Technion - Israel Institute of Technology

Numerical Methods in Aeronautical Engineering (086172)

GRADE	OUT OF	CHAPTER
	2	ABSTRACT
	2	CONTENTS, STYLE &C.
	4	PHYSICAL PROBLEM
	4	MATHEMATICAL MODEL
	26	NUMERICAL METHODS
	20	INFLUENCE OF NUMERICAL METHODS
	20	RESULTS
	2	SUMMARY & CONCLUSIONS
	20	COMPUTER PROGRAM
	100	**TOTAL**

Daniel Engelsman, ID 300546173 @ June 16, 2019

= Homework Assignment 3 =

Contents

List of Figures

1 Abstract

Heat transfer is a discipline of thermal engineering that analyzes the generation, use, conversion and exchange of thermal energy between physical systems. In our assignment we will examine 4 cases of different parameters ($\alpha,\ K_{10},\ K_{20}$) and their influence on the heat diffusion, and look for some sensitive factors that impact most.

2 The physical problem

In this report are asked to solve a steady-state heat transfer problem, and provide the heat distribution in a 2-D shape plate. The lower (**AB**) and upper (**ED**) sides are subjected to a zero-derivative condition, that can be translated into <u>insulation</u> (temp. $\sim$ const.). The left side (**AE**) equals **1**, and the right sides (**BC, CD**) equal **0** :

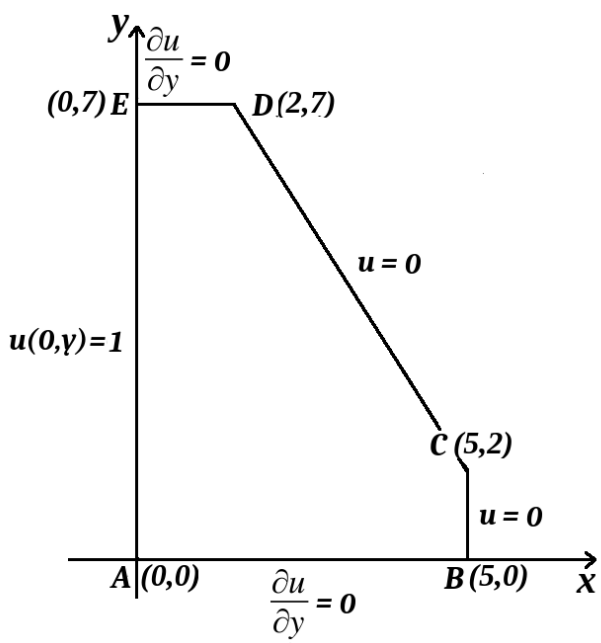


Figure 1: Polygon plate subjected to heat transfer

3 The mathematical model

We'll present the formal approach to the data given in the question :

$$\text{PDE}: \quad \frac{\partial}{\partial x}\Big(K_1(u)\cdot\frac{\partial u}{\partial x}\Big)+\frac{\partial}{\partial y}\Big(K_2(u)\cdot\frac{\partial u}{\partial y}\Big)=0, \quad y'=\begin{cases}[0,\ 7], & 0\le x\le 2\\ \frac{31-5x}{3}, & 2<x<5\\ [0,\ 2], & x=5\end{cases} \tag{3.1}$$

$$\text{Coefficients}: \quad K_1=K_{10}(1+\alpha u), \quad K_2=K_{20}(1+\alpha u) \tag{3.2}$$

$$\text{derivatives}: \quad \frac{\partial K_1}{\partial x}=\alpha K_{10}\cdot\frac{\partial u}{\partial x}, \qquad \frac{\partial K_2}{\partial x}=\alpha K_{20}\cdot\frac{\partial u}{\partial x}$$

$$\text{b.c}: \quad u(0,y)=1, \quad u_{BC}=u_{CD}=0, \quad u_{AB}=u_{DE}=\frac{\partial u}{\partial y}=0 \tag{3.3}$$

Elaborating **Eq. 3.1** we'll get :

$$\Big(\frac{\partial K_1}{\partial x}\cdot\frac{\partial u}{\partial x}+K_1(u)\cdot\frac{\partial^2 u}{\partial x^2}\Big)+\Big(\frac{\partial K_2}{\partial y}\cdot\frac{\partial u}{\partial y}+K_2(u)\cdot\frac{\partial^2 u}{\partial y^2}\Big)=0$$

$$\Big(\alpha K_{10}\cdot(\frac{\partial u}{\partial x})^2+\Big(K_{10}(1+\alpha u)\Big)\cdot\frac{\partial^2 u}{\partial x^2}\Big)+\Big(\alpha K_{20}\cdot(\frac{\partial u}{\partial y})^2+\Big(K_{20}(1+\alpha u)\Big)\cdot\frac{\partial^2 u}{\partial y^2}\Big)=0$$

We got an elliptic PDE (*Laplace*), conventionally presented :

$$\Big(K_{10}(1+\alpha u)\Big)\frac{\partial^2 u}{\partial x^2}+\Big(K_{20}(1+\alpha u)\Big)\frac{\partial^2 u}{\partial y^2}+\alpha K_{10}(\frac{\partial u}{\partial x})^2+\alpha K_{20}(\frac{\partial u}{\partial y})^2=0 \tag{3.4}$$

The PDE classification is dependent on the coefficients values. Additionally, it is non-linear and thus requires iterative methods as learned in class.

