Лабораторная работа №5 по курсу "Численные методы"

Тема ЛР - "Начально-краевые задачи для дифференциального уравнения параболического типа"

Студент - Письменский Данила Владимирович

Группа - М8О-406Б-19

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)*.

Вариант 3

$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2}$
$a > 0$
$u(0,t) = e^{-at}$
$u(\pi,t) = -e^{-at}$
$u(x,0) = cos(x)$
Аналитическое решение: $U(x,t) = e^{-at}cos(x)$

In [14]:
```
# импортируем библиотеки
import numpy as np
import matplotlib.pyplot as plt
```

Зададим начальные условия, а также коэффициенты *a*, *sigma* и *h* - размер сетки для координатной составляющей. Размер сетки для временной составляющей будем искать из формулы $\tau = \sigma\frac{h^2}{a}$.

Константы

In [15]:
```
a = 1

x_start = 0
x_end = np.pi

t_start = 0
t_end = 5

h = 0.01
sigma = 0.37
```

Начальные условия

In [16]:
```
def phi_0(t, a):
    return np.exp(-a * t)

def phi_1(t, a):
    return -np.exp(-a * t)

def psi(x):
    return np.cos(x)

def exact_sol(x, t, a):
    return np.exp(-a * t) * np.cos(x)
```

Аналитическое решение

Найдем аналитическое решение начально-краевой задачи для дифференциального уравнения параболического типа, затем будем сравнивать его с численными методами для того. Это нам пригодится, чтобы визуализировать зависимость максимального модуля ошибки от времени. Для этого реализовал функцию, которая возвращает матрицу *U* со значениями функции для аналитического решения.

In [17]:
```
def analytical_solve(x_start, x_end, t_start, t_end, a, h, sigma):
    tau = sigma * h**2 / a
    x = np.arange(x_start, x_end, h)
    t = np.arange(t_start, t_end, tau)

    U = np.zeros((len(t), len(x)))
    for i_x in range(len(x)):
        for i_t in range(len(t)):
            U[i_t][i_x] = exact_sol(x[i_x], t[i_t], a)

    return U
```

In [18]:
```
anal_solution = analytical_solve(x_start, x_end, t_start, t_end, a, h, sigma)
```

In [19]:
```
anal_solution.shape
```

Out[19]:
```
(135136, 315)
```

Погрешность

В качестве погрешности буду использовать максимальный модуль ошибки.

In [21]:
```
def max_abs_error(U_num, U_anal):
    return abs(U_num - U_anal).max()
```

Реализация функций построения графиков

Для того, чтобы визуализировать решение ДУ численными методами, реализую функцию построения графика функции *U(t)* при заданном времени *t*.

In [22]:
```
def build_numerical_results_graphic(solution, method_name, time, x_start, x_end, t_start
    tau = sigma * h**2 / a
    x = np.arange(x_start, x_end, h)
```

```
    times = np.arange(t_start, t_end, tau)
    cur_t_id = abs(times - time).argmin()

    plt.figure(figsize=(15, 9))
    plt.plot(x, anal_solution[cur_t_id], label='Аналитическое решение')
    plt.plot(x, solution[cur_t_id], label=method_name, color='r')

    plt.xlabel('x')
    plt.ylabel('U(t)')
    plt.legend()
    plt.grid()
    plt.show()
```

Чтобы проверить, насколько точно решение ДУ численными методами, необходимо реализовать функцию построения графика зависимости погрешности (максимального модуля ошибки) от времени.

In [23]:
```
def build_errors_graphic(solution, method_name, t_start, t_end, h, sigma):
    tau = sigma * h**2 / a
    t = np.arange(t_start, t_end, tau)

    plt.figure(figsize=(15, 9))
    max_abs_errors = np.array([max_abs_error(solution[i], anal_solution[i]) for i in ran
    plt.plot(t, max_abs_errors, label=method_name, color='g')

    plt.xlabel('Время')
    plt.ylabel('Максимальный модуль ошибки')

    plt.legend()
    plt.grid()
    plt.show()
```

Численные методы

Явная конечно-разностная схема

Преобразуем исходное уравнение с производными в уравнение с их численными приближениями. Производную второго порядка в правой части уравнения будем аппроксимировать по значениям нижнего временного слоя.

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2};$$

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} => u_j^{k+1} = \tau a\frac{u_{j-1}^k + (h^2 - 2)u_j^k + u_{j+1}^k}{h^2}$$

Получили рекуррентное соотношение. Начальные условия позволяет нам посчитать значения u в нижнем временном ряду. Далее в цикле считаем значения в узлах сетки.

Особенностью метода явной конечно-разностной схемы является условие $\sigma = \frac{a\tau}{h^2} < \frac{1}{2}$, при котором данный метод сходится. В противном случае, погрешность вычисления будет очень большой.

In [24]:
```
def explicit_finite_difference_method(x_start, x_end, t_start, t_end, a, h, sigma, phi_0
    assert sigma < (1/2)
    tau = sigma * h**2 / a
    x = np.arange(x_start, x_end, h)
    t = np.arange(t_start, t_end, tau)

    U = np.zeros((len(t), len(x)))
    # подсчитываем значения на нижней границе (t = 0)
    for i in range(len(x)):
        U[0][i] = psi(x[i])
```

```
    for dt in range(1, len(t)):
        # подсчитываем значения на левой границе (x = 0)
        U[dt][0] = phi_0(t[dt], a)
        for dx in range(1, len(x) - 1):
            U[dt][dx] = sigma * U[dt - 1][dx - 1] + (1 - 2 * sigma) * U[dt - 1][dx] + si
        # подсчитываем значения на правой границе (x = pi)
        U[dt][-1] = phi_1(t[dt], a)
    return U
```

In [25]:
```
explicit_solution = explicit_finite_difference_method(x_start, x_end, t_start, t_end, a,
```

