PROGRAMMING OF MULTIGRID METHODS

LONG CHEN

In this note, we explain the implementation detail of multigrid methods. We will use the approach by space decomposition and subspace correction method; see Chapter: Subspace Correction Method and Auxiliary Space Method. The matrix formulation will be obtained naturally, when the functions’ basis representation is inserted. We also include a simplified implementation of multigrid methods using finite difference approach. To distinguish functions and vectors, we use boldface letter for a matrix representation of an operator or a vector representation of a function.

1. TWO LEVEL METHODS AND TRANSFER OPERATORS

We use two level methods to illustrate how to realize operators by matrices. The space decomposition we choose is
\[ V = V_1 + V_2 \quad \text{with} \quad V_1 \subset V_2 = V. \]

We call \( V \) fine space and \( V_1 \) coarse space since it is usually based on a coarse mesh.

Recall that the PSC for this two level decomposition in operator form is
\begin{align*}
(1) \quad & r = f - Au^k; \\
(2) \quad & e = I_1 R_1 I_1^T r + I_2 R_2 I_2^T r; \\
(3) \quad & u^{k+1} = u^k + e.
\end{align*}

The matrix form of step 1 and 3 is trivial. We only discuss the realization of step 2.

Since \( V_2 = V \), \( I_2 = I_2^T = I \). The solver \( R_2 \) can be choosen as weighted Jacobi method \( R_2 = \omega D^{-1} \) (\( \omega = 0.5 \) is recommended as the default choice) or Gauss-Seidel method \( R_2 = (D + L)^{-1} \).

The transformation to the coarse \( V_1 \) is not easy. There are three operators to realize: \( I_1, R_1, \) and \( I_1^T \).

**Prolongation operator.** Let us first discuss the operator \( I_1 = I_1^T : V_1 \to V_2 \). By the definition, it is the natural inclusion \( V_1 \hookrightarrow V \) i.e. treat a function \( u_1 \in V_1 \) as a function in \( u_1 \in V \) since \( V_1 \subset V_2 \). So the operator is the identity. But the matrix representation is different since we have different basis in \( V_1 \) and \( V_2 \). We use a 1-D two level grids in Figure 1 to illustrate the different basis functions in fine and coarse grids.

In this example \( I_1^2 : \mathbb{R}^3 \to \mathbb{R}^5 \) will map a vector with small size to one with bigger size and thus called prolongation operator. We determine this map by the following two observations:
\begin{itemize}
  \item[(1)] \( u_1 \) and \( u_2 = I_1^2 u_1 \) represent the same function in \( V_2 \);
  \item[(2)] a function in \( V_2 \) is uniquely determined by the values at the nodal points.
\end{itemize}

For nodes in both fine grids and coarse grids,
\[ u_2(1) = u_1(1), \quad u_2(3) = u_1(2), \quad u_2(5) = u_1(3). \]
For the nodes only existing in the fine grids, by (1), values at these nodes can be evaluated in the coarse grids. Since we are using linear elements, we get

\[ u_2(2) = \frac{u_1(1) + u_1(2)}{2}, \quad u_2(4) = \frac{u_1(3) + u_1(5)}{2}. \]

In matrix form, \( I^2_x \in \mathbb{R}^{5 \times 3} \) can be written as

\[
\begin{bmatrix}
1 & 0 & 0 \\
1/2 & 1/2 & 0 \\
0 & 1 & 0 \\
0 & 1/2 & 1/2 \\
0 & 0 & 1 \\
\end{bmatrix}
\]

To define the prolongation, we need to know the correspondences of the index of nodes between two grids. Different index mapping will give different prolongation matrix. A better hierarchical index of the fine grid nodes is \([1 \ 4 \ 2 \ 5 \ 3]\), for which the prolongation matrix is

\[
\begin{bmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 1/2 & 1/2 \\
1/2 & 1/2 & 0 \\
0 & 1/2 & 1/2 \\
\end{bmatrix}
\]

The presentness of the identity matrix can save the computational time of \( I^2_x \).

The construction of the prolongation operator can be easily generalized to high dimensions for the linear element. The information needed is the index map between coarse grid points and fine grids points. We classify the grid points in the fine grid into two groups:

- \( \mathcal{C} \): the points in both fine and coarse grids
- \( \mathcal{F} \): the points in the fine grid only.

