САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Математический анализ

Отчет

по лабораторной работе №2

«Нахождение точек локальных экстремумов функции»

Выполнил(а): Ступников Александр Сергеевич

студ. гр. M3235

Санкт-Петербург

2021

Аналитический метод

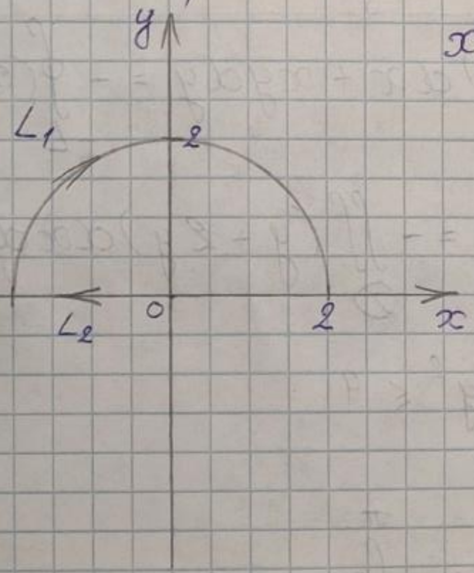
28 ноября.

Матан.

Лаб. работа № 2

Вариант 46.

1.1 $x^2+y^2=4\ ,\ y \geq 0$



1.2 $\begin{cases} x = 2\cos\alpha \\ y = 2\sin\alpha \end{cases}$ $\quad dx = -2\sin\alpha\, d\alpha$, $\quad dy = 2\cos\alpha\, d\alpha$

$$I = \oint\limits_L (x^2+y^2)\,dx + xy\,dy = \int\limits_{L_1} + \int\limits_{L_2}$$

$$\int\limits_{L_1} = \int\limits_{\pi}^{0} 4\cdot(-2)\sin\alpha\, d\alpha + 4\sin\alpha\cos\alpha\, 2\cos\alpha\, d\alpha =$$

$$= 8\int\limits_{0}^{\pi}(-\cos^2\alpha\sin\alpha + \sin\alpha)\,d\alpha =$$

$$= 8\left(\frac{\cos^3\alpha}{3} - \cos\alpha\right)\Big|_0^{\pi} = 8\left(-\frac{1}{3}+1-\frac{1}{3}+1\right) =$$

$$= \frac{32}{3} \qquad \int\limits_{L_2} = \int\limits_{2}^{-2} x^2 dx = \frac{x^3}{3}\Big|_2^{-2} = -\frac{16}{3}$$

1.3 $\quad \boxed{I = \frac{16}{3}}$

$$\oint\limits_{L} (x^2+y^2)\,dx + xy\,dy = -\oint\limits_{-L} (x^2+y^2)\,dx +$$

$$+ xy\,dy = -\iint\limits_{D} (y-2y)\,dx\,dy = \iint\limits_{D} y\,dx\,dy \ominus$$

$$D: \begin{cases} x^2+y^2 \leq 4 \\ y \geq 0 \end{cases}$$

$$\ominus \int\limits_0^2 r\,dr \int\limits_0^{\pi} r\sin\varphi\,d\varphi = 2\int\limits_0^2 r^2\,dr =$$

$$= 2\,\frac{r^3}{3}\Big|_0^2 = \frac{16}{3}$$

Ответы сходятся, ф-ла Грина работает.

1.4 ~~$y=\sqrt{4-x^2}$~~,

L_1: $\bar{\tau} = \bar{r}' = (-2\sin t \quad 2\cos t)$

$\bar{\tau}_0 (\cos\alpha \quad \cos\beta) = \dfrac{\bar{r}'}{|\bar{r}'|} = (-\sin t \quad \cos t)$

$dx = -\sin t\, dl$
$dy = \cos t\, dl$

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

$$= \int\limits_{L_1} (-4\sin t + 4\sin t\cos^2 t)\,dl$$

$$\int\limits_{L_2} (x^2+y^2)\,dx + xy\,dy = \int\limits_{L_2} -x^2\,dl \text{, т.к.}$$

