Национальный исследовательский университет «МЭИ»
Институт информационных и вычислительных технологий
Кафедра ПМ ИИ

Отчет по курсовой работе

по дисциплине: «Численные методы»

Тема курсовой работы

«Моделирование передачи тепла в пластине в зависимости от граничных условий»

Группа: А-05-22

Студенты:

Павлов Н. М.

Володченков Н. Д.

Руководитель:

Амосова О. А.

Москва 2024

Оглавление

Раздел 1. Задача теплопроводности **3**

1.1. Постановка задачи 3

1.2. Общие выкладки по поиску аналитического решения 3

1.3. Первая попытка поиска начальных условий 4

1.4. Вторая попытка поиска начальных условий 5

1.5. Третья попытка поиска начальных условий 5

1.6. Четвертая попытка поиска начальных условий 6

1.7. Итог раздела 6

Раздел 2. Разностные схемы **7**

Раздел 3. Явная разностная схема **8**

3.1. Идея решения 8

3.2. Особенности применения и точность решения 8

Раздел 4. Неявная разностная схема **10**

4.1. Идея решения 10

4.2. Точность метода 11

Раздел 5. Моделирование передачи тепла в пластине в зависимости от начальных и граничных условий **12**

5.1. Первая модель 12

5.2. Вторая модель 13

5.3. Третья модель 13

5.4. Четвертая модель 16

5.5. Пятая модель 16

Раздел 6. Заключение **18**

Приложение **19**

Приложение 1. Код явного метода 19

Приложение 2. Код неявного метода 21

Приложение 3. Ссылки на анимации 25

Литература **26**

Раздел 1. Задача теплопередачи

1.1. Постановка задачи

В ходе курсовой работы будем моделировать теплопередачу внутри квадратной пластины. Для этого требуется составить математическую модель задачи. Положим пластину в декартову систему координат так, чтобы углы пластины соответствовали координатам $(0,0)$, $(0,1)$, $(1,0)$, $(1,1)$. Измерение температуры будем проводить в отрезок времени $[0,T]$.

Нетрудно заметить, что в таком случае у нас получается функция нагрева $u(x,y,t)$, определенная на D. Также обозначим границу пластины как Γ.

$D = \{(x,y,t) \in \mathbb{R}^3 : x \in [0,1],\ y \in [0,1],\ t \in [0,T]\}$

$\Gamma = \{(x,y,t) \in \mathbb{R}^3 : x = 0 \vee y = 0 \vee x = 1 \vee y = 1\}$

Обращаясь к результатам физики, имеем следующее дифференциальное уравнение, описывающее поставленную задачу:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Которое для простоты обозначений перепишем в следующем виде:

$u_t = u_{xx} + u_{yy}$

Но для однозначного решения этого уравнения не хватает начальных и граничных условий, определим их следующим образом:

$u\big|_{t=0} = \phi(x,y)$ - начальное распределение тепла в пластине

$u\big|_{\Gamma} = g(x,y,t)$ - распределение тепла на краях пластины

Поиск общего аналитического решения, зависящего от обоих условий, для такой задачи достаточно сложен, поэтому будем применять численные методы, а именно разностные схемы, которые рассмотрим далее. Далее, для оценки работы численных методов, найдем решение этой задачи при определенных начальных условиях.

1.2. Общие выкладки по поиску аналитического решения

Решение будем искать в виде $u(x,y,t) = X(x) \cdot Y(y) \cdot T(t)$, отсюда

$u_t = X(x) \cdot Y(y) \cdot T'(t)$

$u_{xx} = X''(x) \cdot Y(y) \cdot T(t)$

$u_{yy} = X(x) \cdot Y''(y) \cdot T(t)$

Тогда уравнение примет вид:

$X(x) \cdot Y(y) \cdot T'(t) = X''(x) \cdot Y(y) \cdot T(t) + X(x) \cdot Y''(y) \cdot T(t)$

Поделив обе части уравнения на $X(x) \cdot Y(y) \cdot T(t)$ получаем:

$$\frac{T'(t)}{T(t)} = \frac{X''(x)}{X(x)} + \frac{Y''(y)}{Y(y)}$$

Заметим, что каждое из слагаемых зависит только от одной из переменных, поэтому получаем, что

$\frac{T'(t)}{T(t)} = -\lambda$, $\frac{X''(x)}{X(x)} = -\lambda_1$, $\frac{Y''(y)}{Y(y)} = -\lambda_2$, где $\lambda, \lambda_1, \lambda_2$ - некоторые константы.

Избавимся от знаменателей в каждом из этих уравнений и получим следующие задачи Штурма-Лиувилля:

$$\begin{cases} T'(t) = -\lambda T(t) & (1) \\ X''(x) = -\lambda_1 X(x) & (2) \\ Y''(y) = -\lambda_2 Y(y) & (3) \end{cases}$$

Отдельно решим каждую из них, для начала задачу (1). Если не будем задавать никаких дополнительных условий, то получим решение

$T(t) = C^{(1)} e^{-\lambda t}$

Пока что оставим его в таком виде и перейдем к задаче (2) (ну и аналогичные выводы сможем сделать и для задачи (3)):

$X''(x) = -\lambda_1 X(x)$

И тут уже от параметра λ зависит то как будет выглядеть решение. Для простоты будем брать $\lambda \in \mathbb{R}$.

1.3. Первая попытка поиска начальных условий (неудачная)

Решение получим, если возьмём $\lambda_i = 0$. Тогда общее решение будет очень простое:

$X(x) = C_1^{(2)} x + C_0^{(2)}$

$Y(y) = C_1^{(3)} y + C_0^{(3)}$

А решение для функции тепла будет выглядеть следующим образом:

$$u(x,y,t) = C^{(1)}e^0 \cdot (C_1^{(2)}x + C_0^{(2)}) \cdot (C_1^{(3)}y + C_0^{(3)}) = Axy + Bx + Cy + D$$

По смыслу задачи (тепло никак не уравновешивается) такое решение не подходит.

1.4. Вторая попытка поиска начальных условий (неудачная)

Теперь возьмем $\lambda_i < 0$, в таком случае решение запишется как

$$X(x) = C_1^{(2)}e^{\sqrt{-\lambda_1}x} + C_2^{(2)}e^{-\sqrt{-\lambda_1}x}$$

$$Y(y) = C_1^{(3)}e^{\sqrt{-\lambda_2}y} + C_2^{(3)}e^{-\sqrt{-\lambda_2}y}$$

$$u(x,y,t) = C^{(1)}e^{-\lambda t} \cdot X(x) \cdot Y(y)$$

Так как $\lambda = \lambda_1 + \lambda_2 < 0$, то при увеличении времени будет возрастать и нагрев, что также не подходит по смыслу задачи.

