Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Самарский национальный исследовательский университет
имени академика С.П.Королёва»

Факультет информатики

Кафедра технической кибернетики

Отчет по курсовой работе

Дисциплина «Уравнения математической физики»

Тема: «**АНАЛИТИЧЕСКОЕ РЕШЕНИЕ КРАЕВЫХ ЗАДАЧ МАТЕМАТИЧЕСКОЙ ФИЗИКИ**»

Вариант № 40

Выполнил студент: Белоусов А. А.

Группа: 6409

Проверил: Дегтярев А. А.

Самара 2018

ЗАДАНИЕ К КУРСОВОЙ РАБОТЕ

1. Осуществить математическую постановку краевой задачи для физического процесса, описанного в предложенном варианте курсовой работы.

2. Используя метод разделения переменных (метод Фурье), получить решение краевой задачи в виде разложения в ряд Фурье по собственным функциям оператора Лапласа, соответствующим краевым условиям задачи.

3. Провести анализ погрешности решения.

4. Разработать компьютерную программу расчета функции-решения краевой задачи (суммирования ряда Фурье). При расчете коэффициентов ряда использовать метод численного интегрирования, если это необходимо. Если необходимо, то разработать специальный программный модуль для вычисления используемых собственных чисел оператора Лапласа. Компьютерная программа должна обеспечивать возможность диалогового режима ввода физических, геометрических параметров задачи, числа суммируемых элементов ряда, графическую визуализацию рассчитанного решения задачи.

5. Оформить отчет о выполненной курсовой работе.

***ВАРИАНТ* 40**

Разработать программу расчета на промежутке времени $0 < t \leq T$ малых поперечных колебаний прямоугольной однородной мембраны шириной l_x и длиной l_y. Колебания мембраны возбуждаются начальным отклонением

$$u(x, y, t = 0) = \alpha(x, y), 0 \leq x \leq l_x, 0 \leq y \leq l_y.$$

Края мембраны $x = 0, x = l_x, y = 0$ и $y = l_y$ жестко закреплены, а реакция окружающей среды пренебрежимо мала. Начальные скорости точек мембраны равны нулю.

Поверхностная плотность мембраны и величина натяжения, возникающего в ней в процессе колебаний, равны ρ и η соответственно.

Для решения описанной задачи математической физики применить метод разделения переменных. Для расчетов использовать представление решения задачи в виде ряда Фурье по собственным функциям оператора Лапласа, удовлетворяющим соответствующим краевым условиям.

При проведении расчетов использовать значения параметров l_x, l_y, T, ρ, η, а также выражение функции $\alpha(x, y)$, указанные преподавателем.

Значения параметров, указанные преподавателем:

$$\begin{aligned} l_x &= 4, \\ l_y &= 1, \\ T &= 10, \\ \rho &= 1, \\ \eta &= 1, \\ \alpha(x, y) &= p(x, y) \sin(\frac{\pi y}{l_y}) \\ p(x, y) &= \frac{x^2}{4} + x \end{aligned}$$

РЕФЕРАТ

42 страниц, 12 рисунков, 2 таблиц, 0 источников, 1 приложение

УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ, КРАЕВАЯ ЗАДАЧА, УРАВНЕНИЕ ТЕПЛОПРОВОДНОСТИ, МЕТОД РАЗДЕЛЕНИЯ ПЕРЕМЕННЫХ, СОБСТВЕННЫЕ ФУНКЦИИ ОПЕРАТОРА ЛАПЛАСА, РЯД ФУРЬЕ, ПОЛИНОМ ЛЕЖАНДРА.

Целью курсовой работы является получение решения краевой задачи колебаний прямоугольной мембраны в виде разложения в ряд Фурье по собственным функциям оператора Лапласа и создание компьютерной программы для расчета функции-решения.

Для получения аналитического решения краевой задачи использован метод разделения переменных. Решение задачи получено в виде конечного ряда Фурье.

Разработана компьютерная программа, обеспечивающая расчет и графическую визуализацию процесса колебаний мембраны.

Приведены графические результаты численного решения задачи колебаний мембраны, а также анализ погрешности решения.

Программа написана на языке Python в среде разработки PyCharm на операционной системе Windows.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Метод разделения переменных относится к классу аналитических методов решения краевых задач математической физики. Характеризуя этот метод необходимо выделить его достоинства и недостатки в сравнении с другими методами.

К достоинствам метода разделения переменных следует отнести возможность получения точного решения краевой задачи в виде ряда Фурье. Такая форма решения задачи часто и весьма успешно используется для теоретического исследования свойств этого решения. В случае достаточно быстрой сходимости ряда Фурье она может с успехом использоваться для численного моделирования физического процесса (явления).

К числу недостатков метода следует отнести его невысокую универсальность. Этот метод весьма проблематично использовать для решения нелинейных уравнений математической физики, уравнений с переменными операторными коэффициентами, а также для решения краевых задач в областях со сложными границами.

Суть метода разделения переменных состоит в факторизации по каждой независимой переменной функции, определяющей решение уравнения математической физики. Далее осуществляется переход к так называемой задаче Штурма-Лиувилля, решение которой приводит к получению собственных функций и соответствующих им собственных чисел оператора Лапласа. Затем решение исходной задачи ищется в виде ряда Фурье по этим собственным функциям.

В настоящей работе метод разделения переменных применен для получения аналитической формы решения задачи описания процесса колебаний мембраны. На основе этого результата разработан алгоритм и компьютерная программа численного моделирования процесса колебаний мембраны.

1 Постановка краевой задачи

Построим математическую модель поперечных колебаний тонкой однородной мембраны. Уравнение свободных поперечных колебаний мембраны имеет вид:

$$\frac{\partial u}{\partial t} = \alpha^2(\frac{\partial u}{\partial x} + \frac{\partial u}{\partial y}). \tag{1}$$

В условии задачи указано, что края мембраны жестко закреплены. Отсюда следуют граничные условия:

$$u|_{x=0}, u|_{y=0}, u|_{x=l_x}, u|_{y=l_y} = 0. \tag{2}$$

По условию, в начальный момент времени отклонение мембраны задано функцией $\alpha(x, y)$, а начальная скорость точек мембраны равна нулю. Отсюда получаем начальные условия:

$$u|_{t=0} = p(x, y)\sin(\frac{\pi y}{l_y});$$
$$\frac{\partial u}{\partial t}|_{t=0} = 0.$$

Производим следующую замену:

$$u(x, y, t) = \nu(x, t)\sin(\frac{\pi y}{l_y}). \tag{3}$$

Подставляем в исходное уравнение, переходим к виду:

$$\frac{\partial \nu}{\partial t} = \alpha^2(\frac{\partial \nu}{\partial x} - \frac{\pi^2}{l_y}\nu(x, t)). \tag{4}$$

Пересчитываем начальные условия для $\nu(x, y)$:

$$\nu|_{t=0} = p(x, y);$$
$$\frac{\partial \nu}{\partial t}|_{t=0} = 0.$$

Таким образом, имея уравнение, набор граничных и начальных условий, получим математическую модель, описывающую нашу задачу:

$$\frac{\partial u}{\partial t} = \nu(x, t)\sin(\frac{\pi y}{l_y}); \tag{5}$$

$$
\begin{cases}
\dfrac{\partial \nu}{\partial t} = \alpha^2 (\dfrac{\partial \nu}{\partial x} - \dfrac{\pi^2}{l_y} \nu(x,t)), 0 \leq x \leq {}_x; \\
\nu|_{x=0}, \nu|_{x=l_x} = 0; \\
\nu|_{t=0} = p(x,y); \\
\dfrac{\partial \nu}{\partial t}|_{t=0} = 0;
\end{cases}
\tag{6}
$$

2 Аналитическое решение краевой задачи и получение решения задачи в виде ряда Фурье

Для решения поставленной задачи в уравнении (6) воспользуемся методом разделения переменных:

$$\nu(x,t) = X(x)T(t). \tag{7}$$

Тогда уравнение (6) примет вид:

$$T''X = \alpha^2 \left(X''T - \frac{\pi^2}{l_y^2} XT \right);$$

Разделим обе части на a^2XT и получим следующее уравнение:

$$\frac{T'}{\alpha^2 T} = \frac{X'' - \frac{\pi^2}{l_y^2}X}{X}. \tag{8}$$

В равенстве (9) можно заметить, что левая и правая части зависят только от t и θ соответственно. Тем самым, общая величина выражений - константа, которую мы возьмем равной $-\lambda^2$. Константа берется с отрицательным знаком, чтобы удовлетворять граничным условиям. Таким образом равенство (9) примет вид:

$$\frac{T'}{\alpha^2 T} = \frac{X'' - \frac{\pi^2}{l_y^2}X}{X} = -\lambda^2. \tag{9}$$

