International Journal of Computing and Visualization in Science and Engineering Vol 1.4, 2021.
https://www.doi.org/10.51375/IJCVSE.2021.1.4

A Semi-Algebraic Two Level Solver

Randolph E. Bank · **Robert D. Falgout**

Received: 31 October 2019 / Accepted: 13 December 2019 / Published 4 March 2021

Abstract We develop a simple semi-algebraic 2-level solver built on traditional multigrid ideas. It is designed to be easily incorporated into existing simulation software. It exhibits good convergence for many classes of challenging problems including discontinuous diffusion, convection-diffusion, and Helmholtz equations. It has built-in structure that makes it simple to generalize in several interesting directions.

Keywords Two level solver, Quadtree, Octree.

Mathematics Subject Classification (2010) 65M55, 65F10

1 Introduction

Multilevel iterative methods for solving sparse linear systems of equations arising from discretizations of partial differential equations use a smoothing iteration in combination with a coarse space correction to achieve a method that quickly damps error components across the entire spectrum of eigenvalues of the system matrix A. A common scenario, especially in the algebraic multigrid (AMG) setting, is to select a relatively simple smoother such as Gauss-Seidel or damped Jacobi, and combine it with a sophisticated multilevel coarse space correction. See for example [14,12,10,19] and the references therein. In this work we take an alternative approach, where we choose a sophisticated smoother such as incomplete *LU* used in combination with a very simple coarse space correction, and then accelerate the resulting 2-level preconditioner with some Krylov method such as conjugate gradients. We hope that such an approach will simplify introducing multilevel approaches into existing simulation codes that lack multilevel solvers. One can use the already existing preconditioner as the smoother, and add our simple coarse space correction to create a solver that exhibits multilevel convergence rates.

We construct our coarse space correction using classical multilevel techniques. Thus in many ways there is nothing “new” about the algorithms that we employ, although their use in this particular context might be viewed as unusual. On the other hand, we do demonstrate that a very simple and inexpensive coarse space correction coupled with a sophisticated smoother can have a large positive impact on its convergence rate.

Construction of our coarse space correction matrix is semi-algebraic, requiring certain information beyond just the system matrix and right hand side. In particular we require

---

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-JRNL-795845).

Bank: Department of Mathematics, University of California, San Diego, La Jolla, California 92093-0112 USA. Email:rbank@ucsd.edu.
Falgout: Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, P. O. Box 808, L-561, Livermore, California 94551 USA. Email:falgout2@llnl.gov.

- (x,y) (2D) or (x,y,z) (3D) coordinates for each degree of freedom (DOF).
- Each class of DOFs needs to be labeled.
  - Different physical variables as in a PDE system (pressure, velocity, etc).
  - Different derivatives of the same variable, as in a mixed method (u, $\sigma = \nabla u$, etc).

Such information is likely to be easily available in typical simulation packages. For example, it is necessary input for a graphics package to be used to display images of the solution. Similar information is also used by the *HYPRE* software package [15,13,1,16] For simplicity in this work, we will restrict attention to the case of a single unknown function, as in the general case we can apply the same procedures to each individual variable.

Our method can be briefly summarized as follows. We embed the physical domain of the partial differential equation in a square (2D) or cube (3D). We create an adaptive quadtree (2D) or octree (3D) with approximately N_e or fewer unknowns in each leaf element. N_e is a user selected parameter. We then create a coarse space of dimension N_c using continuous piecewise bilinear (2D) or trilinear (3D) nodal basis functions as a guide. However this coarse space is generally not a space of piecewise bilinear or trilinear functions, but rather is a formal subspace of the approximation space used to discretize the underlying partial differential equation.

In Section 2, we present the quadtree and octree refinement algorithms. These were strongly influenced by the quadtree used in early versions of the *PLTMG* software package, and described in detail in [18,8]. In Section 3, we describe the construction of the restriction, prolongation and coarse space system matrices. In Section 4, we present a selection of examples in 2D and 3D that demonstrate the effectiveness of this approach. Finally, in Section 5, we make some concluding remarks. This includes a discussion of how this approach could be applied to systems of partial differential equations. Also, several remarks describe how the techniques here could be extended to create more sophisticated coarse space corrections, including multilevel solvers.

2 Quadtrees and Octrees

Quadtrees and octrees are widely used in adaptive mesh generation, computer graphics, and other areas of scientific computation. Multi-level solvers have been developed making use of the quadtree or octree level structure. The early versions of the *PLTMG* software package used a quadtree forest based on triangles (called “triangle tree” at that time) for both adaptive meshing and multilevel solver. Our brief summary below is largely based on this early work. See Weiser [18], and Bank, Sherman, and Weiser [8] for a more complete discussion.

Our setup phase employs a quadtree (2D) or octree (3D) to partition the degrees of freedom into blocks of approximate size N_e. To accomplish this, we embed all unknowns in a square (2D) or cube (3D) based on their coordinates. We then create an adaptive quadtree (2D) or octree (3D) by refinement until each leaf element in the tree contains at most approximately N_e degrees of freedom. N_e is a user specified parameter. In the case of a quadtree, a given square element is refined into four smaller squares in the usual way by connecting opposing pairs of midpoints. In the case of an octree, a cube is refined into eight smaller cubes in the usual way using three mutually perpendicular bisecting planes. We note that N_e is only a target, and some leaves may have more than N_e degrees of freedom; if a given element t has $N_t > N_e$ unknowns, it is not refined if $N_t/4 \ll N_e$ (2D) or $N_t/8 \ll N_e$ (3D).

We employ typical definitions of parent, child, and level. The original square or cube is the root of the tree. A given refined element t in the tree has 4 (2D) or 8 (3D) children. Typically the children appear as consecutive members of an ordered list of elements, so the first child is denoted by the pointer s_t; $s_t = 0$ for leaf elements. The parent of a given element t is denoted f_t, another pointer into the list of elements. $f_t = 0$ if t is the root element. The level of and element ℓ_t is defined inductively as follows. $\ell_t = 1$ if t is the root element. The children of an element t with $\ell_t = k$ all have level $k+1$.

A vertex v in the quadtree/octree mesh is called *regular* if it is a corner vertex of each unrefined element it touches; all other vertices are called *irregular* or *hanging nodes*.

For quadtree meshes, a given element t has 4 *neighbors* η_t^j, $1 \le j \le 4$. $\eta_t^j = 0$ if the corresponding edge is a on the boundary. Otherwise, η_t^j points at the smallest element with an edge that completely overlaps the given edge of t. Note this implies $\ell_{\eta_t^j} \le \ell_t$; thus the neighbor relation is not symmetric and is time dependent.

