ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»

Факультет программной инженерии и компьютерной техники

Дисциплина:

«Вычислительная математика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №5

Вариант 10.

Выполнил:

Студент гр. P32151

Понамарев Степан Андреевич

Проверил:

Машина Екатерина Алексеевна

Санкт-Петербург, 2023

Цель работы

Решить задачу интерполяции, найти значения функции при заданных значениях аргумента, отличных от узловых точек.

Вычислительная реализация

1. Таблица конечных разностей.

i	x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
0	2,10	3,7587	0,4274	0,3083	-0,6171	1,0778	-1,7774	2,9757
1	2,15	4,1861	0,7357	-0,3088	0,4607	-0,6996	1,1983	
2	2,20	4,9218	0,4269	0,1519	-0,2389	0,4987		
3	2,25	5,3487	0,5788	-0,087	0,2598			
4	2,30	5,9275	0,4918	0,1728				
5	2,35	6,4193	0,6646					
6	2,40	7,0839						

2. Для вычисления значения функции в точке $X_1 = 2{,}355$ используем вторую интерполяционную формулу Ньютона:

$$t = \frac{x - x_n}{h} = \frac{2{,}355 - 2.40}{0.05} = -0{,}9$$

$$Y_1 = N_6(x) = y_6 + \Delta y_5 t + \frac{\Delta^2 y_4 t(t+1)}{2!} + \frac{\Delta^3 y_3 t(t+1)(t+2)}{3!} + \frac{\Delta^4 y_2 t(t+1)(t+2)(t+3)}{4!} + \frac{\Delta^5 y_1 t(t+1)(t+2)(t+3)(t+4)}{5!} + \frac{\Delta^6 y_0 t(t+1)(t+2)(t+3)(t+4)(t+5)}{6!}$$

$$N_{6(x)} = 7{,}08390 - 0{,}59814 - 0{,}00778 - 0{,}00429 - 0{,}00432 - 0{,}00644 - 0{,}01092 = \mathbf{6{,}45202}$$

3. Для вычисления значения функции в точке $X_2 = 2{,}254$ используем первую интерполяционную формулу Гаусса для интерполирования вперёд ($2{,}254 = X_2 > a = 2{,}25$ – центральная точка):

$$t = \frac{X_2 - a}{h} = \frac{2.254 - 2.250}{0.05} = 0{,}08$$

$$Y_2 = P_6(x) = y_0 + \Delta y_0 t + \frac{t(t-1)}{2!}\Delta^2 y_{-1} + \frac{(t+1)t(t-1)}{3!}\Delta^3 y_{-1} + \frac{(t+1)t(t-1)(t-2)}{4!}\Delta^4 y_{-2} + \frac{(t-2)(t-1)t(t+1)(t+2)}{5!}\Delta^5 y_{-2} + \frac{(t+2)(t+1)t(t-1)(t-2)(t-3)}{6!}\Delta^6 y_{-3}$$

$$P_6(x) = 5{,}3487 + 0{,}0463 - 0{,}00559 + 0{,}00316 - 0{,}00445 + 0{,}00317 - 0{,}00383 = \mathbf{5{,}38747}$$