Откуда получим два уравнения:

$$T'' + \lambda^2\alpha^2 T = 0;$$

$$X'' + (\lambda^2 - \frac{\pi^2}{l_y^2})X = 0.$$

И обозначим, что в силу начальных условий, $z \in [-1;1]$. После подстановки ω из уравнения (7) будет зависеть от z и t. Выразим производные от X по θ через производные от X по z.

$$\frac{\partial X}{\partial z} = \frac{\partial X}{\partial \theta}\frac{\partial \theta}{\partial z};$$
$$\frac{\partial X}{\partial \theta} = -\sin(\theta)\frac{\partial X}{\partial z};$$
$$\frac{\partial^2 X}{\partial \theta^2} = (1-z^2)\frac{\partial^2 X}{\partial z^2} - z\frac{\partial X}{\partial z}.$$

Тогда уравнение (**??**) примет вид:

$$(1-z^2)\frac{\partial^2 X}{\partial x^2} - 2z\frac{\partial X}{\partial z} + X(\frac{-\alpha}{kl} + \frac{\lambda}{k}) = 0. \tag{10}$$

Произведем замену переменных в уравнении (12). Пусть:

$$(\frac{-\alpha}{kl} + \frac{\lambda}{k}) = n(n+1). \tag{11}$$

Тогда:

$$(1-z^2)\frac{\partial^2 X}{\partial x^2} - 2z\frac{\partial X}{\partial z} + n(n+1)X = 0. \tag{12}$$

Нетрудно заметить, что уравнение (14) является уравнением Лежандра, описанное в приложении А, решением которого являются полиномы Лежандра.

Функция Лежандра образует базис в некотором пространстве. Разложим функцию $\omega(z,t)$ в ряд, взяв за базис $P_n(z)$, и пусть $X_n(z) = P_n(z)$. Тогда будем искать решение в виде:

$$\omega(z,t) = \sum_{n=0}^{\infty} \hat{A}_n P_n(z) T_n(t). \tag{13}$$

Тогда для нахождения $\omega(z,t)$ из уравнения (15) требуется вычислить $T_n(t)$. Для этого из левой части уравнения (**??**) выразим T_n.

$$c\frac{T'}{T} = -\lambda;$$
$$\frac{dT}{Tdt} = \frac{-\lambda}{c};$$
$$\int \frac{dT}{T} = \int \frac{-\lambda dt}{c};$$

$$T_n = C_n e^{\frac{-\lambda_n t}{c}}, \tag{14}$$

где C_n - константа для n-го члена.

Тогда из формулы (13) видно, что λ_n, необходимая для вычисления T_n, будет находится следующим образом:

$$\lambda_n = k(n^2 + n + \frac{\alpha}{kl}). \tag{15}$$

Следовательно, уравнение (15) примет вид:

$$\omega(z,t) = \sum_{n=0}^{\infty} A_n P_n(z) e^{\frac{-k(n^2+n+\frac{\alpha}{kl})t}{c}}, \tag{16}$$

где $A_n = \hat{A_n} \cdot C_n$.

Воспользуемся начальным условием и зафиксируем t в момент времени $t = 0$ и получим:

$$\omega(z,t)|_{t=0} = \sum_{n=0}^{\infty} A_n P_n(z) = \psi(z) - u_c \tag{17}$$

Следует указать два важных свойства полиномов Лежандра:

$$\int_{-1}^{1} P_n(x) P_m(x) dx = 0, \text{ при } m \neq n; \tag{18}$$

$$\int_{-1}^{1} [P_n(x)]^2 = \frac{2}{2n+1} \tag{19}$$

Поэтому домножим функцию (19) на полином Лежандра порядка m и проинтегрируем по z, чтобы можно было применить свойство описанное в равенстве (20).

$$\omega(z)|_{t=0} = \sum_{n=0}^{\infty} A_n P_n(z) \Big| \cdot P_m(z);$$

$$(\psi(z) - u_c) P_m(z) = \sum_{n=0}^{\infty} A_n P_n(z) P_m(z);$$

$$\int_{-1}^{1} (\psi(z) - u_c) P_m(z) dz = \int_{-1}^{1} \sum_{n=0}^{\infty} A_n P_n(z) P_m(z) dz;$$

В силу свойства (20):

$$\int_{-1}^{1} (\psi(z) - u_c) P_m(z) dz = A_m \int_{-1}^{1} P_m^2(z) dz;$$

$$A_m = \frac{\int_{-1}^{1} (\psi(z) - u_c) P_m(z) dz}{\int_{-1}^{1} P_m^2(z) dz}.$$

Воспользуемся формулой (21) для нахождения знаменателя A_m и получим:

$$A_m = \frac{(2m+1)\int\limits_{-1}^{1}(\psi(z)-u_c)P_m(z)dz}{2}. \tag{20}$$

Таким образом было получено аналитическое решение поставленной задачи. Конечная формула ряда будет выглядеть следующим образом:

$$\omega(z,t) = \frac{1}{2}\sum_{n=0}^{\infty}(2n+1)e^{\frac{-k(n^2+n+\frac{\alpha}{kl})t}{c}}P_n(z)\int\limits_{-1}^{1}\left[(\psi(z)-u_c)P_n(z)\right]dz. \tag{21}$$

3 Получение расчетной формулы для начальной функции ψ_1 (Кирилин П. Н.)

Преподавателем была выдана функция, задающая начальное условие:

$$\psi(\theta) = u_c + \cos^4\theta. \tag{22}$$

С учетом замены переменной, которая была произведена ранее ($z = cos(\theta)$), функция $\psi(z)$ будет иметь вид:

$$\psi(z) = u_c + z^4.$$

Подставляя полученные данные в формулу (19), получим:

$$\omega(z,t)|_{t=0} = \sum_{n=0}^{\infty} A_n P_n(z) = z^4. \tag{23}$$

В следствии начального условия можно утверждать, что ряд конечен и полиномы Лежандра, включенные в ряд, не превышают 4-ого порядка. Тогда $\omega(z,t)|_{t=0}$ примет вид:

$$\omega(z,t)|_{t=0} = A_0 + A_1 z + A_2\frac{1}{2}(3z^2-1) + A_3\frac{1}{2}(5z^3-3z) + A_4\frac{1}{8}(35z^4-30z^2+3). \tag{24}$$

Раскрыв скобки и приведя подобные слагаемые, получим:

$$A_0 - \frac{1}{2}A_2 + \frac{3}{8}A_4 + z(A_1 - \frac{3}{2}A_3) + z^2(\frac{3}{2}A_2 - \frac{30}{8}A_4) + \frac{5}{2}A_3 z^3 + \frac{35}{8}A_4 z^4 = z^4 \tag{25}$$

Решим (27). Для этого составим следующую систему:

$$\begin{cases} A_0 - \frac{1}{2}A_2 + \frac{3}{8}A_4 = 0, \\ A_1 - \frac{3}{2}A_3 = 0, \\ \frac{3}{2}A_2 - \frac{30}{8}A_4 = 0, \\ \frac{5}{2}A_3 = 0, \\ \frac{35}{8}A_4 = 1. \end{cases} \tag{26}$$

Система (28) решается однозначно, и решение выглядит следующим образом:

$$\begin{cases} A_0 = \dfrac{1}{5}, \\ A_1 = 0, \\ A_2 = \dfrac{4}{7}, \\ A_3 = 0, \\ A_4 = \dfrac{8}{35}. \end{cases} \tag{27}$$

Подставив полученные значения A_n в получим:

$$\omega(z,t) = \frac{1}{5}e^{-\frac{\lambda_0 t}{c}} + \frac{4}{14}(3z^2-1)e^{-\frac{\lambda_2 t}{c}} + \frac{1}{35}(35z^4-30z^2+3)e^{-\frac{\lambda_4 t}{c}}, \tag{28}$$

где λ_n вычисляется по формуле (17).

