Comput Visual Sci (2019) 20:51–57
https://doi.org/10.1007/s00791-017-0280-z

A two level solver for $h-p$ adaptive finite element equations

Randolph E. Bank[1]

Received: 4 April 2017 / Accepted: 15 May 2017 / Published online: 25 May 2017

Abstract Higher order finite elements present certain challenges for multilevel methods. Such matrices have more nonzero elements and special block structure. In the case of $h-p$ adaptive methods, the block structure is more complicated. In this work we present a simple two level solver for such systems, that exploits these special properties. The convergence rate is (empirically) multigrid-like, at least up to piecewise polynomials of degree nine. Numerical illustrations demonstrate its robustness on a wide variety of problems, including convection–diffusion and Helmholtz equations.

Keywords Two level solver · $h-p$ adaptivity · Hierarchical basis

Mathematics Subject Classification 65M55 · 65F10

1 Introduction

As the use of $h-p$ adaptive methods become more widespread, the development of efficient solvers for the resulting linear systems becomes more important. Such matrices typically have many more nonzeroes than the sparse matrices for low order finite element spaces. These nonzeroes generally can be organized in dense blocks of varying sizes. These properties should be taken into account when developing multi level solvers.

In this work, we present a simple two level solver for such matrices. The main components are a smoother and a coarse grid correction. Because of the density of the matrices, we restrict attention to very simple block smoothers, in this case block symmetric Gauss–Seidel. The development of more sophisticated and potentially more effective smoothers is hindered by the density of the matrices. Allowing even a modest degree of fill-in is likely to result in smoothers very close the sparse Gaussian elimination in complexity. In that situation, it is likely that the direct method would become more efficient.

Our coarse grid correction is inspired by the block data structure we use to store the system matrix, and has a hierarchical basis flavor. The coarse grid space always contains the space of piecewise linear finite elements, and if all elements have degree $p \geq 2$, it will contain the complete space of piecewise quadratic finite elements. For the case of two dimensional triangular finite elements and scalar partial differential equations, the dimension of the coarse grid correction space is bounded by approximately $6V$, where V is the number of vertices in the mesh. In more general settings, one should expect the coarse grid space to be bounded in a similar fashion, and thus to become quite small in comparison to the fine grid space as p increases.

This two level method exhibits interesting behavior on adaptive $h-p$ meshes. Initially, all of the elements have degree $p = 1$, and the coarse grid correction space and the fine space are identical. This remains true as long as all elements have degree $p \leq 2$. Typically, these spaces have relatively small dimension, and a simple two level hierarchical basis method can easily solve problems on the coarse grid correction space. In this setting, the smoother is mainly an added benefit. Later, when p becomes larger for many of the elements, the coarse grid space becomes smaller in compari-

The work of R. E. Bank was supported by the National Science Foundation under contract DMS-1318480.

✉ Randolph E. Bank
rbank@ucsd.edu

[1] Department of Mathematics, University of California, San Diego, La Jolla, CA 92093-0112, USA

son with the fine grid, and the smoother plays an increasingly prominent role.

The method empirically exhibits convergence rates that are independent of the mesh size h, and sometimes even improves as h becomes smaller. On the other hand, because the dimension of the coarse grid correction space is bounded independent of p, it does not have an asymptotic convergence rate bounded independent of p, and we expect that the convergence rate will eventually decrease with increasing p and fixed h. On the other hand, there are often good reasons to bound p in practical situations. These could include the availability of quadrature formulas (this limits $p \leq 9$ in the *PLTMG* software package), or the complexity growth in p of common finite element procedures, e.g, matrix and right hand side assembly, that will fail to scale as $O(n)$, n the fine space dimension, unless p is bounded. For this reason, we are not overly concerned about its convergence behavior as $p \to \infty$. As we illustrate in Sect. 4, the method performs quite well in the range $p \leq 9$.

The rest of this paper is organized as follows. In Sect. 2, we describe the point and block sparse matrix data structures that we use, and that motivated our choice of coarse grid correction subspace. In Sect. 3, we define our two level solver for the case of the standard family of two dimensional triangular finite element spaces, and scalar elliptic partial differential equations. In Sect. 4, we present some numerical results for a variety of elliptic equations.

2 Sparse matrix data structures

Let A be an $n \times n$ matrix with elements A_{ij}, and a symmetric sparsity structure; that is, both A_{ij} and A_{ji} are treated as nonzero elements (i.e. stored and processed) if $|A_{ij}|+|A_{ji}| > 0$. All diagonal entries A_{ii} are treated as nonzero regardless of their numerical values.

