Отчет по лабораторной работе №5
по курсу «Численные методы»

Студент группы М8О-406Б-19: Суханов Е.А.
Преподаватель: Пивоваров Д.Е.
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

---

1. Тема работы

ЧИСЛЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА. ПОНЯТИЕ О МЕТОДЕ КОНЕЧНЫХ РАЗНОСТЕЙ. ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И КОНЕЧНО-РАЗНОСТНЫЕ СХЕМЫ.

---

2. Цель работы

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 . Исследовать зависимость погрешности от сеточных параметров.

Вариант: 7

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + 0,5\exp(-0,5t)\cos x,$$

$$u_x(0,t) = \exp(-0,5t),$$

$$u_x(\pi,t) = -\exp(-0,5t),$$

$$u(x,0) = \sin x,$$

Аналитическое решение: $U(x,t) = \exp(-0,5t)\sin x$.

---

3. Ход работы

In [18]:
```python
# Импортируем нужные модули
import numpy as np
import matplotlib.pyplot as plt
```

In [5]:
```python
# Напишем класс для описания задачи
class Task():
    def __init__(self, u0, ul, l, ur, r, f, a, b, c):
        self.u0 = u0
        self.ul = ul
        self.l = l
        self.ur = ur
        self.r = r
        self.f = f
        self.a = a
        self.b = b
        self.c = c

# Вариант 7
task = Task(
    u0=lambda x: np.sin(x),
    ul=lambda t: np.exp(-0.5*t),
    l=0,
    ur=lambda t: -np.exp(-0.5*t),
    r=np.pi,
    f=lambda x,t: 0.5*np.exp(-0.5*t)*np.sin(x),
    a=1,
    b=0,
    c=0,
)

analytic_func = lambda x,t: np.exp(-0.5*t)*np.sin(x)

T_RES = 1000
H_RES = 20
END_TIME = 10
```

In [6]:
```python
# Аналитическое решение
def analytic(l_bound, r_bound, func, end_time, t_res, h_res):
    h = (r_bound - l_bound) / h_res
    tau = end_time / t_res
    u = np.zeros(shape=(t_res, h_res))

    for t_itr in range(0, t_res):
        for x in range(0, h_res):
            u[t_itr][x] = func(l_bound + x * h, t_itr * tau)

    return u
```

In [7]:

```
# Явная схема
def explicit(task: Task, end_time, t_res, h_res, approx='1p1'):
        h = (task.r - task.l) / h_res
        tau = end_time / t_res
        sigma = (task.a * tau)/(h**2)

        if sigma > 0.5:
            raise ValueError(f"Sigma: {sigma}")

        u = np.zeros((t_res, h_res))
        u[0] = task.u0(np.arange(task.l, task.r, h))

        for k in range(1, t_res):
            for j in range(1, h_res - 1):
                u[k][j] = sigma * u[k - 1][j + 1] + (1 - 2 * sigma) * u[k - 1][j] +
                u[k][j] += tau * task.f(task.l + j * h, k * tau)

            if approx == '1p2':
                u[k][0] = u[k][1] - h * task.ul(k * tau)
                u[k][-1] = u[k][-2] + h * task.ur(k * tau)

            elif approx == '2p2':
                u[k][0] = (u[k][1] - h * task.ul(k * tau) + (h ** 2 / (2 * tau) * u
                u[k][-1] = (u[k][-2] + h * task.ur(k * tau) + (h ** 2 / (2 * tau) *


            elif approx == '2p3':
                u[k][0] = (task.ul(k * tau) + u[k][2] / (2 * h) - 2 * u[k][1] / h)
                u[k][-1] = (task.ur(k * tau) - u[k][-3] / (2 * h) + 2 * u[k][-2] /

        return u
```

In [8]:
```
# Метод прогонки
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

# Неявная схема
def implicit(task: Task, end_time, t_res, h_res, approx='1p2'):
    h = (task.r - task.l) / h_res
    tau = end_time / t_res
    sigma = (task.a * tau)/(h**2)

    a = np.zeros(h_res)
    b = np.zeros(h_res)
    c = np.zeros(h_res)
    d = np.zeros(h_res)
    u = np.zeros((t_res, h_res))
    u[0] = task.u0(np.arange(task.l, task.r, h))

    for k in range(1, t_res):
        for j in range(1, h_res - 1):
            a[j] = sigma
            b[j] = -(1 + 2 * sigma)
            c[j] = sigma
            d[j] = -u[k - 1][j] - tau * task.f(task.l + j * h, k * tau)

        if approx == '1p2':
            a[0] = 0
            b[0] = -1 / h
            c[0] = 1 / h
            d[0] = task.ul(k * tau)
            a[-1] = -1 / h
            b[-1] = 1 / h
            c[-1] = 0
            d[-1] = task.ur(k * tau)

        elif approx == '2p2':
            b[0] = 2 * task.a ** 2 / h + h / tau
            c[0] = - 2 * task.a ** 2 / h
            d[0] = (h / tau) * u[k - 1][0] - task.ul(k * tau) * 2 * task.a ** 2
```

```
            a[-1] = -2 * task.a ** 2 / h
            b[-1] = 2 * task.a ** 2 / h + h / tau
            d[-1] = (h / tau) * u[k - 1][-1] + task.ur(k * tau) * 2 * task.a ** 2
        elif approx == '2p3':
            k0 = 1 / (2 * h) / c[1]
            b[0] = (-3 / (2 * h) + a[1] * k0)
            c[0] = 2 / h + b[1] * k0
            d[0] = task.ul(k* tau) + d[1] * k0
            k1 = -(1 / (h * 2)) / a[-2]
            a[-1] = (-2 / h) + b[-2] * k1
            b[-1] = (3 / (h * 2)) + c[-2] * k1
            d[-1] = task.ur(k* tau) + d[-2] * k1

        u[k] = tridiagonal_matrix_algorithm(a, b, c, d)

    return u
```

