# DATA STRUCTURE AND STATIONARY ITERATIVE SOLUTION METHODS FOR LARGE SPARSE LINEAR SYSTEMS

**João Batista Campos Silva**
UNESP/Ilha Solteira, Faculdade de Engenharia, Departamento de Engenharia Mecânica - 15385-000 – Ilha Solteira – SP - Brazil
jbcampos@dem.feis.unesp.br

**João Batista Aparecido**
UNESP/Ilha Solteira, Faculdade de Engenharia, Departamento de Engenharia Mecânica - 15385-000 – Ilha Solteira – SP - Brazil
jbaparec@dem.feis.unesp.br

**Abstract.** *In this work, algorithms have been implemented to solve large sparse linear systems like those resulting from the solution of convective-diffusive problems by numerical methods such as finite difference, finite volume and finite element methods, exploring and preserving the initial sparsity of global matrix. Only the non-zero coefficients of the systems are stored in vector mode. In this first phase of this work, the methods implemented are those of iterative stationary type. A library of computational subroutines has been developed for the purpose of this work. Some numerical tests are implemented and results are obtained to verify the performance of the methods and their computational implementation. No comparison with other public library of solution subroutines has been done yet.*

***Keywords**. Data Structure, Iterative Solution, Sparse Systems, Convective-Diffusive Problem, Gauss-Seidel, SOR.*

## 1. Introduction

Many practical problems in engineering and related fields are governed by partial differential equations of second order. Among these problems are the convective-diffusive problems, generally, solved by numerical methods, such as: finite volume, finite element or finite difference methods. The resultant algebraic systems of equations from application of these numerical methods forms large and sparse matrices. It taking account of the sparsity of the matrices may save storage requirements and reduce computing time for solution of the systems. In this way, iterative methods are advantageous due to their simplicity and power. The simplest iterative method has the form $x^{l+1} = x^l - \tau(Ax^l - b)$, $l = 0, 1, \ldots$, where $x^0$ is the initial guess and $\tau$ is a parameter, (Axelsson, 1996). Most iterative methods require only matrix-vector multiplication, vectors dot product and vectors addition. Furthermore, the sparsity of the matrix *A* can be fully exploited, and computer storage is required only for the nonzero entries of *A* and the vector *x*, plus one to three more vectors to identify the position and the number of nonzero coefficient. Under certain conditions, the sequence $\{x^l\}$ converges to the solution of $Ax = b$. Excellent references on iterative methods are (Golub and van Loan, 1996) and Hackabush (1994).

The iterative methods can be classified as of first-order or second-order, stationary or non-stationary. Some known iterative methods are the Jacobi, Gauss-Seidel, the SOR, Chebyshev, Conjugate Gradient, Lanczos-Type and Generalized Conjugate Gradient. In item 2 is given more insight about some characteristics of iterative methods.

The sparse matrix computation has been subject of study by many authors, for example, see the work of (Kotlyar, 1999). Many computational packages and libraries have been developed for this purpose. Duff (1999) presented a list of software for the parallel solution of large sparse linear systems. Saad & van der Vorst (1999) presented a list of the main research developments in the area of iterative methods for solving linear systems during the 20th century.

The main purpose in this work is to implement some iterative methods and procedures for solutions of large sparse systems originated from the numerical solutions of convective-diffusive problems, in such way, that only nonzero entries of the matrix *A* will be stored, reducing data storage and computing time. Some test problems are solved in order to show the performance of the methods.

## 2. Iterative Solution Methods

In this section we present, in a brief way, the main characteristics of some iterative methods. At this stage of the work the iterative methods that have already been implemented are the Jacobi, the Gauss-Seidel and the SOR methods.

## 2.1. Basic iterative methods

A basic iterative method has the form:

$$Cd^{l+1} = -r^{l}, \quad x^{l+1} = x^{l} + d^{l+1}, \quad l = 0,1,2,... \tag{1}$$

where $r^l$ is the residual defined as $r^l = Ax^l - b$ and $d^{l+1}$ is a correction term at each step. The initial approximation, $x^0$, is sometimes arbitrarily chosen and sometimes taken to be $x^0 = C^{-1}b$. The matrix $C$ is nonsingular, generally, called a preconditioning matrix, that can be chosen in various ways for a faster rate of convergence (Axelsson , 1996).

If a matrix $R$, called a defect matrix, is defined as $A = C - R$, called a splitting of $A$, the Eq. (1) has the alternative form

$$Cx^{l+1} = Rx^{l} + \mathrm{b}, \quad l = 0,1,2,.... \tag{2}$$

## 2.2. Stationary iterative methods

The iterative method given by Eq. (2) can be generalized and improved by introducing parameters. The new methods will be of first-order, or one-step iterative, when it is defined by

$$Cd^{l+1} = -\tau_l r^{l}, \quad x^{l+1} = x^{l} + d^{l+1}, \quad l = 0,1,2,... \tag{3}$$

where $\{\tau_l\}$ is a sequence of parameters. If $\tau_l = \tau, \quad l = 0,1,2,...,$ is constant; the method is called stationary; otherwise it will called non-stationary.

