Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования


```
Лабораторная работа № 5
по курсу «Численные методы».

Тема: «ЧИСЛЕННОЕ РЕШЕНИЕ УРАВНЕНИЙ ПАРАБОЛИЧЕСКОГО ТИПА.
ПОНЯТИЕ О МЕТОДЕ КОНЕЧНЫХ РАЗНОСТЕЙ.
ОСНОВНЫЕ ОПРЕДЕЛЕНИЯ И КОНЕЧНО-РАЗНОСТНЫЕ СХЕМЫ».
```



```
Студент: Кондратьев Е.А.
Группа: 80-406Б
Преподаватель: Ревизников Д.Л.
```





```
Москва, 2022
```



# 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 $\tau$ и $h$.

# Вариант №7

***Инструменты:***

```
import ipywidgets as widgets
from ipywidgets import interact
from IPython.display import display
import random
import matplotlib.pyplot as plt
import math
import sys
import warnings
import numpy as np
from functools import reduce
from mpl_toolkits.mplot3d import Axes3D
```

***Уравнение:***

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + g(x,t),$$

где $g(x,t) = \frac{1}{2} e^{-\frac{1}{2}t} \cos x$

***Граничные условия:***

$$\begin{cases} u'_x(0,\ t) = \phi_0(t) = e^{-\frac{1}{2}t} \\ u'_x(\pi,\ t) = \phi_l(t) = -e^{-\frac{1}{2}t} \\ u(x,\ 0) = \sin x \end{cases}$$

***Аналитическое решение:***

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

Задаем наши данные из условия задачи.

```
def phi_0(t):
    return math.exp(-0.5*t)

def phi_l(t):
    return -math.exp(-0.5*t)

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

# Вместо cos нужно sin, так как ошибка в условии
def g(x, t):
    return 0.5 * math.exp(-0.5*t) * math.sin(x)

def u(x, t):
    return math.exp(-0.5*t)*math.sin(x)
```

## Конечно-разностная схема

Рассмотрим конечно-разностную схему решения краевой задачи на сетке с граничными параметрами $l, T$ и параметрами насыщенности сетки $N, K$. Отсюда размер шага по каждой из координат определяется:

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

Определим 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 $x$:

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

```
class Schema:
    def __init__(self, f0 = phi_0, fl = phi_l,
                 u0 = u_0, g = g,
                 O = 0.5, l0 = 0, l1 = math.pi,
                 T = 5, aprx_cls = None):
        self.fl = fl
        self.f0 = f0
        self.u0 = u0
        self.g = g
        self.T = T
        self.l0 = l0
        self.l1 = l1
        self.tau = None
        self.h = None
        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 SetApprox(self, aprx_cls):
        self._init_approx(self, aprx_cls)

    def Set_l0_l1(self, l0, l1):
        self.l0 = l0
        self.l1 = l1

    def SetT(self, T):
        self.T = T

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

    def CalculateTau(self, K):
        self.tau = self.T / K

    def CalculateSigma(self):
        self.sigma = 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 CalculateLine(self, t, x, lastLine):
        pass

    def __call__(self, N=50, K=70):
        N, K = N - 1, K - 1
        self.CalculateTau(K)
        self.CalculateH(N)
        self.CalculateSigma()
        ans = []
        x = list(np.arange(self.l0, self.l1 + 0.5 * self.h, self.h))
        lastLine = list(map(self.u0, x))
        ans.append(list(lastLine))
        X, Y = [], []
        X.append(x)
        Y.append([0.0 for _ in x])
        for t in np.arange(self.tau, self.T + 0.5 * self.tau, self.tau):
            ans.append(self.CalculateLine(t, x, lastLine))
            X.append(x)
            Y.append([t for _ in x])
            lastLine = 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 для $\forall j \in \{1, \ldots, N-1\}, \forall k \in \{0, \ldots, K-1\}$:

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

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

$$u_j^{k+1} = \sigma u_{j-1}^k + (1 - 2\sigma)u_j^k + \sigma u_{j+1}^k + \tau g(x_j, t^k)$$

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.

