<table>
<tr><td colspan="3">LODZ UNIVERSITY OF TECHNOLOGY
Faculty of Technical Physics, Information Technology and Applied Mathematics
Field of study: Modelling and Data Science
Course name: Computer Simulations</td></tr>
<tr><td>Academic year:
Semester:
Lab day and time:</td><td colspan="2">Task’s no:</td></tr>
<tr><td>Student ID number:</td><td>First name, last name
Marta Haik</td><td>Report grade:</td></tr>
<tr><td>Date of submission of the report:</td><td colspan="2"></td></tr>
</table>

1. Problem description

Infinitely long pipe of outer square cross section (side length = ***a***) has a rectangular hole (side lengths ***b*** and ***c***), which is not coaxial with the pipe. The heat conduction coefficient ***α*** of the pipe material is known. The temperature of a fluid flowing in the pipe is constant and equals to T_0. The heat transfer coefficient λ_i on the inside walls is known. The temperature of the medium in contact with the outer sides of the pipe is different for every face (it's T_1, T_2, T_3, and T_4, and all these temperatures are lower than T_0). The heat exchange coefficient λ_0 is the same for all outside walls. All the boundary conditions are of the second kind.

- Temperature field ***T(x,y)*** inside the rod and ***heat fluxes*** must be computed and presented.
- The accuracy of the solution must be verified.

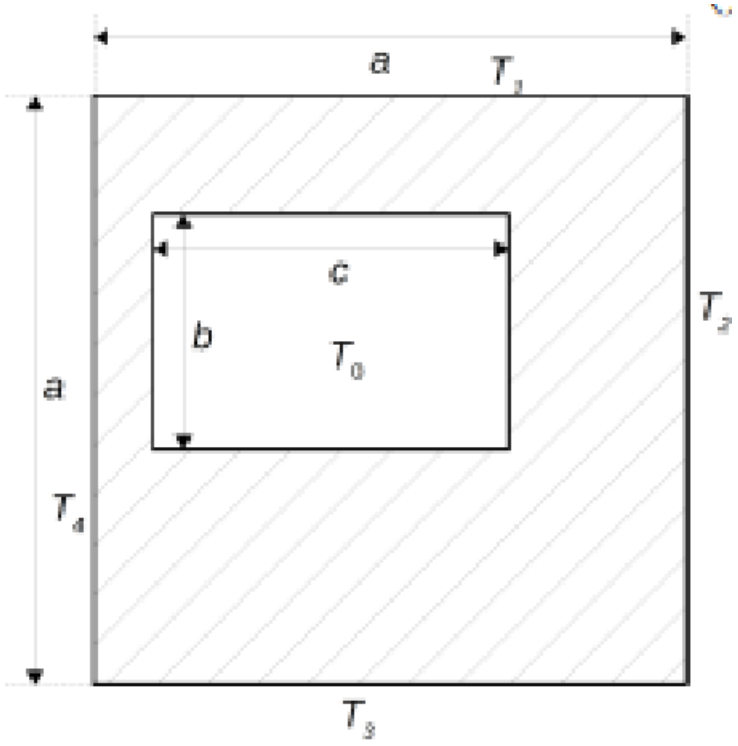

Figure 1 Illustration of the task [1]

2. The Program

Knowns: **a, b, c, α, T_0, λ_i,T_1, T_2, T_3, T_4, λ_0**

Unknowns **T(x,y) inside rod, Φ_q (heat fluxes) + solution's accuracy**

Additional information: Boundary conditions are of second kind and temperature distribution must be tested with Gauss theorem applied to heat fluxes.

There is a pipe with a rectangular hole, and we need to find the temperature inside the pipe and the heat flow within it. We need to divide the pipe and hole into smaller sections and calculate the temperature at each section. To calculate that, we apply heat flow equations and we get temperature at each section. From there we have to get the temperature values at each point, and then we can get the proper temperature distribution and heat flow in pipe.

It's a bit complicated to I did my best to comment the code.

```
# Computer Simulations TASK NO. 23
# MDS 4th semester
# Marta Haik
# index no. 242243

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from time import time
from itertools import count

plt.close("all")

#dimensions of the pipe
a = 1 #outside wall length

#hole dimensions
c = 0.4
b = 0.5

x0 = 0; xe = a
y0 = 0; ye = a

#hole position
c0 = 0.2; ce = c0 + c
b0 = 0.1; be = b0 + b

Nx = 51
x = np.linspace(x0,xe,Nx)
```

```
dx = x[1]-x[0]

Ny = 51
y = np.linspace(y0,ye,Ny)
dy = y[1]-y[0]

if abs(dx-dy)>1e-6:
    print("Grid is not square enough")
    exit(-1)

#indexes of the corners of the inside hole
ic0 = int(Nx*(c0/a))
ice = int(Nx*(ce/a))
ib0 = int(Ny*(b0/a))
ibe = int(Ny*(be/a))

# physical parameters:
k = 1          # [W/K/m] coeff. of heat conduction
lam0 = 10      # [W/K/m^2] coeff. of heat exchange for outside walls
lami = 10      # [W/K/m^2] coeff. of heat exchange for inside walls

# simplification for future calculation
A = k/lam0/dx
IA = 1/(1+A)

Ai = k/lami/dx
IAi = 1/(1+Ai)

# boundary temperatures:

#outside walls
T1 = 100
T2 = 200
T3 = 300
T4 = 400

#inside walls
T5 = 500

# conditions for stopping the computation:
delta_low = 1e-6         # condition for success
delta_high = 1e3          # condition of run-away
max_steps = int(1e5)     # max number of steps
```

```
#Gauss test points

GT_points = [
    [0.05,0.1,0.15,0.3],
    [0.65,0.7,0.8,0.9],
    [0.1,0.7,0.4,0.8],
    [0.65,0.3,0.9,0.5],
]

for i,points in enumerate(GT_points):
    lbx = points[0]
    lby = points[1]
    rtx = points[2]
    rty = points[3]

    #Check if any of the corners of the gauss test are inside or too close to the
inner boundry
    #we check for dx/2 since this it the breaking point after which the test is
conducted on boundry which will fail in most cases
    lb_corner_test_x = lbx > c0 - dx/2 and lbx < ce + dx/2
    lb_corner_test_y = lby > b0 - dy/2 and lby < be + dy/2
    rt_corner_test_x = rtx > c0 - dx/2 and rtx < ce + dx/2
    rt_corner_test_y = rty > b0 - dy/2 and rty < be + dy/2

    if (lb_corner_test_x and lb_corner_test_y) or (rt_corner_test_x and
rt_corner_test_y):
        print(
            f"Gauss test too close or inside the inner boundry which is not
continuous!\
            \nTest {i} will probably fail regardless of the correctness of the
solution"
            )


#Gauss test condition
max_flux = 1e-2

def corn(T):
    #corners for the outside boundry
    T[ 0, 0] = (T[ 0, 1] + T[ 1, 0]) /2
    T[-1,-1] = (T[-1,-2] + T[-2,-1]) /2
    T[ 0,-1] = (T[ 0,-2] + T[ 1,-1]) /2
    T[-1, 0] = (T[-1, 1] + T[-2, 0]) /2

    #corners for the inside boundry
```

```
    T[ic0,ib0] = (T[ic0+1,ib0] + T[ic0,ib0+1])/2
    T[ic0,ibe] = (T[ic0+1,ibe] + T[ic0,ibe-1])/2
    T[ice,ib0] = (T[ice-1,ib0] + T[ice,ib0+1])/2
    T[ice,ibe] = (T[ice-1,ibe] + T[ice,ibe-1])/2
    return T

def step(TT):
    #standard step
    #we can also change the boundry points since they will get overwritten in
boundry calculations
    T = TT.copy()
    T = (np.roll(T,1,axis=0) + np.roll(T,-1,axis=0) + np.roll(T,1,axis=1) +
np.roll(T,-1,axis=1))/4
    return T


def bnd(TT):
    T = TT.copy()

    #Second kind boundry conditions for the outside walls
    T[1:-1, -1 ] = (A*T[1:-1, -2 ] + T1) *IA
    T[ -1 ,1:-1] = (A*T[ -2 ,1:-1] + T2) *IA
    T[1:-1,  0 ] = (A*T[1:-1,  1 ] + T3) *IA
    T[  0 ,1:-1] = (A*T[  1 ,1:-1] + T4) *IA

    #Set inside the hole to T5
    T[ic0+1:ice,ib0+1:ibe] = T5

    #Second kind boundry conditions for the inside walls
    T[ic0,ib0+1:ibe] = (Ai*T[ic0-1,ib0+1:ibe] + T5) * IAi
    T[ice,ib0+1:ibe] = (Ai*T[ice+1,ib0+1:ibe] + T5) * IAi
    T[ic0+1:ice,ib0] = (Ai*T[ic0+1:ice,ib0-1] + T5) * IAi
    T[ic0+1:ice,ibe] = (Ai*T[ic0+1:ice,ibe+1] + T5) * IAi

    return T


X,Y = np.meshgrid(x,y,indexing='ij')
T0 = np.average([T1,T2,T3,T4,T5])
T = np.ones_like(X)*T0


t0 = time()
nT = T.copy()
for i in count():
```

```
    #operations are performed in this order to make sure that the boundry
calculation
    #overwrites the incorrect calculation at boundries performed by the normal
step
    nT = step(nT)
    nT = bnd(nT)
    nT = corn(nT)
    incr = np.max(abs(T-nT))
    T = nT.copy()

    if incr<delta_low:
        info = 'success'
        break
    elif incr>delta_high:
        info = 'run-away'
        break
    elif i==max_steps:
        info = 'too long'
        break

time_to_compute = time() - t0


fig = plt.figure(figsize=(16,9))

#because the boundry is not continous there are can be some bizarre artefacts
#in the 3d plot near the hole boundry
ax = fig.add_subplot(121,projection='3d',azim=30,elev=20)
pl = [ax.plot_surface(X,Y,T,cmap=cm.jet)]
ax.set(xlabel='x',ylabel='y',zlabel='T')
ax.set_title(
    f"Temperature distribution after {i} steps ({info})\
    \nmax incr. = {incr :.4e}\
    \nnumbers of grid points: Nx = {Nx}, Ny = {Ny}, k = {k}, lam0 = {lam0}, lami
= {lami}\
    \nfor initial temp. = {T0} execution time = {time_to_compute:.3f} s"
)

def Gt(i,x1,y1,x2,y2,ax):

    i1 = round((x1-x0)/dx)
    i2 = round((x2-x0)/dx)
    j1 = round((y1-y0)/dy)
    j2 = round((y2-y0)/dy)
    ax.plot(x[[i1,i2,i2,i1,i1]],\
```

```
            y[[j1,j1,j2,j2,j1]],'-k',lw=2,c='w')
    Fi1 = np.trapz(gx[i1,j1:j2+1],y[j1:j2+1])
    Fi2 = np.trapz(gx[i2,j1:j2+1],y[j1:j2+1])
    Fj1 = np.trapz(gy[i1:i2+1,j1],x[i1:i2+1])
    Fj2 = np.trapz(gy[i1:i2+1,j2],x[i1:i2+1])
    flux = Fj1 + Fi1 - Fj2 - Fi2
    print(
        f"Gauss test {i} for rectangle ({x[i1]},{y[j1]})-({x[i2]},{y[j2]})gave
{flux:.5e}\
        \n{'Test successful' if np.abs(flux) < max_flux else 'Test failed'}"
    )
    ax.text(x[i1],y[j1],"{:.4e}".format(flux))
    return flux < max_flux


gx,gy = np.gradient(T,dx,dy)

ax2 = fig.add_subplot(122)

#run gauss test for specified rectangles and save the results
GT_results = []
for j,points in enumerate(GT_points):
    lbx = points[0]
    lby = points[1]
    rux = points[2]
    ruy = points[3]
    ok = Gt(j,lbx,lby,rux,ruy,ax2)
    GT_results.append(ok)
GT_results = np.array(GT_results).all()


pc = ax2.pcolormesh(X,Y,T,cmap=cm.jet,shading='auto')
plt.colorbar(pc)
ax2.set(xlabel='x',ylabel='y')
ax2.axis('scaled')

cn = ax2.contour(X,Y,T,20,linewidths=0.5,colors='k')
plt.clabel(cn,fmt='%d')
every = Nx//10
ax2.quiver(X[::every,::every], Y[::every,::every], -gx[::every,::every], -
gy[::every,::every], zorder=10, units='dots', width=1)

ax2.set_title(
    f"Temperature distribution after {i} steps\
```

```
    \n{'All Gauss tests successful' if GT_results else 'One or more Gauss tests
failed'}\
    \n Showing flux every {every} points"
)

fig.tight_layout()
plt.show()
```