In [9]:

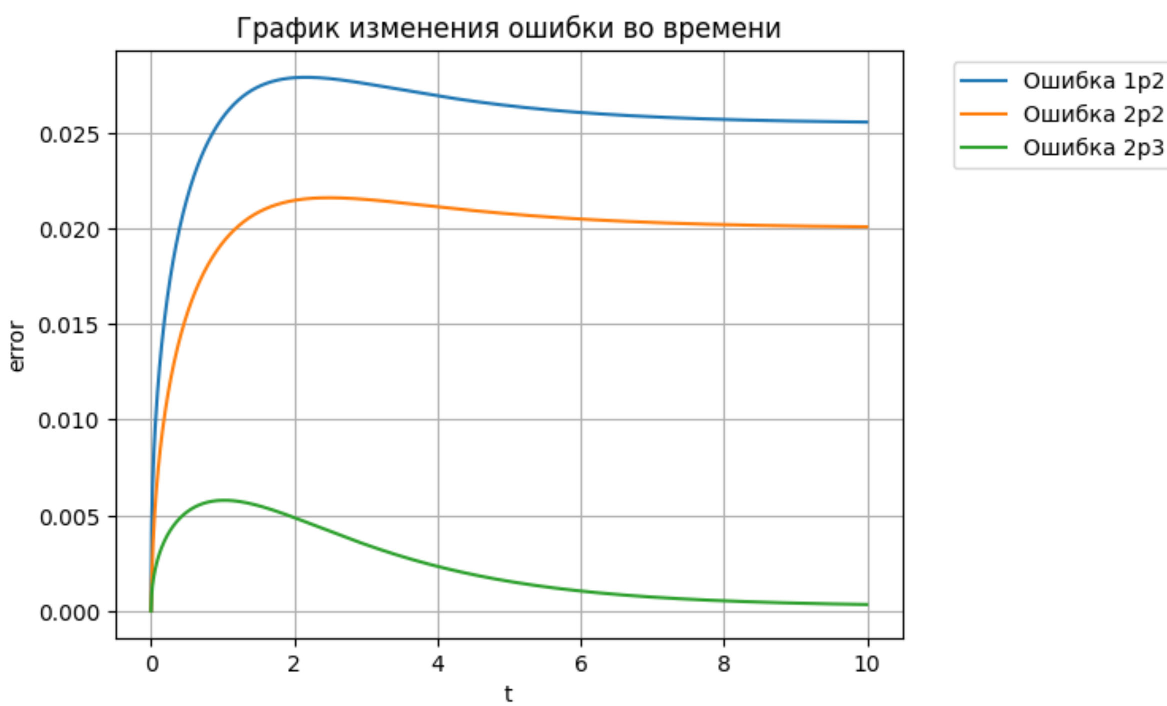
```
# Кобинированный метод
def combined_method(task: Task, end_time, t_res, h_res, theta=0.5, approx='1p2'):
    h = (task.r - task.l) / h_res
    tau = end_time / t_res
    sigma = (task.a * tau)/(h**2)

    a = np.zeros(h_res)
    b = np.zeros(h_res)
    c = np.zeros(h_res)
    d = np.zeros(h_res)
    tmp_imp = np.zeros(h_res)
    u = np.zeros((t_res, h_res))

    u[0] = task.u0(np.arange(task.l, task.r, h))

    for k in range(1, t_res):
        for j in range(1, h_res - 1):
            a[j] = sigma * theta
            b[j] = -(1 + 2 * sigma * theta)
            c[j] = sigma * theta
            d[j] = -u[k - 1][j] - tau * task.f(task.l + j * h, k * tau)
            d[j] -= (1-theta)*sigma*(u[k - 1][j + 1] - 2*u[k - 1][j] + u[k - 1][j -

        if approx == '1p2':
            a[0] = 0
            b[0] = -1 / h
            c[0] = 1 / h
            d[0] = task.ul(k * tau)
            a[-1] = -1 / h
            b[-1] = 1 / h
            c[-1] = 0
            d[-1] = task.ur(k * tau)

        elif approx == '2p2':
            b[0] = 2 * task.a ** 2 / h + h / tau
            c[0] = - 2 * task.a ** 2 / h
            d[0] = (h / tau) * u[k - 1][0] - task.ul(k * tau) * 2 * task.a ** 2
            a[-1] = -2 * task.a ** 2 / h
            b[-1] = 2 * task.a ** 2 / h + h / tau
```

```
            d[-1] = (h / tau) * u[k - 1][-1] + task.ur(k * tau) * 2 * task.a ** 2
        elif approx == '2p3':
            k0 = 1 / (2 * h) / c[1]
```

