Why Multigrid Methods Are So Efficient

Originally introduced as a way to numerically solve elliptic boundary-value problems, multigrid methods, and their various multiscale descendants, have since been developed and applied to various problems in many disciplines. This introductory article provides the basic concepts and methods of analysis and outlines some of the difficulties of developing efficient multigrid algorithms.

Multigrid computational methods are well known for being the fastest numerical methods for solving elliptic boundary-value problems. Over the past 30 years, multigrid methods have earned a reputation as an efficient and versatile approach for other types of computational problems as well, including other types of partial differential equations and systems and some integral equations. In addition, researchers have successfully developed generalizations of the ideas underlying multigrid methods for problems in various disciplines. (See the "Multigrid Methods Resources" sidebar for more details.)

This introductory article presents the fundamentals of multigrid methods, including explicit algorithms, and points out some of the main pitfalls using elementary model problems. This material is mostly intended for readers who have a practical interest in computational methods but little or no acquaintance with multigrid techniques. The article also provides some background for this special issue and other, more advanced publications.

1521-9615/06/$20.00 © 2006 IEEE
Copublished by the IEEE CS and the AIP

Irad Yavneh
Technion – Israel Institute of Technology

Basic Concepts

Let's begin with a simple example based on a problem studied in a different context.[1]

Global versus Local Processes

The hometown team has won the regional cup. The players are to receive medals, so it's up to the coach to line them up, equally spaced, along the goal line. Alas, the field is muddy. The coach, who abhors mud, tries to accomplish the task from afar. He begins by numbering the players 0 to N and orders players 0 and N to stand by the left and right goal posts, respectively, which are a distance L apart. Now, the coach could, for example, order player i, $i = 1, \ldots, N-1$, to move to the point on the goal line that is at a distance iL/N from the left goal post. This would be a *global process* that would solve the problem directly. However, it would require each player to recognize the left-hand goal post, perform some fancy arithmetic, and gauge long distances accurately. Suspecting that his players aren't up to the task, the coach reasons that if each player were to stand at the center point between his two neighbors (with players 0 and N fixed at the goal posts), his task would be accomplished. From the local rule, a global order will emerge, he philosophizes.

With this in mind, the coach devises the following simple iterative *local process*. At each iteration, he blows his whistle, and player i, $i = 1, \ldots, N-1$,

Multigrid Methods Resources

Radii Petrovich Fedorenko[1,2] and Nikolai Sergeevitch Bakhvalov[3] first conceived multigrid methods in the 1960s, but Achi Brandt first achieved their practical utility and efficiency for general problems in his pioneering work in the 1970s.[4–6] Classical texts on multigrid methods include an excellent collection of papers edited by Wolfgang Hackbusch and Ulrich Trottenberg,[7] Brandt's guide to multigrid methods,[8] and the classical book by Hackbusch.[9] Notable recent textbooks on multigrid include the introductory tutorial by William Briggs and his colleagues[10] and a comprehensive textbook by Trottenberg and his colleagues.[11] Many other excellent books and numerous papers are also available, mostly devoted to more specific aspects of multigrid techniques, both theoretical and practical.

References

1. R.P. Fedorenko, "A Relaxation Method for Solving Elliptic Difference Equations," *USSR Computational Mathematics and Mathematical Physics*, vol. 1, no. 4, 1962, pp. 1092–1096.
2. R.P. Fedorenko, "The Speed of Convergence of One Iterative Process," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 3, 1964, pp. 227–235.
3. N.S. Bakhvalov, "On the Convergence of a Relaxation Method with Natural Constraints on the Elliptic Operator," *USSR Computational Mathematics and Mathematical Physics*, vol. 6, no. 5, 1966, pp. 101–135.
4. A. Brandt, "A Multi-Level Adaptive Technique (MLAT) for Fast Numerical Solution to Boundary Value Problems," *Proc. 3rd Int'l Conf. Numerical Methods in Fluid Mechanics*, H. Cabannes and R. Temam, eds., Lecture Notes in Physics 18, Springer, 1973, pp. 82–89.
5. A. Brandt, *Multi-Level Adaptive Techniques (MLAT): The Multigrid Method*, research report RC 6026, IBM T.J. Watson Research Ctr., 1976.
6. A. Brandt, "Multi-Level Adaptive Solutions to Boundary-Value Problems," *Mathematics of Computation*, vol. 31, no. 138, 1977, pp. 333–390.
7. W. Hackbusch and U. Trottenberg, eds., *Multigrid Methods*, Springer-Verlag, 1982.
8. A. Brandt, *1984 Multigrid Guide with Applications to Fluid Dynamics*, GMD-Studie 85, GMD-FIT, Postfach 1240, D-5205, St. Augustin 1, West Germany, 1985.
9. W. Hackbusch, *Multi-Grid Methods and Applications*, Springer, 1985.
10. W.L. Briggs, V.E. Henson, and S.F. McCormick, *A Multigrid Tutorial*, 2nd ed., SIAM Press, 2000.
11. U. Trottenberg, C.W. Oosterlee, and A. Schüller, *Multigrid*, Academic Press, 2001.

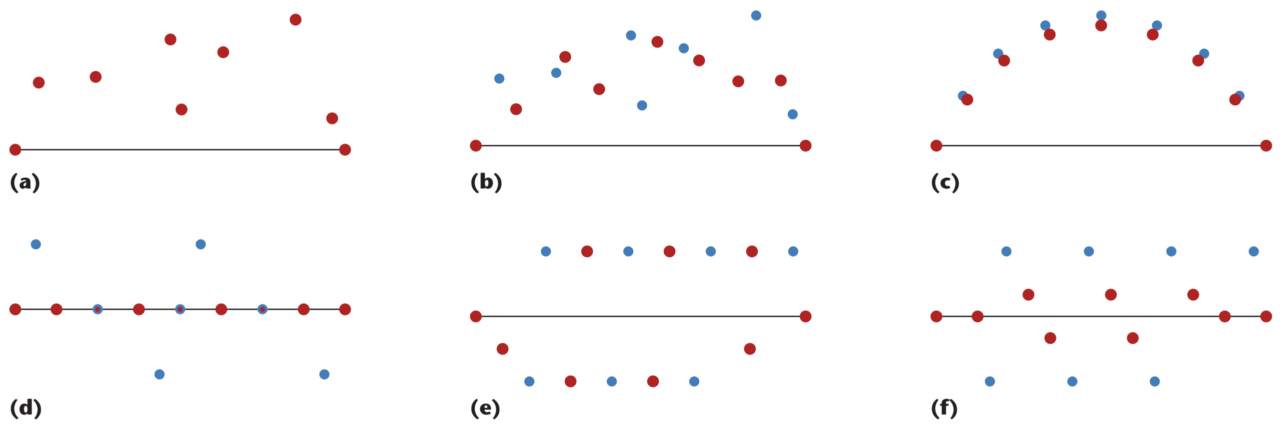

Figure 1. Jacobi's player-alignment iteration. Red disks show the current position, and blue disks show the previous position, before the last whistle blow: (a) initial position, (b) one whistle blow, (c) slow convergence, (d) fast convergence, (e) overshoot, and (f) damped iteration.

moves to the point that's at the center between the positions of player $i - 1$ and player $i + 1$ just before the whistle was blown. These iterations are repeated until the errors in the players' positions are so small that the mayor, who will be pinning the medals, won't notice. This is the *convergence criterion*. Undeniably, the coach deserves a name; we'll call him Mr. Jacobi.

