Лабораторная работа 8

Васютинский В.А.

М8О-408Б-20

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 $U(x, y, t)$. Исследовать зависимость погрешности от сеточных параметров τ, h_x, h_y.

Будем решать задачу на заданной площади от 0 до l_x по координате x, от 0 до l_y по координате y и на промежутке от 0 до заданного параметра T по времени t.

Рассмотрим конечно-разностную схему решения краевой задачи на сетке с граничными парамерами l_x, l_y, T и параметрами насыщенности сетки N_x, N_y, K. Тогда размер шага по каждой из координат определяется:

$$h_x=\frac{l_x}{N_x-1}, h_y=\frac{l_y}{N_y-1}, \tau=\frac{T}{K-1}$$

Конечно-разностная схема решения параболического типа в сетке на временном слое t^{k+1} определяется с помощью 2-ух этапов, на каждом из которых решается трёхдиагональное уравнение с помощью метода прогонки:

- Считая, что значения функции $u_{i,j}^k=u(x_i, y_j, t^k)$ на временном слое t^k известно, попробуем определить значения функции на временном слое $t^{k+\frac{1}{2}}$ путем разностной апроксимации производной по времени: $\frac{\partial u}{\partial t}(x_i, y_j, t^k)=(1+\gamma)\frac{u_{i,j}^{k+\frac{1}{2}}-u_{i,j}^k}{\tau}$, неявной аппроксимацией производной по x: $\frac{\partial^2 u}{\partial x^2}(x_i, y_j, t^k)=\frac{u_{i-1,j}^{k+\frac{1}{2}}-2u_{i,j}^{k+\frac{1}{2}}+u_{i+1,j}^{k+\frac{1}{2}}}{h_x^2}$ и явной аппроксимацией по y: $\frac{\partial^2 u}{\partial y^2}(x_i, y_j, t^k)=\frac{u_{i,j-1}^k-2u_{i,j}^k+u_{i,j+1}^k}{h_y^2}$ получаем уравнение:

$$-a\tau h_x^2\gamma u_{i,j-1}^k-\left((1+\gamma)h_x^2h_y^2-2a\tau h_x^2\gamma\right)u_{i,j}^k-a\tau h_x^2\gamma u_{i,j+1}^k=a\tau h_y^2 u_{i-1,j}^{k+\frac{1}{2}}-\left(2a\tau h_y^2+(1+\gamma)h_x^2h_y^2\right)u_{i,j}^{k+\frac{1}{2}}+a\tau h_y^2 u_{i+1,j}^{k+\frac{1}{2}}$$

- Считая, что значения функции $u_{i,j}^{k+\frac{1}{2}}=u\left(x_i, y_j, t^{k+\frac{1}{2}}\right)$ на временном слое $t^{k+\frac{1}{2}}$ известно из прошлого этапа, попробуем определить значения функции на временном слое t^{k+1} путем разностной апроксимации производной по времени:

$\frac{\partial u}{\partial t}\left(x_i, y_j, t^{k+\frac{1}{2}}\right) = (1+\gamma)\frac{u_{i,j}^{k+1} - u_{i,j}^{k+\frac{1}{2}}}{\tau}$, явной аппроксимацией производной по x:

$\frac{\partial^2 u}{\partial x^2}\left(x_i, y_j, t^{k+\frac{1}{2}}\right) = \frac{u_{i-1,j}^{k+\frac{1}{2}} - 2u_{i,j}^{k+\frac{1}{2}} + u_{i+1,j}^{k+\frac{1}{2}}}{h_x^2}$ и неявной аппроксимацией по y:

$\frac{\partial^2 u}{\partial y^2}\left(x_i, y_j, t^{k+\frac{1}{2}}\right) = \frac{u_{i,j-1}^{k+1} - 2u_{i,j}^{k+1} + u_{i,j+1}^{k+1}}{h_y^2}$ получим второе уравнение:

$$-a\tau h_y^2 \gamma u_{i-1,j}^{k+\frac{1}{2}} - \left((1+\gamma)h_x^2 h_y^2 - 2a\tau h_y^2 \gamma\right) u_{i,j}^{k+\frac{1}{2}} - a\tau h_y^2 \gamma u_{i+1,j}^{k+\frac{1}{2}} = a\tau h_x^2 u_{i,j-1}^{k+1} - \left(2a\tau h_x^2 + (1+\gamma)h_x^2 h_y^2\right) u_{i,j}^{k+1} + a\tau h_x^2 u_{i,j+1}^{k+1}$$

При $\gamma=1$ получаем метод переменных направлений, когда как при $\gamma=0$ - метод дробных шагов.