4 The numerical method

Analyzing the problem on the $u_{i,j}=u(x_0+ih,y_0+jp)$ plane, such that indices are calculated due to the following sub-intervals sizes :

$$i=0:dx:\frac{5.0}{dx}, \quad \mathbf{h}\equiv dx, \qquad j=0:dy:\frac{y'}{dy}, \quad \mathbf{p}\equiv dy \tag{4.1}$$

We'll do the following discretization, both at central F.D :

$$\frac{\partial u}{\partial x} = \frac{u_{i+1,j} - u_{i-1,j}}{2h} + O(h^2), \qquad \frac{\partial^2 u}{\partial x^2} = \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2} + O(h^2) \tag{4.2}$$

$$\frac{\partial u}{\partial y} = \frac{u_{i,j+1} - u_{i,j-1}}{2p} + O(p^2), \qquad \frac{\partial^2 u}{\partial y^2} = \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{p^2} + O(p^2) \tag{4.3}$$

Plug inside our PDE (**Eq. 3.4**) :

$$\begin{aligned}
\rightarrow\ & \Big(K_{10}(1+\alpha u_{i,j}) \cdot \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{h^2}\Big) + \Big(K_{20}(1+\alpha u_{i,j}) \cdot \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{p^2}\Big) + \dots \\
& \dots + \alpha K_{10}(\frac{u_{i+1,j} - u_{i-1,j}}{2h})^2 + \alpha K_{20}(\frac{u_{i,j+1} - u_{i,j-1}}{2p})^2 = 0 \\
\rightarrow\ & (1+\alpha u_{i,j}) \cdot \left(\frac{K_{10}}{h^2}(u_{i+1,j} + u_{i-1,j}) + \frac{K_{20}}{p^2}(u_{i,j+1} + u_{i,j-1}) - 2u_{i,j}\Big[\frac{K_{10}}{h^2} + \frac{K_{20}}{p^2}\Big]\right) = \\
& -\alpha K_{10}(\frac{u_{i+1,j} - u_{i-1,j}}{2h})^2 - \alpha K_{20}(\frac{u_{i,j+1} - u_{i,j-1}}{2p})^2 \\
\rightarrow\ & -2u_{i,j}(1+\alpha u_{i,j})\Big[\frac{K_{10}}{h^2} + \frac{K_{20}}{p^2}\Big] = -(1+\alpha u_{i,j})\left(\frac{K_{10}}{h^2}(u_{i+1,j} + u_{i-1,j}) + \frac{K_{20}}{p^2}(u_{i,j+1} + u_{i,j-1})\right) \\
& -\frac{\alpha}{4}\left(\frac{K_{10}}{h^2}(u_{i+1,j} - u_{i-1,j})^2 + \frac{K_{20}}{p^2}(u_{i,j+1} - u_{i,j-1})^2\right) \\
\Rightarrow\ & u_{i,j} = \frac{1}{2}\Big[\frac{K_{10}}{h^2} + \frac{K_{20}}{p^2}\Big]^{-1}\Bigg[\left(\frac{K_{10}}{h^2}(u_{i+1,j} + u_{i-1,j}) + \frac{K_{20}}{p^2}(u_{i,j+1} + u_{i,j-1})\right) \\
& + \frac{\alpha}{4(1+\alpha u_{i,j})}\left(\frac{K_{10}}{h^2}(u_{i+1,j} - u_{i-1,j})^2 + \frac{K_{20}}{p^2}(u_{i,j+1} - u_{i,j-1})^2\right)\Bigg]
\end{aligned}$$

Since the problem is nonlinear one has to use an iterative method to solve the set of the algebraic equations that replace the original elliptic PDE. To that end, we shall use the SOR (Successive Over-Relaxation) method, based on Gauss-Seidel method (Section 5.4) :

$$L_* \mathbf{u}_{GS}^{(n+1)} = \mathbf{b} - U\mathbf{u}^{(n)} \quad \rightarrow \quad u_{i,GS}^{(n+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}u_j^{(n+1)} - \sum_{j=i+1}^{n} a_{ij}u_j^{(n)}\right) \tag{4.4}$$

Implement in SOR : $\quad \mathbf{u}_{SOR}^{(n+1)} = u_i^{(n)} + \Omega(u_{i,GS}^{(n+1)} - u_i^{(n)})$

$$\text{Such that :} \quad \mathbf{u}_{i,SOR}^{(n+1)} = (1-\Omega)u_i^{(n)} + \frac{\Omega}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}u_j^{(n+1)} - \sum_{j=i+1}^{n} a_{ij}u_j^{(n)}\right) \tag{4.5}$$

Defining auxiliary terms to shorten the notation :

$$k_1 \equiv \frac{K_{10}}{h^2}, \qquad k_2 \equiv \frac{K_{20}}{p^2}, \qquad X_{i,j}^{\pm} \equiv (u_{i+1,j}^n \pm u_{i-1,j}^{n+1}), \qquad Y_{i,j}^{\pm} \equiv (u_{i,j+1}^n \pm u_{i,j-1}^{n+1})$$

And we get the following iterative method :

$$\mathbf{u}_{i,j}^{(n+1)} = \frac{\left(k_1 X_{i,j}^{+} + k_2 Y_{i,j}^{+}\right) + \left(k_1 (X_{i,j}^{-})^2 + k_2 (Y_{i,j}^{-})^2\right) \frac{\alpha}{4}(1 + \alpha \mathbf{u}_{i,j}^{(n)})^{-1}}{2\left[k_1 + k_2\right]} \tag{4.6}$$

