Technische Universität Berlin Winter term 2025/26
Institute of Mathematics
Prof. Dr. T. Breiten, Prof. Dr. S. May, Dr. Juliette Dubois

Numerical Mathematics II for Engineers
Tutorial 10

Topics : Multigrid algorithm

Discussion in the tutorials of the week 12 – 16 January 2026

Disclaimers:

- To test your code, we provide tests. This should help you to locate precisely your errors, and code more efficiently. To run the test, you need to install pytest.
- Exercises should be solved in **fixed groups of 3 students**.
- Hand in the solution in **one folder** labeled **hw[hw_number]_group[group_number]** and containing:
  - **One pdf** for the theoretical questions and comments on the numerical results,
  - **One python file per programming exercise**.
  - Write the group number and all names of your members **in each file**.

Exercise 10.1: 2D Multigrid

In the lecture, you learned about the multigrid algorithm to solve systems of linear equations originating from elliptic PDEs. The basic algorithm, based on using a V-cycle, is shown in Algorithm 1. In this task, you will implement the algorithm in the context of *finite differences* yourself. The specific choices for all the relevant building blocks will be described in detail below.

Problem

Solve the Poisson equation

$$-\Delta u = f(\vec{x}) \qquad \text{in } \Omega, \tag{1a}$$
$$u = 0 \qquad \text{on } \Gamma, \tag{1b}$$

with the *finite difference* method using a *5-point* stencil. The domain Ω is given as a rectangle $[0,a] \times [0,b]$ and the mesh $\mathcal{T}^{(L)}$ is obtained by refining L times a Cartesian (coarse) grid $\mathcal{T}^{(0)}$. The mesh-refinement levels naturally give the multigrid levels, as shown in Figure 1.

Building blocks

In the following, we discuss intergrid transfer operators, level operators, smoothers, and the coarse-grid solver. We refer to these as operators, since we will never explicitly compute the corresponding matrices.

Intergrid transfer operators
The *restriction operator* $R_{(l)}^{(l-1)}$ is given by injection:

$$r_{i,j}^{(l)} = r_{2i,2j}^{(l+1)} \tag{2}$$

and the *prolongation operator* $P_{(l-1)}^{(l)}$ via interpolation from the coarse points to the fine points. Depending on the position of the fine points (coinciding with coarse points, with coarse horizontal or vertical lines, or with the center of coarse cells), one gets different interpolation formulas:

$$x_{2i,2j}^{(l+1)} = x_{i,j}^{(l)}, \tag{3a}$$
$$x_{2i+1,2j}^{(l+1)} = (x_{i,j}^{(l)} + x_{i+1,j}^{(l)})/2, \tag{3b}$$
$$x_{2i,2j+1}^{(l+1)} = (x_{i,j}^{(l)} + x_{i,j+1}^{(l)})/2, \tag{3c}$$
$$x_{2i+1,2j+1}^{(l+1)} = (x_{i,j}^{(l)} + x_{i+1,j}^{(l)} + x_{i,j+1}^{(l)} + x_{i+1,j+1}^{(l)})/4. \tag{3d}$$

Figure 2 gives a visualization of the performed operations and clarifies how the indexing is meant.

Level operators
The discrete version of $-\Delta u$ on all levels is given by the update formula

$$x_{i,j}^{(k+1)} = -\frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)}}{h_x^2} + \left(\frac{2}{h_x^2} + \frac{2}{h_y^2}\right) x_{i,j}^{(k)} - \frac{x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)}}{h_y^2}, \tag{4}$$

where h_x and h_y are different on each level (see Figure 1).

Smoothers

Algorithm 1: Multigrid V-cycle called recursively on each level l to solve $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{x}$ is also used as initial condition. The algorithm is called with $\boldsymbol{x} \leftarrow \texttt{VCycle}(L, \boldsymbol{x}, \boldsymbol{f})$ if used as solver. The pre- and post-smoothers are each run m times. As coarse-grid solver, we apply a smoother n times.

```
if $l = 0$ then
    for $i \leftarrow 1$ to $n$ do
        $\boldsymbol{x} \leftarrow S^{(0)}(\boldsymbol{x}, \boldsymbol{b})$ ;          /* coarse-grid solver:  smooth $n$ times */
    return $\boldsymbol{x}$
else
    for $i \leftarrow 1$ to $m$ do
        $\boldsymbol{x} \leftarrow S^{(l)}(\boldsymbol{x}, \boldsymbol{b})$ ;          /* presmoothing m-times */
    $\boldsymbol{r} \leftarrow \boldsymbol{b} - \boldsymbol{A}^{(l)}\boldsymbol{x}$ ;          /* compute residual */
    $\tilde{\boldsymbol{r}} \leftarrow \boldsymbol{R}_{(l)}^{(l-1)}\boldsymbol{r}$ ;          /* restrict residual */
    $\tilde{\boldsymbol{x}} \leftarrow \texttt{VCycle}(l-1, \boldsymbol{0}, \tilde{r})$ ;          /* recursion */
    $\boldsymbol{x} \leftarrow \boldsymbol{x} + \boldsymbol{P}_{(l-1)}^{(l)}(\tilde{\boldsymbol{x}})$ ;          /* prolongation */
    for $i \leftarrow 1$ to $m$ do
        $\boldsymbol{x} \leftarrow S^{(l)}(\boldsymbol{x}, \boldsymbol{b})$ ;          /* postsmoothing $m$ times */
    return $\boldsymbol{x}$
```

