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

35 страниц, 12 рисунков, 1 таблиц, 0 источников, 1 приложение

УРАВНЕНИЯ МАТЕМАТИЧЕСКОЙ ФИЗИКИ, КРАЕВАЯ ЗАДАЧА, УРАВНЕНИЕ ПОПЕРЕЧНЫХ КОЛЕБАНИЙ ПРЯМОУГОЛЬНОЙ МЕМБРАНЫ, МЕТОД РАЗДЕЛЕНИЯ ПЕРЕМЕННЫХ, СОБСТВЕННЫЕ ФУНКЦИИ ОПЕРАТОРА ЛАПЛАСА, РЯД ФУРЬЕ.

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:

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

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:

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

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

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 x 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:

$$T'' + \lambda^2\alpha^2 T = 0;$$
$$X'' + (\lambda^2 - \frac{\pi^2}{l_y^2})X = 0.$$

Общие решения этих уравнений будут иметь следующий вид:

$$X(x) = A\cos(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}x) + B\sin(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}x), \tag{10}$$

$$T(t) = C\cos(\alpha\lambda t) + D\sin(\lambda t), \tag{11}$$

где A, B, C, D - произвольные постоянные.

Подставляя полученные выражения $X(t), T(t)$ в равенство (7), получаем:

$$\nu(x,t) = (A\cos(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}x) + B\sin(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}x))(C\cos(\alpha\lambda t) + D\sin(\lambda t)). \tag{12}$$

Произведем подбор постоянных A, B, так, чтобы полученное выражение удовлетворяло граничным условиям (6).

Для удовлетворяющей указанным условиям функции $X(x)$ должно выполняться $X(0) = 0, X(l_x) = 0$. Подставляя значения $x = 0, x = l_x$ в (10), получаем:

$$0 = A\cos(0) + B\sin(0), \qquad 0 = A\cos(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}l_x) + B\sin(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}l_x).$$

Из первого уравнения очевидно следует $A = 0$. Из второго уравнения находим:

$$B\sin(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}l_x) = 0. \tag{13}$$

$B \neq 0$, так как в противном случае было бы $X = 0$ и $u = 0$, что противоречит условию. Следовательно,

$$\sin(\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2}l_x) = 0,$$

Откуда находим

$$\sqrt{\frac{\pi^2}{l_y^2} - \lambda^2} = \frac{n\pi}{l_x}, n = 1, 2, ...,$$
$$\lambda = \sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}.$$

Итак, получаем подстановкой λ в (13) вид функции $X(x)$:

$$X(x) = B\sin(\sqrt{\frac{n\pi^2}{l_x^2}}x) = B\sin(\frac{n\pi^2}{l_x^2}x).$$

Теперь, зная значение λ, можем записать для уравнения (11):

$$T(t) = C\cos(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t) + D\sin(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t), n = 1, 2, ...,$$

Подставляя для каждого n полученные выражения в уравнение (7), получаем решение исходного уравнения (6), удовлетворяющее требуемым условиям. Обозначим это решение как $\nu_n(x,t)$:

$$\nu_n(x,t) = (C\cos(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t) + D\sin(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t))\sin(\frac{n\pi}{l_x}x) \tag{14}$$

Так как уравнение (6) является линейным и однородным, решение можно представить в виде ряда:

$$\nu(x,t) = \sum_{n=1}^{\infty}(C\cos(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t) + D\sin(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t))\sin(\frac{n\pi}{l_x}x). \tag{15}$$