In [10]:
```python
# Вывод графика ошибки
def draw_error(analytic, t_end, numericals, suffix_labels):

    t_res = analytic.shape[0]
    t = np.arange(0, t_end, t_end/t_res)
    for n,l in zip(numericals, suffix_labels):
        err = np.max(np.abs(analytic - n), axis=1)
        print(f"mean err {l}: {np.mean(err)}")
        plt.plot(t, err, label = f'Ошибка {l}')
    plt.legend(bbox_to_anchor = (1.05, 1), loc = 'upper left')
    plt.title('График изменения ошибки во времени')
    plt.xlabel('t')
    plt.ylabel('error')
    plt.grid(True)
    plt.show()
```

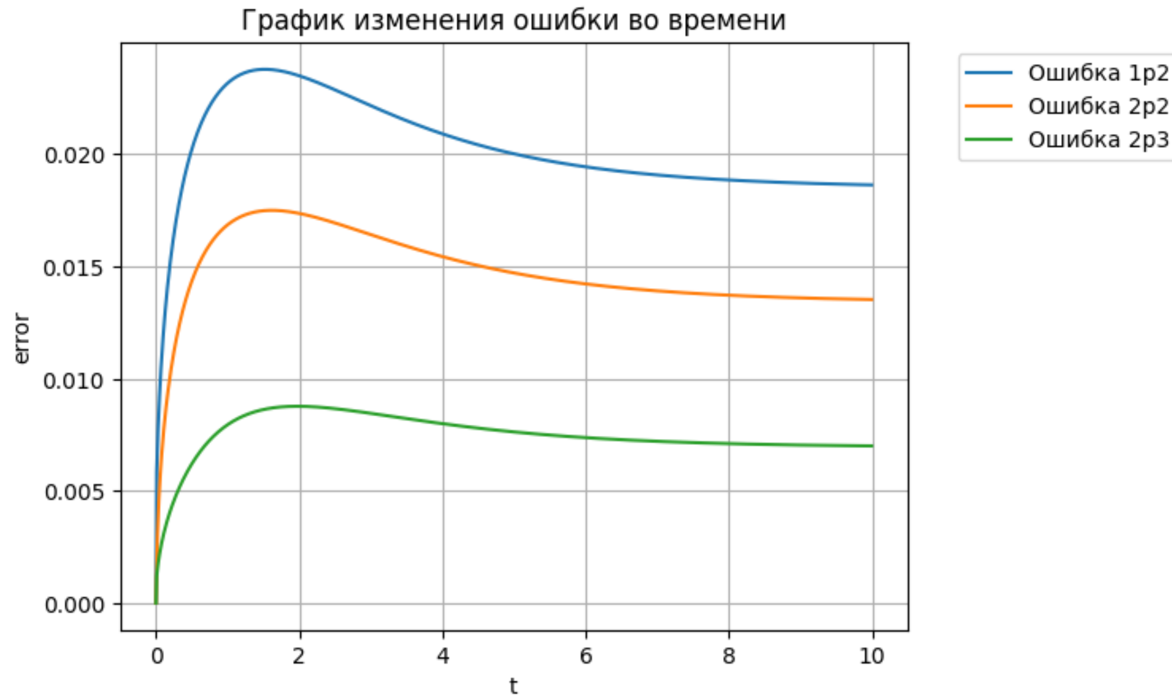
In [11]:
```python
analytic_nodes = analytic(task.l, task.r, analytic_func, END_TIME, T_RES, H_RES)
explicit_nodes_12 = explicit(task, END_TIME, T_RES, H_RES, approx='1p2')
explicit_nodes_22 = explicit(task, END_TIME, T_RES, H_RES, approx='2p2')
explicit_nodes_23 = explicit(task, END_TIME, T_RES, H_RES, approx='2p3')

draw_error(analytic_nodes,  END_TIME, [explicit_nodes_12, explicit_nodes_22, explic
```

```
mean err 1p2: 0.025763541107382423
mean err 2p2: 0.019995796292286497
mean err 2p3: 0.002268599643971792
```



