Отчет по вычислительному практикуму

Шилов Максим

Дано уравнение переноса в виде:

$$\frac{\partial u}{\partial t} + C(x,t)\frac{\partial u}{\partial x} = 0 \quad 0 \le x \le 1, \quad 0 \le t \le 1$$
$$u(x,0) = \mu(x), \quad u(0,t) = \mu_1(t)$$

Схема Лакса (явная центральная трехточечная схема):

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

$$u(x,t) = x^3 - \frac{sin(2\pi t)}{2} + x - 3.5t$$

$$C(x,t) = \frac{\pi\cos(2\pi t) + 3.5}{3x^2+1} > 0 \quad \forall x, t \in [0,1]$$

Разностную схему Эйлера можно сделать устойчивой, если заменить u_k^j на пространственное среднее $\frac{u_{k+1}^j + u_{k-1}^j}{2}$. В результате получим широко известную схему Лакса.

Погрешность аппроксимации $\mathcal{O}(\tau + h)$ в точке (x_k, t_j) при условии $\tau = \mathcal{O}(h)$.

Схема условно устойчива при $\frac{C\tau}{h} \le 1$.

Недотающее граничное условие можно вычислить с помощью многочлена Лагранжа:

$$u_M^{j+1} = 2u_{M-1}^{j+1} - u_{M-2}^{j+1}; \quad \text{(многочлен 1-й степени)}$$

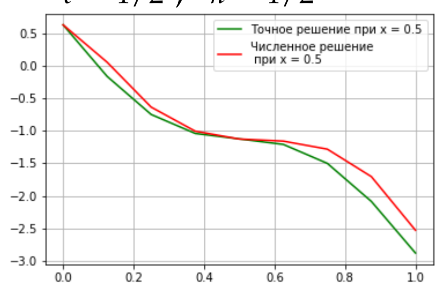
Значения величин u_k^j считаем известными, как посчитанные на предыдущем шаге.
u_k^{j+1} - выражено в (1). Значения на нулевом слое находим из начальных условий.

Ниже приведена таблица значений погрешности при различных h и τ:

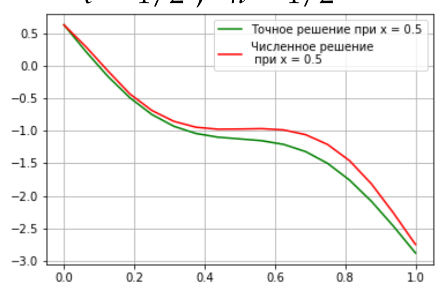
$h \backslash \tau$	$1/2$	$1/2^2$	$1/2^3$	$1/2^4$	$1/2^5$	$1/2^6$	$1/2^7$
$1/2$			0.753294	0.484428	0.350003	0.360096	0.367577
$1/2^2$				0.426900	0.468720	0.542660	0.615828
$1/2^3$					0.361710	0.454929	0.571145
$1/2^4$						0.250940	0.362881
$1/2^5$							0.154340
$1/2^6$							
$1/2^7$							

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

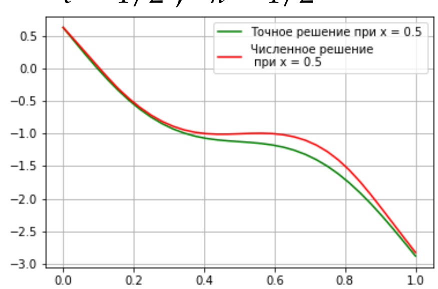
$\tau = 1/2^3, \quad h = 1/2$



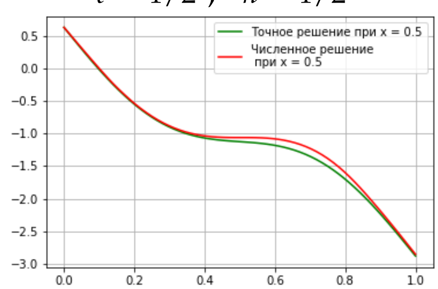
$\tau = 1/2^4, \quad h = 1/2^2$



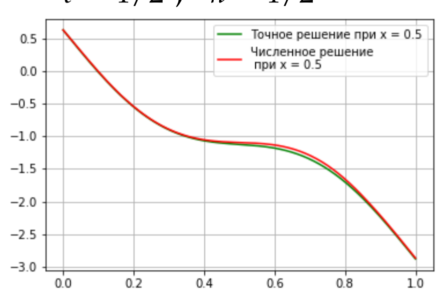
$\tau = 1/2^5, \quad h = 1/2^3$



$\tau = 1/2^6, \quad h = 1/2^4$



$\tau = 1/2^7, \quad h = 1/2^5$



Код программы (Python 3.8)

Преамбула

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

Реализация

```
In [0]: def C(x, t):
            return (np.pi*np.cos(2*np.pi*t) + 3.5)/(3*x**2 + 1)

        def funk_u(x, t):
            return x**3 - np.sin(2*np.pi*t)/2 + x - 3.5*t

        def Lax_method(n, m):

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

            for j in range(0, n):
                for i in range(1, m):
                    u[i][j+1] = (u[i+1][j]+u[i-1][j])/2 -\
                     ((tau*C(i*h, j*tau))/(2*h))*(u[i+1][j]-u[i-1][j])
                u[m][j+1] = 2*u[m-1][j+1] - u[m-2][j+1]

            return u

        if __name__ == "__main__":
            C_max = np.pi + 3.5
            n = 1000 # tau
            m = 100 # h

            if (C_max*m)/(2*n) <= 1:
                u = Lax_method(n, m)
            else:
                print('Не выполнено условие устойчивости')
```

Графики

```
In [0]: t = np.arange(0, 1+1/n, 1/n)
        x = 0.5
        y = x**3 - np.sin(2*np.pi*t)/2 + x - 3.5*t

        fig = plt.figure()
        plt.plot(t, y, label = u'Точное решение при x = 0.5', color='green')

        plt.plot(t, u[int(x*m)], label = u'Численное решение \n при x = 0.5', color='red')

        plt.grid(True)
        plt.legend()
        # save(str('1_') + str(n) + str('int_nit_')+str(1/n), fmt='png')
        plt.show()
```

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()
```

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)))
```