

**Московский Авиационный Институт (Национальный
Исследовательский Университет)**

Факультет: “Информационные технологии и прикладная математика”

Кафедра: 806 “Вычислительная математика и программирование”

**Отчет по лабораторной работе №6
по курсу «Численные методы»**

Студент:

Полей-Добронравова

Амелия Вадимовна

Группа: М8О-407Б,

№ по списку 20

Дата: 05.12.2021

Итоговая оценка:

Подпись преподавателя:

Москва, 2021

Постановка задачи

Лабораторная работа 2

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: двух точечная аппроксимация с первым порядком, трех точечная аппроксимация со вторым порядком, двух точечная 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, h .

Вариант 10

10.

$$\frac{\partial^2 u}{\partial t^2} + 3 \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial u}{\partial x} - u - \cos x \exp(-t),$$

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

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

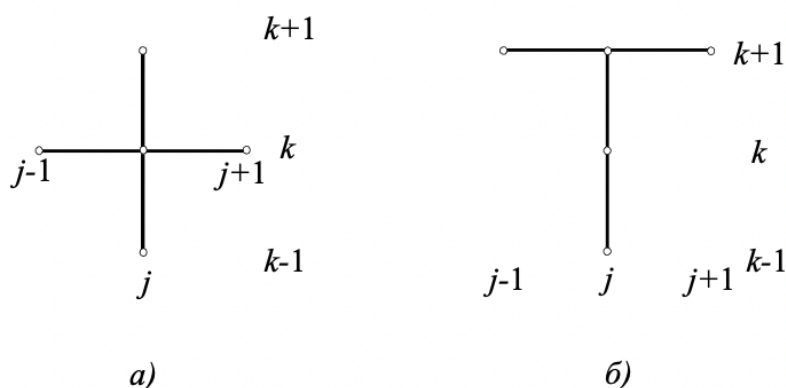
$$u(x, 0) = \sin x,$$

$$u_t(x, 0) = -\sin x.$$

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

Описание программы

Выбор схемы осуществляется с помощью параметра схемы. Если он равен 0, отрисовка явной схемы. Если он равен 1, неявная схема.



При этом схема (5.38) является *явной*. С ее помощью решение u_j^{k+1} , $j = \overline{1, N-1}$, $k = 1, 2, \dots$ определяется сразу, поскольку значения сеточных функции u_j^{k-1} , u_j^k на нижних временных слоях должны быть известны. В соответствии с шаблоном для этой схемы порядок аппроксимации равен двум, как по пространственной, так и по временной переменной. При этом явная конечно-разностная схема (5.38) для волнового уравнения условно устойчива с условием $\sigma = \frac{a^2 \tau^2}{h^2} < 1$, накладываемым на сеточные характеристики τ и h .

Схема (5.39) является *неявной схемой* и обладает абсолютной устойчивостью. Ее можно свести к СЛАУ с трехдиагональной матрицей, решаемой методом прогонки.

В обеих схемах необходимо знать значения u_j^{k-1} , u_j^k , $j = \overline{1, N-1}$, $k = 1, 2, \dots$ на нижних временных слоях. Для $k = 1$ это делается следующим образом:







$$u_j^0 = \psi_1(x_j), \quad j = \overline{0, N}, \quad (5.40)$$

где $\psi_1(x)$ - функция из начального условия (5.36).

В функции **plot_ex()** идёт отрисовка явной схемы двух графиков, в **plot_im()** отрисовка неявной схемы:

- 1) Аналитическое и численные решения с разными типами аппроксимации.
- 2) Погрешность численных решений с разными типами аппроксимации.

Типы аппроксимации:

	$1\pi / 1\pi - 2\tau$
	$1\pi / 2\pi - 2\tau$
	$1\pi / 2\pi - 3\tau$
	$2\pi / 1\pi - 2\tau$
	$2\pi / 2\pi - 2\tau$
	$2\pi / 2\pi - 3\tau$

Функция **true_fval()** - формула аналитического решения.

Функция **explicit()** - численное решение явной схемы, параметр *j* указывает на то, какой тип аппроксимации использовать.

Функция **implicit()** - численное решение неявной схемы, параметр *j* указывает на то, какой тип аппроксимации использовать.

Исходный код

```
import numpy as np
import math
import matplotlib.pyplot as plt
from math import sqrt
from tkinter import *
from PIL import Image, ImageTk
```

```
def norma(a):
    norm = 0
    for i in range(len(a)):
        norm += a[i]**2
    return sqrt(norm)
```

```
def prog(a, b, c, d):
    n = len(d)
    x=np.zeros(n)
    p = [-c[0] / b[0]]
    q = [d[0] / b[0]]
    for i in range(1, n):
        p.append(-c[i] / (b[i] + a[i] * p[i - 1]))
        q.append((d[i] - a[i] * q[i - 1]) / (b[i] + a[i] * p[i - 1]))
    x[-1] = q[-1]
    for i in reversed(range(n-1)):
        x[i] = p[i] * x[i + 1] + q[i]
    return x
```

```
def psi0(x):
    return np.sin(x)
```

```
def dpsi0(x):
    return np.cos(x)
```

```
def d2psi0(x):
    return - np.sin(x)
```

```
def psi1(x):
    return - np.sin(x)
```

```
def phi0(t):
    return np.exp(-t)
```

```
def phi1(t):
    return - np.exp(-t)
```

```
def true_fval(x, t):
    return np.exp(-t)*np.sin(x)
```

```
def f(x,t):
    return -np.cos(x) * np.exp(-t)
```

```
def explicit(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, apr0, apr1):
    x = np.arange(lb, ub + h, h)
    t = np.arange(0, T + tau, tau)
    U = np.zeros((len(t),len(x)))
    for j in range(len(x)):
        U[0,j] = psi0(x[j])
        if apr0 == 1:
```

```

        U[1,j] = U[0,j] + tau*psi1(x[j])
    if apr0 == 2:
        U[1,j] = psi0(x[j]) + tau*psi1(x[j]) +
0.5*(tau**2)*(-d*psi1(x[j]) + a*d2psi0(x[j]) +
b*dpsi0(x[j]) + c*psi0(x[j]) + f(x[j],0) )
    for i in range(2,len(t)):
        for j in range(1,len(x)-1):
            U[i,j] = (4*U[i-1,j] + (d*tau - 2)*U[i-2,j] +
2*a*(tau**2)*(U[i-1,j-1] - 2*U[i-1,j] +
U[i-1,j+1]))/(h**2) + b*(tau**2)*(U[i-1,j+1] -
U[i-1,j-1])/h + 2*(tau**2)*(c*U[i-1,j] + f(x[j],t[i-1]))/(2
+ d*tau)
        if apr1 == 1:
            U[i,0] = (h * phi0(t[i]) - alfa * U[i,1])/(h * beta -
alfa)
            U[i,-1] = (h * phi1(t[i]) + gama * U[i, -2] )/(h *
delta + gama)
        if apr1 == 2:
            U[i,0] = ((h*b - 2*a)*phi0(t[i])/alfa +
2*a*U[i,1]/h + (2*h/(tau**2) + h*d/tau)*U[i-1,0] +
h*U[i-2,0]/(tau**2) + h*f(x[0],t[i]))/(h*b -
2*a)*beta/alfa + 2*a/h + h/(tau**2) + h*d/tau - h*c)
            U[i,-1] = (phi1(t[i])*(2*a + h*b)/gama +
2*a*U[i,-2]/h + (2*h/(tau**2) + h*d/tau)*U[i-1,-1] +
h*U[i-2,-1]/(tau**2) + h*f(x[-1],t[i]))/(2*a +
h*b)*delta/gama + 2*a/h + h/(tau**2) + h*d/tau - h*c)
        if apr1 == 3:
            U[i,0] = (2*h*phi0(t[i]) - 4*alfa*U[i,1] +
alfa*U[i,2])/(2*h*beta - 3*alfa)
            U[i,-1] = (2*h*phi1(t[i]) + 4*gama*U[i,-2] -
gama*U[i,-3])/(2*h*delta + 3*gama)
    return U

def plot_ex(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, sigm, k):
    x = np.arange(lb, ub + h, h)
    t = np.arange(0, T + tau, tau)
    plt.figure(1)
    plt.title('Явная схема, t = ' + str(t[k]))
    plt.grid()
    plt.plot(x, true_fval(x, t[k]), color = 'red', label =
'аналитическое решение')
    colors = ['green', 'lime', 'pink', 'blue', 'orange', 'indigo']
    aprs = ['1π / 1π - 2τ', '1π / 2π - 2τ', '1π / 2π - 3τ', '2π / 1π
- 2τ', '2π / 2π - 2τ', '2π / 2π - 3τ']
    for apr0 in range(1,3):
        for apr1 in range(1,4):
            U = explicit(a, b, c, d, alfa, beta, gama, delta, lb,
ub, h, tau, T, apr0, apr1)
            plt.plot(x, U[k,:], color = colors[6 - (6//apr0) +
apr1 - 1], label = aprs[6 - (6//apr0) + apr1 - 1])
            plt.legend()
            if apr0 == 1 and apr1 == 1:
                A11 = U
            if apr0 == 1 and apr1 == 2:
                A12 = U
            if apr0 == 1 and apr1 == 3:
                A13 = U

```

```

        if apr0 == 2 and apr1 == 1:
            A21 = U
        if apr0 == 2 and apr1 == 2:
            A22 = U
        if apr0 == 2 and apr1 == 3:
            A23 = U
    plt.figure(2)
    plt.title('Погрешность')
    plt.grid()
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A11[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'green', label = aprs[0])
    "eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A12[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'lime', label = aprs[1])
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A13[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'pink', label = aprs[2])"
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A21[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'blue', label = aprs[3])
    "eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A22[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'orange', label = aprs[4])
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i]) - A23[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'indigo', label = aprs[5])
    plt.legend()
    plt.show()
    return

def implicit(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, apr0, apr1):
    x = np.arange(lb, ub + h, h)
    t = np.arange(0, T + tau, tau)
    U = np.zeros((len(t),len(x)))
    for j in range(len(x)):
        U[0,j] = psi0(x[j])
        if apr0 == 1:
            U[1,j] = U[0,j] + tau*psi1(x[j])
        if apr0 == 2:
            U[1,j] = psi0(x[j]) + tau*psi1(x[j]) +
0.5*(tau**2)*(-d*psi1(x[j]) + a*d2psi0(x[j]) +
b*dpsi0(x[j]) + c*psi0(x[j]) + f(x[j],0) )
        if apr1 == 1:
            for i in range(2, len(t)):

```

```

aa = np.zeros(len(x))
bb = np.zeros(len(x))
cc = np.zeros(len(x))
dd = np.zeros(len(x))
dd[0] = h*phi0(t[i])
dd[-1] = h*phi1(t[i])
aa[-1] = -gama
bb[0] = beta*h - alfa
bb[-1] = delta*h + gama
cc[0] = alfa
for j in range(1, len(x) - 1):
    dd[j] = -4*(h**2)*U[i-1,j]/(tau**2) +
(2*(h**2)/(tau**2) - d*(h**2)/tau)*U[i-2,j] -
2*(h**2)*f(x[j], t[i])
    aa[j] = 2*a - h*b
    bb[j] = 2*(h**2)*(-d/(2*tau) - 1/(tau**2) + c )
- 4*a
    cc[j] = h*b + 2*a
    xx = prog(aa, bb, cc, dd)
    for j in range(len(x)):
        U[i, j] = xx[j]
if apr1 == 2:
    for i in range(2, len(t)):
        aa = np.zeros(len(x))
        bb = np.zeros(len(x))
        cc = np.zeros(len(x))
        dd = np.zeros(len(x))
        dd[0] = phi0(t[i])*(h*b - 2*a)/alfa +
(2*h/(tau**2) + d*h/tau)*U[i-1,0] + h/(tau**2)*U[i-2,0]
+ h*f(x[0], t[i])
        dd[-1] = phi1(t[i])*(h*b + 2*a)/gama +
(2*h/(tau**2) + d*h/tau)*U[i-1,-1] +
h/(tau**2)*U[i-2,-1] + h*f(x[-1], t[i])
        aa[-1] = -2*a/h
        bb[0] = 2*a/h + h/(tau**2) + h*d/tau - c*h +
beta*(h*b - 2*a)/alfa
        bb[-1] = 2*a/h + h/(tau**2) + h*d/tau - c*h +
delta*(h*b + 2*a)/gama
        cc[0] = -2*a/h
        for j in range(1, len(x) - 1):
            dd[j] = -4*(h**2)*U[i-1,j]/(tau**2) +
(2*(h**2)/(tau**2) - d*(h**2)/tau)*U[i-2,j] -
2*(h**2)*f(x[j], t[i])
            #dd[j] = -2*d*(h**2)*U[i-1,j]/tau +
2*(h**2)*(-2*U[i-1,j] + U[i-2,j])/(tau**2) -
2*(h**2)*f(x[j], t[i])
            aa[j] = 2*a - h*b
            bb[j] = 2*(h**2)*(-d/(2*tau) - 1/(tau**2) + c )
- 4*a
            #bb[j] = 2*(h**2)*(-d/tau - 1/(tau**2) + c) -
4*a
            cc[j] = h*b + 2*a
            xx = prog(aa, bb, cc, dd)
        for j in range(len(x)):
            U[i, j] = xx[j]
if apr1 == 3:
    for i in range(2, len(t)):

```

```

aa = np.zeros(len(x))
bb = np.zeros(len(x))
cc = np.zeros(len(x))
dd = np.zeros(len(x))
dd[0] = 2*h*phi0(t[i])
dd[-1] = 2*h*phi1(t[i])
aa[-1] = -4*gama
bb[0] = 2*beta*h - 3*alfa
bb[-1] = delta*2*h + 3*gama
cc[0] = alfa*4
for j in range(1, len(x) - 1):
    dd[j] = -4*(h**2)*U[i-1,j]/(tau**2) +
(2*(h**2)/(tau**2) - d*(h**2)/tau)*U[i-2,j] -
2*(h**2)*f(x[j], t[i])
    aa[j] = 2*a - h*b
    bb[j] = 2*(h**2)*(-d/(2*tau) - 1/(tau**2) + c )
- 4*a
    cc[j] = h*b + 2*a
    const1 = alfa/(2*a + h*b)
    bb[0] = bb[0] + aa[1]*const1
    cc[0] = cc[0] + bb[1]*const1
    dd[0] = dd[0] + dd[1]*const1
    const2 = - gama/(2*a - h*b)
    aa[-1] = aa[-1] + bb[-2]*const2
    bb[-1] = bb[-1] + cc[-2]*const2
    dd[-1] = dd[-1] + dd[-2]*const2
    xx = prog(aa, bb, cc, dd)
    for j in range(len(x)):
        U[i, j] = xx[j]
return U

def plot_im(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, sigm, k):
    print('Неявная схема')
    x = np.arange(lb, ub + h, h)
    t = np.arange(0, T + tau, tau)
    plt.figure(1)
    plt.title('Неявная схема, t = ' + str(t[k]))
    plt.grid()
    plt.plot(x, true_fval(x, t[k]), color = 'red', label =
'аналитическое решение')
    colors = ['green', 'lime', 'pink', 'blue', 'orange', 'indigo']
    aprs = ['1π / 1π - 2τ', '1π / 2π - 2τ', '1π / 2π - 3τ', '2π / 1π
- 2τ', '2π / 2π - 2τ', '2π / 2π - 3τ']
    for apr0 in range(1, 3):
        for apr1 in range(1, 4):
            U = explicit(a, b, c, d, alfa, beta, gama, delta, lb,
ub, h, tau, T, apr0, apr1)
            plt.plot(x, U[k,:], color = colors[6 - (6//apr0) +
apr1 - 1], label = aprs[6 - (6//apr0) + apr1 - 1])
            plt.legend()
            if apr0 == 1 and apr1 == 1:
                A11 = U
            if apr0 == 1 and apr1 == 2:
                A12 = U
            if apr0 == 1 and apr1 == 3:
                A13 = U
            if apr0 == 2 and apr1 == 1:

```

```

        A21 = U
        if apr0 == 2 and apr1 == 2:
            A22 = U
        if apr0 == 2 and apr1 == 3:
            A23 = U
plt.figure(2)
plt.title('Погрешность')
plt.grid()
eps = []
for i in range(len(t)):
    a = true_fval(x, t[i]) - A11[i,:]
    eps = np.append(eps, norma(a))
plt.plot(t, eps, color = 'green', label = aprs[0])
"eps = []
for i in range(len(t)):
    a = true_fval(x, t[i]) - A12[i,:]
    eps = np.append(eps, norma(a))
plt.plot(t, eps, color = 'lime', label = aprs[1])
eps = []
eps = []
for i in range(len(t)):
    a = true_fval(x, t[i]) - A21[i,:]
    eps = np.append(eps, norma(a))
plt.plot(t, eps, color = 'blue', label = aprs[3])
plt.show()
return

root = Tk()
root.title("Лабораторная работа №6")
root["bg"] = "grey"
w = root.winfo_screenwidth() # ширина экрана
h = root.winfo_screenheight() # высота экрана
ww = str(int(w/2))
hh = str(int(h-150))
a = ww + 'x' + hh
w = w//2 # середина экрана
h = h//2
w = w - 200 # смещение от середины
h = h - 200
root.geometry(a + '+0+0'.format(w, h))

label4 = Label(text = "Параметры задачи:", justify =
LEFT, font = "Arial 12")
label4.place(x = 10, y = 10)
labela = Label(text = "a = ", justify = LEFT, font = "Arial
12", bg = "grey")
labela.place(x = 10, y = 40)
labelb = Label(text = "b = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labelb.place(x = 10, y = 70)
labelc = Label(text = "c = ", justify = LEFT, font = "Arial
12", bg = "grey")
labelc.place(x = 10, y = 100)
labeld = Label(text = "d = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labeld.place(x = 10, y = 130)

```

```

label1 = Label(text = "l = ", justify = LEFT, font = "Arial
12", bg = "grey")
label1.place(x = 10, y = 160)

entrya = Entry(root, justify = RIGHT)
entrya.place(x = 30, y = 40)

entryb = Entry(root, justify = RIGHT)
entryb.place(x = 30, y = 70)

entryc = Entry(root, justify = RIGHT)
entryc.place(x = 30, y = 100)

entryd = Entry(root, justify = RIGHT)
entryd.place(x = 30, y = 130)

entryl = Entry(root, justify = RIGHT)
entryl.place(x = 30, y = 160)

entrya.insert(0, 1)
entryb.insert(0, 1)
entryc.insert(0, -1)
entryd.insert(0, 3)
entryl.insert(0, np.pi)

label5 = Label(text = "Параметры конечно-разностной
сетки:", justify = LEFT, font = "Arial 12")
label5.place(x = 300, y = 10)
labeln = Label(text = "n = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labeln.place(x = 300, y = 40)
labelsigm = Label(text = "σ = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labelsigm.place(x = 300, y = 70)
labelT = Label(text = "T = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labelT.place(x = 300, y = 100)

entryn = Entry(root, justify = RIGHT)
entryn.place(x = 330, y = 40)

entrysigm = Entry(root, justify = RIGHT)
entrysigm.place(x = 330, y = 70)

entryT = Entry(root, justify = RIGHT)
entryT.place(x = 330, y = 100)

entryn.insert(0, 40)
entrysigm.insert(0, 0.1)
entryT.insert(0, 2)

labelh = Label(text = " ", justify = LEFT, font = "Arial
12", bg = "grey")
labelh.place(x = 650, y = 40)
labeltau = Label(text = " ", justify = LEFT, font = "Arial
12", bg = "grey")
labeltau.place(x = 650, y = 70)

```

```

#labelerror = Label(text = " ", justify = LEFT, font =
"Arial 12", bg = "grey")
#labelerror.place(x = 650, y = 100)
labelK = Label(text = " ", justify = LEFT, font = "Arial
12", bg = "grey")
labelK.place(x = 650, y = 100)

def params():
    try:
        lb = 0
        ub = float(entryl.get())
        a = float(entrya.get())
        T = float(entryT.get())
        n = float(entryn.get())
        sigm = float(entrysigm.get())
        labelh.config(text = "h = {}".format((ub - lb)/n))
        h = (ub - lb)/n
        labeltau.config(text = "τ = {}".format(sigm*h*h/a))
        labelK.config(text = "K =
{}".format(int(T*a/(sigm*h*h))))
    except ValueError:
        labelerror.config(text = "Заполните все поля", fg =
"red")

but = Button(root, text = "Показать все параметры
сетки", command = params, font = "Arial 9")
but.place(x = 650, y = 10)

label6 = Label(text = "Параметр схемы:", justify =
LEFT, font = "Arial 12")
label6.place(x = 10, y = 250)
labelteta = Label(text = "Θ = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labelteta.place(x = 10, y = 280)
entryteta = Entry(root, justify = RIGHT)
entryteta.place(x = 50, y = 280)
entryteta.insert(0, 0)

label7 = Label(text = "Отобразить решение на шаге по
времени:", justify = LEFT, font = "Arial 12")
label7.place(x = 300, y = 250)
labelk = Label(text = "k = ", justify = LEFT, font =
"Arial 12", bg = "grey")
labelk.place(x = 300, y = 280)
entryk = Entry(root, justify = RIGHT)
entryk.place(x = 340, y = 280)

a = float(entrya.get())
ub = float(entryl.get())
lb = 0
T = float(entryT.get())
sigm = float(entrysigm.get())
n = float(entryn.get())
h = (ub - lb)/n
tau = sigm*h*h/a
K = int(T/tau)

```

```

entryk.insert(0, int(K//650))

```

```

def solver(a, b, c, d, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, sigm, teta, k):
    if teta == 0:
        plot_ex(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, sigm, k)
    if teta == 1:
        plot_im(a, b, c, d, alfa, beta, gama, delta, lb, ub, h,
tau, T, sigm, k)
    return

plt.figure()
plt.grid()
plt.scatter(a,b)
plt.show()
return

```

```

def solvv():
    try:
        teta = float(entryteta.get())
        a = float(entrya.get())
        b = float(entryb.get())
        c = float(entryc.get())
        d = float(entryd.get())
        ub = float(entryl.get())
        lb = 0
        T = float(entryT.get())
        sigm = float(entrysigm.get())
        n = float(entryn.get())
        h = (ub - lb)/n
        tau = sigm*h*h/a
        K = T/tau
        k = int(entryk.get())
        alfa = 1
        beta = 0
        gama = 1
        delta = 0
        solver(a, b, c, d, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, sigm, teta, k)
    except ValueError:
        labelerror.config(text = "Заполните все поля", fg =
"red")

```

```

solv = Button(root, text = " Решить ", font = "Arial 14",
command = solvv)
solv.place(x = 650, y = 310)

root.mainloop()

```


Скриншоты выполнения

Лабораторная работа №6

Параметры задачи:

a = 1
b = 1
c = -1
d = 3
l = 3.141592653589793

Параметры конечно-разностной сетки:

n = 5
 σ = 0.1
T = 1

Показать все параметры сетки

h = 0.6283185307179586
 τ = 0.039478417604357434
K = 25

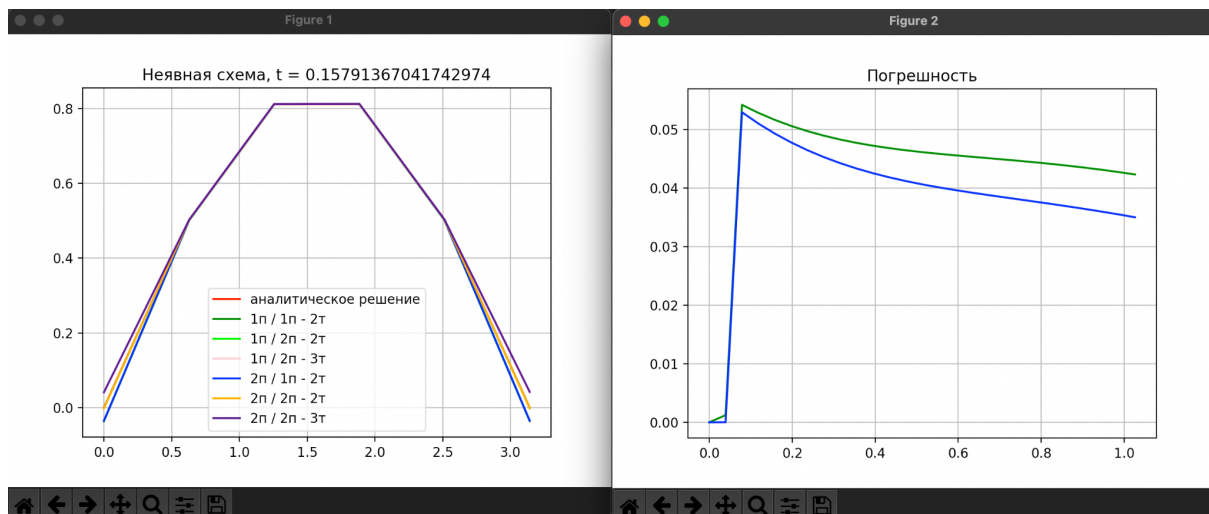
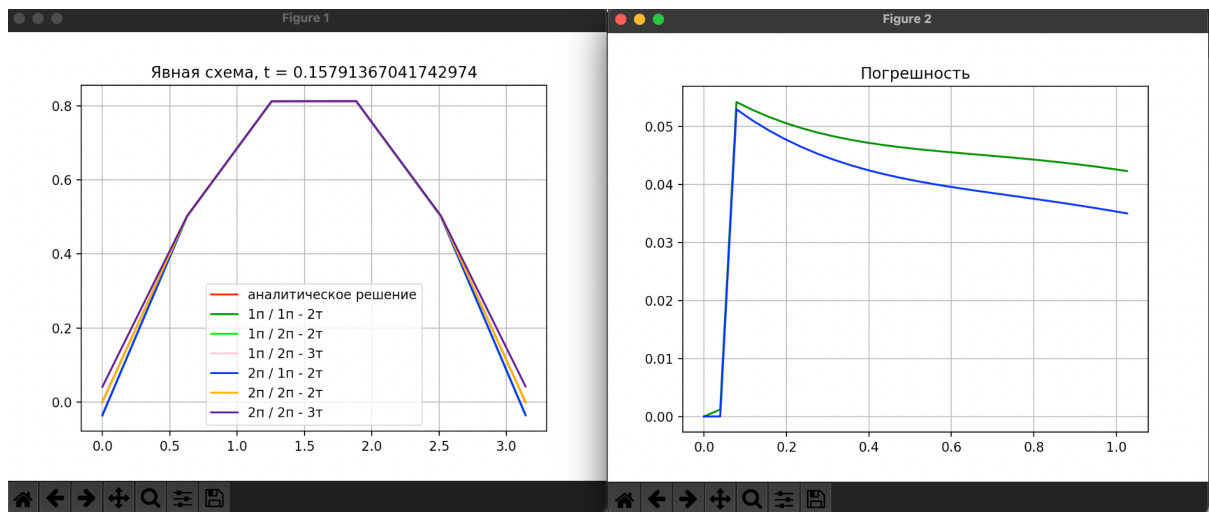
Параметр схемы:

Θ = 0

Отобразить решение на шаге по времени:

k = 4

Решить



Лабораторная работа №6

Параметры задачи:

a =

b =

c =

d =

l =

Параметры конечно-разностной сетки:

n =

σ =

T =

Показать все параметры сетки

h = 1.5707963267948966

τ = 0.24674011002723395

K = 4

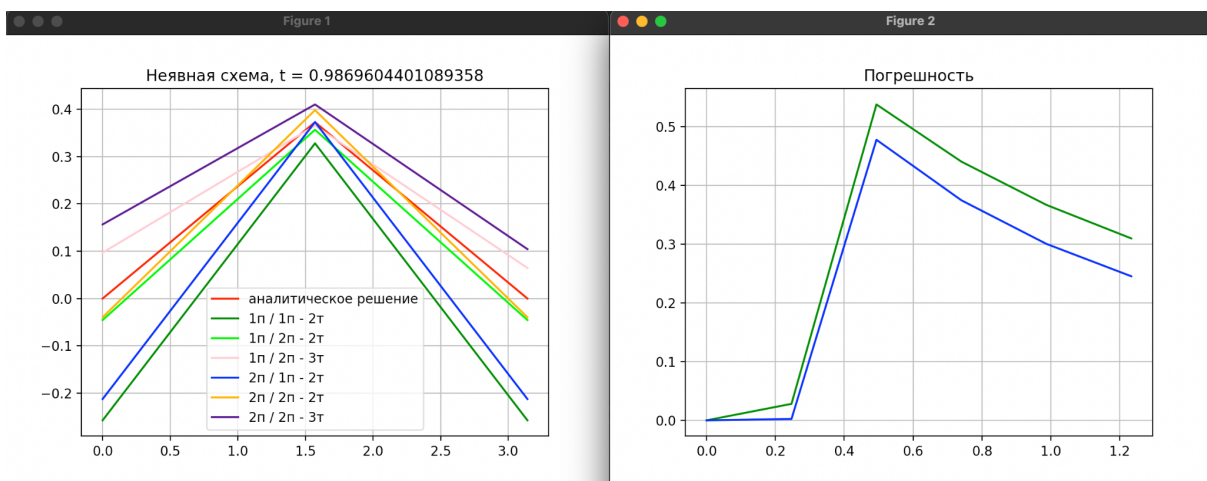
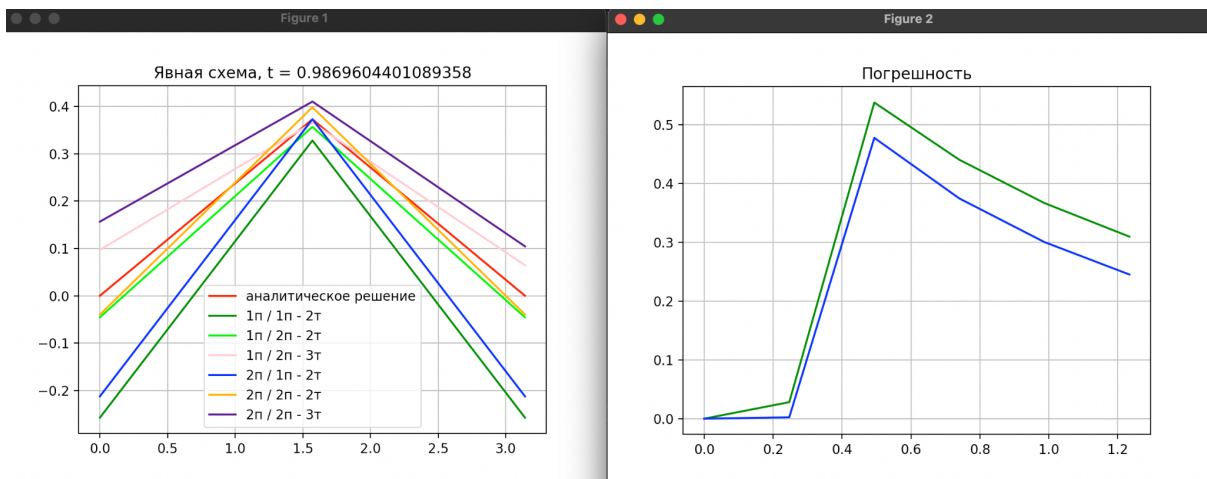
Параметр схемы:

Θ =

Отобразить решение на шаге по времени:

k =

Решить



Лабораторная работа №6

Параметры задачи:

a =

b =

c =

d =

l =

Параметры конечно-разностной сетки:

n =

σ =

T =

Показать все параметры сетки

h = 0.3141592653589793

τ = 0.009869604401089358

K = 101

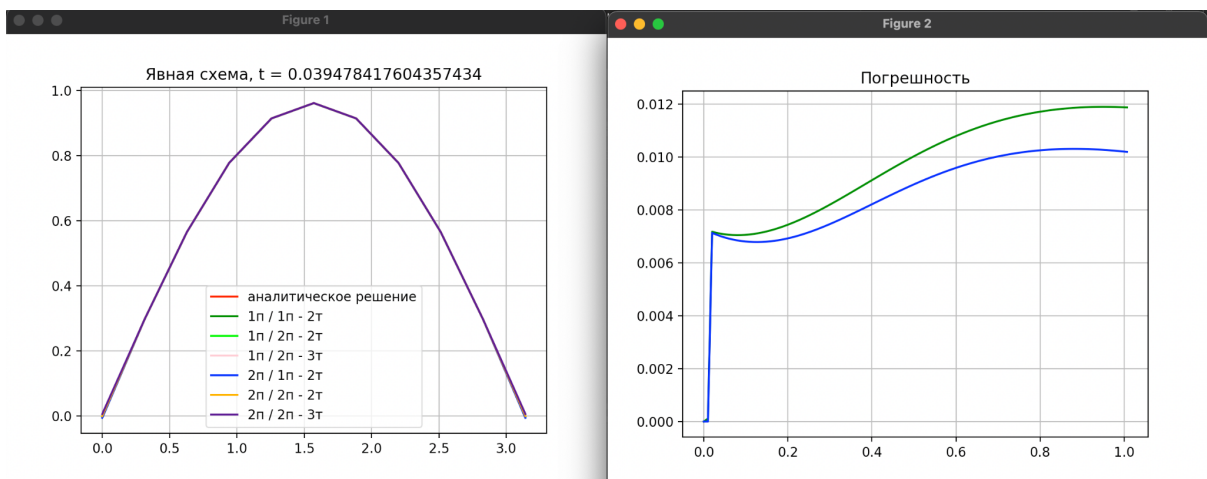
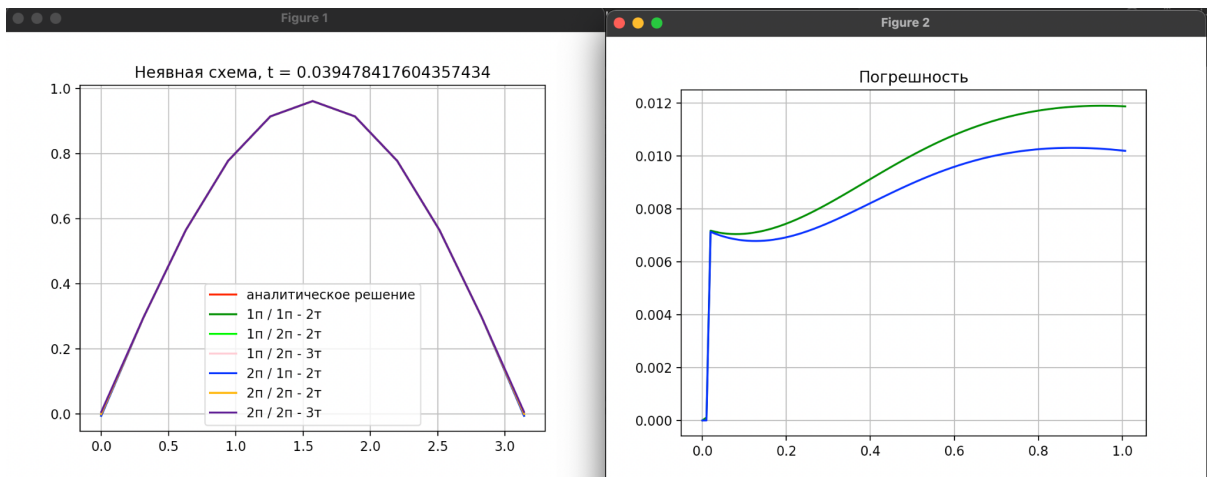
Параметр схемы:

Θ =

Отобразить решение на шаге по времени:

k =

Решить



Выводы

Классическим примером уравнения гиперболического типа является волновое уравнение, которое в области $0 < x < l$, $t > 0$ имеет вид:

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

Данное уравнение описывает, в частности, процесс малых поперечных колебаний струны. В этом случае $u(x, t)$ - поперечные перемещения (колебания) струны, a – скорость распространения малых возмущений в материале, из которого изготовлена струна.