Integration of \( hp \)-adaptivity and a two-grid solver for elliptic problems

D. Pardo, L. Demkowicz *

Institute for Computational Engineering and Sciences (ICES), The University of Texas at Austin, Austin, TX 78712, USA

Received 27 May 2004; received in revised form 13 January 2005; accepted 15 February 2005

Abstract

We present implementation details and analyze convergence of a two-grid solver forming the core of the fully automatic \( hp \)-adaptive strategy for elliptic problems. The solver delivers a solution for a fine grid obtained from an arbitrary coarse \( hp \)-grid by a global \( hp \)-refinement. The classical V-cycle algorithm combines an overlapping block Jacobi smoother with optimal relaxation, and a direct solve on the coarse grid. A simple theoretical analysis is illustrated with extensive numerical experimentation.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Multigrid; Iterative solvers; Preconditioners

1. Introduction

The paper is concerned with a construction and study of an iterative solver for linear systems resulting from \( hp \)-adaptive finite element (FE) discretizations of elliptic boundary-value problems. Here, \( h \) stands for element size, and \( p \) denotes element order of approximation, both varying locally throughout the mesh [15].

The algorithm presented in [14] produces a sequence of optimally \( hp \)-refined meshes that deliver exponential convergence rates in terms of the energy norm error versus the size of the discrete problem (number of degrees-of-freedom (d.o.f.)) or CPU time. A given (coarse) \( hp \)-mesh is first refined globally in both \( h \) and...
to yield a fine mesh, i.e., each element is broken into four element-sons (eight in 3D), and the discretization order $p$ is raised uniformly by one. Then, we solve the problem of interest on the fine mesh. The next optimal coarse mesh is then determined by minimizing the projection based interpolation error of the fine mesh solution with respect to the optimally refined coarse mesh. The algorithm is very general, and it applies not only to $H^1$-conforming but also $H$(curl)- and $H$(div)-conforming discretizations [12,13]. Moreover, since the mesh optimization process is based on minimizing the interpolation error rather than the residual, the algorithm is problem independent, and it can be applied to non-linear and eigenvalue problems as well.

Critical to the success of the proposed adaptive strategy is the solution of the fine grid problem. Typically, in 3D, the global $hp$-refinement increases the problem size at least by one order of magnitude, making the use of an iterative solver inevitable. With a multigrid solver in mind, we choose to implement first a two-grid solver based on the interaction between the coarse and fine $hp$-meshes. The choice is quite natural. The coarse meshes are minimum in size. Also, for wave propagation problems in the frequency domain, the size of the coarsest mesh in the multigrid algorithm is limited by the condition that the mesh has to resolve all eigenvalues below the frequency of interest. Consequently, the sequence of multigrid meshes may be limited to just a few meshes only.

The fine mesh is obtained from the coarse mesh by the global $hp$ -refinement. This guarantees that the corresponding FE spaces are nested, and allows for the standard construction of the prolongation and restriction operators. Notice that the sequence of optimal coarse $hp$-meshes produced by the self-adaptive algorithm discussed above is not nested. The coarse meshes are highly non-uniform, both in element size $h$ and order of approximation $p$, and they frequently include anisotropically refined elements (construction of multigrid algorithms for such anisotropically refined meshes is sometimes difficult [22]), but the global refinement simplifies the logic of implementation greatly.

Customary, any work on iterative methods starts with self-adjoint and positive definite problems, and this is also the subject of the present work. We include 2D and 3D examples of problems with highly non-homogeneous and anisotropic material data, as well as problems including corners and edge singularities.

The structure of our presentation is as follows: We begin with a formulation of the algorithm, and an elementary study of its convergence related to the standard Domain Decomposition methods and the Schwartz framework [24,3,7,18]. In Section 3 we present some implementation details. Section 4 is devoted to an extensive numerical experimentation, with conclusions drawn in Section 5.

Notice that the two-grid solver is intended to be used not only as a solver by itself, but also as a crucial part of the $hp$-adaptive strategy. Among several implementation and theoretical issues that we address in this paper, one question is especially important for us; is it possible to guide the optimal hp-refinements with a partially converged fine grid solution only, and to what extent?

As it will be shown throughout the paper, it is possible to guide optimal refinements with a partially converged fine grid solution only.

## 2. Formulation of the method

### 2.1. Variational problem

We are interested in solving a variational problem in the standard form:

$$
\begin{align*}
\{ & u \in V, \\
& a(u,v) = l(v) \quad \forall v \in V.
\end{align*}
$$

(2.1)
Here,

- $V$ is a real (complex) Hilbert space with an inner product $(u,v)_V$, and corresponding norm $\|u\|_V$;
- $a(u,v)$ is a bilinear (sesquilinear) and symmetric (Hermitian) form that is assumed to be coercive and continuous on space $V$,
  \[
  \alpha \|u\|_V^2 \leq a(u,u) \leq M \|u\|_V^2, \quad \alpha, M > 0;
  \] (2.2)
- $l \in V'$ is a linear (anti-linear), continuous functional on $V$.

Form $a(u,v)$ itself defines an alternative (energy) inner product on $V$, with the corresponding energy norm. In what follows, we shall restrict ourselves exclusively to the energy inner product and the energy norm. Obviously, with respect to the energy norm, both coercivity and continuity constants are equal to one.

Variational problem (2.1) is equivalent to an operator equation,

\[
Au = l,
\] (2.3)

where $A: V \to V'$ is the linear operator corresponding to the bilinear form,

\[
Au(v) = a(u,v) \quad \forall u,v \in V.
\] (2.4)

The variational problem has a unique solution $u$ that depends continuously on data $l$, see e.g. [21].

2.2. Abstract convergence results

Let $V$ be a Hilbert space with inner product $(u,v)$, and the corresponding norm $\|u\|$. Let $B : V \to V$ be a self-adjoint (with respect to the inner product), continuous, and coercive operator:

\[
\lambda_{\min}(B) \|u\|^2 \leq (Bu,u) \leq \lambda_{\max}(B) \|u\|^2,
\] (2.5)

where $\lambda_{\min}(B) > 0$.

Let $e \in V$, $e \neq 0$ be a specific vector in space $V$. We shall study operator $I - \alpha B$, where coefficient $\alpha$ minimizes norm

\[
\| (I - \alpha B)e \| \to \min.
\] (2.6)

A simple calculation leads to the following result:

\[
\alpha = \frac{(e,Be)}{\|Be\|^2},
\]

\[
\| (I - \alpha B)e \|^2 = \|e\|^2 \left( 1 - \frac{|(Be,e)|^2}{\|e\|^2 \|Be\|^2} \right).
\] (2.7)

Consequently,

\[
\sup_{e \in V, e \neq 0} \left( 1 - \frac{|(Be,e)|^2}{\|e\|^2 \|Be\|^2} \right) = 1 - \inf_{e \in V, e \neq 0} \frac{|(Be,e)|^2}{\|e\|^2 \|Be\|^2}.
\] (2.8)

It is known from the study of the steepest descent (SD) method [23, p. 36] that

\[
\sup_{e \in V, e \neq 0} \left( 1 - \frac{|(Be,e)|^2}{\|e\|^2 \|Be\|^2} \right) \leq \left( \frac{\lambda_{\max}(B) - \lambda_{\min}(B)}{\lambda_{\max}(B) + \lambda_{\min}(B)} \right)^2.
\] (2.9)
and, therefore,
\[
\frac{\|(I - zB)e\|}{\|e\|} \leq \frac{\kappa - 1}{\kappa + 1},
\]
where \(\kappa = \lambda_{\text{max}}(B)/\lambda_{\text{min}}(B)\) is the condition number of operator \(B\).

Consequently, starting with an arbitrary vector \(e^{(0)}\), the (SD) iterations,
\[
e^{(n+1)} = (I - z^{(n)}B)e^{(n)},
\]
with \(z^{(n)}\) computed using \(e^{(n)}\), will converge to zero, with the contraction constant (2.10) dependent upon condition number of operator \(B\) only.

