11th World Congress on Computational Mechanics (WCCM XI)
5th European Conference on Computational Mechanics (ECCM V)
6th European Conference on Computational Fluid Dynamics (ECFD VI)
E. Oñate, J. Oliver and A. Huerta (Eds)

SEBSM-BASED RESIDUAL ITERATIVE METHOD FOR SOLVING LARGE SYSTEMS OF LINEAR EQUATIONS AND ITS APPLICATIONS IN COMPUTATIONAL MECHANICS

X. W. Gao[1], J. X. Hu[1], J. Liu[1] and M. Cui[1]*

[1] State Key Laboratory of Structural Analysis for Industrial Equipment
Dalian University of Technology, Dalian 116024, P.R. China
e-mail: xwgao@dlut.edu.cn; miaocui@dlut.edu.cn

Key Words: *System of linear equations, Simultaneous Elimination and Back-Substitution Method (SEBSM), Residual iterative method, Gauss-Seidel iterative method.*

Abstract. In this paper, a new iterative method, named residual iterative method, for solving sparse non-symmetrical systems of linear equations is proposed based on the Simultaneous Elimination and Back-Substitution Method (SEBSM), and the method is applied to solve systems resulted in solid mechanics problems solved using Finite Element Method (FEM). First, SEBSM is introduced for solving general linear systems using the direct method. And, then a double-iterations method for reducing the residual of the system of equations and modifying the value of the solution change is presented based on a Newton-Raphson iterative technique. In each system residual iteration, another inner iterative process is applied for modifying the solution change. Two numerical examples are given to demonstrate the behaviour of the proposed method.

1 INTRODUCTION

In science computation and engineering problems, people often need to solve linear systems of equations. In fact, in the most popular numerical methods, such as the finite element method (FEM) [1], boundary element method (BEM) [2], and the meshless method [3], the eventual task is to solve a system of equations. Choosing an efficient solver for a linear system is of significant importance both for improving computational efficiency and for saving storage requirements, especially for large-scale sparse equation systems.

The methods of solving linear systems can be classified into two categories [4,5]: direct methods and iterative methods. The frequently used direct methods are Gaussian elimination and LU-factorization methods, and the iterative methods [6,7] are represented by the Jacobi method, Gauss-Seidel method, SOR algorithm, Conjugate gradients, and GMRES. The advantage of the direct method is that the solution of the system can be definitely obtained by executing a finite number of operations, and the disadvantage is that as the size of the system become large, both the computational efficiency and the accuracy become deteriorated. Comparing to the direction methods, the iterative methods have the advantage that large systems can be solved since it is based on matrix-vector product operations, and the disadvantage that if the iteration cannot be converged, the solution usually cannot be accepted.

However, once a convergent solution can be obtained, iterative methods are usually less expensive than direct methods in solving large systems of equations. It can be seen that convergence is a critical issue in iterative methods.

In direct methods, a novel method for solving large sparse systems of linear equations was proposed recently by Gao and co-workers [8,9] based on a simultaneous elimination and back-substitution method (SEBSM). In the method, both elimination and back-substitution procedures are completed in a same row treatment, and, therefore, no final back-substitution procedure is required. SEBSM requires much less amount of storage and can also save the computational time by a few orders of magnitude comparing to the Gaussian elimination method. Nevertheless, when the system of equations is huge, the storage is still a big issue. In this case, the iterative method is a good alternative.

In iterative methods, the Jacobi and Gauss-Seidel methods [10] require much less storage than other methods. This is because the Jacobi method only employs the diagonal terms of the coefficient matrix *A* in the system *Ax=b* as the iterative matrix and the Gauss-Seidel method utilizes the lower triangular matrix of *A* as the iterative matrix. Generally, the Gauss-Seidel method is more stable than the Jacobi method. The main disadvantage of these two methods is that the convergence is not guaranteed for some systems of equations.

