Московский Авиационный Институт (Национальный Исследовательский Университет)

Факультет: “Информационные технологии и прикладная математика”
Кафедра: 806 “Вычислительная математика и программирование”

Отчет по лабораторной работе №5 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 1

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: 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

$$\frac{\partial u}{\partial t} = a\frac{\partial^2 u}{\partial x^2} + b\frac{\partial u}{\partial x} + cu, \quad a > 0, \quad b > 0, \quad c < 0.$$

$$u_x(0,t) + u(0,t) = \exp((c-a)t)(\cos(bt) + \sin(bt)),$$

$$u_x(\pi,t) + u(\pi,t) = -\exp((c-a)t)(\cos(bt) + \sin(bt)),$$

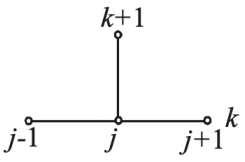
$$u(x,0) = \sin x.$$

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

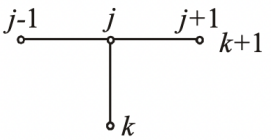
Описание программы

Выбор схемы осуществляется с помощью параметра схемы. Если он равен 0, отрисовка явной схемы. Если он равен 1, неявная схема.
Если 0.5, схема Кранка-Николсона.

Шаблоном конечно-разностной схемы называют ее геометрическую интерпретацию



шаблон явной схемы



шаблон неявной схемы

Рассмотрим неявно-явную схему с весами для простейшего уравнения теплопроводности

$$\frac{u_j^{k+1}-u_j^k}{\tau}=\theta a^2\frac{u_{j+1}^{k+1}-2u_j^{k+1}+u_{j-1}^{k+1}}{h^2}+(1-\theta)a^2\frac{u_{j+1}^{k}-2u_j^{k}+u_{j-1}^{k}}{h^2}, \qquad (5.20)$$

где θ - вес неявной части конечно-разностной схемы, 1–θ - вес для явной части, причем $0\le\theta\le1$. При θ=1 имеем полностью неявную схему, при θ=0 - полностью явную схему, и при θ=1/2 - схему *Кранка-Николсона*.

Для схемы Кранка-Николсона (θ=1/2) порядок аппроксимации составляет $O(\tau^2+h^2)$, т.е. на один порядок по времени выше, чем обычные явная или неявная схемы.

В функции **plot_ex()** идёт отрисовка явной схемы двух графиков, в **plot_im()** отрисовка неявной схемы, **plot_exim()** схема Кранка-Николсона:
1) Аналитическое и численные решения с 3 типами аппроксимации.
2) Погрешность численных решений с 3 типами аппроксимации.

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.
Функция **eximplicit()** - численное решение схемы Кранка-Николсона, 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 psi(x):
    return  np.sin(x)

def phi0(t, a, b, c):
    return np.exp((c - a)*t) * (np.cos(b*t) + np.sin(b*t))

