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)

Вычислительные алгоритмы.
Лабораторная работа №5.
«Построение и программная реализация алгоритмов численного интегрирования»

Студент **Трошкин Николай Романович**

Группа **ИУ7-46Б**

Студент ____________________ Трошкин Н.Р.
подпись, дата *фамилия, и.о.*

Преподаватель ____________________ Градов В.М.
подпись, дата *фамилия, и.о.*

2021 г.

Цель работы

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

Задание

Исходные данные

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

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

где $\qquad \dfrac{l}{R} = \dfrac{2\cos\theta}{1 - \sin^2\theta \cos^2\varphi},$

Используется метод последовательного интегрирования, по одному направлению используется формула Гаусса, а по другому - формулу Симпсона. На входе количество узлов сетки по каждому из двух направлений и значение параметра τ.

Требуемый результат

Описание алгоритма нахождения n корней полинома Лежандра степени n.

Исследование влияния количества узлов сетки по каждому направлению на точность расчетов.

Построение графика зависимости $\varepsilon(\tau)$в диапазоне τ= 0.05...10 при конкретном количестве узлов.

Краткий алгоритм

Интегрирование по углу θ производится по формуле Гаусса, затем производится интегрирование по φ по формуле Симпсона.

Изначально интервал аргумента φ от 0 до $\frac{\pi}{2}$ разбивается на равные части, которых обязательно четное количество (узлов, соответственно, нечетное). Для каждого узла требуется вычислить интеграл по аргументу θ. Для этого используется формула Гаусса:

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

В этой формуле t_i - узлы, в которых рассматривается функция, интегрируемая на отрезке [-1, 1]. В качестве этих узлов используются корни полинома Лежандра степени n_θ, равной количеству узлов в направлении θ. Полином имеет вид:

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

Этот полином n-й степени, во-первых, всегда имеет n действительных различных корней на отрезке [-1, 1], а во-вторых, его можно вычислять рекуррентно по формуле:

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

При этом $P_0(x) = 1$, $P_1(x) = x$.

Алгоритм определения корней полинома Лежандра степени n:

- Разбиваем отрезок [-1, 1] на 2n равных частей,
- Рассматривая i-й отрезок (от 0 до 2n - 1), проверяем знак выражения $P_n(x_i)P_n(x_{i+1})$. Если оно меньше нуля, внутри отрезка есть корень, увеличиваем счетчик корней.
- Рассмотрев все отрезки, проверяем количество найденных корней: если их n штук, находим их методом половинного

деления, в ином случае увеличиваем количество разбиений отрезка в два раза и просматриваем отрезок заново.

Коэффициенты A_i из формулы Гаусса определяются из системы n_θ уравнений:

$$\sum_{i=1}^{n} A_i = 2,$$

$$\sum_{i=1}^{n} A_i t_i = 0,$$

$$\dots\dots\dots\dots\dots$$

$$\sum_{i=1}^{n} A_i t_i^n = 0, \text{ если } n \text{ нечетно, и } \frac{2}{n+1}, \text{ если } n \text{ четно}$$

СЛАУ решается методом Гаусса.

Так как пределы интегрирования в данной задаче не равны [-1, 1], то требуется сделать преобразование:

$$\theta_i = \frac{b+a}{2} + \frac{b-a}{2} t_i = \frac{\pi}{4}(1 + t_i), \; i = 1, 2, \dots, n$$

И тогда значение интеграла по аргументу θ равно

$$\int_0^{\pi/2} [1 - exp(-\tau \frac{l}{R})] \, cos\theta \, sin\theta \, d\theta = \frac{b-a}{2} \sum_{i=1}^{n} A_i f(\theta_i) = \frac{\pi}{4} \sum_{i=1}^{n} A_i f(\theta_i)$$

Это значение по полученным значениям A_i и θ_i вычисляется для каждого узла направления φ. После этого происходит второе интегрирование по направлению φ по формуле Симпсона. Значение двукратного интеграла будет равно

$$\frac{h_x}{3} \sum_{i=0}^{n_\varphi/2 - 1} (F_{2i} + 4F_{2i+1} + F_{2i+2}),$$ где F_i - значение интеграла, вычисленного по формуле Гаусса для значения $\varphi = \varphi_i, \; i = 0, 1, \dots, n_\varphi$.

