Метод сеток для решения уравнения параболического типа

Курдоякова Марина

Задание: Найти решение задачи:

$$\frac{\partial u}{\partial t} = Lu + f(x,t), \quad 0 < x < 1, \quad 0 < t \leqslant 0.1$$

$$u(x,0) = \varphi(x), \quad 0 \leqslant x \leqslant 1$$

$$\left(\alpha_1(t)u - \alpha_2(t)\frac{\partial u}{\partial x}\right)\bigg|_{x=0} = \alpha(t), \quad 0 \leqslant t \leqslant 0.1$$

$$\left(\beta_1(t)u + \beta_2(t)\frac{\partial u}{\partial x}\right)\bigg|_{x=1} = \beta(t), \quad 0 \leqslant t \leqslant 0.1$$

Где, в моем случае, $Lu = a(x,t)\frac{\partial^2 u}{\partial x^2} + b(x,t)\frac{\partial u}{\partial x} + c(x,t)u, \quad a(x,t) \in C_{[0,1]\times[0,T]}, \quad a(x,t) \geqslant a_0 > 0$

используя различные разностные схемы:
1. Явную схему
2. Схему с весами при $\sigma = 0, \sigma = 1, \sigma = 1/2$.

Алгоритм решения задачи:

1. Явный метод:
   Аппроксимируем начальное уравнение в узле (x_i, t_{k-1}):

   $$\frac{u_i^k - u_i^{k-1}}{\tau} = L_h u_i^{k-1} + f\left(x_i, t_{k-1}\right), \quad i = \overline{1, N-1}, \quad k = \overline{1, M}$$

   Из начального условия имеем:

   $$u_i^0 = \varphi\left(x_i\right), \quad i = \overline{0, N}$$

   Граничные условия (3), (4) примут такой вид:

   $$\alpha_1\left(t_k\right) u_0^k - \alpha_2\left(t_k\right) \frac{-3u_0^k + 4u_1^k - u_2^k}{2h} = \alpha\left(t_k\right)$$
   $$\beta_1\left(t_k\right) u_N^k + \beta_2\left(t_k\right) \frac{3u_N^k - 4u_{N-1}^k + u_{N-2}^k}{2h} = \beta\left(t_k\right)$$

   Причем, обозначим за

   $$A = \begin{cases} \max\{a(x,t) \mid 0 \leqslant x \leqslant 1, 0 \leqslant t \leqslant T\} \text{ в случае } a) \\ \max\{p(x) \mid 0 \leqslant x \leqslant 1\} \text{ в случае } \sigma) \end{cases}$$

   И пусть $\nu = \frac{\tau}{h^2}$, тогда условие устойчивости примет вид $A\nu \leqslant \frac{1}{2}$.

   Отсюда следует алгорит вычисления:

   (a) Находим $u_i^0, i = \overline{0, N}$
   (b) Далее находим на формулам $u_i^k = u_i^{k-1} + \tau\left(L_h u_i^{k-1} + f\left(x_i, t_{k-1}\right)\right), i = \overline{1, N-1}$ при $k = 1$.
   (c) Находим u_0^k при $k = 1$
   (d) И u_M^k при $k = 1$

   Тем самым, решение при k = 1 найдено, увеличиваем k на единицу и переходим к пункту 2 до тех пор, пока $k \leqslant M$.

2. Схема с весами:
Пусть σ — вещественный параметр. Рассмотрим однопараметрическое семейство разностных схем

$$\frac{u_i^k - u_i^{k-1}}{\tau} = L_h\left(\sigma u_i^k + (1-\sigma)u_i^{k-1}\right) + f\left(x_i, \overline{t_k}\right), \quad i = \overline{1, N-1}, \quad k = \overline{1, M}$$

Так же, как и в явном методе из начального условия имеем:

$$u_i^0 = \varphi\left(x_i\right), \quad i = \overline{0, N}$$

Для упрощения алгоритма производные в краевых условиях аппроксимируем с первым порядком и получим:

$$\begin{array}{c} \alpha_1\left(t_k\right) u_0^k - \alpha_2\left(t_k\right) \frac{u_1^k - u_0^k}{h} = \alpha\left(t_k\right) \\ \beta_1\left(t_k\right) u_N^k + \beta_2\left(t_k\right) \frac{u_N^k - u_{N-1}^k}{h} = \beta\left(t_k\right) \end{array}$$

Теперь рассмотрим различные значения параметра σ.