Для удовлетворения начальным условиям требуется подобрать значения C_n и D_n. По условию $\nu|_{t=0} = p(x,y)$. Соответственно, подставив в (15) значение $t = 0$, получим:

$$p(x,y) = \sum_{n=1}^{\infty}\sin(\frac{n\pi}{l_x}x)C_n. \tag{16}$$

Если функция $p(x,y)$ такова, что в интервале $(0, l_x)$ ее можно разложить в ряд Фурье, условие (16) будет выполняться при:

$$C_n = \frac{2}{l_x}\int_0^{l_x} p(x)\sin(\frac{n\pi}{l_x}x)dx. \tag{17}$$

По условию $\frac{\partial\nu}{\partial t}|_{t=0} = 0$. Дифференцируем члены уравнения (15) по t и учитываем это условие:

$$0 = \sum_{n=1}^{\infty} D_n\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}\sin(\frac{n\pi}{l_x}x). \tag{18}$$

Очевидно, что $\alpha \neq 0$, следовательно, из (18) получаем $D_n = 0$.

Таким образом, приходим к аналитическому решению поставленной задачи (6) в следующем виде:

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

$$\nu(x,t) = \sum_{n=1}^{\infty}(C_n\cos(\alpha\sqrt{\frac{n\pi^2}{l_x^2} + \frac{\pi^2}{l_y^2}}t))\sin(\frac{n\pi}{l_x}x), \tag{19}$$

$$C_n = \frac{2}{l_x}\int_0^{l_x} p(x)\sin(\frac{n\pi}{l_x}x)dx.$$

3 Получение расчетной формулы

Преподавателем была выдана функция $p(x, y)$, задающая начальное отклонение мембраны и значение поверхностной плотности ρ:

$$p(x, y) = \frac{x^2}{4} + x, \quad \rho = 1.$$

Размеры мембраны были заданы как $l_x = 4, \quad l_y = 1$.

Из вывода уравнения колебаний мембраны следует, что коэффициент α^2 равен $\frac{1}{\rho}$. Следовательно, в нашем случае $\alpha = 1$ и в дальнейших расчетах не оказывает влияния.

Вычислим значение коэффициента C_n в сумме ряда (19) с учетом выданных значений.

$$C_n = \frac{2}{l_x} \int_0^{l_x} p(x) \sin(\frac{n\pi}{l_x} x) dx = \frac{1}{2} \int_0^4 p(x) \sin(\frac{n\pi}{4} x) dx = 8 \frac{n\pi \sin(n\pi) + 2\cos(n\pi) - 2}{n^3 \pi^3}.$$

Перепишем решение задачи с учетом вычисленного коэффициента C_n и выданных значений, получая итоговые расчетные формулы:

$$u(x, y, t) = \nu(x, t) \sin(\pi y),$$

$$\nu(x, t) = \sum_{n=1}^{\infty} 8 \frac{n\pi \sin(n\pi) + 2\cos(n\pi) - 2}{n^3 \pi^3} \cos(\sqrt{\frac{n\pi^2}{16} + \pi^2} t) \sin(\frac{n\pi}{4} x). \tag{20}$$

4 Оценка остатка ряда

Оценим остаток полученного ранее ряда Фурье (20). Далее обозначаем n-ый элемент ряда как $D_n(x,t)$. Остаток оценивается как сумма элементов ряда, начиная с $N+1$-го:

$$|R_n| = \left| \beta \sum_{n=N+1}^{\infty} D_n(x,t) \right|.$$

Тригонометрические функции не принимают значения по модулю больше единицы, поэтому могут быть заменены на 1:

$$|\sin(n\pi)| < 1,$$
$$|\cos(n\pi)| < 1,$$
$$|\cos(\sqrt{\frac{n\pi^2}{16} + \pi^2 t})| < 1.$$

Тогда оценка остатка ряда приобретает следующий вид:

$$|R_n| \leq \sum_{n=N+1}^{\infty} \left| 8\frac{n\pi + 4}{n^3\pi^3} \right| \leq \frac{8}{\pi^2} \int_N^{\infty} \frac{dk}{k^2} + \frac{32}{\pi^3} \int_N^{\infty} \frac{dk}{k^3} \leq \left| 8\frac{\pi N + 16}{\pi^3 N^2} \right| = \Phi(N). \tag{21}$$

Для контроля точности принимаем $\Phi(N) \leq E$.