Значения на слое $u_{i,j}^0$ и на границах сетки определяются с помощью заданных граничных условий и их аппроксимаций.

Для того, чтобы получить 2-ой порядок аппроксимации будем аппроксимировать верхнюю границу по y трёхточечной аппроксимацией в явном виде и двухточечной второго порядка в неявном методе.

Трёхточечная аппроксимация второго порядка

Трёхточечная апроксимация второго порядка в точке $y=l_y$ равна соответственно:

$$\frac{3u_{i,N_y}^{k+\frac{1}{2}} - 4u_{i,N_y-1}^{k+\frac{1}{2}} + u_{i,N_y-2}^{k+\frac{1}{2}}}{2h_y} = \psi_1\left(x_i, t^{k+\frac{1}{2}}\right)$$

Тогда, поскольку мы знаем значения для внутренних узлов, получаем выражение для граничного значения при явном методе:

$$u_{i,N_y}^{k+\frac{1}{2}} = \frac{2h_y \psi_1\left(x_i, t^{k+\frac{1}{2}}\right) + 4u_{i,N_y-1}^{k+\frac{1}{2}} - u_{i,N_y-2}^{k+\frac{1}{2}}}{3}$$

Двухточечная аппроксимация второго порядка

Двухточечная апроксимация второго порядка в точке $y=l_y$ равна соответственно:

$$\frac{u_{i,N_y+1}^{k+1} - u_{i,N_y-1}^{k+1}}{2h_y} = \psi_1\left(x_i, t^{k+1}\right)$$

Тогда, поскольку мы знаем значения для внутренних узлов, получаем выражение для граничного значения при неявном методе:

$$-a\tau h_y^2 \gamma u_{i-1,N_y}^{k+\frac{1}{2}} - \left((1+\gamma)h_x^2 h_y^2 - 2a\tau h_y^2 \gamma\right) u_{i,N_y}^{k+\frac{1}{2}} - a\tau h_y^2 \gamma u_{i+1,N_y}^{k+\frac{1}{2}} - 2a\tau h_x^2 h_y \psi_1\left(x_i, t^{k+1}\right) = 2a\tau h_x^2 u_{i,N_y-1}^{k+1} - (2a\tau h_x^2 + (1+\gamma$$

Двухточечная аппроксимация первого порядка

Впрочем можно аппроксировать граничное условие в обоих случаях двухточечной аппроксимацией первого порядка.

Тогда очевидны формулы для определения значений функции при $y = l_y$ в обоих направлениях прогонки.

Вариант 5:

dU/dt = (dU)^2 / (dx)^2 + (du)^2 / (dy) ^ 2 - x * y * sin(t)

U(0, y, t) = 0

U(1, y, t) = y * cos(t)

U(x, 0, t) = 0

U(x, 1, t) = x * cos(t)

U(x, y, 0) = x * y

Аналитическое решение:

U(x, y, t) = x * y * cos(t)