A second-order, or two-step iterative, method is defined by

$$Cd^{l} = r^{l}, \quad x^{l+1} = \alpha_l x^{l} + (1-\alpha_l)x^{l-1} - \beta_l d^{l}, \quad l = 0,1,2,..., \tag{4}$$

where $\{\alpha_l\}$, $\{\beta_l\}$ are sequences of parameters with $\alpha_0 = 1$.

## 2.3. The Jacobi, Gauss-Seidel and SOR methods

In some cases, a matrix A can be partitioned into the following block matrix form

$$A = \begin{bmatrix} D_1 & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & D_2 & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n,1} & A_{n,2} & \cdots & D_n \end{bmatrix} \tag{5}$$

where $D_i$ are square nonsingular matrices. Detailed considerations about the characteristics those matrices can be found in Chapter 6 of Axelsson (1996).

When $A = D_A - B_A$, with $D_A = diag(D_1,\ D_2,\ ...,\ D_n)$, the following iterative method, called the (block) Jacobi method or simultaneous iteration method, is convergent:

$$D_A x^{l+1} = B_A x^{l} + \mathrm{b}, \quad l = 0,1,2,..., \tag{6}$$

an alternative form for Jacobi method is:

$$x_i^{l+1} = \frac{1}{a_{i,i}} \sum_{\substack{j=1 \\ j \neq i}}^{n} (b_i - a_{i,j} x_j^{l}), \quad i = 1,\ 2,...,n, \quad l = 0,\ 1,... \tag{7}$$

In he above equations $B = I - D_A^{-1}A = D_A^{-1}B_A$ is called the Jacobi matrix.

Now, consider the splitting of $A$ as: $A = D - L - U$, where $D = D_A$ and $L,\ U$ are the lower and upper block triangular parts of $A$, respectively. The following iterative method, called the (block) Gauss-Seidel method or successive iteration method, is convergent

$$(D-L)x^{l+1} = Ux^{l} + \mathrm{b}, \quad l = 0,1,2,... \qquad . \tag{8}$$

In a more suitable form to computational implementation, Eq. (8) can be rewritten as

$$x_i^{l+1} = \frac{1}{a_{i,i}}(b_i - \sum_{j=1}^{i-1} a_{i,j}x_j^{l+1} - \sum_{j=i+1}^{n} a_{i,j}x_j^{l}), \ i = 1,\ 2,...,n, \quad l = 0,\ 1,... \tag{9}$$

The Equation (8) can be generalized as

$$(D-\omega L)x^{l+1} = [(1-\omega)D + \omega U]x^{l} + \omega b, \quad l = 0,1,2,..., \tag{10}$$

or

$$(\frac{D}{\omega} - L)x^{l+1} = [(\frac{1}{\omega} - 1)D + U]x^{l} + b, \quad l = 0,1,2,..., \tag{11}$$

where $\omega \in (0,2)$ is called the relaxation parameter. The method is known as the successive relaxation method. This method when $1 < \omega < 2$ is called over-relaxation, and the method of Eq. (10) is then called successive over-relaxation (SOR) method, (Axelsson, 1996). For $0 < \omega < 1$, the method may be called under-relaxation, but many authors call the method as SOR method for the whole open interval (0,2). For $\omega = 1$ the SOR reduces to the Gauss-Seidel method.

Equation (11) can written in the following alternative form

$$(\frac{D}{\omega} - L)(x^{l+1} - x^{l}) = -(Ax^{l} - b), \quad l = 0,1,2,... \tag{12}$$

The comparison of Eq. (12) with Eq. (2) shows that $C = D/\omega - L$ and $R = (1/\omega - 1)D + U$. The iteration matrix of Eq. (10) is defined as

$$L_{\omega} = (D-\omega L)^{-1}[(1-\omega)D + \omega U]. \tag{13}$$

### 2.4. The Conjugate Gradient Method

Just for the completeness of the work we also present the formulation of Conjugate Gradient Method. The conjugate gradient methods can be seen as iterative solution methods to solve linear systems of equations $Ax = b$ by minimizing a quadratic functional, such as $f(x) = (Ax-b)^T (Ax-b)$ over certain vector spaces called Krylov spaces, (Axelsson, 1996). A more standard form of conjugate gradient method, when *A* is positive definite, is a minimization of the functional

$$f(x) = \frac{1}{2} < r, A^{-1}r > \tag{14}$$

where $r = Ax - b$ and $<,>$ is an inner product.

