

Multigrid Methods

The multigrid method provides algorithms which can be used to accelerate the rate of convergence of iterative methods, such as Jacobi or Gauss-Seidel, for solving elliptic partial differential equations.

Iterative methods start with an approximate guess for the solution to the differential equation. In each iteration, the difference between the approximate solution and the exact solution is made smaller.

One can analyze this difference or *error* into components of different wavelengths, for example by using Fourier analysis. In general the error will have components of many different wavelengths: there will be short wavelength error components and long wavelength error components.

Algorithms like Jacobi or Gauss-Seidel are *local* because the new value for the solution at any lattice site depends only on the value of the previous iterate at neighboring points. Such local algorithms are generally more efficient in reducing short wavelength error components.

The basic idea behind multigrid methods is to reduce long wavelength error components by updating blocks of grid points. This strategy is similar to that employed by cluster algorithms in Monte Carlo simulations of the Ising model close to the phase transition temperature where long range correlations are important. In fact, multigrid algorithms can also be combined with Monte Carlo simulations.

Multigrid method for Poisson's equation in 2-D

With a small change in notation, Poisson's equation in 2-D can be written:

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = -f(x, y) ,$$

where the unknown solution $u(x, y)$ is determined by the given *source term* $f(x, y)$ in a closed region. Let's consider a square domain $0 \leq x, y \leq 1$ with homogeneous Dirichlet boundary conditions $u = 0$ on the perimeter of the square. The equation is discretized on a grid with $L + 2$ lattice points, i.e., L interior points and 2 boundary points, in the x and y directions. At any interior point, the exact solution obeys

$$u_{i,j} = \frac{1}{4} [u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} + h^2 f_{i,j}] .$$

The algorithm uses a succession of lattices or grids. The number of different grids is called the number of multigrid levels ℓ . The number of interior lattice points in the x and y directions is then taken to be 2^ℓ , so that $L = 2^\ell + 2$, and the lattice spacing $h = 1/(L - 1)$. L is chosen in this manner so that the downward multigrid iteration can construct a sequence of coarser lattices with

$$2^{\ell-1} \rightarrow 2^{\ell-2} \rightarrow \dots \rightarrow 2^0 = 1$$

interior points in the x and y directions.

Suppose that $u(x, y)$ is the approximate solution at any stage in the calculation, and $u_{\text{exact}}(x, y)$ is the exact solution which we are trying to find. The multigrid algorithm uses the following definitions:

- The *correction*

$$v = u_{\text{exact}} - u$$

is the function which must be added to the approximate solution to give the exact solution.

- The *residual* or *defect* is defined as

$$r = \nabla^2 u + f .$$

Notice that the correction and the residual are related by the equation

$$\nabla^2 v = [\nabla^2 u_{\text{exact}} + f] - [\nabla^2 u + f] = -r .$$

This equation has exactly the same form as Poisson's equation with v playing the role of unknown function and r playing the role of known source function!

Simple V-cycle algorithm

The simplest multigrid algorithm is based on a *two-grid* improvement scheme. Consider two grids:

- a *fine grid* with $L = 2^\ell + 2$ points in each direction, and
- a *coarse grid* with $L = 2^{\ell-1} + 2$ points.

We need to be able to move from one grid to another, i.e., given any function on the lattice, we need to be able to

- *restrict* the function from fine $\rightarrow$ coarse, and
- *prolongate* or *interpolate* the function from coarse $\rightarrow$ fine.

Given these definitions, the multigrid V -cycle can be defined recursively as follows:

- If $\ell = 0$ there is only one interior point, so solve exactly for

$$u_{1,1} = (u_{0,1} + u_{2,1} + u_{1,0} + u_{1,2} + h^2 f_{1,1})/4 .$$

- Otherwise, calculate the current $L = 2^\ell + 2$.
- Perform a few pre-smoothing iterations using a local algorithm such as Gauss-Seidel. The idea is to damp or reduce the short wavelength errors in the solution.
- Estimate the correction $v = u_{\text{exact}} - u$ as follows:

- Compute the residual

$$r_{i,j} = \frac{1}{h^2} [u_{i+1,j} + u_{i-1,j} + u_{i,j+1} + u_{i,j-1} - 4u_{i,j}] + f_{i,j} .$$

- Restrict the residual $r \rightarrow R$ to the coarser grid.
- Set the coarser grid correction $V = 0$ and improve it recursively.
- Prolongate the correction $V \rightarrow v$ onto the finer grid.
- Correct $u \rightarrow u + v$.
- Perform a few post-smoothing Gauss-Seidel iterations and return this improved u .

