Coupled Problems Computer Assignment

By
Ahmed Saeed Sherif

May 18, 2018

Contents

1 Question 1: Solving a single heat problem

(a) Studying the effect of changing the value of the thermal diffusion coefficient

It is noticed that the higher the value of the diffusion coefficient, the more diffusive the solution is. This means that as the diffusion coefficient increase, the maximum temperature in the domain is going to be smaller, see Figure 1.

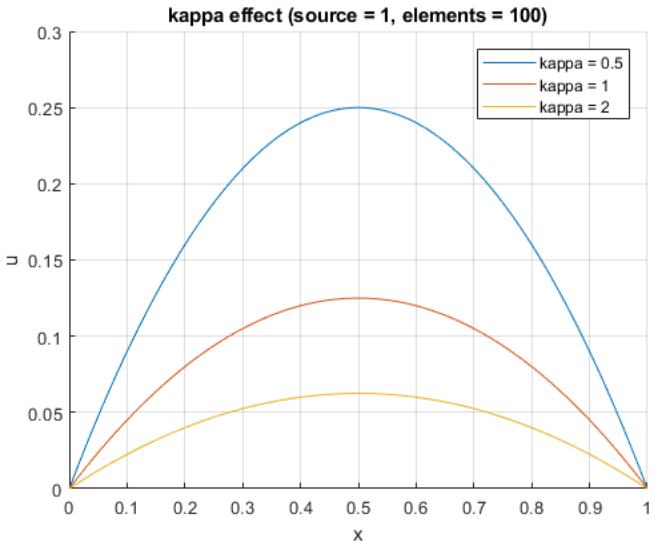

Figure 1: Effect of diffusion coefficient (kappa)

(b) Studying the effect of changing the source term value

By increasing the value of the source term, the temperature profile is increased as seen in Figure 2.

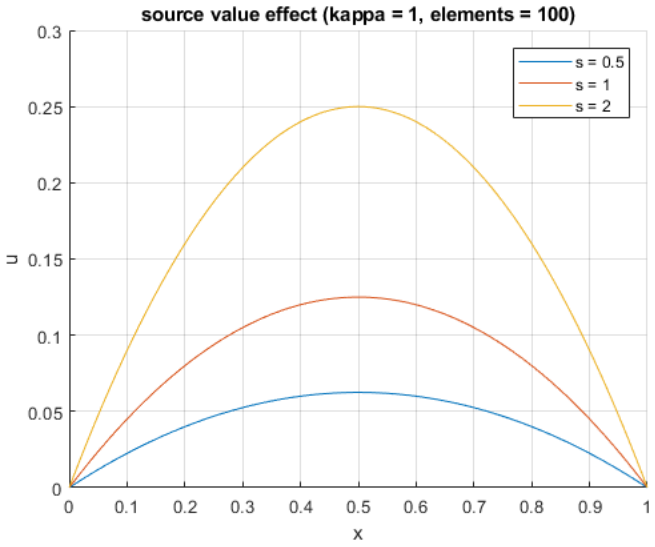

Figure 2: Effect of source term value (s)

(c) **Studying the effect of changing the number of elements**

By increasing the number of elements, the accuracy of the solution is increased as seen in Figure 3. The effect of changing the number of elements is studied by observing the convergence of the maximum temperature value which happens for this problem to be in the middle of the domain. For this, different number of elements have been used and the error in the maximum value of the temperature has been recorded and plotted against the element size using a log-log plot as seen in Figure 4. It is observed that the optimal convergence rate $(p+1)$ is obtained, for p being the order of the finite element. The slope of the plot is 2 because linear elements were used, i.e. $p = 1$. In this analysis, it is important to use odd number of elements so that no node is positioned in the middle of the domain, otherwise the maximum value of the temperature associated to that node would be exact and the analysis wouldn't give meaningful results.

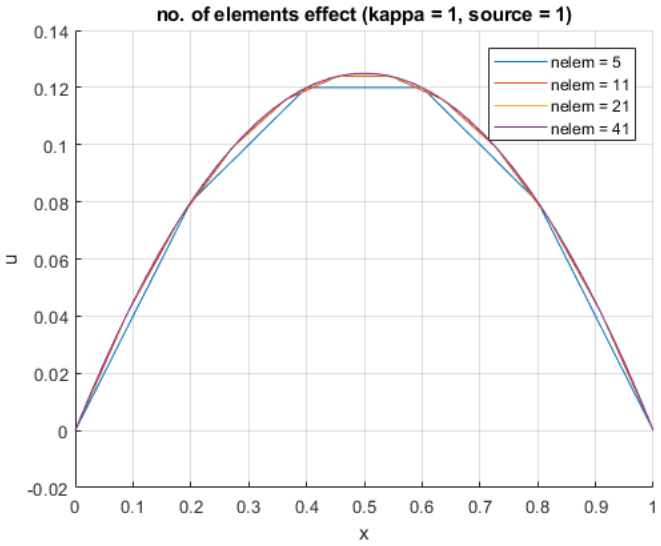

Figure 3: Effect of number of elements (nelem)