1.5. Третья попытка поиска начальных условий (удачная)

Будем считать $\lambda_i > 0$, тогда получаем решение

$$X(x) = C_1\cos(\sqrt{\lambda_1}x) + C_2\sin(\sqrt{\lambda_1}x)$$

Также будем считать $g(x,y,t) = 0$, что означает:

- $X(0) = 0$, тогда $C_1 = 0$
- $X(1) = 0$, тогда $\sin(\sqrt{\lambda_1}) = 0 \implies \lambda_1 = \pi^2 n^2$

И решение примет вид

$$X(x) = C^{(2)}\sin(\pi n x)$$

Аналогично

$$Y(y) = C^{(3)}\sin(\pi m y)$$

Таким образом,

$$u(x,y,t) = Ae^{-(m^2+n^2)\pi^2 t}\sin(\pi n x)\sin(\pi m y)$$

Вообще говоря, это целое семейство функций, линейная комбинация которых также будет решением, но если возьмем начальное и граничное условие как

$$\phi(x,y) = \sin(\pi x)\sin(\pi y)$$

$$g(x,y,t) = 0$$

то решением такой задачи будет

$u(x, y, t) = e^{-2\pi^2 t} \sin(\pi x) \sin(\pi y).$

1.6. Четвертая попытка поиска начальных условий (неудачная)

Заметим, что вторая попытка была откинута только потому, что итоговое $\lambda = \lambda_1 + \lambda_2$ оказалось отрицательным, благодаря чему можно было с легкостью утвердить, что такое решение не соответствует смыслу задачи. Но что, если $\lambda = \lambda_1 + \lambda_2$ будет с одной стороны положительным, но с другой стороны одно из слагаемых λ_i окажется отрицательным?

Не ограничивая общности, примем, что $\lambda_1 < 0$, $\lambda_2 > 0$, $|\lambda_1| < |\lambda_1|$, тогда решения примут вид:

$X(x) = C_1^{(2)} e^{\sqrt{-\lambda_1} x} + C_2^{(2)} e^{-\sqrt{-\lambda_1} x}$

$Y(y) = C_1^{(3)} \cos(\sqrt{\lambda_2} y) + C_2^{(3)} \sin(\sqrt{\lambda_2} y)$

Наложим сюда то же краевое условие, что и в пункте 5, тогда, как мы уже знаем

$Y(y) = C^{(3)} \sin(\pi m y)$

А для $X(x)$ имеем:

$$\begin{cases} C_1^{(2)} + C_2^{(2)} = 0 \\ C_1^{(2)} e^{\sqrt{-\lambda_1}} + C_2^{(2)} e^{-\sqrt{-\lambda_1}} = 0 \end{cases}$$

Выразим из первого уравнения $C_2^{(2)} = -C_1^{(2)}$ и подставим во второе, получим:

$C_1^{(2)} \cdot \left(e^{\sqrt{-\lambda_1}} - e^{-\sqrt{-\lambda_1}} \right) = 0$

Так как приняли $\lambda_1 < 0$, то второй множитель никогда не обращается в 0, значит $C_1^{(2)} = 0$, но тогда в силу первого уравнения $C_2^{(2)} = 0$, следовательно $X(x) = 0$, $u = 0$. Этот случай тривиален, а значит рассматривать его не будем.

1.7. Итог раздела

Если зададим граничное и начальное условия как

$\phi(x, y) = \sin(\pi x) \sin(\pi y)$

$g(x, y, t) = 0$

То аналитическим решением будет

$u(x, y, t) = e^{-2\pi^2 t} \sin(\pi x) \sin(\pi y).$

Раздел 2. Разностные схемы

Решение задачи теплопроводности будем искать в виде сеточной функции, с чем нам помогут разностные схемы. Для этого разобьем исходное пространство D на прямоугольную сетку точек (x_i, y_j, t_k), где $i = 0, \dots, n$; $j = 0, \dots, n$; $k = 0, \dots, m$

$x_i = ih$, $y_j = jh$, $t_k = k\tau$

Обозначим

$U^k_{i,j} = u(x_i, y_j, t_k)$

Заметим, что благодаря краевым условиям при решении задачи будем знать

$U^k_{i,0}, U^k_{i,n}, U^k_{j,0}, U^k_{j,n}, \ \forall i, j, k$

А благодаря начальному условию известно

$U^0_{i,j}, \ \forall i, j$

Далее будем заменять производные по формулам, их приближающим:

$f'(x) \approx \dfrac{f(x+h) - f(x)}{h}$ - правая разностная производная

$f'(x) \approx \dfrac{f(x) - f(x-h)}{h}$ - левая разностная производная

В обоих случаях предполагается $h > 0$.

Тогда вторую производную можно приблизить следующим образом:

$$f''(x) \approx \frac{f'(x) - f'(x-h)}{h} \approx \frac{\dfrac{f(x+h) - f(x)}{h} - \dfrac{f(x) - f(x-h)}{h}}{h} = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Оттуда сможем выражать значения на последующем слое U^{k+1} через значения на предыдущем(-их) U^k.

Раздел 3. Явная разностная схема

3.1. Идея решения

Заменим u_t, u_{xx}, u_{yy} на приближенные производные в точках (x_i, y_j, t_k):

$$\frac{U_{i,j}^{k+1} - U_{i,j}^k}{\tau} = \frac{U_{i+1,j}^k - 2U_{i,j}^k + U_{i-1,j}^k}{h^2} + \frac{U_{i,j+1}^k - 2U_{i,j}^k + U_{i,j-1}^k}{h^2}$$

Отсюда сразу выражается значения температуры в следующий момент времени:

$$U_{i,j}^{k+1} = \frac{\tau}{h^2}\left(U_{i-1,j}^k + U_{i+1,j}^k + U_{i,j-1}^k + U_{i,j+1}^k - 4U_{i,j}^k\right) + U_{i,j}^k$$

$\forall i, j \in \{1, 2, \ldots, n-1\},\ k \in \{0, 1, \ldots, m-1\}$

Как видно, эта формула справедлива только для внутренних точек пластины, а для значений на краях, как было отмечено в разделе 2, используем краевые условия.

В приложении 1 приведен код, реализующий этот метод.