How does this recursive algorithm scale with L ? The pre-smoothing and post-smoothing Jacobi or Gauss-Seidel iterations are the most time consuming parts of the calculation. Recall that a single Jacobi or Gauss-Seidel iteration scales like $\mathcal{O}(L^2)$. The operations must be carried out on the sequence of grids with

$$2^\ell \rightarrow 2^{\ell-1} \rightarrow 2^{\ell-2} \rightarrow \dots \rightarrow 2^0 = 1$$

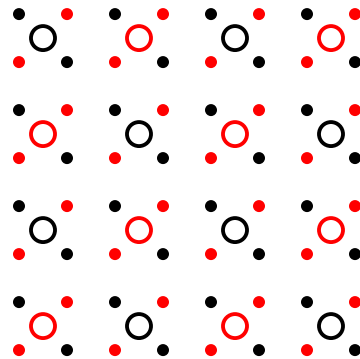
interior lattice points in each direction. The total number of operations is of order

$$L^2 \sum_{n=0}^{\ell} \frac{1}{2^{2n}} \leq L^2 \frac{1}{1 - \frac{1}{4}} .$$

Thus the multigrid V -cycle scales like $\mathcal{O}(L^2)$, i.e., linearly with the number of lattice points N !

Restricting the Residual to a Coarser Lattice

The coarser lattice with spacing $H = 2h$ is constructed as shown. A simple algorithm for *restricting* the residual to the coarser lattice is to set its value to the average of the values on the four surrounding lattice points (cell-centered coarsening):



$$R_{I,J} = \frac{1}{4} [r_{i,j} + r_{i+1,j} + r_{i,j+1} + r_{i+1,j+1}] , \quad i = 2(I-1) , \quad j = 2(J-1) .$$

Prolongation of the Correction to the Finer Lattice

Having restricted the residual to the coarser lattice with spacing $H = 2h$, we need to solve the equation

$$\nabla^2 V = -R(x, y) ,$$

with the initial guess $V(x, y) = 0$. This is done by two-grid iteration

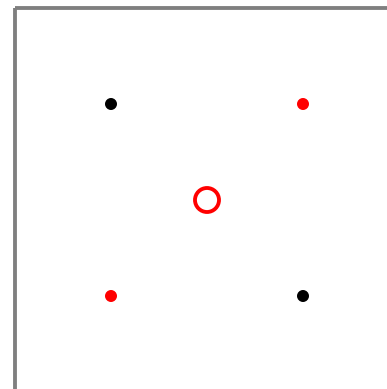
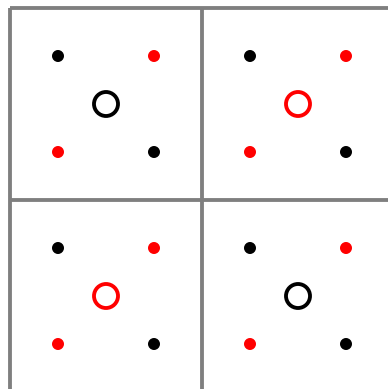
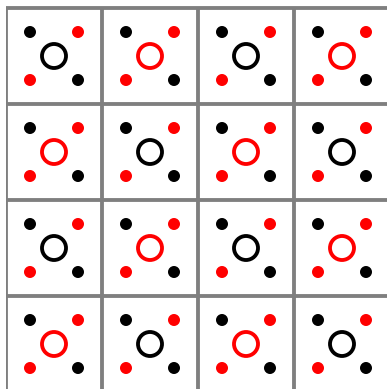
$$V = \text{twoGrid}(H, V, R) .$$

The output must now be interpolated or *prolongated* to the finer lattice. The simplest procedure is to copy the value of $V_{I,J}$ on the coarse lattice to the 4 neighboring cell points on the finer lattice:

$$v_{i,j} = v_{i+1,j} = v_{i,j+1} = v_{i+1,j+1} = V_{I,J} , \quad i = 2(I-1) , \quad j = 2(J-1) .$$

