Пищик Е.С. М8О-406Б-19

Лабораторная работа №6

Задание: Используя явную схему крест и неявную схему, решить начально-краевую задачу для дифференциального уравнения гиперболического типа. Аппроксимацию второго начального условия произвести с первым и со вторым порядком. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. 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 и h.

Вариант №2

Уравнение:

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

Граничные условия:

$$\begin{cases} u_x(0,\ t) - u(0,\ t) = 0 \\ u_x(\pi,\ t) - u(\pi,\ t) = 0 \\ u(x,\ 0) = \sin x + \cos x \\ u_t(x,\ 0) = -a(\sin x + \cos x) \end{cases}$$

Аналитическое решение:

$$u(x,t) = \sin{(x - at)} + \cos{(x + at)}$$

Файл **tools.py** содержит функции для граничных условий (**phi**, **psi0**, **psi1**) аналитического решения (**analitic_solution**), аналитической сетки (**analitic_grid**), алгоритма прогонки (**tridiagonal_matrix_algorithm**), второй производной (**second_derivative**, ошибки (**error**).

In [ ]:

```
import numpy as np
```

In [ ]:

```
def phi(x, t, a):
    return -psi0(x, t, a)
```

In [ ]:

```
def psi0(x, t, a):
    return np.sin(x) + np.cos(x)
```

In [ ]:

```
def psi1(x, t, a):
    return -a * (np.sin(x) + np.cos(x))
```

In [ ]:

```
def analitic_solution(x, t, a):
    return np.sin(x - a * t) + np.cos(x + a * t)
```

In [ ]:

```
def analitic_grid(x, t, a):
    grid = np.zeros((len(t), len(x)))

    for i in range(len(t)):
        for j in range(len(x)):
            grid[i, j] = analitic_solution(x[j], t[i], a)
    return grid
```

In [ ]:

```
def second_derivative(lst, step):
    der = np.zeros(lst.shape)
    for i in range(1, len(lst) - 1):
        der[i] = (lst[i - 1] - 2.0 * lst[i] + lst[i + 1]) / (step * step)
    return der
```

In [ ]:

```
def tridiagonal_matrix_algorithm(matrix, target):
    n = matrix.shape[1]

    p = np.zeros(n)
    q = np.zeros(n)

    p[0] = -matrix[0, 1] / matrix[0, 0]
    q[0] = target[0] / matrix[0, 0]

    for i in range(1, n - 1):
        den = matrix[i, i] + matrix[i, i - 1] * p[i - 1]
        p[i] = -matrix[i, i + 1] / den

        den = matrix[i, i] + matrix[i, i - 1] * p[i - 1]
        q[i] = (target[i] - matrix[i, i - 1] * q[i - 1]) / den

    p[-1] = 0

    res = np.zeros(n + 1)

    for i in range(n - 1, -1, -1):
        res[i] = p[i] * res[i + 1] + q[i]

    return res[:-1]
```

In [ ]:

```
def error(approx, analytic):
    errors = []

    for v1, v2 in zip(approx, analytic):
        errors.append(np.mean(np.abs(v1 - v2)))

    return np.array(errors)
```

Файл **plot.py** содержит функцию для отрисовки графиков (**plot**).

In [ ]:

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

In [ ]:

```
def plot(approx, analytic, x, t, err, filenames):
    fig = plt.figure(figsize=plt.figaspect(0.7))
    x_mesh, t_mesh = np.meshgrid(x, t)

    ax = fig.add_subplot(1, 2, 1, projection='3d')

    plt.title('approx')
    ax.set_xlabel('x', fontsize=14)
    ax.set_ylabel('t', fontsize=14)
    ax.set_zlabel('u', fontsize=14)

    ax.plot_surface(x_mesh,
                    t_mesh,
                    approx,
                    cmap=cm.BrBG)

    ax = fig.add_subplot(1, 2, 2, projection='3d')

    plt.title('analytic')
    ax.set_xlabel('x', fontsize=14)
    ax.set_ylabel('t', fontsize=14)
    ax.set_zlabel('u', fontsize=14)

    ax.plot_surface(x_mesh,
                    t_mesh,
                    analytic,
                    cmap=cm.BrBG)

    plt.savefig(filenames[0])

    fig = plt.figure(figsize=plt.figaspect(0.7))

    plt.title('error')
    plt.xlabel('t')
    plt.ylabel('error')
    plt.plot(t, err, color='green')

    plt.savefig(filenames[1])
```

Файл **explicit.py** содержит реализацию явной схемы крест.

In [ ]:

```
import numpy as np

from tools import phi, psi0, psi1, second_derivative, error, analitic_grid
from plot import plot
```

In [ ]:

```
def explicit(params):
    psi0 = params['psi0']
    psi1 = params['psi1']
    phi = params['phi']
    a = params['a']
    h = params['h']
    tau = params['tau']
    lhs = params['lhs']
    rhs = params['rhs']
    t_last = params['t_last']
    approx = params['approx']

    sigma = (a * tau * tau) / (h * h)

    if sigma > 1.0:
        raise ValueError(f'Явная схема не устойчива sigma = {sigma}')

    x = np.arange(lhs, rhs + h / 2.0, step=h)
    t = np.arange(0, t_last + tau / 2.0, step=tau)
    u = np.zeros((len(t), len(x)))

    u[0] = psi0(x, 0, 0)
    u[1] = u[0] + tau * psi1(x, 0, a) + tau * tau * phi(x, 0, 0) / 2.0

    for i in range(1, len(t) - 1):
        val1 = 2.0 * u[i]
        val2 = u[i - 1]
        val3 = tau**2 * a * second_derivative(u[i], step=h)
        u[i + 1] = val1 - val2 + val3

        if approx == 21:
            u[i + 1, 0] = u[i + 1, 1] / (1.0 + h)
            u[i + 1, -1] = u[i + 1, -2] / (1.0 - h)
        elif approx == 32:
            u[i + 1, 0] = (4.0 * u[i + 1][1] - u[i + 1][2]) / (3.0 + 2.0 * h)
            den = (3.0 - 2.0 * h)

            u[i + 1, -1] = (4.0 * u[i + 1][-2] - u[i + 1][-3]) / den
        elif approx == 22:
            add1 = sigma * (2.0 * u[i][1] - (2.0 + 2.0 * h) * u[i][0])
            add2 = 2.0 * u[i][0] - u[i - 1][0]
            u[i + 1, 0] = add1 + add2

            add1 = sigma * (2.0 * u[i][-2] + (2.0 * h - 2.0) * u[i][-1])
            add2 = 2.0 * u[i][-1] - u[i - 1][-1]
            u[i + 1, -1] = add1 + add2

    return u
```