In view of the analysis of the advantages and disadvantages on the direct and iterative methods, this paper presents a new iterative method for solving all types of linear systems of equations: dense, sparse, unsymmetrical, and non-positive definite ones. In the method, the coefficient matrix *A* is split into lower and upper matrices. The lower matrix is served as the iterative matrix and the upper one is used to form the right-hand side of the solvable equation set. The feature of the proposed method is that the lower matrix has a certain bandwidth along the diagonal line. By making use of the band storing feature of SEBSM [8], the iterative convergence can be controlled by selecting a suitable bandwidth of the lower matrix. The proposed method overcomes the weaknesses of the Gauss-Seidel iterative method which can only use a lower triangular matrix of *A* as the iterative matrix, and breaks through the limitation of the Golub and Kahan's method which reduce *A* to a lower bi-diagonal form [11], and solves the bottle neck problems of the Thomas method which only can solve the tri-diagonal system of equations. Since the size of the working array needing to be stored in the iteration is as small as the upper bandwidth off the diagonal line of the lower matrix, large-scale linear systems can be solved using the proposed method.

2 SIMULTANEOUS ELIMINATION AND BACK-SUBSTITUTION METHOD (SEBSM)

The systems of linear equations are usually expressed in the component or matrix form as:

$$\sum_{j=1}^{n} a_{ij} x_j = b_i \qquad \text{or} \qquad A\boldsymbol{x} = \boldsymbol{b} \tag{1}$$

Using SEBSM to solve the above system is through a row by row operation procedure [8,9] (this is why the method was called REBSM in [8,9]). It is assumed that, after the operation of the first *k-1* rows, the first *k-1* unknowns can be expressed in terms of the remaining unknowns as follows:

Using SEBSM to solve the above system is through a row by row operation procedure [8,9] (this is why the method was called REBSM in [8,9]). It is assumed that, after the operation of the first *k-1* rows, the first *k-1* unknowns can be expressed in terms of the remaining unknowns as follows:

$$x_i = b_i^{(k-1)} - \sum_{j=k}^{n} a_{ij}^{(k-1)} x_j \qquad (i=1,2,\dots,k\text{-}1) \tag{2}$$

For the *k*-th row in equation (1), by eliminating the first *k-1* unknowns using (2), the *k*-th unknown x_k can be expressed in terms of the remaining unknowns as follows

$$x_k = b_k^{(k)} - \sum_{j=k+1}^{n} a_{kj}^{(k)} x_j \tag{3}$$

where

$$a_{kj}^{(k)} = \frac{a'_{kj}}{a'_{kk}}, \qquad b_k^{(k)} = \frac{b'_k}{a'_{kk}} \tag{4}$$

$$a'_{kj} = a_{kj} - \sum_{l=1}^{k-1} a_{kl} a_{lj}^{(k-1)}, \qquad b'_k = b_k - \sum_{l=1}^{k-1} a_{kl} b_l^{(k-1)} \tag{5}$$

And substituting (3) back to (2), the expressions for the previous unknowns can be updated as :

$$x_i = b_i^{(k)} - \sum_{j=k+1}^{n} a_{ij}^{(k)} x_j \quad (i=1,2,\dots,k\text{-}1) \tag{6}$$

where

$$a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{ik}^{(k-1)} a_{kj}^{(k)} \tag{7}$$

$$b_i^{(k)} = b_i^{(k-1)} - a_{ik}^{(k-1)} b_k^{(k)} \tag{8}$$

Equations (6) and (3) are the results after the treatment of the *k*-th row. Comparing them to equation (2), it can be found that the number of unknowns on the right-hand sides is reduced by one. Equations (6) and (3) are then repeated for all remaining rows of equation (1) one by one, until *k* reaches the final row, i.e., *k=n*. Whenever the row treatment goes forward by one, the unknowns on the right-hand side is reduced by one. When all rows are finished for treatment, the solutions of the system are obtained, which are $x_i = b_i^{(n)}$, *i=1, …, n*.

It is noted that during treatment of each row, the coefficients $a_{ij}^{(k)}$ need to be stored. However, the storage size for each row is changed and determined only by the non-zero elements after the diagonal line. Fig. 1 is a diagrammatic sketch of the storage spaces needed for a sparse matrix *A*, when *k* takes values from 1 to 10, in which the elements within the rectangles imply the storage spaces required for each *k* value.