In [26]:
```
explicit_solution.shape
```

Out[26]:
```
(135136, 315)
```

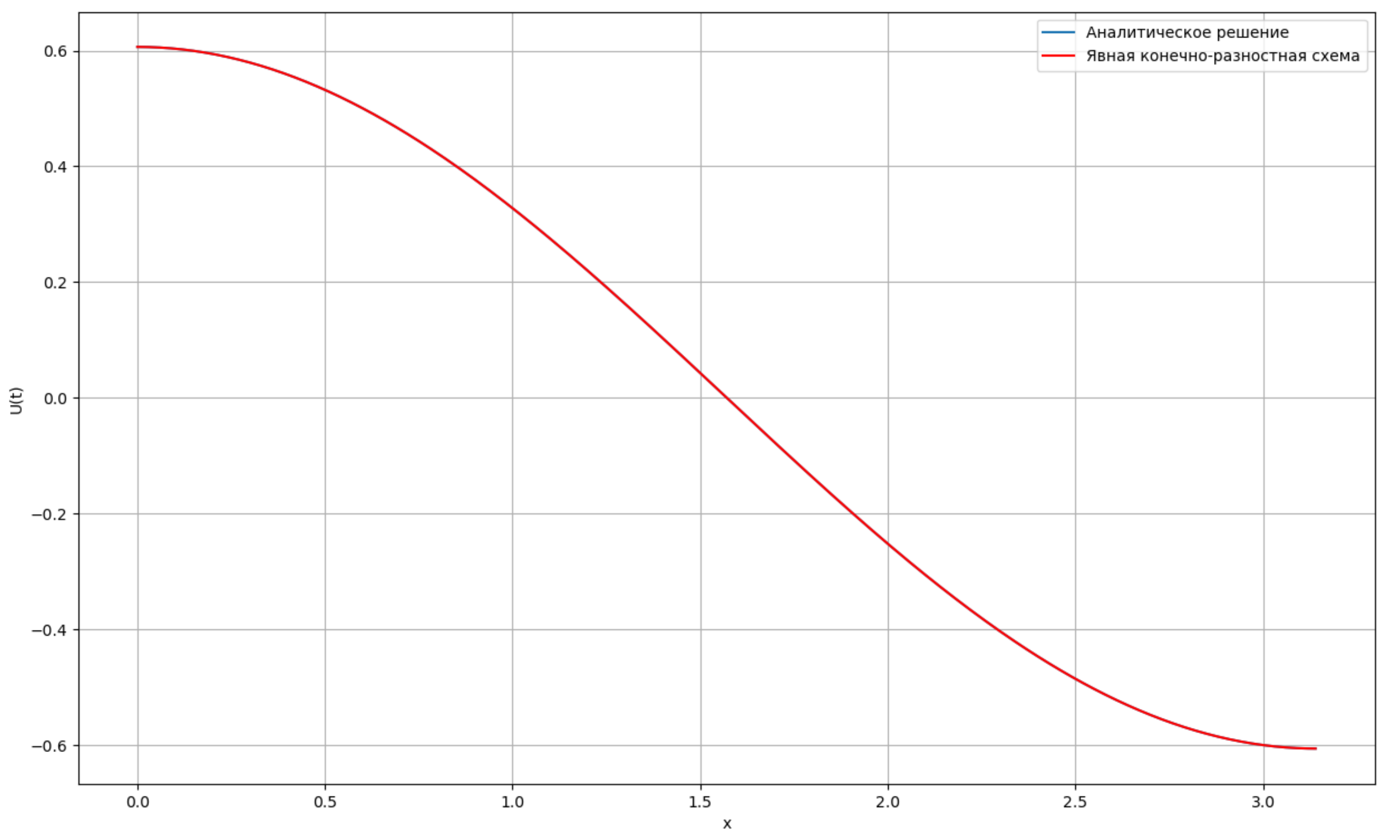
Максимальный модуль ошибки

In [27]:
```
print(f'Максимальный модуль ошибки = {max_abs_error(explicit_solution, anal_solution)}')
```

```
Максимальный модуль ошибки = 1.3637662959475882e-06
```

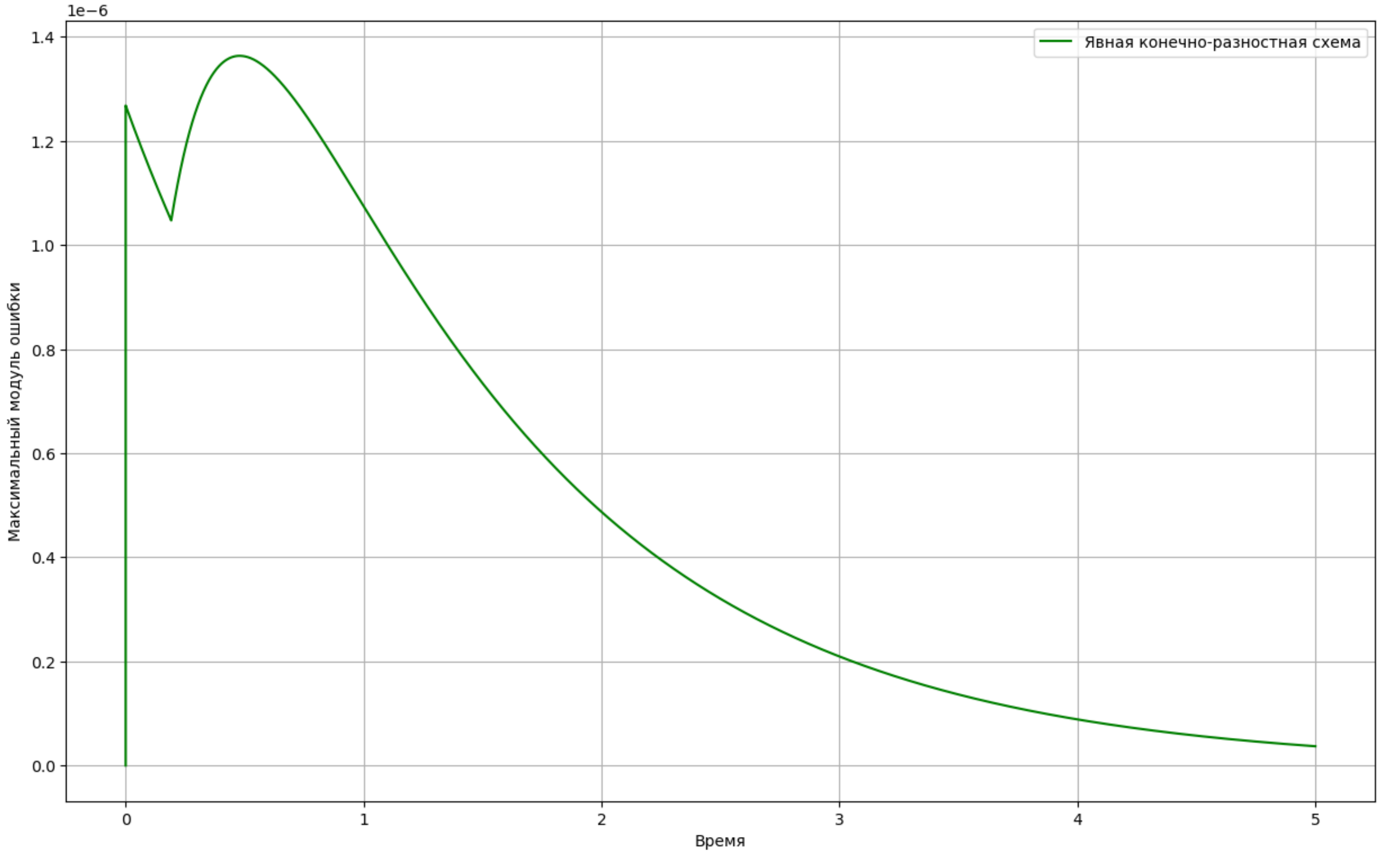
Визуализация решения ДУ с помощью явной конечно-разностной схемы

