МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Костромской государственный университет»

(КГУ)

ИАСТ

Кафедра автоматизированных систем и технологий

09.03.02

Направление подготовки/Специальность Информационные системы и технологии

Дисциплина Численные методы

# Лабораторная №7.

# Интерполяция.

Выполнил студент

Копосов Лев Владимирович

Группа 22-ИСбо-1б

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_

Кострома

**Постановка задачи.**

Разработать алгоритм и программу, которые по 10 точкам, полученным в задании для решения вашего варианта ОДУ (любым методом) строят интерполяционный параболический полином.

****

Рис.1. Задача N19

**Краткая теория используемых методов.**

Интерполяционный параболический полином - это полином второй степени, который используется для интерполяции набора трех точек. Для построения параболического полинома, общая теория базируется на понятиях интерполяции Лагранжа и Ньютона. Перед построением полинома, применяются табличные разности для уменьшения сложности полинома. Табличные разности определяют степень полинома и указывают, что для его построения достаточно использовать первые n разделенных разностей.

1. Полином Лагранжа.

Интерполяционный многочлен Лагранжа - многочлен минимальной степени, принимающий данные значения в данном наборе точек. Для n+1 пар чисел , , …, , где все xi различны, существует единственный многочлен L(x) степени не более n, для которого L(xi) = yi.

В общем виде интерполяционный многочлен Лагранжа записывается в следующем виде:

(1.1),

где , i = 0, …, n - полином Лагранжа, вычисляемый как (1.2), и yi = y(xi), i = 0, …, n - значения функции в узлах интерполяции, то есть в значениях аргумента xi.

1. Полином Ньютона.

Интерполяционный многочлен Ньютона - это многочлен n-ой степени, который проходит через все заданные точки из набора значений. Интерполяционный полином для заданного набора данных является единственным.

В общем виде интерполяционный многочлен Ньютона записывается в следующем виде:

(2.1),

где *n* - степень полинома, - разделенная разность *k*-го порядка, вычисляемая как , или как (2.2).

**Алгоритм построения интерполяционного параболического полинома.**

1. **Полином Лагранжа.**

Шаг 1. Задаем узлы интерполяции (x, y), где x - значения аргументов, y - значения функции в этих точках, шаг h;

Шаг 2. Определяем степень искомого полинома, используя табличные разности;

Шаг 3. Выделяем необходимое число характерных точек, взяв середины отрезков на каждом шаге h;

Шаг 4. Вычисляем базисный полином Лагранжа для каждого узла по формуле (1.2);

Шаг 5. Строим интерполяционный полином Лагранжа по формуле (1.1), где lni(x) - найденный базисный полином Лагранжа;

Шаг 6. Вычисляем значения интерполяционного полинома для нужных точек;

Шаг 7. Вычисляем погрешность как сумму модулей разностей между значениями точек, не вошедших в построение полинома и значениями построенного полинома в этих точках, Конец.

1. **Полином Ньютона.**

Шаг 1. Задаем узлы интерполяции (x, y), где x - значения аргументов, y - значения функции в этих точках, шаг h;

Шаг 2. Определяем степень искомого полинома, используя табличные разности;

Шаг 3. Выделяем необходимое число характерных точек, взяв середины отрезков на каждом шаге h;

Шаг 4. Вычисляем разделенные разности для заданных узлов по формуле (2.2);

Шаг 5. Строим интерполяционный полином Ньютона по формуле (2.1), где f(x0, …, xk) - найденная разделенная разность;

Шаг 6. Вычисляем значения интерполяционного полинома для нужных точек, Конец.

Шаг 7. Вычисляем погрешность как сумму модулей разностей между значениями точек, не вошедших в построение полинома и значениями построенного полинома в этих точках, Конец.

**Вывод результата решения задачи.**

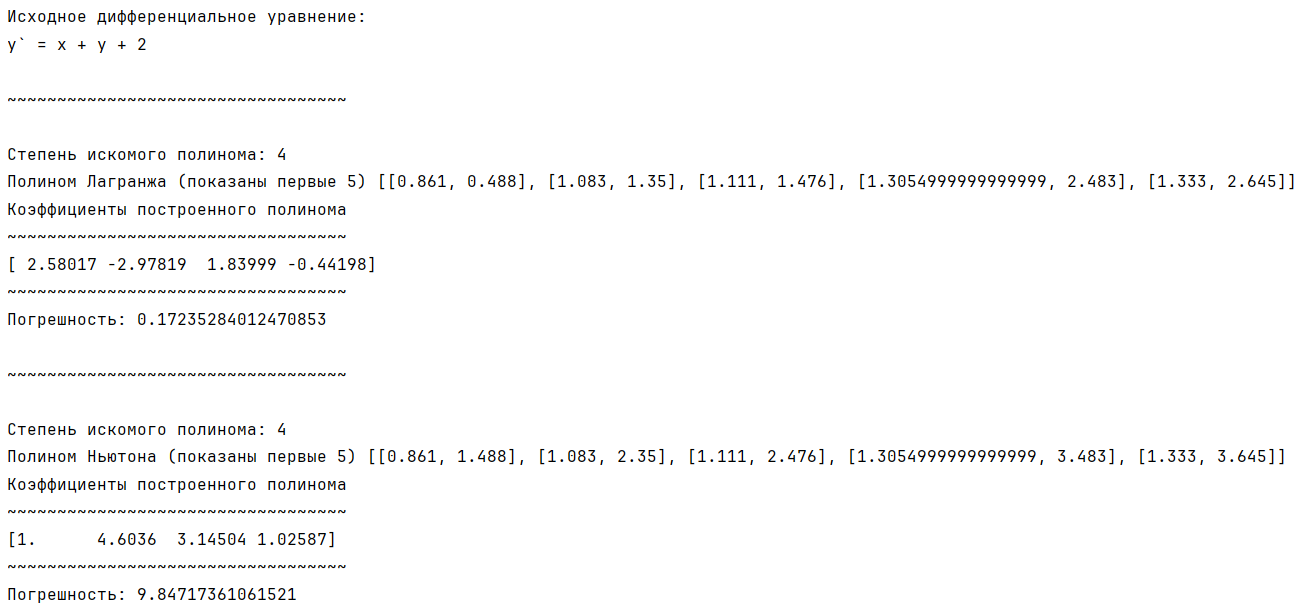
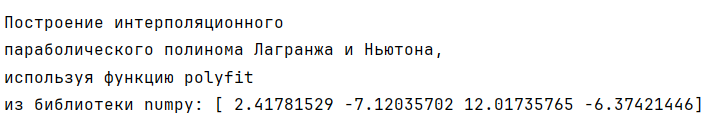
****

Рис.2. Вывод результата задачи

**Проверка правильности решения.**

Правильность решений была проверена с помощью функции polyfit из библиотеки numpy, где параметр deg - степень искомого полинома.



**Выводы.**

Разработали алгоритм и программу, которые по 10 точкам, полученным в задании для решения ОДУ строят интерполяционный параболический полином.