def phi1(t, a, b, c):
    return - np.exp((c - a)*t) * (np.cos(b*t) + np.sin(b*t))
```

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

def f(x,t):
    return 0

def explicit(a, b, c, alfa, beta, gama, delta, lb, ub, n, K, h,
tau, T, apr, sigm):
    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] = psi(x[j])
    for i in range(1, len(t)):
        for j in range(1, len(x) - 1):
            U[i,j] = U[i-1,j] + (a*tau/h**2)*(U[i-1,j-1] -
2*U[i-1,j] + U[i-1,j+1]) + (b*tau*0.5/h)*(- U[i-1,j-1] +
U[i-1,j+1]) + c*tau*U[i-1,j] + tau*f(x[j],t[i-1])
            #U[i,j] = (a*tau/(h**2) - b*tau/(2*h))*U[i-1,j-1]
+ (tau*c + 1 - 2*a*tau/(h**2))*U[i-1,j] + (a*tau/(h**2) +
b*tau/(2*h))*U[i-1,j+1] + tau*f(x[j],t[i-1])
        if apr == 1:
            U[i,0] = (h * phi0(t[i],a,b,c) - alfa * U[i,1])/(h *
beta - alfa)
            U[i,-1] = (h * phi1(t[i],a,b,c) + gama * U[i, -2]
)/(h * delta + gama)
        elif apr ==2:
            U[i,0] = (h*(2*a - b*h)*phi0(t[i],a,b,c) -
2*alfa*a*U[i,1] - alfa*(h**2)*U[i-1,0]/tau
)/(alfa*c*(h**2) + beta*h*(2*a - b*h) - 2*alfa*a -
alfa*(h**2)/tau)
            U[i,-1] = (h*(2*a + b*h)*phi1(t[i],a,b,c) +
2*gama*a*U[i,-2] + gama*(h**2)*U[i-1,-1]/tau
)/(2*gama*a + gama*(h**2)/tau - c*gama*(h**2) +
delta*h*(2*a + b*h))
        elif apr == 3:
            U[i,0] = (2*h*phi0(t[i],a,b,c) - 4*alfa*U[i,1] +
alfa*U[i,2])/(2*h*beta - 3*alfa)
            U[i,-1] = (2*h*phi1(t[i],a,b,c) + 4*gama*U[i,-2] -
gama*U[i,-3])/(2*h*delta + 3*gama)
    return U

def plot_ex(a, b, c, alfa, beta, gama, delta, lb, ub, n, K, 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],a,b,c), color = 'red', label =
'аналитическое решение')
    colors = ['green', 'orange', 'blue']
    aprs = ['1п - 2т', '2п - 2т', '2п - 3т'] #апроксимации
    for j in range(len(aprs)):
        U = explicit(a, b, c, alfa, beta, gama, delta, lb, ub, n,
K, h, tau, T, j+1, sigm)
        plt.plot(x, U[k,:], color = colors[j], label = aprs[j])
        plt.legend()
        plt.xlim((0,ub))
        if j == 0:
            A1 = U
        if j == 1:
            A2 = U
        if j == 2:
            A3 = U
    plt.figure(2)
    plt.title('Погрешность')
    plt.grid()
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A1[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],a,b,c) - A2[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'orange', label = aprs[1])

    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A3[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'blue', label = aprs[2])
    plt.legend()
    plt.show()
    return

def implicit(a, b, c, alfa, beta, gama, delta, lb, ub, n, K, h,
tau, T, apr, sigm):
    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] = psi(x[j])
    if apr == 1:
        for i in range(1, 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],a,b,c)*h
            dd[-1] = phi1(t[i],a,b,c)*h
            bb[0] = h*beta - alfa
            bb[-1] = h*delta + gama
            cc[0] = alfa
            aa[-1] = - gama
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*tau*0.5/(h**2)
                bb[j] = c*tau - 2*a*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*tau*0.5/(h**2)
                dd[j] = -U[i-1, j]
            xx = prog(aa, bb, cc, dd)
            for j in range(len(x)):
```

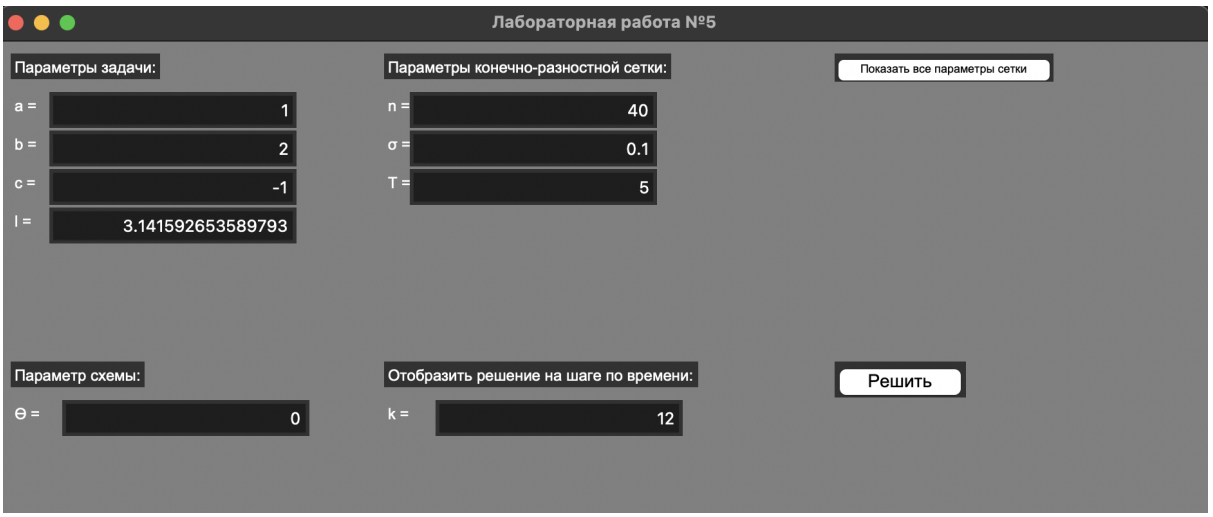
```
            U[i, j] = xx[j]
    if apr == 2:
        for i in range(1, len(t)):
            aa = np.zeros(len(x))
            bb = np.zeros(len(x))
            cc = np.zeros(len(x))
            dd = np.zeros(len(x))
            dd[0] = h*U[i-1,0]/tau -  phi0(t[i],a,b,c)*(2*a -
b*h)/alfa
            dd[-1] = h*U[i-1,-1]/tau + phi1(t[i],a,b,c)*(2*a +
b*h)/gama
            bb[0] = 2*a/h + h/tau - c*h - beta*(2*a - b*h)/alfa
            bb[-1] = 2*a/h + h/tau - c*h + delta*(2*a +
b*h)/gama
            cc[0] = - 2 * a/h
            aa[-1] = - 2 * a / h
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*tau*0.5/(h**2)
                bb[j] = c*tau - 2*a*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*tau*0.5/(h**2)
                dd[j] = -U[i-1, j]
            xx = prog(aa, bb, cc, dd)
            for j in range(len(x)):
                U[i, j] = xx[j]
    if apr == 3:
        for i in range(1, 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],a,b,c)
            dd[-1] = 2*h*phi1(t[i],a,b,c)
            #bb[0] = 2*a/h + h/tau - c*h - beta*(2*a -
b*h)/alfa
            #bb[-1] = 2*a/h + h/tau - c*h + delta*(2*a +
b*h)/gama
            #cc[0] = - 2 * a/h
            #aa[-1] = - 2 * a / h
            bb[0] = (2*beta*h - 3*alfa)
            cc[0] = 4*alfa
            bb[-1] = 2*h*delta + 3*gama
            aa[-1] = - 4*gama
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*tau*0.5/(h**2)
                bb[j] = c*tau - 2*a*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*tau*0.5/(h**2)
                dd[j] = -U[i-1, j]
            const1 = alfa*2*(h**2)/(tau*(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*(h**2)/(tau*(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, alfa, beta, gama, delta, lb, ub, n, K, h,
tau, T, sigm, k):
    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],a,b,c), color = 'red', label =
'аналитическое решение')
    colors = ['green', 'orange', 'blue']
    aprs = ['1п - 2т', '2п - 2т', '2п - 3т']
    for j in range(len(aprs)):
        U = implicit(a, b, c, alfa, beta, gama, delta, lb, ub, n,
K, h, tau, T, j+1, sigm)
        plt.plot(x, U[k,:], color = colors[j], label = aprs[j])
        plt.legend()
        plt.xlim((0,ub))
        if j == 0:
            A1 = U
        if j == 1:
            A2 = U
        if j == 2:
            A3 = U
    plt.figure(2)
    plt.title('Погрешность')
    plt.grid()
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A1[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],a,b,c) - A2[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'orange', label = aprs[1])

    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A3[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'blue', label = aprs[2])
    plt.legend()
    plt.show()
    return


def eximplicit(a, b, c, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, apr, sigm, teta):
    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] = psi(x[j])
    if apr == 1:
```

```
        for i in range(1, 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],a,b,c)*h
            dd[-1] = phi1(t[i],a,b,c)*h
            bb[0] = h*beta - alfa
            bb[-1] = h*delta + gama
            cc[0] = alfa
            aa[-1] = - gama
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*teta*tau*0.5/(h**2)
                bb[j] = c*tau*teta - 2*a*teta*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*tau*teta*0.5/(h**2)
                dd[j] = -U[i-1, j] - (1 -
teta)*((a*tau/h**2)*(U[i-1,j-1] - 2*U[i-1,j] + U[i-1,j+1])
+ (b*tau*0.5/h)*(- U[i-1,j-1] + U[i-1,j+1]) +
c*tau*U[i-1,j])
            xx = prog(aa, bb, cc, dd)
            for j in range(len(x)):
                U[i, j] = xx[j]
    if apr == 2:
        for i in range(1, 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*U[i-1,0]/tau -  phi0(t[i],a,b,c)*(2*a -
b*h)/alfa
            dd[-1] = h*U[i-1,-1]/tau + phi1(t[i],a,b,c)*(2*a +
b*h)/gama
            bb[0] = 2*a/h + h/tau - c*h - beta*(2*a - b*h)/alfa
            bb[-1] = 2*a/h + h/tau - c*h + delta*(2*a +
b*h)/gama
            cc[0] = - 2 * a/h
            aa[-1] = - 2 * a / h
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*teta*tau*0.5/(h**2)
                bb[j] = c*tau*teta - teta*2*a*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*teta*tau*0.5/(h**2)
                dd[j] = -U[i-1, j] - (1 -
teta)*((a*tau/h**2)*(U[i-1,j-1] - 2*U[i-1,j] + U[i-1,j+1])
+ (b*tau*0.5/h)*(- U[i-1,j-1] + U[i-1,j+1]) +
c*tau*U[i-1,j])
            xx = prog(aa, bb, cc, dd)
            for j in range(len(x)):
                U[i, j] = xx[j]
    if apr == 3:
        for i in range(1, 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],a,b,c)
            dd[-1] = 2*h*phi1(t[i],a,b,c)
            bb[0] = (2*beta*h - 3*alfa)
            cc[0] = 4*alfa
            bb[-1] = 2*h*delta + 3*gama
            aa[-1] = - 4*gama
            for j in range(1,len(x)-1):
                aa[j] = (2*a - b*h)*teta*tau*0.5/(h**2)
                bb[j] = c*tau*teta - 2*a*teta*tau/(h**2) - 1
                cc[j] = (2*a + h*b)*tau*teta*0.5/(h**2)
                dd[j] = -U[i-1, j] - (1 -
teta)*((a*tau/h**2)*(U[i-1,j-1] - 2*U[i-1,j] + U[i-1,j+1])
+ (b*tau*0.5/h)*(- U[i-1,j-1] + U[i-1,j+1]) +
c*tau*U[i-1,j])
            const1 = alfa*2*(h**2)/(tau*(2*a + h*b)*teta)
            bb[0] = bb[0] + aa[1]*const1
            cc[0] = cc[0] + bb[1]*const1
            dd[0] = dd[0] + dd[1]*const1
            const2 = - gama*2*(h**2)/(tau*(2*a - h*b)*teta)
            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_exim(a, b, c, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, sigm, teta, k):
    x = np.arange(lb, ub + h, h)
    t = np.arange(0, T + tau, tau)
    plt.figure(1)
    if teta == 0.5:
        plt.title('Схема Кранка - Николсона, t = ' +
str(t[k]))
    else:
        plt.title('Явно-неявная схема, t = ' + str(t[k]))
    plt.grid()
    plt.plot(x, true_fval(x, t[k],a,b,c), color = 'red', label =
'аналитическое решение')
    colors = ['green', 'orange', 'blue']
    aprs = ['1п - 2т', '2п - 2т', '2п - 3т']
    epsil = []
    for j in range(len(aprs)):
        U = eximplicit(a, b, c, alfa, beta, gama, delta, lb, ub,
n, K, h, tau, T, j+1, sigm, teta)
        plt.plot(x, U[k,:], color = colors[j], label = aprs[j])
        plt.legend()
        plt.xlim((0,ub))
        if j == 0:
            A1 = U
        if j == 1:
            A2 = U
        if j == 2:
            A3 = U
    plt.figure(2)
    plt.title('Погрешность')
    plt.grid()
    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A1[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],a,b,c) - A2[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'orange', label = aprs[1])

    eps = []
    for i in range(len(t)):
        a = true_fval(x, t[i],a,b,c) - A3[i,:]
        eps = np.append(eps, norma(a))
    plt.plot(t, eps, color = 'blue', label = aprs[2])
    plt.legend()
    plt.show()
    return

root = Tk()
root.title("Лабораторная работа №5")
root["bg"] = "grey"
w = root.winfo_screenwidth() # ширина экрана
h = root.winfo_screenheight() # высота экрана
ww = str(int(w/2))
hh = str(int(h-250))
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)
labell = Label(text = "l = ", justify = LEFT, font = "Arial
12", bg = "grey")
labell.place(x = 10, y = 130)

entrya = Entry(root, justify = RIGHT)
entrya.place(x = 40, y = 40)
entryb = Entry(root, justify = RIGHT)
entryb.place(x = 40, y = 70)

entryc = Entry(root, justify = RIGHT)
entryc.place(x = 40, y = 100)

entryl = Entry(root, justify = RIGHT)
entryl.place(x = 40, y = 130)

entrya.insert(0, 1)
entryb.insert(0, 2)
entryc.insert(0, -1)
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 = 320, y = 40)

entrysigm = Entry(root, justify = RIGHT)
entrysigm.place(x = 320, y = 70)

entryT = Entry(root, justify = RIGHT)
entryT.place(x = 320, y = 100)

entryn.insert(0, 40)
entrysigm.insert(0, 0.1)
entryT.insert(0, 5)

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, alfa, beta, gama, delta, lb, ub, n, K, h,
tau, T, sigm, teta, k):
    if teta == 0:
        plot_ex(a, b, c, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, sigm, k)
    if teta == 1:
        plot_im(a, b, c, alfa, beta, gama, delta, lb, ub, n, K,
h, tau, T, sigm, k)
    if teta > 0  and teta < 1:
        plot_exim(a, b, c, alfa, beta, gama, delta, lb, ub, n,
K, h, tau, T, sigm, teta, 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())
        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 = 1
        gama = 1
        delta = 1
        solver(a, b, c, 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 = 250)

root.mainloop()
```

Скриншоты выполнения


Лабораторная работа №5
Параметры задачи:
a =
1
b =
2
c =
-1
l =
3.141592653589793
Параметры конечно-разностной сетки:
n =
40
σ =
0.1
T =
5
Показать все параметры сетки
Параметр схемы:
Θ =
0
Отобразить решение на шаге по времени:
k =
12
Решить

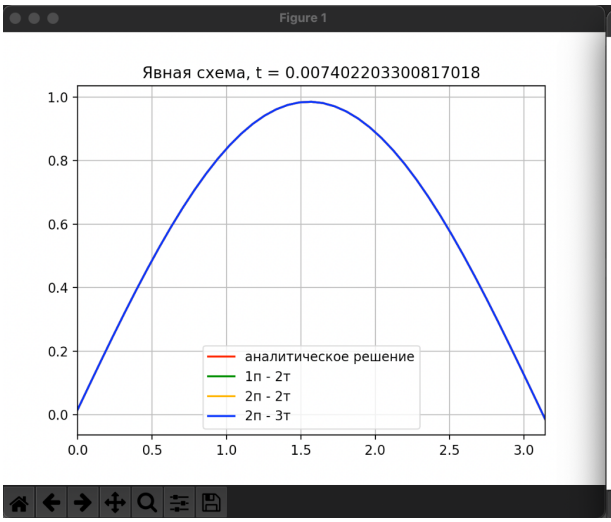


Figure 1
Явная схема, t = 0.007402203300817018
аналитическое решение
1п - 2т
2п - 2т
2п - 3т

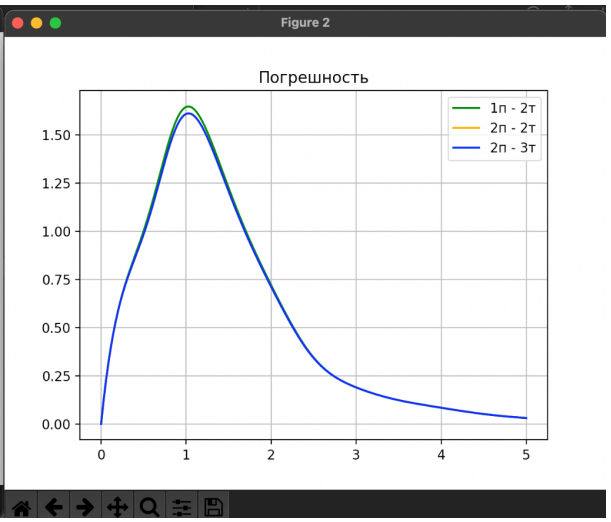


Figure 2
Погрешность
1п - 2т
2п - 2т
2п - 3т

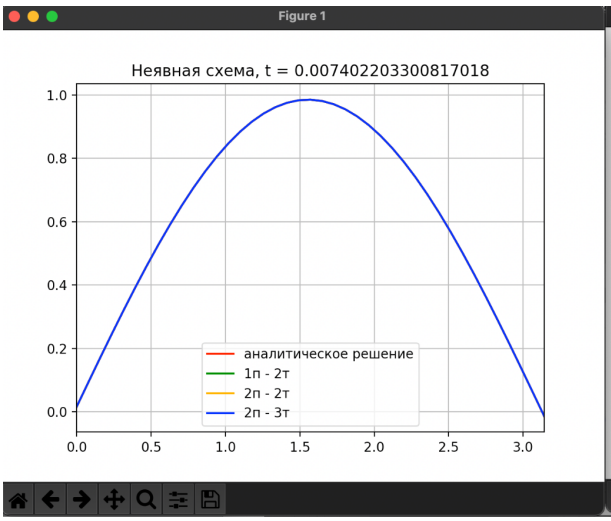


Figure 1
Неявная схема, t = 0.007402203300817018
аналитическое решение
1п - 2т
2п - 2т
2п - 3т

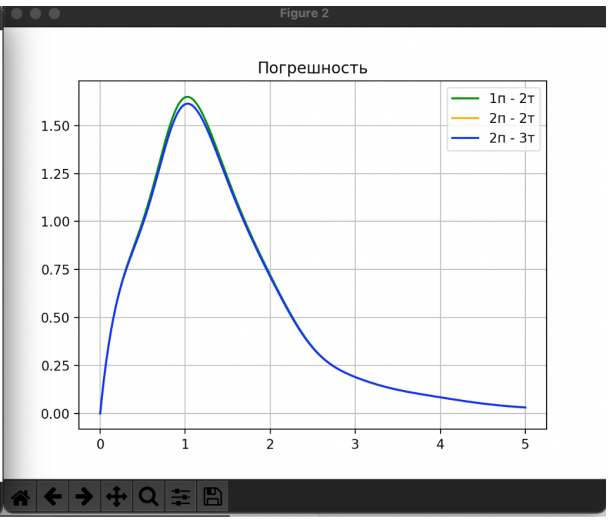
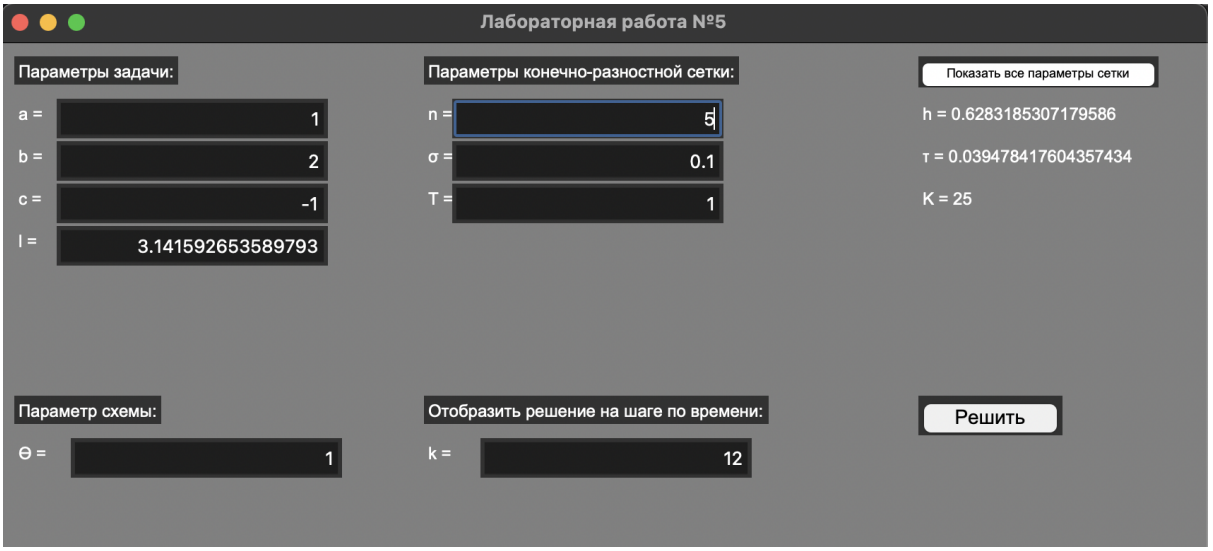


