Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №5

по «Вычислительной математике»

Вариант 4

Выполнил:

Студент группы P3208

Дашкевич Егор Вячеславович

Преподаватели:

Машина Екатерина Алексеевна

Санкт-Петербург

2024

Оглавление

Цель работы

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

Задание:

<u>Обязательное задание (до 80 баллов)</u>

Вычислительная реализация задачи:

1. Выбрать из табл. 1 заданную по варианту таблицу $y = f(x)$ (таблица 1.1 – таблица 1.5);

2. Построить таблицу конечных разностей для заданной таблицы. Таблицу отразить в отчете;

3. Вычислить значения функции для аргумента $X1$ (см. табл.1), используя первую или вторую интерполяционную формулу Ньютона. Обратить внимание какой конкретно формулой необходимо воспользоваться;

4. Вычислить значения функции для аргумента $X2$ (см. табл. 1), используя первую или вторую интерполяционную формулу Гаусса. Обратить внимание какой конкретно формулой необходимо воспользоваться;

5. Подробные вычисления привести в отчете.

Программная реализация задачи:

1. Исходные данные задаются тремя способами:

a) в виде набора данных (таблицы x, y), пользователь вводит значения с клавиатуры;

b) в виде сформированных в файле данных (подготовить не менее трех тестовых вариантов);

c) на основе выбранной функции, из тех, которые предлагает программа, например, $\sin x$. Пользователь выбирает уравнение, исследуемый интервал и количество точек на интервале (не менее двух функций).

2. Сформировать и вывести таблицу конечных разностей;

3. Вычислить приближенное значение функции для заданного значения аргумента, введенного с клавиатуры, указанными методами (см. табл. 2). Сравнить полученные значения;

4. Построить графики заданной функции с отмеченными узлами интерполяции и интерполяционного многочлена Ньютона/Гаусса (разными цветами);

5. Программа должна быть протестирована на различных наборах данных, в том числе и некорректных.

6. Проанализировать результаты работы программы.

Необязательное задание (до 20 баллов)

1. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Стирлинга;

2. Реализовать в программе вычисление значения функции для заданного значения аргумента, введенного с клавиатуры, используя схемы Бесселя.

Вычислительная часть

Исходные данные:

x_i	**1.05**	**1.15**	**1.25**	**1.35**	**1.45**	**1.55**	**1.65**
y_i	**0.1213**	**1.1316**	**2.1459**	**3.1565**	**4.1571**	**5.1819**	**6.1969**

$$X_1 = 1,051, \qquad X_2 = 1,277$$

Таблица конечных разностей:

x_i	y_i	Δy_i	$\Delta^2 y_i$	$\Delta^3 y_i$	$\Delta^4 y_i$	$\Delta^5 y_i$	$\Delta^6 y_i$
1.05	**0.1213**	**1.0103**	**0.004**	**-0.0077**	**0.0014**	**0.0391**	**-0.1478**
1.15	**1.1316**	**1.0143**	**-0.0037**	**-0.0063**	**0.0405**	**-0.1087**	
1.25	**2.1459**	**1.0106**	**-0.01**	**0.0342**	**-0.0682**		
1.35	**3.1565**	**1.0006**	**0.0242**	**-0.034**			

1.45	**4.1571**	**1.0248**	**-0.0098**				
1.55	**5.1819**	**1.015**					
1.65	**6.1969**						

Метод Ньютона:

$$t = \frac{x - x_0}{h} = \frac{1.051 - 1.05}{0.1} = 0.01$$

$$N_6(x) = y_0 + t\Delta y_0 + \frac{t(t-1)}{2!}\Delta^2 y_0 + \frac{t(t-1)(t-2)}{3!}\Delta^3 y_0 + \frac{t(t-1)(t-2)(t-3)}{4!}\Delta^4 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)}{5!}\Delta^5 y_0 + \frac{t(t-1)(t-2)(t-3)(t-4)(t-5)}{6!}\Delta^6 y_0$$

$$y(X_1) = 0.1213 + (0.01 * 1.0103) + 0.004\frac{0.01(0.01-1)}{2} - 0.0077\frac{0.01(0.01-1)(0.01-2)}{6} + 0.0014\frac{0.01(0.01-1)(0.01-2)(0.01-3)}{24} + 0.0391\frac{0.01(0.01-1)(0.01-2)(0.01-3)(0.01-4)}{120} - 0.1478\frac{0.01(0.01-1)(0.01-2)(0.01-3)(0.01-4)(0.01-5)}{720} \approx 0.1317$$