Листинг программы

Main.py:

```
import numpy as np

from InputManager import InputManager
from solvers import Lagrange, Gaussian, StirlingAndBessel,
PreparedFunctionSolver

working_flag = True
points = []


def stop_program():
    global working_flag
    working_flag = False


def sort_and_delete_dublicates(points):
    points.sort()
    i = 1
    while i < len(points):
        if points[i][0] == points[i - 1][0]:
            points[i - 1] = (points[i - 1][0], (points[i - 1][1] +
points[i][1]) / 2)
            points.pop(i)
        else:
            i += 1
    return points


def enter_points():
    global points
    if InputManager.yes_or_no_input("Хотите считать точки из
файла?"):
        while True:
            filename = InputManager.string_input("Введите название
файла: ")
            try:
                file = open(filename, "r")
                n = int(file.readline())
                if n < 2:
                    print("Точек должно быть не менее двух!")
                    raise ValueError
                points = []
                for i in range(n):
                    x, y = map(float, file.readline().split())
                    points.append((x, y))
                file.close()
                break
```

```
            except FileNotFoundError:
                print("Не удалось найти указанный файл.")
            except ValueError:
                print("Не удалось считать данные из файла.")
    else:
        n = InputManager.int_input("Введите количество точек: ")
        while n < 2:
            print("Точек должно быть не менее двух!")
            n = InputManager.int_input("Введите количество точек:
")
        points = []
        print(f"Введите x y координаты {n} точек через пробел.")
        for i in range(n):
            x, y = InputManager.point_input(f"Координаты точки {i +
1}: ")
            points.append((x, y))

    points = sort_and_delete_dublicates(points)

    if len(points) < 2:
        print("Различных точек должно быть не менее двух!")
        return enter_points()
    print("Исходный набор точек:", points)
    return points


def save_points():
    global points
    filename = InputManager.string_input("Введите название файла:
")
    try:
        with open(filename, "w") as f:
            f.writelines(str(len(points)) + "\n")
            f.writelines([f"{x} {y}\n" for x, y in points])
    except Exception:
        print("Не удалось сохранить точки в указанный файл.")


def lagrange_solver():
    solver = Lagrange(points)
    solver.solve()


def gaussian_solver():
    solver = Gaussian(points)
    solver.solve()


def prepared_function_solver():
    global points
    variants = ["y = sin(x)",
```

```
                "y = x * e^(sin(3x/5)) - 3.44",
                "y = x^3 - 3x^2 + 2"]
    values = [PreparedFunctionSolver(lambda x: np.sin(x)),
              PreparedFunctionSolver(lambda x: x * np.exp(np.sin(3
/ 5 * x)) - 3.44),
              PreparedFunctionSolver(lambda x: x ** 3 - 3 * x ** 2
+ 2)]

    chosen_variant = InputManager.multiple_choice_input(variants,
values, "Выберите функцию:")

    if InputManager.yes_or_no_input("Показать график выбранной
функции?"):
        chosen_variant.draw_init_graphic()

    n = InputManager.int_input("Введите количество точек для
интерполирования: ")
    while n < 2:
        print("Точек должно быть не менее двух!")
        n = InputManager.int_input("Введите количество точек для
интерполирования: ")
    points_x = []
    points = []
    for i in range(n):
        x = InputManager.float_input(f"Введите x-координату точки
{i + 1}: ")
        while x in points_x:
            print("Такая координата уже была.")
            x = InputManager.float_input(f"Введите x-координату
точки {i + 1}: ")
        points.append((x, chosen_variant.f(x)))
        points_x.append(x)
    del points_x
    points = sort_and_delete_dublicates(points)
    solver = Lagrange(points)

    solver.solve(draw_graphic=False)

    chosen_variant.draw_two_graphics(solver.f, points)


def stirling_and_bessel_solver():
    solver = StirlingAndBessel(points)
    solver.solve()


if __name__ == '__main__':
    variants = ["Ввести точки", "Интерполировать функцию из
предложенных"] + ["Выйти из программы"]
    values = [enter_points, prepared_function_solver] +
[stop_program]
```

```
    chosen_variant = InputManager.multiple_choice_input(variants,
values, "Выберите нужный вариант:")
    chosen_variant()

    variants = ["Ввести точки заново", "Сохранить введённый набор
точек", "Решить методом Лагранжа",
                "Решить методом Гаусса", "Интерполировать функцию
из предложенных",
                "Вычислить значение функции, используя схемы
Стирлинга и Бесселя"] + ["Выйти из программы"]
    values = [enter_points, save_points, lagrange_solver,
gaussian_solver, prepared_function_solver,
              stirling_and_bessel_solver] + [stop_program]

    while working_flag:
        chosen_variant =
InputManager.multiple_choice_input(variants, values, "Выберите
нужный вариант:")
        chosen_variant()
        print()
```

