Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №5 по курсу «Численные методы»

Студент:	Маринин И.С.
Группа:	М8О-408Б-20
Преподаватель:	Пивоваров Д.Е.

Москва, 2023

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: 14

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

$$\begin{cases} u'_x(0,\ t) = \phi_0(t) = e^{-at} \\ u'_x(\pi,\ t) = \phi_l(t) = -e^{-at} \\ u(x,\ 0) = \sin x \end{cases}$$

Аналитическое решение:

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

In [ ]:
```
# conditions:
def phi_0(t, a = 1.0):
    return math.exp(-a*t)

def phi_l(t, a = 1.0):
    return -math.exp(-a*t)

def u_0(x):
    return math.sin(x)

# analytic solve
def u(x, t, a = 1.0):
    return math.exp(-a*t)*math.sin(x)
```

Конечно-разностная схема

Общая концепция

2

Будем решать задачу на заданном промежутке от 0 до l по координате x 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, T и параметрами насыщенности сетки N, K. Тогда размер шага по каждой из координат определяется:

$$h = \frac{l}{N},\ \tau = \frac{T}{K}$$

Считая, что значения функции $u_j^k = u(x_j, t^k)$ для всех координат $x_j = jh,\ \forall j \in \{0, \dots, N\}$ на временном слое $t^k = k\tau,\ k \in \{0, \dots, K-1\}$ известно, попробуем определить значения функции на временном слое t^{k+1} путем разностной апроксимации производной:

$$\frac{\partial u}{\partial t}(x_j, t^k) = \frac{u_j^{k+1} - u_j^k}{\tau}$$

И одним из методов апроксимации второй производной по x:

$$\frac{\partial^2 u}{\partial x^2}(x_j, t^k)$$

In [ ]:
```
class Schema:
    def __init__(self, a = 1, f0 = phi_0, fl = phi_l, u0 = u_0,
                 O = 0.5, l0 = 0, l1 = math.pi, T = 5, aprx_cls = None):
        self.fl = lambda t: fl(t, a)
        self.f0 = lambda t: f0(t, a)
        self.u0 = u0
        self.T = T
        self.l0 = l0
        self.l1 = l1
        self.tau = None
        self.h = None
        self.a = a
        self.O = O
        self.approx = None
        if aprx_cls is not None:
            self._init_approx(aprx_cls)
        self.sigma = None

    def _init_approx(self, a_cls):
        self.approx = a_cls(self.f0, self.fl)

    def set_approx(self, aprx_cls):
        self._init_approx(self, aprx_cls)

    def set_l0_l1(self, l0, l1):
        self.l0 = l0
        self.l1 = l1

    def set_T(self, T):
        self.T = T                    3

    def _compute_h(self, N):
        self.h = (self.l1 - self.l0) / N

    def _compute_tau(self, K):
```

```
        self.tau = self.T / K

    def _compute_sigma(self):
        self.sigma = self.a * self.tau / (self.h * self.h)

    @staticmethod
    def nparange(start, end, step = 1):
        now = start
        e = 0.00000000001
        while now - e <= end:
            yield now
            now += step

    def _compute_line(self, t, x, last_line):
        pass

    def __call__(self, N=30, K=110):
        # compute t and h
        N, K = N-1, K-1
        self._compute_tau(K)
        self._compute_h(N)
        self._compute_sigma()
        ans = []
        # compute x:
        x = list(self.nparange(self.l0, self.l1, self.h))
        # compute first line
        last_line = list(map(self.u0, x))
        # add copy
        ans.append(list(last_line))
        X = []
        Y = []
        X.append(x)
        Y.append([0.0 for _ in x])
        # main loop
        for t in self.nparange(self.tau, self.T, self.tau):
            # append new line
            ans.append(self._compute_line(t, x, last_line))
            X.append(x)
            Y.append([t for _ in x])
            # take new line as last
            last_line = ans[-1]
        return X, Y, ans
```

Явная конечно-разностная схема

Апроксимируем вторую производную по значениям нижнего временного слоя t^k, а именно:

$$\frac{\partial^2 u}{\partial x^2}(x_j, t^k) = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2}$$

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2}, \forall j \in \{1, \dots, N-1\}, \forall k \in \{0, \dots, K-1\}$$

Обозначим $\sigma = \dfrac{a\tau}{h^2}$, тогда:

$$u_j^{k+1} = \sigma u_{j-1}^k + (1 - 2\sigma) u_j^k + \sigma u_{j+1}^k$$

Граничные же значения u_0^{k+1} и u_N^{k+1} определяются граничными условиями $u_x(0, t) = \phi_0(t)$ и $u_x(l, t) = \phi_l(t)$ при помощи апроксимации производной.

Значение σ используется для анализа устойчивости решения, а именно решение устойчиво, если $\sigma \leq \frac{1}{2}$.