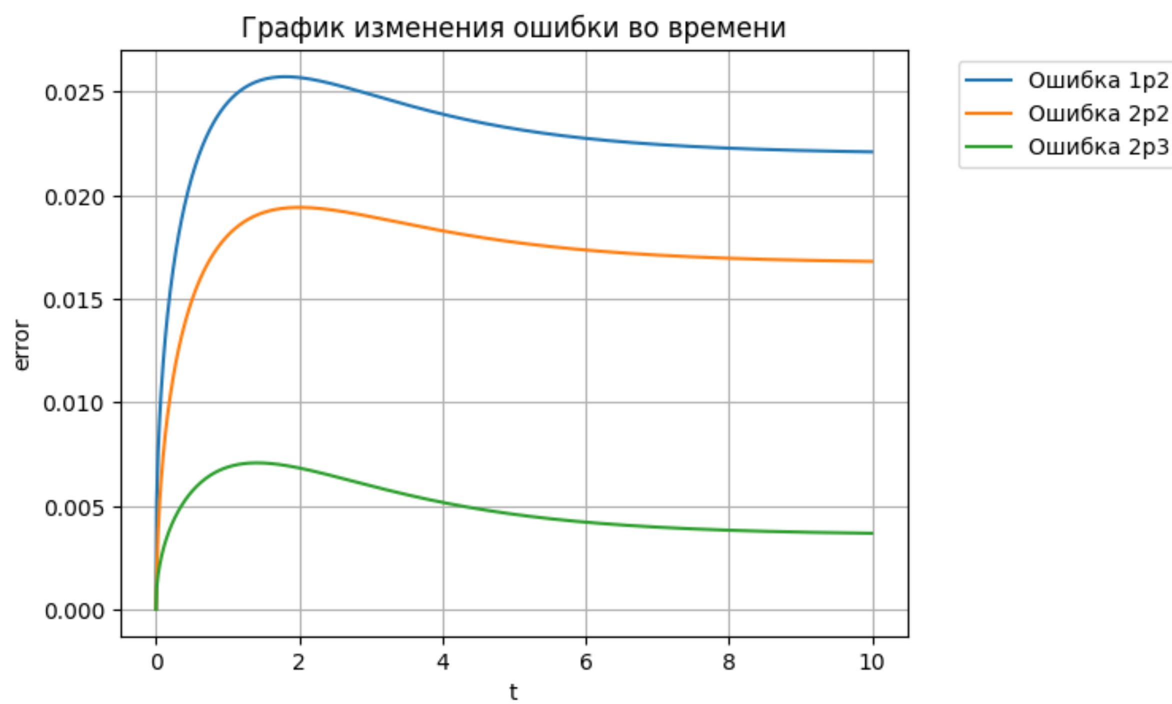
In [12]:
```python
implicit_nodes_11 = implicit(task, END_TIME, T_RES, H_RES, approx='1p2')
implicit_nodes_12 = implicit(task, END_TIME, T_RES, H_RES, approx='2p2')
implicit_nodes_13 = implicit(task, END_TIME, T_RES, H_RES, approx='2p3')

draw_error(analytic_nodes,  END_TIME, [implicit_nodes_11, implicit_nodes_12, implic
```

```
mean err 1p2: 0.02022978428305627
mean err 2p2: 0.014772002296248567
mean err 2p3: 0.007490723652054878
```



In [13]:
```
combined_nodes_11 = combined_method(task, END_TIME, T_RES, H_RES, approx='1p2')
combined_nodes_12 = combined_method(task, END_TIME, T_RES, H_RES, approx='2p2')
combined_nodes_13 = combined_method(task, END_TIME, T_RES, H_RES, approx='2p3')

draw_error(analytic_nodes,  END_TIME, [combined_nodes_11, combined_nodes_12, combin
```

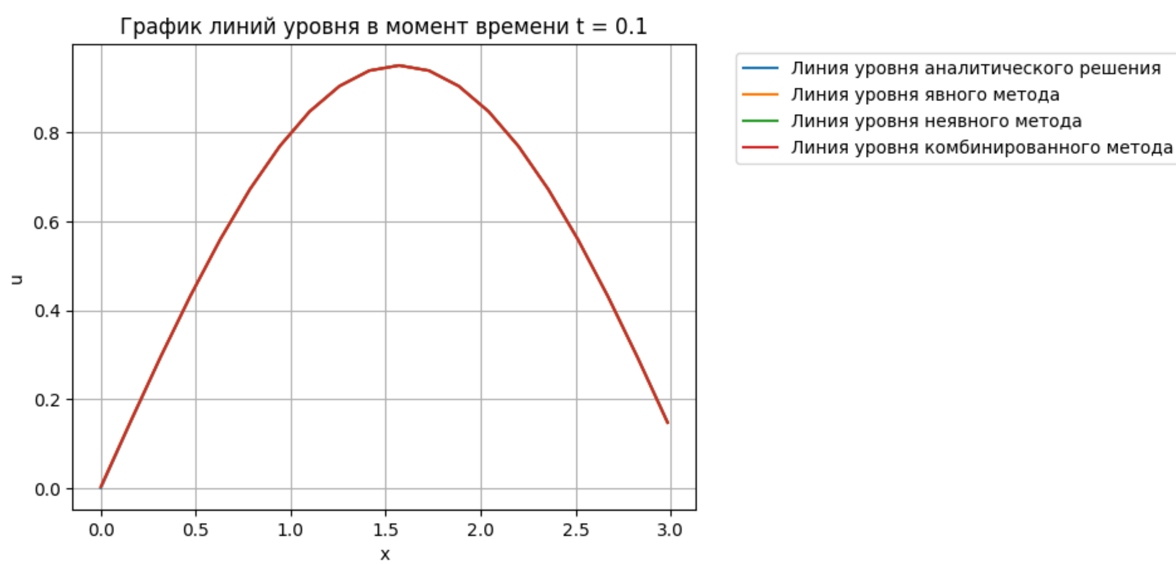
```
mean err 1p2: 0.02299666226416123
mean err 2p2: 0.01738381257119265
mean err 2p3: 0.004879658929933494
```