Отсюда получаем формулу для расчета числа суммирований ряда для достижения заданной точности:

$$N = \frac{4(\sqrt{\pi E + 1} + 1)}{\pi^2 E}. \tag{22}$$

5 Экспериментальное исследование качества оценки остатка ряда

Проведём экспериментальное исследование качества оценки остатка ряда следующим образом:

1. Будем менять требуемую точность вычислений от 10^{-1} до 10^{-8}.

2. Для каждой требуемой точности рассчитаем $N_{\text{избыточное}}$ с помощью формулы (22).

3. Рассчитаем $N_{\text{достаточное}}$, уменьшая количество слагаемых в сумме, начиная с $N_{\text{избыточное}}$ до тех пор, пока результат сохраняет требуемую точность.

4. Шаги 1-3 проделаем для моментов времени $t = 0$, $t = 5$ и $t = 10000$. Это необходимо для того, чтобы проследить за изменением $N_{\text{достаточное}}$ на разных временных промежутках.

Таблица 1 – Значения $N_{\text{избыточное}}$ и $N_{\text{достаточное}}$ при разных ϵ и t.

ϵ	$t = 0$		$t = 5$		$t = 10000$	
	$N_{\text{избыточное}}$	$N_{\text{достаточное}}$	$N_{\text{избыточное}}$	$N_{\text{достаточное}}$	$N_{\text{избыточное}}$	$N_{\text{достаточное}}$
10^{-1}	2	0	2	0	2	0
10^{-2}	8	0	8	0	8	2
10^{-3}	25	0	25	2	25	3
10^{-4}	78	0	78	9	78	5
10^{-5}	246	0	246	25	246	8
10^{-6}	778	0	778	65	778	15
10^{-7}	2462	0	2462	152	2462	25
10^{-8}	7784	0	7784	310	7784	38

Как видно из таблицы 1 в моменты времени t близкие к 0, достаточно небольшого количества слагаемых. Причиной этого является тот факт, что в начальный момент времени концентрация в трубке нулевая, а следовательно достаточно нуля слагаемых для описания системы.

Разница между моментами времени $t = 5$ и $t = 10000$ объясняется тем, что в итоговой формуле (**??**) весь ряд умножается на $e^{g_{2n+1}t}$, где g_{2n+1} - отрицательная величина. Следовательно, в моменты времени $t \gg 0$ слагаемые убывают с большей скоростью, чем в моменты

времени более близкие к моменту $t = 0$. Это означает что при больших t достаточно меньшего количества слагаемых, так как каждый следующий член ряда убывает всё быстрее и быстрее.

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”.

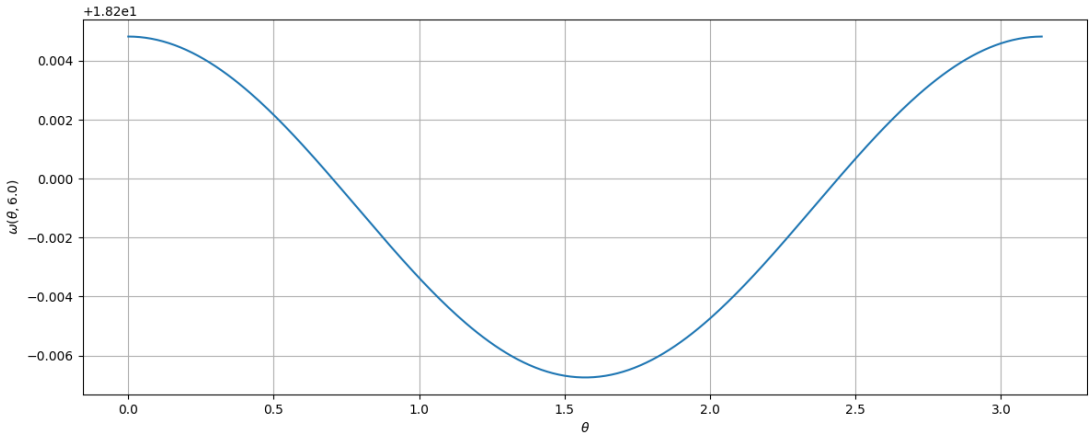
Для варьирования параметров, выданных преподавателем, следует передавать их как ”флаги” программы. Например, для изменения параметра ”a”(α) следует написать ”python umf.py pic –a=0.4”, теперь при расчетах коэффициент теплопроводности α будет равен $0.4\left[\frac{\text{Вт}}{\text{см}^2 \cdot \text{К}}\right]$.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