For group \( \mathcal{F} \), we can use \( \text{HB} \) (hierarchical basis) matrix with \( \text{HB}(:, 2:3) \) being two parent nodes of the node \( \text{HB}(:, 1) \). Note that \( \text{HB}(:, 1) \) is the index in the fine level while
HB(:,2:3) are in the coarse level. Then the interpolation at the grids points in \( \mathcal{F} \) can be realized

\[
uf(HB(1:end,1)) = \frac{(uc(HB(1:end,2)) + uc(HB(1:end,3)))/2}{.}
\]

For group \( C \), although those grid points are in both coarse and fine grids, their indices could be different. For example, in Fig 1, the 3-rd point in the fine grid is the 2-nd one in the coarse grid. Therefore we need an index mapping, say \( \text{coarseNodeFineIdx} \), for points in group \( C \). In the example in Fig 1, \( \text{coarseNodeFineIdx} = [1 3 5] \). The interpolation is straightforward

\[
uf(\text{coarseNodeFineIdx}) = uc;
\]

Using \( HB \) and \( \text{coarseNodeFineIdx} \), the prolongation matrix do not need to be formed explicitly. On the other hand, the prolongation matrix can be easily formed by the following self-explained code

1

\[
ii = [\text{coarseNodeFineIdx}; HB(:,1); HB(:,1)];
\]

2

\[
jj = [\text{coarseNode}; HB(:,2); HB(:,3)];
\]

3

\[
ss = [\text{ones(nCoarseNode,1); 0.5*ones(nFineNode,1); 0.5*ones(nFineNode,1)}];
\]

4

\[
Pro = \text{sparse}(ii,jj,ss,nFineNode,nCoarseNode);
\]

Remark 1.1. Interpolation and restriction matrix must be altered at boundary points or neighbors of boundary points to imposing the correct boundary condition.

Restriction operator. How to compute \( I^T_1 = Q_1 \)? To compute the \( L^2 \) projection, we need to invert the mass matrix which is not cheap. Fortunately, we are not really computing the \( L^2 \) projection of a function. Instead we are dealing with a functional! Let’s recall the definition

\[
(Q_1 r, u_1) = (r, u_1) = (r, I^1_1 u_1).
\]

\( Q_1 r \) is simply to restrict the action of the dual \( r \in \mathcal{V}_2^\prime \) to the elements in \( \mathcal{V}_1 \) only. It is better to write as \( I^1_2 : \mathcal{V}_2^\prime \to \mathcal{V}_1^\prime \) and call it restriction. Note that \( \mathcal{V}_1 \subset \mathcal{V}_2 \) implies that \( \mathcal{V}_2^\prime \subset \mathcal{V}_1^\prime \). So the operator \( I^1_2 \) is also a natural inclusion of the functional. Again \( r \) and \( I^1_2 r \) will have different vector representations. The matrix form of (1) is

\[
(I^1_2 r)^T u_1 = r^T I^1_1 u_1,
\]

which implies

\[
I^1_2 = (I^1_1)^T.
\]

If we have the prolongation matrix \( Pro \) formed explicitly, the restriction matrix will be simply its transpose, i.e., \( \text{Res} = Pro' \).

Exercise 1.2. Use \( HB \) and \( \text{coarseNodeFineIdx} \) to code the restriction without forming the matrix.

Smother in the coarse space. The last component is the smoother \( R_1 \). If we know \textit{a priori} the information on the PDE and the discretization, we can easily code one. For example, for the 5-point stencil discretization of Poisson equation, one step of Gauss-Seidel iteration can be implemented using \texttt{for} loops:

1

\[
\text{for } i = 2:N-1
\]

2

\[
\text{for } j = 2:N-1
\]

3

\[
u(i,j) = (b(i,j)+(u(i-1,j)+u(i+1,j)+u(i,j-1)+u(i,j+1))/4;
\]

4

\[
\text{end}
\]

5

\[
\text{end}
\]
For general operator $A$, if we want to choose more accurate local subspace solver, say Gauss-Seidel method, we need to know the matrix $A_1$. Of course we can assemble one if we have the coarse grid. But there are several reasons to abandon this approach. First, the assembling is time consuming. Indeed this is one of the criticism of finite element methods comparing with finite difference scheme. Second it requires the information of the mesh and PDE. Then it will be problem dependent. Third, we have a better way to do it.

Recall that the operator $A_1$ is just the restriction of $A$ to the space $V_1$. Namely
\[
(A_1 u_1, v_1) := (Au_1, v_1) = (AI_1 u_1, I_1 v_1) = (I_1^T AI_1 u_1, v_1),
\]
which implies $A_1 = I_1^T AI_1$ and in the matrix form
\[
A_1 = I_1^T A I_1 = \text{Res} \ast A \ast \text{Pro}.
\]
So we can apply a triple product to form the matrix on the coarse grid. This approach is often referred as Galerkin method or variational method since the coarse matrix is a restriction of the fine one to a subspace.

2. SSC AND Multigrid method

In this section, we discuss implementation of successive subspace correction method when the subspaces are nested. Let $V = \sum_{i=1}^{J} V_i$ be a space decomposition into nested subspaces, i.e. the subspaces are nested, i.e.
\[
V_1 \subset V_2 \subset \cdots \subset V_J = V.
\]
Denoted by $N_i = \dim V_i$ and in practice $N_i = \gamma N_{i-1}$ for a factor $\gamma > 1$. For example, spaces based on a sequence of nested meshes in $\mathbb{R}^d$, the factor $\gamma \approx 2^d$.

