Exercise 1 - Heat Conduction with multiple layers and variable properties

Florian Schnabel

Apr 16, 2023

CONTENTS

This script aims to calculate the temperature field across a multilayered building component. The building component can have an arbitrary number of layers with varying properties. Furthermore, the thickness of the layer may change as well as the chosen number of finite volumes used to discretize the domain. Such a multilayered building component is depicted in Fig. **??**.

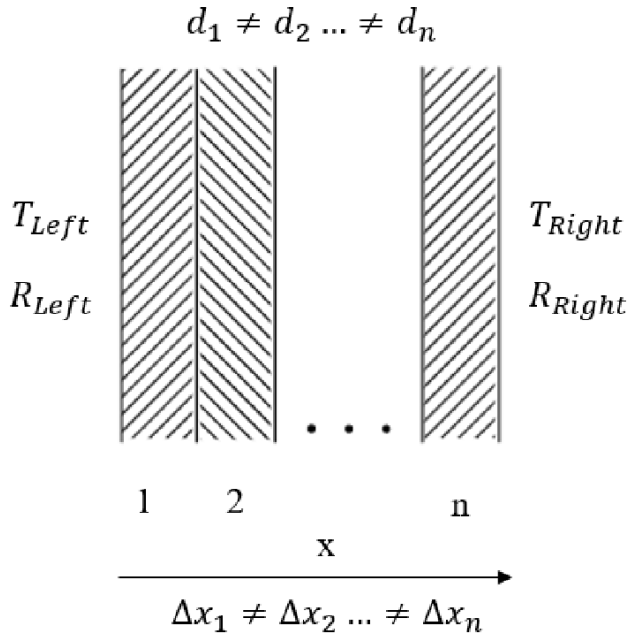


Fig. 1: Multilayered building Component and discretisation requirement [excersise Description]

To allow a numerical solution of the problem the partial differential equation for heat conduction needs to be discretised. Equation for heat conduction can be seen in (**??**) [3]:

$$\rho \cdot c \cdot \frac{dT}{dt} = \nabla(\lambda \nabla T) \tag{1}$$

reduced to one dimension:

$$\rho \cdot c \cdot \frac{dT}{dt} = \frac{d}{dx}(\lambda \frac{dT}{dx}) \tag{1}$$

0.1 Space discretication

$R_{ci} = 0$

To calculate the temperaturefield cells and cunduction between them are represented by RC-Networks. Each cell is represented by a resistance and the conductivity to the neighboring cells. The conductivity between Interrior cells are calculated as follows (For sake of simplicity the surface resistance between layers is neglected. $R_{ci} = 0$) [2]:

$$K_{i-0.5} = \frac{1}{\frac{0.5\Delta x_{i-1}}{\lambda_{i-1}} + R_{ci} + \frac{0.5\Delta x_i}{\lambda_i}} \tag{1}$$

For a cell inside a layer ($\lambda_{i-1} = \lambda_i = \lambda_{i+1}$ and $\Delta x_{i-1} = \Delta x_i = \Delta x_{i+1}$) the conductivity to neighbouring cells collapses to:

$$K_{i-0.5} = K_{i+0.5} = \frac{\lambda_i}{\Delta x_i} \tag{1}$$

And for Cells at the boundary:

$$K_{BC,1} = K_{0.5} = \frac{1}{R_{s,1} + \frac{0.5\Delta x_1}{\lambda_1}} \tag{2}$$

0.2 Time Integration

using the explicit euler sheme the next Timestep can be calculated as follows:

For cells at the left boundary: $T_1^{n+1} = T_1^n + F_{o,1}^* \cdot (K_{BC,1} \cdot T_{BC}^n - (K_{BC,l} + K_2) \cdot T_1^n + K_2 \cdot T_2^n)$ (1)

For interior cells: $T_i^{n+1} = T_1^n + F_{o,i}^* \cdot (K_{i-1} \cdot T_{i-1}^n - (K_{i-1} + K_{i+1}) \cdot T_i^n + K_{i+1} \cdot T_{i+1}^n)$ (1)

