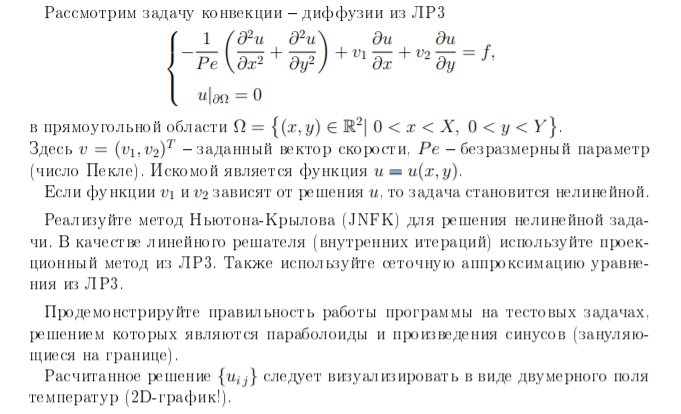
1. Постановка задачи



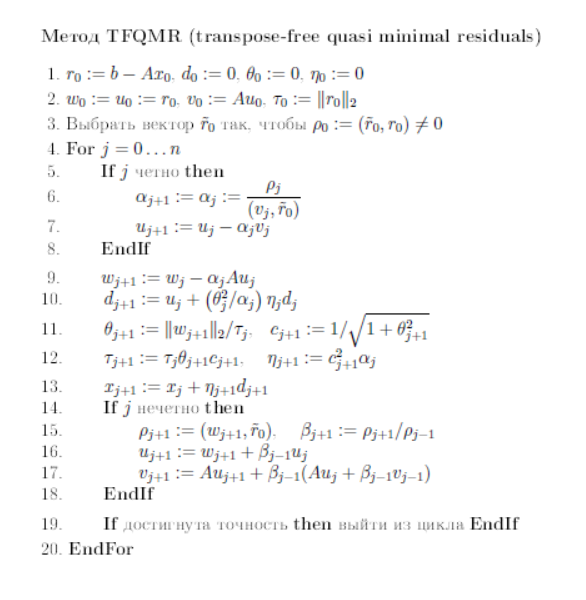
Мой вариант:

V1(u,x,y) = -3 - y

V2(u,x,y) = 21-u

1. Алгоритм метода TFQMR

Реализуется данный проекционный метод решения СЛАУ



1. Метод Ньютона-Крылова

Метод Ньютона имеет следующий вид

Получаем алгоритм:

В данной задаче реализуется метод Ньютона-Крылова, во внутренних итерациях которого решатся задача

проекционным методом, а во внешних - переходим к новому приближению

В проекционных методах решения СЛАУ основной операцией является умножение матрицы **A** на произвольный вектор **p.**

Произведём замену на где

1. Тесты

Тест 1

u(x,y) = x2 + y2

v1(u,x,y) = -3 - y

v2(u,x,y) = 2\*\*(1-u)

Pe = 1

hx = 0.1

hy = 0.1

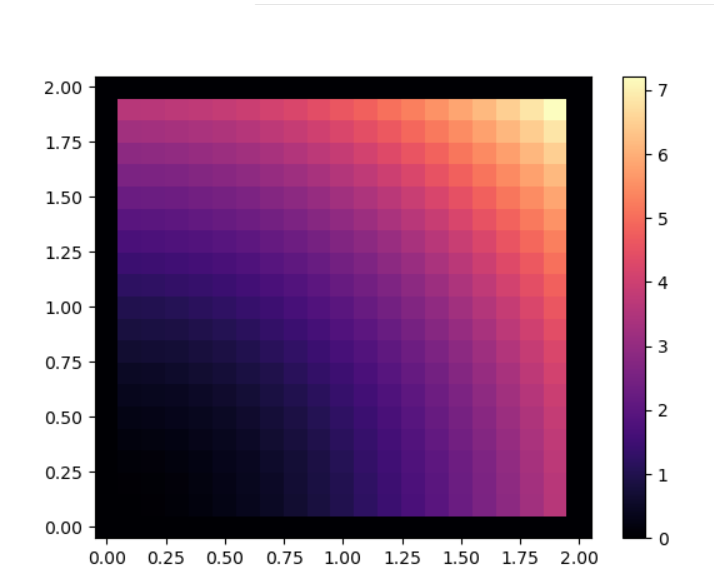
eps = 0.001

Nx = 20

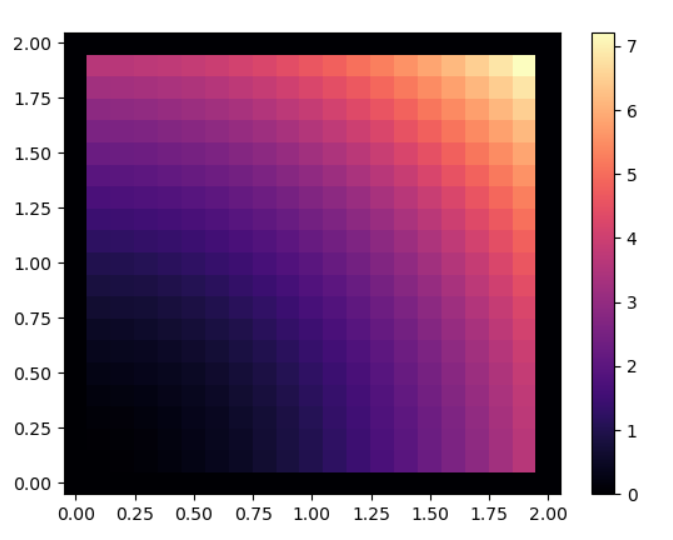
Ny = 20

epsN = 0.0001

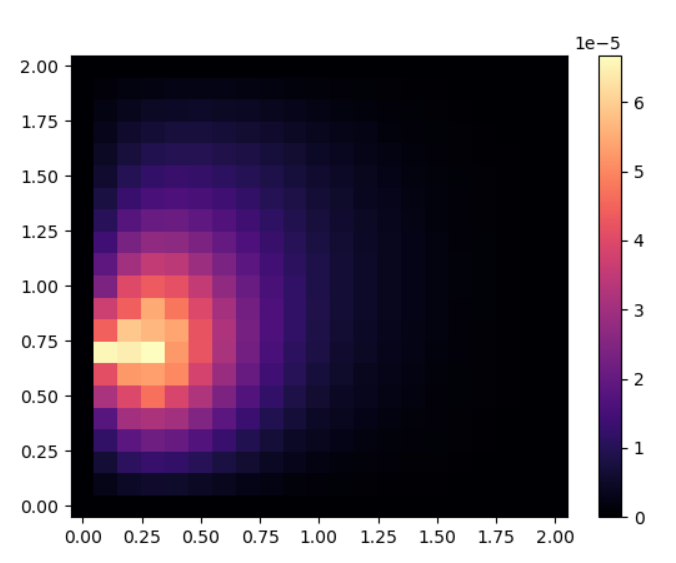
Найденное решение:



Точное решение:



Погрешности:



Затраченное время: 223.31

Итераций метода Ньютона: 26

Погрешность: 0.000318

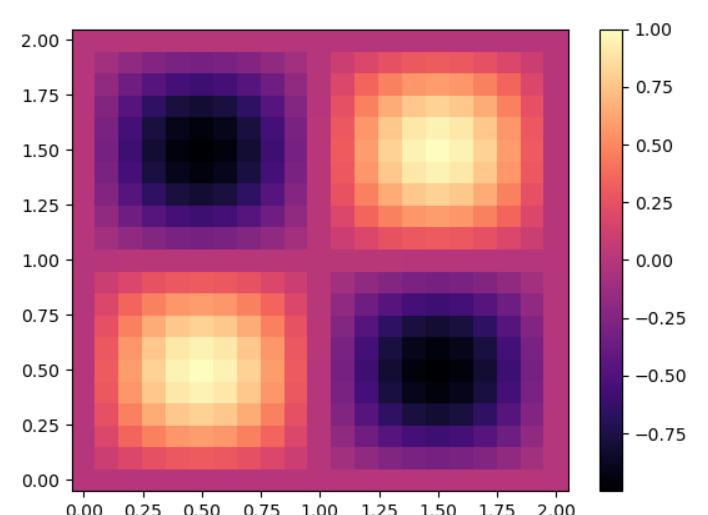
Тест 2

u(x,y) = sin(x) + sin(y)

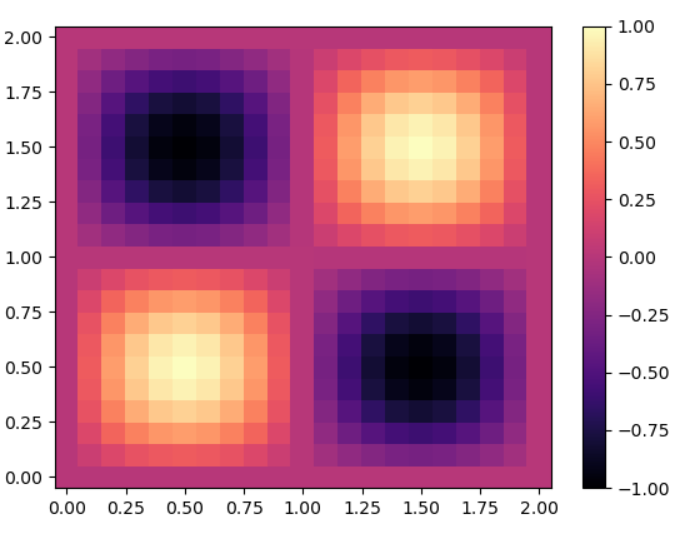
v1(u,x,y) = -3 - y

v2(u,x,y) = 2\*\*(1-u)

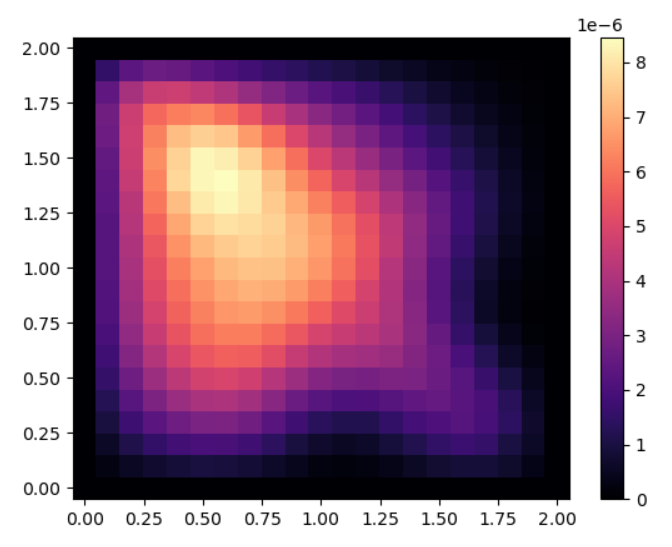
Найденное решение:



Точное решение:



Погрешности:



Затраченное время: 149.967

Итераций метода Ньютона: 18

Погрешность: 7.679e-05

1. Код программы

import numpy as np

import matplotlib.pyplot as plt

import numpy.linalg as lin

import time

import copy

import random

#Скалярное произведение

def Scalar(a,b):

s = 0

for i in range(a.shape[0]):

for j in range(a.shape[1]):

s += a[i][j]\*b[i][j]

return s

def Mult(u,hx,hy,flag): # def Mult(u,pe,hx,hy,a,b,flag):

v1, v2 = Form\_v(u,hx,hy) # определение вектора скорости

if flag == True:

Aptoch = Mult(tochresh,hx,hy,False)

m = u.shape[0]

n = u.shape[1]

res = copy.deepcopy(u)

for i in range(1,m - 1):

for j in range(1,n - 1):

if v1[i][j] > 0:

Dx = (u[i][j] - u[i-1][j]) / hx

else:

Dx = (u[i+1][j] - u[i][j]) / hx

if v2[i][j] > 0:

Dy = (u[i][j] - u[i][j-1]) / hy

else:

Dy = (u[i][j+1] - u[i][j]) / hy

res[i][j] = ((-1/Pe) \* ((u[i-1][j] - 2\*u[i][j] + u[i+1][j]) / (hx\*\*2) + (u[i][j-1] - 2\*u[i][j] + u[i][j+1]) / (hy\*\*2)) + v1[i][j] \* Dx + v2[i][j] \* Dy)

if flag == True:

res -= Aptoch

return res

# создание матрицы с границей g

def Matrix(hx,hy,g,U\_or\_F): #False - вернёт F, True - вернёт U

if not U\_or\_F:

U\_or\_F = True

U = Matrix(hx,hy,g,U\_or\_F)

n = Nx + 1

m = Ny + 1

x = hx

y = hy

matr = np.zeros((m,n))

# формируем центральную часть матрицы(за исключением первых и последних строк и столбцов)

for i in range(1,m-1):

x = hx

for j in range(1,n-1):

if not U\_or\_F:

matr[i][j] = f\_Func(x,y,v1\_Func(U[i,j],x,y),v2\_Func(U[i,j],x,y))

else:

matr[i][j] = U\_Func(x,y,v1\_Func(0,0,0),v2\_Func(0,0,0))

x += hx

y += hy

y = 0

# заполняем 1 и последний столбец

for i in range(m):

x = 0

matr[i][0] = g(x,y)

x = hx\*Nx

matr[i][n-1] = g(x,y)

y += hy

x = 0

# заполняем 1 и последнюю строку

for j in range(n):

y = 0

matr[0][j] = g(x,y)

y = hy\*Ny

matr[m-1][j] = g(x,y)

x += hx

return matr

# Функция определения вектора скорости

def Form\_v(u,hx,hy):

n = Nx + 1

m = Ny + 1

x = 0

y = 0

v11 = np.zeros((m, n))

v22 = np.zeros((m, n))

for i in range(m):

x = 0

for j in range(n):

v11[i][j] = v1\_Func(u[i,j],x, y)

v22[i][j] = v2\_Func(u[i,j],x, y)

x += hx

y += hy

return v11, v22

#Вычисление Dhf в точке x по направлению w

def DhF(x,w,b\_,hx,hy):

res = np.zeros((len(x),len(x[0])))

norm\_w = np.linalg.norm(w)

norm\_x = np.linalg.norm(x)

x\_real = Matrix(hx,hy,g\_Func,True)#False - вернёт F, True - вернёт U

#normx\_real = np.linalg.norm(x\_real)

if norm\_w > 10\*\*(-12):# w > 0

if norm\_x < 10\*\*(-12): # x = 0

xw = np.zeros((len(x),len(x[0])))

xw += hx\*w/norm\_w

res = (Mult(xw,hx,hy,True)- Mult(x,hx,hy,True))/ (hx/norm\_w)

else:# x > 0

xw = x + hx\*w\*norm\_x/norm\_w

res = (Mult(xw,hx,hy,True) - Mult(x,hx,hy,True)) / (hx\*norm\_x/norm\_w)

return res

def TFQMR(Un0,hx,hy,eps):

n = Nx + 1

m = Ny + 1

b\_0 = np.zeros((m,n))

x0 = np.zeros((m,n))

b\_ = -Mult(Un0,hx,hy,True)

r0 = b\_ - DhF(Un0,x0,b\_,hx,hy)

d0 = 0

teta0 = 0

eta0 = 0

w0 = copy.deepcopy(r0)

u0 = copy.deepcopy(r0)

v0 = DhF(Un0,u0,b\_,hx,hy)

tau0 = lin.norm(r0)

r0v = copy.deepcopy(r0)

ro0 = Scalar(r0v, r0)

count = 0

while True:

if (count % 2 == 0):

alpha1 = ro0 / Scalar(v0, r0v)

alpha0 = copy.deepcopy(alpha1)

u1 = u0 - np.dot(alpha0, v0)

ro01 = copy.deepcopy(ro0) # на шаге 0 запоминаем ро чтобы использовать на шаге 1

w1 = w0 - np.dot(alpha0, DhF(Un0,u0,b\_,hx,hy))

w0 = copy.deepcopy(w1)

