NASA Contractor Report 181860

ICASE Report No. 89-37

ICASE

BUNCH-KAUFMAN FACTORIZATION FOR REAL SYMMETRIC INDEFINITE BANDED MATRICES

Mark T. Jones

Merrell L. Patrick

(NASA-CR-181860) BUNCH-KAUFMAN FACTORIZATION FOR REAL SYMMETRIC INDEFINITE BANDED MATRICES Final Report (ICASE) 15 p CSCL 12A

N89-28341

Unclas

G3/64 0224027

Contract No. NAS1-18605

May 1989

Institute for Computer Applications in Science and Engineering

NASA Langley Research Center

Hampton, Virginia 23665-5225

Operated by the Universities Space Research Association

National Aeronautics and Space Administration

Langley Research Center

Hampton, Virginia 23665-5225

Recently, ICASE has begun differentiating between reports with a mathematical or applied science theme and reports whose main emphasis is some aspect of computer science by producing the computer science reports with a yellow cover. The blue cover reports will now emphasize mathematical research. In all other aspects the reports will remain the same; in particular, they will continue to be submitted to the appropriate journals or conferences for formal publication.

Bunch-Kaufman Factorization for Real Symmetric Indefinite Banded Matrices

Mark T. Jones*and Merrell L. Patrick*†

Abstract

The Bunch-Kaufman algorithm for factoring symmetric indefinite matrices has been rejected for banded matrices because it destroys the banded structure of the matrix. Herein, it is shown that for a subclass of real symmetric matrices which arise in solving the generalized eigenvalue problem using Lanczos's method, the Bunch-Kaufman algorithm does not result in major destruction of the bandwidth. Space time complexities of the algorithm are given and used to show that the Bunch-Kaufman algorithm is a significant improvement over LU factorization.

*Department of Computer Science, Duke University, Durham, NC 27706

†This research was supported by the National Aeronautics and Space Administration under NASA contract Nos. NAS1-18107 and NAS1-18605 and the Air Force Office of Scientific Research under AFOSR grant No. 88-0117 while the authors were in residence at ICASE. Additional support was provided by NASA grant No. NAG-1-466.

1 Introduction

The Bunch-Kaufman algorithm is considered one of the best methods for factoring full, symmetric, indefinite matrices [BK77], [BG76]. It has also been modified and successfully used to factor sparse matrices [DRMN79]. However, to date it has been rejected for banded, symmetric indefinite matrices because it destroys the banded structure of the matrix [BK77]. Herein it is shown that for a subclass of real symmetric indefinite matrices, which arise in solving the generalized eigenvalue problem using Lanczos's method, the Bunch-Kaufman algorithm does not result in major destruction of the bandwidth. Furthermore, for our class of problems, the Bunch-Kaufman factorization algorithm is a significant improvement over LU factorization, the standard of comparison for such methods [BK77]. In addition to taking advantage of symmetry, the Bunch-Kaufman algorithm yields the inertia of the matrix essentially for free [BK77], which is important in eigenvalue calculations. LU factorization does not yield the inertia as a by-product and destroys the symmetry of the matrix, thus increasing storage requirements for its implementation.

In section 2 we give one of the several variations of the Bunch-Kaufman algorithm and in section 3 describe a subclass of matrices to which we apply it. An efficient implementation of the method is described in section 4 and the space/time complexity of the implementation is disussed in section 5. Conclusions are drawn in section 6.

2 The Bunch-Kaufman Algorithm

The Bunch-Kaufman algorithm factors A, an $n \times n$ real symmetric indefinite matrix, into LDL^T while doing symmetric permutations on A to maintain stability, resulting in the following equation:

$$PAP^T = LDL^T. \tag{1}$$

Although several variations of the algorithm exist, algorithm D of the Bunch-Kaufman paper is the least destructive of the banded structure [BK77]. The algorithm is shown in figure 1.

