JOURNAL OF THEORETICAL
AND APPLIED MECHANICS
4, **37**, 1999

NEW DESIGN OF PIVOTED BANDED LINEAR EQUATION SOLVER VERSUS CLAPACK LIBRARY ROUTINES

MAREK M. STABROWSKI

Electrical Engineering Department, Lublin University of Technology
e-mail: mmst@bravo.pol.lublin.pl

A new method of pivoting applicable to banded unsymmetric linear equation systems has been introduced. It limits the fill-in, preserves basic band structure and makes full use of allocated memory. Two new features of this method include multiple pivoting and the threshold factor use for pivoting decision. The stability or rather the solution precision has been investigated basing on the examples of classic ill conditioned equation system (Hilbert) and quasi-random system with still more difficult conditioning characteristic. Comprehensive test results have been presented in the easily readable graphic form. They prove the overall efficiency of new solvers and compare them favourably with the software from CLAPACK library.

Key words: banded unsymmetric linear equation systems, partial pivoting

1. Introduction

The paper presents new algorithms of linear equation solving, aimed at large unsymmetric banded systems. The solvers feature exceptional stability outperforming other similar software (cf Demmel, 1997; Engeln-Muellges and Uhlig, 1996). The stability is maintained also in out-of-core version of the solvers in a wide range of equation system sizes.

The solvers of such type are developed in order to handle efficiently large banded unsymmetric linear equation systems without diagonal dominance (cf Demmel, 1997; Engeln-Muellges and Uhlig, 1996; George and Liu, 1981). Systems of this type may arise in the non-linear problems in diverse science and technology fields (FEM, FDM, ODE) (cf Crotty, 1982; Demmel, 1997; George and Liu, 1981; Rigby and Aliabadi, 1995; Stabrowski, 1998). The lack of

diagonal dominance makes pivoting absolutely necessary in most cases. It should be observed at the very beginning that the pivoting leads to more or less extensive filling of zeros outside of the original non-zero band. The bandwidth grows then, increasing the memory requirements of the solver. It is, along with numerical stability, most critical problem in the design of the solvers for this class of problems.

There exist several more or less popular libraries of numerical software offering the tools for solving of these problems. Most comprehensive library in the field of linear algebra is the LAPACK/CLAPACK library (cf Demmel, 1997) – a successor to renowned older LINPACK library (cf Bjoerck and Dahlquist, 1974; Demmel, 1997). It includes the routine for solving of banded linear equation systems with partial pivoting usage in the form of **sgbsv/dgbsv** (single/double precision) routine with associated co-routines and auxiliary BLAS library (cf Demmel, 1997). However, it should be never taken for granted that the tools from this library or from the other ones (cf Engeln-Muellges and Uhlig, 1996) are absolutely reliable and fully tested. The research reported in this paper has proved quite the contrary with respect to the **sgbsv/dgbsv** routines. At the same time reliable and numerically stable alternatives to the CLAPACK routines will be presented.

Fundamental characteristics of reliable software should include:

- numerical stability for ill-conditioned (in a broad sense) cases;
- low consumption of computer resources, i.e. of operating memory and processor time;
- optimised out-of-core operation for solving large problems.

There is little room for improvement with respect to basic solution method. However seemingly minor details of algorithm implementation can influence dramatically all the three above mentioned characteristics of reliable software. Some time ago, the author of the present paper, has developed two out-of-core solvers based on the Crout method (cf Stabrowski, 1998). This method seemed very promising. In the first place, compact elimination formulas justified the hope of reducing the computation time, while maintaining good numerical stability. More thorough further investigation has led to the conclusion that it was overly optimistic assumption. Next, careful analysis of memory requirements and especially confrontation with the CLAPACK library has led to the conclusion that the Gauss method may be more useful in the design of high performance pivoted solvers of banded systems. Nevertheless, the research carried up to date (cf Stabrowski, 1998) has been very fruitful, leading

to extremely important and fully documented conclusion that direct rows interchange during pivoting is more profitable (in the sense of processor time) than indirect indexed addressing based on pivoting information.

The present paper will be focused on new design of pivoted banded solver using the Gauss method with several important implementation enhancements. They include quite common implicit scaling as well as multi-pivoting and thresholding. These new techniques will be extensively tested experimentally. New solvers will be compared with the counterpart from the CLAPACK library. Special attention has been devoted to the problems of numerical stability. As the test systems two examples of banded linear equation systems have been used. The first one is a system with quasi-random coefficient matrix and slight, if any, diagonal dominance. The second one is a classic ill-conditioned Hilbert system or rather a banded version of such a system.

2. Memory management in Gauss and Crout banded solvers

The Gauss and Crout methods are algebraically equivalent but there are distinct differences in purely numerical performance and in memory consumption. Differences in numerical performance are quite obvious, as in both methods computations are made in a different order. Also finite precision of computers and possible ill conditioning of the system to be solved may enhance the differences in computation results. Different pivoting decisions (the reasons will be pointed out latter) influence also numerical results. However, this time another problem – that of memory usage and management – will be analysed.