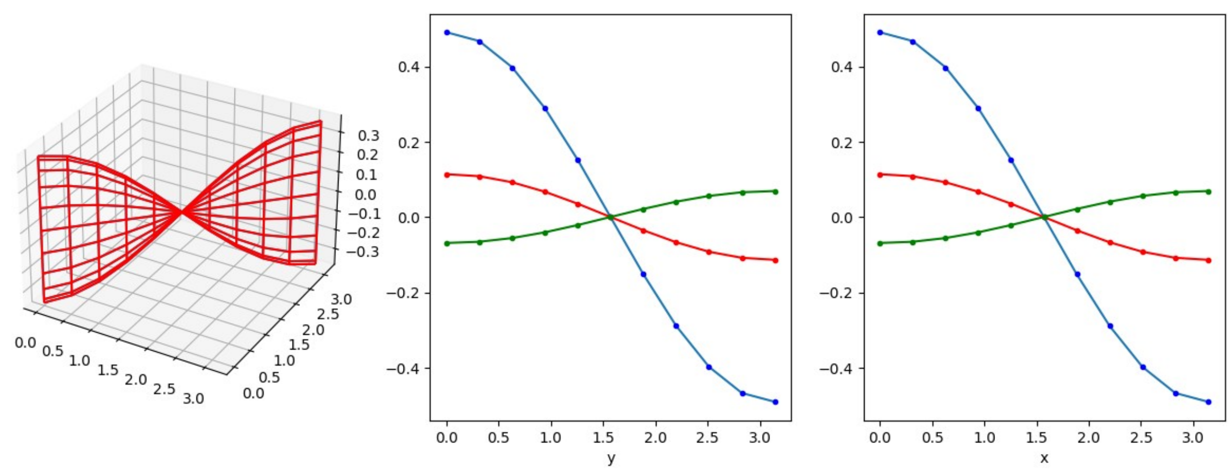
```
import numpy as np
import matplotlib.pyplot as plt

nu1 = 1
nu2 = 1
lx = np.pi
ly = np.pi
nx = 11
ny = 11
tau = 0.01
nt = 100

hx = lx / (nx - 1)
hy = ly / (ny - 1)
x = [i*hx for i in range(nx)]
y = [i*hy for i in range(ny)]

def U_0_y_t(y,t):
    return np.cos(nu2*y)*np.exp(-(nu1**2 + nu2**2)*t)

def U_l_y_t(y,t):
    return (-1)**nu1*np.cos(nu2*y)*np.exp(-(nu1**2 + nu2**2)*t)

def U_x_0_t(x,t):
    return np.cos(nu1*x)*np.exp(-(nu1**2 + nu2**2)*t)

def U_x_l_t(x,t):
```

```
    return (-1)**nu2*np.cos(nu1*x)*np.exp(-(nu1**2 + nu2**2)*t)

def U_x_y(x,y):
    return np.cos(nu1*x)*np.cos(nu2*y)

def U_ans(x,y,t):
    return np.cos(nu1*x)*np.cos(nu2*y)*np.exp(-(nu1**2 + nu2**2)*t)

def trid_alg(matrix, vec):
    s = len(vec)
    ans = np.zeros(s)
    p = np.zeros(s)
    q = np.zeros(s)

    p[0] = -matrix[0][1] / matrix[0][0]
    q[0] = vec[0] / matrix[0][0]

    for i in range(1, s - 1):
        p[i] = -matrix[i][i + 1] / (matrix[i][i] + matrix[i][i - 1] *
p[i - 1])
        q[i] = (vec[i] - matrix[i][i - 1] * q[i - 1]) / (
            matrix[i][i] + matrix[i][i - 1] * p[i - 1]
        )

    p[s - 1] = 0
    q[s - 1] = (vec[s - 1] - matrix[s - 1][s - 2] * q[s - 2]) / (
        matrix[s - 1][s - 1] + matrix[s - 1][s - 2] * p[s - 2]
    )

    ans[s - 1] = q[s - 1]

    for i in range(s - 2, -1, -1):
        ans[i] = p[i] * ans[i + 1] + q[i]

    return

ans = np.zeros((nt,nx,ny))
for i in range(nx):
    for j in range(ny):
        ans[0][i][j] = U_x_y(hx*i, hy*j)

for k in range(nt - 1):
    Uk12 = np.zeros((nx,ny))
    for i in range(nx):
        ans[k+1][i][0] =  U_x_0_t(i*hx,tau*(k+1))
        ans[k+1][i][-1] =  U_x_l_t(i*hx,tau*(k+1))
        Uk12[i][0] = U_x_0_t(i*hx,tau*k + tau/2)
        Uk12[i][-1] = U_x_l_t(i*hx,tau*k + tau/2)
    for i in range(ny):
        ans[k+1][0][i] =  U_0_y_t(i*hy,tau*(k+1))
        ans[k+1][-1][i] =  U_l_y_t(i*hy,tau*(k+1))
```

```python
        Uk12[0][i] =  U_0_y_t(i*hy,tau*k + tau/2)
        Uk12[-1][i] =  U_l_y_t(i*hy,tau*k + tau/2)


    for j in range(1, ny-1):
        solve_mat = np.zeros((nx,nx))
        vec = np.zeros(nx)
        denominator = 2*tau*hy**2 + 2*hy**2*hx**2
        solve_mat[0][0] = 1
        solve_mat[-1][-1] = 1
        vec[0] = U_0_y_t(hy*j, tau*k+tau/2)
