<u>МГТУ им. Н.Э. Баумана</u>

ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

Лабораторный практикум №5

по теме: «Построение и программная реализация алгоритмов численного интегрирования»

Студент: Нгуен Фыок Санг
Группа: ИУ7И-46
Преподаватель: Градов В.М.

2020

Цель работы: Получение навыков построения алгоритма вычисления двукратного интеграла с использованием квадратурных формул Гаусса и Симпсона.

Задание:

Построить алгоритм и программу для вычисления двукратного интеграла при фикчированном значении параметра τ:

$$\epsilon(\tau) = \frac{\pi}{4}\int_0^{\frac{\pi}{2}} d\phi \int_0^{\frac{\pi}{2}} [1 - \exp(-\tau \frac{1}{R})] \cos\theta sin\theta d\theta d\varphi$$

$$\frac{1}{R} = \frac{2cos\theta}{1 - cos^2\theta sin^2\theta}$$

Применить метод последовательного интегрирования. По одному направлению использовать формулу Гаусса, а по другому – формулу Симпсона.

Описание алгоритма:

Имеем $\int_{-1}^{1} f(t)dt = \sum_{i=1}^{n} A_i f(t_i)$, положим $\int_{-1}^{1} t^k dt = \sum_{i=1}^{n} A_i f(t_i^k), k = 0, 1, 2, \dots, 2n-1$

Имеем систему:

$$\begin{cases} \sum_{i=1}^{n} A_i = 2 \\ \sum_{i=1}^{n} A_i t_i = 0 \\ \sum_{i=1}^{n} A_i t_i^2 = \frac{2}{3} \\ \dots \\ \sum_{i=1}^{n} A_i t_i^{2n-1} = 0 \end{cases}$$

Системая нелинейная, найти решение сложно. Для нахождения A_i и t_i можно воспользоваться полиномом Лежандра. Формула полинома:

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}[(x^2 - 1)^n], n = 0, 1, 2$$

Узлами формулы Гаусса являются нули полинома Лежандра $P_n(t)$, а A_i можно найти из вышеуказанной системы уравнений

При вычислении интеграла на произвольном интервале $[a, b]$, для применения квадратурной формулы Гаусса необходимо выполнить преобразование переменной:

$$x = \frac{a+b}{2} + \frac{b-a}{2}t$$

В таком случае, получаем конечную формулу для произвольного интервала $[a, b]$:

$$\int_a^b f(x)dx = \frac{b-a}{2}\sum_{i=1}^{n} A_i f(x_i)$$

Так же, существует квадратнурная формула Симпсона:

$$\int_a^b f(x)dx \approx \frac{h}{3}\sum_{i=0}^{\frac{N}{2}-1}(f_{2i} + 4f_{2i+1} + f_{2i+2})$$

Однако, эти методы можно применять и для приближенной оценки двукратных (и не только) интегралов. Рассмотрим интеграл по прямоугольной области:

$$I = \int_c^d \int_a^b f(x, y)dxdy = \int_a^b F(x)dx \text{, где } F(x) = \int_c^d f(x, y)dy$$

По каждой координате введем сетку узлов. Каждый однократный интеграл вычисляют по квадратурным формулам. Для разных направлений можно использовать квадратурные формулы разных порядков точности, в т.ч. и Гаусса.

Конечная формула:

$$I = \int_c^d \int_a^b f(x,y)dxdy = \sum_{i=1}^{n}\sum_{j=1}^{m} A_i B_{ij} f(x_i, y_i)$$

где A_i B_{ij} – известные постоянные.

Результаты:

1. Описать алгоритм вычисления n корней полинома Лежандра n-ой степени $P_n(x)$ при реализации формулы Гаусса.