It should be noted, at first, that pivoting and row interchange causes expansion of non-zero band. In the Crout method this expansion is limited to twice the original bandwidth (cf Stabrowski, 1998); the non-zero band expands leftward and rightward. The largest array for system coefficients storing sholud be dimensioned in in-core Crout solver as

$$N_{CI} = 4n_{band}n_{eq} \tag{2.1}$$

where

n_{band} – semi-bandwidth

n_{eq} – number of equations.

For the out-of-core solver, initial assumption about the size of computational window should be made at first. It is quite natural to assume that

computations in n_{band} rows should be made before swapping in and out to the disc memory some part of the already processed non-zero band. Under such assumption the out-of-core Crout solver consumes for its largest array approximately

$$N_{CO} = 16n_{band}^2 \tag{2.2}$$

For another choice of the number of processed rows, e.g. n_{rows} before resorting to disc memory support, this parameter is equal to

$$N_{CO} = 4n_{band}(3n_{band} + n_{rows}) \tag{2.3}$$

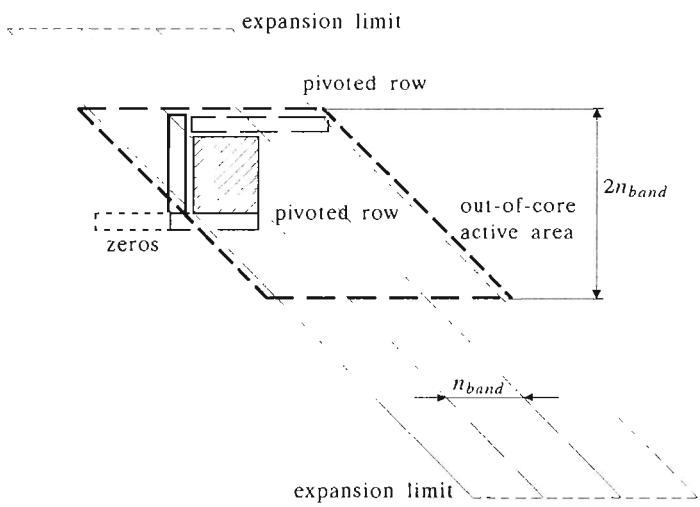

Fig. 1. Active front and out-of-core window in the Gauss banded pivoted solver

All these parameters are lower in the case of Gauss solver. At first it may be observed (Fig.1) that in the Gauss method, during processing of the kth row, all column elements in the lower (left) semi-band up to the column no. $k-1$ have been already zeroed. Thus pivoting down of the kth row causes no expansion of left (lower) semi-band. Only the right (upper) semi-band may expand up to $2n_{band}$ value. The main part of memory consumption for the in-core Gauss solver is approximately

$$N_{GI} = 3n_{band}n_{eq} \tag{2.4}$$

It follows from comparison with Eq (2.1) that memory requirements of the Gauss solver are lower by 25% than that of the Crout solver. The advantages

of Gauss method are still more prominent in the case of out-of-core solver. It is easy to observe (Fig.1) that for n_{band} rows processed for a single disc memory access, the largest array should be dimensioned approximately as

$$N_{GO} = 6n_{band}^2 \tag{2.5}$$

This value is almost 3 times lower than the corresponding value Eq (2.2) for the out-of-core Crout solver.

The comparison of memory consumption by the Gauss and Crout pivoted banded solvers leads to the conclusion about superiority of the Gauss method. This advantage alone may decide about preferring of the Gauss algorithm in this application area. Moreover, other properties of the Gauss method are favouring it in the case of pivoted banded solvers.

3. New and old improvements of basic numerical technique

The pivoting alone is rarely sufficient for achieving numerical stability of solution, especially for not so well-conditioned systems. The established and accepted technique improving the pivoting efficiency is scaling and equilibration of the equation system. It has been proved (cf Bjoerck and Dahlquist, 1974) that optimum scaling and equilibration should start with minimising of the maximum norm

$$\min[\kappa(\mathbf{D}_2\mathbf{A}\mathbf{D}_1)] \tag{3.1}$$

Optimum matrices $\mathbf{D}_1$ and $\mathbf{D}_2$ depend on an inverse matrix $\mathbf{A}^{-1}$, which is unknown before obtaining the system solution (cf Bjoerck and Dahlquist, 1974; Cormen et al., 1994). Moreover, the scaling changes the norm used in error criterion. Thus, this approach has no practical implications. In practice, quite frequently the so called "implicit scaling" is used. In this method both the current diagonal element and potential candidate for pivot are normalised according to the formulas (cf Engeln-Muellges and Uhlig, 1996)

$$a'_{kk} = \frac{|a_{kk}|}{\sum_{i=1}^{n} |a_{ki}|} \qquad a'_{jk} = \frac{|a_{jk}|}{\sum_{i=1}^{n} |a_{ji}|} \tag{3.2}$$

Normalisation means simply division by the sum of absolute values of the current row elements (cf Engeln-Muellges and Uhlig, 1996). Pivoting decision is based on the comparison of the normalised values.