In [ ]:

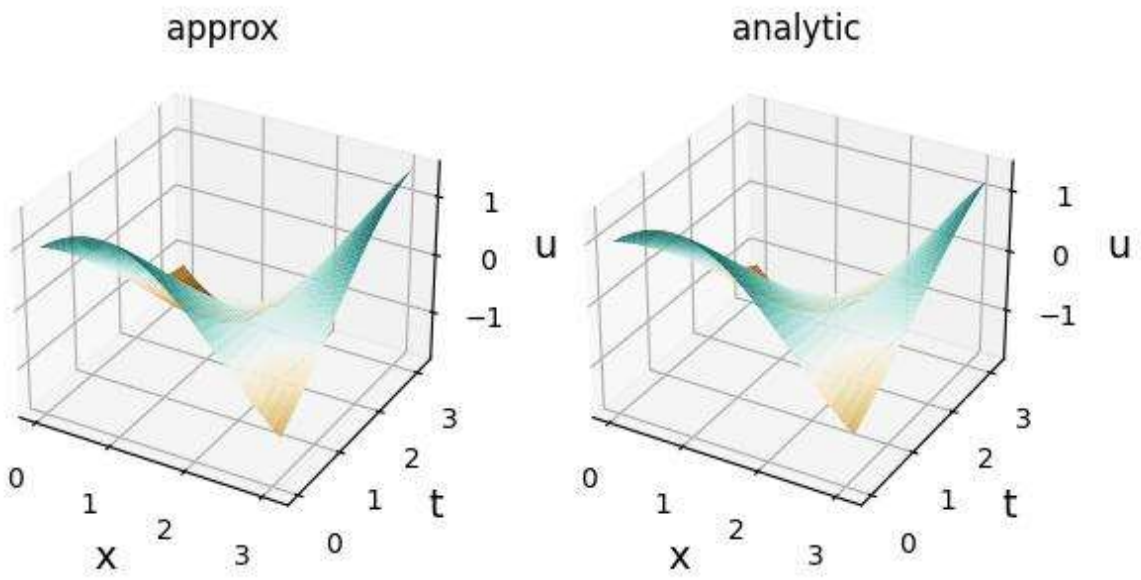
```
def main(params):
    a = params['a']
    h = params['h']
    tau = params['tau']
    lhs = params['lhs']
    rhs = params['rhs']
    t_last = params['t_last']
    approx_type = params['approx']

    x = np.arange(0, rhs + h / 2.0, step=h)
    t = np.arange(0, t_last + tau / 2.0, step=tau)

    analytic = analitic_grid(x, t, a)
    approx = explicit(params)
    err = error(approx, analytic)

    filename_u = f'images/explicit_function_a={a}_h={h}_tau={tau}_lhs={lhs}'
    filename_u += f'_rhs={rhs}_t_last={t_last}_approx={approx_type}.jpg'

    filename_e = f'images/explicit_error_a={a}_h={h}_tau={tau}_lhs={lhs}'
    filename_e += f'_rhs={rhs}_t_last={t_last}_approx={approx_type}.jpg'

    filenames = (filename_u, filename_e)

    plot(approx, analytic, x, t, err, filenames)
```

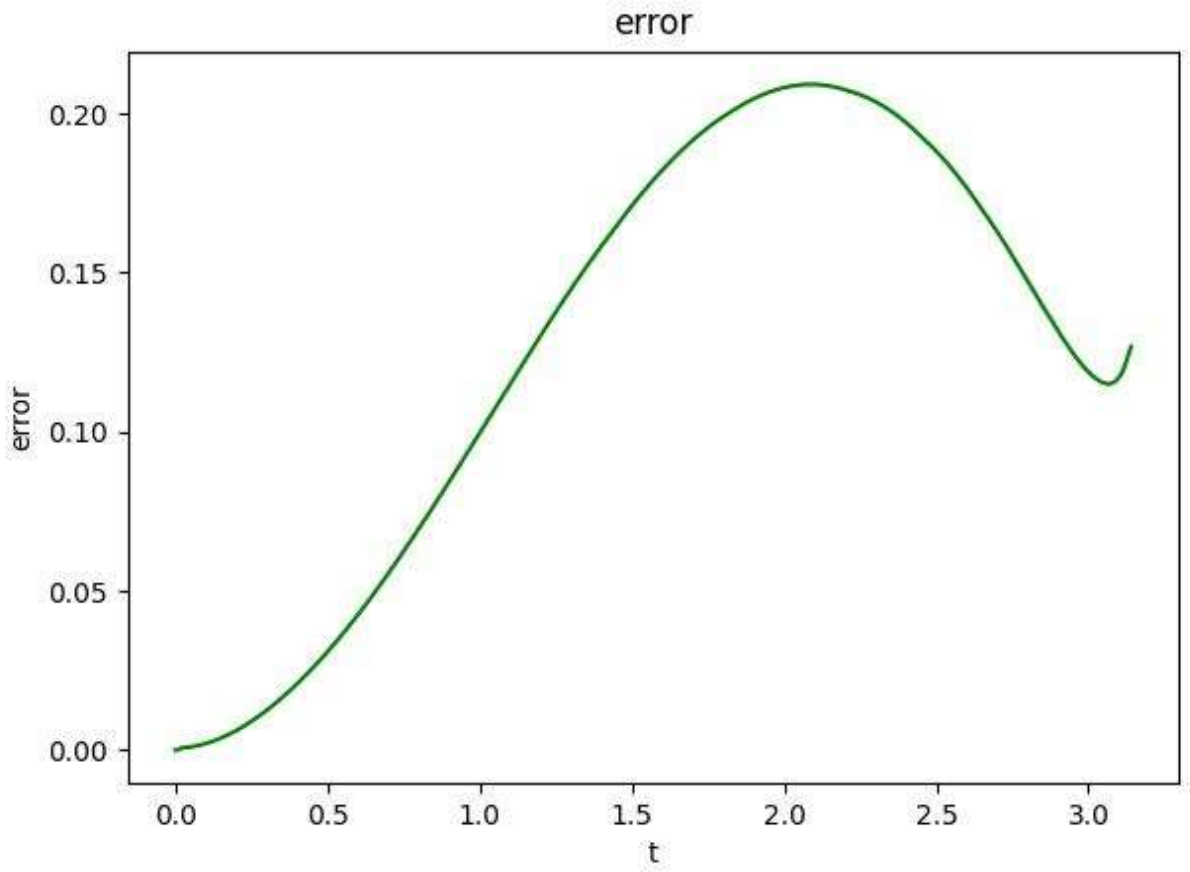
In [ ]:

```
if __name__ == '__main__':
    run1 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.6,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 21,
    }

    run2 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.6,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 32,
    }

    run3 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.6,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 22,
    }

    main(run1)
    main(run2)
    main(run3)
```

