Fundamentals of Simulation Methods

Exercise Sheet 5

Daniel Rosenblüh, Janosh Riebesell

November 27th, 2015

Multigrid solver for linear systems

1 Galerkin coarse grid approximation

Consider the one-dimensional problem

$$\frac{\partial^2 \Phi}{\partial^2 x} = 4\pi G \rho(x), \tag{1}$$

which we want to solve using a multigrid accelerated iterative method on a grid of size $N = 8$ with spacing h. The problem can be rephrased in the form

$$\mathbf{A}\boldsymbol{x} = \boldsymbol{b}, \tag{2}$$

where $\boldsymbol{x} \mathrel{\hat{=}} \Phi$ and $\boldsymbol{b} \mathrel{\hat{=}} 4\pi G \rho h^2$ are vectors of size N and $\mathbf{A}$ is an $N \times N$-matrix. Given the operator

$$\mathbf{A}^{(h)} = \begin{pmatrix} -2 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 1 & -2 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & -2 \end{pmatrix} \tag{3}$$

on the finest grid level ($N = 8$), find the operator $\mathbf{A}^{(2h)}$ on the next coarser grid level ($N = 4$, spacing $2h$). In order to do so, carry out the following steps:

(a) Find the prolongation and restriction operators in matrix form,

$$\mathbf{I}^{(h)}_{(2h)} : \begin{cases} x^{(h)}_{2i} = x^{(2h)}_{i}, \\ x^{(h)}_{2i+1} = \frac{1}{2}\left(x^{(2h)}_{i} + x^{(2h)}_{i+1}\right), \end{cases} \tag{4}$$

$$\mathbf{I}^{(2h)}_{(h)} : x^{(2h)}_{i} = \frac{1}{4}\left(x^{(h)}_{2i-1} + 2x^{(h)}_{2i} + x^{(h)}_{2i+1}\right), \tag{5}$$

where $0 \le i < N/2$. Hint: $\mathbf{I}^{(h)}_{(2h)}\boldsymbol{x}^{(h)} = \boldsymbol{x}^{(2h)}$ and vice versa.

(b) Now calculate the operator $\mathbf{A}^{(2h)}$ on the coarse grid using the Galerkin coarse grid approximation $\mathbf{A}^{(2h)}_{\text{Gal}} = \mathbf{I}^{(2h)}_{(h)}\mathbf{A}^{(h)}\mathbf{I}^{(h)}_{(2h)}$.

(c) Compare the result to an operator obtained by direct discretization on the coarse grid.

(a) The notation employed in part (a) is quite confusing. We attempt a more natural way of expressing eqs. (4) and (5) by introducing

$$\begin{aligned} \mathbf{I}_{(2h)}^{(h)} &\equiv \mathbf{P}^{(2h\rightarrow h)}, \\ \mathbf{I}_{(h)}^{(2h)} &\equiv \mathbf{R}^{(h\rightarrow 2h)}, \end{aligned} \tag{6}$$

and resorting to the well-known index notation:

$$x_i^{(h)} = \sum_{j=1}^{N} P_{ij}^{(2h\rightarrow h)} x_j^{(2h)}, \tag{7}$$

$$x_i^{(2h)} = \sum_{j=1}^{N} R_{ij}^{(h\rightarrow 2h)} x_j^{(h)}. \tag{8}$$

Looking at eqs. (7) and (8), it is now clear that $\mathbf{P}^{(2h\rightarrow h)}$ and $\mathbf{R}^{(h\rightarrow 2h)}$ are $N \times N/2$- and $N/2 \times N$-matrices, respectively.

To implement eqs. (4) and (5), $\mathbf{P}^{(2h\rightarrow h)}$ and $\mathbf{R}^{(h\rightarrow 2h)}$ must take the form

$$\mathbf{P}^{(2h\rightarrow h)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1/2 & 1/2 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1 \\ 1/2 & 0 & 0 & 1/2 \end{pmatrix} \tag{9}$$

$$\mathbf{R}^{(h\rightarrow 2h)} = \begin{pmatrix} 1/2 & 1/4 & 0 & 0 & 0 & 0 & 0 & 1/4 \\ 0 & 1/4 & 1/2 & 1/4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/4 & 1/2 & 1/4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/4 & 1/2 & 1/4 \end{pmatrix} \tag{10}$$

(b) Multiplying the operator $\mathbf{A}^{(h)}$ from the left with a restriction and from the right with a prolongation, we find

$$\mathbf{A}_{\text{Gal}}^{(2h)} = \mathbf{R}^{(h\rightarrow 2h)}\mathbf{A}^{(h)}\mathbf{P}^{(2h\rightarrow h)} = \begin{pmatrix} -1/2 & 1/4 & 0 & 1/4 \\ 1/4 & -1/2 & 1/4 & 0 \\ 0 & 1/4 & -1/2 & 1/4 \\ 1/4 & 0 & 1/4 & -1/2 \end{pmatrix} = \frac{1}{4}\begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 1 & 0 & 1 & -2 \end{pmatrix}. \tag{11}$$

(c) Setting out from the coarse grid instead with only $N/2$ points per dimension spaced $2h$ apart, the discretized Poisson equation would be given by

$$\frac{\Phi_{i+1} - 2\Phi_i + \Phi_{i-1}}{(2h)^2} = 4\pi G\rho_i, \tag{12}$$

where the index i now runs from $i \in \{0, 1, 2, \ldots, \frac{N}{2} - 1\}$ if N is still taken to be the number of points on the fine grid. If we then write this Poisson equation in the form of eq. (2), i.e. as a system of linear equations with the components of $\boldsymbol{b}$ now given by

$$b_i = 4\pi G(2h^2)\rho_i, \tag{13}$$

we see that $\mathbf{A}^{(2h)}$ has to be

$$\mathbf{A}^{(2h)} = \begin{pmatrix} -2 & 1 & 0 & 1 \\ 1 & -2 & 1 & 0 \\ 0 & 1 & -2 & 1 \\ 1 & 0 & 1 & -2 \end{pmatrix} = 4\mathbf{A}^{(2h)}_{\mathrm{Gal}}. \tag{14}$$

This $\mathbf{A}^{(2h)}$ is larger by a factor of 4 than the one we found in part (b), as it should be since its corresponding $\boldsymbol{b}$ is also four times larger.

2 Iteratively solving Poisson's equation

(a) Suppose we want to solve for the gravitational potential Φ on a *periodic* mesh with square-shaped domain $[0, L] \times [0, L]$ (with $L = 1$ for definiteness) containing $N \times N$ cells and housing the mass density distribution

$$\rho(\boldsymbol{r}) = \rho_0 \exp\left(-\frac{\boldsymbol{r}^2}{2\eta^2}\right), \tag{15}$$

where $\eta = \frac{L}{10}$ is the spread of the mass spike, and $\rho_0 = 10$. Discretization of the Poisson equation gives an iteration rule where the left-hand side gives the updated values in terms of the old values on the right-hand side, in the form

$$x^{(n+1)}_{i,j} = \frac{1}{4}\left(x^{(n)}_{i-1,j} + x^{(n)}_{i+1,j} + x^{(n)}_{i,j-1} + x^{(n)}_{i,j+1} - b_{i,j}\right), \tag{16}$$

where $x_{i,j} = \Phi_{i,j}$ and $b_{i,j} = 4\pi G h^2 \rho_{i,j} - \bar{b}$. This can be readily used for Jacobi or Gauss-Seidel iteration schemes. Write a function `jacobi_step(x[], b[], N)` that replaces the input array `x[]` with a correspondingly updated array after one iteration step.

(b) To characterize the current error in the solution, write a function that calculates the residual

$$r_{i,j} = b_{i,j} - (\mathbf{A}\boldsymbol{x})_{i,j}. \tag{17}$$

Also, write a function that computes the norm

$$S = \left(\sum_{i,j} r^2_{i,j}\right)^{1/2} \tag{18}$$

of this vector and returns it. We can use S as a simple measure of the overall error.

(c) Now specialize to the case of $N = 256$ and write a routine that suitably initializes the density field, with the density peak placed in the center of the box. Set the initial guess for the potential to $\boldsymbol{x} = \mathbf{0}$. Carefully think about how to initialize the array $\boldsymbol{b}$ (for a periodic system, eq. (2) is only solvable for $\langle \boldsymbol{b} \rangle = \mathbf{0}$ because the matrix $\mathbf{A}$ is singular, therefore $\bar{b}$ has to be chosen appropriately). Now add a loop that calls the Jacobi iteration $N_{\mathrm{steps}} = 2000$ times, and after each step, determines the norm of the residual. You may use the C or Python template on the Moodle site and fill in the missing parts in order to reduce the coding work, or write everything from scratch if you prefer. Make a plot of the decay of the log of this residual as a function of step number. What do you expect to get for $\sum_{i,j} \Phi_{i,j}$?

(d) Now produce a second version of your program in which the Jacobi iteration is replaced by Gauss-Seidel iteration where new values for elements of $\boldsymbol{x}$ are used as soon as they become available, replacing any old value in the array.

(e) Change the Gauss-Seidel scheme to a red-black Gauss-Seidel iteration, i.e. you first update the red cells in a chess-board pattern overlaid over the mesh, then the black cells in a second pass. Produce a common plot of the decay rates of the residual for N_{steps} steps with the three iteration variants considered thus far. Interpret the results. The red-black scheme result will perhaps disappoint you at first – what's going on?

(f) We now solve the problem with multigrid acceleration. First, write a function that carries out a restriction step of a mesh with dimension $N \times N$ (where N is a power of 2) onto a coarser mesh with dimension $N/2 \times N/2$. Make each point in the coarser mesh a weighted average of neighboring points, according to the stencil

$$\mathbf{T} = \begin{bmatrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{bmatrix}. \tag{19}$$

Give this function a calling signature `do_restrict(N, fine[], NN, coarse[])` (only `NN` = `N/2` is allowed when calling this function).

(g) Next, write a function that carries out a prolongation step of a mesh of dimension $N \times N$ onto a finer mesh of dimension $2N \times 2N$. Each point in the coarser mesh is additively injected into the finer mesh with weights

$$\mathbf{U} = \left] \begin{matrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{matrix} \right[. \tag{20}$$

Note that points of the fine mesh that see a coarse point with weight 1/4 will get contributions from 4 coarse points in total, similarly for the points with weight 1/2, while the fine-mesh point with weight 1 simply inherits the value of one of the coarse mesh points. Give this function a calling signature `do_prolong(NN, coarse[], N, fine[])` (again, only `NN` = `N/2` is allowed when calling this function).

(h) Now write a function `do_v_cycle(x[], b[], N)` that carries out a V-cycle multigrid iteration on the current solution vector. The steps of the function should be

1. Do a Gauss-Seidel step.
2. Calculate the residuum.
3. Restrict the residuum to a coarser mesh $N' = N/2$, and scale it by a factor of 4 to effectively take care of the $1/h \leftarrow 1/(2h)$ scaling of the differential operator (whose explicit form we leave invariant, for simplicity).
4. Call `do_v_cycle` recursively for the coarser mesh, with a zero error vector as starting guess for `x`, and the coarsened residual for `b`.
5. Now prolong the returned error vector to the fine mesh N.
6. Add this to the current solution vector on the fine mesh.
7. Do another Gauss-Seidel step.

Steps 2 to 6 are only done provided $N > 4$. To simplify the coding, you may use one of the code templates provided on the Moodle site.

(i) Finally, solve the original problem by repeatedly calling your V-cycle iteration. Again, plot the residual as a function of the number of steps taken, and compare with the results obtained earlier for plain Jacobi and Gauss-Seidel iteration.

For parts (a) and (b), see `jacobi.c`.

(c) A half-logarithmic plot of the decay rate of the norm S of the residual is shown in fig. 1.

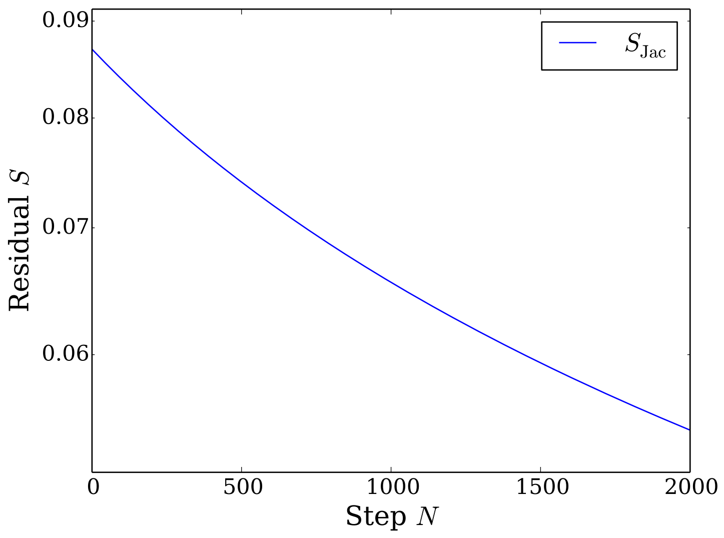


Figure 1: Decay of Jacobian residual norm S

(d) See `gausseidel.c`.

(e) Figure 2 displays the decay rates of the residual norm as produced by the Jacobi method and the Gauss-Seidel method with and without chessboard scheme.