Figure 2
Погрешность
1п - 2т
2п - 2т
2п - 3т


Лабораторная работа №5
Параметры задачи:
a = 1
b = 2
c = -1
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 = 1
Отобразить решение на шаге по времени:
k = 12
Решить

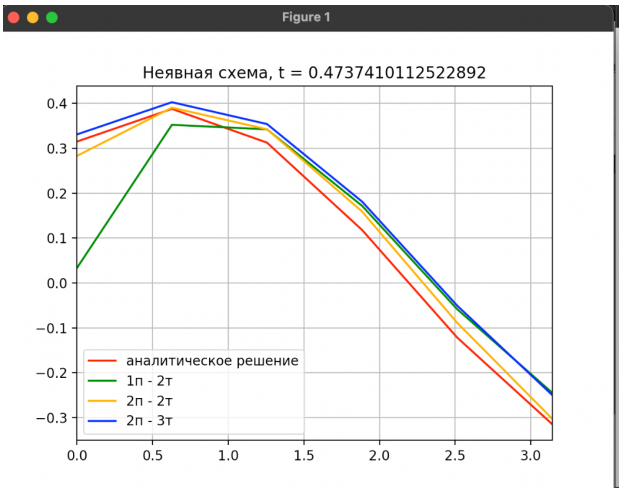


Figure 1
Неявная схема, t = 0.4737410112522892
аналитическое решение
1п - 2т
2п - 2т
2п - 3т