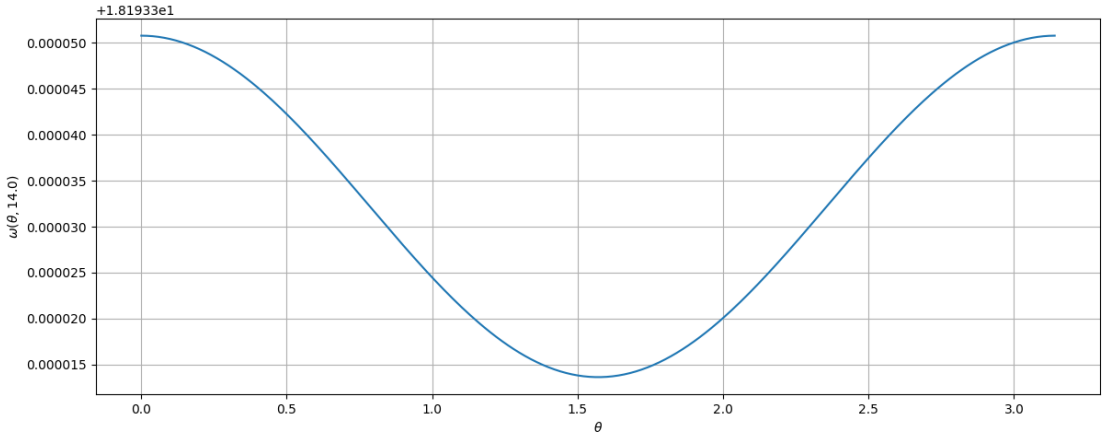
Для вызова подсказок следует дописать ”– –help” в конец команды. Система подсказок работает для любой команды. Все подсказки написаны на английском языке, для корректного встраивания текста программы в файл отчета.

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.



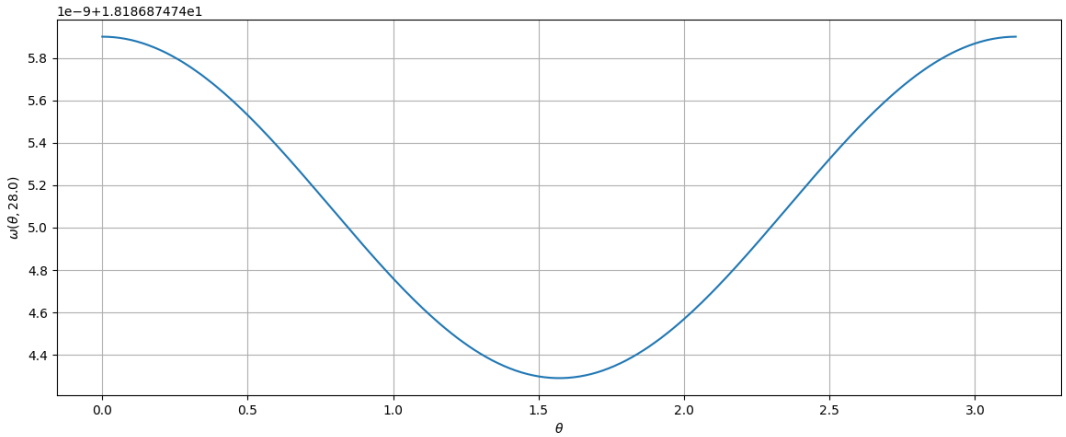
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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



