Tutorial 2 - Heat exchanger

1 Introduction

Heat exchangers (HE) appear in numerous technical applications and industrial processes, e.g. air conditioning of cars as well as heating and cooling of process fluids in chemical engineering. Within the present tutorial you are going to write a program, which evaluates the temperature field in a double tube HE based on given inflow conditions (temperature, mass flow rate) and geometrical data. Both, parallel flow and counter flow HE are considered and the numerical results are compared to the analytical solutions. Although being one of the simplest configurations, the double tube configuration teaches us the basic concepts of a HE in a sufficient way. Based on this practical example, you will learn how to control the program flow in a *C++* code using loops and conditional statements. The goals of this tutorial are:

- understand principal concepts of heat exchangers
- understand principal concepts of **looping** in *C++*
- understand principal concepts of **conditional statements** in *C++*

2 Problem description

In the following a brief introduction to the basic concepts of HEs is given, with further details given in [1]. The naming convention in this tutorial reflects the nomenclature used in [1].

In a double tube HE a hot and a cold fluid stream are physically separated by a wall. The temperature difference between the two fluid streams induces heat transfer across the wall, such that the cold fluid is heated up, while the hot fluid is cooled down. The basic concept of this kind of HE is shown in Fig. 1. The heat flux through the wall is determined as

$$\dot{Q}_w = kA\theta_m \tag{1}$$

where k, A and θ_m are the heat transfer coefficient, the area of the wall and the mean driving temperature difference, respectively. Following the first law of thermodynamics the energy balances for the two fluid streams are

$$\dot{Q}_1 = \dot{m}_1 c_{p,1}(\vartheta_1' - \vartheta_1'') \tag{2}$$

and

$$\dot{Q}_2 = \dot{m}_2 c_{p,2}(\vartheta_2'' - \vartheta_2'), \tag{3}$$

where $\dot{m}$, c_p and ϑ are the mass flow rate, the specific heat capacity and the temperature, respectively. Note, that the heat flows are defined such that they are always positive, i.e. the difference between the higher and the lower temperature is used.

Table 1: Input data.

symbol	value	unit
Inflow conditions and fluid properties		
$\dot{m}_1$	1	kg/s
$\dot{m}_2$	2	kg/s
ϑ_1'	423.15	K
ϑ_2'	293.15	K
$c_{p,1}$	4305.335	J/(kg K)
$c_{p,2}$	4184.051	J/(kg K)
Heat transfer		
α_1	2000.0	W/(m^{2}K)
α_2	2000.0	W/(m^{2}K)
λ_{Al}	236.0	W/(m K)
Pipe geometry		
r_{in}	80	mm
r_{out}	82	mm
L	10	m
Pollution		
s_{pol}	2	mm
λ_{pol}	0.5	W/(m K)

The following conventions are applied throughout this tutorial:

1. The subscripts 1 and 2 represent the two separate fluid streams.
2. Fluid 1 is at a higher temperature than fluid 2.
3. The superscript $'$ represents the inflow conditions and superscript $''$ represents the outflow conditions of the respective fluid stream.
4. We define the "heat capacity flows" $\dot{W}_1 = \dot{m}_1 c_{p,1}$ and $\dot{W}_2 = \dot{m}_2 c_{p,2}$.

Additionally the following assumptions are made to simplify the analysis:

- The system is adiabatic with respect to its surroundings.
- The fluid properties are constant within each tube of the HE.
- The HE is in a steady-state and unsteady effects such as heating of walls during the initial stages are not considered.

The global heat balance is then

$$\dot{Q}_w = \dot{Q}_1 = \dot{Q}_2. \tag{4}$$

General task

The final goal of this tutorial is to determine the temperature field in a given double tube HE, where all initial and boundary conditions are known. The general tasks are:

- Determine the temperature field of a parallel flow HE.
- Determine the temperature field of a counter flow HE.
- Analyse the influence of pollution for the counter flow HE (optional).