3.2. Особенности применения и точность решения при их соблюдении

При решение сразу захотелось решать поставленную уравнение на большом промежутке времени и при большом разрешении пластины, всё конечно же упирается в вычислительные мощности, поэтому пространство времени было решено всегда разбивать на 100 интервалов, так как именно количество интервалов времени соотвествует количеству сгенерированных кадров анимации.

Затем выбрали $T = 0.1$, а пластину разбили на 100 кусочков.

Будем сразу брать начальные и краевые условия из пункта 5 раздела 1.

И тогда получаем анимацию, которая на первых кадрах более менее похожа на теплопередачу в пластине, но затем достаточно быстро вся пластина заполняется чередующимися очень горячими, или очень холодными точками - следовательно, нарушена устойсчивость, анимацию можно найти по ссылке (если кликабельность будет потеряна, то ссылки можно найти в приложении):

Нестабильный явный метод

Опытным путем обнаружили, что отношение $\frac{\tau}{h^2} \approx 1$ чтобы задача была устойчива, следовательно, чтобы решить задачу хотя бы для размера пластины 50 на 50, требуется уменьшить время до $T = 0.01$, и тогда получаем следующую анимацию:

Стабильный явный метод

Погрешность в таком случае никогда не превышает 0.000104, её график можно посмотреть по той же ссылке.

Также из теории известно, что эта схема имеет 2-й порядок точности для аппроксимации производных про пространству $\frac{\partial}{\partial x}$ и $\frac{\partial}{\partial y}$ (в нашем случае $h = 0.01 \implies R \approx C \cdot 10^{-4}$), и 1-й порядок точности для аппроксимации производных по времени $\frac{\partial}{\partial t}$ (в нашем случае $\tau = 10^{-4} \implies R \approx C \cdot 10^{-4}$). Как видим, полученный результат удовлетворяет теории.

Раздел 4. Неявная разностная схема

4.1. Идея решения

Также будем заменять u_t на приближенную производную в точке (x_i, y_j, t_k), но u_{xx} и u_{yy} заменим на разностную формулу в точке (x_i, y_j, t_{k+1}) - получим неявную схему

$$\frac{U_{i,j}^{k+1} - U_{i,j}^k}{\tau} = \frac{U_{i+1,j}^{k+1} - 2U_{i,j}^{k+1} + U_{i-1,j}^{k+1}}{h^2} + \frac{U_{i,j+1}^{k+1} - 2U_{i,j}^{k+1} + U_{i,j-1}^{k+1}}{h^2}$$

Немного преобразовав это уравнение, получаем

$$-\tau U_{i,j-1}^{k+1} - \tau U_{i-1,j}^{k+1} + (h^2 + 4\tau)U_{i,j}^{k+1} - \tau U_{i+1,j}^{k+1} - \tau U_{i,j+1}^{k+1} = h^2 U_{i,j}^k$$

Добавив сюда краевые условия получаем систему n^2 уравнений из n^2 переменных:

$$\begin{cases} U_{0,0}^{k+1} = g(0, 0, t_{k+1}) \\ U_{1,0}^{k+1} = g(x_1, 0, t_{k+1}) \\ \dots \\ U_{n,0}^{k+1} = g(1, 0, t_{k+1}) \\ U_{0,1}^{k+1} = g(0, y_1, t_{k+1}) \\ -\tau U_{1,0}^{k+1} - \tau U_{0,1}^{k+1} + (h^2 + 4\tau)U_{1,1}^{k+1} - \tau U_{2,1}^{k+1} - \tau U_{1,2}^{k+1} = h^2 U_{1,1}^k \\ -\tau U_{2,0}^{k+1} - \tau U_{1,1}^{k+1} + (h^2 + 4\tau)U_{2,1}^{k+1} - \tau U_{3,1}^{k+1} - \tau U_{2,2}^{j+1} = h^2 U_{2,1}^k \\ \dots \end{cases}$$

Переименовав переменные для решения этой системы как $U_{i,j}^{k+1} = x_{(n+1)j+i}$ можем переписать эту систему как $Ax = b$ для:

> $$n = 2 \implies \dim(A) = 9$$
>
> $$A = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -\tau & 0 & -\tau & h^2 + 4\tau & -\tau & 0 & -\tau & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}; \; b = \begin{pmatrix} g(x_0, y_0, t_{k+1}) \\ g(x_1, y_0, t_{k+1}) \\ g(x_2, y_0, t_{k+1}) \\ g(x_0, y_1, t_{k+1}) \\ h^2 U_{1,1}^k \\ g(x_2, y_1, t_{k+1}) \\ g(x_0, y_2, t_{k+1}) \\ g(x_1, y_2, t_{k+1}) \\ g(x_2, y_2, t_{k+1}) \end{pmatrix}$$

> $$n = 3 \implies \dim(A) = 16$$
>
> Эту матрицу опишем только по строкам, тогда понятно, что строкам, которые соответствуют граничным условиям, то есть строки с номерами $(n+1)j + i$, когда $i = 0 \vee i = n \vee j = 0 \vee j = n$, будут иметь вид
>
> $$(0 \quad \dots \quad 0 \quad 1 \quad 0 \quad \dots \quad 0)$$

где единица стоит прямо на главной диагонали.

А остальные строки будут иметь вид

$$(0 \quad \dots \quad 0 \quad -\tau \quad 0 \quad 0 \quad -\tau \quad h^2+4\tau \quad -\tau \quad 0 \quad 0 \quad -\tau \quad 0 \quad \dots \quad 0)$$

где на главной диагонали стоит $h^2 + 4\tau$

Вектор $b = (b_\xi)$, где $b_\xi = g(\dots)$ на границах, $b_\xi = U^k$ во внутренних точках.

$$n = 4 \implies \dim(A) = 25$$

Аналогично, только количество нулевых элементов между $(-\tau)$ для строк, соответствующих внутренним точкам пластины, станет равно 3 и будет увеличиваться на 1 по мере увеличения n.

Нетрудно заметить, что такая система обладает диагональным преобладанием, так что её можно решать методом Зейделя, но в коде будем использовать функцию `spsolve`, который достаточно быстро решает такие системы даже для размера 40000 (которая соответствует $n = 200$).

4.2. Точность метода

Из теории известно, что неявный метод является абсолютно устойчивым, то есть с нашей задачей проблем возникать не должно, но так как мы ограничены в вычислительных мощностях (памятью для хранения матриц, производительностью процессора для вычислений), решили использовать параметры $T = 0.1,\ \tau = 10^{-3},\ h = 5 \cdot 10^{-3}$ (что соответствует разбиению пластины на 200 точек)

Эти параметры позволяют просмотреть теплопередачу в пластине вплоть до теплового равновесия (если граничные условия постоянны) при очень хорошем разрешение за сравнительно недолгое время вычисления. Анимацию теплопередачи для условий из пункта 5 раздела 1 можно посмотреть по ссылке:

Неявный метод

Получаем погрешность, никогда не превышающую 0.0036, которая, кстати говоря, достигает максимума на слое $k = 51$, что можно примерно наблюдать на анимации этой погрешности по той же ссылке.

Теоретически этот метод имеет те же порядки точности, что и неявный метод, получаем что $R_{x,y} \approx C \cdot 10^{-5}$, $R_t \approx C \cdot 10^{-3}$, выбирая наибольший получаем, что полученная нами точность совпадает с теорией.

Раздел 5. Моделирование передачи тепла в пластине в зависимости от начальных и граничных условий.

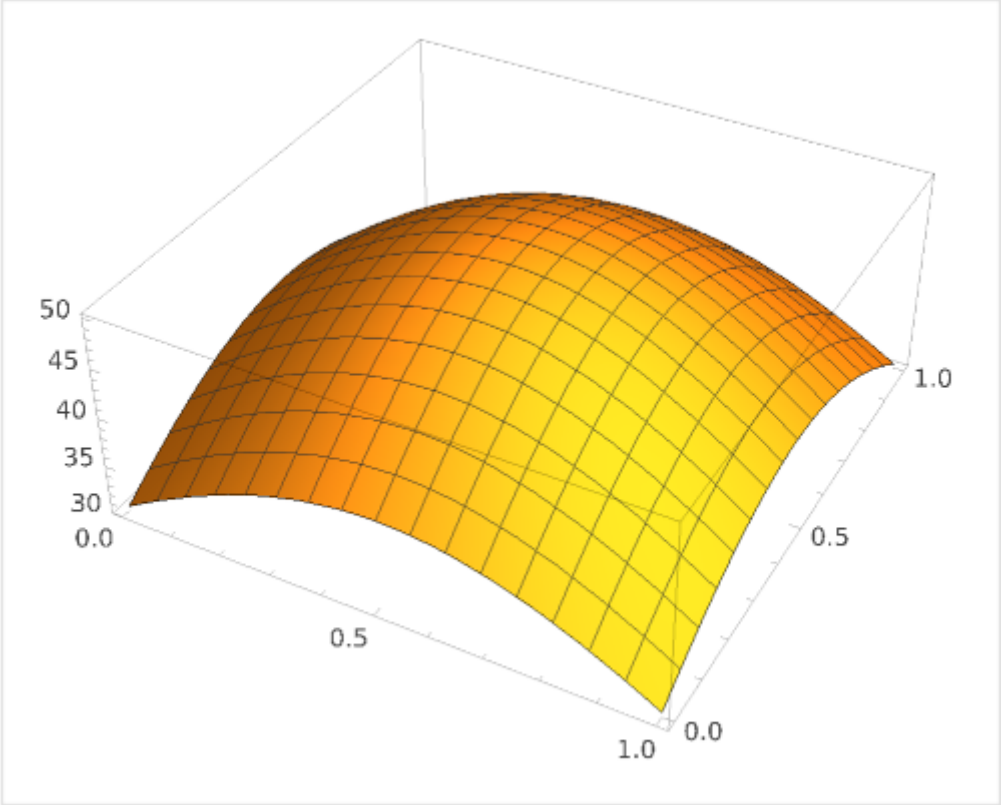
В этом разделе посмотрим на разные начальные и краевые условия, все вычисления были проведены при $T = 0.1$, $\tau = 10^{-3}$, $h = \dfrac{1}{150} \approx 6.7 \cdot 10^{-3}$.

Давайте смоделируем нагревание пластины при 5-ти различных условиях.

5.1. Первая модель

Начальным нагревом пластины будет

$$50 \exp\left\{-\left(x - \frac{1}{2}\right)^2 - \left(y - \frac{1}{2}\right)^2\right\}$$



по графику видно, что это нагретый центр с постепенным спадом температуры к краям.

По краям будем подавать температуру

$500t$

это постепенное возрастание нагрева краев.

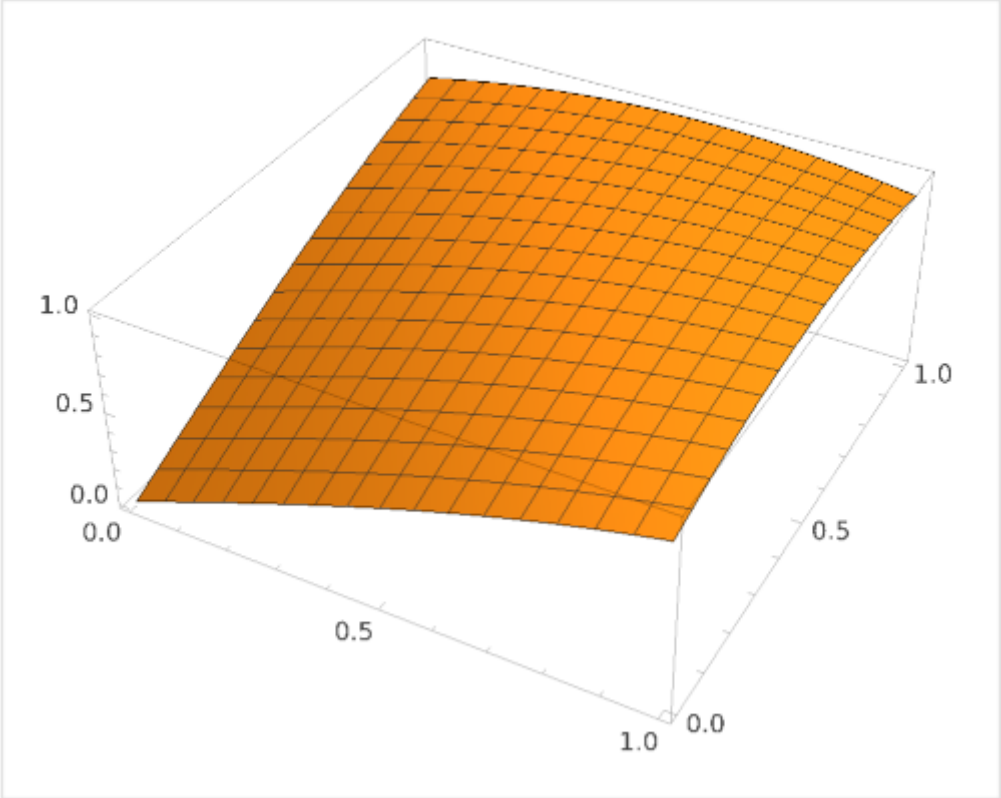
Получаем следующую анимацию:

ссылка

5.2. Вторая модель

Начальным нагревом пластины будет

$\sin(x+y)$



гладкое изменение температуры по диагонали (изменяется по синусоиде).

По краям зададим температуру

$\cos(x+y+t)$

которая в случае с $T = 0.1$ просто немного изменится (где-то увеличится, где-то уменьшится, где-то и уменьшится, и увеличится) на краях.

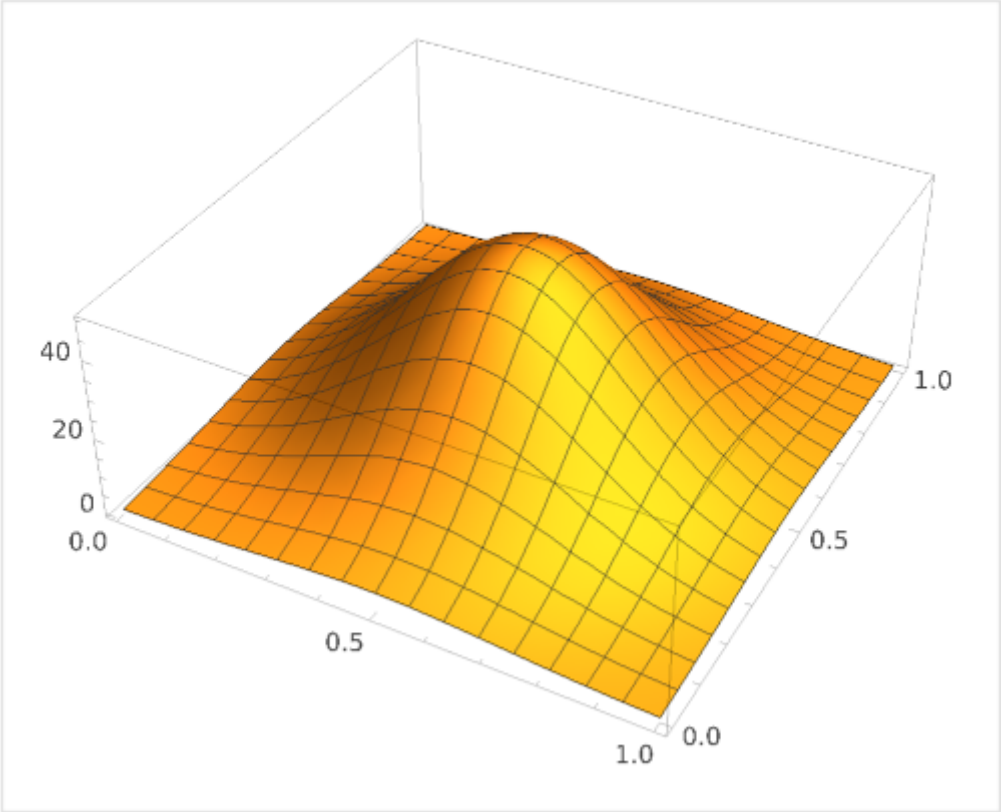
Вот что как выглядит анимация этого процесса:

ссылка

5.3. Третья модель

Начальным нагревом пластины будет

$$50 \exp\left\{10 \cdot \left(-\left(x-\frac{1}{2}\right)^2 - \left(y-\frac{1}{2}\right)^2\right)\right\}$$



это то же самое, что и в первой модели, просто центр теперь нагрет намного сильнее краев.

На границах подадим температуру

$$50\sin(100t)$$

которая благодаря коэффициенту 100 под синусом в процессе наблюдения $T = 0.1$ много раз изменит свою температуру от -50 до $+50$.

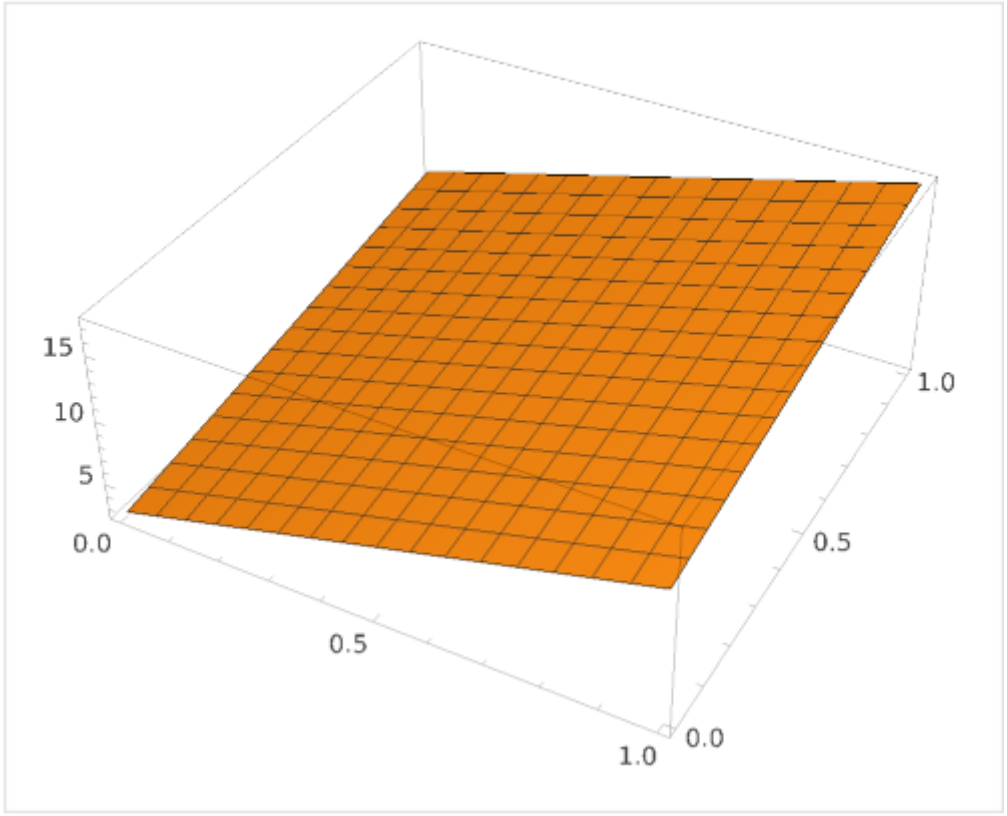
Вот какая анимация получается:

ссылка

5.4. Четвертая модель

Начальным нагревом пластины будет

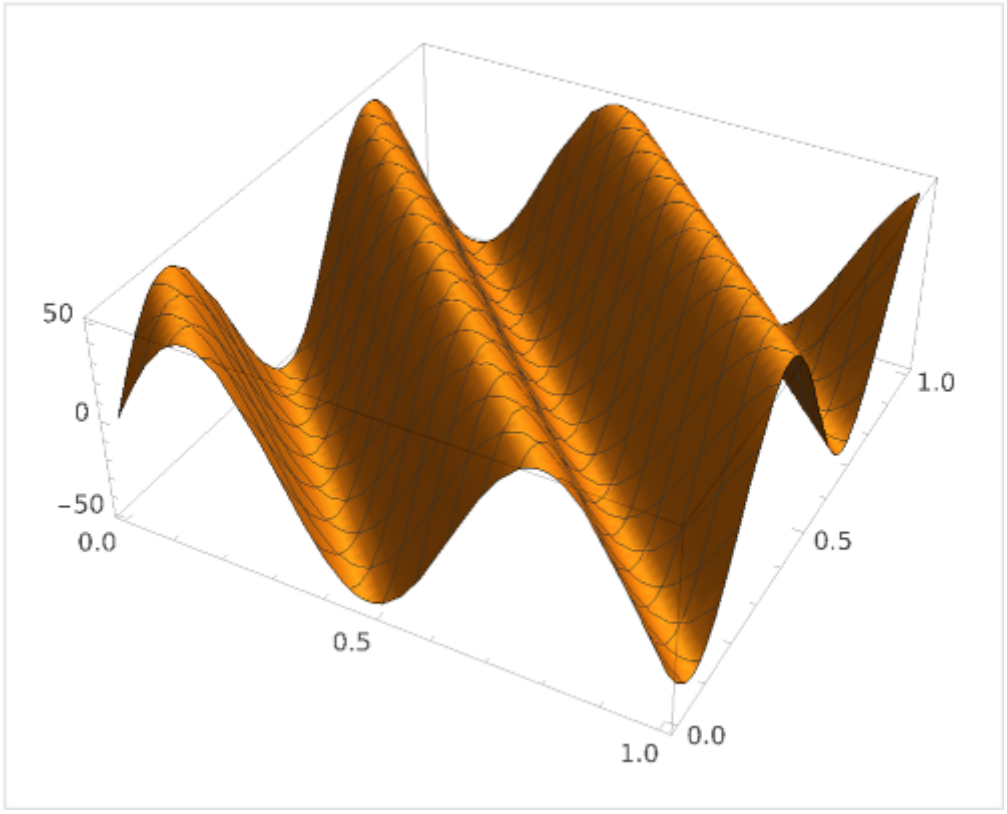
$$10x + 5y + 2.5$$



это просто линейное изменение температуры вдоль пластины.

На границах возьмем функцию

$50 \sin(10x + 10y)$



которая создает на границах много горячих и холодных точек.

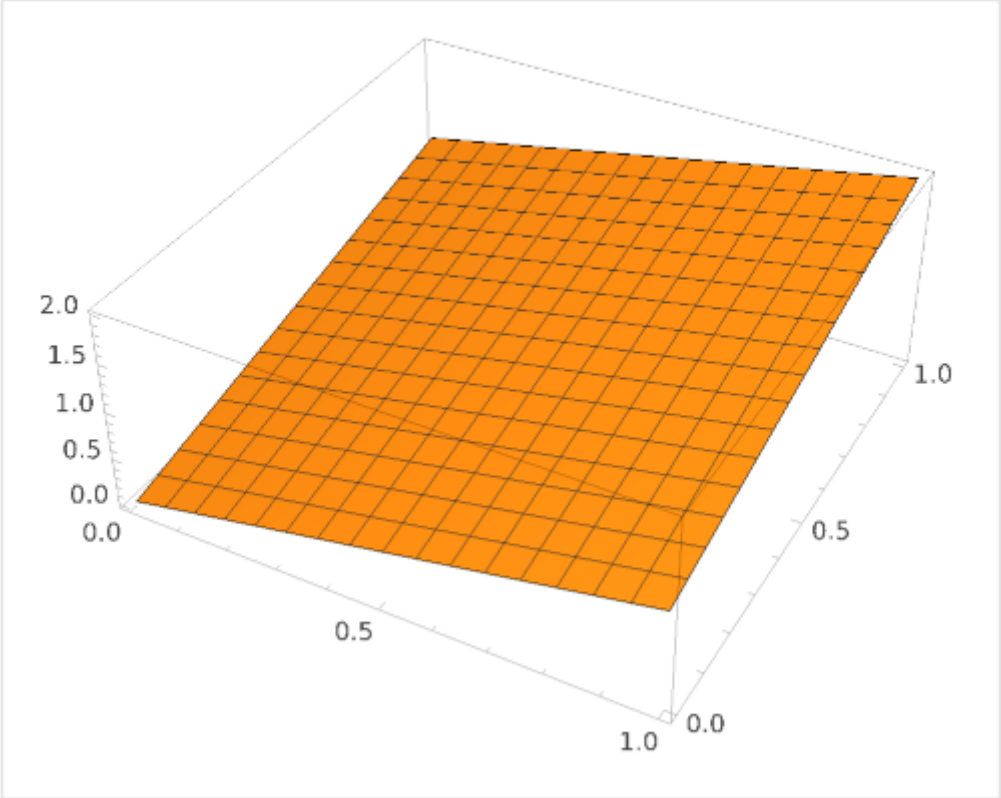
Полученную анимацию можно увидеть по ссылке:

ссылка

5.5. Пятая модель

Начальным нагревом возьмем

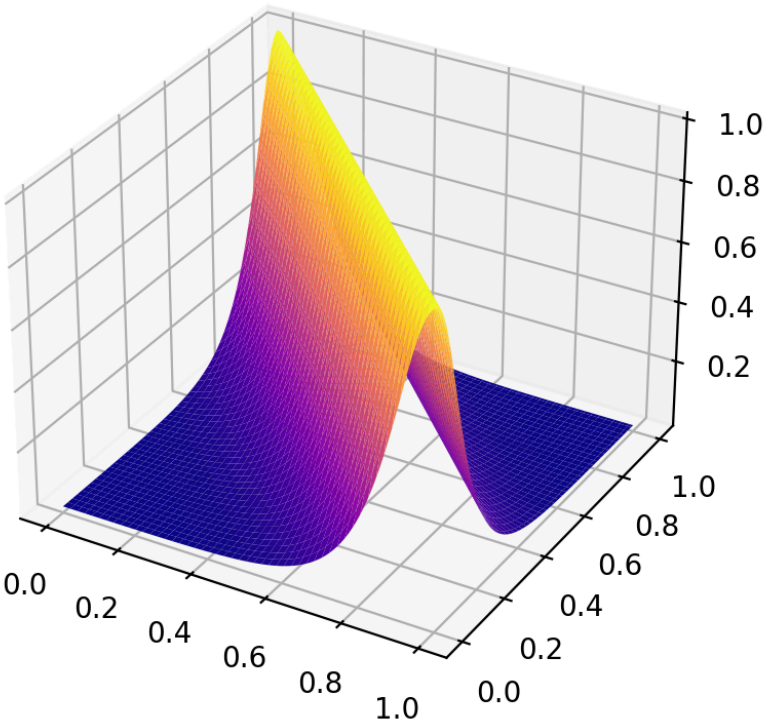
$x + y$



более гладкое линейное распределение температуры в сравнении с четвертой моделью.

На границах возьмем функцию

$\operatorname{csch}(10x + 10y - 10)$ — гиперболический косеканс



который дает более экстремальные (по перепаду температур) граничные условия.

Полученную анимацию можно увидеть по ссылке:

ссылка

Раздел 6. Заключение.

В курсовой работе рассмотрена модель передачи тепла в прямоугольной пластине, которая описывается уравнением теплопроводности.

Для численного решения были построены две разностные схемы:

1. **Явная схема**:
   - Простая в реализации, но требует ограничения на шаг по времени τ и по пространству h для обеспечения устойчивости.
2. **Неявная схема**:
   - Не требует жестких ограничений на шаг, устойчива при любом τ, h, но требует решения системы линейных алгебраических уравнений на каждом временном шаге.

Проделанное исследование:

1. Проведена дискретизация задачи с использованием центральных разностей по пространственным переменным и левых для временной переменной.
2. Осуществлено программное моделирование в среде Python с использованием библиотек `numpy` и `matplotlib`, включая реализацию алгоритмов для обеих схем.
3. Проведены вычислительные эксперименты для различных сеточных параметров h, τ, что позволило исследовать влияние сеточной дискретизации на точность и устойчивость метода.
4. Построены тепловые карты распределения температуры в пластине для различных временных шагов. Отчетливо видно, как начальное распределение температуры трансформируется под действием граничных условий и внутренних процессов теплопередачи.

Приложение

Приложение 1. Код явного метода

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
from scipy.sparse.linalg import spsolve
from scipy.sparse import csr_matrix
import time
from matplotlib.animation import FuncAnimation

try_number = 0
path = 'Forward/'

####!!!
####Очень важно сохранять отношение t к h как 1 к 100
####T.e. размер в 10000 больше чем конечнее время
####!!!

###Так как метод Абсоютно устойчив только в малекнькой окружности

T = 0.01
field_size = 50
time_duration = 100
t = T / time_duration
h = 1 / (field_size - 1)

u = np.zeros((field_size, field_size, time_duration))

def phi(x, y):
    return np.sin(np.pi * x) * np.sin(np.pi * y)

def g(x, y, t):
    return 0

def Solution():
    for k in range(0, int(T / t)):
        for i in range(field_size):
            for j in range(field_size):
                if k == 0:
                    u[i][j][k] = phi(i * h, j * h)
                elif i == 0 or i == field_size - 1 or j == 0 or j ==
field_size - 1:
                    u[i][j][k] = g(i * h, j * h, k * t)
                else:
                    u[i][j][k] = t * ((u[i - 1][j][k - 1] - 2 * u[i][j][k -
1] + u[i + 1][j][k - 1]) / h ** 2 + \
```

```
                                + (u[i][j - 1][k - 1] - 2 * u[i][j][k - 1] +
u[i][j + 1][k - 1]) / h ** 2) + u[i][j][k - 1]

Solution()

fig, ax = plt.subplots()
frames = time_duration - 3
interval = 100
heatmap = ax.imshow(u[:, :, 0], cmap='plasma', interpolation='nearest', vmin
= -1, vmax = 1)

def update(frame):
    heatmap.set_array(u[:, :, frame - 1])
    return [heatmap]

plt.colorbar(heatmap)
plt.axis('off')
ani = FuncAnimation(fig, update, frames=frames, interval=interval,
blit=True)
ani.save(path + f'Method animation{try_number}.gif',
writer=PillowWriter(fps=10))

u_solve = np.zeros_like(u)
R = np.zeros_like(u)

def solve(x, y, t):
    return np.exp(-2 * np.pi ** 2 * t) * np.sin(np.pi * x) * np.sin(np.pi *
y)

def create_u_solve_and_R():
    global max_R
    global time_R
    for k in range(time_duration):
        for i in range(field_size):
            for j in range(field_size):
                u_solve[i, j, k] = solve(i * h, j * h, k * t)
                R[i, j, k] = abs(u_solve[i, j, k] - u[i, j, k])
                if R[i, j, k] > max_R:
                    max_R = R[i, j, k]
                    time_R = k


max_R = 0.0
time_R = 0.0
create_u_solve_and_R()

fig, ax = plt.subplots()
frames = time_duration - 3
interval = 100
heatmap = ax.imshow(u[:, :, 0], cmap='plasma', interpolation='nearest', vmin
```

```
= -1, vmax = 1)

def update(frame):
    heatmap.set_array(u_solve[:, :, frame - 1])
    return [heatmap]

plt.colorbar(heatmap)
plt.axis('off')
ani = FuncAnimation(fig, update, frames=frames, interval=interval,
blit=True)
ani.save(path + f'solve animation{try_number}.gif',
writer=PillowWriter(fps=10))

print(f'Максимальная погрешность равна {max_R} на итерации {time_R}')

x = np.arange(field_size)
y = np.arange(field_size)
x, y = np.meshgrid(x, y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_zlim(0, 0.0001)