For cells at the right boundary: $T_j^{n+1} = T_j^n + F_{o,j}^* \cdot (K_{j-1} \cdot T_{j-1}^n - (K_{j-1} + K_{BC,2}) \cdot T_j^n + K_{BC,2} \cdot T_{BC,2}^n)$ (1)

(1)

or generally written (based on [3]):

$$T^{n+1} = T^n + F_o^* \cdot K \cdot T^n \tag{1}$$

With the conductivity matrix K:

$$K = \begin{bmatrix} 0 & 0 & 0 & 0 & (...) & 0 & 0 \\ 0 & 0 & & & & & \\ K_{BC,1} & -(K_{BC,1} + K_w) & K_w & 0 & (...) & 0 & 0 \\ 0 & 0 & & & & & \\ 0 & K_e & -(K_e + K_w) & K_w & (...) & 0 & 0 \\ 0 & 0 & & & & & \\ (...) & (...) & (...) & (...) & (...) & (...) & (...) \\ (...) & (...) & & & & & \\ 0 & 0 & 0 & 0 & (...) & K_e & -(K_e + K_w) \\ K_w & 0 & & & & & \\ 0 & 0 & 0 & 0 & (...) & 0 & K_e \\ -(K_e + K_{BC,2}) & K_{BC,2} & & & & & \\ 0 & 0 & 0 & 0 & (...) & 0 & 0 \\ 0 & 0 & & & & & \end{bmatrix}$$

and the adapted "resistance" matrix F_o^* (layers a,b,c):

$$F_o^* = \begin{bmatrix} F_{o,a}^* & 0 & & & & & \\ 0 & F_{o,a}^* & 0 & & & & \\ & 0 & (...) & 0 & & & \\ & & 0 & F_{o,b}^* & 0 & & \\ & & & 0 & F_{o,b}^* & 0 & \\ & & & & 0 & (...) & 0 \\ & & & & & 0 & F_{o,c}^* \\ 0 & & & & & & \end{bmatrix} \qquad F_{o,i}^* = \frac{\Delta t}{\Delta x_i} \cdot \frac{1}{\rho_i \cdot c_i}$$

with the heat capacity c_i and the density ρ_i.

0.2.1 Stability

0.2.2 Steady state criteria

The simulation stops once a steady state is achieved. The ISO 10211 [1] norm gives a steady state criterea for thermal bridges in building constructions. For iterative solution techniques the following limit is provided:

$$\sum q_{in} - \frac{\sum q}{2} \leq 0.0001 \tag{2}$$

As derived in the exercise the heatflow through the surface of a finite volume can be approximated as follows:

$$q_e = \lambda_e \cdot \frac{T_E - T_P}{\Delta x_e} \tag{3}$$

$$q_w = \lambda_w \cdot \frac{T_P - T_W}{\Delta x_w} \tag{3}$$

Specifing for the first and last cells with $\Delta x/2$ and R_{se} / R_{si} the heatflow in and out of the construction can be written as:

$$q_{in} = K_{BC1} \cdot (T_{BC1} - T_1) \tag{3}$$

$$q_{out} = K_{BC2} \cdot (T_{BC2} - T_n) \tag{4}$$

0.3 Implementation

```
%reset

#Libraries
import numpy as np
import matplotlib.pyplot as plt
```

```
class boundary:
    def __init__(self, temperature1, temperature2, resistance1, resistance2):
        self.temperature1 = temperature1
        self.temperature2 = temperature2
        self.resistance1 = resistance1
        self.resistance2 = resistance2
```

```
class layer:
    # n summarizes the number of all cells
    # start value for n = number of boundaries, makes place for boundary temp
    n=2
    # width sumarizes the thickness of all layers
    width_sum =0

    def __init__(self, material, width, n_cells, thermConduct, heatCap, density):
        self.material = material
        self.width = width
        self.n_cells = n_cells
        self.thermConduct = thermConduct
        self.heatCap = heatCap
```

(continues on next page)

(continued from previous page)

```
        self.density = density

        # Adding number of cells and width to class counters
        layer.n += self.n_cells
        layer.width_sum += self.width
```

0.3.1 construction of the layer and boundary objects

```
boundaries = boundary(0,20,0.04,0.10)

#Beware, each layer must have at least 3 cells!
layers = []
layers.append(layer("Dämmung", 0.2, 20, 0.04, 1470, 1000))
layers.append(layer("STB", 0.15, 15, 2.3, 1000, 2500))
layers.append(layer("Putz", 0.02, 4, 0.9, 1000, 2000))
```

0.3.2 Definition of the simulation time and the step size.

```
#Time steps
ts = 800000
#simulation time
tsim= 3000000 #s
#step sizes
deltat = tsim/ts
```

0.3.3 Construction of the K and F_o^* matrices

```
# reference number of cells to increase readability
n = layer.n

# Initialising conductivity matrix K, "resistance" array Fo, and the error array␣
 ↪deltaGrad
K = np.zeros((n,n))
Fo = np.zeros((n,n))

# iterater to keep track of cell numbers
counter = 1
for layer in layers:
    #step size for the current layer
    deltaX=layer.width / layer.n_cells

    #initialize empty conductivity matrix for the current layer
    K_current = np.zeros((n,n))

    #defining the "east" and "west" conductivity for interior cells
    K_current[range(counter+1,counter+layer.n_cells),range(counter,counter+layer.n_
 ↪cells-1)]= layer.thermConduct / deltaX
    K_current[range(counter,counter+layer.n_cells-1),range(counter+1,counter+layer.n_
 ↪cells)]= layer.thermConduct / deltaX
```

(continues on next page)

(continued from previous page)

```
    #defining the "east" and "west" conductivity for the last and first cell of each␣
 ↪layer
    #these are added up between layers, with half the cell distance
    K_current[(counter,counter+layer.n_cells),(counter-1,counter+layer.n_cells-1)]=␣
 ↪deltaX / (2 * layer.thermConduct)
    K_current[(counter-1,counter+layer.n_cells-1),(counter,counter+layer.n_cells)]=␣
 ↪deltaX / (2 * layer.thermConduct)

    # Filling the "resistance" matrix for each layer
    Fo[range(counter,counter+layer.n_cells),range(counter,counter+layer.n_cells)] =␣
 ↪deltat /(layer.density * layer.heatCap * deltaX)

    K += K_current

    #Inverting the Konductivity between surfaces
    K[(counter),(counter-1)] = 1/K[(counter),(counter-1)]
    K[(counter-1),(counter)] = 1/K[(counter-1),(counter)]

    counter += layer.n_cells
```

```
# Boundary Conditions
K_l = 1 / (boundaries.resistance1 + 0.5* (layers[0].width / layers[0].n_cells) /␣
 ↪layers[0].thermConduct)
K_r = 1 / (boundaries.resistance2 + 0.5* (layers[-1].width / layers[-1].n_cells) /␣
 ↪layers[-1].thermConduct)

K[1,0] = K_l
K[len(K)-2,len(K)-1] = K_r
```

```
#The diagonal of the conductivity matrix is defined as the negative sum of the
#"east" and "west" conductivity for the cell
K[range(0,len(K)),range(0,len(K))]= -np.sum(K,axis=1)

#free first and last row; this leads to constant temperatures at the boundaries
K[0,:]=0
K[-1,:]=0
```

```
#initialize the temperaturefield
T = np.ones(layer.n) * 10

#Insert Boundary Konditions
T[0]= boundaries.temperature1
T[-1]= boundaries.temperature2
```

```
#Calculate the adjusted Matrix outside the loop as both are time independent
K=np.dot(Fo,K)
T_plus =  np.dot(K,T) + T

#Initialize heatflow for the loop criteria
qin = 1
qout = 0
```