Figure 1a shows the players' initial haphazard position, and Figure 1b the position after one iteration (whistle blow). (For clarity, we assume that the players aren't even on the goal line initially. This pessimistic view does not alter our analysis significantly. We can envision purely lateral movements by projecting the players' positions onto the goal line.) As I'll show, Jacobi's method is con-

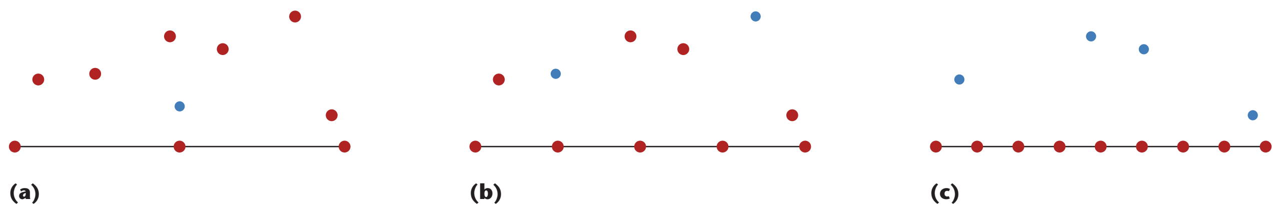

Figure 2. Multiscale player alignment. Red disks show the current position, and blue disks show the previous position, before the last whistle blow: (a) large, (b) medium, and (c) small scales.

vergent—that is, the errors in the players' positions will eventually be as small as we wish to make them.

However, this might take a long time. Suppose the players' positions are as shown in Figure 1c. In this case, the players move short distances at each iteration. Sadly, this slow crawl toward convergence will persist for the duration of the process. In contrast, convergence is obtained in a single iteration in the position shown in Figure 1d.

What is the essential property that makes the convergence slow in the position of Figure 1c, yet fast in Figure 1d? In Figure 1c, we have a large-scale error. But Jacobi's process is local, employing only small-scale information—that is, each player's new position is determined by his near neighborhood. Thus, to each player, it seems (from his myopic viewpoint) that the error in position is small, when in fact it isn't. The point is that we cannot correct what we cannot detect, and a large-scale error can't be detected locally. In contrast, Figure 1d shows a small-scale error position, which can be effectively detected and corrected by using the local process.

Finally, consider the position in Figure 1e. It clearly has a small-scale error, but Jacobi's process doesn't reduce it effectively; rather, it introduces an overshoot. Not every local process is suitable. However, this problem is easily overcome without compromising the process's local nature by introducing a constant *damping factor*. That is, each player moves only a fraction of the way toward his designated target at each iteration; in Figure 1f, for example, the players' movements are damped by a factor 2/3.

Multiscale Concept

The main idea behind multigrid methods is to use the simple local process but to do so at all scales of the problem. For convenience, assume N to be a power of two. In our simple example, the coach begins with a large-scale view in which the problem consists only of the players numbered 0, $N/2$, and N. A whistle blow puts player $N/2$ halfway between his "large-scale neighbors" (see Figure 2a). Next, addressing the intermediate scale, players $N/4$ and $3N/4$ are activated, and they move at the sound of the whistle to the midpoint between their "mid-scale neighbors" (see Figure 2b). Finally, the small scale is solved by Jacobi's iteration at the remaining locations (Figure 2c). Thus, in just three whistle blows—generally, $\log_2(N)$—and only $N - 1$ individual moves, the problem is solved, even though the simple local process has been used. Because the players can move simultaneously at each iteration—in parallel, so to speak—the number of whistle blows is important in determining the overall time, and $\log_2(N)$ is encouraging. For 1 million players, for example, Jacobi's process would require a mere 20 whistle blows.

Researchers have successfully exploited the idea of applying a local process at different scales in many fields and applications. Our simple scenario displays the concept but hides the difficulties. In general, developing a multiscale solver for a given problem involves four main tasks:

- choosing an appropriate local process,
- choosing appropriate coarse (large-scale) variables,
- choosing appropriate methods for transferring information across scales, and
- developing appropriate equations (or processes) for the coarse variables.

Depending on the application, each of these tasks can be simple, moderately difficult, or extremely challenging. This is partly the reason why multigrid continues to be an active and thriving research field. We can examine these four tasks for developing a multiscale solver by using a slightly less trivial problem. Before doing this, let's analyze Jacobi's player-alignment algorithm and give some mathematical substance to our intuitive notions on the dependence of its efficiency on the scale of the error.

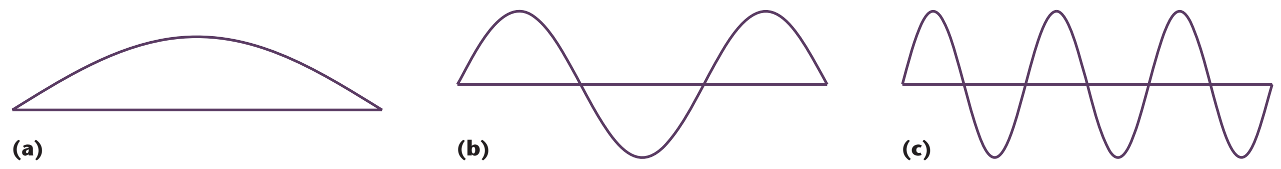

Figure 3. Three eigenvectors. We associate scale with the wavelength, which is inversely proportional to k: (a) $v^{(1)}$, (b) $v^{(3)}$, and (c) $v^{(6)}$.