Метод Гаусса:

$$t = \frac{x - x_3}{h} = \frac{1.277 - 1.35}{0.1} = -0.73$$

$$P_6(x) = y_3 + t\Delta y_2 + \frac{t(t+1)}{2!}\Delta^2 y_2 + \frac{t(t+1)(t-1)}{3!}\Delta^3 y_1 + \frac{t(t+1)(t-1)(t+2)}{4!}\Delta^4 y_1 + \frac{t(t+1)(t-1)(t+2)(t-2)}{5!}\Delta^5 y_0 + \frac{t(t+1)(t-1)(t+2)(t-2)(t+3)}{6!}\Delta^6 y_0$$

$$y(X_2) = 3.1565 - 0.73 * 1.0106 + 0.01\frac{0.73 * 0.27}{2} - 0.0063\frac{0.7 * 0.27 * 1.73}{6} + 0.0405\frac{0.7 * 0.27 * 1.73 * 1.27}{24} - 0.0391\frac{0.7 * 0.27 * 1.73 * 1.27 * 2.73}{120} + 0.1478\frac{0.7 * 0.27 * 1.73 * 1.27 * 2.73 * 2.27}{720} \approx 2.4203$$

Листинг программы:

Лагранж:

```
def __init__(self, points):
    super().__init__(points, list(zeros(len(points))),  name: "lagrange")

    for i in range(len(points)):
        numerator = []
        buff = points[i][1]

        for j in range(len(points)):
            if i == j:
                continue
            buff /= points[i][0] - points[j][0] # считаем знаменатель
            numerator.append(-1 * points[j][0]) # собираем числитель

        _polynom = expand_brackets(numerator)
        _polynom = [elem * buff for elem in _polynom]
        self.koofs = [self.koofs[i] + _polynom[i] for i in range(len(self.koofs))]
```

Ньютон:

```python
def __init__(self, points):
    super().__init__(points, list(zeros(len(points))), name: "newton")

    self.tree.append([])

    for i in range(1, len(points)):
        self.tree[-1].append((points[i][1] - points[i - 1][1]) / (points[i][0] - points[i - 1][0]))

    for depth in range(1, len(points) - 1):
        self.tree.append([])
        left, right = 0, depth + 1
        for i in range(1, len(self.tree[-2])):
            self.tree[-1].append((self.tree[-2][i] - self.tree[-2][i - 1]) /
                                 (points[right][0] - points[left][0]))
            right += 1
            left += 1

    self.koofs[0] = points[0][1]
    for i in range(len(points) - 1):
        _polynom = [-1 * points[j][0] for j in range(i + 1)]

        _polynom = expand_brackets(_polynom)
        _polynom = [elem * self.tree[i][0] for elem in _polynom]

        while len(_polynom) < len(self.koofs):
            _polynom = _polynom + [0]
        self.koofs = [self.koofs[i] + _polynom[i] for i in range(len(self.koofs))]
```


```python
    self.koofs[0] = points[0][1]
    for i in range(len(points) - 1):
        _polynom = [-1 * points[j][0] for j in range(i + 1)]

        _polynom = expand_brackets(_polynom)
        _polynom = [elem * self.tree[i][0] for elem in _polynom]

        while len(_polynom) < len(self.koofs):
            _polynom = _polynom + [0]
        self.koofs = [self.koofs[i] + _polynom[i] for i in range(len(self.koofs))]
```


```python
def print_tree(self):
    print()
    print_table_header(["x\\y"] + list(range(len(self.tree))))
    for x_i in range(len(self.tree)):
        buff = [x_i]
        for j in range(len(self.tree) - x_i):
            buff.append(self.tree[j][x_i])
        print_table_row(buff)
```