For octree meshes, there are two types of neighbors. First, there are 6 *face neighbors* η_t^j, $1 \le j \le 6$. Face neighbors are analogous to neighbors in the 2D case, $\eta_t^j = 0$ if face j of t is on the boundary. Otherwise, η_t^j is a pointer to the smallest cube in the octree with a face that completely overlaps face j of element t. Element t also has 12 *edge neighbors* $\hat{\eta}_t^j$, $1 \le j \le 12$. For each of the 12 edges of element t, generally there are 2 face neighbors of t that also share the given edge. Additionally there is a fourth element, besides t and the 2 face neighbors, that shares the given edge. As usual $\hat{\eta}_t^j = 0$ if edge j is on the boundary, and otherwise is a pointer to the fourth element sharing edge j. We remark that in the case of more general forests of less structured octrees it is possible to have more than one edge neighbor for a given edge. As in the case of 2D neighbors, the face and edge neighbors relations are not symmetric and are time dependent.

It is advantageous to sometimes refine additional elements in order for the quadtree or octree mesh to achieve certain properties. This was shown in [8] for the quadtree case, where we imposed the following additional refinement rules.

- **1-irregular rule.** Refine any unrefined element t that has more than one irregular vertex on any edge.
- **3-neighbor rule.** Refine any unrefined element t that has one irregular vertex on three or more of its edges.

These refinement rules are illustrated in Figure 1.

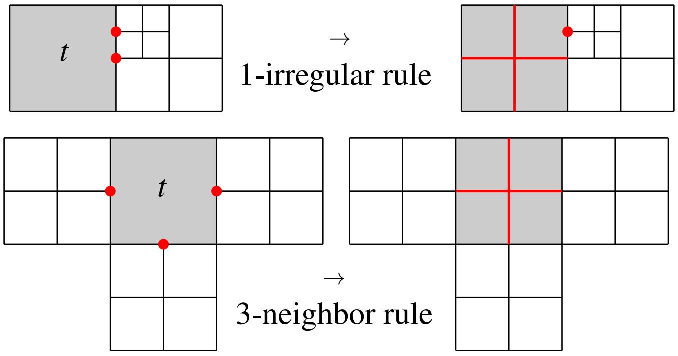


Fig. 1 Top: element t has two irregular vertices (marked in red) on one edge, and thus is refined due to the 1-irregular rule. Bottom: element t has one irregular vertex on three of its edges and thus is refined due to the 3-neighbor rule.

The 3-neighbor rule mainly addresses the issue of irregular nodes. In particular, if three edges of element t have irregular nodes, refining t changes them to regular nodes, adds a regular node at the center, but creates one new potentially irregular node on the fourth edge; however, if the fourth edge is a boundary edge, it will be a regular node. Thus there is a net reduction of at least 2 irregular nodes, and the addition of at least 4 regular nodes. If all edges of t contain irregular nodes, there is net reduction of 4 irregular nodes, and the addition of 5 regular nodes. In the case of continuous piecewise bilinear finite elements, regular nodes stand in correspondence with the degrees of freedom, and thus it is beneficial to have as few irregular nodes as possible.

In some sense the 1-irregular rule is more critical. For continuous piecewise bilinear finite elements, the 1-irregular rule guarantees that there are at most 4 nodal basis functions with support in any leaf element t, and the support of nodal basis functions centered on the regular nodes intersect the support of at most 12 other basis functions. This guarantees the sparsity of the finite element stiffness matrix and makes element assembly comparatively simple. Finally, the 1-irregular rule ensures that all nodes of non-leaf tree elements ($s_t \neq 0$) are regular. See [8] for further discussion of these points and for additional properties.

In the octree case we impose similar refinement rules:

- **1-irregular rule.** Refine any unrefined element t that has more than one irregular vertex on any edge, or more than one irregular vertex in the interior of any face.
- **4-neighbor rule.** Refine any unrefined element t that has one irregular vertex on four or more of its face interiors.

If 4 faces of element t contain irregular nodes, then 11 or 12 of the edges of element t also contain irregular nodes at their midpoints, depending on the arrangement of faces containing irregular nodes. Refining t changes 4 irregular face nodes to regular, adds a regular node at the center of t, and creates 2 potentially irregular nodes in face centers, and possibly 1 potentially irregular edge node. The result is a net gain of at least 2 regular nodes. Some of the 11 or 12 irregular edge nodes may also become regular as well, but this is not guaranteed, as this also depends on the state of the edge neighbors $\hat{\eta}_t^j$ of t. Refining elements with 5 or 6 irregular face nodes is even more advantageous.

Similar to the quadtree case, for continuous trilinear finite elements, the 1-irregular rule guarantees sparsity of the stiffness matrix, as well as 8 nodal basis functions with support in each leaf element. As in the quadtree case, all nodes of non-leaf elements are regular.

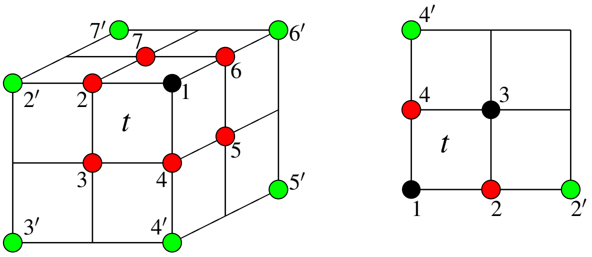


Fig. 2 A refined 3D cube (left) and 2D square (right). The colored vertices are relevant for child element t. Black vertices are known to be regular. The red vertices are possible irregular vertices, depending on the state of neighboring elements. The green vertices, all corners of the parent element and known to be regular, are possible “replacements” for red vertices that are irregular.

In Figure 2, we examine the construction of bilinear (2D) and trilinear (3D) nodal basis functions in a given leaf element t, assuming that the mesh satisfies the 1-irregular rule. We first consider the 2D case, illustrated in Figure 2 right. In this case the corner vertex v_1 shared with the parent element f_t and the center vertex v_3 are known to be regular. The red vertices v_2 and v_4 could be irregular, depending on the neighbor elements of t. Let ϕ_j denote the usual bilinear nodal basis function restricted to element t associated with vertex v_j. For simplicity, suppose that v_2 is irregular and v_4 is regular. Then ϕ_3 and ϕ_4 are just the usual nodal bilinear basis functions for t. ϕ_1 is modified and $\phi_{2'}$ replaces ϕ_2 as a basis function for t as follows.

$$\begin{aligned}
&\phi_{2'} \leftarrow \phi_2/2,\\
&\phi_1 \leftarrow \phi_1 + \phi_2/2.
\end{aligned}$$

If v_4 is irregular then ϕ_1 and $\phi_{4'}$ are modified in a similar fashion.