Analyzing Jacobi's Player-Alignment Algorithm

First, observe that Jacobi's iteration acts similarly and independently in the directions parallel and perpendicular to the goal line. Therefore, it suffices to consider the convergence with respect to just one of these directions. Accordingly, let the column vector $e^{(0)} = [e_1^{(0)}, \ldots, e_{N-1}^{(0)}]^T$ denote the players' initial errors, either in the direction perpendicular or parallel to the goal line. We can define the error as the player's present coordinate, subtracted from the correct coordinate (to which he should eventually converge.) For convenience, we don't include players 0 and N in this vector, but we keep in mind that $e_0^{(0)} = e_N^{(0)} = 0$. For $n = 1, 2, \ldots$, the new error in the location of player i after iteration n is given by the average of the errors of his two neighbors just before the iteration—namely,

$$e_i^{(n)} = \frac{1}{2}\left(e_{i+1}^{(n-1)} + e_{i-1}^{(n-1)}\right). \tag{1}$$

We can write this in vector-matrix form as

$$e^{(n)} = \boldsymbol{R}e^{(n-1)}, \tag{2}$$

where $\boldsymbol{R}$ is an $N-1$ by $N-1$ tridiagonal matrix with one-half on the upper and lower diagonals and 0 on the main diagonal. For example, for $N = 6$, we have

$$\boldsymbol{R} = \frac{1}{2}\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}. \tag{3}$$

The behavior of the iteration and its relation to scale is most clearly observed for an error, e, which is an eigenvector of $\boldsymbol{R}$, satisfying $\boldsymbol{R}e = \lambda e$, for some (scalar) eigenvalue, λ. The matrix $\boldsymbol{R}$ has $N-1$ eigenvalues; for $k = 1, \ldots, N-1$, they are given by

$$\lambda^{(k)} = \cos\left(\frac{k\pi}{N}\right). \tag{4}$$

The $N-1$ corresponding eigenvectors are given by

$$v^{(k)} = \begin{bmatrix} \sin\left(\frac{k\pi}{N}\right) \\ \sin\left(\frac{2k\pi}{N}\right) \\ \vdots \\ \sin\left(\frac{(N-1)k\pi}{N}\right) \end{bmatrix}. \tag{5}$$

(To verify this, use the trigonometric identity,

$$\sin\left(\frac{k(i+1)\pi}{N}\right) + \sin\left(\frac{k(i-1)\pi}{N}\right) = 2\cos\left(\frac{k\pi}{N}\right)\sin\left(\frac{ki\pi}{N}\right)$$

to show that $\boldsymbol{R}v^{(k)} = \lambda^{(k)}v^{(k)}$.) Figure 3 shows the first, third, and sixth eigenvectors. We associate scale with the wavelength, which is inversely proportional to k. Thus, small k corresponds to large-scale eigenvectors, and vice versa.

The eigenvectors $v^{(k)}$ are linearly independent, spanning the space $\boldsymbol{R}^{N-1}$, so we can write any initial error as a linear combination of the eigenvectors,

$$e^{(0)} = \sum_{k=1}^{N-1} c_k v^{(k)}, \tag{6}$$

for some real coefficients, c_k. Repeated multiplication by the iteration matrix, $\boldsymbol{R}$, yields the error after n iterations:

$$\begin{aligned} e^{(n)} &= \boldsymbol{R}^n e^{(0)} = \boldsymbol{R}^n \sum_{k=1}^{N-1} c_k v^{(k)} \\ &= \sum_{k=1}^{N-1} c_k \boldsymbol{R}^n v^{(k)} = \sum_{k=1}^{N-1} c_k (\lambda^{(k)})^n v^{(k)} \\ &= \sum_{k=1}^{N-1} c_k \left[\cos\left(\frac{k\pi}{N}\right)\right]^n v^{(k)}. \end{aligned} \tag{7}$$

Because $|\lambda^{(k)}| = |\cos(k\pi/N)| < 1$ for $1 \le k \le N-1$,

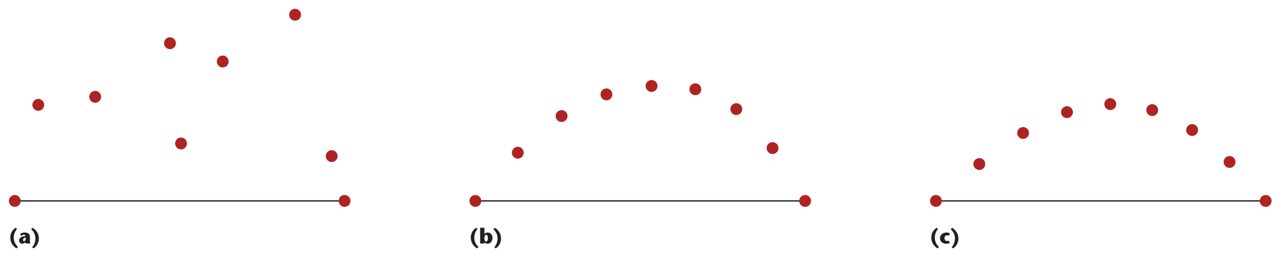

Figure 4. The smoothing property of the damped Jacobi relaxation. (a) Initial position, (b) after four relaxations, and (c) after eight relaxations.

we conclude that Jacobi's method converges because $|\lambda^{(k)}|^n$ tends to zero as n tends to infinity. It's illuminating to examine the rate of convergence for different values of k. For $k/N << 1$, we have

$$\left(\lambda^{(k)}\right)^n = \left[\cos\left(\frac{k\pi}{N}\right)\right]^n \approx e^{-\frac{n}{2}\left(\frac{k\pi}{N}\right)^2}, \tag{8}$$

where we can verify the approximation by using a Taylor series expansion. We find that the coefficients of eigenvectors corresponding to small k values change only slightly in each iteration (see Figure 1c). Indeed, we see here that $n = O(N^2)$ iterations are required to reduce such an error by an order of magnitude. For 1 million players, for example, hundreds of billions of whistle blows are required, compared to just 20 using the multiscale approach.

For $k = N/2$, $\lambda^{(k)} = \cos(k\pi/N) = 0$, and convergence is immediate (see Figure 1d). As k approaches N, the scale of the error becomes even smaller, and $\cos(k\pi/N)$ approaches −1, which is consistent with the overshoot we saw in Figure 1e—that is, at each iteration, the error mainly changes sign, with hardly any amplitude change.