Recall that the operator formation of SSC method is

```matlab
function e = SSC(r)
% Solve the residual equation Ae = r by SSC method
e = 0; rnew = r;
for i = J:-1:1
    ri = I_i'*rnew; % restrict the residual to subspace
    ei = R_i*ri; % solve the residual equation in subspace
    e = e + I_i*ei; % prolongate the correction to the big space
    rnew = r - A*e; % update residual
end
```

Here we change the for loop from $J:-1:1$ to reflect to the ordering from fine to coarse. The operators $I_i^T = Q_i : V \to V_i$ and $I_i : V_i \to V$ are related to the finest space. As $N_i$ is geometrically decay, the number of level $J = O(\log N)$. At each level, the prolongation matrix is of size $N \times N_i$ and thus the operation cost at each level is $O(N)$. The total cost of the direct implementation of SSC is $O(N \log N)$.

When the subspaces are nested, we do not need to return to the finest space every time. Suppose $r_i = I_i^T (r - Ae_{\text{old}})$ in the subspace $V_1$ is known, and the correction $e_i$ is used to update $e_{\text{new}} = e_{\text{old}} + e_i$. We can compute $r_{i-1}$ by the relation:
\[
r_{i-1} = Q_{i-1}(r - Ae_{\text{new}})
= Q_{i-1}Q_i(r - Ae_{\text{old}} - Ae_i)
= Q_{i-1}(r_i - Q_i A Q_i^T e_i)
= Q_{i-1}(r_i - A_i e_i).
\]
Here in the second step, we make use of the nested property $V_{i-1} \subset V_i$ to write $Q_{i-1} = Q_{i-1}^2$. Similarly the correction step can be also done accumulatively. Let us rewrite the correction as
\[ e = e_J + I_{J-1}e_{J-1} + \ldots + I_1e_1. \]

The correction can be computed by the loop
\[ e_i = e_i + I_{i-1}e_{i-1}, \quad i = 2 : J \]

Therefore only the prolongation and restriction operators between consecutive levels are needed. The cost at each level is reduced to $O(N_i)$ and the total cost is $O(N)$.

From this point of view, SSC on a nested space decomposition will result in a V-cycle multigrid method. We summarize the algorithm below. We use notation $e_i, r_i$ to emphasize that in each level we are solving the residual equation $A_i e_i = r_i$ and assume the transfer operators and discretization matrices have been computed already using the method discussed in the previous section.

```matlab
function e = Vcycle(r,J)
ri = cell(J,1); ei = cell(J,1);
ri{J} = r;
for i = J:-1:2
    ei{i} = R{i}*ri{i}; % pre-smoothing
    ri{i-1} = Res{i-1}*(ri{i}-Ai{i}*ei{i}); % update and restrict residual
end
ei{1} = Ai{1}\ri{1}; % exact solver in the coarsest level
for i = 2:J
    ei{i} = ei{i} + Pro{i}*ei{i-1}; % prolongate and correct
    ei{i} = ei{i} + R{i}'*(ri{i}-Ai{i}*ei{i}); % post-smoothing
end
e = ei{J};
```

In the second loop (/) part, we add a post-smoothing step and choose $R_i'$ as the smoother which is the transpose of the pre-smoothing operator. For example, if $R_i = (D_i + L_i)^{-1}$ is the forward Gauss-Seidel method, then the post-smoothing is backward Gauss-Seidel $(D_i + U_i)^{-1}$. This choice will make the corresponding iterator $B$ symmetric and thus can be used as preconditioner.

The function $e = Vcycle(r, \ldots)$ suggests that the mg V-cycle is used to solve the residual equation $A e = r$ and will be used as an iterator in the residual-correction method $u = u + Vcycle(f-A*u,J)$.

Due to the linearity of the iteration, we can also formulate a direct update form of multigrid method $u = Vcycle(u,f,J)$; see the next section.

3. SIMPLIFIED IMPLEMENTATION FOR FINITE DIFFERENCE METHODS

We discuss implementation of main components, especially the prolongation and restriction operator, for multigrid methods on uniform grids.

In order to evaluate the residual, we need to compute the matrix-vector product $A u$ which has been discussed in Chapter: Programming of Finite Difference Methods. A typical smoother Gauss-Seidel is also discussed in detail there. We now discuss the transfer operators: prolongation and restriction.

