Российский университет дружбы народов им. П. Лумумба

Факультет физико-математических и естественных наук

Лабораторная работа №2

Дисциплина: Вычислительные методы

Студент: Шуплецов Александр Андреевич

Группа: НФИбд-01-22

Москва

2024 г.

Оглавление

Задание

Интерполяция

1. Построить равномерное разбиение отрезка $[a,b]$ из задания на $N=10$ частей точками $a=x_0, x_1, \dots, x_N=b$
   * параметр N должен задаваться в одном месте в программе.

2. Рассчитать значения функции $f(x)$ из задания в узлах интерполяции: $y_0=f(x_0), y_1=f(x_1), \dots, y_N=f(x_N)$.

3. Построить интерполяционный полином Ньютона $P_N(x)$ согласно значениям из п.1, 2.
   *полином должен быть оформлен в виде отдельной функции (или отдельного метода класса)

4. Построить равномерное разбиение отрезка $[a,b]$ из задания на $M=3N$ частей точками $a=\overline{x}_0, \overline{x}_1, \dots, \overline{x}_M=b$
   * параметр M должен задаваться в одном месте в программе.

5. Посчитать значения исходной функции $f(x)$ из задания и построенного в п.3 полинома Ньютона $P_N(x)$ в точках $\overline{x}_0, \overline{x}_1, \dots, \overline{x}_M$, полученных в п.4
   * в программе вывести таблицу данных следующего вида:

$$\begin{array}{cccc} \overline{x}_0 & f(\overline{x}_0) & P_N(\overline{x}_0) & \delta(\overline{x}_0) \\ \overline{x}_1 & f(\overline{x}_1) & P_N(\overline{x}_1) & \delta(\overline{x}_1) \\ \vdots & \vdots & \vdots & \vdots \\ \overline{x}_M & f(\overline{x}_M) & P_N(\overline{x}_M) & \delta(\overline{x}_M) \end{array}$$

   где $\delta(\overline{x}_j)=\left|P_N(\overline{x}_j)-f(\overline{x}_j)\right|$ – погрешность интерполяции в точке $\overline{x}_j$.

6. Подобрать такое значение N, при котором $\max\left\{\delta(\overline{x}_j)\right\}\le 10^{-1}$, $j=\overline{0,M}$.

Буду использовать вариант 1:

	$f(x)$	Отрезок
1.	$\lvert x\rvert\sin(x)$	$x\in[0,1]$

Теоретическая справка

Данная лабораторная работа представляет собой дополнение к первой лабораторной работе. В ней нам нужно добавить в таблицу вывод работы с полиномом Ньютона.

Формула полинома Ньютона:

$$P_n(x) = A_0 + A_1(x - x_0) + A_2(x - x_0)(x - x_1) + \dots$$
$$+ A_i(x - x_0)\dots(x - x_{i-1}) + \dots + A_n(x - x_0)\dots(x - x_{n-1})$$

Где x_0, x_1, ..., x_n - узлы интерполяции, A – числовой коэффициент.

Такое представление удобно для вычислителя, поскольку увеличение n на единицу требует только добавления к «старому» многочлену одного дополнительного слагаемого. Такое представление интерполяционного полинома P(x) называют интерполяционным полиномом в форме Ньютона.

Полный программный код, с подробным описанием функции, реализующей полином Ньютона.

```
import numpy as np

a = 0
b = 1
N = 10
M = 3 * N

x_N = np.linspace(a, b, N + 1)
x_M = np.linspace(a, b, M + 1)

# Определяем функцию f(x) вариант 1
def f(x):
    return np.abs(x) * np.sin(x)

# Расчитываем значения функции f(x) в точках x_M
y_M = f(x_M)

# Функция для вычисления интерполяционного полинома Лагранжа
def lagrange_polynomial(x, x_points, y_points):
    def basis_polynomial(i):
        p = [(x - x_points[j]) / (x_points[i] - x_points[j]) for j in
range(len(x_points)) if j != i]
        return np.prod(p, axis=0)

    L = sum(y_points[i] * basis_polynomial(i) for i in range(len(x_points)))
    return L

# Функция для вычисления интерполяционного полинома Ньютона
def newton_polynomial(x, x_points, y_points):
    def divided_diff(x_points, y_points):
        n = len(y_points)
        coef = np.zeros([n, n])
        coef[:,0] = y_points

        for j in range(1,n):
            for i in range(n-j):
                coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x_points[i+j]
- x_points[i])

        return coef[0, :]

    def newton_basis(x, xi, x_points):
        n = x_points.size
        newton_terms = 1
        y_interpolated = xi
        for i in range(1, n):
            newton_terms = newton_terms * (x - x_points[i-1])
            y_interpolated = y_interpolated + xi * newton_terms
        return y_interpolated

    coeffs = divided_diff(x_points, y_points)
    newton_poly_vals = np.zeros_like(x)
    for i in range(len(x)):
        newton_poly_vals[i] = np.polyval(coeffs[::-1], x[i])
```

```
    return newton_poly_vals

L_values = lagrange_polynomial(x_M, x_N, f(x_N))  # Значения полинома
Лагранжа в точках x_M
N_values = newton_polynomial(x_M, x_N, f(x_N))  # Значения полинома Ньютона в
точках x_M

# Вычисляем погрешности
delta_L_values = np.abs(L_values - y_M)
delta_N_values = np.abs(N_values - y_M)
print(np.max(delta_L_values))
print(np.max(delta_N_values))

# Выводим таблицу данных
print(f"{'x_j':^10} | {'f(x_j)':^15} | {'L(x_j)':^15} | {'N(x_j)':^15} |
{'δ_L(x_j)':^15} | {'δ_N(x_j)':^15}")
print('-' * 85)
for i in range(len(x_M)):
    print(f"{x_M[i]:^10.5f} | {y_M[i]:^15.5f} | {L_values[i]:^15.5f} |
{N_values[i]:^15.5f} | {delta_L_values[i]:^15.5f} |
{delta_N_values[i]:^15.5f}")'
```

Функция newton_polynomial

python def newton_polynomial(x, x_points, y_points):

Определяет функцию newton_polynomial, которая принимает аргументы:

x: точки, в которых мы хотим оценить полином.

x_points: узлы интерполяции (x_i).

y_points: значения функции в узлах интерполяции (y_i).