InputManager.py:

```
class InputManager:
    @staticmethod
    def string_input(message=""):
        buf = ""
        while buf == "":
            buf = input(message).strip()
        return buf

    @staticmethod
    def _check_number(buf):
        try:
            float(buf.replace(',', '.'))
            return True
        except ValueError:
            return False

    @staticmethod
    def _convert_to_number(num):
        try:
            return float(num.replace(',', '.'))
        except ValueError:
            return None

    @staticmethod
    def float_input(message=""):
        number = None
        while number is None:
```

```
            number =
InputManager._convert_to_number(InputManager.string_input(message))
        return number

    @staticmethod
    def int_input(message=""):
        return int(InputManager.float_input(message))

    @staticmethod
    def yes_or_no_input(message=""):
        answer = "0"
        while answer[0].lower() not in ["y", "n", "д", "н"]:
            answer = InputManager.string_input(message + " [y/n]:
")
        return answer[0].lower() in ["y", "д"]

    @staticmethod
    def enum_input(variants_list, message=""):
        variants_list = [str(i) for i in variants_list]
        buf = ""
        while buf not in variants_list:
            buf = InputManager.string_input(message)
        return buf

    @staticmethod
    def multiple_choice_input(variant_list, values_list,
message=""):
        n = len(variant_list)
        if n == 0 or len(variant_list) != len(values_list):
            raise ValueError

        if message != "":
            print(message)

        for i in range(n):
            s = f"{i + 1}."
            lines = variant_list[i].split('\n')
            print('\t' + s, lines[0])
            for line in lines[1:]:
                print("\t" + ' ' * len(s), line)

        i = int(InputManager.enum_input([*range(1, n + 1)],
f"Введите число от 1 до {n}: "))
        return values_list[i - 1]

    @staticmethod
    def epsilon_input(message=""):
        e = InputManager.float_input(message)
        while not (0 < e and e <= 1):
            print("Эпсилон должно быть в промежутке от 0 до 1!")
            e = InputManager.float_input(message)
```

```
        return e

    @staticmethod
    def int_input_with_borders(left, right, message=""):
        if left >= right:
            raise ValueError
        if message != "":
            print(message)
        i = int(InputManager.enum_input([*range(left, right + 1)],
f"Введите число от {left} до {right}: "))
        return i

    @staticmethod
    def point_input(message=""):
        x, y = None, None
        while x is None or y is None:
            line = InputManager.string_input(message).split()
            if len(line) != 2:
                continue
            x, y = InputManager._convert_to_number(line[0]),
InputManager._convert_to_number(line[1])
        return x, y

    @staticmethod
    def float_input_with_borders(left, right, message=""):
        if not left < right:
            raise ValueError
        x = InputManager.float_input(message)
        while not (left < x < right):
            print(f"Значение должно быть от {left} до {right}!")
            x = InputManager.float_input(message)
        return x
```

