Coupled Problems

Master in Numerical Methods in Engineering

Numerical Homework

Iterative schemes for coupling in space

Head Professor: Ramón Codina

Student: Marcello Rubino

Date: 6 June 2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

1987-2017 CIMNE 30 years International Centre for Numerical Methods in Engineering

1. Introduction

The following report shows the implementation of MatLab code for different iterative (and non-iterative) schemes for the resolution of the heat transfer equation in one-dimensional domain (i.e. [0,1]): the equation shows the following structure:

$$-\kappa \frac{\partial^2 u}{\partial x^2} = f$$

In which *k* represents the thermal diffusivity coefficient, *u* is the solution of the problem (which corresponds to the temperature in the domain, and *f* is the source term which is physically a volumetric heat source). The model implemented and shown below are the following:

- *Single domain problem* model
- *Two sub-independent problems* model
- *Monolithic scheme* model
- *Interation-by-subdomain Dirichlet-Neumann iterative* model
- *Relaxation scheme with fixed parameter* model
- *Aitken relaxation scheme* model

2. Single domain problem

In this first case the whole problem is studied in a whole single domain [0,1], discretized in 100 elements. This corresponds to a typical FE code without any particular method. In this case the fixed parameters at the boundaries are the value of the u (precisely u=0 in both x=0 and x=1). This will mean that the temperature will be 0 in both boundaries. The source term is set to 1. Few studies on some parameters have been computed.

2.1 – Different values of the diffusivity k

The first one consists in the change of the thermal diffusivity *k* and plot the results in a complete graph. As shown in figure 1, the value of the diffusivity varies significantly: the main effect is the reduction of the temperature seen (as expected, due to the symmetry of the problem) in the middle. This is physically consistent because it shows the particularities of the metals (high values of the diffusivity) to dissipate the heat given from the external world (the *f*) much more easily than wood or rubber.

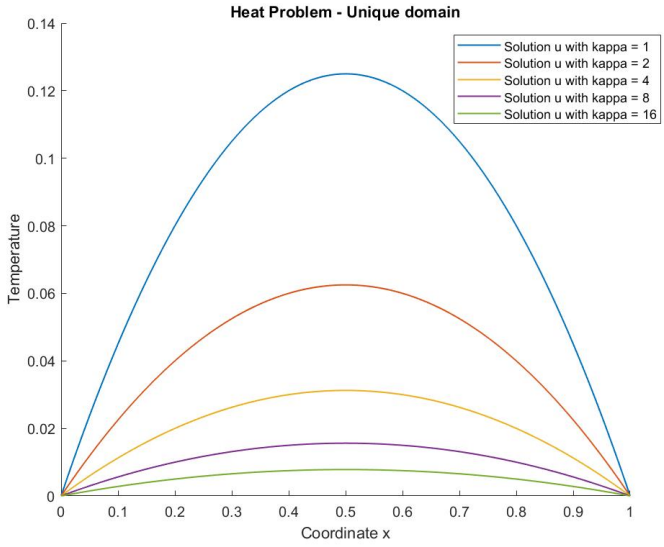


Figure 1 - Unique domain - different diffusivity k values

2.2 – Different values of the source term f

The second study is the effect of increasing the volumetric heat source *f*. The figure 2 below represents this effect, and varies the value of the source between 1 and 16, while keeping constant the diffusivity *k* to 1. In this case the situation is different: the material modeled is the same, while the external heat is set by the scientist. Increasing the heat shows an increase of the internal temperature, since a bigger quantity of the heat must be dissipated inside the domain through a higher value of the temperature.

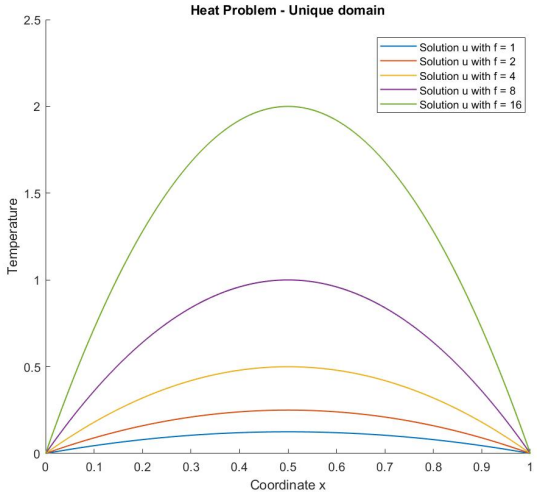


Figure 2 - Unique domain – different source terms f

2.3 – Convergence of the problem – Number of elements

The last discussion in this model is a numerical study: increasing the number of the elements which discretize the domain it's possible to make a convergence analysis and see the good results given by a finer mesh. In fact, as shown in figure 3, the increase of the number of element shows a more precise solution for the same problem and gets closer to the analytical solution given by the parabolic equation:

$$u = -\frac{f}{2k}x^2 + \frac{f}{2k}x$$

Which shows the maximum value (as already reported) in the middle of the domain (x=0.5):

$$u|_{x=0.5} = -\frac{f}{2k}0.5^2 + \frac{f}{2k}0.5 = 0.125 \text{ (if } k\text{=}f\text{=1)}$$

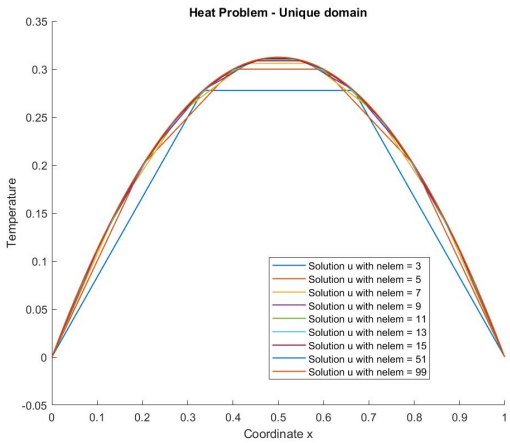


Figure 3 - Unique domain - different numbers of elements

The convergence analysis has been done in terms of the relative error between the maximum value of the temperature for the coarse-discretized problem (for symmetry reasons the value of the elements is odd), compared to the analytical solution. On the x-axis the mesh size has been reported. As expected, both figure 4 and 5 (logarithmic reproduction) show that the convergence is satisfied and a good solution is acquired already for 100 elements.

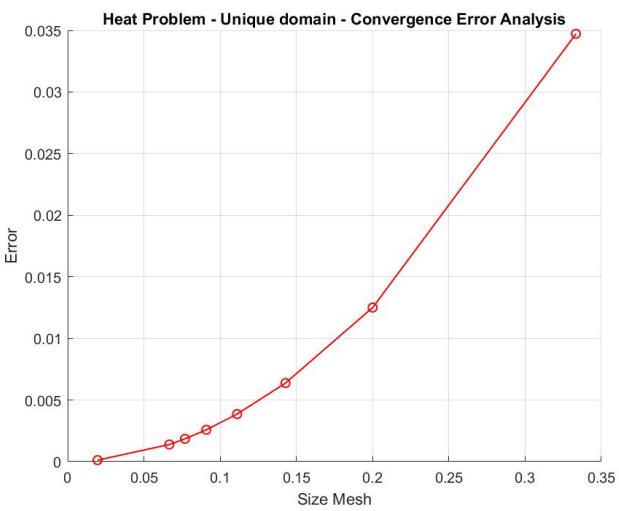


Figure 4 - Unique domain – Convergence Analysis

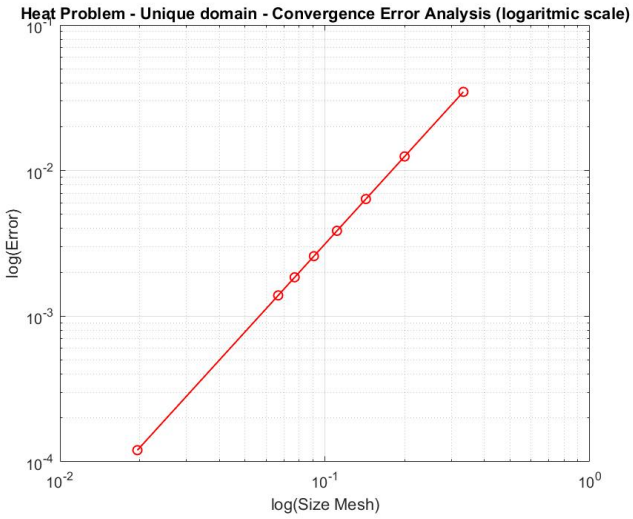


Figure 5 – Unique domain – Convergence Analysis (logarithmic scale)

3. Two sub-independent problems

In this case the domain [0,1] has been split in two subdomains: the first one goes between 0 and 0.25 and has 25 elements, while the second is between 0.25 and 1, and presents 75 elements. In this problem the coefficients kappa and the source term for both subdomains are set to 1. The Dirichlet boundary conditions are given to the extremal nodes of the old big domain (u=0 in x=0, left value for the left sbd; u=0 in x=1, right value for the right sbd). The rest of the boundary conditions (Neumann or interface Dirichlet) are not set. The result, as shown in figure 6, presents a big jump at the interface point, due to the fact that each subdomain solves its own problem with only one B.C. and doesn't care of conditions (Transmission) coming from the other subdomain.

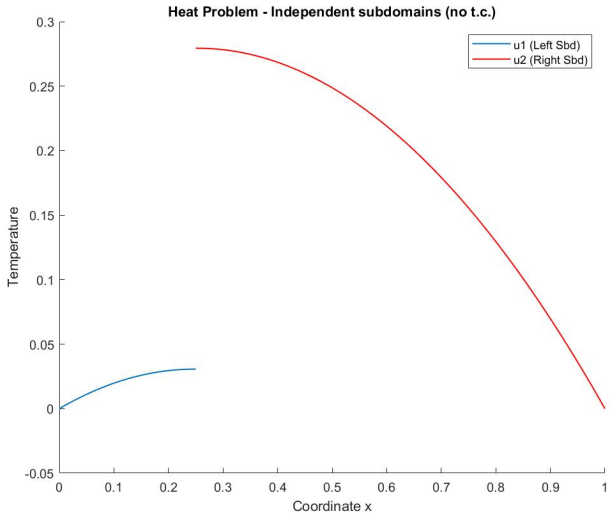


Figure 6 - Independent subdomains

4. Monolithic scheme

4.1 – Theoretical considerations

In this particular scheme the domain, partitioned into two subdomains (which have the same size as properties described before), is characterized by a big stiffness matrix *K* and rhs vector *f*, simply coming from the assembly of the two subdomains interacting. This is the case in which the coupling is done by enforcing a Neumann-Neumann coupling approach. In this case we have the first subdomain which shows the following weak form:

$$\left(\frac{\partial v_{1h}}{\partial x}, k_1 \frac{\partial u_{1h}}{\partial x}\right)_{\Omega_1} - < v_{1h}, k_1 \frac{\partial u_{1h}}{\partial x} >_{interface} = < v_{1h}, f >_{\Omega_1}$$

The Neumann – Neumann coupling consists in the following fact:

$$- k_2 \frac{\partial u_{2h}}{\partial x} + k_1 \frac{\partial u_{1h}}{\partial x} = 0$$

And this can be insert in the equation for the second subdomain:

$$\left(\frac{\partial v_{2h}}{\partial x}, k_2 \frac{\partial u_{2h}}{\partial x}\right)_{\Omega_2} - < v_{2h}, -k_1 \frac{\partial u_{1h}}{\partial x} >_{interface} = < v_{2h}, f >_{\Omega_2}$$

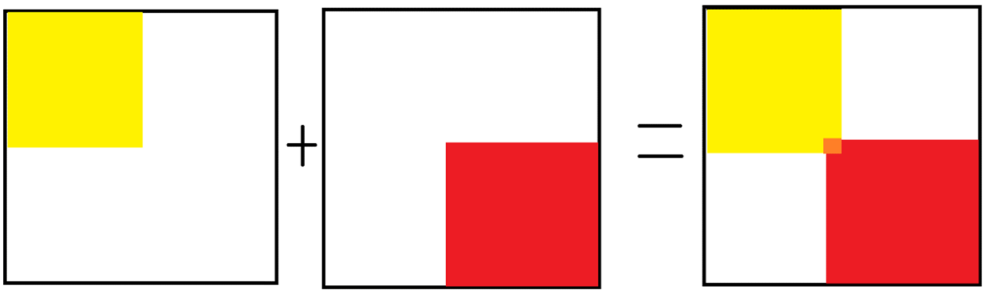
Since the meshes match at the interface and the interpolation space V_h used in this approach is the same, we obtain this equality:

$$-< v_{2h}, -k_1 \frac{\partial u_{1h}}{\partial x} >_{interface} = < v_{1h}, k_1 \frac{\partial u_{1h}}{\partial x} >_{interface} = \left(\frac{\partial v_{1h}}{\partial x}, k_1 \frac{\partial u_{1h}}{\partial x}\right)_{\Omega_1} - < v_{1h}, f >_{\Omega_1}$$

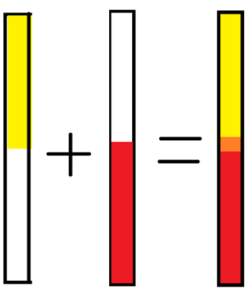
So we can rewrite the equation for the problem in the second subdomain:

$$\left(\frac{\partial v_{2h}}{\partial x}, k_2 \frac{\partial u_{2h}}{\partial x}\right)_{\Omega_2} + \left(\frac{\partial v_{1h}}{\partial x}, k_1 \frac{\partial u_{1h}}{\partial x}\right)_{\Omega_1} = < v_{1h}, f >_{\Omega_1} + < v_{2h}, f >_{\Omega_2}$$

Which means that the two subdomains can be assembled into a big global problem, defined by the following stiffness matrix structure:



And the following right hand side global vector:



In which the yellow parts are linked to the nodes in the first subdomain, the red to the second subdomain while the orange part to the interface. This scheme is well visible in the *HP_SolveMonolithic.m* Matlab function (figure 7)in which, before solving the global problem, the program assembles the two subdomains, considering twice the contribution for the interface node (*npoin1* position) coming from the two subdomains as explained by theory above:

```
function [HeatProblem,HeatProblem2] = HP_SolveMonolithic(HeatProblem,HeatProblem2)

npoin1 = HeatProblem.Data.nelem+1;
npoin2 = HeatProblem2.Data.nelem+1;

%Monolithic system
KMono = zeros(npoin1+npoin2-1);
FMono = zeros(npoin1+npoin2-1,1);

KMono(1:npoin1,1:npoin1) = HeatProblem.Matrices.K;
FMono(1:npoin1) = HeatProblem.Matrices.F;

KMono(npoin1:npoin1+npoin2-1,npoin1:npoin1+npoin2-1) = KMono(npoin1:npoin1+npoin2-1,npoin1:npoin1+npoin2-1) + HeatProblem2.Matrices.K;
FMono(npoin1:npoin1+npoin2-1) = FMono(npoin1:npoin1+npoin2-1) + HeatProblem2.Matrices.F;

%Solve Monolithic
UMono = KMono\FMono;

%Give it back to each problem
HeatProblem.Solution.U = UMono(1:npoin1);
HeatProblem = HP_PostSol(HeatProblem);

HeatProblem2.Solution.U = UMono(npoin1:end);
HeatProblem2 = HP_PostSol(HeatProblem2);
```

Figure 7 - Matlab code for HP_SolveMonolithic.m

4.2 – Solutions computed in the scheme

As shown in figure 8 the solution is the same of the unique domain problem shown in figure 1. Here the two subdomains solutions are visible in two different colors and the values for k and f are equal to 1 in both subdomains.

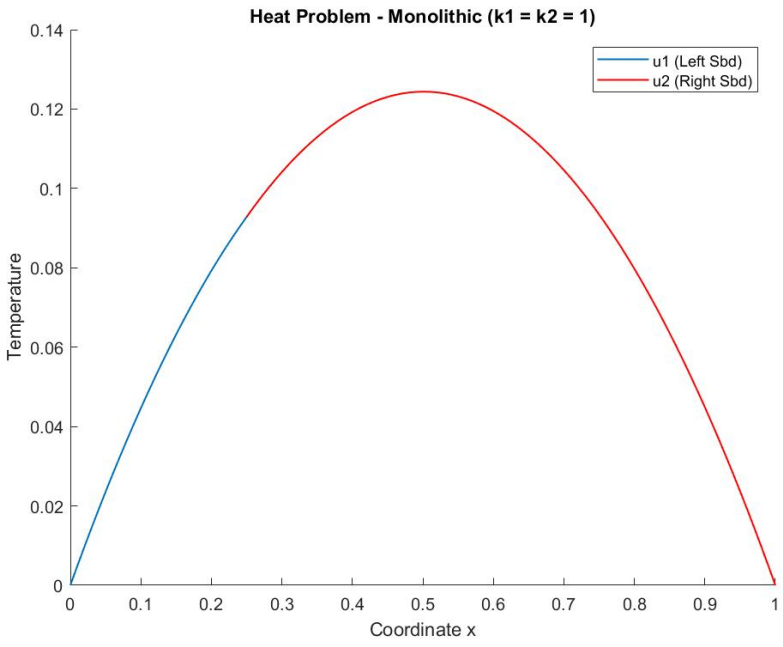
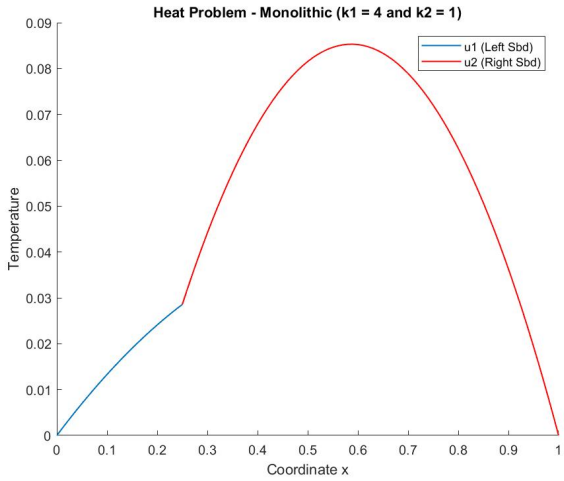


Figure 8 – Monolithic scheme – $k_1 = k_2 = 1$

If one of the two subdomains has a different value for the diffusivity, the solution is much different. In fact, the growth of the temperature in the subdomain that has the smaller diffusivity tends to be much slower, while the subdomain with big diffusivity has a higher change of the temperature along the space. This creates a jump for the derivatives visible at the interface. It's important to say that the second transmission condition (Neumann-Neumann) is always satisfied, since the

derivatives, if scaled (multiplied) for a same value of the diffusivity, would get the smooth shape for the previous case, so the normal flux has a continuity. In figure 9 and 10 there are the monolithic solutions respectively for $k_1 = 4$ and $k_2 = 1$, and $k_1 = 1$ and $k_2 = 4$. The values of the temperature reached in the second case are smaller since the biggest subdomain has the biggest diffusivity *k*.