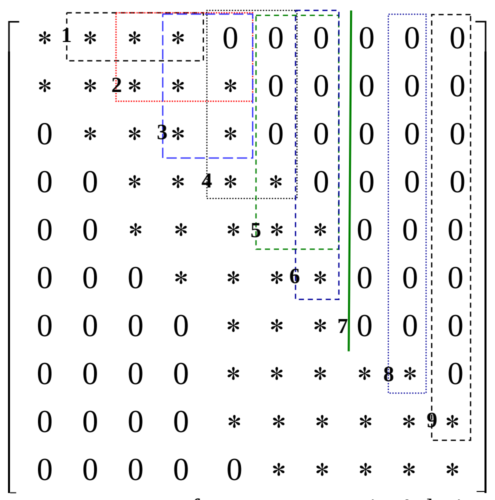

Figure 1: Coefficient storage spaces for a sparse matrix *A* during each row treatment

From Fig.1 it can be seen that, for a particular row, the required storage size for storing the coefficients involved in equation (7) depends on the position of the last non-zero element of the row under consideration. This gives us a chance to construct an iterative method to solve a huge system of equations.

3 DOUBLE-ITERATIONS METHOD BASED ON SEBSM

The first page must contain the Title, Author(s), Affiliation(s), Key words and the Summary. The Introduction must begin immediately below, following the format of this template.

Although SEBSM requires less storage space than other solvers, data storing is still a big issue in solving huge system of equations. Besides, when a system closes to singular, the direct method described above may result in a large computational round-off error. In these cases, the iterative method is a good alternative [12, 13].

According to the storing feature of SEBSM that only the spaces from the diagonal element to the last non-zero element of the row are needed, a double-iterations method for system residuals and solution changes is developed in the following by making use of this feature.

After *n*-th residual iteration, the solution of equation (1) is denoted by x^n, and the residual of equation (1) is written as follows:

$$R^n = b - Ax^n \tag{9}$$

At the *(n+1)-th* iteration, by modifying the solution, we enforce the residual to be zero, i.e. $R^{n+1} = 0$. Thus, utilizing Tayloy's series expansion, it can be obtained that:

$$R^{n+1} = R^n + \frac{\partial R^n}{\partial \mathbf{x}} \Delta x + O(\Delta x^2) = 0 \tag{10}$$

where Δx is the modification value of the solution, the solution change.

Noticing that $\partial R^n / \partial x = A$, ignoring the high order terms in above equation, the following equation can be obtained for determining the solution change:

$$A\Delta x = R^n \tag{11}$$

Solving the above equation, we can obtain the modification value of the solution, Δx. For a huge system of equations, it may be difficult to store the whole coefficient matrix *A* into the computer memory. To overcome this difficulty, we solve equation (11) through an iteration process by splitting *A* into a lower matrix A^L and an upper matrix A^U as follows

$$A = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} = A^L + A^U \tag{12}$$

where the lower matrix A^L has a certain bandwidth off the diagonal line, and A^U is the remaining part of *A*. As an example, following equations illustrate the coefficient splitting pattern for the case in which the lower matrix has the half bandwidth of $N_{1/2}^b = 3$.

$$A^L = \begin{pmatrix} \overbrace{a_{11} \quad a_{12} \quad a_{13}}^{N_{1/2}^b = 3} & 0 & \cdots & \cdots & 0 \\ a_{21} \quad a_{22} \quad a_{23} & a_{24} & 0 & \cdots & 0 \\ a_{31} \quad a_{32} \quad a_{33} & a_{34} & a_{35} & 0 \cdots & 0 \\ \vdots & & \vdots & & \vdots \\ a_{n1} \quad a_{n2} \quad a_{n3} & \cdots & \cdots & & \cdots a_{nn} \end{pmatrix} \tag{13}$$

$$A^U = \begin{pmatrix} 0 & 0 & 0 & a_{14} & \cdots & \cdots & \cdots & a_{1n} \\ 0 & 0 & 0 & 0 & a_{25} & \cdots & \cdots & a_{2n} \\ 0 & 0 & 0 & 0 & 0 & a_{36} & \cdots\cdots & a_{2n} \\ & \vdots & & & \vdots & & \vdots & \\ 0 & 0 & 0 & 0 & 0 & 0 & \cdots \quad \cdots & 0 \end{pmatrix} \tag{14}$$

After the coefficient matrix is split into two parts as shown in (12), the iterative formulation for solving the solution change can be written as

$$A^L \Delta x^{(I+1)} = R^n - A^U \Delta x^{(I)} \tag{15}$$

where the superscript *I* denotes the *I*-th iteration for Δx.

For a certain residual R^n obtained from equation (9) in the *n*-th residual iteration, equation (15) is used repeatedly, until the values of Δx does not change any more.

It is noted that using the SEBSM formulation to solve equation (15), the working space required for storing the coefficients of remaining unknowns is as large as the marked region in equation (13) rather than all non-zero elements in the lower matrix A^L. The maximum required working space is $N_{1/2}^{b} \cdot n$. This indicates that the current iterative method needs a relatively small storage.

To obtain a fast iterative scheme for the solution change, one can invert the lower matrix A^L in (15) and obtain a high efficient iterative scheme. However, the inverse matrix of A^L may be a dense matrix. In this case, if the order of *A* is huge, the required storage space may be unallowable for a computer resource. Therefore, whether directly using equation (15) or using the inverse matrix of A^L to perform the iteration process for the modification value Δx depends on how large the coefficient matrix is and the computer resource.

Once the iteration for the solution change is converged using equation (15), the approximation values of unknowns are updated using the following expression:

$$x^{n+1} = x^n + \lambda \Delta x \tag{16}$$

where λ is a relaxation factor which makes the norm of the residual of equation (1) minimum.

To determine λ, substituting equation (16) into equation (1) and the updated residual of the system can be expressed as

$$R^{n+1} = b - Ax^{n+1} = b - Ax^n - \lambda A \Delta x = R^n - \lambda A \Delta x \tag{17}$$

The norm of R^{n+1} can be written as

$$S = (R^{n+1})^T R^{n+1} = (R^n - \lambda A \Delta x)^T (R^n - \lambda A \Delta x) \tag{18}$$

To make *S* minimum, let

$$\frac{\partial S}{\partial \lambda} = 0 \tag{19}$$

Thus, from above equation, it can be easily solved that

$$\lambda = \frac{(R^n)^T (A\Delta x)}{(A\Delta x)^T (A\Delta x)} \tag{20}$$

After obtain the value of λ using equation (20), the updated solution can be obtained using equation (16). Then, substituting x^{n+1} into equation (9) results in the new residual of the system of equations. If the new residual is not small enough, a new residual iteration process is performed by using equations (9),(15),(20) and (16) again, until the norm of the residual is reduced to a specified value. In fact, when λ is less than a very small value or becomes minus, the iteration is stopped.

It can be seen that the iterative method described above is a double iterations scheme for the residual of the system and for the change of the solution, respectively. In the residual iteration, it has the advantage of the fast convergence in the Newton-Raphson method, and in the solution change iteration it has the advantage of requiring small storage as occurred in the Gauss-Seidel method [10]. If only one iteration is forced in the residual iteration, i.e. always keeping $R^n = b$, then the current method becomes the general matrix iterative method. On the other hand, if only one iteration is carried out for the solution change, that is, the right-hand side of equation (15) is always taken as R^n, then the current method reduced to a pure Newton-Raphson iterative method.

On inspection of equation (15), it can be seen that the iterative formulation in the solution change is similar to the Jacobi and Gauss-Seidel iterative algorithms [10]. The only difference is that in Jacobi method, the lower matrix A^L is the diagonal terms of *A*, and in the Gauss-Seidel method, A^L is the lower triangular part of *A*. However, in the current method, the lower matrix A^L can be formed by selecting a suitable half bandwidth size of $N_{1/2}^b$. This can guarantee the iteration converged. It is noted that the bigger $N_{1/2}^b$ is, the faster the convergence is, of course, the larger the storage space is required for containing the working coefficients of remaining unknowns.