In [28]:
```
build_numerical_results_graphic(explicit_solution, "Явная конечно-разностная схема", 0.5
```



Визуализация погрешности метода явной конечно-разностной схемы

In [29]:
```
build_errors_graphic(explicit_solution, "Явная конечно-разностная схема", t_start, t_end
```



Неявная конечно-разностная схема

Преобразуем исходное уравнение с производными в уравнение с их численными приближениями. Производную второго порядка в правой части уравнения будем аппроксимировать по значениям нижнего временного слоя.

Необходимо решить систему уравнений для того, чтобы получить значения *u* в одном временном ряду.
Система уравнений имеет вид:

$$\begin{cases} b_1 u_1^{k+1} + c_1 u_2^{k+1} = d_1,\ j = 1 \\ a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j,\ j = 2...(n-2) \\ a_{n-1} u_{n-2}^{k+1} + b_{n-1} u_{n-1}^{k+1} = d_{n-1},\ j = n-1 \end{cases} \text{, где}$$

$a_j = c_j = \sigma = \frac{a\tau}{h^2}$
$b_j = -2\sigma - 1$
$d_j = -u_j^k,\ j = 2...(n-2)$
$d_1 = -\sigma\phi_0(t^{k+1}) - u_1^k$
$d_{n-1} = -\sigma\phi_1(t^{k+1}) - u_{n-1}^k$

Данная система уравнений представляет собой трёхдиагональную СЛАУ, которую можно решить, используя метод прогонки.

Метод прогонки

In [30]:
```
def run_through_method(A, b):
    n = len(A)
```

```
    v = [0 for _ in range(n)]
    u = [0 for _ in range(n)]
    v[0] = A[0][1] / -A[0][0]
    u[0] = b[0] / A[0][0]
    for i in range(1, n - 1):
        v[i] = A[i][i + 1] / (-A[i][i] - A[i][i - 1] * v[i - 1])
        u[i] = (A[i][i - 1] * u[i - 1] - b[i]) / (-A[i][i] - A[i][i - 1] * v[i - 1])
    v[n - 1] = 0
    u[n - 1] = (A[n - 1][n - 2] * u[n - 2] - b[n - 1]) / (-A[n - 1][n - 1] - A[n - 1][n

    x = [0 for _ in range(n)]
    x[n - 1] = u[n - 1]
    for i in range(n - 1, 0, -1):
        x[i - 1] = v[i - 1] * x[i] + u[i - 1]
    return np.array(x)
```

In [31]:
```
def implicit_finite_difference_method(x_start, x_end, t_start, t_end, a, h, sigma, phi_0
    assert sigma < (1/2)
    tau = sigma * h**2 / a
    x = np.arange(x_start, x_end, h)
    t = np.arange(t_start, t_end, tau)
    U = np.zeros((len(t), len(x)))

    # подсчитываем значения на нижней границе (t = 0)
    for i in range(len(x)):
        U[0][i] = psi(x[i])

    for dt in range(1, len(t)):
        A = np.zeros((len(x) - 2, len(x) - 2))
        A[0][0] = -2 * sigma - 1
        A[0][1] = sigma
        for i in range(1, len(A) - 1):
            A[i][i - 1] = sigma
            A[i][i] = -2 * sigma - 1
            A[i][i + 1] = sigma
        A[-1][-2] = sigma
        A[-1][-1] = -2 * sigma - 1

        b = -U[dt - 1][1:-1]
        # подсчитываем значения, используя граничные условия
        b[0] -= sigma * phi_0(t[dt], a)
        b[-1] -= sigma * phi_1(t[dt], a)

        U[dt][0] = phi_0(t[dt], a)
        U[dt][-1] = phi_1(t[dt], a)
        # метод прогонки
        U[dt][1:-1] = run_through_method(A, b)
    return U
```

In [32]:
```
implicit_solution = implicit_finite_difference_method(x_start, x_end, t_start, t_end, a,
```