Damped Jacobi Iteration's Smoothing Factor

To overcome the overshoot problem, we introduce a damping factor. If we damp the players' movements by a factor 2/3, we obtain

$$e_i^{(n)} = \frac{1}{3}e_i^{(n-1)} + \frac{2}{3}\left[\frac{1}{2}\left(e_{i+1}^{(n-1)} + e_{i-1}^{(n-1)}\right)\right], \tag{9}$$

and in vector-matrix form,

$$e^{(n)} = \boldsymbol{R}_d e^{(n-1)} \equiv \left(\frac{1}{3}I + \frac{2}{3}\boldsymbol{R}\right)e^{(n-1)}, \tag{10}$$

with $\boldsymbol{R}_d$ denoting the damped Jacobi iteration matrix and $\boldsymbol{I}$ the identity matrix. The eigenvectors clearly remain the same, but the eigenvalues of $\boldsymbol{R}_d$ are given by

$$\lambda_d^{(k)} = \frac{1}{3} + \frac{2}{3}\lambda^{(k)} = \frac{1}{3} + \frac{2}{3}\cos\left(\frac{k\pi}{N}\right). \tag{11}$$

We can now quantify the claim that Jacobi's iteration, damped by a factor 2/3, is efficient in reducing small-scale errors. To do this, let's (somewhat arbitrarily at this point) split the spectrum down the middle, roughly, and declare all eigenvectors with $1 \le k < N/2$ *smooth* (large scale), and eigenvectors with $N/2 \le k < N$ *rough* (small scale). Observe that, as we vary k from $N/2$ to $N - 1$, spanning the rough eigenvectors, $\cos(k\pi/N)$ varies from 0 to (nearly) −1, and therefore $\lambda_d^{(k)}$ varies from 1/3 to (nearly) −1/3. The fact that $|\lambda_d^{(k)}| \le 1/3$ for all the rough eigenvectors implies that the size of the coefficient of each rough eigenvector is reduced in each iteration to at most 1/3 its size prior to the iteration independently of N. We say that the iteration's *smoothing factor* is 1/3. If we begin with an initial error that's some haphazard combination of the eigenvectors, then after just a few iterations, the coefficients of all the rough eigenvectors will be greatly reduced. Thus, the damped Jacobi iteration (usually called *relaxation*) is efficient at smoothing the error, although, as we've seen, it is inefficient at reducing the error once it has been smoothed. Figure 4 exhibits these properties. Relaxation, one observes, irons out the wrinkles but leaves the fat.

1D Model Problem

The idea of using a local rule to obtain a global order is, of course, ancient. Perhaps the most common example in mathematics is differential equations, which have a global solution determined by a local rule on the derivatives (plus boundary conditions). To advance our discussion, we require an example that's slightly less trivial than player alignment. Consider the 1D boundary-value problem:

$$Lu \equiv \frac{d^2u}{dx^2} = f(x), \quad x \text{ in } (0,1),$$

$$u(0) = u_\ell, \quad u(1) = u_r, \qquad (12)$$

where f is a given function, u_ℓ and u_r are the boundary conditions, and u is the solution we're seeking. We adopt the common numerical approach of finite difference discretization. A uniform mesh is defined on the interval [0, 1], with mesh size $h = 1/N$ and $N + 1$ mesh points given by $x_i + ih$ for $i = 0, 1, ..., N$.

Our vector of unknowns—the discrete approximation to u—is denoted as $u^h = [u_1^h, ...u_{N-1}^h]^T$, where we exclude the boundary values for convenience but keep in mind that $u_0^h = u_\ell$, $u_N^h = u_r$. The right-hand side vector is denoted by $f^h = [f_1^h, ..., f_{N-1}^h]^T$, with $f_i^h = f(x_i)$. Alternatively, we could use some local averaging of f. The standard second-order accurate finite difference discretization of the second derivative now yields a system of equations for u^h, which we can write as

$$\begin{aligned}(L^h u^h)_i &\equiv \frac{u_{i-1}^h - 2u_i^h + u_{i+1}^h}{h^2} \\ &= f_i^h, \quad i = 1, ..., N-1,\end{aligned}$$

$$u_0^h = u_\ell, \quad u_N^h = u_r. \qquad (13)$$

We can also write this system more concisely as $L^h u^h = f^h$. The system is tridiagonal and easy to solve by standard methods, but it's useful for examining the multigrid solver's aforementioned tasks. Observe the similarity to the player-alignment problem: if we set $f \equiv 0$, the solution u is obviously a straight line connecting the point $(0, u_\ell)$ with $(1, u_r)$. The discrete solution is then a set of values along this line, each equal to the average of its two neighbors.

We can easily adapt Jacobi's algorithm to Equation 13. Let $u^{h(0)}$ denote an initial approximation to u^h that satisfies the boundary conditions. Then, at the nth iteration, we set $u_i^{h(n)}$, $i = 1, ..., N - 1$, to satisfy the ith equation in the system in Equation 13, with the neighboring values taken from the $n - 1$st iteration:

$$\begin{aligned}u_i^{h(n)} &= \frac{1}{2}\left(u_{i+1}^{h(n-1)} + u_{i-1}^{h(n-1)} - h^2 f_i^h\right), \\ i &= 1, ..., N-1.\end{aligned} \qquad (14)$$

The convergence behavior of Jacobi's algorithm is identical to its behavior in the player-alignment problem. Denote the error in our approximation after the nth iteration by

$$e^{h(n)} = u^h - u^{h(n)}, \qquad (15)$$

with $e_0^{h(n)} = e_N^{h(n)} = 0$ because all our approximations satisfy the boundary conditions. Now, subtract Equation 14 from the equation

$$u_i^h = \frac{1}{2}\left(u_{i+1}^h + u_{i-1}^h - h^2 f_i^h\right),$$

which we get by isolating u_i^h in Equation 13. This yields

$$e_i^{h(n)} = \frac{1}{2}\left(e_{i+1}^{h(n-1)} + e_{i-1}^{h(n-1)}\right), \qquad (16)$$

which is exactly like Equation 1. Similarly, damped Jacobi relaxation, with a damping factor of 2/3, is defined by

$$u_i^{h(n)} = \frac{1}{3}u_i^{h(n)} + \frac{2}{3}\left[\frac{1}{2}\begin{pmatrix}u_{i+1}^{h(n-1)} + u_{i-1}^{h(n-1)} \\ -h^2 f_i^h\end{pmatrix}\right], \qquad (17)$$

and its error propagation properties are identical to those of the player-alignment problem in Equations 9 through 11.

Naïve Multiscale Algorithm

We next naïvely extend the multiscale algorithm of the player-alignment problem to the 1D model problem. The main point of this exercise is to show that in order to know what to do at the large scales, we must first propagate appropriate information from the small scales to the large scales.