Let \(V_0 \subset V\) be now a closed subspace of \(V\), with the corresponding orthogonal projection \(P_0 : V \rightarrow V_0\). We can replace the original operator with the sum \(P_0 + B\), and arrive at the same result as above. The (SD) iterations,
\[
e^{(n+1)} = (I - z^{(n)}(P_0 + B))e^{(n)},
\]
with \(z^{(n)}\) computed using \(e^{(n)}\), will converge to zero, with the contraction constant (2.10) dependent upon condition number of operator \(P_0 + B\) only. If \(B\) is interpreted as the action of a smoother and \(P_0\) as a coarse space projection, this establishes the well known result that the additive coupling of the smoother and the coarse grid correction, ‘accelerated’ with the SD\(^1\) always converges with the rate depending upon the condition number of operator \(P_0 + B\).

We replace now the additive coupling with a multiplicative coupling, and study the operator \((I - zB)(I - P_0)\). Here, \(z\) is obtained by minimizing the norm,
\[
\|(I - zB)(I - P_0)e\| = \|(I - zB)g\| \rightarrow \min,
\]
where again, \(e \in V\) is a fixed vector. Notice that it makes no sense to introduce a relaxation factor in the first, coarse grid projection step, which is already optimal.\(^2\)

We follow now essentially the reasoning of Jan Mandel [19, Lemma 3.2]. We have
\[
\|(I - zB)(I - P_0)e\|^2 = \|(I - zB)g\|^2 = \|g\|^2\left(1 - \frac{\|(Bg, g)\|^2}{\|g\|^2\|Bg\|^2}\right) = \|g\|^2\left(1 - \frac{\|(P_0 + B)g, g\|^2}{\|g\|^2\|Bg\|^2}\right)
\]
\[
\leq \|e\|^2 \sup_{e \in V} \left(1 - \frac{\|(P_0 + B)e, e\|^2}{\|e\|^2\|Be\|^2}\right).
\]
In the last line, we have used the fact that \(P_0g = 0\), and that \(\|g\| = \|(I - P_0)e\| \leq \|e\|\).

In summary, we have established the following facts:

- The multiplicative coupling of coarse grid correction and smoothing with optimal relaxation parameter will always converge.
- The corresponding error will decrease with at least the same rate as for the additive coupling.
- The contraction constant \(\gamma\) be estimated in terms of the condition number of sum \(P_0 + B\).

\(^1\) There is no acceleration as with the conjugate gradient (CG) method, however SD forces convergence.
\(^2\) In other words, solving minimization problem \(\|(I - zP_0)e\| \rightarrow \min\), leads to optimal \(z = 1\).
2.3. Galerkin discretization

Replacing space \( V \) with a finite-dimensional subspace of \( V \), we obtain the corresponding discrete variational problem, with analogous discussion and results as for the continuous case. From now on, we will switch entirely to the finite-dimensional case, and assume that \( V \) denotes the discrete space. Let \( e_1, \ldots, e_N \) be a basis for space \( V \), with \( e_1^*, \ldots, e_N^* \) denoting the corresponding dual basis for dual space \( V^* \). Consequently, each \( v \in V \) and \( l \in V^* \) can be represented in the form,

\[
u = \sum_{j=1}^{N} x_j e_j, \quad l = \sum_{i=1}^{N} b_i e_i^*.
\]

The linear problem translates into a system of linear equations,

\[
A x = b,
\]

where

\[
x = \{x_1, \ldots, x_N\}^T, \quad b = \{b_1, \ldots, b_N\}, \quad A_{ij} = a(e_j, e_i).
\]

Introducing the canonical (discrete \( \ell^2 \)) inner product in \( \mathbb{R}^N \), \( (x, y) = y^T x \), we can translate the energy inner product into the coefficient space,

\[
a(u, v) = (Ax, y)_2 = y^T Ax,
\]

with the corresponding energy norm,

\[
\|x\|_d^2 = (Ax, x)_2 = x^T Ax.
\]

The linear problem translates into a system of linear equations,

\[
A x = b,
\]

where

\[
x = \{x_1, \ldots, x_N\}^T, \quad b = \{b_1, \ldots, b_N\}, \quad A_{ij} = a(e_j, e_i).
\]

We shall assume that the subspaces \( V_k \) are constructed by taking spans of subsets of basis functions \( e_j \), \( j = 1, \ldots, N \) introduced earlier:

\[
V_k = \text{span}\{e_{j(l)}, l = 1, \ldots, M_k\}.
\]

2.4. Overlapping block Jacobi smoother

We assume that space \( V \) can be represented as an algebraic (in general not direct) sum of subspaces \( V_k \), \( k = 1, \ldots, M \),

\[
V = V_1 + V_2 + \cdots + V_M,
\]

with corresponding inclusions and their transposes denoted by

\[
i_k : V_k \rightarrow V, \quad i_k^T : V^* \rightarrow V_k^*.
\]

Given now an \( n \)th iterate \( u^{(n)} \), and the corresponding residual \( r^{(n)} := I - A u^{(n)} \), for each subspace \( V_k \), we solve a local variational problem,

\[
\left\{ \begin{align*}
\Delta u_k^{(n+1)} & \in V_k, \\
a(\Delta u_k^{(n+1)}, v) & = r^{(n)}(v) \quad \forall v \in V_k.
\end{align*} \right.
\]

Notice that, formally, solution \( \Delta u_k^{(n+1)} \) and test function \( v \) should be replaced with their inclusions into \( V \) (the forms are defined on the whole space).

In the coefficient space, the same operations look as follows. Given \( n \)th iterate \( x^{(n)} \), and the corresponding residual \( r^{(n)} = b - Ax^{(n)} \), increments \( \Delta x_k^{(n+1)} \) are computed as

\[
\Delta x_k^{(n+1)} = D_k^{-1} i_k^T r^{(n)}.
\]
Here, \( D_k \) denotes the diagonal subblock of global stiffness matrix \( A \) corresponding to basis functions that span subspace \( V_k \), and we use the same symbols \( t_k, k = 1, \ldots, M \) for the inclusions in the coefficient space. The matrix representation \( t_k \) of inclusion \( t_k \) is a Boolean matrix,

\[
(t_k)_{il} = \delta_{i,j(l)},
\]

(2.25)

The global increment \( \Delta x^{(n+1)} \) is then obtained by adding the local contributions:

\[
\Delta x^{(n+1)} = \sum_{k}^{M} t_k \Delta x_k^{(n+1)}.
\]

(2.26)

The corresponding notation in the function space is

\[
\Delta u^{(n+1)} = \sum_{k}^{M} t_k \Delta u_k^{(n+1)}.
\]

(2.27)

In summary, given \( n \)th residual \( r^{(n)} \), we compute the \((n+1)\)th correction as

\[
\Delta x^{(n+1)} = \sum_{k}^{M} t_k D_k^{-1} t_k^T r^{(n)}.
\]

(2.28)

with \( S \) denoting the (global) smoother. The final operations of updating the solution and residual reduce to

\[
x^{(n+1)} = x^{(n)} + S r^{(n)},
\]

\[
r^{(n+1)} = b - A x^{(n+1)} = b - A x^{(n)} - A S r^{(n)} = (I - AS) r^{(n)}.
\]

(2.29)

Operator \( B \) discussed in Section 2.2 corresponds to \( B = SA \), and it is self-adjoint and positive definite.

At this point, we would like to simplify our notation and drop the boldface symbols for the matrices and vectors in the coefficient space. All matrix operations can simultaneously be interpreted as operations on the solution in the original space \( V \) and residual in dual space \( V^* \).

Notice that, once the smoother is assembled into a global matrix, one iteration requires just two matrix–vector multiplies only.