solvers.py:

```
import math
from abc import abstractmethod
from functools import lru_cache

import numpy as np
from matplotlib import pyplot as plt
from prettytable import PrettyTable

from InputManager import InputManager


def multiply(arr):
    res = 1
    for i in arr:
        res *= i
    return res
```

```
class Solver:
    def __init__(self, points):
        self.f = lambda x: 0
        self.points_x = [p[0] for p in points]
        self.points_y = [p[1] for p in points]
        self.n = len(points)

    def draw_graphics(self, plot_name=""):
        X = np.linspace(min(self.points_x), max(self.points_x),
100)
        Y = [self.f(x) for x in X]
        fig, ax = plt.subplots(1, 1)
        # ax.set_ylim([min(self.points_y) - 1, max(self.points_y) +
1])
        ax.scatter(self.points_x, self.points_y, c='r', marker='o')
        ax.plot(X, Y)
        plt.title(plot_name)
        plt.show()

    @abstractmethod
    def solve(self):
        pass


class Lagrange(Solver):

    def solve(self, draw_graphic=True):
        one_brace = lambda x, i, j: (x - self.points_x[j]) /
(self.points_x[i] - self.points_x[j])
        l_i = lambda x, i: multiply([one_brace(x, i, j) if i != j
else 1 for j in range(self.n)])
        L_n = lambda x: sum(np.multiply(self.points_y, [l_i(x, i)
for i in range(self.n)]))
        self.f = L_n
        if draw_graphic:
            self.draw_graphics("Интерполяционный многочлен
Лагранжа")


class Gaussian(Solver):

    @lru_cache
    def calculate_gaussian_delta_y(self, d, i):
        if abs(i) > len(self.points_x) // 2 or d > self.n:
            return None
        if self.n // 2 + i + d >= self.n:
            return None

        if d == 0:
            return self.points_y[i + self.n // 2]
```

```
        return self.calculate_gaussian_delta_y(d - 1, i + 1) -
self.calculate_gaussian_delta_y(d - 1, i)

    @staticmethod
    @lru_cache(maxsize=None)
    def get_first_t_coefficient(t, n):
        """
        Возвращает число (в зависимости от n)
        для n = 0, n = 1, и т.д.:
        1, t, t(t-1), (t+1)t(t-1), (t+1)t(t-1)(t-2), ...
        """
        if n == 0:
            return 1
        res = 1
        for i in range((n + 1) // 2):
            res *= (t + i)
        for i in range(1, n // 2 + 1):
            res *= (t - i)
        return res / math.factorial(n)

    @staticmethod
    @lru_cache(maxsize=None)
    def get_second_t_coefficient(t, n):
        """
        1, t, (t+1)t, (t+1)t(t-1), (t+1)(t+2)t(t-1), ...
        """
        if n == 0:
            return 1
        res = 1
        for i in range((n + 1) // 2):
            res *= (t - i)
        for i in range(1, n // 2 + 1):
            res *= (t + i)
        return res / math.factorial(n)

    def solve(self):

        h = self.points_x[1] - self.points_x[0]

        for i in range(1, self.n - 1):
            if self.points_x[i] - self.points_x[i - 1] - h > 10 **
(-9):
                print("Невозможно применить метод Гаусса. Узлы не
равноотстоящие.")
                return

        if len(self.points_x) % 2 == 0:
            print("Невозможно применить метод Гаусса. Количество
точек должно быть нечётным.")
            return
```