In [33]:
```
implicit_solution.shape
```

Out[33]:
```
(135136, 315)
```

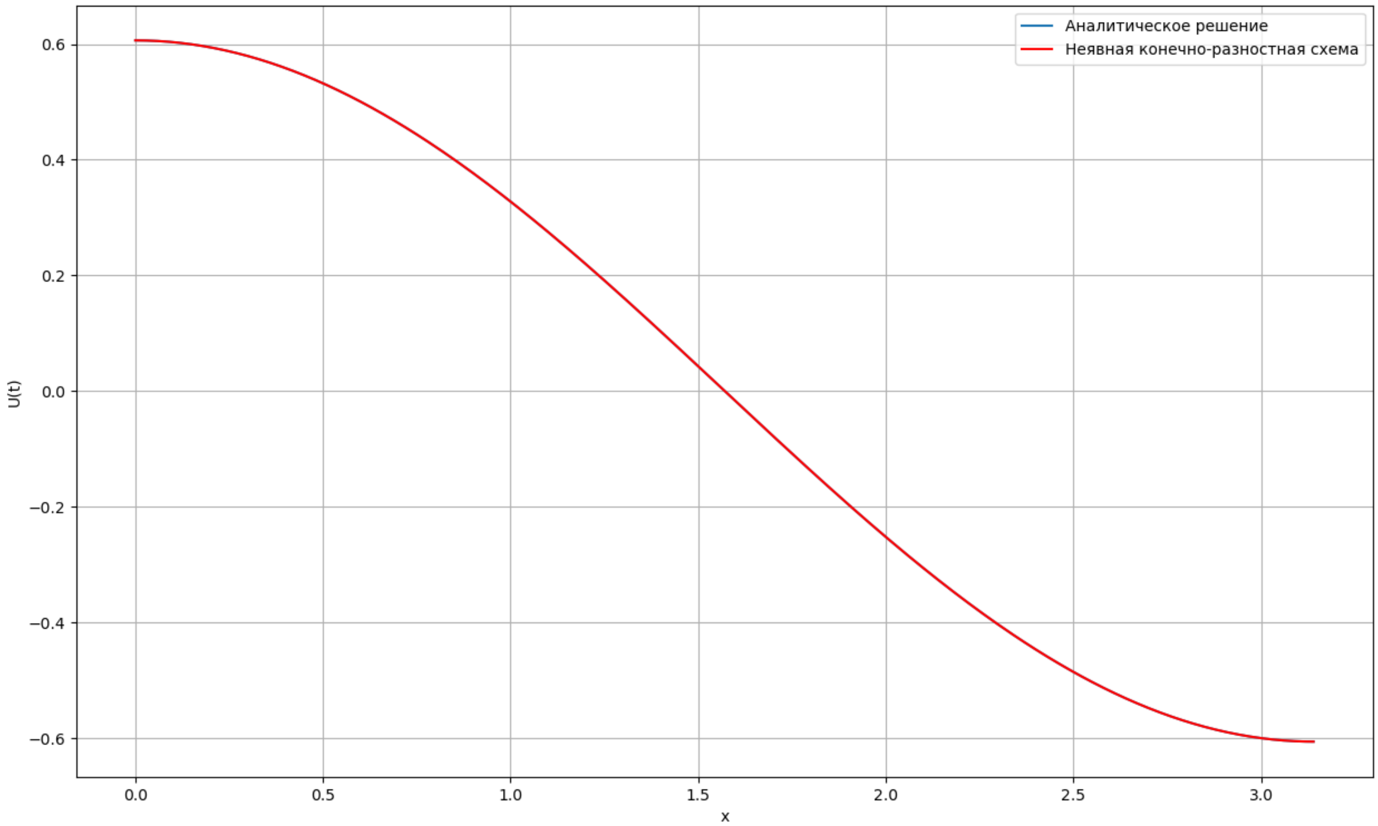
Максимальный модуль ошибки

In [34]:
```
print(f'Максимальный модуль ошибки = {max_abs_error(implicit_solution, anal_solution)}')
```

```
Максимальный модуль ошибки = 4.034005250619366e-06
```

Визуализация решения ДУ с помощью неявной конечно-разностной схемы

In [35]:
```
build_numerical_results_graphic(implicit_solution, "Неявная конечно-разностная схема", 0
```



Визуализация погрешности метода явной конечно-разностной схемы

In [36]:
```
build_errors_graphic(implicit_solution, "Неявная конечно-разностная схема", t_start, t_e
```

