**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра систем автоматизированного проектирования**

**ОТЧЁТ**

**по лабораторной работе №2  
по дисциплине «Компьютерная графика»**

**Тема: Формирования различных кривых с использованием**

**ортогонального проектирования на плоскость визуализации (экране дисплея)**

|  |  |
| --- | --- |
| Студенты \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  |
| Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Матвеева И.В. |

Санкт-Петербург

2024

**Задание на лабораторную работу**

Вариант 9: Сформировать на плоскости В-сплайновую кривую различной степени (1, 2, 3, 4, 5, 6) на основе 7 неповторяющихся задающих точек. Обеспечить редактирование координат задающих точек с перерисовкой сплайна.

**Теоретические положения**

Дано n + 1 контрольных точек p0, p1, ..., pn и узловой вектор U = { u0, u1, ..., um }, кривая B-spline степени p, определяемая этими точками и узловым вектором U, это , где Ni,p(u) - это базисные функции кривой B-spline степени p. Вид кривой B-spline очень похож на кривые Безье. В отличие от кривых Безье, кривые B-spline требуют больше информации, а именно: набор контр. точек, узловой вектор, и степень. Заметьте, что n, m и p должны удовлетворять условию m = n + p + 1. Точнее, если мы хотим описать кривую B-spline степени p с n + 1 контр. точками, нам нужно обеспечить n + p + 2 узлов u0, u1, ..., un+p+1. С другой стороны, если узловой вектор состоит из m + 1 узлов и дано n + 1 контр. точек, то степень кривой B-spline равна p = m - n - 1.

Хотя Ni,p(u) похоже на Bn,i(u), степень кривых B-spline - это входные данные, тогда как степень кривой Безье зависит от количества контр. точек. Поэтому, чтобы изменить форму кривой B-spline, можно изменить один или несколько управляющих параметров: положения контрольных точек, положения узлов, степень кривой.

Если узловой вектор не имеет конкретной структуры, полученная кривая не будет касаться первого и последнего сегмента контрольной ломаной. Этот тип кривых B-spline называется открытыми кривыми B-spline. Мы, возможно, захотим закрепить кривую, то есть сделать так, чтобы она была касательна первому и последнему сегментам, как в случае с кривыми Безье. Чтобы это сделать, первый и последний узлы должны повторяться p+1 раз (т.e., быть множественности p+1). Таким образом, получим так называемую фиксированную кривую B-spline. Дублируя некоторые узлы и контрольные точки, можно сделать полученную кривую замкнутой. В этом случае начало и конец полученной кривой соединяются, образуя замкнутую петлю. Таким образом получаем фиксированную кривую.

Свойства:

1. Кривая B-spline p(u) является кусочной кривой, каждый компонент которой - кривая степени p
2. Равенство m = n + p + 1 должно выполняться
3. Фиксированный кривая B-spline p(u) проходит через две крайние контр. точки p0 и pn
4. Схема Локального Изменения [Local Modification Scheme] : изменение положения контрольной точки pi влияет только на кривую p(u) на интервале [ui, ui+p+1)
5. Кривые Безье - это Особый Случай Кривых B-spline

**Ход работы**

* По нажатию кнопки запускается главная функция программы start\_func, которая получает введенные координаты точек и степень кривизны.
* Далее запускается функция curve\_generator, откуда мы получим координаты для новых точек:

def curve\_generator(n, k, ctrl\_x, ctrl\_y):

# Формируем вектор узлов по количеству точек и степени кривизны

t = knot\_values(n, k)

print("t: ", t)

# Массив чисел с шагом в 0.001 от 0 до n - k + 1, где n - количество точек, k - кривизна

u = np.arange(t[0][k - 1], t[0][n + 1], 0.001)

print("u: ", )

length = u.shape

# Массивы нулей для заполнения

x = np.zeros((1, length[0]))

y = np.zeros((1, length[0]))

# Заполняем массивы

# ctrl x и y координаты оригинальных введенных точек

for i in range(0, n + 1):

for j in range(0, length[0]):

x[0][j] = x[0][j] + basis\_spline(u[j], t, i, k) \* ctrl\_x[i]

y[0][j] = y[0][j] + basis\_spline(u[j], t, i, k) \* ctrl\_y[i]

print("x: ", x)

print("y: ", y)

return x, y

* Внутри основной функции также вызывается функция knot\_values, которая формирует массив узлов по количеству точек и степени кривизны:

def knot\_values(n, k):

# Количество узлов по формуле i = n + k + 1

t = np.zeros((1, n + k + 1))

for i in range(0, n + k + 1):

if i < k:

t[0][i] = 0

if k <= i <= n:

t[0][i] = i - k + 1

if i > n:

t[0][i] = n - k + 2

return t

* Еще одна используемая функция basis\_spline это расчет коэффициентов B-сплайна, которые используются для формирования новых точек для кривой:

def basis\_spline(u, t, i, k):

# Если степень кривизны 1

if k == 1:

if t[0][i] <= u < t[0][i + 1]:

sol = 1

else:

sol = 0

# Степень кривизны больше 1

# i - 1 из 7 рассматриваемых точек

else:

a = (u - t[0][i]) \* basis\_spline(u, t, i, k - 1)

b = t[0][i + k - 1] - t[0][i]

c = (t[0][i + k] - u) \* basis\_spline(u, t, i + 1, k - 1)

d = t[0][i + k] - t[0][i + 1]

if b == 0:

temp1 = 0

else:

temp1 = a / b

if d == 0:

temp2 = 0

else:

temp2 = c / d

sol = temp1 + temp2

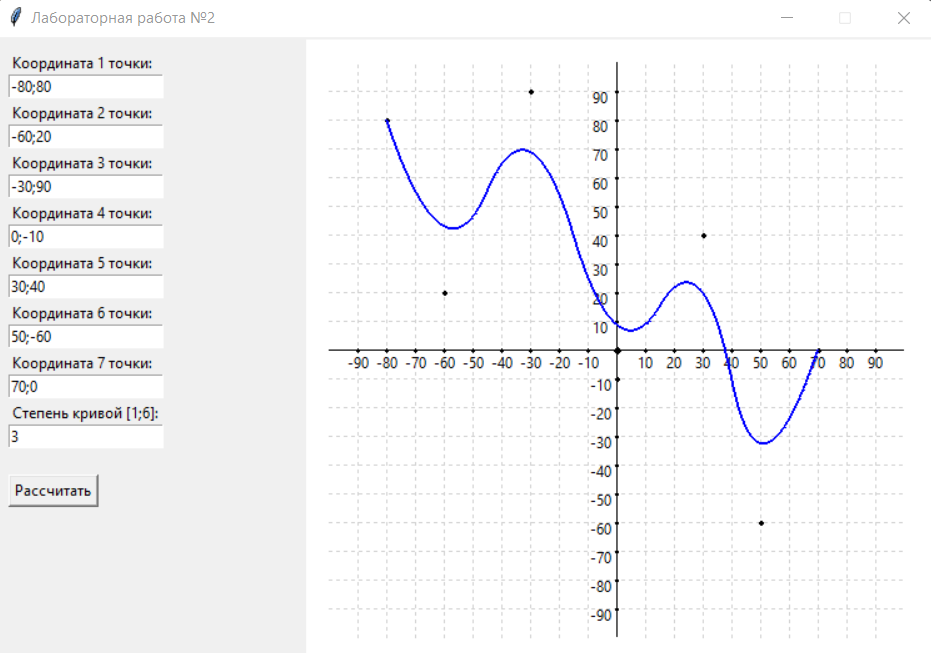
return sol

**Результаты лабораторной работы**

Программа была реализована на языке python с использованием графической библиотеки Tkinter.

Для работы программы нужно ввести координаты 7-ми точек, а также степень кривой и нажать кнопку “Рассчитать”.

В программе предусмотрена проверка: когда точка лежит за диапазоном, различные точки, некорректная степень кривизны. Также предусмотрена возможность редактирования входных параметров.



**Вывод**

В данной работе была реализована программа, которая строит на плоскости В-сплайновую кривую различной степени (1, 2, 3, 4, 5, 6) на основе 7 не повторяющихся задающих точек. С возможностью редактирования.

Значения по осям x и y: от -90 до 90.

**Приложение 1: исходный код программы**

import tkinter as tk

from tkinter.messagebox import showerror

import numpy as np

def axis\_y(k):

for i in range(457, 20, -23):

if i == 250:

continue

if k == 0:

k += 10

canvas.create\_line(20, i, 480, i, width=1, fill="lightgrey", dash=(2, 2))

canvas.create\_oval(250, i, 250, i, width=2)

canvas.create\_text(237, i + 5, text=k)

k += 10

def axis\_x(k):

for i in range(43, 480, 23):

if i == 250:

continue

if k == 0:

k += 10

canvas.create\_line(i, 480, i, 20, width=1, fill="lightgrey", dash=(2, 2))

canvas.create\_oval(i, 250, i, 250, width=2)

canvas.create\_text(i, 260, text=k)

k += 10

def axis():

axis\_y(-90)

axis\_x(-90)

def draw():

canvas.create\_line(250, 20, 250, 480, width=1, fill="black") # y

canvas.create\_line(20, 250, 480, 250, width=1, fill="black") # x

axis()

canvas.create\_oval(250, 250, 250, 250, width=4)

def checkX(point):

x = (point - 250) / 2.3

if x > 90 or x < -90:

return False

return True

def checkY(point):

y = (250 - point) / 2.3

if y > 90 or y < -90:

return False

return True

# Получаем введенные данные каждой из точек и возвращаем их раздельно по х и у

def extract\_numbers(s):

parts = s.split(";")

return 250 + float(int(parts[0])) \* 2.3, 250 - float(int(parts[1])) \* 2.3

# Формируем вектор узлов по количеству точек и степени кривизны

def knot\_values(n, k):

# Количество узлов по формуле i = n + k + 1

t = np.zeros((1, n + k + 1))

for i in range(0, n + k + 1):

if i < k:

t[0][i] = 0

if k <= i <= n:

t[0][i] = i - k + 1

if i > n:

t[0][i] = n - k + 2

return t

# Расчет коэффициентов B-сплайна

# Расчет рекурсивный

def basis\_spline(u, t, i, k):

# Если степень кривизны 1

if k == 1:

if t[0][i] <= u < t[0][i + 1]:

sol = 1

else:

sol = 0

# Степень кривизны больше 1

# i - 1 из 7 рассматриваемых точек

else:

a = (u - t[0][i]) \* basis\_spline(u, t, i, k - 1)

b = t[0][i + k - 1] - t[0][i]

c = (t[0][i + k] - u) \* basis\_spline(u, t, i + 1, k - 1)

d = t[0][i + k] - t[0][i + 1]

if b == 0:

temp1 = 0

else:

temp1 = a / b

if d == 0:

temp2 = 0

else:

temp2 = c / d

sol = temp1 + temp2

return sol

# Генератор кривой (получаем новые значения для построения)

def curve\_generator(n, k, ctrl\_x, ctrl\_y):

# Формируем вектор узлов по количеству точек и степени кривизны

t = knot\_values(n, k)

print("t: ", t)

# Массив чисел с шагом в 0.001 от 0 до n - k + 1, где n - количество точек, k - кривизна

u = np.arange(t[0][k - 1], t[0][n + 1], 0.001)

print("u: ", )

length = u.shape

# Массивы нулей для заполнения

x = np.zeros((1, length[0]))

y = np.zeros((1, length[0]))

# Заполняем массивы

# ctrl x и y координаты оригинальных введенных точек

for i in range(0, n + 1):

for j in range(0, length[0]):

x[0][j] = x[0][j] + basis\_spline(u[j], t, i, k) \* ctrl\_x[i]

y[0][j] = y[0][j] + basis\_spline(u[j], t, i, k) \* ctrl\_y[i]

print("x: ", x)

print("y: ", y)

return x, y

# Рисование кривой через Canvas

def draw\_spline\_curve(canvas, x, y):

num\_points = len(x)

for i in range(1, num\_points):

canvas.create\_line(x[i - 1], y[i - 1], x[i], y[i], fill="blue", width=2)