For convenience, we still assume N is a power of two. Similarly to the player-alignment process, we begin by performing Jacobi's iteration for the problem defined at the largest scale, which consists of the variables $[u_0^h, u_{N/2}^h, u_N^h]$. The mesh size at this scale is 0.5, and u_0^h and u_N^h are prescribed by the boundary conditions. The Jacobi step thus fixes $u_{N/2}^h$:

$$u_{N/2}^h = \frac{1}{2}\left(u_0^h + u_N^h - 0.25 f_{N/2}^h\right).$$

Next, based on the values of the largest scale, we perform a Jacobi iteration at the second-largest scale, obtaining $u_{N/4}^h$ and $u_{3N/4}^h$. We continue until we have determined all variables, just as in the player-alignment problem.

This algorithm is obviously computationally efficient. Unfortunately, it fails to give the correct answer (unless f is trivial) because we aren't solving the right equations on the coarse grids. More precisely, the right-hand sides are wrong.

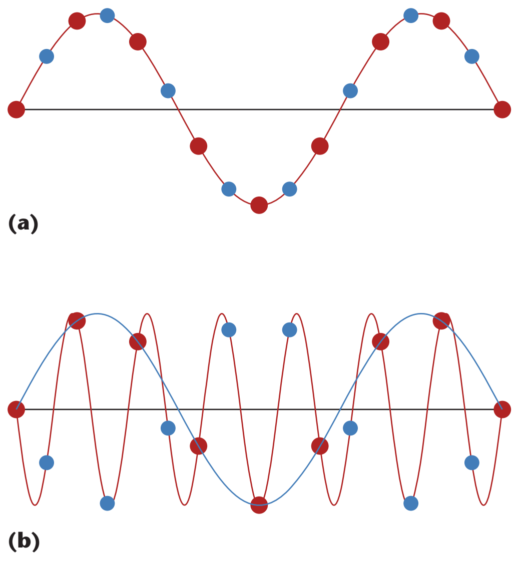

Figure 5. Aliasing. Red disks correspond to the coarse grid: (a) $v^{(3)}$ and (b) $-v^{(13)}$.

Aliasing

Figure 5a shows a smooth eigenvector ($k = 3$) sampled on a grid with $N = 16$ and also on a twice-coarser grid, $N = 8$, consisting of the fine grid's even-indexed mesh points. This repeats in Figure 5b, but for a rough eigenvector ($k = 13$ on the finer grid), with coefficient –1. Evidently, the two are indistinguishable on the coarse grid, which demonstrates the well-known phenomenon of aliasing. The coarse grid affords roughly half as many eigenvectors as the fine grid, so some pairs of fine-grid eigenvectors must alias. Specifically, by Equation 5, the even-indexed terms in eigenvectors $v^{(k)}$ and $v^{(N-k)}$ are equal to the negative of each other. Thus, if f^h contains some roughness—that is, if we write f^h as a linear combination of eigenvectors, it will have rough eigenvectors with nonzero coefficients—it won't be represented correctly on the coarse grid. As a somewhat extreme example, if f^h is a linear combination of $v^{(k)}$ and $v^{(N-k)}$ for some k, with weights of equal magnitude but opposite sign, it will vanish when sampled on the coarser grid.

The definition of the smoothing factor, which was based on a partitioning of the spectrum into smooth and rough parts, foreshadowed our use of a twice-coarser grid. Smooth eigenvectors are those that can be represented accurately on the twice-coarser grid, whereas rough eigenvectors alias with smooth ones.

We can conclude that constructing correct equations for coarse grids requires information from the fine grid. For the 1D model problem, we can get around the aliasing problem by successive local averaging. However, in more general problems, we can't do this without spending a substantial computational effort, which would defeat the purpose of the multiscale approach. We must come to grips with the fact that coarse grids are useful only for representing smooth functions. This means that we must take care in choosing our coarse-grid variables.

Taxonomy of Errors

We've seen that damped Jacobi relaxation smoothes the error. In this context, it is important to distinguish between two distinct types of error. The *discretization error*, given at mesh point i by $u(x_i) - u_i^h$, is the difference between the solution of the differential equation (sampled on the grid) and that of the discrete system. This error is determined by the problem, the discretization, and the mesh size. Iteration has no effect on this error, of course, because it only affects our approximation to u^h. Only the *algebraic error*, which is the difference between the discrete solution and our current approximation to it (Equation 15), is smoothed. Our plan, therefore, is to first apply damped Jacobi relaxation, which smoothes the algebraic error, and then construct a linear system for this error vector and approximate it on a coarser grid. This can continue recursively, such that, on each coarser grid, we solve approximate equations for the next-finer grid's algebraic error.

The motivation for using the algebraic error as our coarse-grid variable is twofold. Because we're limited in our ability to compute an accurate approximation to u^h on the coarse grid due to aliasing, we need to resort to iterative improvement. We should therefore use the coarse grid to approximately correct the fine-grid algebraic error, rather than to dictate the fine-grid solution. Moreover, relaxation has smoothed the algebraic error, and thus we can approximate it accurately on the coarse grid.

Let $\tilde{u}^h$ denote our current approximation to u^h after performing several damped Jacobi relaxations, and let $e^h = u^h - \tilde{u}^h$ denote the corresponding algebraic error, which is known to be smooth (see Figure 4). Subtract from Equation 13 the trivial relation

$$\left(L^h \tilde{u}^h\right)_i = \left(L^h \tilde{u}^h\right)_i, \quad i = 1, \ldots, N-1,$$

$$\tilde{u}_0^h = u_\ell, \quad \tilde{u}_N^h = u_r,$$

to obtain a system for the algebraic error,

$$\left(L^h e^h\right)_i = f_i^h - \left(L^h \tilde{u}^h\right)_i \equiv r_i^h, \quad i = 1,\dots N-1,$$

$$e_0^h = 0, \quad e_N^h = 0,$$

where r_i^h is the ith *residual*. This is the problem we must approximate on the coarser grid. We can write it concisely as $L^h e^h = r^h$. Once we compute a good approximation to e^h, we then add it to $\tilde{u}^h$, and thus improve our approximation to u^h.

