

Chapter 9

Elliptic Equations and Multigrid

9.1 Elliptic equations

The simplest elliptic PDE is *Laplace's equation*:

$$\nabla^2 \phi = 0 \tag{9.1}$$

Only slightly more complex is *Poisson's equation* (Laplace + a source term):

$$\nabla^2 \phi = f \tag{9.2}$$

These equations can arise in electrostatics (for the electric potential), solving for the gravitational potential from a mass distribution, or enforcing a divergence constraint on a vector field (we'll see this when we consider incompressible flow).

Another common elliptic equation is the *Helmholtz equation*:

$$(\alpha - \nabla \cdot \beta \nabla) \phi = f \tag{9.3}$$

A Helmholtz equation can arise, for example, from a time-dependent equation (like diffusion) by discretizing in time.

Notice that there is no time-dependence in any of these equations. The quantity ϕ is specified instantaneously in the domain subject to boundary conditions. This makes the solution methods very different than what we saw for hyperbolic problems.

9.2 Fourier Method

A direct way of solving a constant-coefficient elliptic equation is using Fourier transforms. Using a general Fourier transform (which we consider here) works only for

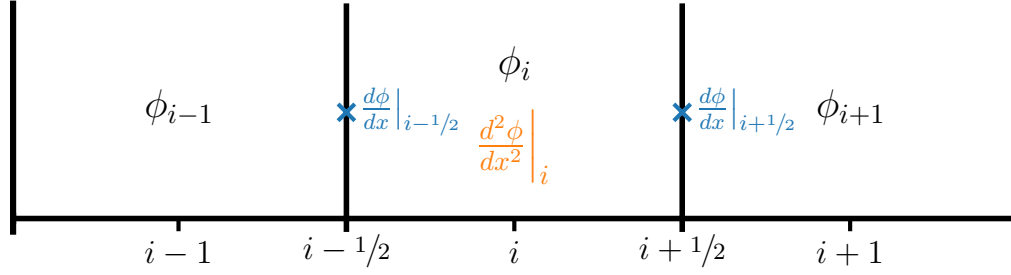


Figure 9.1: The centerings of the first and second derivatives for a standard Laplacian discretization. Our data, ϕ , is cell-centered. The first-derivatives, $d\phi/dx$, are edge-centered, and the second-derivative, $d^2\phi/dx^2$, is cell-centered.

periodic boundary conditions, but other basis functions (e.g., all sines or all cosines) can be used for other boundary conditions.

Consider the Poisson equation:

$$\nabla^2 \phi = f \quad (9.4)$$

We will difference this in a second-order accurate fashion—see Figure 9.1. In 1-d, the Laplacian is just the second-derivative. If our solution is defined at cell-centers, then we first compute the first-derivative on cell edges:

$$\left. \frac{d\phi}{dx} \right|_{i-1/2} = \frac{\phi_i - \phi_{i-1}}{\Delta x} \quad (9.5)$$

$$\left. \frac{d\phi}{dx} \right|_{i+1/2} = \frac{\phi_{i+1} - \phi_i}{\Delta x} \quad (9.6)$$

These are second-order accurate on the interface. We can then compute the second-derivative at the cell-center by differencing these edge values:

$$\left. \frac{d^2\phi}{dx^2} \right|_i = \frac{d\phi/dx|_{i+1/2} - d\phi/dx|_{i-1/2}}{\Delta x} \quad (9.7)$$

The extension to 2-d is straightforward. Thinking of the Laplacian as $\nabla^2 \phi = \nabla \cdot \nabla \phi$, we first compute the gradient of ϕ on edges:

$$[\nabla \phi \cdot \hat{x}]_{i+1/2,j} = \frac{\phi_{i+1,j} - \phi_{i,j}}{\Delta x} \quad (9.8)$$

$$[\nabla \phi \cdot \hat{y}]_{i,j+1/2} = \frac{\phi_{i,j+1} - \phi_{i,j}}{\Delta y} \quad (9.9)$$

Again, since this is defined on edges, this represents a centered difference, and is therefore second-order accurate. We then difference the edge-centered gradients to

the center to get the Laplacian at cell-centers:

$$\begin{aligned} [\nabla^2 \phi]_{i,j} &= \frac{[\nabla \phi \cdot \hat{x}]_{i+1/2,j} - [\nabla \phi \cdot \hat{x}]_{i-1/2,j}}{\Delta x} + \frac{[\nabla \phi \cdot \hat{y}]_{i,j+1/2} - [\nabla \phi \cdot \hat{y}]_{i,j-1/2}}{\Delta y} \\ &= \frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \end{aligned} \quad (9.10)$$

Again, since we used a centered-difference of the edge values, this expression is second-order accurate. This is the standard *5-point stencil* for the 2-d Laplacian*.

We now assume that we have an FFT subroutine (see § 1.2.6) that can take our discrete real-space data, $\phi_{i,j}$ and return the discrete Fourier coefficients, Φ_{k_x,k_y} , and likewise for the source term:

$$\Phi_{k_x,k_y} = \mathcal{F}(\phi_{i,j}) \quad F_{k_x,k_y} = \mathcal{F}(f_{i,j}) \quad (9.11)$$

The power of the Fourier method is that derivatives in real space are multiplications in Fourier space, which makes the solution process in Fourier space straightforward.

We now express $\phi_{i,j}$ and $f_{i,j}$ as sums over their Fourier components. Here we define M as the number of grid points in the x -direction and N as the number of grid points in the y -direction. As before, we are using i as the grid index, we will use I as the imaginary unit:

$$\phi_{i,j} = \frac{1}{MN} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} \Phi_{k_x,k_y} e^{2\pi I i k_x / M} e^{2\pi I j k_y / N} \quad (9.12)$$

$$f_{i,j} = \frac{1}{MN} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} F_{k_x,k_y} e^{2\pi I i k_x / M} e^{2\pi I j k_y / N} \quad (9.13)$$

Inserting these into the differenced equation, we have:

$$\begin{aligned} \frac{1}{MN} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} \left\{ \frac{\Phi_{k_x,k_y}}{\Delta x^2} e^{2\pi I j k_y / N} \left[e^{2\pi I (i+1) k_x / M} - 2e^{2\pi I i k_x / M} + e^{2\pi I (i-1) k_x / M} \right] + \right. \\ \left. \frac{\Phi_{k_x,k_y}}{\Delta y^2} e^{2\pi I i k_x / M} \left[e^{2\pi I (j+1) k_y / N} - 2e^{2\pi I j k_y / N} + e^{2\pi I (j-1) k_y / N} \right] \right\} = \\ \frac{1}{MN} \sum_{k_x=0}^{M-1} \sum_{k_y=0}^{N-1} F_{k_x,k_y} e^{2\pi I i k_x / M} e^{2\pi I j k_y / N} \end{aligned} \quad (9.14)$$

We can bring the righthand side into the sums on the left, and we can then look at

*There are other possible second-order accurate stencils, including a 9-point stencil in 2-d, that are less commonly used.

just a single (k_x, k_y) term in the series:

$$e^{2\pi I k_x / M} e^{2\pi I j k_y / N} \left\{ \frac{\Phi_{k_x, k_y}}{\Delta x^2} \left[e^{2\pi I k_x / M} + e^{-2\pi I k_x / M} - 2 \right] + \frac{\Phi_{k_x, k_y}}{\Delta y^2} \left[e^{2\pi I k_y / N} + e^{-2\pi I k_y / N} - 2 \right] - F_{k_x, k_y} \right\} = 0 \quad (9.15)$$

Simplifying, we have:

$$\Phi_{k_x, k_y} = \frac{1}{2} \frac{F_{k_x, k_y}}{[\cos(2\pi k_x / M) - 1] \Delta x^{-2} + [\cos(2\pi k_y / N) - 1] \Delta y^{-2}} \quad (9.16)$$

This is the algebraic solution to the Poisson equation in Fourier (frequency) space. Once we evaluate this, we can get the real-space solution by doing the inverse transform:

$$\phi_{i,j} = \mathcal{F}^{-1}(\Phi_{k_x, k_y}) \quad (9.17)$$

We can test this technique with the source term:

$$f = 8\pi^2 \cos(4\pi y) [\cos(4\pi x) - \sin(4\pi x)] - 16\pi^2 [\sin(4\pi x) \cos(2\pi y)^2 + \sin(2\pi x)^2 \cos(4\pi y)] \quad (9.18)$$

which has the analytic solution[†]:

$$\phi = \sin(2\pi x)^2 \cos(4\pi y) + \sin(4\pi x) \cos(2\pi y)^2 \quad (9.19)$$

Note that this solution has the required periodic behavior. Figure 9.2 shows the solution.