Our point data structure is a modified and generalized version of the data structure introduced in the (symmetric) Yale Sparse Matrix Package [4,5]. In our data structure, the nonzero entries of A are stored in a linear array a, and accessed through an integer array ja. In the real array a, we store the diagonal first, followed by the strict upper triangle stored row-wise. If $A^T \neq A$, this is followed by the strict lower triangle stored column-wise. In the corresponding integer array ja, pointers to the column indices and the nonzeroes in each row of the strict upper triangle of A are stored in the first $n+1$ places, while the remaining storage in ja contains the respective column indices themselves. Since A is structurally symmetric, the column indices for the upper triangle are identical to the row indices for the lower triangle, and hence need not be duplicated in storage.

Let η_i be the number of nonzeros in the strict upper triangular part of row i, and set $\eta = \sum_{i=1}^{n} \eta_i$. The array ja is of length $n+1+\eta$ and the array a is of length $n+1+\eta$ if $A^T = A$. If $A^T \neq A$, then the array a is of length $n+1+2\eta$. The entries of $ja(i)$, $1 \leq i \leq n+1$, are pointers defined as follows:

$$ja(1) = n+2$$
$$ja(i+1) = ja(i) + \eta_i, \quad 1 \leq i \leq n$$

The locations $ja(i)$ to $ja(i+1)-1$ contain the η_i column indices corresponding to the row i in the strictly upper triangular matrix.

In a similar manner, the array a is defined as follows:

$$a(i) = A_{ii}, \quad 1 \leq i \leq n$$
$$a(n+1) \quad \text{is arbitrary}$$
$$a(k) = A_{ij}, \quad 1 \leq i \leq n,$$
$$j = ja(k), \quad ja(i) \leq k \leq ja(i+1)-1$$

If $A^T \neq A$, then

$$a(k+\eta) = A_{ji}, \quad 1 \leq i \leq n,$$
$$j = ja(k), \quad ja(i) \leq k \leq ja(i+1)-1$$

As an example, let

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & 0 & 0 \\ A_{21} & A_{22} & 0 & A_{24} & 0 \\ A_{31} & 0 & A_{33} & A_{34} & A_{35} \\ 0 & A_{42} & A_{43} & A_{44} & 0 \\ 0 & 0 & A_{53} & 0 & A_{55} \end{pmatrix}$$

Then

	1	2	3	4	5	6
ja	7	9	10	12	12	12
a	A_{11}	A_{22}	A_{33}	A_{44}	A_{55}	
	Diagonal					

	7	8	9	10	11
ja	2	3	4	4	5
a	A_{12}	A_{13}	A_{24}	A_{34}	A_{35}
	Upper triangle				

	12	13	14	15	16
ja					
a	A_{21}	A_{31}	A_{42}	A_{43}	A_{53}
	Lower triangle				

To illustrate the use of this data structure, the following algorithm computes $y = Ax$ or $y = A^T x$. If $A = A^T$, the parameters $lshift = ushift = 0$. If $A^T \neq A$ and we wish to compute $y = Ax$, then $ushift = 0$ and $lshift = \eta = ja(n+1) - ja(1)$. If $A^T \neq A$ and we wish to compute $y = A^T x$, then $ushift = \eta$ and $lshift = 0$.

```
for i = 1, n
  y(i) ← a(i) × x(i)
end for
for i = 1, n
  for k = ja(i), ja(i + 1) − 1
  j = ja(k)
  y(i) ← y(i) + a(k + ushift) × x(j)
  y(j) ← y(j) + a(k + lshift) × x(i)
  end for
end for
```

A similar data structure can be used to store an LDU or $U^T DU$ factorization, arising from either Gaussian elimination or an incomplete factorization. In this case, only the strictly lower (upper) triangular part of L (U) needs to be stored as the diagonal is an identity matrix.

Now suppose matrix A is a block $m \times m$ matrix, with block i having dimension n_i and

$$\sum_{i=1}^{m} n_i = n.$$

We assume the block structure is structurally symmetric, and that each block A_{ij} is a dense $n_i \times n_j$ matrix, and is stored as a linear array within the data structure a. Our block ja data structure is analogous to the point ja data structure described above, but now the column indices refer to blocks rather than individual matrix elements. Because entries in array a will no longer correspond in a simple fashion to those in ja, we need two additional integer data structures. First, the array b of size m that stores the sizes of individual blocks

$$b(i) = n_i, \quad 1 \leq i \leq m.$$

We need a second array jp of pointers that indicate the starting point for various matrix blocks in the array a. Array jp is the same size as ja. For $1 \leq i \leq m$, $jp(i)$ points to the beginning of the ith diagonal block, and $jp(i+1) - 1$ points to the end of that block. For $ja(1) \leq i \leq ja(m+1) - 1$, $jp(i-1)$ points to the beginning of the corresponding off-diagonal block in the upper triangle and $jp(i) - 1$ points to its end.