Посмотрим на линии уровей

In [14]:
```
def draw_cmp_levels(analytic, a, b, end_time, curr_time, numericals, labels):
    t_res = analytic.shape[0]
    h_res = analytic.shape[1]

    h = (b - a) / h_res
    tau = end_time / t_res

    curr_t_idx = int(curr_time / tau)
    curr_time = curr_t_idx * tau

    x = np.arange(a, b, h)
    plt.plot(x, analytic[curr_t_idx], label = f'Линия уровня аналитического решения
    for n,l in zip(numericals, labels):
        plt.plot(x, n[curr_t_idx], label = f'Линия уровня {l}')
    plt.legend(bbox_to_anchor = (1.05, 1), loc = 'upper left')
    plt.title(f'График линий уровня в момент времени t = {curr_time}')
    plt.xlabel('x')
    plt.ylabel('u')
    plt.grid(True)
    plt.show()

def draw_cmp_levels_helper(time):
    explicit_nodes = explicit(task, END_TIME, T_RES, H_RES, approx='2p3')
    implicit_nodes = implicit(task, END_TIME, T_RES, H_RES, approx='2p3')
    combined_nodes = combined_method(task, END_TIME, T_RES, H_RES, approx='2p3')
    draw_cmp_levels(analytic_nodes, task.l, task.r,  END_TIME, time, [explicit_node
```

In [15]:
```
draw_cmp_levels_helper(0.1)
draw_cmp_levels_helper(0.5)
draw_cmp_levels_helper(1)
draw_cmp_levels_helper(3)
draw_cmp_levels_helper(6)
draw_cmp_levels_helper(9)
```



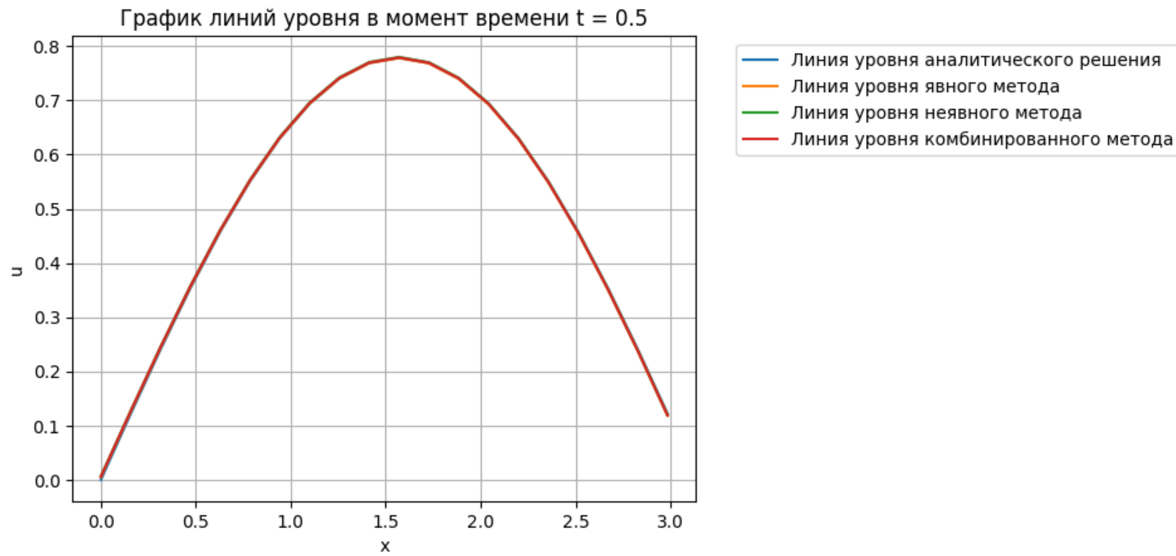


График линий уровня в момент времени t = 0.5
u
x
Линия уровня аналитического решения
Линия уровня явного метода
Линия уровня неявного метода
Линия уровня комбинированного метода

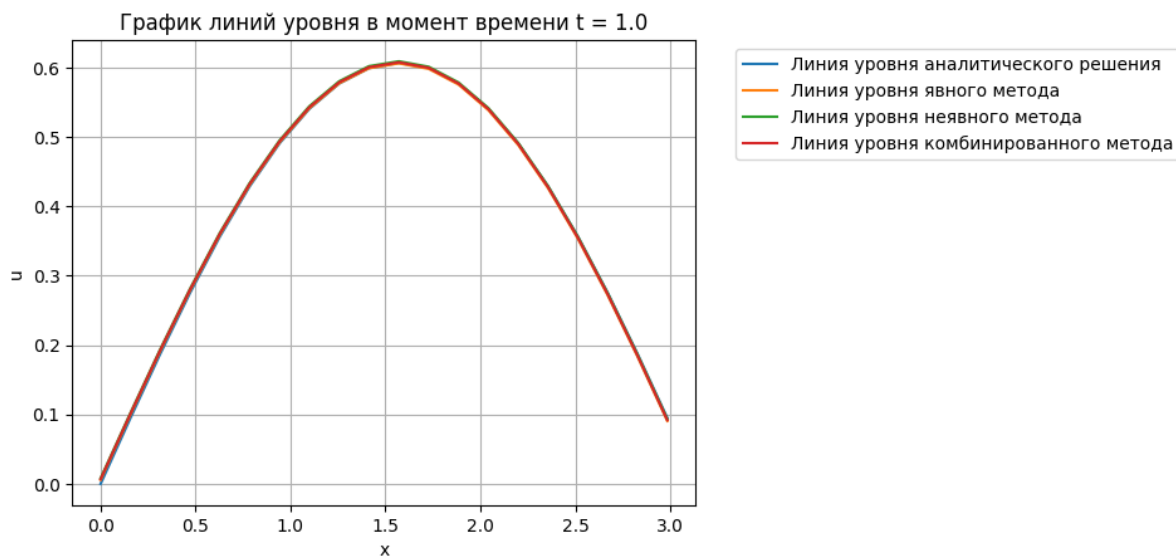


График линий уровня в момент времени t = 1.0
u
x
Линия уровня аналитического решения
Линия уровня явного метода
Линия уровня неявного метода
Линия уровня комбинированного метода

