

Лабораторная работа №5

Задание: Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. 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:

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

Границные условия:

$$\begin{cases} u'_x(0, t) = \phi_0(t) = 0 \\ u'_x(1, t) = \phi_l(t) = 1 \\ u(x, 0) = x + \sin \pi x \end{cases}$$

Аналитическое решение:

$$u(x, t) = x + e^{-\pi^2 at} \sin \pi x$$

Файл **tools.py** содержит функции для граничных условий (**phi**, **psi**) аналитического решения (**analytic_solution**), алгоритма прогонки (**tridiagonal_matrix_algorithm**).

In []:

```
import numpy as np
```

In []:

```
def phi(x, t, a):
    if x == 0.0 or x == 1.0:
        return x
    else:
        raise ValueError('incorrect border value')
```

In []:

```
def psi(x, t=0.0, a=0.0):
    return x + np.sin(np.pi * x)
```

In []:

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

In []:

```
def tridiagonal_matrix_algorithm(a, b, c, d):
    n = len(a)

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

    p[0] = -c[0] / b[0]
    q[0] = d[0] / b[0]

    for i in range(1, n):
        p[i] = -c[i] / (b[i] + a[i] * p[i - 1])
        q[i] = (d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1])

    x = np.zeros(n)
    x[-1] = q[-1]

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

    return x
```

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

In []:

```
import numpy as np
import matplotlib.pyplot as plt
from tools import analitic_solution
```

In []:

```
def plot_stat(name_u, name_e, a, n, cnt, step, x_min, x_max, u):
    """
    Parameters:
        name_u: название файла с графиком сеточных функций
        name_e: название файла с графиком ошибки
        a: коэффициент температуропроводности
        n: количество точек в пространстве
        cnt: количество временных точек
        step: временной шаг
        x_min: левая граница
        x_max: правая граница
        u: сеточная функция
    """
    t = np.zeros(cnt)

    for i in range(cnt):
        t[i] = i * step

    space = np.zeros(n)
    h = (x_max - x_min) / n

    for i in range(n):
        space[i] = x_min + i * h

    t_idx = np.linspace(0, t.shape[0] - 1, 6, dtype=np.int32)

    sx, sy = 3, 2
    fig, ax = plt.subplots(sx, sy)
    fig.tight_layout()
    fig.suptitle('Сеточная функция')
    fig.set_figwidth(12)
    fig.set_figheight(7)

    k = 0
    for i in range(sx):
        for j in range(sy):
            idx = t_idx[k]
            sol = [analytic_solution(x, t[idx], a) for x in space]

            ax[i][j].plot(space, u[idx], label='Численный метод')
            ax[i][j].plot(space, sol, label='Аналитическое решение')

            ax[i][j].grid(True)
            ax[i][j].set_xlabel('x')
            ax[i][j].set_ylabel('u')
            ax[i][j].set_title(f'Решения при t = {t[idx]}')

            k += 1

    plt.legend()
    plt.savefig(name_u, dpi=300)

    plt.clf()
    plt.cla()

    error = np.zeros(cnt)
    for i in range(cnt):
        sol = [analytic_solution(x, t[i], a) for x in space]
        error[i] = np.max(np.abs(u[i] - np.array(sol)))
```

```
plt.figure(figsize=(12, 7))
plt.plot(t[1:], error[1:], 'violet', label='Ошибка')
plt.title('График изменения ошибки во времени')
plt.xlabel('t')
plt.ylabel('error')
plt.grid(True)
plt.legend()

plt.savefig(name_e, dpi=300)

plt.clf()
plt.cla()
```

Файл **explicit.py** содержит реализацию явной конечно-разностной схемы.

In []:

```
import numpy as np
from tools import psi, phi
from plot import plot_stat
```

In []:

```

def explicit(a, n, cnt, step, x_min, x_max, approx):
    """
    Parameters:
        a: коэффициент температуропроводности
        n: количество точек в пространстве
        cnt: количество временных точек
        step: временной шаг
        x_min: левая граница
        x_max: правая граница
        approx: тип аппроксимации
            approx = 21: двухточечная аппроксимация с первым порядком
            approx = 32: трехточечная аппроксимация со вторым порядком
            approx = 22: двухточечная аппроксимация со вторым порядком

    Return:
        u: сеточная функция
    """
    h = (x_max - x_min) / n
    sigma = (a * a * step) / (h * h)

    if sigma > 0.5:
        raise ValueError(f'Явная схема не устойчива sigma = {sigma}')

    u = np.zeros((cnt, n))

    for i in range(1, n - 1):
        u[0][i] = psi(x_min + i * h)

    for i in range(1, cnt):
        for j in range(1, n - 1):
            add = u[i - 1][j + 1] + u[i - 1][j - 1]
            sub = 1 - 2 * sigma
            u[i][j] = sigma * add + sub * u[i - 1][j]

    if approx == 21:
        u[i][0] = u[i][1] - h * phi(0.0, i * step, a)
        u[i][-1] = u[i][-2] + h * phi(1.0, i * step, a)
    elif approx == 32:
        phi1 = phi(0.0, i * step, a)
        phi2 = phi(1.0, i * step, a)

        val1 = phi1 + u[i][2] / (2 * h) - 2 * u[i][1] / h
        val2 = phi2 - u[i][-3] / (2 * h) + 2 * u[i][-2] / h

        u[i][0] = val1 * 2 * h / -3
        u[i][-1] = val2 * 2 * h / 3
    elif approx == 22:
        add1 = h * h / (2 * step) * u[i - 1][0]
        add2 = h * h / (2 * step) * u[i - 1][-1]

        val1 = u[i][1] - h * phi(0.0, i * step, a) + add1
        val2 = u[i][-2] + h * phi(1.0, i * step, a) + add2

        u[i][0] = val1 / (1 + h * h / (2 * step))
        u[i][-1] = val2 / (1 + h * h / (2 * step))
    else:
        str = 'Такого типа аппроксимации граничных условий не существует'
        raise ValueError(str)

    return u

```

In []:

```

def main(params):
    a = params['a']
    n = params['n']
    cnt = params['cnt']
    step = params['step']
    x_min = params['x_min']
    x_max = params['x_max']
    approx = params['approx']

    filename_u = f'images/explicit_function_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_u += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    filename_e = f'images/explicit_error_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_e += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    u = explicit(a, n, cnt, step, x_min, x_max, approx)
    plot_stat(filename_u, filename_e, a, n, cnt, step, x_min, x_max, u)

