Отчет по вычислительному практикуму

Шилов Максим

Дано уравнение теплопроводности:

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + f(x,t) \quad 0 \le x \le 1, \quad 0 \le t \le 1$$
$$u(x,0) = \mu(x), \quad u(0,t) = \mu_1(t), \quad u(1,t) = \mu_2(t)$$

Схема Кранка-Николсон:

$$\frac{u_k^{j+1} - u_k^j}{\tau} = a(\frac{u_{k+1}^{j+1} - 2u_k^{j+1} + u_{k-1}^{j+1}}{2h^2} + \frac{u_{k+1}^j - 2u_k^j + u_{k-1}^j}{2h^2}) + f_k^{j+\frac{1}{2}} \qquad (1)$$

$$u(x,t) = -2x^4 - 3t^4 + 3t^2x + e^x, \quad a = 0.019$$

$$\Rightarrow \quad f(x,t) = -12t^3 + 6tx - a(-24x^2 + e^x)$$

Погрешность аппроксимации: $\mathcal{O}(\tau^2, h^2)$

Порядок аппроксимации разностной схемы Кранка-Николсон выше, чем порядок аппроксимации явной и неявной разностных схем, т.е. результаты, получаемые при использовании разностной схемы Кранка-Николсон, будут более точными.

Условие устойчивости разностных схем в данном случае выполняется при любых значениях τ и h, то есть разностная схема Кранка-Николсон является абсолютно устойчивой.

Для решения данной разностной схемы необходимо использовать метод прогонки.

$$\text{из (1)} \quad \Rightarrow \quad -\frac{\tau a}{2h^2}u_{k+1}^{j+1} + (1 + \frac{\tau a}{h^2})u_k^{j+1} - \frac{\tau a}{2h^2}u_{k-1}^{j+1} = \frac{\tau a}{2h^2}(u_{k+1}^j - 2u_k^j + u_{k-1}^j) + f_k^{j+\frac{1}{2}}\tau + u_k^j$$

$$\Rightarrow \quad a = -\frac{\tau a}{2h^2}, \quad b = (1 + \frac{\tau a}{h^2}), \quad c = -\frac{\tau a}{2h^2}$$

$$\Rightarrow \quad au_{k+1}^{j+1} + bu_k^{j+1} + cu_{k-1}^{j+1} = \frac{\tau a}{2h^2}(u_{k+1}^j - 2u_k^j + u_{k-1}^j) + f_k^{j+\frac{1}{2}}\tau + u_k^j = \xi_k^j$$

Теорема. Достаточным условием сходимости метода прогонки к решению исходной дифференциальной задачи является выполнение неравенства:

$$|a| + |c| < |b|,$$

где a, c, b - коэффициенты уравнения $au_{k+1}^{j+1} + bu_k^{j+1} + cu_{k-1}^{j+1} = \xi_k^j$

Легко видеть, что для разностной схемы Кранка-Николсон достаточное условие сходимости прогонки выполняется:

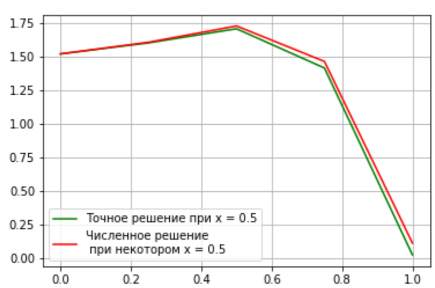
$$|a_j| + |c_j| = \frac{a\tau}{h^2} < 1 + \frac{a\tau}{h^2} = |b_j|$$

Ниже приведена таблица значений погрешности при различных h и τ:

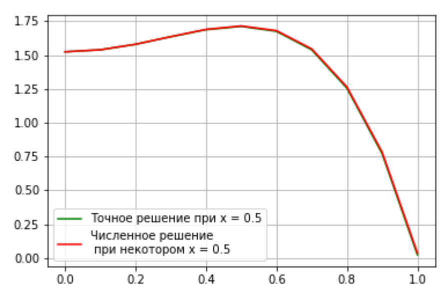
$h \backslash \tau$	0.25	0.1	0.01	0.001
0.25	0.088024	0.010334	0.004331	0.004477
0.1	0.092692	0.014229	0.000578	0.000726
0.01	0.093564	0.014973	0.000142	5.813e-06
0.001	0.093573	0.014981	0.000149	1.425e-06

