# Notes on multigrid

*These summarize multigrid on cell-centered grids. The text "A Multigrid Tutorial" [1] is an incredible reference. These notes discuss the basics and point out some specific details for cell-centered grids.*

## 1 Elliptic equations

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

$$\nabla^2 \phi = 0 \tag{1}$$

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

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

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*:

$$(\alpha - \nabla \cdot \beta \nabla)\phi = f \tag{3}$$

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 $\phi$ is specified instantaneously in the domain subject to boundary conditions.

## 2 Relaxation

Consider Poisson's equation differenced as:

$$\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2} = f_{i,j} \tag{4}$$

This is a 5-point stencil: for each zone $(i, j)$, we couple in the zones $\pm 1$ in $x$ and $\pm 1$ in $y$. This discretization uses the standard form for the second derivative, and is second-order accurate in space.

For the moment, consider the case where $\Delta x = \Delta y$. If we solve this discretized equation for $\phi_{i,j}$, then we have:

$$\phi_{i,j} = \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{5}$$

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

$$\phi_{i,j}^{(1)} = \frac{1}{4}(\phi_{i+1,j}^{(0)} + \phi_{i-1,j}^{(0)} + \phi_{i,j+1}^{(0)} + \phi_{i,j-1}^{(0)} - \Delta x^2 f_{i,j}) \tag{6}$$

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

$$\phi_{i,j}^{(k+1)} = \frac{1}{4}(\phi_{i+1,j}^{(k)} + \phi_{i-1,j}^{(k)} + \phi_{i,j+1}^{(k)} + \phi_{i,j-1}^{(k)} - \Delta x^2 f_{i,j}) \tag{7}$$

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 $\phi$—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:

$$\phi_{i,j} \leftarrow \frac{1}{4}(\phi_{i+1,j} + \phi_{i-1,j} + \phi_{i,j+1} + \phi_{i,j-1} - \Delta x^2 f_{i,j}) \tag{8}$$

This only requires a single copy of $\phi$ to be stored. This technique is called *Gauss-Seidel iteration*. A host of other relaxation methods exist, including linear combinations of these two. The text by Briggs is an excellent reference for this, and discusses the strengths of these different approaches.

Next consider the Helmholz equation with constant coefficients:

$$(\alpha - \beta\nabla^2)\phi = f \tag{9}$$

We can discretize this as:

$$\alpha\phi_{i,j} - \beta\left(\frac{\phi_{i+1,j} - 2\phi_{i,j} + \phi_{i-1,j}}{\Delta x^2} + \frac{\phi_{i,j+1} - 2\phi_{i,j} + \phi_{i,j-1}}{\Delta y^2}\right) = f_{i,j} \tag{10}$$

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

$$\phi_{i,j} \leftarrow \left(f_{i,j} + \frac{\beta}{\Delta x^2}\phi_{i+1,j} + \frac{\beta}{\Delta x^2}\phi_{i-1,j} + \frac{\beta}{\Delta y^2}\phi_{i,j+1} + \frac{\beta}{\Delta y^2}\phi_{i,j-1}\right) \Big/ \left(\alpha + \frac{2\beta}{\Delta x^2} + \frac{2\beta}{\Delta y^2}\right) \tag{11}$$

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

## 2.1 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. 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. For example, consider $\phi_{xx} = 0$. The solution to this is a line, $\phi = ax + b$. We can specify different Neumann boundary conditions on each end, $\phi_x|_{x=x_l} = p$, $\phi_x|_{x=x_r} = q$, because this specifies two incompatible slopes for our line.

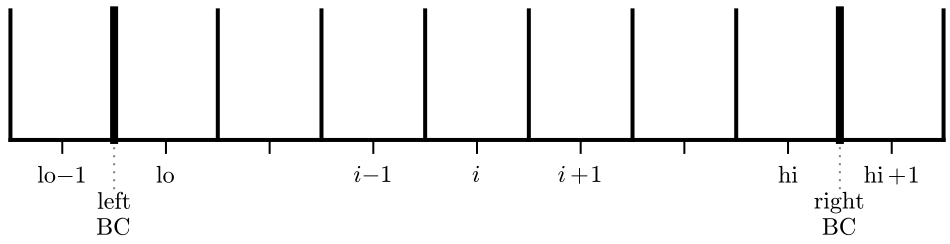


Figure 1: The cell-centered grid showing the cells (and ghost cells) surrounding the boundaries and indicating that the boundary conditions are actually specified right at the boundary itself.