        vec[-1] = U_l_y_t(hy*j, tau*k+tau/2)
        for i in range(1,nx-1):
            solve_mat[i][i] = denominator
            solve_mat[i][i-1] = -tau*hy**2
            solve_mat[i][i+1] = -tau*hy**2
            vec[i] = (
                ans[k][i][j+1]*tau*hx**2
                + ans[k][i][j-1]*tau*hx**2
                + ans[k][i][j]*(2*hx**2*hy**2 - 2*tau*hx**2)
                )
        x_solve = np.linalg.solve(solve_mat, vec)
        Uk12[:,j] = x_solve

    for i in range(1, nx-1):
        solve_mat = np.zeros((ny,ny))
        vec = np.zeros(ny)
        denominator = 2*tau*hx**2 + 2*hy**2*hx**2
        solve_mat[0][0] = 1
        solve_mat[-1][-1] = 1
        vec[0] = U_x_0_t(hx*i, tau*(k+1))
        vec[-1] = U_x_l_t(hx*i, tau*(k+1))
        for j in range(1,ny-1):
            solve_mat[j][j] = denominator
            solve_mat[j][j-1] = -tau*hx**2
            solve_mat[j][j+1] = -tau*hx**2
            vec[j] = (
                Uk12[i+1][j]*tau*hy**2
                + Uk12[i-1][j]*tau*hy**2
                + Uk12[i][j]*(2*hx**2*hy**2 - 2*tau*hy**2)
                )
        y_solve = np.linalg.solve(solve_mat, vec)
        ans[k+1][i,:] = y_solve

z_ans = np.zeros((nt,nx,ny))
for k in range(nt):
    for i in range(nx):
        for j in range(ny):
            z_ans[k][i][j] = U_ans(hx*i, hy*j, tau*k)
plt.rcParams['figure.figsize'] = [15, 5]
```

```
fig = plt.figure()
ax_3d = fig.add_subplot(1,3,1, projection='3d')
ax_3d.plot_wireframe(x, y, ans[nt//2])
ax_3d.plot_wireframe(x, y, z_ans[nt//2], color = 'r')
axx = fig.add_subplot(1,3,2)
axx.plot(y, ans[nt // 4][nx // 4])
axx.plot(y, z_ans[nt // 4][nx // 4], '.b')
# axx.plot(y, ans[5][nx // 4])
# axx.plot(y, z_ans[5][nx // 4], '.b')
axx.plot(y, ans[nt // 4 * 2][nx // 4 * 2], 'r')
axx.plot(y, z_ans[nt // 4 * 2][nx // 4 * 2], '.r')
axx.plot(y, ans[nt // 4 * 3][nx // 4 * 3], 'g')
axx.plot(y, z_ans[nt // 4 * 3][nx // 4 * 3], '.g')
plt.xlabel('y')
axy = fig.add_subplot(1,3,3)
axy.plot(x, ans[nt // 4][:, ny // 4])
axy.plot(x, z_ans[nt // 4][:, ny // 4], '.b')
axy.plot(x, ans[nt // 4 * 2][:, ny // 4 * 2], 'r')
axy.plot(x, z_ans[nt // 4 * 2][:, ny // 4 * 2], '.r')
axy.plot(x, ans[nt // 4 * 3][:, ny // 4 * 3], 'g')
axy.plot(x, z_ans[nt // 4 * 3][:, ny // 4 * 3], '.g')
plt.xlabel('x')
```