In the array a, each diagonal block is stored diagonal entries first, followed by upper triangular entries stored row-wise. If $A^T \neq A$, this is followed by the lower triangular entries stored column-wise. Off diagonal blocks in the upper triangle are stored row-wise. If $A^T \neq A$, the upper triangle is followed by the lower triangular blocks stored column-wise analogous to the point data structure described above.

3 Two level solver

We consider scalar elliptic partial differential equations of the form

$$-\nabla \cdot (\alpha u) + \beta \cdot u + \gamma u = f$$

in $\Omega \subset \mathbb{R}^2$ with

$$u = 0$$

on $\partial\Omega$. Our $h-p$ adaptive algorithm is based on conforming triangular Lagrange finite elements of degrees 1–9. The upper bound of 9 was imposed by the family of quadrature rules implemented in the *PLTMG* package [2].

There is a natural block structure associated with such an $h-p$ adaptive mesh. Suppose the triangular mesh has V vertices. Each vertex is associated with a nodal basis function, each giving rise of a block of size $n_i = 1$ in the stiffness matrix A. If a triangle edge is associated with a polynomial of degree $p > 1$, then there are $p-1$ nodal basis functions (bump functions) associated with interior nodes on that edge; the two endpoint basis functions are counted as vertex functions. These $p-1$ basis functions give rise to a block of size $n_i = p-1$ in A. Finally, if the element itself is of degree $p > 2$ there will be $(p-1)(p-2)/2$ nodal basis functions (bubble functions) associated with the strict interior of that triangle. These give rise to a block of size $n_i = (p-1)(p-2)/2$ in A.

For a mesh consisting uniformly of elements of degree p, the order of the stiffness matrix A is approximately $n \approx p^2 V$. However, the block dimension of the matrix is approximately $m \approx 6V$ independently of p. Although vertex, bump, and bubble functions typically have different numbers of non zeroes per row, on average, A has $(p^2+6p+7)/2$ non zeroes per row, indicating that A becomes significantly more dense with increasing p. If $p \leq 2$, all of the blocks have $n_i = 1$ and the block data structure is unnecessary and the point data structure is more efficient. The block data structure begins to pay dividends as p increases. The combined storage for all of the integer data structures will become smaller than the single ja array used in the point data structure. Perhaps more important, the ratio of overhead operations (indirect addressing) to floating point operations used in assembling and solving the linear system is increasingly reduced with increasing p as the blocks become larger.

Our two level solver consists of a smoother and a coarse grid correction. The smoother is block symmetric Gauss–Seidel, using the block structure of the matrix A. For each of the diagonal blocks we use (dense) Gaussian elimination to compute an LDU factorization (or U^TDU factorization if $A^T = A$).

The coarse grid correction is motivated by interpreting the block *ja* data structure for the matrix A as a point *ja* data structure for the coarse grid correction matrix H. In particular, we compute a hierarchical basis for the coarse grid correction space as follows:

- For each vertex v in the mesh, we associate a continuous piecewise linear nodal basis (pyramid) function. This implies that the coarse grid correction space always contains the usual space of continuous piecewise linear polynomials.
- For each interior edge e associated with piecewise polynomials of degree $p \geq 2$, we add a single quadratic bump function associated with the midpoint of edge e.
- For each element t associated with piecewise polynomials of degree $p \geq 3$, we add a single cubic bubble p function associated with the barycenter of element t.

With these definitions, the block *ja* data structure can also be used as the point *ja* data structure for H, with the block column indices for A interpreted as point column indices for H. Additionally, the coarse grid correction space has a maximum dimension of approximately $6V$, achieved when all elements in the mesh have degree $p \geq 3$. The average number of non zeroes per row in H in this maximal situation is 23/2. The basis associated with the coarse grid correction has a natural hierarchical structure, allowing standard hierarchical basis solvers to be employed for solving linear systems associated with the coarse grid correction. Often the matrix H is sufficiently small relative to A that LDU or U^TDU factorizations are feasible and attractive. The prolongation and restriction operators are just the usual Galerkin (change of basis) operators.