   Для вычисления корней полинома Лежандра :

   Вычисляются итеративно по методу Ньютона

$$x_i^{(k+1)} = x_i^{(k)} - \frac{P_n(x_i^{(k)})}{P'_n(x_i^{(k)})}$$

   причем начальное приближение для i-го корня (i = 1, 2, …, n) берется по формуле:

$$x_i^0 = cos(\frac{\pi(4i-1)}{(4n+2)})$$

   Значение полинома можно вычислять используя рекуррентную формулу для конкретного значения x.

$$P_n(x) = \frac{1}{n}[(2n-1)xP_{n-1}(x) - (n-1)P_{n-2}(x)]$$

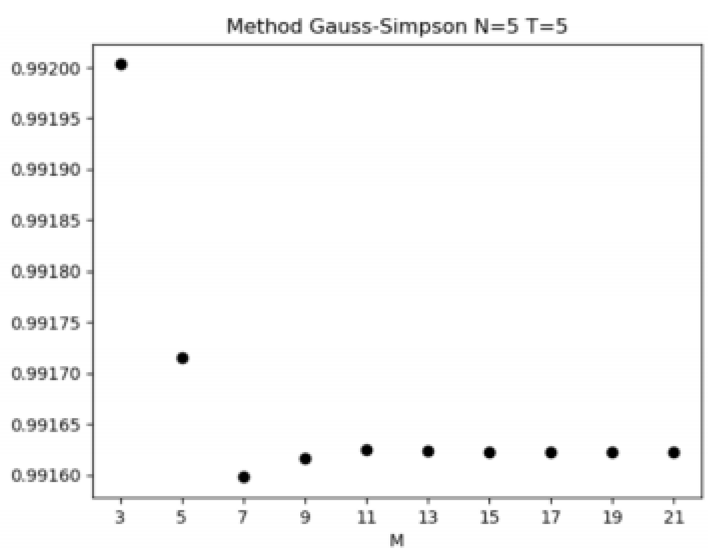
   Причем:

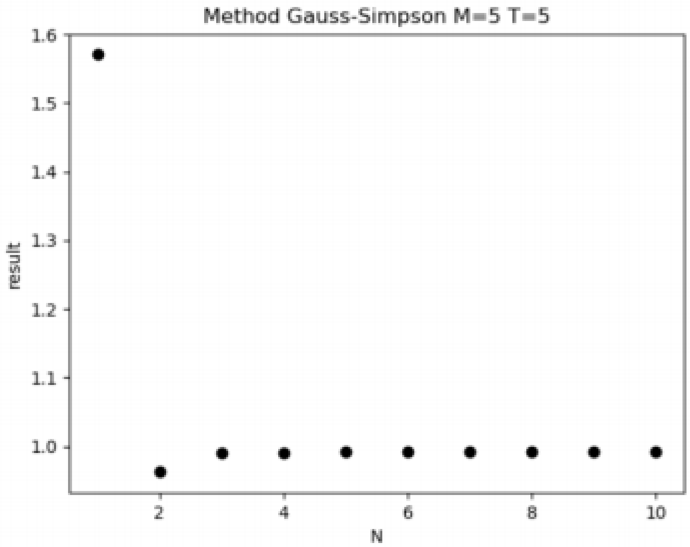
$$P_0(x) = 1$$
$$P_1(x) = x$$

   Производную также можно вычислять для конкретного значения x, используя формулу для производной:

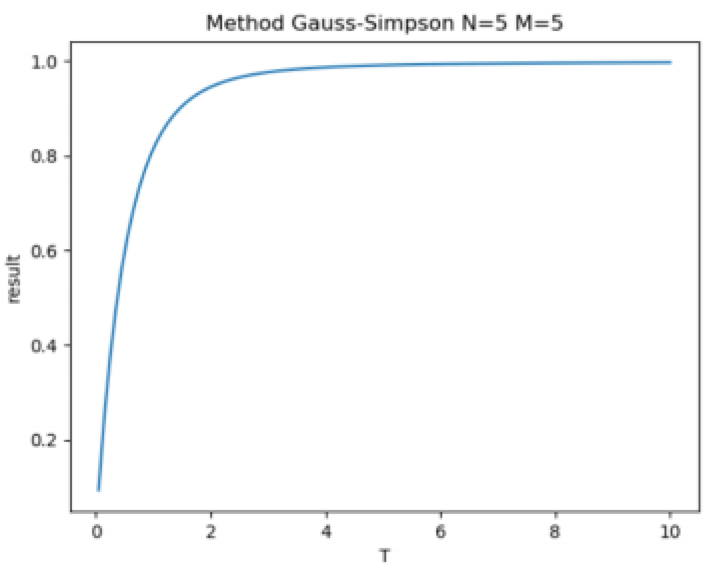
$$P'_n(x) = \frac{n}{1-x^2}[P_{n-1}(x) - xP_n(x)]$$

2. Исследовать влияние количества выбираемых узлов сетки по каждому направлению на точность расчетов.