```

In []:

```

if __name__ == '__main__':
    run1 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 21}

    run2 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 32}

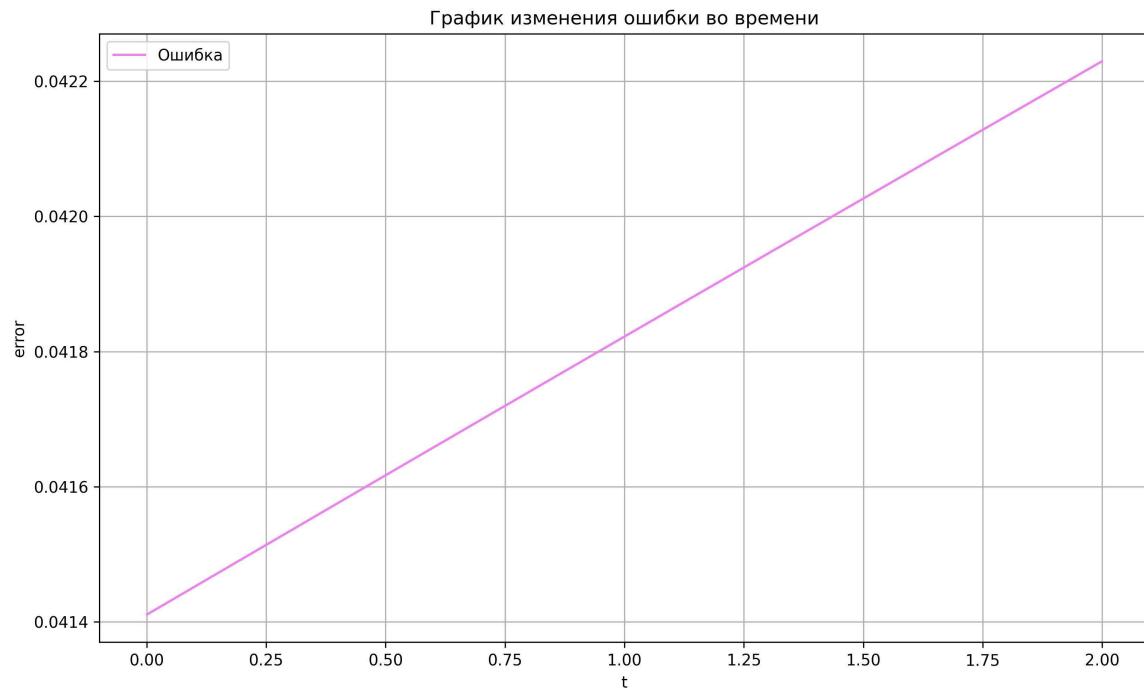
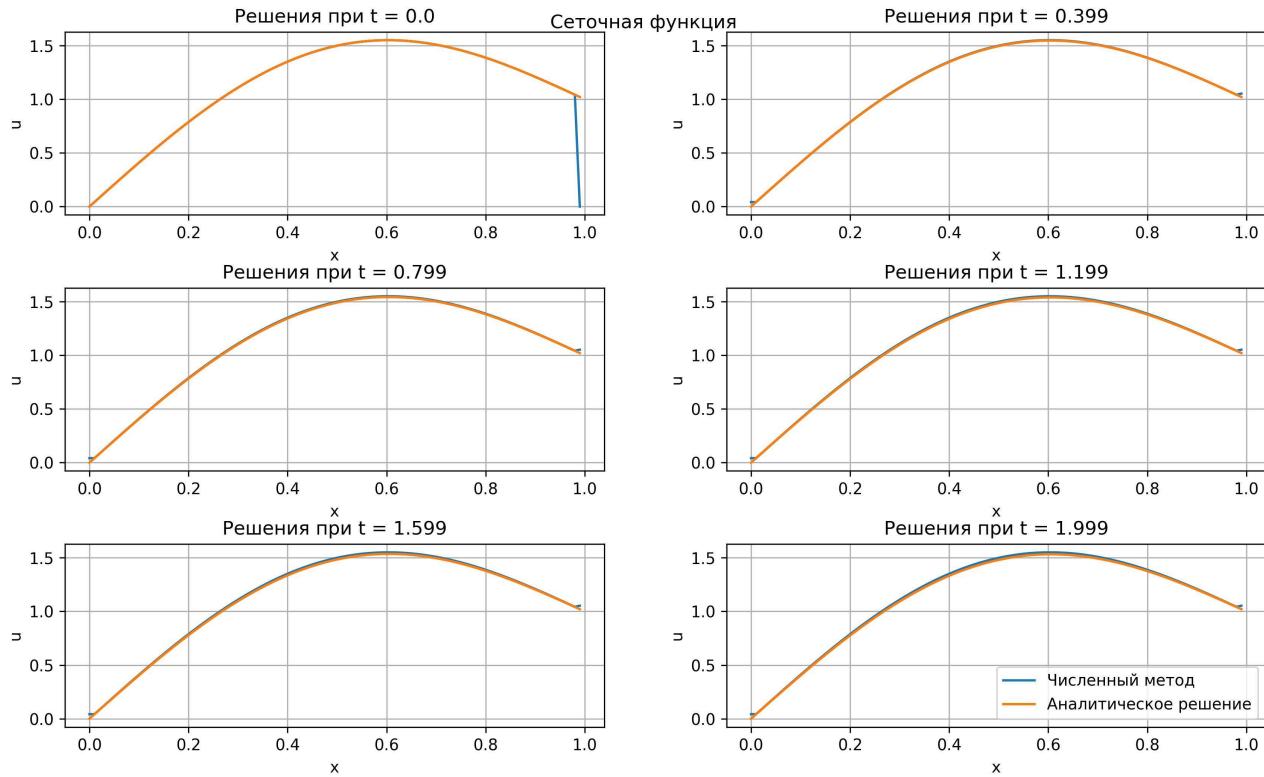
    run3 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 22}

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

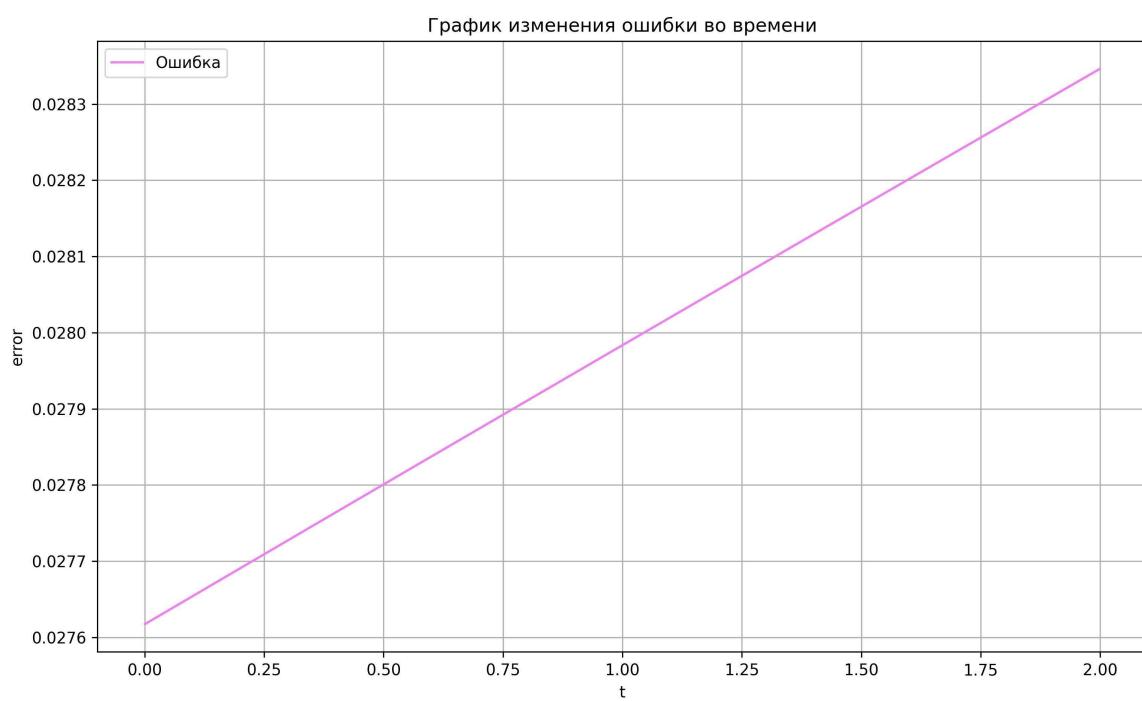
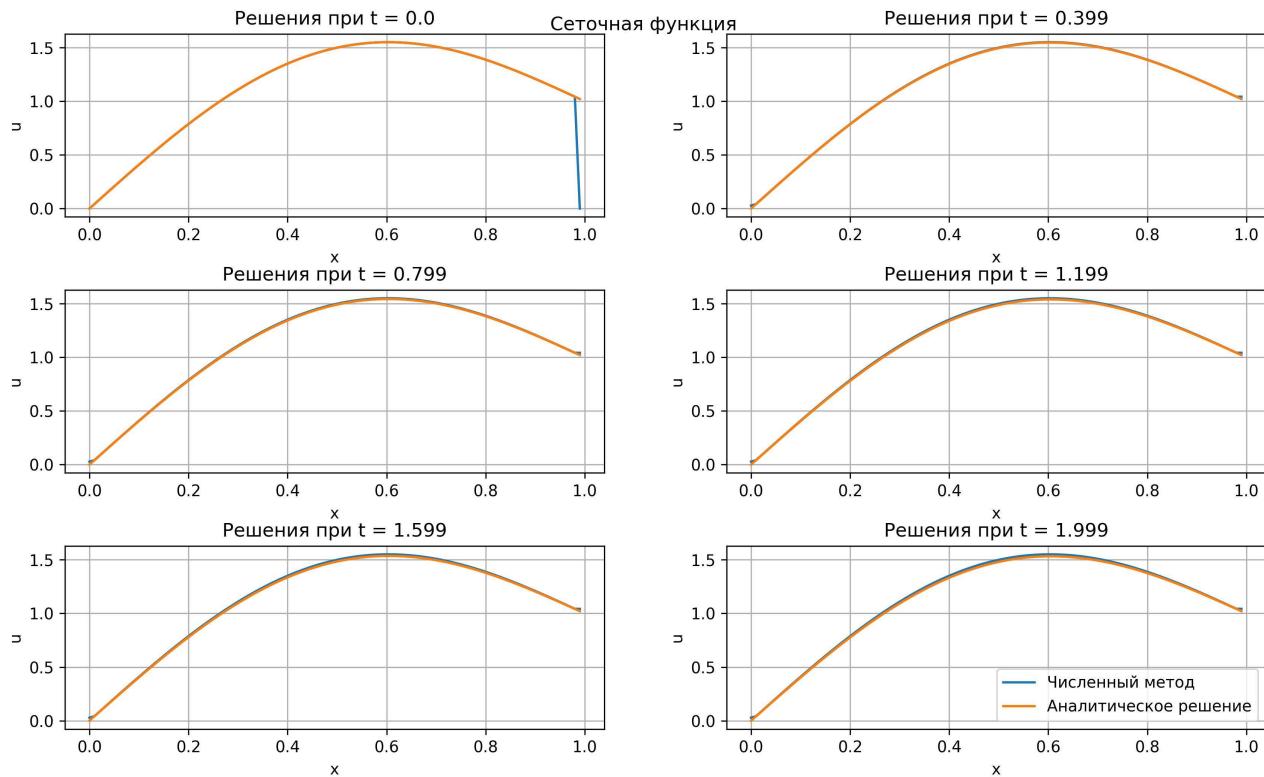
```