In terms of convergence, many of the standard two level proofs apply when the mesh is quasiuniform [1,6,7]. However, it should be clear that the rate of convergence cannot be independent of p as $p \to \infty$. On the other hand, as a practical matter p should be bounded. In *PLTMG*, $p \leq 9$ is required due to the suite of quadrature rules used by the package. More generally, if we again consider the case of uniform p, the subspace dimension scales as $n \approx p^2V$. The cost of numerical quadrature scales as $p^4V = O(p^2n)$, and the cost of Gaussian elimination for the diagonal blocks of A scales as $p^6V = O(p^4n)$. In this situation, if one seeks complexity estimates that scale as $O(n)$, then p should be bounded, e.g., as $p \leq p_{max}$. As a side remark, for d dimensional simplicial elements, the space dimension scales as $n \approx p^dV$, while the quadrature and diagonal block factorization costs scale as $p^{2d}V$ and $p^{3d}V$, respectively.

4 Numerical experiments

In this section, we present a selection of numerical results that illustrate the robustness of this two level $h-p$ solver. To have a well controlled experimental environment, we used uniform square meshes and constant $p = 8$ in all of these experiments. In particular, we set $\Omega = (0,1) \times (0,1)$, and employed uniform meshes of size 41×41, 81×81, and 161×161, resulting in finite element spaces of approximate dimension 103 K, 410 K, and 1.64 M, respectively, using continuous piecewise polynomials of degree $p = 8$.

We solved the six linear scalar partial differential equations given below:

$$
\begin{aligned}
&-\Delta u = 1\\
&-10^{-3}u_{xx} - u_{yy} = 1\\
&-\Delta u + 10^3 u_x = 1\\
&-\Delta u + 10^3((y-.5)u_x - (x-.5)u_y) = 1\\
&-\Delta u - 10^3 u = 1\\
&-\Delta u + 10^3 u = 1.
\end{aligned}
$$

For all these equations we impose homogeneous Dirichlet boundary conditions $u = 0$ on $\partial\Omega$. The solutions of these six problems are shown in Fig. 1.

We solved each of the example problems for the three values of n, and for two different values of the error tolerance, $\varepsilon = 10^{-3}$ and $\varepsilon = 10^{-6}$. We started all iterations with initial guess zero, and used the criteria

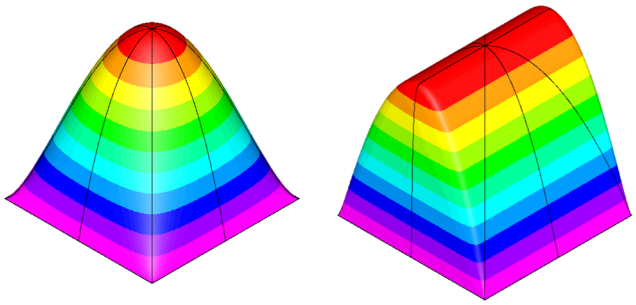
$$e_k = \frac{\|r_k\|_{\ell_2}}{\|r_0\|_{\ell_2}} \leq \varepsilon \tag{1}$$

to measure convergence. Here r_k is the preconditioned residual. The problems were accelerated using composite step conjugate gradients [3] for symmetric problems and composite step biconjugate gradients for nonsymmetric problems. The coarse grid problems were solved using one iteration of *ILU* where the drop tolerance was made sufficiently small that as a practical matter it became a direct solve.

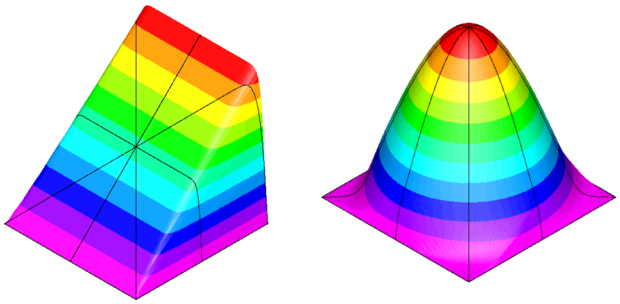
The results are reported in Tables 1 and 2. The reported data are K, the number of iterations required to satisfy (1), and

$$\text{Digits} = -\log e_K.$$

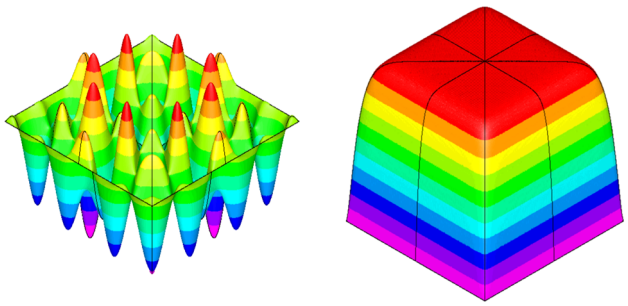
For $\varepsilon = 10^{-3}$, all of the problems were solved in few iterations, often with $e_K \ll \varepsilon$. This is a reflection of the