Consider Dirichlet boundary conditions, specifying values $\phi_l$ on the left and $\phi_r$ on the right boundaries. To second order, we can define these via:

$$\phi_l = \frac{1}{2}(\phi_{\rm lo} + \phi_{\rm lo-1}) \tag{12}$$

$$\phi_r = \frac{1}{2}(\phi_{\rm hi} + \phi_{\rm hi+1}) \tag{13}$$

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

$$\phi_{\mathrm{lo}-1} = 2\phi_l - \phi_{\mathrm{lo}} \tag{14}$$
$$\phi_{\mathrm{hi}+1} = 2\phi_r - \phi_{\mathrm{hi}} \tag{15}$$

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:

$$\phi_x|_l = \frac{\phi_{\mathrm{lo}} - \phi_{\mathrm{lo}-1}}{\Delta x} \tag{16}$$
$$\phi_x|_r = \frac{\phi_{\mathrm{hi}+1} - \phi_{\mathrm{hi}}}{\Delta x} \tag{17}$$

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

$$\phi_{\mathrm{lo}-1} = \phi_{\mathrm{lo}} - \Delta x\, \phi_x|_l \tag{18}$$
$$\phi_{\mathrm{hi}+1} = \phi_{\mathrm{hi}} + \Delta x\, \phi_x|_r \tag{19}$$

## 2.2 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:

$$r_{i,j} = f_{i,j} - (L\phi)_{i,j} \tag{20}$$

and the residual error as:

$$\epsilon^{(r)} = \|r\| \tag{21}$$

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 $\phi^{\mathrm{true}}$ satisfies $\nabla^2\phi^{\mathrm{true}} = f$, then the true error in each zone is

$$e_{i,j} = \phi^{\mathrm{true}}(x_i, y_j) - \phi_{i,j} \tag{22}$$

and

$$\epsilon^{\mathrm{true}} = \|e_{i,j}\| \tag{23}$$

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

Note that since our operator is linear,

$$Le = L\phi^{\mathrm{true}} - L\phi = f - L\phi = r \tag{24}$$

so the error in our solution obeys a Poisson equation with the residual as the source.

## 2.3 Performance

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 (as the exercise shows), but many iterations are needed to remove the long wavelength noise (see Figure 2). Here the wavelength is in terms of the number of zones across the feature, and not a physical measure.

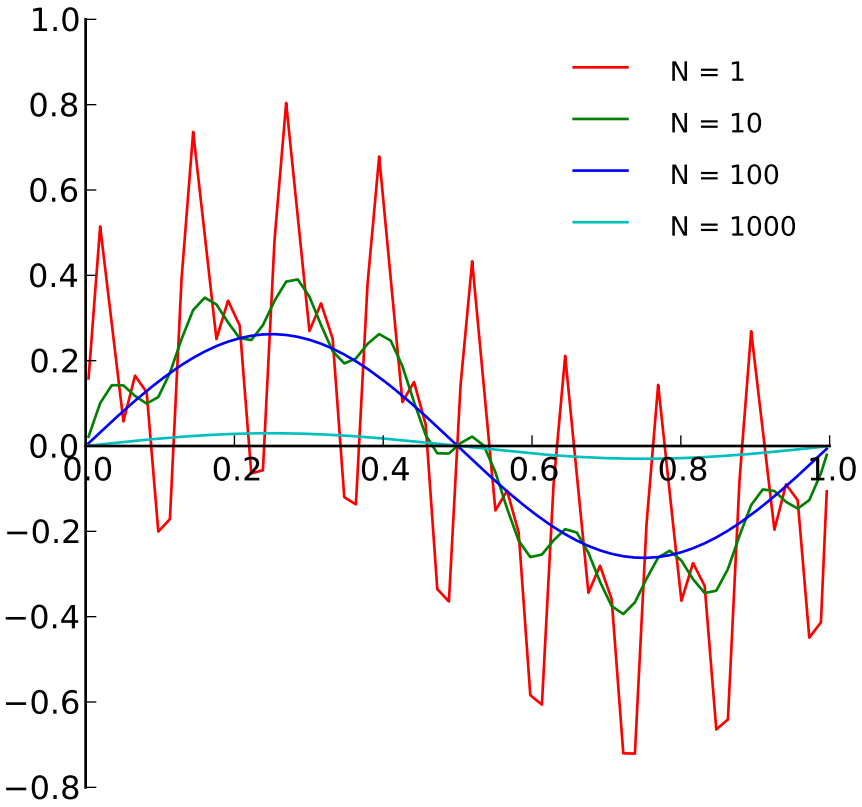


Figure 2: Error in the solution to $\phi'' = 0$ given an initial guess with 3 different wavenumbers of noise. The different curves are different numbers of smoothing iterations.

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

$$\phi_0(x) = \frac{1}{3}(\sin(2\pi x) + \sin(2\pi\, 8x) + \sin(2\pi\, 16x)) \tag{25}$$

*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 2.*

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 is the core idea behind multigrid.

## 3 Multigrid

The text *A Multigrid Tutorial* [1] provides an excellent introduction to the mechanics of multigrid. The basic idea 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.

### 3.1 Prolongation and restriction on cell-centered grids

Multigrid relies on transferring the problem up and down a hierarchy of grids. Consider the following grid. The finer grid is superposed over the center coarse cell, and the fine grid cells are marked in red.