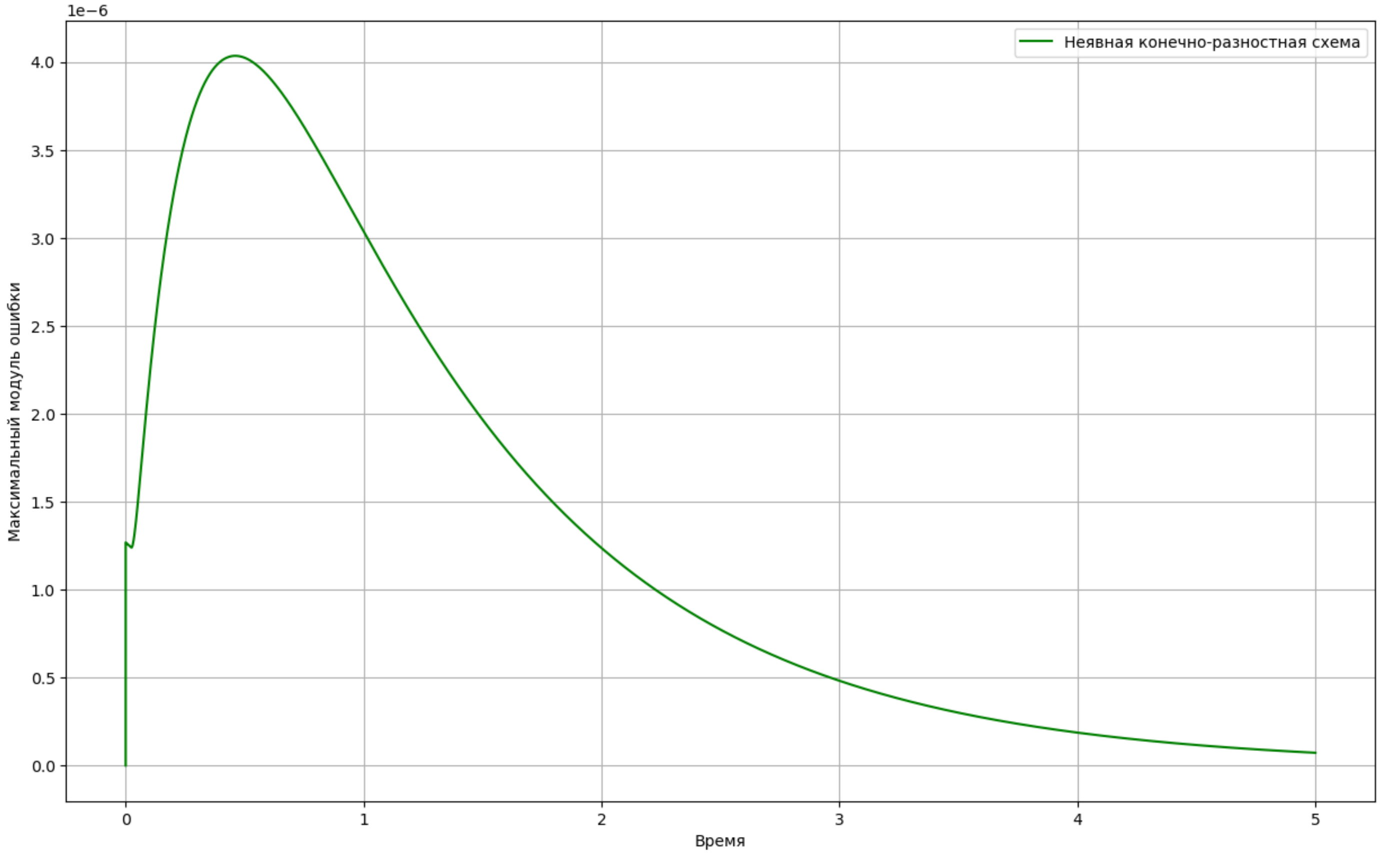


Схема Кранка-Николсона

Скорректируем трехдиагональную СЛАУ из предыдущего метода с той разницей, что производную второго порядка будем аппроксимировать, используя выпуклую комбинацию (по значениям нижнего и верхнего временного слоя).

Выпуклая комбинация - ЛК, при которой коэффициенты ≥ 0 и в сумме дают единицу. Будем использовать коэффициенты θ и $1-\theta$. При $\theta = \frac{1}{2}$ имеем схему Кранка-Николсона.

Скорректированя система уравнений будет иметь вид:

$$\begin{cases} b_1 u_1^{k+1} + c_1 u_2^{k+1} = d_1,\ j = 1 \\ a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j,\ j = 2...(n-2) \\ a_{n-1} u_{n-2}^{k+1} + b_{n-1} u_{n-1}^{k+1} = d_{n-1},\ j = n-1 \end{cases} \text{, где}$$

$a_j = c_j = \theta\sigma$
$b_j = -2\theta\sigma - 1$
$d_j = -(1-\theta)\sigma(u_{j-1}^k - 2u_j^k + u_{j+1}^k) - u_j^k,\ j = 2...(n-2)$
$d_1 = -\sigma\phi_0(t^{k+1}) - u_1^k$
$d_{n-1} = -\sigma\phi_1(t^{k+1}) - u_{n-1}^k$

Данная система уравнений представляет собой трёхдиагональную СЛАУ, которую можно решить, используя метод прогонки

In [37]:
```
def crank_nicolson_method(x_start, x_end, t_start, t_end, a, h, sigma, phi_0, phi_1, psi
    assert sigma < (1/2)
    tau = sigma * h**2 / a
    x = np.arange(x_start, x_end, h)
```

```
    t = np.arange(t_start, t_end, tau)
    U = np.zeros((len(t), len(x)))

    # подсчитываем значения на нижней границе (t = 0)
    for i in range(len(x)):
        U[0][i] = psi(x[i])

    for dt in range(1, len(t)):
        A = np.zeros((len(x) - 2, len(x) - 2))
        A[0][0] = -2 * sigma * theta - 1
        A[0][1] = sigma * theta
        for i in range(1, len(A) - 1):
            A[i][i - 1] = sigma * theta
            A[i][i] = -2 * sigma * theta - 1
            A[i][i + 1] = sigma * theta
        A[-1][-2] = sigma * theta
        A[-1][-1] = -2 * sigma * theta - 1

        b = np.array([-(U[dt - 1][i] + (1 - theta) * sigma * (U[dt - 1][i - 1] - 2 * U[d
        ) for i in range(1, len(U[dt - 1]) - 1)])
        # подсчитываем значения, используя граничные условия
        b[0] -= sigma * theta * phi_0(t[dt], a)
        b[-1] -= sigma * theta * phi_1(t[dt], a)

        U[dt][0] = phi_0(t[dt], a)
        U[dt][-1] = phi_1(t[dt], a)
        # метод прогонки
        U[dt][1:-1] = run_through_method(A, b)

    return U
```

In [38]:
```
crank_nicolson_solution = crank_nicolson_method(x_start, x_end, t_start, t_end, a, h, si
```