Совершим обратные замены (**??**) и (**??**) для получения конечной формулы, удовлетворяющей поставленной задаче:

$$\omega(\theta,t) = \frac{1}{5}e^{-\frac{\lambda_0 t}{c}} + \frac{4}{14}(3\cos^2(\theta)-1)e^{-\frac{\lambda_2 t}{c}} + \frac{1}{35}(35\cos^4(\theta)-30\cos^2(\theta)+3)e^{-\frac{\lambda_4 t}{c}};$$

$$\begin{aligned} v(\theta,t) = \omega(\theta,t) + u_c = &\frac{1}{5}e^{-\frac{\lambda_0 t}{c}} + \frac{4}{14}(3\cos^2(\theta)-1)e^{-\frac{\lambda_2 t}{c}} + \frac{1}{35}(35\cos^4(\theta)- \\ &- 30\cos^2(\theta)+3)e^{-\frac{\lambda_4 t}{c}} + u_c. \end{aligned} \tag{29}$$

Из-за конечного количества элементов в ряде (31) остаток ряда отсутствует.

4 Получение расчетной формулы для начальной функции ψ_2 (Посоха К.А.)

Преподавателем была выдана функция, задающая начальное условие:

$$\psi(\theta) = u_c + \cos^5\theta. \tag{30}$$

После проведенной замены переменной, которая была произведена ранее ($z = cos(\theta)$), функция $\psi(z)$ примет вид:

$$\psi(z) = u_c + z^5.$$

Подставляя полученные данные в формулу (19), получим:

$$\omega(z,t)|_{t=0} = \sum_{n=0}^{\infty} A_n P_n(z) = z^5. \tag{31}$$

Из-за начального условия можно полагать, что ряд является конченым, состоящий из 6 слагаемых, а полиномы Лежандра, включенные в ряд, не будут превышать 5-ого порядка. Тогда $\omega(z,t)|_{t=0}$ примет вид:

$$\omega(z,t)|_{t=0} = A_0 + A_1 z + A_2\frac{1}{2}(3z^2-1) + A_3\frac{1}{2}(5z^3-3z) + A_4\frac{1}{8}(35z^4-30z^2+3) + \\ + A_5\frac{1}{8}(63z^5-70z^3+15z). \tag{32}$$

Раскрыв скобки и приведя подобные слагаемые, получим:

$$A_0 - \frac{1}{2}A_2 + \frac{3}{8}A_4 + z(A_1 - \frac{3}{2}A_3 + \frac{15}{8}A_5) + z^2(\frac{3}{2}A_2 - \frac{30}{8}A_4) + z^3(\frac{5}{2}A_3 - \frac{70}{8}A_5) + \\ + \frac{35}{8}A_4 z^4 + \frac{63}{8}A_5 z^5 = z^5 \tag{33}$$

Решим (35). Для этого составим следующую систему:

$$\begin{cases} A_0 - \frac{1}{2}A_2 + \frac{3}{8}A_4 = 0, \\ A_1 - \frac{3}{2}A_3 + \frac{15}{8}A_5 = 0, \\ \frac{3}{2}A_2 - \frac{30}{8}A_4 = 0, \\ \frac{5}{2}A_3 - \frac{70}{8}A_5 = 0, \\ \frac{35}{8}A_4 = 0, \\ \frac{63}{8}A_5 = 1. \end{cases} \tag{34}$$

Система (36) решается однозначно, и решение выглядит следующим образом:

$$\begin{cases} A_0 = 0, \\ A_1 = \dfrac{27}{63}, \\ A_2 = 0, \\ A_3 = \dfrac{28}{63}, \\ A_4 = 0, \\ A_5 = \dfrac{8}{63} \end{cases} \tag{35}$$

Подставив полученные значения A_n в получим:

$$\omega(z,t) = \frac{27}{63} z e^{-\frac{\lambda_1 t}{c}} + \frac{14}{63}(5z^3 - 3z)e^{-\frac{\lambda_3 t}{c}} + \frac{1}{63}(63z^5 - 70z^3 + 15z)e^{-\frac{\lambda_5 t}{c}}, \tag{36}$$

где λ_n вычисляется по формуле (17).

Совершим обратные замены (**??**) и (**??**) для получения конечной формулы, удовлетворяющей поставленной задаче:

$$\omega(\theta,t) = \frac{27}{63}\cos(\theta)e^{-\frac{\lambda_1 t}{c}} + \frac{14}{63}(5\cos^3(\theta) - 3\cos(\theta))e^{-\frac{\lambda_3 t}{c}} + \\ + \frac{1}{63}(63\cos^5(\theta) - 70\cos^3(\theta) + 15\cos(\theta))e^{-\frac{\lambda_5 t}{c}};$$

$$v(\theta,t) = \omega(\theta,t) + u_c = \frac{27}{63}\cos(\theta)e^{-\frac{\lambda_1 t}{c}} + \frac{14}{63}(5\cos^3(\theta) - 3\cos(\theta))e^{-\frac{\lambda_3 t}{c}} + \\ + \frac{1}{63}(63\cos^5(\theta) - 70\cos^3(\theta) + 15\cos(\theta))e^{-\frac{\lambda_5 t}{c}} + u_c. \tag{37}$$

Из-за конечного количества элементов в ряде (39) остаток ряда отсутствует.

5 Анализ погрешности решения и вычислительной сложности алгоритма

При помощи формулы (22) вычислим $\overline{A_0, A_4}$ для функции $\psi_1(z)$ и $\overline{A_0, A_5}$ для функции $\psi_2(z)$, проведем анализ погрешности решения и запишем результаты в таблицы 1 и 2 соответственно. Численное интегрирование будем проводить при помощи метода квадратур Гаусса-Лежандра, описанный в приложении В, который реализован в библиотеке Mpmath для языка программирования Python в функции “quad“. Как можно заметить, коэффициенты A_n, вычисленные с помощью системы уравнений (36) совпадают с коэффициентами, вычисленными с помощью аналитически выведенной формулы (22), следовательно погрешность решения будет равна погрешности вычислений.

Таблица 1 – Результаты исследования погрешности для $\psi_1(z)$.

	Результаты, высчитанные при помощи расчетной формулы	Результаты, высчитанные при помощи численного интегрирования	Δ
A_0	$\frac{1}{5}$	0.2	0
A_1	0	0	0
A_2	$\frac{4}{7}$	0.571	$4.440 \cdot 10^{-}16$
A_3	0	0	0
A_4	$\frac{8}{35}$	0.229	$4.440 \cdot 10^{-}16$

Таблица 2 – Результаты исследования погрешности для $\psi_2(z)$.

	Результаты, высчитанные при помощи расчетной формулы	Результаты, высчитанные при помощи численного интегрирования	Δ
A_0	0	0	0
A_1	$\frac{27}{63}$	0.428	$4.440 \cdot 10^-16$
A_2	0	0	0
A_3	$\frac{27}{63}$	0.(444)	$4.440 \cdot 10^-16$
A_4	0	0	0
A_5	$\frac{8}{63}$	0.127	$2.775 \cdot 10^-17$

Можно заметить, что погрешность вычисления метода численного интегрирования очень мала, однако необходимо проанализировать вычислительную сложность метода.

Алгоритм вычисления A_n состоит из 2 блоков, составляющих наибольшую вычислительную сложность: вычисление P_n и численное интегрирование, остальные операции тривиальны и выполняются за $O(1)$. Так как вычисления производятся последовательно, то асимптотической вычисления A_n будет сумма сложностей вычисления P_n и численного интегрирования.

Полиномы Лежандра, как описано в приложении А,вычисляются рекурсивной формулой. При правильной реализации вычисления n-го полинома Лежандра можно добиться асимптотической сложности $O(n)$. Для этого функции вычисления полиномов следует запоминать результаты вычисления полинома порядка $n - 1$ и возвращать два значения: P_n и P_{n-1}.

Для вычисления A_n также требуется алгоритм численного интегрирования. В данной курсовой работе использовался метод квадратур Гаусса-Лежандра. При правильной реализации данного метода потребуется 3 вложенных цикла, следовательно вычислительная сложность данного алгоритма будет составлять $O(n^3)$. Для того чтобы добиться такой асимптотической сложности следует воспользоваться модифицированной функцией вычислении полиномов Лежандра, которая была описана ранее и вычисляется за $O(n)$. У полинома Лежандра

порядка n существует n корней, которые находятся с помощью модифицированного метода Ньютона, упомянутого в приложении В и асимптотическая сложность которого, с учетом заранее вычисленных полиномов, будет составлять $O(1)$. Но, так как корней n сложность вычисления всех корней полинома Лежандра порядка n будет равна $O(n)$. Суммируя все результаты по n узлам интегрирования, получим общую асимптотическую сложность алгоритма численного интегрирования равную $O(n^3)$. Следовательно общая вычислительная сложность нахождения коэффициентов A_n будет составлять $O(n) + O(n^3) = O(n^3)$

6 Результаты вычислительных экспериментов

Ниже показан пример использования программы для моделирования теплового процесса в тонкой сфере. Для расчета и визуализации функции $\omega(\theta, t)$ программа вызывается с аргументом ”pic”.

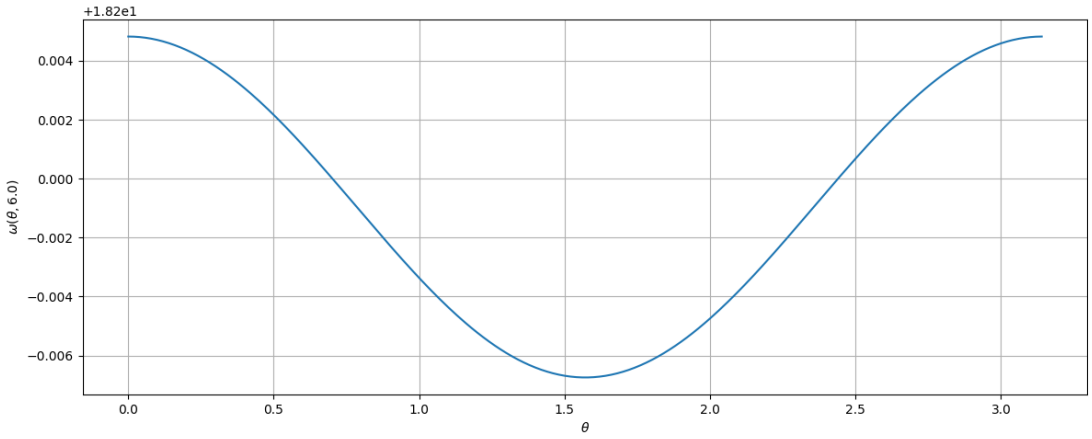
Для варьирования параметров, выданных преподавателем, следует передавать их как ”флаги” программы. Например, для изменения параметра ”a”(α) следует написать ”python umf.py pic –a=0.4”, теперь при расчетах коэффициент теплопроводности α будет равен $0.4\left[\frac{\text{Вт}}{\text{см}^2 \cdot \text{К}}\right]$. Стандартные параметры указаны преподавателем.

Для генерации видео программа должна быть запущена с параметром ”draw”.

Для расчетов коэффициентов A_n программа запускается с аргументом ”find-a”.

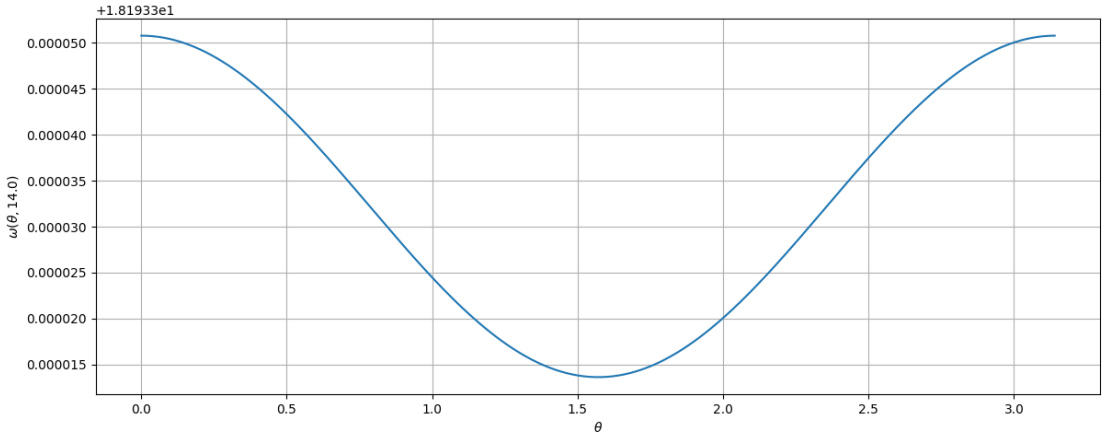
Для вызова подсказок следует дописать ”– –help” в конец команды. Система подсказок работает для любой команды. Все подсказки написаны на английском языке, для корректного встраивания текста программы в файл отчета.

Ниже, на рисунках 1, 2, 3 для начального условия $\psi_1(\theta)$ и 4, 5, 6 для $\psi_2(\theta)$, представлены выводы программы зависимости теплоты от угловой координаты в различные моменты времени t.



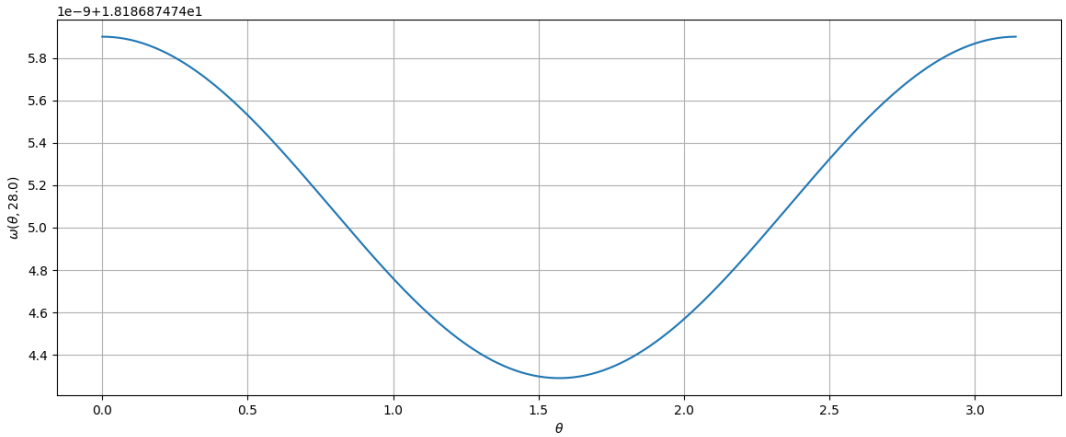
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 1 – График температуры от времени t=6c при $\psi_1(\theta)$



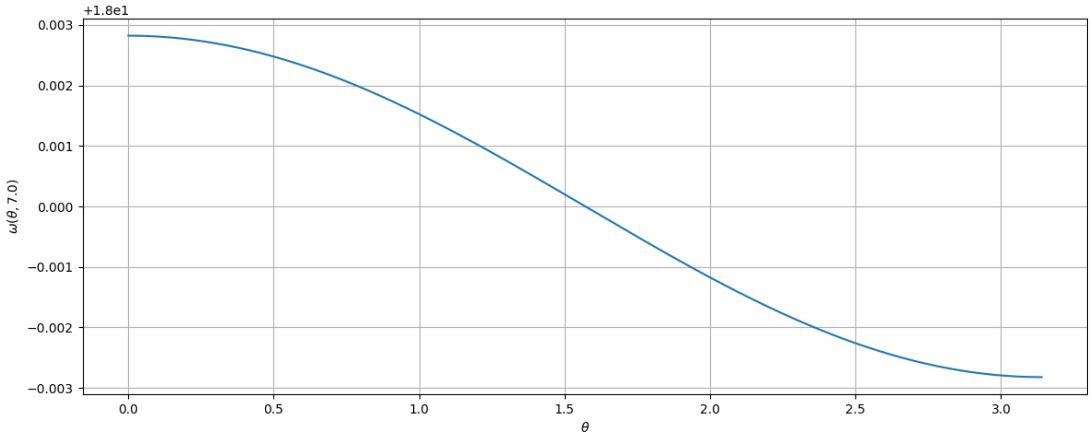
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 2 – График температуры от времени t=14c при $\psi_1(\theta)$



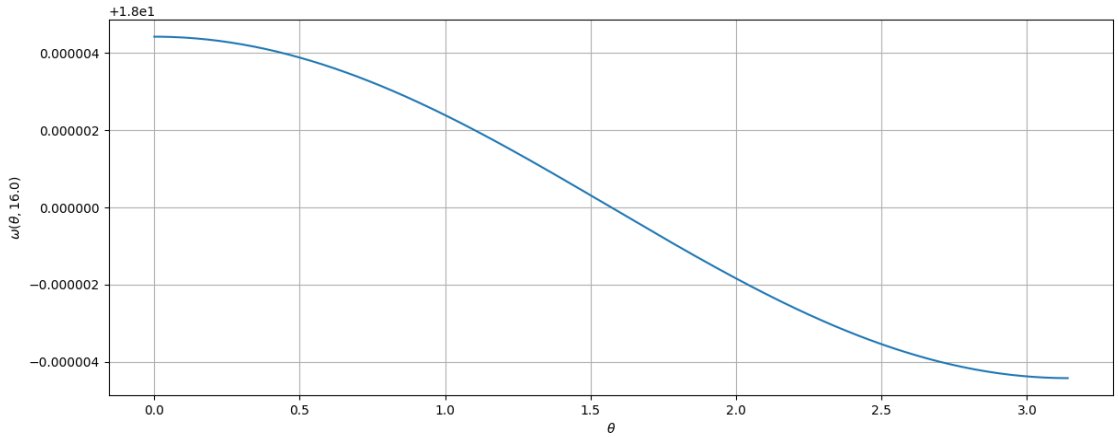
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 3 – График температуры от времени t=28c при $\psi_1(\theta)$