1) for $i = 1, n$
begin
2) if the previous step was not a 2x2 pivot then
begin
3) $\lambda = \max_{j=i+1,n} \mid a_{j,i} \mid$
4) set r to the row number of this value
5) if $\lambda \alpha < \mid a_{i,i} \mid$ then
begin
6) perform a 1x1 pivot
end
else
begin
7) $\sigma = \max_{j=i+1,n} \mid a_{r,j} \mid$
8) if $\alpha \lambda^2 < \sigma \mid a_{i,i} \mid$ then
begin
9) perform a 1x1 pivot
end
else
begin
10) exchange rows and columns r and $i + 1$
11) perform a 2x2 pivot
end
end
end
12) end
13) if inertia is desired, then scan the D matrix

Figure 1: The Bunch-Kaufman Factorization Algorithm

The parameter, α, is chosen such that stability is maximized and has been shown by Bunch and Kaufman to be approximately 0.525 [BK77]. The exchange of rows and columns in step 10 maintains the symmetry of the matrix, unlike LU factorization which destroys the symmetry of the matrix by permuting only rows.

3 Applicable Set of Matrices

Bunch and Kaufman show that, in general, if m is the semi-bandwidth of a matrix being factored, then a 2x2 pivot can increase the semi-bandwidth from m to $(2m) - 2$ and that this can happen at every step thus resulting in the complete destruction of the band structure due to fill-in outside the band [BK77]. However, it will be shown in section four that for a subclass of matrices this fill-in can be controlled. The number of 2x2 pivots is bounded above by the number of negative eigenvalues of A, because each 2x2 pivot represents a positive-negative eigenvalue pair [BK77]. Also, the increase of the semi-bandwidth from m to $(2m) - 2$ is a worst case that in practice is not likely to occur. Therefore, for matrices with a small number of negative eigenvalues (in relation to the size of the matrix), it is possible to use Bunch-Kaufman factorization with very little fill-in. Such matrices arise in eigenvalue calculations where the smallest eigenvalues are sought. Methods such as inverse iteration and Lanczos's method are often used to find the smallest eigenvalues of a matrix, A. To do so, they often require the factorization of a matrix, $(A - \sigma I)$, where σ is normally very near the left end of A's spectrum, but may not be to the left of the smallest eigenvalue, thus the matrix is indefinite [NOPT83] but has only a small number of negative eigenvalues. These matrices can be banded, as they are in structural mechanics [BH87]. The difficulty is that the location and amount of the fill-in outside the band is not possible to predict a priori. In the following section, a detailed algorithm which dynamically allows for fill-in during factorization will be presented.

4 Efficient Implementation of the Algorithm

```
.   .                                                  .   .
.   .                                                  .   .
• • • • • x                      ← row k →      • • • • • x
0 • • • • x x                  ← row k+1 →      0 • • • • x x
0 0 • • • x x x                                 0 0 • • • x x x
0 0 0 • • x x x x                               0 0 0 • • x x x x
0 0 0 0 • x x x x x                             0 0 0 0 • x x x x x
0 0 0 0 0 x x x x x x            ← row r →      0 0 0 0 0 x x x x x x
0 0 0 0 0 0 x x x x x x                         0 0 0 0 0 f x x x x x x
0 0 0 0 0 0 0 x x x x x x                       0 0 0 0 0 f f x x x x x x
0 0 0 0 0 0 0 0 x x x x x x                     0 0 0 0 0 f f f x x x x x x
0 0 0 0 0 0 0 0 0 x x x x x x                   0 0 0 0 0 f f f f x x x x x x
0 0 0 0 0 0 0 0 0 0 x x x x x x                 0 0 0 0 0 f f f f f x x x x x x
0 0 0 0 0 0 0 0 0 0 0 x x x x x x               0 0 0 0 0 0 0 0 0 0 0 x x x x x x
.   .   .                                       .   .   .
.   .   .                                       .   .   .
```