In the 3D case, the vertex v_1 shared with f_t and the center vertex v_8 (not displayed in Figure 2) are known to be regular. The other six vertices could be irregular, with three face center vertices (v_3, v_5, v_7) and three edge midpoints (v_2, v_4, v_6). Assume that face center v_3 is irregular; then all four edge midpoints on that face of f_t are irregular, including v_2 and v_4. For convenience we assume v_5, v_6, and v_7 are all regular. With the ϕ_j now representing trilinear nodal basis functions on t, then ϕ_j, $5 \leq j \leq 8$ remain unchanged, while the other four are modified as follows.

$$\begin{aligned}
&\phi_{3'} \leftarrow \phi_3/4,\\
&\phi_{2'} \leftarrow \phi_2/2 + \phi_3/4,\\
&\phi_{4'} \leftarrow \phi_4/2 + \phi_3/4,\\
&\phi_1 \leftarrow \phi_1 + \phi_2/2 + \phi_4/2 + \phi_3/4.
\end{aligned}$$

If other face center vertices are irregular, the usual nodal basis functions would be modified in an analogous fashion. Even if all three face centers are regular, it is still possible for an edge midpoint vertex to be irregular, based on the edge neighbor $\hat{\eta}_t^j$ of t along that edge. These cases are handled analogously to the 2D example described above.

As a final remark, imposing the 1-irregular and {3,4}-neighbor rules does not change the overall complexity of creating the quadtree or octree. In [8], a one pass algorithm is described and proven to create quadtree meshes in $O(T)$ work, where T is the total number of elements in the tree. As each element t is processed and evaluated for possible refinement due to the number of degrees of freedom N_t contained in the element relative to N_e, one examines the four neighbors of t (a fixed bounded number) for possible violations of the 1-irregular and 3-neighbor rules. Since all operation at this step are $O(1)$ the overall complexity is $O(T)$. Octree meshes are created using the same strategy.

3 Two Level Solver

Consider the linear system

$$Au = b \tag{1}$$

where A is the large sparse $N \times N$ stiffness matrix corresponding to the discretization of some partial differential equation. We begin the development of our 2-level preconditioner for (1) with the restriction matrix R in the 2D case.

We start with the quadtree mesh, and discard (ignore) any leaf elements that contain zero degrees of freedom from the fine mesh. The remaining mesh of leaf elements contains N_c regular nodes, and this becomes the order of our coarse grid space. Thus the restriction matrix R is $N_c \times N$, and is sparse, with exactly four nonzeroes in each column. Suppose the k-th fine grid degree of freedom corresponding to coordinates (x_k, y_k) is contained in the given leaf element. We evaluate the four nonzero bilinear nodal basis functions, as described in Section 2, at the point (x_k, y_k), and these four nonzeroes are entered in appropriate rows of column k of the matrix R. Since the four nodal basis functions on a given leaf element form a partition of unity, the column sum of each column of R is one. In terms of implementation, the array R representing matrix R is a $4 \times N$ dense array with appropriate pointers to indicate the leaf element and corresponding node numbers.

In the 3D case, the matrix R is constructed in a completely analogous way. The main difference is that each leaf element has eight nodes, and thus each column of R will contain eight nonzeroes. The column sums remain one as the eight nodal basis functions on each cube form a partition of unity.

The prolongation matrix is taken as R^t, and the course grid correction matrix A_c is given by

$$A_c = RAR^t. \tag{2}$$

In the following discussion we restrict attention to finite element stiffness matrices A in 2D, but the discussion has relevance to the 3D case and also to other classes of PDE discretizations.

We begin by noting that although bilinear finite elements have played a central role in the development of our coarse grid matrix A_c, the coarse subspace is not a space of piecewise bilinear polynomials. In particular, from (2) it is clear that the coarse space consists of linear combinations for the fine grid basis functions, and the coefficients of these linear combinations are determined by the bilinear finite element space. Thus this is a classic nested 2-level solver.

The coarse space basis functions do have some relation to the piecewise bilinear basis. Fine space basis functions associated with (x, y) coordinates close to regular vertex v_j will have larger coefficients than those more distant, so the coarse grid basis function centered at v_j may have some qualitative visual resemblance to the pyramid structure of the bilinear basis function centered at v_j, but this also strongly depends on the shapes of the fine space basis functions.

The support of a coarse space basis function consists of the union of the supports of the fine space basis functions with (x, y) coordinates lying within one of the leaf elements forming the support of the corresponding piecewise bilinear basis function. In particular, if the regular vertex v_j associated with a given coarse space function lies on the boundary or outside of the physical domain of the PDE, its support will automatically conform to the boundary as needed.

Because of the 1-irregular rule, the stiffness matrix $\hat{A}_c$ arising from the nodal bilinear basis associated with the quadtree mesh is sparse, with at most 13 nonzeroes in any row. If the supports of the coarse space basis functions overlap in a similar way, then the matrix A_c in (2) will also have this property. This is most likely to be achieved when the support of individual fine space functions is small relative to the size of the quadtree leaf that contains its (x,y) coordinates, and that its support is largely contained within that leaf. To the extent that this does not occur the density of A_c will likely increase. In the extreme case that A itself is dense, A_c will also be dense.

The 2-level cycle is classical, with one pre-smoothing iteration using a smoother M, followed by the coarse grid correction, and one post smoothing iteration. Below we summarize one cycle for solving (1). Let u_0 be the initial guess; normally $u_0 = 0$ and the right hand side b will be the residual from the previous cycle. The output is the approximate solution u_1,

$$\begin{aligned} M\delta_0 &= b - Au_0 \\ p_1 &= u_0 + \delta_0 \\ A_c\delta_1 &= R(b - Ap_1) \\ p_2 &= z_1 + R^t\delta_1 \\ M\delta_2 &= b - Ap_2 \\ u_1 &= p_2 + \delta_2. \end{aligned} \tag{3}$$

In our prototype implementation, we allow a choice of two smoothers.

- An incomplete *LU* factorization (*ILU*) with user specified drop tolerance δ_f. Minimum degree ordering is used.
- Block symmetric Gauss-Seidel (*SGS*) with blocks of size $N_f \geq N_c$, N_f user specified. Diagonal blocks are solved using sparse *LU* factorization with minimum degree ordering.

The coarse space solver implemented in our prototype code is

- *ILU* with user specified drop tolerance δ_c. Minimum degree ordering is used. $\delta_c = 0$ results in sparse *LU* factorization.

Our 2-level cycle is used as a preconditioner. If A is symmetric we employ the composite step conjugate gradient method (CSCG) [4] appropriate for symmetric indefinite as well as positive definite matrices. If A is not symmetric, we employ the composite step biconjugate gradient method (CSBCG) [4].

The theoretical analysis of 2-level methods has a long and expansive history in the multigrid literature. See [14, 12, 19, 10, 20, 17, 11, 5] and their references for a selection of theoretical analyses.

