LAB REPORT: LAB 3-4
Heat transfer simulation
MODELING OF PHYSICAL SYSTEMS

Patryk Gaczyski

Tuesday 27th March, 2018

1 Aim

The aim of this laboratory was to simulate, visualize and analyze heat transfer process in three different materials - copper, stainless steel and aluminum with two methods.

2 Methods

For the laboratory purpose various tools and methods were selected to achieve good simulation results and easy in depth analysis. Simulation model has been tested against its theoretical model, its numerical stability as well as state stability after time.

2.1 Simulation description

This simulation considers metal plate with size of 25 centimeter by 25 centimeter which is being heated on highlighted area figure 3.6 (which is 5 centimeters by 5 centimeters) in couple different ways. Heater area is square shaped and placed in center of the plate.

- Situation 1 - heater area (the red one) is having constant temperature during whole simulation equal to 80^o of Celsius. Outer edge has fixed temperature as well and is equal to 10^o of Celsius all the time. This situation assumes that plate is infinitely thin.
- Situation 2 - heater area provides constant amount of energy - 100 Wats for first 10 seconds. It is assumed that outer edge is thermally isolated from environment, that means it does not exchanges energy. This time, plate does have finite thinness.

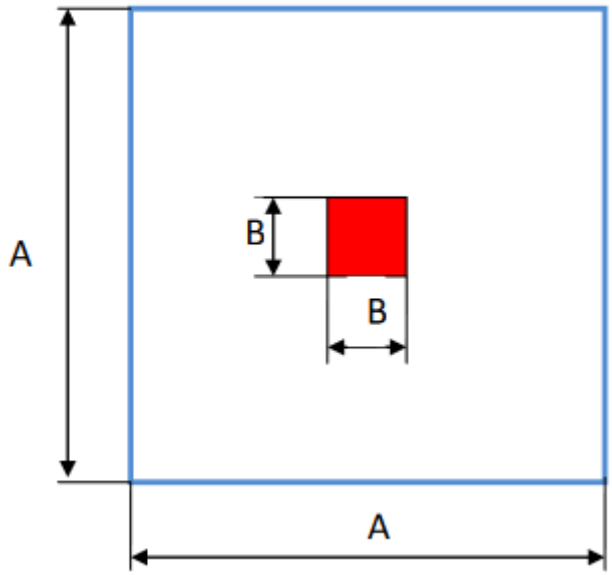

Figure 2.1: Metal plate that is being heated

2.2 Software

To simulate, compute and visually represent heat transfer process MATLAB software was involved.

3 Results

3.1 Simulation of heat transfer phenomena code

Following code has been used to simulate and visually represent heat transfer in two described situations. First part, which is declaration of constant values, stays the same for both situations:

Listing 1: initial values for matlab scripts

```
% time variables
number_of_steps = 100;          % num
dt = 0.1;                       % s
dx = 0.01;                      % m
dy = 0.01;                      % m


```

```
9  % metal variables
10 K = 237;                                % W/mK
11 cw = 900;                               % J/kgK
12 ro = 2700;                              % kg/m3
13
14 % plane variables
15 outer_size = 25;                        % cm
16 inner_size = 5;                         % cm
17 % only these differs for different situations
18 outer_temp = 10;                        % C
19 inner_temp = 80;                        % C
20 small_start = (outer_size - inner_size)/2;
```

In first situation (boundary condition 1), temperature of heater area and outer boarder has to be fixed, which is done in lines 2 and 3 for initial condition and in lines 18 to 22 for every timestamp. Temperature on every point belonging to plane is calculated with formula provided in lab instruction. At the very end surface plot of last calculated state is being used to visualize final result.

Listing 2: matlab script implementing first boundary condition

```
1  % assign initial plate state
2  plane(1:outer_size,1:outer_size,1) = outer_temp;
3  plane(small_start:small_start+inner_size,small_start:
       small_start+inner_size, 1) = inner_temp;
4
5  % iterate over time
6  for i = 2:number_of_steps
7      % iterate over plate points
8      for x = 2:outer_size-1
9          for y = 2:outer_size-1
10             % change plane state according to given
                   formula
11             plane(x,y,i) = ...
12                 plane(x,y,i-1) + ...
13                 (K*dt)*(plane(x+1,y,i-1) - 2*plane(x,y,i
                       -1) + plane(x-1,y,i-1))/(cw*ro*dx*dx) +
                       ...
14                 (K*dt)*(plane(x,y+1,i-1) - 2*plane(x,y,i
                       -1) + plane(x,y-1,i-1))/(cw*ro*dy*dy);
15         end
16     end
17     % restore initial conditions at heater area and outer
           edge
```

```
18      plane(small_start:small_start+inner_size,small_start:
            small_start+inner_size, i) = inner_temp;
19      plane(1,:,i) = outer_temp;
20      plane(:,1,i) = outer_temp;
21      plane(outer_size,:,i) = outer_temp;
22      plane(:,outer_size,i) = outer_temp;
23
24  end
25
26  % plot awesome graph
27  [XX, YY] = meshgrid(1:outer_size,1:outer_size);
28  surf(XX,YY,plane(:,:,number_of_steps));
```

Result of this script execution is the following chart

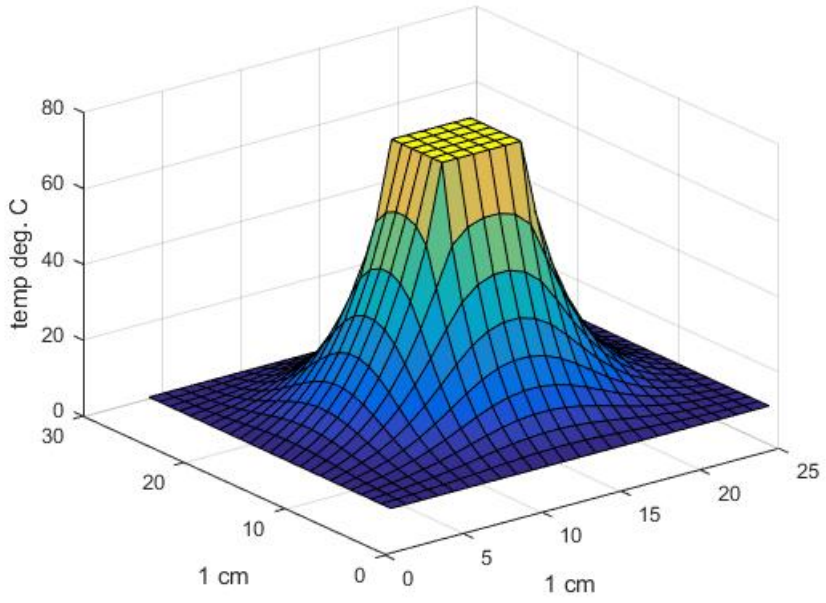

Figure 3.2: Heat transfer for first boundary condition using copper plate after certain number of time steps

There are several conclusions to be made. First of all, it is clearly visible that the hottest part is always in place of heater. Secondly, area near the heater is propagating energy to its neighbors, which results in warmer area then in beginning of the simulation.

Code for second situation is mostly the same as for first one, however different parts are:

- $inner\_temp = outer\_temp = 20^oC$
- in line 12 is if statement checking whether heater should be in on state.
- surface plot changes over time to show how temperature of plane is changing
- $heating\_power$ is equal to $100W$

Listing 3: matlab script implementing second boundary condition

```
1
2  plane(1:outer_size,1:outer_size,1) = outer_temp;
3  plane(small_start:small_start+inner_size,small_start:
       small_start+inner_size, 1) = inner_temp;
4
5  % loop over loop
6  for i = 2:number_of_steps
7      % iterate over plate points
8      for x = 2:outer_size-1
9          for y = 2:outer_size-1
10
11             % for first 10s transfer heat designated area
12             if i * dt < t && ...
13                x >= small_start && ...
14                x <= small_start + inner_size && ...
15                y >= small_start && ...
16                y <= small_start + inner_size
17                 plane(x,y,i) = plane(x, y, i-1) + ...
18                                (heating_power * dt) /
                                      ...
19                                (cw * inner_size *
                                   inner_size *
                                   thickness * ro);
20             else
21                 % and propagate heat
22                 plane(x,y,i) = ...
23                     plane(x,y,i-1) + ...
24                     (K*dt)*(plane(x+1,y,i-1) - 2*plane(x,y
                           ,i-1) + plane(x-1,y,i-1))/(cw*ro*dx
                           *dx) + ...
25
26
```

```
                    (K*dt)*(plane(x,y+1,i-1) - 2*plane(x,y
                        ,i-1) + plane(x,y-1,i-1))/(cw*ro*dy
                        *dy);
                end
            end
        end
        [XX, YY] = meshgrid(1:outer_size,1:outer_size);
        % plot as surface
        surf(XX,YY,plane(:,:,i));
        title(strcat('Simulation after ', num2str(i*dt), 's'))
            ;
        zlim([20, 30]);
        % draw as animation
        drawnow;
        % with 0.1s frame rate
        pause(0.1);
end
```