(a) $\sigma = 0, \bar{t}_k = t_{k-1}$ - в этом случае разностная схема явная
(b) Если $\sigma \neq 0$, то схема называется неявной двуслойной схемой. В частности, при $\sigma = 1/2, \bar{t}_k = t_k - \tau/2$ получаем разностную схему Кранка-Никольсона с шеститочечным шаблоном.
(c) При $\sigma = 1, \bar{t}_k = t_k$ получаем разностную схему с опережением или чисто неявную схему с четырехточечным шаблоном.

Тогда алгорит вычисления слелующтй:

(a) Находим $u_i^0, i = \overline{0, N}$
(b) Находим $u_i^k, i = \overline{0, N}$ при k = 1 методом прогонки.

Тем самым, решение при k = 1 найдено, увеличиваем k на единицу и переходим к пункту 2 до тех пор, пока $k \leqslant M$.

В методе прогонки коэффициенты:

$$\begin{aligned} -B_0 u_0^k + C_0 u_1^k &= G_0^k \\ A_i u_{i-1}^k - B_i u_i^k + C_i u_{i+1}^k &= \quad G_i^k, \quad i = \overline{1, N-1} \\ A_N u_{N-1}^k - B_N u_N^k &= G_N^k \end{aligned}$$

Будут выражаться следующим образом:

$$A_i^k = \sigma \frac{a(x_i, \bar{t}_k)}{h^2} - \frac{b(x_i, \bar{t}_k)}{h^2}$$

$$C_i^k = \sigma \frac{a(x_i, \bar{t}_k)}{h^2} + \frac{b(x_i, \bar{t}_k)}{h^2}$$

$$B_i^k = \sigma \frac{2a(x_i, \bar{t}_k)}{h^2} - c(x_i, \bar{t}_k) + \frac{1}{\tau}$$

$$G_i^k = -\frac{1}{\tau} u_i^{k-1} - (1-\sigma) L_h u_i^{k-1} - f\left(x_i, \overline{t_k}\right)$$

Код программы: .
Здесь некоторые строчки закомментированны, так как считают другое решение.