### 2.4.1. Optimal relaxation

According to the abstract results discussed earlier, we modify the smoothing step by introducing the optimal relaxation parameter \( z \),

\[
x^{(n+1)} = x^{(n)} + z^{(n)} S r^{(n)},
\]

\[
r^{(n+1)} = (I - z^{(n)} A S) r^{(n)}.
\]

(2.30)

where relaxation parameter \( z^{(n)} \) solves the minimization problem,

\[
\| A^{-1} r^{(n+1)} \|_d = \| x - x^{(n+1)} \|_A \rightarrow \min.
\]

(2.31)

The parameter is given by the formula

\[
z^{(n)} = \frac{(r^{(n)}, S r^{(n)})_2}{(A S r^{(n)}, S r^{(n)})_2}.
\]

(2.32)

Recall again that this is equivalent to applying the steepest descent method to the original matrix preconditioned with the smoother.
2.5. Coarse grid correction

We assume that the coarse grid space \( V_0 \) is a subspace of \( V \) but it is specified in terms of different basis functions,

\[
V_0 = \text{span}\{g_i, i = 1, \ldots, N_0\}.
\]  
(2.33)

Each of the coarse grid basis functions \( g_i \) can be represented as a linear combination of the fine grid basis functions,

\[
g_i = \sum_{j=1}^{N_0} Q_{ij} e_j.
\]  
(2.34)

Consequently, for \( u \in V_0 \)

\[
u = \sum_{i=1}^{N_0} x_i g_i = \sum_{i=1}^{N_0} x_i \sum_{j=1}^{N} Q_{ij} e_j = \sum_{j=1}^{N} \left( \sum_{i=1}^{N_0} Q_{ij} x_i \right) e_j
\]  
(2.35)

and matrix \( Q \) defines the prolongation operator from the coarse grid coefficient space to the fine grid coefficient space, with its transpose defining the restriction operator.

Given an iterate \( x^{(n)} \) and the corresponding residual \( r^{(n)} \), we compute the corresponding coarse grid correction and update the residual in the following way:

\[
\Delta x^{(n+1/2)} = Q A_0^{-1} Q^T r^{(n)};
\]

\[
x^{(n+1/2)} = x^{(n)} + \Delta x^{(n+1/2)};
\]

\[
r^{(n+1/2)} = r^{(n)} - A \Delta x^{(n+1/2)}.
\]  
(2.36)

The coarse grid projection operator discussed in Section 2.2 is \( P_0 = QA_0^{-1}Q^T A \).

2.6. The two-grid algorithm

In summary, the two-grid iteration includes the following steps. Given current solution \( x \), and residual \( r \),

(1) restrict the residual to the coarse grid dual space,

\[
r_0 = Q^T r;
\]  
(2.37)

(2) solve the coarse grid problem for coarse grid correction \( \Delta x_0 \),

\[
A_0 \Delta x_0 = r_0;
\]  
(2.38)

(3) prolong the coarse grid correction to the fine grid space, and compute the corresponding correction for the residual,

\[
\Delta x = QA_0^{-1}Q^T \Delta x_0, \quad \Delta r = A \Delta x;
\]  
(2.39)

(4) update the fine grid solution and residual,

\[
x = x + \Delta x, \quad r = r - \Delta r;
\]  
(2.40)

(5) compute the smoothing correction and the corresponding correction for the residual,

\[
\Delta x = S r, \quad \Delta r = A \Delta x;
\]  
(2.41)
(6) compute the optimal relaxation parameter, \( \alpha \),

\[
\alpha = \frac{(r, Sr)}{(A Sr, Sr)} = \frac{(r, \Delta x)}{(A \Delta x, \Delta x)} = \frac{(r, \Delta r)}{(A \Delta r, \Delta r)};
\]  

(7) update the solution and residual,

\[
x = x + \alpha \Delta x, \\
r = r - \alpha \Delta r.
\]

2.7. Stopping criterion

Choosing an adequate stopping criterion is one of the most sensitive issues for all iterative methods. Ideally, we wish to stop the iterations when the difference between the unknown solution \( x \) and \( n \)th iterate reaches a prescribed tolerance,

\[
\|x - x^{(n)}\|_d \leq \varepsilon_{\text{TOL}}.
\]

As

\[
x - x^{(n)} = A^{-1}(b - Ax^{(n)}) = A^{-1}r^{(n)},
\]

the ideal stopping criterion translates into condition,

\[
\|A^{-1}r^{(n)}\|_d = (AA^{-1}r^{(n)}, A^{-1}r^{(n)})^{1/2} = (r^{(n)})^T A^{-1}r^{(n)})^{1/2} \leq \varepsilon_{\text{TOL}}.
\]

Obviously, this quantity is not computable, and a stopping criterion can only be based on an approximation to it.

3. Implementation

This section discusses several implementation issues, such as assembling, sparse storage, matrix–vector multiplication algorithms, selection of blocks for the block Jacobi smoother, and construction of the prolongation operator. The issues of element-by-element computations and matrix–vector multiplication in context of parallel implementations have been studied by Carey et al. in [9,4,20].

3.1. To assemble or not to assemble

A standard finite element code generates a number of dense element stiffness matrices. These matrices can be stored in an element-by-element fashion, or they can be assembled in a global stiffness matrix.

The main advantages of assembling the global stiffness matrix are:

(1) Considerable savings in storage for low order \( p \) meshes. Storing the (unassembled) element matrices results in increased memory requirements, due to the repetition of nodes shared by neighboring elements. The amount of extra storage becomes significant in 3D but it decreases with increasing order \( p \). We illustrate this assertion with a grid composed of \( 4 \times 4 \times 4 \) brick elements shown in Fig. 1. For uniform order of approximation \( p \), Table 1 compares the number of non-zero entries in a globally assembled stiffness matrix against the number of non-zero entries in the corresponding set of element stiffness matrices.
The logical complexity of manipulations with the assembled matrix is lower than that for the unassembled element matrices. Matrix–vector multiplication in the element fashion requires two extra subroutines: one that extracts element vectors from a global vector, and another one, that assembles back the element vectors into a global vector.

On the other hand, the main disadvantages of assembling are:

1. Assembling the global matrix requires a sparse storage format. A global (assembled) stiffness matrix is sparse. Therefore, we should store it by using a sparse storage pattern. In $h$-adaptive codes, this is usually implemented by using a bandwidth storage pattern. For $hp$-adaptive elements, the bandwidth depends not only upon the topology of the mesh but order of approximation as well, and it is determined by the elements of highest order in the mesh. This would result in prohibitive storage requirements.
requirements. Using the example of Fig. 1, if we take the bandwidth corresponding to \( p = 4 \), with a majority of elements, however, being of first order only, we are storing over 40,000\% more entries than necessary! Motivated with these simple observations, we have decided to use the compressed column storage (CCS) pattern [5], also known as Harwell–Boeing sparse matrix format [17]. As we will see in Section 3.2, using the sparse pattern storage requires an extra amount of memory, and affects performance of matrix–vector operations.

(2) *The parallelization becomes more challenging.* A standard parallel finite element code is based on assigning a number of elements to each processor. Therefore, it is more natural to assign element matrices to each processor as opposed to a partitioning of the global matrix between processors. Nevertheless, a partitioning of the global stiffness matrix (or partial assembling) can be performed and efficient parallel linear algebra libraries can be utilized [25].

In summary, the decision whether to assemble or not seems to be mostly a function of personal preference. In our case, we have decided to use the CCS pattern and assemble the global matrix.

3.2. CCS pattern for storing matrices

The CCS pattern requires the storage of three vectors: the first one stores the values of the non-zero entries; the second one saves the row numbers of the non-zero entries; and the third one contains the number of non-zero entries per column. This information determines completely a matrix. Thus, a Fortran 90 object CCS matrix can be defined as follows:

- **Object:** type CCSMatrix
  - integer :: idimcolumn. Integer with the dimension number of columns.
  - integer :: idimrow. Integer with the dimension number of rows.
  - integer :: non-zero. Integer with the number of non-zero entries in the matrix.
  - double precision, dimension (:), pointer :: value. Array of dimension non-zero with the values of the non-zero entries of the matrix.
  - integer, dimension (:), pointer :: nrrow. Array of dimension non-zero such that \( \text{nrrow}(i) = \text{row where value}(i) \) is located.
  - integer, dimension (:), pointer :: nrcolumn. Array of dimension number of columns + 1 such that \( \text{value}(\text{nrcolumn}(i)) \) is the first non-zero entry on the column \( i \). Note that \( \text{nrcolumn} \text{(number of columns + 1)} = \text{non-zero} + 1 \).

In order to fully understand the CCS pattern, we illustrate it with an example (see Fig. 2).