The main downside of this approach is that, because we solve for a single component independently (Eq. 9.16), this only works for linear problems with constant coefficients. This makes it an excellent choice for cosmological problems solving the gravitational Poisson equation with periodic boundaries on all sides of the domain (see, e.g., [41]). However, for a problem like:

$$\nabla \cdot (\beta \nabla \phi) = f \quad (9.20)$$

there would be “cross-talk” between the Fourier modes of β and ϕ , and we would not be able to solve for a single mode of Φ_{k_x, k_y} independently. We discuss more general methods that work for these forms next.

[†]Note: throughout this chapter, we devise test problems by picking a function that meets the desired boundary conditions and then inserting it into the analytic equation we are solving to find the righthand side

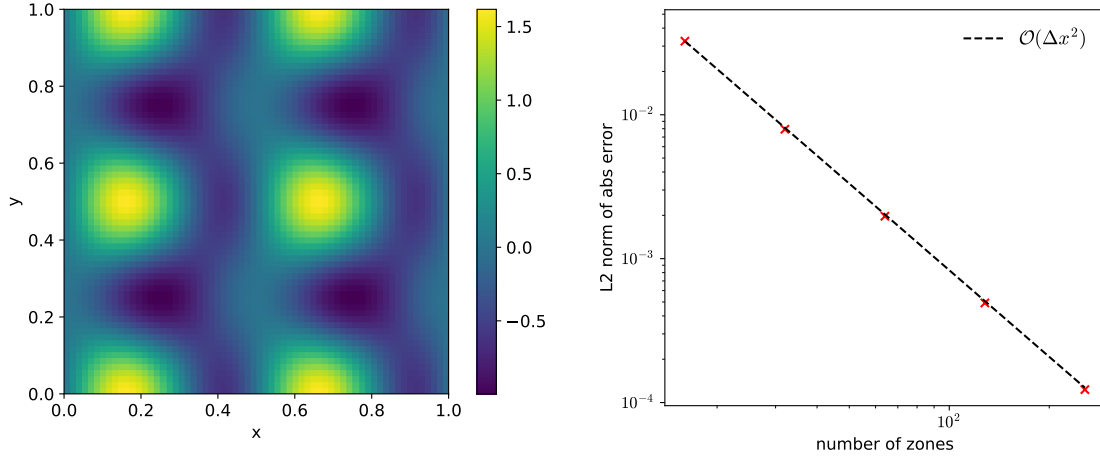


Figure 9.2: (left) Solution to the Poisson equation on a 64^2 grid with source from Eq. 9.18. (right) Error vs. the true solution as a function of resolution for the Fourier method, showing second-order convergence. `hydro_examples: poisson_fft.py`

9.3 Relaxation

Relaxation is an iterative technique, and as we will see shortly, it provides the basis for the multigrid technique.

Consider Poisson's equation, again differenced as:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \quad (9.21)$$

For each zone (i, j) , we couple in the zones ± 1 in x and ± 1 in y . For the moment, consider the case where $\Delta x = \Delta y$. If we solve this discretized equation for $\phi_{i,j}$, then we have:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \quad (9.22)$$

A similar expression exists for every zone in our domain, coupling all the zones together. We can't separate the solution of $\phi_{i,j}$ for the neighboring zones, but instead can apply an iterative technique called *relaxation* (also sometimes called *smoothing* because generally speaking the solution to elliptic equations is a smooth function) to find the solution for ϕ everywhere.

Imagine an initial guess to ϕ : $\phi_{i,j}^{(0)}$. We can improve that guess by using our difference equation to define a new value of ϕ , $\phi_{i,j}^{(1)}$:

$$\phi_{i,j}^{(1)} = \frac{1}{4}(\phi_{i+1,j}^{(0)} + \phi_{i-1,j}^{(0)} + \phi_{i,j+1}^{(0)} + \phi_{i,j-1}^{(0)} - \Delta x^2 f_{i,j}) \quad (9.23)$$

or generally, the $k+1$ iteration will see:

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)} - \Delta x^2 f_{i,j}) \quad (9.24)$$

This will (slowly) converge to the true solution[†], since each zone is coupled to each other zone (and to the boundary values that we need to specify—more on that in a moment). This form of relaxation is called *Jacobi iteration*. To implement this, you need two copies of ϕ —the old iteration value and the new iteration value.

An alternate way to do the relaxation is to update $\phi_{i,j}$ in place, as soon as the new value is known. Thus the neighboring cells will see a mix of the old and new solutions. We can express this in-place updating as:

$$\phi_{i,j} \leftarrow \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \quad (9.25)$$

This only requires a single copy of ϕ to be stored. This technique is called *Gauss-Seidel iteration*. A host of other relaxation methods exist, including linear combinations of these two. An excellent discussion of these approaches, and their strengths and weaknesses is given in [17].

Next consider the Helmholtz equation with constant coefficients:

$$(\alpha - \beta \nabla^2)\phi = f \quad (9.26)$$

We can discretize this as:

$$\alpha\phi_{i,j} - \beta \left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} \right) = f_{i,j} \quad (9.27)$$

and the update of $\phi_{i,j}$ through relaxation is:

$$\phi_{i,j} \leftarrow \left(f_{i,j} + \frac{\beta}{\Delta x^2}\phi_{i+1,j} + \frac{\beta}{\Delta x^2}\phi_{i-1,j} + \frac{\beta}{\Delta y^2}\phi_{i,j+1} + \frac{\beta}{\Delta y^2}\phi_{i,j-1} \right) / \left(\alpha + \frac{2\beta}{\Delta x^2} + \frac{2\beta}{\Delta y^2} \right) \quad (9.28)$$

Notice that if $\alpha = 0$, $\beta = -1$, and $\Delta x = \Delta y$, we recover the relaxation expression for Poisson's equation from above.

9.3.1 Boundary conditions

When using a cell-centered grid, no points fall exactly on the boundary, so we need to use ghost cells to specify boundary conditions. A single ghost cell is sufficient for the 5-point stencil used here. The common types of boundary conditions are *Dirichlet* (specified value on the boundary), *Neumann* (specified first derivative on the boundary), and periodic. Some restrictions apply (see discuss this later, in § 9.5).

Consider Dirichlet boundary conditions, specifying values ϕ_l on the left and ϕ_r on the right boundaries.[§] We'll label the first zone inside the domain, at the left boundary,

[†]Formally, convergence is only guaranteed if the matrix in our linear system is *diagonally dominant*. The Laplacian used here is not quite diagonally dominant, but these methods still converge.

[§]If the value, ϕ_l or ϕ_r is zero, we call this a *homogeneous boundary condition*. Otherwise we call it an *inhomogeneous boundary condition*.