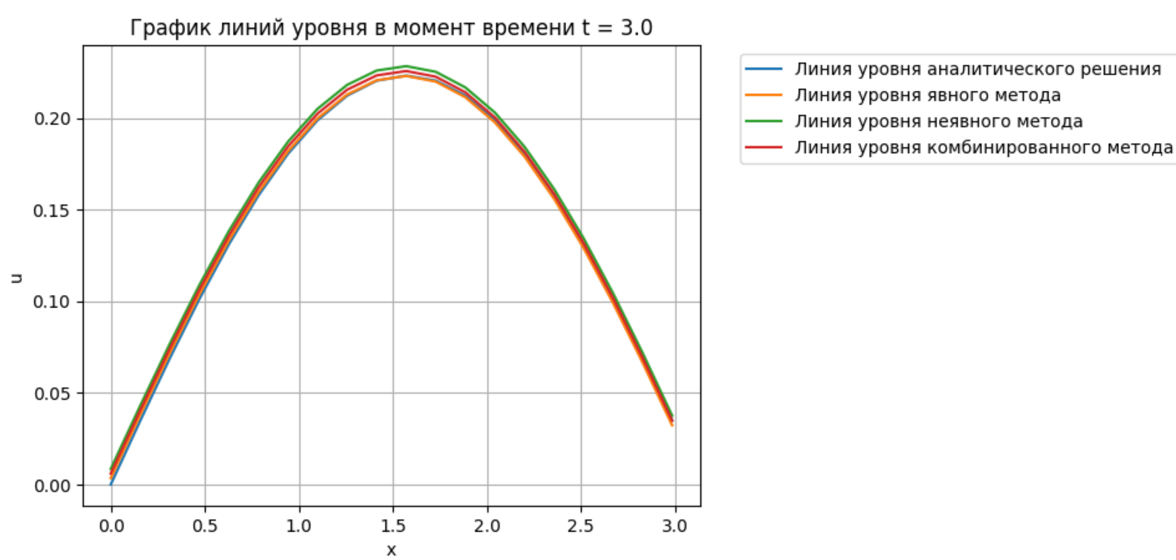
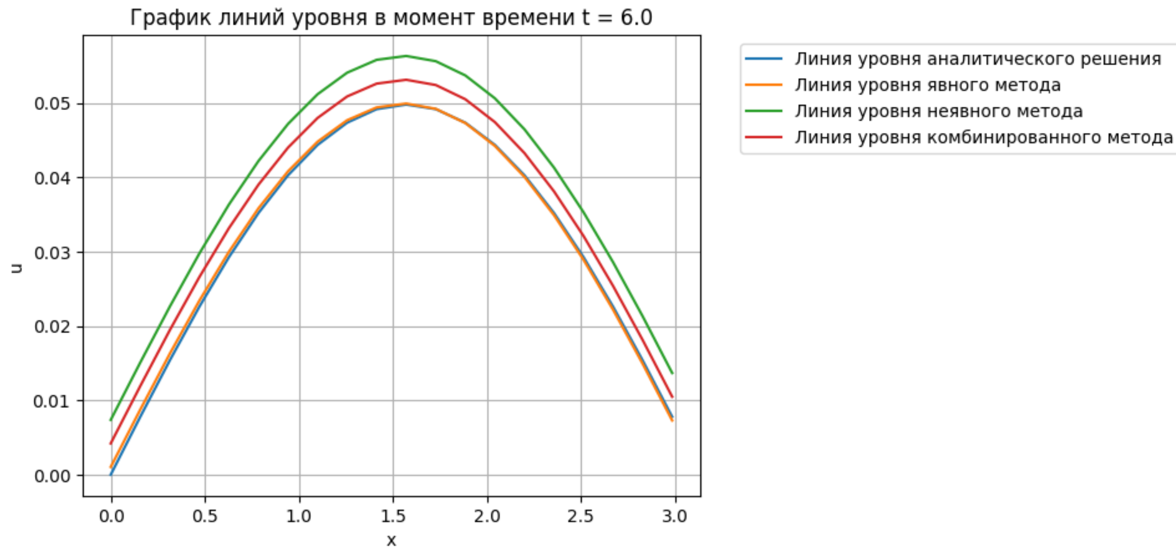
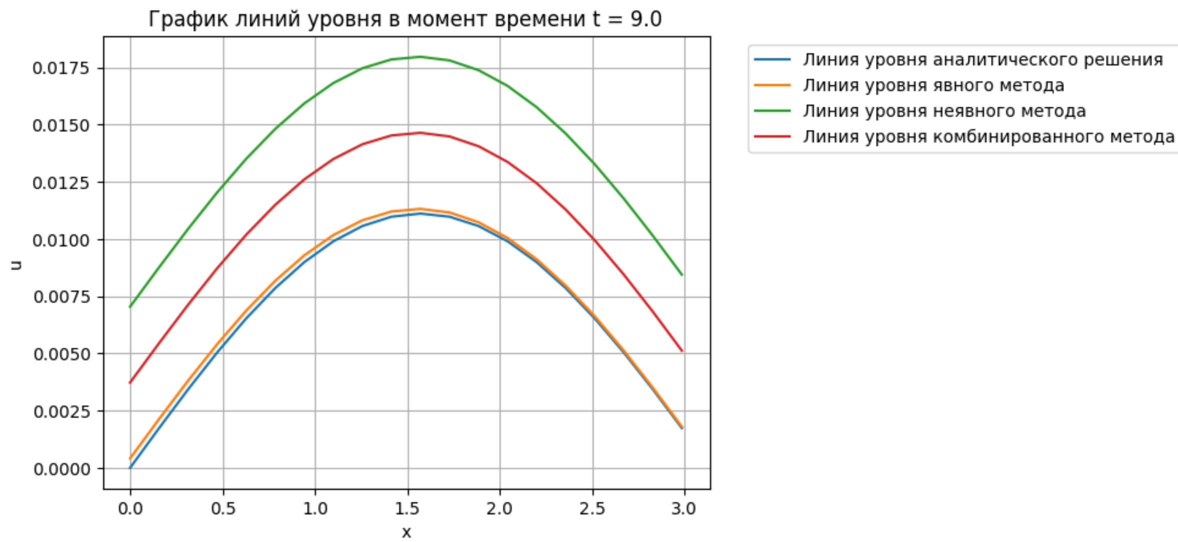