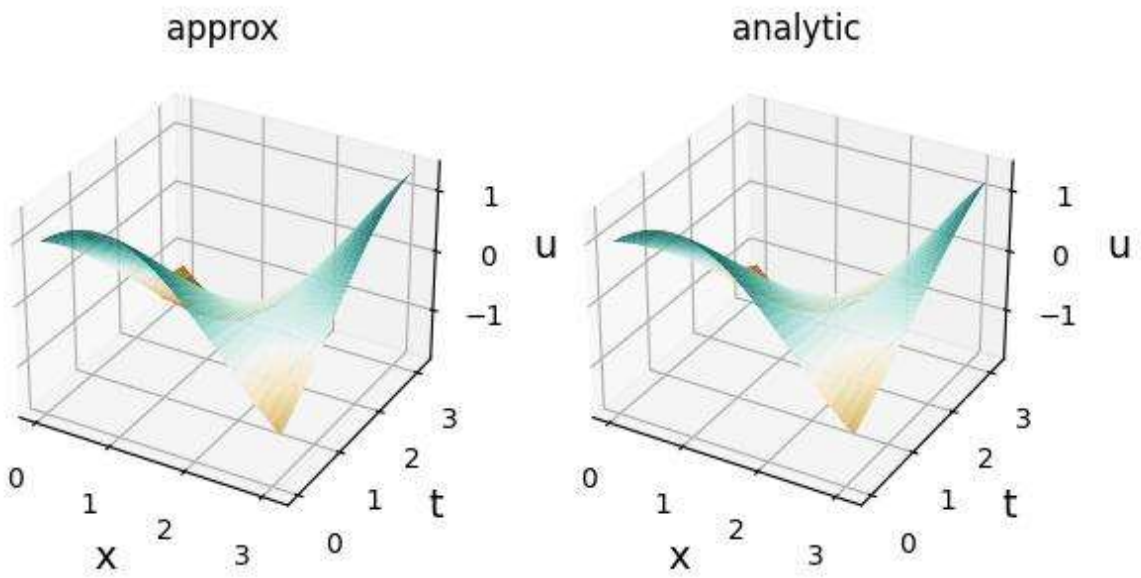
Графики сравнения аналитического решения и численного метода, значения ошибки для явной схемы крест при разных вариантах аппроксимации (**двухточечная аппроксимация с первым порядком**, **трехточечная аппроксимация со вторым порядком**, **двухточечная аппроксимация со вторым порядком**).