In [ ]:
```
class Explict_Schema(Schema):
    def _compute_sigma(self):
        self.sigma = self.a * self.tau / (self.h * self.h)
        if self.sigma > 0.5:
            warnings.warn("Sigma > 0.5")

    def _compute_line(self, t, x, last_line):
        line = [None for _ in last_line]
        for i in range(1, len(x) - 1):
            line[i] = self.sigma*last_line[i-1]
            line[i] += (1 - 2*self.sigma)*last_line[i]
            line[i] += self.sigma*last_line[i+1]
        line[0] = self.approx.explict_0(t, self.h, self.sigma,
                                        last_line, line, t - self.tau)
        line[-1] = self.approx.explict_l(t, self.h, self.sigma,
                                         last_line, line, t - self.tau)
        return line
```

Неявная конечно-разностная схема

Апроксимируем вторую производную по значениям верхнего временного слоя t^{k+1}, а именно:

$$\frac{\partial^2 u}{\partial x^2}(x_j, t^k) = \frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2}$$

Тогда получим явную схему конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - u_j^k}{\tau} = a \frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2}, \ \forall j \in \{1, \ldots, N-1\}, \forall k \in \{0, \ldots, K-1\}$$

Обозначим $\sigma = \frac{a\tau}{h^2}$. Тогда значения функции на слое можно найти эффективны образом с помощью методом прогонки, где **СЛАУ**, кроме крайних двух уравнений, определяется коэффицетнами $a_j = \sigma$, $b_j = -(1 + 2\sigma)$, $c_j = \sigma$, $d_j = -u_j^k$ уравнений:

$$a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j, \ \forall j \in \{1, \ldots, N-1\}$$

Первое и последнее уравнение системы содержащие u_0^{k+1} и u_N^{k+1} определяются граничными условиями $u_x(0, t) = \phi_0(t)$ и $u_x(l, t) = \phi_l(t)$ при помощи апроксимации производной.

Неявная схема является абсолютно устойчивой.

Схема Кранка-Николсона

Поскольку как правило решение в зависимости от времени лежит между значениями явной и неявной схемы, имеет смысл получить смешанную апроксимацию пространственных производных.

graphic

Явно-неявная схема для $\forall j \in \{1, \ldots, N-1\}, \forall k \in \{0, \ldots, K-1\}$ будет выглядеть следующим образом:

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

При значении параметра $\theta = \dfrac{1}{2}$ схема являет собой *схему Кранка-Николсона*.

Обозначим $\sigma = \dfrac{a\tau}{h^2}$. Тогда значения функции на слое можно найти эффективны образом с помощью методом прогонки, где **СЛАУ**, кроме крайних двух уравнений, определяется коэффицетнами $a_j = \sigma\theta$, $b_j = -(1 + 2\theta\sigma)$, $c_j = \sigma\theta$, $d_j = -(u_j^k + (1-\theta)\sigma(u_{j-1}^k - 2u_j^k + u_{j+1}^k))$ уравнений:

$$a_j u_{j-1}^{k+1} + b_j u_j^{k+1} + c_j u_{j+1}^{k+1} = d_j,\ \forall j \in \{1, \ldots, N-1\}$$

Первое и последнее уравнение системы содержащие u_0^{k+1} и u_N^{k+1} определяются граничными условиями $u_x(0,t) = \phi_0(t)$ и $u_x(l,t) = \phi_l(t)$ при помощи апроксимации производной.

Схема Кранка-Николсона является абсолютно устойчивой.

In [ ]:

```
class Explict_Implict(Schema):
    def set_O(self, O):
        self.O = O

    # method from old labs
    @staticmethod
    def race_method(A, b):
        P = [-item[2] for item in A]
        Q = [item for item in b]

        P[0] /= A[0][1]
        Q[0] /= A[0][1]

        for i in range(1, len(b)):
            z = (A[i][1] + A[i][0] * P[i-1])
            P[i] /= z
            Q[i] -= A[i][0] * Q[i-1]
            Q[i] /= z

        x = [item for item in Q]

        for i in range(len(x) - 2, -1, -1):
            x[i] += P[i] * x[i + 1]

        return x
```

```
    # compute line using race method
    def _compute_line(self, t, x, last_line):
        a = self.sigma * self.O
        b = -1 - 2 * self.sigma * self.O

        # coeffs
        A = [(a, b, a) for _ in range(1, len(x)-1)]
        w = [
            # expression
            -(last_line[i] +
                (1 - self.O) * self.sigma*
                (last_line[i-1] - 2*last_line[i] + last_line[i+1]))
            # for this idxs
            for i in range(1, len(x)-1)
        ]
        # compute coeffst for first and last equation
        koeffs = self.approx.nikolson_0(t, self.h, self.sigma,
                                        last_line, self.O, t - self.tau)
        A.insert(0, koeffs[:-1])
        w.insert(0, koeffs[-1])
        koeffs = self.approx.nikolson_l(t, self.h, self.sigma,
                                        last_line, self.O, t - self.tau)
        A.append(koeffs[:-1])
        w.append(koeffs[-1])

        return self.race_method(A, w)
```

In [ ]:
```
Krank_Nikolson = Explict_Implict
```

Апроксимация первых производных

In [ ]:
```
class Approx:
    def __init__(self, f0, fl):
        self.f0 = f0
        self.fl = fl

    def explict_0(self, t, h, sigma, l0, l1, t0):
        pass
    def explict_l(self, t, h, sigma, l0, l1, t0):
        pass

    def nikolson_0(self, t, h, sigma, l0, O, t0):
        pass
    def nikolson_l(self, t, h, sigma, l0, O, t0):
        pass
```