3. Построить график зависимости ε (τ) в диапазоне изменения τ =0.05-10. Указать при каком количестве узлов получены результаты.



Ответы контрольные вопросы:

1. ***В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается?***
   Если подынтегральная функция не имеет соответствующих производных. Например, если на отрезке интегрирования не существует 3-я и 4-я производные, то порядок тончости формула Симпсона будет только 2-ой.

2. ***Построить формулу Гаусса численного интегрирования при одном узле.***
   Имеем формулу Гаусса:

$$\boldsymbol{\int_{-1}^{1} f(t)dt = \sum_{i=1}^{n} A_i f(t_i)}$$

   При одном узле:

$$n = 1$$
$$A_1 = 2$$
$$P_1(x) = x \ \rightarrow t_1 = 0$$

   Получим:

$$\int_{-1}^{1} f(t)dt = 2f(0)$$

3. ***Построить формулу Гаусса численного интегрирования при двух узлах. при двух узлах:***

$$n = 1$$

$$P_2(x) = \frac{1}{2}(3x^2 - 1) \to t_1 = \frac{1}{\sqrt{3}}; t_2 = -\frac{1}{\sqrt{3}}$$

$$\begin{cases} A_1 + A_2 = 2 \\ A_1 t_1 + A_2 t_2 = 0 \end{cases} \to A_1 = A_2 = 1$$

$$\int_{-1}^{1} f(t)dt = f\left(\frac{1}{\sqrt{3}}\right) + f\left(-\frac{1}{\sqrt{3}}\right)$$

4. Получить обобщенную кубатурную формулу, на основе методе трапеций, с тремя узлами на каждом направлении:

$$\int_c^d \int_a^b f(x,y)dxdy = h_x\left(\frac{1}{2}(F_0 + F_2) + F_1\right)$$

$$= h_x h_y [\frac{1}{4}\left(f(x_0,y_0) + f(x_0,y_2) + f(x_2,y_0) + f(x_2,y_2)\right) + \frac{1}{2}(f(x_0,y_1)$$
$$+ f(x_2,y_1) + f(x_1,y_0) + f(x_1,y_2) + f(x_1,y_1)]$$