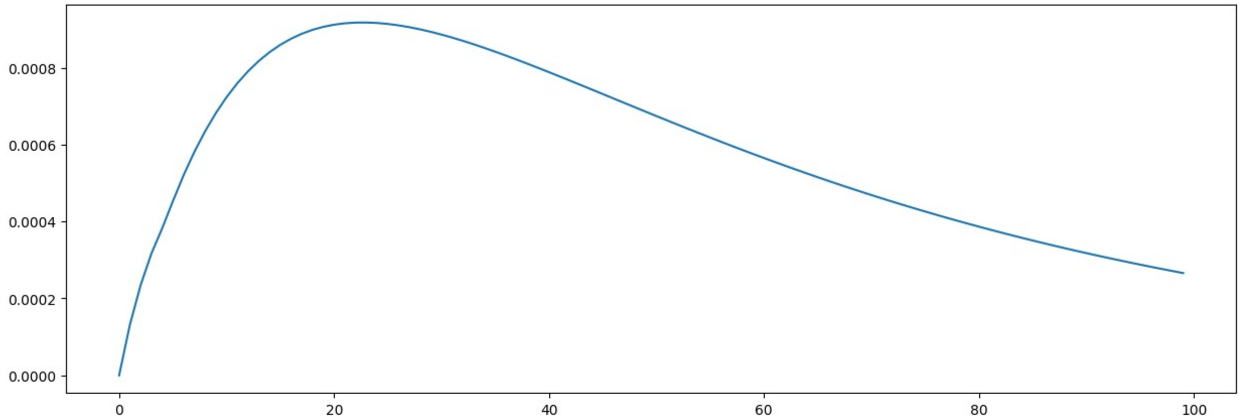
```
Text(0.5, 0, 'x')
```



Time check

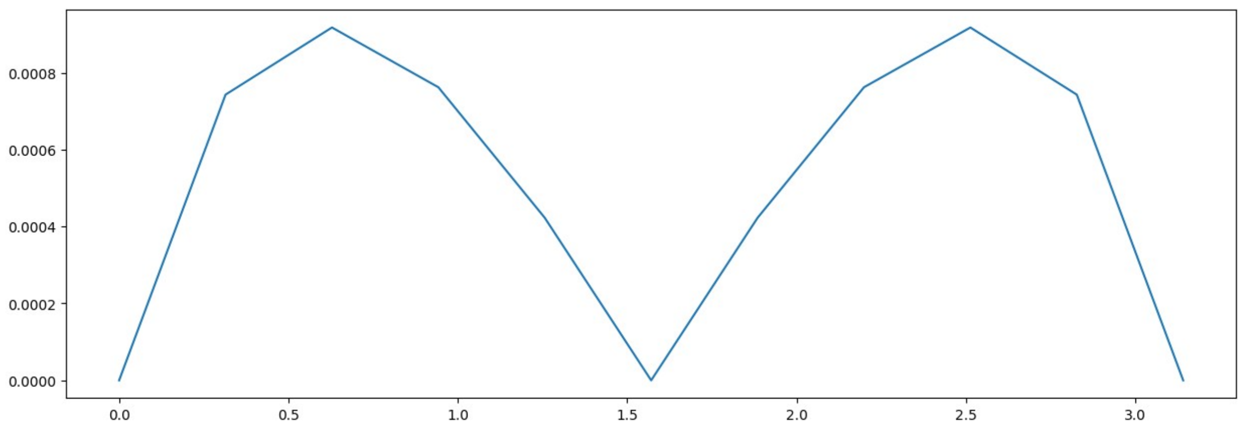
```
plt.plot(range(nt), [np.max(np.abs(ans-z_ans)[i]) for i in range(nt)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b15b74850>]
```



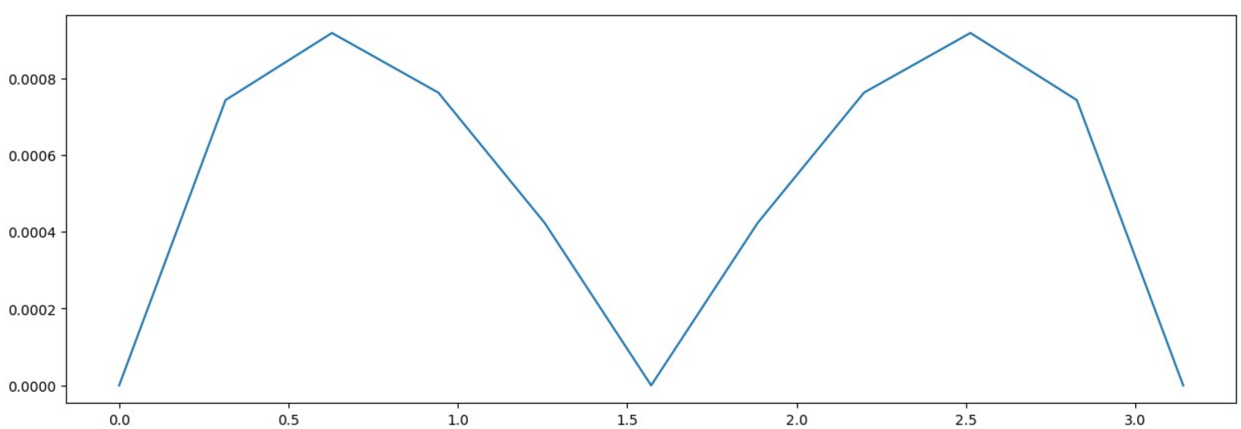

```
plt.plot(y, [np.max(np.abs(ans-z_ans)[:,:,i]) for i in range(ny)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b15ada3d0>]
```




```
plt.plot(x, [np.max(np.abs(ans-z_ans)[:,i,:]) for i in range(nx)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b15aba3d0>]
```