There are other variants of the conjugate gradient method depending on the characteristics of matrices involved in the linear system, see Axelsson, (1996). This part of the research is currently being done.

## 3. Mathematical Model of Convective-Diffusive Problems

Generally, in a convective-diffusive problem the variable of interest can be defined by a partial differential equation, the transport equation, in the form:

$$\frac{\partial \phi}{\partial t} + \frac{\partial (u_j \phi)}{\partial x_j} = \frac{\partial (\Gamma_{\varphi} \partial \phi)}{\partial x_j \partial x_j} + S_{\phi} \tag{15}$$

where $\phi$ is a scalar variable, $u_j$ is the velocity component in direction $x_j$, $\Gamma_{\phi}$ is a diffusion coefficient and $S_{\phi}$ is a source/sink term.

In Table (1) is shown variables $\phi$, coefficient of diffusion $\Gamma_\phi$ and source term $S_\phi$, for continuity, momentum and energy equations that govern a convective-diffusive heat transfer problem.

The numerical methods, generally, used to solve Eq. (15) are finite volumes, finite differences and finite elements methods. Whichever, the method employed to the solution, an algebraic system of equations will be obtained after the time and spatial discretization of Eq. (15). That system when written in a matrix form will be

$$A\Phi = S \tag{16}$$

Table 1. Variables, coefficients and source/sink terms in Eq. (15) for a convective-diffusive heat transfer problem.

| Equation | $\phi$ | $\Gamma_\phi$ | $S_\phi$ |
|---|---|---|---|
| Continuity | $\rho$ | *0* | *0* |
| Momentum | $u_i$ - velocity in axis $x_i$ | *Kinematic viscosity* | *Pressure gradient in axis* $x_i$ |
| Energy | *T - temperature* | *Thermal diffusivity* | *Source/sink of energy* |

In Eq. (16), $A$ is a coefficient matrix that depends on $U_j$, $\Gamma_\phi$, the time step and size of elements in spatial discretization. Generally, the matrix $A$ has very high sparsity (defined as the ratio of quantity of zero coefficients by $n^2$; where $n$ is the dimension of $A$) of more than 99 percent for $n > 1000$. The vector of unknowns at spatial nodes is $\Phi$ and $S$ is a vector of independent terms ( known a *priori).*

**4. Compressed Row Storage Data Structure**

When the coefficient matrix $A$ is sparse, large-scale linear systems of the form $Ax = b$ can be most efficiently solved if the zero elements of $A$ are not stored, Barret et al. (1994). Sparse storage schemes allocate contiguous storage in computer random access memory for the nonzero elements of the matrix, and perhaps a small number of zeros. Of course, some scheme for knowing where the elements fit into the full matrix is necessary.

There are many of such methods for storing the data. Some of them are Compressed Row and Column Storage, Block Compressed Row Storage, Diagonal Storage, Jagged Diagonal Storage and Skyline Storage. In this work we have implemented the Compressed Row Storage (CRS).

The Compressed Row and Column Storage formats are the most general, because they make absolutely no assumptions about the sparsity structure of the matrix, and they don't store any unnecessary zero elements. Maybe their main drawback is the needing of an indirect addressing step for every single scalar operation in a matrix-vector product or pre-conditioner solve.

The Compressed Row Storage (CRS) format puts the subsequent nonzero elements of the matrix rows into contiguous memory locations. For example, for a non-symmetric sparse matrix *A*, we create these vectors: one for floating-point numbers, and the other two for integers. If we call *val* the vector for real values, this vector stores the nonzero elements of the matrix *A,* as they are traversed in a row-wise fashion. If we name *col_ind* and *row_ptr* the other two vectors for integers, in *col_ind* vector are stored the columns indexes of the elements in *val.* That is, if $val(k) = a_{i,j}$ then $col\_ind(k) = j$. The *row_ptr* vector stores the number of nonzero coefficients in each row of matrix *A*. The first element in *row_ptr* may be any integer number, in this case, it was defined as unity. For example, the location "*i"* in *row_ptr* stores the quantity of all nonzero coefficients that have already been counted in all previous rows, plus one of the first location. Since the matrix *A* has n rows, the dimension of *row_ptr* will be n+1. As a consequence of that definition *row_ptr(n+1) = nnz+1,* where *nnz* is the number of nonzero coefficients in the matrix *A*. The storage savings for this approach is very significant. Instead of storing $n^2$ elements, we need only to store *nnz* floating point elements and *nnz+n+1* integer elements.

