Московский авиационный институт

(национальный исследовательский институт)

Лабораторные работы

По курсу

Численные методы

Выполнил: Смирнов Д.А.

Группа: М8О-403Б-18

Москва 2021

Лабораторная №4

Задание

Используя схемы переменных направлений и дробных шагов, решить двумерную начально-краевую задачу для дифференциального уравнения параболического типа. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x,t)$.Исследовать зависимость погрешности от сеточных параметров t, h_x, h_y.

Вариант 8

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} - xy\sin(t) \\ u(0,y,t) = 0 \\ u(1,y,t) - u_x(1,y,t) = 0 \\ u(x,0,t) = 0 \\ u(x,1,0) - u_x(x,1,0) = 0 \\ u(x,y,0) = xy \end{cases}$$

Аналитическое решение:

$$u(x,y,t) = xy\cos(t)$$

Теоретический материал

При численном решении многомерных задач математической физики исключительно важным является вопрос об экономичности используемых методов. Конечно - разностную схему будем называть экономичной, если число выполняемых операций (операций типа умножения) пропорционально числу узлов сетки.

За последние 50 лет разработано значительное количество экономичных разностных схем численного решения многомерных задач математической физики, основанных на расщеплении пространственных дифференциальных операторов по координатным направлениям и использовании метода скалярной прогонки вдоль этих направлений.

Из экономичных конечно-разностных схем, получивших наибольшее распространение, в данном разделе рассматриваются схема метода переменных направлений и схема метода дробных шагов. Все эти методы будем называть общим термином - методы расщепления.

Для начала необходимо ввести пространственно-временную сетку:

$$\omega^{\tau}_{h_1 h_2} = \{x_i = ih_1, i = \overline{0, I}, j = \overline{0, J} : t^k = k\tau, k = 0,1,2,...\}$$

Рассмотрим методы решения

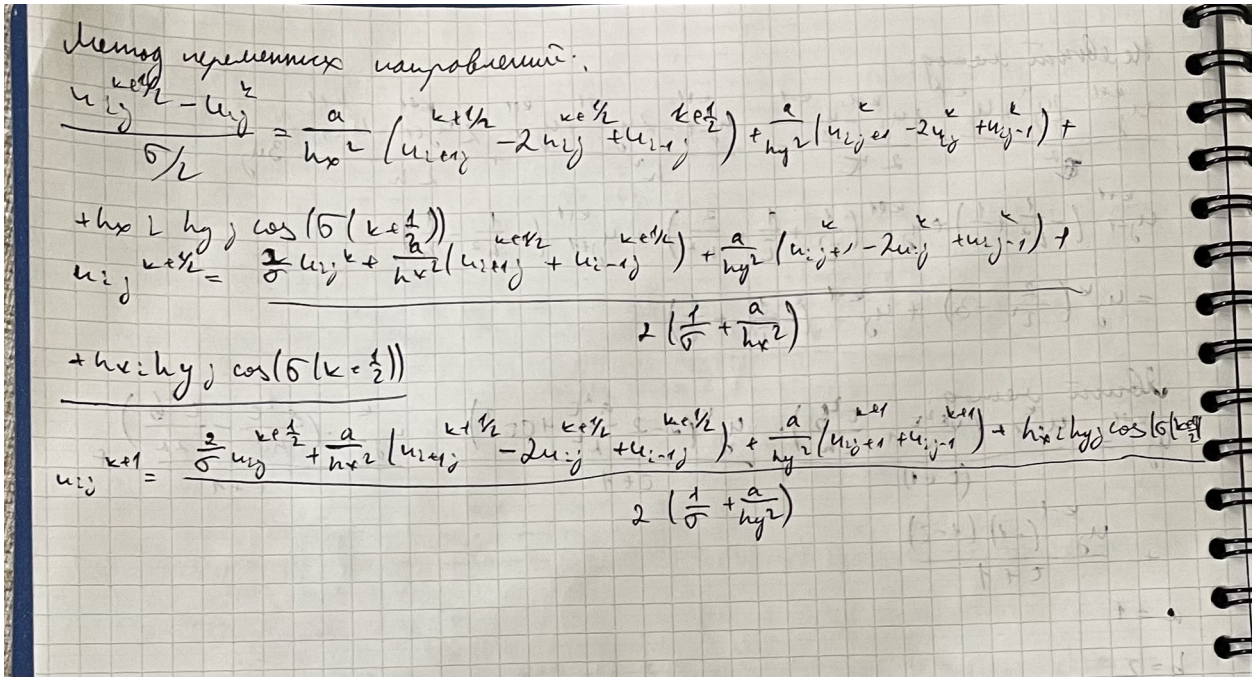
- Метод переменных направлений

Шаг по времени разбивается на число независимых переменных. На каждом дробном слое один из операторов аппроксимируется неявно, а другой явно. Вид для двумерного случая:

$$u_{ij}^{k+1/2} - u_{ij}^{k} = \sigma_x(u_{i+1\,j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1\,j}^{k+1/2}) + \sigma_y(u_{ij+1}^{k} - 2u_{ij}^{k} + u_{ij-1}^{k}) + f_{ij}^{k+1/2}(\tau/2)$$

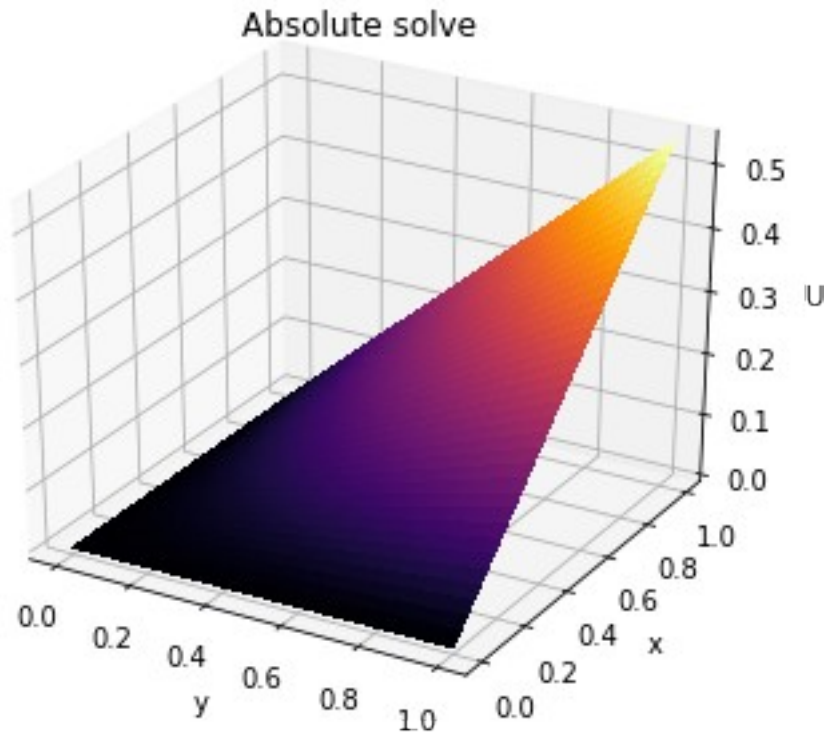
$$u_{ij}^{k+1} - u_{ij}^{k+1/2} = \sigma_x(u_{i+1\,j}^{k+1/2} - 2u_{ij}^{k+1/2} + u_{i-1\,j}^{k+1/2}) + \sigma_y(u_{ij+1}^{k+1} - 2u_{ij}^{k+1} + u_{ij-1}^{k+1}) + f_{ij}^{k+1/2}(\tau/2)$$

$$\sigma_x = \frac{a\tau}{2h_x^2}, \sigma_y = \frac{a\tau}{2h_y^2}$$



Программа

Для начала построим график абсолютного решения



Определяем начально-краевые условия

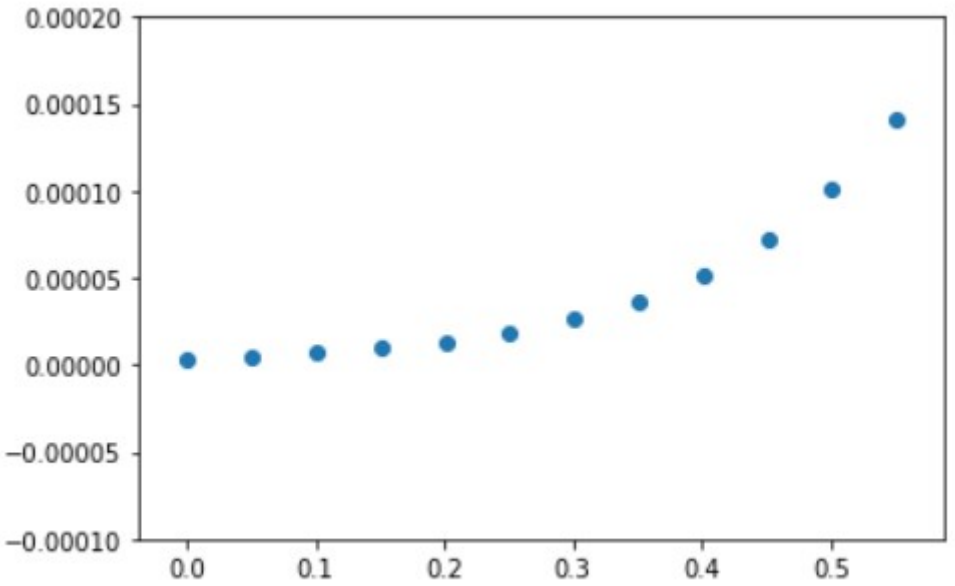
```
def function(x,y,t):
    return -x*y*np.sin(t)

def Psi(x,y):
    return x*y

def U(x, y, t,par_a=1):
    return x*y*np.cos(t)

def Uy_left(x,y,par_a=1):
    return 0

def Uy_right(x,y,par_a=1):
    return 0

def Ux_left(y, t, par_a=1):
    return 0

def Ux_right(y, t, par_a=1):
    return 0
```

Метод дробных шагов

```
def FractionalStepsMethod(x_step, y_step, r,  t, par_a):
    x = np.arange(x0, xl + x_step, x_step)
    y = np.arange(y0, yl + y_step, y_step)
    time = np.arange(0, t + r/2, r/2)

    Solve = np.zeros((len(time), len(x), len(y)))
    ###
    for i in range(len(x)):
        for j in range(len(y)):
            Solve[0][i][j] = Psi(x[i], y[j])

    ###

    for k in range(2, len(time), 2):

        for j in range(1, len(y) - 1):

            a = np.zeros(len(x))
            b = np.zeros(len(x))
            c = np.zeros(len(x))
            d = np.zeros(len(x))

            tmp = par_a * r / x_step**2
            for i in range(1, len(x) - 1):
                a[i] = tmp
                b[i] = -2 * tmp - 1
                c[i] = tmp
                d[i] = - Solve[k - 2][i][j]  - (r/2)*function(x[i],y[j],time[k-2])

            alpha = 0
            betta = 1
            gamma = -1
            delta = 1
            b[0] = betta - alpha / x_step
            c[0] = alpha / x_step
            d[0] = Ux_left(y[j], time[k-1], par_a)

            a[-1] = - gamma / x_step
            b[-1] = delta + gamma / x_step
            d[-1] = Ux_right(y[j], time[k-1], par_a)

            ans = runMethod(a, b, c, d, len(d))

            for i in range(1, len(x) - 1):
                Solve[k-1][i][j] = ans[i]

        for j in range(len(y)):
            Solve[k-1][0][j] =  Ux_left(y[j], time[k-1], par_a)
            Solve[k-1][-1][j] = (x_step*Ux_right(y[j], time[k-1], par_a) - Solve[k-1]
[-2][j])/(x_step - 1)
        # k + 1
        for i in range(1, len(x)):
            a = np.zeros(len(y))
            b = np.zeros(len(y))
            c = np.zeros(len(y))
            d = np.zeros(len(y))


            tmp = par_a * r / y_step**2
            for j in range(1, len(y) - 1):
                a[j] = tmp
                b[j] = -2 * tmp - 1
                c[j] = tmp
                d[j] = - Solve[k-1][i][j] - (r/2)*function(x[i],y[j],time[k])
            alpha = 0
            betta = 1
            gamma = -1
            delta = 1
            b[0] = betta - alpha / y_step
```

```
            c[0] = alpha / y_step
            d[0] = Uy_left(x[i], time[k], par_a)

            a[-1] = - gamma / y_step
            b[-1] = delta + gamma / y_step
            d[-1] = Uy_right(x[i], time[k], par_a)

            ans = runMethod(a, b, c, d, len(d))
            for j in range(len(y)):
                Solve[k][i][j] = ans[j]

        for j in range(len(y)):
            Solve[k][0][j] =  Ux_left(y[j], time[k], par_a)
            Solve[k][-1][j] = (x_step*Ux_right(y[j], time[k], par_a) - Solve[k][-2]
[j])/(x_step - 1)

    return Solve[-1].transpose()
```

График зависимости ошибки от размера шага по пространству:



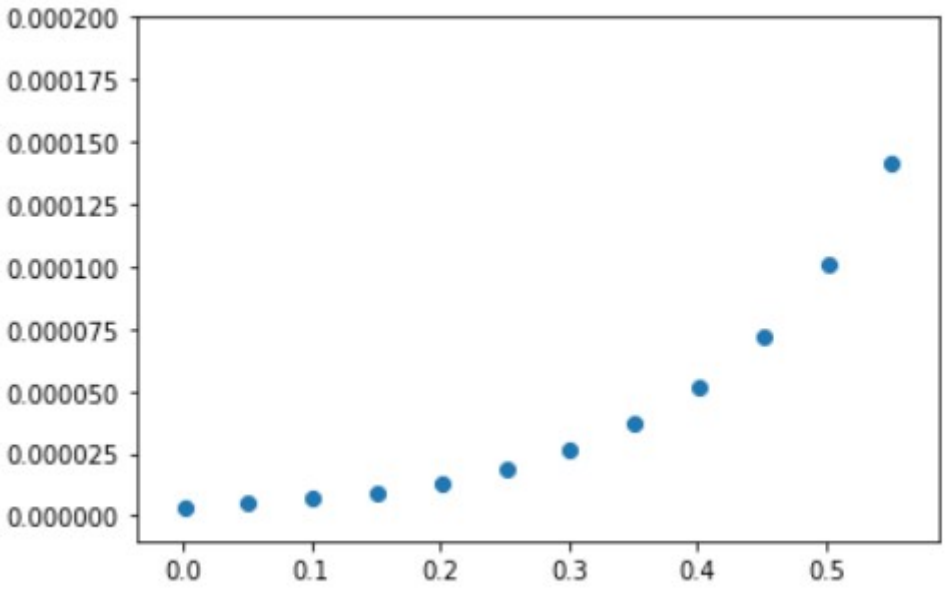
Метод переменных направлений

```
def AlternatingDirectionsMethod(x_step, y_step, r,  t, par_a):  # Переменных
направлений
    x = np.arange(x0, xl + x_step, x_step)
    y = np.arange(y0, yl + y_step, y_step)
    time = np.arange(0, t + r/2, r/2)

    Solve = np.zeros((len(time), len(x), len(y)))
    ###
    for i in range(len(x)):
        for j in range(len(y)):
            Solve[0][i][j] = Psi(x[i], y[j])

    ###

    for k in range(2, len(time), 2):

        for i in range(1, len(x)):
            a = np.zeros(len(y))
            b = np.zeros(len(y))
```

```
            c = np.zeros(len(y))
            d = np.zeros(len(y))

            tmp2 = par_a * r / (2*x_step**2)
            tmp1 = par_a * r / (2*y_step**2)
            for j in range(1, len(y) - 1):
                a[j] = tmp2
                b[j] = -2 * tmp2 - 1
                c[j] = tmp2
                d[j] = - Solve[k-2][i][j] - (r/2)*function(x[i],y[j],time[k-2]) - \
                tmp1*(Solve[k-2][i][j+1] - 2*Solve[k-2][i][j] + Solve[k-2][i][j-1])

            b[0] = 1
            c[0] = 0
            d[0] = Uy_left(x[i], time[k-1], par_a)

            a[-1] = 1 / y_step
            b[-1] = 1 - 1 / y_step
            d[-1] = Uy_right(x[i], time[k-1], par_a)

            ans = runMethod(a, b, c, d, len(d))
            for j in range(len(y)):
                Solve[k-1][i][j] = ans[j]

        for i in range(len(x)):
            Solve[k-1][i][0] =  Uy_left(x[i], time[k-1], par_a)
            Solve[k-1][i][-1] = (y_step*Uy_right(x[i], time[k-1], par_a) - Solve[k-1]
[i][-2])/(y_step - 1)
        for j in range(len(y)):
            Solve[k-1][0][j] =  Ux_left(y[j], time[k-1], par_a)
            Solve[k-1][-1][j] = (x_step*Ux_right(y[j], time[k-1], par_a) - Solve[k-1]
[-2][j])/(x_step - 1)

        for j in range(1, len(y) - 1):

            a = np.zeros(len(x))
            b = np.zeros(len(x))
            c = np.zeros(len(x))
            d = np.zeros(len(x))

            tmp2 = par_a * r / (2*x_step**2)
            tmp1 = par_a * r / (2*y_step**2)
            for i in range(1, len(x) - 1):
                a[i] = tmp1
                b[i] = - 2 * tmp1 - 1
                c[i] = tmp1
                d[i] = - Solve[k - 1][i][j]  - (r/2)*function(x[i],y[j],time[k-1]) - \
                tmp2*(Solve[k-1][i+1][j] - 2*Solve[k-1][i][j] + Solve[k-1][i-1][j])


            b[0] = 1
            c[0] = 0
            d[0] = Ux_left(y[j], time[k], par_a)

            a[-1] = 1 / x_step
            b[-1] = 1 - 1 / x_step
            d[-1] = Ux_right(y[j], time[k], par_a)

            ans = runMethod(a, b, c, d, len(d))

            for i in range(len(x)):
                Solve[k][i][j] = ans[i]

        for j in range(len(y)):
            Solve[k][0][j] =  Ux_left(y[j], time[k], par_a)
            Solve[k][-1][j] = (x_step*Ux_right(y[j], time[k], par_a) - Solve[k][-2]
[j])/(x_step - 1)
        for i in range(len(x)):
            Solve[k][i][0] =  Uy_left(x[i], time[k], par_a)
            Solve[k][i][-1] = (y_step*Uy_right(x[i], time[k], par_a) - Solve[k][i][-
2])/(y_step - 1)
```

```
        # k + 1

    return Solve[-1].transpose()
```

График зависимости ошибки от размера шага по пространству:



Вывод: используя схемы переменных направлений и дробных шагов, научился решать двумерную начально-краевую задачу для дифференциального уравнения параболического типа.