Двухточечная первого порядка </h2>

Двухточечная апроксимация первого порядка в точке $x = 0$ и $x = l$ равны соответственно:

$$\frac{u_1^{k+1} - u_0^{k+1}}{h} = \phi_0(t^{k+1})$$

$$\frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} = \phi_l(t^{k+1})$$

Тогда, поскольку мы знаем значения для внутренних узлов, получаем выражения для граничных значений при явном методе:

$$u_0^{k+1} = -h\phi_0(t^{k+1}) + u_1^{k+1}$$

$$u_N^{k+1} = h\phi_l(t^{k+1}) + u_{N-1}^{k+1}$$

И крайние уравенения для методда прогонки в неявном методе и в *схеме Кранка-Николсона*:

$$-u_0^{k+1} + u_1^{k+1} = h\phi_0(t^{k+1})$$

$$-u_{N-1}^{k+1} + u_N^{k+1} = h\phi_l(t^{k+1})$$

In [ ]:
```
class approx_two_one(Approx):
    def explict_0(self, t, h, sigma, l0, l1, t0):
        return -h * self.f0(t) + l1[1]

    def explict_l(self, t, h, sigma, l0, l1, t0):
        return h * self.fl(t) + l1[-2]

    def nikolson_0(self, t, h, sigma, l0, O, t0):
        return 0, -1, 1, h*self.f0(t)

    def nikolson_l(self, t, h, sigma, l0, O, t0):
        return -1, 1, 0, h*self.fl(t)
```

Трёхточечная второго порядка

Трёхточечна апроксимация второго порядка в точке $x = 0$ и $x = l$ равны соответственно:

$$\frac{-3u_0^{k+1} + 4u_1^{k+1} - u_2^{k+1}}{2h} = \phi_0(t^{k+1})$$

$$\frac{3u_N^{k+1} - 4u_{N-1}^{k+1} + u_{N-2}^{k+1}}{2h} = \phi_l(t^{k+1})$$

Тогда, поскольку мы знаем значения для внутренних узлов, получаем выражения для граничных значений при явном методе:

$$u_0^{k+1} = \frac{-2h\phi_0(t^{k+1}) + 4u_1^{k+1} - u_2^{k+1}}{3}$$

$$u_N^{k+1} = \frac{2h\phi_l(t^{k+1}) + 4u_{N-1}^{k+1} - u_{N-2}^{k+1}}{3}$$

Крайние уравенения для методда прогонки в неявном методе:

$$-2\sigma u_0^{k+1} + (2\sigma - 1)u_1^{k+1} = 2\sigma h\phi_0(t^{k+1}) - u_1^k$$

$$(2\sigma - 1)u_{N-1}^{k+1} - 2\sigma u_N^{k+1} = -2\sigma h\phi_l(t^{k+1}) - u_{N-1}^k$$

И крайние уравнения для *схемы Кранка-Николсона*:

$$-2\sigma\theta u_0^{k+1} + (2\sigma\theta - 1)u_1^{k+1} = 2\sigma\theta h\phi_0(t^{k+1}) - (u_1^k + (1-\theta)\sigma(u_0^k - 2u_1^k + u_2^k)$$

$$(1-2\sigma\theta)u_{N-1}^{k+1} + 2\sigma\theta u_N^{k+1} = 2\sigma\theta h\phi_l(t^{k+1}) + (u_{N-1}^k + (1-\theta)\sigma(u_{N-2}^k - 2u_{N-1}^k + u_N^k)$$

In [ ]:
```
class approx_three_two(Approx):
    def explict_0(self, t, h, sigma, l0, l1, t0):
        return (-2*h*self.f0(t) + 4*l1[1] - l1[2]) / 3

    def explict_l(self, t, h, sigma, l0, l1, t0):
        return (2*h*self.fl(t) + 4*l1[-2] - l1[-3]) / 3

    def nikolson_0(self, t, h, sigma, l0, O, t0):
        d = 2*sigma*O*h*self.f0(t)
        d -= l0[1] + (1 - O)*sigma*(l0[0] - 2*l0[1] + l0[2])
        return 0, -2*sigma*O, 2*sigma*O - 1, d

    def nikolson_l(self, t, h, sigma, l0, O, t0):
        d = 2*sigma*O*h*self.fl(t)
        d += l0[-2] + (1 - O)*sigma*(l0[-3] - 2*l0[-2] + l0[-1])
        return 1 - 2*sigma*O, 2*sigma*O, 0, d
```

Двухточечная второго порядка

Двухточечная апроксимация второго порядка в точке $x = 0$ и $x = l$ равны соответственно:

$$\frac{u_1^{k+1} - u_{-1}^{k+1}}{2h} = \phi_0(t^{k+1})$$

$$\frac{u_{N+1}^{k+1} - u_{N-1}^{k+1}}{2h} = \phi_l(t^{k+1})$$

Тогда, используя апроксимацию на предыдущем временном слое, а именно при $t = t^k$, и выразив значения, выходящие за пределы сетки с помощью уравнения: $\frac{u_j^{k+1} - u_j^k}{\tau} = a\frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2}$ для значений $j = 0$ и $j = N$ мы получим формулу граничных значений для явной схемы:

$$u_0^{k+1} = -2\sigma h\phi_0(t^k) + 2\sigma u_1^k + (1-2\sigma)u_0^k$$

$$u_N^{k+1} = 2\sigma h\phi_l(t^k) + 2\sigma u_{N-1}^k + (1-2\sigma)u_N^k$$

Используя аппроксимацию на слое t^{k+1} получим крайние уравенения для методда прогонки в неявном методе:

$$-(2\sigma+1)u_0^{k+1} + 2\sigma u_1^{k+1} = 2\sigma h\phi_0(t^{k+1}) - u_0^k$$

$$2\sigma u_{N-1}^{k+1} - (2\sigma+1)u_N^{k+1} = -2\sigma h\phi_l(t^{k+1}) - u_N^k$$

И крайние уравнения для *схемы Кранка-Николсона*:

$$-(2\theta\sigma+1)u_0^{k+1} + 2\sigma\theta u_1^{k+1} = 2\theta\sigma h\phi_0(t^{k+1}) - (u_0^k + 2(1-\theta)\sigma(u_1^k - u_0^k - h\phi_0(t^k)))$$

$$2\sigma\theta u_{N-1}^{k+1} - (2\theta\sigma+1)u_N^{k+1} = -2\theta\sigma h\phi_l(t^{k+1}) - (u_N^k + 2(1-\theta)\sigma(u_{N-1}^k - u_N^k + h\phi_l(t^k)$$

In [ ]:
```
class approx_two_two(Approx):
    def explict_0(self, t, h, sigma, l0, l1, t0):
        return -2*sigma*h*self.f0(t0) + \
                    2*sigma*l0[1] + (1 - 2*sigma)*l0[0]

    def explict_l(self, t, h, sigma, l0, l1, t0):
        return 2*sigma*h*self.fl(t0) + \
                    2*sigma*l0[-2] + (1 - 2*sigma)*l0[-1]

    def nikolson_0(self, t, h, sigma, l0, O, t0):
        d = 2*sigma*O*h*self.f0(t) - l0[0]
        d -= 2*(1 - O)*sigma*(l0[1] - l0[0] - h*self.f0(t0))
        return 0, -(2*sigma*O + 1), 2*sigma*O, d

    def nikolson_l(self, t, h, sigma, l0, O, t0):
        d = -2*sigma*O*h*self.fl(t) - l0[-1]
        d -= 2*(1 - O)*sigma*(l0[-2] - l0[-1] + h*self.fl(t0))
        return 2*sigma*O, -(2*sigma*O + 1), 0, d
```

Результаты работы

Зависимость погрешности от параметра h

Вычисление погрешностей

Вычисление погрешности: $e = \|\hat{z} - z\|_2$, где $\hat{z}$, z - матрицы вычесленных и реальных значений функции в сетке соответственно.

In [ ]:
```
def epsilon(x, y, z, f):
    ans = 0.0
    for i in range(len(z)):
        for j in range(len(z[i])):
            ans += (z[i][j] - f(x[i][j], y[i][j]))**2
    return ans**0.5
```

Постоение зависимости погрешности от шага h.

In [ ]:
```
def get_graphic_h(solver, real_f):
    h = []
    e = []
    for N in range(3, 50):
        x, y, z = solver(N)
        h.append(solver.h)
        e.append(epsilon(x, y, z, real_f))
    return h, e
```