После вычисления интеграла нужно не забыть домножить результат на $\frac{4}{\pi}$.

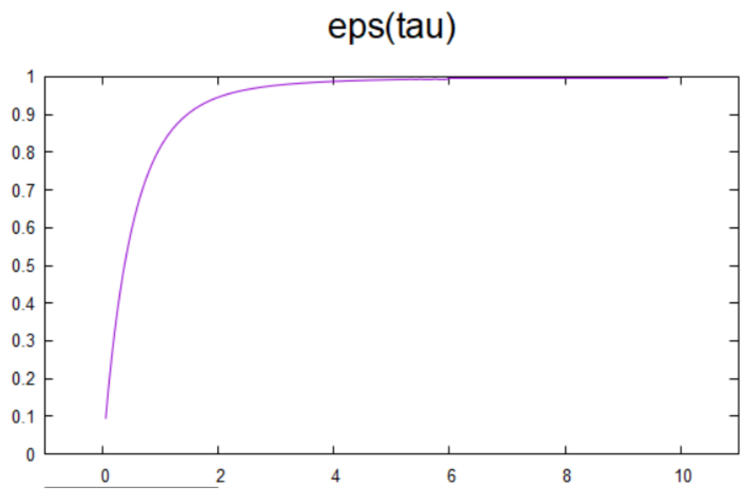
Полученный результат

Алгоритм нахождения n корней полинома Лежандра степени n описан выше. Исследуем влияние количества узлов сетки по каждому из направлений на точность расчетов. Возьмем $\tau = 0.5$. По расчетам программы WolframAlpha, точное значение выражения $\varepsilon(\tau) = 0.595953$.Получим значения выражений для разных n_θ и n_φ.

$n_\theta \rightarrow$ $n_\varphi \downarrow$	3	5	7	9	11
3	0.58518	0.58716	0.58708	0.58708	0.58708
5	0.59278	0.59509	0.59524	0.59523	0.59523
7	0.59427	0.59580	0.59598	0.59602	0.59602
9	0.59422	0.59593	0.59599	0.59603	0.59605
11	0.59407	0.59594	0.59596	0.59599	0.59600

Таким образом, уже при 7 узлах в каждом направлении алгоритм показывает высокую точность (4 знака после точки), это позволяет достаточно быстро вычислять двукратные интегралы.

График зависимости $\varepsilon(\tau)$в диапазоне τ= 0.05...10 при 7 узлах в каждом из направлений интегрирования.



Ответы на вопросы

1. В каких ситуациях теоретический порядок квадратурных формул численного интегрирования не достигается?

Теоретический порядок точности квадратурных формул не достигается в случае, если подынтегральная функция не имеет соответствующих производных, например, если функция не имеет 3-й и 4-й производных, то порядок точности формулы Симпсона будет второй, хотя теоретический порядок равен 4.

2. Построить формулу Гаусса численного интегрирования при одном узле.

Один узел - корень полинома Лежандра первой степени:
$P_1(t) = t;\ t_1 = 0.$

Тогда $A_1 = 2$ и $\int_{-1}^{1} f(t)dt = 2f(0)$

3. Построить формулу Гаусса численного интегрирования при двух узлах.