In [39]:
```
crank_nicolson_solution.shape
```

Out[39]:
```
(135136, 315)
```

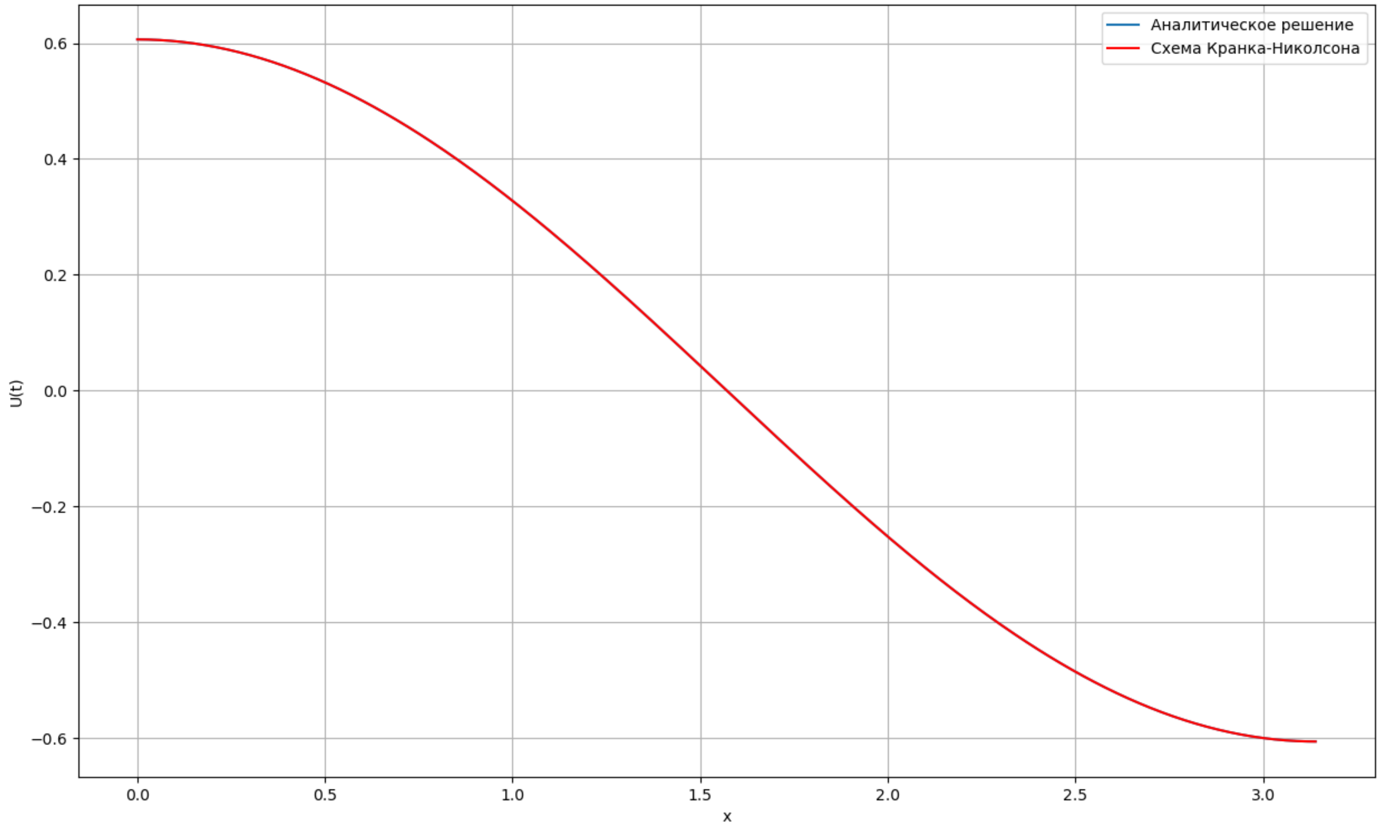
Максимальный модуль ошибки

In [40]:
```
print(f'Максимальный модуль ошибки = {max_abs_error(crank_nicolson_solution, anal_soluti
```

```
Максимальный модуль ошибки = 1.6059616916308528e-06
```

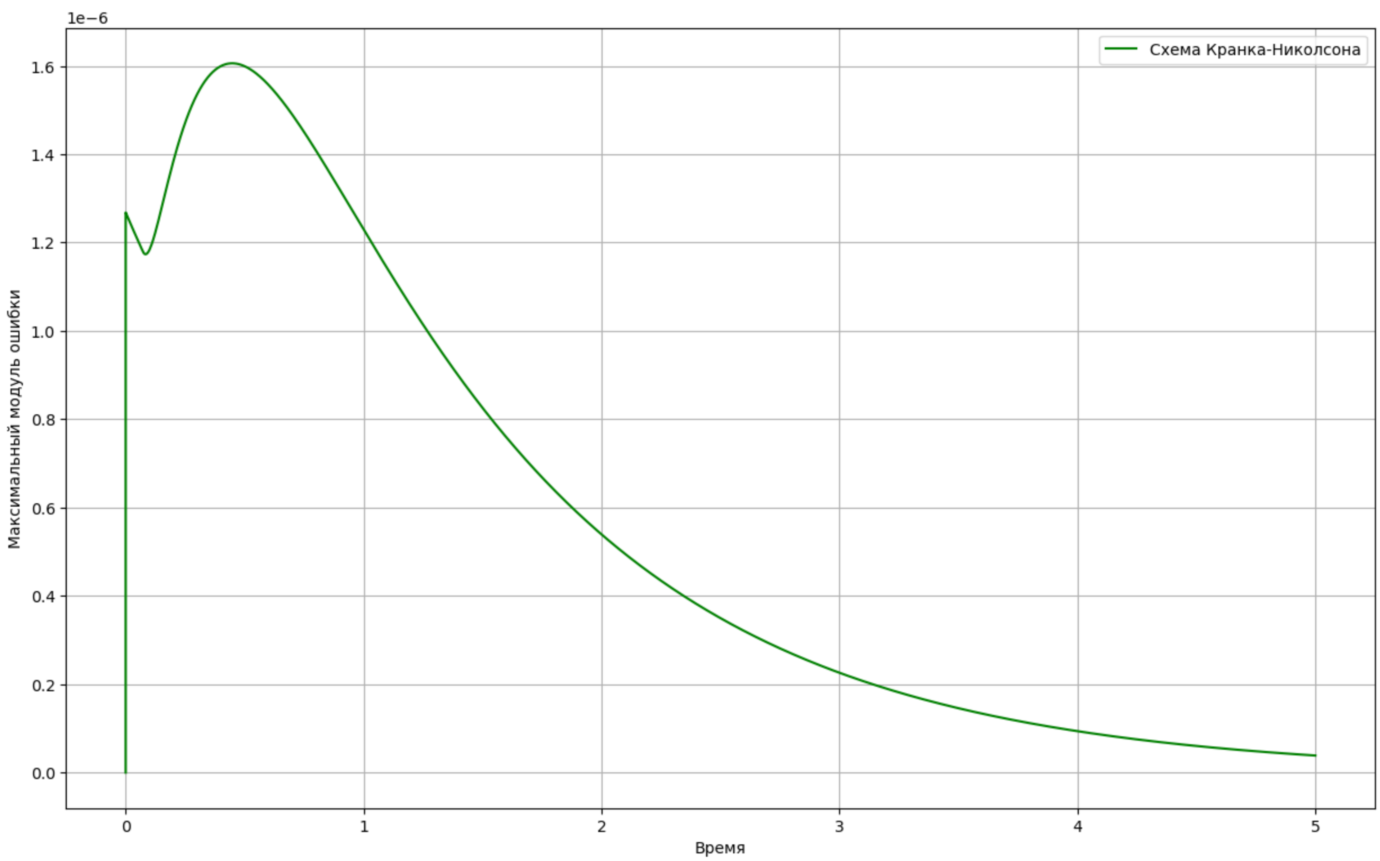
Визуализация решения ДУ с помощью схемы Кранка-Николсона