```
class ExplictSchema(Schema):
    def CalculateSigma(self):
        self.sigma = self.tau / (self.h * self.h)
        if self.sigma > 0.5:
            warnings.warn("Sigma > 0.5")

    def CalculateLine(self, t, x, lastLine):
```

```
        line = [None for _ in lastLine]
        for i in range(1, len(x) - 1):
            line[i] = self.sigma * lastLine[i-1]
            line[i] += (1 - 2 * self.sigma) * lastLine[i]
            line[i] += self.sigma * lastLine[i + 1]
            line[i] += self.tau * self.g(x[i], t - self.tau)
        line[0] = self.approx.explict_0(t, self.h, self.sigma,
                                      self.g, self.l0, lastLine,
                                      line, t - self.tau)
        line[-1] = self.approx.explict_l(t, self.h, self.sigma,
                                       self.g, self.l0, lastLine,
                                       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 для $\forall j \in \{1, \ldots, N-1\}, \forall k \in \{0, \ldots, K-1\}$:

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

Обозначим $\sigma = \frac{\tau}{h^2},\ g_j^k = g(x_j, t^k)$. Тогда значения функции на верхнем временном слое можно найти из решения **СЛАУ** с трехдиагональной матрицей. Сделаем это с помощью метода прогонки, где **СЛАУ**, кроме крайних двух уравнений, определяется коэффициентами $a_j = \sigma, b_j = -(1 + 2\sigma), c_j = \sigma, d_j = -u_j^k - \tau g_j^{k+1}$ уравнений:

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

## Схема Кранка-Николсона

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

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

$\theta$ - вес неявной части конечно-разностной схемы, $(1 - \theta)$ - вес для явной части.

При значении параметра $\theta = \frac{1}{2}$ мы имеем *схему Кранка-Николсона*.

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

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

```
class ImplictExplict(Schema):
    def SetO(self, O):
        self.O = O

    @staticmethod
    def SweepMethod(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

    def CalculateLine(self, t, x, lastLine):
        a = self.sigma * self.O
        b = -1 - 2 * self.sigma * self.O
        A = [(a, b, a) for _ in range(1, len(x)-1)]
        w = [-(lastLine[i] + self.O * self.tau * self.g(x[i], t) + (1 - self.O) * self.sigma * (lastLine[i - 1] - 2 * lastLine[i] + lastLine[i + 1] + self.h * self.h * self.g(
        koeffs = self.approx.nikolson_0(t, self.h, self.sigma,
                                        self.g, self.l0, lastLine,
                                        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,
                                        self.g, self.l1, lastLine,
                                        self.O, t - self.tau)
        A.append(koeffs[:-1])
        w.append(koeffs[-1])

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

## Апроксимация первых производных

Граничные условия 1-го рода аппроксимируются точно в узлах на границе расчетной области.

Граничные условия 2-го и 3-го рода отличаются тем, что в них присутствует производная первого порядка искомой функции по пространственной переменной. Поэтому для замыкания конечно-разностной схемы необходима их аппроксимация. Простейшим способом является аппроксимация производных направленными разностями первого порядка.

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

    def explict_0(self, t, h, sigma, g, x0, l0, l1, t0):
        pass
    def explict_l(self, t, h, sigma, g, xN, l0, l1, t0):
        pass

    def nikolson_0(self, t, h, sigma, g, x0, l0, O, t0):
```

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

## Двухточечная первого порядка

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

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

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*:

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

```
class Approx2pointFirstOrder(Approx):
    def explict_0(self, t, h, sigma, g, x0, l0, l1, t0):
        return -h * self.f0(t) + l1[1]

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

    def nikolson_0(self, t, h, sigma, g, x0, l0, O, t0):
        return 0, -1, 1, h * self.f0(t)

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

## Трёхточечная второго порядка

Трёхточечная аппроксимация второго порядка в точке $x = 0$ и $x = l$ равны соответственно:

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

Получаем выражения для граничных значений при явном методе:

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

И крайние уравнения для *схемы Кранка-Николсона*:

$$-2\sigma\theta u_0^{k+1} + (2\sigma\theta - 1)u_1^{k+1} = 2\sigma\theta h\phi_0(t^{k+1}) - (u_1^k + \theta\tau g_1^{k+1} + (1-\theta)\sigma(u_0^k - 2u_1^k + u_2^k + h^2 g_1^k))$$
$$(1 - 2\sigma\theta)u_{N-1}^{k+1} + 2\sigma\theta u_N^{k+1} = 2\sigma\theta h\phi_l(t^{k+1}) + (u_{N-1}^k + \theta\tau g_{N-1}^{k+1} + (1-\theta)\sigma(u_{N-2}^k - 2u_{N-1}^k + u_N^k + h^2 g_{N-1}^k))$$