Графики приближенного и точного решения для значений $0 \le t \le 1$ для $x = 0.5$

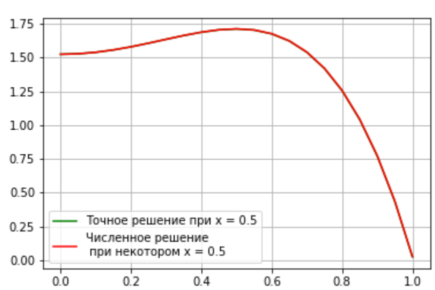
$\tau = h = 0.25$



$\tau = h = 0.1$



$\tau = h = 0.05$



Код программы (Python 3.8)

Преамбула

```
In [0]: import numpy as np
        import matplotlib.pyplot as plt
        import os
```

Реализация

```
In [0]: def f(x, t, a = 0.019):
            return -12*(t**3)+6*t*x-a*(-24*(x**2)+np.exp(x))


        def funk_u(x, t):
            return -2*(x**4)-3*(t**4)+3*(t**2)*x+np.exp(x)


        def Thomas_algorithm(u, j, n, m, a = 0.019):

            tau = 1/n
            h = 1/m

            b = []
            alfa = []
            beta = []

            e_c = (a*tau)/(2*h**2)

            b.append(e_c*(u[2][j]-2*u[1][j]+u[0][j]) +\
                     f(h, (j+1/2)*tau)*tau + u[1][j] + e_c*u[0][j+1])

            for i in range(2, m-1):
                b.append(e_c*(u[i+1][j]-2*u[i][j]+u[i-1][j]) +\
                         f(i*h, (j+1/2)*tau)*tau + u[i][j])

            b.append(e_c*(u[m][j]-2*u[m-1][j]+u[m-2][j]) +\
                     f((m-1)*h, (j+1/2)*tau)*tau + u[m-1][j] + e_c*u[m][j+1])

            e_c = -(a*tau)/(2*h**2)
            d = 1+(a*tau)/(h**2)

            alfa.append(-e_c/d)
            beta.append(b[0]/d)

            for i in range(1, m-2):
                alfa.append(-e_c/(d+e_c*alfa[i-1]))
                beta.append((-e_c*beta[i-1]+b[i])/(d+e_c*alfa[i-1]))
```

```
    x = np.zeros(m-1)

    x[m-2] = (-e_c*beta[m-3]+b[m-2])/(d+e_c*alfa[m-3])

    for i in range(m-3, -1, -1):
        x[i] = alfa[i]*x[i+1]+beta[i]

    return x


def Crank_Nicolson_method(n, m, a = 0.019):

    tau = 1/n
    h = 1/m
    u = np.zeros((m+1, n+1))

    # t = 0
    for i in range(m+1):
        u[i][0] = funk_u(i*h, 0)

    # x = 0 & x = 1
    for i in range(n+1):
        u[0][i] = funk_u(0, i*tau)
        u[m][i] = funk_u(1, i*tau)

    for j in range(0, n):
        x = Thomas_algorithm(u, j, n, m)
        for i in range(1, m):
            u[i][j+1] = x[i-1]

    return u


if __name__ == "__main__":

    n = 10
    m = 10
    u = Crank_Nicolson_method(n, m)
```

Графики

```
In [0]: t = np.arange(0, 1+1/n, 1/n)
        x = 0.5
        y = -2*(x**4)-3*(t**4)+3*(t**2)*x+np.exp(x)

        fig = plt.figure()
        plt.plot(t, y, color='green')

        plt.plot(t, u[5], color='red')

        plt.grid(True)

        # save(str('int_nit_')+str(n), fmt='png')

        plt.show()
```

Погрешность

```
In [0]: tau = 1/n
        h = 1/m

        real_u = np.zeros(u.shape)

        for j in range(0, n+1):
            for i in range(0, m+1):
                real_u[i][j] = funk_u(i*h, j*tau)

        print(np.max(np.abs(real_u - u)))
```

Код для сохранения графиков

```
In [0]: %matplotlib inline
        def save(name='', fmt='png'):
            pwd = os.getcwd()
            iPath = './{}'.format(fmt)
            #     print(iPath)
            if not os.path.exists(iPath):
                os.mkdir(iPath)
            os.chdir(iPath)
            plt.savefig('{}.{}'.format(name, fmt), fmt='png')
            os.chdir(pwd)
            #plt.close()
```