

Scientific Computing II

Multigrid Methods

Programming Exercise 2: Multigrid for Poisson Equation

We want to solve the two-dimensional finite difference approximation

$$a \frac{u_{i+1j} - 2u_{ij} + u_{i-1j}}{h_x^2} + b \frac{u_{ij+1} - 2u_{ij} + u_{ij-1}}{h_y^2} = 0, \quad i \in \{2, \dots, N_x\}, j \in \{2, \dots, N_y\}$$

$$u_{ij} = 0, \quad i = 1 \vee j = 1 \vee i = N_x + 1 \vee j = N_y + 1 \quad (1)$$

which is defined on a unit square using a Cartesian mesh $((i-1)h_x, (j-1)h_y)$ with $h_x = 1/N_x, h_y = 1/N_y$ ¹. The number of grid points in each direction should initially be chosen as multiple of two, that is $N_x = N_y = 2^l, l \in \mathbb{N}$. The coefficients $a, b \in \mathbb{R}$ can be chosen arbitrarily. For the time being, we set them to unity, $a = b = 1$.

The following two matlab functions are provided on the webpage of this course:

- `gaussSeidel(a,b,u,rhs)` carries out one iteration of the Gauss-Seidel method. The input arguments are given by the scalar coefficients a, b from above, the two-dimensional solution $u \in \mathbb{R}^{N_x+1 \times N_y+1}$ and a right hand side $rhs \in \mathbb{R}^{N_x+1 \times N_y+1}$. The mesh size of the current solution u is determined from the number of grid points $N_x + 1, N_y + 1$. The function returns the updated solution of u .
- `residual(a,b,u,rhs)` evaluates the residual for the approximate solution u of Eq. (1). The function returns the residual $r \in \mathbb{R}^{N_x+1 \times N_y+1}$.

In the following, we will prepare the different subroutines required for a multigrid solver and finally put them together in the *V-cycle* algorithm.

- (a) Implement a matlab function `restrict_fullweighting(r)` to perform the full-weighting restriction. The argument of the function is the residual $r \in \mathbb{R}^{N_x+1 \times N_y+1}$. The function should return a restricted residual $r^c \in \mathbb{R}^{\frac{N_x}{2}+1 \times \frac{N_y}{2}+1}$ which arises from

$$r_{ij}^c := \frac{1}{4} r_{2(i-1)+1, 2(j-1)+1}$$

$$\frac{1}{8} (r_{2(i-1)+1, 2(j-1)} + r_{2(i-1), 2(j-1)+1} + r_{2(i-1)+2, 2(j-1)+1} + r_{2(i-1)+1, 2(j-1)+2}) \quad (2)$$

$$\frac{1}{16} (r_{2(i-1), 2(j-1)} + r_{2(i-1)+2, 2(j-1)} + r_{2(i-1), 2(j-1)+2} + r_{2(i-1)+2, 2(j-1)+2})$$

¹The enumeration has been slightly adapted to stay more compatible with the MATLAB enumeration of arrays and matrix structures

for all inner points. The value r_{ij}^c can be considered to be zero for all points on the boundary strip.

(b) Implement a matlab function `restrict_injection(r)` to perform the full-weighting restriction. The function should return a restricted residual $r^c \in \mathbb{R}^{\frac{N_x}{2}+1 \times \frac{N_y}{2}+1}$. For the injection method, see the lecture slides. The value r_{ij}^c can be considered to be zero for all points on the boundary strip.

(c) Implement a matlab function `interpolate(e)` which prolongates the error $e \in \mathbb{R}^{\frac{N_x}{2}+1 \times \frac{N_y}{2}+1}$ to the fine grid. The interpolated error $e^f \in \mathbb{R}^{N_x+1 \times N_y+1}$ is zero on the boundary strip and defined on the inner points via bilinear interpolation as

$$e_{ij}^f := \begin{cases} \frac{1}{4}(e_{i/2,j/2} + e_{i/2+1,j/2} + e_{i/2,j/2+1} + e_{i/2+1,j/2+1}) & \text{if (i,j) is centered between} \\ & \text{4 coarse grid points} \\ \frac{1}{2}(e_{(i+1)/2,j/2} + e_{(i+1)/2,j/2+1}) & \text{if (i,j) is centered between} \\ & \text{2 y-aligned coarse grid points} \\ \frac{1}{2}(e_{i/2,(j+1)/2} + e_{i/2+1,(j+1)/2}) & \text{if (i,j) is centered between} \\ & \text{2 x-aligned coarse grid points} \\ e_{(i+1)/2,(j+1)/2} & \text{if (i,j) coincides with a coarse} \\ & \text{grid point.} \end{cases} \quad (3)$$