In practical implementation, the solution change iteration does not need to be completely converged. A suitable maximum number of iterations may be given for the solution change iteration and the final global convergence of the problem can be controlled by the residual iteration of the system of equations. Using this strategy, the global computational efficiency can be improved considerably.

4 EXAMPLES

To examine the performance of the proposed method, SEBSM, two examples are analyzed in this section.

Example 4.1. The systems of equations are generated in such a way that

$$a_{ij} = 1/(|j-i|+1)$$

b_i is formed by setting $x_i = 1$ (i=1, …, n).

To make the coefficient matrix band sparse, when generated a_{ij} is smaller than 0.001, it is set to zero. For the purpose of comparison, Gauss-Seidel iterative method is also adopted here to solve the generated equation sets. Table 1 lists the computational time and numbers of iterations for the equation order n=10000, in which Sebsm-200 means the results using SEBSM with the half bandwidth of $N_{1/2}^b = 200$. Figs. 2 and 3 are the plots of the computational time and the number of iterations for different value of $N_{1/2}^b$, respectively. In iteration, the convergence tolerance is taken as 10^{-6}.

Table 1: Computational time using different bandwidth size

Iterative Method	Computational Time (s)	Number of Residual Iteration	Number of solution change iteration
Gauss-Seidel	91	x	57
Sebsm-20	77	7	1
Sebsm-20	119	6	2
Sebsm-20	129	5	3
Sebsm-20	139	4	4
Sebsm-20	138	3	6
Sebsm-20	138	3	7
Sebsm-800	211	3	1
Sebsm-1000	86	1	1

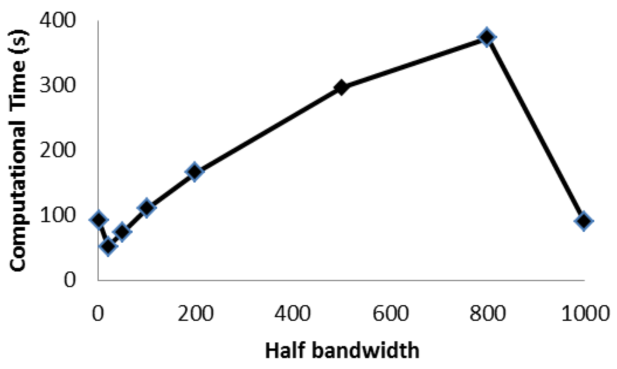

Figure 2: CPU time for different size of half bandwidth

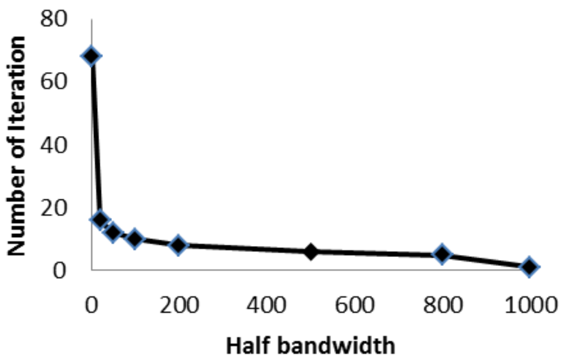

Figure 3: Number of iterations for different size of half bandwidth

From Table 1, it can be seen that the computational time varies with different size of the half bandwidth. The best performance occurs in the case of the number iteration for solution change being 1.

Example 4.2

In order to investigate the effectiveness of the presented method to solve the actual engineering problems, the second example is considered for steady heat transfer problem of a cube with the dimension of $1\text{m}\times1\text{m}\times1\text{m}$. The thermal conductivity is $k=50\ \text{W/(m}\cdot{}^\circ\text{C)}$. The temperature on the top of the cube is $T=500\ {}^\circ\text{C}$, while the temperature on the bottom surface is $T=50\ {}^\circ\text{C}$, other surfaces are subject to the insulated Neumann condition $q=0\ \text{W/m}^2$. This problem is analyzed using the finite element method (FEM), in which the system of equations is solved using the proposed method in this paper. Two FEM models with different elements are used as shown in Figure 4. Table 2 shows the detailed information of the two sets of meshes. The computed temperature distribution on the cube is shown in Figure 4 (c). Tables 3 and 4 list the computational time and numbers of iterations for the equation order is 6137 and 16224, respectively.