In [41]:
```
build_numerical_results_graphic(crank_nicolson_solution, "Схема Кранка-Николсона", 0.5,
```



Визуализация погрешности метода схемы Кранка-Николсона

In [42]:
```
build_errors_graphic(crank_nicolson_solution, "Схема Кранка-Николсона", t_start, t_end,
```



Сравнение численных методов с аналитическим решением

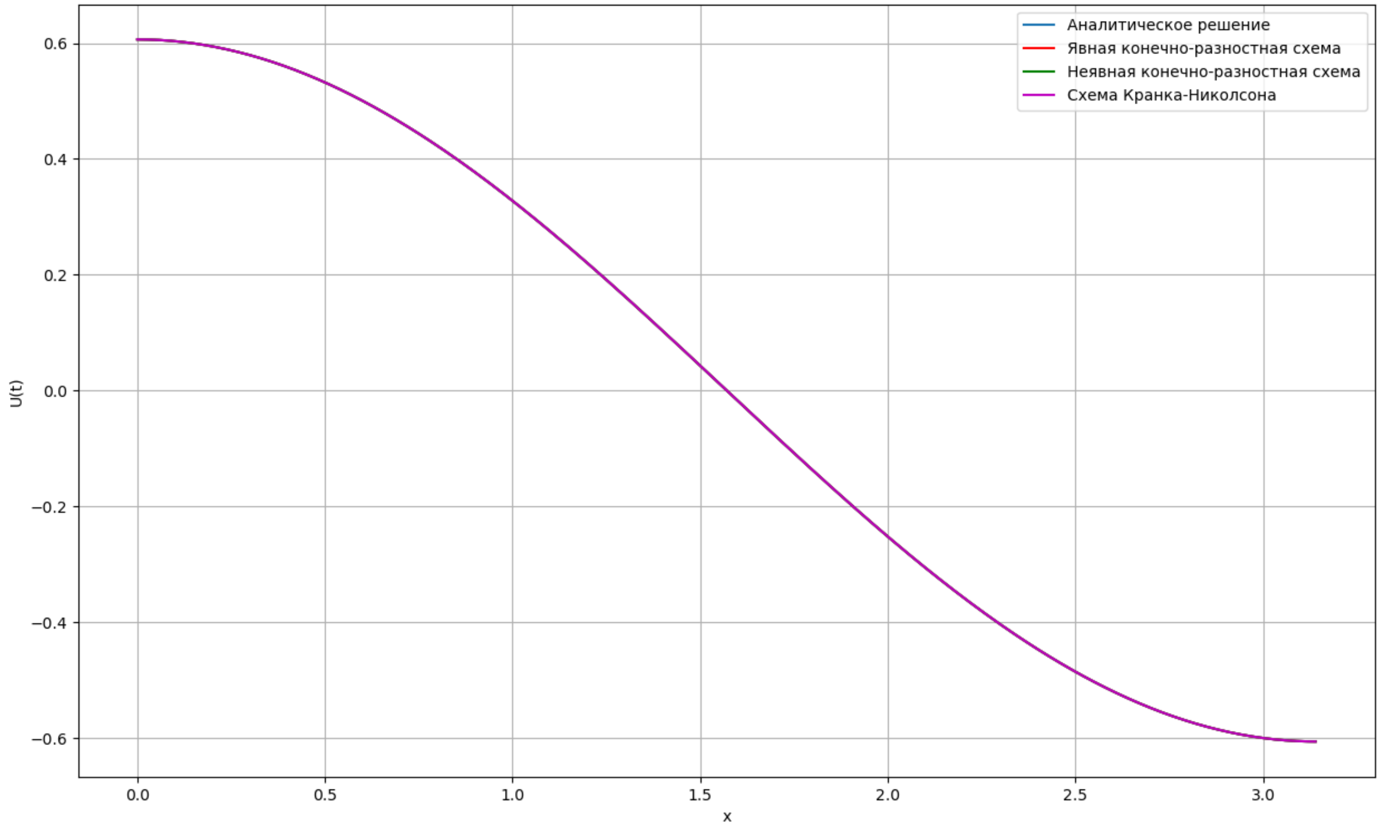
In [43]:
```
def build_all_numerical_results_graphic(sol1, sol2, sol3 , m_n1, m_n2, m_n3, time, x_sta
    tau = sigma * h**2 / a
```

```
    x = np.arange(x_start, x_end, h)
    times = np.arange(t_start, t_end, tau)
    cur_t_id = abs(times - time).argmin()

    plt.figure(figsize=(15, 9))
    plt.plot(x, anal_solution[cur_t_id], label='Аналитическое решение')
    plt.plot(x, sol1[cur_t_id], label=m_n1, color='r')
    plt.plot(x, sol2[cur_t_id], label=m_n2, color='g')
    plt.plot(x, sol3[cur_t_id], label=m_n3, color='m')

    plt.xlabel('x')
    plt.ylabel('U(t)')
    plt.legend()
    plt.grid()
    plt.show()
```