Another technique improving stability of computations can be called multi-pivoting. The multi-pivoting technique allows one to pivot down a row several

times, i.e. already pivoted row may be pivoted down again. However the Gauss and Crout methods differ widely in the potential of multi-pivoting application. In the Gauss method there are no limitations imposed on multi-pivoting, i.e. the first row may be pivoted down even to the very bottom of the matrix without expansion of non-zero band beyond the semi-bandwidth value (Fig.1). In the Crout solver pivoting down is limited to the semi-bandwidth value n_{band}. It prevents excessive expansion of non-zero band beyond $4n_{band}+1$ value. Therefore the Crout solver is potentially less stable numerically than the Gauss solver. This thesis will be proved in the course of numerical experiments.

Less obvious technique enhancing numerical stability can be nicknamed "thresholding". This technique introduces the limitation of pivoting through the usage of threshold factor t_h in the course of pivoting. The decision about pivoting is normally made (forget for the moment the scaling) if potential pivot is larger than the actual candidate for pivot, i.e. when

$$a_{kk} < a_{jk} \qquad j > k \tag{3.3}$$

After introduction of the threshold factor t_h the pivoting is performed only when

$$t_h a_{kk} < a_{jk} \qquad j > k \tag{3.4}$$

where t_h is a coefficient larger than 1 and depending usually on the problem at hand. The rationale behind thresholding is twofold. First of all, it can be expected that a threshold factor larger than 1 will reduce the number of pivotal row interchanges. The computational workload is reduced in this way. Moreover, important is another factor. Thresholding helps oneto avoid less profitable (in the sense of computational stability) interchanges, which may mask more advantageous ones. Also this thesis will be proved through numerical experiments.

4. Basic testing of new solvers

A new linear equation solver has been developed in C, using the Gauss method and enhanced pivoting strategies. Both in-core and out-of-core versions have been developed. Also a new in-core version of the Crout solver has been developed. All this software has been compared with the well known procedure **dgbsv** from the CLAPACK linear algebra library of C programs. Classic perturbation analysis (cf Bjoerck and Dahlquist, 1974; Forsythe et al.,

1977) may offer only a general insight into numerical stability of linear equation solvers. Such problems as the order of elementary arithmetic operations, processor properties and specific ill-conditioning may be investigated only in a series of numerical experiments.

Two sample coefficient matrices have been tested for the numerical stability of solution. The first one was a quasi-random matrix with rather negligible traces of diagonal domination. The coefficients of this matrix have been generated using the formula

$$a_{ij} = \frac{1 + 2 \cdot rand}{n_{eq} - (i + j - 1)} \tag{4.1}$$

where *rand* is a random number (uniform distribution in the range 0,...,1) supplied by C library. Diagonal coefficients have been multiplied by 3.0, but it is hard to call the matrix diagonally dominant. Quite obvious precautions preventing division by 0 in Eq (4.1) have been taken. The values generated depend on the properties of library routine and on the processor. At first, it was expected that this matrix is quite well-conditioned, but the experiments have shown, that rather contrary thesis is true; it surpassed in ill conditioning the classic Hilbert matrix.

The second test matrix was a classic ill conditioned coefficient matrix of the Hilbert type. The coefficients are generated here with the aid of well known formula

$$a_{ij} = \frac{1}{i + j - 1} \tag{4.2}$$

The tests have been carried out on the Pentium II with 266 MHz clock, running Linux RedHat 5.0 operating system. The solvers have been compiled with the aid of GNU gcc compiler. Another testing platform was Sun's Ultra 10 workstation featuring 300 MHz clock, Solaris 2.6 operating system and standard Sun's C compiler. Throughout all tests the double precision has been applied. Testing and comparing numerical stability is not an easy and unambiguous task, as there are no generally accepted standards and criteria. It has been decided to use the criterion of residues distribution. There exist assorted examples of linear equation systems solutions (cf Bjoerck and Dahlquist, 1974; Demmel, 1997; Engeln-Muellges and Uhlig, 1996) leading to small residues but at the same time completely diverging from the exact and correct solutions. However, in real-world cases small residues are almost invariably proving the solutions correctness and precision. Moreover, there is evident that large residues, beyond any doubt, are indicating incorrectness of the solutions.