График линий уровня в момент времени t = 3.0
u
x
Линия уровня аналитического решения
Линия уровня явного метода
Линия уровня неявного метода
Линия уровня комбинированного метода

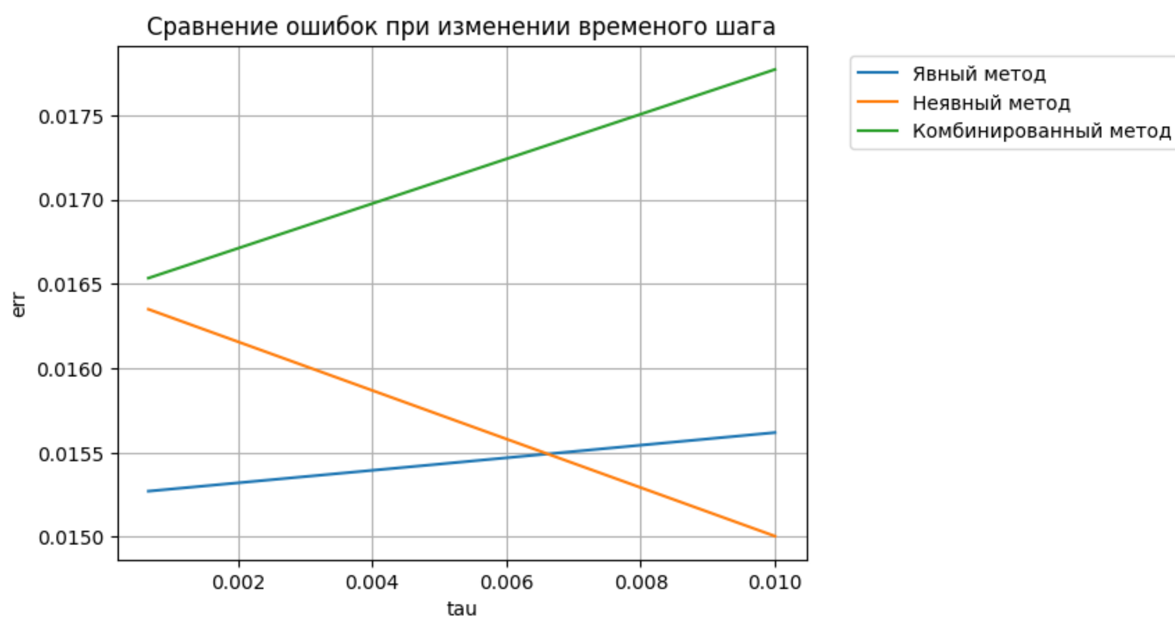




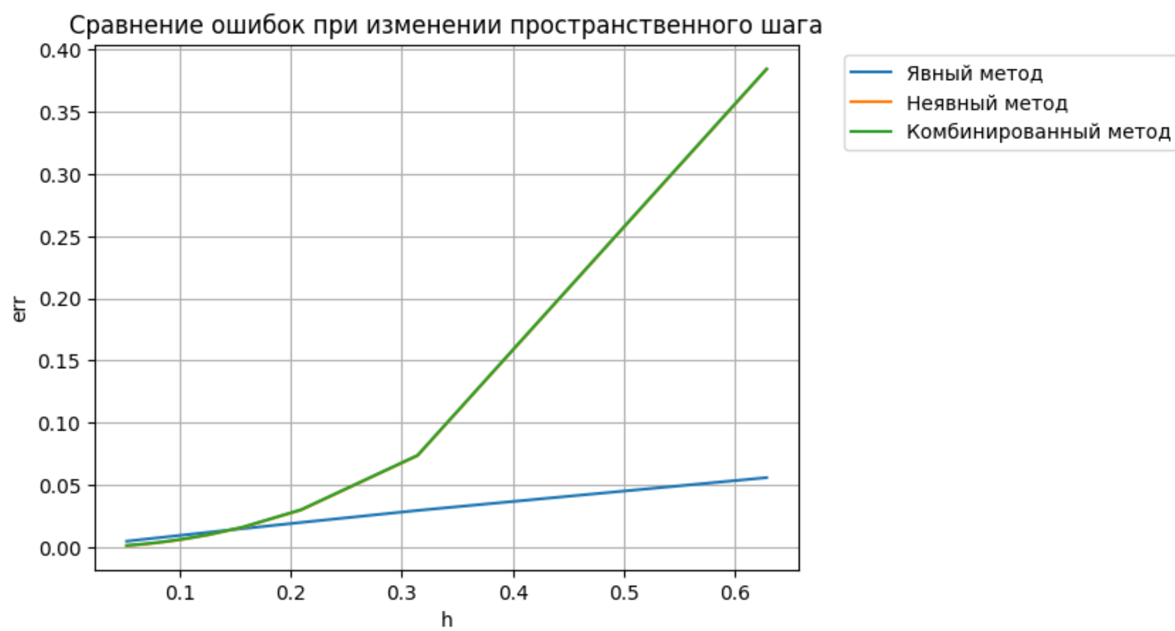
Исследование зависимости погрешности от сеточных параметров tau и h

In [16]:

```
# Исследуем зависимость от tau при неизменном h
# tau зависит от t_res: чем больше t_res => тем меньше tau
H_RES = 20
T_RES_MIN = 1000
T_RES_MAX = 15000
T_RES_STEP = 500
END_TIME = 10
t_res_list = [t_res for t_res in range(T_RES_MIN, T_RES_MAX + T_RES_STEP // 2, T_RE

tau_list = [END_TIME / t_res for t_res in t_res_list]
exp_v = []
imp_v = []
com_v = []
for t_res in t_res_list:
    analytic_nodes = analytic(task.l, task.r, analytic_func, END_TIME, t_res, H_RES
    explicit_nodes = explicit(task, END_TIME, t_res, H_RES)
    implicit_nodes = implicit(task, END_TIME, t_res, H_RES)
    combined_nodes = combined_method(task, END_TIME, t_res, H_RES)
    get_err = lambda nodes: (np.mean(np.abs(analytic_nodes - nodes)))
    exp_v.append(get_err(explicit_nodes))
    imp_v.append(get_err(implicit_nodes))
    com_v.append(get_err(combined_nodes))

plt.plot(tau_list, exp_v, label = 'Явный метод')
plt.plot(tau_list, imp_v, label = 'Неявный метод')
plt.plot(tau_list, com_v, label = 'Комбинированный метод')
plt.legend(bbox_to_anchor = (1.05, 1), loc = 'upper left')
plt.title(f'Сравнение ошибок при изменении временого шага')
plt.xlabel('tau')
plt.ylabel('err')
plt.grid(True)
plt.show()
```



In [17]:
```
# Исследуем зависимость от h при неизменном tau
# h зависит от h_res: чем больше h_res => тем меньше h
T_RES = 10000