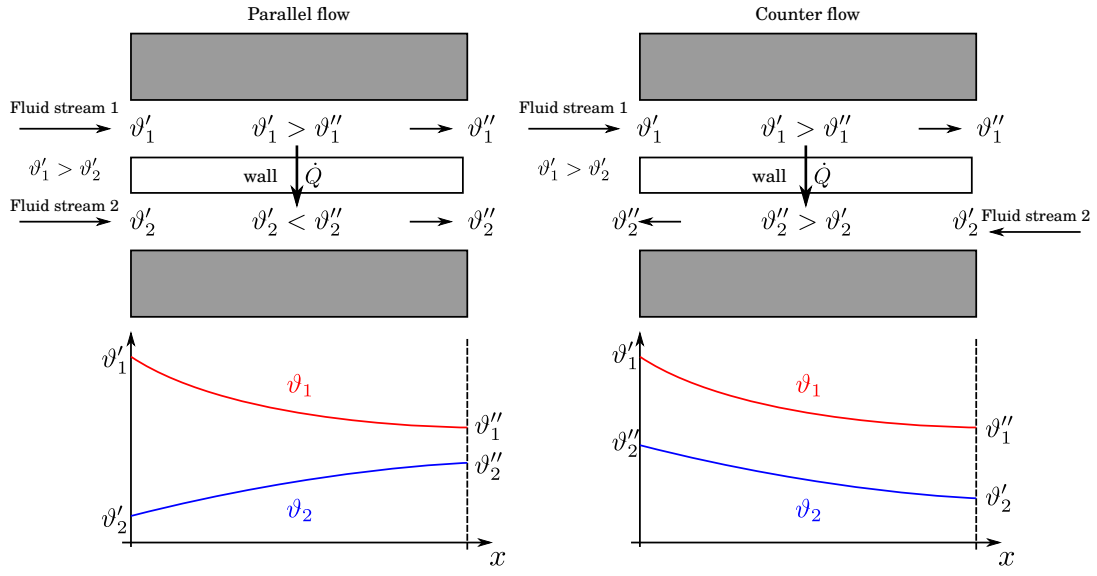

Figure 1: Schematic of heat exchangers (recuperator). A hot fluid 1 is separated from a cold fluid 2 by a wall. Due to the heat transfer through the wall, fluid 1 is cooled down, while fluid 2 is heated up.

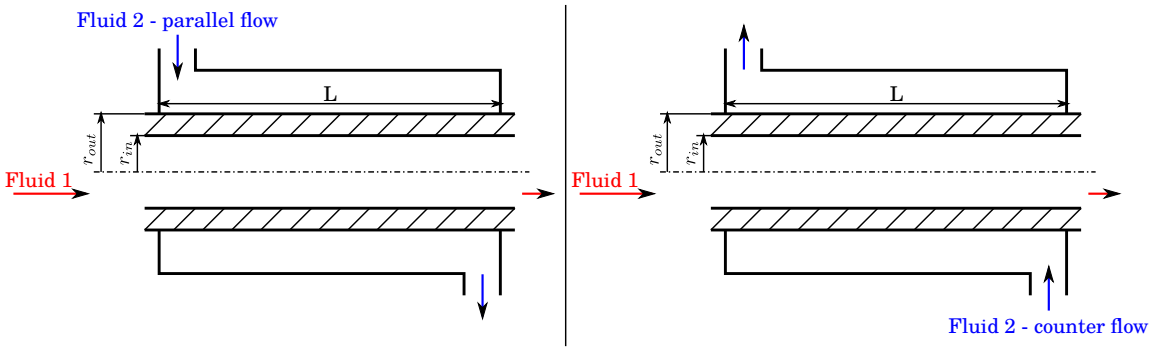

Figure 2: Geometrical setups for the double tube heat exchanger. Fluid 1 passes the inner tube from the left to the right. Fluid 2 passes the outer tube, while the direction depends on the flow mode. Left: Parallel flow HE. Right: Counter flow HE.

2.1 Input data

All necessary input data including mass flow rates, temperatures, dimensions of the inner pipe as well as heat transfer and heat diffusivity coefficients are given in Tab. 1. Both fluids are liquid water. Note that fluid 1 is at a higher pressure allowing for temperatures beyond 373.15 K (boiling point). The inner pipe is made of aluminium. The geometrical setup is depicted in Fig. 2. The parameters s_{pol} and λ_{pol} in Tab. 1 describe the thickness of a pollution layer and its heat diffusivity which is going to be considered in the optional part of the tutorial.

The reference area for the heat transfer calculation is the outer area of the inner pipe:

$$A_{\mathrm{ref}} = 2\pi r_{\mathrm{out}} L. \tag{5}$$

In this case, the heat transfer coefficient k is calculated by means of this equation

$$\frac{1}{k} = \frac{r_{\mathrm{out}}}{r_{\mathrm{in}}}\frac{1}{\alpha_{\mathrm{in}}} + r_{\mathrm{out}} \sum_j \frac{\ln(r_{\mathrm{out,j}}/r_{\mathrm{in,j}})}{\lambda_j} + \frac{1}{\alpha_{\mathrm{out}}}. \tag{6}$$

The summation over index j accounts for a wall consisting of multiple layers. In the optional task, the pollution on the pipe wall can be addressed by considering the pollution layer as an additional pipe wall layer. For all other (mandatory) tasks, the pipe wall consists of a single layer.