![Fig. 2. Example of a 5×6 matrix stored in the CCS fashion.](image-url)
We discuss now the necessary matrix vector operations:

- **Algorithm 1: CCSMatrixVector**
  - **Description**: This algorithm describes the multiplication of a matrix stored in the CCS format times a vector.
  - **Input**: CCSMatrix, Vectorinput.
  - **Output**: Vectoroutput.
  - **Implementation**:
    
    ```
    idimrow = CCSMatrix%idimrow
    idimcolumn = CCSMatrix%idimcolumn
    Vectoroutput(1:idimrow) = 0
    do icolumn = 1,idimcolumn
       alpha = Vectorinput(icolumn)
       do irow = nrrow(icolumn),nrrow(icolumn + 1) - 1
          Vectoroutput(irow) = Vectoroutput(irow) + alpha * CCSMatrix%value(irow)
       enddo
    enddo
    ```

- **Algorithm 2: CCSTMatrixVector**
  - **Description**: This algorithm describes the multiplication of the transpose of a matrix stored in the CCS format times a vector.
  - **Input**: CCSMatrix, Vectorinput.
  - **Output**: Vectoroutput.
  - **Implementation**:
    
    ```
    idimcolumn = CCSMatrix%idimcolumn
    do icolumn = 1,idimcolumn
       do irow = nrrow(icolumn),nrrow(icolumn + 1) - 1
          alpha = alpha + Vectorinput(irow) * CCSMatrix%value(irow)
       enddo
       Vectoroutput(icolumn) = alpha
    enddo
    ```

Both the memory requirements and the performance of these matrix vector operations can be improved in several ways:

- By taking advantage of the *logical* symmetry of the stiffness matrix. Although the stiffness matrix may not be symmetric, the logical structure (position of non-zero entries) is symmetric, since it does not depend upon the particular bilinear form, but upon the topology of mesh only.
- By using a block CCS pattern, see [5].
- By calling BLAS [8] or other specialized linear algebra subroutines.

### 3.3. Assembling of the stiffness matrix

We present now the assembling algorithm for the stiffness matrix:

- **Algorithm: CCSAssembling**
  - **Description**: This algorithm describes the assembling of element matrices into one assembled stiffness matrix stored in the CCS format.
- **Input**: Set of element stiffness matrices.
- **Output**: CCSMatrix.
- **High Level Implementation**:

  We construct a global denumeration for element d.o.f. We use the concept of the natural ordering of nodes (and the corresponding d.o.f.) discussed, e.g., in [10]. The goal is to determine, for each node \( \text{nod} \), number \( n_{\text{first}}(\text{nod}) \) (in the global ordering) of the first d.o.f. associated with the node. Using the element nodal connectivities, we create for each node a list of interacting nodes. Notice that element nodes include both regular nodes and parent nodes of the element constrained nodes, comp. the notion of modified element discussed in [10].

  Using the nodal interactions and the global denumeration for element d.o.f., we save the number of non-zero entries per column. We also utilize this information to determine the row position for each non-zero entry.

  We reorder the row positions for each column according to the global ordering of d.o.f. This step simplifies the final assembling of the non-zero entries.

  We generate the entries of the global stiffness matrix by following the next steps:

  ```plaintext
  Loop through elements
  Compute the modified element matrix
  Loop through the (modified) element nodes
    Establish the bijection between local and global d.o.f.
    Loop 1 through local d.o.f.
      Assemble (accumulate) for the load vector
      Loop 2 through local d.o.f.
      Assemble (accumulate) for the stiffness matrix
  enddo
  enddo
  enddo
  ```

  3.4. Construction of the smoother matrix

  We assemble the smoother into a global matrix. The necessary steps are as follows:

  (1) **Determine the blocks.** Typically, we define blocks by considering d.o.f. corresponding to all basis functions whose supports are contained within already specified patches of elements. The corresponding subspace \( V_k \) (see Section 2) can then be identified not only as a span of the selected basis functions but also as a subspace of all FE functions that vanish outside the patch. This yields a definition of space \( V_k \) that is independent of the choice of shape functions and, consequently, simplifies the convergence analysis. The first two of our choices follow this idea. A block corresponds to the span of all fine grid basis functions whose supports are contained within

  - the support of a coarse grid vertex basis function, or
  - the support of a fine grid vertex basis functions.

  In our third choice the blocks are defined by considering all d.o.f. corresponding to a particular (modified) element. The corresponding space \( V_k \) does depend upon the way the basis functions extend into neighboring elements and it is not independent of the selection of basis functions. In each of the discussed cases, a block is specified by listing all nodes contributing to it.

  Advantages and disadvantages of each of the corresponding smoothers are discussed in Section 4.4.
(2) Extract each block from the assembled stiffness matrix. This operation requires the construction, for each block, of the corresponding list of global d.o.f. This is done by using the list of block nodes discussed above, and the global array n_first(nod) determined earlier.

(3) Invert each block. Since the order of approximation \( p \) in our code cannot exceed a rather small number \( (p = 9) \), the size of the blocks is relatively small. For example, for blocks defined by element basis functions, the size of the blocks is equal to \((p + 1)^{\text{dimension}}\). The cost of inverting the block matrices is negligible in comparison with the overall cost of the solver.

(4) Assemble the global smoother. We use the same logic as for assembling the global stiffness matrix, and assemble the inverted block matrices, back into a global CCS matrix. In the case of the third choice of patches, the stiffness matrix and the smoother not only share the same logic of assembling, but also the same CCS integer arrays.

3.5. The prolongation (restriction) matrix

Determination of the prolongation matrix reduces to determining all coarse grid basis functions in terms of fine grid basis functions, comp. (2.35). This corresponds exactly to the initiation of new d.o.f. during \( h \)-refinements, and it is related to the constrained approximation technique. Consequently, we generate the prolongation matrix coefficients, node by node, as we refine coarse mesh elements. The only technicality comes from the fact that, during the global \( hp \)-refinement, some intermediate nodes may appear, i.e., parent nodes of fine mesh nodes may not necessary belong to the coarse mesh themselves, but they may result from an earlier refinement of the coarse grid nodes. The d.o.f. corresponding to the intermediate nodes are eliminated using a logic similar to that used for implementing multiply constrained nodes [11].

4. Numerical experiments

This section is devoted to an experimental study of convergence and performance of our two-grid solver. The study will be based on six model elliptic problems that we will introduce shortly. Using these examples, we will try to address the following issues:

- importance of a particular selection of shape functions,
- importance of the relaxation parameter,
- influence of the selection of blocks for the block Jacobi smoother,
- possible use of an averaging operator,
- error estimation for the two-grid solver,
- difference between performance of the two-grid solver versus smoothing operations only,
- efficiency and scalability of the two-grid solver, and
- the possibility of guiding the optimal \( hp \)-refinements with a partially converged solution.

4.1. Examples

We shall work with six examples of elliptic boundary-value problems. For each model problem, we describe the geometry, governing equations, material coefficients, and boundary conditions. We also display the exact or approximate solution, and we briefly explain the relevance of each problem in this research.

4.1.1. L-shape domain

Geometry: see Fig. 3.

Governing equation: Laplace equation \((-\Delta u = 0)\).
4.1.2. 2D shock problem

**Geometry**: unit square \(([0,1]^2)\). See Fig. 4.

**Governing equation**: Laplace equation \((-\Delta u = 0)\).

**Boundary conditions**: see Fig. 4 (N—Neumann, D—Dirichlet).

**Exact solution**: \(u = \arctan[60(r - 1)]\), where \(r = \sqrt{(x - 1.25)^2 + (y + 0.25)^2}\). Solution is displayed in Fig. 4.

**Observations**: this problem is regular (smooth), but it incorporates an internal layer. Thus, from the numerical point of view, it shows a dual behavior: in the preasymptotic range, the problem is close to a singular problem, while in the asymptotic range, the problem is smooth.
4.1.3. Isotropic heat conduction in a thermal battery

Geometry: see Fig. 5.

Governing equation: $\nabla (K \nabla u) = f^{(k)}$.

