Simulation of Heat transfer
Daniel Stefanik

1. Explanation of aim

The aim of this exercise was to create model of physical phenomenon - heat transfer in a specified object. Implementation of computer simulation helps to understand the mechanisms describing the phenomenon and it theoretical concept. Another objective is to test the program for various parameters and check the numerical stability of this system.

2. Used tools

Simulation was created in python language using Jupyter Notebook application. Libraries and their use:

- *matplotlib* - creating plot of heat transfer result,
- *Axes3D* - creating 3d diagram,
- *numpy* - operations on matrices.

3. Results

a. Simulation parameters

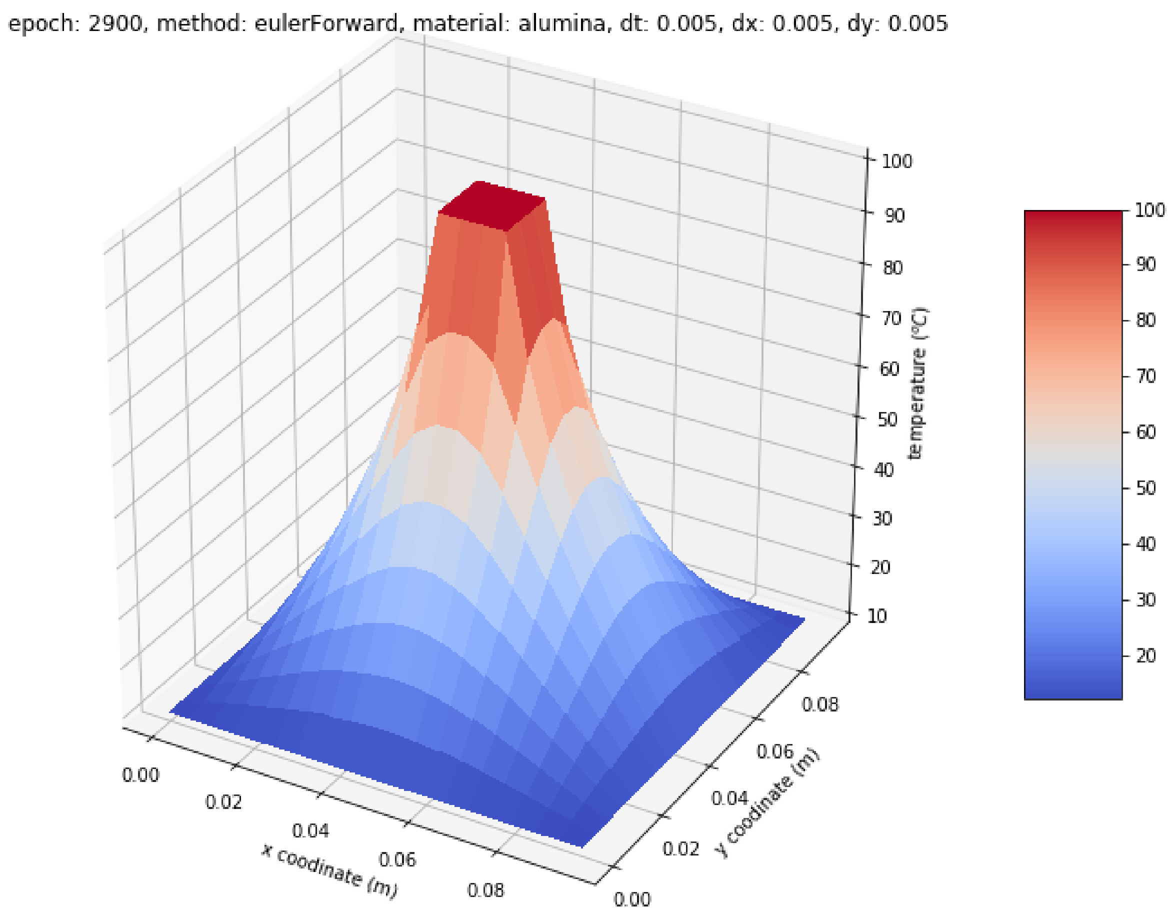
Result of simulation of heat transfer presented below contains the following parameters:

- surface size *A* = 0.1 m,
- material size *B* = 0.02 m,
- surface temperature *T0* = 20 Celsius degrees,
- material temperature *T1* = 100 Celsius degrees,
- boundary temperature *T2* = 10 Celsius degrees,
- material: alumuina,
- sensity *ro* = 2700 kg/m3,
- specific heat *cw* = 900 J/kgK,
- thermal conductivity *K* = 237 W/mK,
- time step *dt* = 0.005,

- distance between computational nodes in x direction *dx* = 0.005,
- distance between computational nodes in y direction *dy* = 0.005,
- limit of minimal change of temperature, when reached the simulation stops *threshold* = 0.01.

Parameters *dt, dx, dy* and *threshold* were selected based on application behavior.

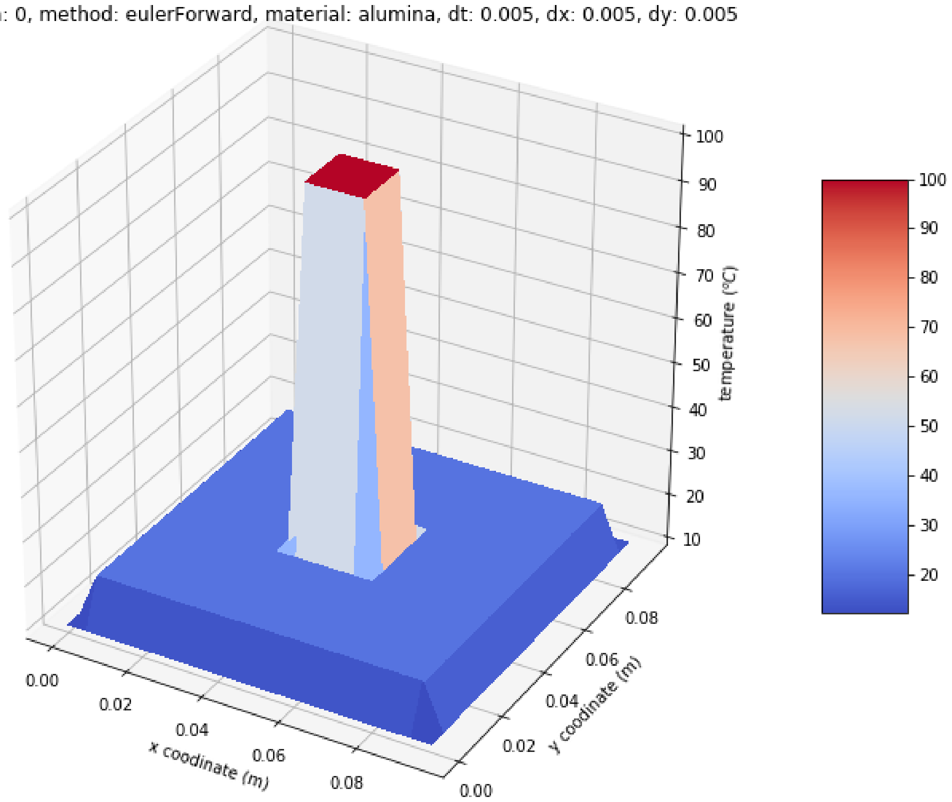
- *dt* - if set to 0.05 simulation does not visualize well the distribution of temperature because it ends after few iterations, and value of 0.01 destabilize whole simulation, when set to 0.0001 simulation take a lot of time to see the whole process;
- *dx* and *dy* - if set to 0.02 number of simulated points on surface is 5 and the material do not looks like square, when set to 0.002 program calculations take some time and application works slowly, value of 0.001 destabilize calculation and the visualization shows incorrect results;
- *threshold* - after change of temperature reaches threshold less or equal to 0.01, we can not observe significant changes on the diagram.



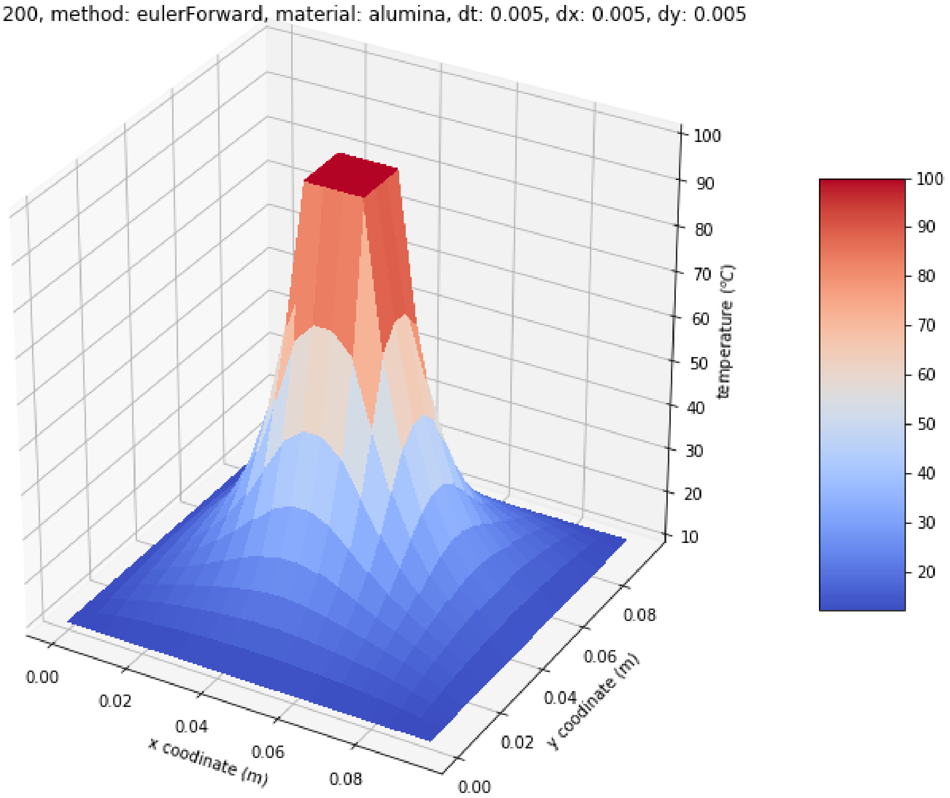
b. temperature distribution

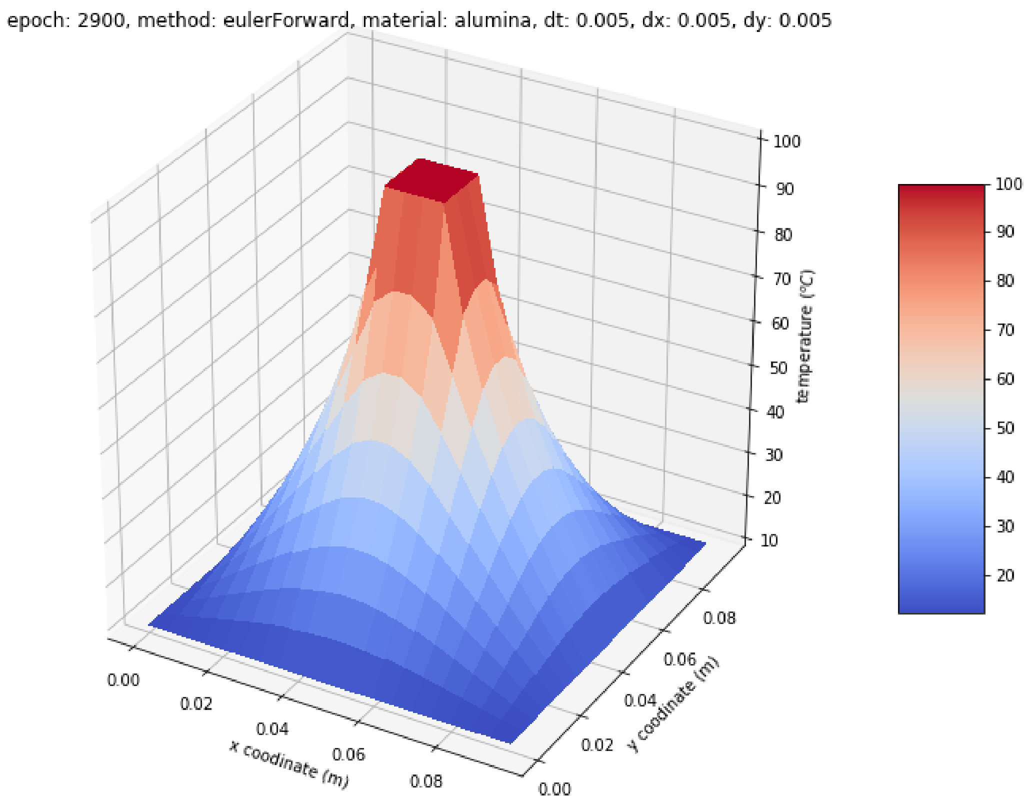
Three diagrams below shows temperature distribution in a metal plate made of alumina. We can observe that after some amount of time steps temperature stabilizes.

epoch: 0, method: eulerForward, material: alumina, dt: 0.005, dx: 0.005, dy: 0.005



epoch: 200, method: eulerForward, material: alumina, dt: 0.005, dx: 0.005, dy: 0.005





c. Stability

Parameter *dt* has the biggest influence on stabilization of the system. For alumina when set to 0.005 stabilization for threshold = 0.01 occurs after 2900 time steps. Increasing *dt* to 0.01 reduces the amount of time steps to 1900 and for *dt* = 0.02 only 9000 time steps is needed. Similarly for the cooper material:

- *dt* = 0.005, minimum time steps to stabilization = 2400,
- *dt* = 0.01, minimum time steps to stabilization = 1400,
- *dt* = 0.02, minimum time steps to stabilization = 800.

The stainless steel material needs more time steps:

- *dt* = 0.005, minimum time steps to stabilization = 12500,
- *dt* = 0.01, minimum time steps to stabilization = 7100,
- *dt* = 0.02, minimum time steps to stabilization = 4000.

Above calculations was conducted for *dx* and *dy* equal to 0.005. For *dx = 0.01, dy = 0.01* results are shown below:

Alumina:

- *dt* = 0.005, minimum time steps to stabilization = 2200,
- *dt* = 0.01, minimum time steps to stabilization = 1300,
- *dt* = 0.02, minimum time steps to stabilization = 700.

Cooper:

- *dt* = 0.005, minimum time steps to stabilization = 1900,
- *dt* = 0.01, minimum time steps to stabilization = 1100,
- *dt* = 0.02, minimum time steps to stabilization = 600.

Stainless steel:

- *dt* = 0.005, minimum time steps to stabilization = 8600,
- *dt* = 0.01, minimum time steps to stabilization = 5200,
- *dt* = 0.02, minimum time steps to stabilization = 3000.

4. Discussion

a. About simulation

Script contains four main parts:

- variable initiation including initial conditions and other necessary parameters,
- diagram methods responsible for initialization and visualizing the temperature distribution,
- computation methods - initialize matrix values with proper temperatures and calculates steps of simulation,
- main loop in which above methods are called and when threshold is reached applications ends work.

Simulation was run with various parameters which was described in subsections 3.a and 3.c.

b. Conclusions

Reaching set objectives helped us to understand theoretical concept and mechanisms of this physical phenomenon. Trying various parameters show us that numerical computation can be easily destabilized what leads to incorrect results.