H_RES_MIN = 5
H_RES_MAX = 60
H_RES_STEP = 5
END_TIME = 10
h_res_list = [h_res for h_res in range(H_RES_MIN, H_RES_MAX + H_RES_STEP // 2, H_RE

h_list = [(task.r - task.l) / h_res for h_res in h_res_list]
exp_v = []
imp_v = []
com_v = []
for h_res in h_res_list:
    analytic_nodes = analytic(task.l, task.r, analytic_func, END_TIME, T_RES, h_res
    explicit_nodes = explicit(task, END_TIME, T_RES, h_res)
    implicit_nodes = implicit(task, END_TIME, T_RES, h_res)
    combined_nodes = combined_method(task, END_TIME, T_RES, h_res)
    get_err = lambda nodes: (np.mean(np.abs(analytic_nodes - nodes)))
    exp_v.append(get_err(explicit_nodes))
    imp_v.append(get_err(implicit_nodes))
    com_v.append(get_err(combined_nodes))

plt.plot(h_list, exp_v, label = 'Явный метод')
plt.plot(h_list, imp_v, label = 'Неявный метод')
plt.plot(h_list, com_v, label = 'Комбинированный метод')
plt.legend(bbox_to_anchor = (1.05, 1), loc = 'upper left')
plt.title(f'Сравнение ошибок при изменении пространственного шага')
plt.xlabel('h')
plt.ylabel('err')
plt.grid(True)
plt.show()
```



4. Выводы

Как видно из сравнений аппроксимаций, 3-х точечная аппроксимация второго порядка дает наименьшую ошибку.

Явный метод показывает наименьшую ошибку, однако на него накладываются ограничения на максимальный шаг сетки. Так как при сигме большей 0.5 метод теряет устойчивость.

Как видно по графикам, уменьшение временного шага уменьшает ошибку явного и комбинированного мтеода Но увеличивает ошибку неявного метода.
Пространственный шаг влияет прямо пропорционально на ошибку.