Cell-centered and Vertex-centered Grids and Coarsenings

In the *cell-centered* prescription, the spatial domain is partitioned into discrete cells. Lattice points are defined at the *center* of each cell as shown in the figure:



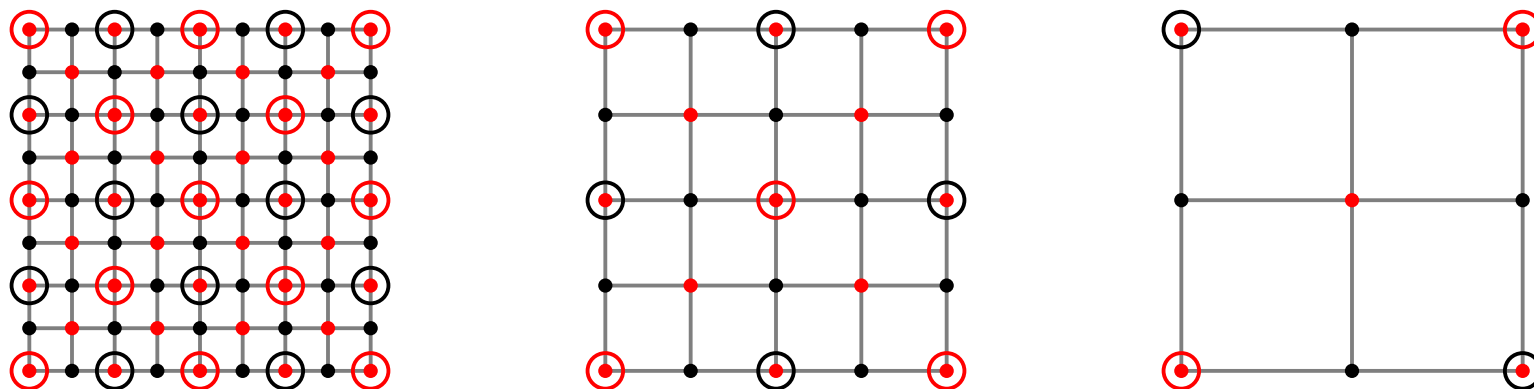
The coarsening operation is defined by doubling the size of a cell in each spatial dimension and placing a coarse lattice point at the center of the doubled cell.

Note that the number of lattice points or cells in each dimension must be a power of 2 if the coarsening operation is to terminate with a single cell. In the figure, the finest lattice has $2^3 = 8$ cells in each dimension, and 3 coarsening operations reduce the number of cells in each dimension

$$2^3 = 8 \rightarrow 2^2 = 4 \rightarrow 2^1 = 2 \rightarrow 2^0 = 1 .$$

Note also that with the cell-centered prescription, the spatial location of lattice sites *changes* with each coarsening: coarse lattice sites are spatially displaced from fine lattice sites.

A *vertex-centered* prescription is defined by partitioning the spatial domain into discrete cells and locating the discrete lattice points at the *vertices* of the cells as shown in the figure:



The coarsening operation is implemented simply by dropping every other lattice site in each spatial dimension.

Note that the number of lattice points in each dimension must be one greater than a power of 2 if the coarsening operation is to reduce the number of cells to a single coarsest cell. In the example in the figure the finest lattice has $2^3 + 1 = 9$ lattice sites in each dimension, and 2 coarsening operations reduce the number of vertices in each dimension

$$2^3 + 1 = 9 \rightarrow 2^2 + 1 = 5 \rightarrow 2^1 + 1 = 3 .$$

The vertex-centered prescription has the property that the spatial locations of the discretization points are not changed by the coarsening operation.

Boundary points

Let's assume that the outermost perimeter points are taken to be the boundary points. The behavior of these boundary points is different in the two prescriptions:

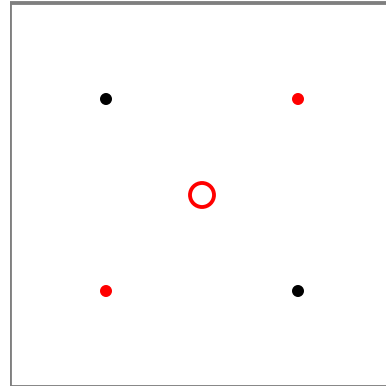
- **Cell-centered Prescription:** The boundary points move in space towards the center of the region at each coarsening. This implies that one has to be careful in defining the “boundary values” of the solution.
- **Vertex-centered Prescription:** The boundary points do not move when the lattice is coarsened. This makes it easier in principle to define the boundary values.