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

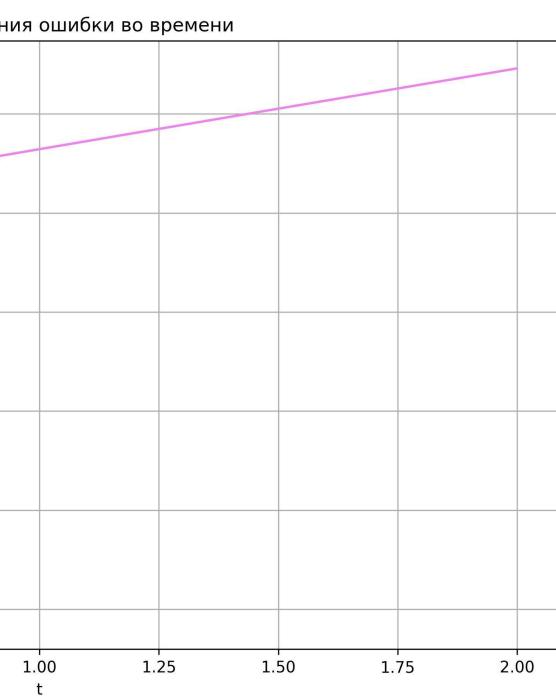
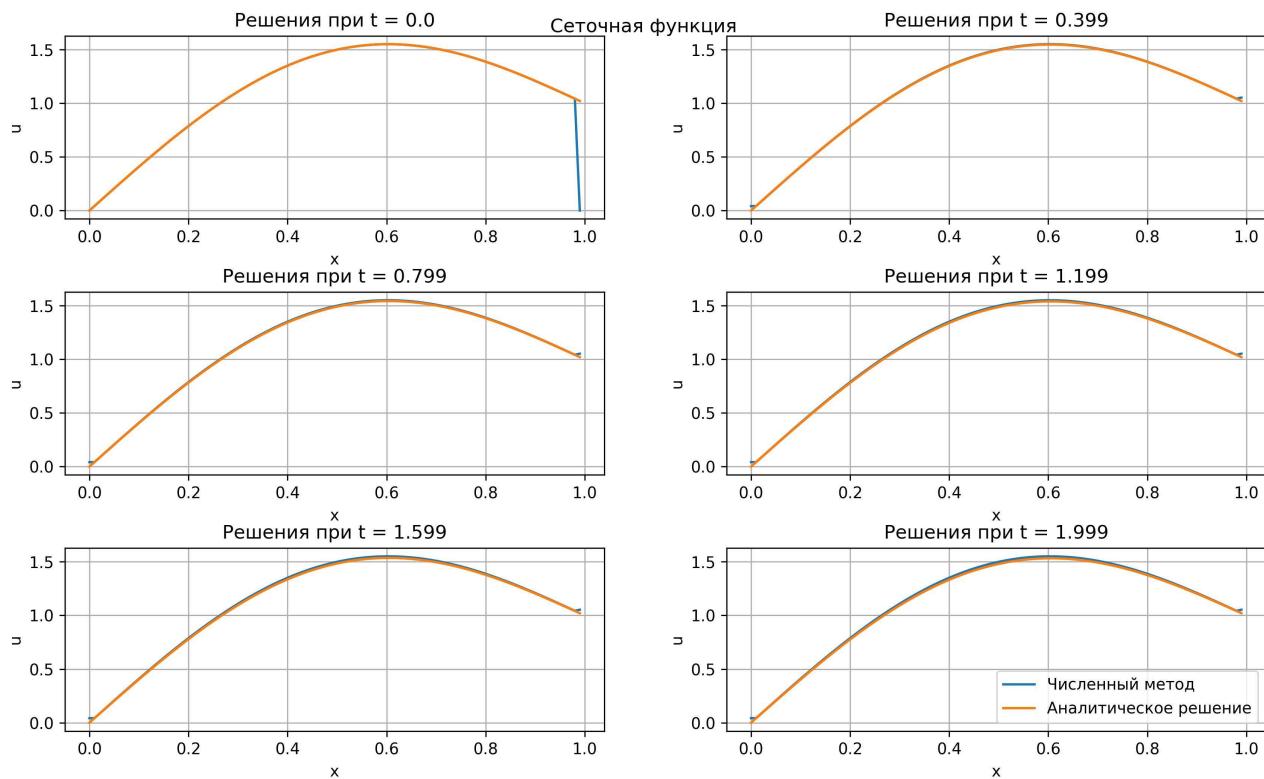
Двухточечная аппроксимация с первым порядком.



Трехточечная аппроксимация со вторым порядком.



Двухточечная аппроксимация со вторым порядком.



Файл **implicit.py** содержит реализацию неявной конечно-разностной схемы.

In []:

```
import numpy as np
from tools import psi, phi, tridiagonal_matrix_algorithm
from plot import plot_stat
```

In []:

```
def implicit(a, n, cnt, step, x_min, x_max, approx):
    """
    Parameters:
        a: коэффициент температуропроводности
        n: количество точек в пространстве
        cnt: количество временных точек
        step: временной шаг
        x_min: левая граница
        x_max: правая граница
        approx: тип аппроксимации
            approx = 21: двухточечная аппроксимация с первым порядком
            approx = 32: трехточечная аппроксимация со вторым порядком
            approx = 22: двухточечная аппроксимация со вторым порядком

    Return:
        u: сеточная функция
    """
    eta = 1.0

    u = np.zeros((cnt, n))
    h = (x_max - x_min) / n
    sigma = (a * a * step) / (h * h)

    for i in range(1, n - 1):
        u[0][i] = psi(x_min + i * h)

    for i in range(1, cnt):
        alst, blst, clst, dlst = np.zeros(
            n), np.zeros(n), np.zeros(n), np.zeros(n)

        for j in range(1, n - 1):
            alst[j] = sigma
            blst[j] = -(1 + 2 * sigma)
            clst[j] = sigma
            dlst[j] = -u[i - 1][j]

        if approx == 21:
            blst[0] = -1 / h
            clst[0] = 1 / h
            dlst[0] = phi(0.0, (i + 1) * step, a)
            alst[-1] = -1 / h
            alst[-1] = 1 / h
            dlst[-1] = phi(1.0, (i + 1) * step, a)
        elif approx == 32:
            k0 = 1 / (2 * h) / clst[1]
            blst[0] = (-3 / (2 * h) + alst[1] * k0)
            clst[0] = 2 / h + blst[1] * k0
            dlst[0] = phi(0.0, (i + 1) * step, a) + dlst[1] * k0
            k1 = -(1 / (h * 2)) / alst[-2]
            alst[-1] = (-2 / h) + blst[-2] * k1
            blst[-1] = (3 / (h * 2)) + clst[-2] * k1
            dlst[-1] = phi(1.0, (i + 1) * step, a) + dlst[-2] * k1
        elif approx == 22:
            blst[0] = 2 * a * a / h + h / step
            clst[0] = -2 * a * a / h
            dsub = phi(0.0, (i + 1) * step, a) * 2 * a * a
            dlst[0] = (h / step) * u[i - 1][0] - dsub
            alst[-1] = -2 * a * a / h
            blst[-1] = 2 * a * a / h + h / step
```

```

        dadd = phi(1.0, (i + 1) * step, a) * 2 * a * a
        dlst[-1] = (h / step) * u[i - 1][-1] + dadd
    else:
        str = 'Такого типа аппроксимации граничных условий не существует'
        raise ValueError(str)

    u[i] = eta * tridiagonal_matrix_algorithm(alst, blst, clst, dlst)
    explicit_part = np.zeros(n)

    for j in range(1, n - 1):
        val1 = sigma * (u[i - 1][j + 1] + u[i - 1][j - 1])
        val2 = (1 - 2 * sigma) * u[i - 1][j]
        explicit_part[j] = val1 + val2

    if approx == 21:
        explicit_part[0] = explicit_part[1] - h * phi(0.0, i * step, a)
        explicit_part[-1] = explicit_part[-2] + h * phi(1.0, i * step, a)
    elif approx == 32:
        sub1 = 2 * explicit_part[1] / h
        val1 = phi(0.0, i * step, a) + explicit_part[2] / (2 * h) - sub1
        explicit_part[0] = val1 * 2 * h / -3

        add2 = 2 * explicit_part[-2] / h
        val2 = phi(1.0, i * step, a) - explicit_part[-3] / (2 * h) + add2
        explicit_part[-1] = val2 * 2 * h / 3
    elif approx == 22:
        add1 = h * h / (2 * step) * u[i - 1][0]
        val1 = explicit_part[1] - h * phi(0.0, i * step, a) + add1
        explicit_part[0] = val1 / (1 + h * h / (2 * step))

        add2 = h * h / (2 * step) * u[i - 1][-1]
        val2 = explicit_part[-2] + h * phi(1.0, i * step, a) + add2
        explicit_part[-1] = val2 / (1 + h * h / (2 * step))
    else:
        str = 'Такого типа аппроксимации граничных условий не существует'
        raise ValueError(str)

    u[i] += (1 - eta) * explicit_part

return u