4 Numerical Experiments

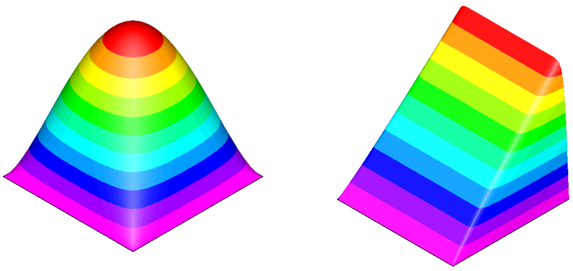
In this section, we present some numerical illustrations of this method. For simplicity, we have chosen a mostly uniform selection of parameters that apply to all of the examples. In particular the coarse grid target $N_e = 25$ is chosen in all of the experiments. The fine grid smoother is *ILU* based on minimum degree ordering and with drop tolerance $\delta_f = 10^{-4}$. The coarse space solver is sparse Gaussian elimination with minimum degree ordering (*ILU* with $\delta_c = 0$). The 2-level solver is a preconditioner for CSCG when A is symmetric, and for CSBCG when A is nonsymmetric. All computations were done on a Apple Mac Pro desktop (2013). The prototype codes were written in FORTRAN 2003, and compiled using the gcc8 compiler. In all 2D experiments, the sparse linear systems (1) were generated using the adaptive finite element package *PLTMG* [2].

In the first experiment, we consider five PDEs that challenge the solver in various ways. These problems are the standard Poisson equation, a convection-diffusion equation, a Helmholtz equation, an anisotropic diffusion equation, and a discontinuous diffusion equation.

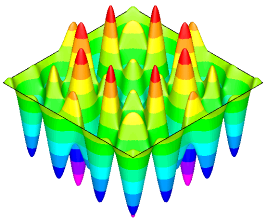
$$\begin{aligned} -\Delta u &= 1 \\ -\Delta u + 1000u_x &= 1 \\ -\Delta u - 1000u &= 1 \\ -.001u_{xx} - u_{yy} &= 1 \\ -\nabla(a\nabla u) &= 1 \qquad \text{with } a = \{1, .001\} \end{aligned}$$

The domain for all equations is the unit square, and the finite element discretization in all cases is continuous piecewise linear triangular finite elements. Homogeneous Dirichlet boundary conditions are applied in all problems. The meshes are uniform $n \times n$ with $n = 161, 321, 641, 1281$, and $N = n^2 = 25291, 103041, 410881, 1640961$, respectively. The solutions are shown in Figure 3.

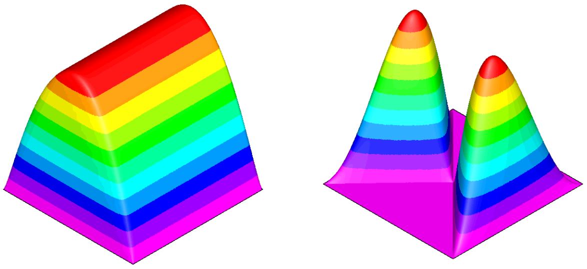
The results are given in Table 1. In Table 1, K is the number of iterations required to satisfy $\|r_K\|_{\ell_2} \leq \varepsilon\|r_0\|_{\ell_2}$, where $\varepsilon = 10^{-6}$ and r_j is the preconditioned residual, and Digits $= -\log(\|r_K\|_{\ell_2}/\|r_0\|_{\ell_2})$. $|A|$, $|A_c|$ are the number of nonzeroes in A and A_c, respectively. $|ILU|$ is the number of nonzeroes in the incomplete *LU* factorization of A, and $|L_cU_c|$ is the number of nonzeroes in the sparse *LU* factorization of A_c.



$-\Delta u = 1$ $-\Delta u + 1000 u_x = 1$



$-\Delta u - 1000 u = 1$



$-.001 u_{xx} - u_{yy} = 1$ $-\nabla \cdot (a \nabla u) = 1,\ a = \{1, .001\}$

Fig. 3

N	N_c	K	Digits	$\frac{\lvert A\rvert}{N}$	$\frac{\lvert A_c\rvert}{N}$	$\frac{\lvert ILU\rvert}{N}$	$\frac{\lvert L_cU_c\rvert}{N}$
			$-\Delta u = 1$				
25921	1089	2	7.23	6.95	0.98	42.73	2.72
103041	4225	2	7.22	6.98	0.99	46.10	4.50
410881	16641	2	7.47	6.99	0.99	48.02	6.74
1640961	66049	2	7.65	6.99	1.00	48.85	10.14
			$-\Delta u + 1000 u_x = 1$				
25921	1089	1	6.32	6.95	0.98	31.79	2.72
103041	4225	2	11.01	6.98	0.99	42.97	4.50
410881	16641	2	9.07	6.99	0.99	55.05	6.74
1640961	66049	2	7.12	6.99	1.00	64.08	10.15
			$-\Delta u - 1000 u = 1$				
25921	1089	9	6.87	6.95	0.98	64.04	2.72
103041	4225	15	6.49	6.98	0.99	69.96	4.50
410881	16641	16	6.97	6.99	0.99	50.69	6.74
1640961	66049	4	6.06	6.99	1.00	49.35	10.15
			$-.001 u_{xx} - u_{yy} = 1$				
25921	1089	2	6.43	6.95	0.98	12.32	2.72
103041	4225	3	7.93	6.98	0.99	12.88	4.50
410881	16641	3	7.23	6.99	0.99	13.23	6.74
1640961	66049	3	7.15	6.99	1.00	13.42	10.14
			$-\nabla(a \nabla u) = 1$				
25921	1089	3	7.89	6.95	0.98	39.34	2.72
103041	4225	3	6.75	6.98	0.99	44.30	4.50
410881	16641	3	6.49	6.99	0.99	47.06	6.74
1640961	66049	3	6.41	6.99	1.00	48.35	10.14

Table 1

Table 1 illustrates traditional multigrid-like convergence independent of N for four of the problems. The exception is the Helmholtz equation. For multilevel analysis of such indefinite problems, one typically makes some assumption that requires the coarse grid to be sufficiently fine. In this example, the coarse grid was not sufficiently fine for the three smaller vales of N, yet the solver still exhibited reasonable and largely N independent convergence. When $N = 1640961$, the coarse grid was sufficiently fine, and convergence behavior became more like the other four examples.