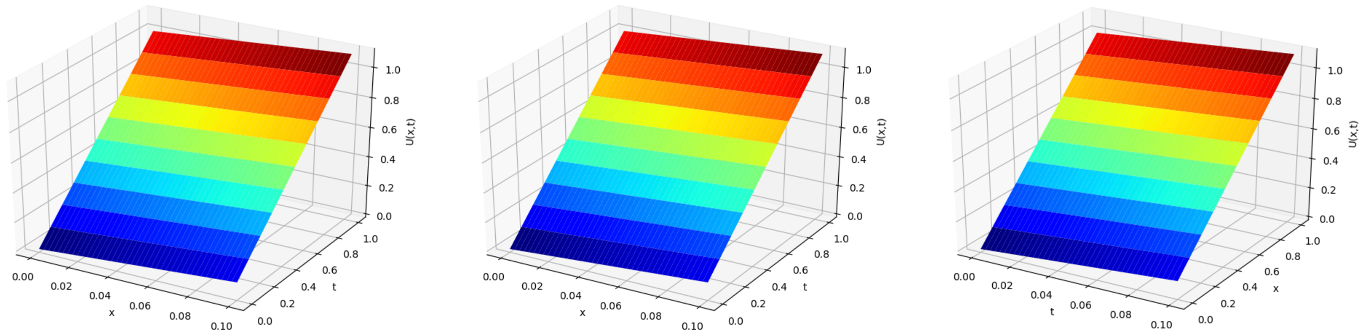
```
import numpy as np
from math import sqrt, sin, pi, cos, log, exp
import matplotlib.pyplot as plt
import pylab
from matplotlib import cm
```

```
from mpl_toolkits.mplot3d import Axes3D

aa, bb = [0,1]
T = 0.1 # in our case
N = 20 #, 10,20
M = 120 #,10,20,40,80
h = (bb-aa)/N
tau = T/M
# tau = 0.5/(M**2)
nu = tau/h**2
sigma = 1 #, 0.5, 0

x = [i*h for i in range((N)+1)]
# t = [i*tau*M/5 for i in range((M)+1)]
t = [i*tau for i in range((M)+1)]

def solution(x,t):
    return x**3 + t**3
    # return x + t

phi = lambda x: solution(x,0)

fi = lambda x,t: -6*x - 3*x**4 - 3*x*x + 3*t*t
a = lambda x,t: 1
b = lambda x,t: x*x + 1
c = lambda x,t: 0
alfa = lambda t: 0
alfa1 = lambda t: 0
alfa2 = lambda t: -1
beta = lambda t: 3
beta1 = lambda t: 0
beta2 = lambda t: 1

# fi = lambda x,t: - x**2
# a = lambda x,t: 1
# b = lambda x,t: x**2 + 1
# c = lambda x,t: 0
# alfa = lambda t: 1
# alfa1 = lambda t: 0
# alfa2 = lambda t: -1
# beta = lambda t: 1
# beta1 = lambda t: 0
# beta2 = lambda t: 1

def Lh(i,k,h,U):
    L1 = (U[i+1][k-1] - 2*U[i][k-1] + U[i-1][k-1])/(h**2)
    L2 = (U[i+1][k-1] - U[i-1][k-1])/(2*h)
    L = a(x[i],t[k])*L1 + b(x[i],t[k])*L2 + c(x[i],t[k])*U[i][k-1]
    return L

def tkst(t,k,sigma):
    if sigma == 1:
        return t[k]
    if sigma == 0:
```