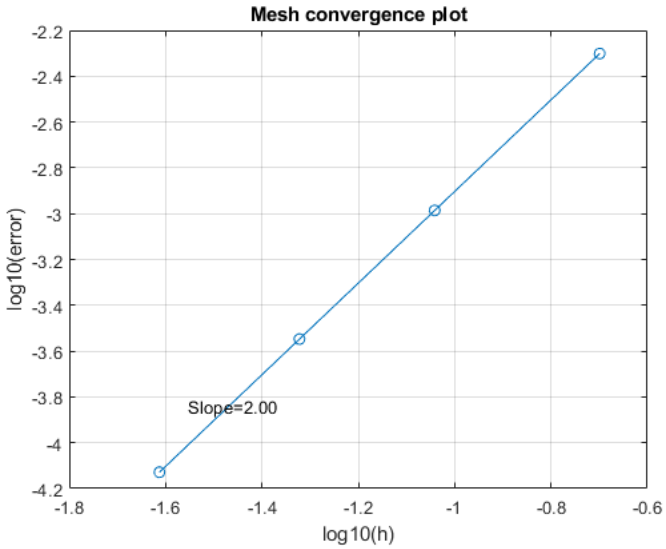

Figure 4: Mesh convergence plot

2 Question 2: Solving two independent heat problems in two adjacent subdomains with no conditions applied at the interface

By solving two independent problems in two adjacent subdomains without imposing any transmission condition at the interface, the obtained solution is shown in Figure 5. It is clearly seen that the solution at the interface doesn't match. This happens because there is no restriction from one problem on the other. This in fact corresponds to solving the two problems with zero Neumann flux at the interface, that's why the slope of the solution is zero at the interface from both sides.

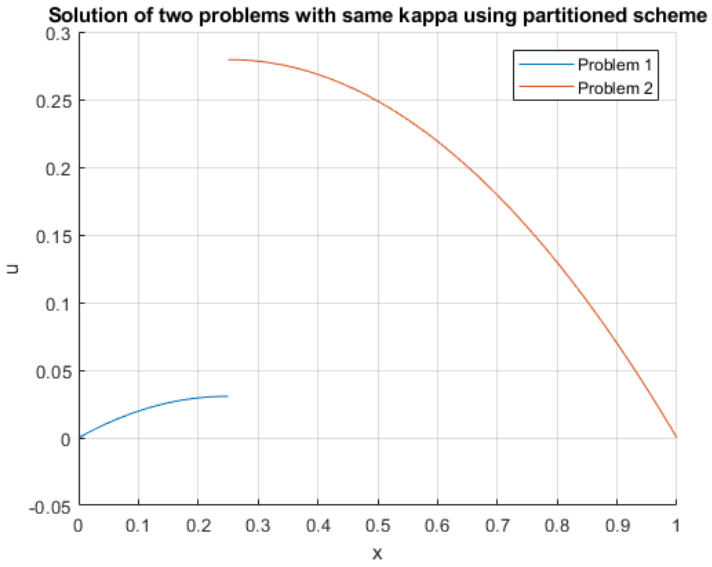

Figure 5: Solution of two uncoupled problems

3 Question 3: Solving two heat problems in two adjacent subdomains in a Monolithic way

(a) **Solving using the Monolithic solver with same κ parameter in the two subdomains**

By solving the two problems using a Monolithic solver, the obtained solution is shown in Figure 6. It is clearly seen that the solution matches at the interface. The reason is that the Monolithic solver solves the two problems as one and combines the two degrees of freedom at the interface from both subdomains into one degree of freedom, this in fact assure the continuity of the solution and the fluxes at the interface. Since the flux is defined as $\boldsymbol{n} \cdot \kappa \boldsymbol{\nabla} u$, and since the value of κ is the same for both subdomains, this means that the gradient of the solution (the slope) will be continuous at the interface.

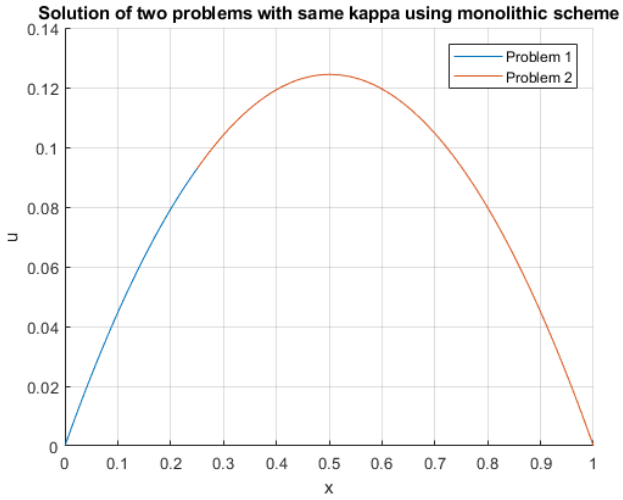

Figure 6: Solution of the two problems (same κ) using Monolithic solver

(b) **Solving using the Monolithic solver with different κ parameter in the two subdomains**

By setting the value of κ to different values in the two subdomain, the obtained solution is shown in Figure 7. Again, it is noticed that the solution matches at the interface, but the slope of the solution is now discontinuous at the interface. the reason is that the fluxes are continuous while κ is different, so the slope varies on each side to assure the continuity of the fluxes at the interface.

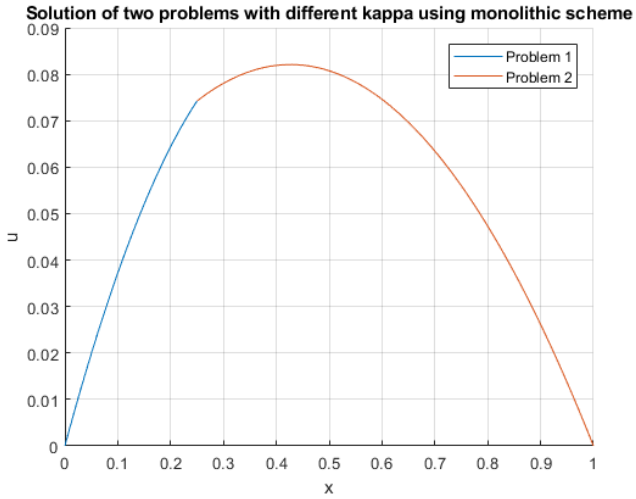

Figure 7: Solution of the two problems (different κ) using Monolithic solver

4 Question 4: Solving two heat problems in two adjacent subdomains using Dirchlet-Neumann iterations

(a) **Evaluating the convergence of the iterative scheme**

The iterative scheme is designed so that Neumann boundary conditions (from the second subdomain) are applied at the interface in the first (left) subdomain, and Dirichlet boundary conditions (from the first subdomain) at the interface in the second (right) subdomain. The iterations goes on until convergence to a certain tolerance (∼1e-6) is achieved. The obtained solution is shown in Figure 8. It is noticed that the solution is very much the same as the solution obtained using the Monolithic solver shown in Figure 6. The converged solution was obtained after 13 iterations and the difference in the solution at the interface from the left and the right is (5.231926e-15).

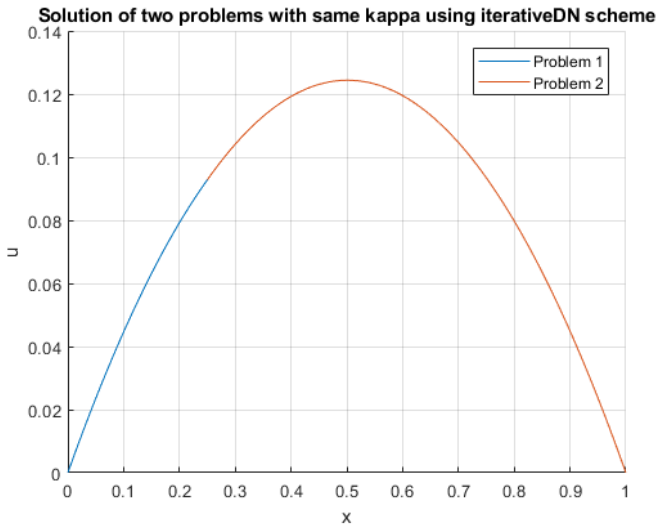

Figure 8: Solution of the two problems ($\kappa_1 = \kappa_2 = 1$) using iterative solver

(b) **Increasing the value for kappa at subdomain 1 (x100)**

By increasing the value of κ_1 (the diffusion coefficient in subdomain 1) to 100 and keeping κ_2 equal to 1, the obtained solution is shown in Figure 9. The convergence of the scheme was much faster as it took only 4 iterations to converge to the selected tolerance (1e-6). The jump in the solution at the interface is (9.202708e-16).

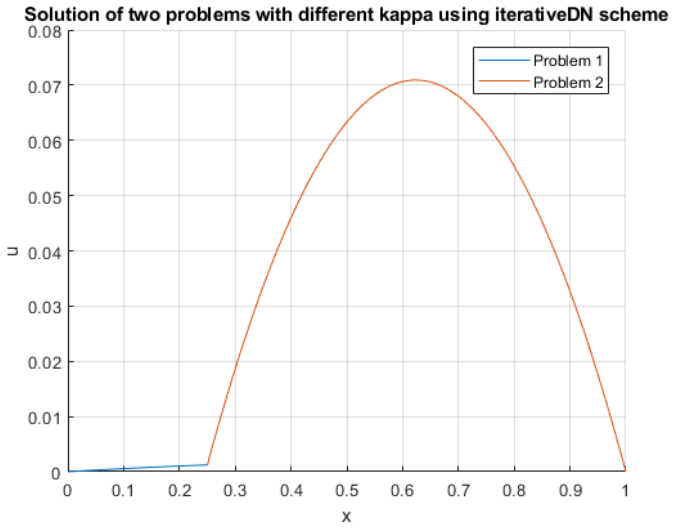

Figure 9: Solution of the two problems ($\kappa_1 = 100, \kappa_2 = 1$) using iterative solver

