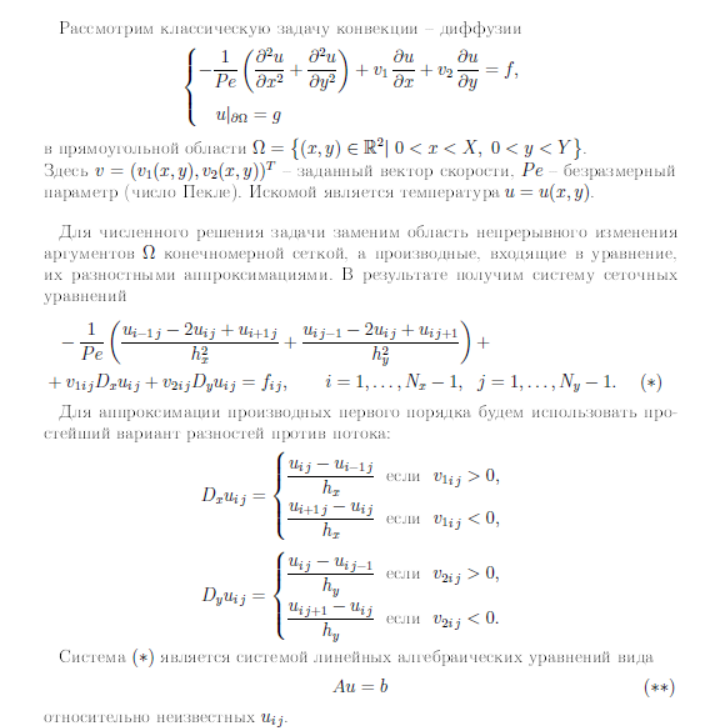
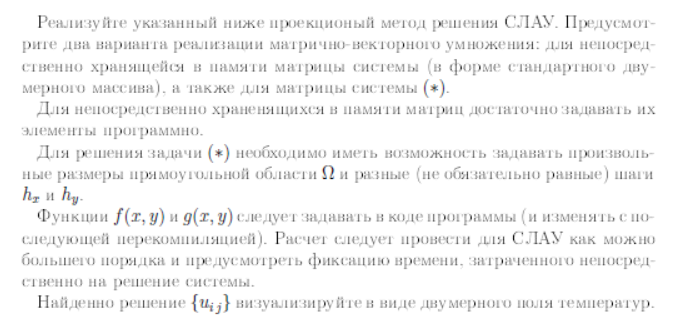
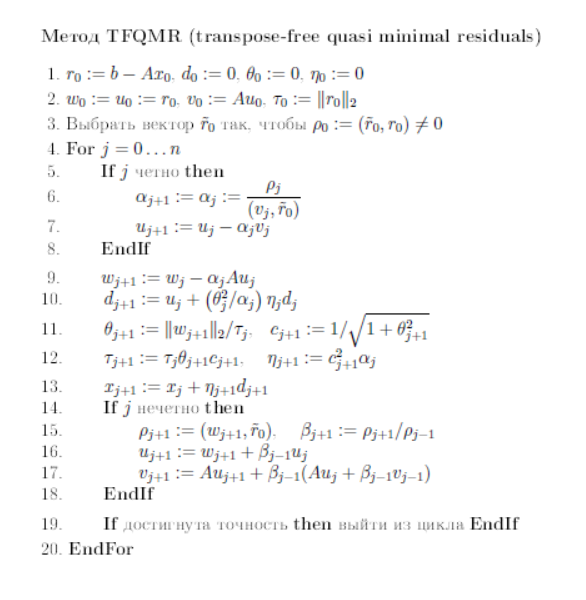
1. Постановка задачи





1. Алгоритм метода TFQMR



1. Решение задачи методом TFQMR

Решаем задачу конвекции-диффузии проекционным методом.

Pe = 1

hx = 0.1

hy = 0.1

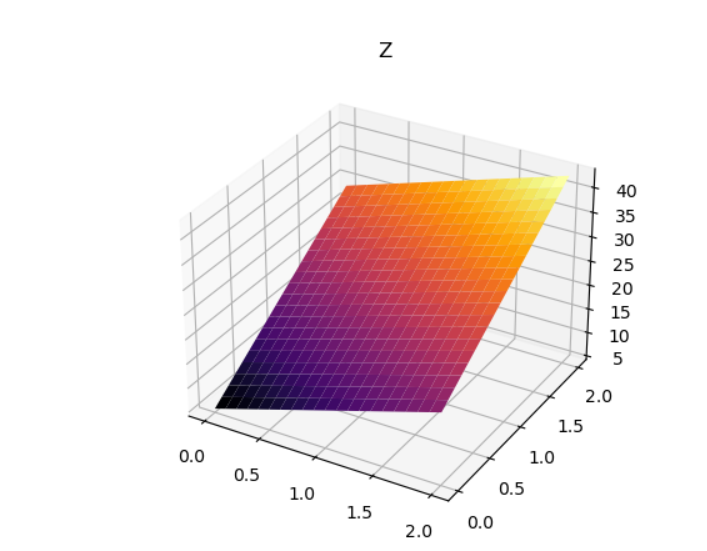
eps = 0.0001

Nx = 20

Ny = 20

U(x) = 12x + 7y + 5

Найденное решение:



Точное решение:

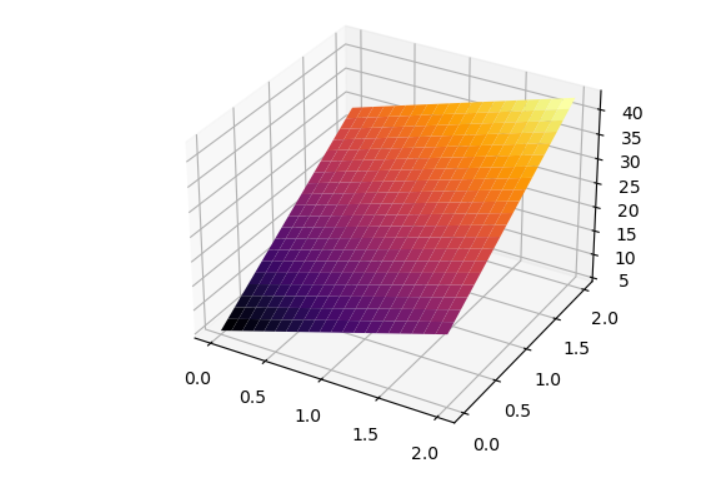
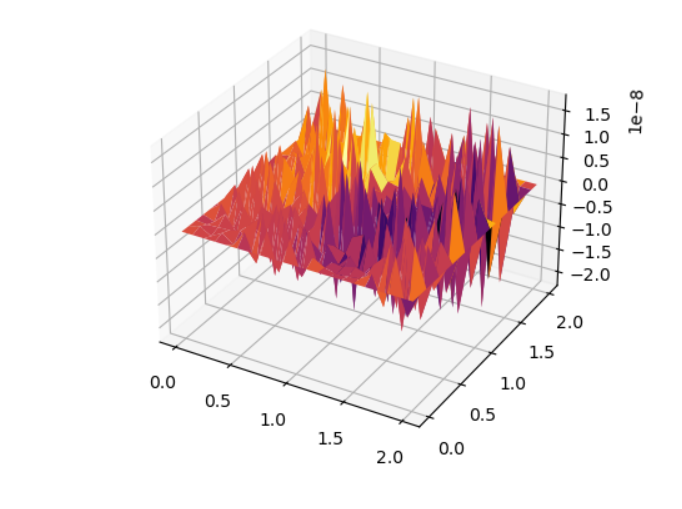


График погрешностей:



Норма невязки: 4.970909у-08

Число итераций: 112

1. Код программы

import numpy.linalg as lin

import numpy as np

import copy

from mpl\_toolkits.mplot3d import Axes3D

from matplotlib import cm

from matplotlib import ticker

import matplotlib.pyplot as plt

import time

# 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

# Функция определения вектора скорости

def Form\_v(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(x,y)

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

x += hx

y += hy

return v11, v22

# умножение вектора u на матрицу A

def Mult(u):

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

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)

return res

# формирование матрицы с границей g

def Matrix(hx,hy):

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):

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

x += hx

y += hy

y = 0

# заполянем 1 и последний столбец

for i in range(m):

x = 0

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

#x = 1

x = hx\*Nx

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

y += hy

x = 0

# заполняем 1 и последнюю строку

for j in range(n):

y = 0

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

#y = 1

y = hy\*Ny

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

x += hx

return matr

# -------------------------------------------------------------

def Norm(X):

m = 0

for i in X:

for j in i:

m += j\*j

m = math.sqrt(m)

return m

# -------------------------------------------------------------

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

n = Nx + 1

m = Ny + 1

tochresh = Matrix(hx,hy)

b\_ = Mult(tochresh)

x0 = np.dot(0.4,tochresh) #начальное приближение

r0 = b\_ - Mult(x0)

d0 = 0

teta0 = 0

eta0 = 0

w0 = copy.deepcopy(r0)

u0 = copy.deepcopy(r0)

v0 = Mult(u0)

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, Mult(u0))

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 == 1):

ro1 = Scalar(w1, r0v)

ro0 = copy.deepcopy(ro1)

beta01 = ro1 / ro01

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

v1 = Mult(u1) + (beta01 \* (Mult(u0) + 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):

break

count += 1

return (x1,count,norm)

# Построение графика

def Plot(matrix, hx, hy):

fig = plt.figure()

ax = fig.add\_subplot(projection='3d')

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)

ax.set\_title('Z')

surf = ax.plot\_surface(x,y,Z,cmap='inferno')

plt.show()

#Определение функций

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

return 12\*v1\_Func(x, y) + 7\*v2\_Func(x, y) + 5

#return 0

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

return 12\*x + 7\*y + 5

def v1\_Func(x, y): # первая компонента вектора скорости

return x

def v2\_Func(x, y): # вторая компонента вектора скорости

return y

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

return 12\*v1\_Func(x, y) + 7\*v2\_Func(x, y)

# Задача конвекции-диффузии

Pe = 1

hx = 0.1

hy = 0.1

eps = 0.0001

Nx = 20

Ny = 20

resh,chislo,norma = TFQMR(Pe, hx, hy, eps)

print("Число итераций для достижения заданной точности: ", chislo)

print("Норма невязки: ", norma)

print("График найденного решения:")

Plot(resh,hx,hy)

U = Matrix(hx,hy)

print("График точного решения:")

Plot(U,hx,hy)

pogr = resh - U

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

Plot(pogr,hx,hy)