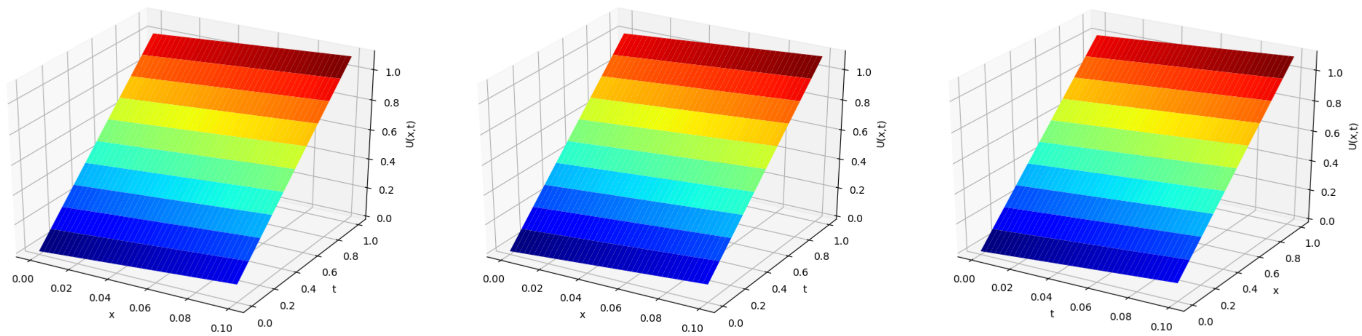
```
        return t[k-1]
    if sigma == 0.5:
        return t[k] - tau/2

def explicit_method():
    U = np.zeros((N+1, M+1))
    for i in range(N+1):
        U[i][0] =  phi(x[i])
    for k in range(1,M+1):
        for i in range(1, N):
            U[i][k] = U[i][k-1] + tau*(Lh(i,k,h,U) + fi(x[i],t[k-1]))
        U[0][k] = (alfa(t[k]) + alfa2(t[k])*(4*U[1][k] - U[2][k])/(2*h)) / (alfa1(t[k]) + (3*alfa2(t[k]))/
        U[N][k] = (beta(t[k]) + beta2(t[k])*(4*U[N-1][k] - U[N-2][k])/(2*h)) / (beta1(t[k]) + (3*beta2(t[k
    return U

def progonka(k,U):
    A = np.zeros(N+1); B = np.zeros(N+1); C = np.zeros(N+1); D = np.zeros(N+1)

    A[0] = 0; C[N] = 0
    B[0] = - (alfa1(t[k]) + alfa2(t[k])/h)
    C[0] = - alfa2(t[k])/h
    D[0] = alfa(t[k])
    B[N] = - (beta1(t[k]) + beta2(t[k])/h)
    A[N] = - beta2(t[k])/h
    D[N] = beta(t[k])

    for i in range(1,N):
        A[i] = sigma*(a(x[i],tkst(t,k,sigma))/h**2 - b(x[i],tkst(t,k,sigma))/(2*h))
        C[i] = sigma*(a(x[i],tkst(t,k,sigma))/h**2 + b(x[i],tkst(t,k,sigma))/(2*h))
        B[i] = sigma*(2*a(x[i],tkst(t,k,sigma))/h**2 - c(x[i],tkst(t,k,sigma))) + 1/tau
        D[i] = - (1/tau)*U[i][k-1] - (1-sigma)*Lh(i,k-1,h,U) - fi(x[i],tkst(t,k,sigma))

    s = np.zeros(N+1); p = np.zeros(N+1); UU = np.zeros(N+1)
    s[0] = C[0]/B[0]; p[0] = -D[0]/B[0]
    for i in range(1,N+1):
        s[i] = C[i]/(B[i] - A[i]*s[i-1])
        p[i] = (A[i]*p[i-1]-D[i])/(B[i]-A[i]*s[i-1])

    UU[N]=p[N]
    for i in range(N-1,-1,-1):
        UU[i] = s[i]*UU[i+1] + p[i]
    return UU

def implicit_method():
    U = np.zeros((N+1, M+1))
    for i in range(N+1):
        U[i][0] = phi(x[i])

    A = np.zeros(N+1); B = np.zeros(N+1); C = np.zeros(N+1); D = np.zeros(N+1)
    for k in range(1,M+1):
        Y = progonka(k,U)
        for i in range(N+1):
            U[i][k] = Y[i]
    return U
```

```
def plot_surface(x, t, U):
    fig = pylab.figure()
    ax = Axes3D(fig)
    x, t = np.meshgrid(x, t)
    ax.plot_surface(x, t, U,cmap = cm.jet)
    ax.set_xlabel('t')
    ax.set_ylabel('x')
    ax.set_zlabel('U(x,t)')
    pylab.show()

def main():
    Q = np.zeros((N+1, M+1))
    J_ex1 = np.zeros((N+1, M+1))
    J_ex2 = np.zeros((N+1, M+1))
    U = explicit_method()
    Y = implicit_method()
    for i in range(N+1):
        for k in range(M+1):
            Q[i][k] = solution(x[i],t[k])
            J_ex1[i][k] = abs(Q[i][k] - U[i][k])
            J_ex2[i][k] = abs(Q[i][k] - Y[i][k])

    # print(J_ex1)
    # print(J_ex2)

    maxx = 0
    Jex = 0
    for i in range(N+1):
        for k in range(M+1):
            if Jex >= maxx:
                Jex = abs(Q[i][k] - U[i][k])
                maxx = Jex
    print(maxx)

    print(h, tau)
    print('sigma = ', sigma)
    # print('N = ', N)
    # print('M = ', M)
    # plot_surface(t, x, Q)
    # plot_surface(t, x, U)
    # plot_surface(t, x, Y)

main()
```