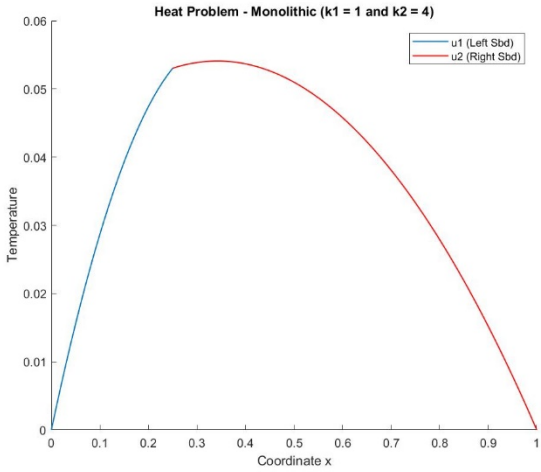


Figure 9 – Monolithic scheme – $k_1 = 4$, $k_2 = 1$

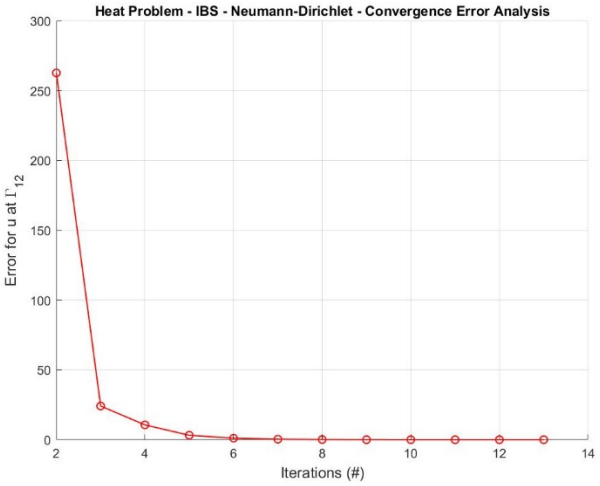
Figure 10 – Monolithic scheme – $k_1 = 1$, $k_2 = 4$

5. Iteration-by-subdomain scheme (Dirichlet-Neumann)

In this more sophisticated case, the scheme represents an iterative method between the two subdomains: the first subdomain solve a Neumann problem and sends the solution at the interface to the second subdomain, which takes this value as a Dirichlet boundary condition (it solves a Dirichlet problem). Then, the second subdomain sends the value of the flux calculated at the interface, which will be collected by the first subdomain, which will solve again a Neumann problem (and so on…).

5.1 – Constant diffusivity k = 1 in the whole domain

In this case both subdomains present a $k = 1$ and a source $f = 1$. The solution of this problem, is not reported as a figure, since it's the same that can be seen in figure 8 for the Monolithic case. More interesting is the evaluation of the error at the interface during the iterations, and plot this against the number of iterations computed during the solving process. The figure 11 and 12 show the results in normal and logarithmic scale. It's possible to see that, without any relaxation method, the pure D-N scheme needs many iterations before converging to the solution.



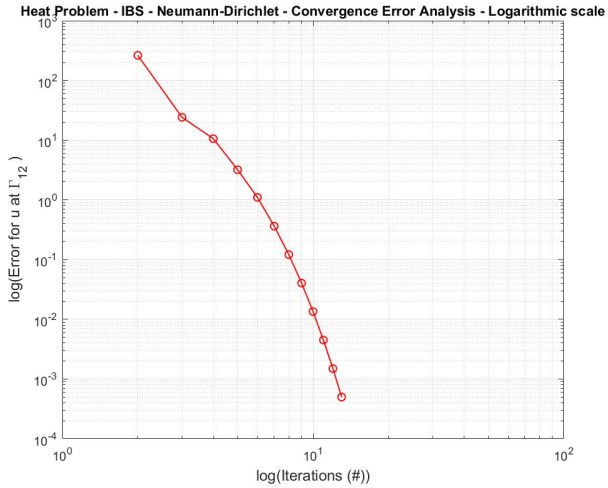


Figure 11 – Dirichlet–Neumann, $k_1=k_2=1$ – Conv. Analysis

Figure 12 – Dirichlet–Neumann, $k_1=k_2=1$ – Conv. Analysis (log. scale)

5.2 – Diffusivity $k_1 = 100*k_2$

The same scheme has been used for a big difference in terms of diffusivity in the two subdomains. The first study shows a value for the first subdomain equal to 100, while k_2 remains equal to 1. In this case, the solution is a stressed case for the one shown in figure 9 (in this case the growth of the temperature on the left is very much slow, and almost imperceptible) [figure 13].

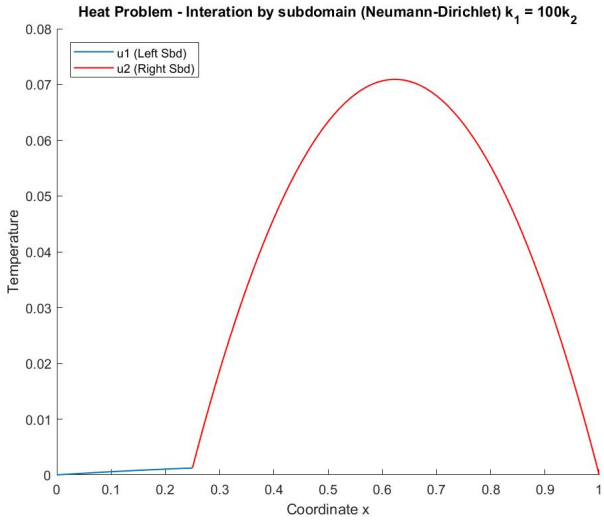
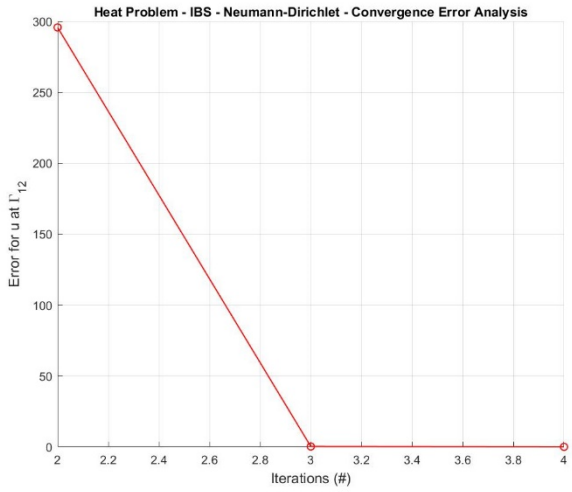


Figure 13 – Dirichlet–Neumann – $k_1 = 100$, $k_2 = 1$

As for the convergence analysis, it's interesting to see that the convergence in this particular case is much faster than the previous case (much below, an analytical explanation will be illustrated). The figures 14 and 15 confirm this fact (the number of iterations are equal to 3).



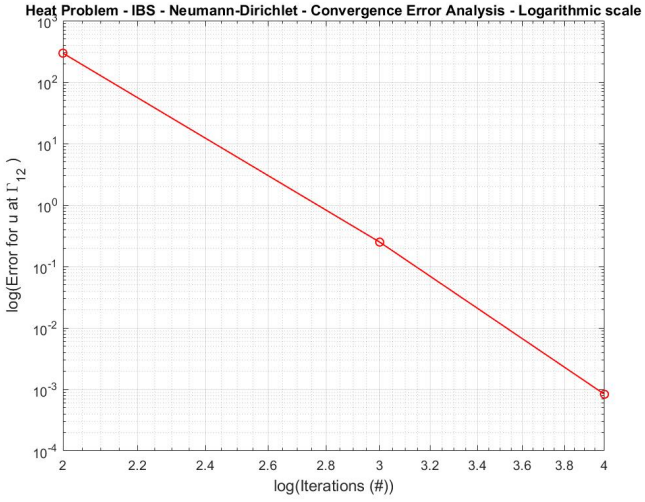


Figure 14 – Dirichlet–Neumann, $k_1=100$ / $k_2=1$ – Conv. Analysis

Figure 15 – Dirichlet–Neumann, $k_1=100$ / $k_2=1$ – Conv. Analysis (log. scale)

5.3 – Diffusivity $k_1 = (1/100)*k_2$

If one inverts this position and gives 100 to the diffusivity in the second subdomain (which makes the Dirichlet problem) and gives 1 to k_1 the situation is completely different: in this case the iterative scheme is completely unstable from the very beginning and the solution doesn't converge. In figure 16, the solution proposed is the one found in the secon iteration, which is the first iteration in which the solution blows up and does not converge anymore. The reason will be explained below the graphics.

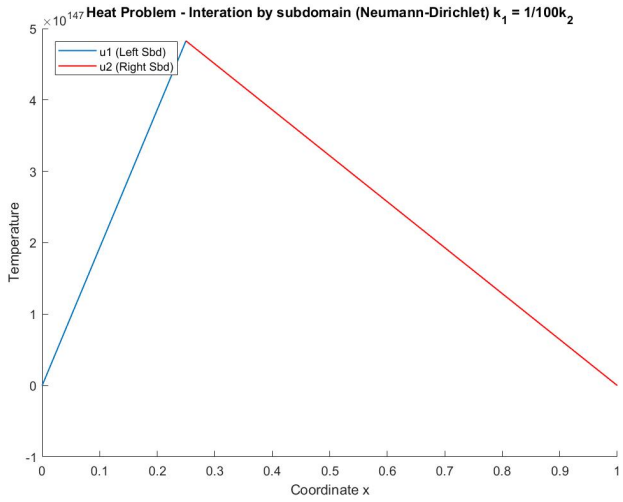


Figure 16 – Dirichlet–Neumann – $k_1 = 1$, $k_2 = 100$

The convergence analysis shows a huge value of the error which remains constant and does not decrease. This behavior is visible in both Cartesian and logarithmic plots (figure 17 and 18).

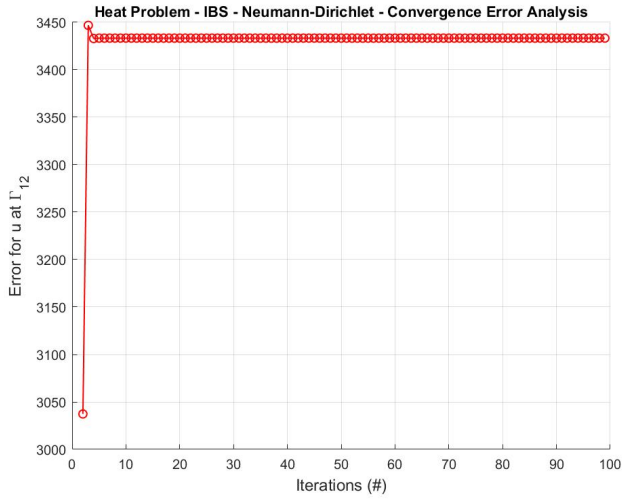


Figure 16 – Dirichlet–Neumann, k_1=1 / k_2=100 – Conv. Analysis

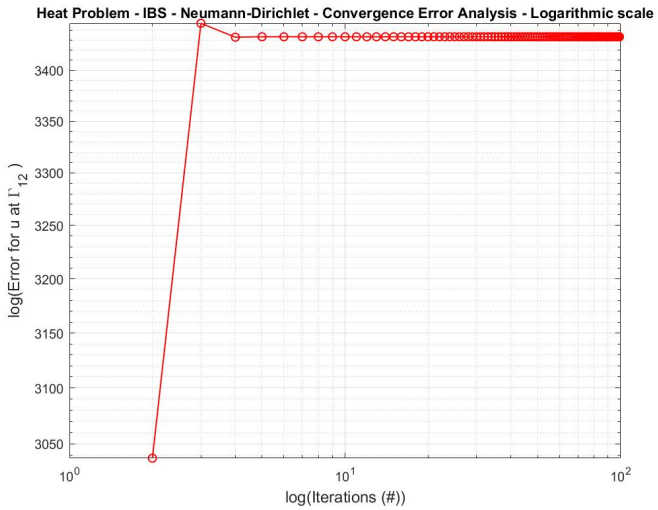


Figure 17 – Dirichlet–Neumann, k_1=1 / k_2=100 – Conv. Analysis (log. scale)

5.4 – Theoretical explaination

Given the Dirichlet-Neumann scheme (sequential (Gauss-Seidel) case):

$$\begin{cases} -k_1 \frac{\partial^2 u_1^i}{\partial x^2} = f_1 & in\ \Omega_1 \\ u_1 = 0 & on\ \partial\Omega_1 \backslash \Gamma_{12} \\ k_1 \frac{\partial u_1^i}{\partial x} = -k_2 \frac{\partial u_2^{i-1}}{\partial x} & on\ \Gamma_{12} \end{cases} \Rightarrow \begin{cases} -k_2 \frac{\partial^2 u_2^i}{\partial x^2} = f_1 & in\ \Omega_2 \\ u_2 = 0 & on\ \partial\Omega_2 \backslash \Gamma_{21} \\ u_2^i = u_1^i & on\ \Gamma_{21} \end{cases}$$

If the attention is paid in the left subdomain, the continuity of fluxes, as reported:

$$k_1 \frac{\partial u_1^i}{\partial x} = -k_2 \frac{\partial u_2^{i-1}}{\partial x}$$

Will make the program evaluate the spatial first derivative on the interface from the first subdomain point-of-view:

$$\frac{\partial u_1^i}{\partial x} = -\frac{k_2}{k_1} \frac{\partial u_2^{i-1}}{\partial x}$$

When the diffusion in subdomain 1 is equal to 100, and the same is equal to 1 on the subdomain 2, it means that $k_1 >> k_2$, or, in the same way that $\frac{k_2}{k_1}$ is very small, or at the limit, $\rightarrow 0^+$, this makes the left term decrease very fast and the convergence is reached very soon since the scheme becomes much more stable than the constant *k* over the whole domain.

On the other hand, when the diffusion in subdomain 1 is equal to 1, and the same is equal to 100 on the subdomain 2, it means that $k_1 << k_2$, or, in the same way that $\frac{k_2}{k_1}$ is very big, or at the limit, $\rightarrow +\infty$, this makes the left term increase instantaneously and the convergence cannot be reached since the scheme becomes unstable.

6. Relaxation scheme with fixed parameter ω

In order to decrease significantly the total number of iterations, a relaxation scheme is implemented to the classic IBS scheme. In this case, the Dirichlet B.C. on the interface for the second subdomain problem (Dirichlet problem) is relaxed using the transmission condition and the interface value evaluated at the previous iteration, this will make a faster convergence in the computational process.

$$u_{\Gamma 21}^i = \omega \cdot u_{\Gamma 12}^i + (1-\omega) \cdot u_{\Gamma 21}^{i-1}$$

6.1 – Diffusivity constant k = 1 in the whole domain (different ω)

In this case, different values for ω (which remains constant) have been given: if ω is equal to 0, as shown in figure 18, makes a disconnection of the transmission condition for the second subdomain, since there is no connection between the two subdomains. In this particular case, the whole process converges to a solution in one iteration because it's similar to a normal FE independent scheme. Except for the last figures, here the value of the diffusivity term *k* is = 1 in both sbds.

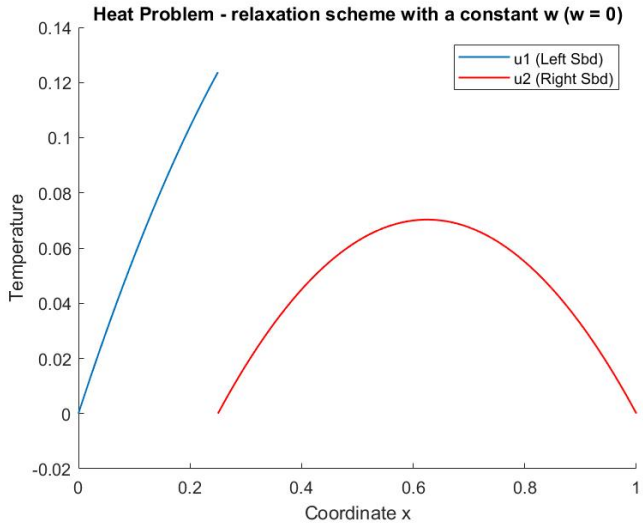


Figure 18 – Simple relaxation method (ω = 0)

For a small value of the ω (0.1) the number of iterations computed are very big (around 60). The figures 19 and 20 show this problematic case, in which the relaxation is not actuated since the value of its coefficient is too small.

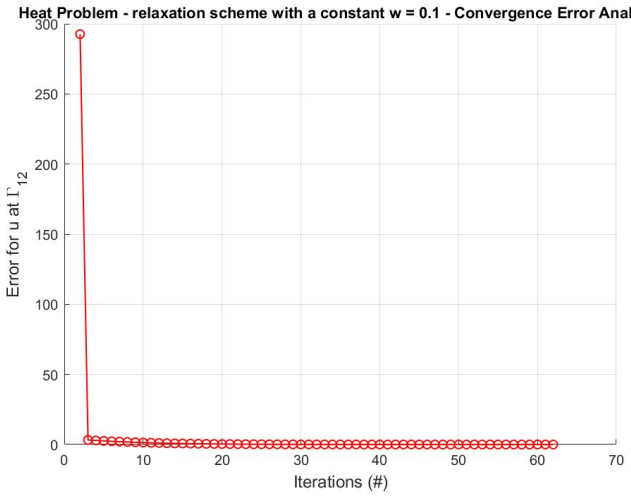


Figure 19 – Simple relaxation method (ω = 0.1) – Conv. Analysis

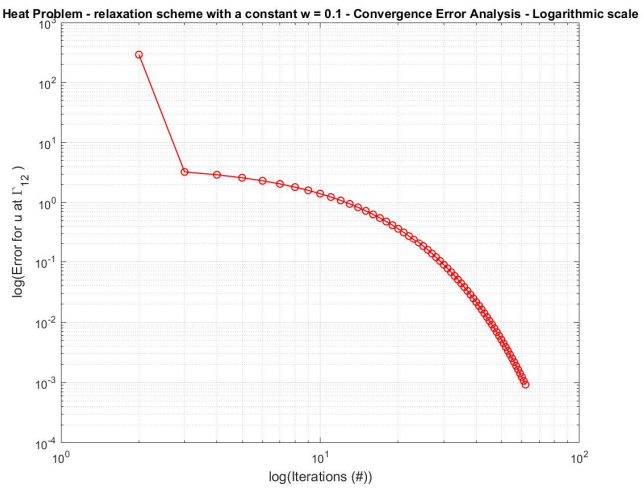


Figure 20 – Simple relaxation method (ω = 0.1) – Conv. Analysis (log. scale)

Increasing the value of ω to 0.25, the situation is much nicer, but still not optimal. Again the relaxation is not helping the convergence (# iterations = 26) [figures 21-22].