$dx = \cos\alpha = +1 \; dl$
$dy = y = 0$

$$\underline{I} = \int\limits_{L_1} (-4\sin t + 4\sin t\cos^2 t)\,dl + \int\limits_{L_2} +x^2\,dl =$$

$$= -8\int\limits_{\pi}^{0} (\sin t - \sin t\cos^2 t)\,dt + 2\int\limits_{2}^{-2} x^2\,dx =$$

$$= \frac{32}{3} - \frac{16}{3} = \frac{16}{3}$$

Численный метод

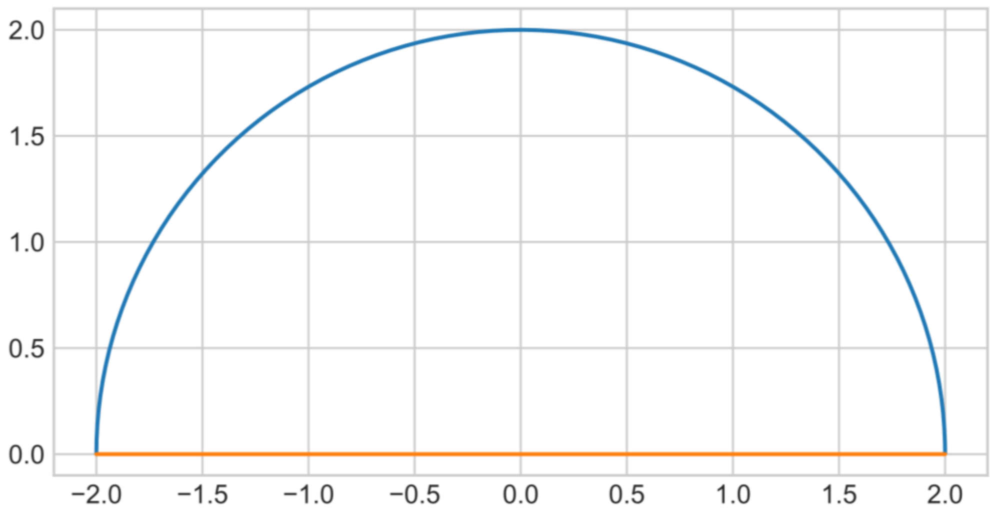


Рисунок №1 – кривая интегрирования L

Программа находит приблизительное численное значение криволинейного интеграла второго рода тремя способами:

1. по определению
2. по формуле Грина
3. путём сведения криволинейного интеграла второго рода к криволинейному интегралу первого рода

На вход программе подаётся (все входные данные определяются путём внесения изменений в исходный код программы):

1. точность вычисления интеграла δ
2. кривая L в виде массива гладких параметризованных кривых $(x(t), y(t))$, которые составляют L
3. потенциал $F = (P(x, y), Q(x, y))$
4. Значение интеграла, вычисленное аналитически (опционально)
5. множество D
6. ориентация кривой

Код снабжён подробными комментариями. Программа строит график кривой интегрирования L, результат сохраняется в файл «Curves.svg». Предполагается, что программа должна находить числовое значение двойного интеграла для широкого класса кривых и потенциалов (есть надежда, что программа может находить значение криволинейного интеграла второго рода для любых кусочно-гладких кривых и непрерывных потенциалов (если отключить код, использующий теорему Грина)). Следует заметить, что вычисление двойного интеграла происходит за $O(n^2)$ в силу необходимости разбиения плоскости на квадраты. Поэтому для получения ответа для маленьких δ можно закомментировать часть кода, ответственную за вычисление двойного интеграла и вычислять только криволинейные интегралы.