3. Typical parameters

The parameters are rather standard. They are simple (using full numbers, 1s, 10s and 100s) to not interfere with calculations, and inspired by programs we have already written in class.

There were some precalculations of constants to make the loop go faster, e.g:.

```
A = k/lam0/dx
IA = 1/(1+A)

Ai = k/lami/dx
IAi = 1/(1+Ai)
```

Which works, but it makes it a bit harder in case of finding a mistake here later on.

4. Results and correctness of the solution

The program appears to be correct in terms of syntax and logic. There are boundary conditions of the second kind (the task required only them), for the inside and outside of pipe's walls. It produces the following plots (*Fig. 2, Fig. 3*), showing the temperature distribution and heat flow:

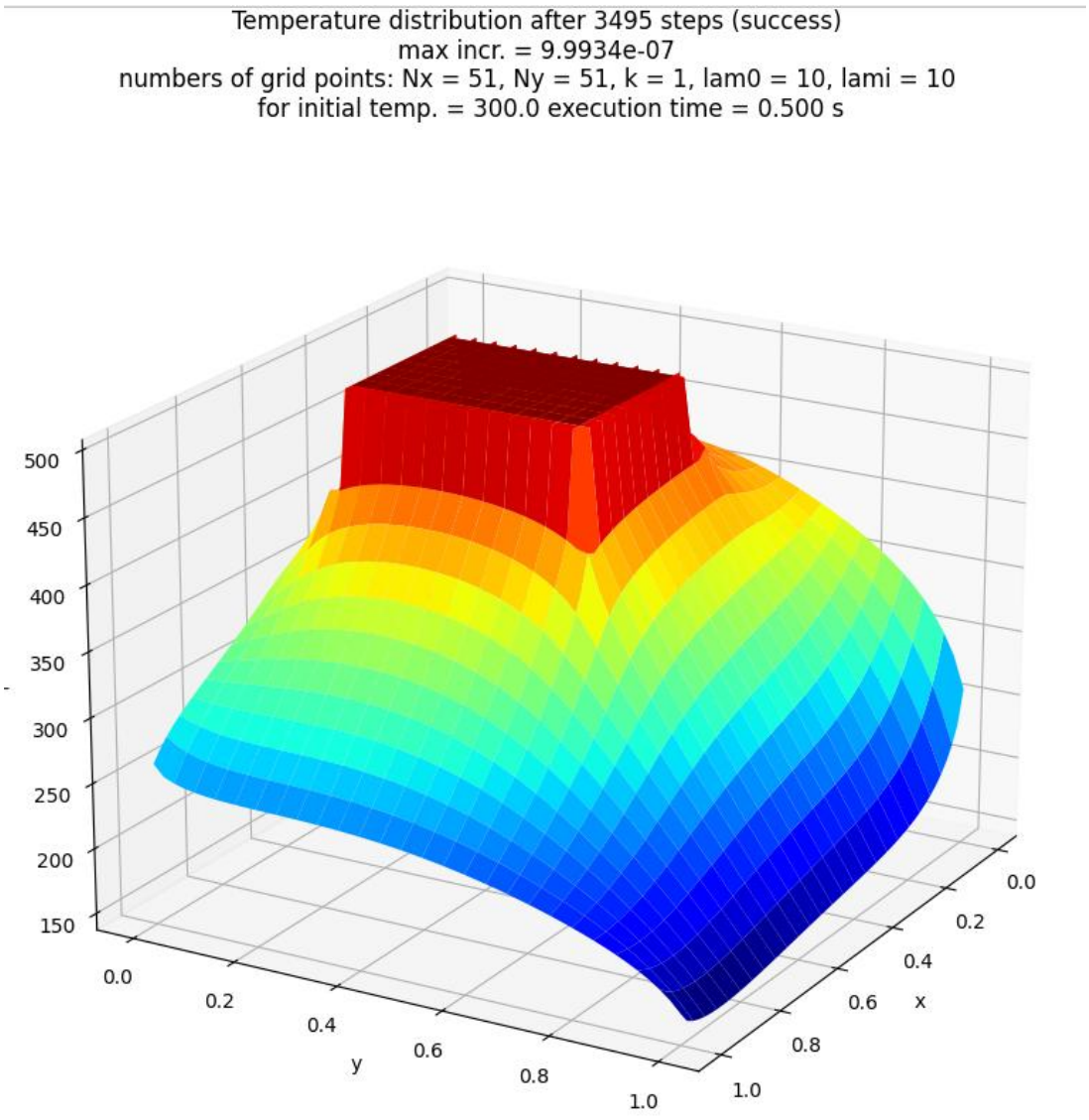

Figure 2 Temperature distribution T(x,y)

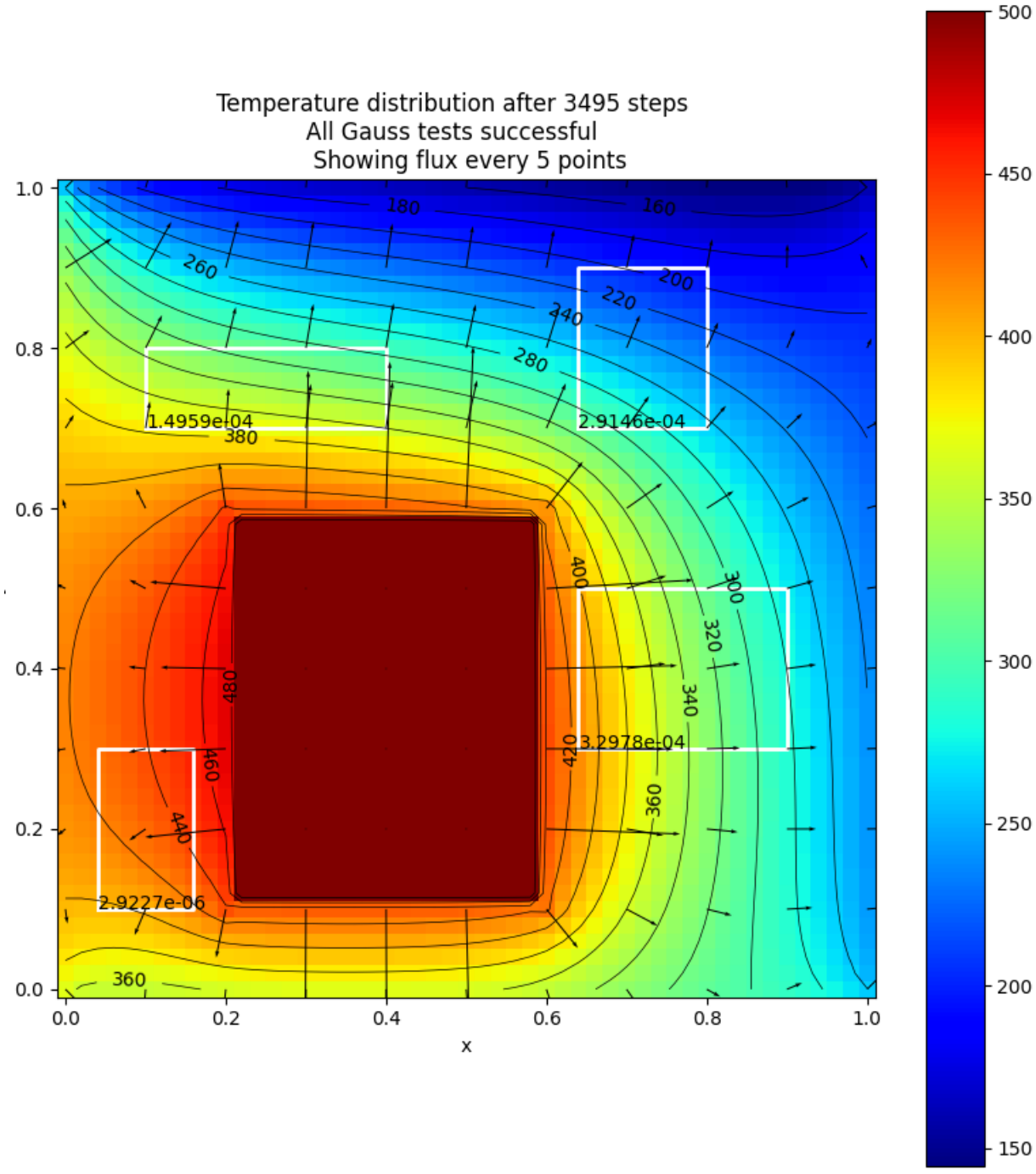

Figure 3 Temperature distribution and heat flow

There were Gauss tests performed for four rectangles (which they pass), they all have sufficiently small values - hopefully making it good-enough solution.

Initially, I have made one rectangle too close to the hole (*Fig. 4*), which impacted the Gauss test:

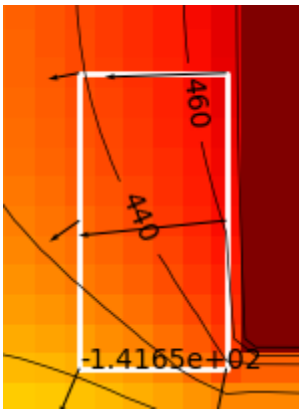

Figure 4 Wrong placement of rectangle

Because the rectangle was too close to the border it has been showing the error. The new placement is more representative and successfully passes the Gauss test (*Fig. 5*).

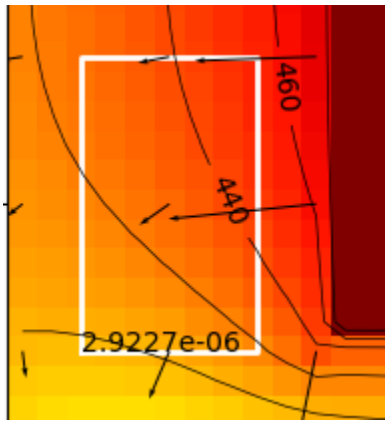

Figure 5 Fixed position of rectangle

5. Sources

[1] pdf file “assignment equations” on the FTIMS WIKAMP platform