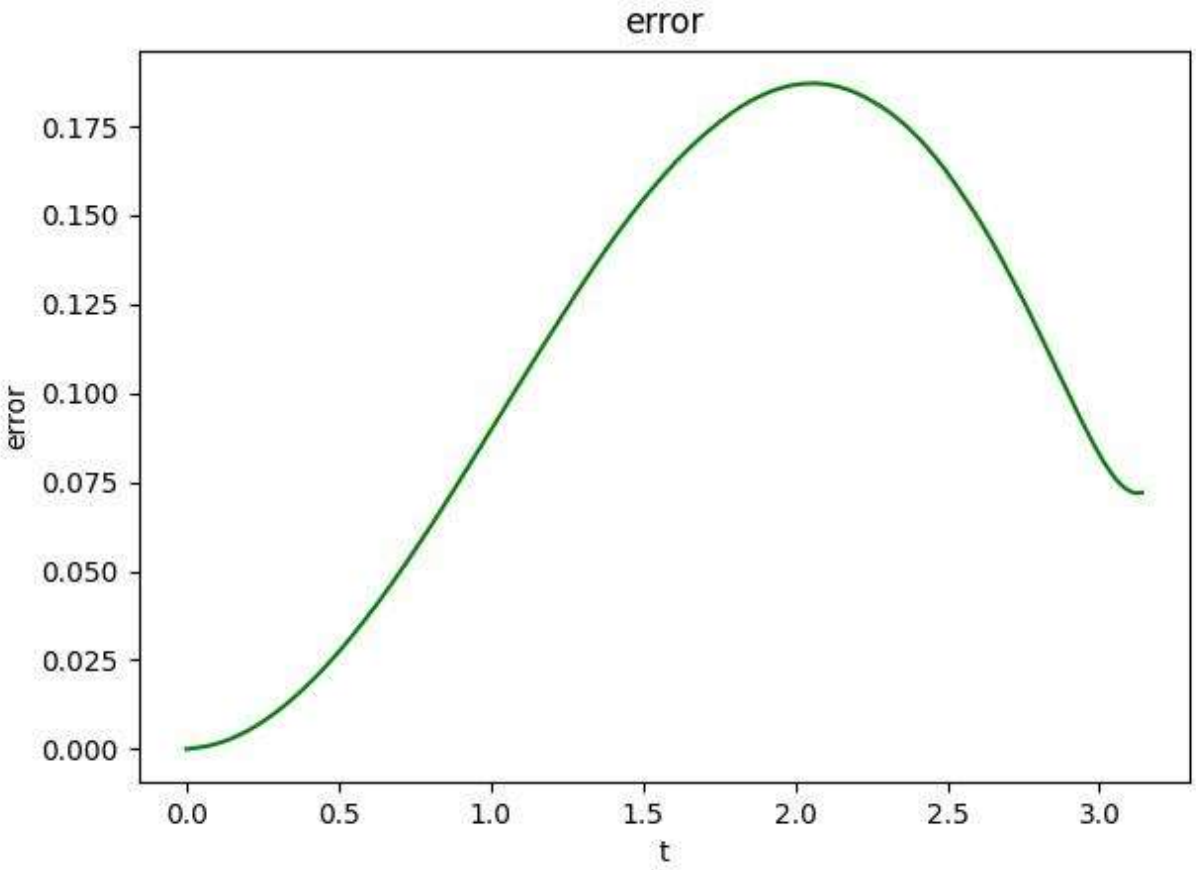
Двухточечная аппроксимация с первым порядком.


