Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Отчет по лабораторной работе №1**

**Дисциплина: «**Практикум по вычислительной математике».

Выполнил

студент гр. 5130901/20003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.А. Вагнер

(подпись)

Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.Н. Цыган

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург   
2024

**ЗАДАНИЕ**

**Вариант 24:**

Для 1≤ x ≤ 3 с шагом h=0.2 вычислить значения функции f(x) с использованием программы QUANC8, где По полученным точкам построить сплайн-функцию и полином Лагранжа 10-й степени. Сравнить значения сплайн-функции и полинома с точным значением f(x) (вычислить интеграл по QUANC8 с высокой точностью) в точках xk=1.1+0.2k для k=0,1,…9.

**ХОД РАБОТЫ**

Для выполнения работы был использован язык программирования Python версии 3.11 и следующие библиотеки:

* SciPy для вычисления сплайна.
* NumPy для работы с массивами точек.
* Matplotlib для представления данных в виде графиков.
* Ctypes для использования функции QUANC8.

Во время работы были поставлены следующие цели:

1. При помощи программы QUANC8 определить значения функции f(x) в заданных точках
2. По полученным точкам построить полином Лагранжа десятой степени и сплайн-функцию.
3. С помощью программы QUANC8 вычислить значение интеграла в другом наборе заданных точек с высокой точностью.
4. Сравнить результаты интерполирования со значениями полученными с помощью QUANC8.

Сплайн

Сплайн функцию можно построить с помощью функции CubicSpline, из библиотеки SciPy. Эта функция возвращает коэффициенты b, c и d:

# Вычисление коэффициентов сплайн-функции  
spline = CubicSpline(x, y)

Полином Лагранжа

Функцию для построения полинома Лагранжа нетрудно написать самостоятельно:

# сама функция  
def lagrange\_polynomial(x, y):  
 def L(k):  
 return lambda x\_val: np.prod([(x\_val - x[j]) / (x[k] - x[j]) for j in range(len(x)) if j != k])  
  
 return lambda x\_val: np.sum(y[k] \* L(k)(x\_val) for k in range(len(x)))

# применение

poly = lagrange\_polynomial(x, y)

QUANC8

Для использования QUANC8 нужно создать библиотеку на основании готовой библиотеки на языке C из открытого доступа:

# Функция, реализирующая функцию QUANC8  
lib = ctypes.CDLL('./quanc8.dll')  
  
# определение аргументов функции  
lib.quanc8.argtypes = [ctypes.CFUNCTYPE(ctypes.c\_double, ctypes.c\_double),  
 ctypes.c\_double, ctypes.c\_double, ctypes.c\_double,  
 ctypes.c\_double, ctypes.POINTER(ctypes.c\_double),  
 ctypes.POINTER(ctypes.c\_double), ctypes.POINTER(ctypes.c\_int),  
 ctypes.POINTER(ctypes.c\_double), ctypes.POINTER(ctypes.c\_int)]  
lib.quanc8.restype = ctypes.c\_int  
  
  
def quanc8\_wrapper(fun, a, b, abserr, relerr):  
 resultR = ctypes.c\_double()  
 errestR = ctypes.c\_double()  
 nofunR = ctypes.c\_int()  
 posnR = ctypes.c\_double()  
 flag = ctypes.c\_int()  
  
 lib.quanc8(fun, a, b, abserr, relerr, resultR, errestR, nofunR, posnR, flag)  
  
 return resultR.value, errestR.value, nofunR.value, posnR.value, flag.value

**РЕЗУЛЬТАТЫ РАСЧЕТОВ**

|  |  |
| --- | --- |
| **x** | **F(x)** |
| **1.0** | **0.3642230481001669** |
| **1.2** | **0.5509333397637297** |
| **1.4** | **0.7703173048134445** |
| **1.6** | **1.014220772198156** |
| **1.8** | **1.2723156677864331** |
| **2.0** | **1.5326513289449823** |
| **2.2** | **1.7822785020529512** |
| **2.4** | **2.0079231674221076** |
| **2.6** | **2.1966822860135164** |
| **2.8** | **2.3367109104093666** |
| **3.0** | **2.4178693994784086** |

Таблица результатов работы QUANC8 для функции f(x)

По полученным точкам построим полином Лагранжа и сплайн-функцию. Результаты представим при помощи библиотеки MatplotLib в виде графиков.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Как видим, графики сплайна и полинома Лагранжа не имеют заметных отличий на данном наборе точек и с данной точностью построения.