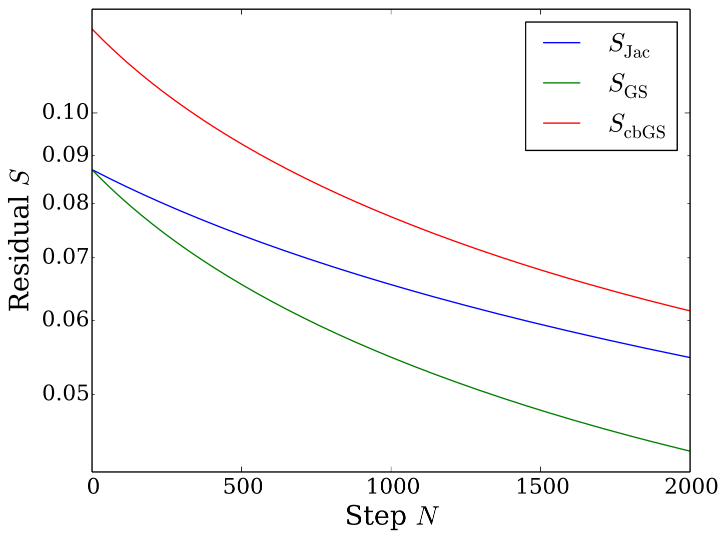


Figure 2: Gauss-Seidel residual decay rates compared to to that of the Jacobi method

As expected, we see that Gauss-Seidel's superior utilization of available information leads to the norm of its residual S_{GS} decaying considerably faster than that of Jacobi. The same improvement is not observed by further implementing a chessboard-like update scheme for the potential. The decay rate appears largely unchanged except for a spike during the very first step suggesting an ill-suited initial density distribution for this method.

For parts (f) to (h), see `multigrid.c`.

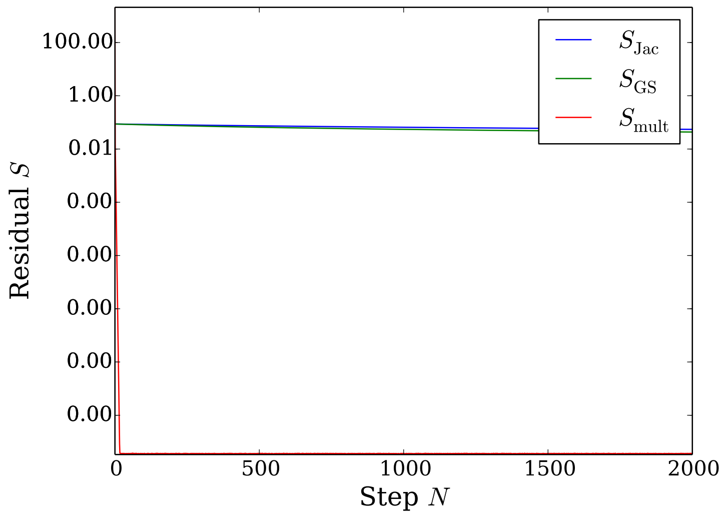
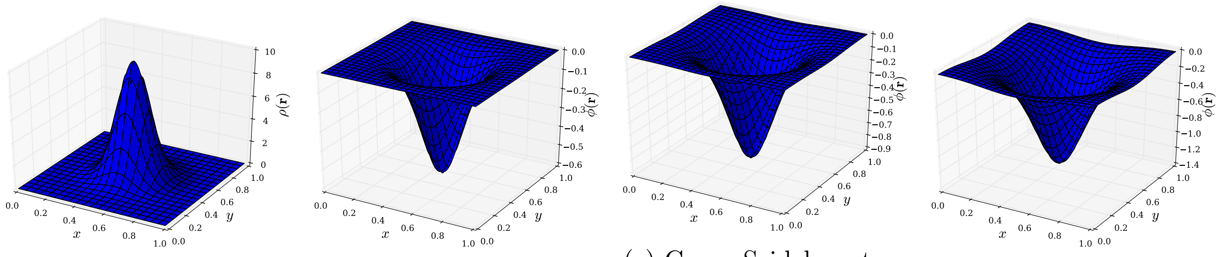


Figure 3: Residual decay of the multigrid method compared to Jacobi and Gauss-Seidel

(i) Figure 3 compares the decay rate produced by the multigrid method to those of the Jacobi and Gauss-Seidel methods. We indeed find a dramatic increase in the convergence rate.

Note: A surface plot of the initial density distribution $\rho_{\text{i}}(\boldsymbol{r})$ over the entire simulation domain is shown in fig. 4a. Figures 4b to 4d display the noticeably differing corresponding potentials according to the Jacobi, Gauss-Seidel and multigrid methods, respectively. Since each method convergences at a different rate, plotting the potential after a constant number of 2000 sweeps is akin to plotting it at different times.



(a) Initial density (b) Jacobi potential (c) Gauss-Seidel potential (d) Multigrid potential

Figure 4: Surface plots of the initial density and calculated potentials