Résolution des Systèmes Linéaires issus des EDP

ENSEEIHT/INSA — ModIA

TP: Alternating and Parallel Schwarz methods

Année scolaire 2024–2025

Alena Kopaničáková Carola Kruse Ronan Guivarch

1 Introduction

In this TP, you will focus on the preconditioning techniques and implement alternating and parallel Schwarz methods to solve a model problem. The goal is to deepen the understanding of Schwarz methods, and analyze their dependence on algorithmic parameters (e.g., overlap).

Model problem

Consider a heat diffusion problem posed on a square domain Ω. The boundary $\partial\Omega$ is decomposed into the left part Γ_L, the right part Γ_R, as well as $\partial\Omega \setminus \Gamma_L \setminus \Gamma_R$ the top and bottom part. We set the temperature to be equal to 0 on all parts of the boundary, i.e., $gl = gr = 0$. Inside the room, there is a radiator whose temperature is equal to 50. The radiator is modeled by a source term $f(x, y) = 50$ if $(x, y) \in [0.0, 1.0] \times [0.4, 0.6]$ and zero otherwise.

We want to find the temperature distribution inside the room described by the following equation:

$$\begin{aligned} -\Delta u &= f, \quad \text{in } \Omega, \\ u &= gl, \quad \text{on } \Gamma_L, \\ u &= gr, \quad \text{on } \Gamma_R, \\ u &= 0, \quad \text{on } \partial\Omega \setminus \Gamma_L \setminus \Gamma_R. \end{aligned} \tag{1}$$

The script `Problem.m` provides a discretization of the model problem.

2 Preconditioning

Get familiar with scripts `ex1a.m, ex1b.m, ex1c.m`, which solves the model problem from the `Problem.m` class. Solve the model problem for a fixed discretization and plot the solution.
Tasks:

- In the script `ex1a.m`, we study the behavior of the model problem while increasing the resolution of our discretization. To this aim, discretize the problem with increasing N and compute the condition number of the assembled stifness matrix. Study how the condition number changes with the increasing problem resolution.

```
for i = 1: size(Ns, 2)
   N = Ns(i);
   problem = Problem(N, N, global_domain_start, global_domain_end);

   % Extract mesh size
   hs(i) = problem.h;

   % Compute condition number of the stifness matrix
   condition_numbers(i) = % ....

end
```

- In the script `ex1b.m`, we solve the model problem using `pcg` function of Matlab. Documentation of `pcg` function can be found online. To do so, please complete function `solve_cg` in `Problem.m` script. What can you observe regarding the convergence as the problem size increases?

```
      function [u, iter] = solve_cg(obj)
         % Solve the boundary value problem represented by A and force
         % term f using matlab's cg solver
         obj.f(1:obj.Ny, 1) = obj.f(1:obj.Ny, 1) + obj.gl / obj.h^2;
         obj.f(1:obj.Ny, end) = obj.f(1:obj.Ny, end) + obj.gr / obj.h^2;

         % Define tolerance
         tol=1e-10;
         maxit=size(obj.A,1);

         % Call PCG solver with no preconditioner
         [u,~,~,iter] = % ....

         u = reshape(u, obj.Ny, obj.Nx);
         u = [obj.gl u obj.gr];
      end
```

- In the script `ex1c.m`, we solve the model problem using `pcg` function with incomplete Cholesky preconditioner (`ichol` function). To do so, please complete function `solve_pcg` in `Problem.m` script. Compare the convergence of preconditioned and non-preconditioned variants. What do you observe?

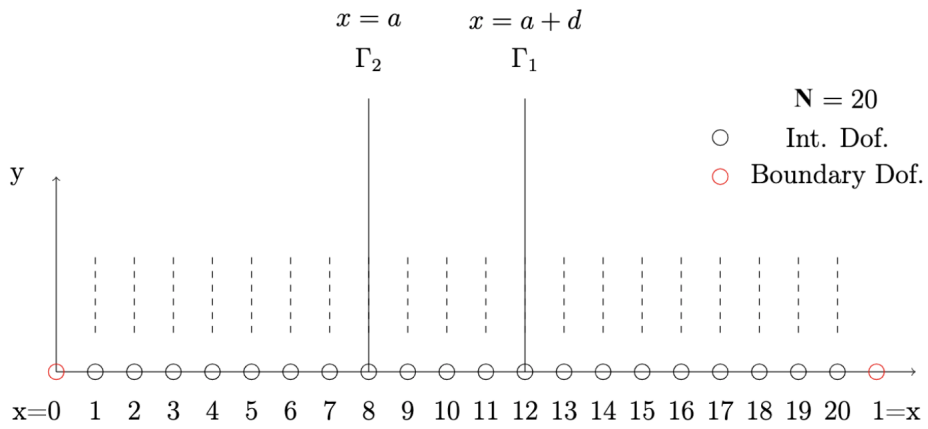

Figure 1: Sketch of the decomposition of Ω into Ω_1 and Ω_2.

```
function [u, iter] = solve_pcg(obj)
   % Solve the boundary value problem represented by A and force
   % term f using matlab's preconditioned cg solver
   obj.f(1:obj.Ny, 1) = obj.f(1:obj.Ny, 1) + obj.gl / obj.h^2;
   obj.f(1:obj.Ny, end) = obj.f(1:obj.Ny, end) + obj.gr / obj.h^2;

   % Define tolerance
   tol=1e-10;
   maxit=size(obj.A,1);

   % Get Cholesky preconditioner
   M = % ....

   % Call pcg solver with preconditioner M
   [u,~,~,iter] = % .....

   u = reshape(u, obj.Ny, obj.Nx);
   u = [obj.gl u obj.gr];
end
```

3 Alternating Schwarz method

We will use script `ex2.m` to test the alternating Schwarz method.

- Study script `ex2.m` and understand the decomposition of domain Ω into Ω_1 and Ω_2. Note that the domain is split only in the x-direction, as depicted in Figure 1.

- Implement alternating Schwarz method as part of *alternating_Schwarz* function. The three main steps are:
  - Solve the first subproblem with the correctly prescribed interface/boundary condition.

- Solve the second subproblem with the correctly prescribed interface/boundary condition.
- Construct global iterations by merging together solutions obtained from subdomain solves. Pay attention to overlapping region, where the subdomain solutions have to be averaged.

To solve each subproblem, you can utilize one of solve functions from `Problem.m` class. Reminder: The interface condition for each subdomain is prescribed using the solution obtained by solving another subdomain.

```
function [error, residual, iter, u_global] = alternating_Schwarz(params, ...
    problem_global, u_global_exact, problem_subdomain1, ...
    problem_subdomain2, u1, u2, overlap_size)

iter=0;
converged=false;
error=zeros(params.maxiter,1);
residual=zeros(params.maxiter,1);


% Additive Schwarz solver
while converged==false

   iter = iter+1;

   %% Solve on the first subdomain
   % Assign correct BC conditions
   problem_subdomain1.gr = % ...
   % Solve
   u1 = % ...

   %% Solve on the second subdomain
   % Assign correct BC conditions
   problem_subdomain2.gl = % ...
   % Solve
   u2 = % ...


   % Construct global iterate by merging together subdomain contributions
   % Pay attention to the overlapping part!!!
   u_global= % ...


   % compute error and residual between iterate and exact solution
   error(iter+1)=problem_global.compute_error(u_global_exact, u_global);
   residual(iter)=problem_global.compute_residual(u_global);

   fprintf('it: %d || r || %d || e || %d \n', iter, residual(iter), ...
       error(iter));

   if(iter > params.maxiter || residual(iter) < params.atol)
      converged=true;
   end


end
end
```

- Run the script with different sizes of overlap and interface positions. Enable plotting, i.e., set `params.plt=1`, and study how the iterates evolve. What is the influence of the overlap on the convergence, and why?

- Check that, asymptotically, the observed error decrease is as predicted by the following theoretical convergence factor:

$$\rho := \max_{k\in[\pi,N\pi]} \frac{\sinh(k\beta)}{\sinh(k(1-\beta))}\frac{\sinh(k(1-\alpha))}{\sinh(k\alpha)},$$

  where α and β are the location of Γ_1 and Γ_2, respectively.

  To do so, implement the required formula in `ex2.m` script. What do you observe?

```
%=== Verify the convergence factor
figure(2);
% Plot error
semilogy(error_alternating)
hold on
% Location on the boundary of \Omega_2
beta=a*problem_global.h;
% Location on the boundary of \Omega_1
alpha=(a+d)*problem_global.h;
k=(1:N)*pi;
% Convergence factor predicted by the theory
rho= % ...
iters=1:1:iters_alternating+1;
% Plot theoretical convergence rate
semilogy(iters,rho.^iters,'r')
grid on
xlabel('Iterations');
ylabel('Error');
legend('Error','Theoretical decay')
title('Alternating Schwarz method')
```

4 Alternating vs. Parallel Schwarz method

We will use script `ex3.m` to compare the alternating and parallel Schwarz method. We consider same decomposition of Ω as for alternating Schwarz method implemented in previous step.
Tasks:

- Implement parallel Schwarz method as part of *parallel_Schwarz* function. The three main steps are:

  – Solve the first subproblem with the correctly prescribed interface/boundary condition.

- Solve the second subproblem with the correctly prescribed interface/boundary condition.
- Construct global iterations by merging together solutions obtained from subdomain solves. Pay attention to overlapping region, where the subdomain solutions have to be averaged.

To solve each subproblem, you can utilize one of the solve functions from `Problem.m` class. Reminder: The interface condition for each subdomain is prescribed using the solution obtained by solving another subdomain.

```
function [error, residual, iter, u_global] = parallel_Schwarz(params, ↵
    ↳ problem_global, u_global_exact, problem_subdomain1, ↵
    ↳ problem_subdomain2, u1, u2, overlap_size)

iter=0;
converged=false;

error=zeros(params.maxiter,1);
residual=zeros(params.maxiter,1);


% Parallel Schwarz solver
while converged==false

   iter = iter+1;

   %% Solve on the first subdomain
   % Assign correct BC conditions
   problem_subdomain1.gr = % ...
   % Solve the first subproblem
   u1n = % ...

   %% Solve on the second subdomain
   % Assign correct BC conditions
   problem_subdomain2.gl = % ...
   % Solve the second subproblem
   u2n = % ...

   u1=u1n;
   u2=u2n;

   % Construct global iterate by merging together subdomain contributions
   % Pay attention to the overlapping part!!!
   u_global= % ...

   % compute error and residual between iterate and exact solution
   error(iter+1)=problem_global.compute_error(u_global_exact, u_global);
   residual(iter)=problem_global.compute_residual(u_global);

   fprintf('it: %d || r || %d || e || %d \n', iter, residual(iter), ↵
       ↳ error(iter));

   if(iter > params.maxiter || residual(iter) < params.atol)
      converged=true;
   end


end
end
```

- Run the script with different sizes of overlap and interface positions. Enable plotting, i.e., set `params.plt=1`, and study how the iterates evolve. What is the influence of the overlap on the convergence, and why?

- Compare the convergence speed and performance of the alternating and parallel Schwarz methods. What do you observe and why?