# Начальное значение поверхности
z = R[:, :, 0]
surf = ax.plot_surface(x, y, z, cmap='viridis')

def update(frame):
    ax.clear()
    ax.set_zlim(0, 0.0001)
    z = R[:, :, frame]
    surf = ax.plot_surface(x, y, z, cmap='viridis')
    return surf,

ani = FuncAnimation(fig, update, frames=frames, blit=False)
ani.save(path + f'Error animation{try_number}.gif', writer='imagemagick',
fps=15)
plt.show()
```

Приложение 2. Код неявного метода

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation, PillowWriter
from scipy.sparse.linalg import spsolve
from scipy.sparse import csr_matrix
import time
from matplotlib.animation import FuncAnimation
```

```
try_number = 5
path = 'Backward/'

T = 0.1
field_size = 200
time_duration = 100

t = T / time_duration
h = 1 / (field_size - 1)
u = np.zeros((field_size, field_size, time_duration))

def phi(x, y):
    # Method animation1 return np.sin(np.pi * x) * np.sin(np.pi * y)
    # test1 return 50 * np.exp(-(x - 1/2) ** 2 - (y - 1/2) ** 2)
    # test2  return np.sin(x + y)
    # test3 return 50 * np.exp((-(x - 1/4) ** 2 - (y - 1/3) ** 2) / 0.1)
    # test4 return 10 * x + 5 * y + 2.5
    # test5
    return x + y

def g(x, y, t):
    # Method animation1 return 0
    # test1 return 500 * t
    # test2 return np.cos(x + y + t
    # test3 return 50 * np.sin(100 * t)
    # test4 return 50 * np.sin(10 * x + 10 * y)
    # test5 return
    1 / np.cosh(10 * ((x - 1 / 2) + (y - 1 / 2)))

def generate_A(n, h, t):
    A = np.eye((n * n), dtype='float')
    for i in range(1, n - 1):
        for j in range(1, n - 1):
            row = n * j + i
            A[row][n * j + i] = h ** 2 + 4 * t
            A[row][n * j + (i - 1)] = -t
            A[row][n * j + (i + 1)] = -t
            A[row][n * (j - 1) + i] = -t
            A[row][n * (j + 1) + i] = -t
    return A


def generate_b(u, n, h, t, k):
    b = np.zeros((n, n))

    boundary_mask = np.zeros((n, n), dtype=bool)
    boundary_mask[0, :] = boundary_mask[-1, :] = True
    boundary_mask[:, 0] = boundary_mask[:, -1] = True
```

```
    inner_mask = ~boundary_mask
    i, j = np.where(boundary_mask)
    b[boundary_mask] = g(i * h, j * h, k * t)

    b[inner_mask] = h ** 2 * u[inner_mask][:, k]
    b = b.ravel()

    return b

A = csr_matrix(generate_A(field_size, h, t))

def Solution():
    for i in range(field_size):
        for j in range(field_size):
            if i == 0 or j == 0 or i == field_size - 1 or j == field_size -
1:
                u[i][j][0] = g(i * h, j * h, 0)
            else:
                u[i][j][0] = phi(i * h, j * h)

    for k in range(1, time_duration):
        b = generate_b(u, field_size, h, t, k - 1)
        y = spsolve(A, b)
        u[:, :, k] = y.reshape(field_size, field_size)

Solution()

fig, ax = plt.subplots()
frames = time_duration - 3
interval = 100
heatmap = ax.imshow(u[:, :, 0], cmap='plasma', interpolation='nearest', vmin
= -50, vmax = 50)

def update(frame):
    heatmap.set_array(u[:, :, frame - 1])
    return [heatmap]

plt.colorbar(heatmap)
plt.axis('off')
ani = FuncAnimation(fig, update, frames=frames, interval=interval,
blit=True)
ani.save(path + f'test{try_number}.gif', writer=PillowWriter(fps=10))

u_solve = np.zeros_like(u)
R = np.zeros_like(u)

def solve(x, y, t):
    return np.exp(-2 * np.pi ** 2 * t) * np.sin(np.pi * x) * np.sin(np.pi *
y)
```

```
def create_u_solve_and_R():
    global max_R
    global time_R
    for k in range(time_duration):
        for i in range(field_size):
            for j in range(field_size):
                u_solve[i, j, k] = solve(i * h, j * h, k * t)
                R[i, j, k] = abs(u_solve[i, j, k] - u[i, j, k])
                if R[i, j, k] > max_R:
                    max_R = R[i, j, k]
                    time_R = k


max_R = 0.0
time_R = 0.0
create_u_solve_and_R()

fig, ax = plt.subplots()
frames = time_duration - 3
interval = 100
heatmap = ax.imshow(u[:, :, 0], cmap='plasma', interpolation='nearest', vmin
= 0, vmax = 1)

def update(frame):
    heatmap.set_array(u_solve[:, :, frame - 1])
    return [heatmap]

plt.colorbar(heatmap)
plt.axis('off')
ani = FuncAnimation(fig, update, frames=frames, interval=interval,
blit=True)
ani.save(path + f'solve animation{try_number}.gif',
writer=PillowWriter(fps=10))

print(f'Максимальная погрешность равна {max_R} на итерации {time_R}')

x = np.arange(field_size)
y = np.arange(field_size)
x, y = np.meshgrid(x, y)

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
ax.set_zlim(0, 0.005)

# Начальное значение поверхности
z = R[:, :, 0]
surf = ax.plot_surface(x, y, z, cmap='viridis')

def update(frame):
    ax.clear()
```

```
    ax.set_zlim(0, 0.005)
    z = R[:, :, frame]
    surf = ax.plot_surface(x, y, z, cmap='viridis')
    return surf,

ani = FuncAnimation(fig, update, frames=frames, blit=False)
ani.save(path + f'Error animation{try_number}.gif', writer='imagemagick',
fps=15)
plt.show()
```

Приложение 3. Ссылки на анимации

- Нестабильный явный метод: https://imgur.com/a/IGnPehv
- Стабильный явный метод: https://imgur.com/a/MtAoKjN
- Неявный метод: https://imgur.com/a/wpB2Pmr
- 1-я модель: https://imgur.com/a/t94V6S1
- 2-я модель: https://imgur.com/a/1x6v1B0
- 3-я модель: https://imgur.com/a/IZTcCT0
- 4-я модель: https://imgur.com/a/SI8KQD8
- 5-я модель: https://imgur.com/a/4dhIAaC

Литература

1. Численное решение задач математической физики. Нестационарные уравнения: учебно-методическое пособие / К.О. Казёнкин, О.А. Амосова, — М.: Издательство МЭИ, 2016. — 36 с.