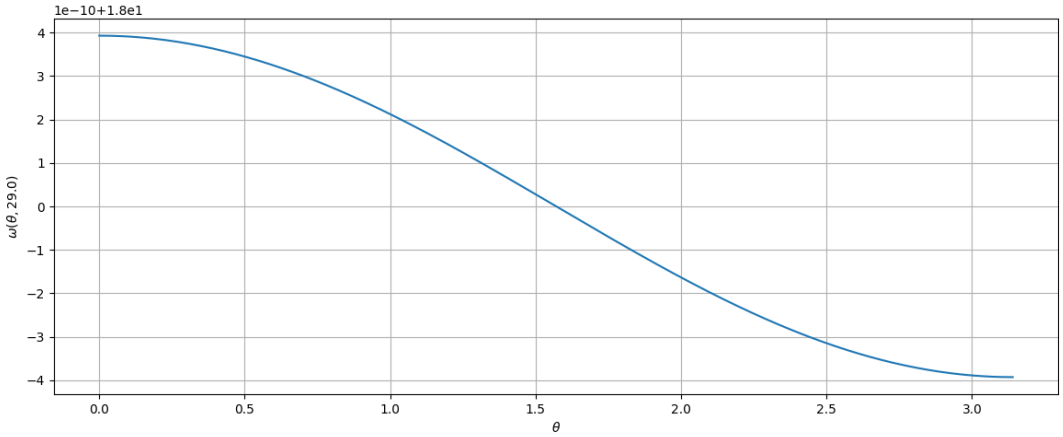
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 4 – График температуры от времени t=7c при $\psi_2(\theta)$



Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

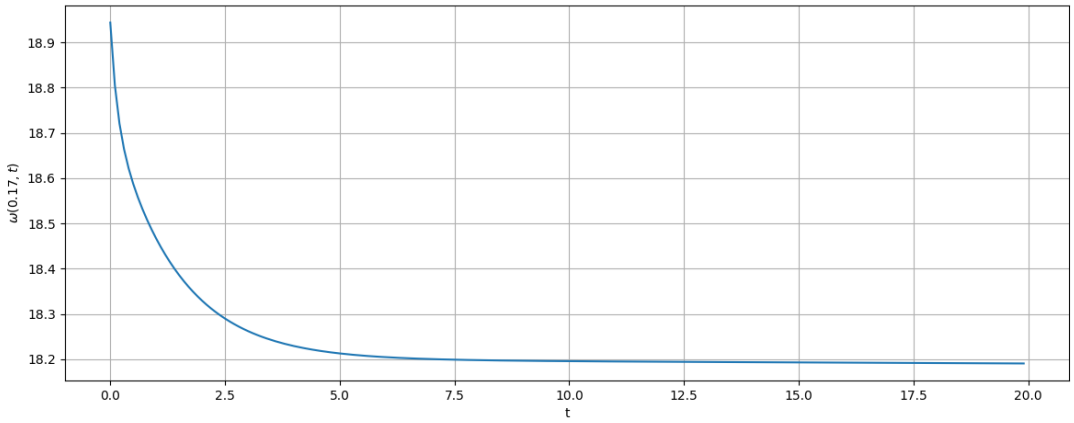
Рисунок 5 – График температуры от времени t=16c при $\psi_2(\theta)$



Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

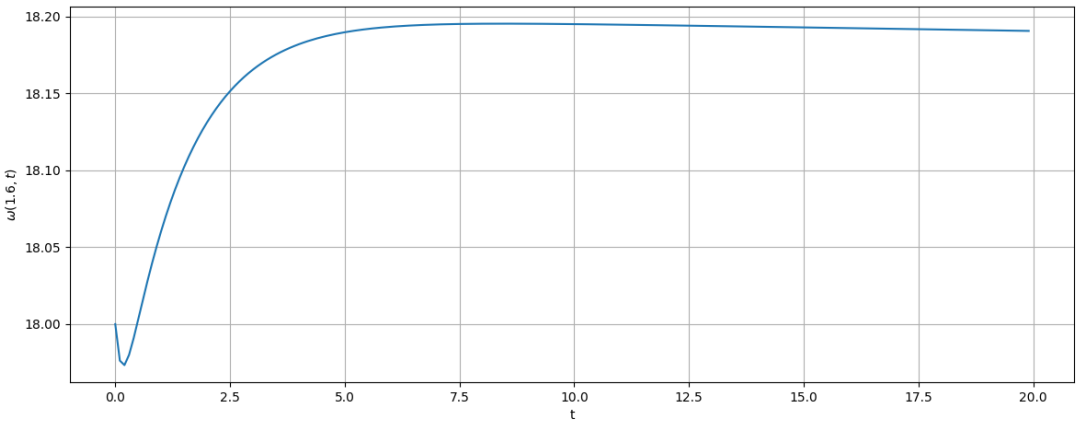
Рисунок 6 – График температуры от времени t=29c при $\psi_2(\theta)$

На рисунках 7, 8, 9 для начального условия $\psi_1(\theta)$ и 10, 11, 12 для $\psi_2(\theta)$, представлены выводы программы зависимости теплоты от времени для различной угловой координаты θ.



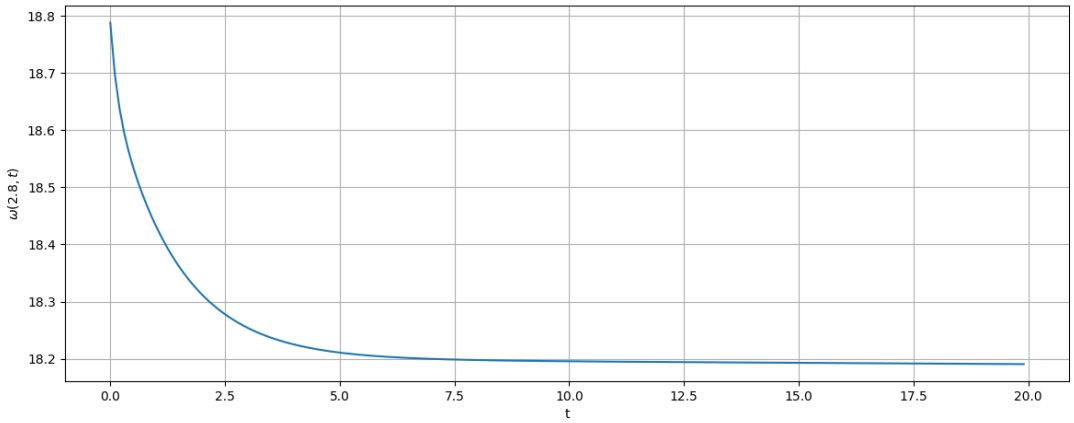
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 7 – График температуры от времени $\theta = 0.17$ *при* $\psi_1(\theta)$



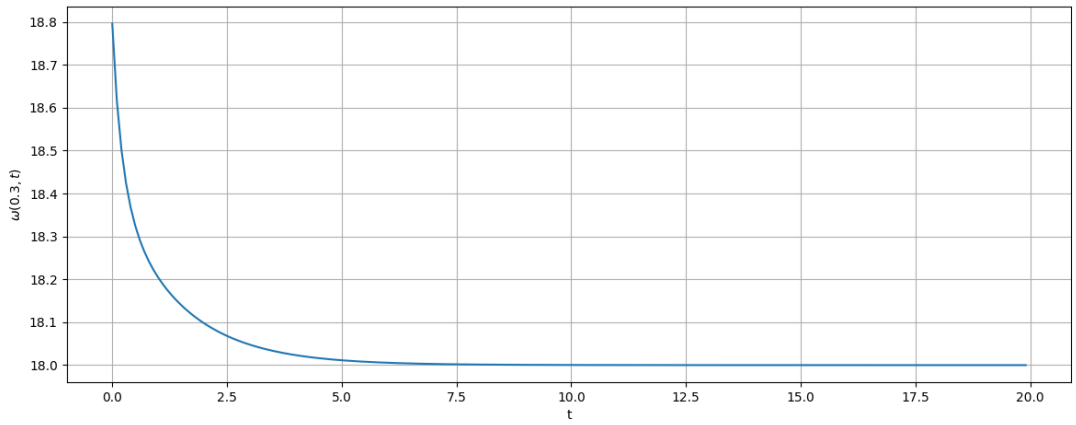
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 8 – График температуры от времени $\theta = 1.6$ *при* $\psi_1(\theta)$