(c) **Diminishing the value for kappa at subdomain 1 (/100)**

By decreasing the value of κ_1 to 0.01 and keeping κ_2 equal to 1, the iterative scheme simply doesn't converge, see Figure 10 that shows the obtained solution after 100 iterations. The solution grows to (∼4.8e+149). The solution at the interface is in fact not-matching, where the jump in the solution is (∼2.6e+135).

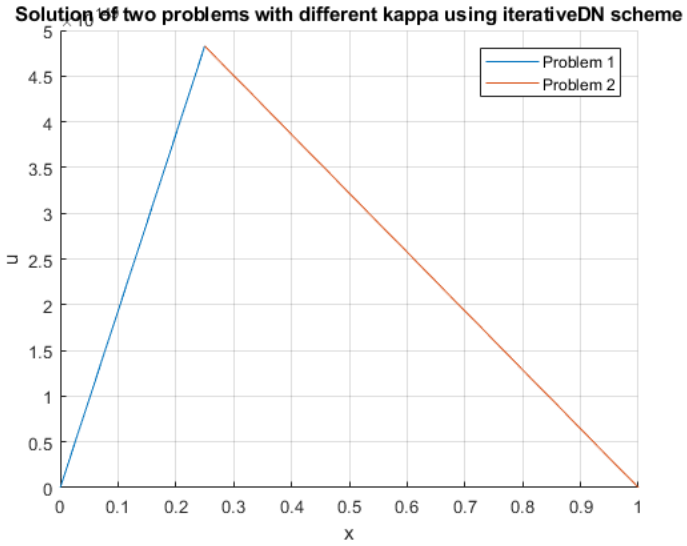

Figure 10: Solution of the two problems ($\kappa_1 = 0.01, \kappa_2 = 1$) using iterative solver

(d) **Stability of the coupling scheme**

As seen from the results shown in Figures 9 and 10, the Dirchlet-Neumann iteration scheme gives stable solution only when the Dirchlet conditions are applied at the interface to the subdomain of the lower diffusion coefficient κ. This in fact is similar to the added mass effect in fluid-structure interaction problems where the Dirchlet boundary conditions should be applied to the subdomain of the higher density.

In order for the scheme to work in the case shown in point (c), a relaxation scheme is used, where the Dirchlet boundary condition applied at the interface of subdomain 2 is computed from the solution from subdomain 1 at the current iteration and the solution at the interface of subdomain 2 at the previous iteration.

5 Question 5: Relaxation for the iterative Dirchlet-Neumann coupling

(a) **Relaxation scheme in terms of a fixed relaxation parameter** w

In a relaxation scheme with a fixed relaxation parameter w, the Dirchlet boundary condition applied to the second (right) subdomain is relaxed, meaning that the value obtained from subdomain 1 is not applied automatically, but an average of it and the solution from the previous iteration is applied. The implemented code is shown in Appendix D.

- **Same values of diffusion coefficient in the two subdomains,** $\kappa_1 = \kappa_2 = 1$

Using a fixed relaxation parameter (w) for the case of equal κ gives a stable solution similar to what was obtained before without relaxation in Figure 8. Setting $w = 1$ means no relaxation and exactly the same solution as shown in Figure 8 is obtained in 13 iterations. It is important to note that the relaxation parameter should be chosen closer to 1 to obtain the solution in less iterations. For example, setting $w = 0.8$ yields convergence after 7 iterations (jump at the interface is 2.381345e-08) while setting $w = 0.1$ yields convergence after 68 iterations (jump at the interface is 8.514346e-06). As a conclusion, the use of a fixed relaxation scheme with a parameter closer to 1 improves the convergence compared to the case of having no relaxation.

- **Different values of diffusion coefficient in the two subdomains, $\kappa_1 = 100$ and $\kappa_2 = 1$**

  The case of having $\kappa_1 > \kappa_2$ in fact doesn't need relaxation because it already gives a stable solution without relaxation as was shown earlier in Figure 9. However, if a fixed relaxation scheme is used, the parameter w should be chosen closer to 1. By setting the value of w to 1, 0.8, and 0.1, the converged solutions is obtained in 4, 7, and 48 iterations, respectively, and the jumps in the solution at the interface are 9.202708e-16, 5.904766e-08, and 8.382958e-06, respectively. This shows that the drawbacks of using a fixed relaxation for this case.

- **Different values of diffusion coefficient in the two subdomains, $\kappa_1 = 0.01$ and $\kappa_2 = 1$**

  For this case, the use of the relaxation scheme is of a great benefit where it allows to obtain a stable solution, unlike what was shown earlier in Figure 10. It is important for this case that we choose a small value for the relaxation parameter. By setting $w = 0.01$, the converged solution shown in Figure 11 is obtained after 30 iterations and the jump in the solution at the interface is (8.599098e-05). It is also observed that the stability of the scheme is very sensitive to the value of w, where the highest possible value that could be used to obtain a stable solution is $w \approx 0.054$ in about 95 iterations.

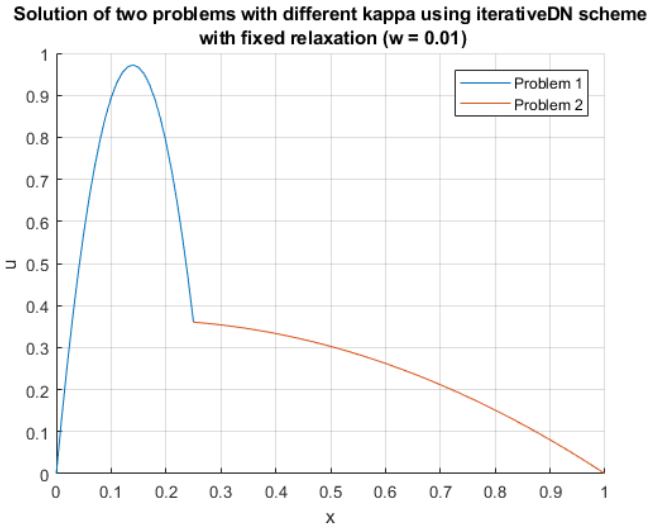

Figure 11: Solution of the two problems ($\kappa_1 = 0.01, \kappa_2 = 1$) using iterative solver with fixed relaxation ($w = 0.01$)

(b) **Aitken relaxation scheme**

Aitken relaxation scheme overcomes the drawbacks of the fixed relaxation scheme, where the value of the relaxation parameter (w) is computed automatically from the solution at the interface from both subdomains considering two previous iterations and the current iteration. Therefore, there is no need to adjust the parameter w manually. The implemented code is shown in Appendix D.

- **Same values of diffusion coefficient in the two subdomains,** $\kappa_1 = \kappa_2 = 1$

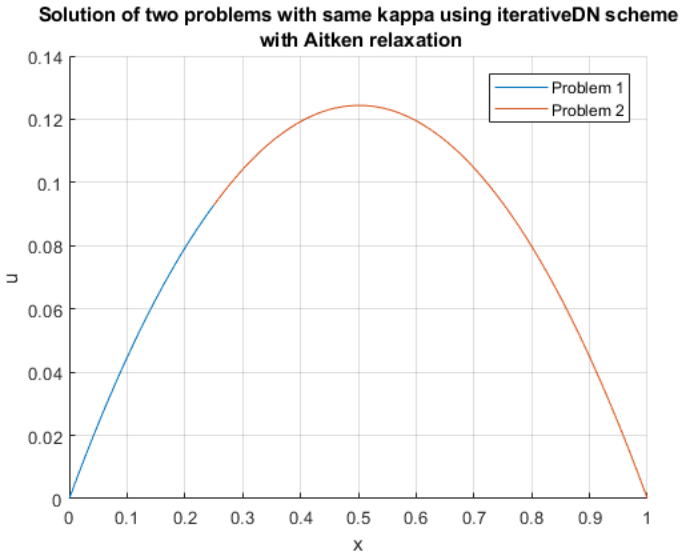

Figure 12: Solution of the two problems ($\kappa_1 = \kappa_2 = 1$) using iterative solver with Aitken relaxation

- **Different values of diffusion coefficient in the two subdomains,** $\kappa_1 = 100$ **and** $\kappa_2 = 1$

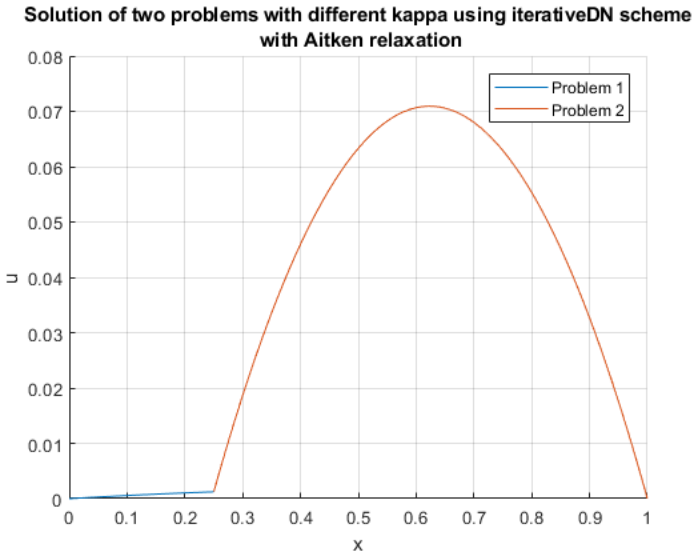

Figure 13: Solution of the two problems ($\kappa_1 = 100, \kappa_2 = 1$) using iterative solver with Aitken relaxation