$-\Delta = u = 1$ *(left) and* $-10^{-3}u_{xx} - u_{yy} = 1$ *(right).*



$-\Delta u + 10^3 u_x = 1$ *(left) and* $-\Delta u + 10^3((y-.5)u_x - (x-.5)u_y) = 1$ *(right).*



$-\Delta u - 10^3 u = 1$ *(left) and* $-\Delta u + 10^3 u = 1$ *(right).*

Fig. 1 Solutions of example problems

effectiveness of the coarse grid correction. Starting from an initial guess of zero, on the first iteration, the coarse grid correction computes a solution that is essentially as good as the finite element solution for uniform $p = 2$. Thus, we should expect an exceptional decrease in the error on the first iteration that is not necessarily a reflection of the asymptotic behavior of the method. For $h-p$ adaptive meshes, the coarse grid space always includes the piecewise linear subspace, but not necessarily the complete quadratic subspace, so one can still expect an exceptional decrease of the error on the first iteration. We note that in the context of an adaptive feedback loop, only a modest error reduction is needed in any solve step due to a good initial guess. (In *PLTMG*, the default is $\varepsilon = 10^{-2}$.) Thus for these classes of problems, this solver is effective in this context.

The results for $\varepsilon = 10^{-6}$ reveal a bit more about the asymptotic behavior of the method. The results are still quite good for the Poisson equation, and for the two equations $-\Delta u \pm 10^3 u = 1$. Particularly impressive is the highly indefinite Helmholtz equation, systems that are often very difficult to solve. The number if iterations decreases with increasing n. This effect is likely due to improvements in the coarse grid correction. As h is decreased, the approximation properties provided by the piecewise quadratic finite element space contained within the coarse grid correction improves.

The two convection dominated equations also perform well but show a significant increase in the number of iterations for the case $n \approx 1.6m$. After some investigation, it appears that this is due to growth in the condition number of A with increasing n. Since condition numbers for all these problems are increasing, this effect will eventually appear in all of the problems as n increases. The use of higher precision floating point arithmetic could delay the onset of this effect. In all these experiments, we used standard double precision arithmetic.

As a side remark, as the size of the problems addressed by adaptive methods continues to increase, roundoff error issues will become increasingly important in many parts of the calculation, not just the solver. For example, it is now common for h-refined meshes to locally become so refined that the coordinates of all the vertices of some elements can be identical to the level of roundoff error. This implies that many routine finite element calculations, such as mapping such

Table 1 $\varepsilon = 10^{-3}$

	$n = 103{,}041$		$n = 410{,}881$		$n = 1{,}640{,}961$	
	K	Digits	K	Digits	K	Digits
$-\Delta u = 1$	1	4.95	1	5.59	1	6.21
$-10^{-3}u_{xx} - u_{yy} = 1$	2	3.50	1	3.51	1	4.45
$-\Delta u + 10^3 u_x = 1$	1	6.53	1	6.63	1	5.85
$-\Delta u + 10^3((y-.5)u_x - (x-.5)u_y) = 1$	1	6.78	1	6.24	1	5.78
$-\Delta u - 10^3 u = 1$	3	3.54	1	3.55	1	4.45
$-\Delta u + 10^3 u = 1$	1	3.50	1	4.13	1	4.74

Table 2 $\varepsilon = 10^{-6}$

	$n = 103{,}041$		$n = 410{,}881$		$n = 1{,}640{,}961$	
	K	Digits	K	Digits	K	Digits
$-\Delta u = 1$	3	6.12	3	6.74	1	6.21
$-10^{-3}u_{xx} - u_{yy} = 1$	63	6.15	46	6.00	25	6.05
$-\Delta u + 10^3 u_x = 1$	1	6.49	1	6.64	3	6.59
$-\Delta u + 10^3((y-.5)u_x - (x-.5)u_y) = 1$	1	6.79	1	6.24	4	6.42
$-\Delta u - 10^3 u = 1$	8	6.46	6	6.59	4	6.18
$-\Delta u + 10^3 u = 1$	6	6.42	5	6.54	4	6.40