These two different behaviors of the boundary points make the vertex-centered prescription a little more convenient to use in multigrid applications. However, there is no reason why the cell-centered prescription should not work as well.

Restriction and Prolongation Operators

In the multigrid method it is necessary to move functions from a fine grid to the next coarser grid (*Restriction*), and from a coarse grid to the next finer grid (*Prolongation*). Many prescriptions for restricting and prolongating functions have been studied. Let's consider two of the simplest prescriptions appropriate for cell- and vertex-centered coarsening:

- **Cell-centered Coarsening:** In this prescription, a coarse lattice point is naturally associated with 2^d neighboring fine lattice points in d -dimensions.
- Suppose that $f(\vec{x})$ is a function on the fine lattice at spatial position $\vec{x}$, and $F(\vec{X})$ is the corresponding function on the coarse lattice, then this diagram suggests a simple prescription for restriction and prolongation.



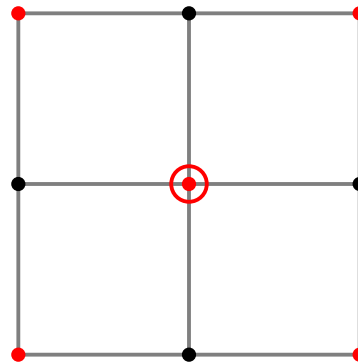
- **Restriction:** Average the function values at the 4 neighboring fine lattice sites $\vec{x}_i$:

$$F(\vec{X}) = \frac{1}{4} \sum_{i=1}^4 f(\vec{x}_i) .$$

- **Prolongation:** Inject the value of the function at the coarse lattice site to the 4 neighboring fine lattice sites:

$$f(\vec{x}_i) = F(\vec{X}) , \quad i = 1 \dots 4$$

- **Vertex-centered Coarsening:** Consider a coarse lattice point and the 9 neighboring fine lattice points shown in the figure:



In this prescription, a coarse lattice point can naturally associated (in 2-D) with

- the corresponding fine lattice point, or
- the four nearest neighbor fine lattice points, left, right, up, and down, or
- with the four diagonally nearest fine lattice points, etc.

It is a little more complicated here to define transfer operators. The problem is that the fine lattice points are associated with *more than one* coarse lattice point, unlike the cell-centered case:

- The single red fine lattice point in the center coincides with an unique coarse lattice point.
- Each of the 4 black fine lattice points however is equidistant from *two* coarse lattice points.
- Each of the 4 red fine lattice points is equidistant from *four* coarse lattice points.

This sharing of lattice points suggests the following prescriptions:

- **Prolongation:** use *bilinear interpolation* in which the value of F at a coarse grid point is copied to 9 neighboring fine-grid points with the following weights:

$$\begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}.$$

This matrix is called the *stencil* for the prolongation.

- **Restriction:** The restriction operator is taken to be the *adjoint* of the prolongation operator:

$$\begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix}.$$

This choice of restriction operator is called *full weighting*.

Improvements and More Complicated Multigrid Algorithms

The algorithm implemented above is the simplest multigrid scheme with a single *V-cycle*. Section 19.6 of *Numerical Recipes* discusses various ways of improving this algorithm:

- One can repeat the two-grid iteration more than once. If it is repeated *twice* in each multigrid level one obtains a *W-cycle* type of algorithm.
- The *Full Multigrid Algorithm* starts with the coarsest grid on which the equation can be solved exactly. It then proceeds to finer grids, performing one or more *V-cycles* at each level along the way. *Numerical Recipes* gives a program `mglin(u,n,ncycle)` which accepts the source function $-f$ in the first argument and implements the full multigrid algorithm with $\ell = \log_2(n - 1)$ levels, performing `ncycle` V-cycles at each level, and returning the solution in the array parameter `u`. Note that this program assumes that the number of lattice points in each dimension L is odd, which leads to vertex centered coarsening:

