Application of Sparse Matrix Methods to Partial Differential Equations†

S. C. Eisenstat, M. H. Schultz,
and A. H. Sherman *

Research Report #47

---

† This paper has appeared in the Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, June 1975.

* Department of Computer Science, Yale University, New Haven, Connecticut 06520.
This research was supported in part by NSF Grant GJ-43157 and ONR Grant N0014-67-A-0097-0016.

1. Introduction

We consider the system of linear equations $Ax = b$ where the coefficient matrix A is an $N \times N$ symmetric, positive definite matrix. Such systems arise frequently in scientific computation and their solution often represents a significant portion of the total calculation.

When N is small or the matrix A is dense (i.e. most elements a_{ij} are nonzero), the usual way to solve the system is with symmetric Gaussian elimination [1]. Equivalently, we factor A into the product U^tDU, where D is a positive diagonal matrix and U is unit upper triangular, and then successively back-solve the triangular systems

$$U^ty = b, \quad Dz = y, \quad Ux = z$$

for the solution x. However, when A is large and sparse (i.e. most elements a_{ij} are zero), symmetric Gaussian elimination is extremely inefficient, both in terms of storage (because of the need to store the large number of zero elements in A and U) and work (because of the need to perform a large number of arithmetic operations on zero operands).

Historically, such large sparse symmetric systems have been solved with iterative methods, in which we choose an initial guess $x^{(0)}$ and successively generate a sequence of iterates $x^{(k)}$ that converges to the exact solution. The iteration stops when some iterate satisfies an estimated error criterion, usually based on the residual $Ax^{(k)} - b$, and that iterate is taken as an approximate solution. Such methods require very little extra storage (typically one or two temporary vectors) beyond

that needed for A, x, and b, and converge sufficiently rapidly as to be practical for certain large classes of problems. Unfortunately, the total work required (i.e. the number of arithmetic operations) is highly dependent on the initial guess; the stopping criterion is a delicate balance between speed and accuracy; and the rate of convergence is critically dependent on iteration parameters that are rarely known a priori. These difficulties have been overcome for some problems arising out of finite difference approximations to elliptic partial differential equations [14], but not in general.

More recently, a great deal of attention has been focused on variants of symmetric Gaussian elimination that attempt in varying degrees to avoid storing or operating on zero elements occurring during the elimination process. In this paper, we shall examine three such methods -- band elimination [11], profile or envelope elimination [6], and (general) sparse elimination [9] -- to see how well they meet this goal and how efficiently they can be implemented.

In Section 2, we introduce several linear systems typical of sparse systems arising in practice. In Section 3, we present the three variations of symmetric Gaussian elimination within the context of avoiding the storage and work associated with zero elements and show the importance of the ordering of the unknowns and equations in this regard. In Section 4, we investigate the relative efficiencies of these variations by looking closely at their innermost loops and, for sparse elimination, at the additional pointer storage required to store sparse matrices. Last, in Section 5, we draw some tentative

conclusions as to what methods should be used in practice.

2. Model Problems

In this section, we introduce several model problems that lead quite naturally to sparse symmetric linear systems of the kind typically encountered in practice. Although they are all very specialized, nearly all that will be said about them is valid for more general problems.

Our first model problem arises from the Poisson equation on the unit square $D = (0,1)\times(0,1)$ with homogeneous Dirichlet boundary conditions:

$$-\Delta v = f \text{ in } D; \qquad v = 0 \text{ on } \partial D.$$

To solve this problem, we must reduce the continuous problem to a discrete one. We cover the unit square with a uniform grid with mesh-width $h = 1/(n+1)$ and seek an approximation V_{ij} to $v(ih,jh)$ at each interior meshpoint. Replacing the differential operator by the familiar five-point difference approximation at each interior meshpoint, we obtain

$$4V_{ij} - V_{i,j-1} - V_{i,j+1} - V_{i-1,j} - V_{i+1,j} = h^2F_{ij} \qquad 1 \le i, \; j \le n$$

where $V_{ij} = 0$ if $i = 0,n+1$ or $j = 0,n+1$ and $F_{ij} = f(ih,jh)$ [7]. Numbering the unknowns in the natural row-by-row ordering (see Figure 3.1a), we get the $n\times n$ block tridiagonal system of linear equations

$$\begin{bmatrix} T & -I & & & \\ -I & T & -I & & \\ & \ddots & \ddots & \ddots & \\ & & & -I & \\ & & -I & T & \end{bmatrix} \qquad V = h^2 F$$

where T is the $n \times n$ tridiagonal matrix

$$T = \begin{bmatrix} 4 & -1 & & & \\ -1 & 4 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & & -1 & \\ & & -1 & 4 & \end{bmatrix}$$

and I is the $n \times n$ identity matrix. We shall refer to this system as the five-point model problem on an $n \times n$ grid. It can also be derived using the Rayleigh-Ritz-Galerkin method with linear right-triangular elements.

A slightly different linear system comes from using a nine-point finite difference operator to approximate the differential operator:

$$(\Delta v)_{ij} = -8V_{ij} + V_{i-1,j-1} + V_{i-1,j} + V_{i-1,j+1} + V_{i,j-1} + V_{i+1,j-1}$$

$$+ V_{i+1,j} + V_{i+1,j+1} + V_{i,j+1} \quad .$$

The corresponding matrix problem is again an $n \times n$ block tridiagonal system

$$\begin{bmatrix} B & -C & & & \\ -C & B & -C & & \\ & \ddots & \ddots & \ddots & \\ & & -C & B & -C \\ & & & -C & B \end{bmatrix} V = h^2 F$$

where B and C are $n \times n$ tridiagonal matrices

$$B = \begin{bmatrix} 8 & -1 & & & \\ -1 & 8 & -1 & & \\ & \ddots & \ddots & \ddots & \\ & & -1 & 8 & -1 \\ & & & -1 & 8 \end{bmatrix}, \qquad C = \begin{bmatrix} 1 & 1 & & & \\ 1 & 1 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & 1 & 1 \\ & & & 1 & 1 \end{bmatrix}.$$

We shall refer to this system as the nine-point model problem on an $n \times n$ grid. It can also be derived using Rayleigh-Ritz-Galerkin with a basis of tensor products of piecewise linear functions.

Our third model problem arises from the biharmonic problem

$$\Delta^2 v = f \text{ in } D; \quad v = 0, \quad \partial v/\partial n = 0 \text{ on } \partial D$$

on the unit square, where $\partial/\partial n$ denotes the normal derivative. Replacing the differential operator Δ^2 by the familiar thirteen-point difference operator and using centered differences to approximate the normal derivative along the boundary, we are led to an $n \times n$ block penta-diagonal system where the diagonal blocks are penta-diagonal, the sub- and super-diagonal blocks are tridiagonal, and the remaining blocks are diagonal [7]. We shall refer to this system as the thirteen-point model problem on an $n \times n$ grid.

3. Variations on Gaussian Elimination: The Importance of Ordering

When systems of linear equations are formulated, there is usually some natural ordering of the variables and equations. Yet for any $N \times N$ permutation matrix P, Gaussian elimination applies equally well to the permuted system

$$PAP^t y = Pb, \quad Px = y$$

in which both the variables and equations have been permuted [2]. As we shall see in this section, the ordering has an extremely large effect on the storage and work required when we attempt to avoid storing and operating on zero elements, and indeed may dictate the form of symmetric Gaussian elimination to be used.

As an illustration, consider the five-point model problem on an $n \times n$ grid. With the natural ordering of unknowns and equations (see Figure 3.1a), the coefficient matrix A is best described as a band matrix; i.e. all the nonzero elements of A lie in a band about the diagonal:

$$a_{ij} \neq 0 \text{ implies } |i-j| \leq n$$

(see Figure 3.1b). Moreover, U is again a band matrix with the same bandwidth as A and nearly all the elements of the band of U are nonzero (see Figure 3.1c). Band elimination methods are just Gaussian elimination with the assumption that all elements within the band of A or U are nonzero and must be stored and operated on, but that all elements outside the band of A and U are zero and can be ignored. Such methods are easy to implement [11] and widely used.

Figure 3.1a Natural ordering for a 3×3 grid.

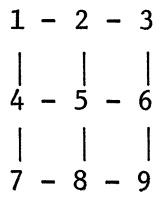

Figure 3.1b Corresponding zero structure of A for five-point model problem.

X	X	0	X	0	0	0	0	0
X	X	X	0	X	0	0	0	0
0	X	X	0	0	X	0	0	0
X	0	0	X	X	0	X	0	0
0	X	0	X	X	X	0	X	0
0	0	X	0	X	X	0	0	X
0	0	0	X	0	0	X	X	0
0	0	0	0	X	0	X	X	X
0	0	0	0	0	X	0	X	X

Figure 3.1c Band structure of U (30 elements).

X	X	X	X	0	0	0	0	0
	X	X	X	X	0	0	0	0
		X	X	X	X	0	0	0
			X	X	X	X	0	0
				X	X	X	X	0
					X	X	X	X
						X	X	X
							X	X
								X

With the diagonal ordering (see Figure 3.2a), the coefficient matrix A is best described as a variable bandwidth or profile matrix (see Figure 3.2b). Letting f_j^A denote the row index of the first nonzero element in the j-th column of A, we find that

$$a_{ij} \neq 0 \quad \text{and} \quad i \leq j \quad \text{imply} \quad f_j^A \leq i \leq j$$

(the set of elements $\{(i,j) \mid f_j^A \leq i \leq j\}$ is called the profile or envelope of A). The envelope of U is the same as the envelope of A and contains no zero entries (see Figure 3.2c). Envelope elimination methods have been developed analagous to band methods to avoid storing and operating on zero elements in A and U [6]. Note that there are fewer elements in the profile of U than in the band of U (every envelope matrix is a band matrix!) so that one would expect to need less storage and fewer arithmetic operations to factor A. However, whereas one number (namely the bandwidth) sufficed to represent the band structure of A or U, a vector of length N (namely the f_j^A's) is required to describe the envelope. In general, this trade-off of storage for U for storage for the structure of U is quite beneficial.

With the alternate diagonal ordering (see Figure 3.3a), the coefficient matrix A has no discernible structure (see Figure 3.3b). However, even though U also has no regular structure (see Figure 3.3c), it has significantly fewer nonzero elements than we obtained with either of the two previous orderings. But, to take advantage of this, we must store A and U as sparse matrices (i.e. store only the nonzeroes) and do (general symmetric) sparse elimination (i.e. keep track of nonzeroes as they occur during the elimination process and operate only on nonzeroes).

Figure 3.2a Diagonal ordering for a 3×3 grid.

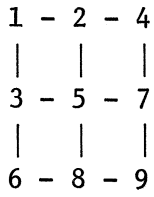

Figure 3.2b Corresponding zero structure of A for five-point model problem.

X	X	X	0	0	0	0	0	0
X	X	0	X	X	0	0	0	0
X	0	X	0	X	X	0	0	0
0	X	0	X	0	0	X	0	0
0	X	X	0	X	0	X	X	0
0	0	X	0	0	X	0	X	0
0	0	0	X	X	0	X	0	X
0	0	0	0	X	X	0	X	X
0	0	0	0	0	0	X	X	X

Figure 3.2c Envelope structure of U (28 elements).

X	X	X	0	0	0	0	0	0
	X	X	X	X	0	0	0	0
		X	X	X	X	0	0	0
			X	X	X	X	0	0
				X	X	X	X	0
					X	X	X	0
						X	X	X
							X	X
								X

Figure 3.3a Alternate diagonal ordering for a 3×3 grid.

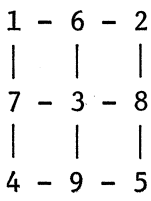

Figure 3.3b Corresponding zero structure of A for five-point model problem.

X	0	0	0	0	X	X	0	0
0	X	0	0	0	X	0	X	0
0	0	X	0	0	X	X	X	X
0	0	0	X	0	0	X	0	X
0	0	0	0	X	0	0	X	X
X	X	X	0	0	X	0	0	0
X	0	X	X	0	0	X	0	0
0	X	X	0	X	0	0	X	0
0	0	X	X	X	0	0	0	X

Figure 3.3c Zero structure of U (27 nonzero elements).

X	0	0	0	0	X	X	0	0
	X	0	0	0	X	0	X	0
		X	0	0	X	X	X	X
			X	0	0	X	0	X
				X	0	0	X	X
					X	X	X	X
						X	X	X
							X	X
								X

Again we have a trade-off, between storage for the nonzero elements of U and storage to describe the nonzero structure of U. Although the process is somewhat more complex than either band or envelope elimination, general sparse elimination can be implemented without great difficulty by preceding the numeric factorization (NUMFAC) with a symbolic factorization (SYMFAC) to determine which elements of U are nonzero. Details are given in a companion paper in these proceedings [4,13].

But if we are being forced to use sparse elimination, why not seek an ordering of the variables and equations that minimizes the work and storage? The work required to eliminate the k-th equation is proportional to the square of the number of nonzeroes in the k-th row of U. Thus, we could locally minimize the work by choosing at the k-th stage to eliminate the variable and corresponding equation, which leads to the smallest number of nonzeroes in the k-th row of U. This heuristic is known as the minimum degree algorithm [12] and it does indeed lead to a further savings (see Figures 3.4a-c).

To further illustrate the results of this section, we solved a series of sample model problems with different orderings using band, envelope, and general sparse elimination. In each case we counted the number of nonzeroes in U (or the number of elements in the band or profile of U, whichever was appropriate) and the total number of multiply-add pairs to factor A. The figures are given in Table 3.5. The important points to note are that general sparse elimination with the minimum degree ordering is always best and that envelope is never worse than band with the same ordering.

Figure 3.4a Minumum degree ordering for a 3×3 grid.

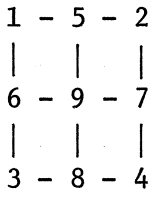

Figure 3.4b Corresponding zero structure of A for five-point model problem.

```
X  0  0  0  X  X  0  0  0
0  X  0  0  X  0  X  0  0
0  0  X  0  0  X  0  X  0
0  0  0  X  0  0  X  X  0
X  X  0  0  X  0  0  0  X
X  0  X  0  0  X  0  0  X
0  X  0  X  0  0  X  0  X
0  0  X  X  0  0  0  X  X
0  0  0  0  X  X  X  X  X
```

Figure 3.4c Zero structure of U (26 nonzero elements).

```
X  0  0  0  X  X  0  0  0
   X  0  0  X  0  X  0  0
      X  0  0  X  0  X  0
         X  0  0  X  X  0
            X  X  X  0  X
               X  X  X  X
                  X  X  X
                     X  X
                        X
```

Table 3.5 Number of elements in band or envelope of U or nonzeroes in U and number of multiply-adds to factor A for several model problems.

	Band Elimination		Envelope Elimination		Sparse Elimination	
16×16 Five-point:	Non-zeroes	Work	Non-zeroes	Work	Non-zeroes	Work
Natural	4,216	37,536	4,111	36,206	4,111	36,206
Diagonal	4,216	37,536	3,096	22,136	3,096	22,136
Alternate diagonal	--	--	--	--	2,200	13,442
Minimum degree	--	--	--	--	2,047	12,149
32×32 Five-point:						
Natural	33,264	562,496	32,799	551,646	32,799	551,646
Diagonal	33,264	562,496	23,344	307,056	23,344	307,056
Alternate diagonal	--	--	--	--	14,384	166,146
Minimum degree	--	--	--	--	12,031	131,893
16×16 Nine-point:						
Natural	4,455	41,838	4,336	40,151	4,336	40,151
Minimum degree	--	--	--	--	3,209	25,259
32×32 Nine-point:						
Natural	34,255	596,190	33,760	583,855	33,760	583,855
Minimum degree	--	--	--	--	18,855	245,346
16×16 Thirteen-point:						
Natural	7,920	131,648	7,724	126,909	7,724	126,909
Diagonal	7,920	131,648	5,890	78,308	5,890	78,308
Minimum degree	--	--	--	--	5,619	80,671
32×32 Thirteen-point:						
Natural	64,480	2,104,960	63,580	2,064,005	63,580	2,064,005
Diagonal	64,480	2,104,960	45,570	1,154,500	45,570	1,154,500
Minimum degree	--	--	--	--	38,865	1,052,964

4. Efficiency: The Innermost Loop

In Section 3, we saw that sparse Gaussian elimination can offer significant savings both in terms of storage (i.e. the number of nonzeroes in U) and work (i.e. the number of multiply-adds to factor A) over either band or envelope elimination. Details on how to implement the method are given in a companion paper in these proceedings [4; also 13]. In this section, we shall show that the implementation is efficient in the sense that the additional overhead (over band or envelope) per multiply-add and the pointer storage per nonzero element of U are both relatively small.

As is often observed empirically [10], the bulk of the running time of a program is spent in its innermost loops. Thus we begin by examining the innermost loops from the several different variations of Gaussian elimination (see Figure 4.1). By way of comparison, we have also presented the loop from an inner-product routine. Each loop is written in an extended version of FORTRAN-IV accepted by most existing compilers and is expressed in a most natural form (as opposed to the form that a particular compiler would translate into the most nearly optimal machine code). The loops are arranged in order of increasing complexity but all are very much similar. To compare the efficiencies of these program fragments abstractly, we must consider how to implement them in machine code for a "typical" modern computer. We shall assume that there are enough arithmetic and index registers available to keep all but subscripted variables in registers throughout execution of the

Figure 4.1 Innermost loops from several variations of Gaussian elimination.

a) Inner-product routine

```
          DO 1 J=JMIN,JMAX
1           SUM = SUM + A(J)*B(J)
```

b) Envelope elimination or band elimination (inner product form)

```
          DO 1 J=JMIN,JMAX
1           SUM = SUM + U(MUI+J) * U(J+MUK)
```

c) Band elimination (outer product form)

```
          DO 1 J=JMIN,JMAX
1           U(MUI+J) = U(MUI+J) + UIK * U(MUK+J)
```

d) Sparse elimination

```
          DO 1 J=JMIN,JMAX
1           D(JU(MU+J)) = D(JU(MU+J)) + UKI * U(J)
```

Figure 4.2 Pseudo-machine-code corresponding to the different inner loops.

```
a)  LOAD    R1,A-1(J)          ;FETCH A(J)
    FMLT    R1,B-1(J)          ;MULTIPLY BY B(J)
    FADD    R2,R1              ;ADD TO RUNNING SUM IN R2
   <LOOP>

b)  LOAD    R1,U+MUI-1(J)      ;FETCH U(MUI+J)
    FMLT    R1,U+MUK-1(J)      ;MULTIPLY BY U(J+MUK)
    FADD    R2,R1              ;ADD TO RUNNING SUM IN R2
   <LOOP>

c)  LOAD    R1,U+MUK-1(J)      ;FETCH U(MUK+J)
    FMLT    R1,UIK             ;MULTIPLY BY UIK
    FADD    R1,U+MUI-1(J)      ;ADD U(MUI+J)
    STORE   R1,U+MUI-1(J)      ;SAVE RESULT AS U(MUI+J)
   <LOOP>

d)  LOAD    R1,JU+MU-1(J)      ;FETCH JU(MU+J)
    LOAD    R2,U-1(J)          ;FETCH U(J)
    FMLT    R2,UKI             ;MULTIPLY BY UKI
    FADD    R2,D-1(R1)         ;ADD IN D(JU(MU+J))
    STORE   R2,D-1(R1)         ;SAVE RESULT AS D(JU(MU+J))
   <LOOP>
```

loop, and shall ignore the common loop-control instruction(s). The corresponding machine code is given in Figure 4.2. Since each loop contains a floating point multiply-add pair, the important points to note are the number of fixed point instructions needed to calculate element addresses, the number of core accesses, and the total number of instructions (with loop-control counted as a single instruction). These figures are presented in Table 4.3. Without specifying instruction times, we can make only qualitative statements about the respective running times: Sparse elimination should cost more per multiply-add than the other variations (but not much more); profile elimination should cost no more than band elimination.

To make quantitative statements, we now restrict our attention to a particular machine, the IBM 370/158, and a particular compiler, the FORTRAN-IV Level-H-Extended optimizing compiler. Each innermost loop was hand-coded in machine language to be as efficient as possible and the running time for each execution of the loop calculated from the timing manual. Next each program was compiled and the same calculation made for the compiler-generated code; the difference between these values is the cost of programming in a high-level language. Last, the algorithms were timed on the 16×16 and 32×32 five-point model problems. The total time divided by the total number of multiply-adds gives the cost per multiply-add, or equivalently the actual cost per execution of the innermost loop; the difference between this and the calculated time for the compiled code is just the overhead associated with the remainder of the algorithm. These figures are presented in Table 4.4. As expected,

Table 4.3 Qualitative comparison of the pseudo-machine-code loops.

	(a)	(b)	(c)	(d)
Number of fixed point instructions	1	1	2*	3*
Number of core references	2	2	3*	4*
Total number of instructions†	4	4	5*	6*

* One fewer if machine has floating point add-to-memory instruction.

† Counting loop control as a single instruction.

Table 4.4 Quantitative comparison of the innermost loops for the IBM 370/158

	Optimum Code (Estimated)	Compiled Code (Estimated)	Compiled Code (Executed)*	Total Execution Time*
16×16 Five-point				
Diagonal ordering				
SYMFAC	--	--	--	0.11 sec
Band elimination	7.1 μs	10.2 μs	10.0 μs	0.38 sec
Envelope elimination	6.3 μs	9.3 μs	11.3 μs	0.25 sec
Sparse elimination	8.1 μs	10.7 μs	14.5 μs	0.32 sec
Minimum degree ordering				
SYMFAC	--	--	--	0.09 sec
Sparse elimination	8.1 μs	10.7 μs	15.6 μs	0.19 sec
32×32 Five-point				
Diagonal ordering				
SYMFAC	--	--	--	0.75 sec
Band elimination	7.1 μs	10.2 μs	10.0 μs	5.62 sec
Envelope elimination	6.3 μs	9.3 μs	10.3 μs	3.16 sec
Sparse elimination	8.1 μs	10.7 μs	12.6 μs	3.87 sec
Minimum degree ordering				
SYMFAC	--	--	--	0.38 sec
Sparse elimination	8.1 μs	10.7 μs	13.3 μs	1.76 sec

* Times are accurate to within only 10-20%.

Table 4.5 Additional storage required for sparse elimination.

	Nonzeroes in U	Total Storage	(Additional) Pointers per Nonzero
16x16 Five-point			
Natural	4,111	8,871	1.16
Diagonal	3,096	6,841	1.21
Alternate diagonal	2,200	5,049	1.29
Minimum degree	2,047	4,218	1.06
32x32 Five-point			
Natural	32,799	68,175	1.08
Diagonal	23,344	49,625	1.11
Alternate diagonal	14,384	31,345	1.18
Minimum degree	12,031	21,287	0.77
16x16 Nine-point			
Natural	4,336	9,081	1.09
Minimum degree	3,209	5,587	0.74
32x32 Nine-point			
Natural	33,760	69,105	1.05
Minimum degree	18,855	29,241	0.55
16x16 Thirteen-point			
Natural	7,724	15,721	1.04
Diagonal	5,890	12,053	1.05
Minimum degree	5,619	8,538	0.52
32x32 Thirteen-point			
Natural	63,580	128,217	1.02
Diagonal	45,570	92,197	1.02
Minimum degree	38,865	52,065	0.34

sparse elimination is much faster for these problems although the time per multiply-add is somewhat higher, reflecting the smaller operation counts.

To estimate the additional pointer storage required for sparse

elimination, we measured the total storage required and divided this by the number of nonzero elements in D and U for each of the problems considered at the end of Section 3 (see Table 4.5). As we can easily see, the additional pointer storage per nonzero element of U is quite small, even though a full word is used for each pointer.

5. Extensions and Conclusions

In the preceding sections, we have seen that sparse elimination offers significant savings in the number of nonzeroes in U and the number of arithmetic operations to factor A. Indeed, for all our model problems on an $n \times n$ grid, it can be shown that sparse elimination with the appropriate variable ordering needs only $0(n^2 \log n)$ storage and $0(n^3)$ work [8], whereas band or envelope elimination requires at least $0(n^3)$ storage and $0(n^4)$ work for any ordering [3,8]. Moreover, the added overhead in implementing sparse elimination is very small, so we conclude that it is preferable for all but very small or very dense problems. Since it does no more work, requires no more storage for U, and has no more overhead, we also conclude that envelope elimination is always preferable to band elimination.

There remains the original question of whether one should be using Gaussian elimination in any form versus an iterative method such as Symmetric Successive Over-Relaxation (SSOR) [14] or the Conjugate Gradient method combined with Strongly ImPlicit factorizations (CG-SIP) [2]. For the five- and nine-point model problems on an $n \times n$ grid,

these methods require $O(\sqrt{n} \log n)$ iterations to reduce the initial error by a factor of h^2 [2]. Since each iteration requires only $O(n^2)$ multiply-adds, the total operation count is $O(n^2 \sqrt{n} \log n)$ with $O(n^2)$ storage, clearly superior asymptotically to even sparse elimination. However, for the corresponding thirteen-point model problem, $O(n \log n)$ iterations are required and sparse elimination is preferable. Nonasymptotically, one can only look at the actual operation counts and run times to choose between these methods. (See Table 5.1.)

Two other points must be made. First, for grid problems in two dimensions, one can reduce the storage from $O(n^2 \log n)$ to $O(n^2)$ at the cost of doubling the work [5,13]. Thus one need not automatically choose an iterative method because of storage considerations. Second, for the corresponding problems in three dimensions, one should probably not use direct methods: It can be shown that Gaussian elimination requires $O(n^6)$ work and $O(n^4)$ storage for any ordering [3,8], whereas SSOR or CG-SIP requires only $O(n^3 \sqrt{n} \log n)$ work and $O(n^3)$ storage for the analogs of the five- and nine-point model problems and $O(n^4 \log n)$ work and $O(n^3)$ storage for the analog of the thirteen-point model problem [3].

Table 5.1 Comparison of Direct and Iterative Methods for the Five-Point Model Problem

	Storage:			
	16 × 16	32 × 32	48 × 48	64 × 64
Sparse (Minimum degree)	4,218	21,287	53,058	100,690
CGSIP (Poisson)	1,280	5,120	11,520	20,480
CGSIP (General)	2,496	10,112	22,848	40,704
	Work:			
	16 × 16	32 × 32	48 × 48	64 × 64
Sparse (Minimum degree)	15,987	154,931	522,524	1,248,748
CGSIP (Poisson)	29,965	185,365	490,777	1,036,318
CGSIP (General)	55,644	347,308	922,612	1,949,502

Bibliography

1. G. Dahlquist and A. Björck. *Numerical Methods*. Prentice-Hall, Englewood Cliffs, New Jersey, 1974.

2. R. Chandra, S.C. Eisenstat, and M.H. Schultz. Conjugate Gradient methods for partial differential equations. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975.

3. S.C. Eisenstat. Complexity bounds for Gaussian elimination. To appear.

4. S.C. Eisenstat, M.H. Schultz, and A.H. Sherman. Efficient implementation of sparse symmetric Gaussian elimination. Proceedings of the AICA International Symposium on Computer Methods for Partial Differential Equations, Bethlehem, Pennsylvania, 1975.

5. S.C. Eisenstat, M.H. Schultz, and A.H. Sherman. Direct methods for the solution of sparse systems of linear equations with limited core storage. Presented at SIAM Fall Meeting, Alexandria, Virginia, 1974.

6. S.C. Eisenstat and A.H. Sherman. Subroutines for envelope solutions of sparse linear systems. Yale Computer Science Research Report #35, 1974.

7. G.E. Forsythe and W.R. Wasow. *Finite Difference Methods for Partial Differential Equations*. John Wiley & Sons, New York, 1960.

8. J.A. George. Nested dissection of a regular finite element mesh. *SINUM* 10:2, 1973.

9. F.G. Gustavson. Basic techniques for solving sparse systems of linear equations. In Rose and Willoughby, *Sparse Matrices and Their Applications*, Plenum Press, New York, 1972.

10. D.E. Knuth. An empirical study of FORTRAN programs. *Software: Practice and Experience* 1:2, 1971.

11. R.S. Martin and J.H. Wilkinson. Solution of symmetric and unsymmetric band equations and the calculation of eigenvectors of band matrices. *Numerische Mathematik* 9:2, 1967.

12.

12. D.J. Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. In Read, *Graph Theory and Computing*, Academic Press, New York, 1972.

13. A.H. Sherman. Ph.D. dissertation, Yale University, 1975.

14. D.M. Young. *Iterative Solution of Large Linear Systems*. Academic Press, New York, 1971.