Figure 2: Example of Fill-in (Note: this is an example of worst case fill-in)

As the following algorithm is executed the original matrix is copied, piece-wise, from one place in memory to another. This allows for dynamic allocation of fill-in as well as only requiring part of the matrix to be in main memory at any particular time. Fill-in only takes place in a small triangle when a 2x2 pivot occurs. If a pivot occurs at step k, this triangle is of the form shown in figure 2, where •'s represent eliminated elements in L, x's are uneliminated non-zeros, 0's are zeros outside the band for which no storage is needed, and fs are areas where fill occurs. The triangle of fill is from row $r+1$ to row $r+m$, where m is the semi-bandwidth (this area may already contain non-zeros depending on the value of r, so no extra memory may be needed). The algorithm is as follows:

0) set *upto* to 0
1) for $i = 1, n$
   begin
2)   if the previous step was not a 2x2 pivot then
      begin
3)   read rows *upto* to $\min(n, i+m)$ of the matrix A into L,

no extra space for fill needs to be added for these rows

4) set *upto* to $\min(n, i + m)$

5) $\lambda = \max_{j=i+1,upto} \mid a_{j,i} \mid$

6) set r to the row number of this value

7) if $\lambda\ \alpha \le \mid a_{i,i} \mid$ then

begin

8) go to 11

end

9) $\sigma = \max_{j=i+1,upto} \mid a_{r,j} \mid$

C (it may be necessary to access some elements that are not

C read in at this point, but the number of elements is small,

C so they may be read into L or simply discarded,

C this is only a concern if i/o is taking place)

10) if $\alpha\ \lambda^2 \le \sigma \mid a_{i,i} \mid$ then

begin

C perform a 1x1 pivot

11) set $p_i = i$

12) set $d_{i,i} = a_{i,i}$

13) set $d_{i,i+1} = 0.0$

14) set $a_{i,i} = 1.0$

15) for $j = i + 1$, *upto*

begin

16) $v_j = a_{j,i}$

17) $vl_j = a_{j,i}/a_{i,i}$

18) $a_{j,i} = vl_j$

end

19) for $j = i + 1$, *upto*

begin

20) for $k = i + 1$, j

begin

21) $a_{j,k} = a_{j,k} - vl_j v_k$

end

end

end

else

begin

C permute the matrix and then perform a 2x2 pivot

22) read rows *upto* to $\min(n,r+m)$ of the matrix A into L, and allocate space for the fill triangle

23) set *upto* to $\min(n,r+m)$

24) exchange rows and columns r and $i+1$

25) set $p_i = i$

26) set $p_{i+1} = r$

27) set $d_{i,i} = a_{i,i}$

28) set $d_{i+1,i+1} = a_{i+1,i+1}$

29) set $d_{i,i+1} = a_{i+1,i}$

30) set $d_{i+1,i+2} = 0.0$

31) set determinant $= (((d_{i,i}d_{i+1,i+1})/d_{i,i+1}) - d_{i,i+1})d_{i,i+1}$

32) for $j = i+2$, *upto*

begin

33) $v_j = a_{j,i}$

34) $v2_j = a_{j,i+1}$

35) $vl_j = a_{j,i}d_{i+1,i+1} - a_{j,i+1}d_{i,i+1}$

36) $vl2_j = -a_{j,i}d_{i,i+1} + a_{j,i+1}d_{i,i}$

37) $a_{j,i} = vl_j$

38) $a_{j,i+1} = vl2_j$

end

39) for $j = i+2$, *upto*

begin

40) for $k = i+2, j$

```
            begin
41)             a_{j,k} = a_{j,k} - (vl_j v_k + vl2_j v2_k)
            end
          end
        end
      end
    end
```

P is a vector representing the permutation matrices. The only time fill outside the band occurs is in step 24 of the algorithm when a 2x2 pivot occurs and then storage for the fill is allocated dynamically.

5 Speed and Storage Analysis

In this section we compare the space/time requirements of our implementation of the Bunch-Kaufman algorithm with LU factorization. The storage requirements for both algorithms will be analyzed for two different situations: 1) when simply factoring a matrix that falls in the subclass described in section 2, and 2) when factoring a matrix pencil such as $(K - \sigma M)$ where K and M are symmetric, K is positive definite and σ is near the left end of K's spectrum.

In the first situation, the storage required by the algorithm presented in section 4 is significantly less than that required by LU factorization for the set of matrices that was described in section 2. The storage required by LU factorization is approximately $3mn$ [BK77]. The storage needed by this implementation of Bunch-Kaufman is mn for the original storage from which the matrix is copied, plus mn for the locations to which the matrix is copied, plus an additional amount C which is the amount of storage necessary for the fill-in triangles. C is much less than mn, because of the small number of negative eigenvalues. In addition, two vectors of length n are needed for storing the D matrix giving a total of $2n(m+1)+C$. So when C is small, approximately $(m-2)n$ storage locations are saved factoring matrices using the Bunch-Kaufman algorithm instead of LU factorization.

In the second situation (which arises in an efficient implementation of Lanczos's method for solving $Kx = \lambda Mx$), the shift σ may change during

Method	adds.	mults.	divisions	sq. roots	comps.	fill
Chol.	44433080	48140336	1824	1824	0	0
B-K	48277445	48686784	1831	0	446326	2083
LU	137241687	137648943	1823	0	409079	$2mn$

Figure 3: Operation Counts for Factorization: n=1824, m=240, 5 negative eigenvalues

Method	adds.	mults.	divisions	sq. roots	comps.	fill
Chol.	44433080	48140336	1824	1824	0	0
B-K	52023663	52445756	1837	0	485452	14837
LU	137412094	137819350	1823	0	409079	$2mn$

Figure 4: Operation Counts for Factorization: n=1824, m=240, 19 negative eigenvalues

execution of the algorithm, so K and M must be saved throughout the computation. In this situation, the storage requirements for LU factorization is increased to $(4mn)$, but the storage needed by Bunch-Kaufman remains the same, namely $2n(m+1)+C$ making it even more attractive in this case.

The operation counts for factorization are the same in both cases. The operation count for Bunch-Kaufman is significantly less than that of LU factorization because symmetry is exploited and the fill-in is limited. For simplicity, the operations added by the fill-in during Bunch-Kaufman are ignored, since the amount that is added is trivial. The high order term in the operation counts for Bunch-Kaufman is approximately nm^2 arithmetic operations plus approximately nm comparisons while the high order term for the operation counts for LU is approximately $4nm^2$ arithmetic operations plus approximately nm comparisons.

The Bunch-Kaufman method also vectorizes well if the semi-bandwidth is large enough. The gains from vectorization are much the same as those

Method	adds.	mults.	divisions	sq. roots	comps.	fill
Chol.	3321051	3434180	1980	1980	0	0
B-K	3322513	3435664	1985	0	142978	22
LU	10342067	10455196	1979	0	115108	$2mn$

Figure 5: Operation Counts for Factorization: n=1980, m=59, 5 negative eigenvalues

Method	adds.	mults.	divisions	sq. roots	comps.	fill
Chol.	3321051	3434180	1980	1980	0	0
B-K	3324670	3437856	1985	0	152370	57
LU	10618321	10731450	1979	0	115108	$2mn$

Figure 6: Operation Counts for Factorization: n=1980, m=59, 15 negative eigenvalues

N	M	No. of neg. Eigenvalues	No. of 2x2 pivots
1824	240	5	4
1824	240	19	7
1980	59	15	3
1980	59	5	3

Figure 7: The number of 2x2 pivots for each problem

for Choleski factorization.