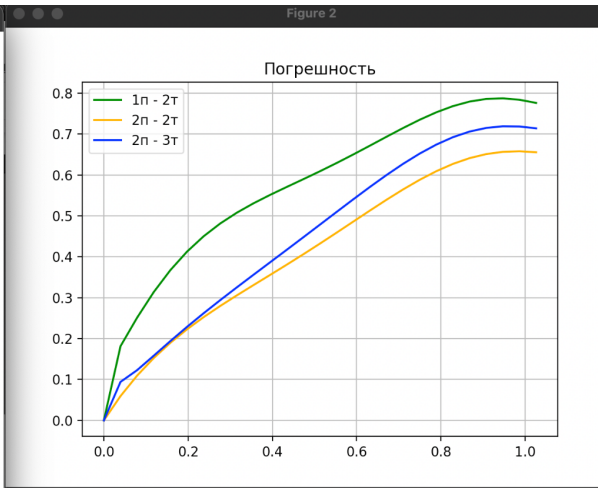


Figure 2
Погрешность
1п - 2т
2п - 2т
2п - 3т

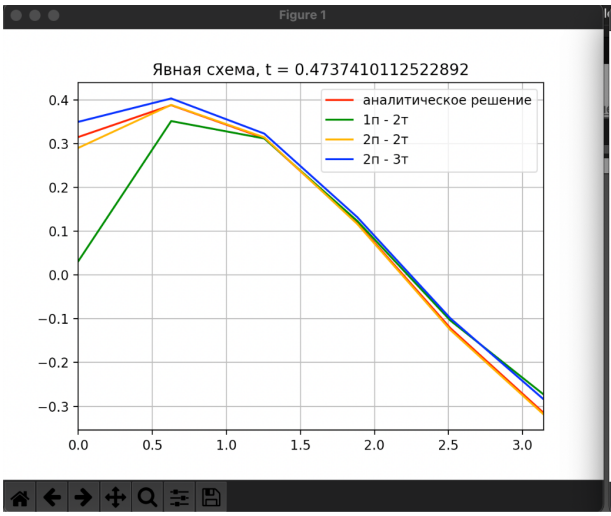


Figure 1
Явная схема, t = 0.4737410112522892
аналитическое решение
1п - 2т
2п - 2т
2п - 3т

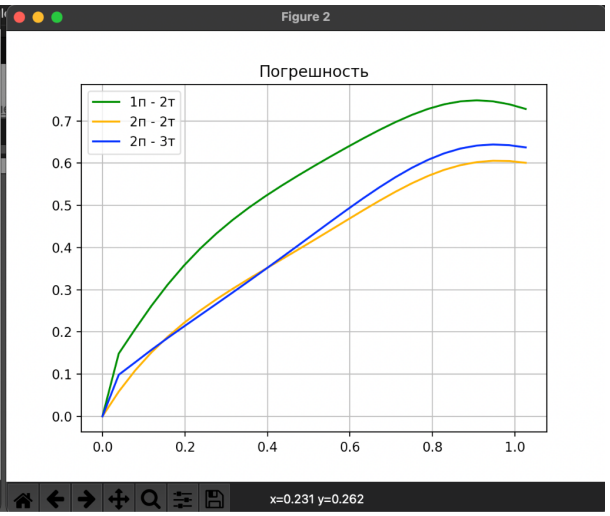
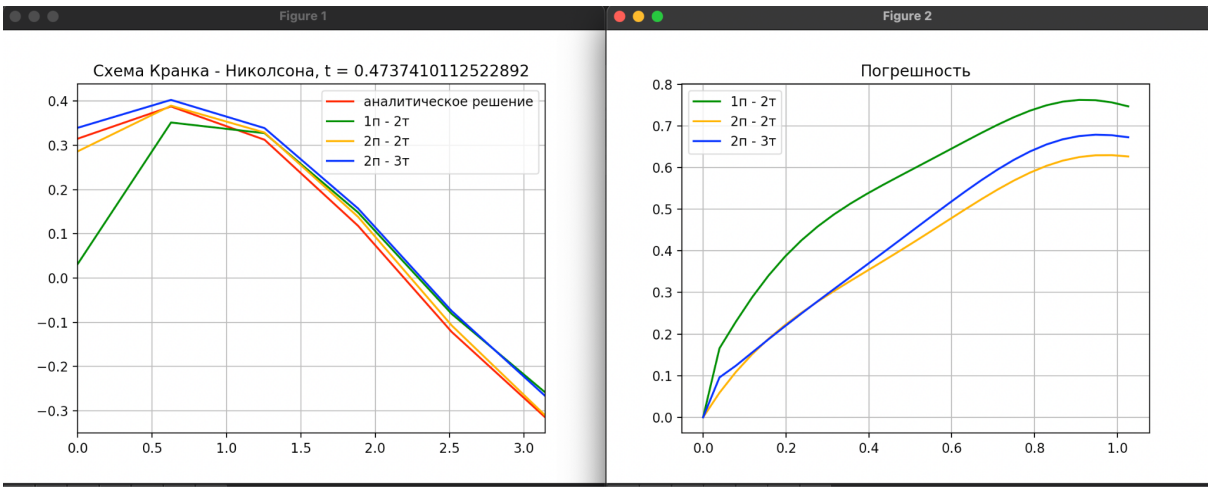


Figure 2
Погрешность
1п - 2т
2п - 2т
2п - 3т
x=0.231 y=0.262



Выводы

Погрешность численных решений уравнений параболического типа достаточно низкая, метод надежный.

В задачах математической физики вообще, и в задачах теплопроводности в частности, 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 вариантом является аппроксимация производных направленными разностями первого порядка.