Московский авиационный институт

(Национальный исследовательский университет)

Институт №8 «Информационные технологии и прикладная математика»

Кафедра вычислительной математики и программирования

Лабораторная работа № 6
по курсу «Численные методы».

Тема: «МЕТОД КОНЕЧНЫХ РАЗНОСТЕЙ ДЛЯ РЕШЕНИЯ УРАВНЕНИЙ ГИПЕРБОЛИЧЕСКОГО ТИПА».

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

Москва, 2022

Лабораторная работа №6

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 №7

```
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:

$$\frac{\partial^2 u}{\partial t^2} + 2\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + 2\frac{\partial u}{\partial x} - 3u$$

Граничные условия:

$$\begin{cases} u(0,\, t) = e^{-t}\cos 2t \\ u(\frac{\pi}{2},\, t) = 0 \\ u(x,\, 0) = \psi_1(x) = e^{-x}\cos x \\ u_t(x, 0) = \psi_2(x) = -e^{-x}\cos x \end{cases}$$

Аналитическое решение:

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

Задаем наши данные из условия задачи.

```
def psi_1(x):
    return math.exp(-x) * math.cos(x)

def psi_2(x):
    return -math.exp(-x) * math.cos(x)

def phi_0(t):
    return math.exp(-t) * math.cos(2 * t)

def phi_l(t):
    return 0

def dpsi2_dx2(x):
    return 2 * math.sin(x) * math.exp(-x)

def u(x, t):
    return math.exp(-x - t) * math.cos(2 * t) * math.cos(x)
```

Конечно-разностная схема

Так как 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 слоях известны, попробуем определить значения функции на следующем временном слое t^{k+1} с помощью 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:

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

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

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

```
class Schema:
    def __init__(self, psi1=psi_1, psi2=psi_2,
                 diffpsi2=dpsi2_dx2, f0=phi_0, fl=phi_l,
                 l0=0, l1=math.pi / 2, T=5, order2nd=True):
        self.psi1 = psi1
        self.diffpsi = diffpsi2
        self.psi2 = psi2
        self.T = T
        self.l0 = l0
        self.l1 = l1
        self.tau = None
        self.h = None
        self.order = order2nd
        self.sigma = None
        self.f0 = f0
        self.fl = fl

    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.tau / (self.h*self.h)

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

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

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

    def CalculateLine(self, t, x, lastLine1, lastLine2):
        pass
```

```
    def __call__(self, N=30, K=70):
        N, K = N - 1, K - 1
        self.CalculateTau(K)
        self.CalculateH(N)
        self.CalculateSigma()
        ans = []
        x = list(self.nparange(self.l0, self.l1, self.h))
        lastLine = list(map(self.psi1, x))
        ans.append(list(lastLine))
        if self.order:
            lastLine = list(map(lambda a: self.psi1(a) + self.tau * self.psi2(a) + self.tau * self.tau * self.diffpsi(a) / 2, x))
        else:
            lastLine = list(map(lambda a: self.psi1(a) + self.tau * self.psi2(a), x))
        ans.append(list(lastLine))
        X = [x, x]
        Y = [[0.0 for _ in x]]
        Y.append([self.tau for _ in x])
        for t in self.nparange(self.tau + self.tau, self.T, self.tau):
            ans.append(self.CalculateLine(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:

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

Выражаем явную схему конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} + \frac{u^{k+1} - u^{k-1}}{\tau} = \frac{u_{j-1}^k - 2u_j^k + u_{j+1}^k}{h^2} + \frac{u_{j+1}^k - u_{j-1}^k}{h} - 3u_j^k$$

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

$$u_j^{k+1} = \frac{\sigma(u_{j+1}^k - 2u_j^k + u_{j-1}^k) + \dfrac{u_{j+1}^k - u_{j-1}^k}{h} - 3\tau^2 u_j^k + \tau u_j^{k-1} + 2u_j^k - u_j^{k-1}}{1+\tau}$$

Граничные же значения u_0^{k+1} и u_N^{k+1} определяются граничными условиями $u(0,t)$ и $u(l,t)$.

```
class ExplictSchema(Schema):
    def CalculateSigma(self):
        self.sigma =  self.tau * self.tau / (self.h * self.h)
        if self.sigma > 1:
            warnings.warn("Sigma > 1")

    def CalculateLine(self, t, x, lastLine1, lastLine2):
        line = [None for _ in lastLine1]
        for i in range(1, len(x) - 1):
            line[i] = self.sigma * (lastLine1[i - 1] - 2 * lastLine1[i] + lastLine1[i + 1])
            line[i] -= 3 * self.tau * self.tau * lastLine1[i]
```

```
            line[i] += 2 * lastLine1[i]
            line[i] += (self.tau - 1) * lastLine2[i]
            line[i] += self.tau * self.tau * (lastLine1[i + 1] - lastLine1[i - 1]) / self.h
            line[i] /= (1 + self.tau)
        line[0] = self.f0(t)
        line[-1] = self.fl(t)
        return line
```

Неявная конечно-разностная схема

Аппроксимируем вторую производную по значениям верхнего временного слоя t^{k+1}:

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

Тогда у нас получается явная схема конечно-разностного метода во внутренних узлах сетки:

$$\frac{u_j^{k+1} - 2u_j^k + u_j^{k-1}}{\tau^2} + \frac{u^{k+1} - u^{k-1}}{\tau} = \frac{u_{j-1}^{k+1} - 2u_j^{k+1} + u_{j+1}^{k+1}}{h^2} + \frac{u_{j+1}^{k+1} - u_{j-1}^{k+1}}{h} - 3u_j^{k+1}$$

Введем обозначение $\sigma = \frac{\tau^2}{h^2}$. Следовательно значения функции на слое можно найти из решения СЛАУ с трехдиагональной матрицей. Сделаем это с помощью метода прогонки, где СЛАУ, кроме крайних двух уравнений, будет определяться коэффициентами:

$a_j = 1 - h$,

$b_j = -2 - 3h^2 - \frac{1+\tau}{\sigma}$,

$c_j = 1 + h$,

$d_j = \frac{-2u_j^k + u_j^{k-1} - \tau u_j^{k-1}}{\sigma}$ уравнений:

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

```
class ImplictSchema(Schema):
    @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, lastLine1, lastLine2):
        a = 1 - self.h
```

```
        c = 1 + self.h
        b = -2 - 3*self.h*self.h - (1 + self.tau)/self.sigma
        A = [(a, b, c) for _ in range(2, len(x) - 2)]
        w = [((1 - self.tau) * lastLine2[i] - 2 * lastLine1[i]) / self.sigma for i in range(2, len(x)-2)]
        koeffs = (0, b, c, (((1 - self.tau) * lastLine2[1] - 2 * lastLine1[1]) / self.sigma) - a * self.f0(t))
        A.insert(0, koeffs[:-1])
        w.insert(0, koeffs[-1])
        koeffs = (a, b, 0, (((1 - self.tau) * lastLine2[-2] - 2 * lastLine1[-2]) / self.sigma) - c * self.fl(t))
        A.append(koeffs[:-1])
        w.append(koeffs[-1])
        ans = self.SweepMethod(A, w)
        ans.insert(0, self.f0(t))
        ans.append(self.fl(t))
        return ans
```

Полученные результаты работы

Зависимость погрешности от параметра

Вычисление погрешностей

```
def Error(x, y, z, f):
    ans = 0.0
    for i in range(len(z)):
        for j in range(len(z[i])):
            tmp = abs(z[i][j] - f(x[i][j], y[i][j]))
            ans = tmp if tmp > ans else ans
    return ans
```

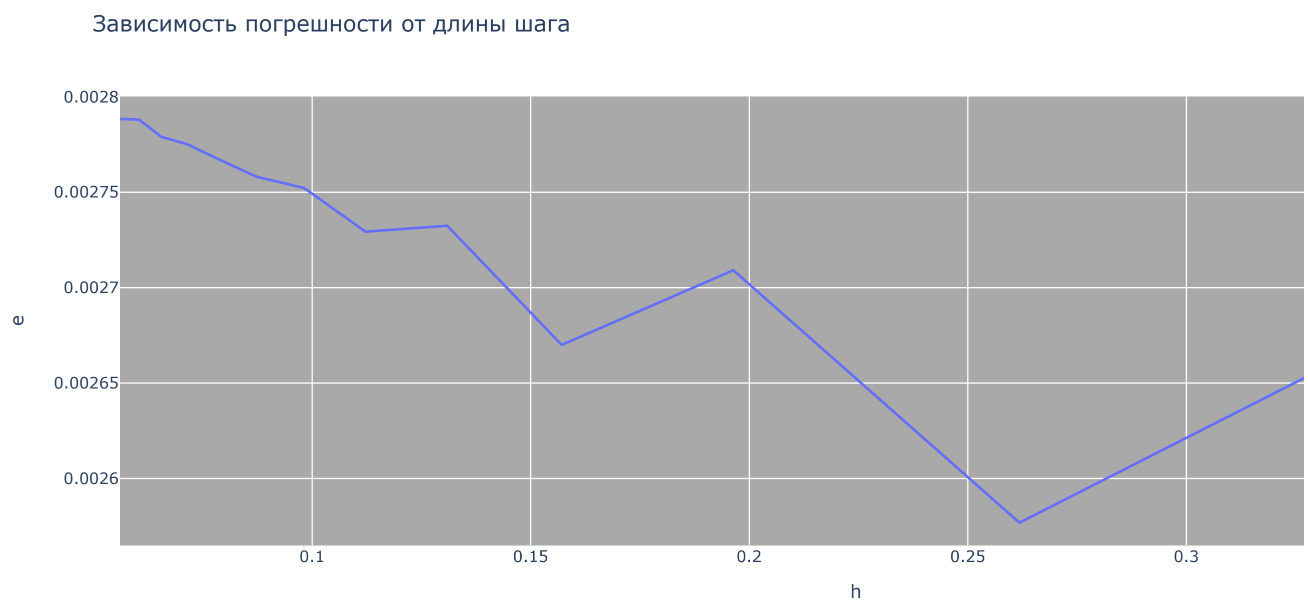
```
def GetStepHandEror(solver, real_f):
    h = []
    e = []
    for N in range(5, 30, 2):
        x, y, z = solver(N, 100)
        h.append(solver.h)
        e.append(Error(x, y, z, real_f))
    return h, e
```

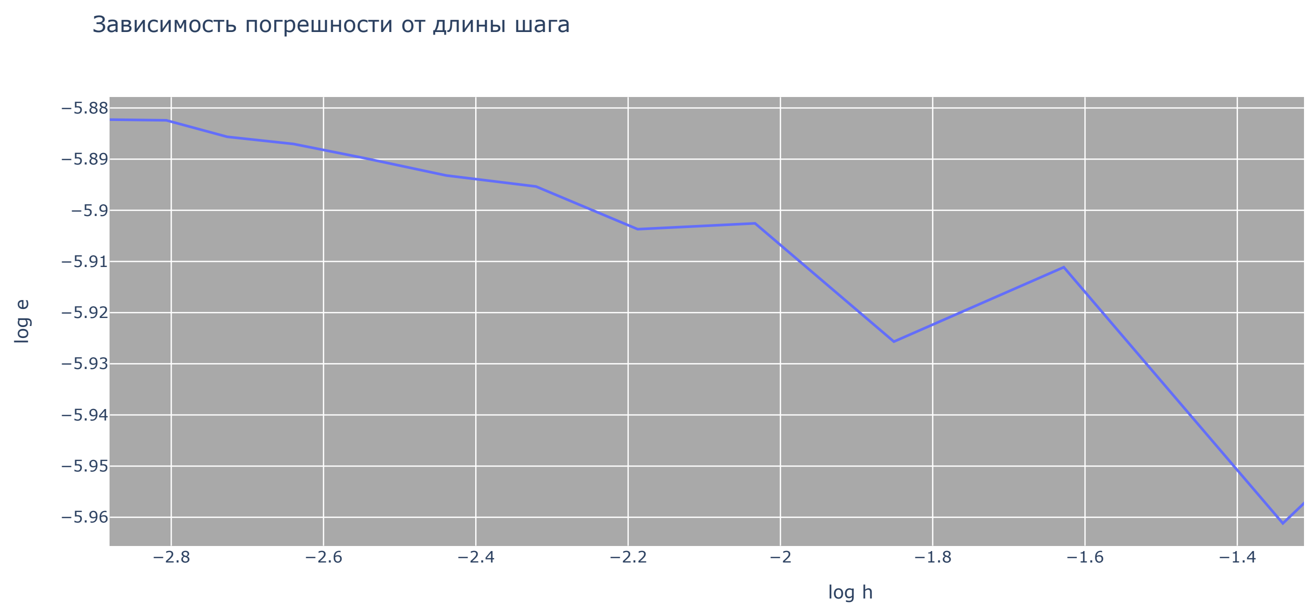
Явная схема

```
explict = ExplictSchema(T=1)
```

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

h, e = GetStepHandEror(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(title = 'h'),
    yaxis = dict(title = 'e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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 = [-3.5, -1],
    y = [-3, -0.5],
    name = 'Зависимость $O(h)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-3.5, -1],
    y = [-3, 2],
    name = 'Зависимость $O(h^2)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от длины шага',
    xaxis = dict(title = 'log h'),
    yaxis = dict(title = 'log e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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





Неявная схема

```
implict = ImplictSchema(T=1)


h, e = GetStepHandEror(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(title = 'h'),
    yaxis = dict(title = 'e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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 = [-3.5, -1],
    y = [-6, -1],
    name = 'Зависимость $O(h)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-3.5, -1],
    y = [-6, 4],
    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(title = 'log h'),
    yaxis = dict(title = 'log e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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

Зависимость погрешности от длины шага

Зависимость погрешности от параметра τ.

Вычисление погрешности

Построение зависимости погрешности от параметра τ.

```
def GetTandError(methodToSolve, realF):
    tau, e = [], []
    for K in range(3, 90):
        x, y, z = methodToSolve(K=K)
        tau.append(methodToSolve.tau)
        e.append(Error(x, y, z, realF))
    return tau, e
```

Явная схема

```
explict = ExplictSchema(T=1)
```

Зависимость погрешности от длины шага

```
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 мелкости разбиения по времени',
    xaxis = dict(title = 't'),
    yaxis = dict(title = 'e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)


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

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

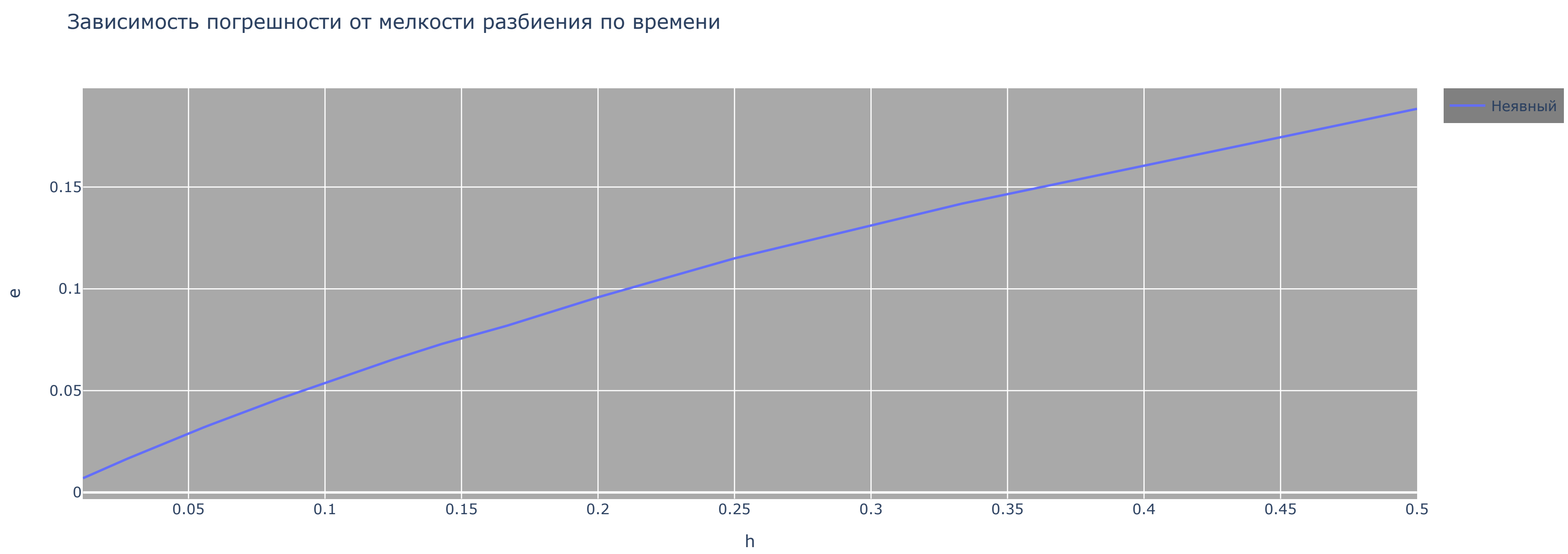
```
    name = 'Явная',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

data = [trace1]

layout = Layout(
    title = 'Зависимость погрешности от мелкости разбиения по времени',
    xaxis = dict(title = 'log t'),
    yaxis = dict(title = 'log e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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

```
<ipython-input-6-d161589d8095>:5: UserWarning:

Sigma > 1
```

Неявная схема

```
implict = ImplictSchema(T=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(title = 'h'),
    yaxis = dict(title = 'e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)


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 = [-4, 0],
    y = [-4.5, -1.5],
    name = 'Зависимость $O(t)$',
    mode = 'lines',
    text = ('(x, y)'),
    showlegend = True
)

trace3 = Scatter(
    x = [-4, 0],
    y = [-4, -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(title = 'log t'),
    yaxis = dict(title = 'log e'),
    plot_bgcolor = 'darkgray',
    paper_bgcolor = 'gray'
)

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

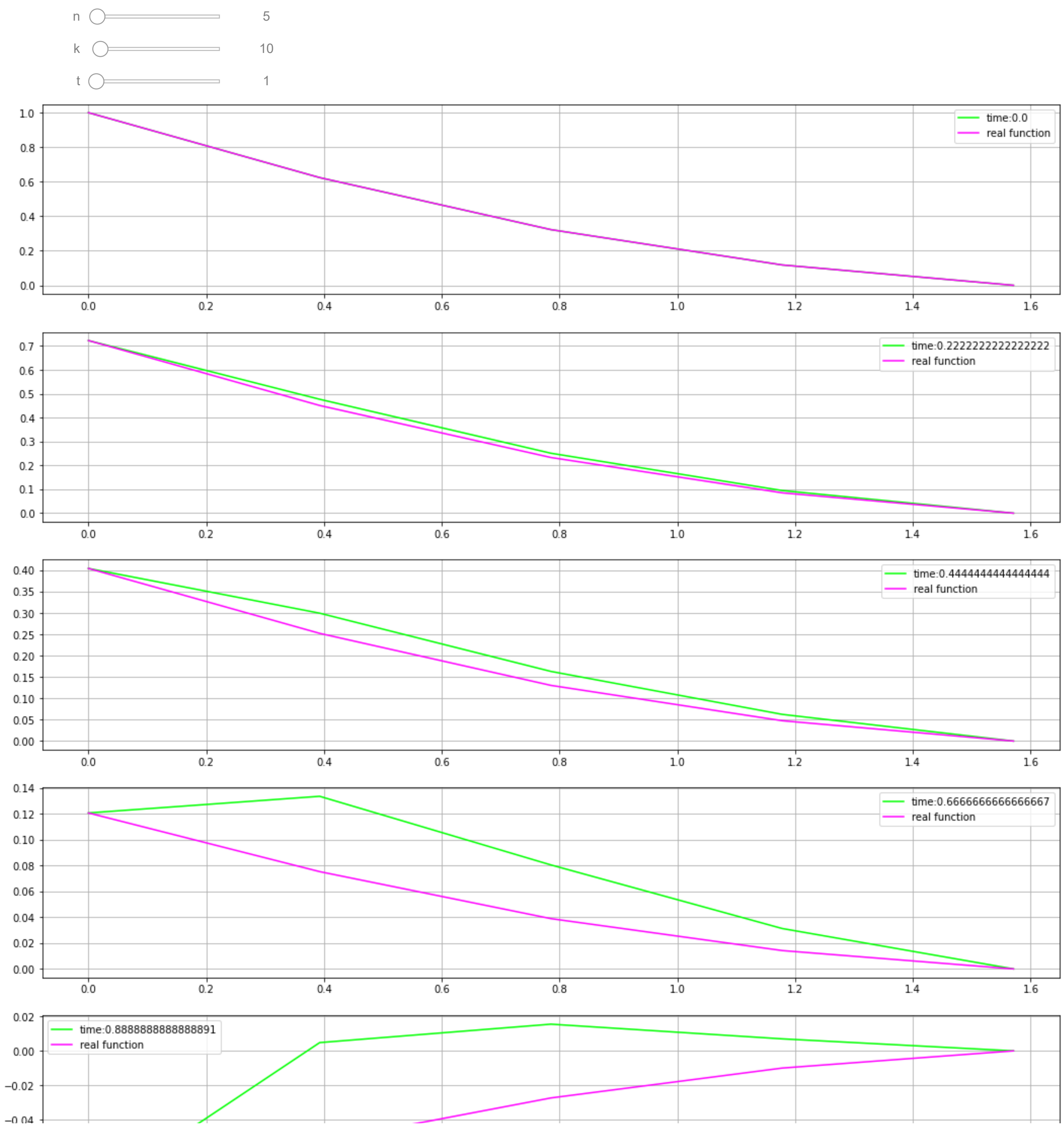


Решение

Зависимость погрешности от мелкости разбиения по времени

!

```
#k - количество разбиений
def plotDependenceT(n=5, k=10, t=1):
    schema = ImplictSchema(T=t)
    x, y, z = schema(N=n, K=k)
    plt.figure(figsize=(18, 20))
    for i in range(1, 6):
        plt.subplot(5, 1, i)
        j = (k * (i - 1)) // 5
        X = x[j]
        Z = z[j]
        T = y[j][0]

        plt.plot(X, Z, label="time:" + str(T), color='#00FF00')
        plt.plot(X, list(map(lambda o: u(o, T), X)), label='real function', color='#FF00FF')
        plt.grid()
        plt.legend()

interact(plotDependenceT, n=(4, 200, 2), k=(5, 200, 3), t=(1, 10, 1))
None
```



n 5
k 10
t 1
time:0.0
real function
time:0.2222222222222222
real function
time:0.4444444444444444
real function
time:0.6666666666666667
real function
time:0.8888888888888891
real function

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.