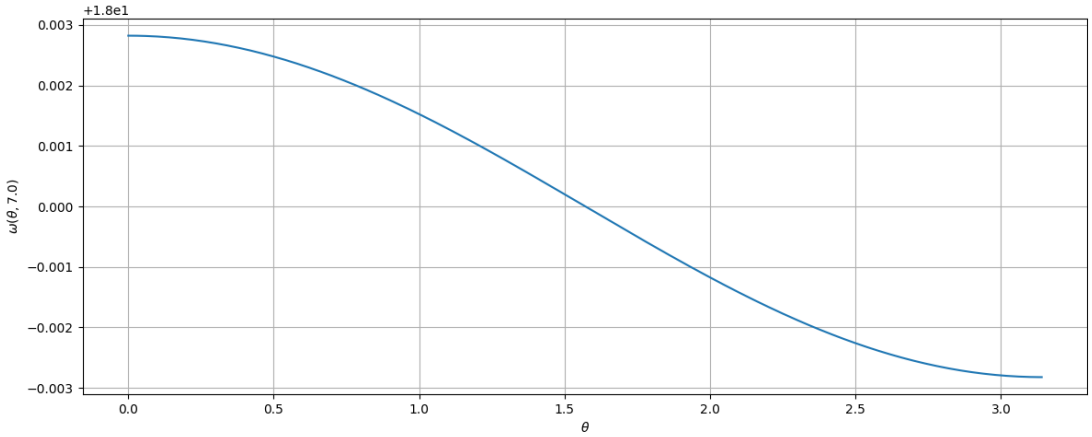
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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



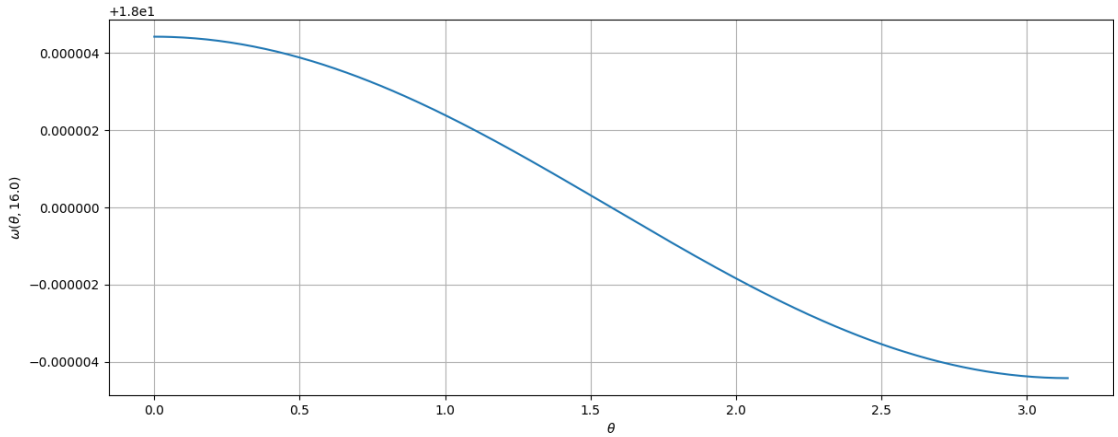
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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