```
class Approx3pointSecondOrder(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, g, x0, l0, O, t0):
        d = 2 * sigma * O * h * self.f0(t)
        d -= l0[1] + O * (t - t0) * g(x0 + h, t)
        d -= (1 - O) * sigma * (l0[0] - 2 * l0[1] + l0[2] + h * h * g(x0 + h, t0))
        return 0, -2 * sigma * O, 2 * sigma * O - 1, d

    def nikolson_l(self, t, h, sigma, g, xN, l0, O, t0):
        d = 2 * sigma * O * h * self.fl(t)
        d += l0[-2] + O * (t - t0) * g(xN - h, t)
        d += (1 - O) * sigma * (l0[-3] - 2 * l0[-2] + l0[-1] + h * h * g(xN - h, t0))
        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: $\frac{u_j^{k+1} - u_j^k}{\tau} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + g_j^k$ для значений $j = 0$ и $j = N$ мы получим формулу граничных значений для явной схемы:

$$u_0^{k+1} = -2\sigma h\phi_0(t^k) + 2\sigma u_1^k + (1 - 2\sigma)u_0^k + \tau g_0^k$$
$$u_N^{k+1} = 2\sigma h\phi_l(t^k) + 2\sigma u_{N-1}^k + (1 - 2\sigma)u_N^k + \tau g_N^k$$

Мы можем использовать аппроксимацию на слое $t^{k+1}$ и тогда получим крайние уравнения для алгоритма 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 явной-неявной схемы:

$$-(2\theta\sigma + 1)u_0^{k+1} + 2\sigma\theta u_1^{k+1} = 2\theta\sigma h\phi_0(t^{k+1}) - (u_0^k + \theta\tau g_0^{k+1} + 2(1-\theta)\sigma(u_1^k - u_0^k - h\phi_0(t^k) + \frac{h^2}{2}g_0^k))$$
$$2\sigma\theta u_{N-1}^{k+1} - (2\theta\sigma + 1)u_N^{k+1} = -2\theta\sigma h\phi_l(t^{k+1}) - (u_N^k + \theta\tau g_N^{k+1} + 2(1-\theta)\sigma(u_{N-1}^k - u_N^k + h\phi_l(t^k) + \frac{h^2}{2}g_N^k))$$

```
class Approx2pointSecondOrder(Approx):
    def explict_0(self, t, h, sigma, g, x0, l0, l1, t0):
```

```
        return -2 * sigma * h * self.f0(t0) + 2 * sigma * l0[1] + \
                    (1 - 2 * sigma) * l0[0] + (t - t0) * g(x0, t0)

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

    def nikolson_0(self, t, h, sigma, g, x0, l0, O, t0):
        d = 2*sigma*O*h*self.f0(t) - l0[0] - O*(t-t0)*g(x0, t)
        d -= 2*(1 - O)*sigma*(l0[1] - l0[0] - h*self.f0(t0) + 0.5*h*h*g(x0, t0))
        return 0, -(2*sigma*O + 1), 2*sigma*O, d

    def nikolson_l(self, t, h, sigma, g, xN, l0, O, t0):
        d = -2*sigma*O*h*self.fl(t) - l0[-1] - O*(t-t0)*g(xN, t)
        d -= 2*(1 - O)*sigma*(l0[-2] - l0[-1] + h*self.fl(t0) + 0.5*h*h*g(xN, t0))
        return 2*sigma*O, -(2*sigma*O + 1), 0, d
```

# Результаты работы

## Зависимость погрешности от параметра $h$

### Вычисление погрешностей

Будем считать погрешность как норму матрицы.

$e = \|\hat{z} - z\|_2$

$\hat{z}, z$ - матрицы вычисленных и реальных значений функции в сетке соответственно.

```
def Error(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
```

Построение зависимости погрешности от шага $h$.

```
def GetStepHandError(solver, real_f):
    h = []
    e = []
    for N in range(3, 20):
        x, y, z = solver(N, 70)
        h.append(solver.h)
        e.append(Error(x, y, z, real_f))
    return h, e
```

### Явная схема

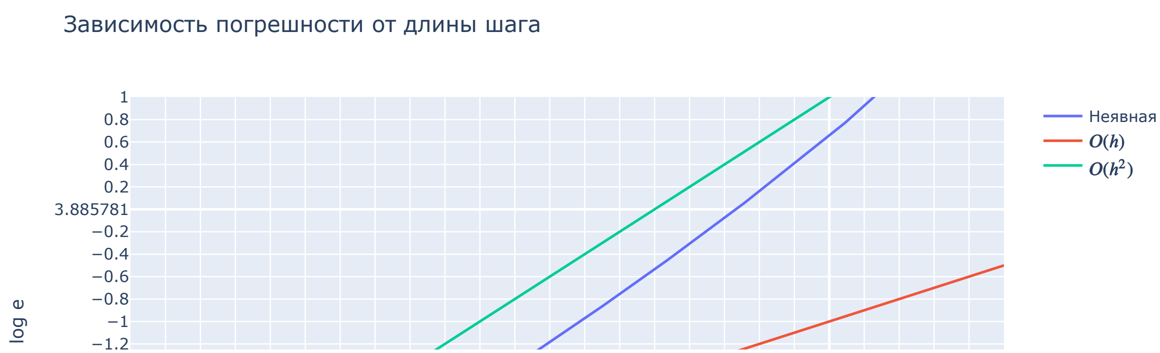
```
explict = ExplictSchema(T = 1, aprx_cls=Approx2pointSecondOrder)

import plotly.offline as offline
from plotly.graph_objs import *

h, e = GetStepHandError(explict, u)

trace1 = Scatter(
    x = h,
    y = e,
    name = 'Явная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(explict.nparange(0, 1.6, 0.1)),
                 tickmode = 'linear',
                 tick0 = 0,
                 dtick = 0.1,
                 range = [0, 1.6],
                 title = 'h'),
    yaxis = dict(tickmode = 'array',
                 tickvals = list(explict.nparange(0, 2.1, 0.2)),
                 range = [0, 2.1],
                 title = 'e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)

trace1 = Scatter(
    x = list(map(math.log, h)),
    y = list(map(math.log, e)),
    name = 'Явная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = [-2, 0.5],
    y = [-3, -0.5],
    name = 'Зависимость $O(h)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-2, 0.5],
    y = [-3, 2],
    name = 'Зависимость $O(h^2)$',
    mode = 'lines',
```

```
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(explict.nparange(-2, 0.5, 0.1)),
                 tickmode = 'linear',
                 tick0 = 0,
                 dtick = 0.1,
                 range = [-2.1, 0.5],
                 title = 'log h'),
    yaxis = dict(tickmode = 'array',
                 tickvals = list(explict.nparange(-3, 1, 0.2)),
                 range = [-3, 1],
                 title = 'log e')
)

fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```


## Неявная схема

```
# ImplictExplict с параметром O=1 это неявная схема
implict = ImplictExplict(T=1, aprx_cls=Approx2pointFirstOrder, O=1)


h, e = GetStepHandError(implict, u)

trace1 = Scatter(
    x = h,
    y = e,
    name = 'Неявная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(implict.nparange(0, 1.6, 0.1)),
                 tickmode = 'linear',
                 tick0 = 0,
                 dtick = 0.1,
                 range = [0, 1.6],
                 title = 'h'),
    yaxis = dict(tickmode = 'array',
                 tickvals = list(implict.nparange(0, 2.1, 0.2)),
                 range = [0, 2.1],
                 title = 'e')
)

fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

```
trace1 = Scatter(
    x = list(map(math.log, h)),
    y = list(map(math.log, e)),
    name = 'Неявная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = [-2, 0.5],
    y = [-3, -0.5],
    name = 'Зависимость $O(h)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-2, 0.5],
    y = [-3, 2],
    name = 'Зависимость $O(h^2)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(implict.nparange(-2, 0.5, 0.1)),
                 tickmode = 'linear',
                 tick0 = 0,
                 dtick = 0.1,
                 range = [-2, 0.5],
                 title = 'log h'),
    yaxis = dict(tickmode = 'array',
                 tickvals = list(implict.nparange(-3, 1, 0.2)),
                 range = [-3, 1],
                 title = 'log e')
)

fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```


## Схема Кранка-Николсона

```
# ImplictExplict с параметром O=0.5 это схема Кранка-Николсона
krank = ImplictExplict(T=1, aprx_cls=Approx2pointSecondOrder, O=0.5)

h, e = GetStepHandError(krank, u)

trace1 = Scatter(
    x = h,
    y = e,
    name = 'Кранк-Николсон',
    mode = 'lines',
    text = ('(x, y)'),
```

```
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(tickmode = 'array',
                 tickvals = list(np.arange(0, 1.6, 0.1)),
                 title = 'h'),
    yaxis = dict(tickmode = 'array',
                 tickvals = list(np.arange(0, 2.1, 0.2)),
                 range = [0, 2.1],
                 title = 'e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)

trace1 = Scatter(
    x = list(map(math.log, h)),
    y = list(map(math.log, e)),
    name = 'Кранк-Николсон',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = [-2, 0.5],
    y = [-3, -0.5],
    name = 'Зависимость $O(h)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-2, 0.5],
    y = [-3, 2],
    name = 'Зависимость $O(h^2)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(krank.nparange(-2, 0.5, 0.1)),
                 tickmode = 'linear',
                 tick0 = -2,
                 dtick = 0.1,
                 range = [-2, 0.5],
                 title = 'log h'),
    yaxis = dict(tickmode = 'array',
```

```
                  tickvals = list(krank.nparange(-3, 1, 0.2)),
                  range = [-3, 1],
                  title = 'log e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

Зависимость погрешности от длины шага

## Зависимость погрешности от параметра $\tau$

### Вычисление погрешности

Построение зависимости погрешности от параметра $\tau$.

```
def GetTandError(solver, real_f):
    tau, e = [], []
    for K in range(3, 70):
        x, y, z = solver(K = K)
        tau.append(solver.tau)
        e.append(Error(x, y, z, real_f))
    return tau, e
```

### Явная схема

```
explict = ExplictSchema(T=5, aprx_cls=Approx2pointSecondOrder)
```

```
tau, e = GetTandError(explict, u)

trace1 = Scatter(
    x = tau,
    y = e,
    name = 'Явная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(tickmode = 'array',
                 tickvals = list(explict.nparange(0, 2.5, 0.1)),
                 range = [0, 2.6],
                 title = 'h'),
    yaxis = dict(title = 'e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)

trace1 = Scatter(
    x = list(map(math.log, tau)),
    y = list(map(math.log, e)),
```

```
    name = 'Явная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от времени',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(explict.nparange(-3, 1, 0.2)),
                 tickmode = 'linear',
                 tick0 = -3,
                 dtick = 0.2,
                 range = [-3, 1],
                 title = 'log h'),
    yaxis = dict(title = 'log e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

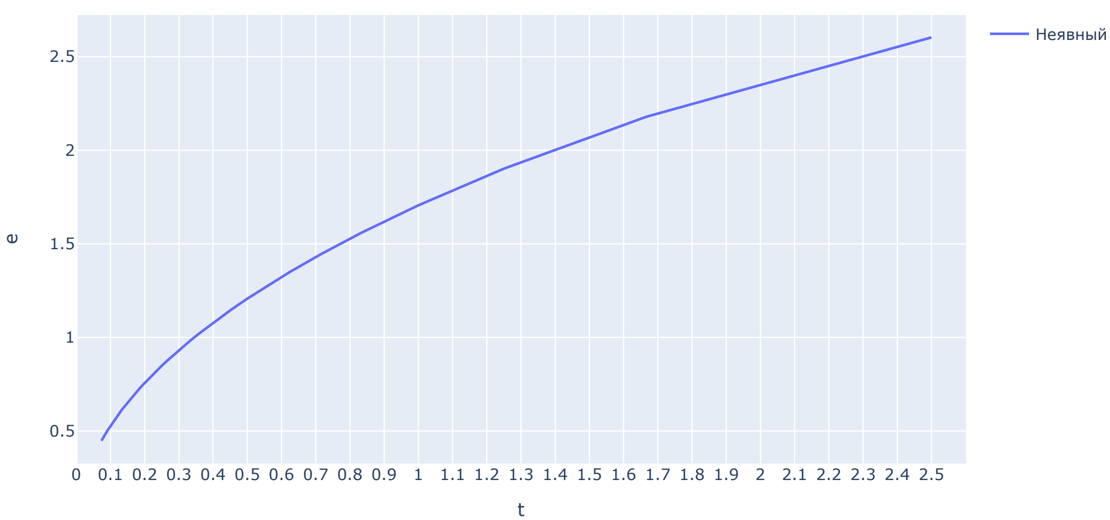
```
<ipython-input-4-1b4bcaef4826>:5: UserWarning:

Sigma > 0.5
```


## Неявная схема

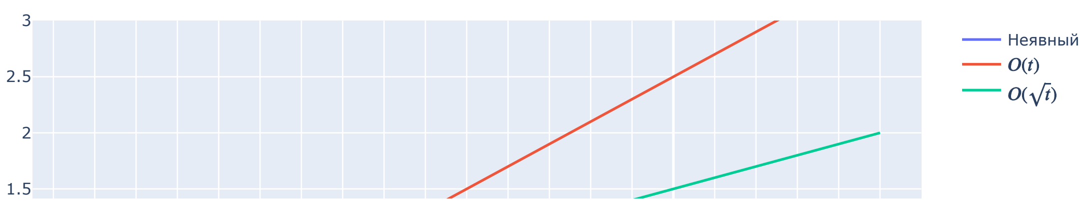
```
# ImplictExplict с параметром O=1 это неявная схема
implict = ImplictExplict(T = 5, aprx_cls=Approx2pointSecondOrder, O=1)


tau, e = GetTandError(implict, u)

trace1 = Scatter(
    x = tau,
    y = e,
    name = 'Неявный',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от мелкости разбиения по времени',
    xaxis = dict(tickmode = 'array',
                 tickvals = list(explict.nparange(0, 2.5, 0.1)),
                 range = [0, 2.6],
                 title = 't'),
    yaxis = dict(title = 'e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)

trace1 = Scatter(
    x = list(map(math.log, tau)),
```

```
    y = list(map(math.log, e)),
    name = 'Неявный',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = [-3, 1],
    y = [-0.5, 3.5],
    name = 'Зависимость $O(t)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-3, 1],
    y = [0, 2],
    name = 'Зависимость $O(\sqrt{t})$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3]

layout = Layout(
    title = 'Зависимость погрешности от мелкости разбиения по времени',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(explict.nparange(-3, 1, 0.2)),
                 tickmode = 'linear',
                 tick0 = -3,
                 dtick = 0.2,
                 range = [-3.1, 1.2],
                 title = 'log t'),
    yaxis = dict(range = [-1, 3],
                 title = 'log e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

Зависимость погрешности от мелкости разбиения по времени




Зависимость погрешности от мелкости разбиения по времени



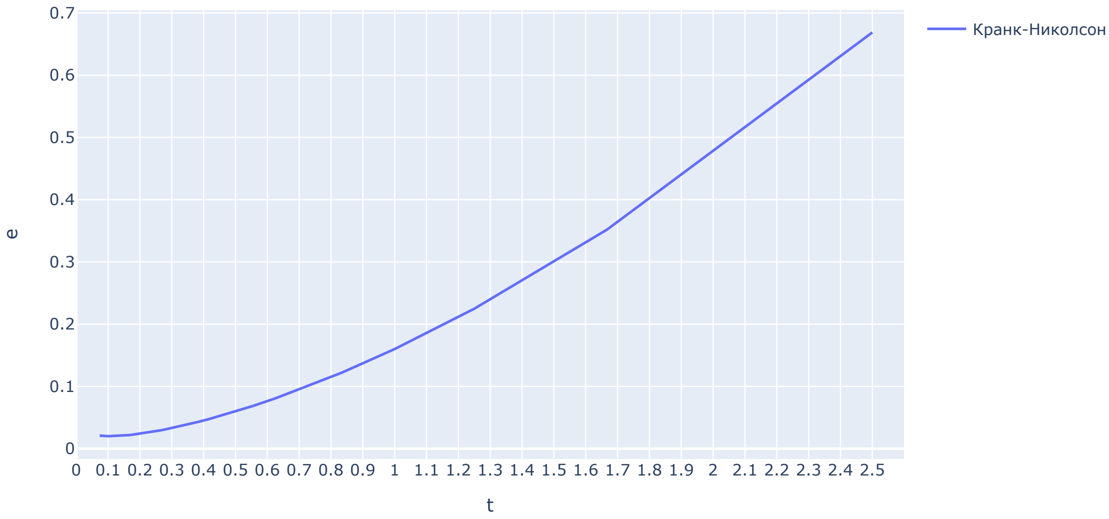

## Схема Кранка-Николсона

```
krank = ImplictExplict(aprx_cls=Approx2pointSecondOrder)

tau, e = GetTandError(krank, u)

trace1 = Scatter(
    x = tau,
    y = e,
    name = 'Кранк-Николсон',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]
```

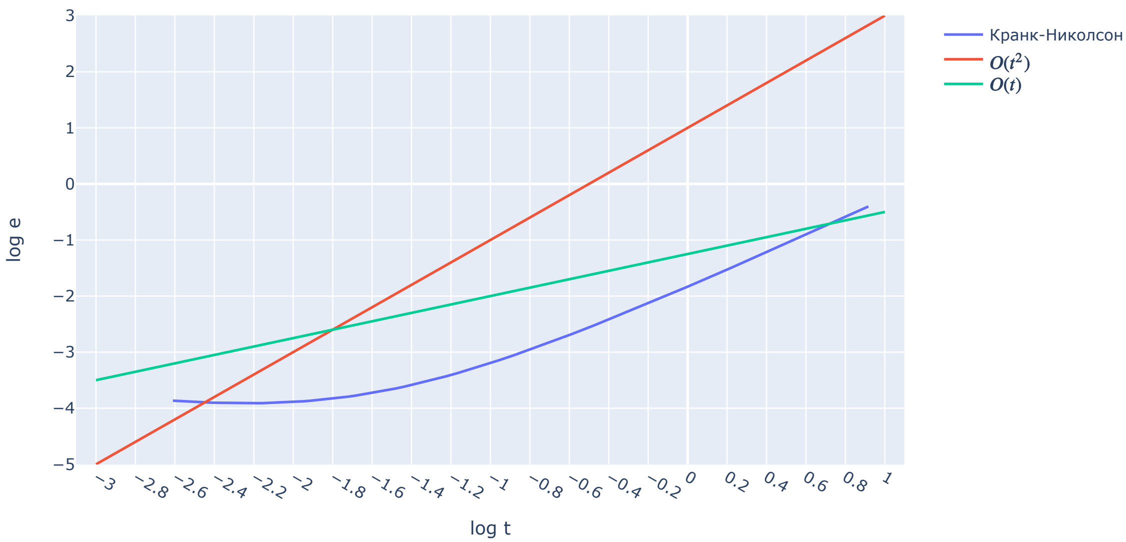
```
layout = Layout(
    title = 'Зависимость погрешности от мелкости разбиения по времени',
    xaxis = dict(tickmode = 'array',
                 tickvals = list(krank.nparange(0, 2.5, 0.1)),
                 range = [0, 2.6],
                 title = 't'),
    yaxis = dict(title = 'e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

```
trace1 = Scatter(
    x = list(map(math.log, tau)),
    y = list(map(math.log, e)),
    name = 'Кранк-Николсон',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace2 = Scatter(
    x = [-3, 1],
    y = [-5, 3],
    name = 'Зависимость $O(t^2)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-3, 1],
    y = [-3.5, -0.5],
    name = 'Зависимость $O(t)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1, trace2, trace3]

layout = Layout(
    title = 'Зависимость погрешности от мелкости разбиения по времени',
    xaxis = dict(#tickmode = 'array',
                 #tickvals = list(krank.nparange(-3, 1, 0.2)),
                 tickmode = 'linear',
                 tick0 = -3,
                 dtick = 0.2,
                 range = [-3.1, 1.1],
                 title = 'log t'),
    yaxis = dict(range = [-5, 3],
                 title = 'log e')
)


fig = Figure(data = data, layout = layout)
offline.iplot(fig)
```

Зависимость погрешности от мелкости разбиения по времени




Зависимость погрешности от мелкости разбиения по времени




Вывод:

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 $\tau$ и $h$.