- **Different values of diffusion coefficient in the two subdomains,** $\kappa_1 = 0.01$ **and** $\kappa_2 = 1$

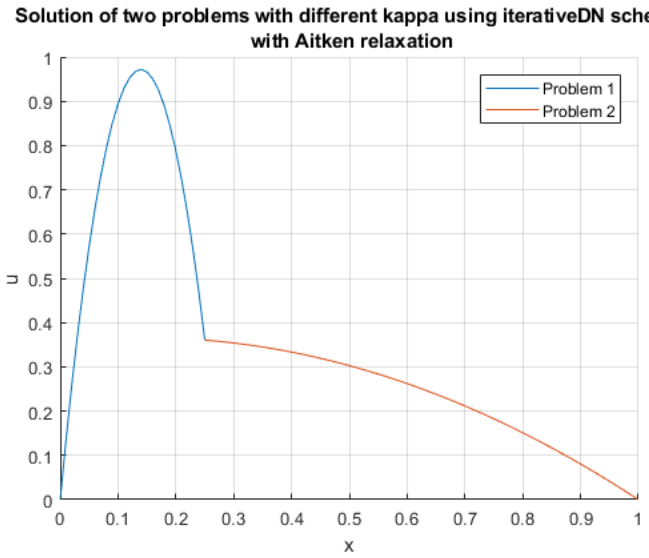

Figure 14: Solution of the two problems ($\kappa_1 = 0.01, \kappa_2 = 1$) using iterative solver with Aitken relaxation

In all of the three cases, the converged solutions were obtained in only 2 iterations which shows the huge improvement in terms of convergence. The last case where $\kappa_1 < \kappa_2$ is the one with the highest benefit from this scheme where the number of iterations is reduced from 30 to 2 when compared to the fixed relaxation scheme with $w = 0.01$. It is also worth noting that the absolute jump in the solution at the interface is much less in case of Aitken relaxation scheme where it is of order $\mathcal{O}\left(10^{-15}\right)$ for both cases of $\kappa_1 = \kappa_2$ and $\kappa_1 > \kappa_2$, and of order $\mathcal{O}\left(10^{-13}\right)$ for the case of $\kappa_1 < \kappa_2$.

A Developed codes for question 1

A.1 New script `mainPoisson.m`

```
% Solve a single Poisson's problem in 1D

clc
close all
clear variables

%% Effect of kappa value
kappaValues = [0.5 1 2];

for i = 1:length(kappaValues)

%Domain
Data.inix = 0;
Data.endx = 1;
Data.nelem = 100;
%Physical
Data.kappa = kappaValues(i);
Data.source = 1;
%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 = 0;

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

fignum = 1;
legendKappa = (['kappa = ',num2str(Data.kappa)]);
HP_PlotQ1(HeatProblem,fignum,legendKappa);
grid on
hold on
title('kappa effect (source = 1, elements = 100)')
legend('show')

end



%% Effect of source term
sourceValues = [0.5 1 2];

for i = 1:length(sourceValues)

%Domain
```

```
Data.inix = 0;
Data.endx = 1;
Data.nelem = 100;
%Physical
Data.kappa = 1;
Data.source = sourceValues(i);
%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 = 0;

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

fignum = 2;
legendSource = (['s = ',num2str(Data.source)]);
HP_PlotQ1(HeatProblem,fignum,legendSource);
grid on
hold on
title('source value effect (kappa = 1, elements = 100)')
legend('show')

end


%% Effect of number of elements
nOfElements = [5 11 21 41]; % odd numbers so that no nodes in the middle ...
    of the domain
error = zeros(length(nOfElements),1);
h = zeros(length(nOfElements),1);

for i = 1:length(nOfElements)

%Domain
Data.inix = 0;
Data.endx = 1;
Data.nelem = nOfElements(i);
%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 =1;
Data.RightValue = 0;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
```

```
Data.RightFluxes = 0;

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

fignum = 3;
legendElem = (['nelem = ',num2str(Data.nelem)]);
HP_PlotQ1(HeatProblem,fignum,legendElem);
grid on
hold on
title('no. of elements effect (kappa = 1, source = 1)')
legend('show')

% Data needed to produce the mesh convergence plot
Umax = max(HeatProblem.Solution.U);
error(i) = abs(0.125 - Umax);
h(i) = (Data.endx - Data.inix)/nOfElements(i);

end

% Plotting a mesh convergence plot
figure
plot(log10(h),log10(error),'-o');
grid on
xlabel('log10(h)')
ylabel('log10(error)')
title('Mesh convergence plot')
% Compute slopes
n = length(h);
for i=1:n-1
    Slopes(i) = log10(error(i+1)/error(i)) / log10(h(i+1)/h(i));
end
% Write the slopes over the plot
x = log10(h(end))+0.2*abs(log10(h(end))-log10(h(end-1)));
y = log10(error(end))+0.5*abs(log10(error(end))-log10(error(end-1)));
text(x,y,['Slope = ',num2str(Slopes(end),'%2.2f')],'FontSize',10)
```