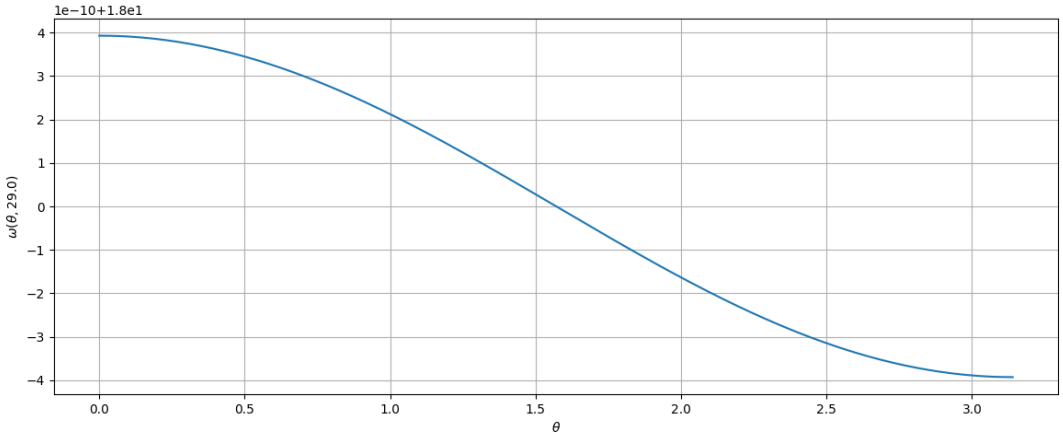
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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

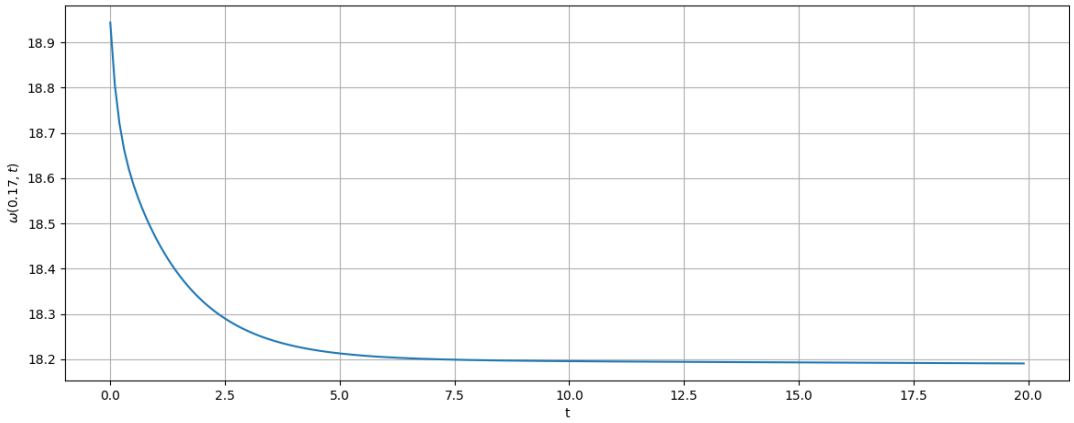
Рисунок 5 – График температуры от времени t=16c при $\psi_2(\theta)$



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

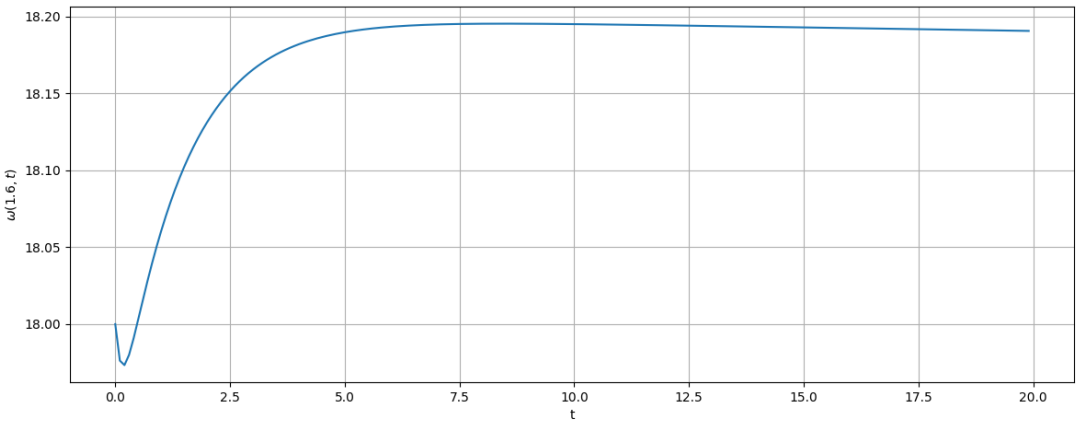
Рисунок 6 – График температуры от времени t=29c при $\psi_2(\theta)$

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.