We illustrate below as the CRS is implemented for a non-symmetric small matrix *A* defined by

$$A = \begin{bmatrix} 10 & 0 & 0 & -2 \\ 3 & 9 & 0 & 0 \\ 0 & 7 & 8 & 0 \\ 3 & 0 & 8 & 7 \end{bmatrix} \tag{17}$$

The CRS format for this matrix is then specified by the arrays {*val, col_ind, row_ptr*} given as follows, in Table 2:

Table 2. Vectors defined in a CRS data structure for the matrix *A*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| val | 10 | -2 | 3 | 9 | 7 | 8 | 3 | 8 | 7 |
| col_ind | 1 | 4 | 1 | 2 | 2 | 3 | 1 | 3 | 4 |
| row_ptr | 1 | 3 | 5 | 7 | 10 | | | | |

## 5. Test Problems and Results

As mentioned at Introduction, the main purpose of this work is to implement iterative solution algorithms for large sparse systems, with storage of only nonzero elements in matrix *A*. Until this stage of the work the iterative methods implemented have been the stationary ones: Jacobi, Gauss-Seidel and SOR. The technique of storage of nonzero elements of matrix *A* is the RCS (row compressed storage). The tests were performed considering matrices that, generally, appear in finite difference and finite element discretizations of convective-diffusive problems in two-dimensional domains. These matrices present a penta-diagonal structure. It is quite straightforward to solve also linear systems presenting hepta-diagonal-like matrices that appear frequently in three-dimensional problems. In all simulations the vector of starting was taken as: $x_0 = 0$ .

The convergence criteria is based on the following conditions: $\| r_k \| \le \varepsilon_r (\| A \| \| x^k \| + \| b \|)$ or $\| r_k \| \le \varepsilon_r \| b \|$ or $\| r_k \| \le \varepsilon_r$ . The norm $\| \ \|$ is defined in Appendix A, *r* is the residual and $\varepsilon_r$ is a given parameter, small enough to be represented by the tiny computer word. We can say that a sequence $\{x^k, k = 1 : \infty\}$ converges to *x* if the $\lim_{x \to \infty} \| x^k - x \| = 0$. The stopping criteria may be one of the following: the norm of residual satisfies the convergence criteria; or there is no evolution of the sequence $x^k$, expressed in mathematical form as $\| x^{k+1} - x^k \| < \varepsilon_x$ ; or the number of iterations exceed a maximum number specified. The parameter $\varepsilon_x$ has the nature of $\varepsilon_r$ and is used computationally to control the evolution of the series $\{x^k, k = 1 : \infty\}$ .

### 5.1. Test using the Gauss-Seidel method with CRS data structure

The Gauss-Seidel method has been implemented with compressed row storage data structure. For purpose of test, it was taken an arbitrary penta-diagonal matrix that has a structure similar to matrices originated in application of finite difference or finite element methods. The matrix *A* and vector *b* have the following forms:

$$a_{ij} = \begin{cases} -1, & if \quad i = j + 2n/3 \\ -1, & if \quad i = j + n/3 \\ +4, & if \quad i = j \\ -1, & if \quad j = i + n/3 \\ -1, & if \quad j = i + 2n/3 \end{cases} , \qquad and \qquad b_i = -1/i \tag{18a,b}$$

Figure (1) shows the number of iterations to reach the stopping criteria as a function of the matrix order and using variables of four bytes (single precision) and eight bytes (double precision) respectively. In computation of actual problems, we recommend the use of double precision variables, because the calculation using single precision variables generally presents low accuracy due to round-off errors. The upper curves correspond to variables of double precision and the lower curves correspond to single precision variables. The oscillation on the number of iterations with the increasing of matrix order may be due that the problems are actually different when the order of the systems are changed. In spite of the coefficient matrices be of equal structures, the problems are different for different orders, n, of the matrices. The Figure (1) also shows that the SOR method with optimal relaxation factor converges faster than the Gauss-Seidel method. All computations were performed using the parameters $\varepsilon_r$ and $\varepsilon_x$ settled to $\varepsilon_r$ = $\varepsilon_x$ = $10^{-7}$ when using variables of 4 bytes and $\varepsilon_r$ = $\varepsilon_x$ = $10^{-15}$ when using variables of 8 bytes.

Figures (2) and (3) show the variation of residual norm with the order of matrix for variables of 4 bytes and 8 bytes, respectively. The residual norm used to obtain the results presented is the $L_2$ norm as defined in the Appendix A. For variables of 8 bytes, the calculated residual norm in SOR method presented low values than the calculated residual norm in the Gauss-Seidel method to the most range of the matrix order.