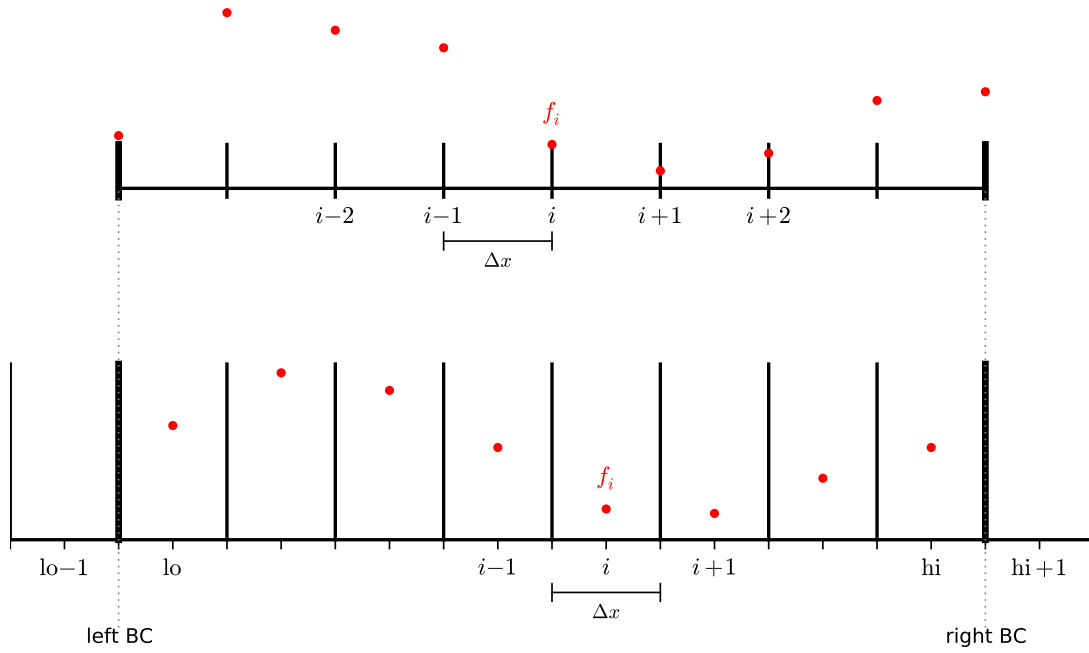


Figure 9.3: A node-centered (top) and cell-centered (bottom) finite difference grid showing the data and domain boundaries. Notice that for the cell-centered grid, there is no data point precisely at the boundary.

lo , and the last zone inside the domain, at the right boundary, hi —see Figure 9.3. To second order, we can average the zone values on either side of the interface to get the boundary condition:

$$\phi_l = \frac{1}{2}(\phi_{lo} + \phi_{lo-1}) \quad (9.29)$$

$$\phi_r = \frac{1}{2}(\phi_{hi} + \phi_{hi+1}) \quad (9.30)$$

This then tells us that the values we need to assign to the ghost cells are:

$$\phi_{lo-1} = 2\phi_l - \phi_{lo} \quad (9.31)$$

$$\phi_{hi+1} = 2\phi_r - \phi_{hi} \quad (9.32)$$

If we instead consider Neumann boundary conditions, we specify values of the derivative on the boundaries: $\phi_x|_l$ on the left and $\phi_x|_r$ on the right. We note that a single difference across the boundary is second-order accurate on the boundary (it is a centered-difference there), so to second-order:

$$\phi_x|_l = \frac{\phi_{lo} - \phi_{lo-1}}{\Delta x} \quad (9.33)$$

$$\phi_x|_r = \frac{\phi_{hi+1} - \phi_{hi}}{\Delta x} \quad (9.34)$$

This then tells us that the ghost cells are filled as:

$$\phi_{lo-1} = \phi_{lo} - \Delta x \phi_x|_l \quad (9.35)$$

$$\phi_{hi+1} = \phi_{hi} + \Delta x \phi_x|_r \quad (9.36)$$

9.3.2 Residual and true error

The *residual error* is a measure of how well our discrete solution satisfies the discretized equation. For the Poisson equation, we can the residual as:

$$r_{i,j} = f_{i,j} - (L\phi)_{i,j} \quad (9.37)$$

and the residual error as:

$$\epsilon^{(r)} = \|r\| \quad (9.38)$$

where L represents our discretized Laplacian. Note that r is the error with respect to the discrete form of the equation. The true error is the measure of how well our discrete solution approximates the true solution. If ϕ^{true} satisfies $\nabla^2 \phi^{\text{true}} = f$, then the true error in each zone is

$$e_{i,j} = \phi^{\text{true}}(x_i, y_j) - \phi_{i,j} \quad (9.39)$$

and

$$\epsilon^{\text{true}} = \|e_{i,j}\| \quad (9.40)$$

We can make $\epsilon^{(r)}$ approach machine precision by performing more and more relaxation iterations, but after some point, this will no longer improve ϵ^{true} . The only way to improve ϵ^{true} is to make Δx and Δy smaller. In practice we do not know the true solution so we cannot compute ϵ^{true} and will instead have to rely on $\epsilon^{(r)}$ to monitor our error.

Note that since our operator is linear,

$$Le = L\phi^{\text{true}} - L\phi = f - L\phi = r \quad (9.41)$$

so the error in our solution obeys a Poisson equation with the residual as the source—we'll see this in the next section.

9.3.3 Norms

There are several different norms that are typically used in defining errors on the grid. The L_∞ norm (or 'inf'-norm) is just the maximum error on the grid:

$$\|e\|_\infty = \max_{i,j} \{|e_{i,j}|\} \quad (9.42)$$

This will pick up on local errors.

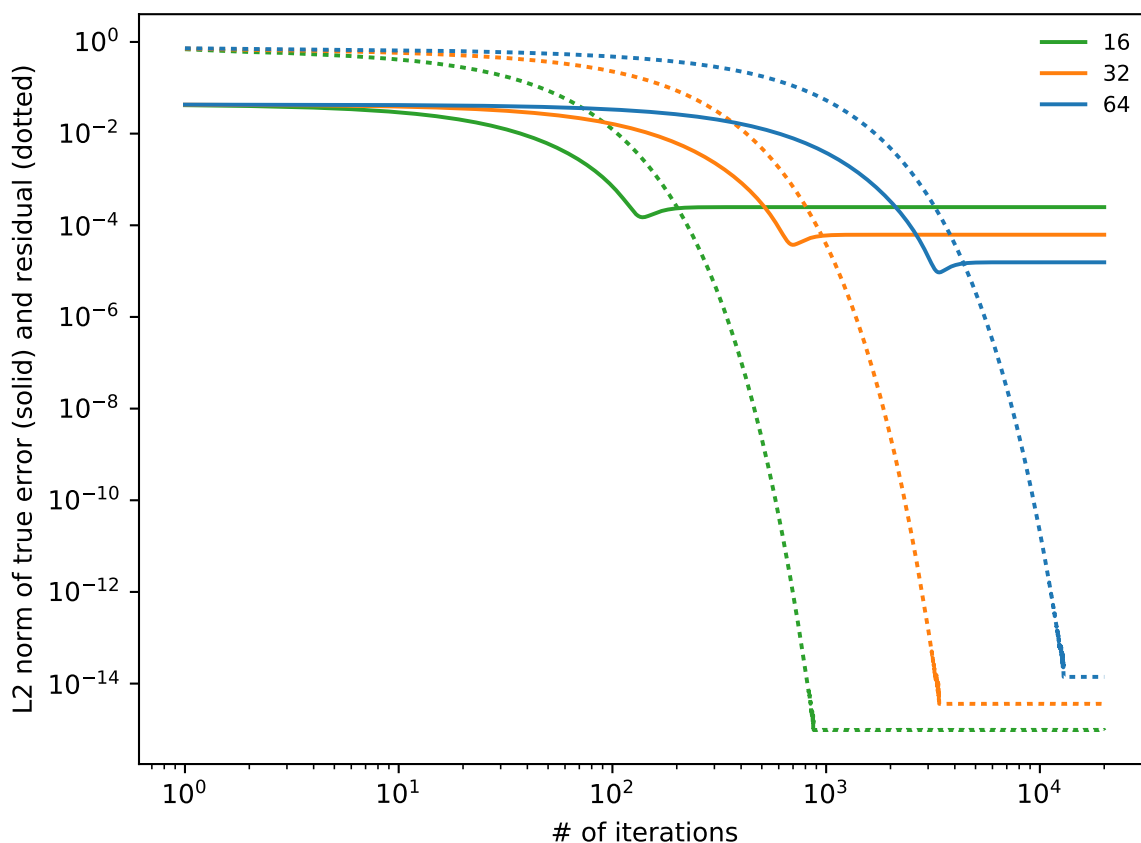


Figure 9.4: Gauss-Seidel relaxation applied to $\phi_{xx} = \sin(x)$ with $\phi(0) = \phi(1) = 0$. Shown are the L2 norm of the error compared with the true solution (solid lines) and the L2 norm of the residual (dotted lines) for 3 different resolutions (16, 32, and 64 zones).

 hydro_examples: smooth.py

The L_1 norm and L_2 norms are more global.

$$\|e\|_1 = \frac{1}{N} \sum_{i,j} |e_{i,j}| \quad (9.43)$$

$$\|e\|_2 = \left(\frac{1}{N} \sum_{i,j} |e_{i,j}|^2 \right)^{1/2} \quad (9.44)$$

Generally, the measure in L_2 falls between L_∞ and L_1 . Regardless of the norm used, if the problem converges, it should converge in all norms. For reference, the AMReX library[¶] uses L_∞ in its multigrid solvers.