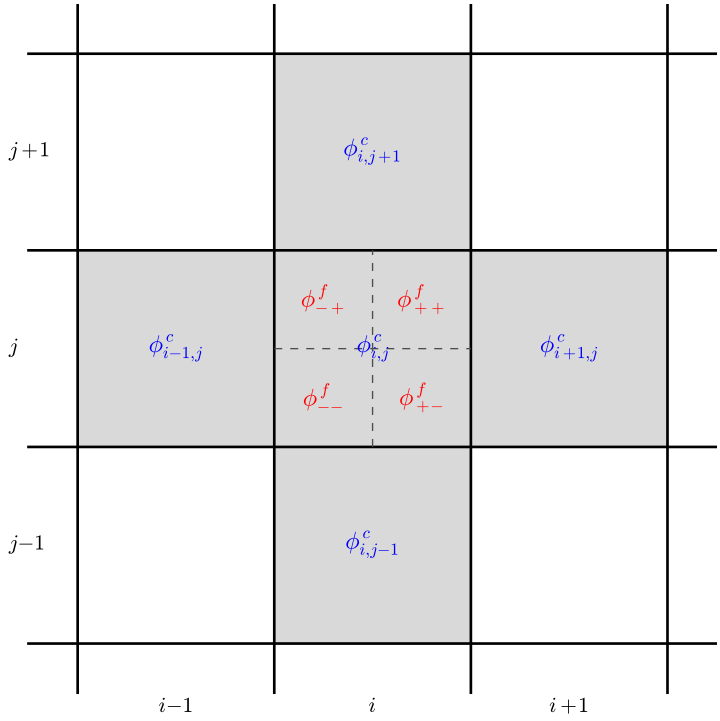


Figure 3: Four fine cells and the underlying coarse grid. For prolongation, the fine cells in red are initialized from a coarse parent. The gray coarse cells are used in the reconstruction of the coarse data. For restriction, the fine cells are averaged to the underlying coarse cell.

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:

$$\phi^c_{i,j} = \frac{1}{4}(\phi^f_{--} + \phi^f_{+-} + \phi^f_{-+} + \phi^f_{++}) \tag{26}$$

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:

$$\phi(x,y) = \frac{m_x}{\Delta x}(x - x^c_i) + \frac{m_y}{\Delta y}(y - y^c_j) + \phi^c_{i,j} \tag{27}$$

with slopes:

$$m_x = \frac{1}{2}(\phi^c_{i+1,j} - \phi^c_{i-1,j}) \tag{28}$$

$$m_y = \frac{1}{2}(\phi^c_{i,j+1} - \phi^c_{i,j-1}) \tag{29}$$

When averaged over the coarse cell, $\phi(x,y)$ recovers the average, $\phi^c_{i,j}$ 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,

$$x^f_\pm = x^c_i \pm \frac{\Delta x^c}{4} \tag{30}$$

$$y^f_\pm = y^c_j \pm \frac{\Delta y^c}{4} \tag{31}$$

$$\tag{32}$$

This gives

$$\phi^f_{\pm\pm} = \phi^c_{i,j} \pm \frac{1}{4} m_x \pm \frac{1}{4} m_y \tag{33}$$

(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.

### 3.2 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 $2 \times 2$ grid is created. 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.

### 3.3 Stopping criteria

Repeated V-cycles are done until:

$$\|r\| < \epsilon \|f\| \tag{34}$$

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

$$\|r\| < \epsilon \tag{35}$$

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.

## 4 Other notes

- *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 the grid to be a checkerboard. 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.

- *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:

$$\int_\Omega f d\Omega = \int_\Omega \nabla^2\phi d\Omega = \int_{\partial\Omega} \nabla\phi \cdot n dS = 0 \tag{36}$$

- *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:

$$\frac{\phi_{\mathrm{lo}-1} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{37}$$

  Inhomogeneous BCs at the left boundary would give the condition:

$$\phi_{\mathrm{lo}-1} = 2\phi_l - \phi_{\mathrm{lo}} \tag{38}$$

  Substituting this into the discrete equation, we have:

$$\frac{2\phi_l - \phi_{\mathrm{lo}} - 2\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} \tag{39}$$

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

$$\frac{-3\phi_{\mathrm{lo}} + \phi_{\mathrm{lo}+1}}{\Delta x^2} = f_{\mathrm{lo}} - \frac{2\phi_l}{\Delta x^2} \tag{40}$$

  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 [2].

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

$$\|e\|_\infty = \max\{|e_{i,j}|\} \tag{41}$$

  This will pick up on local errors.

  The $L_1$ norm and $L_2$ norms are more global.

$$\|e\|_1 = \frac{1}{N}\sum_{i,j} |e_{i,j}| \tag{42}$$

$$\|e\|_2 = \left(\frac{1}{N}\sum_{i,j} |e_{i,j}|^2\right)^{1/2} \tag{43}$$

  Generally, the measure in $L_2$ falls between $L_\infty$ and $L_1$. Regardless of the norm used, if the problem converges, it should converge in all norms.

  For reference, the BoxLib library uses $L_\infty$ in its multigrid solvers.

## acknowledgements

Thanks to Andy Nonaka and Ann Almgren for helpful discussions.

## References

[1] W. L. Briggs, V-E. Henson, and S. F. McCormick. *A Mutigrid Tutorial, 2nd Ed.* SIAM, 2000.

[2] P. Colella and E. G. Puckett. *Modern Numerical Methods for Fluid Flow*. unpublished manuscript. obtained from http://www.amath.unc.edu/Faculty/minion/class/puckett/.