Résolution des Systèmes Linéaires issus des EDP

ENSEEIHT/INSA — ModIA

TP: One-level and two-level Schwarz methods

Année scolaire 2024–2025

Alena Kopaničáková   Carola Kruse   Ronan Guivarch

1 Introduction

In this TP, you will focus on implementation of one-level and two-level additive Schwarz methods to solve a model problem. The goal is to deepen the understanding of the methods seen in the class, and analyze their convergence behavior.

Model problem

Consider a diffusion problem posed on a domain $\Omega = [0, 1]$ with the boundary $\partial\Omega$, given as

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

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

2 Overlapping Additive Schwarz

Get familiar with scripts `ex1a.m, ex1b.m, ex1c.m`, which solve the model problem from the `Problem.m` class using Richardson iteration with Additive Schwarz (AS) preconditioner.
Tasks:

- In the script `ex1a.m`, we study the behavior of the AS preconditioner with an increasing number of subdomains. To this aim, we first have to complete the routine `assembly_interpolation_subdomains`, which assembles the interpolation matrices related to each subdomain. Moreover, we need to complete `apply_AS` function, which implements a single step of the one-level AS method.

```
function [obj] = assembly_interpolation_subdomains(obj)
   % Assembly interpolation matrices for local spaces
   obj.I = cell(obj.n_subd,1);
   for i = 1:obj.n_subd
      obj.I{i} = sparse(obj.n_dofs, obj.n_loc);
      left = (i-1)*(obj.n_loc-obj.overlap) + 1;
      right = left + obj.n_loc-1;
      % obj.I{i}(left:right,1:obj.n_loc) = ...
   end
end
```

```
function corr = apply_AS(obj, rhs)
   % One step of the one-level additive Schwarz method.
   % INPUT PARAMETERS:
   % * rhs: right-hand side/residual
   % OUTPUT:
   % * corr: preconditioned residual / correction


   % Restrict residual to V_0,V_1,...,V_S
   rloc = zeros(obj.n_loc, obj.n_subd);
   for i = 1:obj.n_subd
      % rloc(:,i) = ...
   end

   % Solve subproblems in V_0,V_1,...,V_S
   corrloc = zeros(obj.n_loc, obj.n_subd);
   for i = 1:obj.n_subd
      % A_loc = ..
      % corrloc(:,i) = ..
   end

   % Add corrections
   corr = zeros(obj.n_dofs,1);
   for i = 1:obj.n_subd
      % corr =
   end

end
```

- In the script `ex1b.m`, we study the performance of the Richardson method with AS preconditioner with respect to different overlaps, i.e.,
  - Set step-size to be equal to 1 and study the convergence behavior of the AS preconditioner with respect to different overlaps.
  - Use adaptive step-size for Richardson iteration. This can be specified by passing "-1" as the second argument to `solve_richardson("AS", -1)` function. Note that we

compute the ideal step size on each iteration using a formula derived during the class. Study the convergence behavior of the AS preconditioner with respect to different overlaps.

  - Discuss the different behavior of the AS method for both test cases.

- In the script `ex1c.m`, we implement the restricted Additive Schwarz method (RAS). To this aim, we have to complete the routine `assembly_subdomain_scaling_matrices`, which assembles the scaling matrices for ensuring the partition of unity in the overlap. Moreover, we need to complete `apply_RAS` function, which implements one step of the one-level RAS method. Please refer to the lecture slides for details and the simple example.

```
function [obj] = assembly_subdomain_scaling_matrices(obj)
   % Assembly scaling matrices for local spaces
   obj.D = cell(obj.n_subd,1);
   for i = 1:obj.n_subd
      d= ones(obj.n_loc, 1);

      % TODO:: modify d accordingly, so it scales corrections in
      % overlapping region following the partition of unity
      % approach
      % .......

      obj.D{i} = diag(d);
   end
end
```

```
      function corr = apply_RAS(obj, rhs)
         % One step of the one-level restricted additive Schwarz method.
         % INPUT PARAMETERS:
         % * rhs: right-hand side/residual
         % OUTPUT:
         % * corr: preconditioned residual / correction


         % Restrict residual to V_0,V_1,...,V_S
         rloc = zeros(obj.n_loc, obj.n_subd);
         for i = 1:obj.n_subd
            % rloc(:,i) = ...
         end

         % Solve subproblems in V_0,V_1,...,V_S
         corrloc = zeros(obj.n_loc, obj.n_subd);
         for i = 1:obj.n_subd
            % A_loc = ...
            % corrloc(:,i) = ...
         end

         % Add corrections
         corr = zeros(obj.n_dofs,1);
         for i = 1:obj.n_subd
            % corr = ...
         end

      end
```

- Study the performance of AS and RAS preconditioners. Provide the reasoning behind their different convergence behavior.