For our second example, we consider the problem

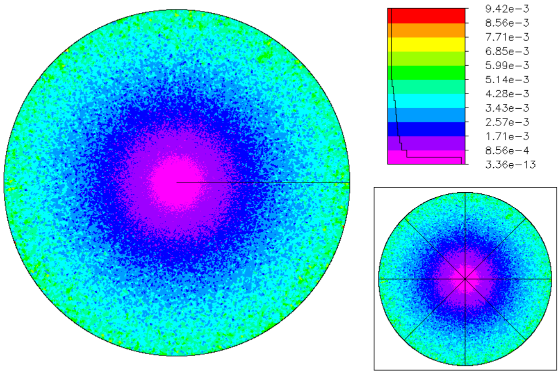
$$\begin{aligned} -\Delta u &= 0 && \text{for } x \in \Omega \\ u &= r^{1/4}\sin(\theta/4) && \text{for } x \in \partial\Omega_1 \\ \nabla u \cdot n &= 0 && \text{for } x \in \partial\Omega_2 \end{aligned}$$

where Ω is unit circle with crack on x-axis for $0 \le x \le 1$. $\partial\Omega_2$ is $0 \le x \le 1$ below the crack. $\partial\Omega_1 = \partial\Omega - \partial\Omega_2$. This problem was solved in *PLTMG* using h-refinement for continuous piecewise linear elements. This required 11 feedback loops to move from $N_0 = 10$ to $N = 250000$. In this experiment we used the same solution algorithm and parameter settings as in the previous example ($N_e = 25$, $\delta_f = 10^{-4}$, $\delta_c = 0$). The main difference was the convergence criteria was reduced from $\varepsilon = 10^{-6}$ to $\varepsilon = 10^{-2}$. This weaker criteria is more reasonable in the context of an adaptive feedback loop, where very good initial guesses are available for every solution step after the first.

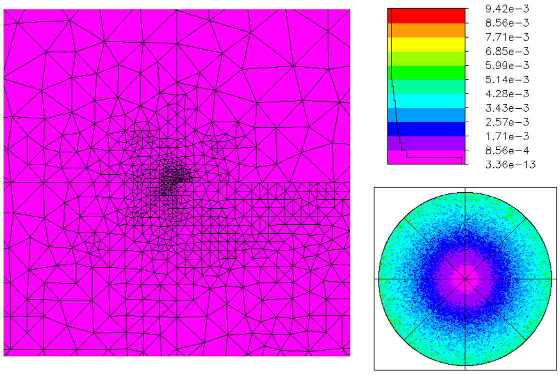
In terms of the PDE, there are three aspects of this problem that differ from the previous example. First, the outer boundary of the domain is a circle, so this boundary does not align with the quadtree mesh; thus some regular nodes in the coarse space will lie outside of the boundary. Second, this mesh is highly nonuniform, with elements that vary in diameter from $9.42 \cdot 10^{-3}$ to $3.36 \cdot 10^{-13}$, with the smallest elements clustered at the crack tip. This is illustrated in Figure 4. Third, the domain is not convex due the the crack; this means that coarse space basis functions corresponding to regular nodes near the crack may be linear combinations of functions lying on both sides of the crack. As illustrated in Figure 4, these coarse space functions will typically have a jump discontinuity that coincides with the crack.

N	N_c	K	Digits	$\frac{\lvert A\rvert}{N}$	$\frac{\lvert A_c\rvert}{N}$	$\frac{\lvert ILU\rvert}{N}$	$\frac{\lvert L_cU_c\rvert}{N}$
10	4	0	15.65	4.50	1.70	1.10	1.70
22	4	1	16.19	5.50	0.77	1.86	0.77
53	4	1	15.42	6.19	0.32	5.81	0.32
142	9	1	15.57	6.56	0.58	11.26	0.58
296	21	1	6.99	6.70	1.05	16.67	1.01
670	60	1	4.60	6.79	2.24	21.58	2.52
1526	92	1	3.22	6.86	1.64	26.83	1.87
3696	134	1	2.39	6.89	1.04	32.20	1.19
9935	276	2	3.36	6.93	0.80	38.56	1.47
28447	745	2	2.13	6.96	0.73	43.63	2.32
84645	2266	3	2.60	6.98	0.73	46.93	3.76
250000	6797	2	2.17	6.99	0.71	48.40	5.52

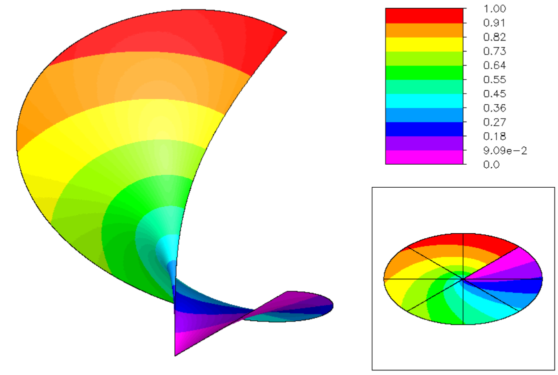
Table 2



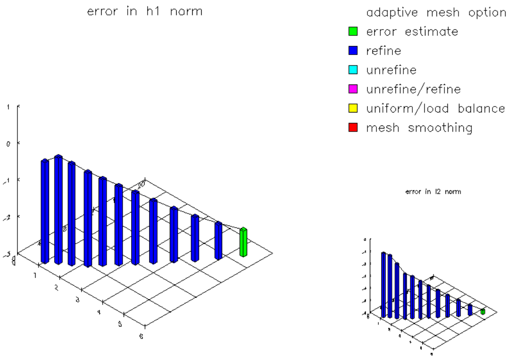
498286 elements colored by size.



Zoom by 10^{10}.



The solution $u = r^{1/4}\sin(\theta/4)$.



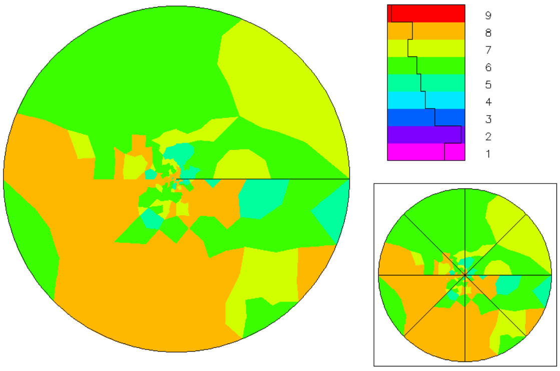
Convergence history of h-adaptive calculation.

Fig. 4

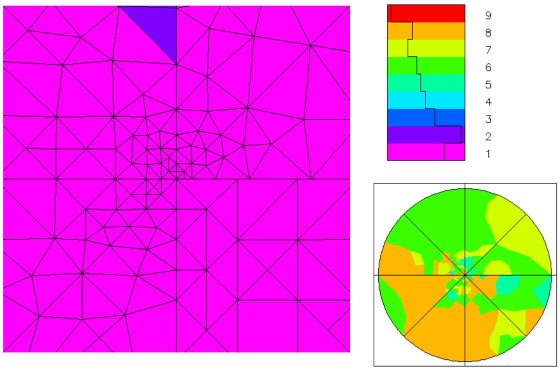
The results are shown in Table 2. The smaller values of N are not so interesting but are included for completeness. The entries for larger values of N show at most a slight dependence on N. The convergence rate is a bit slower than for the Poisson equation in the first example, likely due to the three factors mentioned above.