9.3.4 Performance

Consider the simple Poisson problem¹¹ on $x \in [0, 1]$:

$$\phi_{xx} = \sin(x), \quad \phi(0) = \phi(1) = 0 \quad (9.45)$$

The analytic solution to this is simply

$$\phi^a(x) = -\sin(x) + x \sin(1) \quad (9.46)$$

We can perform smoothing and compute both the error against the analytic solution (the ‘true’ error), $e \equiv \|\phi^a(x_i) - \phi_i\|_2$ and the residual error, $\|r_i\|_2$. Figure 9.4 shows these errors as a function of the number of smoothing iterations for 3 different resolutions.

Notice that the true error stalls at a relatively high value—this is the truncation error of the method. From one resolution to the next, the true error changes as Δx^2 , indicating that we are converging as our method should. No additional amount of smoothing will change this—we are getting the best answer to the problem we can with our choice of discretization. In contrast, the residual error decreases to machine precision levels—this is indicating that our solution is an exact solution to the discrete equation (to roundoff-error). In practice, we can only monitor the residual error, not the true error, and we hope that small residual error implies a small true error.

Figure 9.5 shows the error with respect to the true solution and of the residual for pure smoothing in three different norms. The overall behavior is qualitative similar regardless of the choice of norm.

To demonstrate the influence of the boundary conditions, Figure 9.6 shows the norm of the true error for the same problem, but this time with a naive implementation of the boundary conditions—simply initializing the ghost cell to the boundary value, instead of averaging to the interface:

$$\phi_{lo-1} = \phi_{lo} \quad (9.47)$$

$$\phi_{hi+1} = \phi_{hi} \quad (9.48)$$

We see that with this mistake at the boundaries, the error of the entire solution is affected, and we get only first-order convergence with resolution (this can be seen by looking at the spacing of the curves). The previous solution, with the correct BCs is shown for reference, and shows second-order convergence. It is important to note that because every zone is linked to every other zone (and the boundary) in elliptic problems, an error at the boundary can pollute the global solution.

¹¹<https://github.com/amrex-codes>

¹¹This is the test problem used throughout *A Multigrid Tutorial* [17]. We use the same problem here to allow for easy comparison to the discussions in that text.

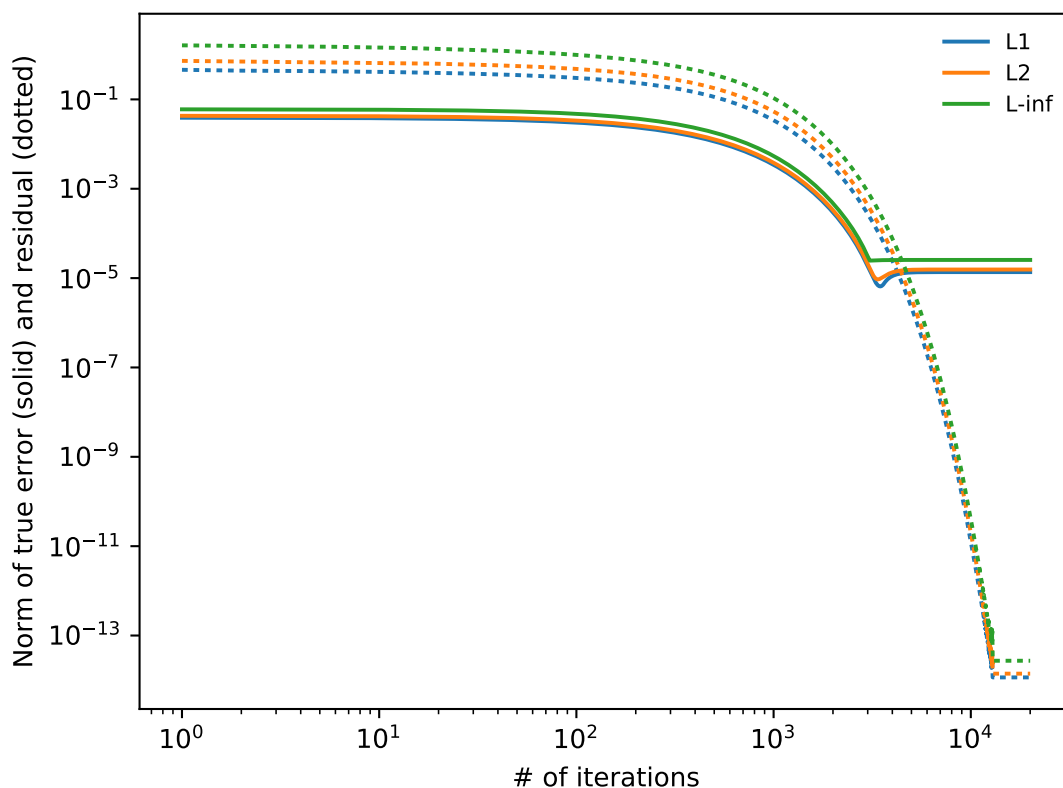



Figure 9.5: Gauss-Seidel relaxation applied to $\phi_{xx} = \sin(x)$ with $\phi(0) = \phi(1) = 0$. This is like figure 9.4, but now we show the error in 3 different norms: L_1 , L_2 , and L_∞ .

 hydro_examples: smooth-norms.py

9.3.5 Frequency/wavelength-dependent error

We can think of the error in the solution as a superposition of high (short) and low (long) frequency (wavelength) modes. Smoothing works really well to eliminate the short wavelength noise quickly, but many iterations are needed to remove the long wavelength noise (see Figure ??). A very important concept to understand here is that when we talk about long wavelength error, we are expressing it in terms of the number of zones across the feature, not as physical length. We can get an intuitive feel for this behavior by thinking about how smoothing works. Every zone is coupled to every other zone, and we can think about each zone seeing one more zone away for each iteration. When the error is short wavelength, that means that there are only a few zones across it, and after a few iterations, all of the zones have seen the short wavelength error, and can eliminate it. For a very long wavelength error, many iterations will be needed until the smoothing couples one zone to another that is a wavelength away.