2.2 Numerical solution

The numerical solution for the temperature field requires a division of the heat exchanger into several sub-cells as it is indicated in Fig. 3. The global heat balance in Eq. (4) can be applied to each cell separately allowing for a stepwise solution of the temperature fields. With

$$\dot{W}_1(\vartheta_1' - \vartheta_1'') = kA\theta_m \tag{7}$$

and

$$\dot{W}_2(\vartheta_2'' - \vartheta_2') = kA\theta_m \tag{8}$$

we obtain

$$\vartheta_{1,i+1} = \vartheta_{1,i} - \frac{(kA)_i}{\dot{W}_1}\theta_{m,i} \tag{9}$$

and

$$\vartheta_{2,i+1} = \vartheta_{2,i} + \frac{(kA)_i}{\dot{W}_2}\theta_{m,i}. \tag{10}$$

Note: In the present tutorial an equidistant distribution of mesh cells is used and effects such as boundary layer development are neglected when estimating the heat transfer coefficients α_1 and α_2. The values of $(kA)_i$ in Eqs. (9) and (10) are therefore the same for each cell.

For the numerical solution we approximate

$$\theta_{m,i} = \vartheta_{1,i} - \vartheta_{2,i}. \tag{11}$$

ATTENTION: Since the temperature difference θ_m between fluid 1 and fluid 2 is a function of x, Eq. (11) is only valid, if the number of cells N is sufficiently high, i.e. the cell width Δx is sufficiently small.

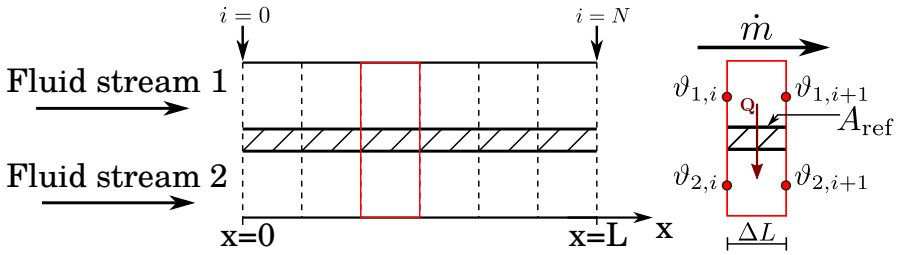

Figure 3: Sketch of the numerical setup with several cells.

3 C++ Concepts

Within the present tutorial you are going to learn the basics of **looping** and **conditional statements**, which are integral parts of the majority of computer codes in engineering. In the following we introduce C++ concepts for looping and conditional statements. A detailed description is found, e.g. in Chapters 4 and 6 in [2]. In section 3.4 a brief introduction to the use of *arrays* is given, since (the elements of) arrays are often addressed within loops. Details are found in Chapter 7 in [2]. The program examples in Listings 1-3 are made available for the tutorial in folder **CPP/TEST** and can be compiled using *g++*. In case you struggle employing any of the *C++* concepts introduced here and applied in the (larger) main tutorial code, revert to the simple codes in **TEST** and use them as a testbed.

3.1 Loops

"Many programming tasks are accomplished by doing the same thing either a fixed number of times or until a specific condition is met. A block of code that is executed more than once in a row in a program is called a *loop*. Each pass through the loop is called an *iteration*". [2]

Available loop types in *C++* are *while*-loops, *do-while*-loops and *for*-loops. An example for each loop is shown in Listing 1. In the *C++* while-loop, an execution condition is defined at the beginning of the loop and the operation(s) inside is (are) executed as long as the condition is true. In the example in Listing 1, the local position `xLoc` is increased by `dx` as long as it is smaller than 2.5. The *do-while*-loop works similarly but the execution condition is evaluated at the end of the loop. If you execute the code from Listing 1, you will notice that both options give the same number of iterations. The difference becomes important when either the execution condition is not met before the first iteration is started, or if a quantity used in the condition is not available at the beginning of the loop.

In the *for*-loop a fixed number of iterations is executed. In the example `xLoc` is determined `N` times. If `dx` is decreased, the operation within the *while* and the *do-while* loop will be executed more frequently. The operation in the *for* loop is still called `N` times, giving smaller values for `xLoc`.

Note: Every loop type can also be stopped at a certain iteration by using the command `break`. See also [2] for more details on this. However, breaking out of loops is generally not a recommended programming procedure.