The test equation system has been composed of 4800 equations with the semi-bandwidth of 240 and 3 right hand sides. Basically, the in-core versions

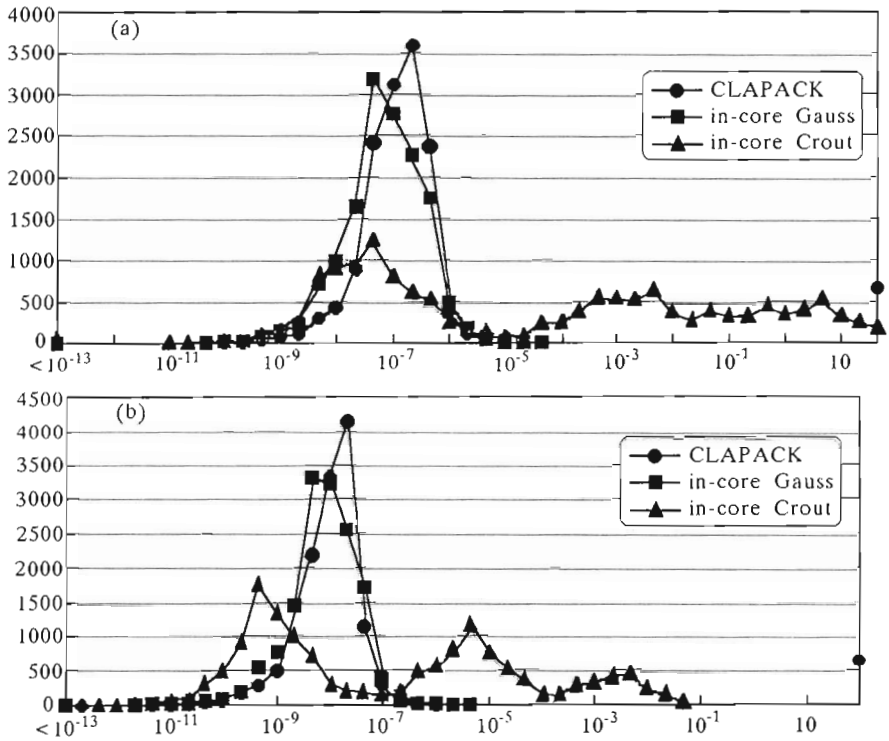

Fig. 2. Basic test results for the **dgbsv** (CLAPACK), new Gauss and new Crout solvers, respectively, for quasi-random coefficient matrices on Sun's Ultra 10 workstation (a) and Pentium II Linux box (b)

of new Gauss and Crout solvers have been used. The out-of-core versions are fully equivalent in the sense of numerical algorithm. In the first series of tests only multi-pivoting and implicit scaling have been switched on. The results of the tests for quasi-random system are presented in Fig.2. It should be observed that in the diagram showing distribution of relative residues there appears isolated island of extremely large relative residues (> 50) for the **dgbsv** solver from the CLAPACK library. The CLAPACK routines are simply useless, at least for this test matrix. Moreover, the maximum of residues distribution of new Gauss solver is shifted distinctly to the left (lower values) with respect to maximum for the **dgbsv**. The problem of numerical stability of **dgbsv** routine will be investigated latter in details. Modest Pentium II is not inferior to Sun's Ultra 10 in the sense of numerical stability but the computation time (Table 1) is approximately 3 times longer. The Crout solver performance is less satisfactory than that of new Gauss solver, especially on the Sun plat-

form. Larger equation systems (9600 and 14400 equations) have been tested on Ultra 10 (only 9600 equations) and E450 (the same UltraSparc processor with the slightly slower 247 MHz clock but with the larger RAM of 250 Mb per processor). Computation time depends roughly on the fourth power of system size (see E450 column). However, if the system size or rather dynamically adjusted storage requirements exceed the RAM size, the effect of deteriorating performance due to virtual memory switching on is very spectacular. Before more detailed presentation of thresholding let us apply this technique to the Crout solver. The effect of introducing the threshold factor $t_h = 10$ is very spectacular (Fig.3). Largest residues are now limited to $5 \cdot 10^{-3}$ as opposed to more than 50 in the previous version. Maximum of residues distribution is shifted to the region of larger values but it remains inside fully acceptable limits.

Table 1. Execution time (sec) for the quasi-random coefficient matrix for three hardware platforms

		Sun's Ultra 10	Sun's E450	Pentium II
4800 equations	**dgbsv** solver	57	61	168
semibandwidth=240	new Gauss solver	64	42	196
3 right hand sides	new Crout solver	137	149	267
9600 equations	**dgbsv** solver	1395	498	
semibandwidth=480	new Gauss solver	977	472	
3 right hand sides	new Crout solver	1792	1261	
14400 equations	**dgbsv** solver		1656	
semibandwidth=720	new Gauss solver		1668	
3 right hand sides	new Crout solver		4649	

5. Multi-pivoting and implicit scaling

In the next series of tests the influence of multi-pivoting and implicit scaling on solver performance has been investigated. The size of test system remained unchanged (4800 equations), both types of matrices have been involved (quasi-random and Hilbert) but the solver was only the new in-core Gauss.

