Московский авиационный институт
(национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №6 по курсу «Численные методы»

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

Москва, 2023

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: 14

Уравнение:

$$\frac{\partial^2 u}{\partial t^2} = \frac{\partial^2 u}{\partial x^2} - 5u$$

$$\begin{cases} u'_x(0,\ t) = 2u(0,t) \\ u'_x(1,\ t) = 2u(1,t) \\ u(x,\ 0) = \psi_1(x) = e^{2x} \\ u_t(x,0) = \psi_2(x) = 0 \end{cases}$$

Аналитическое решение:

$$u(x,t) = e^{2x}\cos t$$

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:

$$h = \frac{l}{N-1},\ \tau = \frac{T}{K-1}$$

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 предыдущих временных известно, попробуем определить значения функции на временном слое t^{k+1} путем разностной апроксимации производной:

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

И одним из методов апроксимации второй производной по x:

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

Для расчета u_j^0 и u_j^1 можно использовать следующие формулы:

$$u_j^0 = \psi_1(x_j)$$

$$u_j^1 = \psi_1(x_j) + \tau\psi_2(x_j) + \frac{\tau^2}{2}\psi_1''(x_j) + O(\tau^2)$$

$$u_j^1 = \psi_1(x_j) + \tau\psi_2(x_j) + O(\tau^1)$$

2

In [2]:
```
# analytic solve
def u(x, t):
    return math.exp(2*x)*math.cos(t)
```

In [3]:
```
# class will return grid of values
class Schema:
```

```
    def __init__(self, T = 5, order2nd = True, aprx_cls = None):
        self.psi1 = lambda x: math.exp(2*x)
        self.diffpsi = lambda x: 4 * math.exp(2*x)
        self.psi2 = lambda x: 0
        self.T = T

        self.l0 = 0
        self.l1 = 1
        self.tau = None
        self.h = None
        self.approx = None
        self.order = order2nd
        if aprx_cls is not None:
            self._init_approx(aprx_cls)
        self.sigma = None

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

    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

    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.tau*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_line1, last_line2):
        pass

    def __call__(self, N=30, K=200):
        # 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))
```

3

```
        # compute first line
        last_line = list(map(self.psi1, x))
        # add copy
        ans.append(list(last_line))
        # compute second line
```

```
        if self.order:
            last_line = list(map(
                lambda a: self.psi1(a) + self.tau*self.psi2(a) + self.tau*se
                x
            ))
        else:
            last_line = list(map(lambda a: self.psi1(a) + self.tau*self.psi2
        # add copy
        ans.append(list(last_line))

        # create grid
        X = [x, x]
        Y = [[0.0 for _ in x]]
        Y.append([self.tau for _ in x])
        # main loop
        for t in self.nparange(self.tau + self.tau, self.T, self.tau):
            # append new line
            ans.append(self._compute_line(t, x, ans[-1], ans[-2]))
            X.append(x)
            Y.append([t for _ in x])
        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:

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} - 5u_j^k,\ \forall j \in \{1, \ldots, N-1\}, \forall k \in \{0, \ldots,$$

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

$$u_j^{k+1} = \sigma(u_{j+1}^k - 2u_j^k + u_{j-1}^k) - 5\tau^2 u_j^k + 2u_j^k - u_j^{k-1}$$

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

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

In [4]:
```
class Explict_Schema(Schema):
    def _compute_sigma(self):
        self.sigma =  self.tau*self.tau / (self.h * self.h)
        if self.sigma > 1:
            warnings.warn("Sigma > 1")

    def _compute_line(self, t, x, last_line1, last_line2):
        line = [None for _ in last_line1]
        for i in range(1, len(x) - 1):
```

```
            line[i] = self.sigma*(last_line1[i-1] - 2*last_line1[i] + last_l
            line[i] -= 5*self.tau*self.tau*last_line1[i]
            line[i] += 2*last_line1[i]
            line[i] -= last_line2[i]
        line[0] = self.approx.explict_0(self.h, self.sigma, line, last_line1
        line[-1] = self.approx.explict_l(self.h, self.sigma, line, last_line
        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:

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} = \frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} - 5u_j^{k+1},\ \forall j \in \{1, \dots, N-1\}, \forall k \in \{0,$$

Обозначим $\sigma = \frac{\tau^2}{h^2}$. Тогда значения функции на слое можно найти эффективны образом с помощью методом прогонки, где **СЛАУ**, кроме крайних двух уравнений, определяется коэффицетнами $a_j = 1$, $b_j = -(2 + 5h^2 + \frac{1}{\sigma})$, $c_j = 1$, $d_j = \frac{-2u_j^k + u_j^{k-1}}{\sigma}$ уравнений:

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

Первое и последнее уравнение системы содержащие u_0^{k+1} и u_N^{k+1} определяются граничными условиями при помощи апроксимации производной.

Неявная схема является абсолютно устойчивой.

In [5]:
```
class Implict_Schema(Schema):
    # 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]
```

5

```
        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_line1, last_line2):
        A = [(1, -(2 + 5*self.h*self.h + 1/self.sigma), 1) for _ in range(1,
        w = [(last_line2[i] - 2*last_line1[i]) / self.sigma for i in range(1

        # compute coeffst for first and last equation
        koeffs = self.approx.implict_0(self.h, self.sigma, last_line1, last_
        A.insert(0, koeffs[:-1])
        w.insert(0, koeffs[-1])

        koeffs = self.approx.implict_l(self.h, self.sigma, last_line1, last_
        A.append(koeffs[:-1])
        w.append(koeffs[-1])

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

Апроксимация первых производных

In [6]:
```
class Approx:
    def __init__(self):
        pass
    def explict_0(self, h, sigma, line, last_line1, last_line2, tau):
        pass
    def explict_l(self, h, sigma, line, last_line1, last_line2, tau):
        pass
    def implict_0(self, h, sigma, l0, l1):
        pass
    def implict_l(self, h, sigma, l0, l1):
        pass
```

Двухточечная первого порядка

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

$$\frac{u_1^{k+1} - u_0^{k+1}}{h} = 2u_0^{k+1}$$

$$\frac{u_N^{k+1} - u_{N-1}^{k+1}}{h} = 2u_N^{k+1}$$

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

$$u_0^{k+1} = \frac{u_1^{k+1}}{(1+2h)}$$

6

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

И крайние уравенения для методда прогонки в неявном методе:

$$u_0^{k+1}(1+2h) - u_1^{k+1} = 0$$

$$-u_{N-1}^{k+1} + (1-2h)u_N^{k+1} = 0$$

In [7]:
```
class approx_two_one(Approx):
    def explicit_0(self, h, sigma, line, last_line1, last_line2, tau):
        return line[1] / (1 + 2*h)
    def explicit_l(self, h, sigma, line, last_line1, last_line2, tau):
        return line[-2] / (1 - 2*h)
    def implict_0(self, h, sigma, l0, l1):
        return 0, (1 + 2*h), -1, 0
    def implict_l(self, h, sigma, l0, l1):
        return -1, (1 - 2*h), 0, 0
```

Трёхточечная второго порядка

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

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

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

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

$$u_0^{k+1} = \frac{4u_1^{k+1} - u_2^{k+1}}{3+4h}$$

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

Крайние уравенения для методда прогонки в неявном методе:

$$-(2+4h)u_0^{k+1} - (5h^2 - 2 + \frac{1}{\sigma})u_1^{k+1} = \frac{(-2u_1^k + u_1^{k-1})}{\sigma}$$

$$-(5h^2 - 2 + \frac{1}{\sigma})u_{N-1}^{k+1} - (2-4h)u_N^{k+1} - = \frac{(-2u_{N-1}^k + u_{N-1}^{k-1})}{\sigma}$$

In [8]:
```
class approx_three_two(Approx):
    def explicit_0(self, h, sigma, line, last_line1, last_line2, tau):
        return (4*line[1] - line[2]) / (3 + 4*h)
    def explicit_l(self, h, sigma, line, last_line1, last_line2, tau):
        return (4*line[-2] - line[-3]) / (3 - 4*h)
    def implict_0(self, h, sigma, l0, l1):
        return 0, -(2 + 4*h), -(5*h*h + 1/sigma - 2), (-2*l0[1] + l1[1])/sig
    def implict_l(self, h, sigma, l0, l1):
        return -(5*h*h + 1/sigma - 2), -(2 - 4*h), 0, (-2*l0[-2] + l1[-2])/s
```

Двухточечная второго порядка

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

$$\frac{u_1^{k+1} - u_{-1}^{k+1}}{2h} = 2u_0^{k+1}$$

$$\frac{u_{N+1}^{k+1} - u_{N-1}^{k+1}}{2h} = 2u_N^{k+1}$$

Тогда, используя апроксимацию на предыдущем временном слое, а именно при $t = t^k$, и выразив значения, выходящие за пределы сетки с помощью уравнения: $\dfrac{u_j^{k-1} - 2u_j^k + u_j^{k+1}}{\tau^2} = \dfrac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} - 5u_j$ для значений $j = 0$ и $j = N$ мы получим формулу граничных значений для явной схемы:

$$u_0^{k+1} = \sigma(2u_1^k - (2+4h)u_0^k) + (2 - 5\tau^2)u_0^k - u_0^{k-1}$$

$$u_0^{k+1} = \sigma(2u_{N-1}^k + (4h-2)u_N^k) + (2 - 5\tau^2)u_N^k - u_N^{k-1}$$

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

$$-(2 + 5h^2 + 4h + \frac{1}{\sigma})u_0^{k+1} + 2u_1^{k+1} = \frac{(-2u_0^k + u_0^{k-1})}{\sigma}$$

$$2u_{N-1}^{k+1} - (2 + 5h^2 - 4h + \frac{1}{\sigma})u_N^{k+1} = \frac{(-2u_N^k + u_N^{k-1})}{\sigma}$$

In [9]:
```
class approx_two_two(Approx):
    def explicit_0(self, h, sigma, line, last_line1, last_line2, tau):
        ans = sigma*(2*last_line1[1] - (2 + 4*h)*last_line1[0])
        ans += (2 - 5*tau*tau)*last_line1[0] - last_line2[0]
        return ans

    def explicit_l(self, h, sigma, line, last_line1, last_line2, tau):
        ans = sigma*(2*last_line1[-2] + (4*h - 2)*last_line1[-1])
        ans += (2 - 5*tau*tau)*last_line1[-1] - last_line2[-1]
        return ans

    def implict_0(self, h, sigma, l0, l1):
        return 0, -(2 + 5*h*h + 4*h + 1/sigma), 2, (-2*l0[0] + l1[0])/sigma

    def implict_l(self, h, sigma, l0, l1):
        return 2, -(2 + 5*h*h - 4*h + 1/sigma), 0, (-2*l0[-1] + l1[-1])/sigm
```

Вычисление погрешностей

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

In [10]:
```
def epsilon(x, y, z, f):
    ans = 0.0
```

```
    for i in range(len(z)):
        for j in range(len(z[i])):
            temp = abs(z[i][j] - f(x[i][j], y[i][j]))
            ans = temp if temp > ans else ans
    return ans
```

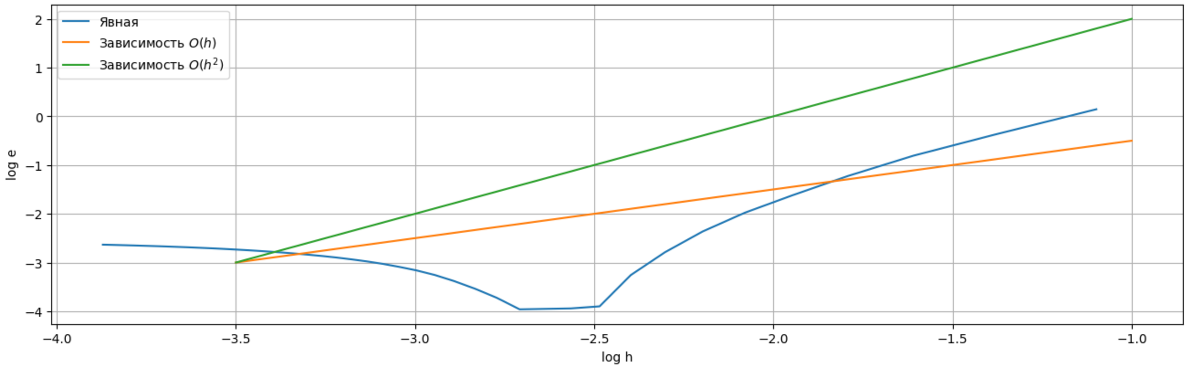
Постоение зависимости погрешности от шага h.

In [11]:
```
def get_graphic_h(solver, real_f):
    h,e = [],[]
    for N in range(4, 50, 1):
        x, y, z = solver(N)
        h.append(solver.h)
        e.append(epsilon(x, y, z, real_f))
    return h, e
```

Явная схема

In [12]:
```
explict = Explict_Schema(T = 1, aprx_cls=approx_two_two)
```

In [13]:
```
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.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Явная")
plt.plot([-3.5, -1], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-3.5, -1], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.legend()
plt.grid()
```

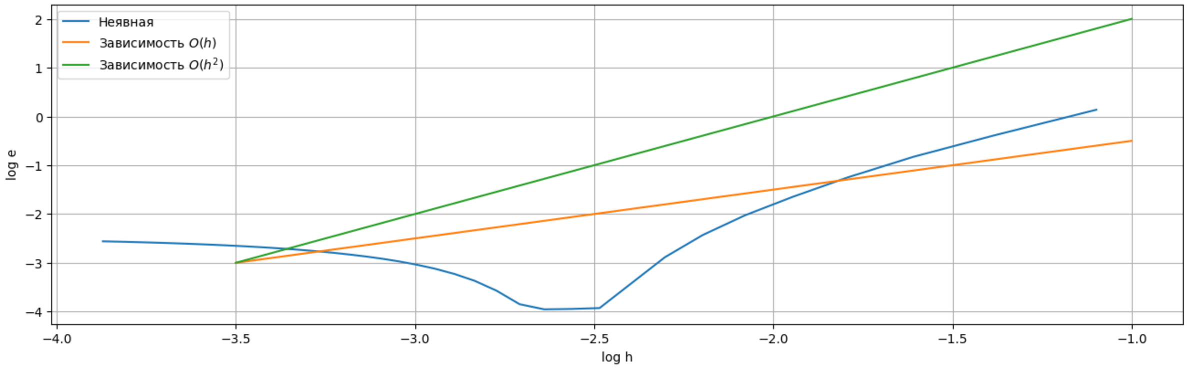


Неявная схема

In [14]:
```
# Krank Nikolson with O = 1 is implict schema
implict = Implict_Schema(T = 1, aprx_cls=approx_two_two)
```

In [15]:
```
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.legend()
plt.grid()

plt.subplot(2, 1, 2)
plt.plot(list(map(math.log, h)), list(map(math.log, e)), label="Неявная")
plt.plot([-3.5, -1], [-3, -0.5], label="Зависимость $O(h)$")
plt.plot([-3.5, -1], [-3, 2], label="Зависимость $O(h^2)$")
plt.xlabel("log h")
plt.ylabel("log e")
plt.legend()
plt.grid()
```

10



Вычисление погрешности

Построение зависимости погрешности от параметра τ.

In [16]:
```
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
```

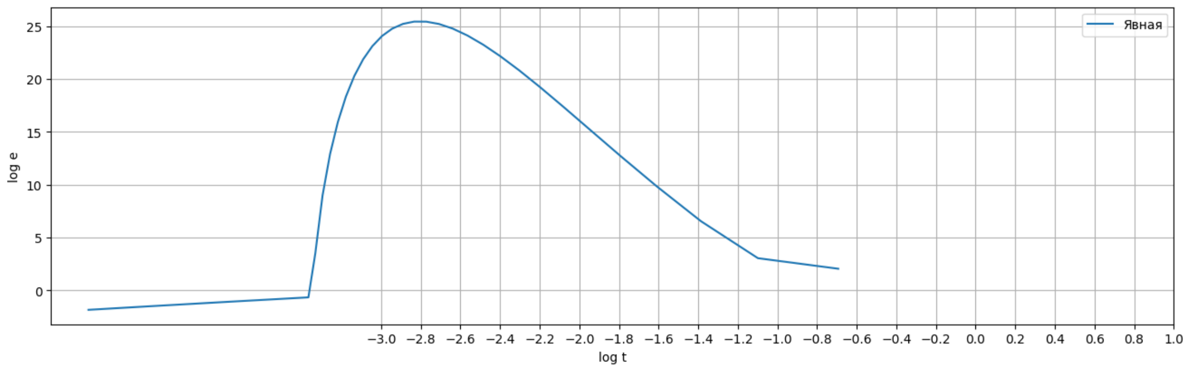
Явная схема

In [17]:
```
explict = Explict_Schema(T = 1, aprx_cls=approx_two_two)
```

In [18]:
```
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.xticks(list(explict.nparange(0, 2.5, 0.1)))
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.xticks(list(explict.nparange(-3, 1, 0.2)))
```

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

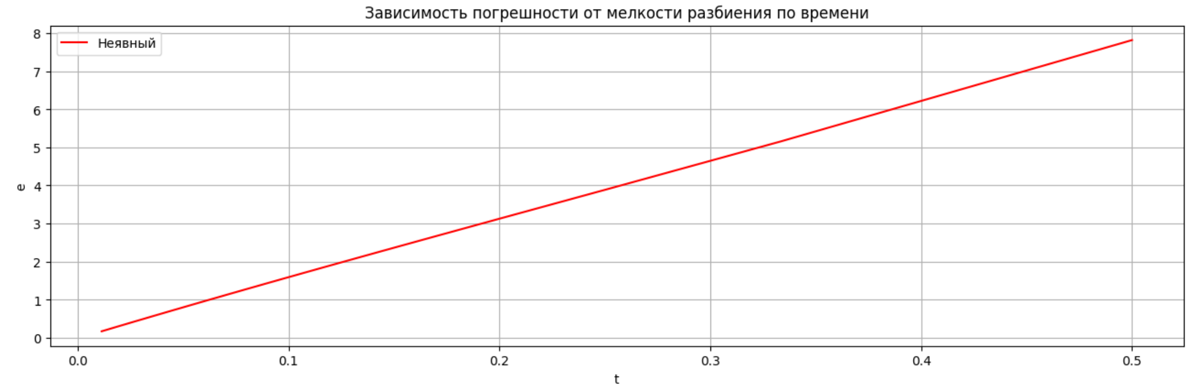
```
<ipython-input-4-5ffb4a1b1180>:5: UserWarning: Sigma > 1
  warnings.warn("Sigma > 1")
```

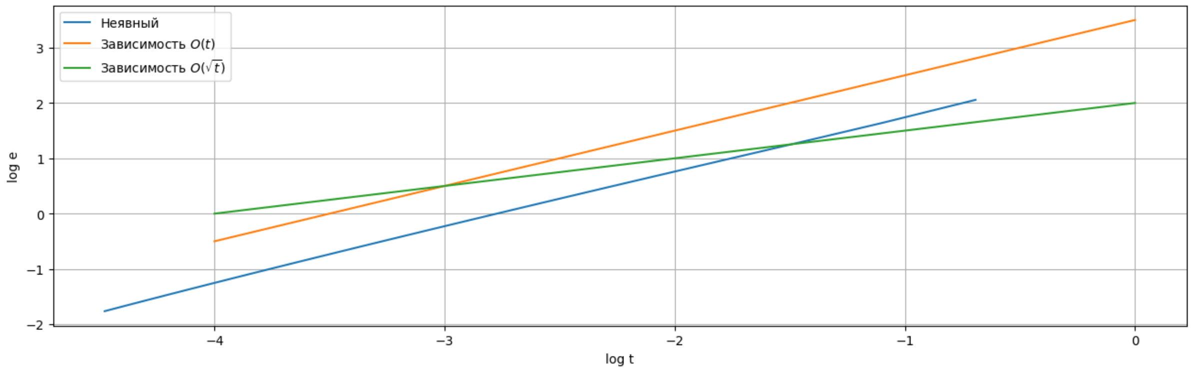


Неявная схема

In [19]:
```
implict = Implict_Schema(T = 1, aprx_cls=approx_two_two, order2nd=True)
```

In [20]:
```
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")
plt.ylabel("e")
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([-4, 0], [-0.5, 3.5], label="Зависимость $O(t)$")
plt.plot([-4, 0], [0, 2], label="Зависимость $O(\sqrt{t})$")
plt.xlabel("log t")
plt.ylabel("log e")

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


Зависимость погрешности от мелкости разбиения по времени
Неявный
e
t
0.0
0.1
0.2
0.3
0.4
0.5
0
1
2
3
4
5
6
7
8


Неявный
Зависимость $O(t)$
Зависимость $O(\sqrt{t})$
log e
log t
−4
−3
−2
−1
0
−2
−1
0
1
2
3

Вывод Выполнив данную лабораторную работу, изучил явную схему крест и неявную схему для решения начально-краевой задачи для дифференциального уравнения гиперболического типа. Выполнил три варианта аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком и двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислил погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $u(x, t)$. Также исследовал зависимость погрешности от сеточных параметров τ и h.