3 Two-level Additive Schwarz

Get familiar with scripts `ex2a.m, ex2b.m, ex2c.m`, which solves the model problem from the `Problem.m` class using Richardson iteration with a two-level Additive Schwarz (TL-AS) preconditioner.
Tasks:

- In the script `ex2a.m`, we study the behavior of the TL-AS preconditioner, where the coarse space is included in an additive manner. To this aim, we have to first complete the routine `assembly_interpolation_coarse`, which assembles the interpolation matrices used to construct coarse space quantities. For the purpose of this exercise, we use the interpolation operators assembled using the Nicolaides approach; see lecture notes for the details. Moreover, we need to complete `apply_two_level_AS_additive` function, which implements one step of the TL-AS method with additively added coarse space.

```
    function [obj] = assembly_interpolation_coarse(obj)
       % Assembly interpolation matrix for coarse space
       % Use Nicolaides approach
       obj.I_coarse = zeros(obj.n_dofs, obj.n_subd);

       for i = 1:obj.n_subd
          % Add appropriate entries of interpolation operator
       end
    end
```

```
    function corr = apply_two_level_AS_additive(obj, rhs)
       % One step of the two-level additive Schwarz method with ↩
           ↪ additively added coarse-space.
       % INPUT PARAMETERS:
       % * rhs: right-hand side/residual
       % OUTPUT:
       % * corr: preconditioned residual / correction

       corr = obj.apply_AS(rhs);
 % r_coarse = ...
 % A_coarse = ...
 % corr_coarse = ..

        % corr = corr + ...
    end
```

- In the script `ex2b.m`, we study the behavior of TL-AS preconditioner, where the coarse space is included in a multiplicative manner before or after solving on subdomains.

  To this aim, we need to complete `apply_two_level_AS_multiplicative_coarse_first` (coarse step before subdomain step) and `apply_two_level_AS_multiplicative_coarse_second` (coarse step after subdomain step) functions, which implement one step of TL-AS method with multiplicatively added coarse space step.

```
    function corr = 
        apply_two_level_AS_multiplicative_coarse_first(obj, rhs)
        % One step of the two-level additive Schwarz method with 
            additively added coarse-space before the subdomain solve.
        % INPUT PARAMETERS:
        % * rhs: right-hand side/residual
        % OUTPUT:
        % * corr: preconditioned residual / correction


        corr = obj.apply_AS(rhs);

        % r = ...
        % r_coarse = ...
        % A_coarse = ...
        % corr_coarse = ...
        %
        % corr = ...
    end
```

```
    function corr = 
        apply_two_level_AS_multiplicative_coarse_second(obj, rhs)
        % One step of the two-level additive Schwarz method with 
            additively added coarse-space after the subdomain solve.
        % INPUT PARAMETERS:
        % * rhs: right-hand side/residual
        % OUTPUT:
        % * corr: preconditioned residual / correction

        corr = 0.0*rhs;

        % r_coarse = ...
        % A_coarse = ...
        % corr_coarse = ...

        % corr = ...
        % r = ...
        % corr = corr + obj.apply_AS(r);

    end
```

  – Compare the performance of the TL-AS method with additively and multiplicatively added coarse spaces.
  – Discuss the reasons for their different convergence behavior.

- In script ex2c.m, we compare the performance of AS method and TL-AS method with respect to different numbers of subdomains. Run the script to study the obtained results and discuss the observed behavior.

4 Two-level Additive Schwarz as a special case of two-level multigrid

The goal of this excercise is to implement a two-level multigrid (TG) with additive Schwarz smoother and different type of coarse spaces.
Tasks:

- Implement a two-level multigrid with AS smoother (without overlap). To this aim, you need to complete routine `apply_TG` in `Problem.m`. As a first step, you can use the coarse space generated using the Nicolaides approach, for which we already have an interpolation operator available. The developed method can be tested using script script `ex3a.m`.

```
function corr = apply_TG(obj, rhs)
   % One step of two-level multigrid method with AS smoother
   % INPUT PARAMETERS:
   % * rhs: right-hand side/residual
   % OUTPUT:
   % * corr: preconditioned residual / correction

   corr = 0*rhs;

   % % Call smoother
   % corr = ...
   %
   % % Coarse-step
   % r = ...
   % r_coarse = ...
   % A_coarse = ...
   % corr_coarse = ...
   % corr = ...
   %
   % % Call smoother
   % r = ...
   % corr = ...
end
```

- As a next step, we again consider the two-level multigrid with AS smoother (without overlap). However, this time, we will use geometrically assembled coarse space. For simplicity, you can use coarsening by a factor of 2 compared to the original (fine-level) grid.

  You need to complete routine `apply_TG_geo` in `Problem.m`, implementing two-level multigrid with AS smoother. Your implementation can closely resemble the code in `apply_TG` function but use transfer operators $obj.I\_coarse\_geo$, which you have to assemble in routine `assembly_interpolation_coarse_geo`. The method can be tested using `ex3b.m` script.

```
    function [obj] = assembly_interpolation_coarse_geo(obj)
       % Assembly interpolation matrix for coarse space
       % Use geometric approach, where you coarsen by factor of two
       obj.I_coarse_geo = zeros(obj.n_dofs, floor(obj.n_dofs/2-1));

       % TODO:: implement correctly entries of obj.I_coarse_geo

    end
```

```
    function corr = apply_TG_geo(obj, rhs)
       % One step of two-level multigrid method with AS smoother
       % INPUT PARAMETERS:
       % * rhs: right-hand side/residual
       % OUTPUT:
       % * corr: preconditioned residual / correction

       corr = 0*rhs;

       % Call smoother
       % corr = ...

       % Call coarse-step
       %r = ...
       %r_coarse = ...
       %A_coarse = ...
       %corr_coarse = ...
       %corr = ...

       % Call smoother
       %r = ...
       %corr = ...
    end
```

- Finally, compare the convergence, performance, and computational cost (per iteration) of AS and TG methods with geometric (coarsening the original grid) and algebraic coarse-space (Nicolaides approach). Is the cost of algebraic and geometric coarse spaces comparable? Can the cost of geometric coarse space be reduced? Summarize your numerical observations and discuss your conclusions in the report.