3.2 Conditional statements

Conditional statements are used to decide between different operations or routines, or to decide whether one routine is called or not. One option is an *if*-condition. An example is shown in Listing 2 (top). If the condition defined within the brackets written behind `if` is true, the operation is executed. If it is not true, the code moves on to the next set of commands. One possible extension for an *if* statement is an *else if* statement. The condition in an *else if* is only evaluated if the previous *if* condition gave false. The *else* statement is called, if all previous conditions were false. An *if* condition can

- stand alone,
- be combined with an *else if* statement only
- be combined with an *else* statement only
- be combined with both, *else if* and *else*.

Another possibility to decide between different options (i.e. paths through the code) is to use the *switch-case* environment. Listing 2 (bottom) shows an example implementation. A routine is called depending on the value of a variable, here *modelNumber*. It is possible to define a default routine, which is called if the input variable does not match any of the predefined cases. This is particularly helpful for debugging your code.

3.3 Nesting

Loops and conditions can be nested in arbitrary combinations. If you nest one *for* loop in another *for* loop, within each iteration of the outer loop, all iterations of the inner loop are executed. Loops can be embedded into conditions and conditions can be embedded into loops. Note that the total number of iterations is the product of the iterations of each nested loop. For complex problems, this can result in considerable computational times.

3.4 Arrays

In the previous examples we have assigned one single value to a certain variable. E.g. the variable `TMax` in Listing 2 has the value 343.15. If we want to create an ordered list of variables of the same type we can us *arrays* (we will discuss possible alternatives in a later tutorial). An example for the use of differently sized arrays is found in Listing 3. In this example we have declared two different arrays. The first array `x` is one-dimensional, i.e. it is a simple list of elements. When declaring an array, we define the number of elements of the array. Array `x` in Listing 3 consists of `N` elements. For addressing a certain value of the array, we specify its index (ID number) in the square bracket.

Note: Different from many other programming languages, in C++ an array with N elements usually ranges from element 0 to element $N-1$. Thus, *for* loops are typically defined from $i = 0$ to $i < N$ ($\rightarrow i = N-1$).

```cpp
#include <iostream>
using namespace std;

//******************** Example program ********************//
int main()
{
    int         i;
    const int   N=10;
    double      dx,xLoc;

    dx=0.5;

    xLoc=0.0;
    while(xLoc < 2.5)                   // Define the condition for execution
    {                                   // Open loop
        xLoc+=dx;                       // Calculate x for each iteration
        cout << "x=" << xLoc << endl;   // Write the current xLoc
    }                                   // Close loop

    xLoc=0.0;
    do
    {                                   // Open loop
        xLoc+=dx;                       // Calculate x for each iteration
        cout << "x=" << xLoc << endl;   // Write the current xLoc
    }                                   // Close loop
    while(xLoc < 2.5);                  // Define the condition for execution

                                        // Define number of iterations:
    for(i=0 ; i<N ; ++i)                // Start at i=0, finish at i=N-1,
                                        // advance with i=i+1
    {                                   // Open loop
        xLoc=double(i)*dx;              // Calculate x for each iteration
        cout << "x=" << xLoc << endl;   // Write the result
    }                                   // Close loop

    return 0;
}
```

Listing 1: Basic loop syntax

```cpp
#include <iostream>
using namespace std;

//******************** Example program ********************//
int main()
{
    int     modelNumber;
    double TLoc,TMin,TMax;

    TLoc=323.15;
    TMin=303.15;
    TMax=343.15;

    if(TLoc < TMin)                          // Define condition for execution
    {                                        // Open conditional statement
        TLoc = TMin;                         // Reset TLoc
    }                                        // Close the condition
    else if(TLoc > TMax)                     // Define second condition
    {                                        // Open conditional statement
        TLoc = TMax;                         // Reset TLoc
    }                                        // Close the condition
    else                                     // For any other case:
    {                                        // Open conditional statement
        cout << "TLoc not changed" << endl;  // Show output
    }                                        // Close the condition

    modelNumber = 1;
    switch (modelNumber)                     // Set the switch variable
    {                                        // Start switch cases
        case 1:                              // Model 1: ABC
            cout << "Use model ABC." << endl;
            break;
        case 2:                              // Model 2: DEF
            cout << "Use model DEF." << endl;
            break;
        default:                             // Default, if no case matches
            cout << "Attention: No model with number "
                 << modelNumber
                 << " implemented." << endl;
    }                                        // End switch cases

    return 0;
}
```