```

In []:

```

def main(params):
    a = params['a']
    n = params['n']
    cnt = params['cnt']
    step = params['step']
    x_min = params['x_min']
    x_max = params['x_max']
    approx = params['approx']

    filename_u = f'images/implicit_function_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_u += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    filename_e = f'images/implicit_error_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_e += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    u = implicit(a, n, cnt, step, x_min, x_max, approx)
    plot_stat(filename_u, filename_e, a, n, cnt, step, x_min, x_max, u)

```

In []:

```
if __name__ == '__main__':
    run1 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 21}

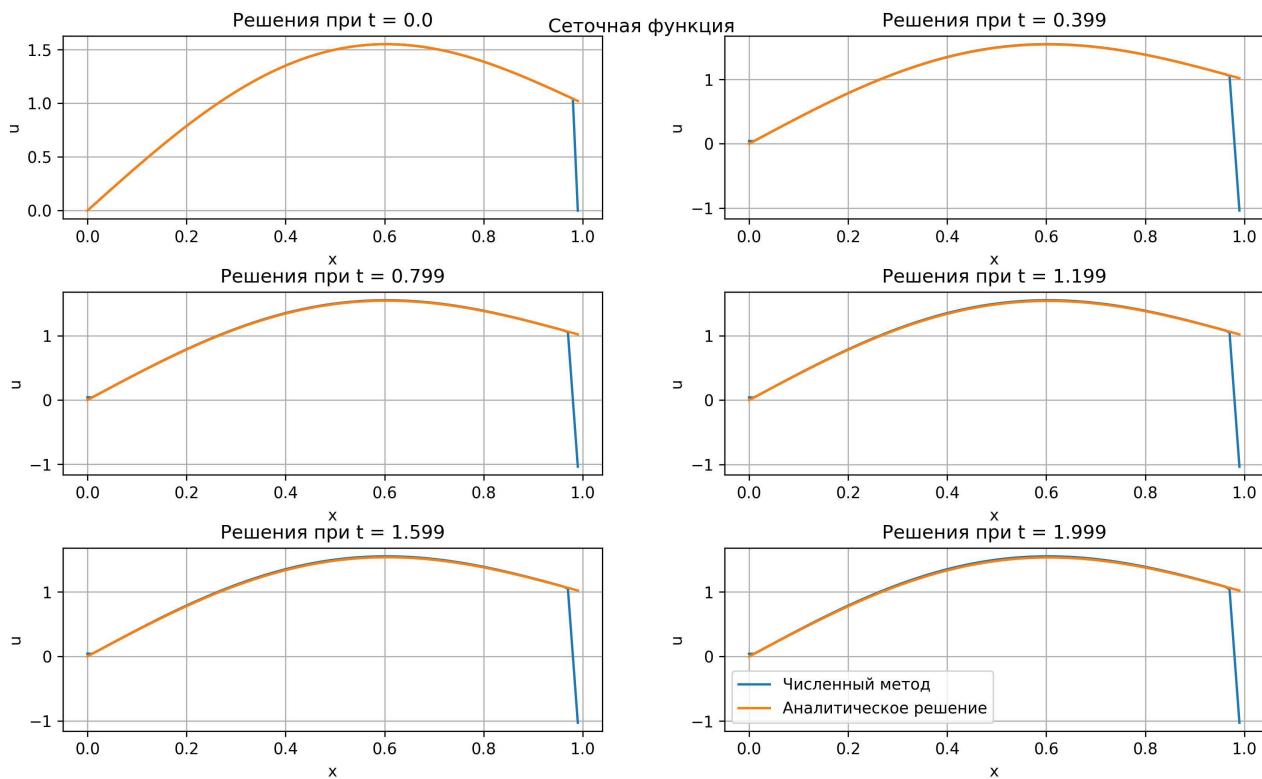