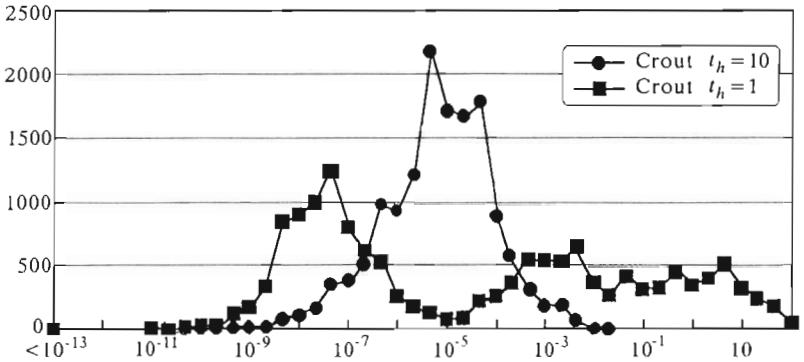

Fig. 3. Thresholding boosts Crout solver performance

Table 2. Execution time (sec) for the quasi-random and Hilbert coefficient matrices (4800 equations, semi-bandwidth= 240, 3 right hand sides), two hardware platforms and various combinations of numerical enhancements in new Gauss solver

	Sun's Ultra 10	Pentium II
quasi-random matrix		
simple solver	32	130
implicit scaling	53	168
multi-pivoting	33	132
scaling + multi-pivoting	64	196
dgbsv solver (CLAPACK)	57	168
Hilbert matrix		
simple solver	35	136
implicit scaling	51	174
multi-pivoting	36	138
scaling + multi-pivoting	67	191
dgbsv solver (CLAPACK)	65	166

For the Hilbert-type system the test results on both the hardware platforms are different (Fig.4). For Pentium II the residues distribution is most satisfactory for the solver with multi-pivoting switched on. Additional switching on of implicit scaling introduces no distinct numerical improvement – only computation time is longer (138 seconds vs. 191 seconds). A simple version of solver is numerically somewhat inferior. Still worse but wholly acceptable is the solver with implicit scaling. The last result of numerical experiments devia-

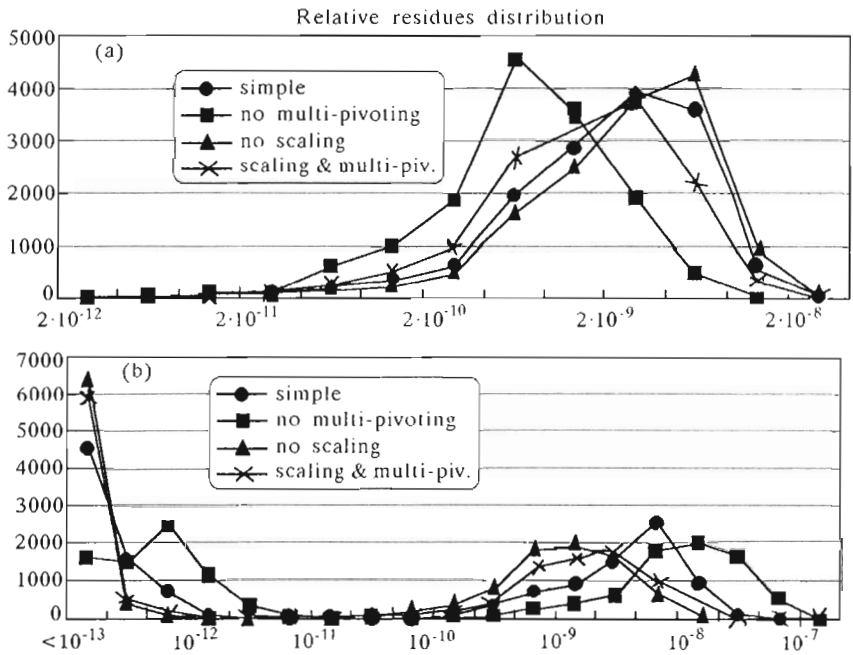

Fig. 4. Multi-pivoting and implicit scaling for the Hilbert coefficient matrix on Sun's Ultra 10 workstation (a) and Pentium II Linux box (b)

tes unexpectedly from common numerical wisdom (cf Bjoerck and Dahlquist, 1974; Demmel, 1997; Engeln-Muellges and Uhlig, 1996; George and Liu, 1981; Martin and Wilkinson, 1971). For Sun's workstation the results are different. The best numerical performance is obtained through switching on of implicit scaling only. The worst performer is the solver with multi-pivoting only. Simple solver and scaled plus multi-pivoted one perform in an intermediate way. Thus selection of these numerical enhancements depends clearly on the hardware platform.

The test results for quasi-random coefficient matrix are almost ideally identical for both the hardware platforms (Fig.5). Best numerical performance is obtained in the case of multi-pivoting switched on or for additional switching on of implicit scaling. As implicit scaling introduces no improvement but increases computation time the choice is rather clear cut. Implicitly scaled and simple solver (no enhancements) are inferior on both hardware platforms. Pentium II is here a better performer than Sun's workstation.

