A Two-Grid Method

Fotios Kasolis
PH.D. IN COMPUTATIONAL SCIENCE & ENGINEERING
kasolis@uni-wuppertal.de

Based on the observation that a simple iterative solver quickly decreases the high frequency components of the error but does a poor job on the low frequency components, we first use the Richardson method and then we use a direct solver on a coarser grid. A two-grid method is summarized in the following algorithm.

1. **for** $k = 0, 1, 2, \ldots$, until convergence do:
2. Compute $\mathbf{x}_h^{(k)}$ by doing m iterations on the linear system $\mathbf{A}_h \mathbf{x}_h = \mathbf{b}_h$.
3. Compute the residual $\mathbf{r}_h^{(k)} = \mathbf{b}_h - \mathbf{A}_h \mathbf{x}_h^{(k)}$.
4. Restrict the residual to the coarse grid, $\mathbf{r}_H^{(k)} = \mathbf{R}_{h \to H} \mathbf{r}_h^{(k)}$.
5. Solve the linear system $\mathbf{A}_H \mathbf{d}_H^{(k)} = \mathbf{r}_H^{(k)}$ for the nodal error correction $\mathbf{d}_H^{(k)}$.
6. Prolongate the correction to the fine grid, $\mathbf{d}_h^{(k)} = \mathbf{P}_{H \to h} \mathbf{d}_H^{(k)}$.
7. Update the solution $\mathbf{x}_h^{(k+1)} = \mathbf{x}_h^{(k)} + \mathbf{d}_h^{(k)}$.
8. **end for**

Here, for the iterations on the fine grid we use the Richardson method, naively implemented in GNU Octave / Matlab as `Richardson`.

```
function x = Richardson(A, x, b, maxit, alpha)
for k = 1:maxit
  x = x + alpha*( b - A*x );
end
```

As a small example, consider the one-dimensional boundary value problem

$$-u'' = 1, \qquad u(0) = u(1) = 0$$

on a uniform grid $0 = x_0 < x_1 < x_2 < \cdots < x_n = 1$. The fine grid consists of nodes labeled $1, 2, \ldots, n-1$ with $h = 1/n$, so that

$$A_h = \frac{1}{h} \begin{pmatrix} 2 & -1 & 0 & 0 & \cdots & 0 \\ -1 & 2 & -1 & 0 & \cdots & 0 \\ 0 & -1 & 2 & -1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix}, \quad b_h = h \begin{pmatrix} 1 \\ 1 \\ 1 \\ \vdots \\ 1 \\ 1 \end{pmatrix},$$

whereas the coarse grid has $H = 2h$. Given a function on the coarse grid, we transfer it to the fine grid using linear interpolation. In doing so, we let the nodes of the coarse grid that are also in the fine grid keep their values. Further, the values at the nodes of the fine grid that are not in the coarse grid are given by averaging their left and right neighbors, resulting in a prolongation matrix of the form

$$\mathbf{P}_{H \to h} = \begin{pmatrix} 1/2 & & \\ 1 & & \\ 1/2 & 1/2 & \\ & 1 & \\ & 1/2 & 1/2 \\ & & 1 \\ & & 1/2 \end{pmatrix}$$

Utilizing the prolongation matrix, the restriction matrix is given by

$$\mathbf{R}_{h \to H} = \frac{1}{2} \mathbf{P}_{H \to h}^{\top},$$

where we have divided by a factor two since the rows have to sum up to unity, a natural property for any interpolation or averaging operator.

Study, complete, and implement the following code snippet.

```
nf = 2*25 - 1;          % number of fine nodes
nc = ( nf - 1 )/2;      % coarse nodes
h  = ...;               % mesh size
```

```
x  = 0:h:1;              % mesh
e  = ones(nf, 1);

                         % fine matrix
A = (1/h)*spdiags([-e 2*e -e], -1:1, nf, nf);
b = h*ones(nf, 1);       % load vector
u = zeros(nf, 1);        % solution guess
P = sparse(nf, nc);      % prolongation matrix
for i = 1:nc
  P(2*i-1,i) = ...;
  P(2*i,i)   = ...;
  P(2*i+1,i) = ...;
end

R  = 0.5*transpose(P);  % prolongation matrix
RAP= R*A*P;              % coarse-grid matrix, why?
for k = 1:5              % outer iteration loop
  u = ...                % fine-grid solution
  r = R*(b-A*u);         % residual
  e = mldivide(RAP, r); % correction
  u = u + P*e;           % solution update
end

plot(x, [0, transpose(u), 0]);
```