Below three slices of continuous chart are being presented

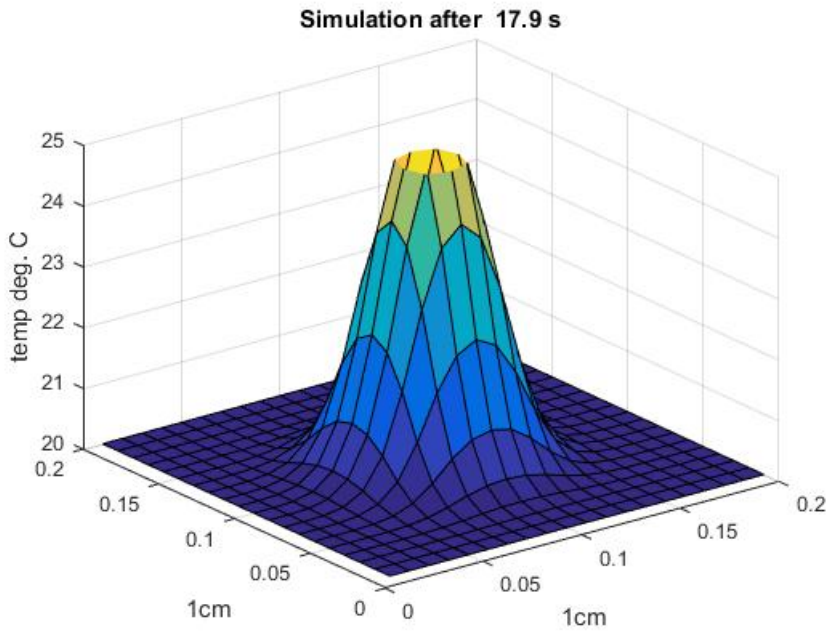

Figure 3.3: Heat transfer for second boundary condition using copper plate after time

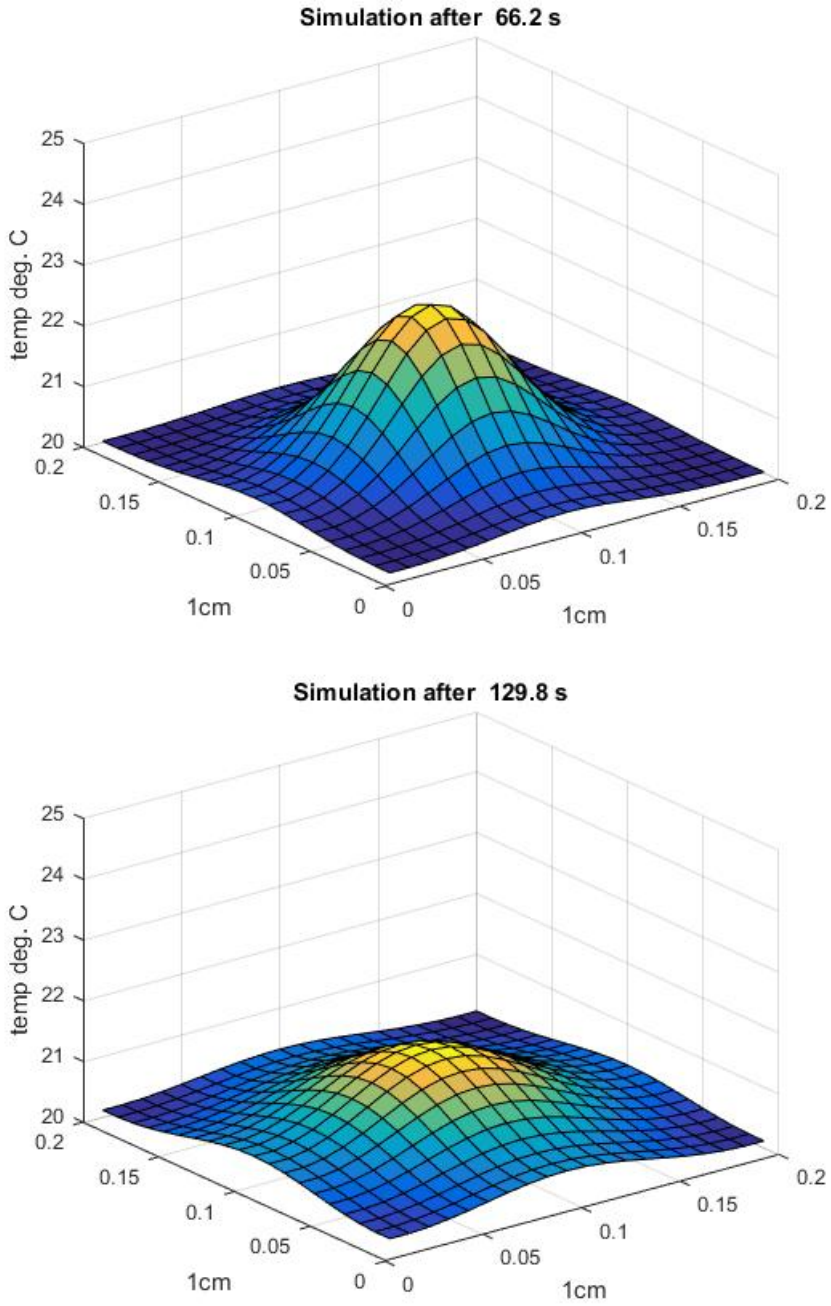

Figure 3.4: Heat transfer for second boundary condition using copper plate after time

In both situations rapid temperature increase is observable nearby heater area. However in the second situation, much more dissipation can be observed. After running such script against copper and alumina, it resolves that copper plate

gives heat off much faster than alumina.

3.2 Testing numerical stability

System is numerically stable if small change in system property is resulting in small change to system behavior. To test against numerical stability given model, *binary-search-ish* algorithm is being used. Following code describes how it has been achieved.

Listing 4: finding numerical stability point

```
% test for different dt values from 0.1 to 1.0
for i=0.1:0.1:1.0
    dt = i
    figure;
    % run simulation code
    tests;
    title(['dt = ' num2str(dt)]);
end
```

To find when system is starting to be unstable, output chart is being visually tested for evident oscillations. First iteration of this code gave result that oscillations are starting for dt between 0.2 and 0.3. To get more precise result, same code can be run with different iterator definition:

Listing 5: iteration example

```
for i=0.2:0.01:0.3
    ...
```

and so on. After couple iterations, results came up as follows:

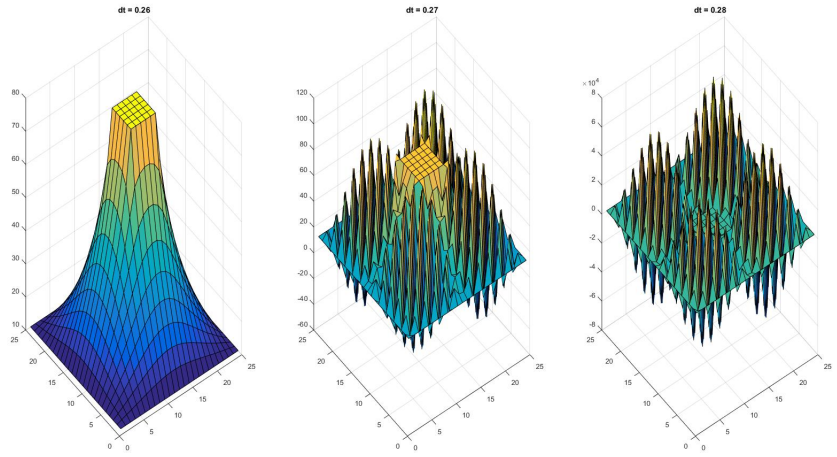

Figure 3.5: osculations for dt between 0.2s and 0.3s

Where it is clearly visible that system is loosing its stability somewhere between 0.26s and 0.28s for dt values. Same thing can be done for spatial resolutions and the results came up as follows