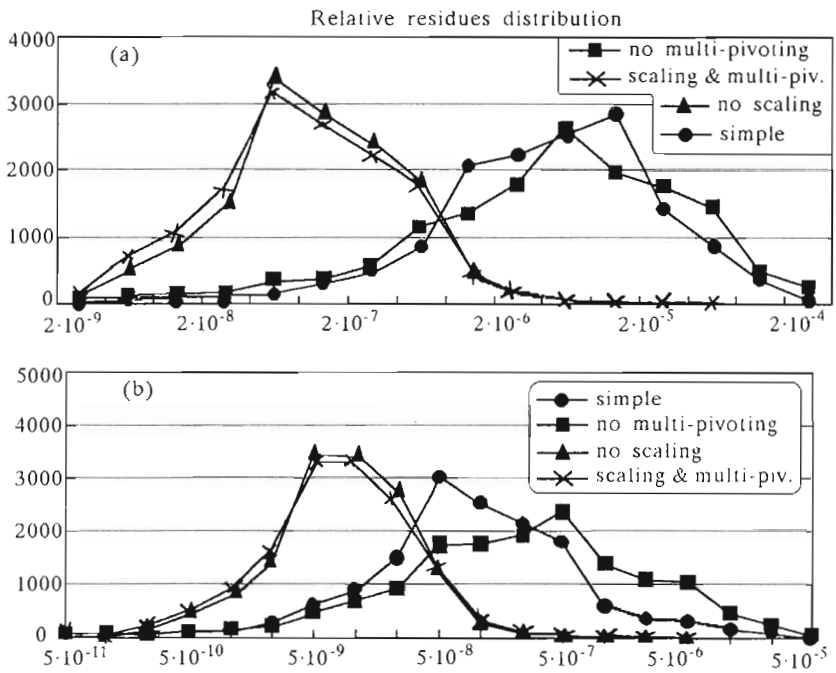

Fig. 5. Multi-pivoting and implicit scaling for quasi-random coefficient matrix on Sun's Ultra 10 workstation (a) and Pentium II Linux box (b)

Final conclusions of this series of tests can be formulated as follows:

- application of multi-pivoting and implicit scaling depends on the character of coefficient matrix;
- another factor in selecting of these numerical enhancements is the type of processor.

6. Thresholding technique

A very interesting method improving numerical stability relies on the thresholding introduced in Section 3. First of all one may expect that thresholding will eliminate pivoting of little value and weight for maintaining numerical stability. If the threshold factor is $t_h = 1$, several interchanges for small actual pivot ratio a_{jk}/a_{kk} (compare Eq (3.3) can mask and eliminate more profitable

pivoting with larger value of a_{jk}/a_{kk}. The second reason for application of thresholding is the expected reduction of the overhead for performing pivotal interchanges. For larger values of t_h (see Eq (3.3)) the interchanges are less frequent and execution time is lower.

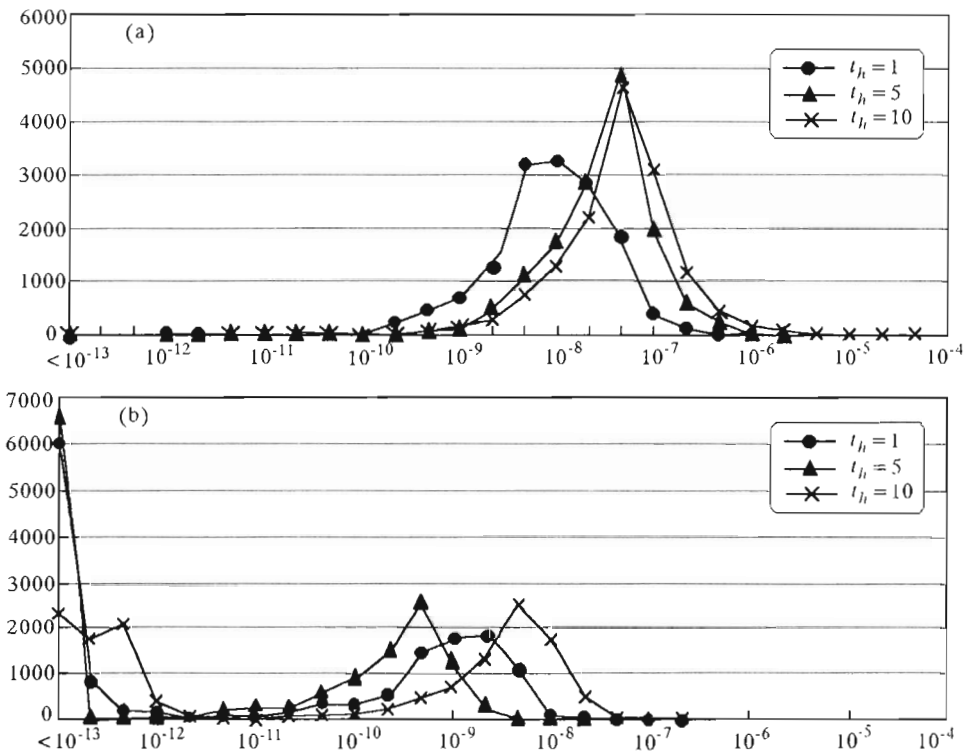

Fig. 6. Thresholding effects for the Hilbert (a) and quasi-random (b) coefficient matrices on Pentium II Linux box