Material coefficients: $K = K^{(k)} = K^{(k)}_x$. For each material $k$, we define

$$f^{(k)} = \begin{cases} 
0, & k = 1, \\
1, & k = 2, \\
1, & k = 3, \\
0, & k = 4, \\
0, & k = 5,
\end{cases} \quad K^{(k)}_x = \begin{cases} 
25, & k = 1, \\
7, & k = 2, \\
5, & k = 3, \\
0.2, & k = 4, \\
0.05, & k = 5.
\end{cases}$$

Boundary conditions: we order each of the four parts of the boundary clockwise, starting with the left-hand side boundary. Then, we impose the following boundary conditions:

$$K \nabla u \cdot n = g^{(i)} - \varphi^{(i)} u,$$

where

$$\varphi^{(i)} = \begin{cases} 
0, & i = 1, \\
1, & i = 2, \\
2, & i = 3, \\
3, & i = 4,
\end{cases} \quad g^{(i)} = \begin{cases} 
0, & i = 1, \\
3, & i = 2, \\
2, & i = 3, \\
0, & i = 4.
\end{cases}$$

Exact solution: the exact solution is unknown. An FE approximation to the solution is displayed in Fig. 5.

Observations: this is a Sandia\textsuperscript{3} benchmark problem, in which we solve the heat equation in a thermal battery with large jumps in the material coefficients.\textsuperscript{4}

4.1.4. Orthotropic heat conduction in a thermal battery

Geometry: see Fig. 6.

Governing equation: $\nabla (K \nabla u) = f^{(k)}$.

Material coefficients:

$$K = K^{(k)} = \begin{bmatrix} K^{(k)}_x & 0 \\ 0 & K^{(k)}_y \end{bmatrix}.$$ 

For each material $k$, we define

$$f^{(k)} = \begin{cases} 
0, & k = 1, \\
1, & k = 2, \\
1, & k = 3, \\
0, & k = 4, \\
0, & k = 5,
\end{cases} \quad K^{(k)}_x = \begin{cases} 
25, & k = 1, \\
7, & k = 2, \\
5, & k = 3, \\
0.2, & k = 4, \\
0.05, & k = 5,
\end{cases} \quad K^{(k)}_y = \begin{cases} 
0.8, & k = 2, \\
0.0001, & k = 3, \\
0.2, & k = 4, \\
0.05, & k = 5.
\end{cases}$$

Boundary conditions: we order each of the four parts of the boundary clockwise, starting with the left-hand side boundary. Then, we impose the following boundary conditions:

\textsuperscript{3} Sandia National Laboratories, USA.
\textsuperscript{4} We thank Dr. Babuška and Strouboulis for the example.
\[ \mathbf{K} \nabla u \cdot n = g^{(i)} - x^{(i)} u, \]

where

\[ x^{(i)} = \begin{cases} 0, & i = 1, \\ 1, & i = 2, \\ 2, & i = 3, \\ 3, & i = 4, \end{cases} \quad g^{(i)} = \begin{cases} 0, & i = 1, \\ 3, & i = 2, \\ 2, & i = 3, \\ 0, & i = 4. \end{cases} \]

**Exact solution:** the exact solution is unknown. An FE approximation to the solution is displayed in Fig. 6.

Fig. 5. Geometry and FE solution of the isotropic heat conduction problem in a thermal battery.

Fig. 6. Geometry and FE solution of the orthotropic heat conduction problem in a thermal battery.
Observations: this is a Sandia\textsuperscript{3} benchmark problem, in which we solve the heat equation in a thermal battery with large and orthotropic jumps in the material coefficients (up to six orders of magnitude).

4.1.5. 3D shock problem

Geometry: unit cube ([0,1]\textsuperscript{3}). See Fig. 7.

Governing equation: Laplace equation (−Δu = 0).

Boundary conditions: Dirichlet.

Exact solution: \( u = a \tan(20r - \sqrt{3}) \), where \( r = \sqrt{(x - .25)^2 + (y - .25)^2 + (z - .25)^2} \). The solution is displayed in Fig. 7.

Observations: this 3D problem is regular (smooth), but it incorporates an internal layer. Thus, from the numerical point of view, it shows a dual behavior: in the preasymptotic range, the problem is close to a singular problem, while in the asymptotic range, the problem is smooth.

4.1.6. The Fichera problem

Geometry: see Fig. 8.

Governing equation: Laplace equation (−Δu = 0).

Boundary conditions: Homogeneous Dirichlet and Neumann.

Exact solution: The exact solution is unknown. An FE approximation to the solution is displayed in Fig. 8.

Fig. 7. Geometry and exact solution of the 3D shock problem.

Fig. 8. Geometry and exact solution of the Fichera problem.
Observations: This classical problem is a generalization of the L-shape domain problem to three dimensions. It incorporates a corner singularity at the origin, and three edge singularities located on the edges adjacent to the singular corner.

4.2. Sensitivity of the solution with respect to the selection of shape functions

Before we endeavor with the iterative solution matters, we study the behavior of direct solvers and their sensitivity to conditioning of the stiffness matrix corresponding to various choices of the element shape functions. This is to assure our confidence in using the direct solver

- to test the iterative solver, and
- to solve the coarse grid problem.

Three different solvers have been tested and compared with each other. A frontal solver with no pivoting,5 SuperLU [16], and a dense solver provided by LAPACK (subroutine dgetrs [2]).

Two different sets of shape functions have been used, the standard Peano shape functions, and integrated Legendre polynomials known also as Lobatto polynomials [10].

The 2D shock problem and the L-shape domain problems were used for testing. In Fig. 9, we display the difference (measured in the relative energy norm) between solutions obtained using the frontal solver and superLU for both choices of shape functions. Similar results were obtained by comparing LAPACK against superLU or LAPACK against the frontal solver.

We observe that Peano shape functions produce a very poorly conditioned stiffness matrix and, consequently, they result in a rather unstable behavior of the solvers. This is avoided by switching to Lobatto shape functions and, consequently, we have every reason to believe that the solutions coming out from either of the three direct solvers provide a reliable basis for studying the convergence of the iterative solver.

4.3. Importance of relaxation

We study the effect of using the relaxation parameter on a collection of (initial) uniform \( hp \)-grids for the L-shape domain problem and Fichera problem, varying the number of elements and their order of approximation. We have used the third choice of blocks (spans of element basis functions), and we focused on studying the smoothing operation only. We compare the block Jacobi iterations for different fixed values of the relaxation parameter with the optimal relaxation. Recall that using the optimal relaxation is equivalent to ‘accelerating’ the standard block Jacobi (with no relaxation at all) with the SD method.

Figs. 10 and 11 summarize the results of the experiment in terms of the number of iterations required to attain a given (fixed) tolerance error, as a function of mesh size and order of approximation. From these plots we conclude the following:

- For a fixed relaxation parameter, whether the method converges or not, depends almost exclusively upon \( p \) (and not upon \( h \)).
- For a fixed relaxation parameter, convergence rate of the method (provided that the method converges) depends almost exclusively upon \( h \) (and not upon \( p \)).
- The optimal relaxation guarantees faster convergence than any fixed relaxation parameter.

---

5 The solver was developed by Prof. E. Becker in the beginning of 1980s, and has been in use at ICES for over 20 years [6].
Fig. 9. Difference between the frontal solver and superLU solutions, measured in the relative (with respect to the energy norm of the solution) energy norm for the L-shape domain (left) and 2D shock (right) problems.

Fig. 10. L-shape domain problem. Convergence of the smoother using different relaxation parameters.
In summary, without the optimal relaxation parameter, the method diverges for sufficiently large order of approximation $p$. The condition number of the stiffness matrix preconditioned with the block Jacobi smoother seems to be independent of order $p$ but it does depend upon the mesh size. These observations seem to be consistent with the results on preconditioning of $hp$-methods, see e.g. [1].

4.4. Selection of patches for the block Jacobi smoother

We use the 2D shock problem, and a 11,837 d.o.f. mesh to study the effect of selecting different blocks for the block Jacobi and the two-grid iterations. Fig. 12 presents convergence results for the three choices discussed in Section 3.4, using smoothing only, whereas Fig. 13 shows the convergence results for the same mesh and choice of smoother, for the two-grid solver. All three smoothers included the optimal relaxation.

As expected, the convergence accelerates with the size of the block. The smoother based on the coarse grid vertex patches performs the best, and the smoother based on the fine grid vertex patches outperforms the smoother based on the spans of the element basis functions. It is interesting to see how the coarse grid corrections improve dramatically the performance of the second smoother based on the fine grid vertex nodes patches.