We consider the 1-D case first. A coarse grid is refined by adding the middle points. It is easy to figure out the index map from coarse to fine: $i \rightarrow 2i - 1$. We use the linear interpolation to construct the prolongation. The matrix-free implementation will be
One can construct a sparse matrix $I_x$

Then $uf = I_x * u$ will produce the desired result. One advantage of using the matrix form of the prolongation operator is that the restriction operator can be simply taken as the transpose of the prolongation matrix (with a possible scaling related to $h$ depending on the scaling used in the 5-point stencil).

When move to 2-D, we can still use subscript to implement a matrix-free version. We will use the bilinear interpolation to get function values at fine grids. In stencil notation, the prolongation for fine nodes on horizontal lines, on vertical lines, and in the center of coarse cells can be summarized as

$$(0.5, *, 0.5), \begin{pmatrix} 0.5 \\ * \\ 0.5 \end{pmatrix}, \begin{pmatrix} 0.25 & 0.25 \\ * & * \\ 0.25 & 0.25 \end{pmatrix}.$$ \[2.05\]

Here $*$ indicates the position of the fine grid point.

A more elegant way using tensor product structure is

$$uf = I_x * u * I_x'$$

Here recall that the unknown vector $u(1:n,1:n)$ is stored as a matrix. The left product $u \text{temp} = I_x * u$ will prolongate the function value along the column direction and thus $u \text{temp}$ is of size $N_f \times N$. The right product $u \text{temp} * I_x'$ will prolongate the function value along the row direction. One can chose different prolongation for different directions if the mesh size is non-uniform in each direction.

For a $d$-dimensional grid, a coarse grid point will have $3^d - 1$ neighboring points in the fine grid. Working on the subscript system is more tedious while the tensor product matrix version still works efficiently by using the permutation trick we mentioned in Chapter: Programming of Finite Difference Methods.
Due to the matrix-free implementation, for finite difference methods, the direct update form of iteration is preferable. For example, the G-S smoother is better coded as $u = GS(u,f)$. The direct update form of V-cycle is sketched below. The matrix-vector product and the exact solver in the coarsest level can be implemented in matrix-free or tensor product way.

```matlab
function u = MG(u,f,J,mu)
  % Direct update form of Multigrid Method
  if J == 1 % coarsest level: exact solve
    u = A(J);
  end
  % Presmoothing
  for i = 1:mu
    u = R(u,f,J);
  end
  % Restriction
  rc = Res(f-A(J)*u);
  % Coarse grid correction
  ec = MG(0,rc, J-1,mu);
  if W-cycle
    ec = MG(ec,rc,J-1,mu); % W-cycle
  end
  % Prolongation
  u = u + Pro(ec);
  % Postsmoothing
  for i = 1:mu
    u = R'(u,f,J);
  end
end
```

4. Algebraic Multigrid Method

The multigrid methods discussed in the previous sections depends heavily on the geometry of the underlying meshes and therefore called geometric multigrid methods. In most applications, the grid could be totally unstructured and a hierarchical structure is not available. In some cases, only the matrix is given without any grid information. It would be desirable to still solve the algebraic equation using the idea of multigrid.

Looking the procedure carefully, the hierarchical structure of grids is used to construct the transfer operators. After that, the matrix equation in the coarse grid can be computed by triple product and the smoother can be algebraically taking as G-S $\text{tril}(A)$ or weighted Jacobi $\omega * \text{diag}(A)$. Two essential ingredients are needed to construct the prolongation operator from a coarse grid to a fine grid

1. Index map from coarse nodes to fine nodes.
2. Interpolation of the fine variables.

Let us revisit these two ingredients in an algebraic way. Suppose $A$ is an $N \times N$ matrix. The fine nodes are the index set $V = \{1, 2, \cdots, N\}$. From the given matrix $A$, we could construct a weighted graph $G = G(A) = (V, E)$. The edge $[i,j]$ exists if $a_{ij} \neq 0$. As the matrix is symmetric, the graph $G$ is undirected.

**Coarsening.** Recall that the node in the fine level can be classified into $\mathcal{C}$ and $\mathcal{F}$. Now if only matrix is given, a node will be understood as an abstract point. No coordinate is associated to it. The fine nodes are the index set $V = \{1, 2, \cdots, N\}$. A subset $\mathcal{C}$ of $\mathcal{N}$ will
be identified as the nodes of a ‘coarse grid’ and the rest is \( \mathcal{F} \), i.e. \( \mathcal{N} = \mathcal{C} \cup \mathcal{F} \). In addition, for any \( i \in \mathcal{F} \), the neighboring ‘coarse nodes’ \( \mathcal{J}(i) \subseteq \mathcal{C} \) should be found. In hierarchical meshes case, \( \mathcal{J}(i) \) is simply \( \mathbb{H}_B \) array which only contains two coarse nodes. In summary we need to pick up \( \mathcal{C} \) and construct \( \mathcal{J}(i) \) for all \( i \in \mathcal{F} \).