approx
analytic
u
x
t
0
1
2
3
−1

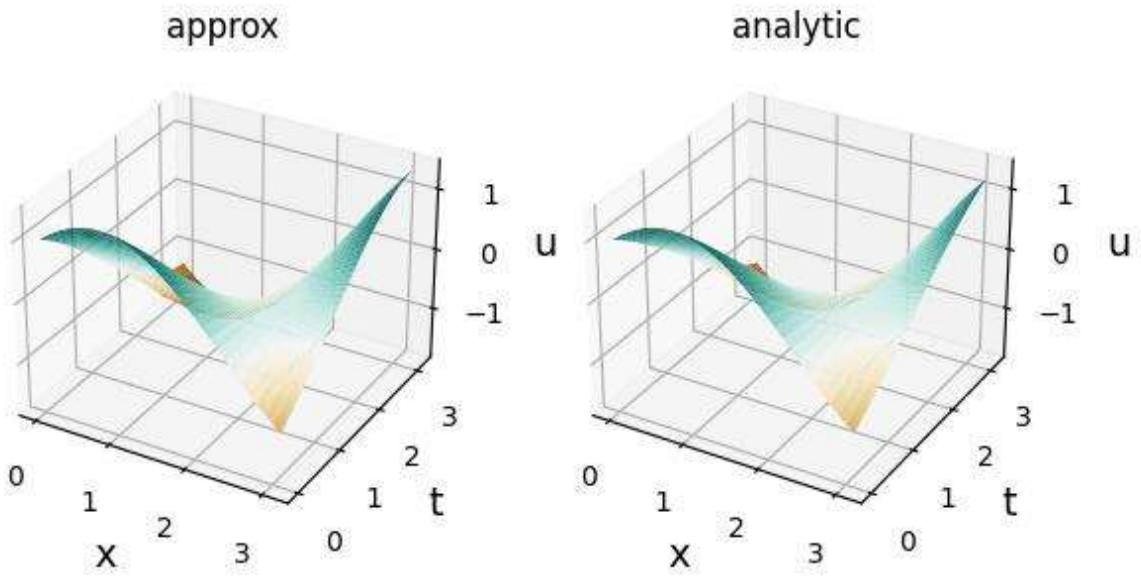


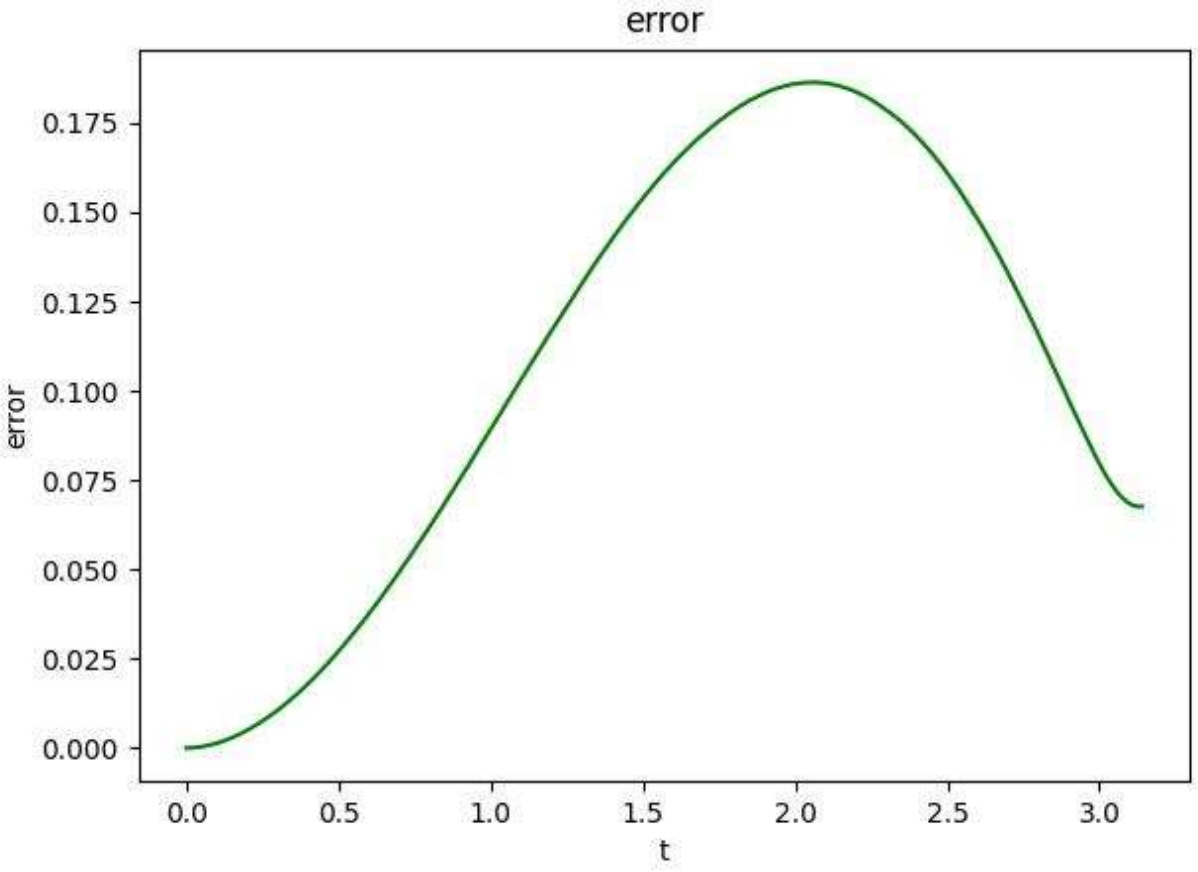
Трехточечная аппроксимация со вторым порядком.


approx
analytic
u
x
t
0
1
2
3
−1


error
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0.000
0.0
0.5
1.0
1.5
2.0
2.5
3.0
t

Двухточечная аппроксимация со вторым порядком.





Файл **implicit.py** содержит реализацию неявной схемы.

In [ ]:

```
import numpy as np

from tools import phi, psi0, psi1
from tools import tridiagonal_matrix_algorithm, error, analitic_grid

from plot import plot
```

In [ ]:

```
def implicit(params):
    psi0 = params['psi0']
    psi1 = params['psi1']
    phi = params['phi']
    a = params['a']
    h = params['h']
    tau = params['tau']
    lhs = params['lhs']
    rhs = params['rhs']
    t_last = params['t_last']
    approx = params['approx']

    sigma = (a * tau * tau) / (h * h)
    x = np.arange(lhs, rhs + h / 2.0, step=h)
    t = np.arange(0, t_last + tau / 2.0, step=tau)
    u = np.zeros((len(t), len(x)))

    u[0] = psi0(x, 0, 0)
    u[1] = u[0] + tau * psi1(x, 0, a) + tau * tau * phi(x, 0, 0) / 2.0

    for i in range(1, len(t) - 1):
        if approx == 21:
            lst0 = ((1.0 + h), -1.0, 0.0)
            lstl = (-1.0, (1.0 - h), 0.0)
        elif approx == 32:
            lst0 = (-(2.0 + 2.0 * h),
                    -(1.0 / sigma - 2.0),
                    (-2.0 * u[i][1] + u[i - 1][1]) / sigma)

            lstl = (-(1.0 / sigma - 2.0),
                    -(2.0 - 2.0 * h),
                    (-2.0 * u[i][-2] + u[i - 1][-2]) / sigma)
        elif approx == 22:
            lst0 = (-(2.0 + 2.0 * h + 1.0 / sigma),
                    2.0,
                    (-2.0 * u[i][0] + u[i - 1][0]) / sigma)

            lstl = (2.0,
                    -(2.0 - 2.0 * h + 1.0 / sigma),
                    (-2.0 * u[i][-1] + u[i - 1][-1]) / sigma)

        matrix = np.zeros((len(x), len(x)))

        matrix[0] += np.array([lst0[0], lst0[1]] + [0.0] * (len(matrix) - 2))
        target = [lst0[2]]

        for j in range(1, len(matrix) - 1):
            add1 = [0.0] * (j - 1)
            add2 = [1.0, -(2.0 + 1.0 / sigma), 1.0]
            add3 = [0.0] * (len(matrix) - j - 2)

            matrix[j] += np.array(add1 + add2 + add3)
            target += [(-2.0 * u[i][j] + u[i - 1][j]) / sigma]

        add1 = [0.0] * (len(matrix) - 2)
        add2 = [lstl[0], lstl[1]]

        matrix[-1] += np.array(add1 + add2)
```

```
        target += [lstl[2]]
        target = np.array(target)

        u[i + 1] = tridiagonal_matrix_algorithm(matrix, target)

    return u
```

In [ ]:

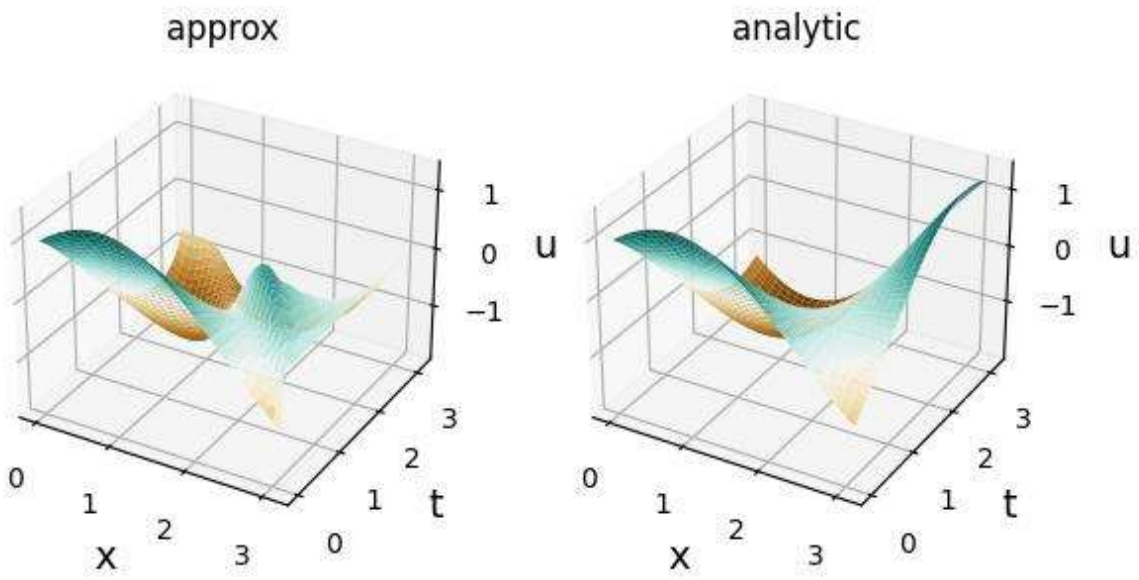
```
def main(params):
    a = params['a']
    h = params['h']
    tau = params['tau']
    lhs = params['lhs']
    rhs = params['rhs']
    t_last = params['t_last']
    approx_type = params['approx']

    x = np.arange(0, rhs + h / 2.0, step=h)
    t = np.arange(0, t_last + tau / 2.0, step=tau)

    analytic = analitic_grid(x, t, a)
    approx = implicit(params)
    err = error(approx, analytic)

    filename_u = f'images/implicit_function_a={a}_h={h}_tau={tau}_lhs={lhs}'
    filename_u += f'_rhs={rhs}_t_last={t_last}_approx={approx_type}.jpg'

    filename_e = f'images/implicit_error_a={a}_h={h}_tau={tau}_lhs={lhs}'
    filename_e += f'_rhs={rhs}_t_last={t_last}_approx={approx_type}.jpg'

    filenames = (filename_u, filename_e)

    plot(approx, analytic, x, t, err, filenames)
```

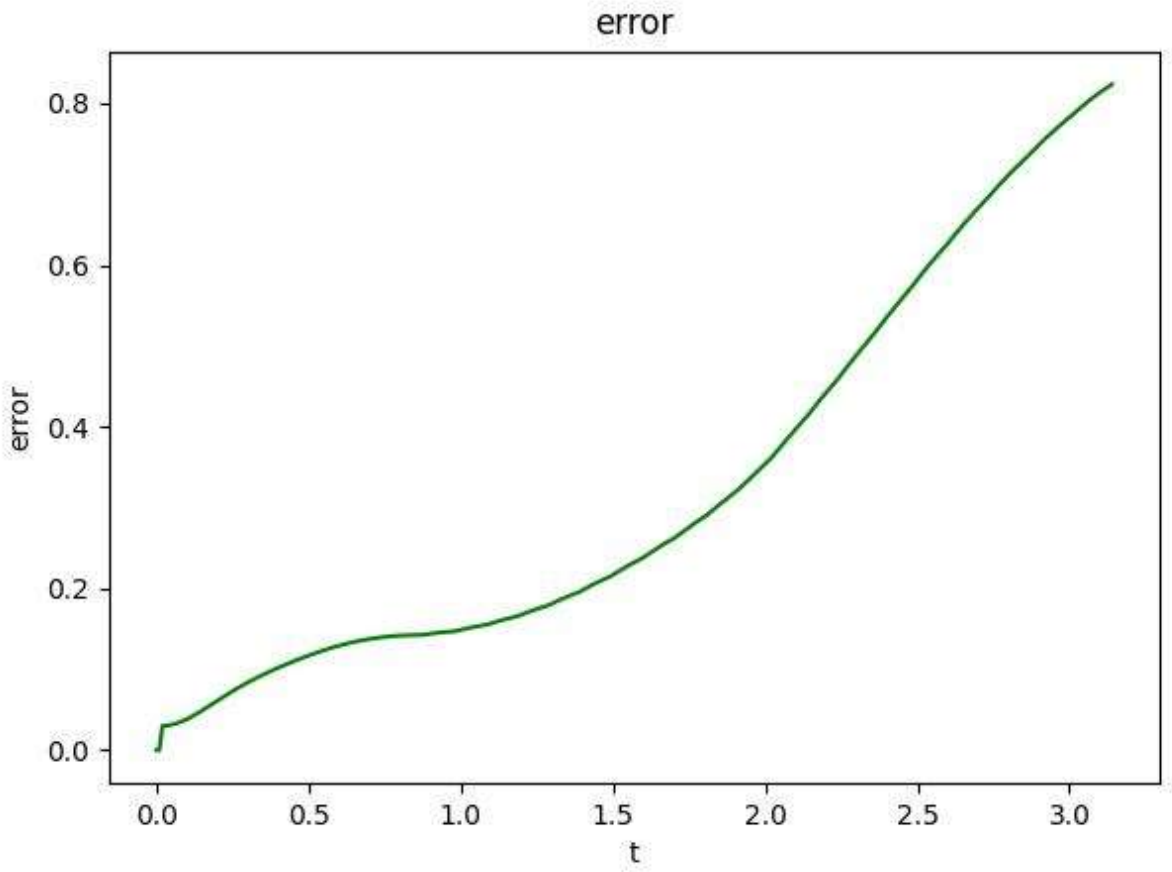
In [ ]:

```
if __name__ == '__main__':
    run1 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.9,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 21,
    }

    run2 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.9,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 32,
    }

    run3 = {
        "psi0": psi0,
        "psi1": psi1,
        "phi": phi,
        "a": 0.9,
        "h": 0.1,
        "tau": 0.01,
        "lhs": 0.0,
        "rhs": 3.14,
        "t_last": 3.14,
        "approx": 22,
    }

    main(run1)
    main(run2)
    main(run3)
```