These expectations have been already verified with a spectacular example of the Crout solver (Fig.3). More comprehensive results for both types of coefficient matrices (quasi-random and Hilbert) and Pentium II hardware are presented in Fig.6. Distribution of relative residues is shown for the threshold factor values t_h equal to 1, 5 and 10. For the Hilbert matrix (Fig.6a) introduction of $t_h = 5$ improves the computational accuracy. For $t_h = 10$ the opposite is true – maximum of residues distribution is shifted to the region of larger, less acceptable values. The picture for quasi-random matrix is completely different (Fig.6b). Best performance is obtained for straightforward solver with threshold factor $t_h = 1$. Increasing of threshold factor shifts the

maximum of distribution curve into less advantageous region of larger relative residues. In both cases of quasi-random and Hilbert matrices the changes resulting from introduction of thresholding are far from spectacular. Nevertheless the conclusion from the previous section can be reiterated – also the impact of thresholding depends on the character of coefficient matrix. Introduction of thresholding may shorten the computation time. It is the direct result of reduction of actual row interchanges for larger values of the threshold factor.

Summing up these observations the conclusion about thresholding application should depend on the nature of the problem in hand and on the processor type. Both new solvers provide the flexibility to use or switch off the usage of thresholding.

7. Comparative analysis of clapack library routine and new Gauss solver

Poor performance of **dgbsv** routine from the CLAPACK library was completely unexpected, as this library is considered at present as the state of the art linear algebra toolkit. Other results, not reported in Section 4, have been similarly unfavourable for the **dgbsv**. Thus, it has been decided to perform more thorough comparative tests of **dgbsv** and new Gauss solver. Both solvers use the same basic method (Gauss) but the new one performs implicit scaling before pivoting, is fully multi-pivoted and capable of thresholding.

The series of comparative tests has been performed on two hardware platforms – Sun's Ultra 10 and Pentium II – and the classic ill-conditioned Hilbert coefficient matrix has been used. The test system has been composed of 2400 equations in order to shorten the computation time. The value of the semibandwidth has been selected as the parameter of these tests; it ranged from 61 to 99. RMS value of relative residues has been used as the criterion of numerical stability.

The diagram presenting the stability of **dgbsv** and new Gauss solver for Ultra 10 workstation (Fig.7a) is really crushing for the **dgbsv**. The new Gauss solver is a clear winner with consistent stable value of error criterion. The criterion value is limited to the range of 10^{-15} up to 10^{-13} and is more than acceptable. The CLAPACK routine **dgbsv** exhibits the criterion value larger by approximately 10^{11}. It is far from acceptability, as there are also the regions of residues larger than 1. Extremely annoying are the dips in error criterion for the semibandwidth value of 74, 79 and 95.

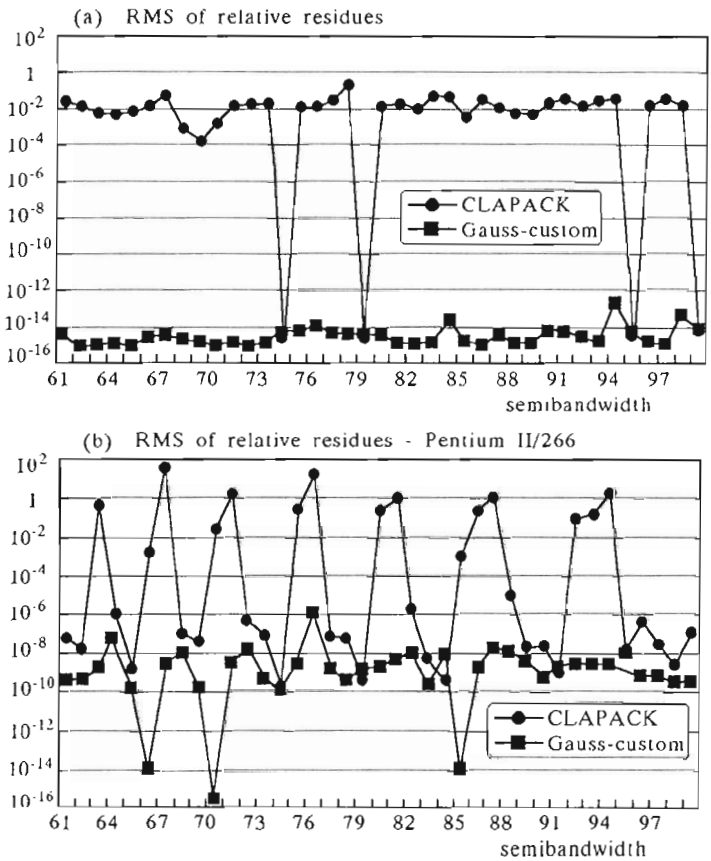

Fig. 7. Comparison between new Gauss and **sgbsv** (CLAPACK) routine for various values of the semi-bandwidth on Sun's Ultra 10 workstation (a) and Pentium II Linux box (b)

The corresponding diagram (Fig.7b) for Pentium II running Linux is rather different but fundamental differences between the **dgbsv** and new Gauss solver are also distinct and unfavourable for the **dgbsv**. It may be observed that basic error criterion for the new Gauss solver is in the range of 10^{-9} to 10^{-7} with three dips down to 10^{-14}. The error criterion for the **dgbsv** wildly oscillates between 10^{-9} up to absolutely unacceptable values of 1 or even 10^{2}.