Вложенная функция divided_diff

python def divided_diff(x_points, y_points): n = len(y_points) coef = np.zeros([n, n]) coef[:,0] = y_points

divided_diff вычисляет разделенные разности для заданных узлов интерполяции и значений функции.

Инициализируется двумерный массив coef, где будут храниться коэффициенты разделенных разностей.

В первый столбец coef[:,0] заносятся значения функции (y_i).

python for j in range(1, n): for i in range(n - j): coef[i][j] = (coef[i+1][j-1] - coef[i][j-1]) / (x_points[i+j] - x_points[i])

Двойной цикл вычисляет разделенные разности. Это рекуррентное уравнение даёт возможность вычислять все коэффициенты в таблице.

Возвращает первую строку массива coef, содержащую все необходимые коэффициенты (a_0, a_1, \ldots, a_n).

Вычисление значений полинома Ньютона

python coeffs = divided_diff(x_points, y_points)

Вызывается функция divided_diff, чтобы получить коэффициенты полинома Ньютона.

python newton_poly_vals = np.zeros_like(x) for i in range(len(x)): newton_poly_vals[i] = np.polyval(coeffs[::-1], x[i])

newton_poly_vals инициализируется массивом нулей такого же размера, как x.

В цикле для каждой точки x_i вычисляется значение полинома Ньютона, используя коэффициенты, рассчитанные функцией divided_diff.

python return newton_poly_vals

Возвращается массив значений полинома в точках x.

Численные расчеты

```
6.23945339839338e-13
0.03602031244608911
   x_j     |     f(x_j)      |     L(x_j)      |     N(x_j)      |    δ_L(x_j)     |    δ_N(x_j)
-----------------------------------------------------------------------------------------------
 0.00000   |     0.00000     |     0.00000     |     0.00000     |     0.00000     |     0.00000
 0.03333   |     0.00111     |     0.00111     |     0.00442     |     0.00000     |     0.00331
 0.06667   |     0.00444     |     0.00444     |     0.01102     |     0.00000     |     0.00657
 0.10000   |     0.00998     |     0.00998     |     0.01975     |     0.00000     |     0.00977
 0.13333   |     0.01773     |     0.01773     |     0.03060     |     0.00000     |     0.01287
 0.16667   |     0.02765     |     0.02765     |     0.04351     |     0.00000     |     0.01586
 0.20000   |     0.03973     |     0.03973     |     0.05845     |     0.00000     |     0.01872
 0.23333   |     0.05395     |     0.05395     |     0.07538     |     0.00000     |     0.02142
 0.26667   |     0.07027     |     0.07027     |     0.09423     |     0.00000     |     0.02396
 0.30000   |     0.08866     |     0.08866     |     0.11495     |     0.00000     |     0.02630
 0.33333   |     0.10906     |     0.10906     |     0.13749     |     0.00000     |     0.02842
 0.36667   |     0.13145     |     0.13145     |     0.16178     |     0.00000     |     0.03032
 0.40000   |     0.15577     |     0.15577     |     0.18775     |     0.00000     |     0.03198
 0.43333   |     0.18196     |     0.18196     |     0.21533     |     0.00000     |     0.03337
 0.46667   |     0.20996     |     0.20996     |     0.24445     |     0.00000     |     0.03449
 0.50000   |     0.23971     |     0.23971     |     0.27502     |     0.00000     |     0.03531
 0.53333   |     0.27115     |     0.27115     |     0.30697     |     0.00000     |     0.03582
 0.56667   |     0.30420     |     0.30420     |     0.34022     |     0.00000     |     0.03602
 0.60000   |     0.33879     |     0.33879     |     0.37467     |     0.00000     |     0.03589
 0.63333   |     0.37483     |     0.37483     |     0.41024     |     0.00000     |     0.03541
 0.66667   |     0.41225     |     0.41225     |     0.44683     |     0.00000     |     0.03458
 0.70000   |     0.45095     |     0.45095     |     0.48434     |     0.00000     |     0.03339
 0.73333   |     0.49086     |     0.49086     |     0.52269     |     0.00000     |     0.03183
 0.76667   |     0.53187     |     0.53187     |     0.56177     |     0.00000     |     0.02990
 0.80000   |     0.57388     |     0.57388     |     0.60147     |     0.00000     |     0.02759
 0.83333   |     0.61681     |     0.61681     |     0.64171     |     0.00000     |     0.02489
 0.86667   |     0.66055     |     0.66055     |     0.68236     |     0.00000     |     0.02181
 0.90000   |     0.70499     |     0.70499     |     0.72333     |     0.00000     |     0.01833
 0.93333   |     0.75003     |     0.75003     |     0.76451     |     0.00000     |     0.01447
 0.96667   |     0.79556     |     0.79556     |     0.80579     |     0.00000     |     0.01022
 1.00000   |     0.84147     |     0.84147     |     0.84706     |     0.00000     |     0.00559
```

Как можно заметить значение N = 10, указанное в задание, уже удовлетворяет нашему условию из пункта 6.

Вывод

Я построил интерполяционный полином Ньютона на языке программирования Python.