Multigrid Algorithm

To transfer information between grids, we need suitable operators. Consider just two grids: the fine grid, with mesh size h and with $N-1$ mesh points excluding boundary points, and the coarse grid, with mesh size $2h$ and $N/2-1$ mesh points. We use a *restriction operator*, denoted I_h^{2h}, for fine-to-coarse data transfer. Common choices are *injection*, defined for a fine-grid vector, g^h, by

$$\left(I_h^{2h} g^h\right)_i = g_{2i}^h, \quad i = 1,\dots,\frac{N}{2}-1,$$

that is, the coarse-grid value is simply given by the fine-grid value at the corresponding location, or *full weighting*, which is defined by

$$\left(I_h^{2h} g^h\right)_i = \frac{1}{4}\left(g_{2i-1}^h + 2g_{2i}^h + 2_{2i+1}^h\right),$$
$$i = 1,\dots,\frac{N}{2}-1,$$

where we apply a local averaging. For coarse-to-fine data transfer, we use a prolongation operator, denoted by I_{2h}^h. A common choice is the usual linear interpolation, defined for a coarse-grid vector g^{2h} by

$$\left(I_{2h}^h g^{2h}\right)_i = \begin{cases} g_{i/2}^{2h}, & \text{if } i \text{ is even,} \\ \frac{1}{2}\left(g_{(i+1)/2}^{2h} + g_{(i-1)/2}^{2h}\right), & \text{if } i \text{ is odd,} \end{cases}$$
$$i = 1,\dots,N-1.$$

Additionally, we must define an operator on the coarse grid, denoted L^{2h}, that approximates the fine grid's L^h. A common choice is to simply discretize the differential equation on the coarse grid using a scheme similar to the one applied on the fine grid. Armed with these operators, and using recursion to solve the coarse-grid problem approximately, we can now define the multigrid *V-Cycle* (see Figure 6). We require two parameters, commonly denoted by ν_1 and ν_2, that designate, respectively, the number of damped Jacobi relaxations performed before and after we apply the coarse-grid correction. The multigrid algorithm is defined as

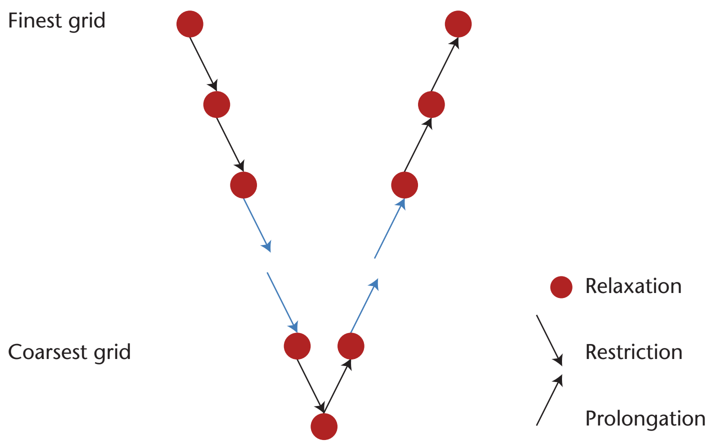

Figure 6. A schematic description of the V-Cycle. The algorithm proceeds from left to right and top *(*finest grid*)* to bottom *(*coarsest grid*)* and back up again. On each grid but the coarsest, we relax ν_1 times before transferring *(*restricting*)* to the next-coarser grid and ν_2 times after interpolating and adding the correction.

```
Algorithm V-Cycle: $u^h$ = VCycle($u^h$, $h$, $N$, $f^h$, $\nu_1$, $\nu_2$)
  1. If $N == 2$, solve $L^h u^h = f^h$ and return $u^h$;
  2. $u^h$ = Relax($u^h$, $h$, $N$, $f^h$, $\nu_1$);
     relax $\nu_1$ times.
  3. $r^h = f^h - L^h u^h$;
     compute residual vector.
  4. $f^{2h} = I_h^{2h} r^h$;
     restrict residual.
  5. $u^{2h} = 0$;
     set $u^{2h}$ to zero, including boundary conditions.
  6. $u^{2h}$ = VCycle($u^{2h}$, $2h$, $N/2$, $f^{2h}$, $\nu_1$, $\nu_2$);
     treat coarse-grid problem recursively.
  7. $u^h = u^h + I_{2h}^h u^{2h}$;
     interpolate and add correction.
  8. $u^h$ = Relax($u^h$, $h$, $N$, $f^h$, $\nu_2$);
     relax $\nu_2$ times;
  9. Return $u^h$.
```

Let's examine the algorithm line by line. In line 1, we check whether we've reached the coarsest grid, $N = 2$. If so, we solve the problem. Because there's only a single variable there, one Jacobi iteration without damping does the job. In line 2, ν_1 damped Jacobi iterations are applied. We compute the residual in line 3, and restrict it to the coarse grid in line 4. We now need to tackle the coarse-grid problem and obtain the coarse-grid approximation to the fine-grid error. Noting that the coarse-grid problem is similar to the fine-grid one, we apply a recursive call in line 6. Note that the boundary conditions on the coarse grid are zero (line 5) because there's no error in the fine-grid boundary values, which are prescribed. Also, the initial approximation to u^{2h} must be zero. After returning from the recursion, we interpolate and add the correction to the fine-grid approximation (line 7). In line 8, we relax the fine-grid equations ν_2 times, and the resulting approximation to the solution is returned in line 9.

Generally, the V-Cycle doesn't solve the problem exactly, and it needs to be applied iteratively. However, as I'll demonstrate later, each such iteration typically reduces the algebraic error dramatically.

2D Model Problem

The multiscale algorithm isn't restricted to one dimension, of course. Consider the 2D boundary-value problem:

$$Lu \equiv \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = f(x, y), \quad (x, y) \text{ in } \Omega,$$

$$u = b(x, y), \quad (x, y) \text{ on } \partial\Omega, \tag{18}$$

where f and b are given functions and u is the solution we're seeking. For simplicity, our domain is the unit square, $\Omega = (0, 1)^2$, with $\partial\Omega$ denoting the boundary of Ω. This is the well-known Poisson problem with Dirichlet boundary conditions. To discretize the problem, we define a uniform grid with mesh size h and mesh points given by

$$(x_i, y_j) = (ih, jh), \quad i, j = 0, 1, \ldots, N,$$

where $h = 1/N$. Our vector of unknowns—the discrete approximation to u—is again denoted u^h, b^h denotes the discrete boundary conditions and f^h is the discrete right-hand side function. The standard second-order-accurate finite difference discretization now yields a system of equations for u^h, which we can write as

$$(L^h u^h)_{i,j} \equiv \frac{\begin{array}{c} u^h_{i-1,j} + u^h_{i+1,j} + u^h_{i,j-1} \\ + u^h_{i,j+1} - 4u^h_{i,j} \end{array}}{h^2} = f^h_{i,j}, \quad i, j = 1, \ldots N-1,$$

$$u^h_{i,j} = b^h_{i,j}, \quad i = 0 \text{ or } i = N \text{ or } j = 0 \text{ or } j = N, \tag{19}$$

or more concisely as $L^h u^h = f^h$.