![Fig. 11. Fichera problem. Convergence of the smoother using different relaxation parameters.](image-url)
The improved performance of the smoothers based on the larger blocks is overshadowed with the dramatic increase of the storage. Compared with the last smoother, the first one asks on average for 16 times more memory (in 2D), and the second one for 4 times more memory (in 2D). The memory needed to store the third smoother is exactly the same as for the stiffness matrix. Moreover, their logical structure and assembling is identical. The increase in size of the smoother results not only in the increased memory

Fig. 12. L-shape domain problem. Convergence of the block Jacobi iterations using different blocks.

Fig. 13. L-shape domain problem. Convergence of the two-grid solver using different blocks for smoothing.
demands but also affects the CPU time for the matrix vector multiplies, in proportion to the memory increase. Thus, it is a misunderstanding to judge the quality of the smoothers based on the number of iterations only. As the storage requirements become critical for solving large problems, the better convergence rates of the first two smoothers do not outweigh the dramatically bigger storage requirements and comparable execution time,\footnote{At least for convergence in the range of .01–.001 relative error, which is the range of interest.} and we pick the last smoother as our choice.

4.5. Effect of an averaging operator

During construction of an overlapping block Jacobi smoother, it is possible to build into the formulation an extra operator that averages the contribution of each basis function. More precisely, we define

$$S_C = \sum_{k} C_{ik} D^{-1} i_k^T,$$

where $C$ is the averaging matrix given by $C_{ij} = \delta_{ij} n_i$ with $n_i$ equal to the number of blocks that $i$th basis function $e_i$ is contributing to. In particular, matrix $C$ is equal to the identity matrix if and only if the block Jacobi smoother is non-overlapping.

Adding the averaging operator certainly complicates the theoretical analysis, as corresponding smoother $S_C$ is not longer self-adjoint (in energy product). On the other hand, introducing the averaging operator has been motivated by related results for a posteriori error estimation based on partition of unity approach (the averaging can be interpreted as a discrete realization of a partition of unity) and, so to say, general intuition based on old iterative methods for solving structural mechanics problems.

It came as a rather big surprise for us that the averaging seems to have no positive effect on the convergence. Figs. 14 and 15 present the number of iterations required by the two-grid solver for all $hp$-meshes produced by the adaptive strategy for the L-shape domain and orthotropic heat conduction problems. In fact, for the case of anisotropic material data, the averaging increases significantly the number of iterations.

Consequently, the averaging operator seems to complicate the formulation of the algorithm, while no improvement on the convergence is obtained. As a consequence, we have decided to not include any averaging operator in the formulation of our two-grid solver.

4.6. Error estimation

In the following, we focus on error estimation and selection of the stopping criterion discussed in Section 2.7.

We have two sources of error:

1. the discretization error, representing the difference between the exact and (fully converged) discrete solutions, $\|x - x_{h,p}\|_A$; and
2. the iterative solver error $\|x^{(n)}_{h,p} - x_{h,p}\|_A$.

It is usually the case that

$$0.01 \leq \frac{\|x - x_{h/2,p+1}\|_A}{\|x - x_{h,p}\|_A} \leq 0.1.$$  

(4.47)
Therefore, it makes sense to impose the following condition for approximation \( x_{h/2,p+1}^{(n)} \) to the solution of the discretized problem:

\[
0.01 \leq \frac{\| x_{h/2,p+1}^{(n)} - x_{h/2,p+1}^{(0)} \|_A}{\| x_{h,p} - x_{h/2,p+1}^{(0)} \|_A} \leq 0.1.
\]  

(4.48)

We always start the two-grid iterations with the coarse grid solution prolongated to the fine grid. Thus, our last equation becomes

\[
0.01 \leq \frac{\| x_{h/2,p+1}^{(n)} - x_{h/2,p+1}^{(0)} \|_A}{\| x_{h/2,p+1}^{(0)} - x_{h/2,p+1}^{(0)} \|_A} \leq 0.1.
\]  

(4.49)

Fig. 14. L-shape domain problem. Number of iterations for the two-grid solver with (solid curve) and without (dashed curve) averaging, for a tolerance error of 0.001.

Fig. 15. Orthotropic heat conduction. Number of iterations for the two-grid solver with (solid curve) and without (dashed curve) averaging, for a tolerance error of 0.001.
The upper bound indicates the maximum acceptable error, while the lower bound points to the value below which the discretization error will dominate the convergence error by an order of magnitude, and further iterations make little sense.

As discussed in Section 2.7, the last equation is equivalent to

$$0.01 \leq \frac{\|A^{-1}r^{(n)}\|_d}{\|A^{-1}r^{(0)}\|_d} \leq 0.1.$$  \hfill (4.50)

Since $A^{-1}$ is not available, we replace it with the smoother $x^{(n)}S$,

$$0.01 \leq \frac{\|x^{(n)}Sr^{(n)}\|_d}{\|x^{(0)}Sr^{(0)}\|_d} \leq 0.1.$$  \hfill (4.51)

Notice that due to the very special choice of relaxation parameter $x^{(n)}$, both of the approximations below lead to the same value:

$$(x^{(n)}Sr^{(n)}, r^{(n)})_2 = (x^{(n)}ASr^{(n)}, x^{(n)}Sr^{(n)})_2.$$  \hfill (4.52)

At each step $n$, we can estimate the error corresponding to step $n - 1$ without performing any additional matrix–vector operations.

Figs. 16–19 show the accuracy of this estimate for different $hp$-grids corresponding to both the L-shape domain and the 2D shock problems. We have solved each problem twice: using our third block Jacobi smoother (smoothing only), and the two-grid iterations with the same smoother.

The error estimate is significantly more accurate when we use the two-grid solver. Unfortunately, in both cases, the error estimate may lose accuracy for large grids. As a consequence, we will construct a second error estimator for our two-grid solver algorithm.

First, we define the sequence $e^{n+1} = (I - x^n SA)e^n$. Then, a simple calculation using the optimal relaxation parameter $x^n = (e^n, SAe^n)_d / \|SAe^n\|_d^2$ leads to

$$\frac{\|e^{n+1}\|_d^2}{\|e^n\|_d^2} = 1 - x^n (e^n, SAe^n)_d / \|e^n\|_d^2.$$  \hfill (4.53)

![Fig. 16. L-shape domain problem. Exact (dashed curve) versus estimated (solid curve) error for the block Jacobi (left) and two-grid (right) iterations for a 1889 d.o.f. mesh.](image-url)
Rearranging terms and taking the square root, we obtain

\[
(e^n, e^n)_{A} = \sqrt{1 - \frac{\|e^n+1\|_A^2}{\|e^n\|_A^2}}
\]  

(4.54)

Then

\[
\frac{\|e^n\|_A^2}{\|e^0\|_A^2} = \frac{\varphi^n(e^n, S Ae^n)_{A}}{\varphi^0(e^0, S Ae^0)_{A}} \sqrt{1 - \frac{\|e^1\|_A^2}{\|e^0\|_A^2}} = E^{(n)}(1) \sqrt{1 - f^2(0)} \sqrt{1 - f^2(n)},
\]  

(4.55)

Fig. 17. L-shape domain problem. Exact (dashed curve) versus estimated (solid curve) error for the block Jacobi (left) and two-grid (right) iterations for a 11,837 d.o.f. mesh.

Fig. 18. Shock problem. Exact (dashed curve) versus estimated (solid curve) error for the block Jacobi (left) and two-grid (right) iterations for a 2821 d.o.f. mesh.
where \( f(i) \) = \( \|e^{i+1}\|_{k} / \|e^{i}\|_{k} \), and \( E^{n}(1) \) is our previous error estimate given by
\[
E^{n}(1) = \frac{\|x^{(n)}Sr^{(n)}\|_{A}}{\|x^{(0)}Sr^{(0)}\|_{A}}.
\]
(4.56)

Clearly, \( f(i) \) cannot be calculated (unless the exact solution is known). Nevertheless, \( \frac{\sqrt{1-f(0)^{2}}}{\sqrt{1-f(m)^{2}}} \) can be approximated. Although a rigorous mathematical approximation could not be obtained by the author, numerical experiments indicated that an adequate approximation may be given by
\[
\frac{\sqrt{1-f(0)^{2}}}{\sqrt{1-f(n)^{2}}} \approx \frac{\sqrt{1 - \left( \frac{g(1) + g(0)}{2} \right)^{2}}}{\sqrt{1 - \left( \frac{g(n) + g(n-1)}{2} \right)^{2}}},
\]
(4.57)

where \( g(n) = \frac{E^{n}(1)}{E^{n-1}(1)} \). Thus, we define our second error estimate \( E^{n}(2) \) as
\[
E^{n}(2) = E^{n}(1) \cdot \frac{\sqrt{1 - \left( \frac{g(1) + g(0)}{2} \right)^{2}}}{\sqrt{1 - \left( \frac{g(n) + g(n-1)}{2} \right)^{2}}},
\]
(4.58)

**Remark.** The correcting factor in (4.58) depends upon the rate of convergence and not error itself. The original error estimate underestimates the error but provides an accurate estimate of the rate. Hence, replacing in (4.58) the exact rate with the estimated rate leads to a superior error estimate.