Now we shall translate the boundary conditions to a numerical properties :

$$u(0,y) = u_{0,j} = 1, \qquad u(5,y) = u_{N+1,j} = 0, \qquad u(2 < x < 5,\ y_{CD}) = u_{i',j} = 0$$

$$\left.\frac{\partial u}{\partial y}\right|_{AB} = \left.\frac{\partial u}{\partial y}\right|_{0\le x\le 5}^{y=0} = 0 \ \rightarrow \quad \frac{u_{i,2} - u_{i,0}}{2p} = 0 \ \rightarrow \quad \underline{\underline{u_{i,0}}} = u_{i,2}$$

$$\left.\frac{\partial u}{\partial y}\right|_{DE} = \left.\frac{\partial u}{\partial y}\right|_{0\le x\le 2}^{y=7} = 0 \ \rightarrow \ \frac{u_{i,N+2} - u_{i,N}}{2p} = 0 \ \rightarrow \ \underline{\underline{u_{i,N+2}}} = u_{i,N}$$

Solution will be obtained by integrating the y axis along each single dx, where y(x) dictates the boundary edges subjected to the polygon shape. Note that each iteration calculates the <u>whole</u> x-y plane matrix, and not just a single solution vector. As seen below, convergence is obtained along iteration axis, until reaching desired criterion.

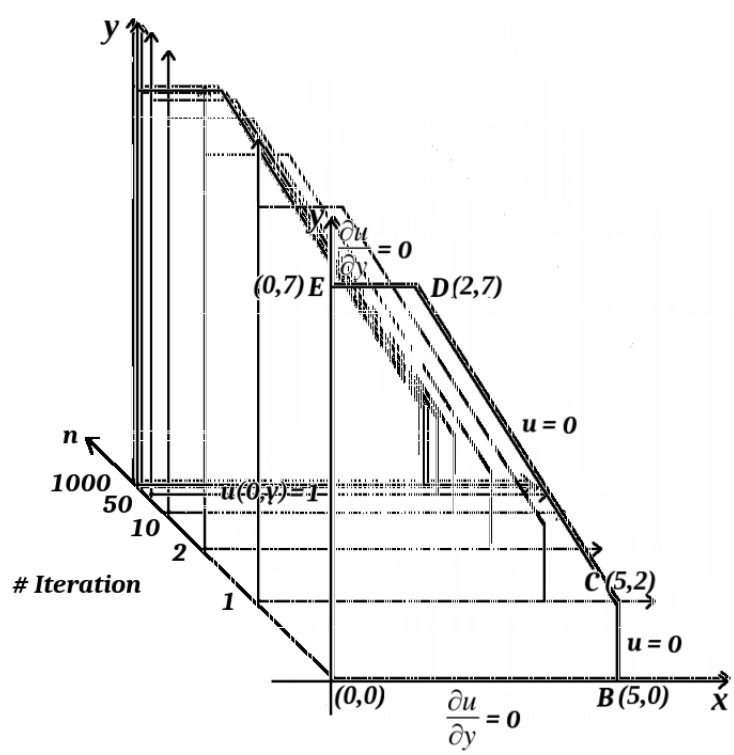


Figure 2: Tentative illustration of the solution approach

5 Influence of the numerical methods

Comprehensive examination of the influence of the the chosen method is a tough task for the high number of degrees of freedom. The best we can do is to take a "frozen" study case with constant parameters, and commence an optimization process in sought of optimality.

Along the process I marked (*pale-yellow*) each chosen value, such that it will be implemented in the next examined criterion.

5.1 Convergence criterion

In the beginning of the course we've seen 3 types of common numerical errors :

$$\text{Absolute error :} \qquad \left| u_{i,j}^{(n+1)} - u_{i,j}^{(n)} \right| < \epsilon \tag{5.1}$$

$$\text{Relative error :} \qquad \left| 1 - \frac{u_{i,j}^{(n)}}{u_{i,j}^{(n+1)}} \right| < \epsilon \tag{5.2}$$

$$\text{RMSE :} \qquad \frac{1}{N}\sqrt{\sum_{n=1}^{N}(u_{i,j}^{(n+1)} - u_{i,j}^{(n)})^2} < \epsilon \tag{5.3}$$

	Iterations per #method			
ϵ	I	II	III	min/max :
1.00E-03	2512	328	99	0.03941
1.00E-04	3314	987	704	0.21243
1.00E-05	4116	1774	1506	0.36589
1.00E-06	4918	2575	2308	0.46930
1.00E-07	5721	3376	3110	0.54361

Figure 3: Iterations number vs. convergence criterion

The different errors expressions perform differently in means of required number of iterations, and thus elapsed time. From right we can see the dramatic difference between the third and the first methods. So from here on out, we shall work with the RMSE criterion (III), and a sufficient convergence threshold of $\epsilon = 10^{-4}$.

5.2 Grid's step size

Now we shall check the grid's step sizes influence upon convergence, and the elapsed time to fulfill that requirement. For the sake of convenience, I set steps similarly ($\Delta x = \Delta y$) :

Step size	Iterations	Time [s]	Rate [itr/s]
0.1667	501	0.6525	767.8161
0.1250	704	1.6791	419.2722
0.1000	867	3.4250	253.1387
0.0833	982	5.5616	176.5679
0.0667	1068	9.3391	114.3579
0.0500	1074	16.8505	63.7370
0.0400	998	25.2550	39.5169
0.0286	814	39.9696	20.3655
0.0200	684	54.2031	12.6192

Figure 4: Mesh size influence upon convergence rate

Unsurprisingly, smaller steps ($\Delta x/y \downarrow$) take more time ($t \uparrow$), and more iteration are needed. But, from a certain magnitude ($\Delta x/y < 0.05$), less iterations are required, but indeed taken more time. From this point, we shall set step size to be $\Delta x/y = 0.0833$.

5.3 Initial Guess

In order to "trigger" the iterative method we should first off initialize the matrix entries, regardless the given initial and boundary conditions. Let us examine the development of the converged solution as function of the initial guess matrix. **Left** below, is a matrix initialized with positive values, and from **right** is initialized with same value, but negative :

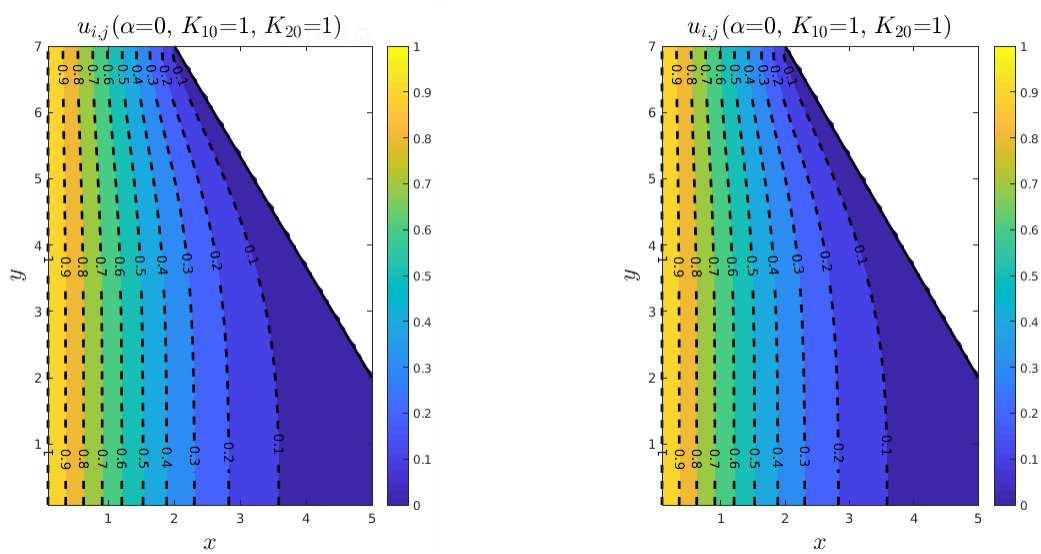


Figure 5: $u_{i,j}^{(1)} = \pm\epsilon = \pm 2.22 \cdot 10^{-16}$ @ (982, 982) iterations

Using an almost zero initial guess, makes no difference in the sign. Both cases show similar distribution regime, which is a layer-like gradient from left to right.

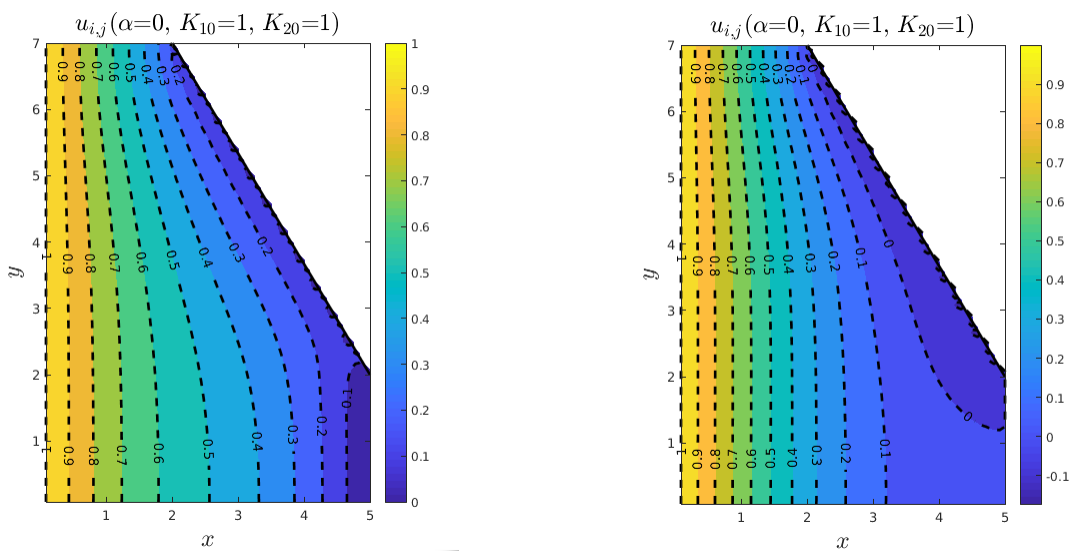


Figure 6: $u_{i,j}^{(1)} = \pm 0.5$ @ ($\underline{\underline{408}}$, 1523)

When the matrix is initialized with ± 0.5 values, convergence is obtained quicker than any other value. Additionally, the layers are start shifting rightwards, towards the diagonal.

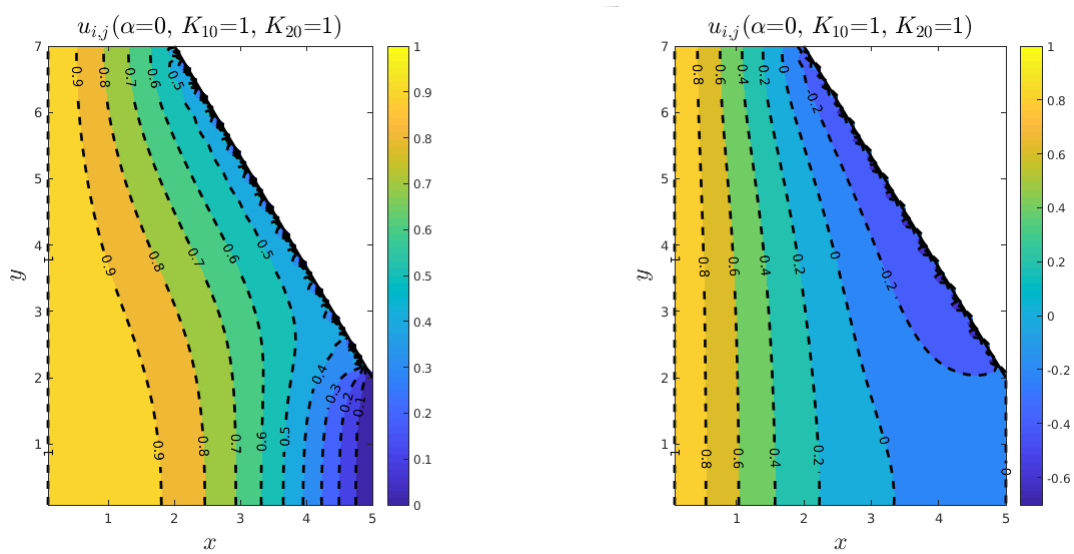


Figure 7: $u_{i,j}^{(1)} = \pm 2$ @ (935, 2881)

Here, the initial guess is bigger and causes a distinctive shifting of the heat distribution. The diagonal shows problematic numerical sensitivity, for its "**memory**" of the initialization values, such that they are roughly change along the iterations.

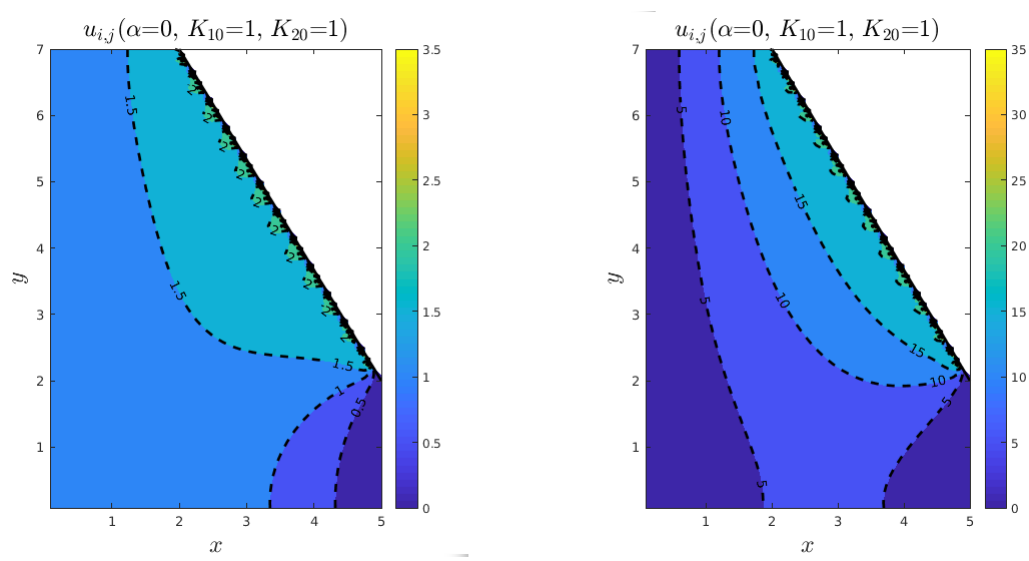


Figure 8: $u_{i,j}^{(1)} = \pm 100$ @ (3217, 4103)

Now when initial guess is insanely high, divergence rises quickly, deleting any initial and boundary conditions. To sum up several empiric attempts :

Initialize	Iterations	
± eps	982	982
± 0.1	827	1119
± 0.25	590	1293
± 0.4	440	1438
± 0.5	**408**	1523
± 0.7	511	1669
± 1	688	1991
± 2	935	2881
± 100	3127	4103

We've seen that the initial guess has a dramatic influence on the solution's development. Moreover, the initial guess is tightly related to the diffusion regime, and will be discussed thoroughly further. We shall therefore work with an initialized matrix of +0.5 values.

5.4 Relaxation factor

The successive over-relaxation (SOR) : $\mathbf{u}_{SOR}^{(n+1)} = u_i^{(n)} + \Omega(u_{i,GS}^{(n+1)} - u_i^{(n)})$, see Eq. 4.5, is an accelerator tool for convergence, using a suitable Ω factor. Examine each of the cases :

	Iterations			
Ω	**Case 1**	**Case 2**	**Case 3**	**Case 4**
1.0	408	449	462	492
1.1	390	429	442	478
1.15	381	689	690	471
1.2	374	411	423	#Diverge
1.28	375	331	299	#Diverge
1.3	362	#Diverge	#Diverge	#Diverge
1.33	#Diverge	#Diverge	#Diverge	#Diverge
1.36	#Diverge	#Diverge	#Diverge	#Diverge

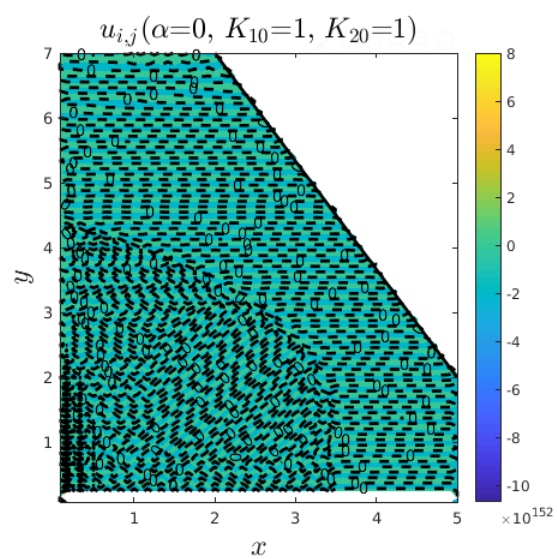
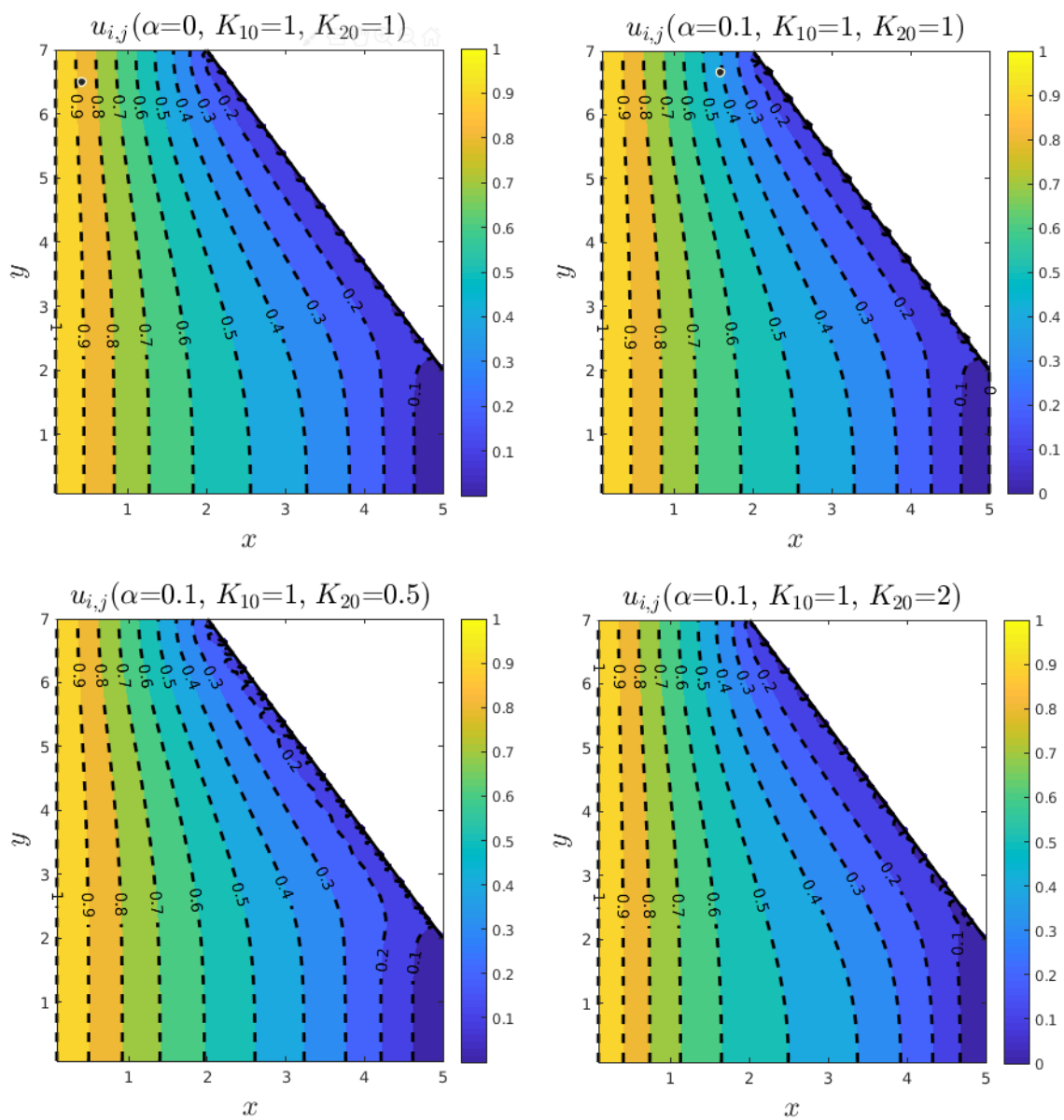


Figure 9: **Right** : Divergence threshold **Left** : Divergence example

The different cases impose different divergence threshold which is already quite low. One of the reasons for that sensitivity is the initial guess matrix that is used, and thus restrain the Ω factor. From here on, we shall use the strict threshold $\Omega_{max} = 1.15$.

6 Results

Following are the 4 combinations running under the setup previously developed :



As seen, the results are almost identical, apart from minor changes in the heat distribution in the lower layers (right side). The heat distributes in a layer-like form from a high potential (left), where the source is $u(0, y) = 1$, to a $u_{BC} = u_{CD} = 0$ in the left boundary. The upper and the lower boundaries are considered as insulated ($\frac{\partial u}{\partial y} = 0$) since no heat change occur, and thus might explain the vertical isotherm layer.

Let us check for any differences (Δu), if at all, on a closer look :

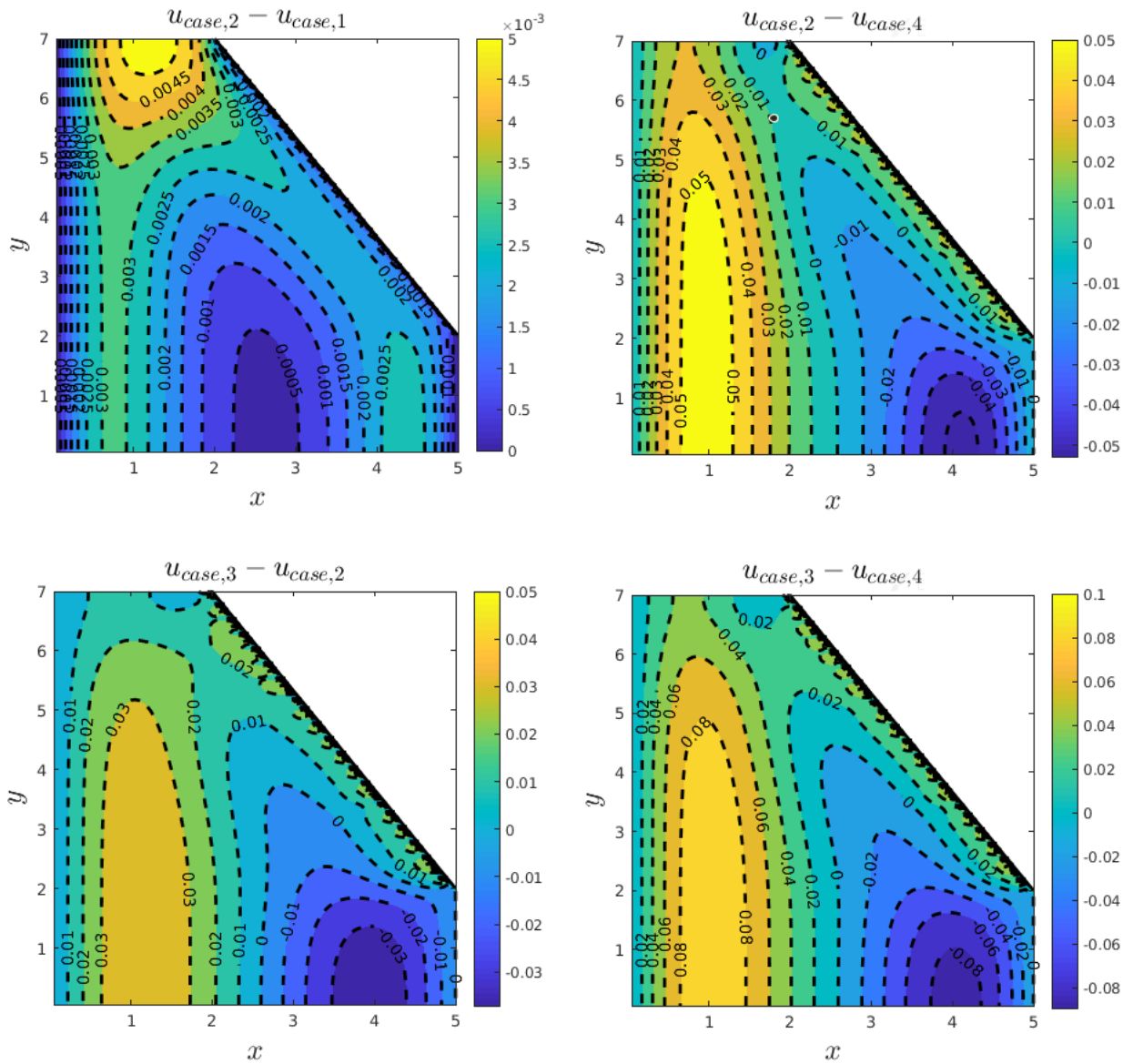


Figure 10: Differences (Δu) between solutions

Generally, the solutions' differences exhibit quite a similar Δu patterns, but noteworthy :

1. The main difference is the magnitude between the cases.
2. Case$_2$ - Case$_1$ differ only by small numerical noise. Hence, α is quite negligible.
3. Case$_4$ tends to perform most different, probably for its highest factor ($K_{20} = 2$).

The different values of the conduction factors dictate the distribution regime, and can thus explain the differences we have seen.

7 Summary and conclusion

• The problem was solved using an implicit central finite differences method with a local truncation error of $O(h^2 + p^2)$, using a mesh size of $\Delta x = \Delta y = 1/12$.

• The nonlinear nature of the PDE required an iterative solution for the set of the algebraic equations, and thus the *Gauss-Seidel* method was used, combined with *SOR* accelerator.

• Given no analytic / validation reference, the reliability of the numeric method was examined per se (Δu), in sought of sensitivity regions and suspicious areas.

• The implicit method diverges (=not robust) under some numerical parameters and initializations, as seen in the initial guess matrix and the relaxation parameter (Ω).

• The geometrical nature of the problem impose numerical constraints that stem from the dependency of the integration steps in one another.

• The diagonal's area exhibits the most numerical sensitivity for the difference between calculation with adjacent cells, and the boundary condition ($u_{BC} = u_{CD} = 0$) it must fulfill.

Main script :

```
clc; close all; clear; set(0, 'defaultfigurecolor', [1 1 1]);
% ----------------------- Description ---------------------- %
%          --- The main script of the program ---            %
% ----------- Initialize values of the problem ------------- %

[xf, yf, step_size] = deal(5, 7, 1/20);
h = step_size; p = h;
[x, y] = deal(h:h:5, p:p:7);                        % x and y axes
alpha = [0, 0.1, 0.1, 0.1];
K_10  = [1,   1,   1,   1];
K_20  = [1,   1, 1/2,   2];

% ------------ Solve 4 variations of the problem ----------- %
for i = 1:length(alpha)
    u = ones(length(x), length(y))*initial_guess;   % b.c. :: Compliant matrix
    u(1, :) = 1;                                    % i.c. :: u_{AE} = 1
    [u_sol(:, :, i), n_iter(i)] = u_solve( u, alpha(i), K_10(i), K_20(i), h, p,
    u_sol(:, :, i) = Clear_excess(u_sol(:, :, i), h, p);  % Clear polygon

    figure('rend', 'painters', 'pos', fig_loc);
    [~, c] = contourf(x, y, u_sol(:, :, i)', '--', 'ShowText','on');
    % ----------- plot diagonal boundaries (?) of the polygon ----------- %
    x_diag = 2:h:5;   y_diag = (31 - 5.*x_diag )./3 ;
    plot(x_diag, y_diag, 'linewidth', 2, 'color', 'black');
    ind(1) = title(['$u_{i,j}$($\alpha$=', num2str(alpha(i)), ', $K_{10}$=']);
    ind(2) = xlabel('$x$'); ind(3) = ylabel('$y$');
end
```

The method :

```
function [u_n, n_iter] = u_solve( u_n, alpha, K_10, K_20, h, p, Omega, err )
% -------------------------- Description ------------------------- %
%                                                                   %
%  This function receives a desired system of equations to solve   %
%  and return the calculted matrix using GS method by chosen err   %
%                                                                   %
% ---------------------- Define the problem ---------------------- %

[k1, k2] = deal(K_10/h^2, K_20/p^2);
X_ij = @(U_n_1, U_n, i, j, sgn) ( U_n(i+1,j) +sgn * U_n_1(i-1,j) );
Y_ij = @(U_n_1, U_n, i, j, sgn) ( U_n(i,j+1) +sgn * U_n_1(i,j-1) );

% ------------------- Define governing method -------------------- %
F = @(U_n, U_n_1, i, j) ( ( k1*X_ij(U_n, U_n_1, i, j, +1) + ...
    k2*Y_ij(U_n,U_n_1,i,j,1) ) + (alpha/(1+alpha*U_n(i,j))/4)*(k1*X_ij...
    (U_n,U_n_1,i,j,-1)^2 + k2*Y_ij(U_n,U_n_1,i,j,-1)^2))/(2*(k1+k2));

[n_iter, u_SOR, u_GS] = deal(1, u_n/100, 0);        % Initialization
```

```
[err_abs, err_rel, err_RMSE] = deal(1, 2, 3);
% --------------------- Iterative convergence -------------------- %
while Error_calc(u_SOR, u_n, err_RMSE) > err
    if (n_iter > 1)
        u_n = u_SOR;
    end
    u_GS = Gauss_Seidel(u_n, F, h, p);
    u_SOR = u_n + Omega * ( u_GS - u_n );
    n_iter = n_iter + 1;
end
u_n = u_SOR;
```

The Gauss-Seidel method :

```
function U_n_1 = Gauss_Seidel( U_n, F, h, p)
% ----------------------- Description ----------------------- %
%                                                              %
%       Gauss-Seidel method iteratively calculates             %
%       the u^(n+1)_{i, j} element along (i, j) grid           %
%       returns : Matrix at iteration (n)                      %
%                                                              %
% --------------------- Initilization --------------------- %

x = @(i) (i*p);
y = @(x) (31-5*x)/3;
A = size(U_n); [i_max, j_max] = deal(A(1), A(2) ); % Framework size

U_n_1 = U_n;                                % Init. Matrices
% ------------------------ 0 < x < 5 ------------------------ %
U_n_1(:, 1) = U_n_1(:, 1+2);                % AB :: du(0) = 0
% -------------------- (x=0, 0 < y < 7 ) ------------------- %
U_n_1(1, :) = 1;                            % AE ::  u    = 1

for i = 2:i_max - 1
    % ------------------- 0 < x < 2 ----------------------- %
    if  x(i) <= 2
        for j = 2:j_max - 1                 % 0 <= y <= 5
            U_n_1(i, j) = F(U_n, U_n_1, i, j);
        end
        U_n_1(i, j+1) = U_n_1(i, j-1);      % DE :: du(0) = 0
        % --------------- 2 < x < 5 ----------------------- %
    elseif x(i) < 5
        for j = 2:floor( y(x(i))/p )        % 0 <= y <= (31-5x)/3
            U_n_1(i, j) = F(U_n, U_n_1, i, j);
        end
        U_n_1(i, j+1) = 0;                  % CD ::  u    = 0
    end
U_n = U_n_1;                            % Swap Matrices
end
% ------------------------- x == 5 ------------------------- %
U_n_1(i+1, 1:j) = 0;                          % BC ::  u    = 0
```