We can easily adapt Jacobi's algorithm to the 2D problem: at each iteration, we change our approximation to $u^h_{i,j}$, $i, j = 1, \ldots, N-1$, to satisfy the (i,j)th equation in the system in Equation 19, with the neighboring values taken from the previous iteration. We can generalize our smoothing analysis of damped Jacobi relaxation to this case. By minimizing the smoothing factor with respect to the damping, we then find that the best achievable smoothing factor is 0.6, obtained with a damping factor of 0.8.

To apply the multigrid V-Cycle to the 2D model problem, we still need to define our coarse grids, along with appropriate intergrid transfer operators. We coarsen naturally by eliminating all the mesh points with either i or j odd, such that roughly one-fourth of the mesh points remain. For restriction, we can once again use simple injection, defined for a fine-grid vector, g^h, by

$$\left(I_h^{2h} g^h\right)_{i,j} = g^h_{2i,2j}, \quad i, j = 1, \ldots, \frac{N}{2} - 1,$$

that is, the coarse-grid value is simply given by the fine-grid value at the corresponding location, or full weighting, defined by

$$\left(I_h^{2h} g^h\right)_{i,j} = \frac{1}{16}\left(\begin{array}{l} g^h_{2i-1,2j-1} + g^h_{2i-1,2j+1} \\ + g^h_{2i+1,2j-1} + g^h_{2i+1,2j+1} \\ +2\left(\begin{array}{l} g^h_{2i-1,2j} + g^h_{2i+1,2j} \\ + g^h_{2i,2j-1} + g^h_{2i,2j+1} \end{array} \right) \\ +4 g^h_{2i,2j} \end{array} \right),$$

$$i, j = 1, \ldots, \frac{N}{2} - 1,$$

where a local averaging is applied. For coarse-to-fine data transfer, a common choice is bilinear interpolation, defined for a coarse-grid vector g^{2h} by

$$\left(I_{2h}^h g^{2h}\right)_{i,j} = \begin{cases} g^{2h}_{i/2,j/2}, & \text{if } i \text{ is even and } j \text{ is even}, \\ \frac{1}{2}\left(g^{2h}_{(i+1)/2,j/2} + g^{2h}_{(i-1)/2,j/2}\right), & \text{if } i \text{ is odd and } j \text{ is even}, \\ \frac{1}{2}\left(g^{2h}_{i/2,(j+1)/2} + g^{2h}_{i/2,(j-1)/2}\right), & \text{if } i \text{ is even and } j \text{ is odd}, \\ \frac{1}{4}\left(g^{2h}_{(i+1)/2,(j+1)/2} + g^{2h}_{(i-1)/2,(j+1)/2} + g^{2h}_{(i+1)/2,(j-1)/2} + g^{2h}_{(i-1)/2,(j-1)/2}\right) & \text{if } i \text{ is odd and } j \text{ is odd}, \end{cases}$$

for $i = 1, \ldots, N-1$.

Numerical Tests

To demonstrate the multigrid algorithm's effectiveness, we test it on the 1D and 2D model problems with $N = 128$ and compare its convergence behavior to that of simple Jacobi relaxation. We apply V-Cycles with $\nu_1 = 2$, $\nu_2 = 1$—denoted $V(2, 1)$—to a problem with a known solution, beginning with a random initial guess for the solution. (It's easy to show that the asymptotic convergence behavior is independent of the particular solution.) We use (bi)linear interpolation and full-weighting restriction and plot the fine-grid algebraic error's root mean square (RMS) versus the number of iterations. Figure 7 shows the results for the 1D and 2D model problems. The V-Cycle's computational cost is similar to that of a few Jacobi iterations, as I will show in the next section. Hence, for fairness, we consider every 10 Jacobi relaxations as a single iteration in the comparison, so that 100 iterations marked in the plot actually correspond to 1,000 Jacobi relaxations. Thus, a V-Cycle in these plots is no more costly than a single Jacobi iteration.

We can see that each V-Cycle reduces the error by roughly an order of magnitude, whereas Jacobi relaxation converges slowly. The smoothing analysis can approximately predict the fast multigrid convergence: on the fine grid, the reduction factor of rough errors per each relaxation sweep is bounded from above by the smoothing factor, whereas the smooth errors are nearly eliminated altogether by the coarse-grid correction.

Computational Complexity

The number of operations performed in the V-Cycle on each grid is clearly proportional to the number of mesh points because this holds for each subprocess—relaxation, residual computation, restriction, and prolongation. Because the number of variables at each grid is approximately a constant fraction of that of the next finer grid (one-half in 1D and one-fourth in 2D), the total number of operations per V-Cycle is greater than the number of operations performed on just the finest grid only by a constant factor. In 1D, this factor is bounded by $1 + 1/2 + 1/4 + \ldots = 2$, and in 2D by $1 + 1/4 + 1/16 + \ldots = 4/3$. The amount of computation required for a single Jacobi relaxation on the finest grid (or a calculation of the residual, which is approximately the same) is called a *work unit* (WU). Our V-Cycle requires three WUs for relaxation on the finest grid and one more for computing the residual. The prolongation plus the restriction operations together require no more than one WU. Thus, the fine-grid work is roughly five WUs per V-Cycle. Therefore, the entire work is approximately 10 WUs per V-Cycle in 1D and seven in 2D.

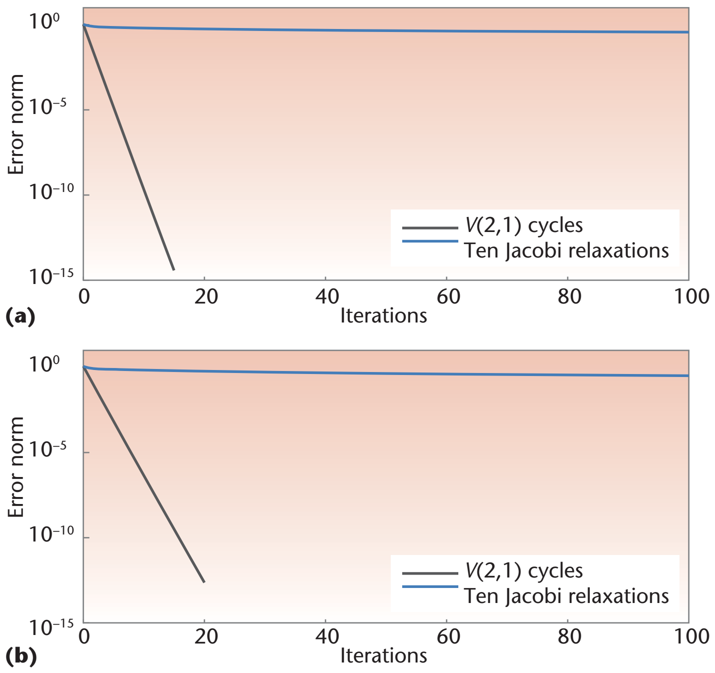

Figure 7. Convergence of the algebraic error. Multigrid $V(2, 1)$ cycles versus Jacobi iterations comprised of 10 relaxations each, with $N = 128$: (a) 1D and (b) 2D.