Два узла - два корня полинома Лежандра второй степени:
$P_2(t) = \frac{1}{2}(3x^2 - 1);\quad t_{1,2} = \pm\frac{\sqrt{3}}{3}$
Тогда имеем систему уравнений:
$A_1 + A_2 = 2,$
$\frac{\sqrt{3}}{3}A_1 - \frac{\sqrt{3}}{3}A_2 = 0.$
Следовательно, $A_1 = A_2 = 1$.Формула Гаусса имеет вид:
$\int_{-1}^{1} f(t)dt = f(\frac{\sqrt{3}}{3}) + f(-\frac{\sqrt{3}}{3})$

4. Получить обобщенную 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.
Имеем по три узла: $x_0, x_1, x_2, y_0, y_1, y_2$.

$$\int_a^b\int_c^d f(x, y)dxdy = \int_a^b dx\int_c^d f(x, y)dy = \int_a^b F(x)dx = h_x[\frac{F_0 + F_2}{2} + F_1]$$

При этом $F_i = \int_c^d f(x_i, y)dy = h_y[\frac{f(x_i, y_0) + f(x_i, y_2)}{2} + f(x_i, y_1)]$

В итоге:

$$\int_a^b\int_c^d f(x, y)dxdy = \frac{h_x h_y}{4}[f(x_0, y_0) + f(x_0, y_2) + f(x_2, y_0) + f(x_2, y_2) +$$
$$+ 2f(x_1, y_0) + 2f(x_1, y_2) + 2f(x_0, y_1) + 2f(x_2, y_1) + 4f(x_1, y_1)]$$

где $h_x = \frac{x_2 - x_0}{2}$, $h_y = \frac{y_2 - y_0}{2}$

Код программы

Основная функция приложения:

```
int main(void)
{
    show_info(); // информация о программе
    double tau = get_argument(); // получение параметра t

    // количество узлов для формулы Симпсона должно быть нечетно
    size_t n_fi = get_nodes("fi");
    // количество узлов для формулы Гаусса
    size_t n_theta = get_nodes("theta");

    // разбиваем направление fi на равные подынтервалы
    double *fi_arr = malloc(sizeof(double) * n_fi);
    if (!fi_arr)
        return EXIT_FAILURE;

    fi_arr[0] = 0;
    double step = M_PI_2 / (n_fi - 1);
    for (size_t i = 1; i < n_fi; i++)
        fi_arr[i] = fi_arr[i - 1] + step;
```

```
    // вычисляем интегралы для каждого fi по формуле Гаусса
    double *fi_integrals = gauss_integral(tau, fi_arr, n_fi, n_theta);

    if (!fi_integrals)
    {
        free(fi_arr);
        return EXIT_FAILURE;
    }

    // вычисление итогового интеграла формулой Симпсона
    double res = 0;
    for (size_t i = 0; i < (n_fi - 1) / 2; i++)
        res += fi_integrals[2 * i] + 4 * fi_integrals[2 * i + 1] \
            + fi_integrals[2 * i + 2];

    res *= step / 3.0 / M_PI_4;

    printf("%.3lf", res);
    free(fi_integrals);
    free(fi_arr);
    return EXIT_SUCCESS;
}
```