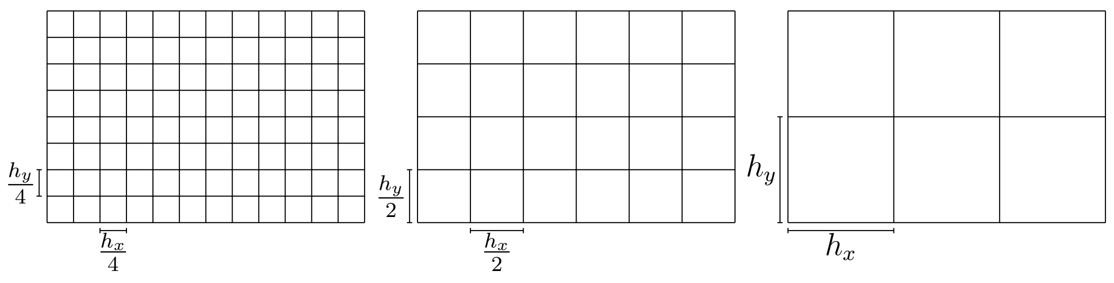

Figure 1: Sequence of meshes $\mathcal{T}^{(l)}$ obtained by refining a coarse grid that contains 3×2 cells.

For smoothing, we use the relaxation scheme:

$$\vec{x}^{(k+1)} = \vec{x}^{(k)} + \omega C^{-1}(b - A\vec{x}^{(k)}) = (1-\omega)\vec{x}^{(k)} + \omega(\underbrace{\vec{x}^{(k)} + C^{-1}(b - A\vec{x}^{(k)})}_{\vec{y}^{(k+1)}}) \tag{5}$$

with the relaxation parameter ω and an invertible matrix C. We use $\omega = 0.7$. We now want to accelerate this by a good choice for C. As simple base smoother, we consider Jacobi: take as C the matrix D made of the diagonal components of A. This choice is motivated by the matrix decomposition $A = L + D + U$. This results in

$$\vec{y}^{(k+1)} = \vec{x}^{(k)} + D^{-1}(b - (L+D+U)\vec{x}^{(k)}) = D^{-1}(b - (L+U)\vec{x}^{(k)}).$$

This can be rewritten in a matrix-free way as

$$y_{i,j}^{(k+1)} = \frac{1}{\frac{2}{h_x^2} + \frac{2}{h_y^2}} \left(b_{i,j} + \frac{x_{i-1,j}^{(k)} + x_{i+1,j}^{(k)}}{h_x^2} + \frac{x_{i,j-1}^{(k)} + x_{i,j+1}^{(k)}}{h_y^2} \right). \tag{6}$$

As **coarse-grid solver**, we will use the implemented smoothers. We simply run them n times.

Note: In this problem, we will not implement MG as a stand-alone solver, but instead use it as a preconditioner for a CG-algorithm. You don't need to understand CG though to do the homework.

Tasks

Complete the following tasks:

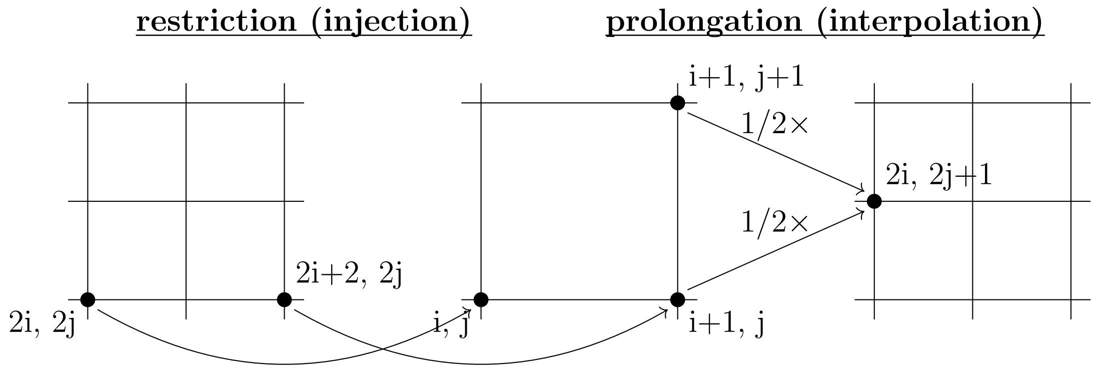

Figure 2: Visualization of restriction and prolongation.

(a) First, implement the smoother: relaxation (5) and Jacobi (6). Implement everything in a matrix-free style and, in particular, use vectorized operations.

(b) Solve (1) for `setup == "unit source"` with the conjugate gradient method using the implemented smoothers as preconditioner and $f(\vec{x}) = 1$. (Everything is already implemented. You just need to choose the preconditioner "Jacobi" and run the code you implemented in (a).) Vary the number of levels, which corresponds to solving on a finer mesh. What do you observe?

(c) Repeat task (b) and set $f(\vec{x}) = 0$. The solution is obviously $u(\vec{x}) = 0$. Vary the initial guess. What do you see for low-frequency initial guess (`setup == "low-frequency error"`) and high-frequency ones (`setup == "high-frequency error"`)?

(d) Implement the missing building blocks of the multigrid algorithm: prolongator (3), restrictor (2), and V-cycle (Algorithm 1). Like before, do not assemble any matrices and use vectorized operations where possible.

(e) Repeat task (b) for `setup == "unit source"`, this time with the multigrid algorithm (`preconditioner = "multigrid"`). What do you see now? Any differences to the behavior in (b)?