In our next example, we solve the same problem as in the second example, but now using the hp-adaptive strategy in *PLTMG*. Starting with the same initial mesh, we arrive at a final mesh with a target of 100000 degrees of freedom in 19 adaptive feedback loops. As illustrated in Figure 5, the element polynomial degrees varies from one to eight. and the element diameter varies from $1.4\cdot 10^{-1}$ to $2.2\cdot 10^{-22}$. As illustrated in Figure 5, there are small linear elements at the crack trip, that grow in size and degree as distance to the singularity increases. Thus this problem has a much wider range of element sizes, and the stiffness matrices have many more nonzero elements. Elements of degree eight have 45×45 dense element stiffness matrices. This problem exhibits the exponential convergence that is a hallmark of hp adaptive methods.

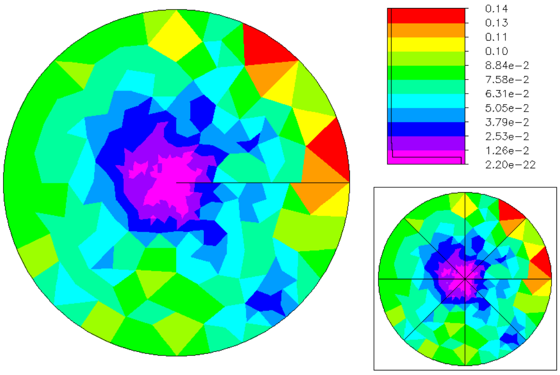
We solved this problem twice, once with the same parameters as in the h-adaptive case, $N_e = 25$, $\delta_f = 10^{-4}$, $\delta_c = 0$, and $\varepsilon = 10^{-2}$. In the second case $N_e = 100$ and the other parameters remain unchanged. The results are shown in Table 3. As before, the results for smaller values of N are not interesting but are included for completeness. For the case $N_e = 25$, the results for larger values of N are a bit erratic with respect to the growth in N, but are still reasonable given the challenging aspects of this problem. We think this erratic behavior is a consequence of the support for the coarse space functions and the support of the high



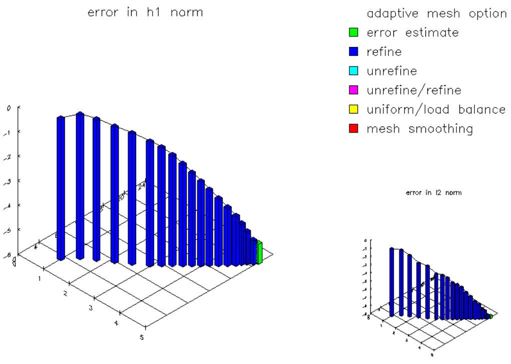
11255 elements colored by degree.



Zoom by 10^{20}.



11255 elements colored by size.



Convergence history of *hp*-adaptive calculation.

Fig. 5

N	N_c	K	Digits	$\frac{\lvert A\rvert}{N}$	$\frac{\lvert A_c\rvert}{N}$	$\frac{\lvert ILU\rvert}{N}$	$\frac{\lvert L_cU_c\rvert}{N}$
			$N_e = 25$				
10	4	0	15.65	4.50	1.70	1.10	1.70
22	4	1	16.19	5.50	0.77	1.86	0.77
53	4	1	15.42	6.19	0.32	5.81	0.32
142	9	1	15.57	6.56	0.58	11.26	0.58
363	36	1	6.20	8.08	2.05	17.10	2.15
921	68	1	4.64	9.72	1.90	23.55	2.16
1549	100	1	3.27	10.33	1.78	27.57	2.05
2461	130	1	2.25	11.36	1.50	32.21	1.75
4012	169	2	4.32	12.94	1.23	35.16	1.43
6072	210	2	3.24	14.43	1.02	38.46	1.20
9053	289	2	2.89	16.83	0.97	41.27	1.29
13097	360	1	2.00	19.60	0.83	43.58	1.21
19674	590	1	2.29	23.05	0.91	46.93	1.92
29352	788	2	2.02	27.33	0.80	50.28	1.96
42173	1161	5	2.25	30.06	0.82	52.36	2.38
51580	1480	3	2.09	31.53	0.86	53.69	2.62
63605	1742	3	2.22	33.36	0.81	55.11	2.74
83863	2249	6	2.33	35.90	0.78	56.90	2.80
99993	2741	8	2.02	39.86	0.79	59.63	3.33
			$N_e = 100$				
10	4	0	15.65	4.50	1.70	1.10	1.70
22	4	1	16.19	5.50	0.77	1.86	0.77
53	4	1	15.42	6.19	0.32	5.81	0.32
142	4	1	15.28	6.56	0.12	11.26	0.12
363	9	1	5.99	8.08	0.23	17.40	0.23
921	9	1	3.76	9.72	0.09	24.27	0.09
1549	47	1	2.72	10.33	0.70	28.01	0.79
2461	76	1	2.31	11.36	0.81	31.61	0.93
4012	119	2	4.30	12.94	0.84	35.43	0.97
6072	159	2	3.21	14.43	0.76	37.98	0.88
9053	202	2	2.78	16.83	0.66	41.24	0.77
13097	242	1	2.02	19.60	0.55	43.54	0.65
19674	302	1	2.36	23.05	0.46	47.09	0.56
29352	364	2	2.36	27.33	0.37	50.25	0.48
42173	425	2	2.65	30.06	0.31	52.40	0.41
51580	484	2	2.11	31.53	0.28	53.63	0.39
63605	584	2	2.68	33.36	0.28	54.78	0.43
83863	788	3	2.44	35.90	0.29	56.75	0.59
99993	890	4	2.46	39.86	0.27	59.33	0.58

Table 3

degree basis functions are comparable. In this case, the matrix A_c becomes denser and its sparsity pattern departs significantly from that of the bilinear functions on the same quadtree mesh. When $N_e = 100$ the leaf elements are larger so the supports of the coarse grid basis functions are larger in comparison with those of the high degree fine grid functions, and the matrix A_c is sparser. For the case $N_e = 100$, the convergence behavior is less erratic with the growth of N, and the convergence is usually better than the case $N_e = 25$. As a final remark, we note that the 2-level solver described in [3] converges faster on this example than either set of parameters. As a geometric method, it had complete knowledge of each element in the mesh, the shape of the boundary, as well as the degree of each basis function on the fine grid. Thus it is not surprising that it outperforms this semi algebraic method.