Comparison of the tests performed on these two hardware platforms shows clear superiority of Sun's Ultra 10. It is no wonder, as this workstation boasts 64-bit RISC processor as opposed to modest 32-bit CISC in Pentium II. Moreover there is still a question of compiler robustness, especially in the field of

double precision. And last, but not least – is the floating point operation of Pentiums processor absolutely bug-free? However, these questions cannot be answered at this stage.

8. Conclusions

Two new solvers using the Gauss and Crout methods have been presented. Both have been developed in the in-core and out-of-core versions. Thorough tests have demonstrated satisfying performance, especially that of the Gauss solver, which due to inherent features of the method, imposes also lower memory requirements than the Crout solver. Both solvers outperform standard tools from the CLAPACK library, which are evidently unstable and unreliable.

There have been introduced two new techniques of thresholding and multi-pivoting, complementing quite standard implicit scaling. All these techniques can improve, sometimes very spectacularly, the numerical stability. However, they introduce some overhead, especially significant in the case of scaling. Moreover the improvements achieved with these techniques depend on the nature of the problem in hand and on some fine properties of the hardware. Both new solvers, in stark opposition to the CLAPACK library routines, provide the user with full control of these features.

Now, it is time to arrive at more practical and ordered conclusions. It seems that the following three key points may sum up the situation:

1. It is beyond any doubt that the **dgbsv** and **sgbsv** solvers from the CLAPACK library are absolutely unreliable in the sense of numerical stability.
2. The stability of the solver depends on the intricate properties of the processor and the problem at hand.
3. The solver should be designed with reasonable flexibility of usage, enabling switching on or off such features as scaling or pivoting.

The new Gauss solver meets the requirements formulated in point 3. Thus it surpasses existing software and adapts well to the problem and to the processor.

References

1. BJOERCK A., DAHLQUIST B., 1974, *Numerical Methods*, Prentice Hall, Englewood Cliffs

2. CORMEN T.H., LEISERSON C.E., RIVEST R.L., 1994, *Introduction to Algorithms*, The Massachusetts Institute of Technology, Massachusets
3. CROTTY J.M., 1982, A Block Equation Solver for Large Unsymmetric Matrices Arising in the Boundary Integral Equation Method, *IJNME*, **18**, 997-1017
4. DEMMEL J.W., 1997, *Applied Numerical Linear Algebra*, SIAM, Philadelphia
5. ENGELN-MUELLGES G., UHLIG F., 1996, *Numerical Algorithms with C*, Springer Verlag, Berlin
6. FORSYTHE G.E., MALCOLM M.M., MOLER C.B., 1977, *Computer Methods for Mathematical Computations*, Prentice Hall, Englewood Cliffs
7. GEORGE A., LIU W.H., 1981, *Computer Solution of Large Sparse Positive Definite Systems*, Prentice Hall, Englewood Cliffs
8. MARTIN R.S., WILKINSON J.H., 1971, Symmetric Decomposition of Positive Definite Band Matrices, In: J.H. Wilkinson, C. Reinsch (edit.), *Handbook for Automatic Computation*, **II**. Springer Verlag, Berlin
9. RIGBY R.H., ALIABADI M.H., 1995, Out-of-Core Solver for Large, Multi-Zone Boundary Element Matrices, *IJNME*, **38**, 1507-1533
10. STABROWSKI M.M., 1998, New Method of Pivoting in the Block Solvers for Large Banded Linear Equation Systems, *Journal of Theoretical and Applied Mechanics*, **36**, 1, 97-108

Nowy program rozwiązywania pasmowych układów równań liniowych z wyborem elementów głównych a procedury z biblioteki CLAPACK

Streszczenie

Przedstawiono nową metodę wyboru elementów głównych użyteczną w przypadku pasmowych niesymetrycznych układów równań liniowych. Metoda ta ogranicza wypełnianie, zachowuje podstawową strukturę pasmową i w pełni wykorzystuje zaalokowaną pamięć. Dwie nowe właściwości tej metody to wielokrotne wybieranie elementów głównych oraz stosowanie współczynników progowych przy podejmowaniu decyzji o wyborze elementów głównych, Stabilność, a właściwie dokładność rozwiązań zbadano w oparciu o przykład klasycznego źle uwarunkowanego układu równań (Hilbert) i pseudo-przypadkowego układu, jeszcze gorzej uwarunkowanego. Rezultaty ekstensywnych testów przedstawiono w dobrze czytelnej postaci graficznej. Potwierdzają one efektywność nowego oprogramowania, zwłaszcza w porównaniu z oprogramowaniem z biblioteki CLAPACK.

Manuscript received Februar 15, 1999; accepted for print April 30, 1999