    run2 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 32}

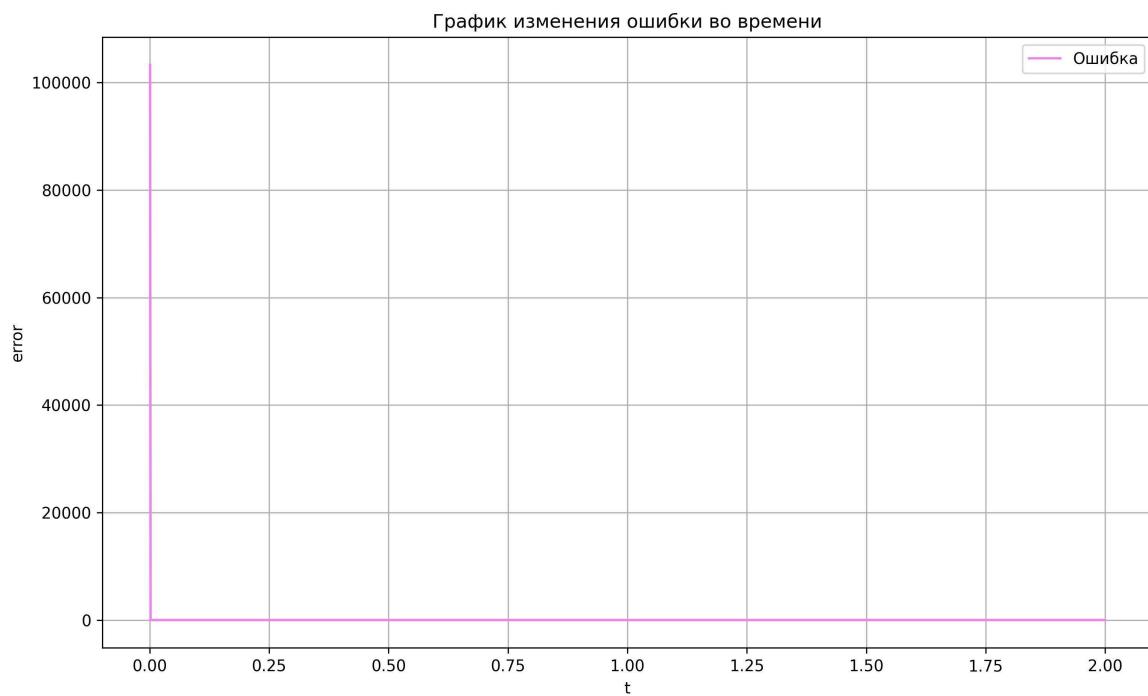
    run3 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 22}

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

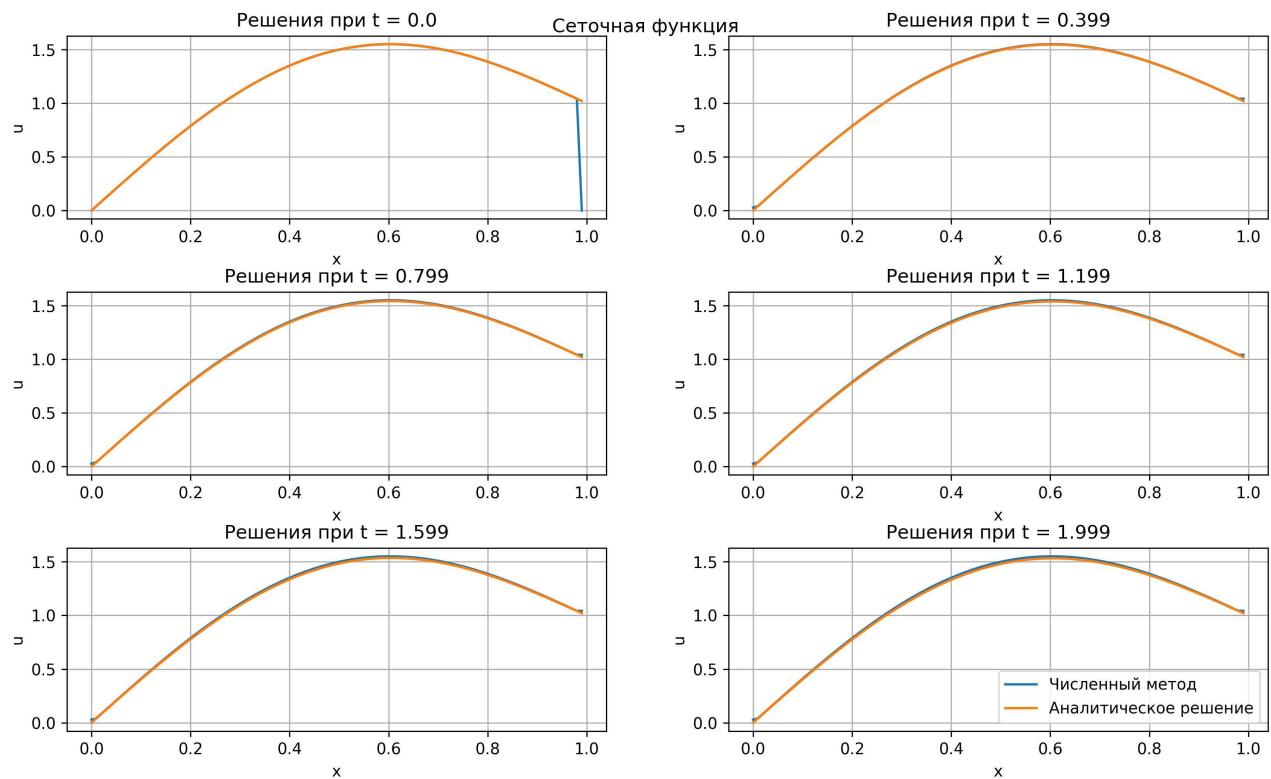
Графики сравнения аналитического решения и численного метода, значения ошибки для неявной конечно-разностной схемы при разных вариантах аппроксимации (**двуточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком**).

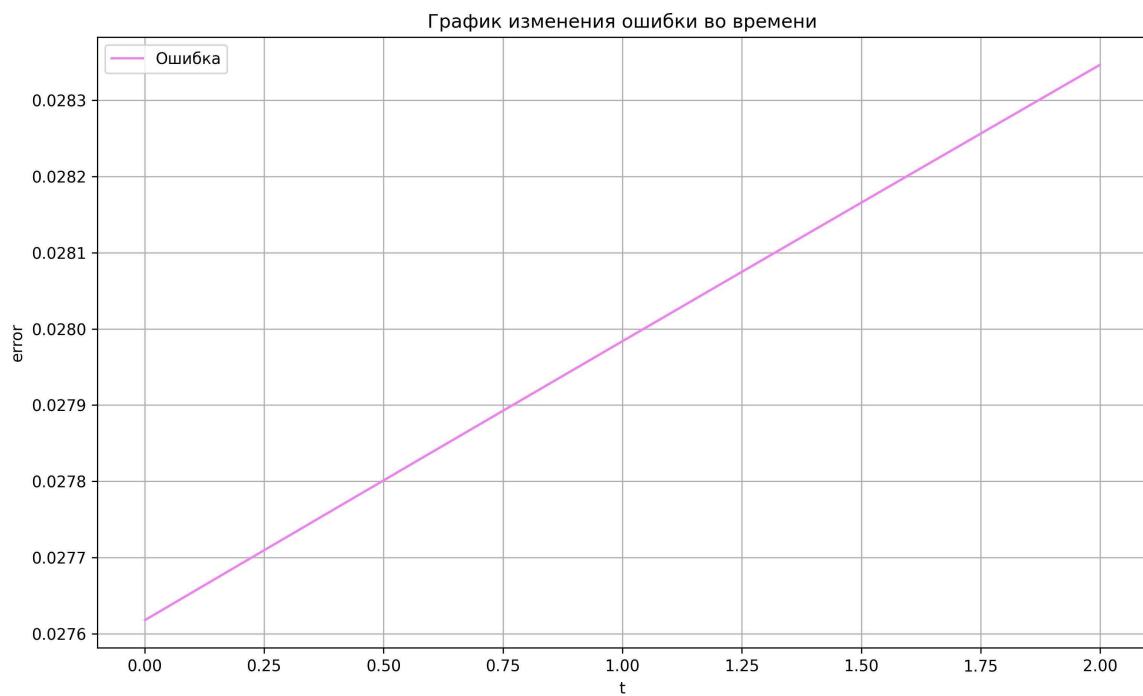
Двуточечная аппроксимация с первым порядком.



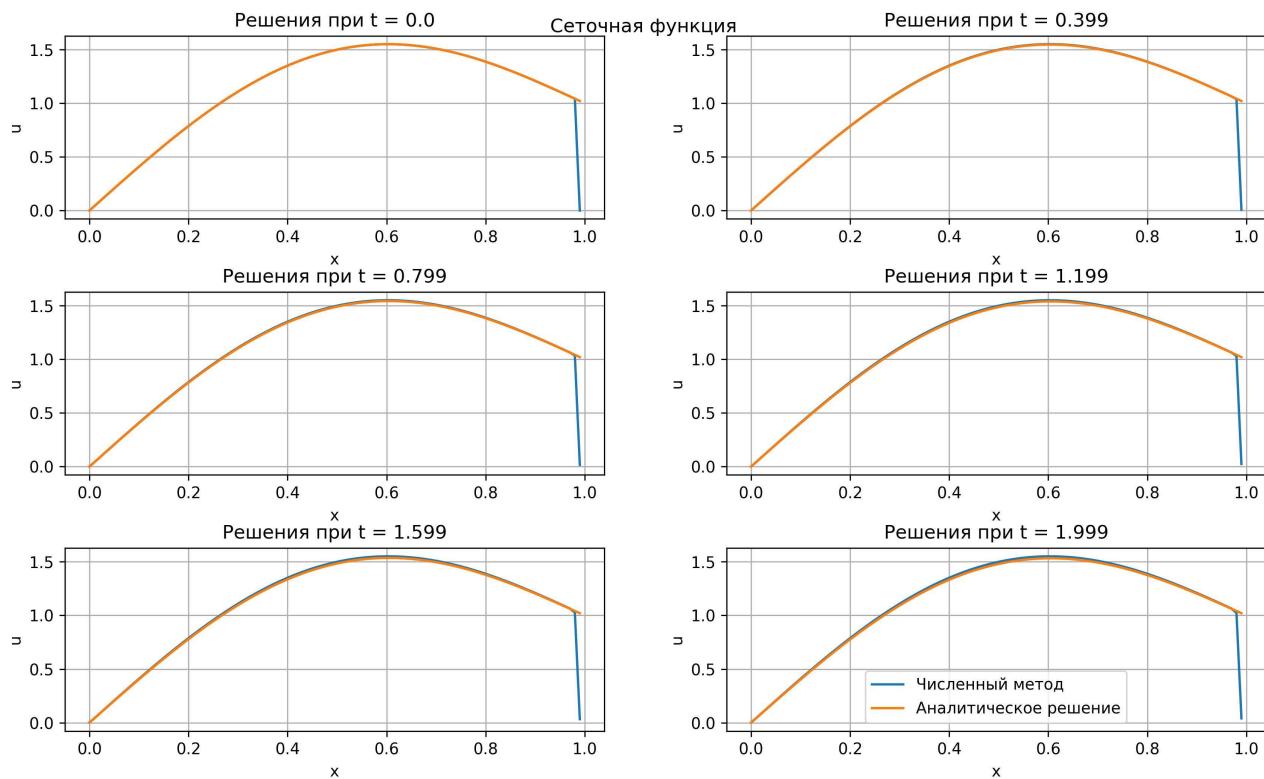


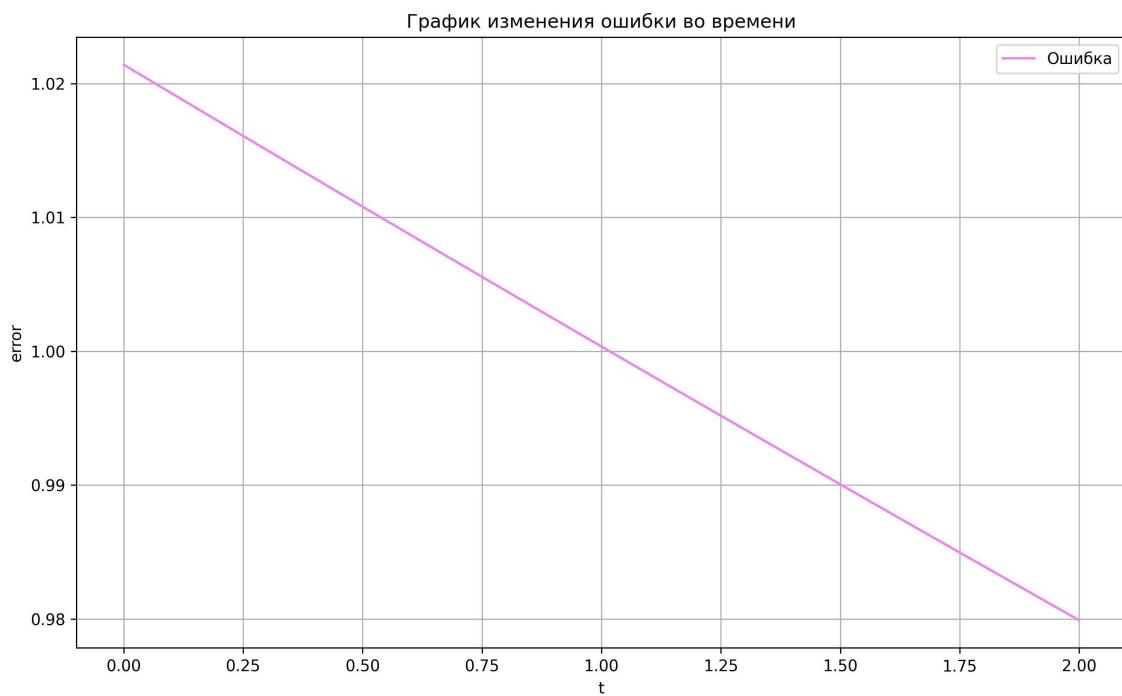
Трехточечная аппроксимация со вторым порядком.





Двухточечная аппроксимация со вторым порядком.





Файл **crank.py** содержит реализацию схемы Кранка - Николсона.

In []:

```
import numpy as np
from tools import psi, phi, tridiagonal_matrix_algorithm
from plot import plot_stat
```

In []:

```
def crank(a, n, cnt, step, x_min, x_max, approx):
    """
    Parameters:
        a: коэффициент температуропроводности
        n: количество точек в пространстве
        cnt: количество временных точек
        step: временной шаг
        x_min: левая граница
        x_max: правая граница
        approx: тип аппроксимации
            approx = 21: двухточечная аппроксимация с первым порядком
            approx = 32: трехточечная аппроксимация со вторым порядком
            approx = 22: двухточечная аппроксимация со вторым порядком

    Return:
        u: сеточная функция
    """
eta = 0.5

u = np.zeros((cnt, n))
h = (x_max - x_min) / n
sigma = (a * a * step) / (h * h)

for i in range(1, n - 1):
    u[0][i] = psi(x_min + i * h)

for i in range(1, cnt):
    alst, blst, clst, dlst = np.zeros(
        n), np.zeros(n), np.zeros(n), np.zeros(n)

    for j in range(1, n - 1):
        alst[j] = sigma
        blst[j] = -(1 + 2 * sigma)
        clst[j] = sigma
        dlst[j] = -u[i - 1][j]

    if approx == 21:
        blst[0] = -1 / h
        clst[0] = 1 / h
        dlst[0] = phi(0.0, (i + 1) * step, a)
        alst[-1] = -1 / h
        alst[-1] = 1 / h
        dlst[-1] = phi(1.0, (i + 1) * step, a)
    elif approx == 32:
        k0 = 1 / (2 * h) / clst[1]
        blst[0] = (-3 / (2 * h) + alst[1] * k0)
        clst[0] = 2 / h + blst[1] * k0
        dlst[0] = phi(0.0, (i + 1) * step, a) + dlst[1] * k0
        k1 = -(1 / (h * 2)) / alst[-2]
        alst[-1] = (-2 / h) + blst[-2] * k1
        blst[-1] = (3 / (h * 2)) + clst[-2] * k1
        dlst[-1] = phi(1.0, (i + 1) * step, a) + dlst[-2] * k1
    elif approx == 22:
        blst[0] = 2 * a * a / h + h / step
        clst[0] = -2 * a * a / h
        dsub = phi(0.0, (i + 1) * step, a) * 2 * a * a
        dlst[0] = (h / step) * u[i - 1][0] - dsub
        alst[-1] = -2 * a * a / h
        blst[-1] = 2 * a * a / h + h / step
```

```
dadd = phi(1.0, (i + 1) * step, a) * 2 * a * a
dlst[-1] = (h / step) * u[i - 1][-1] + dadd
else:
    str = 'Такого типа аппроксимации граничных условий не существует'
    raise ValueError(str)

u[i] = eta * tridiagonal_matrix_algorithm(alst, blst, clst, dlst)
explicit_part = np.zeros(n)

for j in range(1, n - 1):
    val1 = sigma * (u[i - 1][j + 1] + u[i - 1][j - 1])
    val2 = (1 - 2 * sigma) * u[i - 1][j]
    explicit_part[j] = val1 + val2

if approx == 21:
    explicit_part[0] = explicit_part[1] - h * phi(0.0, i * step, a)
    explicit_part[-1] = explicit_part[-2] + h * phi(1.0, i * step, a)
elif approx == 32:
    sub1 = 2 * explicit_part[1] / h
    val1 = phi(0.0, i * step, a) + explicit_part[2] / (2 * h) - sub1
    explicit_part[0] = val1 * 2 * h / -3

    add2 = 2 * explicit_part[-2] / h
    val2 = phi(1.0, i * step, a) - explicit_part[-3] / (2 * h) + add2
    explicit_part[-1] = val2 * 2 * h / 3
elif approx == 22:
    add1 = h * h / (2 * step) * u[i - 1][0]
    val1 = explicit_part[1] - h * phi(0.0, i * step, a) + add1
    explicit_part[0] = val1 / (1 + h * h / (2 * step))

    add2 = h * h / (2 * step) * u[i - 1][-1]
    val2 = explicit_part[-2] + h * phi(1.0, i * step, a) + add2
    explicit_part[-1] = val2 / (1 + h * h / (2 * step))
else:
    str = 'Такого типа аппроксимации граничных условий не существует'
    raise ValueError(str)

u[i] += (1 - eta) * explicit_part

return u
```

In []:

```
def main(params):
    a = params['a']
    n = params['n']
    cnt = params['cnt']
    step = params['step']
    x_min = params['x_min']
    x_max = params['x_max']
    approx = params['approx']

    filename_u = f'images/crank_function_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_u += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    filename_e = f'images/crank_error_a={a}_n={n}_cnt={cnt}_step={step}'
    filename_e += f'_xmin={x_min}_xmax={x_max}_approx={approx}.jpg'

    u = crank(a, n, cnt, step, x_min, x_max, approx)
    plot_stat(filename_u, filename_e, a, n, cnt, step, x_min, x_max, u)
```

In []:

```
if __name__ == '__main__':
    run1 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 21}