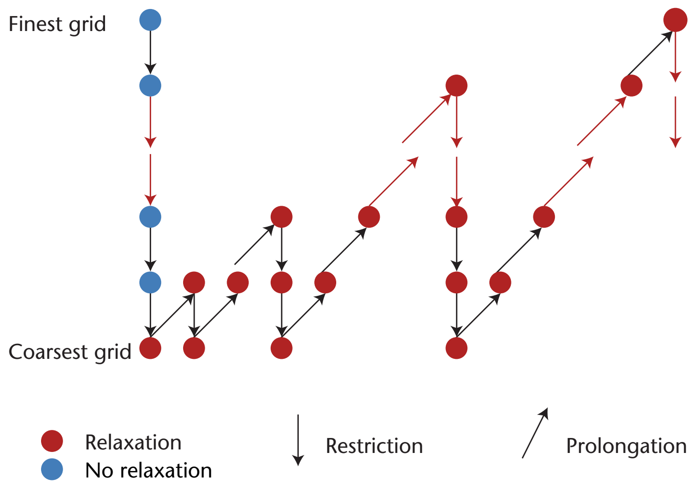

Figure 8. A schematic description of the full multigrid algorithm for $\mu = 1$. The algorithm proceeds from left to right and top (finest grid) to bottom (coarsest grid.) Initially, we transfer the problem down to the coarsest grid. We solve the problem on the coarsest grid and then interpolate this solution to the second-coarsest grid and perform a V-Cycle. These two steps are repeated recursively to finer and finer grids, ending with the finest grid.

Finally, observe that on each grid in the hierarchy, the operations (relaxation, prolongation and restriction) can be performed in parallel. Thus, the number of sequential steps required by the V-Cycle is proportional to the number of grids, which is merely $O(\log N)$.

The Full Multigrid Algorithm

In the numerical experiments, we've used a random initial guess for the solution for testing purposes. In practice, we want to enlist the best possible initial guess. A natural and effective approach is to first solve the problem approximately on a coarser grid and interpolate this solution to the fine grid to be used as a first approximation. This is applied recursively, yielding the full multigrid (FMG) algorithm. (Figure 8 gives a schematic description.) We need a positive parameter, μ, which denotes the number of V-Cycles applied at each level of the algorithm. The FMG algorithm is then defined as

```
Algorithm FMG: $u^h$ = FMG($h, N, f^h, b^h, \nu_1, \nu_2, \mu$)
  1. If $N == 2$, solve $L^h u^h = f^h$ and return $u^h$;
  2. $f^{2h} = I_h^{2h} f^h$;
     restrict $f^h$.
  3. $b^{2h} = \hat{I}_h^{2h} b^h$;
     restrict boundary conditions.
  4. $u^{2h}$ = FMG($2h, N/2, f^{2h}, b^{2h}, \nu_1, \nu_2, \mu$);
     recursive call;
  5. $u^h = \hat{I}_{2h}^h u^{2h}$;
     interpolate initial approximation, excluding
     boundary conditions.
  6. Repeat $\mu$ times: $u^h$ = VCycle($u^h, h, N, f^h,$
     $\nu_1, \nu_2$);
     apply $\mu$ V-Cycles.
  7. Return $u^h$.
```

Here, $\hat{I}_h^{2h}$ is a restriction operator for the boundary values, and $\hat{I}_{2h}^h$ is a prolongation operator that might differ from the one we use in the V-Cycle. Generally, it must be more accurate, so a common choice is (bi)cubic interpolation. Recall the distinction we made earlier between the discretization and algebraic errors. The total error is the sum of these two errors—that is, the difference between the solution to the differential equation (sampled on the fine grid) and the current discrete approximation. This is the relevant measure of the current approximation's accuracy. Once the algebraic error becomes smaller than the discretization error, the latter begins to dominate the total error.

Clearly, no advantage exists in working hard to reduce the algebraic error much further. Indeed, if we want to spend an additional computational effort at this point, we should reduce the discretization error by appealing to a yet finer grid. For many problems, researchers have shown that the FMG algorithm can yield small algebraic errors compared to the discretization error even with just $\mu = 1$. The FMG algorithm's computational complexity is linear (an FMG algorithm with $\mu = 1$ costs about twice as much as a V-Cycle in 1D and about one and a third as much in 2D). The FMG algorithm, therefore, yields an accurate solution to the problem at the cost of just a few WUs—typically only several dozen operations per fine-grid variable.

The multigrid algorithms I describe in this article provide a glimpse into the complex and often subtle machinery of multiscale computational methods. These algorithms are immediately useful for simple linear elliptic problems on simple domains. With relatively minor modifications, they can be generalized to handle nonlinear, anisotropic, or moderately inhomogeneous problems as well as elliptic systems. Still more elaborate modifications are required to obtain efficient solvers for singular-perturbation problems and for problems with discontinuous coefficients, complex domains, or unstructured grids. These topics have been researched widely over the last 30 years, and several of them are addressed in this special issue. Researchers have resolved many of the difficulties, while others remain open and subject to research. Exploiting these ideas outside the realm of differential equations is particularly challenging.

While the details of the methods that turn out to be optimal for different problems vary widely, the concepts and fundamental tasks I describe in this article are at the basis of most efficient multiscale approaches.

Reference

1. I.A. Wagner and A.M. Bruckstein, "Row Straightening via Local Interactions," *Circuits, Systems, and Signal Processing*, vol. 16, no. 3, 1997, pp. 287–305.

***Irad Yavneh** is a professor in the Faculty of Computer Science, Technion – Israel Institute of Technology. His research interests include multiscale computational techniques, scientific computing and computational physics, image processing and analysis, and geophysical fluid dynamics. Yavneh has a PhD in applied mathematics from the Weizmann Institute of Science, Israel. Contact him at irad@cs.technion.ac.il.*