# Главная функция

def start\_func():

point1 = extract\_numbers(point1\_entry.get())

point2 = extract\_numbers(point2\_entry.get())

point3 = extract\_numbers(point3\_entry.get())

point4 = extract\_numbers(point4\_entry.get())

point5 = extract\_numbers(point5\_entry.get())

point6 = extract\_numbers(point6\_entry.get())

point7 = extract\_numbers(point7\_entry.get())

k = int(k\_entry.get())

canvas.delete("all")

draw()

canvas.create\_oval(point1[0], point1[1], point1[0], point1[1], width=3)

canvas.create\_oval(point2[0], point2[1], point2[0], point2[1], width=3)

canvas.create\_oval(point3[0], point3[1], point3[0], point3[1], width=3)

canvas.create\_oval(point4[0], point4[1], point4[0], point4[1], width=3)

canvas.create\_oval(point5[0], point5[1], point5[0], point5[1], width=3)

canvas.create\_oval(point6[0], point6[1], point6[0], point6[1], width=3)

canvas.create\_oval(point7[0], point7[1], point7[0], point7[1], width=3)

ctrl\_x = np.array([point1[0], point2[0], point3[0], point4[0], point5[0], point6[0], point7[0]])

ctrl\_y = np.array([point1[1], point2[1], point3[1], point4[1], point5[1], point6[1], point7[1]])

n = len(ctrl\_x) - 1

if (not checkY(point1[1]) or not checkX(point1[0])) or (not checkY(point2[1]) or not checkX(point2[0])) \

or (not checkY(point3[1]) or not checkX(point3[0])) or (not checkY(point4[1]) or not checkX(point4[0])) \

or (not checkY(point5[1]) or not checkX(point5[0])) or (not checkY(point6[1]) or not checkX(point6[0])) \

or (not checkY(point7[1]) or not checkX(point7[0])):

showerror(title="Ошибка",

message="Точка находится вне диапазона! Введите корректные координаты.")

return 1

if not (1 <= k <= 6):

showerror(title="Ошибка",

message="Некорректная степень кривой! Введите корректные данные.")

return 1

for i in range(len(ctrl\_x)):

for j in range(len(ctrl\_y)):

if ctrl\_x[i] == ctrl\_x[j] and ctrl\_y[i] == ctrl\_y[j] and i != j:

showerror(title="Ошибка",

message="Есть точки с одинаковыми координатами! Введите корректные координаты.")

return 1

# Генератор кривой, откуда получим значения для новых точек

[x, y] = curve\_generator(n, k, ctrl\_x, ctrl\_y)

n = x.shape

a = [0] \* n[1]

b = [0] \* n[1]

for i in range(0, n[1]):

a[i] = x[0][i]

b[i] = y[0][i]

# Рисует новую кривую

draw\_spline\_curve(canvas, a, b)

root = tk.Tk()

root.title("Лабораторная работа №2")

root.geometry("750x500")

root.resizable(False, False)

point1\_label = tk.Label(root, text="Координата 1 точки:")

point1\_label.place(x=10, y=10)

point1\_entry = tk.Entry(root)

point1\_entry.place(x=10, y=30)

point2\_label = tk.Label(root, text="Координата 2 точки:")

point2\_label.place(x=10, y=50)

point2\_entry = tk.Entry(root)

point2\_entry.place(x=10, y=70)

point3\_label = tk.Label(root, text="Координата 3 точки:")

point3\_label.place(x=10, y=90)

point3\_entry = tk.Entry(root)

point3\_entry.place(x=10, y=110)

point4\_label = tk.Label(root, text="Координата 4 точки:")

point4\_label.place(x=10, y=130)

point4\_entry = tk.Entry(root)

point4\_entry.place(x=10, y=150)

point5\_label = tk.Label(root, text="Координата 5 точки:")

point5\_label.place(x=10, y=170)

point5\_entry = tk.Entry(root)

point5\_entry.place(x=10, y=190)

point6\_label = tk.Label(root, text="Координата 6 точки:")

point6\_label.place(x=10, y=210)

point6\_entry = tk.Entry(root)

point6\_entry.place(x=10, y=230)

point7\_label = tk.Label(root, text="Координата 7 точки:")

point7\_label.place(x=10, y=250)

point7\_entry = tk.Entry(root)

point7\_entry.place(x=10, y=270)

k\_label = tk.Label(root, text="Степень кривой [1;6]:")

k\_label.place(x=10, y=290)

k\_entry = tk.Entry(root)

k\_entry.place(x=10, y=310)

calculate\_button = tk.Button(root, text="Рассчитать", command=start\_func)

calculate\_button.place(x=10, y=350)

canvas = tk.Canvas(root, width=500, height=500, bg="white")

canvas.pack(anchor="e")

draw()

root.mainloop()