In [44]:
```
def build_all_errors_graphic(sol1, sol2, sol3, m_n1, m_n2, m_n3, t_start, t_end, h, sigm
    tau = sigma * h**2 / a
    t = np.arange(t_start, t_end, tau)

    plt.figure(figsize=(15, 9))
    max_abs_errors1 = np.array([max_abs_error(sol1[i], anal_solution[i]) for i in range(
    max_abs_errors2 = np.array([max_abs_error(sol2[i], anal_solution[i]) for i in range(
    max_abs_errors3 = np.array([max_abs_error(sol3[i], anal_solution[i]) for i in range(
    plt.plot(t, max_abs_errors1, label=m_n1, color='g')
    plt.plot(t, max_abs_errors2, label=m_n2, color='r')
    plt.plot(t, max_abs_errors3, label=m_n3, color='m')

    plt.xlabel('Время')
    plt.ylabel('Максимальный модуль ошибки')

    plt.legend()
    plt.grid()
    plt.show()
```

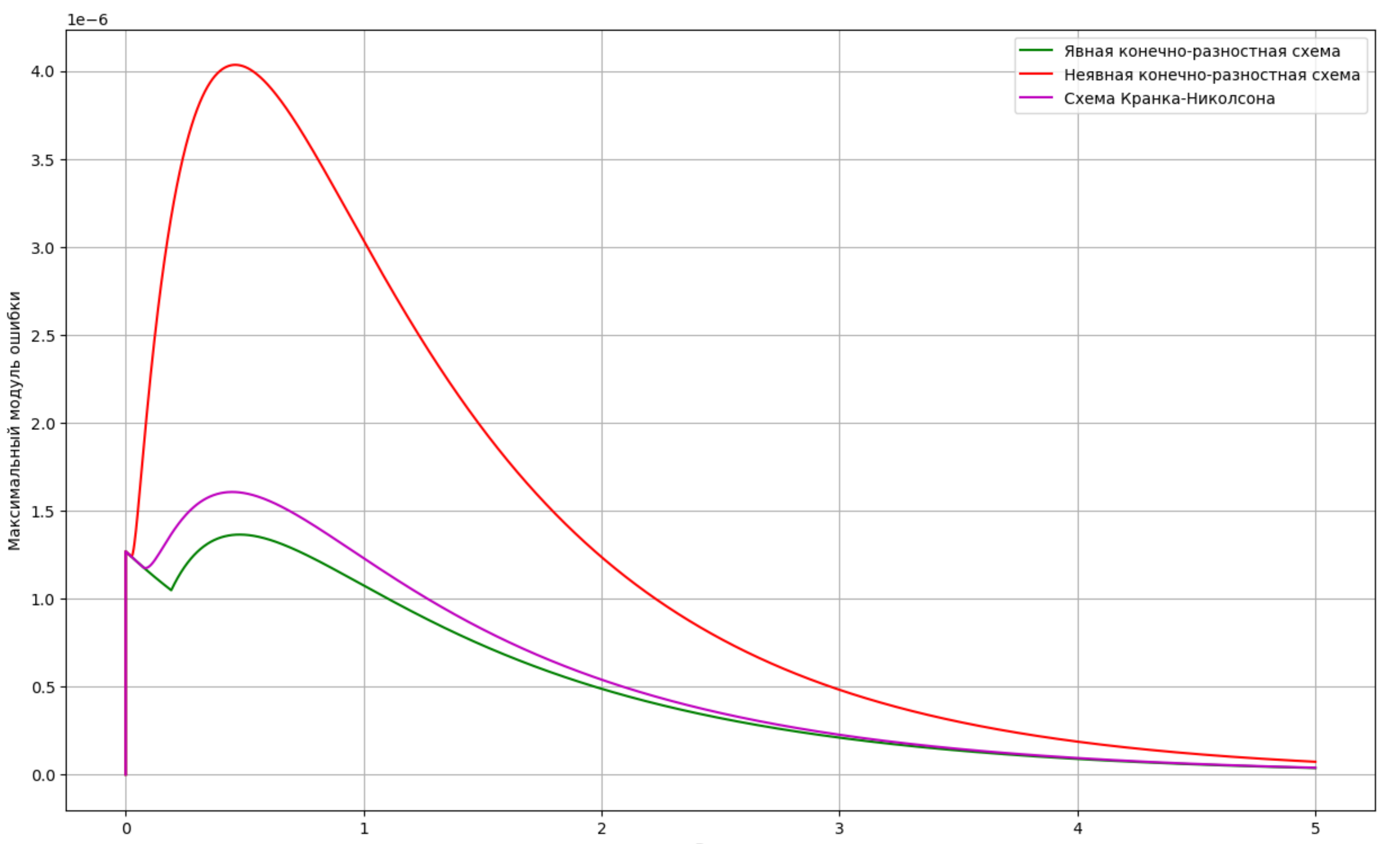
Визуализация результатов работы численных методов

In [45]:
```
build_all_numerical_results_graphic(explicit_solution, implicit_solution, crank_nicolson
```



Визуализация зависимости погрешности от времени для численных методов

In [46]:
```
build_all_errors_graphic(explicit_solution, implicit_solution, crank_nicolson_solution ,
```



Выводы

В данной лабораторной работе изучил три метода для решения начально-краевой задачи для дифференциального уравнения параболического типа:

- явная конечно-разностная схема;
- неявная конечно-разностная схема;
- схема Кранка-Николсона.

Полученное решение заданного ДУ данными методами имеет довольно большую точность. Построил графики зависимости погрешности от времени для всех численных методов.

В процессе выполнения данной лабораторной работы изучил преимущества и недостатки численных методов.

Для явной конечно-разностной схемы обязательным требованием является условие сходимости метода $\sigma < \frac{1}{2}$, тем не менее, данный метод является простым в реализации.
Неявная конечно-разностная схема обуславливается сложностью вычислений, так как необходимо решать большое количество СЛАУ. Однако для данного метода, как и для схемы Кранка-Николсона ограничение $\sigma < \frac{1}{2}$ не требуется.
Схема Кранка-Николсона "сочетает" в себе два предыдущих метода и вследствие этого имеет наименьшую погрешность, хотя всё также использует сложные вычисления, как неявная конечно-разностная схема.