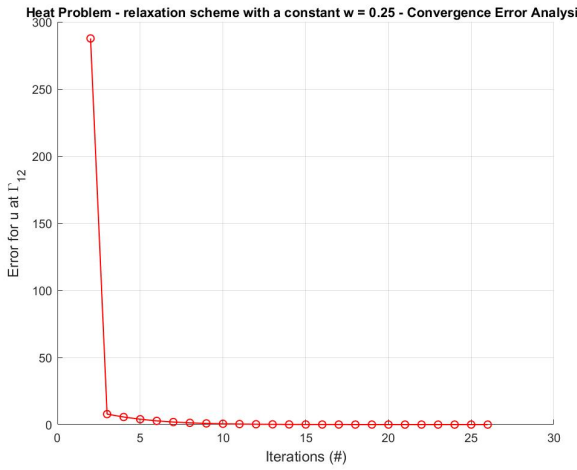


Figure 21 – Simple relaxation method (ω = 0.25) – Conv. Analysis

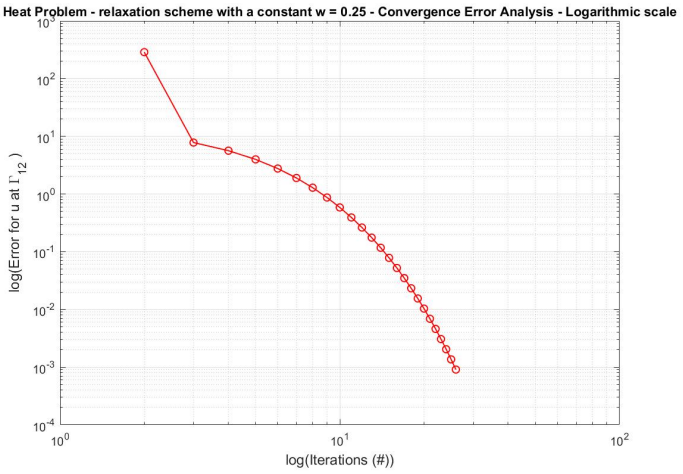


Figure 22 – Simple relaxation method (ω = 0.25) – Conv. Analysis (log. scale)

For ω equal to 0.5 it's possible to recover almost the same situation given in the D-N IBS scheme (# iterations = 12). The figures 23 and 24 contain less points.

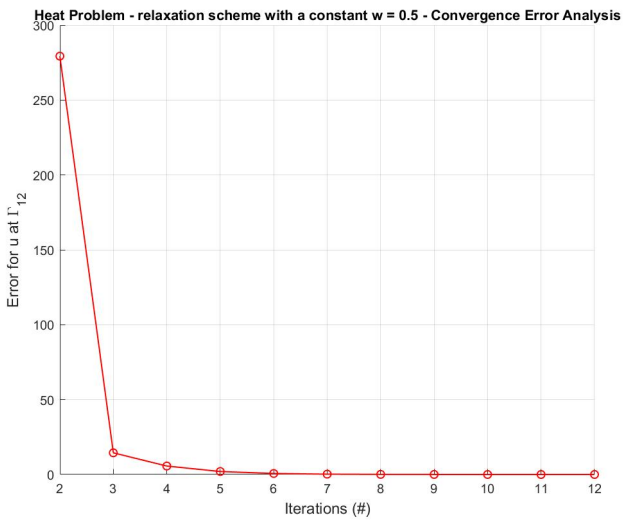


Figure 23 – Simple relaxation method ($\omega = 0.5$) – Conv. Analysis

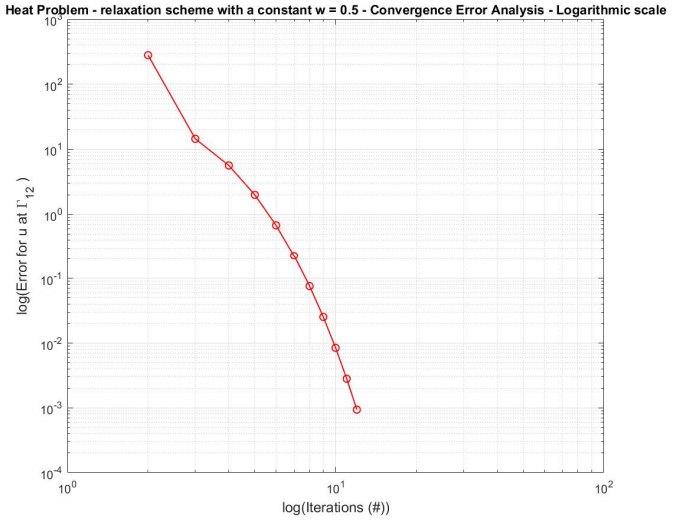


Figure 24 – Simple relaxation method ($\omega = 0.5$) – Conv. Analysis (log. scale)

It's expected to get an optimal solution if ω is equal to 0.75. In fact, the solution converges in 4 iterations, and this represents an optimal solution if one uses a relaxation method (figures 25 and 26).

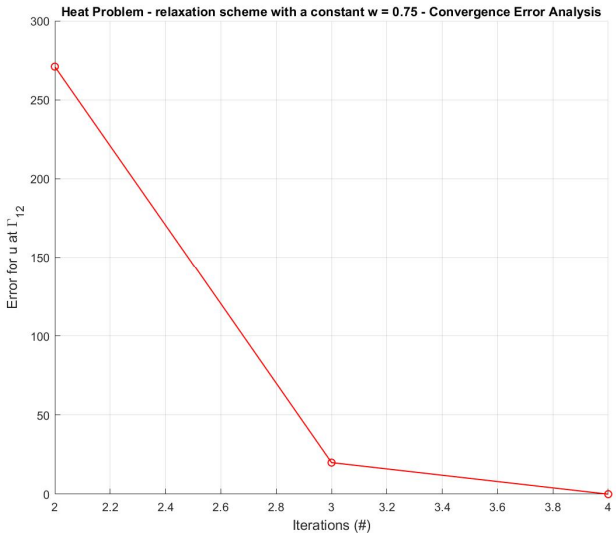


Figure 25 – Simple relaxation method ($\omega = 0.75$) – Conv. Analysis

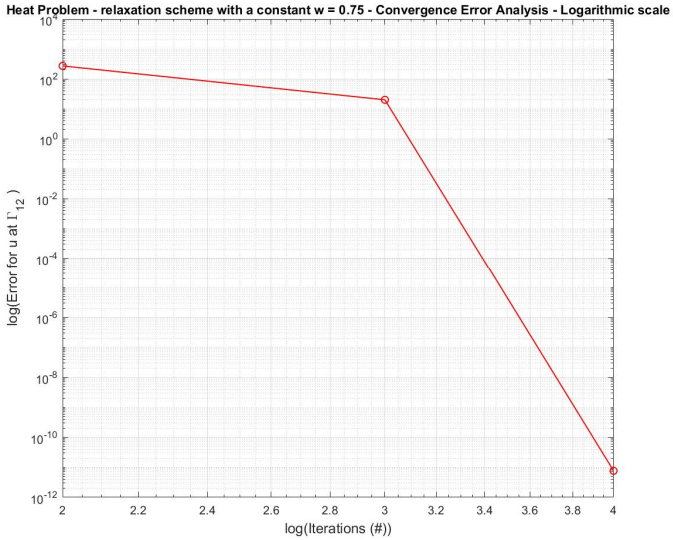


Figure 26 – Simple relaxation method ($\omega = 0.75$) – Conv. Analysis (log. scale)

Arriving to the value of 1, the relaxation mode becomes again the classic Dirichlet-Neumann IBS scheme (# iterations = 13). This is confirmed if one looks at the figures 27 and 28, which show its convergence analysis.

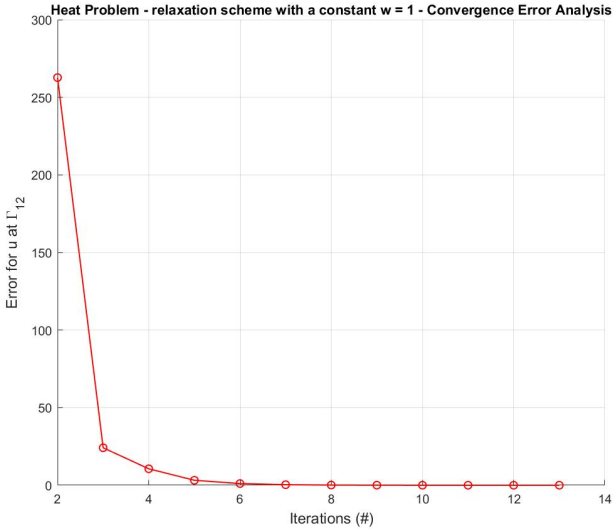


Figure 27 – Simple relaxation method ($\omega = 1$) – Conv. Analysis

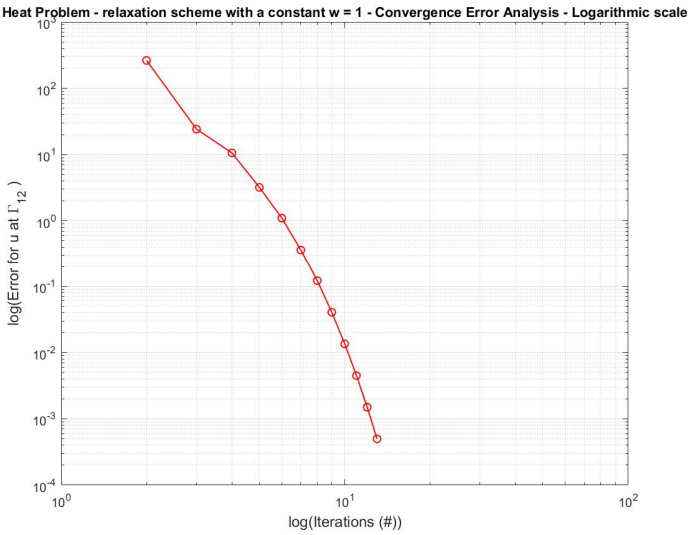


Figure 28 – Simple relaxation method ($\omega = 1$) – Conv. Analysis (log. scale)

6.2 – Diffusivity $k_1 = (1/100)*k_2$ (optimal $\omega = 0.75$)

As conclusion is possible to see that the relaxation method creates a faster solution in this case for a parameter ω around 0.75. Still this doesn't confirm that the method is stable always. In fact, for the problematic case in which $k_1 << k_2$ the solution is not reached, and the number of iterations blows up (figures 29-30-31).

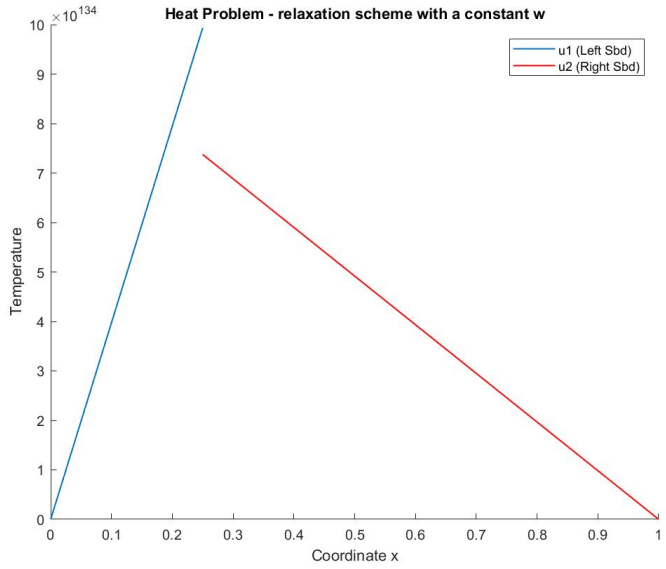
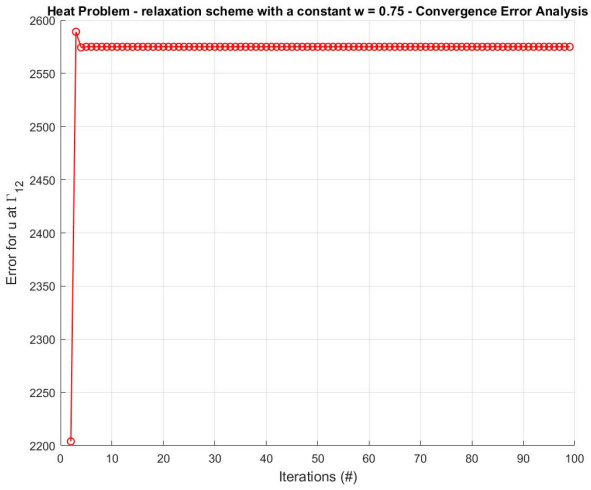


Figure 29 – Simple relaxation method ($\omega = 0.75$) – $k_1 = 1$, $k_2 = 100$

The jump seen in this figure (29) is due to the structure of the relaxation methods, which don't completely take the whole value coming from the left, but only a partition (0.75); this is the first total iteration in which the value of $u_{\Gamma 21}$ for the previous iteration is equal to 0.

The figures 30 and 31 show the impossibility to reach convergence for this problem. Need of a different relaxation method (Aitken).



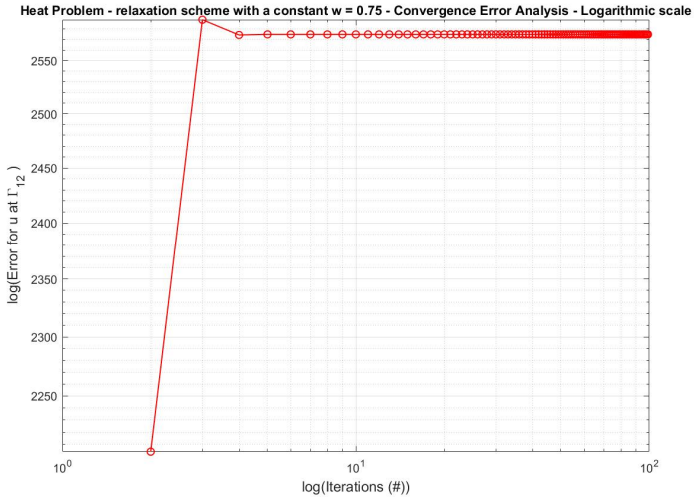


Figure 30 – Simple relaxation method ($\omega = 0.75$) – Conv. Analysis

Figure 31 – Simple relaxation method ($\omega = 0.75$) – Conv. Analysis (log. scale) [special case in which $k_1 = 1$ and $k_2 = 100$]

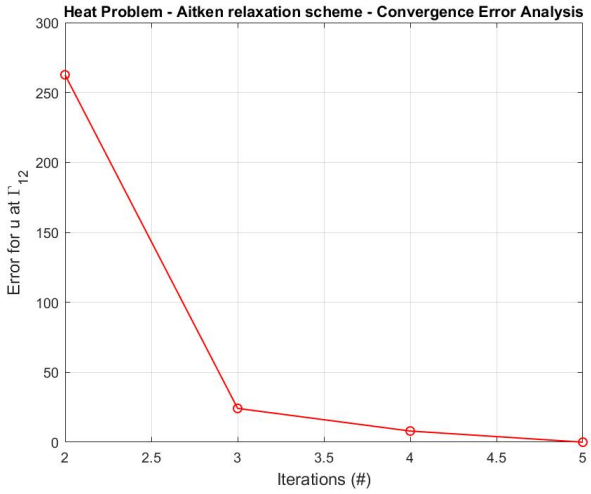
7. Aitken Relaxation scheme

The Aitken relaxation scheme represents a good solution which stabilizes many problematic codes. In this case, the value of the relaxation parameter ω is not constant except for the first two iterations. In the following iterations the computation for the ω is a combination of the values at the interface in both subdomains coming until from two iterations before: this means that the scheme provides at least 3 iterations in order to converge, but it converges very fast:

$$\omega = \frac{u_{\Gamma 21}^{i-1} - u_{\Gamma 21}^{i-2}}{(u_{\Gamma 21}^{i-1} - u_{\Gamma 21}^{i-2}) + (u_{\Gamma 12}^{i} - u_{\Gamma 12}^{i-1})}$$

7.1 – Diffusivity constant k = 1 in the whole domain

Computing the general case in which in both subdomains the diffusivity *k* = 1, it's possible to see (in figures 32 and 33) that the number of iterations is very low (= 5). The advantage of this method is that the choice of initial ω is free (except for 0 – wrong!), and the system search and finds the correct and optimal value of ω at each iteration.



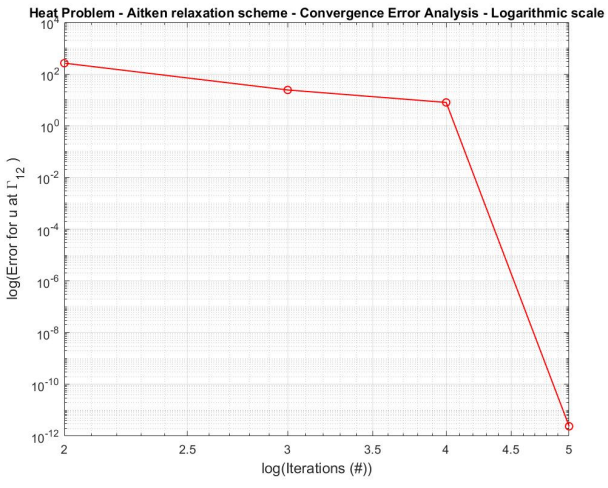


Figure 32 – Aitken relaxation method (k = 1) – Conv. Analysis

Figure 33 – Aitken relaxation method (k = 1) – Conv. Analysis (log. scale)

7.2 – Diffusivity $k_1 = (1/100)*k_2$

Once computed the second option (in which $k_1 = 1$ and $k_2 = 100$) it's possible to see that in this case the system converges, and it's found useful for this problematic cases. The number of iterations remains very small (5) and this means that this method is very adaptive. The solution and the convergence analysis are shown in figures 34, 35 and 36.

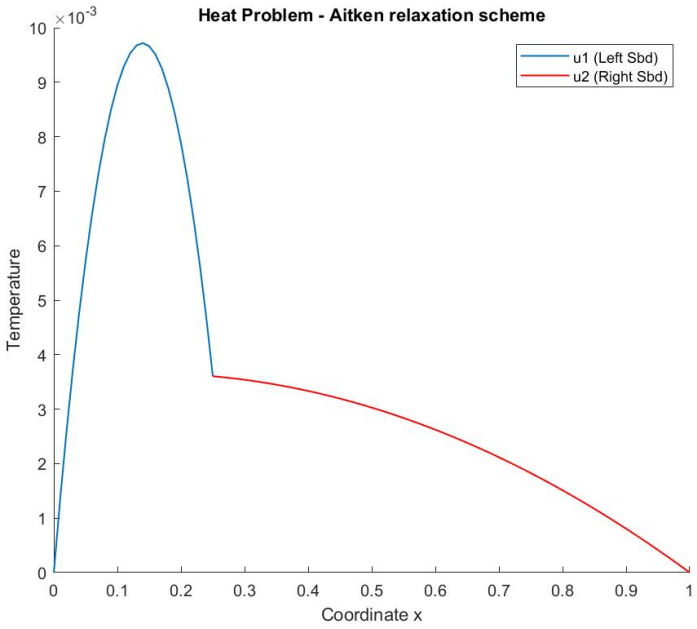
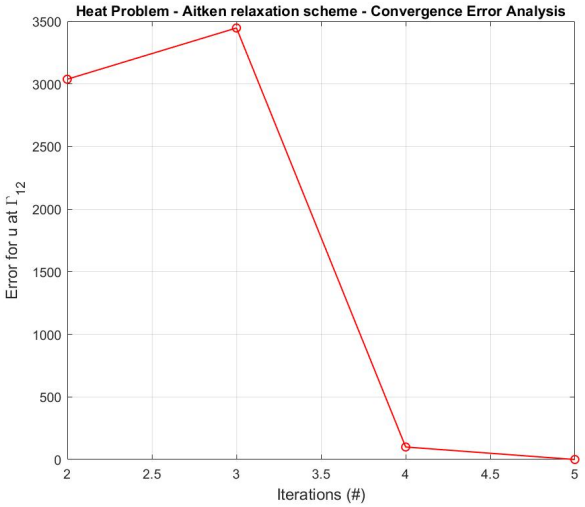


Figure 34 – Aitken relaxation method – $k_1 = 1$, $k_2 = 100$



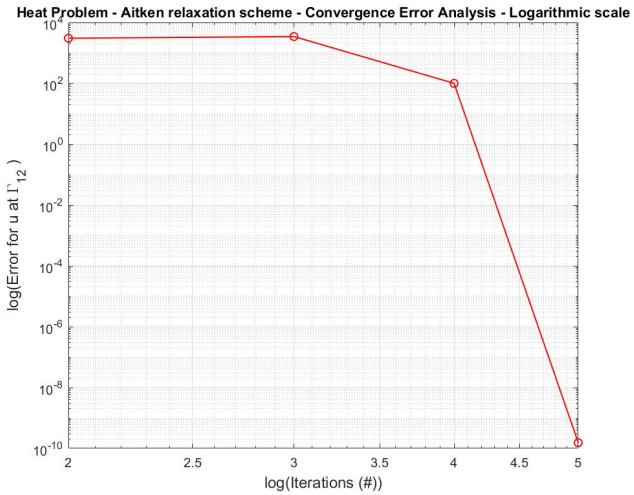


Figure 35 – Aitken relaxation method ($k_1 = 1$, $k_2 = 100$) – Conv. An. //// Figure 36 – Aitken relaxation method ($k_1 = 1$, $k_2 = 100$) – Conv. An. (log. scale)

8. Conclusions

As seen for this particular problem (Heat transfer problem), the advantage given by the classic iterative schemes are not so underlined for particular extremal cases. Solutions like Aitken can be a good advantage and make the system converge fast and for any case. As for a Monolithic scheme, it's possible to see that the system is larger than the others and this, sometimes, is a disadvantage for bigger and more complicated problems. The need of an iterative scheme for big problems is very much recommended (with sophisticated method like Aitken if needed).

9. Annex of the codes

A] Unique Domain – FEM

A1. Change value of *k*

```
clc
close all
clear variables

%Domain
Data.inix = 0;
Data.endx = 1;
Data.nelem = 100;

%Physical
kappa_vector = [1,2,4,8,16];
Data.source = 1;

%Boundary conditions
%Dirichlet
Data.FixLeft = 1;
Data.LeftValue = 0;
Data.FixRight = 1;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 25;

for i=1:length(kappa_vector)
Data.kappa = kappa_vector(i);
%Initialization
HeatProblem = HP_Initialize(Data);

%Building
HeatProblem = HP_Build(HeatProblem);

%Solving
HeatProblem = HP_Solve(HeatProblem);

%Plotting
HP_Plot(HeatProblem,1,1);
legendInfo{i} = sprintf('Solution u with kappa = %u',kappa_vector(i));
end
legend(legendInfo,'Location','best'); % Legend
title('Heat Problem - Unique domain') % Title
```

A2. Change value of *f*

```
clc
close all
clear variables

%Domain
Data.inix = 0;
Data.endx = 1;
Data.nelem = 100;

%Physical
Data.kappa = 1;
source_vector = [1,2,4,8,16];

%Boundary conditions
%Dirichlet
Data.FixLeft = 1;
Data.LeftValue = 0;
Data.FixRight = 1;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
```

```
Data.RightFluxes = 25;

for i=1:length(source_vector)
Data.source = source_vector(i);
%Initialization
HeatProblem = HP_Initialize(Data);

%Building
HeatProblem = HP_Build(HeatProblem);

%Solving
HeatProblem = HP_Solve(HeatProblem);

%Plotting
HP_Plot(HeatProblem,1,1);
legendInfo{i} = sprintf('Solution u with f = %u',source_vector(i));
end
legend(legendInfo,'Location','best'); % Legend
title('Heat Problem - Unique domain') % Title
```

A3. Change number of elements - convergence

```
clc
close all
clear variables

%Domain
Data.inix = 0;
Data.endx = 1;
nelem_vector =  [3 5 7 9 11 13 15 51 99];

%Physical
Data.kappa = 4;
Data.source = 10;

%Boundary conditions
%Dirichlet
Data.FixLeft = 1;
Data.LeftValue = 0;
Data.FixRight = 1;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 25;

for i=1:length(nelem_vector)
Data.nelem = nelem_vector(i);
%Initialization
HeatProblem = HP_Initialize(Data);

%Building
HeatProblem = HP_Build(HeatProblem);

%Solving
HeatProblem = HP_Solve(HeatProblem);

%Plotting
HP_Plot(HeatProblem,1,1);
legendInfo{i} = sprintf('Solution u with nelem = %u',nelem_vector(i));

Tmax(i) = max(HeatProblem.Solution.U); % Calculating the maximum value of the temperature
end
legend(legendInfo,'Location','best'); % Legend
title('Heat Problem - Unique domain') % Title

% Temperature max - analytical
T_an = -
Data.source/(2*Data.kappa)*((Data.endx+Data.inix)/2)^2+Data.source/(2*Data.kappa)*((Data.endx+Data.inix)/2);
err = abs(T_an - Tmax); % Error
size_mesh = 1./nelem_vector; % Size of the mesh

figure(2) % Plot Convergence - Cartesian
hold on
```

```
plot(size_mesh(1:end-1),err(1:end-1),'linewidth',1,'color','r','marker','o');
grid on
xlabel('Size Mesh','fontsize',11);
ylabel('Error','fontsize',11);
title('Heat Problem - Unique domain - Convergence Error Analysis')

figure(3) % Plot Convergence - LogLog
loglog(size_mesh(1:end-1),err(1:end-1),'linewidth',1,'color','r','marker','o');
grid on
xlabel('log(Size Mesh)','fontsize',11);
ylabel('log(Error)','fontsize',11);
title('Heat Problem - Unique domain - Convergence Error Analysis (logaritmic scale)')
```

B] Independent Subdomains – FEM

```
clc
close all
clear variables

%SubDomain1
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;

%SubDomain2
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;

%Physical
Data.kappa = 1;
Data.source = 1;
Data2.kappa = 1;
Data2.source = 1;

%Boundary conditions
%Dirichlet
%Subdomain1
Data.FixLeft = 1;
Data.LeftValue = 0;
Data.FixRight = 0;
Data.RightValue = 0;

%Subdomain2
Data2.FixLeft = 0;
Data2.LeftValue = 0;
Data2.FixRight = 1;
Data2.RightValue = 0;

%Neumann
%Subdomain1
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 0;

%Subdomain2
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;

%Problem 1
HeatProblem = HP_Initialize(Data); %Initialization
HeatProblem = HP_Build(HeatProblem); %Building
HeatProblem = HP_Solve(HeatProblem); %Solving

%Problem 2
HeatProblem2 = HP_Initialize(Data2); %Initialization
HeatProblem2 = HP_Build(HeatProblem2); %Building
HeatProblem2 = HP_Solve(HeatProblem2); %Solving

%Plotting
HP_Plot(HeatProblem,1,1); %Plot Problem1
HP_Plot(HeatProblem2,2,1); %Plot Problem2 (in the same figure)
legend('u1 (Left Sbd)','u2 (Right Sbd)','Location','northeast'); % Legend
title('Heat Problem - Independent subdomains (no t.c.)') % Title
```

C] Monolithic Scheme

```
clc
close all
clear variables

%SubDomain1
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;

%SubDomain2
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;

%Physical
Data.kappa = 1;
Data.source = 1;
Data2.kappa = 4;
Data2.source = 1;

%Boundary conditions
%Dirichlet
%Subdomain1
Data.FixLeft = 1;
Data.LeftValue = 0;
Data.FixRight = 0;
Data.RightValue = 0;

%Subdomain2
Data2.FixLeft = 0;
Data2.LeftValue = 0;
Data2.FixRight = 1;
Data2.RightValue = 0;

%Neumann
%Subdomain1
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 0;

%Subdomain2
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;

%Problem 1
HeatProblem = HP_Initialize(Data); %Initialization
HeatProblem = HP_Build(HeatProblem); %Building

%Problem 2
HeatProblem2 = HP_Initialize(Data2); %Initialization
HeatProblem2 = HP_Build(HeatProblem2); %Building

%Solve and plot
[HeatProblem,HeatProblem2] = HP_SolveMonolithic(HeatProblem,HeatProblem2); %Solving the monolithic
HP_Plot(HeatProblem,1,1); %Plot Problem1
HP_Plot(HeatProblem2,2,1); %Plot Problem2 (in the same figure)
legend('u1 (Left Sbd)','u2 (Right Sbd)','Location','northeast'); % Legend
title('Heat Problem - Monolithic (k1 = 1 and k2 = 4)') % Title
```