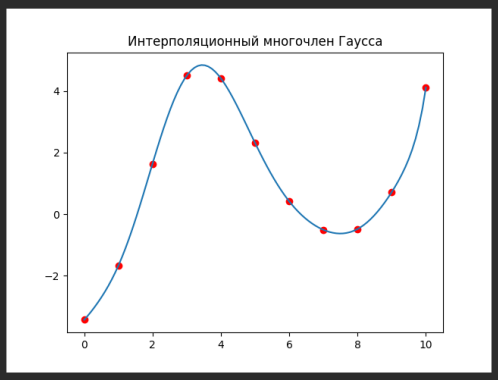
```
        self.calculate_gaussian_delta_y(self.n - 1, -(self.n // 2))
        table = PrettyTable(["x_i", "y_i", "d y_i"] + [f"d^{i} y_i"
for i in range(2, self.n)])
        for i in range(self.n):
            table.add_row([self.points_x[i]] + ["-" if u is None
else round(u, 4) for u in

[self.calculate_gaussian_delta_y(j, i - self.n // 2) for j in
                                                    range(self.n)]])
        print("Таблица конечных разностей:")
        print(table)

        center_of_section = (self.points_x[0] + self.points_x[-1])
/ 2

        first_fg = lambda t: sum(
            [Gaussian.get_first_t_coefficient(t, i) *
self.calculate_gaussian_delta_y(i, -(i // 2))
             for i in range(self.n)])
        second_fg = lambda t: sum(
            [Gaussian.get_second_t_coefficient(t, i) *
self.calculate_gaussian_delta_y(i, (-i // 2))
             for i in range(self.n)])

        self.f = lambda x: first_fg((x - center_of_section) / h) if
x < center_of_section else second_fg(
            (x - center_of_section) / h)

        if InputManager.yes_or_no_input("Показать график
функции?"):
            self.draw_graphics("Интерполяционный многочлен Гаусса")

        if InputManager.yes_or_no_input("Хотите узнать значение
функции в конкретной точке?"):
            x =
InputManager.float_input_with_borders(self.points_x[0],
self.points_x[-1],
                                                    "Введите
координату x точки: ")
            t = (x - center_of_section) / h
            print("Значение функции в точке x:", self.f(x))

            # погрешность считается по формуле из лекции
            print("Оценка погрешности:",
abs(self.calculate_gaussian_delta_y(self.n - 1, -(self.n // 2)) *
multiply(
                [(t - i) for i in range(self.n // 2)]) /
math.factorial(self.n // 2 + 1)))
```

```
class StirlingAndBessel(Solver):
    def solve(self):
        print("Интерполяция по схемам Стирлинга и Бесселя пока не
реализована.")


class PreparedFunctionSolver:
    def __init__(self, f):
        self.f = f
        self.interpolated_function = lambda x: 0

    def draw_init_graphic(self):
        X = np.linspace(-5, 5, 100)
        Y = self.f(X)
        fig, ax = plt.subplots(1, 1)
        ax.plot(X, Y)
        ax.grid(True)
        plt.title("Выбранная функция")
        plt.show()

    def draw_two_graphics(self, f_second, points):
        X = np.linspace(points[0][0], points[-1][0], 100)
        Y1 = self.f(X)
        Y2 = [f_second(x) for x in X]
        fig, ax = plt.subplots(1, 1)
        ax.scatter([p[0] for p in points], [p[1] for p in points],
c='r', marker='o')
        ax.plot(X, Y2, c='g')
        ax.plot(X, Y1, color=(0, 0, 0, 0.2), linestyle='-.')
        ax.grid(True)
        plt.title("Результат интерполяции:")
        plt.show()
```