This behavior suggests that if we could represent our problem on a coarser grid, the error will now be of shorter wavelength, and smoothing will be more efficient. This

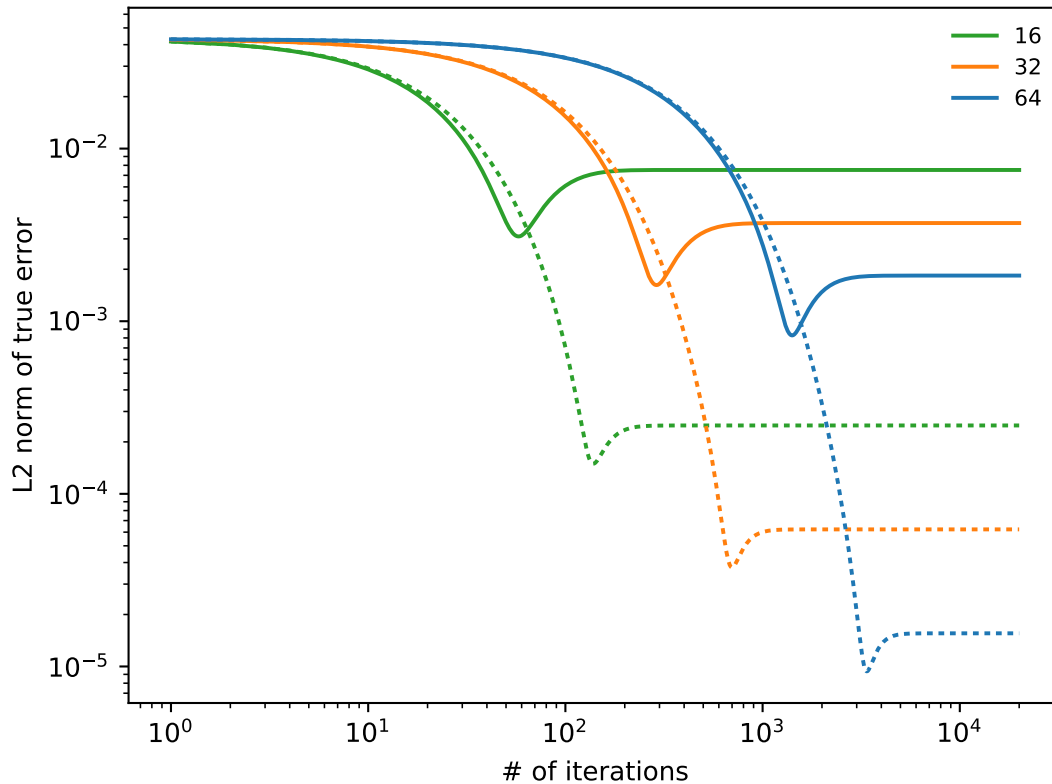


Figure 9.6: The same problem as in figure 9.4, but now we done with a naive first-order boundary conditions—just initializing the ghost cell to the boundary value (solid lines). We see that this achieves only first-order convergence in the true error. The correct second-order implmention is shown as the dotted lines.

 hydro_examples: smooth-badbcs.py

is the core idea behind multigrid, which we see next.

Exercise 9.1

Implement 1-d smoothing for the Laplace equation on cc-grid. Use an initial guess for the solution:

$$\phi_0(x) = \frac{1}{3}(\sin(2\pi x) + \sin(2\pi 8x) + \sin(2\pi 16x)) \quad (9.49)$$

on a 128 zone grid with Dirichlet boundary conditions. This initial guess has both high-frequency and low-frequency noise. Observe that the high-frequency stuff goes after only a few smoothing iterations, but many iterations are needed to remove the low-frequency noise. You should see something like Figure ??.

Figure 9.7 illustrates this behavior. We are solving the Laplace equation, $\phi'' = 0$ in 1-

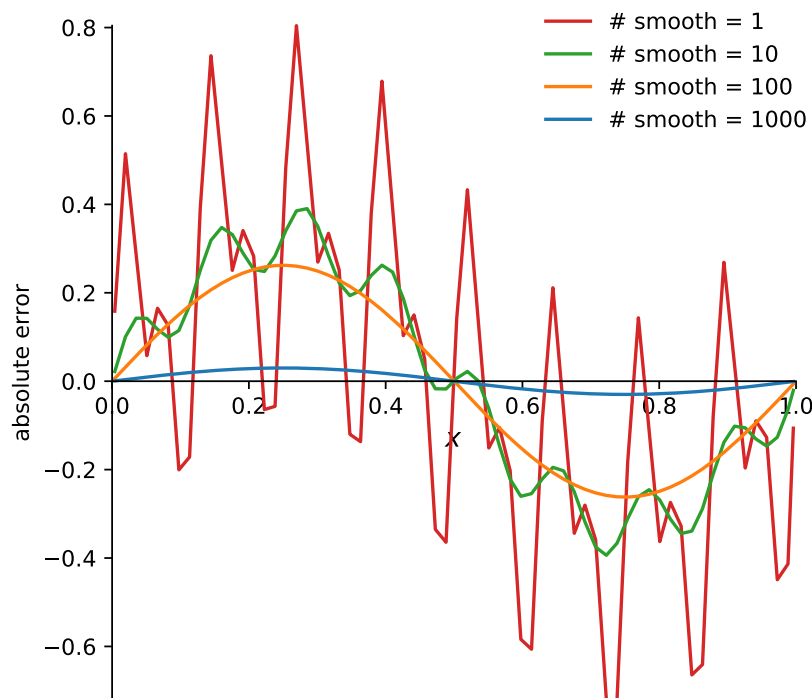


Figure 9.7: Error in the solution to $\phi'' = 0$ given an initial guess with 3 different wavenumbers of noise. A 128 zone grid was used. The different curves are different numbers of smoothing iterations.

 hydro_examples: smooth-modes.py

d on $[0, 1]$ with homogeneous Dirichlet boundary conditions. The solution is simply $\phi = 0$. We use Eq. 9.49 as an initial guess—this is a superposition of 3 different modes. We see that the after just a few iterations, the shortest wavelength mode, $\sin(2\pi 16x)$ is no longer apparent in the error, and just the two longer wavelength modes dominate. By 100 iterations, the error appears to be only due to the longest wavelength mode, $\sin(2\pi x)$. Even after 1000 iterations, we still see this mode in the error. This demonstrates that the longest wavelength modes in the error take the longest to smooth away.

9.4 Multigrid

The text *A Multigrid Tutorial* [17] provides an excellent introduction to the mechanics of multigrid. The discussion here differs mainly in that we are dealing with cell-centered/finite-volume data. We already saw that the treatment of boundary condi-

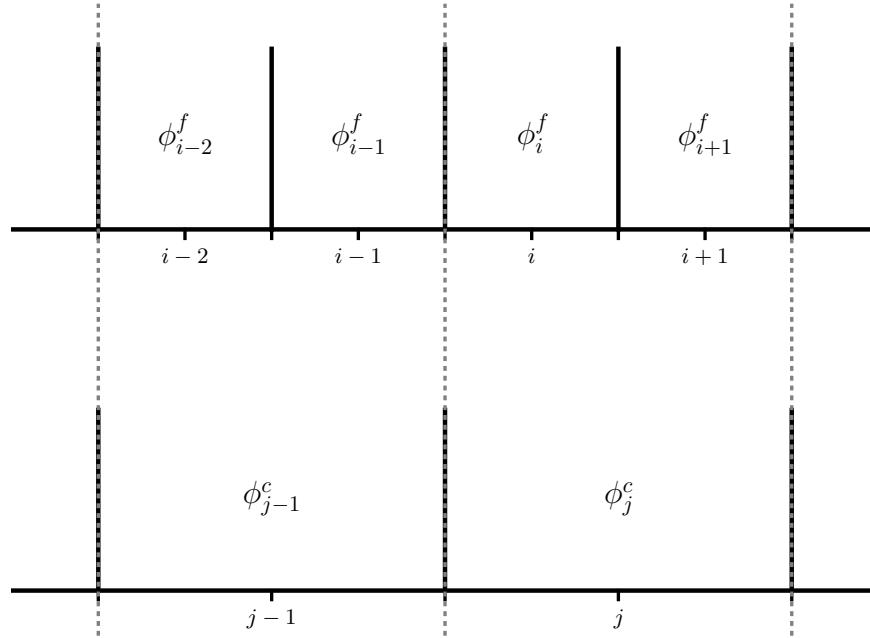


Figure 9.8: A fine grid and corresponding underlying coarse grid.

tions is more complicated because we do not have a point directly on the boundary. The other complication comes in transferring the data from a fine grid to a coarser grid, and back. Before we discuss the multigrid technique, we'll understand how the operations between grids work.