d1 = u0 + (teta0 \*\* 2 / alpha0) \* eta0 \* d0

d0 = copy.deepcopy(d1)

teta1 = lin.norm(w1) / tau0

c1 = 1 / np.sqrt(1 + teta1 \*\* 2)

tau1 = tau0 \* teta1 \* c1

teta0 = copy.deepcopy(teta1)

tau0 = copy.deepcopy(tau1)

eta1 = (c1 \*\* 2) \* alpha0

eta0 = copy.deepcopy(eta1)

x1 = x0 + eta1 \* d1

if (count % 2 != 0):

ro1 = Scalar(w1, r0v)

ro0 = copy.deepcopy(ro1)

beta01 = ro1 / ro01

u1 = w1 + np.dot(beta01, u0)

v1 = DhF(Un0,u1,b\_,hx,hy) + (beta01 \* (DhF(Un0,u0,b\_,hx,hy) + beta01 \* v0))

v0 = copy.deepcopy(v1)

u0 = copy.deepcopy(u1)

norm = lin.norm(x1 - x0)

x0 = copy.deepcopy(x1)

if tau1 < eps:

break

elif (tau1 > 10\*\*10) or (count > 100):

break

count += 1

return x1

# Функция вывода графика

def Plot(matrix, hx, hy):

fig = plt.figure() # fig = plt.figure(figsize=(10,10))

x = np.linspace(0,Nx\*hx,Nx+1)

y = np.linspace(0,Ny\*hy,Ny+1)

x,y = np.meshgrid(x,y)

Z = np.transpose(matrix).reshape(x.shape) # Z = matrix

plt.pcolormesh(x,y,Z,cmap='magma',shading='auto') # plt.pcolormesh(x,y,matrix,cmap='magma',shading='auto')

plt.colorbar()

plt.show()

# алгоритм метода Ньютона-Крылова

def Newton\_Krylov():

iter = 0

n = Nx + 1

m = Ny + 1

t1 = time.time()

x0 = np.zeros((m,n))

norm = 1

while norm > epsN:

# внутренние итерации

xdelta = TFQMR(x0,hx,hy,eps)

norm = np.linalg.norm(xdelta)

print('Newton norm = ', norm)

x1 = x0 + xdelta

x0 = x1.copy()

if iter > 100:

print('всего итераций - ',iter)

break

iter += 1

t2 = time.time()

print('Затраченное время на рассчёты: ',t2-t1)

return (iter,x1)

# Тестовые задачи 1 - параболоид

# Тестовые задачи 2 - произведение синусов

def g\_Func(x,y): #g функция на границе области

return 0

def U\_Func(x,y,v1,v2): #функция температур U для тестового примера

#return x\*\*2 + y\*\*2 # тест 1

return np.sin(np.pi\*x) \* np.sin(np.pi\*y) # тест 2

def v1\_Func(u,x,y): #Функция v1 определения вектора скорости

return -3-y

def v2\_Func(u,x,y): #Функция v2 определения вектора скорости

return 2\*\*(1-u)

# получили после подстановки в исходное дифференциальное уравнение

def f\_Func(x,y,v1,v2): #f функция правой части

#return (1/Pe)\*4 - v1\*2\*x - v2\*2\*y # тест 1

return ((2\*np.pi\*\*2)/Pe)\*np.sin(np.pi\*x)\*np.sin(np.pi\*y)+v1\*np.pi\*np.cos(np.pi\*x)\*np.sin(np.pi\*y)+v2\*np.pi\*np.sin(np.pi\*x)\*np.cos(np.pi\*y) # тест 2

# Задача конвекции-диффузии

Pe = 1

hx = 0.1

hy = 0.1

eps = 0.001

Nx = 20

Ny = 20

epsN = 0.0001

tochresh = Matrix(hx,hy,g\_Func,True)

iter,x1 = Newton\_Krylov()

print('Newton итераций - ',iter)

print("График найденного решения:")

Plot(x1,hx,hy)

print("График точного решения:")

Plot(tochresh,hx,hy)

pogr = x1 - tochresh

print('pogr: ',lin.norm(pogr))

print("График погрешности точного решения и найденного:")

Plot(pogr,hx,hy)