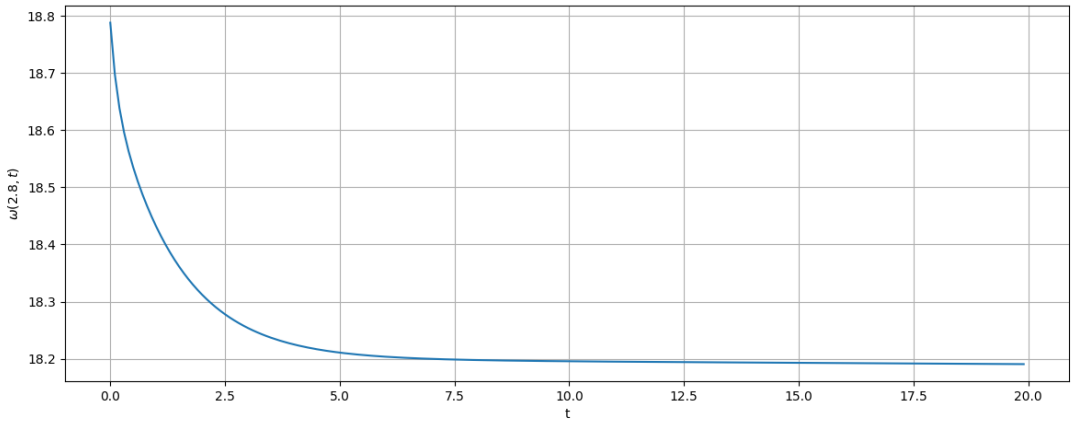
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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



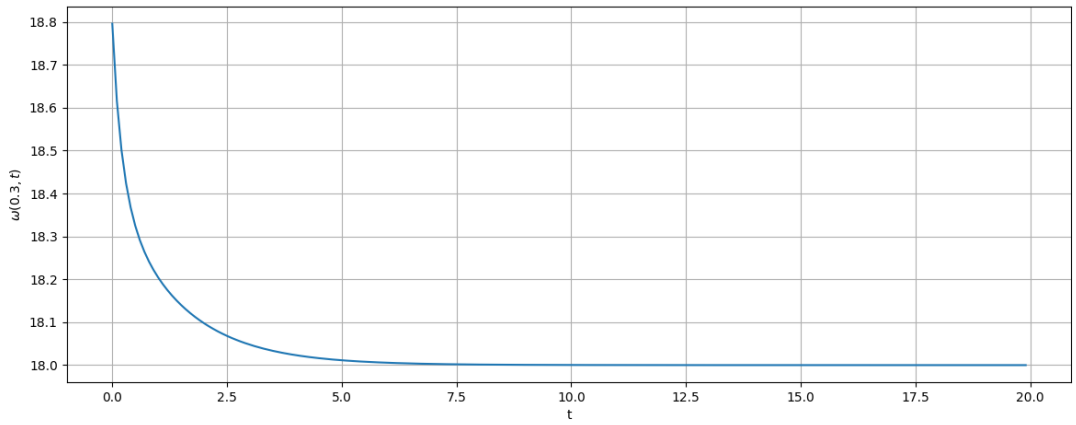
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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