9.4.1 Prolongation and restriction on cell-centered grids

Multigrid relies on transferring the problem up and down a hierarchy of grids. The process of moving the data from the fine grid to the coarse grid is called *restriction*. The reverse process, moving data from the coarse grid to the fine grid is called *prolongation*. If the data on our grid is a conserved quantity, we want restriction and prolongation to conserve the amount of stuff when transitioning data between the fine and coarse grids.

1-d

Consider a 1-d finite-volume/cell-centered finite-difference grid shown in Figure 9.8—we see a fine grid and the corresponding coarser grid. If ϕ represents a density, then conservation requires:

$$\phi_j^c = \frac{1}{\Delta x^c} (\Delta x^f \phi_i^f + \Delta x^f \phi_{i+1}^f) \quad (9.50)$$

or, for a jump in 2 in resolution ($\Delta x^c = 2\Delta x^f$),

$$\phi_j^c = \frac{1}{2}(\phi_i^f + \phi_{i+1}^f) \quad (9.51)$$

This latter form appears as a simple average to the interface of the two fine cells / center of the corresponding coarse cell.

The simplest type of prolongation is simply *direct injection*:

$$\phi_i^f = \phi_j^c \quad (9.52)$$

$$\phi_{i+1}^f = \phi_j^c \quad (9.53)$$

A higher-order method is to do linear reconstruction of the coarse data and then average under the profile, e.g.,

$$\phi(x) = \frac{\phi_{j+1}^c - \phi_{j-1}^c}{2\Delta x^c} (x - x_j^c) + \phi_j^c \quad (9.54)$$

To second-order, we can find the values of ϕ_i^f and ϕ_{i+1}^f by evaluating $\phi(x)$ at the x -coordinate corresponding to their cell-centers,

$$x_i^f = x_j^c - \frac{\Delta x^c}{4} \quad (9.55)$$

$$x_{i+1}^f = x_j^c + \frac{\Delta x^c}{4} \quad (9.56)$$

giving

$$\phi_i^f = \phi_j^c - \frac{1}{8}(\phi_{j+1}^c - \phi_{j-1}^c) \quad (9.57)$$

$$\phi_{i+1}^f = \phi_j^c + \frac{1}{8}(\phi_{j+1}^c - \phi_{j-1}^c) \quad (9.58)$$

Notice that this is conservative, since $\Delta x^f(\phi_i^f + \phi_{i+1}^f) = \Delta x^c \phi_j^c$.

2-d

Restriction from the fine grid to the coarse grid is straightforward. Since the fine cells are perfectly enclosed by a single coarse cell, we simply average:

$$\phi_{i,j}^c = \frac{1}{4}(\phi_{--}^f + \phi_{+-}^f + \phi_{-+}^f + \phi_{++}^f) \quad (9.59)$$

Prolongation requires us to reconstruct the coarse data and use this reconstruction to determine what the fine cell values are. For instance, a linear reconstruction of the coarse data in x and y is:

$$\phi(x, y) = \frac{m_x}{\Delta x^c} (x - x_i^c) + \frac{m_y}{\Delta y^c} (y - y_j^c) + \phi_{i,j}^c \quad (9.60)$$

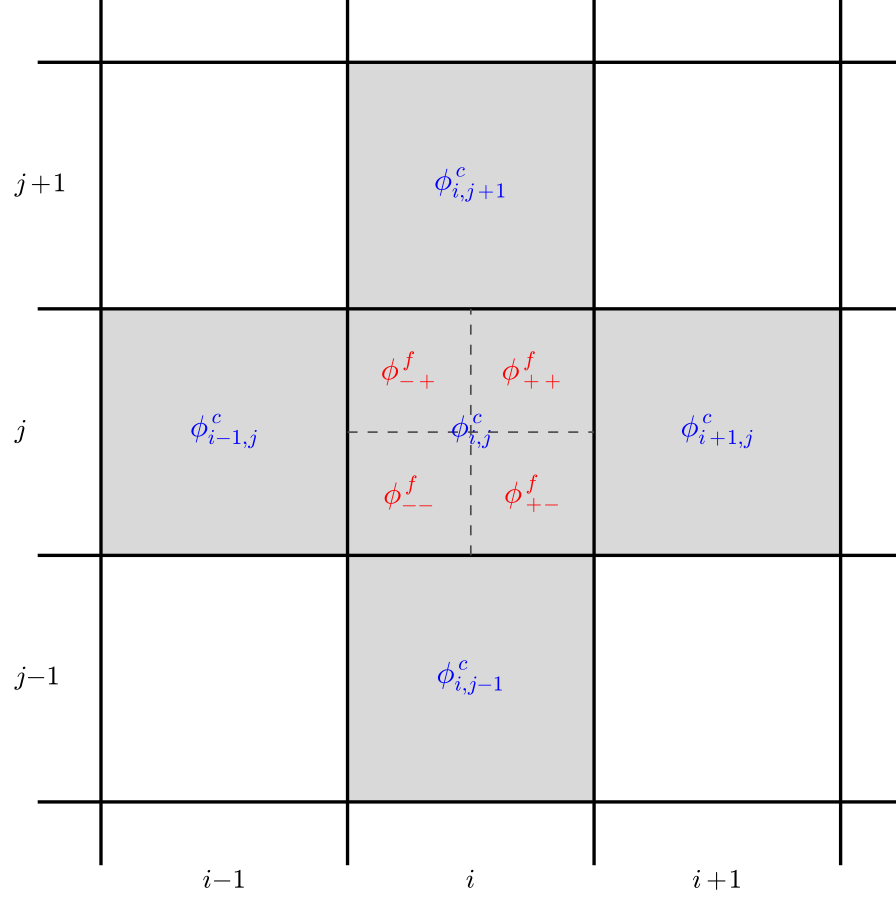


Figure 9.9: Four fine cells and the underlying coarse grid. For prolongation, the fine cells in red are initialized from a coarse parent. The gray coarse cells are used in the reconstruction of the coarse data. For restriction, the fine cells are averaged to the underlying coarse cell.

with slopes:

$$m_x = \frac{1}{2}(\phi_{i+1,j}^c - \phi_{i-1,j}^c) \quad (9.61)$$

$$m_y = \frac{1}{2}(\phi_{i,j+1}^c - \phi_{i,j-1}^c) \quad (9.62)$$

When averaged over the coarse cell, $\phi(x, y)$ recovers the average, $\phi_{i,j}^c$ in that cell (this means that our interpolant is conservative). We can evaluate the value in the fine cells by evaluating $\phi(x, y)$ at the center of the fine cells,

$$x_{\pm}^f = x_i^c \pm \frac{\Delta x^c}{4} \quad (9.63)$$

$$y_{\pm}^f = y_j^c \pm \frac{\Delta y^c}{4} \quad (9.64)$$

$$(9.65)$$

This gives

$$\phi_{\pm\pm}^f = \phi_{ij}^c \pm \frac{1}{4}m_x \pm \frac{1}{4}m_y \quad (9.66)$$

(Note: you would get the same expression if you averaged $\phi(x, y)$ over the fine cell.)

There are other options for prolongation and restriction, both of higher and lower order accuracy. However, the methods above seem to work well.

9.4.2 Multigrid cycles

The basic idea of multigrid** is to smooth a little on the current grid solving $L\phi = f$, compute the residual, r , then *restrict* r to a coarser grid and smooth on that grid solving $Le = r$, restrict again, Once you reach a sufficiently coarse grid, the problem solved exactly. Then the data is moved up to the finer grids, a process called *prolongation*. The error on the coarse grid, e , is prolonged to the finer grid. This error is then used to correct the solution on the finer grid, some smoothing is done, and then the data is prolonged up again.

Note: on the coarse grids, you are not solving the original system, but rather an error equation. If the boundary conditions in the original system are inhomogeneous, the boundary conditions for the error equations are now homogeneous. This must be understood by any ghost cell filling routines.

There are many different forms of the multigrid process. The simplest is called the *V-cycle*. Here you start of the fine grid, restrict down to the coarsest, solve, and then prolong back up to the finest. The flow looks like a 'V'. You continue with additional V-cycles until the residual error is smaller than your tolerance.