Далее сравним точки, полученные интерполированием, с точками, вычисленными при помощи QUANC8. Результат сравнения представим также в виде графика. Точки, в которых проведено сравнение отмечены красным. Ввиду точности метода Лагранжа погрешность на графике пришлось умножить на 1000, чтобы отклонения были заметны.

Изображение выглядит как текст, линия, График, диаграмма

Автоматически созданное описание

Очевидно, погрешность интерполирования возрастает при отдалении от середины набора опорных точек. При этом полином Лагранжа десятой степени многократно точнее сплайн-функции.

**ВЫВОД**

В ходе выполнения данной работы, был изучен функционал математических библиотек, для языка программирования Python, и применен на практике. В ходе сравнения было определено, что оба метода достаточно точны на данном наборе точек. При этом погрешность сплайн-функции оказалась на несколько порядков больше таковой для полинома Лагранжа десятой степени.

**ПРИЛОЖЕНИЕ**

import numpy as np  
import matplotlib.pyplot as plt  
from scipy.interpolate import CubicSpline  
import ctypes  
from math import sin  
  
# Load the shared library  
lib = ctypes.CDLL('./quanc8.dll')  
  
# Define the argument types and return type of the quanc8 function  
lib.quanc8.argtypes = [ctypes.CFUNCTYPE(ctypes.c\_double, ctypes.c\_double),  
 ctypes.c\_double, ctypes.c\_double, ctypes.c\_double,  
 ctypes.c\_double, ctypes.POINTER(ctypes.c\_double),  
 ctypes.POINTER(ctypes.c\_double), ctypes.POINTER(ctypes.c\_int),  
 ctypes.POINTER(ctypes.c\_double), ctypes.POINTER(ctypes.c\_int)]  
lib.quanc8.restype = ctypes.c\_int  
  
  
def quanc8\_wrapper(fun, a, b, abserr, relerr):  
 resultR = ctypes.c\_double()  
 errestR = ctypes.c\_double()  
 nofunR = ctypes.c\_int()  
 posnR = ctypes.c\_double()  
 flag = ctypes.c\_int()  
  
 lib.quanc8(fun, a, b, abserr, relerr, resultR, errestR, nofunR, posnR, flag)  
  
 return resultR.value, errestR.value, nofunR.value, posnR.value, flag.value  
  
  
def my\_func(t):  
 return abs(t) \*\* 0.5 \* sin(t)  
  
  
def my\_fun(t):  
 result, errest, nofun, posn, flag = quanc8\_wrapper(  
 ctypes.CFUNCTYPE(ctypes.c\_double, ctypes.c\_double)(my\_func), 0.0, t, 1e-6, 1e-6  
 )  
 return result  
  
  
  
  
def lagrange\_polynomial(x, y):  
 def L(k):  
 return lambda x\_val: np.prod([(x\_val - x[j]) / (x[k] - x[j]) for j in range(len(x)) if j != k])  
  
 return lambda x\_val: np.sum(y[k] \* L(k)(x\_val) for k in range(len(x)))  
  
  
  
x = np.array([i / 10 for i in range(10, 30, 2)])  
y = np.array([my\_fun(x) for x in x])  
  
poly = lagrange\_polynomial(x, y)  
spline = CubicSpline(x, y)  
  
  
points = np.array([i / 10 for i in range(11, 30, 2)])  
# Plot data points  
plt.scatter(points, [0.5] \* 10, color='red', label='Data Points')  
  
# Plot given function interpolation  
plt.plot(points, [my\_fun(x) for x in points], label='f(x)')  
  
# Plot Lagrange polynomial interpolation  
plt.plot(points, [poly(x) for x in points], label='Lagrange Interpolation')  
  
# Plot cubic spline interpolation  
plt.plot(points, [spline(x) for x in points], label='Cubic Spline Interpolation')  
  
plt.xlabel('x')  
plt.ylabel('y')  
plt.legend()  
plt.title('Lagrange vs. Cubic Spline Interpolation')  
plt.grid(True)  
plt.figure()  
  
# Plot data points  
plt.scatter(points, [0.0] \* 10, color='red', label='Data Points')  
  
# Plot Lagrange polynomial errors  
plt.plot(points, [1000\*(my\_fun(x) - poly(x)) for x in points], label='1000х Lagrange Errors')  
  
# Plot cubic spline errors  
plt.plot(points, [my\_fun(x) - spline(x) for x in points], label='Cubic Spline Errors')  
  
plt.xlabel('x')  
plt.ylabel('y')  
plt.legend()  
plt.title('Lagrange vs. Cubic Spline Interpolation Errors')  
plt.grid(True)  
plt.show()