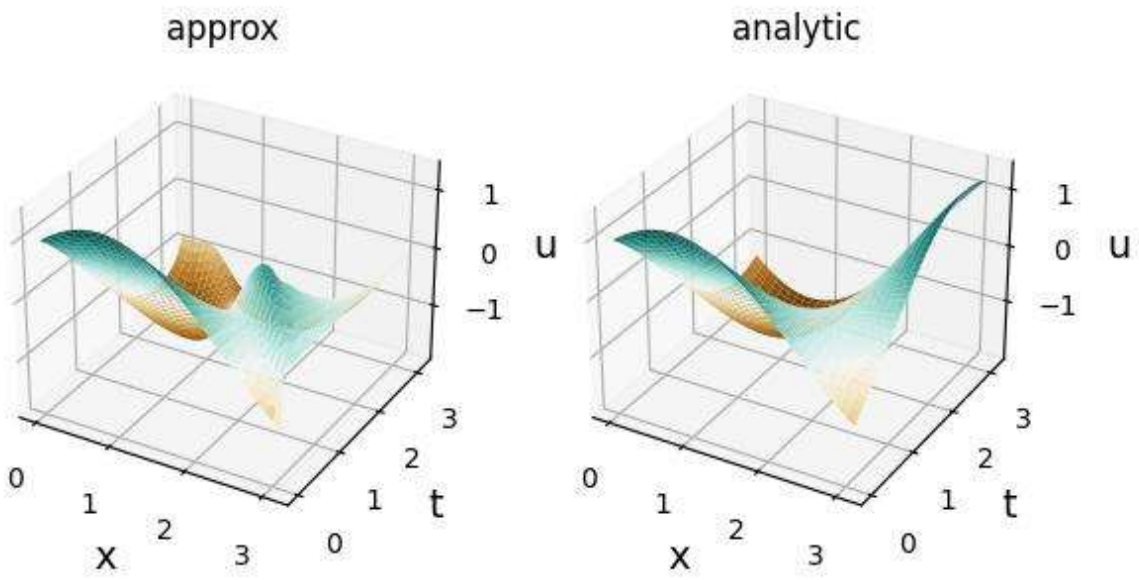
Графики сравнения аналитического решения и численного метода, значения ошибки для неявной схемы при разных вариантах аппроксимации (**двухточечная аппроксимация с первым порядком**, **трехточечная аппроксимация со вторым порядком**, **двухточечная аппроксимация со вторым порядком**).

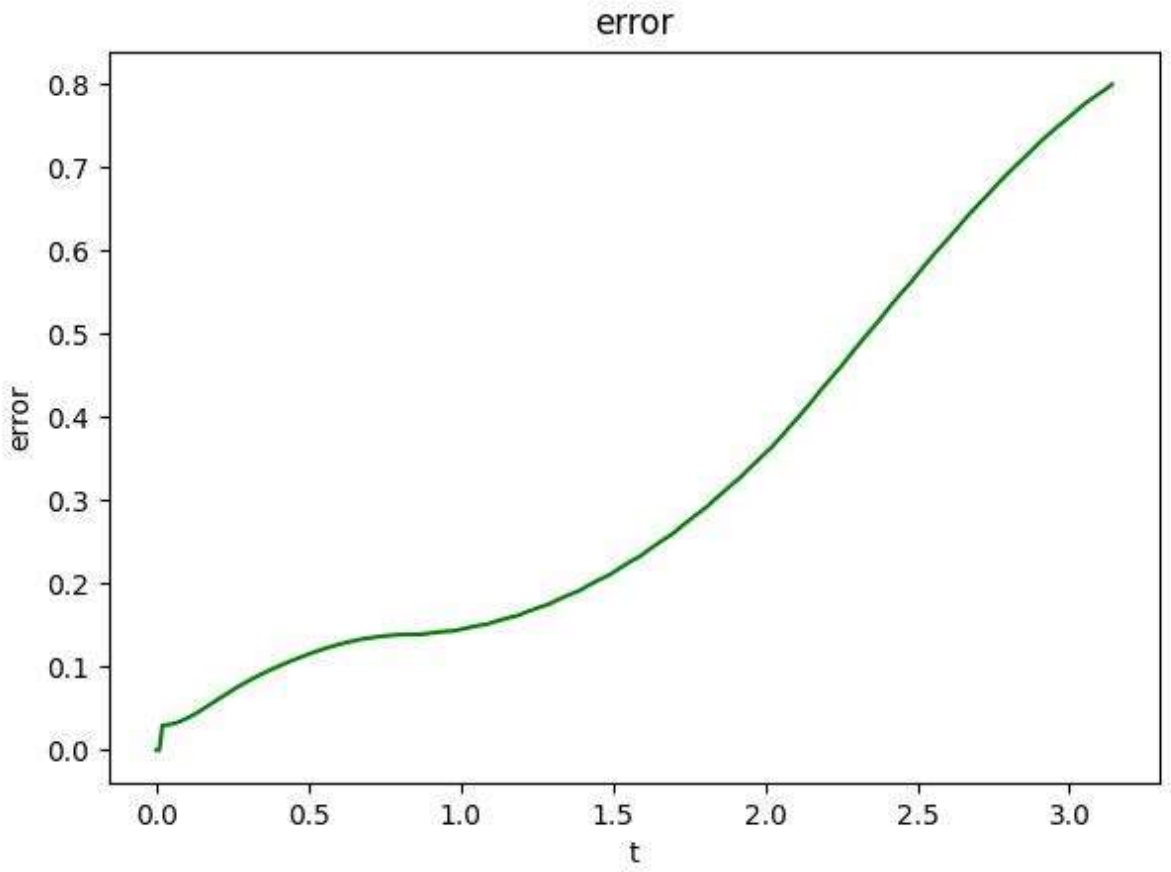
Двухточечная аппроксимация с первым порядком.