**Приложение: код программы.**

| **import numpy as np**  **from scipy.integrate import odeint**  **def dydx(x, y):**  ***""" Исходное дифференциальное уравнение """***  **return x + y + 2**  **def ODE\_solution(x\_segm, n):**  ***'''***  ***Решение дифференциального уравнения,***  ***используя функцию odeint из библиотеки scipy***  ***'''***  **y0 = 1**  **x0 = 1**  **x\_linspace = np.linspace(x\_segm[0], x\_segm[1], n)**  **correct\_result = odeint(dydx, y0, x\_linspace)**  **result = np.round([[x, \*y] for x, y in zip(x\_linspace, correct\_result)], 3)**  **return result**  **class \_\_interpolation\_parabolic\_polynomial:**  ***'''***  ***Построение интерполяционного***  ***параболического полинома***  ***'''***  **def \_\_init\_\_(self, x\_mas, y\_mas, h = 0.5,**  **accuracy = 1e-1):**  **self.h = h**  **self.accuracy = accuracy**  **self.x\_mas = x\_mas**  **self.y\_mas = y\_mas**  **self.poly\_method = None**  **self.indices = self.\_\_extraction\_notable\_points(x\_mas)**  **def inaccuracy(self):**  ***''' вычисление погрешности '''***  **n\_count = 10**  **other\_indices = [1, 2]**  **correct\_values = ODE\_solution(other\_indices, n\_count)**  **intervals = np.linspace(other\_indices[0], other\_indices[1], n\_count)**  **inter\_values = [self.poly\_method(other\_ind) for other\_ind in intervals]**  **errors = np.abs([inter\_values[i] - correct\_values[i][1] for i in range(n\_count)]).sum()**  **return errors**  **def \_\_extraction\_notable\_points(self, x\_mas):**  ***# выделение характерных точек***  **index\_center = [(x\_mas[i - 1] + x\_mas[i]) / 2**  **for i in range(1, len(x\_mas))]**  **indices = [ind\_c - self.h / 2 for ind\_c in index\_center]**  **indices.extend(ind\_c for ind\_c in index\_center)**  **indices.extend(ind\_c + self.h / 2 for ind\_c in index\_center)**  **indices.sort()**  **return indices**  **def degree\_desired\_polynomial(self):**  ***''' определение степени искомого полинома '''***  **degree = len(y\_mas) - 1**  **y\_mas\_copy = y\_mas.copy()**  **while len(y\_mas\_copy) > self.accuracy:**  **new\_mas = []**  **for i in range(1, len(y\_mas\_copy)):**  **new\_mas.append(y\_mas\_copy[i] - y\_mas\_copy[i - 1])**  **degree -= 1**  **y\_mas\_copy = new\_mas[:-1]**  **return degree**  **def display\_coefficients\_polynomials(self, coef):**  ***''' вывод коэффициентов построенных полиномов '''***  **print("Коэффициенты построенного полинома\n"**  **"~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")**  **print(coef)**  **print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~")**  **class method\_Lagrange(\_\_interpolation\_parabolic\_polynomial):**  ***''' построение полинома***  ***по методу Лагранжа '''***  **def \_\_init\_\_(self):**  **super().\_\_init\_\_(x\_mas, y\_mas, h)**  **self.degree = self.degree\_desired\_polynomial()**  **self.poly\_method = self.\_\_polynomial\_Lagrange**  **self.main\_coef\_poly = None**  **def \_\_basic\_polynomial(self, x\_val, k):**  **div = 1**  **res = 1**  **for j in range(self.degree):**  **if j != k:**  **res \*= (x\_val - x\_mas[j])**  **div \*= (x\_mas[k] - x\_mas[j])**  **return res / div**  **def \_\_polynomial\_Lagrange(self, x\_val):**  **res = 0**  **self.bps = [0.] \* self.degree**  **for i in range(self.degree):**  **self.bps[i] = self.\_\_basic\_polynomial(x\_val, i)**  **res += self.bps[i] \* y\_mas[i]**  **if self.main\_coef\_poly is None:**  **self.main\_coef\_poly = np.round(self.bps, 5)**  **return res**  **def solve(self):**  **poly\_lagrange = [[]] \* len(self.indices)**  **for i in range(len(self.indices)):**  **poly\_lagrange[i] = [self.indices[i], np.round(self.\_\_polynomial\_Lagrange(self.indices[i]), 3)]**  **return poly\_lagrange**  **class method\_Newton(\_\_interpolation\_parabolic\_polynomial):**  ***''' построение полинома***  ***по методу Ньютона '''***  **def \_\_init\_\_(self):**  **super().\_\_init\_\_(x\_mas, y\_mas, h)**  **self.degree = self.degree\_desired\_polynomial()**  **self.poly\_method = self.\_\_polynomial\_Newton**  **self.main\_coef\_poly = None**  **def \_\_divided\_difference(self, k):**  **result = 0**  **for i in range(k):**  **mul = 1**  **for j in range(k):**  **if j != i:**  **mul \*= (x\_mas[i] - x\_mas[j])**  **result += y\_mas[i] / mul**  **return result**  **def \_\_polynomial\_Newton(self, x\_val):**  **res = y\_mas[0]**  **self.dds = [0.] \* self.degree**  **for k in range(1, self.degree + 1):**  **self.dds[k - 1] = self.\_\_divided\_difference(k)**  **mul = 1**  **for j in range(k - 1):**  **mul \*= (x\_val - x\_mas[j])**  **res += self.dds[k - 1] \* mul**  **if self.main\_coef\_poly is None:**  **self.main\_coef\_poly = np.round(self.dds, 5)**  **return res**  **def solve(self):**  **poly\_newton = [[]] \* len(self.indices)**  **for i in range(len(self.indices)):**  **poly\_newton[i] = [self.indices[i], np.round(self.\_\_polynomial\_Newton(self.indices[i]), 3)]**  **return poly\_newton**  **if \_\_name\_\_ == "\_\_main\_\_":**  **print(f"Исходное дифференциальное уравнение:\n"**  **f"y` = x + y + 2")**  **h = 0.5**  **x\_segment = [1, 3]**  **set\_points = ODE\_solution(x\_segment, 10)**  **x\_mas = np.array([point[0] for point in set\_points])**  **y\_mas = np.array([point[1] for point in set\_points])**  **print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")**  **ipp\_lagrange = method\_Lagrange()**  **print(f"Степень искомого полинома: {ipp\_lagrange.degree}")**  **poly\_lagrange = ipp\_lagrange.solve()**  **print(f"Полином Лагранжа (показаны первые 5) {poly\_lagrange[:5]}")**  **ipp\_lagrange.display\_coefficients\_polynomials(ipp\_lagrange.main\_coef\_poly)**  **inaccuracy\_poly\_lagrange = ipp\_lagrange.inaccuracy()**  **print(f"Погрешность: {inaccuracy\_poly\_lagrange}")**  **print("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n")**  **ipp\_newton = method\_Newton()**  **print(f"Степень искомого полинома: {ipp\_newton.degree}")**  **poly\_newton = ipp\_newton.solve()**  **print(f"Полином Ньютона (показаны первые 5) {poly\_newton[:5]}")**  **ipp\_newton.display\_coefficients\_polynomials(ipp\_newton.main\_coef\_poly)**  **inaccuracy\_poly\_newton = ipp\_newton.inaccuracy()**  **print(f"Погрешность: {inaccuracy\_poly\_newton}")**  **correct\_result\_poly = np.polyfit(x\_mas, y\_mas, deg=ipp\_newton.degree-1)**  **print(f"\nПостроение интерполяционного\n"**  **f"параболического полинома Лагранжа и Ньютона,\n"**  **f"используя функцию polyfit\n"**  **f"из библиотеки numpy: {correct\_result\_poly}\n")** |
| --- |