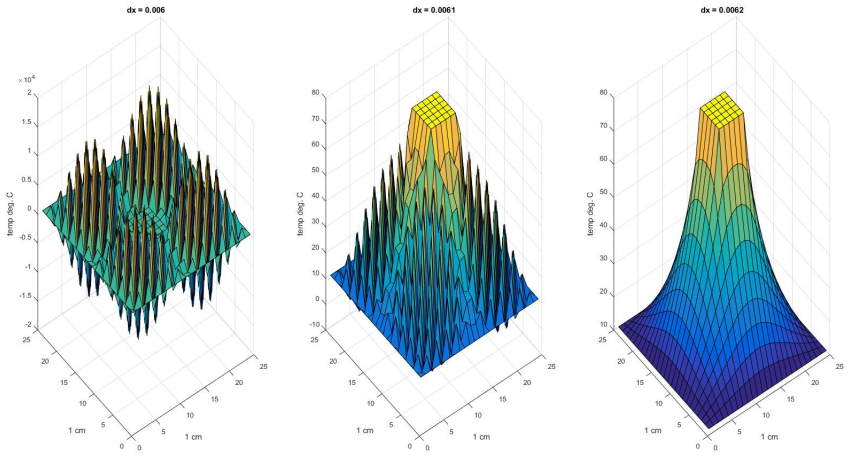

Figure 3.6: osculations for dx between 0.0060m and 0.0062m

System is starting to osculate for dx value between 0.0060 and 0,0062 m.

3.3 Steady state calculation

According to *steady state* definition, system is in steady state when its properties are unchanging in time. In this case temperature change between time steps

should be taken into consideration. Summing up, criterion for steady state of this system would be

$$max(abs(system\_state\_matrix(i) - system\_state\_matrix(i-1))) = 0 \quad (1)$$

. However, because floating point numbers comparison is not so precise when it comes to programming languages, it is safe to assume that this equation shall be written like this:

$$max(abs(system\_state\_matrix(i) - system\_state\_matrix(i-1))) < \epsilon \quad (2)$$

Where epsilon is some fixed value that is suitable in this case, i.e. 0.001^oC. Given that formula, and adding it to code from listing 3, it is testable when system reaches steady state:

Listing 6: matlab snippet to find whether system is stable yet

```
1  if (max(max(abs(plane(:,:,i) - plane(:,:,i-1))) <
       0.001)) && (i * dt > 10)
2      disp(i);
3      break;
```

is resulting in i = 260, which corresponds to 26.0 seconds for copper and i = 1302, which is 130.2 seconds for alumina. Such result can be explained by very small ϵ value, which is fine in this case.

3.4 Theory model vs simulation model

This section purpose is to compare simulated temperature delta to calculated from stated formula:

$$\Delta T_T = \frac{P \cdot t_{heat}}{cw \cdot A^2 \cdot h \cdot \rho} = \frac{100W \cdot 10s}{900\frac{J}{kgK} \cdot 0.25m^2 \cdot 0.002m \cdot 2700\frac{kg}{m3}} \approx 3.2922K \quad (3)$$

To get ΔT from simulation model, following code can be executed

Listing 7: Calculation of delta T in simulated heat transfer model for boundary condition 2

```
1  >> max(max(max(plane))) - start_temp
2  ans =
3
4      8.0658
```

ΔT for theoretical model and simulation model is very different from each other. That means, that either equation is inaccurate (which is VERY unlikely) or the simulation parameters are a bit messed up (which is most probable). However, it is still same order of magnitude. That being said, it can be conclude that this result is acceptable.

4 Conclusion

In this report heat transfer and its properties was covered. Various matlab scripts were written to illustrate heat transfer itself, its relation to theoretical model and system stabilization property. This lab showed us that heat transfer phenomena can be simulated fairly easy however its accuracy in this implementation is something to think about. It is also worth mentioning that simulation of different materials is reasonably simple to achieve.