In Figure (4) is presented the number of nonzero (*nnz*) elements with the increasing of matrices order or the required number of locations to storage the real values of vector *val*. In Figure (4) is also showed the number of locations required to store the supposed full matrix. We can see that the total capacity of storage required drops deeply using the compressed row data structure. We start the calculation with matrices of order n = 1000, and even, in this case the sparsity is greater than 99%. The sparsity was defined as *$(1-nnz/n^2)$*. To take in account the sparsity of matrices it was used the compressed row storage. The compressed row storage, as mentioned above, decreases drastically the memory used during the calculation, and also the computing time. An inspection showed that, while, the solution with CRS data structures used about 2 Mb for the solution, the calculation with full matrices used until 700 Mb, for a matrix of order n = 20000. Also for measure of the time of computation was solved a system of dimension such as 200000 by 200000. Using a personal computer, Pentium III with RAM of 768 Mb, it takes about 3 and half hours of elapsed time to solve a such system.

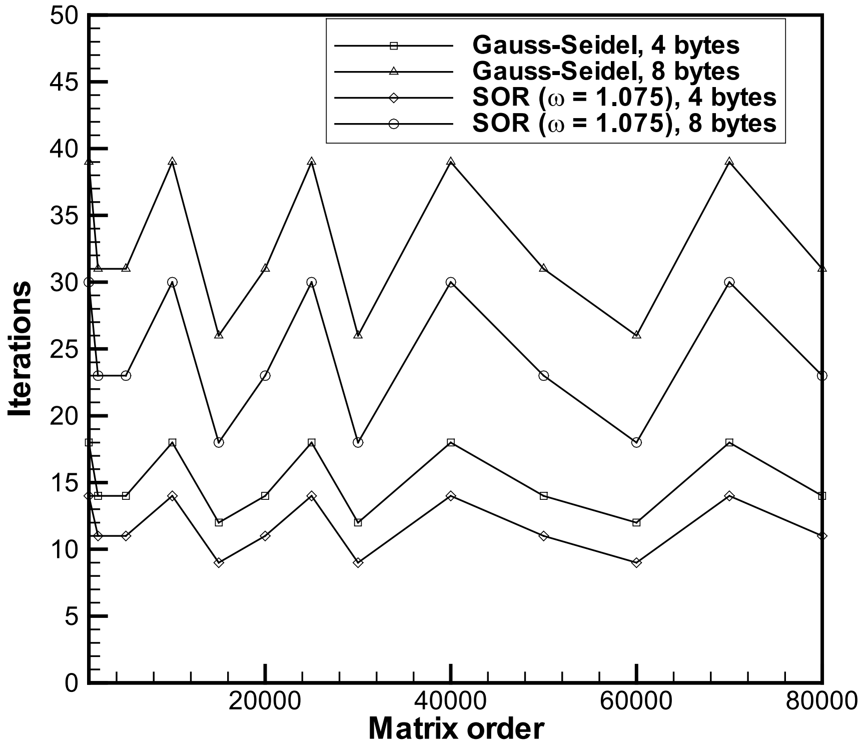

Figure 1. Number of iterations as functions of matrix order for different methods and computer word length.

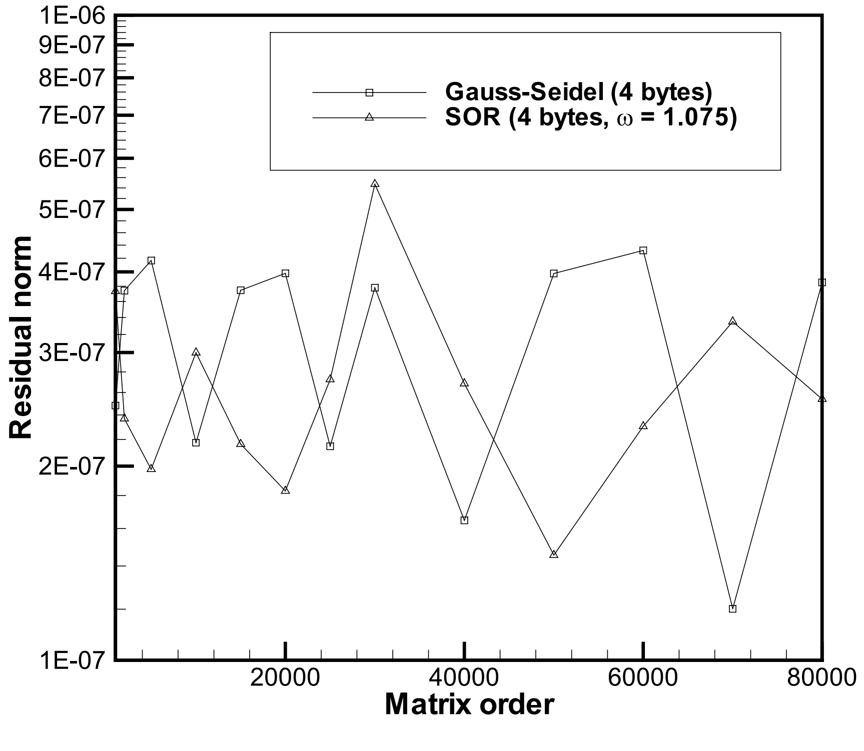

Figure 2. Variation of residual norm with matrix order; variables of single precision.

**5.2. Test using the SOR method with CRS data structure**

The Gauss-Seidel method, Eq. (8), is a particular case of relaxation methods, Eq. (10), when $\omega = 1$. Table (3) shows a comparison of results for both Gauss-Seidel method and SOR method with $\omega = 1$. As expected the results are equals. The relaxation factor has the function to force the residual goes to zero faster than in Gauss-Seidel method. But if ω value is badly chosen the SOR can be worse than Gauss-Seidel. This fact puts the necessity for an automatic process to setup the ω value, optimally. According with Barret et al. (1994) the factor of relaxation must be in the open interval (0,2). Figure (5) shows the number of iterations as a function of the relaxation factor. Those curves suggest, clearly, an optimal value for ω. The system solved in the case showed in Fig. (5) had dimension original of 30000 by 30000; with the using of the CRS data structure only about $9x10^4$ floating points values (the nonzero coefficients) need to be stored, while for the full matrix would be necessary store $9x10^8$ real values. In this case the sparsity of the matrix is something like 99.999999 per cent. Less than 0.1% of real data and more than 99.9% of zeroes were stored if the full matrix data structure was used. The dimension of the vectors *val* and *col_ind* would be $9x10^4$ locations each one for floating points and integers respectively, while the vector *row_ptr* would have 30001 integer locations. In memory size, the full matrix would require about 6.9 Gbytes for storing the double precision data. With the CRS data structure we need about 0.69 Mb for double precision data in vector *val* plus 0.46 Mb for long integer data in vectors *col_ind* and *row_ptr*.

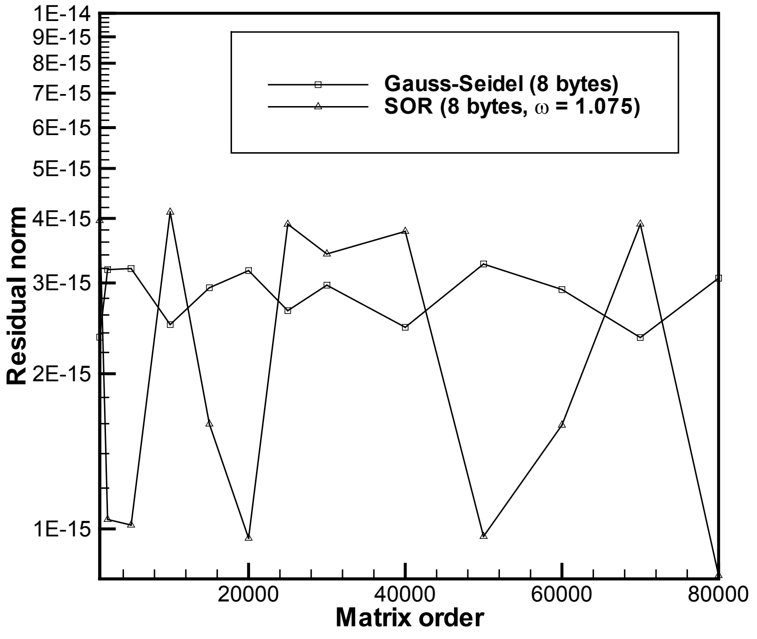

Figure 3. Variation of residual norm with matrix order; variables of double precision.

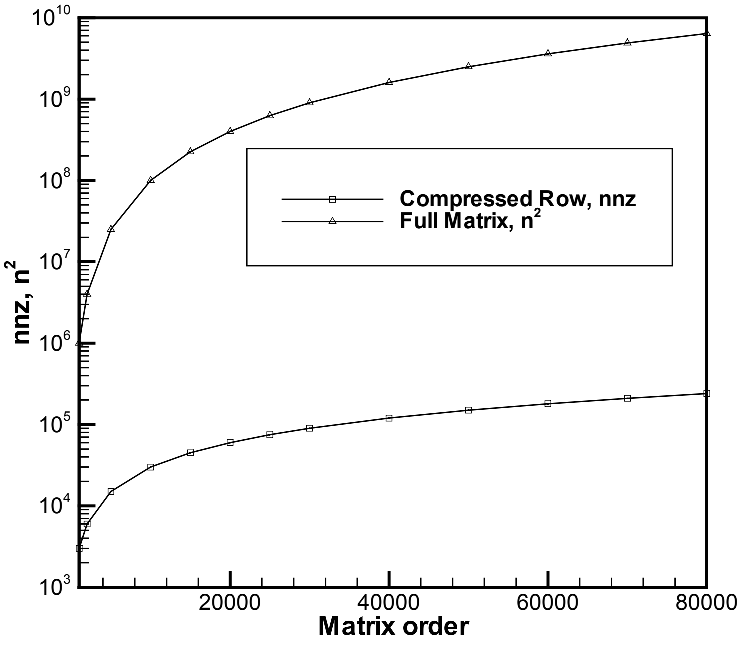

Figure 4. Storage requirements as a function of matrices order, for full matrix and compressed row storage