B Developed codes for questions 2 and 3

B.1 New script `main_partitioned_and_monolithic.m`

```
% Solving two Poisson's problems independently or using monolithic scheme

clc
close all
clear variables

method = 'monolithic'; % 'monolithic' , 'partitioned'
sameKappa = 1;         % 1 --> 'yes'  ,  0 --> 'no'

%Domain 1
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 = 1;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 0;
Data.RightFluxes = 25;

%Domain 2
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
if sameKappa == 1
    Data2.kappa = Data.kappa;
    strKappa = 'same kappa';
else
    Data2.kappa = 2*Data.kappa;
    strKappa = 'different kappa';
end
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;
```

```
%Problem 1
HeatProblem = HP_Initialize(Data);
HeatProblem = HP_Build(HeatProblem);
%Problem 2
HeatProblem2 = HP_Initialize(Data2);
HeatProblem2 = HP_Build(HeatProblem2);

if strcmp(method,'partitioned')
HeatProblem = HP_Solve(HeatProblem);
HeatProblem2 = HP_Solve(HeatProblem2);
elseif strcmp(method,'monolithic')
[HeatProblem,HeatProblem2] = HP_SolveMonolithic(HeatProblem,HeatProblem2);
end

fignum = 1;
legendName = 'Problem 1';
HP_PlotQ1(HeatProblem,fignum,legendName);
legendName = 'Problem 2';
HP_PlotQ1(HeatProblem2,fignum,legendName);
title(['Solution of two problems with ',strKappa,' using ',method, ' ...
    scheme'])
grid on
legend('show','Location','northeast')
```

C Developed codes for question 4

C.1 New script `main_iterative.m`

```
% Solving two Poisson problems using iterative Dirchlet-Neumann scheme

clc
close all
clear variables

method = 'iterativeDN';      % 'iterativeDN'
sameKappa = 1;               % 1 --> 'yes'  ,  0 --> 'no'

%Domain 1
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 = 1;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

%Domain 2
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
if sameKappa == 1
    Data2.kappa = Data.kappa;
    strKappa = 'same kappa';
else
    Data2.kappa = 2*Data.kappa;
%     Data2.kappa = 1;
    strKappa = 'different kappa';
end
Data2.source = 1;
%Boundary conditions
%Dirichlet
Data2.FixLeft = 1; %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;

% Dirchlet-Neumann Iterations
iter = 1;
tol = 1e-6;
maxIter = 20;
while iter < maxIter

    fprintf('Iteration = %d\n',iter);

    % The previous value of the solution at the left side of domain 2
    Uprev = Data2.LeftValue;

    % Solve in domain 1
    HeatProblem = HP_Initialize(Data);
    HeatProblem = HP_Build(HeatProblem);
    HeatProblem = HP_Solve(HeatProblem);
    % Transfer Dirichlet BC from domain 1 to domain 2:
    Data2.LeftValue = HeatProblem.Solution.uRight;

    % Solve in domain 2
    HeatProblem2 = HP_Initialize(Data2);
    HeatProblem2 = HP_Build(HeatProblem2);
    HeatProblem2 = HP_Solve(HeatProblem2);
    % Transfer Neumann BC from domain 2 to domain 1:
    Data.RightFluxes = - HeatProblem2.Solution.FluxesLeft;

    % Check convergence
    solChange = abs(Data2.LeftValue - Uprev);
    disp(['Solution change = ',num2str(solChange)])

    if solChange ≤ tol
        fprintf('\nConvergence achieved in iteration number %g\n',iter);
        break
    end

    iter = iter + 1;
end

% Plot the converged solution
fignum = 2;
legendName = 'Problem 1';
HP_PlotQ1(HeatProblem,fignum,legendName);
legendName = 'Problem 2';
HP_PlotQ1(HeatProblem2,fignum,legendName);
title(['Solution of two problems with ',strKappa,' using ',method,' scheme'])
grid on
legend('show','Location','northeast')

% Difference in the solution at the interface from the left and the right
differenceU = HeatProblem2.Solution.uLeft - HeatProblem.Solution.uRight;
fprintf('difference in U at the interface = %e\n',abs(differenceU));
```