Примеры и результат работы программы

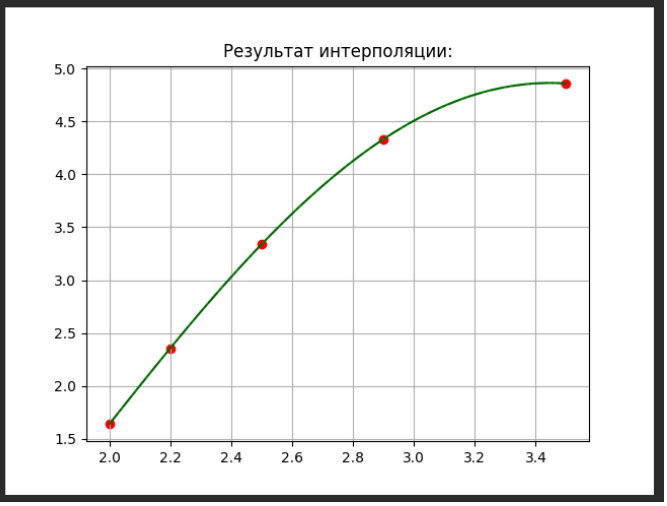
Пример 1

```
Выберите нужный вариант:
    1. Ввести точки
    2. Интерполировать функцию из предложенных
    3. Выйти из программы
Введите число от 1 до 3: 1
Хотите считать точки из файла? [y/n]: y
Введите название файла: examples/gauss1.txt
Исходный набор точек: [(0.0, -3.44), (1.0, -1.6812), (2.0, 1.6394), (3.0, 4.5043), (4.0, 4.4198), (5.0, 2.3178), (6.0,
 0.4145), (7.0, -0.512), (8.0, -0.4857), (9.0, 0.7156), (10.0, 4.1223)]
Выберите нужный вариант:
    1. Ввести точки заново
    2. Сохранить введённый набор точек
    3. Решить методом Лагранжа
    4. Решить методом Гаусса
    5. Интерполировать функцию из предложенных
    6. Вычислить значение функции, используя схемы Стирлинга и Бесселя
    7. Выйти из программы
Введите число от 1 до 7: 4
Таблица конечных разностей:
+------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+----------+
| x_i  |   y_i   |  d y_i  | d^2 y_i | d^3 y_i | d^4 y_i | d^5 y_i | d^6 y_i | d^7 y_i | d^8 y_i | d^9 y_i | d^10 y_i |
+------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+----------+
| 0.0  |  -3.44  |  1.7588 |  1.5618 | -2.0175 | -0.4762 |  3.9018 | -6.0431 |  5.462  | -1.5225 | -5.3631 | 14.2962  |
| 1.0  | -1.6812 |  3.3206 | -0.4557 | -2.4937 |  3.4256 | -2.1413 | -0.5811 |  3.9395 | -6.8856 |  8.9331 |    -     |
| 2.0  |  1.6394 |  2.8649 | -2.9494 |  0.9319 |  1.2843 | -2.7224 |  3.3584 | -2.9461 |  2.0475 |    -    |    -     |
| 3.0  |  4.5043 | -0.0845 | -2.0175 |  2.2162 | -1.4381 |  0.636  |  0.4123 | -0.8986 |    -    |    -    |    -     |
| 4.0  |  4.4198 |  -2.102 |  0.1987 |  0.7781 | -0.8021 |  1.0483 | -0.4863 |    -    |    -    |    -    |    -     |
| 5.0  |  2.3178 | -1.9033 |  0.9768 |  -0.024 |  0.2462 |  0.562  |    -    |    -    |    -    |    -    |    -     |
| 6.0  |  0.4145 | -0.9265 |  0.9528 |  0.2222 |  0.8082 |    -    |    -    |    -    |    -    |    -    |    -     |
| 7.0  |  -0.512 |  0.0263 |  1.175  |  1.0304 |    -    |    -    |    -    |    -    |    -    |    -    |    -     |
| 8.0  | -0.4857 |  1.2013 |  2.2054 |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -     |
| 9.0  |  0.7156 |  3.4067 |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -     |
| 10.0 |  4.1223 |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -    |    -     |
+------+---------+---------+---------+---------+---------+---------+---------+---------+---------+---------+----------+
Показать график функции? [y/n]: y
Хотите узнать значение функции в конкретной точке? [y/n]:
```



Пример 2

```
Выберите нужный вариант:
    1. Ввести точки заново
    2. Сохранить введённый набор точек
    3. Решить методом Лагранжа
    4. Решить методом Гаусса
    5. Интерполировать функцию из предложенных
    6. Вычислить значение функции, используя схемы Стирлинга и Бесселя
    7. Выйти из программы
Введите число от 1 до 7: 5
Выберите функцию:
    1. y = sin(x)
    2. y = x * e^(sin(3x/5)) - 3.44
    3. y = x^3 - 3x^2 + 2
Введите число от 1 до 3: 2
Показать график выбранной функции? [y/n]: y
Введите количество точек для интерполирования: 5
Введите x-координату точки 1: 2
Введите x-координату точки 2: 2.2
Введите x-координату точки 3: 2.5
Введите x-координату точки 4: 2.9
Введите x-координату точки 5: 3.5

Выберите нужный вариант:
```



Выводы

Я научился строить интерполяцию по табличным данным, узнал методы интерполяции и реализовал их в программном коде.