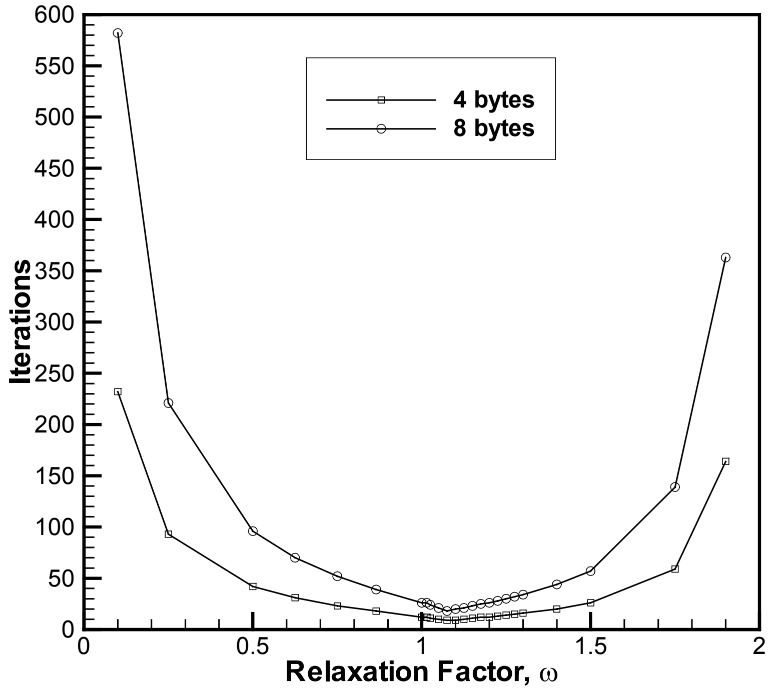

Figure 5. Variation of the number of iterations depending on the relaxation factor ω.

Table 3. Results for Gauss-Seidel method and SOR method with ω =1

| | Matrix order, n =30000 | |
|---|---|---|
| | Gauss-Seidel Method | SOR Method (ω =1) |
| Iterations for single precision | 12 | 12 |
| Norm of residual (4 bytes) | 3.780947E-07 | 3.780947E-07 |
| Norm of increment (4 bytes) | 9.501666E-08 | 9.501666E-08 |
| Iterations for double precision | 26 | 26 |
| Norm of residual (8 bytes) | 2.969775432495640E-015 | 2.969775432495640E-015 |
| Norm of increment (8 bytes) | 7.271132236455295E-016 | 7.271132236455295E-016 |

## 6. Acknowledgement

The authors would like to thank **Fundação de Amparo a Pesquisa do Estado de São Paulo - FAPESP** for supporting this project, **Process Number 00/06513-0** and the **Fundação para o Desenvolvimento da UNESP – FUNDUNESP**.

## 7. References

Axelsson, O., 1996, "Iterative Solutions Methods", Cambridge University Press, New York, USA, 644 p.

Barret, R., Berry, M., Chan, T.F., Demmel, J., Donato, J.M. Dongarra, J. Eijkhout, V., Pozo, R. Romine, C. & Van der Vorst, H., 1994, "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods", SIAM, Philadelphia.

Duff, I. S., 1999, "A Brief Bibliografy of Recent Research and Software for the Parallel Solution of Large Sparse Linear Equations", Technical Report TR/PA/99/12, CERFACS, Toulouse, Fr.

Golub, G. H., Loan, C. F. Van, (1996), "Matrix Computations", 3rd Edition, John Hopkins Univ. Press, 692p.

Hackabush, W., (1994), "Iterative Solution of Large Sparse Systems of Equations", Applied Mathematical Sciences, V. 95), Springer Verlag

Kotlyar, V., 1999, "Relational Algebraic Techniques for the Synthesis of Sparse Matrix Programs", Ph.D. Dissertation, Cornell University, 246p.

Saad, Y., van der Vorst, H.A. (1999) "Iterative Solution of Linear Sparse Systems in the 20-th Century", Technical Report UMSI-99-152, Minnesota Supercomputer Institute, University of Minnesota.

## 7. Copyright Notice

**Appendix A**

In the following we present some definitions and properties of vector and matrix norms.