In our last example, we consider some 3D PDEs. In particular, we reprise the Poisson equation and the convection-diffusion equation of the first example generalized to three dimensions. The discretization in this case is a 7-point star finite difference stencil. We considered a unit cube with $16 \times 16 \times 16 = 4096$, $32 \times 32 \times 32 = 32768$, and $64 \times 64 \times 64 = 262144$ uniform meshes; larger values of N could not be accommodated due to space constraints of our computer. For these octree meshes, we initially chose the same parameters as in the first experiment, $N_e = 25$, $\delta_f = 10^{-4}$, $\delta_c = 0$, and $\varepsilon = 10^{-6}$. We then repeated the experiment three more times, with $N_e = 1000$, $\delta_f = 10^{-4}, 10^{-2}, 1$, and the other parameters unchanged. The results are given in Table 4. The convergence varies depending on the choice of parameters. As expected, it is quite good in the first case. In the case $N_e = 1000$, $\delta_f = 10^{-4}$, the convergence is not quite as good as the case $N_e = 25$, but the dimension of the coarse space N_c is much smaller. Then as δ_f is further increased, the convergence rate is more severely impacted. In the final case where $\delta_f = 1$, the *ILU* factorization is reduced to a simple Jacobi-like smoother. This is combined with a relatively weak coarse space correction, placing much reliance on the CSCG and CSBCG algorithms to speed convergence.

N	N_c	K	Digits	$\frac{\lvert A\rvert}{N}$	$\frac{\lvert A_c\rvert}{N}$	$\frac{\lvert ILU\rvert}{N}$	$\frac{\lvert L_cU_c\rvert}{N}$
$N_e = 25$, $\delta_f = 10^{-4}$ $-\Delta u = 1$							
4096	125	2	9.68	6.63	1.67	69.32	2.33
32768	729	2	6.53	6.81	1.81	106.80	6.48
262144	4913	3	8.85	6.91	1.88	130.39	19.27
$N_e = 1000$, $\delta_f = 10^{-4}$ $-\Delta u = 1$							
4096	27	2	8.46	6.63	0.18	74.54	0.18
32768	125	3	8.27	6.81	0.21	110.65	0.29
262144	729	3	7.62	6.91	0.23	131.85	0.81
$N_e = 1000$, $\delta_f = 10^{-2}$ $-\Delta u = 1$							
4096	27	5	6.42	6.63	0.18	16.88	0.18
32768	125	6	6.54	6.81	0.21	18.13	0.29
262144	729	6	6.32	6.91	0.23	18.82	0.81
$N_e = 1000$, $\delta_f = 1$ $-\Delta u = 1$							
4096	27	9	6.18	6.63	0.18	1.00	0.18
32768	125	14	6.34	6.81	0.21	1.00	0.29
262144	729	15	6.16	6.91	0.23	1.00	0.81
$N_e = 25$, $\delta_f = 10^{-4}$ $-\Delta u + 1000u_x = 1$							
4096	125	1	7.27	6.63	1.67	26.50	2.33
32768	729	1	6.00	6.81	1.81	44.33	6.48
262144	4913	2	10.28	6.91	1.88	81.65	19.27
$N_e = 1000$, $\delta_f = 10^{-4}$ $-\Delta u + 1000u_x = 1$							
4096	27	1	7.07	6.63	0.18	28.10	0.18
32768	125	2	11.96	6.81	0.21	46.55	0.29
262144	729	2	9.97	6.91	0.23	84.09	0.81
$N_e = 1000$, $\delta_f = 10^{-2}$ $-\Delta u + 1000u_x = 1$							
4096	27	2	8.18	6.63	0.18	14.32	0.18
32768	125	3	8.10	6.81	0.21	16.71	0.29
262144	729	4	6.01	6.91	0.23	19.42	0.81
$N_e = 1000$, $\delta_f = 1$ $-\Delta u + 1000u_x = 1$							
4096	27	18	6.43	6.63	0.18	1.00	0.18
32768	125	28	6.10	6.81	0.21	1.00	0.29
262144	729	40	6.01	6.91	0.23	1.00	0.81

Table 4

5 Concluding Remarks

In this section, we make several remarks related to this study, and present some possible directions for future research. First we remark on our choice of parameters. Our goal in choosing parameters was to have one set that produced reasonable results for all of the example problems. With broad ranges of parameters, on any given problem certain combinations of parameters can produce very poor convergence, while others will exhibit the type of convergence one expects from a good 2-level solver. For all of the example problems here, different choices of parameters will produce faster convergence rates and/or less computational cost per iteration. However, better or optimal parameter choices would likely be different for each example. One can see a bit of this effect in the different choices of N_e and δ_f that we made in the the last two examples in Section 4. Thus like many multi-level methods, one should not view this method as a black box, but rather as an effective approach that requires tuning for each environment in which it is used.

One could consider using the coordinates of the degrees of freedom as vertices in a graph of the matrix A; that is, an edge in the graph connecting vertices v_i and v_j corresponds to nonzero off-diagonal elements A_{ij} and A_{ji} in the matrix A. This graph provides some indication of the shape of the physical domain of the PDE, and in some simple cases displays exactly the structure of the underlying elements in the mesh. One can add weights to the edges that correspond to the size of the corresponding matrix elements. All of this information could be used in addition to the value of N_e in the creation of the quadtree or octree mesh. One could imagine starting from a macro mesh of elements (triangles, quadrilaterals, tetrahedrons, hexahedras, etc) and create a forest of trees rather than just a single tree as we did here. All of these generalizations align this approach more closely with more traditional algebraic multigrid (AMG) approaches, where the graph of the matrix A and matrix element size play a prominent role in the creation of coarse subspaces. For example, such information would have revealed the crack in the second and third examples on Section 4, and thus we could have avoided creating coarse basis functions with support on both sides of the crack. While it seems clear that such enhancements could improve the quality of the coarse subspaces generated by the method, at this point it is unknown what the effect of such improvements might have on the convergence of the solver. As with other algebraic methods, one must consider the computational cost and complexity of the setup phase of the algorithm in relation to possible improvements in the solve phase, and try to achieve an appropriate balance.

As a related point, one could choose other partitions of unity to define R, and in the nonsymmetric case, perhaps use different partitions of unity for restriction and prolongation matrices. As a simple experiment in this direction, in our 2D code, we implemented an option to choose continuous piecewise biquadratic finite elements in place of the bilinear elements. On problems with the same quadtree mesh, the biquadratic elements often outperformed the bilinear elements, but with a coarse space roughly four times as large and a matrix A_c with more nonzeroes per row. When we compensated for subspace dimension by choosing coarser quadtree meshes for the biquadratic case, such that the values of N_c were comparable, convergence rates were typically quite similar, but the matrix A_c for the biquadratic case still had more nonzeroes per row. So while certainly more research is necessary to make a definitive conclusion, at this point this particular option does not appear to be a promising direction.