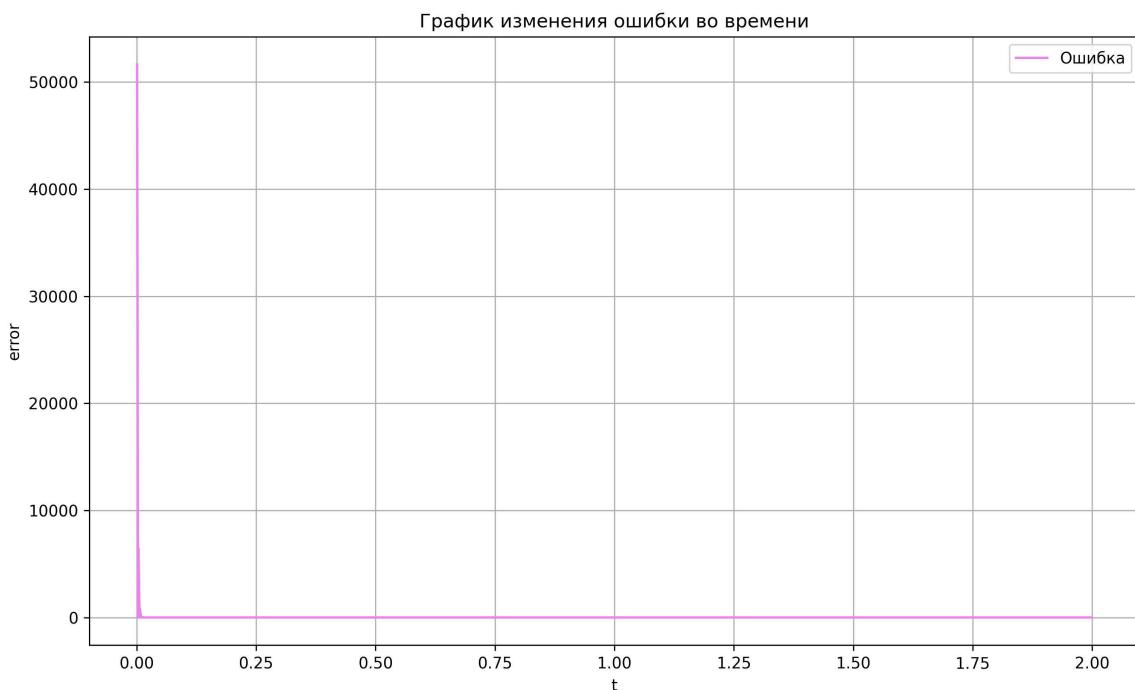
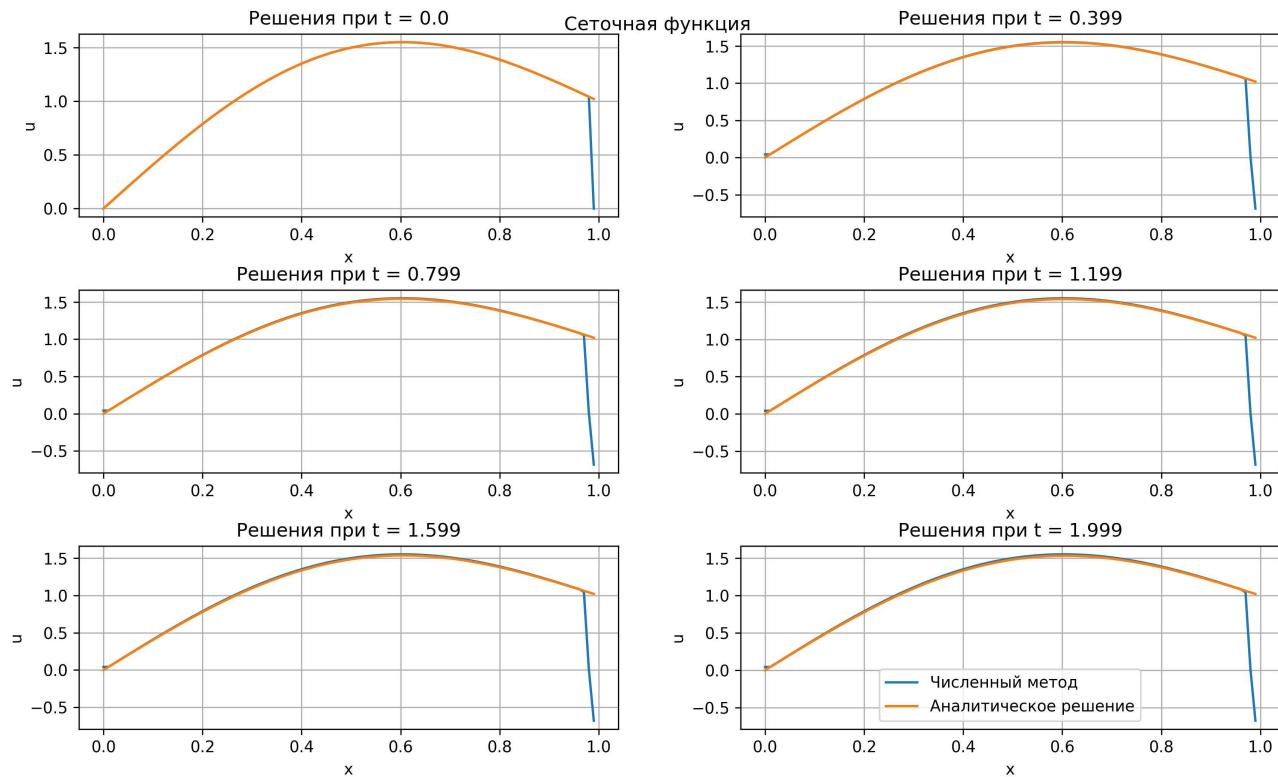
    run2 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 32}

    run3 = {'a': 0.001,
            'n': 100,
            'cnt': 2000,
            'step': 0.001,
            'x_min': 0.0,
            'x_max': 1.0,
            'approx': 22}

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

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

Двуточечная аппроксимация с первым порядком.



Трехточечная аппроксимация со вторым порядком.