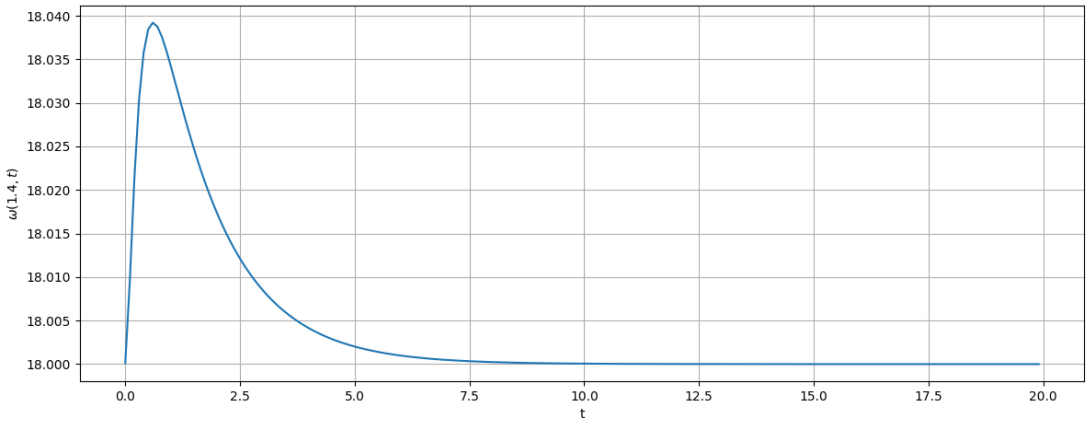
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 9 – График температуры от времени $\theta = 2.8$ *при* $\psi_1(\theta)$



Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 10 – График температуры от времени $\theta = 0.3$ при $\psi_2(\theta)$



Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

Рисунок 11 – График температуры от времени $\theta = 1.4$ при $\psi_2(\theta)$

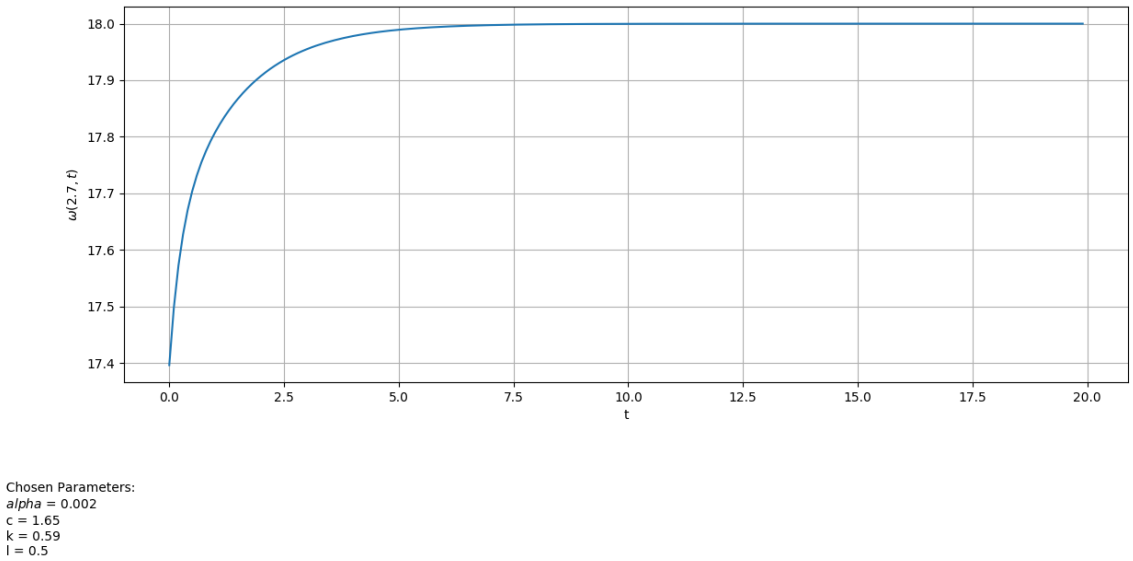


Рисунок 12 – График температуры от времени $\theta = 2.7$ при $\psi_2(\theta)$

На рисунках с 7 по 12 можно заметить, что в любой точке сферы с увеличением времени будет устанавливаться тепловой баланс. А на рисунках с 1 по 6 видна зависимость распределения температуры на оболочке сферы от начального условия.

Заключение

При выполнении данной работы были изучены аналитические методы решения краевой задачи теплопроводности. Составлена постановка краевой задачи, а также было получено аналитеческое решение в виде конечного ряда Фурье-Лежандра.

В ходе работы для решения задачи была написана программа численного моделирования теплового процесса нагревания тонкой оболочки сферы. Данная программа выполняет расчеты темпиратуры с заданными параметрами и позволяет провести анализ погрешности решения для функции начального условия.

Анализ полученных результатов показал, что была достигнута достаточно высокая точность решения, при относительно низкой вычислительной сложности. Что является хорошим результатом выполненной работы.

Список литературы

1. **Дегтярев А.А.** Примеры построения и исследования разностных схем. – Электронное учебное пособие [Электронный ресурс] / А.А Дегтярев, 2011. - 54с.

2. **Тихонов А.Н., Самарский А.А**. Уравнения математической физики(5-е изд.) [Текст] / Тихонов А.Н., Самарский А.А. М.: Наука, 1977. - 742 с.

3. Numpy and Scipy Documentation [Электронный ресурс]: Официальный сайт документации библиотек Numpy и Scipy. - URL: https://docs.scipy.org/doc/

4. MpMath Documentation [Электронный ресурс]: Официальный сайт документации библиотеки MpMath. - URL: http://mpmath.org/doc/current/

Приложение А

Полиномы Лежандра

Полином Лежандра[2] — многочлен, который в наименьшей степени отклоняется от нуля в смысле среднего квадратического.

Рекуррентная формула для вычисления n-го многочлена:

$$\begin{cases} P_0(x) = 1; \\ P_1(x) = x; \\ P_{n+1}(x) = \dfrac{2n+1}{n+1} x P_n(x) - \dfrac{n}{n+1} P_{n-1}(x). \end{cases} \tag{38}$$

Первые многочлены Лежандра:

$$P_0(x) = 1;$$
$$P_1(x) = x;$$
$$P_2(x) = \frac{1}{2}(3x^2 - 1);$$
$$P_3(x) = \frac{1}{2}(5x^3 - 3x);$$
$$P_4(x) = \frac{1}{8}(35x^4 - 30x^2 + 3);$$
$$P_5(x) = \frac{1}{8}(63x^5 - 70x^3 + 15x).$$

Приложение Б

Код программы

```
from functools import lru_cache

import fire
import numpy
from matplotlib import animation
from matplotlib import pyplot as plt
from mpmath import quad, legendre
from sympy import Symbol, legendre_poly


@lru_cache(maxsize=123)
def legendre_p(n, x):
    if n == 0:
        return 1
    elif n == 1:
        return x


def lam_n(n, k, a, l):
    return k * (n ** 2 + n + (a / (k * l)))


def draw_graphics(variant, const_k, const_c, const_a, const_l, frames, fps):
    def animate(val):
        time = val / 10

        def super_exp(n):
            return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

        x_s = []
        y_s = []
        for j in numpy.arange(-1, 1, 0.01):
            if variant == 4:
                summary = (1 / 5) * super_exp(0) + (2 / 7) * (3 * (j ** 2) - 1) * super_exp
```

```
                ↪ (1) + (1 / 35) * (
                    35 * (j ** 4) - 30 * (j ** 2) + 3) * super_exp(4)
            else:
                summary = (27 / 63) * j * super_exp(1) + (14 / 63) * (5 * (j ** 3) - 3 * j)
                ↪ * super_exp(3) + (
                    1 / 63) * (63 * (j ** 5) - 70 * (j ** 3) + 15 * j) * super_exp(5)
            x_s.append(j)
            y_s.append(summary)
        ax1.clear()
        ax1.set_xlabel("Z")
        ax1.set_ylabel("ω(z,)".format(time))
        ax1.plot(x_s, y_s)

    plt.rc('font', **{'serif': ['Computer Modern']})
    plt.rc('text', usetex=True)
    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)

    anim = animation.FuncAnimation(fig, animate, interval=1, frames=frames)

    anim.save('variant-time-{}.mp4'.format(variant), fps=fps, extra_args=['-vcodec', '
        ↪ libx264'])
    plt.show()


def draw_graphics2(variant, const_k, const_c, const_a, const_l, frames, fps, u_c):
    def animate(val):
        time = val / 10

        def super_exp(n):
            return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

        x_s = []
        y_s = []
        for j in numpy.arange(0, numpy.pi, 0.01):
            if variant == 4:
                summary = (1 / 5) * super_exp(0) + (2 / 7) * (3 * (numpy.cos(j) ** 2) - 1) *
                    ↪  super_exp(1) + (1 / 35) * (
```

```
                    35 * (numpy.cos(j) ** 4) - 30 * (numpy.cos(j) ** 2) + 3) * super_exp
                        ↪ (4)
            else:
                summary = (27 / 63) * numpy.cos(j) * super_exp(1) + (14 / 63) * (
                        5 * (numpy.cos(j) ** 3) - 3 * numpy.cos(j)) * super_exp(3) + (
                                1 / 63) * (63 * (numpy.cos(j) ** 5) - 70 * (numpy.cos(j) **
                                    ↪ 3) + 15 * numpy.cos(
                    j)) * super_exp(5)
            x_s.append(j)
            y_s.append(summary + u_c)
        ax1.clear()
        plt.grid()
        ax1.set_xlabel("$theta$")
        ax1.set_ylabel("$\omega(theta,)$".format(time))
        ax1.plot(x_s, y_s)

    plt.rc('font', **{'serif': ['Computer Modern']})
    plt.rc('text', usetex=True)
    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)

    anim = animation.FuncAnimation(fig, animate, interval=1, frames=frames)

    anim.save("variant-time-{}.mp4".format(variant), fps=fps, extra_args=['-vcodec', '
        ↪ libx264'])
    plt.show()


def draw_graphics_by_z(variant, const_k, const_c, const_a, const_l, frames, fps):
    def animate(val):
        z = val % 100
        if z < 50:
            z = 50 - z
            z *= -1
        elif z == 50:
            z = 0
        else:
            z -= 50
```

```
        z /= 50

        def super_exp(n, time):
            return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

        x_s = []
        y_s = []
        for j in numpy.arange(0, 20, 0.01):
            if variant == 4:
                summary = (1 / 5) * super_exp(0, j) + (2 / 7) * (3 * (z ** 2) - 1) *
                    ↪ super_exp(1, j) + (1 / 35) * (
                        35 * (z ** 4) - 30 * (z ** 2) + 3) * super_exp(4, j)
            else:
                summary = (27 / 63) * z * super_exp(1, j) + (14 / 63) * (5 * (z ** 3) - 3 *
                    ↪ z) * super_exp(3, j) + (
                        1 / 63) * (63 * (z ** 5) - 70 * (z ** 3) + 15 * z) * super_exp(5, j)
            x_s.append(j)
            y_s.append(summary)
        ax1.clear()
        ax1.set_xlabel("t")
        ax1.set_ylabel("ω(,t)".format(z))
        ax1.plot(x_s, y_s)

    plt.rc('font', **{'serif': ['Computer Modern']})
    plt.rc('text', usetex=True)
    fig = plt.figure()
    ax1 = fig.add_subplot(1, 1, 1)

    anim = animation.FuncAnimation(fig, animate, interval=1, frames=frames)

    anim.save('variant-z-{}.mp4'.format(variant), fps=fps, extra_args=['-vcodec', 'libx264
        ↪ '])
    plt.show()


def draw_graphics_by_z2(variant, const_k, const_c, const_a, const_l, frames, fps, u_c):
    def animate(val):
        theta = (numpy.pi * val) / frames
```

```
    def super_exp(n, time):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

    x_s = []
    y_s = []
    for j in numpy.arange(0, 20, 0.01):
        if variant == 4:
            summary = (1 / 5) * super_exp(0, j) + (2 / 7) * (3 * (numpy.cos(theta) ** 2)
                ↪  - 1) * super_exp(1, j) + (
                    1 / 35) * (
                            35 * (numpy.cos(theta) ** 4) - 30 * (numpy.cos(theta) ** 2)
                                ↪ + 3) * super_exp(4, j)
        else:
            summary = (27 / 63) * numpy.cos(theta) * super_exp(1, j) + (14 / 63) * (
                   5 * (numpy.cos(theta) ** 3) - 3 * numpy.cos(theta)) * super_exp(3, j)
                       ↪  + (
                            1 / 63) * (
                            63 * (numpy.cos(theta) ** 5) - 70 * (numpy.cos(theta) ** 3)
                                ↪ + 15 * numpy.cos(
                         theta)) * super_exp(5, j)
        x_s.append(j)
        y_s.append(summary + u_c)
    ax1.clear()
    ax1.set_xlabel("t")
    ax1.set_ylabel("ω(,t)".format(theta))
    ax1.plot(x_s, y_s)

plt.rc('font', **{'serif': ['Computer Modern']})
plt.rc('text', usetex=True)
fig = plt.figure()
ax1 = fig.add_subplot(1, 1, 1)

anim = animation.FuncAnimation(fig, animate, interval=1, frames=frames)

anim.save('variant-z-{}.mp4'.format(variant), fps=fps, extra_args=['-vcodec', 'libx264
    ↪ '])
plt.show()
```

```
def show_image(const_k, const_c, const_a, const_l, x_s, y_s, label_x, label_y):
    plt.figure()
    plt.grid()
    plt.gca().set_position((.1, .3, .8, .6))
    plt.rc('font', **{'serif': ['Computer Modern']})
    plt.rc('text', usetex=False)
    plt.plot(y_s, x_s)
    plt.xlabel(label_y)
    plt.ylabel(label_x)
    plt.figtext(
        .0, .0,
        " Chosen Parameters:\n $alpha$ = {0}\n c = {1}\n k = {2}\n l = {3}\n".format(const_a,
            ↪  const_c, const_k, const_l)
    )
    plt.show()


def show_by_time(variant, const_k, const_c, const_a, const_l, time):
    def super_exp(n):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

    x_s = []
    y_s = []
    for j in numpy.arange(-1, 1, 0.01):
        if variant == 4:
            summary = (1 / 5) * super_exp(0) + (2 / 7) * (3 * (j ** 2) - 1) * super_exp(1)
                ↪ + (1 / 35) * (
                    35 * (j ** 4) - 30 * (j ** 2) + 3) * super_exp(4)
        else:
            summary = (27 / 63) * j * super_exp(1) + (14 / 63) * (5 * (j ** 3) - 3 * j) *
                ↪ super_exp(3) + (
                    1 / 63) * (63 * (j ** 5) - 70 * (j ** 3) + 15 * j) * super_exp(5)
        x_s.append(j)
        y_s.append(summary)
    show_image(const_k, const_c, const_a, const_l, x_s, y_s, "z", "$\omega(z,t)$")
```

```
def show_by_time2(variant, const_k, const_c, const_a, const_l, time, u_c):
    def super_exp(n):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

    x_s = []
    y_s = []
    for j in numpy.arange(0, numpy.pi, 0.01):
        if variant == 4:
            summary = (1 / 5) * super_exp(0) + (2 / 7) * (3 * (numpy.cos(j) ** 2) - 1) *
                ↪ super_exp(1) + (1 / 35) * (
                    35 * (numpy.cos(j) ** 4) - 30 * (numpy.cos(j) ** 2) + 3) * super_exp(4)
        else:
            summary = (27 / 63) * numpy.cos(j) * super_exp(1) + (14 / 63) * (
                    5 * (numpy.cos(j) ** 3) - 3 * numpy.cos(j)) * super_exp(3) + (
                            1 / 63) * (
                            63 * (numpy.cos(j) ** 5) - 70 * (numpy.cos(j) ** 3) + 15 * numpy
                                ↪ .cos(j)
                    ) * super_exp(5)
        x_s.append(summary + u_c)
        y_s.append(j)
    show_image(const_k, const_c, const_a, const_l, x_s, y_s, "ω(theta,)".format(time), "
        ↪ theta")


def show_by_z(variant, const_k, const_c, const_a, const_l, z):
    def super_exp(n, time):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

    x_s = []
    y_s = []
    for j in numpy.arange(0, 20, 0.1):
        if variant == 4:
            summary = (1 / 5) * super_exp(0, j) + (2 / 7) * (3 * (z ** 2) - 1) * super_exp
                ↪ (1, j) + (1 / 35) * (
                    35 * (z ** 4) - 30 * (z ** 2) + 3) * super_exp(4, j)
        else:
            summary = (27 / 63) * z * super_exp(1, j) + (14 / 63) * (5 * (z ** 3) - 3 * z)
```

```
            ↪ * super_exp(3, j) + (
                1 / 63) * (63 * (z ** 5) - 70 * (z ** 3) + 15 * z) * super_exp(5, j)
        x_s.append(summary)
        y_s.append(j)
    show_image(const_k, const_c, const_a, const_l, x_s, y_s, "ω(,t)".format(z), "t")


def show_by_z2(variant, const_k, const_c, const_a, const_l, theta, u_c):
    def super_exp(n, time):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * time / const_c)

    x_s = []
    y_s = []
    for j in numpy.arange(0, 20, 0.1):
        if variant == 4:
            summary = (1 / 5) * super_exp(0, j) + (2 / 7) * (3 * numpy.cos(theta) ** 2 - 1)
                ↪  * super_exp(1, j) + (
                1 / 35) * (
                        35 * (numpy.cos(theta) ** 4) - 30 * (numpy.cos(theta) ** 2) + 3)
                            ↪  * super_exp(4, j)
        else:
            summary = (27 / 63) * numpy.cos(theta) * super_exp(1, j) + (14 / 63) * (
                  5 * (numpy.cos(theta) ** 3) - 3 * numpy.cos(theta)) * super_exp(3, j) +
                      ↪ (
                          1 / 63) * (
                          63 * (numpy.cos(theta) ** 5) - 70 * (numpy.cos(theta) ** 3) + 15
                              ↪  * numpy.cos(
                       theta)) * super_exp(5, j)
        x_s.append(summary + u_c)
        y_s.append(j)
    show_image(const_k, const_c, const_a, const_l, x_s, y_s, "ω(,t)".format(theta), "t")


def get_A_n(variant, n, accuracy):
    return (quad(lambda x: legendre(n, x) * x ** variant, [-1, 1], method='gauss-legendre'
        ↪ , maxdegree=accuracy) * (
            (2 * n) + 1)) / 2
```

```
def find_const(variant, count, accuracy):
    A_ns = []
    for i in range(count):
        A_n = get_A_n(variant, i, accuracy)
        print("A_{0} = {1}".format(i, A_n))
        A_ns.append(A_n)
    return A_ns


def calculate_series(z, t, variant, const_k, const_c, const_a, const_t, const_l, count):
    def super_exp(n):
        return numpy.exp(-lam_n(n, const_k, const_a, const_l) * const_t / const_c)

    z = Symbol('z')
    for i in range(count):
        poly = legendre_poly(i, z)
        sum += (2 * i) * super_exp(i) * poly * get_A_n(variant, i)


class CLI(object):
    """
    To show graphics call "draw" as argument of program.
    To calculate A_n call "find-a" as argument of program.
    To calculate ω(z,t) series, call "series" as argument of program.
    """

    def draw(self, power=4, k=0.59, c=1.65, a=2e-3, l=0.5, u_c=18, frames=200, fps=60,
        ↪ gtype='time'):
        """
        Function to visualize heat process.
        :param power: Power of given function. ψ. ψ(z) = z^power
        :param k: Thermal conductivity coefficient.
        :param c: bulk heat capacity.
        :param a: heat transfer coefficient.
        :param l: shell thickness.
        :param frames: The number of frames of animation.
        :param fps: Frames per second.
```

```
        """
        print("-" * 20)
        print("Chosen params:")
        print('k = {}'.format(k))
        print('c = {}'.format(c))
        print('alpha = {}'.format(a))
        print('l = {}'.format(l))
        print('frames will be shown: {}'.format(frames))
        print('Frames per second: {}'.format(fps))
        if gtype == 'time':
            draw_graphics2(int(power), float(k), float(c), float(a), float(l), int(frames),
                ↪  int(fps), float(u_c))
        else:
            draw_graphics_by_z2(int(power), float(k), float(c), float(a), float(l), int(
                ↪ frames), int(fps), float(u_c))

    def find_a(self, power=4, count=5, accuracy=2):
        """
        Function will calculate A_n for chosen power.
        :param power: Power of given function. ψ. ψ(z) = z^power
        :param count: Quantity of A_n. A_0 ... A_count will be calculated.
        :param accuracy: Calculus accuracy.
        """
        find_const(int(power), count, accuracy)

    def pic(self, power=4, k=0.59, c=1.65, a=2e-3, l=0.5, time=20, z=0, gtype='time', u_c
        ↪ =18):
        """
        Function to show picture of ω(z,t) with t=time
        :param gtype: type of graphic. Possible values: 'time' and 'z'.
        :param z: z value
        :param power: Power of given function. ψ. ψ(z) = z^power
        :param k: Thermal conductivity coefficient.
        :param c: bulk heat capacity.
        :param a: heat transfer coefficient.
        :param l: shell thickness.
        :param time: time parameter.
        :param u_c: Environment temperature.
```

```
        """
        print("-" * 20)
        print("Chosen params:")
        print('k = {}'.format(k))
        print('c = {}'.format(c))
        print('alpha = {}'.format(a))
        print('l = {}'.format(l))
        if gtype == 'time':
            show_by_time2(int(power), float(k), float(c), float(a), float(l), float(time),
                ↪ float(u_c))
        else:
            show_by_z2(int(power), float(k), float(c), float(a), float(l), float(z), float(
                ↪ u_c))

    def series(self, power=4, k=0.59, c=1.65, a=2e-3, l=0.5, t=20, count=5):
        """
        Function to calculate series $\omega(z,t)|_{t=0} = \sum_{n=0}^{\text{count}} A_n P_n(z)$
        :param power: Power of given function. $\psi$. $\psi(z) = z^{\text{power}}$
        :param count: Series elements quantity.
        :param k: Thermal conductivity coefficient.
        :param c: bulk heat capacity.
        :param a: heat transfer coefficient.
        :param l: shell thickness.
        """
        calculate_series(int(power), float(k), float(c), float(a), float(t), float(l),
            ↪ float(count))


if __name__ == "__main__":
    fire.Fire(CLI)
```

Приложение В

Метод квадратур Гаусса-Лежандра

Метод квадратур Гаусса-Лежандра - метод численного интегрирования, в основе которого лежит метод квадратур Гаусса. Особенностью метода является то, что он позволяет повысить алгебраический порядок точности, без увеличения числа используемых значений подынтегральной функции. Тем самым может быть достигнута максимальная точность для выбранного числа узлов интегрирования. Численное интегрирование методом Гаусса вычисляется по формуле:

$$I \approx \sum_{i=1}^{n} \omega_i f(x_i), \tag{39}$$

где x_i - узлы интегрирования, а ω_i - ненулевая константа, называемая “вес квадратуры“.

Для вычисления методом Гаусса-Лежандра веса вычисляются по следующей формуле:

$$\omega_i = \frac{2}{(1 - x_i^2)\left[P_n'(x_i)\right]^2}, \tag{40}$$

где $P_n'(x_i)$ - первая производная n-го полинома Лежандра, который описан в приложении А, x_i - корни полинома Лежандра.

Для нахождения производной многочлена Лежандра n-го порядка используется следующая рекуррентная формула:

$$P_n'(x) = \frac{n}{x^2 - 1}\left(xP_n(x) - P_{n-1}(x)\right) \tag{41}$$

Многочлены Лежандра, в основном, не имеют аналитического решения и находятся через аппроксимирующие численные методы для нахождения корней. На практике чаще всего используется метод касательной (Модификация метода Ньютона), который вычисляется по формуле:

$$x_{i+1} = x_i - \frac{P_n(x_i)}{P_n'(x_i)} \tag{42}$$

Для начального приближения i-го корня полинома Лежандра степени n берется

$$x_0 = \cos\left(\pi \frac{i - \frac{1}{4}}{n + \frac{1}{2}}\right)$$

Конечная формула для численного интегрирования:

$$\int_a^b f(x)dx \approx \frac{b-a}{2} \sum_{i=1}^{n} \omega_i f\left(\frac{b-a}{2}x_i + \frac{a+b}{2}\right) \tag{43}$$