, max_sum = 0, 0, 0

        y_start = 1.5
        x_start = -2.5
        while y_start - delta >= -1.5:
            x = x_start
            while x + delta <= 1.5:
                center_x, center_y = x + delta / 2, y_start - delta / 2
                integral_sum -= calculate_integral_sum_conditionally(center_x, center_y, delta, is_center_inside)
                min_sum += calculate_integral_sum_conditionally(center_x, center_y, delta, is_all_inside)
                max_sum += calculate_integral_sum_conditionally(center_x, center_y, delta, is_corner_inside)
                x += delta
            y_start -= delta

        elapsed_time = time() - start_time
        print(
            f"Results for delta = {delta}: Integral Sum: {integral_sum}, Error: {integral_value - integral_sum}, Time: {elapsed_time}, Delta (max-min): {max_sum - min_sum}")


if __name__ == "__main__":
    main()
```

Результаты вычислений

Таблица 1: Результаты вычислений для п.1

Δ	Интегральные суммы	Отклонение от истинного значения	Время работы
0.1	0.43718	0.03422	0.0s
0.01	0.46788	0.00352	0.004s
0.001	0.47105	0.00035	0.032s

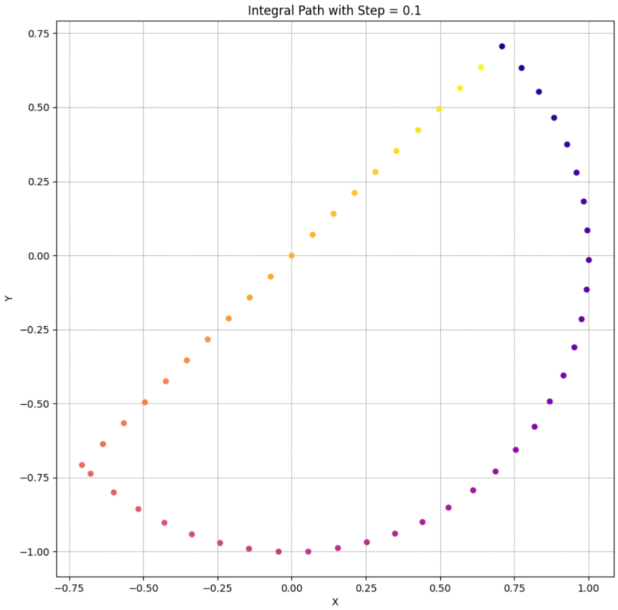