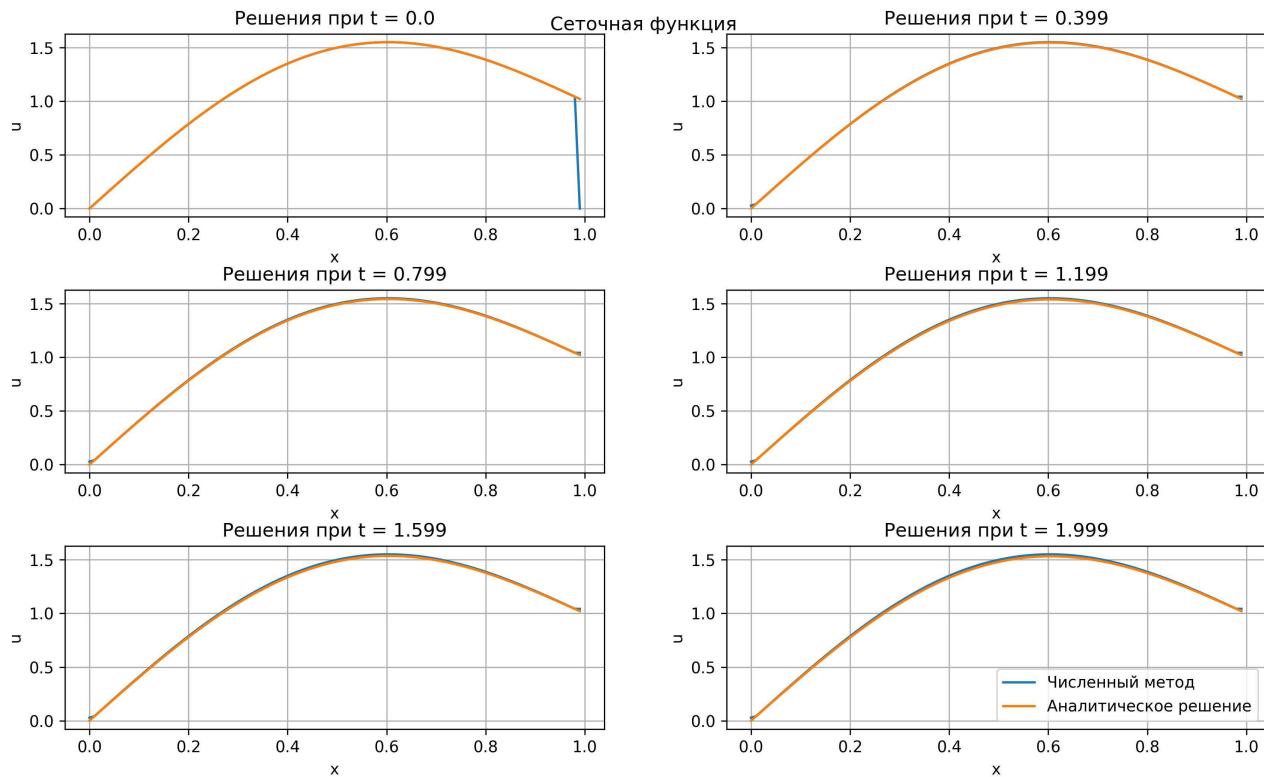
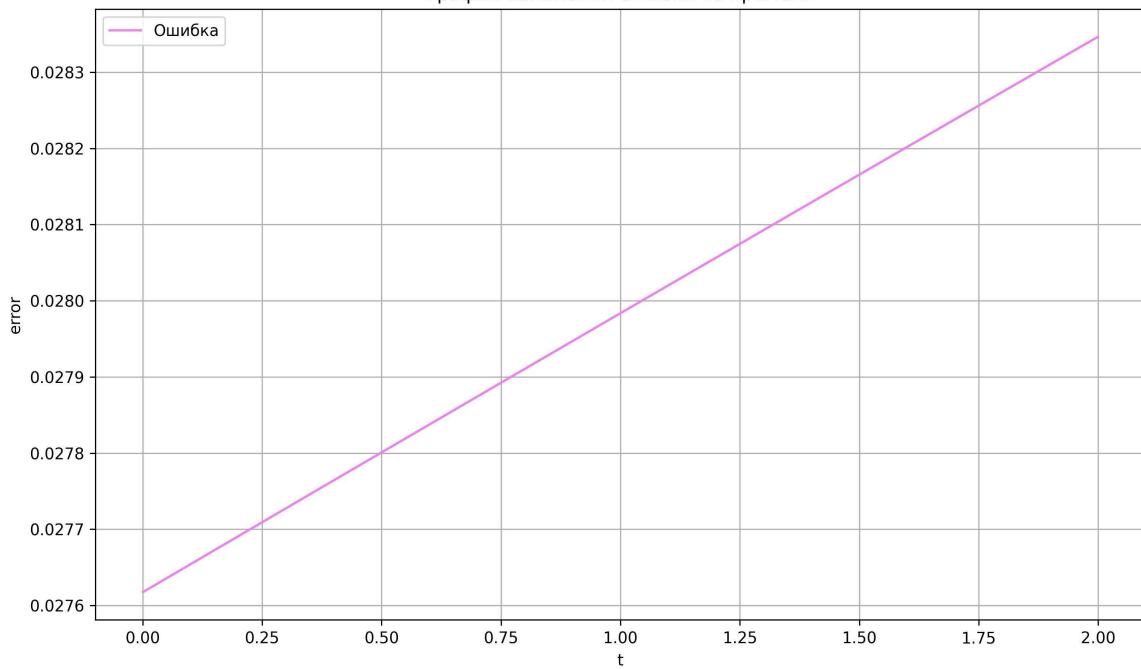
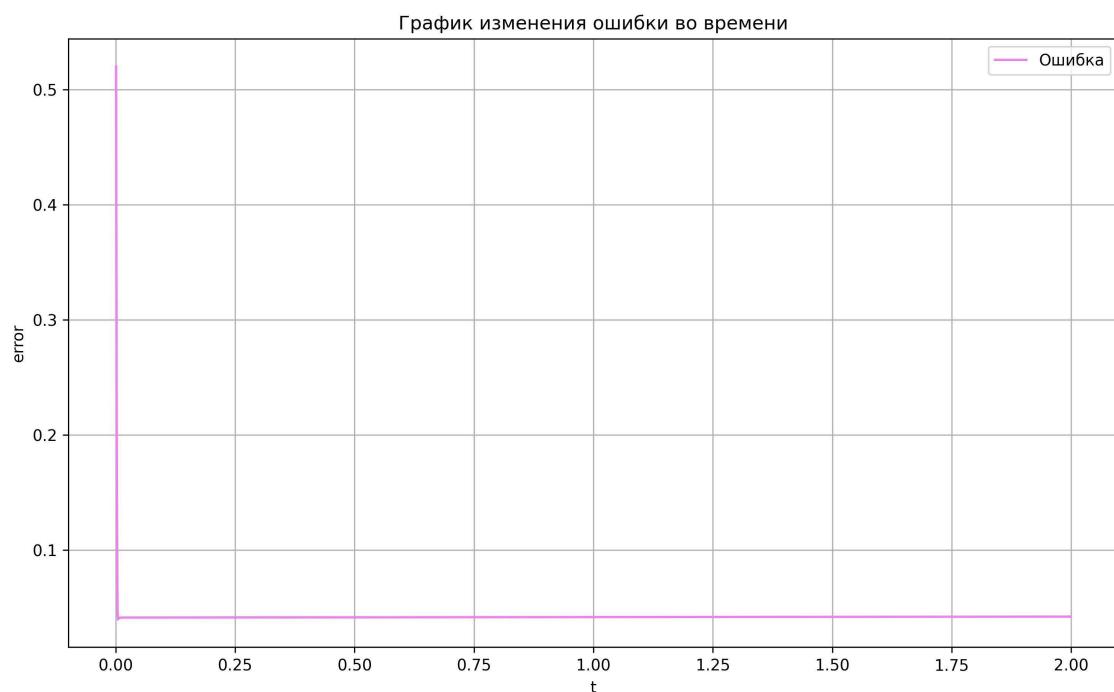
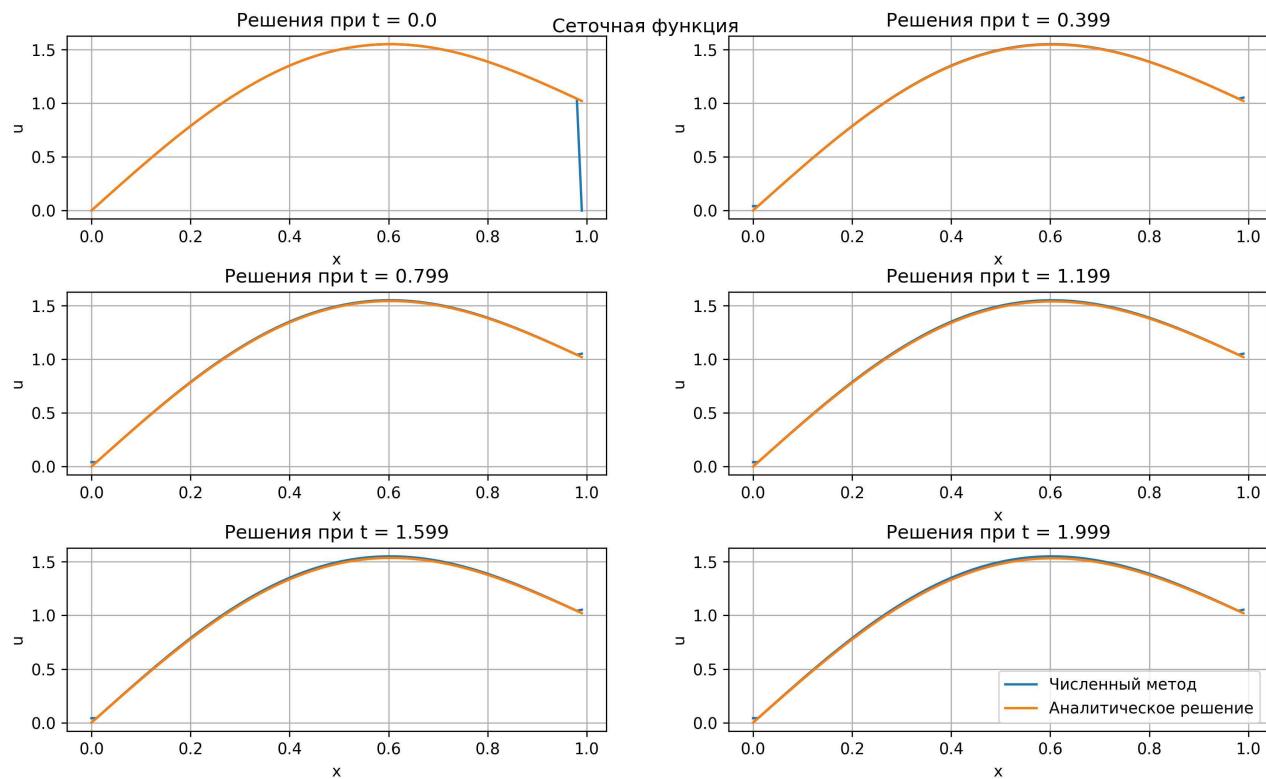


График изменения ошибки во времени



Двухточечная аппроксимация со вторым порядком.



In []: