Московский авиационный институт
(Национальный исследовательский университет)
Факультет прикладной математики и физики
Кафедра вычислительной математики и программирования

Лабораторная работа № 8

по курсу «Численные методы»

Студент: Аксенов А. Е.
Группа: М80-408Б-20
Преподаватель: Пивоваров Д. Е.
Оценка:

Москва, 2023

Лабораторная №8

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 τ, h_x, h_y.

<u>Вариант 1</u>

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + a\frac{\partial^2 u}{\partial y^2},\ a > 0,$$

$$u(0,y,t) = \cos(\mu_2 y)\exp(-(\mu_1^2 + \mu_2^2)at),$$
$$u(\pi,y,t) = (-1)^{\mu_1}\cos(\mu_2 y)\exp(-(\mu_1^2 + \mu_2^2)at),$$

$$u(x,0,t) = \cos(\mu_1 x)\exp(-(\mu_1^2 + \mu_2^2)at),$$
$$u(x,\pi,t) = (-1)^{\mu_2}\cos(\mu_1 x)\exp(-(\mu_1^2 + \mu_2^2)at),$$

$$u(x,y,0) = \cos(\mu_1 x)\cos(\mu_2 y).$$

Аналитическое решение: $U(x,y,t) = \cos(\mu_1 x)\cos(\mu_2 y)\exp(-(\mu_1^2 + \mu_2^2)at)$.

1). $\mu_1 = 1,\ \mu_2 = 1$.

2). $\mu_1 = 2,\ \mu_2 = 1$.

3). $\mu_1 = 1,\ \mu_2 = 2$.

Теоретический материал

Для начала необходимо ввести пространственно-временную сетку:

$$\omega_{h_1 h_2}^{\tau} = \{\mathrm{x}_i = \mathrm{ih}_1, \mathrm{i} = \overline{0,\mathrm{I}}, \mathrm{j} = \overline{0,J} : \mathrm{t}^k = k\tau, k = 0,1,2,...\}$$

<u>Далее опишем два метода</u>

- 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:

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

- Метод дробных шагов

В отличие от МПН данный метод использует только неявные конечно-разностные операторы, что делает его абсолютно устойчивым в задачах, не содержащих смешанные производные.

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

Ключевые моменты программы

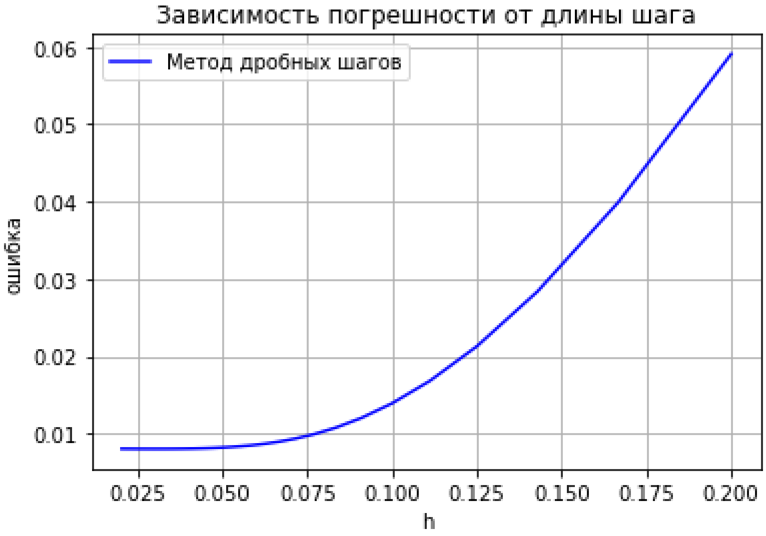
Реализация методов

```
def first_step(i, X, Y, last_ans_line, ans_line, h_x, h_y, tau, a, N_x, N_y, N_t, method):
    hy2 = h_y * h_y
    hx2 = h_x * h_x

    c_x = a * tau
    c_y = a * tau

    if method == "MFS":
        b_x = -2 * a * tau - hx2
        b_y = -2 * a * tau - hy2
    elif method == "MVD":
        b_x = -2 * hy2 * (a * tau + hx2)
        b_y = -2 * hx2 * (a * tau + hy2)
        c_x *= hy2
        c_y *= hx2
    else:
        print("undefined method")


    A = [(0, b_x, c_x)]
    if method == "MFS":
        w = [
            -hx2 * last_ans_line[i][1] - c_x*ans_line[i][0]
        ]
    elif method == "MVD":
        w = [
            -c_y*last_ans_line[i-1][1] -
            (2*hx2*hy2 - 2*c_y)*last_ans_line[i][1] -
            c_y*last_ans_line[i+1][1] - c_x*ans_line[i][0]
        ]

    A.extend([(c_x, b_x, c_x) for _ in range(2, N_x - 2)])
    if method == "MFS":
        w.extend([
            -hx2 * last_ans_line[i][j]
            for j in range(2, N_x - 2)
        ])
    elif method == "MVD":
        w.extend([
            -c_y*last_ans_line[i-1][j] -
            (2*hx2*hy2 - 2*c_y)*last_ans_line[i][j] -
            c_y*last_ans_line[i+1][j]
            for j in range(2, N_x - 2)
        ])

    A.append((c_x, b_x, 0))
    if method == "MFS":
        w.append(
            -hx2 * last_ans_line[i][-2] - c_x*ans_line[i][-1]
        )
    elif method == "MVD":
        w.append(
            -c_y*last_ans_line[i-1][-2] -
            (2*hx2*hy2 - 2*c_y)*last_ans_line[i][-2] -
            c_y*last_ans_line[i+1][-2] - c_x*ans_line[i][-1]
        )

    line = race_method(A, w)

    for j in range(1, N_x - 1):
        ans_line[i][j] = line[j - 1]
```

```python
def second_step(j, X, Y, t, last_ans_line, ans_line, h_x, h_y, tau, a, N_x, N_y, N_t, method):
    hy2 = h_y * h_y
    hx2 = h_x * h_x

    c_x = a * tau
    c_y = a * tau

    if method == "MFS":
        b_x = -2 * a * tau - hx2
        b_y = -2 * a * tau - hy2
    elif method == "MVD":
        b_x = -2 * hy2 * (a * tau + hx2)
        b_y = -2 * hx2 * (a * tau + hy2)
        c_x *= hy2
        c_y *= hx2
    else:
        print("undefined method")


    A = [(0, b_y, c_y)]
    if method == "MFS":
        w = [
            -hy2 * last_ans_line[1][j] - c_y*ans_line[0][j]
        ]
    elif method == "MVD":
        w = [
            -c_x*last_ans_line[1][j - 1] -
            (2*hx2*hy2 - 2*c_x)*last_ans_line[1][j] -
            c_x*last_ans_line[1][j + 1] - c_y*ans_line[0][j]
        ]

    A.extend([(c_y, b_y, c_y) for _ in range(2, N_y - 1)])
    if method == "MFS":
        w.extend([
            -hy2 * last_ans_line[i][j]
            for i in range(2, N_y - 1)
        ])
    elif method == "MVD":
        w.extend([
            -c_x*last_ans_line[i][j - 1] -
            (2*hx2*hy2 - 2*c_x)*last_ans_line[i][j] -
            c_x*last_ans_line[i][j + 1]
            for i in range(2, N_y - 1)
        ])

    A.append((c_y, b_y, 0))
    if method == "MFS":
        w.append(
            -hy2 * last_ans_line[-2][j] - c_x*ans_line[-1][j]
        )
    elif method == "MVD":
        w.append(
            -c_x*last_ans_line[-2][j-1] -
            (2*hx2*hy2 - 2*c_x)*last_ans_line[-2][j] -
            c_x*last_ans_line[-2][j+1] - c_x*ans_line[-1][j]
        )

    line = race_method(A, w)

    for i in range(1, N_y):
        ans_line[i][j] = line[i - 1]
```

```python
def finite_difference_scheme(
        t_begin = 0.0, t_end = 1.0,
        x_begin = 0.0, x_end = math.pi,
        y_begin = 0.0, y_end = math.pi,
        N_t = 30, N_x = 30, N_y = 30,
        epsilon = 1e-5,
        mu = [1, 1],
        a = 1.0,
        method = "MFS"):

    tau = (t_end - t_begin) / (N_t - 1)
    h_x = (x_end - x_begin) / (N_x - 1)
    h_y = (y_end - y_begin) / (N_y - 1)

    x = np.linspace(x_begin, x_end, N_x)
    y = np.linspace(y_begin, y_end, N_y)

    X = [x for _ in range(N_y)]
    Y = [[y[i] for _ in x] for i in range(N_y)]
    T = [0.0]
    line = [[0.0 for _ in range(N_x)] for _ in range(N_y)]
    for i in range(N_y):
        for j in range(N_x):
            line[i][j] = init_cond(X[i][j], Y[i][j])
    ans = [line]

    for t in np.linspace(t_begin, t_end, N_t):
        last_ans_line = ans[-1]
        ans_line = [[0.0 for _ in range(N_x)] for _ in range(N_y)]

        left_boundary_y(Y, N_y, t - 0.5*tau, ans_line, mu, a)
        right_boundary_y(Y, N_y, t - 0.5*tau, ans_line, mu, a)
        left_boundary_x(X, N_x, t - 0.5*tau, ans_line, mu, a)
        right_boundary_x(X, N_x, t - 0.5*tau, ans_line, mu, a)

        for i in range(1, N_y - 1):
            first_step(i, X, Y, last_ans_line, ans_line, h_x, h_y, tau, a, N_x, N_y, N_t, method)

        last_ans_line = ans_line
        ans_line = [[0.0 for _ in range(N_x)] for _ in range(N_y)]

        left_boundary_y(Y, N_y, t, ans_line, mu, a)
        right_boundary_y(Y, N_y, t, ans_line, mu, a)
        left_boundary_x(X, N_x, t, ans_line, mu, a)
        right_boundary_x(X, N_x, t, ans_line, mu, a)

        for j in range(1, N_x - 1):
            second_step(j, X, Y, t, last_ans_line, ans_line, h_x, h_y, tau, a, N_x, N_y, N_t, method)

        ans.append(ans_line)

        T.append(t)

    return X, Y, T, ans
```

Ошибки

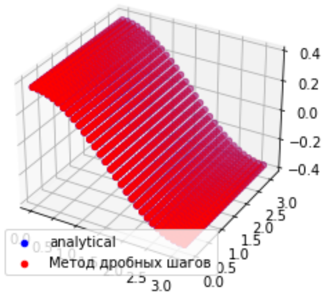



```
]:   1  error_plot(method="MVD", method_name="Метод переменных направлений")
```

Численные и аналитические решения

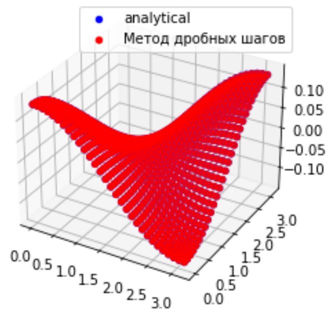
```
R3_plot(method="MFS", method_name="Метод дробных шагов", time=30, mu=[1,0])
```

RMSE = 0.0023939538098467594



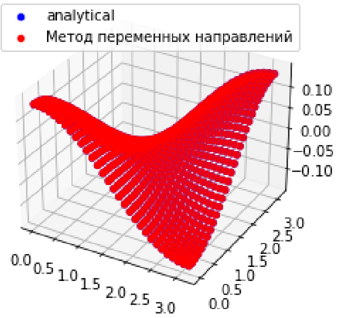

```
R3_plot(method="MFS", method_name="Метод дробных шагов", time=30, mu=[1,1])
```

RMSE = 0.00037798579337682404



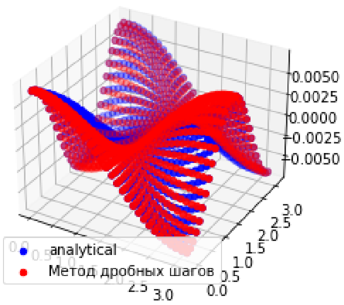
```
R3_plot(method="MVD", method_name="Метод переменных направлений", time=30, mu=[1,1])
```

RMSE = 4.081463094748494e-05



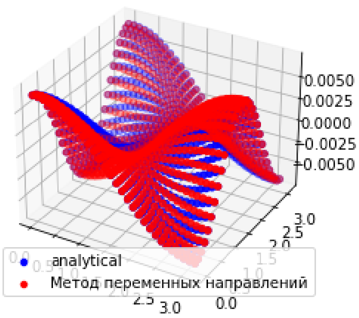

```
R3_plot(method="MFS", method_name="Метод дробных шагов", time=30, mu=[1,2])
```

RMSE = 0.0011424709720930712



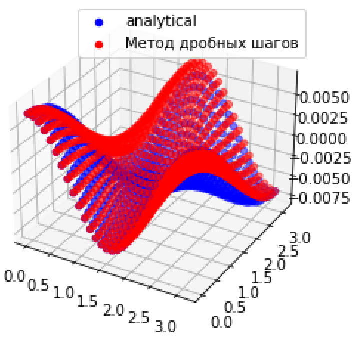

```
R3_plot(method="MVD", method_name="Метод переменных направлений", time=30, mu=[1,2])
```

RMSE = 0.0008956945519685107



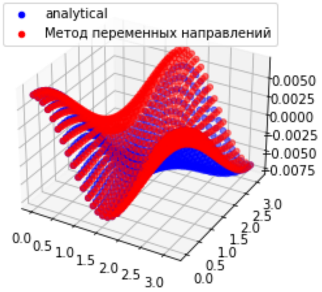

```
R3_plot(method="MFS", method_name="Метод дробных шагов", time=30, mu=[2,1])
```

RMSE = 0.0030333864614875677



```
R3_plot(method="MVD", method_name="Метод переменных направлений", time=30, mu=[2,1])
```

RMSE = 0.0034288198147122183



Вывод

Выполнив лабораторную работу я освоил метод переменных направлений и метод дробных шагов для двумерной начально-краевой задаче для дифференциального уравнения параболического типа. Оба метода достаточно хорошо аппроксимируют нашу задачу. Также хочется обратить внимание, что при увеличении размера максимального шага ошибка возрастает, что говорит о том, что методы сходятся.