Таблица 2: Результаты вычислений для п.2 и п.4

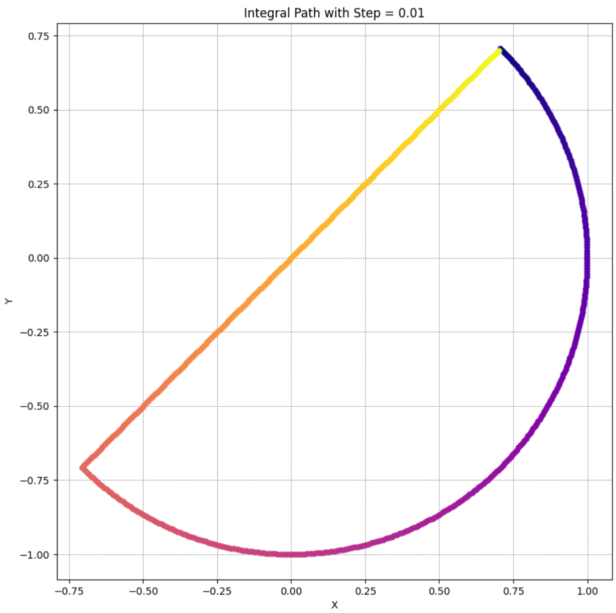
Δ	Интегральные суммы	Отклонение от истинного значения	Время работы	Разница (min-max)
0.1	0.47450	-0.00310	0.005s	-0.16000
0.01	0.47165	-0.00025	0.467s	-0.01782
0.001	0.47142	-0.00002	47.245s	-0.00179

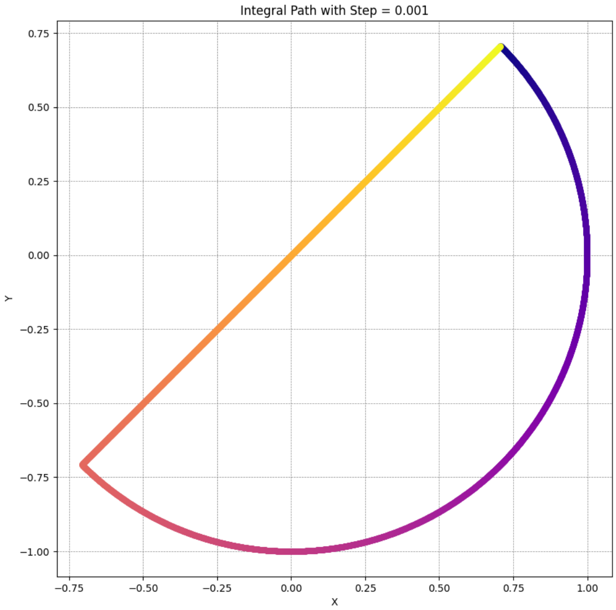
Вывод

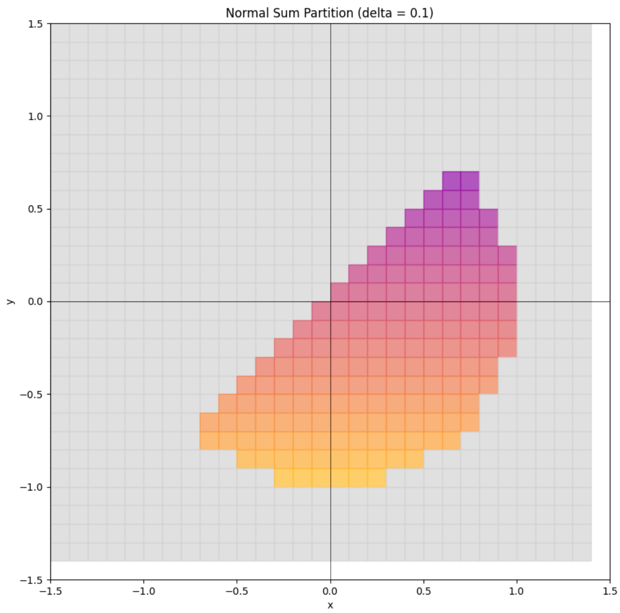
Значения получились близкие к полученному в аналитическом методе, все хорошо.

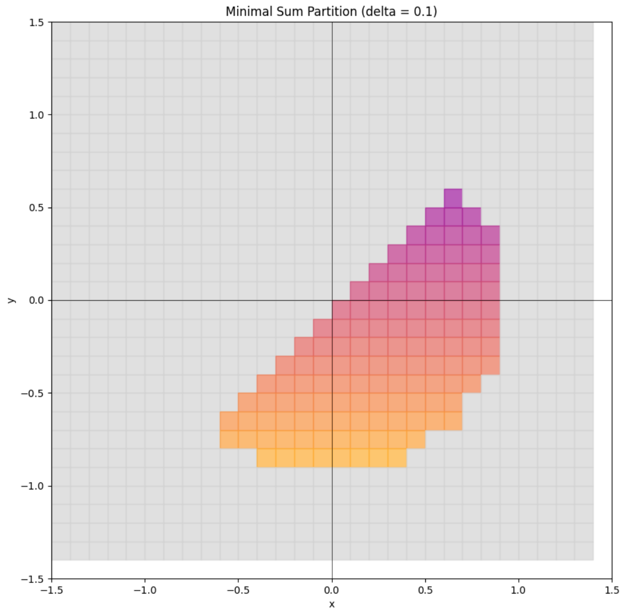
Графики

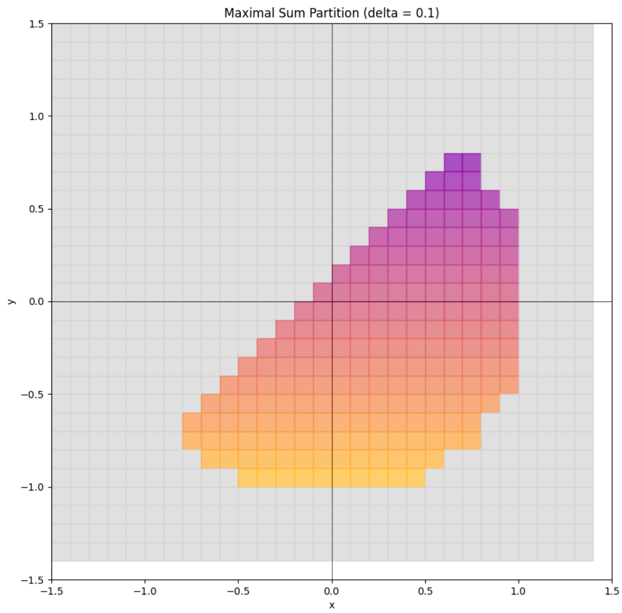

Integral Path with Step = 0.1
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Integral Path with Step = 0.01
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Integral Path with Step = 0.001
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
Y
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
X


Normal Sum Partition (delta = 0.1)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x


Minimal Sum Partition (delta = 0.1)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x


Maximal Sum Partition (delta = 0.1)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
y
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
x