D] IBS Dirichlet-Neumann Scheme

```
clc
close all
clear variables

% Problem 1
% Geometry
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;
```

```
% Physical properties
Data.kappa = 1;
Data.source = 1;
% Boundary conditions - Dirichlet
Data.FixLeft = 1; % 0: do not fix, 1: fix
Data.LeftValue = 0;
Data.FixRight = 0;
Data.RightValue = 0;
% Boundary conditions - Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

% Problem 2
% Geometry
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
% Physical properties
Data2.kappa = 1;
Data2.source = 1;
% Boundary conditions - Dirichlet
Data2.FixLeft = 1; % 0: do not fix, 1: fix
Data2.LeftValue = 0;
Data2.FixRight = 1;
Data2.RightValue = 0;
% Boundary conditions - Neumann
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;

% Initialization
uGamma12_1 = 0;

% Cycle data initialization
i = 1; % Iteration Counter initialization
imax = 100; % Maximum iterations
err = 100; % Error initialization
tol = 10^-3; % Tolerance

while (i < imax && err > tol)

    % Problem 1
    HeatProblem = HP_Initialize(Data); % Initialization
    HeatProblem = HP_Build(HeatProblem); % Building
    HeatProblem = HP_Solve(HeatProblem); % Solving

    Data2.LeftValue = HeatProblem.Solution.uRight;

    % Problem 2
    HeatProblem2 = HP_Initialize(Data2); % Initialization
    HeatProblem2 = HP_Build(HeatProblem2); % Building
    HeatProblem2 = HP_Solve(HeatProblem2); % Solving

    % Updating the flux going to Sbd 1 (for the following step)
    Data.RightFluxes = -HeatProblem2.Solution.FluxesLeft;

    % Error calculation
    err = abs(abs(HeatProblem.Solution.uRight - uGamma12_1)/uGamma12_1)*100;
    fprintf('The current error is: %f\n',err);
    err_vector(i) = err;

    % Recalculating the interface value (Sbd 1) for the following step (it will be i-1 in the next iteration)
    uGamma12_1 = HeatProblem.Solution.uRight;

    % Incrementation of the iteration index
    i = i + 1;
end

% Plotting
HP_Plot(HeatProblem,1,1); %Plot Problem1
HP_Plot(HeatProblem2,2,1); %Plot Problem2 (in the same figure)
legend('u1 (Left Sbd)','u2 (Right Sbd)','Location','northwest'); % Legend
title('Heat Problem - Interation by subdomain (Neumann-Dirichlet) k_1 = 100k_2') % Title

figure(2) % Plot of the convergence - Cartesian
```

```
hold on
plot(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('Iterations (#)','fontsize',12);
ylabel('Error for u at \Gamma_1_2','fontsize',12);
title('Heat Problem - IBS - Neumann-Dirichlet - Convergence Error Analysis')

figure(3) % Plot of the convergence - LogLog
%hold on
loglog(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('log(Iterations (#))','fontsize',12);
ylabel('log(Error for u at \Gamma_1_2 )','fontsize',12);
title('Heat Problem - IBS - Neumann-Dirichlet - Convergence Error Analysis - Logarithmic scale')
```

E] Relaxation Scheme with constant ω

```
clc
close all
clear variables

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

% Problem 2
% Geometry
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
% Physical properties
Data2.kappa = 100;
Data2.source = 1;
% Boundary conditions - Dirichlet
Data2.FixLeft = 1; % 0: do not fix, 1: fix
Data2.LeftValue = 0;
Data2.FixRight = 1;
Data2.RightValue = 0;
% Boundary conditions - Neumann
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;

% Initialization
w = 0.75; % Initial relaxation
uGamma21_1 = 0; %Initial Sbd 2 interface value at i-1 / = 0
uGamma21_2 = 0; %Initial Sbd 2 interface value at i-2 / = 0
uGamma12_1 = 0; %Initial Sbd 1 interface value at i-1 / = 0

% Cycle data initialization
i = 1; % Iteration Counter initialization
imax = 100; % Maximum iterations
err = 100; % Error initialization
tol = 10^-3; % Tolerance


% Cycle "while"
 while (i < imax && err > tol)
```

```
    % Problem 1
    HeatProblem = HP_Initialize(Data); % Initialization
    HeatProblem = HP_Build(HeatProblem); % Building
    HeatProblem = HP_Solve(HeatProblem); % Solving


    % Calculation of uGamma21 at the current iteration
    Data2.LeftValue = w*HeatProblem.Solution.uRight + (1-w)*uGamma21_1;

    % Updating the interface values from the Sbd 2
    uGamma21_2 = uGamma21_1; % the i-1 becomes i-2 for the following step
    uGamma21_1 = Data2.LeftValue; % the i becomes i-1 for the following step

    % Problem 2
    HeatProblem2 = HP_Initialize(Data2); % Initialization
    HeatProblem2 = HP_Build(HeatProblem2); % Building
    HeatProblem2 = HP_Solve(HeatProblem2); % Solving

    % Updating the flux going to Sbd 1 (for the following step)
    Data.RightFluxes = -HeatProblem2.Solution.FluxesLeft;

    % Error calculation
    err = abs(abs(HeatProblem.Solution.uRight - uGamma12_1)/uGamma12_1)*100;
    fprintf('The current error is: %f\n',err);
    err_vector(i) = err;

    % Recalculating the interface value (Sbd 1) for the following step (it will be i-1 in the next iteration)
    uGamma12_1 = HeatProblem.Solution.uRight;

    % Incrementation of the iteration index
    i = i + 1;

 end

% Plotting
HP_Plot(HeatProblem,1,1); %Plot Problem1
HP_Plot(HeatProblem2,2,1); %Plot Problem2 (in the same figure)
legend('u1 (Left Sbd)','u2 (Right Sbd)','Location','northeast'); % Legend
title('Heat Problem - relaxation scheme with a constant w') % Title

figure(2) % Plot of the convergence - Cartesian
hold on
plot(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('Iterations (#)','fontsize',12);
ylabel('Error for u at \Gamma_1_2','fontsize',12);
title(['Heat Problem - relaxation scheme with a constant w = ',num2str(w),' - Convergence Error Analysis'])

figure(3) % Plot of the convergence - LogLog
%hold on
loglog(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('log(Iterations (#))','fontsize',12);
ylabel('log(Error for u at \Gamma_1_2 )','fontsize',12);
title(['Heat Problem - relaxation scheme with a constant w = ',num2str(w),' - Convergence Error Analysis -
Logarithmic scale'])
```

F] Aitken Relaxation Scheme

```
clc
close all
clear variables

% Problem 1
% Geometry
Data.inix = 0;
Data.endx = 0.25;
Data.nelem = 25;
% Physical properties
Data.kappa = 1;
Data.source = 1;
% Boundary conditions - Dirichlet
Data.FixLeft = 1; % 0: do not fix, 1: fix
Data.LeftValue = 0;
Data.FixRight = 0;
```

```
Data.RightValue = 0;
% Boundary conditions - Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

% Problem 2
% Geometry
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
% Physical properties
Data2.kappa = 100;
Data2.source = 1;
% Boundary conditions - Dirichlet
Data2.FixLeft = 1; % 0: do not fix, 1: fix
Data2.LeftValue = 0;
Data2.FixRight = 1;
Data2.RightValue = 0;
% Boundary conditions - Neumann
Data2.FixFluxesLeft = 0;
Data2.LeftFluxes = 0;
Data2.FixFluxesRight = 0;
Data2.RightFluxes = 0;

% Initialization
w = 1; % Initial relaxation
uGamma21_1 = 0; %Initial Sbd 2 interface value at i-1 / = 0
uGamma21_2 = 0; %Initial Sbd 2 interface value at i-2 / = 0
uGamma12_1 = 0; %Initial Sbd 1 interface value at i-1 / = 0

% Cycle data initialization
i = 1; % Iteration Counter initialization
imax = 100; % Maximum iterations
err = 100; % Error initialization
tol = 10^-3; % Tolerance

% Cycle "while"
 while (i < imax && err > tol)

    % Problem 1
    HeatProblem = HP_Initialize(Data); % Initialization
    HeatProblem = HP_Build(HeatProblem); % Building
    HeatProblem = HP_Solve(HeatProblem); % Solving

    % Calculation of Aitken relaxation (if iter counter is > 2)
    if i > 2
        w = (uGamma21_2 - uGamma21_1)/(uGamma21_2 - uGamma21_1 + HeatProblem.Solution.uRight - uGamma12_1);
    end

    % Calculation of uGamma21 at the current iteration
    Data2.LeftValue = w*HeatProblem.Solution.uRight + (1-w)*uGamma21_1;

    % Updating the interface values from the Sbd 2
    uGamma21_2 = uGamma21_1; % the i-1 becomes i-2 for the following step
    uGamma21_1 = Data2.LeftValue; % the i becomes i-1 for the following step

    % Problem 2
    HeatProblem2 = HP_Initialize(Data2); % Initialization
    HeatProblem2 = HP_Build(HeatProblem2); % Building
    HeatProblem2 = HP_Solve(HeatProblem2); % Solving

    % Updating the flux going to Sbd 1 (for the following step)
    Data.RightFluxes = -HeatProblem2.Solution.FluxesLeft;

    % Error calculation
    err = abs(abs(HeatProblem.Solution.uRight - uGamma12_1)/uGamma12_1)*100;
    fprintf('The current error is: %f\n',err);
    err_vector(i) = err;

    % Recalculating the interface value (Sbd 1) for the following step (it will be i-1 in the next iteration)
    uGamma12_1 = HeatProblem.Solution.uRight;

    % Incrementation of the iteration index
    i = i + 1;

 end
```

```
% Plotting
HP_Plot(HeatProblem,1,1); %Plot Problem1
HP_Plot(HeatProblem2,2,1); %Plot Problem2 (in the same figure)
legend('u1 (Left Sbd)','u2 (Right Sbd)','Location','northeast'); % Legend
title('Heat Problem - Aitken relaxation scheme') % Title

figure(2) % Plot of the convergence - Cartesian
plot(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('Iterations (#)','fontsize',12);
ylabel('Error for u at \Gamma_1_2','fontsize',12);
title('Heat Problem - Aitken relaxation scheme - Convergence Error Analysis')

figure(3) % Plot of the convergence - LogLog
%hold on
loglog(1:i-1,err_vector,'linewidth',1,'color','r','marker','o');
grid on
xlabel('log(Iterations (#))','fontsize',12);
ylabel('log(Error for u at \Gamma_1_2 )','fontsize',12);
title('Heat Problem - Aitken relaxation scheme - Convergence Error Analysis - Logarithmic scale')
```

G] *Change* in *HP_Plot.m*

```
function HP_Plot(HeatProblem,n,fignum)

    figure(fignum)
    hold on
    if n == 1 %number of sbd
    plot(HeatProblem.Solution.coord,HeatProblem.Solution.U,'linewidth',1);
    elseif n == 2 %number of sbd
    plot(HeatProblem.Solution.coord,HeatProblem.Solution.U,'linewidth',1,'color','r');
    end
    xlabel('Coordinate x','fontsize',11);
    ylabel('Temperature','fontsize',11);
end
```