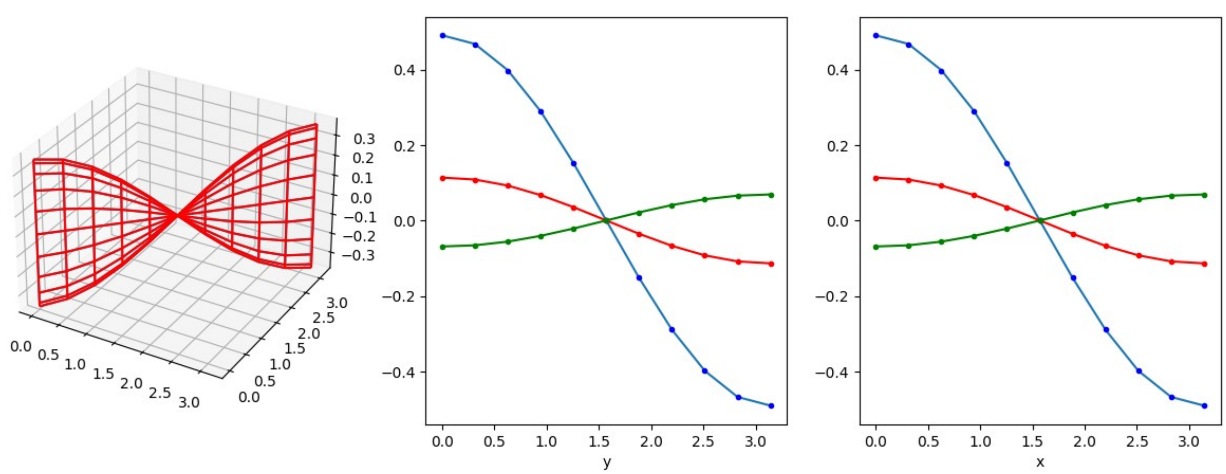


Метод дробных шагов

```
ans = np.zeros((nt,nx,ny))
for i in range(nx):
    for j in range(ny):
        ans[0][i][j] = U_x_y(hx*i, hy*j)

for k in range(nt - 1):
    Uk12 = np.zeros((nx,ny))
    for i in range(nx):
        ans[k+1][i][0] =  U_x_0_t(i*hx,tau*(k+1))
        ans[k+1][i][-1] =  U_x_l_t(i*hx,tau*(k+1))
        Uk12[i][0] = U_x_0_t(i*hx,tau*k + tau/2)
        Uk12[i][-1] = U_x_l_t(i*hx,tau*k + tau/2)
    for i in range(ny):
        ans[k+1][0][i] =  U_0_y_t(i*hy,tau*(k+1))
        ans[k+1][-1][i] =  U_l_y_t(i*hy,tau*(k+1))
        Uk12[0][i] =  U_0_y_t(i*hy,tau*k + tau/2)
        Uk12[-1][i] =  U_l_y_t(i*hy,tau*k + tau/2)


    for j in range(1, ny-1):
        solve_mat = np.zeros((nx,nx))
        vec = np.zeros(nx)
        denominator = 2*tau + hx**2
        solve_mat[0][0] = 1
        solve_mat[-1][-1] = 1
        vec[0] = U_0_y_t(hy*j, tau*k+tau/2)
        vec[-1] = U_l_y_t(hy*j, tau*k+tau/2)
        for i in range(1,nx-1):
            solve_mat[i][i] = denominator
            solve_mat[i][i-1] = -tau
            solve_mat[i][i+1] = -tau
            vec[i] = ans[k][i][j]*hx**2
        x_solve = np.linalg.solve(solve_mat, vec)
        Uk12[:,j] = x_solve

    for i in range(1, nx-1):
        solve_mat = np.zeros((ny,ny))
        vec = np.zeros(ny)
        denominator = 2*tau + hy**2
        solve_mat[0][0] = 1
        solve_mat[-1][-1] = 1
        vec[0] = U_x_0_t(hx*i, tau*(k+1))
        vec[-1] = U_x_l_t(hx*i, tau*(k+1))
        for j in range(1,ny-1):
            solve_mat[j][j] = denominator
            solve_mat[j][j-1] = -tau
            solve_mat[j][j+1] = -tau
            vec[j] = Uk12[i][j]*hy**2
```