Table 3 Storage $\times 10^{-3}$

	$n = 103{,}041$			$n = 410{,}881$			$n = 1{,}640{,}961$		
	$m = 9761$			$m = 38{,}721$			$m = 154{,}241$		
	Int	Sym	Nonsym	Int	Sym	Nonsym	Int	Sym	Nonsym
A	135	3101	6100	539	12,398	24,385	2154	45,976	97,512
H	63	63	116	250	250	462	1000	1000	1846
$GE(A)$	526	6628	13,153	2820	32,495	64,579	14,744	158,375	315,108
$GE(H)$	247	247	484	1361	1361	2684	7295	7217	14,280
$GE(D)$	0	878	1652	0	3505	6600	0	14,012	26,384

elements to a reference element, can suffer from catastrophic cancellation.

Perhaps the most significant difference in Table 2 is the anisotropic problem, which showed a large increase in the number of iterations. This is likely due to the smoother. A block smoother more like a line smoother for point matrices would be a better choice from the viewpoint of convergence. While implementation of such a smoother might be relatively straightforward for this particular problem, it could be more challenging in the case of unstructured meshes with variable p. Also as we discuss below, using an incomplete factorization that allows some fill-in as a smoother to overcome this problem might make this two-level solver unattractive in comparison with sparse Gaussian elimination. However, with the low convergence tolerances typical of the feedback loops used in adaptive methods, development of such a smoother is not such an important issue in practice.

In Table 3 we provide some data on the size of various data structures used in these calculations. Indirectly, this also provides some indication of the computational complexity. Information is provided on A, H, sparse Gaussian elimination for both A and H using a minimum degree ordering, and on factorization of the diagonal blocks of A used in the smoother of our two level iteration. Three numbers are provided. In the case of A and its Gaussian elimination factorization, *int* reports the combined storage of the *ja* array, the corresponding pointer array *jp* (same size as *ja*), and the block size array *b* of size *m*. The factorization of the diagonal blocks of A can make use of the *int* data structures of A so no additional integer data structure are needed. For H and its factorization, only the point *ja* data structure is needed. The *sym* and *nonsym* columns reports the number of nonzeroes stored, essentially the size of the *a* array.

From the data we note that the total integer storage for the block matrices is far less than the real storage. Thus the integer overhead operations are greatly reduced in relation to the floating point operations needed for common matrix operations, so the block data structure is far more efficient in these cases than a point data structure, where the integer and real data structures are of comparable size.

Gaussian elimination of the coarse grid matrix H is less costly and requires less space than the factorization of the diagonal blocks of A needed for the block symmetric Gauss–Seidel smoother. Thus the cost of our two level solver is dominated by the cost of the smoother. Solving linear systems involving H by an iterative method, e.g. a hierarchical basis method, might be locally more efficient than direct solution, but that efficiency will have a smaller impact on the global complexity of the two level solver.

Also note that size of the LDU or $U^T DU$ factorizations of A by Gaussian elimination is only 2–3 times that of the original matrix. Thus, preconditioners or smoothers based on incomplete factorization allowing even modest fill-in are likely to result in factorizations very close to Gaussian elimination.

References

1. Bank, R.E.: Hierarchical bases and the finite element method. In: Iserles, A. (ed.) Acta Numerica, pp. 1–43. Cambridge University Press, Cambridge (1996)
2. Bank, R.E.: PLTMG: A software package for solving elliptic partial differential equations, users' guide 12.0. Technical report, Department of Mathematics, University of California at San Diego (2016)
3. Bank, R.E., Chan, T.F.: An analysis of the composite step biconjugate gradient method. Numer. Math. **66**, 295–319 (1993)
4. Eisenstat, S.C., Gursky, M.C., Schultz, M.H., Sherman, A.H.: Yale sparse matrix package I: the symmetric codes. Int. J. Numer. Methods Eng. **18**, 1145–1151 (1982)
5. Eisenstat, S.C., Schultz, M.H., Sherman, A.H.: Algorithms and data structures for sparse symmetric Gaussian elimination. SIAM J. Sci. Stat. Comput. **2**, 225–237 (1982)
6. Xu, J.: Iterative methods by space decomposition and subspace correction. SIAM Rev. **34**(4), 581–613 (1992)
7. Yserentant, H.: Old and new convergence proofs for multigrid methods. In: Iserles, A. (ed.) Acta Numerica, pp. 285–326. Cambridge University Press, Cambridge (1993)