Figs. 20–23 compare the accuracy of both error estimates (\( E^{n}(1) \) and \( E^{n}(2) \)) for different \( hp \)-grids corresponding to the L-shape domain, the 2D shock problem, the orthotropic heat conduction, and the Fichera problems. \( E^{n}(2) \) seems to be considerably more accurate than \( E^{n}(1) \).
In Section 4.9, we will try to adjust the admissible tolerance error constant to make it as large as possible, without affecting the overall exponential convergence property of the whole $hp$-algorithm.

4.7. Smoothing versus two-grid solver

In this section, we want to compare the convergence of the block Jacobi smoother with the convergence of the two-grid iterations. We shall also study the effect of coupling the coarse grid solve with more than one smoothing iterations. Again, we shall use only our third smoother (based on spans of element basis

---

Fig. 20. L-shape domain problem. Exact (dashed curve) versus two estimated (dotted curve, $E(1)$ and solid curve, $E(2)$) errors for the two-grid solver with a 1889 (left) and 11,837 (right) d.o.f. mesh.

Fig. 21. Shock problem. Exact (dashed curve) versus two estimated (dotted curve, $E(1)$ and solid curve, $E(2)$) errors for the two-grid solver with a 2821 (left) and 12,093 (right) d.o.f. mesh.

In Section 4.9, we will try to adjust the admissible tolerance error constant to make it as large as possible, without affecting the overall exponential convergence property of the whole $hp$-algorithm.

4.7. Smoothing versus two-grid solver

In this section, we want to compare the convergence of the block Jacobi smoother with the convergence of the two-grid iterations. We shall also study the effect of coupling the coarse grid solve with more than one smoothing iterations. Again, we shall use only our third smoother (based on spans of element basis
functions). The study will be performed for the L-shape domain and the 2D shock problems, on several hp-grids resulting from the automatic hp-refinement strategy.

We solve each (fine grid) problem three times using different methods:

- one smoothing iteration followed by the coarse grid solve (1-1) (our standard approach),
- three smoothing iterations followed by the coarse grid solve (3-1), and
- smoothing iterations only.

Results of the computations are displayed in Table 2. As expected, the two-grid solver behaves significantly better than the smoothing iterations only. Increasing the number of smoothing steps in the algorithm does not improve the convergence, in fact, we observe a (very) slight deterioration.

Fig. 22. Orthotropic heat conduction problem. Exact (dashed curve) versus two estimated (dotted curve, $E(1)$ and solid curve, $E(2)$) errors for the two-grid solver with a 561 (left) and 10,883 (right) d.o.f. mesh.

Fig. 23. Fichera problem. Exact (dashed curve) versus two estimated (dotted curve, $E(1)$ and solid curve, $E(2)$) errors for the two-grid solver with a 13,897 d.o.f. mesh.
We emphasize that in each of the reported cases, the iterations were stopped by monitoring the difference between the exact solution (obtained from a direct solver) and the current iterate. Table 3 presents results for the same meshes using smoother $A_{S}$ in place of $A^{-1}$ in the stopping criterion. The results for the ‘smoothing only’ case seem to be confusing as the number of iterations drops with the size of the problem. This is an effect of the deterioration of the error estimate discussed in Section 4.6.

### 4.8. Efficiency of the two-grid solver algorithm

In this section, we first present a simple scalability analysis for $hp$-multigrid solvers. Then, we illustrate numerically the efficiency of our two-grid solver.

It is well known that scalability of a two-grid solver for $h$-finite elements is given by

$$\text{Speed} = O(N_C^2) + O(N),$$

where $N$ and $N_C$ are the number of elements in the fine and coarse grids, respectively.

Unfortunately, for $hp$-finite elements, the situation becomes more difficult, since scalability of a two-grid solver is given by

$$\text{Speed} = O(N_C^2 p^6_C) + O(N p^9),$$

where $p$ and $p_C$ are the average order of approximation $p$ in the fine and coarse grids, respectively. Notice that average $p$ and $p_C$ should be computed in the right norm, that is, in $l_2$-norm for $p$, and in $l_6$-norm for $p_C$. The term $N p^9$ is associated to the cost of inverting patches corresponding to a block Jacobi smoother (if exact solves on each patch are performed).

It is clear from this simple scalability analysis that, in order to implement an efficient two-grid (or multigrid) solver for $hp$-methods, it is necessary either to control the maximum $p$, or to compute somehow a very inexpensive block Jacobi smoother. In our case, we selected $p = 9$ as our maximum allowable order of approximation, since our current $hp$-code already has this limitation. It turns out that for real world problems, $p > 9$ is (almost) never needed.

### Table 2

Number of iterations using exact solution in the stopping criterion with tolerance .01 and .001 (in parentheses)

<table>
<thead>
<tr>
<th>Example</th>
<th>No. of d.o.f.</th>
<th>1-1</th>
<th>3-1</th>
<th>Only smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-shape</td>
<td>1889</td>
<td>13 (24)</td>
<td>14 (25)</td>
<td>34 (96)</td>
</tr>
<tr>
<td>L-shape</td>
<td>11,837</td>
<td>12 (24)</td>
<td>13 (24)</td>
<td>18 (74)</td>
</tr>
<tr>
<td>Shock</td>
<td>2821</td>
<td>6 (11)</td>
<td>6 (11)</td>
<td>478 (732)</td>
</tr>
<tr>
<td>Shock</td>
<td>12,093</td>
<td>8 (19)</td>
<td>9 (20)</td>
<td>326 (908)</td>
</tr>
<tr>
<td>Shock</td>
<td>34,389</td>
<td>12 (28)</td>
<td>13 (30)</td>
<td>18 (257)</td>
</tr>
</tbody>
</table>

### Table 3

Number of iteration using error estimate as the stopping criterion

<table>
<thead>
<tr>
<th>Example</th>
<th>No. of d.o.f.</th>
<th>1-1</th>
<th>3-1</th>
<th>Only smoothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>L-shape</td>
<td>1889</td>
<td>11 (22)</td>
<td>12 (23)</td>
<td>14 (65)</td>
</tr>
<tr>
<td>L-shape</td>
<td>11,837</td>
<td>9 (21)</td>
<td>10 (22)</td>
<td>13 (35)</td>
</tr>
<tr>
<td>Shock</td>
<td>2821</td>
<td>6 (11)</td>
<td>6 (11)</td>
<td>295 (556)</td>
</tr>
<tr>
<td>Shock</td>
<td>12,093</td>
<td>7 (15)</td>
<td>7 (17)</td>
<td>9 (274)</td>
</tr>
<tr>
<td>Shock</td>
<td>34,389</td>
<td>9 (23)</td>
<td>10 (24)</td>
<td>12 (33)</td>
</tr>
</tbody>
</table>
We implemented fast integration rules, and we interfaced with the library LAPACK in order to invert patches for construction of the block Jacobi smoother.

In Fig. 24 we display the time used by the two-grid solver versus the number of d.o.f. for a sequence of optimal $hp$-grids produced by the fully automatic $hp$-adaptive strategy. As expected, the scalability of the method is not linear, which can be appreciated clearly by just looking at the patch inversion times.