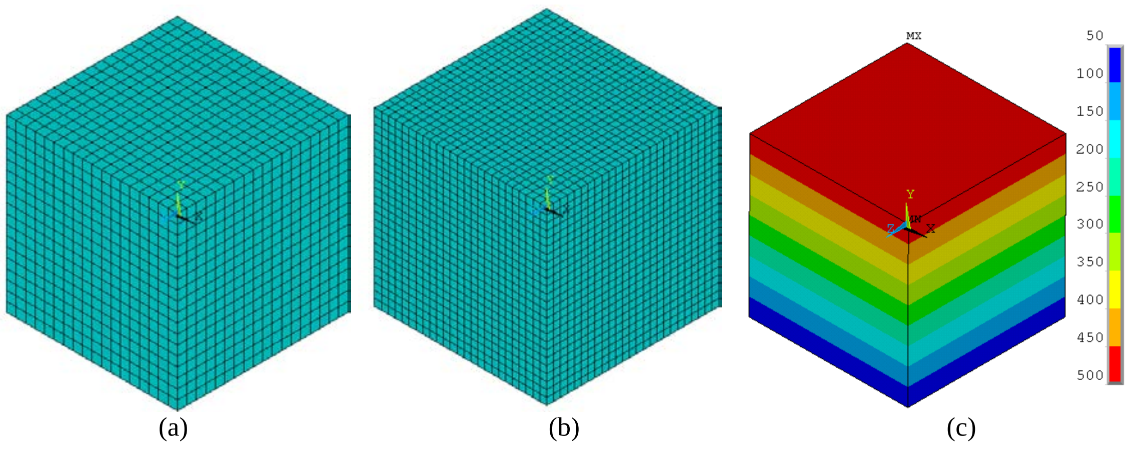

Figure 4: Two kinds of meshes and temperature distribution of the cube:
(a) mesh of 18*18*18, (b) mesh of 25*25*25, and (c) contour plot

Table 2: The detailed information of the two sets of meshes

Meshes	(a)	(b)
The order of the equations	6137	16224
The number of non-zeros	59777	163524
Sparse radio	0.0016	0.00062125

Table 3: Computational time using different bandwidth size and number of iterations

Iterative Method	Computational Time	Number of Residual Iteration	Number of solution change iteration
Gauss-Seidel	21m12s	x	411
Sebsm-1	32s	237	1
Sebsm-1	30s	113	3
Sebsm-5	1m22s	240	1
Sebsm-5	1m18s	121	3
Sebsm-10	1m44s	241	1
Sebsm-10	1m37s	112	3
Sebsm-20	1m56s	169	1
Sebsm-20	2m15s	105	3
Sebsm-50	3m3s	137	1
Sebsm-50	4m6s	113	2
Sebsm-50	3m44s	93	3
Sebsm-50	3m43s	85	4
Sebsm-50	3m54s	85	5
Sebsm-50	3m51s	75	6
Sebsm-50	3m48s	73	7
Sebsm-100	5m48s	137	1
Sebsm-100	7m12s	85	4