9.4.3 Bottom solver

Once the grid is sufficiently coarse, the linear system is small enough to be solved directly. This is the bottom solver operation. In the most ideal case, where the finest grid is some power of 2, $N_x = N_y = 2^n$, then the multigrid procedure can continue down until a 2×2 grid is created (Figure 9.10 illustrates this idea for a one-dimensional grid). This is the coarsest grid upon which one can still impose boundary conditions. With this small grid, just doing additional smoothing is sufficient enough to 'solve' the problem. No fancy bottom solver is needed.

For a general rectangular grid or one that is not a power of 2, the coarsest grid will likely be larger. For the general case, a linear system solver like conjugate gradient (or a variant) is used on the coarsest grid.

**In these discussions, we use *multigrid* to mean *geometric multigrid*, where the coarsening is done to the grid geometry directly. The alternative is *algebraic multigrid*, where it is the structure of the matrix in the linear system itself that is coarsened.

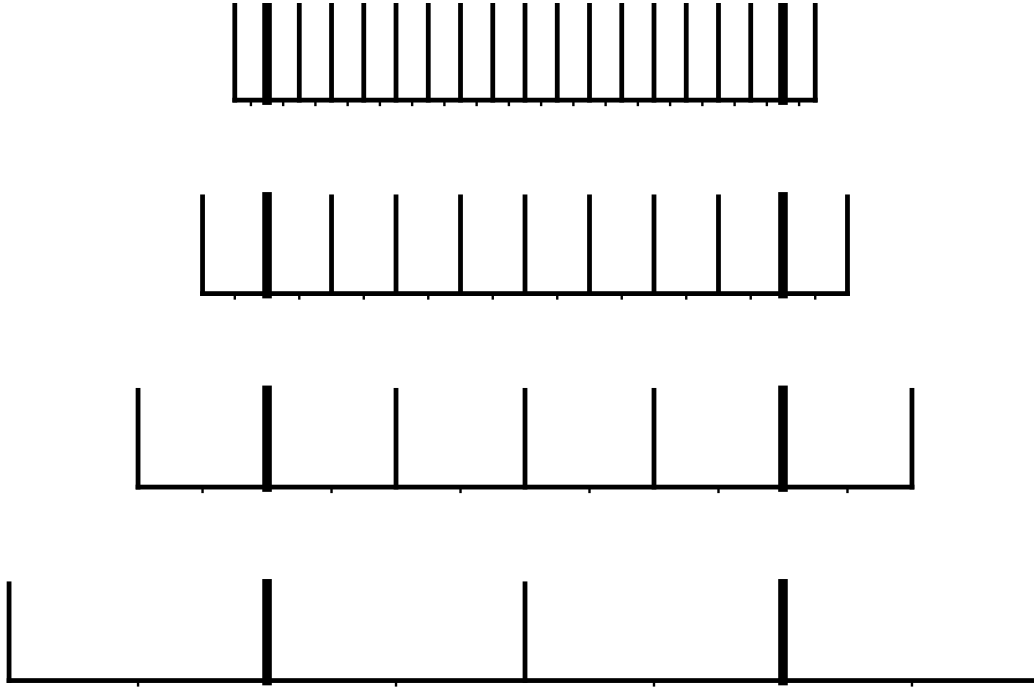


Figure 9.10: Illustration of the hierarchy of grids leading to the coarsest 2-zone grid (in one-dimension). Each grid has a single ghost cell to accommodate boundary conditions.

9.4.4 Boundary conditions throughout the hierarchy

The general inhomogeneous boundary conditions from Eqs. 9.31 and 9.35 apply to the finest level. But because we are solving the residual equation of the coarsest levels in the multigrid hierarchy, the boundary conditions on $Le = r$ are all homogeneous (but of the same type, Dirichlet, Neumann, or periodic, as the fine level).

Implementing these boundary conditions in your multigrid solver means that you will have separate actions for the fine level (where inhomogeneous boundaries may apply) and the coarser levels (where you will always be homogeneous).

An alternate way to enforce boundary conditions is via *boundary charges*. For inhomogeneous boundary conditions, boundary charges can be used to convert the BCs to homogeneous BCs. This has the advantage of allowing the ghost cell filling routines only deal with the homogeneous case.

Consider the one-dimensional Poisson equation, near the left boundary our discretized equation appears as:

$$\frac{\phi_{l_0-1} - 2\phi_{l_0} + \phi_{l_0+1}}{\Delta x^2} = f_{l_0} \quad (9.67)$$

Inhomogeneous BCs at the left boundary would give the condition:

$$\phi_{l0-1} = 2\phi_l - \phi_{l0} \quad (9.68)$$

Substituting this into the discrete equation, we have:

$$\frac{2\phi_l - \phi_{l0} - 2\phi_{l0} + \phi_{l0+1}}{\Delta x^2} = f_{l0} \quad (9.69)$$

Bringing the boundary condition value over to the RHS, we see

$$\frac{-3\phi_{l0} + \phi_{l0+1}}{\Delta x^2} = f_{l0} - \frac{2\phi_l}{\Delta x^2} \quad (9.70)$$

Now the left side looks precisely like the differenced Poisson equation with homogeneous Dirichlet BCs. The RHS has an additional ‘charge’ that captures the boundary value. By modifying the source term, f , in the multigrid solver to include this charge, we can use the homogeneous ghost cell filling routines throughout the multigrid algorithm. This technique is discussed a bit in [26].

Note that the form of the boundary charge will depend on the form of the elliptic equation—the expressions derived above apply only for $\nabla^2\phi = f$.

9.4.5 Stopping criteria

Repeated V-cycles are done until:

$$\|r\| < \epsilon \|f\| \quad (9.71)$$

on the finest grid, for some user-input tolerance, ϵ . Here, $\|f\|$ is called the *source norm*. If $\|f\| = 0$, then we stop when

$$\|r\| < \epsilon \quad (9.72)$$

Picking the tolerance ϵ is sometimes problem-dependent, and generally speaking, a problem with a large number of zones will require a looser tolerance.

The general rule-of-thumb is that each V-cycle should reduce your residual by about 1 order of magnitude. It is important that your bottom solver solves the coarse problem to a tolerance of 10^{-3} or 10^{-4} in order for the solver to converge. Figure 9.11 shows the true and residual errors for $\phi_{xx} = \sin(x)$ as a function of V-cycle number, illustrating the expected performance.

The overall convergence of the multigrid algorithm is limited by the discretization of the Laplacian operator used and the implementation of the boundary conditions. Figure 9.12 shows the error in the solution as the number of zones is increased—demonstrating second-order convergence for our implementation.