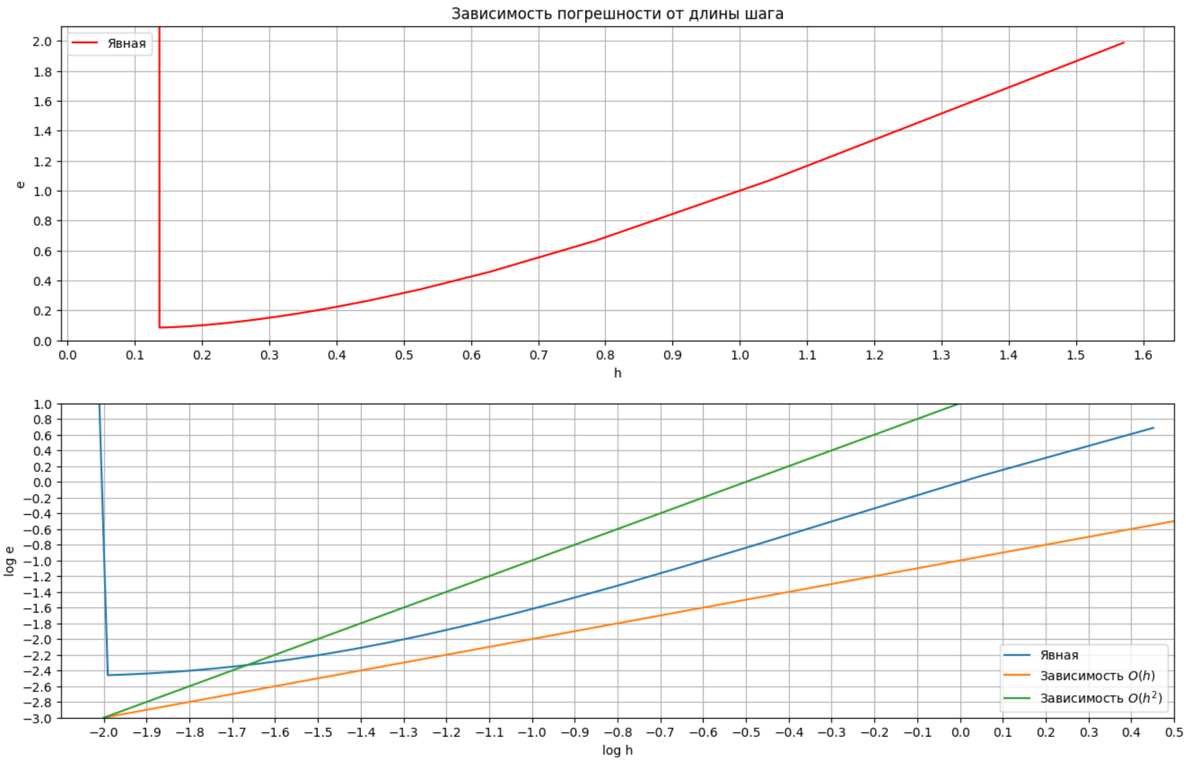
Явная схема

In [ ]:
```
explict = Explict_Schema(T = 1, aprx_cls=approx_two_two)
```


In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от длины шага")
h, e = get_graphic_h(explict, u)
```

```
plt.plot(h, e, label="Явная", color = "red")
plt.xlabel("h")
plt.ylabel("e")
plt.ylim([0, 2.1])
plt.xticks(list(explict.nparange(0, 1.6, 0.1)))
plt.yticks(list(explict.nparange(0, 2.1, 0.2)))
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Явная")
plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.ylim([-3, 1])
plt.xlim([-2.1, 0.5])
plt.xticks(list(explict.nparange(-2, 0.5, 0.1)))
plt.yticks(list(explict.nparange(-3, 1, 0.2)))
plt.legend()
plt.grid()
```

```
<ipython-input-4-6bd7c55a9975>:5: UserWarning: Sigma > 0.5
  warnings.warn("Sigma > 0.5")
```



Неявная схема

In [ ]:
```
# Krank Nikolson with O = 1 is implict schema
implict = Krank_Nikolson(T = 1, aprx_cls=approx_two_two, O=1)
```

In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от длины шага")
h, e = get_graphic_h(implict, u)

plt.plot(h, e, label="Неявная", color = "red")
plt.xlabel("h")
```

```
plt.ylabel("e")
plt.ylim([0, 2.1])
plt.xticks(list(implict.nparange(0, 1.6, 0.1)))
plt.yticks(list(implict.nparange(0, 2.1, 0.2)))
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Неявная")
plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.ylim([-3, 1])
plt.xlim([-2, 0.5])
plt.xticks(list(implict.nparange(-2, 0.5, 0.1)))
plt.yticks(list(implict.nparange(-3, 1, 0.2)))
plt.legend()
plt.grid()
```

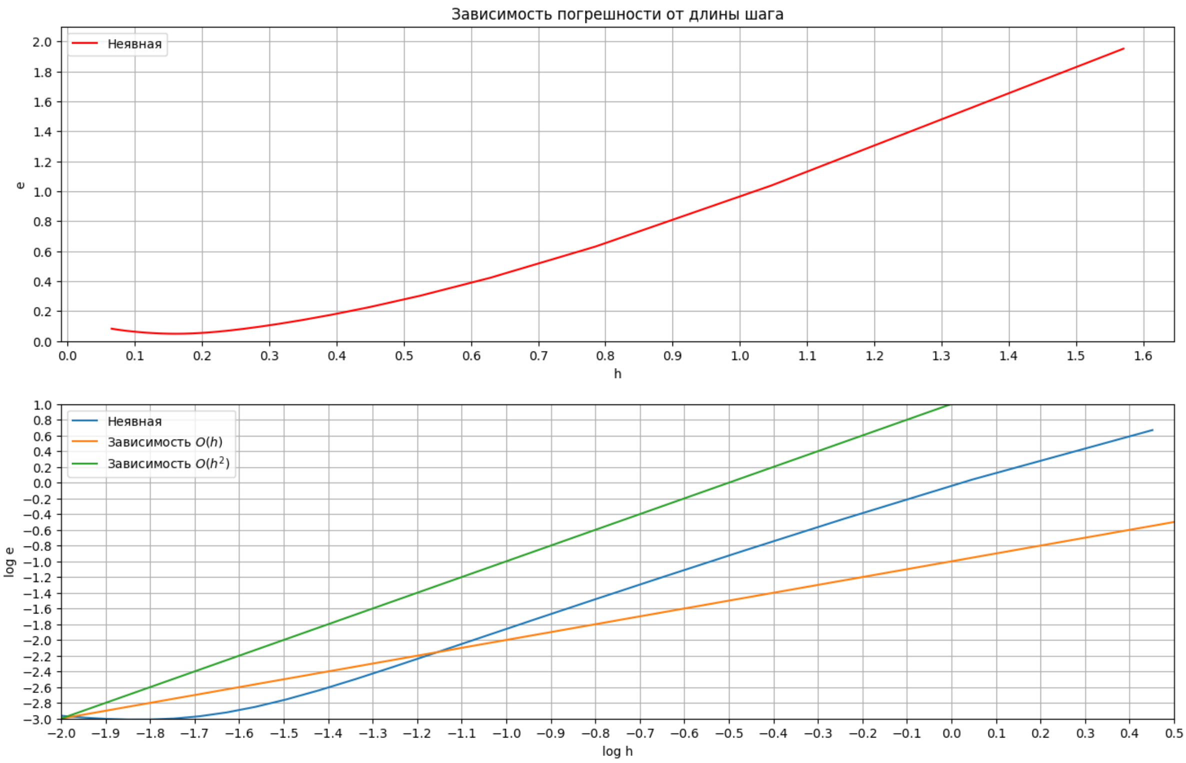
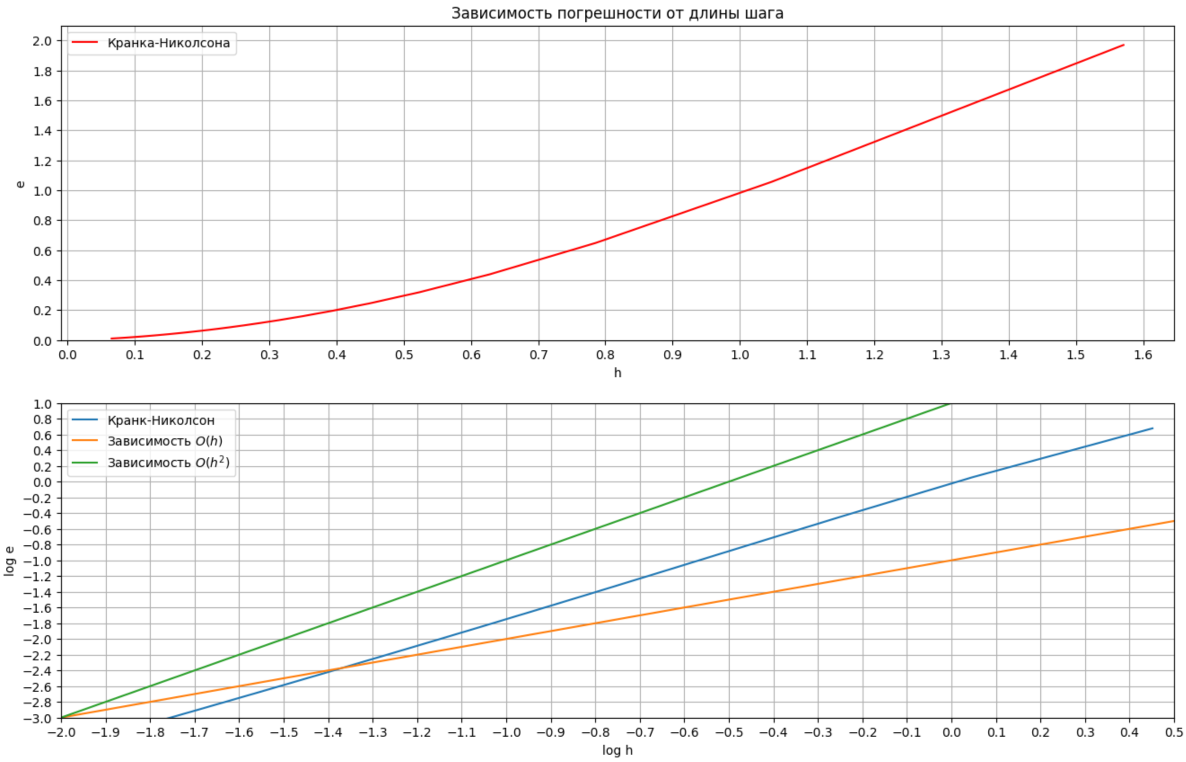


Схема Кранка-Николсона

In [ ]:
```
krank = Krank_Nikolson(T = 1, aprx_cls=approx_two_two)
```

In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от длины шага")
h, e = get_graphic_h(krank, u)

plt.plot(h, e, label="Кранка-Николсона", color = "red")
plt.xlabel("h")
plt.ylabel("e")
plt.ylim([0, 2.1])
plt.xticks(list(krank.nparange(0, 1.6, 0.1)))
plt.yticks(list(krank.nparange(0, 2.1, 0.2)))
plt.legend()
plt.grid()
```

```
plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Кранк-Николс
plt.plot([-2, 0.5], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-2, 0.5], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.ylim([-3, 1])
plt.xlim([-2, 0.5])
plt.xticks(list(krank.nparange(-2, 0.5, 0.1)))
plt.yticks(list(krank.nparange(-3, 1, 0.2)))
plt.legend()
plt.grid()
```



Зависимость погрешности от параметра τ

Вычисление погрешности

Построение зависимости погрешности от параметра τ.

In [ ]:
```
def get_graphic_tau(solver, real_f):
    tau = []
    e = []
    for K in range(3, 90):
        x, y, z = solver(K = K)
        tau.append(solver.tau)
        e.append(epsilon(x, y, z, real_f))
    return tau, e
```

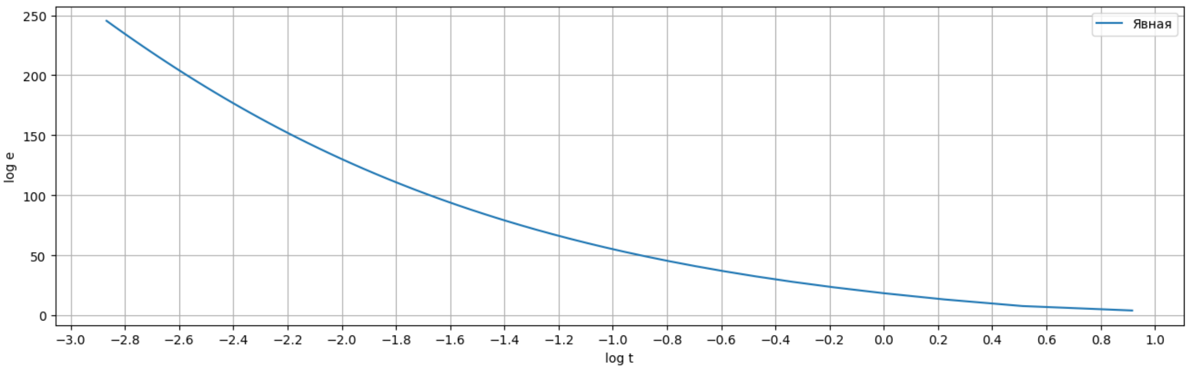
Явная схема

In [ ]:
```
explict = Explict_Schema(T = 5, aprx_cls=approx_two_two)
```


In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
```

```
plt.title("Зависимость погрешности от мелкости разбиения по времени")
tau, e = get_graphic_tau(explict, u)

plt.plot(tau, e, label="Явная", color = "red")
plt.xlabel("t")
plt.ylabel("e")
# plt.ylim([0, 2.1])
plt.xticks(list(explict.nparange(0, 2.5, 0.1)))
# plt.yticks(list(explict.nparange(0, 2.1, 0.2)))
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, tau)), list(map(math.log, e)), label="Явная")
plt.xlabel("log t")
plt.ylabel("log e")
# plt.ylim([-3, 1])
# plt.xlim([-2, 0.5])
plt.xticks(list(explict.nparange(-3, 1, 0.2)))
# plt.yticks(list(explict.nparange(-3, 1, 0.2)))
plt.legend()
plt.grid()
```

```
<ipython-input-4-6bd7c55a9975>:5: UserWarning: Sigma > 0.5
  warnings.warn("Sigma > 0.5")
```



Неявная схема

In [ ]:
```
# Krank Nikolson with O = 1 is implict schema
implict = Krank_Nikolson(T = 5, aprx_cls=approx_two_two, O=1)
```

In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от мелкости разбиения по времени")
tau, e = get_graphic_tau(implict, u)

plt.plot(tau, e, label="Неявный", color = "red")
plt.xlabel("t")
```

14

```
plt.ylabel("e")
# plt.ylim([0, 2.1])
plt.xticks(list(explict.nparange(0, 2.5, 0.1)))
# plt.yticks(list(implict.nparange(0, 2.1, 0.2)))
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, tau)), list(map(math.log, e)), label="Неявный")
plt.plot([-3, 1], [-0.5, 3.5], label="Зависимость $O(t)$")
plt.plot([-3, 1], [0, 2], label="Зависимость $O(\sqrt{t})$")
plt.xlabel("log t")
plt.ylabel("log e")
# plt.ylim([-3, 1])
# plt.xlim([-2, 0.5])
plt.xticks(list(explict.nparange(-3, 1, 0.2)))
# plt.yticks(list(implict.nparange(-3, 1, 0.2)))

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

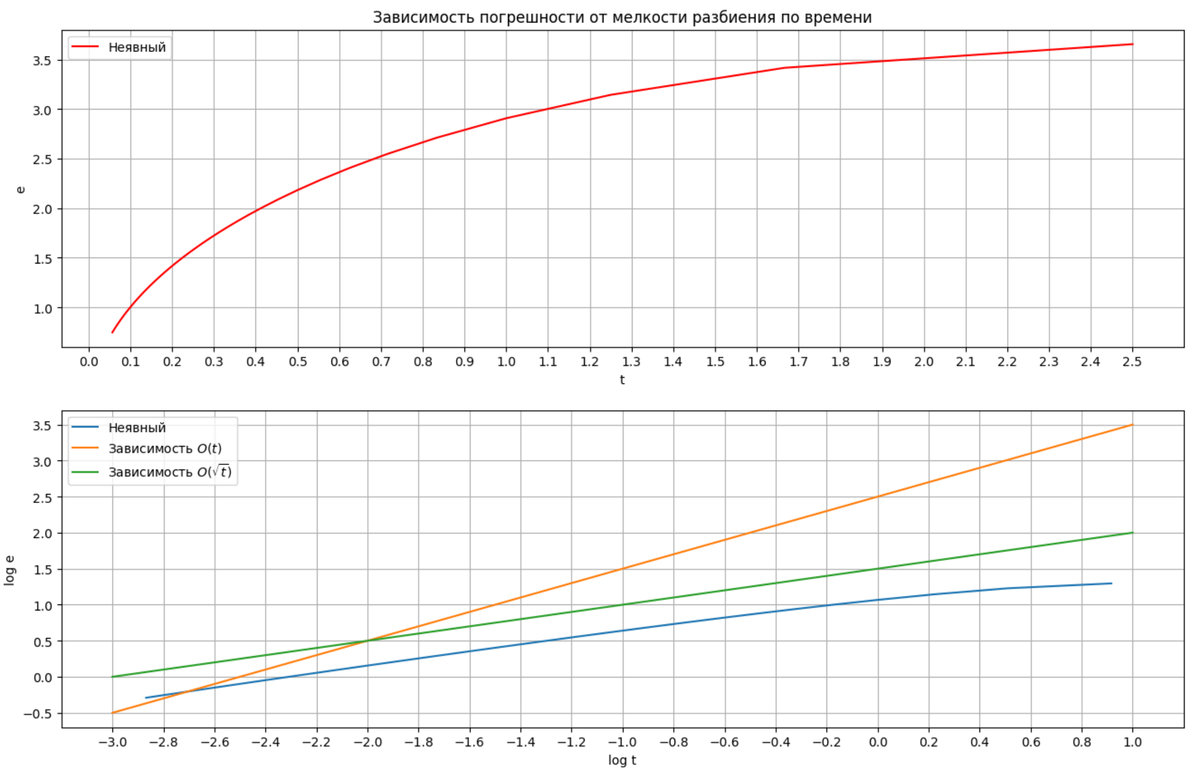
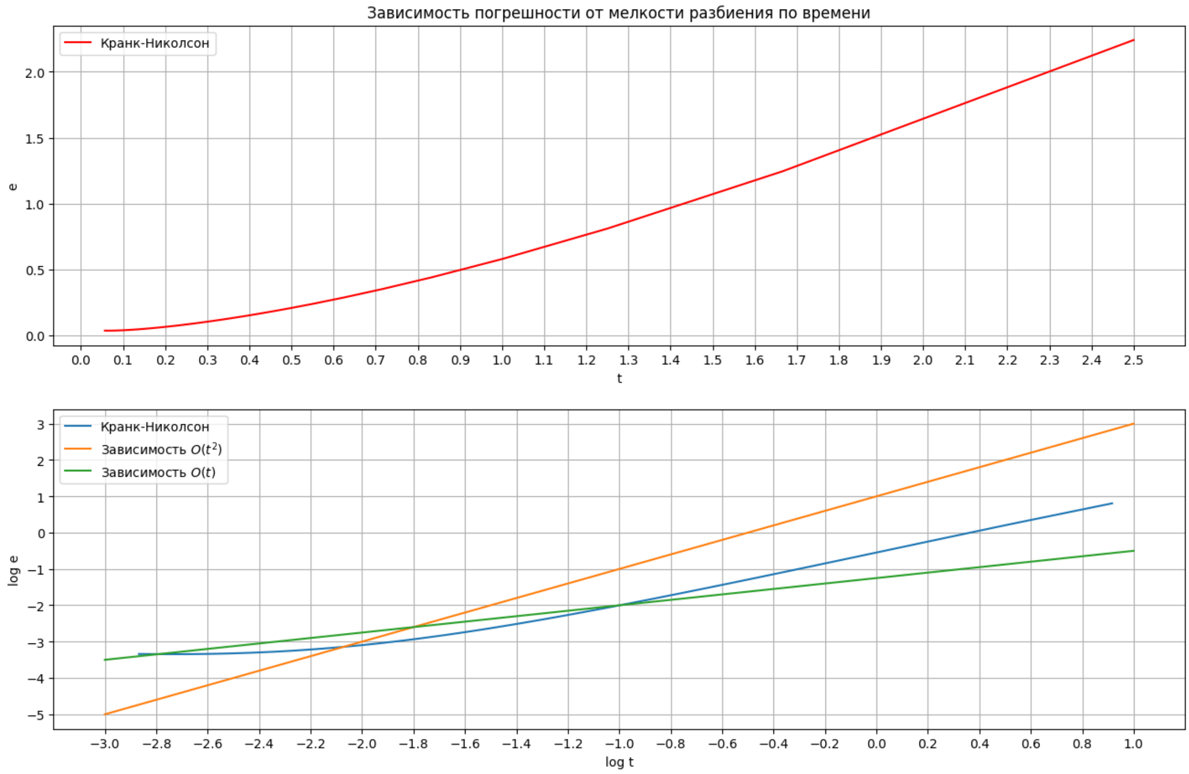


Схема Кранка-Николсона

In [ ]:
```
# Krank Nikolson with O = 1 is implict schema
krank = Krank_Nikolson(T = 5, aprx_cls=approx_two_two)
```

In [ ]:
```
plt.figure(figsize = (16, 10))

plt.subplot(2, 1, 1)
plt.title("Зависимость погрешности от мелкости разбиения по времени")
tau, e = get_graphic_tau(krank, u)

plt.plot(tau, e, label="Кранк-Николсон", color = "red")
plt.xlabel("t")
plt.ylabel("e")
# plt.ylim([0, 2.1])
plt.xticks(list(krank.nparange(0, 2.5, 0.1)))
# plt.yticks(list(implict.nparange(0, 2.1, 0.2)))
```

```python
plt.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, tau)), list(map(math.log, e)), label="Кранк-Нико
plt.plot([-3, 1], [-5, 3], label="Зависимость $O(t^2)$")
plt.plot([-3, 1], [-3.5, -0.5], label="Зависимость $O(t)$")
plt.xlabel("log t")
plt.ylabel("log e")
# plt.ylim([-3, 1])
# plt.xlim([-2, 0.5])
plt.xticks(list(krank.nparange(-3, 1, 0.2)))
# plt.yticks(list(implict.nparange(-3, 1, 0.2)))

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



Выводы

Выполнив данную лабораторную работу, я изучил явные и неявные конечно-разностные схемы, схему Кранка-Николсона для решения начально-краевой задачи для дифференциального уравнения параболического типа. Реализовал три варианта аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. Также исследовал зависимость погрешности от сеточных параметров τ и h.