Функции, использованные в начинке программы:

```
// вычисление значения полинома Лежандра степени n в точке x
static double legendre(double x, int n)
{
    if (n == 0)
        return 1;
    if (n == 1)
        return x;
    double leg0 = 1;
    double leg1 = x;
    double leg2;
    for (int i = 2; i <= n; i++)
    {
        leg2 = ((2 * i - 1) * x * leg1 - (i - 1) * leg0) / i;
        leg0 = leg1;
        leg1 = leg2;
```

```c
    }
    return leg2;
}
// нахождение корня методом половинного деления
static double bisection(double left, double right, int deg)
{
    double middle = (left + right) / 2;
    do
    {
        if (fabs(legendre(middle, deg)) < EPS)    // EPS = 1e-7
            return middle;
        if (legendre(left, deg) * legendre(middle, deg) < 0)
            right = middle;
        else
            left = middle;
        middle = (left + right) / 2;
    }
    while (right - left > EPS);
    return middle;
}

// нахождение корней полинома Лежандра степени deg
static double *find_roots(size_t deg)
{
    double *roots = malloc(sizeof(double) * deg);    // массив корней
    if (!roots)
        return NULL;

    size_t roots_num;
    double step = 2.0 / deg;
    do
    {
        step /= 2.0;    // ширина подынтервала поиска
        roots_num = 0;
        double a = -1;
        double b = a + step;
        while (a < 1)
        {
            if (legendre(a, deg) * legendre(b, deg) < 0)    // разные знаки - корень есть
                roots_num++;
```

```
            a = b;
            b += step;
        }
    }
    while (roots_num < deg);   // должно найтись deg корней

    double a = -1;
    double b = a + step;
    size_t i = 0;
    while (a < 1 && i < deg)
    {
        if (legendre(a, deg) * legendre(b, deg) < 0)
        {
            roots[i] = bisection(a, b, deg);
            i++;
        }

        a = b;
        b += step;
    }

    return roots;
}

// подынтегральная функция
static double func(double fi, double theta, double tau)
{
    double l_div_R = 2 * cos(theta) / (1 - sin(theta) * sin(theta) * cos(fi) * cos(fi));
    return cos(theta) * sin(theta) * (1 - exp(-tau * l_div_R));
}
// получение коэффициентов СЛАУ из найденных корней полинома Лежандра
static int get_slae_coeffs(double **slae_A, double *slae_Y, size_t n_theta, double *roots)
{
    // здесь будут храниться коэффициенты одной строки матрицы СЛАУ
    // сделано, чтобы не приходилось каждый раз высчитывать степень t_i
    double *coeffs = malloc(n_theta * sizeof(double));
    if (!coeffs)
        return -1;
```

```c
    // заполнение правой части СЛАУ
    for (size_t i = 0; i < n_theta; i++)
        if (i % 2 == 0)
            slae_Y[i] = 2.0 / (i + 1);
        else
            slae_Y[i] = 0;

    // инициализация нулевой строки матрицы СЛАУ
    for (size_t i = 0; i < n_theta; i++)
        coeffs[i] = 1;

    // заполнение матрицы СЛАУ
    for (size_t i = 0; i < n_theta; i++)
        for (size_t j = 0; j < n_theta; j++)
        {
            slae_A[i][j] = coeffs[j];
            coeffs[j] *= roots[j];
        }

    free(coeffs);
    return 0;
}

// вычисление интегралов для массива аргументов fi по формуле Гаусса
double *gauss_integral(double tau, double *fi, size_t n_fi, size_t n_theta)
{
    double *roots = find_roots(n_theta);    // корни полинома Лежандра
    // матрицы СЛАУ для нахождения коэффициентов
    double **slae_A = malloc(n_theta * sizeof(double *) + n_theta * n_theta * sizeof(double));
    double *slae_Y = malloc(n_theta * sizeof(double));
    // массив вычисленных значений интеграла при фиксированных fi
    double *integrals = malloc(n_fi * sizeof(double));

    if (!slae_A || !slae_Y || !roots || !integrals)
    {
        free(slae_A);
        free(slae_Y);
        free(roots);
        free(integrals);
        return NULL;
```

```
    }

    // распределение памяти под матрицу СЛАУ
    double *split = (double *)((char *)slae_A + n_theta * sizeof(double *));

    for (size_t i = 0; i < n_theta; i++)
        slae_A[i] = split + i * n_theta;

    int rc = get_slae_coeffs(slae_A, slae_Y, n_theta, roots);

    if (rc)
    {
        free(integrals);
        integrals = NULL;
    }
    else
    {
        // решение СЛАУ
        solve_slae(slae_A, slae_Y, n_theta);

        // преобразование к новому отрезку [0, pi/2]
        for (size_t i = 0; i < n_theta; i++)
            roots[i] = M_PI_4 * (1 + roots[i]);

        // вычисление интегралов для каждого аргумента fi
        for (size_t i = 0; i < n_fi; i++)
        {
            integrals[i] = 0;
            for (size_t j = 0; j < n_theta; j++)
                integrals[i] += slae_Y[j] * func(fi[i], roots[j], tau);
            integrals[i] *= M_PI_4;
        }
    }

    free(slae_A);
    free(roots);
    free(slae_Y);
    return integrals;
}
```