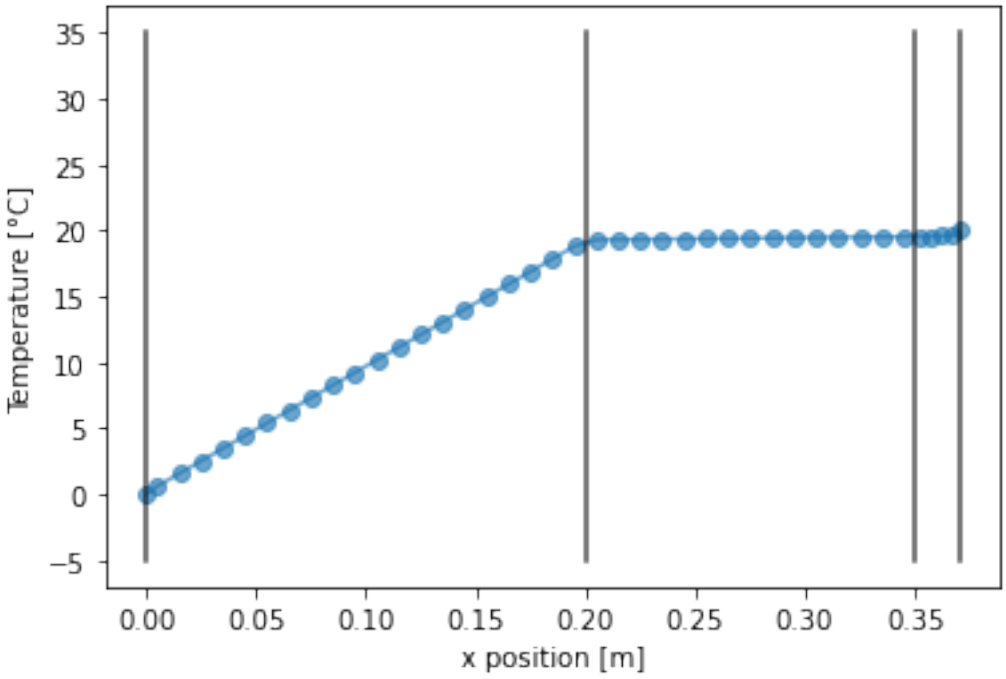
(continues on next page)

(continued from previous page)

```
t=0
while qin - (qin+qout)/2 > 0.0001:
    T=T_plus
    T_plus =  np.dot(K,T) + T

    #Calculate the heatflow for the break criteria
    qout = abs(K_l*(T_plus[1]-T_plus[0]))
    qin = abs(K_r*(T_plus[-1]-T_plus[-2]))

    t += deltat
```

```
x_pos= np.array([0])
counter=0
width_sum=0
for layer in layers:
    width_cell = layer.width / layer.n_cells
    x_pos = np.append(x_pos , np.arange(counter+ width_cell/2,layer.width+counter,␣
 ↪width_cell))
    counter += layer.width
    width_sum += layer.width

x_pos = np.append(x_pos ,np.array([width_sum]))
```

```
plt.xlabel("x position [m]")
plt.ylabel("Temperature [°C]")
plt.plot(x_pos, T_plus, 'o-', alpha=0.65)
x_w=0
for layer in layers:
    plt.plot([x_w,x_w], [-5,35], color='k', alpha=0.7)
    x_w += layer.width
plt.plot([layer.width_sum,layer.width_sum], [-5,35], color='k', alpha=0.7)
plt.show()
a = np.array([1,-1,3])
```



0.3.4 Determine the time in hours when a 3-layered component reaches the steady state

```
print("time until steady state: " + str(round(t/60/60,1)) + " h")
```

```
time until steady state: 473.7 h
```

0.4 References

BIBLIOGRAPHY

[1] ISO/TC 59/SC 13. Iso 10211:2017 thermal bridges in building construction — heat flows and surface temperatures — detailed calculations. International Organization for Standardization, 03 2016.

ISO/TC 59/SC 13. Iso 10211:2017 thermal bridges in building construction — heat flows and surface temperatures — detailed calculations. International Organization for Standardization, 03 2016.

[2] Carl-Eric Hagentoft. *Introduction to building physics*. Studentlitteratur, Lund, 2001. URL: https://buildingphysicshagentoft.com/text-books/introduction-to-building-physics/.

Carl-Eric Hagentoft. *Introduction to building physics*. Studentlitteratur, Lund, 2001. URL: https://buildingphysicshagentoft.com/text-books/introduction-to-building-physics/.

[3] E. Walther. *Building Physics Applications in Python*. DIY Spring, Paris, 2021.

E. Walther. *Building Physics Applications in Python*. DIY Spring, Paris, 2021.