(d) Put the algorithmic components together in a function

`vCycle(a,b,u,rhs,preSmoothing,postSmoothing,level,mgID).`

Besides the coefficients a, b , the approximate solution u and the right hand side rhs , the parameters `preSmoothing` and `postSmoothing` determine the number of pre- and post-smoothing Gauss-Seidel iterations. The parameter `level` corresponds to $N_x = N_y = 2^{level}$. If `level = 1`, we only have one inner grid point (and 3×3 points in total). In this case, the function `vCycle(...)` should carry out one Gauss-Seidel iteration on the current data u and return the new solution. For `level > 1`, the function should carry out the *recursive* v-cycle algorithm, see slide 8 of the multigrid lecture slides. The additional parameter `mgID` can be used to define which restriction-interpolation combination should be used. You may introduce an additional alternative which abstains from the v-cycle and only carries out one Gauss-Seidel iteration (e.g. for testing purposes).

(e) Use your implementations to solve the 2D Poisson problem. Initialise the solution u according to the function $u(x, y) = \sin(\pi x) \sin(\pi y)$ on the unit square. Use two pre- and two post-smoothing steps and iterate until the maximum norm of the residual has reached an accuracy of 10^{-5} . Run the simulation for different grid levels $l \in \{2, 3, 4, 5, 6\}$ and both full-weighting restriction or injection. How many iteration steps do you need for each grid resolution/ solver scheme? How many iteration steps do you need if you use pure Gauss-Seidel iterations to solve the Poisson problem? Create error plots (iteration steps vs. maximum norm of residual) for both multigrid schemes and the Gauss-Seidel scheme for grid level $l = 5$, assuming a tolerance $TOL = 1e - 12$.

Programming Exercise 3: Multigrid for Anisotropic Poisson Equation

So far, we considered the coefficients $a = b = 1$. In the following, we want to consider the case where $a \gg b$ and study the respective behaviour of the multigrid scheme. For this purpose, we set $a = 1$, $b = 1e - 4$ and obtain an anisotropic Poisson-like equation.

- (a) Use the matlab code from the programming exercise 2 (including the same settings such as tolerance ($TOL = 1e - 5$, pre-/post-smoothing=2 steps etc.) to solve the anisotropic Poisson problem. How many iteration steps do you need now for the different v-cycle schemes and the pure Gauss-Seidel method?
- (b) Implement new functions `restrict_semi` and `interpolate_semi` which allow for a semi-coarsening and semi-prolongation of the residual along x-direction:

- `restrict_semi` restricts a residual $r \in \mathbb{R}^{N_x+1 \times N_y+1}$ only along x-direction and returns the restricted residual $r^c \in \mathbb{R}^{\frac{N_x}{2}+1 \times N_y+1}$. The restriction rule reads

$$r_{ij}^c := \frac{1}{4}(r_{2(i-1)+1,j} + r_{2(i-1),j}) + \frac{1}{2}r_{2(i-1)+1,j}. \quad (4)$$

- `interpolate_semi` prolongates an error $e \in \mathbb{R}^{\frac{N_x}{2}+1 \times N_y+1}$ to a finer grid, resulting in a prolonged error $e^f \in \mathbb{R}^{N_x+1 \times N_y+1}$. The interpolation rules read

$$e_{ij}^f := \begin{cases} \frac{1}{2}(e_{i/2,j} + e_{i/2+1,j}) & \text{if } (i,j) \text{ is centered between} \\ & \text{2 x-aligned coarse grid points} \\ e_{(i+1)/2,j} & \text{if } (i,j) \text{ coincides with a coarse} \\ & \text{grid point.} \end{cases} \quad (5)$$

Incorporate the new coarsening and interpolation schemes into the function `vCycle(...)`. The coarsest grid contains more than one inner grid point in case of semi-coarsening; what do you have to change in your current v-cycle implementation so that you obtain convergence? Measure the number of iteration steps for different grid levels. What do you observe?