approx
analytic
u
x
t
0
1
2
3
−1

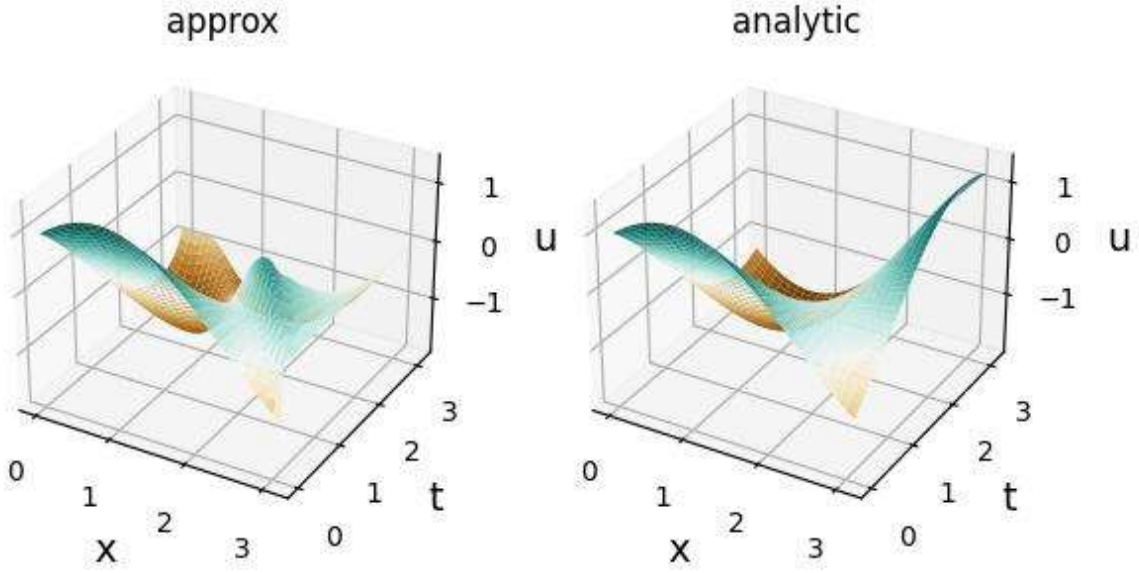


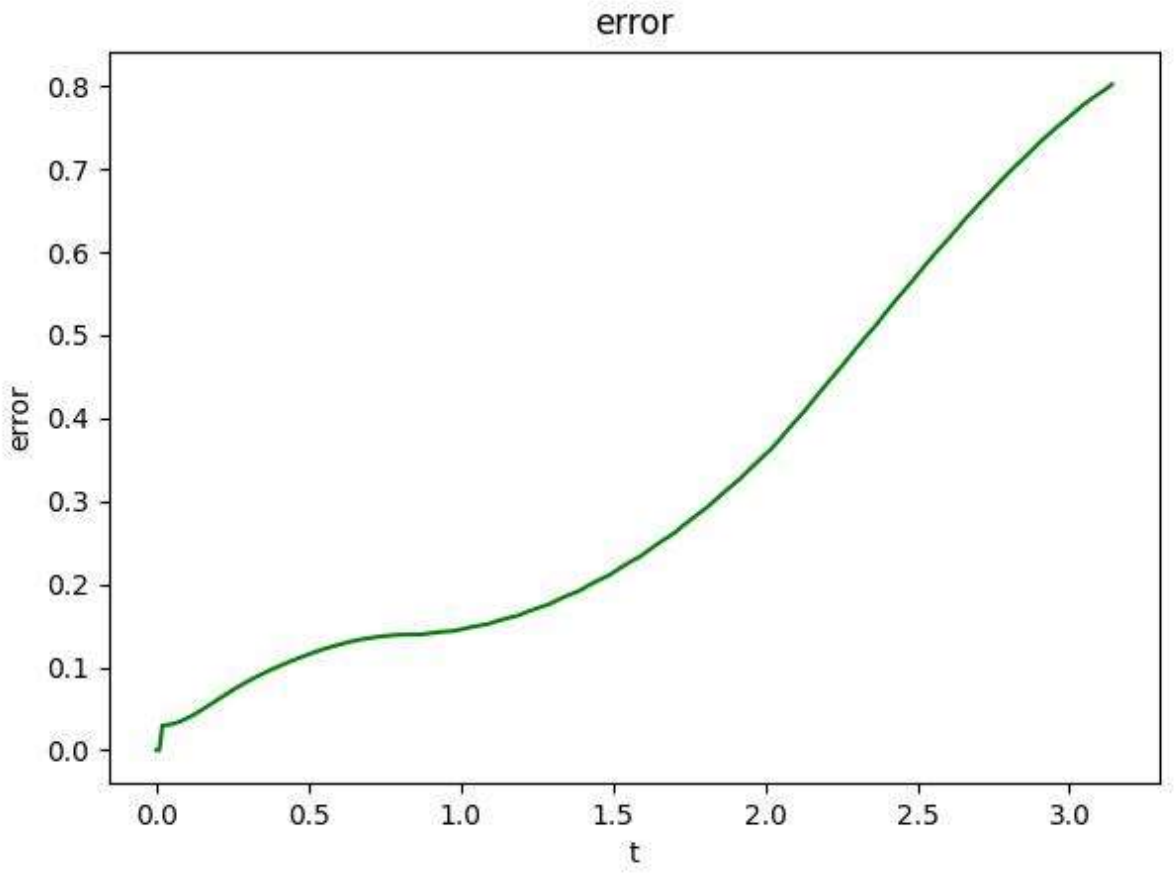
Трехточечная аппроксимация со вторым порядком.


approx
analytic
u
x
t
0
1
2
3
−1



Двухточечная аппроксимация со вторым порядком.





In [ ]: