Coupled Problems
Iterative Schemes for Coupling in Space

Federico Valencia Otálvaro

Master's in Numerical Methods in Engineering
Universitat Politècnica de Catalunya

Academic year 2019 - 2020

June 15^{th}, 2020

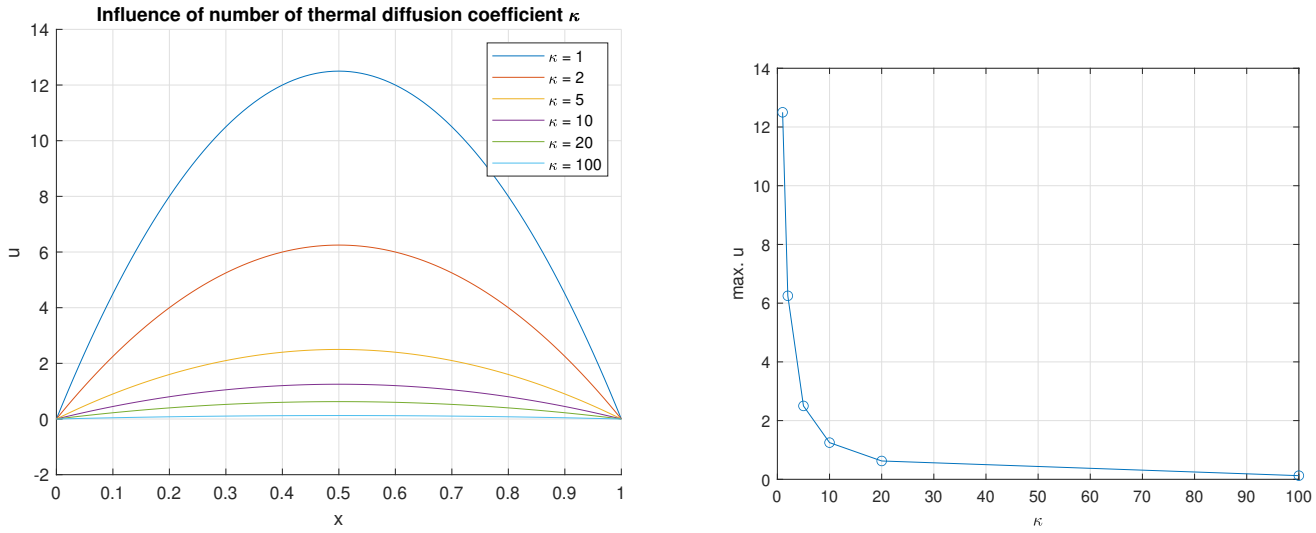
Contents

List of Figures

1 Single Heat Transfer Problem

1.1 Influence of Thermal Diffusion Coefficient κ

The influence of the thermal diffusion coefficient κ in the solution was evaluated while keeping the source term constant as $f = 100$ and a mesh of 100 elements, obtaining the following results.

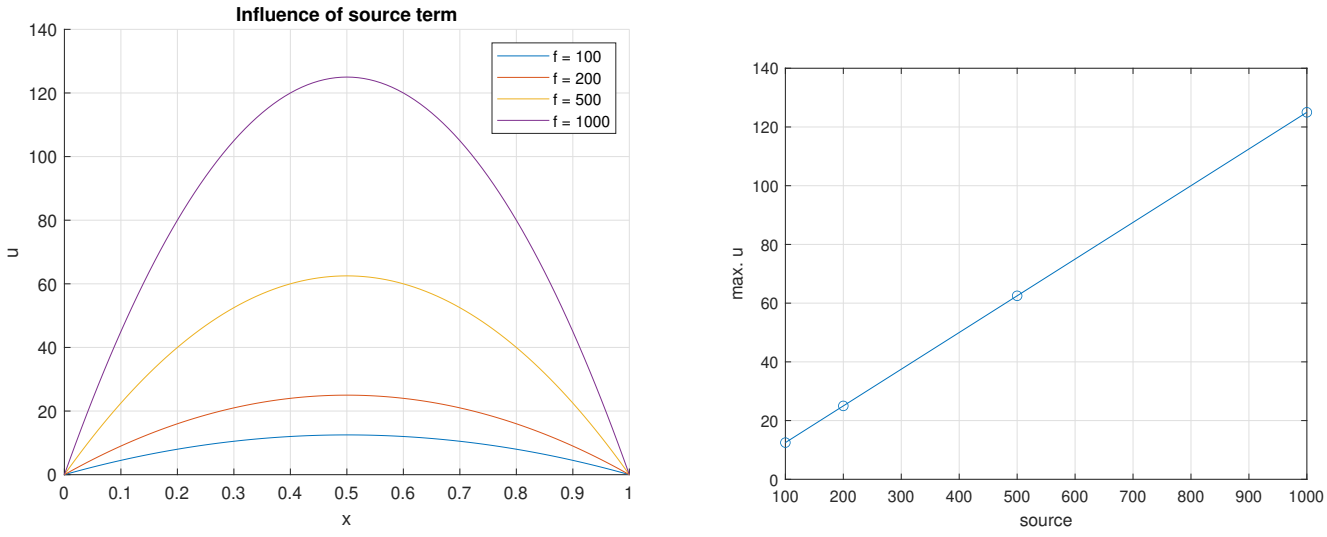


(a) Solution (b) Max. u vs. κ

Figure 1: Influence of number of κ

As expected, there is an inverse relation between the value of κ and the maximum temperature in the domain. Since this relation is not linear, we may observe how the solution is very sensitive to the variation of the diffusion coefficient for small values of κ, and sensibility decreases as the coefficient grows.

1.2 Influence of Source Term Value



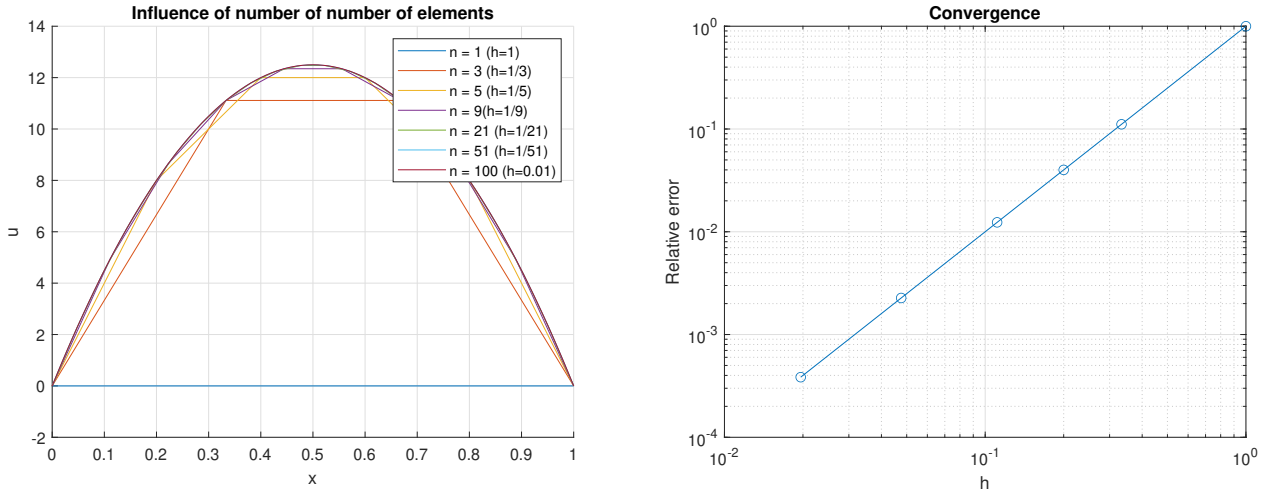
(a) Solution (b) Max. u vs. source term

Figure 2: Influence of source term value

The analysis was performed keeping the thermal diffusion coefficient constant as $\kappa = 1$. A linear direct relation may be seen between the value of the source term and the temperature in the domain.

1.3 Influence of Number of Elements & Convergence Rate

We will now evaluate the error of the maximum temperature in the domain with respect to the element size (h). Due to the fact that the solution of the FE approximation is exact at the nodes, meshes with odd amounts of elements were used to measure the convergence rate of the error in order to avoid having a node at the center of the domain, where the maximum value of the solution is for this particular problem. However, the finest mesh (whose result was taken as the analytical solution) was built with an even number of elements to ensure obtaining the exact solution at the center of the domain.



(a) Solution (b) Convergence

Figure 3: Influence of number of elements

The convergence plot has a slope of 2, which means that we have an error of order $O(h^2)$. Since a linear FE approximation ($p = 1$) is being used and quadratic convergence is being obtained, we may conclude that the optimal convergence rate is reached ($p + 1$).

2 Two Independent Heat Transfer Problems

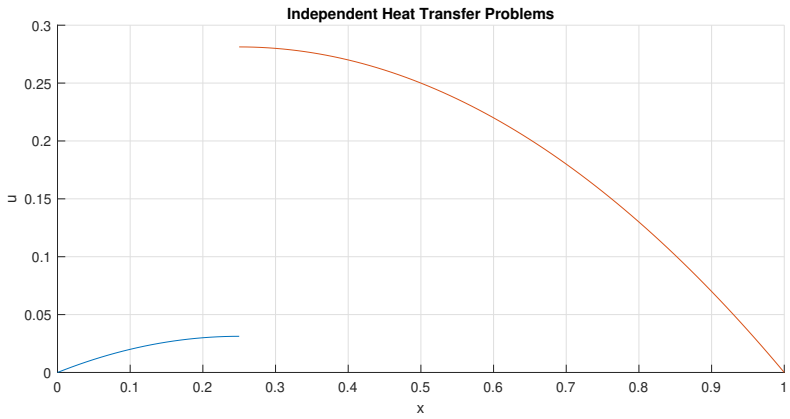


Figure 4: Independent heat transfer problems

This solution corresponds to solving two problems independently with homogeneous Neumann boundary conditions at the "interface" between them. Since we are not imposing any transmission conditions, there is no information passing from one domain to the other, resulting in two completely independent problems which do not interact and evidently, the solutions do not match one another.

3 Monolithic Coupling

We will now employ a monolithic coupling scheme using the `HP_SolveMonolithic.m` Matlab routine to solve the same problem from the previous section. This way, transmission conditions are imposed and therefore, there will be continuity in the solution and fluxes at the interface between the two domains.

3.1 HP_SolveMonolithic.m Function Analysis

The `Hp_SolveMonolithic.m` function takes the discrete system matrices of two problems as input and builds a single global system of equations, adding up the contribution of both subdomains to the stiffness of the node at the interface as well as the forces vector. The system is then solved by simply inverting the global matrix.

The following results were obtained using this function to solve the same problem from the previous section, with a FE discretization of 100 elements (25 for the left subdomain and 75 for the right one), a constant source term of 1 throughout the domain, and homogeneous thermal diffusion coefficient ($\kappa_1 = \kappa_2 = 1$).

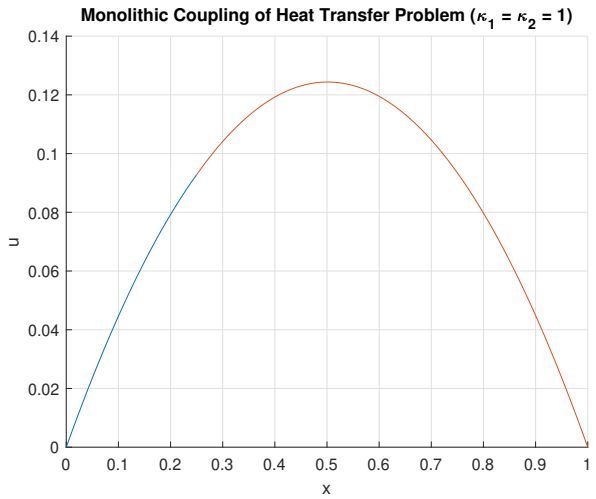


Figure 5: Monolithic coupling ($\kappa_1 = \kappa_2 = 1$)

As expected, both the solution and the fluxes are continuous at the interface between the two subdomains, leading to the same type of results obtained in Section 1, where the problem was being solved in a single domain without any type of partitioning.

3.2 Monolithic Coupling of Problem with Heterogeneous κ

We are now interested in solving a heat transfer problem in a domain composed of two materials with different thermal diffusion coefficients. Therefore, the previous problem is modified, defining $\kappa_1 = 1$ and $\kappa_2 = 50$, obtaining the following results.

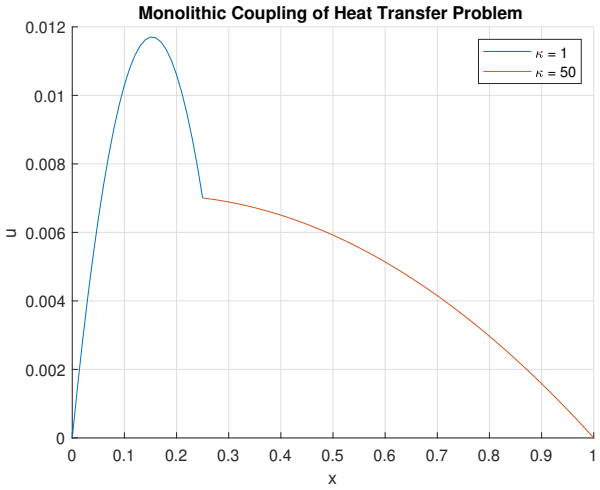


Figure 6: Monolithic coupling ($\kappa_1 = 1, \kappa_2 = 50$)

While the continuity of the solution is evident, it is worth making some comments on the continuity of fluxes in this case. The continuity of fluxes transmission condition at the interface for this problem is given by:

$$\kappa_1 \nabla u_1 = -\kappa_2 \nabla u_2$$

For the given values of κ_1 and κ_2, this results in:

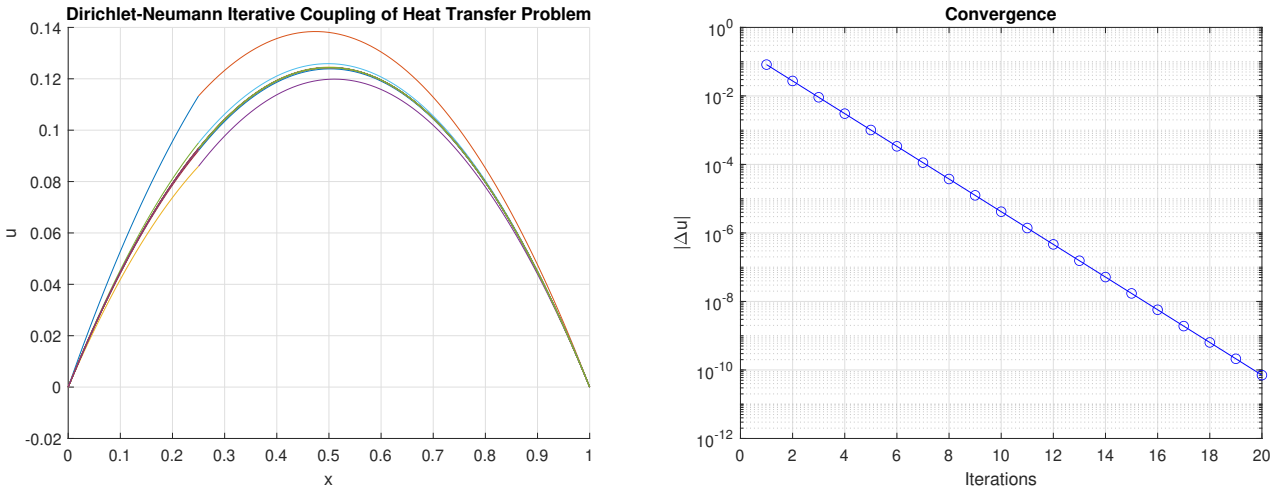
$$\nabla u_1 = -50 \nabla u_2$$

This relation is visibly reflected in Figure 6, allowing us to conclude that the continuity of fluxes is being enforced correctly in the problem.

4 Dirichlet-Neumann Iterative Scheme

We are now interested in solving the problem with homogeneous thermal diffusion coefficient ($\kappa_1 = \kappa_2 = 1$) using a Dirichlet-Neumann iterative coupling scheme, applying a Neumann boundary condition on the left subdomain and a Dirichlet boundary condition on the right one.

a) Evaluate the convergence rate of the iterative scheme (in terms of u at the interface).

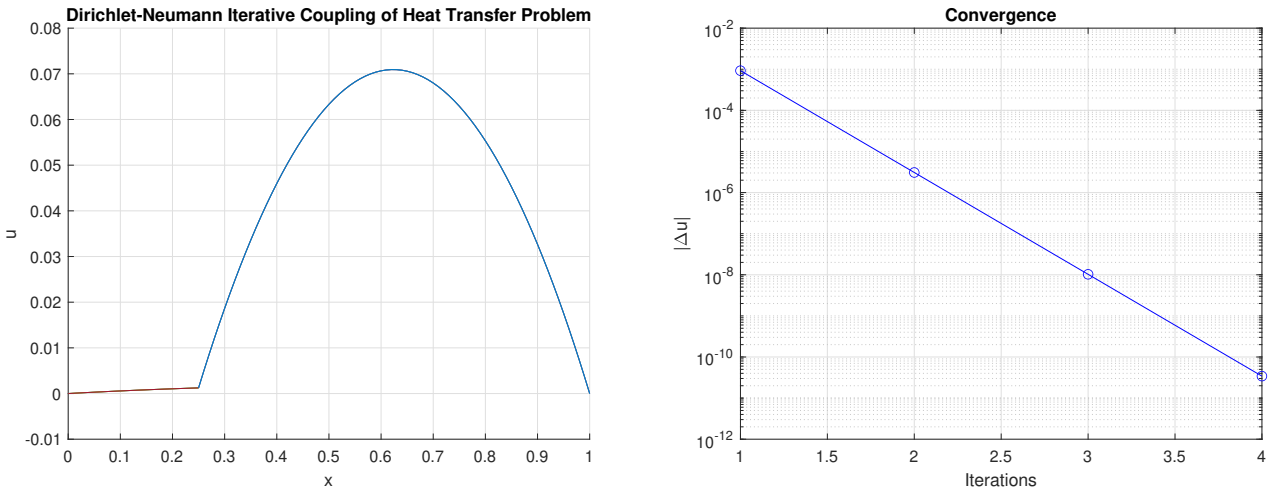


(a) Solution (b) Convergence

Figure 7: DN iterative scheme ($\kappa_1 = \kappa_2 = 1$)

Using $|\Delta u| = |u_\Gamma^{i+1} - u_\Gamma^i|$ as our convergence criteria, the scheme converges to a tolerance of 10^{-10} after 20 iterations, reaching a solution practically identical to the one obtained through the monolithic coupling approach (see Figure 5).

b) Increase the value for κ at subdomain 1 (x100). Comment on the convergence rate.

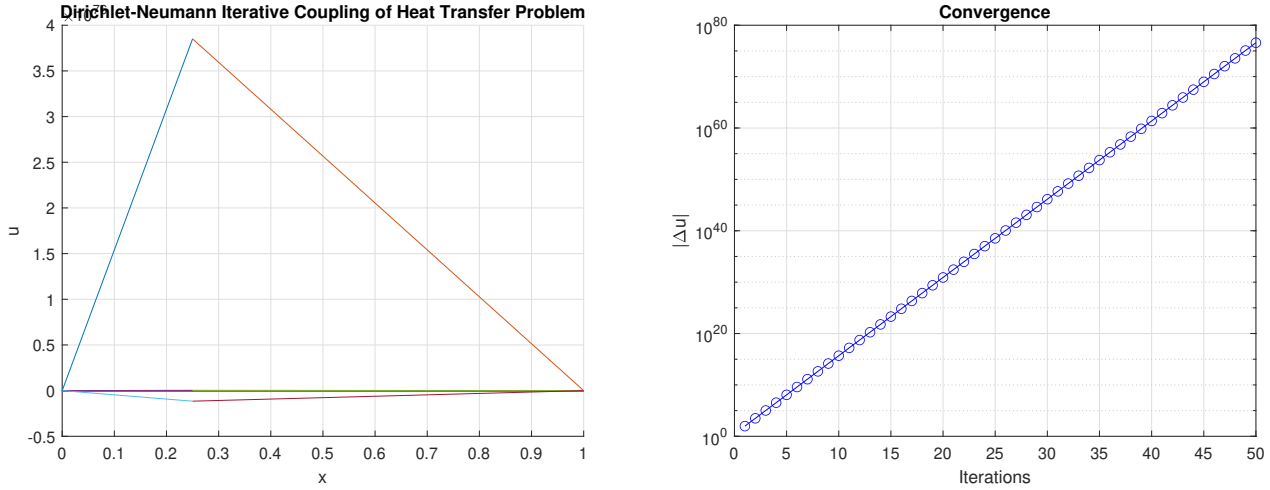


(a) Solution (b) Convergence

Figure 8: DN iterative scheme ($\kappa_1 = 100, \kappa_2 = 1$)

Convergence was much faster in this case, reaching an error of 10^{-10} after only four iterations.

c) Diminish the value for κ at subdomain 1 (/100). Comment on the convergence rate.



(a) Solution (b) Convergence

Figure 9: DN iterative scheme ($\kappa_1 = 0.01, \kappa_2 = 1$)

In this case, where the Dirichlet boundary condition is being applied to the subdomain with the larger thermal diffusion coefficient κ, the scheme fails to reach convergence, as the error steadily grows after every consecutive iteration, reaching an error of order approximately 10^{75} on the 50^{th} and final iteration.

d) Motivate the previous results in terms of the stability of the coupling scheme.

We may conclude that the convergence rate of the scheme depends on the ratio $\frac{\kappa_1}{\kappa_2}$, given that the Dirichlet boundary condition is being applied on subdomain 2. The higher this ratio is, the faster the scheme will converge. However, the scheme will diverge for small enough values of the ratio. This effect may be compared to the added mass effect, which occurs in fluid-structure interaction problems when the fluid has a similar or higher density than the structure, and the Dirichlet boundary condition is being applied on the fluid.

The implementation of a relaxation scheme is an effective way to deal with this inconvenient, as will be shown in the following section.

5 Relaxation Scheme

Let us now focus on sovling the problem which presented convergence problems in the previous section ($\kappa_1 = 0.01, \kappa_2 = 1$), implementing a relaxation scheme to make our Dirichlet-Neumann algorithm more robust.

Two approaches will be explored to obtain the relaxation parameter: Choosing a fixed value and implementing an Aitken scheme, which formulates the relaxation parameter as a function of problem information obtained in previous iterations.

5.1 Fixed Relaxation Parameter Scheme

After a trial and error process, it was observed that for this particular problem, the scheme will only converge for small values of the relaxation parameter ($w < 0.05$). Therefore, the convergence rate of the scheme was studied within the range $0 < w < 0.05$.

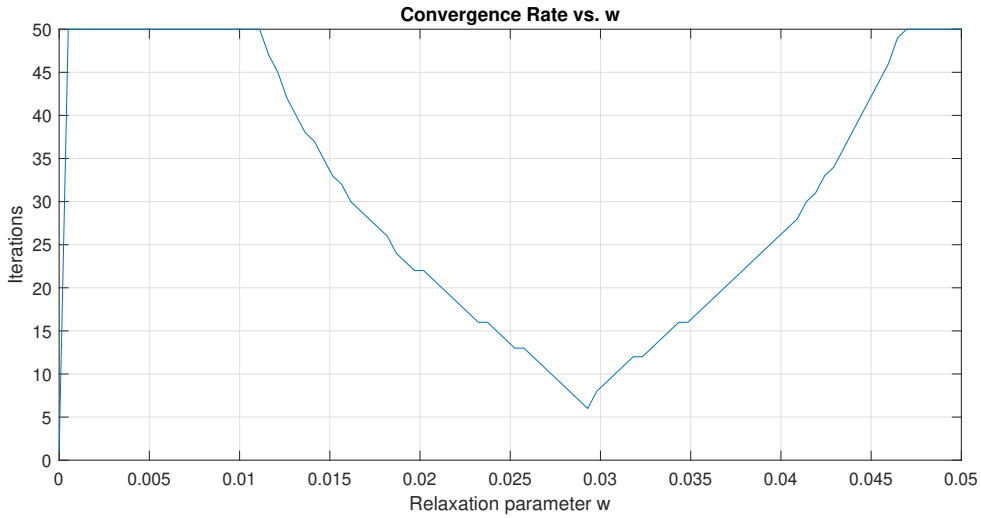
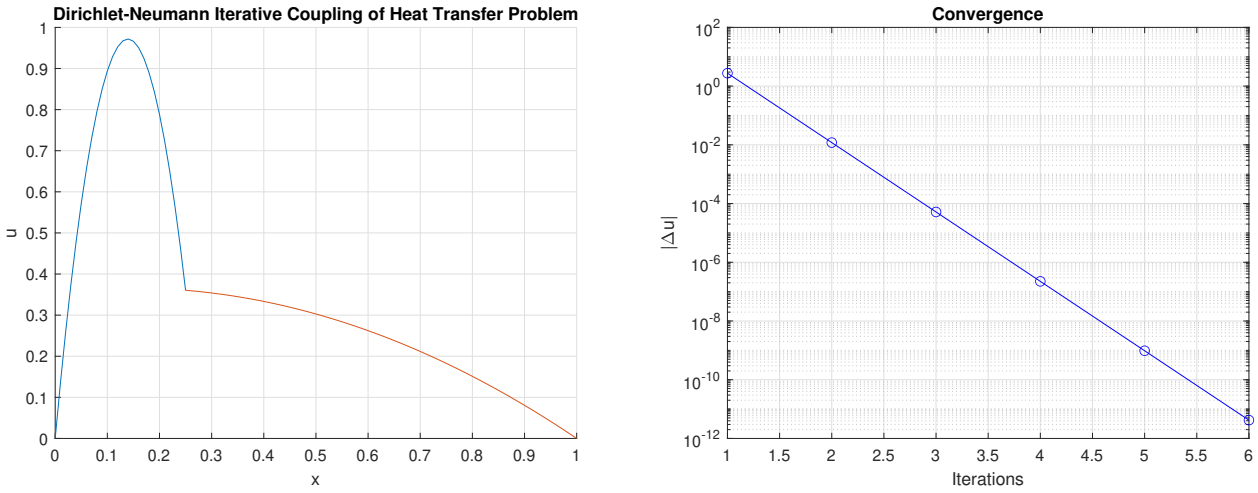


Figure 10: Convergence rate vs. w, fixed value relaxation scheme

The optimal value for w is approximately 0.029. The following results were obtained when solving the problem using this value for the relaxation parameter:

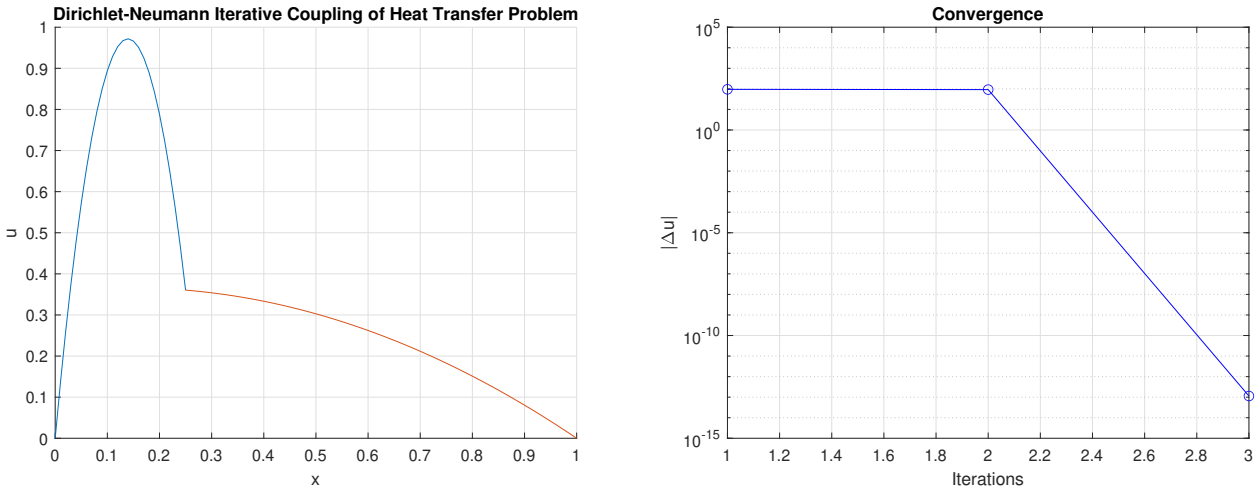


(a) Solution (b) Convergence

Figure 11: DN iterative scheme, fixed value relaxation ($w = 0.029$)

Good results are obtained in a small number of iterations. However, if we were to change any of the problem variables, the optimal value for w would most likely change, and finding it manually again would not be a suitable option for obvious reasons. Therefore, we need an adaptive relaxation scheme which takes information from the problem to find the ideal value for the relaxation parameter w at each iteration. The Aitken relaxation fills this description.

5.2 Aitken Relaxation Scheme



(a) Solution (b) Convergence

Figure 12: DN iterative scheme, Aitken relaxation

Convergence is reached after only 3 iterations (2 in reality since no relaxation is used on the first iteration). It is not only faster than the fixed value scheme but also not problem dependent, since we do not need to modify the scheme depending on the problem variables. We may thus conclude that it is a more efficient and robust method, making it a better choice altogether.

A Annex: Implemented Matlab Codes

A.1 Task 1

```
close all
clear all

%% PROBLEM DATA
%Domain
Data.inix = 0;
Data.endx = 1;
%Data.nelem = 100;
%Physical
Data.kappa = 1;
Data.source = 100;
%Boundary conditions
%Dirichlet
Data.FixLeft = 1;          %0, do not fix it, 1: fix it
Data.LeftValue = 0;
Data.FixRight =1;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 25;

u = [];     % will contain max. value of solution for every mesh
n_elem = [1 3 5 9 21 51 100]; % number of elements of each mesh

%% PROCESSING

for i = 1:length(n_elem)

    Data.nelem = n_elem(i);

    HeatProblem = HP_Initialize(Data);
    HeatProblem = HP_Build(HeatProblem);
    HeatProblem = HP_Solve(HeatProblem);

    HP_Plot(HeatProblem,1);

    u(i) = max(HeatProblem.Solution.U);

end

%% CONVERGENCE

u_analytic = u(end);

error = [];

for i = 1:length(u)
    error(i) = abs((u(i) - u_analytic)/u_analytic);
end

h = ones(size(n_elem))./n_elem;
```

```
figure(2)
loglog(h,error,'o-')
grid on
title('Convergence')
xlabel('h')
ylabel('Relative error')


% figure(2)
% plot(f,u,'o-')
% grid on
% xlabel('source')
% ylabel('max. u')
```

A.2 Task 2

```
%close all
clear all

%% DOMAIN 1 DATA
%Domain
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;
%Physical
Data.kappa = 1;
Data.source = 1;
%Boundary conditions
%Dirichlet
Data.FixLeft = 1;          %0, do not fix it, 1: fix it
Data.LeftValue = 0;
Data.FixRight =0;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

%% DOMAIN 2 DATA
%Domain
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
Data2.kappa = 1;
Data2.source = 1;
%Boundary conditions
%Dirichlet
Data2.FixLeft = 0;          %0, do not fix it, 1: fix it
Data2.LeftValue = 0;
Data2.FixRight =1;
Data2.RightValue = 0;
%Neumann
Data2.FixFluxesLeft = 1;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;
```

```
%% PROCESSING
HeatProblem = HP_Initialize(Data);
HeatProblem = HP_Build(HeatProblem);
HeatProblem = HP_Solve(HeatProblem);

HeatProblem2 = HP_Initialize(Data2);
HeatProblem2 = HP_Build(HeatProblem2);
HeatProblem2 = HP_Solve(HeatProblem2);

%% POST PROCESSING
HP_Plot(HeatProblem,1);
HP_Plot(HeatProblem2,1);
```

A.3 Task 3

```
%close all
clear all

%% DOMAIN 1 DATA
%Domain
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;
%Physical
Data.kappa = 1;
Data.source = 1;
%Boundary conditions
%Dirichlet
Data.FixLeft = 1;          %0, do not fix it, 1: fix it
Data.LeftValue = 0;
Data.FixRight =0;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 0;

%% DOMAIN 2 DATA
%Domain
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
Data2.kappa = 1;
Data2.source = 1;
%Boundary conditions
%Dirichlet
Data2.FixLeft = 0;          %0, do not fix it, 1: fix it
Data2.LeftValue = 0;
Data2.FixRight =1;
Data2.RightValue = 0;
%Neumann
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 25;
```

```
%% PROCESSING

HeatProblem = HP_Initialize(Data);
HeatProblem = HP_Build(HeatProblem);

HeatProblem2 = HP_Initialize(Data2);
HeatProblem2 = HP_Build(HeatProblem2);


[HeatProblem,HeatProblem2] = HP_SolveMonolithic(HeatProblem,HeatProblem2);

%% POST PROCESSING
HP_Plot(HeatProblem,1);
HP_Plot(HeatProblem2,1);
```