Ньютон для равноотстоящих углов:

```python
class Newton_Stable_Polynomial(Polynomial):
    h = 0
    tree = []

    def __init__(self, points):

        super().__init__(points, list(zeros(len(points))), name: "newton_stable")
        self.h = points[1][0] - points[0][0]

        self.tree.append([y for x, y in points])
        self.tree.append([])

        for i in range(1, len(points)):
            self.tree[-1].append(points[i][1] - points[i - 1][1])

        for depth in range(1, len(points) - 1):
            self.tree.append([])
            for i in range(1, len(self.tree[-2])):
                self.tree[-1].append(self.tree[-2][i] - self.tree[-2][i - 1])
```

```python
def calc_straight(self, x):
    t_idx = 0

    for i in range(len(self.points)):
        if (self.points[i][0] < x):
            t_idx = i
        else:
            break

    out = 0
    t = (x - self.points[t_idx][0]) / self.h
    for i in range(len(self.tree) - t_idx):
        buff = self.tree[i][t_idx]
        for j in range(i):
            buff *= t - j
        buff /= math.factorial(i)

        out += buff
    return out
```

```python
def calc_back(self, x):
    t_idx = 0

    for i in range(len(self.points) - 1, 0, -1):
        if self.points[i][0] > x:
            t_idx = i
        else:
            break

    out = 0
    t = (x - self.points[t_idx][0]) / self.h
    for i in range(len(self.tree)):
        if t_idx - i < 0:
            break
        buff = self.tree[i][t_idx - i]
        for j in range(0, i):
            buff *= t + j
        buff /= math.factorial(i)
        out += buff

    return out
```

Стирлинг:

```python
h = 0
tree = []

def __init__(self, points):
    super().__init__(points, list(zeros(len(points))), "stirling")
    self.h = points[1][0] - points[0][0]

    self.tree.append([y for x, y in points])
    self.tree.append([])

    for i in range(1, len(points)):
        self.tree[-1].append(points[i][1] - points[i - 1][1])

    for depth in range(1, len(points) - 1):
        self.tree.append([])
        for i in range(1, len(self.tree[-2])):
            self.tree[-1].append(self.tree[-2][i] - self.tree[-2][i - 1])
```

```python
def calc(self, x):
    xs = [i[0] for i in self.points]
    n = len(self.points) - 1
    alpha_ind = n // 2


    dts1 = [0, -1, 1, -2, 2, -3, 3, -4, 4, -5, 5, -6, 6, -7, 7, -8, 8]
    f1 = lambda x: self.tree[0][alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / self.h + dts1[j] for j in range(k)])
        * self.tree[k][len(self.tree[k]) // 2] / math.factorial(k)
        for k in range(1, n + 1)])
    f2 = lambda x: self.tree[0][alpha_ind] + sum([
        reduce(lambda a, b: a * b,
               [(x - xs[alpha_ind]) / self.h - dts1[j] for j in range(k)])
        * self.tree[k][len(self.tree[k]) // 2 - (1 - len(self.tree[k]) % 2)] / math.factorial(k)
        for k in range(1, n + 1)])
    return (f1(x) + f2(x)) / 2
```

Бессель:

```python
class Bessel_polynom(Polynomial):
    h = 0
    tree = []

    def __init__(self, points):
        super().__init__(points, list(zeros(len(points))),  name: "stirling")
        self.h = points[1][0] - points[0][0]

        self.tree.append([y for x, y in points])
        self.tree.append([])

        for i in range(1, len(points)):
            self.tree[-1].append(points[i][1] - points[i - 1][1])

        for depth in range(1, len(points) - 1):
            self.tree.append([])
            for i in range(1, len(self.tree[-2])):
                self.tree[-1].append(self.tree[-2][i] - self.tree[-2][i - 1])
```

```python
    def calc(self, x):
        xs = [i[0] for i in self.points]
        n = len(self.points) - 1
        alpha_ind = n // 2


        dts1 = [0, -1, 1, -2, 2, -3, 3, -4, 4, -5, 5, -6, 6, -7, 7]
        f = lambda x: (self.tree[0][alpha_ind] + self.tree[0][alpha_ind]) / 2 + sum([
            reduce(lambda a, b: a * b,
                   [(x - xs[alpha_ind]) / self.h + dts1[j] for j in range(k)])
            * self.tree[k][len(self.tree[k]) // 2] / math.factorial(2 * k) +
            ((x - xs[alpha_ind]) / self.h - 1 / 2) *
            reduce(lambda a, b: a * b,
                   [(x - xs[alpha_ind]) / self.h + dts1[j] for j in range(k)])
            * self.tree[k][len(self.tree[k]) // 2] / math.factorial(2 * k + 1)
            for k in range(1, n + 1)])
        return f(x)
```

Вывод

В ходе выполнения работы разобрал интерполяцию различными методами, реализовал их на ЭВМ