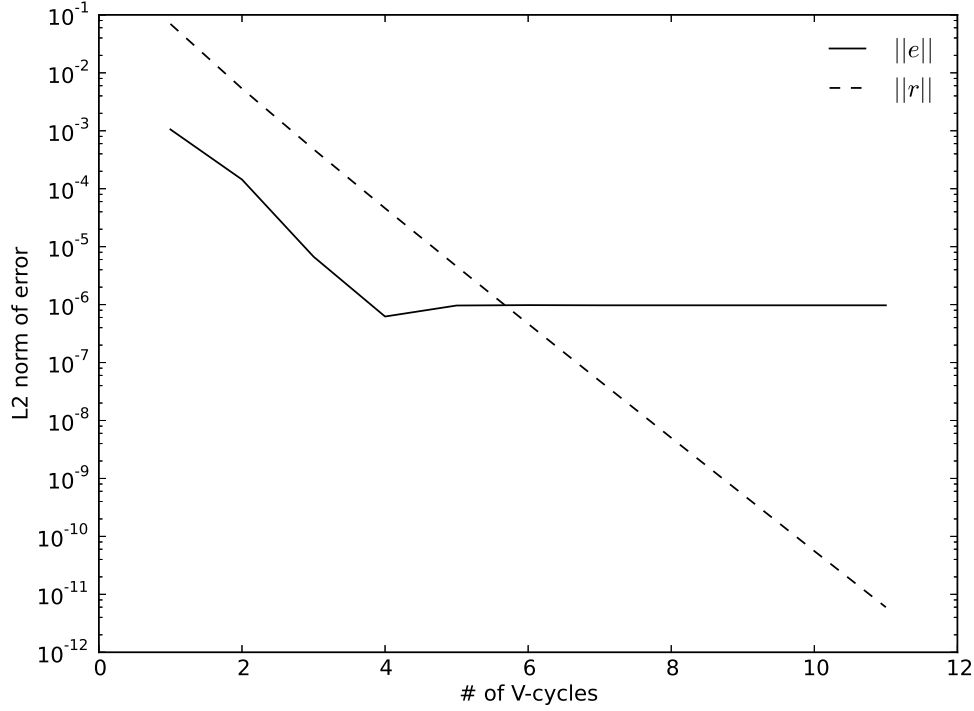


Figure 9.11: Error in the multigrid solution to our model problem ($\phi_{xx} = \sin(x)$) as a function of V-cycle. We see that the true error, $\|e\|$ stalls at truncation error while the residual error, $\|r\|$ reaches roundoff error, the same behavior as seen with smoothing alone (as expected).

 hydro_examples: mg_test.py

9.5 Solvability

For $\nabla^2 \phi = f$ with periodic or Neumann boundaries all around, the sum of f must equal 0 otherwise the solution will not converge. Instead, we will simply find the solution increase each V-cycle. This is seen as follows:

$$\int_{\Omega} f d\Omega = \int_{\Omega} \nabla^2 \phi d\Omega = \int_{\partial\Omega} \nabla \phi \cdot n dS = 0 \quad (9.73)$$

For all homogeneous Neumann boundaries, we have $\nabla \phi \cdot dS = 0$ by construction, so that integral is zero, requiring that the source integrate to zero. If the Neumann boundaries are inhomogeneous, there is still a solvability condition on f based on the sum on the boundary values.

A simple example of solvability is:

$$\phi'' = 0 \quad (9.74)$$

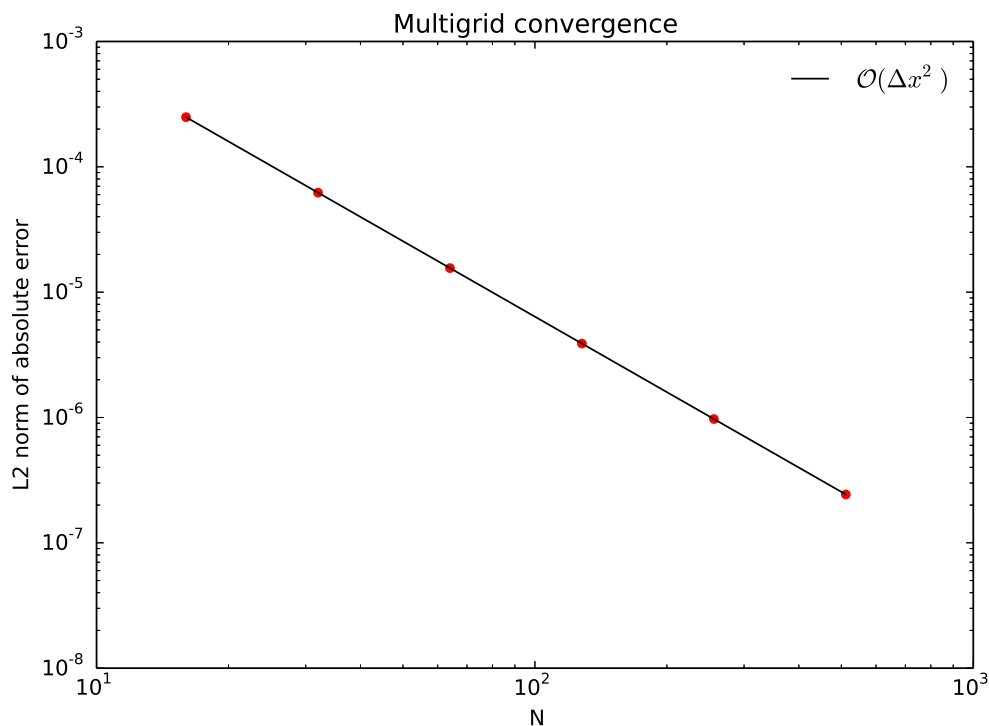


Figure 9.12: Convergence of the multigrid algorithm.

 hydro_examples: mg_converge.py

on $[a, b]$ with

$$\phi'(a) = A \quad (9.75)$$

$$\phi'(b) = B \quad (9.76)$$

We can integrate $\phi'' = 0$ to get $\phi(x) = \alpha x + \beta$ where α and β are integration constants. But note that this is just a straight line, with a single slope, α , so it is not possible to specify two unique slopes, A and B at the boundary unless there is a non-zero source term.

For all periodic boundaries, we have $\nabla\phi|_{\text{left}} = -\nabla\phi|_{\text{right}}$ on the left and right boundaries by definition of the periodicity (and similarly for the top and bottom). Again this implies that f must integrate to zero.

Sometimes, with periodic boundary conditions all around, you need to enforce that f integrate to zero numerically to test convergence. This is discussed in § 15.2.1.

9.6 Going Further

9.6.1 Red-black Ordering

When using domain decomposition to spread the problem across parallel processors, the smoothing is often done as *red-black Gauss-Seidel*. In this ordering, you imagine

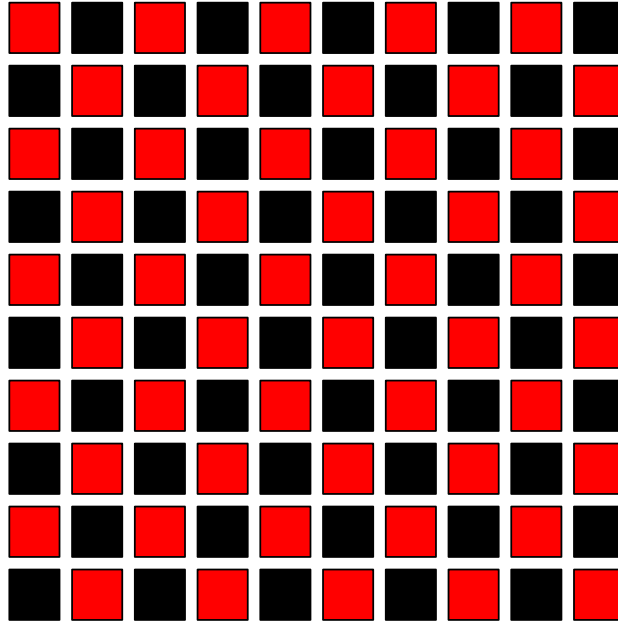


Figure 9.13: The red-black ordering of zones.

the grid to be a checkerboard (see Figure 9.13). In the first Gauss-Seidel pass you update the red squares and in the second, the black squares. The advantage is that when updating the red, you can be sure that none of the zones you depend on (the neighboring black zones) will change. This makes the decomposition parallel. Note: this works for the standard 5-point Laplacian. If you are doing some other operator with a different stencil, then this decomposition may no longer hold.

9.6.2 More General Elliptic Equations

The most general *second-order* elliptic equation takes the form:

$$\alpha\phi + \nabla \cdot (\beta \nabla \phi) + \gamma \cdot \nabla \phi + \nabla \cdot (\zeta \phi) = f \quad (9.77)$$

Here, γ and ζ are vectors. Solving a general elliptic equation of this form can be accomplished with multigrid using the same basic ideas here. The main change is that the smoothing algorithm and the construction of the residual will need to discretize the more general operator, and these coefficients will need to be restricted to the coarser grids (some on edges). This is explored in § 15.2 for a variable-coefficient Poisson equation:

$$\nabla \cdot (\beta \nabla \phi) = f \quad (9.78)$$