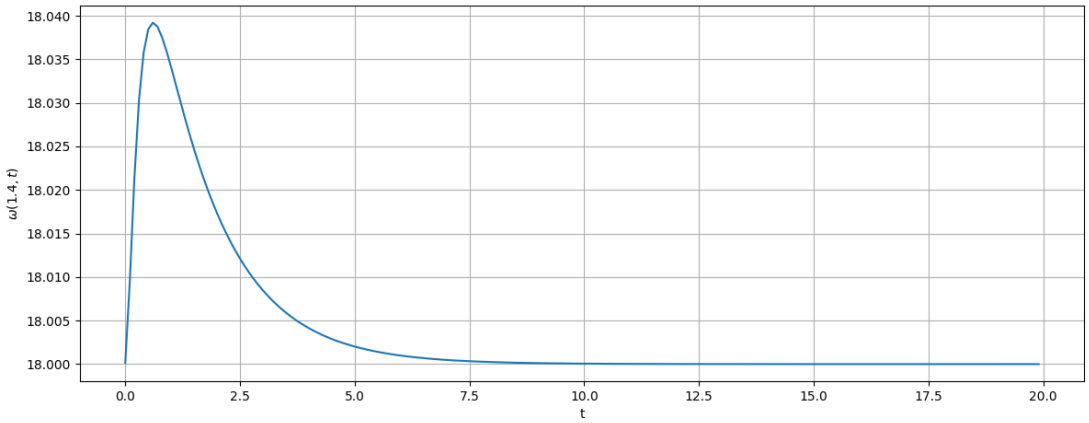
Chosen Parameters:
alpha = 0.002
c = 1.65
k = 0.59
l = 0.5

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 $\psi_2(\theta)$

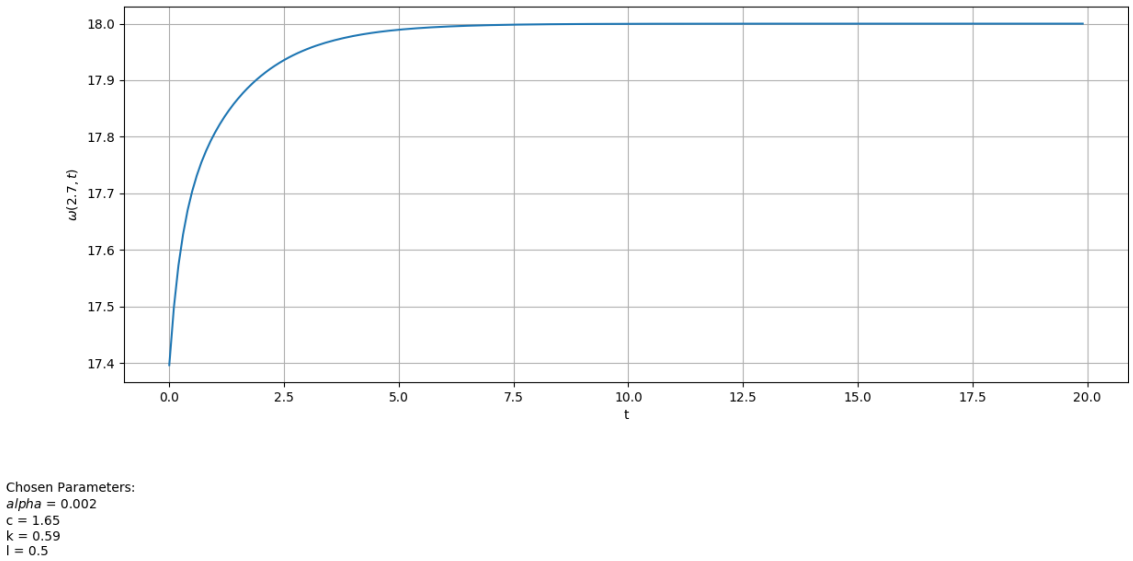


Рисунок 12 – График температуры от времени $\theta = 2.7$ при $\psi_2(\theta)$

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 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)
```