5. Code listening

```
import matplotlib
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import numpy as np

from IPython import display

% matplotlib notebook
% matplotlib inline

# Q = Cin * m * deltaT
```

```python
# P * dt
# m = ro * B^2 *h
# Q = P * dt


class Material:
    def __init__(self, name, density, specificHeat,
thermalConductivity):
        self.name = name
        self.ro = density
        self.cw = specificHeat
        self.K = thermalConductivity

# Variable initiation

A = 0.1  # m
B = 0.02  # m
h = 0.002  # m

epochs = 999  # end defined by threshold
threshold = 0.01

# type 1
T0 = 20.0  # degrees
T1 = 100.0  # degrees
T2 = 10.0  # degrees

# type 2
# P=100W
# th = 10s
# B-edge is isolated

alumina = Material("alumina", 2700.0, 900.0, 237.0)
cooper = Material("cooper", 8920.0, 380.0, 401.0)
stainlessSteel = Material("stainless steel", 7860.0, 450.0, 58.0)

mat = alumina
# mat = cooper
# mat = stainlessSteel

method = 'eulerForward'
# method = 'centralDifference'

dt = 0.005
dx = 0.005
```

```
dy = dx

SIZE_T = int(epochs / dt)
SIZE_X = int(A / dx)
SIZE_Y = int(A / dy)

plotEvery = 10

print("EPOCHS: " + str(SIZE_T) + " SIZE_X: " + str(SIZE_X) + " SIZE_Y: "
+ str(SIZE_Y))

# Diagram methods

def initPlot():
    fig = plt.figure(figsize=(12, 10))
    ax = fig.gca(projection='3d')
    surf = ax.plot_surface(gridX, gridY, matrix, cmap=cm.coolwarm,
linewidth=0, antialiased=False)
    fig.colorbar(surf, shrink=0.5, aspect=5)

    # Customize axis.
    ax.set_zlim(0.0, 120.0)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    showAxis(ax)
    return fig, ax

def showAxis(ax):
    ax.set_xlabel('x coodinate (m)')
    ax.set_ylabel('y coodinate (m)')
    ax.set_zlabel('temperature ($^oC$)')
    ax.set_title('epoch: ' + str(epoch) + ', method: ' + method + ',
material: ' + str(mat.name) + ', dt: ' + str(
        dt) + ', dx: ' + str(dx) + ', dy: ' + str(dy))

def plotMatrix(matrix):
    plt.cla()
    ax.plot_surface(gridX, gridY, matrix, cmap=cm.coolwarm, linewidth=0,
antialiased=False)
    showAxis(ax)
    display.clear_output(wait=True)
    display.display(plt.gcf())

# Computation methods
```

```
def setTempB(matrix, temp):
    matrix[int((A / 2 - B / 2) / dx):int((A / 2 + B / 2) / dx),
    int((A / 2 - B / 2) / dy):int((A / 2 + B / 2) / dy)] = temp

def initMatrix():
    matrix = np.zeros((SIZE_X, SIZE_Y))
    matrix[:, :] = T0
    border_size = 1
    if method == 'eulerForward':
        border_size = 1
    elif method == 'centralDifference':
        border_size = 2

    matrix[0:border_size, :] = T2
    matrix[SIZE_X - border_size:SIZE_X, :] = T2
    matrix[:, 0:border_size] = T2
    matrix[:, SIZE_Y - border_size:SIZE_Y] = T2
    setTempB(matrix, T1)
    print(matrix.astype(int))
    return matrix

def eulerForward(matrix, tmpMatrix):
    for i in range(1, SIZE_X - 1):
        for j in range(1, SIZE_Y - 1):
            Tn = matrix[i, j]
            xsd = (matrix[i + 1, j] - 2 * Tn + matrix[i - 1, j])
            ysd = (matrix[i, j + 1] - 2 * Tn + matrix[i, j - 1])
            Tn1 = Tn + (xParam / (dx * dx)) * xsd + (yParam / (dy * dy))
* ysd
            tmpMatrix[i, j] = Tn1
    setTempB(tmpMatrix, T1)
    matrix = tmpMatrix.copy()
    return matrix

# def centralDifference(matrix, tmpMatrix, m2):
#     for i in range(2,SIZE_X-2):
#         for j in range(2,SIZE_Y-2):
#             Tn = matrix[i,j]
#             xsd = (-matrix[i-2,j] + 16*matrix[i-1,j] - 30*Tn +
16*matrix[i+1,j] - matrix[i+2,j])
#             ysd = (-matrix[i,j-2] + 16*matrix[i,j-1] - 30*Tn +
16*matrix[i,j+1] - matrix[i,j+2])
#             Tn1 = -3*m2[i,j]+4*Tn - 2*((xParam/(dx*dx*144)) * xsd +
(yParam/(dy*dy*144)) * ysd)
# #               print(matrix.round(3))
```

```python
# #             print(" i: "+str(i)+
# #                   " j: "+str(j)+
# #                   " matrix[i+1,j]: "+str(matrix[i+1,j])+
# #                   " matrix[i+2,j]: "+str(matrix[i+2,j])+
# #                   " matrix[i,j+1]: "+str(matrix[i,j+1])+
# #                   " matrix[i,j+2]: "+str(matrix[i,j+2])+
# #                   " Tn: "+str(Tn)+
# #                   " (xParam/(dx*dx)) "+str((xParam/(dx*dx)))+
# #                   " xsd "+str(xsd)+
# #                   " (yParam/(dy*dy)) "+str((yParam/(dy*dy)))+
# #                   " ysd "+str(ysd)+
# #                   " res "+str((xParam/(dx*dx)) * xsd +
(yParam/(dy*dy)) * ysd)+
# #                   " =  "+str(Tn1))
#             tmpMatrix[i,j] = Tn1
#     setTempB(tmpMatrix,T1)
#     matrix = tmpMatrix.copy()
#     return matrix

matrix = initMatrix()

xParam = (mat.K * dt) / (mat.cw * mat.ro)
yParam = (mat.K * dt) / (mat.cw * mat.ro)

epoch = 0
gridX, gridY = np.meshgrid(np.arange(0, A, dx), np.arange(0, A, dy))
fig, ax = initPlot()

change = threshold + 1

newS = 0
tmpMatrix = matrix.copy()

for epoch in range(SIZE_T + 1):
    oldS = matrix.sum()
    if epoch % plotEvery == 0:
        plotMatrix(matrix)
        print(str(oldS) + " - " + str(newS) + " = " + str(change))

        if threshold > change:
            break
    if method == 'eulerForward':
        matrix = eulerForward(matrix, tmpMatrix)
    # elif method == 'centralDifference':
    #     matrix = centralDifference(matrix, tmpMatrix)
    newS = matrix.sum()
    change = abs(oldS - newS)
```