Код программы:

```
from math import *
import numpy as np
import matplotlib.pyplot as plt


def legendre(n, x):
    if n == 0 :
        return 1
    if n == 1 :
        return x
    return 1/n * ((2*n-1) * x * legendre(n-1,x) - (n-1) * legendre(n-2,x))


def derivativeLegendre(n, x):
    return n/(1-x**2) * (legendre(n-1, x) - x*legendre(n, x))
def newton(n, x0):
    eps = 1e-7
    max_iter = 100
    x = x0
    for iter in range(0,max_iter):
        f = legendre(n, x)
        if abs(f) < eps:
            return x
        df = derivativeLegendre(n, x)
        #Zero derivative. No solution found.
        if df == 0:
            return None
        x = x - f/df
    #Exceeded maximum iterations. No solution found.
    return None
```

```
def findRootsLegendre(n):
    if n == 0:
        return None
    roots = []
    for i in range(1, n+1):
        approximateRoot = cos(pi * (4*i - 1) / (4*n + 2))
        roots.append(newton(n, approximateRoot))
    return roots
#Using Gaussian elimination method
#matrix n * (n + 1)
#system of linear equations
def solveSOLE(matrix):
    n = len(matrix)
    res = [0 for i in range(n)]

    #row swapping to shape a "loose" row echelon form
    #the pivot of a non zero row is to the right OR UNDER the pivot of the row above it
    for i in range(n-1):
        for j in range(i+1, n):
            if abs(matrix[i][i]) < abs(matrix[j][i]):
                tmp = matrix[i]
                matrix[i] = matrix[j]
                matrix[j] = tmp
    for i in range(n-1):
        for j in range(i+1, n):
            if matrix[i][i] != 0:
                scalar = matrix[j][i] / matrix[i][i]
                for k in range(n+1):
                    matrix[j][k] -= scalar * matrix[i][k]

    #Backward substitutions
    for i in range(n-1, -1, -1):
        res[i] = matrix[i][n]
        for j in range(i+1, n):
            res[i] -= matrix[i][j] * res[j]
        res[i] /= matrix[i][i]

    return res
```

```
def initCoeffMatrix(n):
    roots = findRootsLegendre(n)
    matrix = [[0 for j in range(n+1)] for i in range(n)] # matrix[n]*[n+2]
    for i in range(n):
        t = list(map(lambda x : x**i, roots))
        for j in range(n):
            matrix[i][j] = t[j]
        if i % 2 == 0 :
            matrix[i][n] = 2 / (i+1)
    return matrix

def getWeights(n):
    matrix = initCoeffMatrix(n);
    return solveSOLE(matrix);


def func(x, y, t):
    exponent = -t * 2 * cos(x) / (1 - sin(x)**2 * cos(y)**2)
    return 4/pi * (1 - e**exponent) * cos(x) * sin(x)
def integrate(method, a, b, c, d, n, m, t):
    res = 0
    #gauss
    if method[0] == "1" :
        weight = getWeights(n)
        root = findRootsLegendre(n)
        for i in range(n):
            tmp = (a + b)/2 + (b - a)/2 *root[i]
            res = res + (b - a)/2 * weight[i] * integrateY(method, c, d, m, tmp, t)
    elif method[0] == "2":
        h = (b - a) / (n - 1)
        for i in range(0, int((n - 1) / 2)):
            x1 = a + 2*i * h
            x2 = a + (2*i + 1)*h
            x3 = a + (2*i + 2)*h
            res = res + h/3 * (integrateY(method, c, d, m, x1, t) + 4*integrateY(method, c, d, m, x2, t) +
integrateY(method, c, d, m, x3, t))
    return res;
```

```
def integrateY(method, c, d, m, x, t):
    res = 0
    #gauss
    if method[1] == "1" :
        weight = getWeights(m)
        root = findRootsLegendre(m)
        for i in range(m):
            tmp = (c + d)/2 + (d - c)/2 *root[i]
            res = res + (d - c)/2 * weight[i] * func(x, tmp, t)
    #simpson
    #m is supposed to be odd
    elif method[1] == "2":
        h = (d - c) / (m - 1)
        for i in range(0, int((m-1) / 2)):
            y1 = c + 2*i * h
            y2 = c + (2*i + 1)*h
            y3 = c + (2*i + 2)*h
            res = res + h/3 * (func(x, y1, t) + 4*func(x, y2, t) + func(x, y3, t))
    return res;

def graphN():
    plt.ylabel('result');
    plt.xlabel('N');
    plt.title('Method Gauss-Simpson M=5 T=5')
    x = np.linspace(1, 10, 10)
    y = [integrate("12", 0, pi/2, 0, pi/2, int(val), 5, 5) for val in x]
    print(y)
    plt.plot(x, y, "ko")
    plt.show()

def graphM():
    plt.ylabel('result');
    plt.xlabel('M');
    plt.title('Method Gauss-Simpson N=5 T=5')
    x = np.linspace(3, 21, 10)
    y = [integrate("12", 0, pi/2, 0, pi/2, 5, int(val), 5) for val in x]
    plt.plot(x, y, "ko")
    plt.xticks(np.arange(3, 23, 2))
    plt.show()
```

```python
def graphT():
    plt.ylabel('result');
    plt.xlabel('T');
    plt.title('Method Gauss-Simpson N=5 M=5')
    x = np.linspace(0.05, 10, 100)
    y = [integrate("12", 0, pi/2, 0, pi/2, 5, 5, val) for val in x]
    plt.plot(x, y)
    plt.show()

def main():
    method = ""
    print("Please chose the method used to integrate with respect to x")
    print("1 - method Gauss")
    print("2 - method Simpson")
    char = (input())
    if (char != "1" and char != "2"):
        print("The option is not existed")
        return
    method += char
    print("Please chose the method used to integrate with respect to y")
    print("1 - method Gauss")
    print("2 - method Simpson")
    char = (input())
    if (char != "1" and char != "2"):
        print("The option is not existed")
        return
    method += char
    n = int(input("Input the number of nodes corresponding to the integral with respect to x : "))
    m = int(input("Input the number of nodes corresponding to the integral with respect to y : "))
    t = int(input("Input tau : "))
    print(integrate(method, 0, pi/2, 0, pi/2, n, m, t));
```