Гафни Даниил, 406 группа, физический факультет МГУ

Решение двумерного уравнения теплопроводности

Постановка задачи

Используя метод переменных направлений, решите краевую задачу:

$$\begin{cases} \frac{\partial u}{\partial t} = \Delta u + (yt)^2, \;\; 0 < x < 2, \;\; 0 < y < 1, \;\; t > 0 \\ \frac{\partial u}{\partial x}\Big|_{x=0} = u|_{x=2} = 0 \\ u|_{y=0} = u|_{y=1} = 0 \\ u|_{t=0} = \cos(\pi x/4) \cdot y(1-y) \end{cases}$$

Численное решение

Сетка

Введем в расчетной области сетку, используя фиктивные узлы в окрестности границ, чтобы получить второй порядок аппроксимации для условий Неймана:

$$\begin{cases} x_0 = 0; \;\; x_n = x_0 + nh_x, \;\; n = 0, 1, \ldots, N; \;\; x_N = 2 \longrightarrow h_x = \frac{2}{N-1} \\ y_0 = 0; \;\; y_m = y_0 + mh_y, \;\; m = 0, 1, \ldots, M; \;\; y_M = 1 \longrightarrow h_y = \frac{1}{M-1} \\ t_j = j\tau, \;\; j = 0, 1, \ldots, J; \;\; t_J = T \longrightarrow \tau = \frac{T}{J} \end{cases}$$

На данной сетке будем рассматривать сеточную функцию

$w^j_{n,m} = u(x_n, y_m, t_j)$.

Аппроксимации

Оператор Лапласа

Аппроксимируем оператор Лапласа $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}$ разностным оператором

$\Lambda w = \Lambda_x w + \Lambda_y w$, где

$$\Lambda_x w = \frac{w_{n-1,m} - 2w_{n,m} + w_{n+1,m}}{h_x^2},$$
$$\Lambda_y w = \frac{w_{n,m-1} - 2w_{n,m} + w_{n,m+1}}{h_y^2}.$$

Данная аппроксимация имеет второй порядок аппроксимации.

Здесь и далее в соответствующих ситуациях для краткости верхний индекс j, соответствующий времени, может быть негласно опущен, как и другие.

Неоднородность

$$f(y,t) = (yt)^2 \longrightarrow f^j_{n,m} = (mjh_yh_t)^2,$$
$$\text{где } \; m = 0, 1, \dots, M, \;\; j = 0, 1, \dots, J.$$

Неоднородность аппроксимируется точно.

Начальное условие

$$u|_{t=0} = \cos(\pi x/4) \cdot y(1-y) \longrightarrow w^0_{n,m} = \cos(\pi n h_x/4) \cdot m h_y (1 - m h_y)$$

Начальное условие аппроксимируется точно.

Граничное условие

- По x : $\begin{cases} w_{0,m} = w_{1,m} \\ w_{N,m} = 0 \end{cases} \qquad m = 0, 1, \dots, M$
- По y : $\begin{cases} w_{n,0} = 0 \\ w_{n,M} = 0 \end{cases} \qquad n = 0, 1, \dots, N$

Условие при $x = 0$ имеет первый порядок аппроксимации; остальные аппроксимируются точно.

Метод переменных направлений

В данном методе переход со слоя j на слой $j+1$ осуществляется в два этапа, с помощью вспомогательного промежуточного слоя $j+1/2$. Схема переменных направлений безусловно устойчива при любых шагах h_x, h_y, τ.

При условии, что для начальных и граничных условий порядки аппроксимации будут не ниже первого, и с учетом вышеописанной аппркосимации дифференциальных операторов, которая имеет первый порядок, метод переменных направлений будет давать первый порядок аппроксимации в данном случае.

Рассмотрим подробно переход со слоя j на промежуточный слой $j+1/2$ и дальнейший переход с промежуточного слоя $j+1/2$ на слой $j+1$.

Переход $j \longrightarrow j+1/2$:

Пусть значения на слое j уже известны (на самом первом шаге значения $w^{\,0}_{n,m}$ известны из начального условия). Перейдем на вспомогательный промежуточный слой $j+1/2$, используя **неявную схему по переменной x и явную - по переменной y:**

- Заменим выражение $\frac{\partial^2}{\partial x^2}$ разностным аналогом, взятым на слое $j+1/2: \;\; \Lambda_x w^{\,j+1/2}$.
- А выражение $\frac{\partial^2}{\partial y^2}$ разностным аналогом, взятым на слое $j: \;\; \Lambda_y w^{\,j}$.

При этом неоднороднось $f(x,y,t)$ в правой части уравнения аппроксимируем на промежуточным слое $j+1/2$.

В результате придем к разностному уравнению:

$$\frac{w^{\,j+1/2} - w^{\,j}}{0.5\tau} = \Lambda_x w^{\,j+1/2} + \Lambda_y w^{\,j} + f^{\,j+1/2}$$

Перейдем к конкретной задаче и добавим соответствующее граничное условие:

$$\begin{cases} w^{\,j+1/2}_{\,n,m} - w^{\,j}_{\,n,m} = \left(\frac{\tau}{2{h_x}^2}\, w^{\,j+1/2}_{\,n+1,\,m} - \frac{\tau}{{h_x}^2}\, w^{\,j+1/2}_{\,n,\,m} + \frac{\tau}{2{h_x}^2}\, w^{\,j+1/2}_{\,n-1,\,m} \right) + \left(\frac{\tau}{2{h_y}^2}\, w^{\,j}_{\,n,\,m+1} - \frac{\tau}{{h_y}^2}\, w^{\,j}_{\,n,\,m} + \right. \\ w^{\,j+1/2}_{\,0,m} = w^{\,j+1/2}_{\,1,m}, \quad w^{\,j+1/2}_{\,N,m} = w^{\,j+1/2}_{\,N-1,m} \end{cases}$$

$$\text{где} \;\; n = 1, 2, \ldots, N-1, \;\; m = 1, 2, \ldots, M-1$$

При каждом фиксированным $n = 0, 1, \ldots, N-1$ можно переписать:

$$\begin{cases} \frac{\tau}{2{h_y}^2}\, w^{\,j+1}_{\,n,m-1} - \left(1 + \frac{\tau}{{h_y}^2}\right)\, w^{\,j+1}_{\,n,m} + \frac{\tau}{2{h_y}^2}\, w^{\,j+1}_{\,n,\,m+1} = -\left[\, w^{\,j+1/2}_{\,n,\,m} + \frac{\tau}{2{h_x}^2}\, \left(w^{\,j+1/2}_{\,n+1,\,m} - \, 2w^{\,j+1/2}_{\,n,\,m} + \, w \right.\right. \\ w^{\,j+1}_{\,0,m} = w^{\,j+1}_{\,1,m}, \quad w^{\,j+1}_{\,N,m} = 0 \end{cases}$$

$$\text{где} \quad m = 1, 2, \ldots, M-1$$

Введем обозначения:

$\chi_n = w^{\,j+1/2}_{\,n,m}, \quad \chi_{n-1} = 0, \quad \chi_{n+1} = w^{\,j+1/2}_{\,n+1,m},$

$A^x = B^x = \frac{\tau}{2{h_x}^2}, \quad C^x = \left(1 + \frac{\tau}{2{h_x}^2}\right),$

$F^x_n \; = \; w^{\,j}_{\,n,\,m} + \frac{\tau}{2{h_y}^2}\, \left(w^{\,j}_{\,n,\,m+1} - \, 2w^{\,j}_{\,n,\,m} + \, w^{\,j}_{\,n,\,m-1}\right) + \frac{\tau}{2}(mjh_y h_t)^2 \,.$

Получим простую систему, состоящую из уравнения, в котором неизвестные связаны рекуррентным соотношением, и граничных условий:

$$\begin{cases} A^x \chi_{n-1} - C^x \chi_n + B^x \chi_{n+1} = -F^x_n, \\ \chi_0 = \chi_1, \quad \chi_N = \chi_{N-1}. \end{cases} \quad n = 1, \ldots, N-1$$

Данную систему можно решить методом прогонки (https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BF%D1%80%D0%BE%D0%B3%D0%BE%D0%BD%D0%BA%D0%B8).

И снова получим простую систему уже для перехода $j + 1/2 \longrightarrow j + 1$,

состоящую из уравнения, в котором неизвестные связаны рекуррентным соотношением, и граничных условий:

$$\begin{cases} A^y \gamma_{n-1} - C^y \gamma_n + B^y \gamma_{n+1} = -F^y_m, \\ \gamma_0 = \gamma_1, \quad \gamma_n = \gamma_{n-1}. \end{cases} \quad m = 1, \ldots, M-1$$

Данная система аналогично решается методом прогонки.

Метод прогонки

Рассмотрим систему для перехода $j \longrightarrow j + 1/2$:

$$\begin{cases} A^x \chi_{n-1} - C^x \chi_n + B^x \chi_{n+1} = -F^x_n, \\ \chi_0 = \chi_1, \quad \chi_N = \chi_{N-1}. \end{cases} \quad n = 1, \ldots, N-1$$

Система для перехода $j + 1/2 \longrightarrow j + 1$ будет решаться абсолютно аналогично.

Прямой ход прогонки

Идея заключается в первоначальном нахождении всех коэффицентов прогонки α_n и β_n через известные α_1 и $\beta _1$.

Рекуррентное соотношение: $\chi_n = \alpha_{n+1}\chi_{n+1} + \beta_{n+1}$

Тогда $\chi_{n-1}(\chi_n)$: $\chi_{n-1} = \alpha_n\chi_n + \beta_n = \alpha_n\alpha_{n+1}\chi_{n+1} + \alpha_n\beta_{n+1} + \beta_n$

В результате после подстановки в первое уравнение системы, получим:

$$A^x\left(\alpha_n\alpha_{n+1}\chi_{n+1} + \alpha_n\beta_{n+1} + \beta_n\right) - C^x\left(\alpha_{n+1}\chi_{n+1} + \beta_{n+1}\right) + B^x\chi_{n+1} = -F_n^x$$

Приравняв коэффициенты при одинаковых степенях χ_{n+1}:

$$\begin{array}{rc} \chi_{n+1}: & A^x\alpha_n\alpha_{n+1} - C^x\alpha_{n+1} + B^x = 0 \\ \chi_{n+1}^0: & A^x\alpha_n\beta_{n+1} + A^x\beta_n - C^x\beta_{n+1} + F_n^x = 0 \end{array}$$

Выразим $\alpha_{n+1}(\alpha_n)$ и $\beta_{n+1}(\beta_n)$:

$$\alpha_{n+1} = \frac{B^x}{C^x - A^x\alpha_n},\ \ \beta_{n+1} = \frac{A^x\beta_n + F_n^x}{C^x - A^x\alpha_n},\ n = 1, 2, 3, \dots, N-1$$

Из первых граничных условий:

$$\chi_0 = k_1\chi_1 + \mu_1 = \chi_1 \Rightarrow \alpha_1 = k_1 = 1, \beta_1 = \mu_1 = 0$$

В итоге получим формулы для прямой прогонки:

$$\begin{cases} \alpha_{n+1} = \frac{B^x}{C^x - A^x\alpha_n},\ \ \beta_{n+1} = \frac{A^x\beta_n + F_n^x}{C^x - A^x\alpha_n},\ n = 1, 2, 3, \dots, N-1 \\ \alpha_1 = 1,\ \ \beta_1 = 0 \end{cases}$$

Обратный ход прогонки

По известному χ_N и найденымм ранее коэффициентам α_n, β_n вычисляем значения χ_n.

$$\chi_n = \alpha_{n+1}\chi_{n+1} + \beta_{n+1}$$

Из вторых граничных условий:

$$\chi_N = k_2\chi_{N-1} + \mu_2 = \chi_{N-1} \Rightarrow \ k_2 = 1,\ \ \mu_2 = 0$$

Откуда получим:

$$\chi_N = \frac{k_2\beta_N + \mu_2}{1 - \alpha_N k_2}$$

Используем, что $k_2 = 1, \mu_2 = 0$, и получим итоговые формулы для обратной прогонки:

$$\begin{cases} \chi_n = \alpha_{n+1}\chi_{n+1} + \beta_{n+1} \\ \chi_N = \frac{\beta_N}{1-\alpha_N} \end{cases}$$

Сложность

Как видим, здесь для прямой прогонки необходимо $0(N)$ действий для одной системы. Поскольку систем таких $M-1$, суммарная сложность будет $O(NM)$.

Аналогично для обратной прогонки: сложность $0(M)$ для одной системы, а систем $N-1$. Таким образом, для обратной прогонки сложность будет $O(MN)$.

Суммарная сложность перехода $j+1 \longrightarrow j+1/2$ будет $O(NM)$.

Такая же сложность будет и для перехода $j+1/2 \longrightarrow j+1$.

В итоге, для перехода $j \longrightarrow j+1$ сложность будет все так же $O(NM)$, а сложность всей задачи $O(NMJ)$. Именно поэтому метод переменных направлений относится к так называемым экономичным схемам.

Экономичные схемы сочетают в себе достоинства явных и неявных схем (требуют при переходе со слоя на слой числа арифметических операций, пропорционального числу узлов сетки, и являются безусловно устойчивыми, соответственно).

Код

На языке Python 3.6.8

Импорт необходимых библиотек

In [1]:
```
import os
from tqdm import tqdm_notebook
import numpy as np
from numba import njit, jitclass
import plotly
import plotly.graph_objs as go
from plotly.offline import iplot
```

Некоторые настройки

In [2]:
```
# Оформление графиков plotly
layout = go.Layout(
    scene = dict( aspectmode='cube', camera = dict(eye=dict(x=-
2, y=1.5, z=1)),
    xaxis = dict(
        title='x',
        gridcolor="rgb(255, 255, 255)",
        zerolinecolor="rgb(255, 255, 255)",
        showbackground=True,
        backgroundcolor="rgb(200, 200, 230)"),

    yaxis = dict(
        title='y',
        gridcolor="rgb(255, 255, 255)",
        zerolinecolor="rgb(255, 255, 255)",
        showbackground=True,
        backgroundcolor="rgb(230, 200,230)"),

    zaxis = dict(
        title='u(x, y, t)',
        gridcolor="rgb(255, 255, 255)",
        zerolinecolor="rgb(255, 255, 255)",
        showbackground=True,
        backgroundcolor="rgb(230, 230,200)",),),
    autosize=False,
    width=800, height=600,
    margin=dict(
        r=20, b=10,
        l=10, t=10),
    )
camera = dict(
    eye=dict(x=-2, y=2, z=1)
)
```

Метод прогонки

Для решения СЛАУ $Ax = F$, где A - трехдиагональная матрица, используется метод прогонки.

In [3]:
```
@njit
def TDMA(coeffs, F):
    '''
    Метод прогонки.

    Параметры:
        coeffs (numpy.array): Трехдианональная матрица коэффицие
нтов уравнений.
        F (numpy.array): Массив правых частей уравнений.
    Вывод:
        (numpy.array): Массив искомых значений неизвестных.
    '''
    N = F.size
    x = np.empty(N)
    A = np.diag(coeffs, -1)
    B = np.diag(coeffs, 1)
    C = np.diag(coeffs, 0)
    alpha = np.empty(N - 1)
    alpha[0] = -B[0]/C[0]
    beta = np.empty(N - 1)
    beta[0] = F[0]/C[0]

    # Прямой ход прогонки
    for i in range(1, N-1):
        alpha[i] = -B[i]/(A[i-1]*alpha[i-1] + C[i])
        beta[i] = ((F[i] - A[i-1]*beta[i-1])/(A[i-1]*alpha[i-1]
+ C[i]))
    x[-1] = (F[-1] - A[-1]*beta[-1]) / (C[-1] + A[-1]*alpha[-1])

    # Обратный ход прогонки
    for i in range(N - 2, -1, -1):
        x[i] = alpha[i]*x[i+1] + beta[i]

    return x
```

Демонстрация работы метода прогонки

In [4]:
```
coeffs = np.array([
    [2, -1, 0, 0, 0],
    [-3, 8, -1, 0, 0],
    [0, -5, 12, 2, 0],
    [0, 0, -6, 18, -4],
    [0, 0, 0, -5, 10]
])
F = np.array([
    -25, 72, -69, -156, 20
])
x = TDMA(coeffs, F)
x
```

```
array([-10.,   5.,  -2., -10.,  -3.])
```

Проверка правильности решения

In [5]:
```
coeffs @ x - F  # Результат ~ 0
```

```
array([ 0.00000000e+00,  0.00000000e+00, -1.42108547e-14,  2.84217094e-14,
       -3.55271368e-15])
```

Основной класс для решения уравнения тепропроводности

In [6]:
```
class HeatEquationSolver_2():
    '''
    Класс для численного решения двумерного уравнения теплопрово
дности.
    '''
    def __init__(self,
                 X_START=0, X_END=2,
                 Y_START=0, Y_END=1,
                 T_START=0, T_END=20,
                 N=5, M=5, J=5):

        self.X_START = X_START
        self.X_END = X_END
        self.Y_START = Y_START
        self.Y_END = Y_END
        self.T_START = T_START
        self.T_END = T_END
        self.N = N
        self.M = M
        self.J = J

        self.x = np.linspace(X_START, X_END, N)
        self.y = np.linspace(Y_START, Y_END, M)
        self.t = np.linspace(T_START, T_END, J)

        self.dx = self.x[1] - self.x[0]
        self.dy = self.y[1] - self.y[0]
        self.dt = self.t[1] - self.t[0]

    def initialize(self, a=1, f=lambda x, y, t:0, fi=lambda x,
y:0,
                   alpha1x=0, alpha2x=0, beta2x=0, beta1x=0, mu1
x=lambda y, t:0, mu2x=lambda y, t:0,
                   alpha1y=0, alpha2y=0, beta2y=0, beta1y=0, mu1
y=lambda x, t:0, mu2y=lambda x, t:0):
        '''
        Задание коэффициентов и функций конкретной задачи.
        '''

        self.a = a  # Коэффициент при операторе Лапласа
        self.f = f  # Функция "источника тепла" - неоднородность
        self.fi = fi  # Начальное условие

        self.alpha1x = alpha1x  # Коэффициент при левом условии
Неймана для x
        self.alpha2x = alpha2x  # Коэффициент при правом условии
Неймана для x
        self.beta1x = beta1x  # Коэффициент при левом условии Ди
```

Задание параметров нашей задачи

In [7]:
```
solver = HeatEquationSolver_2(N = 50, M = 50, J = 50, T_END = 2)
a = 1
def f(x, y, t):
    return (y*t)**2
def fi(x, y):
    return np.cos(np.pi*x/4)*y*(1-y)
alpha1x = 1
alpha2x = 0
alpha1y = 0
alpha2y = 0
beta1x = 0
beta2x = 1
beta1y = 1
beta2y = 1
solver.initialize(a = a, f = f, fi = fi,
                  alpha1x = alpha1x, alpha2x = alpha2x, alpha1y
= alpha1y, alpha2y = alpha2y,
                  beta1x = beta1x, beta2x = beta2x, beta1y = bet
a1y, beta2y = beta2y)
```

Начальное условие

In [8]:
```
solver.plot_initial_state(filename='start')
```

Вычисление решения

In [9]:
```
solver.solve()
```

```
Calculating...
```

```
<ipython-input-6-53a0b5c46e84>:124: TqdmDeprecationWarning:

This function will be removed in tqdm==5.0.0
Please use `tqdm.notebook.tqdm` instead of `tqdm.tqdm_notebook`
```

Состояние функции u(x, y, t) после того, как прошла половина времени:

In [10]:
```
solver.plot_state(n = 50, filename='middle')
```

Конечное состояние функции u(x, y, t):

In [11]:
```
solver.plot_state(n = 100, filename='end')
```

Анимированное численное решение

Иногда необходимо один раз подождать полной прогрузки анимации

In [12]:
```
solver.show_evolution()
```

Play Pause