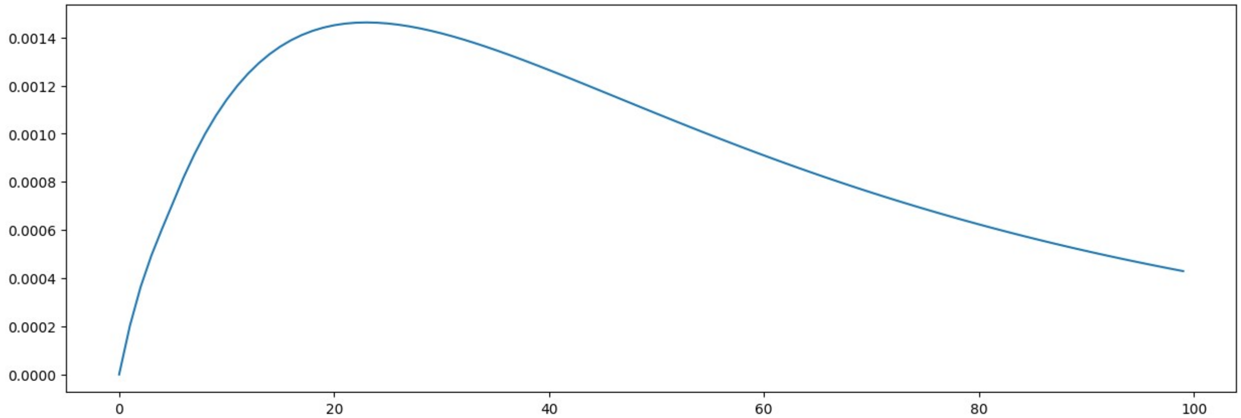
```
        y_solve = np.linalg.solve(solve_mat, vec)
        ans[k+1][i,:] = y_solve

z_ans = np.zeros((nt,nx,ny))
for k in range(nt):
    for i in range(nx):
        for j in range(ny):
            z_ans[k][i][j] = U_ans(hx*i, hy*j, tau*k)
plt.rcParams['figure.figsize'] = [15, 5]
fig = plt.figure()
ax_3d = fig.add_subplot(1,3,1, projection='3d')
ax_3d.plot_wireframe(x, y, ans[nt//2])
ax_3d.plot_wireframe(x, y, z_ans[nt//2], color = 'r')
axx = fig.add_subplot(1,3,2)
axx.plot(y, ans[nt // 4][nx // 4])
axx.plot(y, z_ans[nt // 4][nx // 4], '.b')
# axx.plot(y, ans[5][nx // 4])
# axx.plot(y, z_ans[5][nx // 4], '.b')
axx.plot(y, ans[nt // 4 * 2][nx // 4 * 2], 'r')
axx.plot(y, z_ans[nt // 4 * 2][nx // 4 * 2], '.r')
axx.plot(y, ans[nt // 4 * 3][nx // 4 * 3], 'g')
axx.plot(y, z_ans[nt // 4 * 3][nx // 4 * 3], '.g')
plt.xlabel('y')
axy = fig.add_subplot(1,3,3)
axy.plot(x, ans[nt // 4][:, ny // 4])
axy.plot(x, z_ans[nt // 4][:, ny // 4], '.b')
axy.plot(x, ans[nt // 4 * 2][:, ny // 4 * 2], 'r')
axy.plot(x, z_ans[nt // 4 * 2][:, ny // 4 * 2], '.r')
axy.plot(x, ans[nt // 4 * 3][:, ny // 4 * 3], 'g')
axy.plot(x, z_ans[nt // 4 * 3][:, ny // 4 * 3], '.g')
plt.xlabel('x')
```

```
Text(0.5, 0, 'x')
```



