Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»
ДИСЦИПЛИНА «Вычислительные алгоритмы»

Лабораторная работа № 5

Тема Построение и программная реализация алгоритмов численного интегрирования

Студент Блохин Д.М.

Группа ИУ7-42Б

Оценка (баллы) _____

Преподаватель Градов В.М.

Москва.
2020 г.

1 Цель работы

Получение навыков построения алгоритма вычисления двухкратного интеграла с использованием квадратных формул Гаусса и Симпсона.

2 Задание

Построить алгоритм и программу для вычисления двухкратного интеграла при фиксированном значении параметра τ

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

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

θ, ϕ - углы сферических координат

Применить метод последовательного интегрирования. По одному напрвлению использовать формулу Гаусса, а по другому - формулу Симпсона.

3 Результаты

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

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.

4 Описание алгоритма

4.1 Квадратная формула Гаусса

Пусть интеграл вычисляется на стандартном интервале [1; 1]. Задача состоитвтом,чтобыподобратьточкиt1,t2,...,tn икоэффициентыA1,A2,...,An так, чтобы квадратурная формула

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

Была точной для всех полиномов наивысшей возможной степени. Установлено, что эта наивысшая степень равна N = 2n−1
Согласно вышенаписанной формуле:

$$\int_{-1}^{1} t^k dt = \sum_{i=1}^{n} A_i t_i^k, k = 0, 1, 2, ..., 2n-1$$

Примнимая во внимание нижеописанную формулу

$$\int_{-1}^{1} t^k dt = \frac{1-(-1)^{k+1}}{k+1} = \begin{cases} \frac{2}{k+1}, mod(k,2) = 0 \\ 0, mod(k,2) = 1 \end{cases}$$

Приходим к выводу, что коэффициенты A_i и узлы t_i находятся из системы 2n уравнений.

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

Система нелинейна и найти решения трудно. Для облегчения задачи воспользуемся полиномом Лежандра степени n и найдём его нули. Нули полинома Лежандра буду являться узлами формулы Гаусса. Проще всего найти нули полинома Лежандра учитывая рекуррентное соотношение

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

Далее система решается методом Гаусса, откуда нахоядятся коэффициенты A_i после чего подставляются в следующую формулу:
$\int_{-1}^{1} f(t)dt = \sum_{i=1}^{n} A_i f(t_i)$
Привычислениинтегралванапроизвольноминтервале[a;b],т.е

$$\int_a^b f(x)dx$$

Для применения квадратурной формулы Гаусса необходимо выполнить преобразование переменной x

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

Таким образом получим следущее уравнение

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

где

$$x_i = \frac{b+a}{2} + \frac{b-a}{2}t_i, i = 1, 2, ..., n$$

4.2 Формула Симпсона

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

5 Полученные результаты

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

Для нахождения корней полниома Лежандра я применил метод половинного деления, опираясь на реккурентное свойство полинома Лежандра:

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

Базой рекурсии в таком случае является:

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

Очевидно, что для применения половинного деления необходимо сначала найти такой отрезок, чтобы на его концах знаки функции были различны (либо чтобы произведение обращалось в нуль), т.е. P_n(x)∗ P_n (x + step) <= 0
Далее с этими исходными данными применяется вышеописанный метод половинного деления.

5.2 Исследовать влияние количества выбираемых узлов сетки по каждому направлению на точность расчётов

Исследуем значения є (τ) при τ = 7 Начальное количество узлов N = 2, M = 2
Получим следующий результат є (7) = 1.04718 что, опираясь на физическое содержание задачи, не является точным результатом (є < 1).

Теперь увеличим количество узлов, а именно сделаем N = 4, M = 4
Получим следующий результат є (7) = 1.00185. Уже гораздо лучше, но є всё ещё больше 1!

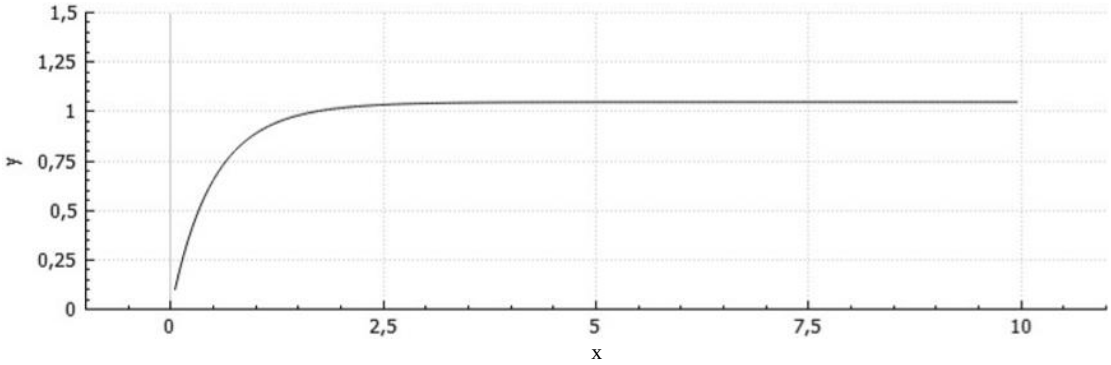
Ещё раз увеличим количество узлов, а именно сделаем N = 16, M = 16
Получим следующий результат є (7) = 0.99657.

Что гораздо ближе к правде, чем предыдущие результаты.

Таким образом можно сделать вывод, что количество узлов напрямую влияет на точность результатов.

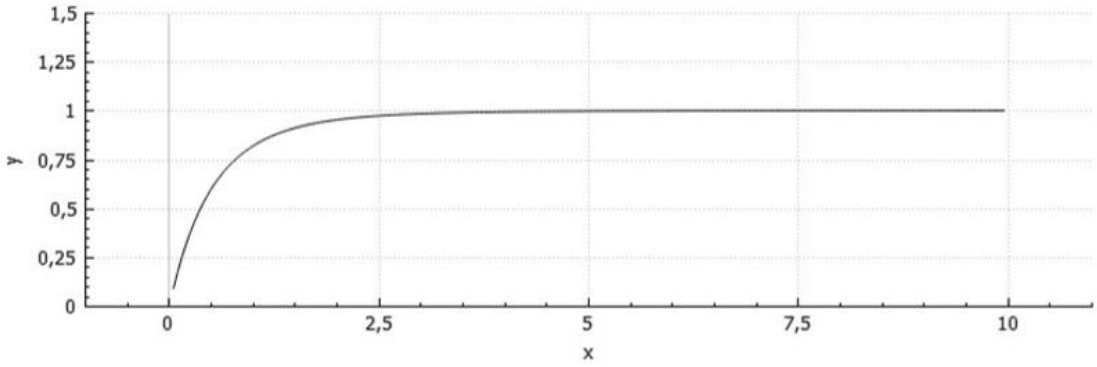
5.3 Построить график зависимости є (τ) в диапазоне изменения τ = 0.05 – 10. Указать при каком количестве узлов получены результаты

N = 2, M = 2



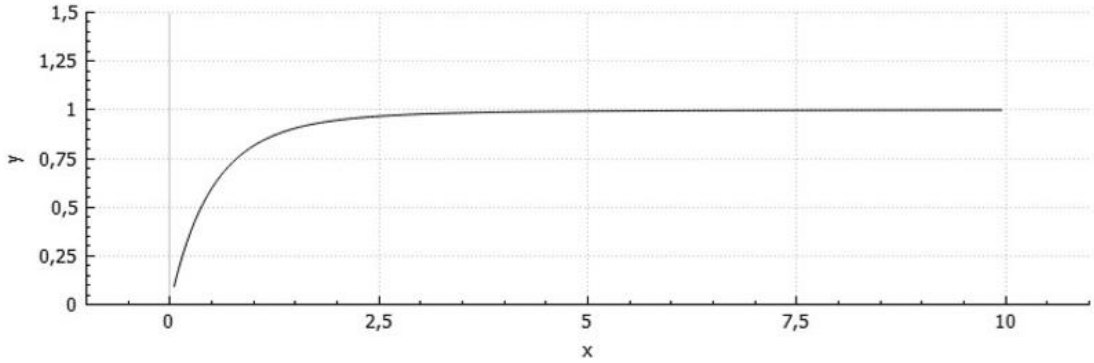
Видно,что присутствуют участки на графике где значения у больше 1

N = 4, M = 4



Лучше, но в некоторых местах у все ешё больше 1.
Это мало заметно, но в этом можно убедиться в выведенных “логах”
1.00223
1.00223
1.00223
1.00224
1.00224
1.00224
1.00224
1.00224

N = 16, M = 16



Приемлимая точность
“Логи”
0.998432
0.998451
0.998474
0.998515
0.998536
0.998557

6 Код программы

6.1 Основной модуль – Формула Гаусса

```
QVector<double> x;
QVector<double> y;
for (double tau = 0.05; tau <= 10; tau += 0.05)
{
    x.push_back(tau);

    N = ui->inputN->text().toInt();
    M = ui->inputM->text().toInt();

    h_y = (end - start) / M;

    int n = N;
    std::vector<std::vector<double>> P(n + 1);

    std::vector<double> t;
    for (double x = start_poly; x < end_poly - step; x += step)
    {
        if (P_i(x, n) * P_i(x + step, n) < 0)
            t.push_back(half_search(x, x + step, n));
    }

    std::vector<std::vector<double>> matrix(2 * n - 1);
    for (int i = 0; i < 2 * n - 1; i++)
    {
        matrix[i] = std::vector <double> (n + 1);
        for (int j = 0 ; j < n; j++)
            matrix[i][j] = pow(t[j], i);

        matrix[i][n] = (1 - pow(-1, i + 1)) / (i + 1);
    }

    for (int iter = 0; iter < n; iter++)
    {
        matrix_max_first(matrix, n, iter);
        matrix_normalize_rows(matrix, n, iter);
    }

    std::vector <double> A(n);
    matrix_get_solutions(matrix, n, A);

    double res = 0;
    for (int i = 0; i < N; i++)
        res += A[i] * F((end + start) / 2 + (end - start) / 2 * t[i], tau, M, h_y);

    res *= (end - start) / 2;
    res *= 4 / M_PI;
    qDebug() << res;
    y.push_back(res);
}
```

6.2 Вспомогательный модуль для нахождения корней полинома Лежандра методом половинного деления

```
double P_i(double x, int i)
{
    if (i == 0)
        return 1;
    else if (i == 1)
        return x;

    return (double)1/i * ((2 * i - 1) * x * P_i(x, i - 1) - (i - 1) * P_i(x, i - 2));
}

double EPS = 1E-7;

double half_search(double start, double end, int n)
{
    double half = (end + start) / 2;
    while (fabs(P_i(half, n) > EPS))
    {
        if (P_i(half, n) * P_i(end, n) < 0)
        {
            start = half;
            half = (end + start) / 2;
        }
        else
        {
            end  = half;
            half = (end + start) / 2;
        }
    }

    return half;
}
```

6.3 Вспомогательный модуль для решения СЛАУ методом Гаусса

```
void matrix_max_first(std::vector <std::vector <double>> &matrix, int x_vars, int iter)
{
    int mx = iter;
    for (int i = iter; i < x_vars; i++)
    {
        if (matrix[i][iter] > matrix[mx][iter])
            mx = i;
    }
    auto tmp = matrix[iter];
    matrix[iter] = matrix[mx];
    matrix[mx] = tmp;
}

void matrix_normalize_rows(std::vector <std::vector <double>> &matrix, int x_vars, int iter)
{
    for (int i = iter; i < x_vars; i++)
    {
        double normalize = matrix[i][iter];
        if (fabs(normalize) < 1E-06)
            continue;
        for (int j = iter; j < x_vars + 1; j++)
            matrix[i][j] /= normalize;
    }

    for (int i = iter + 1; i < x_vars; i++)
    {
        if (matrix[i][iter] < 1E-06)
            continue;
        for (int j = iter; j < x_vars + 1; j++)
            matrix[i][j] -= matrix[iter][j];
    }
}

void matrix_get_solutions(std::vector <std::vector <double>> &matrix, int x_vars, std::vector <double> &x)
{
    for (int i = x_vars - 1; i >= 0; i--)
    {
        double sigma = 0;
        for (int j = x_vars - 1; j > i; j--)
            sigma += matrix[i][j] * x[j];

        if (fabs(matrix[i][i]) <= 1E-06)
        {
            x[i] = 0;
            continue;
        }
        x[i] = (matrix[i][x_vars] - sigma) / matrix[i][i];
    }
}
```

6.4 Вспомогательный модуль для метода Симпсона

```
double f(double phi, double tau, double t) {
    double l_r = (2 * cos(tau)) /
                 (1 - sin(tau) * sin(tau) * cos(phi) * cos(phi));

    return (1 - exp(-t * l_r)) * cos(tau) * sin(tau);
}

double F(double phi, double tau, int M, double h_y)
{
    double res = 0;
    for (int i = 0; i <= M / 2 - 1; i++)
    {
        res += f(phi, start + 2 * i * h_y, tau) +
           4 * f(phi, start + (2 * i + 1) * h_y, tau) +
               f(phi, start + (2 * i + 2) * h_y, tau);
    }

    res *= h_y / 3;
    return res;
}
```

7 Контрольные вопросы

7.1 В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается

Теоретический порядок квадратурных формул численного интегрирования не достигается если подынтегральная функция не имеет соответствующих производных.

7.2 Построить формулу Гаусса численного интегрирования при одном узле

n = 1

$$\begin{cases} \sum_{i=1}^{1} A_i = 2 \\ \sum_{i=1}^{1} A_i t_i = 0 \end{cases}$$

Корень полинома Лежандра первой степени x = 0

$$\begin{cases} A_1 = 2 \\ A_1 * 0 = 0 \end{cases}$$

Решив систему получи A1 = 2

В итоге имеем:

$$\int_a^b f(x)dx = \frac{b-a}{2}(2 * f(\frac{b+a}{2} + \frac{b-a}{2} * 0))$$

7.3 Построить формулу Гаусса численного интегрирования при двух узлах

n = 2

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

Корни полинома Лежандра второй степени степени x = $\pm\sqrt{\frac{1}{3}}$

$$\begin{cases} A_1 + A_2 = 2 \\ A_1 * \sqrt{\frac{1}{3}} + A_2 * -\sqrt{\frac{1}{3}} = 0 \\ A_1 * \sqrt{\frac{1}{3}}^2 + A_2 * (-\sqrt{\frac{1}{3}})^2 = \frac{2}{3} \end{cases}$$

Решив систему получим A1 = A2 = 1

В итоге получим:

$$\int_a^b f(x)dx = \frac{b-a}{2}(1 * f(\frac{b+a}{2} + \frac{b-a}{2}\sqrt{\frac{1}{3}}) + 1 * f(\frac{b+a}{2} + \frac{b-a}{2}(-\sqrt{\frac{1}{3}})))$$

7.4 Получить обощенную кубатурную формулу, аналогичную 6.6 из лекции №6, для вычисления двойного интеграла методом последовательного интегрирования на основе формулы трапеций с тремя узлами по каждому направлению

N = M = 3

$$\int_c^d \int_a^b f(x,y)dxdy == h_x(\tfrac{1}{2}F_0 + F_1 + \tfrac{1}{2}F_2 + F_3)$$

$$F_0 = \int_c^d f(x0,y)dy = h_y(\tfrac{1}{2}f(x0,y0) + f(x0,y1) + \tfrac{1}{2}f(x0,y2) + f(x0,y3))$$

$$F_1 = \int_c^d f(x1,y)dy = h_y(\tfrac{1}{2}f(x1,y0) + f(x1,y1) + \tfrac{1}{2}f(x1,y2) + f(x1,y3))$$

$$F_2 = \int_c^d f(x2,y)dy = h_y(\tfrac{1}{2}f(x2,y0) + f(x2,y1) + \tfrac{1}{2}f(x2,y2) + f(x2,y3))$$

$$F_3 = \int_c^d f(x3,y)dy = h_y(\tfrac{1}{2}f(x3,y0) + f(x3,y1) + \tfrac{1}{2}f(x3,y2) + f(x3,y3))$$

Таким образом получим обобщенную формулу:

$$\int_c^d \int_a^b f(x,y)dxdy == h_x * h_y(\tfrac{1}{4}(f(x_0,y_0) + f(x_0,y_2) + f(x_2,y_0) + f(x_2,y_2)) +$$
$$\tfrac{1}{2}(f(x_0,y_1) + f(x_0,y_3) + f(x_2,y_1) + f(x_2,y_3) + f(x_1,y_0) + f(x_1,y_2) + f(x_3,y_0) +$$
$$f(x_3,y_2)) + (f(x_1,y_1) + f(x_1,y_3) + f(x_3,y_1) + f(x_3,y_3)))$$