Given that the quadtree and octree both produce a tree data structure, a natural possibility is to employ this data structure to construct a multilevel solver to approximately solve the coarse space system. A particularly attractive option, at least for 2D for meshes created using adaptive h-refinement, would be to create a hierarchical basis multigrid solver (HBMG). Geometric multilevel solvers constructed from tree data structures in this way are quite common when a quadtree or octree is used in conjunction with an adaptive mesh refinement scheme applied to the underling PDE, but at this point, such an option has not been implemented in our prototype codes.

Since all of our examples were scalar PDEs, some additional remarks on systems seem appropriate. In the case of a system, we can create a coarse space for each different function independently using the techniques of Section 2. If the unknowns of both the fine and coarse spaces are blocked in a natural way by function, the resulting restriction matrix R will have a block diagonal structure, with the diagonal blocks R_{ii} denoting restriction matrices that were generated for each of the functions in the PDE system. If the matrix A is blocked in the same way, then the coarse space matrix $A_c = RAR^t$ will inherit this block structure. Many PDE systems are solved by some type of inner/outer block Gauss-Seidel like iteration. In such cases, the 2-level solver could be used for the inner iterations involving the diagonal blocks A_{ii} of A, and the corresponding coarse space diagonal blocks $A_{c,ii}$ of A_c. One could also apply a coarse space correction to the outer iteration using the entire matrix A_c, or even apply coarse space corrections in both the inner and outer iterations. Once again, the choice would likely depend on the details of the particular PDE system.

Finally, some aspects of the quadtree and octree meshes generated in Section 2 resemble domain decomposition (DD). Thus it is natural to ask whether some version of this solver could be effectively employed as a DD solver in a parallel computing environment. An especially intriguing point to note here is that the tree data structure could be used to create a solver analogous to the solvers described in [7,9,6]. In addition to the fine space on the piece of the problem that it owns, each processor would have a coarse description of the entire problem, with the coarsening graded and becoming coarser with the distance from the fine region on that processor. The goal of such algorithms is to trade a little extra computation on each processor for a hopefully large reduction in the communication costs in the DD solver. Additionally, since each processor has a coarse description of the entire problem, the separate coarse space correction step common in many DD algorithms is avoided.

References

1. Baker, A.H., Kolev, T.V., Yang, U.M.: Improving algebraic multigrid interpolation operators for elasticity problems. Numer. Linear Algebra Appl. **17**, 495–517 (2010)
2. Bank, R.E.: PLTMG: A software package for solving elliptic partial differential equations, users' guide 13.0. Tech. rep., Department of Mathematics, University of California at San Diego (2018)
3. Bank, R.E.: A two level solver for *hp* adaptive finite element equations. Computing and Visualization in Science (to appear)
4. Bank, R.E., Chan, T.F.: An analysis of the composite step biconjugate gradient method. Numerische Mathematik **66**, 295–319 (1993)

5. Bank, R.E., Dupont, T.F.: Analysis of a two level scheme for solving finite element equations. Tech. Rep. CNA-159, Center for Numerical Analysis, University of Texas at Austin (1980)
6. Bank, R.E., Falgout, R., Jones, T., Manteuffel, T., McCormick, S., Ruge, J.: Algebraic multigrid domain and range decomposition (AMG-DD/AMG-RD). SIAM J. on Scientific Computing **37**, s113–s136 (2015)
7. Bank, R.E., Lu, S.: A domain decomposition solver for a parallel adaptive meshing paradigm. SIAM J. on Scientific Computing **26**, 105–127 (electronic) (2004)
8. Bank, R.E., Sherman, A.H., Weiser, A.: Refinement algorithms and data structures for regular local mesh refinement. In: Scientific Computing (Applications of Mathematics and Computing to the Physical Sciences) (R. S. Stepleman, ed.), pp. 3–17. North Holland (1983)
9. Bank, R.E., Vassilevski, P.S.: Convergence analysis of a domain decomposition paradigm. Computing and Visualization in Science **11**, 333–350 (2008)
10. Bank, R.E., Yserentant, H.: Multigrid convergence: A brief trip down memory lane. Computing and Visualization in Science **13**, 147–152 (2010)
11. Brezina, M., Vaněk, P., Vassilevski, P.S.: An improved convergence analysis of smoothed aggregation algebraic multigrid. Numer. Linear Algebra Appl. **19**(3), 441–469 (2012). DOI 10.1002/nla.775. URL `https://doi.org/10.1002/nla.775`
12. Falgout, R.D.: An introduction to algebraic multigrid. Computing in Science and Engineering **8**(6), 24–33 (2006). Special issue on Multigrid Computing. UCRL-JRNL-220851
13. Falgout, R.D., Jones, J.E., Yang, U.M.: The design and implementation of *hypre*, a library of parallel high performance preconditioners. In: A.M. Bruaset, A. Tveito (eds.) Numerical Solution of Partial Differential Equations on Parallel Computers, *Lecture Notes in Computational Science and Engineering*, vol. 51, chap. 8, pp. 267–294. Springer-Verlag (2006). UCRL-JRNL-205459
14. Hackbusch, W.: Multigrid methods and applications, *Springer Series in Computational Mathematics*, vol. 4. Springer-Verlag, Berlin (1985). DOI 10.1007/978-3-662-02427-0. URL `https://doi.org/10.1007/978-3-662-02427-0`
15. HYPRE: High performance preconditioners. Http://www.llnl.gov/CASC/hypre/
16. Kolev, T.V., Vassilevski, P.S.: Parallel auxiliary space AMG for H(curl) problems. J. Comput. Math. **27**, 604–623 (2009). Special issue on Adaptive and Multilevel Methods in Electromagnetics. UCRL-JRNL-237306
17. MacLachlan, S.P., Olson, L.N.: Theoretical bounds for algebraic multigrid performance: review and analysis. Numer. Linear Algebra Appl. **21**(2), 194–220 (2014). DOI 10.1002/nla.1930. URL `https://doi.org/10.1002/nla.1930`
18. Weiser, A.: Local-mesh, local-order, adaptive finite element methods with a posteriori error estimators. Ph.D. thesis, Computer Science Department, Yale University (1981)
19. Xu, J.: Iterative methods by space decomposition and subspace correction. SIAM Rev. **34**(4), 581–613 (1992). DOI 10.1137/1034116. URL `https://doi.org/10.1137/1034116`
20. Yserentant, H.: Old and new convergence proofs for multigrid methods. In: Acta numerica, 1993, Acta Numer., pp. 285–326. Cambridge Univ. Press, Cambridge (1993). DOI 10.1017/S0962492900002385. URL `https://doi.org/10.1017/S0962492900002385`