The operations counts for both types of factorization, as well as Choleski factorization, when using Lanczos's method for solving the generalized eigenvalue problem are given in figures 3, 4, 5, and 6. The fill-in during factorization is also shown in these figures. The amount of fill-in when using Bunch-Kaufman can be seen to increase when the number of negative eigenvalues increases. The implementation of LU factorization that is used for the comparison is sgbfa from the Linpack package [DBMS78]. The measurements for Choleski factorization are given only as a reference point, the matrices that were solved were shifted to make them positive definite for the Choleski factorization runs, otherwise Choleski factorization would have failed due to the indefiniteness of the system. These matrices arise from a problem in a structural engineering application [BH87]. In figure 7 the number of $2x2$ pivots that occurred in each problem can be examined.

The solution phase that occurs after factorization takes slightly longer for Bunch-Kaufman than for LU factorization due to the fact that three matrices, L, D, and L^t, arise from Bunch-Kaufman (see equation 1) rather than just two matrices, L and U, that arise from LU factorization. This solution phase however takes much less time than factorization, so this is not significant.

6 Conclusions

The Bunch-Kaufman method has been shown to be a more efficient factorization method than LU factorization in terms of time and storage for banded real symmetric indefinite matrices with a small number of eigenvalues. An algorithm has been presented that greatly limits the fill needed for factorization as well as taking advantage of the symmetry of the matrix. This method has been shown to be nearly as stable as LU factorization by Bunch [BK77].

References

[BG76] Victor Barwell and Alan George. A comparison of algorithms for solving symmetric indefinite systems of linear equations.

ACM Transactions on Mathematical Software, 2(3):242–251, September 1976.

[BH87] Charles P. Blankenship and Robert J. Hayduk. Potential supercomputer needs for structural analysis. Presentation at the Second International Conference on Supercomputing, May 3-8 1987. Santa Clara, CA.

[BK77] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, January 1977.

[DBMS78] J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart. *LINPACK User's Guide*, 1978.

[DRMN79] I. S. Duff, J. K. Reid, N. Munksgaard, and H. B. Nielsen. Direct solution of sets of linear equations whose matrix is sparse, symmetric and indefinite. *J. Inst. Maths. Applics.*, 23:235–250, 1979.

[NOPT83] Bahram Nour-Omid, Beresford N. Parlett, and Robert L. Taylor. Lanczos versus subspace iteration for solution of eigenvalue problems. *International Journal for Numerical Methods in Engineering*, 19:859–871, 1983.

NASA National Aeronautics and Space Administration

Report Documentation Page

1. Report No. NASA CR-181860 ICASE Report No. 89-37	2. Government Accession No.	3. Recipient's Catalog No.
4. Title and Subtitle Bunch-Kaufman Factorization for Real Symmetric Indefinite Banded Matrices		5. Report Date May 1989
		6. Performing Organization Code
7. Author(s) Mark T. Jones, Merrell L. Patrick		8. Performing Organization Report No. 89-37
		10. Work Unit No. 505-90-21-01
9. Performing Organization Name and Address Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, VA 23665-5225		11. Contract or Grant No. NAS1-18605
		13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration, Langley Research Center, Hampton, VA 23665-5225		14. Sponsoring Agency Code

15. Supplementary Notes

Langley Technical Monitor:
Richard W. Barnwell

Journal of Approximation Theory

Final Report

16. Abstract

The Bunch-Kaufman algorithm for factoring symmetric indefinite matrices has been rejected for banded matrices because it destroys the banded structure of the matrix. Herein, it is shown that for a subclass of real symmetric matrices which arise in solving the generalized eigenvalue problem using Lanczos's method, the Bunch-Kaufman algorithm does not result in major destruction of the bandwidth. Space time complexities of the algorithm are given and used to show that the Bunch-Kaufman algorithm is a significant improvement over LU factorization.

17. Key Words (Suggested by Author(s)) symmetric, indefinite, banded matrices, Bunch-Kaufman algorithm	18. Distribution Statement 64 - Numerical Analysis; Unclassified - Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	14	A03

NASA FORM 1626 OCT 86