A.4 Tasks 4 and 5

```
close all
clear

%% DOMAIN 1 DATA
%Domain
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;
%Physical
Data.kappa = 0.01;
Data.source = 1;
%Boundary conditions
%Dirichlet
Data.FixLeft = 1;          %0, do not fix it, 1: fix it
Data.LeftValue = 0;
Data.FixRight =0;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

%% DOMAIN 2 DATA
%Domain
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
Data2.kappa = 1;
Data2.source = 1;
%Boundary conditions
%Dirichlet
Data2.FixLeft = 0;          %0, do not fix it, 1: fix it
Data2.LeftValue = 0;
Data2.FixRight =1;
Data2.RightValue = 0;
%Neumann
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
```

```
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 25;

%% PROCESSING
[HeatProblem,HeatProblem2,error,iterations] = HP_SolveIterative(Data, Data2, 2, 0);

%% POST PROCESSING
HP_Plot(HeatProblem,1);
HP_Plot(HeatProblem2,1);

figure(2)
semilogy(error,'bo-')
grid on
title('Convergence')
xlabel('Iterations')
ylabel('|\Deltau|')

% theta = linspace(0,0.05);
% iter = [];
%
% for i = 1:length(theta)
%
%     [HeatProblem,HeatProblem2,error,iterations] = HP_SolveIterative(Data, Data2, ...
%   1, theta(i));
%     iter(i) = iterations;
% end
%
% plot(theta,iter)
% grid on
% title('Convergence Rate vs. w')
% xlabel('Relaxation parameter w')
% ylabel('Iterations')
```

A.5 Iterative Solver (with and without relaxation)

```
function [HeatProblem,HeatProblem2,error,cont] = ...
    HP_SolveIterative(Data, Data2, relax, theta)

% relax = 0 no relaxation
%       = 1 constant relaxation parameter
%       = 2 Aitken

% theta = relaxation parameter for option 1

% Initialize left domain
HeatProblem = HP_Initialize(Data);
HeatProblem = HP_Build(HeatProblem);
HeatProblem = HP_Solve(HeatProblem);

u_i = HeatProblem.Solution.uRight;  % store solution of previous iteration for error computa

% Impose solution of left domain as Dirichlet BC on right domain and initialize
Data2.FixLeft = 1;
Data2.LeftValue = u_i;

HeatProblem2 = HP_Initialize(Data2);
HeatProblem2 = HP_Build(HeatProblem2);
HeatProblem2 = HP_Solve(HeatProblem2);
```

```
% ITERATIVE SCHEME
error = [];
err = 1;
tol = 1e-10;
max_iter = 50;
cont = 0;

while err > tol && cont < max_iter

    cont = cont +1;

    % Problem 1

    Data.RightFluxes = -HeatProblem2.Solution.FluxesLeft;   % update Neumann BC at left doma

    HeatProblem = HP_Initialize(Data);
    HeatProblem = HP_Build(HeatProblem);
    HeatProblem = HP_Solve(HeatProblem);

    switch relax

        case 0 % no relaxation
            u_i1 = HeatProblem.Solution.uRight;

        case 1 % constant relaxation
            w = theta;
            u_i1 = w * HeatProblem.Solution.uRight + (1-w)*u_i;

        case 2 % Aitken
            if cont == 1
                u_0 = u_i;
                u_i1 = HeatProblem.Solution.uRight;
                u_i_trial = HeatProblem.Solution.uRight;
            else
                w = (u_0 - u_i)/(u_0 - u_i + HeatProblem.Solution.uRight - u_i_trial);
                u_i1 = w * HeatProblem.Solution.uRight + (1-w)*u_i;

                u_i_trial = HeatProblem.Solution.uRight;
                u_0 = u_i;
            end
    end

    err = abs(u_i1 - u_i);
    error(cont) = err;

    u_i = u_i1;

    % Problem 2
    Data2.LeftValue = u_i;  % update Dirichlet BC at left domain

    HeatProblem2 = HP_Initialize(Data2);
    HeatProblem2 = HP_Build(HeatProblem2);
    HeatProblem2 = HP_Solve(HeatProblem2);


%     HP_Plot(HeatProblem,1);
%     HP_Plot(HeatProblem2,1);

end
```