In Fig. 25 we solved the 3D shock problem on a grid with order of approximation $p = 2$ and 2.15 million unknowns. The two-grid solver needed only 8 min to converge. As we increase $p$, the stiffness matrix

![Fig. 24. 3D shock problem. Efficiency of the two-grid solver for a sequence of optimal $hp$-grids produced by the fully automatic $hp$-adaptive strategy. In core computations, AMD Athlon 1 GHz processor.](image1)

![Fig. 25. 3D shock problem. The two-grid solver for a grid with 2.15 million d.o.f. and $p = 2$ requires 8 min to converge. 1.0 Gb of memory is needed to store only the non-zero entries of the stiffness matrix. In core computations, IBM Power4 1.3 GHz processor.](image2)

We implemented fast integration rules, and we interfaced with the library LAPACK in order to invert patches for construction of the block Jacobi smoother.

In Fig. 24 we display the time used by the two-grid solver versus the number of d.o.f. for a sequence of optimal $hp$-grids delivered by the fully automatic $hp$-adaptive strategy. As expected, the scalability of the method is not linear, which can be appreciated clearly by just looking at the patch inversion times.

In Fig. 25 we solved the 3D shock problem on a grid with order of approximation $p = 2$ and 2.15 million unknowns. The two-grid solver needed only 8 min to converge. As we increase $p$, the stiffness matrix

---

7 The author wants to acknowledge J. Kurtz for implementation of fast integration rules.
becomes more dense. As a result, for $p = 8$, we are able to solve in 10 min only 0.27 millions unknowns (see Fig. 26). Stiffness matrix contains several hundred million non-zero entries.

Finally, in Fig. 27 we solved the 3D shock problem on a grid with order of approximation $p = 4$ and 2.15 million unknowns. This is the largest problem of all, and the coarse grid solve time becomes dominant, as expected. Indeed, only 9 min are utilized by the remaining components of the two-grid solver. Thus, a full multigrid solver is necessary at this point.

4.9. Guiding $hp$-strategy with a partially converged solution

By now we hope that the reader has agreed with us on the need of using the relaxation, the choice of the smoother, and the necessity of using the two-grid iterations, not only because of faster convergence rates, but also because of the control of the error, i.e., the right stopping criterion.

![Fig. 26. 3D shock problem. The two-grid solver for a grid with 0.27 million d.o.f. and $p = 8$ requires 10 min to converge. 2.0 Gb of memory is needed to store only the non-zero entries of the stiffness matrix. In core computations, IBM Power4 1.3 GHz processor.](image1)

![Fig. 27. 3D shock problem. The two-grid solver for a grid with 2.15 million d.o.f. and $p = 4$ requires 50 min to converge. 3.5 Gb of memory is needed to store only the non-zero entries of the stiffness matrix. In core computations, IBM Power4 1.3 GHz processor.](image2)
We come now to the most crucial question addressed in this chapter. How much can we relax our convergence tolerance for the two-grid solver, without losing the exponential convergence in the overall $hp$-adaptive strategy? Obviously, this is a rather difficult question, and we will try to reach a conclusion via numerical experimentation only.

We will work this time with four examples: the L-shape domain, the two heat conduction problems, and the Fichera problem. For each of these problems we will run the $hp$-adaptive strategy using up to five different strategies to solve the fine grid problem:

1. a direct (frontal) solver,
2. two-grid solver with tolerance set to 0.001 (as described in Section 4.6),
3. two-grid solver with tolerance set to 0.01,
4. two-grid solver with tolerance set to 0.1, and
5. two-grid solver with tolerance set to 0.3.

Fig. 28. L-shape domain problem. Guiding $hp$-refinements with a partially converged solution.
For the L-shape domain problem we do have the exact solution, but for the remaining three problems we do not. So, we have to come up with some approximate ways to measure the error. We will use the following two measures.

(1) **An approximation of the error in the energy norm.** The exact relative error in the energy norm, is given by

\[
\frac{\|x_{h,p} - x_{\text{exact}}\|}{\|x_{\text{exact}}\|} = \sqrt{\frac{\|x_{h,p}\|^2 - \|x_{\text{exact}}\|^2}{\|x_{\text{exact}}\|^2}} .
\]

Thus, a good approximation to it is given by

\[
\frac{\sqrt{\|x_{h,p}\|^2 - \|x_{\text{exact}}\|^2}}{\|x_{\text{exact}}\|} \approx \sqrt{\frac{\|x_{h,p}\|^2 - \|x_{\text{best}}\|^2}{\|x_{\text{best}}\|^2}} .
\]

![Fig. 29. Isotropic heat conduction. Guiding hp-refinements with a partially converged solution.](image)
where $x_{\text{best}}$ is the best approximate solution that we can trust. We will use for $x_{\text{best}}$ the last coarse grid solution (obtained with the direct solver). Obviously, the measure cannot be trusted for meshes produced last.

(2) *An approximation of the error in $H^1$-norm.* We simply compute the $H^1$ norm of the difference between the coarse and (partially converged) fine mesh solutions. Notice that, if the fine mesh solution were fully converged then this would be a very reliable measure. The point is, however, that we are trying to relax the convergence tolerance, and in process of doing it, we also lose the confidence in this a posteriori error estimate.

Both errors will be reported relative to the norm of the fine grid solution, in percent.

Finally, for each of the cases under study, we will report the number of the two-grid iterations necessary to achieve the required tolerance.

![Graphs of relative error and number of iterations vs. number of DOF](image.png)

Fig. 30. Orthotropic heat conduction. Guiding $hp$-refinements with a partially converged solution.
From Figs. 28–31 we draw the following conclusions:

- The two-grid solver with 0.001 error tolerance generates a sequence of $hp$-grids that has identical convergence rates to the sequence of $hp$-grids obtained by using a direct solver.
- As we increase the two-grid solver error tolerance up to 0.3, the convergence rates of the corresponding sequence of $hp$-grids decreases only slightly (in some cases it does not decrease at all). However, the number of iterations decreases dramatically.
- The number of required iterations for our two-grid solver does not increase as the number of degrees of freedom increases.

To summarize, it looks safe to relax the error tolerance to 0.1 value, without loosing the exponential convergence rates of the overall $hp$-mesh optimization procedure.
5. Conclusions and future work

In this paper, we have studied a two-grid solver for solving linear systems resulting from FE $hp$-discretizations of self-adjoint elliptic PDE's in two and three space dimensions. The meshes come in pairs, of a coarse mesh and the corresponding fine mesh obtained via the global $hp$-refinement of the coarse mesh. The coarse meshes are generated by a special $hp$-adaptive algorithm, based on minimizing the projection based interpolation error of the fine mesh solution with respect to the next optimally refined coarse mesh. The solver combines block Jacobi smoothing with an optimal relaxation, with the coarse grid solve.

Instead of using the two-grid iteration for producing a preconditioner for conjugate gradient (CG) only, we chose to accelerate each smoothing operation individually with the SD method, which we interpret as determining the optimal relaxation parameter. We have proved that in the worst case scenario, the method guarantees convergence at least as good as the one corresponding to the SD preconditioned with the smoother and coarse grid solution coupled in the additive way.

Within the described framework, we have studied several critical questions including implementation issues, importance of the relaxation, selection of the blocks for the block Jacobi smoother, stopping criterion, scalability and, first of all, the possibility of guiding the $hp$-strategy with only partially converged solution. Let us emphasize again some of our main conclusions:

- We decided to assemble both the stiffness matrix and smoother into global matrices stored in the CCS pattern.
- We decided to use blocks defined by d.o.f. associated with (modified) elements.
- We verified that the use of the optimal relaxation parameter is necessary for convergence and yields optimal convergence rates in the full range of $hp$-meshes, when compared with fixed relaxation parameters.
- We verified that the use of the two-grid solver, as opposed to smoothing only, is essential in maintaining a fixed number of iterations (for a given tolerance), independently of the problem size. More importantly, we verified that the use of the two-grid solver is essential in controlling the convergence error.
- Finally, we verified that partially converged solution, with a rather large (relative) error tolerance of 0.1, is sufficient to guide the $hp$-strategy. The corresponding number of the two-grid iterations stays then very minimal at a level below 10 iterations per mesh.

A similar but yet different two-grid solver algorithm for electrodynamic problems has also been successfully implemented. The corresponding results will be presented in a forthcoming paper.

Acknowledgments

The work has been partially supported by Air Force under contract F49620-98-1-0255. The computations reported in this work were done through the National Science Foundation’s National Partnership for Advanced Computational Infrastructure.

References