Listing 2: Basic syntax of conditional statements

```cpp
#include <iostream>
using namespace std;

//******************** Example program ********************//
int main()
{
    int          i,j;
    const int    N=3;
    const int    M=3;
    double       dx;        // Single element
    double       x[N];      // 1D array containing N elements   (e.g. a vector)
    double       T[N][M];   // 2D array containing N*M elements (e.g. a matrix)

    dx = 0.1;

    for(i=0 ; i<N ; ++i)
    {
        x[i] = i*dx;      // Write the i-th element of array x
    }

    for(i=0 ; i<N ; ++i)
    {
        for(j=0 ; j<M ; ++j)
        {
            // Write the element identified, by i and j to the array T
            T[i][j] = double(i)*1.0 + double(j)*2.0 + 373.15;
        }
    }

    // Write one element of the array x
    cout << "First element of array x=" << x[0]         << endl;

    // Write one element of the array T
    cout << "Last  element of array T=" << T[N-1][M-1] << endl;

    return 0;
}
```

Listing 3: Basic array syntax

4 Tasks

Tasks:
Part 1 - Parallel flow:

a) Familiarise yourself with the given code structure starting from **main.cpp**. Get an overview on the integral parts of the code by answering the following questions: What are the major routines? What is their purpose? Where are they called from? Where are they implemented? Additionally, point out the significance of the file **declaration.h**.

b) At TODO 1 (in **init.h**) add the input data to the code. Consult Tab. 1 for the correct input values.

c) Considering Eqs. (9) and (10), which other quantities are necessary to obtain the temperature fields in the heat exchanger? How do we need to initialise the temperature field for the solution procedure? Which other quantity is necessary to post-process (i.e. to visualize) the temperature field properly? Implement the calculation of the necessary quantities at TODO 2 (in **init.h**). Hint: If you need a *for*-loop, you find an example in the file **IO.h**.

d) At TODO 3 (in **calcHE.h**) implement Eqs. (9) and (10). Use a *for*-loop to determine the temperatures of both fluid streams for each cell.

e) Compile and run the code using the script **scriptRunAll.sh**. Compare the resulting temperature fields with the analytical solution. Which role does the number of cells play in the solution procedure?

Part 2 - Counter flow:

a) At TODO 4 (in **main.cpp**) implement a conditional statement for switching between parallel flow and counter flow. Use an *if*-condition and introduce a new variable `flowMode` providing the information of the heat exchanger type. The variable `flowMode` can be declared in **declaration.h**. The function for solving the temperature field in the counter flow configuration has the name `solveTFieldCountercurrent` (available but initially commented out in **calcHE.h**).

b) How does the solution procedure change, if we switch to counter flow? Which problem is associated with the counter flow configuration and how can it be solved? Implement Eqs. (9) and (10) for the counter flow configuration at TODO 5 (in **calcHE.h**). *Hint 1:* You might have to apply a *do-while*-loop for properly solving the equations. *Hint 2:* If you have one index i counting from the first mesh cell to the last mesh cell, you can calculate a second index j counting from the last cell to the first cell at the same time.

c) Which consequences does the modified flow configuration have for the initialisation? At TODO 6a (in **init.h**) implement the required modification, by keeping the option to switch between parallel and counter flow. At TODO 6b in **IO.h** change the name of the output file to **thetaOfXCounter.dat**. Again, use an *if* condition to decide between the two HE types.

d) Run the code and compare the resulting temperature fields with the analytical solution. Which differences occur? Can you explain them?

Optional: Part 3 - Pollution

a) Pollution may have a significant effect on the efficiency of a heat exchanger. Assume pollution layers of 2 mm on each side of the tube for the counter flow mode. At TODO 7 (in **init.h**) introduce a new variable `pollMode` providing the information if pollution is considered or not. Which quantities change with pollution? Use the *switch-case* functionality of C++ to decide between the two modes.

b) Run the code and compare the temperature fields with the corresponding results that neglect pollution. What do you observe?

5 Analysis (optional)

Prepare a detailed documentation of your work including answers to all questions in Sec. 4. Include screen shots showing code snippets of your own implementations. In particular, address the following questions:

1) Which differences can be seen in the resulting temperature fields for parallel and counter flow heat exchangers?

2) What is the major difference in the numerical treatment of the counter flow HE compared to the parallel flow HE? How did you change the solution procedure?

3) Why is the rather simple configuration we use probably not suitable for most cases of practical interest? Is there any other quantity of interest besides the heat transfer rate/temperature field?

4) Considering the call of the different functions and the declaration of variables: Which improvements could be made in terms of code implementation?

References

[1] VDI-Wärmeatlas: Berechnungsunterlagen für Druckverlust, Wärme- und Stoffübertragung, 10., bearb. und erw. Auflage, Springer Berlin, Heidelberg, 2006.

[2] Jesse Liberty, Rogers Codenhead: Sams Teach Yourself C++ in 24 Hours, Pearson Education, Inc., 2011.