Sebsm-200	12m26s	137	1
Sebsm-200	15m2s	93	3
Sebsm-6137	9s	1	1

Iterative Method	Computational Time	Number of Residual Iteration	Number of solution change Iteration
Gauss-Seidel	24h34m35s	x	2300
Sebsm-1	5m58s	393	1
Sebsm-1	7m30s	308	2
Sebsm-1	8m10s	284	3
Sebsm-1	8m0s	253	4
Sebsm-5	19m26s	478	1
Sebsm-5	17m12s	218	3
Sebsm-10	23m22s	427	1
Sebsm-10	22m13s	218	3
Sebsm-20	42m58s	361	1
Sebsm-20	46m55s	188	5
Sebsm-20	1h5m13s	150	20
Sebsm-50	59m37s	246	1
Sebsm-50	1h19m5s	119	5
Sebsm-100	3h5m0s	232	1
Sebsm-100	5h46m21s	152	5
Sebsm-200	9h12m43s	231	1
Sebsm-200	10h22m35s	124	3
Sebsm-300	7h41m57s	231	1
Sebsm-300	13h48m14s	138	5
Sebsm-16224	16m4s	1	1

5 CONCLUDING DISCUSSIONS

A novel iterative method for solving large system of linear equations is presented based on the simultaneous elimination and back-substitution method (SEBSM). In the method, double iterative procedures are performed: one being for the residual of the system and the other being for modification value of the solution. The feature of the presented method is that the required storage space and the iterative convergent speed can be controlled by selecting a suitable half bandwidth of the lower matrix. A bigger size of the half bandwidth can accelerate the convergence speed, but the total computational time becomes longer. Therefore, it may be concluded that, under ensuring convergence, the less half bandwidth size can achieve a better computational efficiency.

ACKNOWLEDGMENTS

The authors gratefully acknowledge the National Natural Science Foundation of China under Grant NSFC Nos. 11172055 and 51206014 for financial supports to this work.

REFERENCES

[1] Zienkiewicz, O.C. and Taylor, R.L. *The finite element method*. McGraw Hill, Vol. I., (1989), Vol. II, (1991).

[2] Xiaowei,G. and Davies, T.G. *Boundary element programming in mechanics*, Cambridge University Press, Cambridge, (2002).

[3] Liu, G.R., Khin Z., Wang, Y.Y. and Deng, B. A novel reduced-basis method with upper and lower bounds for real-time computation of linear elasticity problems, *Computer Methods in Applied Mechanics and Engineering* (2008) **198**: 269-279.

[4] Press, W. H., Flannery, B. P., Teukolsky, S.A. and Vetterling, W.T. Gauss–Jordan elimination and gaussian elimination with backsubstitution, *Numerical Recipes in FORTRAN: The Art of Scientific Computing*, Second edition, Cambridge University Press, Cambridge, pp.27-32 and 33-34, (1992).

[5] Strang, G. *Introduction to Linear Algebra*, Third edition, Wellesley-Cambridge Press, Wellesley, MA. pp.74-76, (2003).

[6] Golub,G. H. and Van Loan, C.F. *Matrix Computations*, Third edition, Johns Hopkins Universiy Press, Baltimore, (1996).

[7] Saad,Y. *Iterative Methods for Sparse Linear Systems*, Second edition SIAM, Philadelphia, (2003).

[8] Xiaowei,G. and Lingjie L. A solver of linear systems of equations (REBSM) for large-scale engineering problems. *International Journal of Computational Methods* (2012) **9**(01):1240011(12 pages).

[9] Xiaowei,G., Jinxiu,H. and Miao C. A mdbem based on row elimination-back-substitution method, *Chinese Journal of Theoretical and Applied Mechanics* (2012) **44** (2):361-368.

[10] Li, W. And Sun, W.W. Modified Gauss–Seidel type methods and Jacobi type methods for Z-matrices, *Linear Algebra and its Applications* (2000) **317**:227-240.

[11] Bjorck, A. A bidiagonalization algorithm for solving large and sparse ill-posed systems of linear equations. *BIT Numerical Mathematics* (1988) 28(3): 659-670.

[12] Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M., Dongarra, J., Eijkhout,V., Pozo, R., Romine, C., and Van der Vorst, H. *Templates for the solution of linear systems:* building blocks for iterative methods (No. 43). Society for Industrial and Applied Mathematics, (1987).

[13] Press,W..H., Teukolsky,S.A., Veterling,W.T., Flannery,B.P. *Numerical Recipes in Fortran 77*. Second edition , Cambridge: Cambridge university press, 1992