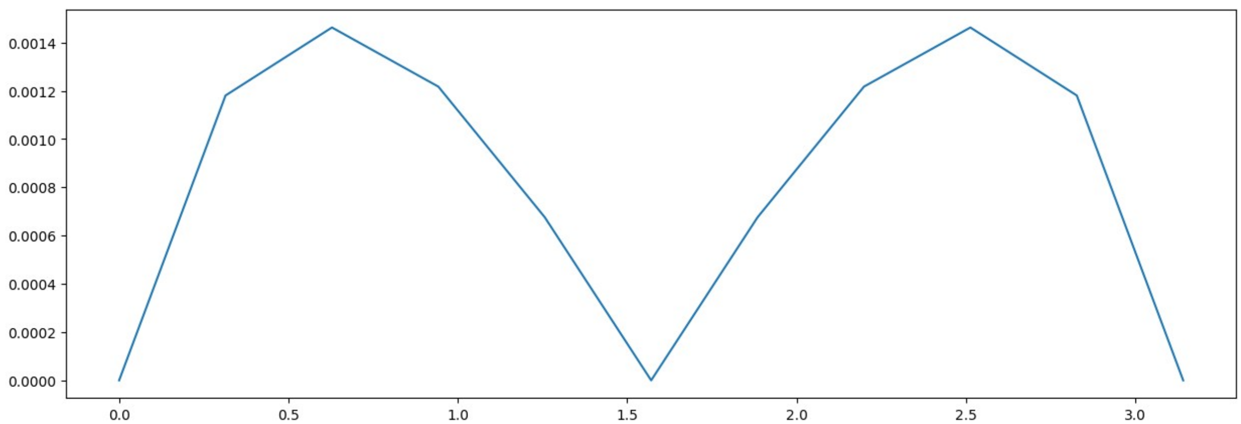

```
plt.plot(range(nt), [np.max(np.abs(ans-z_ans)[i]) for i in range(nt)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b155c4340>]
```



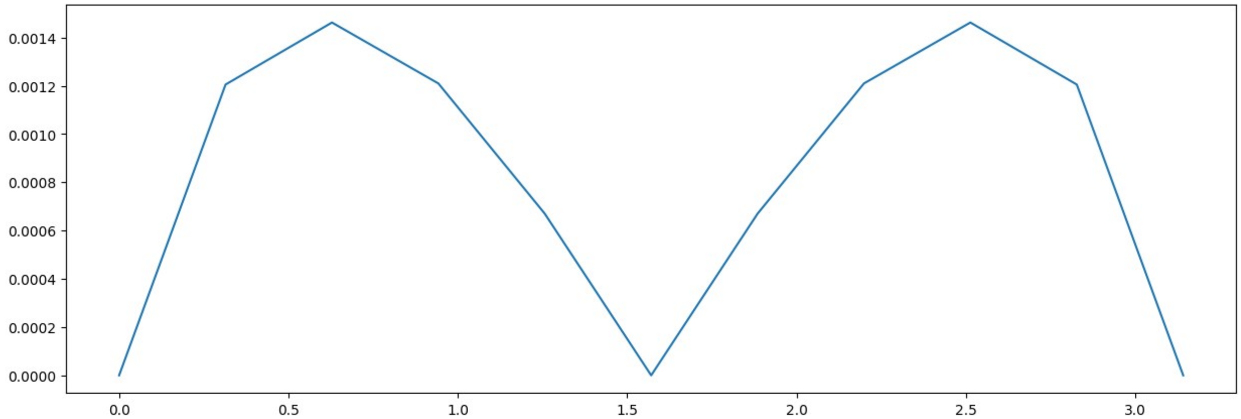

```
plt.plot(y, [np.max(np.abs(ans-z_ans)[:,:,i]) for i in range(ny)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b15525970>]
```




```
plt.plot(x, [np.max(np.abs(ans-z_ans)[:,i,:]) for i in range(nx)])
```

```
[<matplotlib.lines.Line2D at 0x7f1b1550caf0>]
```


0.0014
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000
0.0
0.5
1.0
1.5
2.0
2.5
3.0