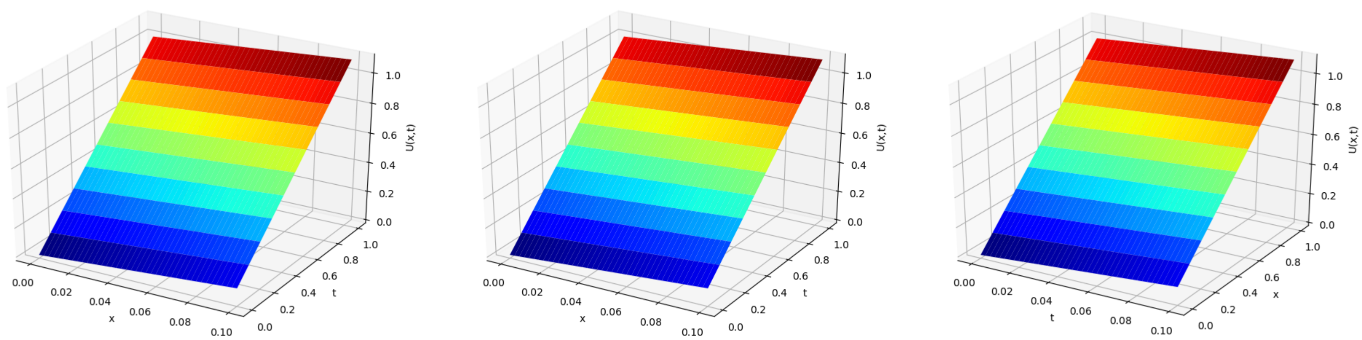
Результаты: Сначала протестируем алгоритм на точном решении $x + t$ Ниже приведены графики для точного решения, речения, найденным явным методом и найденным схемой с весами. N = 10 M = 80 sigma = 0



sigma = 1



sigma = 0.5



Давайте посмотрим на табличку, для точное решения, для явного метода и схемы с веами (они одинаковы):

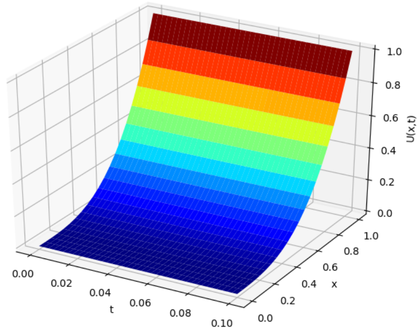
x/t = 0.00	x = 0.20	x = 0.40	x = 0.60	x = 0.80	x = 1.00
t = 0.02	0.22	0.42	0.62	0.82	1.02
t = 0.04	0.24	0.44	0.64	0.84	1.04
t = 0.06	0.26	0.46	0.66	0.86	1.06
t = 0.08	0.28	0.48	0.68	0.88	1.08
t = 0.10	0.30	0.50	0.70	0.90	1.10

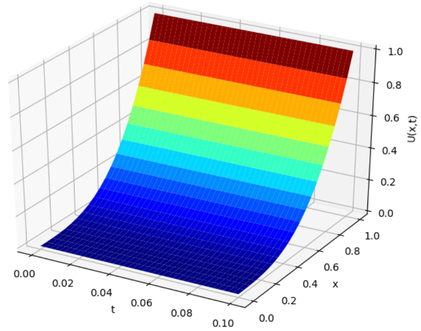
Следующая таблица, характеризующие точность решения в явном случае и в случае схемы с весами, выглядит так (sigma=1)

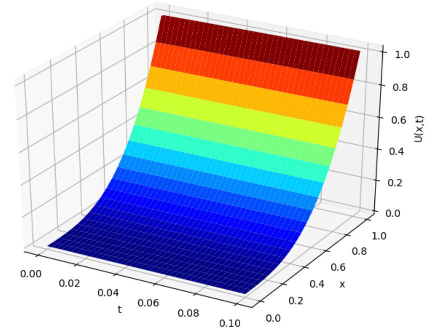
h	tau	max[J_ex1 - u(h,tau)]	max[J_ex2 - u(h,tau)]
0.1	0.00083	1.9984014443252818e-15	7.771561172376096e-15
0.05	0.00083	1.3322676295501878e-15	9.325873406851315e-15
0.2	0.00083	2.220446049250313e-15	6.439293542825908e-15

Теперь рассмотрим неточное решение $x^3 + t^3$:

sigma = 1







h	tau	max[J_ex1 - u(h,tau)]	max[J_ex2 - u(h,tau)]
0.1	3.4723e-05	0.0011462519749942945	0.03933253992828978
0.05	3.4723e-05	0.0005205460972079745	0.014486799716559196
0.2	3.4723e-05	0.011645178477750484	0.12433628755119397

Но с учетом увеличения N и M невязка убывает.