D Developed codes for question 5

D.1 New script `main_iterative_relaxation.m`

```
% Solving two Poisson problems using iterative Dirchlet-Neumann scheme

clc
close all
clear variables

method = 'iterativeDN';      % 'iterativeDN'
relaxation = 'aitken';       % 'no' , 'fixed' , 'aitken'
w_fixedRelaxation = 0.01;
sameKappa = 0;               % 1 --> 'yes'  ,  0 --> 'no'

%Domain 1
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 = 1;
%Neumann
Data.FixFluxesLeft = 0;
Data.LeftFluxes = 0;
Data.FixFluxesRight = 1;
Data.RightFluxes = 0;

%Domain 2
Data2.inix = 0.25;
Data2.endx = 1;
Data2.nelem = 75;
%Physical
if sameKappa == 1
    Data2.kappa = Data.kappa;
    strKappa = 'same kappa';
elseif sameKappa == 0
%     Data2.kappa = 2*Data.kappa;
    Data2.kappa = 1;
    strKappa = 'different kappa';
end
Data2.source = 1;
%Boundary conditions
%Dirichlet
Data2.FixLeft = 1; %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;

% Dirchlet-Neumann Iterations
iter = 1;
tol = 1e-6;
maxIter = 100;
while iter < maxIter

    fprintf('Iteration = %d\n',iter);

    if strcmp(relaxation,'aitken')
        if iter == 1
            U2prev = Data2.LeftValue;
            % Solve in domain 2
            HeatProblem2 = HP_Initialize(Data2);
            HeatProblem2 = HP_Build(HeatProblem2);
            HeatProblem2 = HP_Solve(HeatProblem2);
            % Transfer Neumann BC from domain 2 to domain 1:
            Data.RightFluxes = - HeatProblem2.Solution.FluxesLeft;
            % Solve in domain 1
            HeatProblem = HP_Initialize(Data);
            HeatProblem = HP_Build(HeatProblem);
            HeatProblem = HP_Solve(HeatProblem);
            % Transfer Dirichlet BC from domain 1 to domain 2:
            Data2.LeftValue = HeatProblem.Solution.uRight;
            % Solve in domain 2
            HeatProblem2 = HP_Initialize(Data2);
            HeatProblem2 = HP_Build(HeatProblem2);
            HeatProblem2 = HP_Solve(HeatProblem2);
            % Transfer Neumann BC from domain 2 to domain 1:
            Data.RightFluxes = - HeatProblem2.Solution.FluxesLeft;
        end
        % The previous value of the solution at the right side of domain 1
        U1prev = HeatProblem.Solution.uRight;
        % Before previous value of the solution at the left side of ...
           domain 2
        U2prevprev = U2prev;
        % The previous value of the solution at the left side of domain 2
        U2prev = HeatProblem2.Solution.uLeft;
    else
        % The previous value of the solution at the left side of domain 2
        U2prev = Data2.LeftValue;
    end

    % Solve in domain 1
    HeatProblem = HP_Initialize(Data);
    HeatProblem = HP_Build(HeatProblem);
    HeatProblem = HP_Solve(HeatProblem);
    % Transfer Dirichlet BC from domain 1 to domain 2:
    if strcmp(relaxation,'fixed')
       w = w_fixedRelaxation;
       Data2.LeftValue = w*HeatProblem.Solution.uRight + (1-w)*U2prev;
       strRelax = ([' with fixed relaxation',' (w = ',num2str(w),')']);
    elseif strcmp(relaxation,'aitken')
       w = (U2prevprev - U2prev)/(U2prevprev - U2prev + ...
          HeatProblem.Solution.uRight - U1prev);
```

```
        Data2.LeftValue = U2prev + w*(HeatProblem.Solution.uRight - U2prev);
        strRelax = ' with Aitken relaxation';
    elseif strcmp(relaxation,'no')
        Data2.LeftValue = HeatProblem.Solution.uRight;
        strRelax = '';
    end

    % Solve in domain 2
    HeatProblem2 = HP_Initialize(Data2);
    HeatProblem2 = HP_Build(HeatProblem2);
    HeatProblem2 = HP_Solve(HeatProblem2);
    % Transfer Neumann BC from domain 2 to domain 1:
    Data.RightFluxes = - HeatProblem2.Solution.FluxesLeft;

    % Check convergence
    solChange = abs(Data2.LeftValue - U2prev);
    disp(['Solution change = ',num2str(solChange)])

    if solChange ≤ tol
        fprintf('\nConvergence achieved in iteration number %g\n',iter);
        break
    end

    iter = iter + 1;
end

% Plot the converged solution
fignum = 2;
legendName = 'Problem 1';
HP_PlotQ1(HeatProblem,fignum,legendName);
legendName = 'Problem 2';
HP_PlotQ1(HeatProblem2,fignum,legendName);
title({['Solution of two problems with ',strKappa,' using ',method,' ...
   scheme'],strRelax})
grid on
legend('show','Location','northeast')

% Difference in the solution at the interface from the left and the right
differenceU = HeatProblem2.Solution.uLeft - HeatProblem.Solution.uRight;
fprintf('difference in U at the interface = %e\n',abs(differenceU));
```