Листинг

Интерпретатор Python 3.9.2. Использованы библиотеки matplotlib 3.4.1 и numpy 1.20.1.

main.py

```
import math
import time

import numpy as np
import matplotlib.pyplot as plt
from itertools import product
import sympy
from sympy import diff, Symbol, lambdify


# Generates partition for curve F(x, y) = 0; x, y in R
# xt, yt - parameterization for curve F(x, y) = 0
# r[t1, t2] - range of parameter t (t from t1 to t2)
# delta - max distance between two point in partition
# return value: array of pairs
def gen_curve_partition(xt, yt, r, delta):
    step = delta if r[1] >= r[0] else -delta
    if r[0] == r[1]:
        raise ValueError("r[0] should not be equal r[1]")
    while True:
        n = int((r[1] - r[0]) / step) + 1
        x = np.array([xt(r[0] + i * step) for i in range(n)])
        y = np.array([yt(r[0] + i * step) for i in range(n)])
        delta_x = x[1:] - x[:-1]
        delta_y = y[1:] - y[:-1]
        if max(np.sqrt(delta_x * delta_x + delta_y * delta_y)) <= delta:
            return [x, y]
        step /= 2


# Finds integral sum for curve integral of 2nd kind with potential f
# (x[i], y[i]) - points in partition
# f = (p, q) - potential
def find_2nd_kind_curve_integral_sum(x, y, f):
    delta_x = x[1:] - x[:-1]
    delta_y = y[1:] - y[:-1]
```

```
    return sum(np.array(list(map(f[0], zip(x, y))))[:-1] * delta_x +
               np.array(list(map(f[1], zip(x, y))))[:-1] * delta_y)


# Generates partition of rectangular region [x_min, x_max] x [y_min, y_max] in plane R^2 with equal squares
# x_min, x_max, y_min, y_max - rectangle corners
# step - size of the squares sides
# return value: array of pairs
def gen_center_tagged_plane_partition(x_min, x_max, y_min, y_max, step):
    n = int((x_max - x_min) / step) + 1
    x = [x_min + step * i + step / 2 for i in range(n)]
    n = int((y_max - y_min) / step) + 1
    y = [y_min + step * i + step / 2 for i in range(n)]
    return [x, y]


# Finds double integral on set d with integrand f
# points - points in which value of f evaluates
# step - side of the square in partition
# d - shows if point from points in set d
# f - integrand
def find_double_integral_sum(points, step, d, f):
    s = 0
    for point in points:
        if d(*point):
            s += f(*point)
    return s * np.square(step)


# Finds 1st kind curve integrand
# curve - parametrized curve
# potential - potential of the corresponding 2nd kind integral
# return value: lambda function
def find_1st_kind_curve_integrand(curve, potential):
    t = Symbol("t")
    x_der, y_der = curve[0].diff(t), curve[1].diff(t)
    norm = sympy.sqrt(x_der * x_der + y_der * y_der)
    x_der_norm = x_der / norm
    y_der_norm = y_der / norm
    return lambdify(t,
                    potential[0]([curve[0], curve[1]]) * x_der_norm +
                    potential[1]([curve[0], curve[1]]) * y_der_norm)


# Finds integral to evaluate curve arc length
# curve - parametrized curve
# return value: lambda function
def find_arc_length_integral(curve):
    t = Symbol("t")
    x_der, y_der = curve[0].diff(t), curve[1].diff(t)
    norm = sympy.sqrt(x_der * x_der + y_der * y_der)
    return lambdify(t, sympy.integrate(norm, t))
```

```
# Finds integral sum for curve integral of 1nd kind with integrand f
# curve - parametrized curve
# f - integrand
# arc_length_integral - integral for evaluation arc length between to
points on curve
# delta - determine finest of partition
def find_1st_kind_curve_integral_sum(curve, f, arc_length_integral,
delta):
    t = Symbol("t")
    n = len(gen_curve_partition(*[lambdify(t, curve[0]), lambdify(t,
curve[1]), curve[2]], delta)[0])
    step = (curve[2][1] - curve[2][0]) / n
    res = 0
    for j in range(n - 1):
        length = arc_length_integral(curve[2][0] + step * (j + 1)) -
arc_length_integral(curve[2][0] + step * j)
        res += f(curve[2][0] + step * j) * length
    return res


def main():
    # Determine finest of partitions for integrals
    delta = 0.01
    # Parametrized integral curves
    t = Symbol("t")
    parametrized_sym_curves = [
        [2 * sympy.cos(t), 2 * sympy.sin(t), [np.pi, 0]],
        [t, sympy.S(0), [2, -2]]
    ]
    # Potential for curve integral 2nd kind
    x, y = Symbol("x"), Symbol("y")
    sym_potential = [x * x + y * y, x * y]
    # True integral sum value (optional)
    true_value = 16 / 3
    # Double integral region
    # Should return True if point (x, y) in set d, else should return
False
    def d(x, y): return y >= 0 and x * x + y * y <= 4
    # Initial orientation of the curve (for double integral evaluation
(Green's theorem))
    # 1 - positive orientation, -1 - negative orientation
    orientation = -1

    plt.style.use('seaborn-whitegrid')
    fig, ax = plt.subplots()
    plt.gca().set_aspect('equal', adjustable='box')
    potential = [
        lambdify(((x, y), ), sym_potential[0]),
        lambdify(((x, y), ), sym_potential[1])
    ]
    parametrized_curves = [[
        lambdify(t, parametrized_sym_curves[i][0]),
        lambdify(t, parametrized_sym_curves[i][1]),
```

```
        parametrized_sym_curves[i][2]
    ] for i in range(len(parametrized_sym_curves))]

    # 2nd kind curve integral sum evaluation
    start_time = time.time()
    res = 0
    for curve in parametrized_curves:
        x, y = gen_curve_partition(*curve, delta)
        ax.plot(x, y)
        res += find_2nd_kind_curve_integral_sum(x, y, potential)
    # Plot of parametrized curves
    plt.savefig(f'./Curves.svg')
    print("2nd kind curve integral sum:", res)
    print("Deviation from true value:", abs(res - true_value))
    print(f'Elapsed time: {time.time() - start_time:.3f} s\n')

    # Double integral sum evaluation
    start_time = time.time()
    x, y = Symbol("x"), Symbol("y")
    # f - integrand
    f = lambdify((x, y), orientation * (diff(sym_potential[1], x) -
diff(sym_potential[0], y)))
    x_min, x_max, y_min, y_max = np.inf, -np.inf, np.inf, -np.inf
    for curve in parametrized_curves:
        x, y = gen_curve_partition(*curve, delta)
        x_min = min(x_min, *x)
        x_max = max(x_max, *x)
        y_min = min(y_min, *y)
        y_max = max(y_max, *y)
    x, y = gen_center_tagged_plane_partition(x_min, x_max, y_min, y_max,
delta)
    res = find_double_integral_sum(product(x, y), delta, d, f)
    print("Double integral sum:", res)
    print("Deviation from true value:", abs(res - true_value))
    print(f'Elapsed time: {time.time() - start_time:.3f} s\n')

    # Difference between max and min double integral sums evaluation
    start_time = time.time()
    def d_max(x, y): return x * x + y * y <= 4 + delta / 2
    def d_min(x, y): return x * x + y * y <= 4 - delta / 2 * math.sqrt(2)
    print("Difference between max and min double integral sums: ",
        find_double_integral_sum(product(x, y), delta, d_max, f) -
        find_double_integral_sum(product(x, y), delta, d_min, f))
    print(f'Elapsed time: {time.time() - start_time:.3f} s\n')

    # 1st kind integral sum evaluation
    start_time = time.time()
    res = 0
    for curve in parametrized_sym_curves:
        f = find_1st_kind_curve_integrand(curve, potential)
        arc_length_integral = find_arc_length_integral(curve)
        res += find_1st_kind_curve_integral_sum(curve, f,
arc_length_integral, delta)
    print("1st kind integral sum:", res)
```

```
    print("Deviation from true value:", abs(res - true_value))
    print(f'Elapsed time: {time.time() - start_time:.3f} s\n')


main()
```