**A.1 Definition of a vector norm**

Let x $\in \Re^n$ and y $\in \Re^n$. A vector norm in $\Re^n$ space is a function *f*: $\Re^n \rightarrow \Re$ satisfying the following properties:

- $f(\mathbf{x}) = 0 \quad \text{if} \quad \mathbf{x} = \mathbf{0},$ (19)
- $f(\mathbf{x}) \geq 0,$ (20)
- $f(\mathbf{x}+\mathbf{y}) \leq f(\mathbf{x}) + f(\mathbf{y}),$ (21)
- $f(\alpha\mathbf{x}) = |\alpha| f(\mathbf{x}), \ \alpha \in \Re.$ (22)

This function $f(\mathbf{x})$ is denoted by $f(\mathbf{x}) = \|\mathbf{x}\|$.
A class of useful norms are the *p*-norms, defined as

$$\|\mathbf{x}\|_p = (|x_1|^p + \cdots + |x_p|^p)^{1/p} = (\sum_{i=1}^{n} |x_i|^p)^{1/p}, \quad p \geq 1 \tag{23}$$

The most popular of the norms defined above are the cases $p$ = 1, 2, $\propto$, as follow

$$\|\mathbf{x}\|_1 = |x_1| + \cdots + |x_n| = \sum_{i=1}^{n} |x_i| \tag{24}$$

$$\|\mathbf{x}\|_2 = (|x_1|^2 + \cdots + |x_n|^2)^{1/2} = (\sum_{i=1}^{n} |x_i|^2)^{1/2} = (\mathbf{x}^\mathbf{T}\mathbf{x})^{1/2} \tag{25}$$

$$\|\mathbf{x}\|_\infty = \lim_{p\to\infty} (|x_1|^p + \cdots + |x_n|^p)^{1/p} = \lim_{p\to\infty} (\sum_{i=1}^{n} |x_i|^p)^{1/p} = \max_{1\leq i\leq n} |x_i| \tag{26}$$

**A.2 Definition of a matrix norm**

Let $\mathbf{A} \in \Re^{m\times n}$ and $\mathbf{B} \in \Re^{m\times n}$. Once the vectorial space $\Re^{m\times n}$ is isomorphic with the vectorial space $\Re^{mn}$, the definition of a matrix norm must be equivalent to the definition of a vector norm. In particular, if *f*: $\Re^{m\times n} \rightarrow \Re$ is a norm of a matrix, the following properties are satisfied:

- $f(\mathbf{A}) = 0 \quad \text{if} \quad \mathbf{A} = \mathbf{0},$ (27)
- $f(\mathbf{A}) \geq 0,$ (28)
- $f(\mathbf{A}+\mathbf{B}) \leq f(\mathbf{A}) + f(\mathbf{B}),$ (29)
- $f(\alpha\mathbf{A}) = |\alpha| f(\mathbf{A}), \quad \alpha \in \Re.$ (30)

Adopting the same notation, the norm of a matrix can be defined as $\|\mathbf{A}\| = f(\mathbf{A})$. The most used norms of a matrix in linear algebra are the Frobenius norm

$$\|\mathbf{A}\|_F = (\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2)^{1/2} \tag{31}$$

and a class of *p*-norms

$$\left\|\mathbf{A}\right\|_p = \sup_{\mathbf{x}\neq\mathbf{0}} \frac{\|\mathbf{Ax}\|_p}{\|\mathbf{x}\|_p} \tag{32}$$

In above equation, the term $\|\mathbf{x}\|_p$ is a scalar so we can write

$$\left\|\mathbf{A}\right\|_p = \sup_{\mathbf{x}\neq\mathbf{0}} \|\mathbf{A}\frac{\mathbf{x}}{\|\mathbf{x}\|_p}\|_p \tag{33}$$

Clearly, the term $(\mathbf{x}/\|\mathbf{x}\|_p)$ is unity vector, then the $p$-norms can be also defined as

$$\left\|\mathbf{A}\right\|_p = \max_{\|\mathbf{x}\|_p=1} \|\mathbf{Ax}\|_p \tag{34}$$

The norms $f_1$, $f_2$ e $f_3$ in $\Re^{m\times q}$, $\Re^{m\times n}$, $\Re^{n\times q}$ are called *mutually consistent*, if for all $\mathbf{A}\in\Re^{m\times n}$, $\mathbf{B}\in\Re^{n\times q}$ one has $f_1(\mathbf{AB}) \leq f_2(\mathbf{A})\, f_3(\mathbf{B})$. Notice that neither all matrices satisfy the product property

$$\left\|\mathbf{AB}\right\| \leq \left\|\mathbf{A}\right\| \left\|\mathbf{B}\right\| \tag{35}$$

Some properties of the $p$-norms (specially for $p = 1, 2, \propto$) satisfy equalities and inequalities. If $\mathbf{A}\in\Re^{m\times n}$, then

$$\left\|\mathbf{A}\right\|_1 \leq \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}| \tag{36}$$

$$\left\|\mathbf{A}\right\|_\infty \leq \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}| \tag{37}$$

$$\left\|\mathbf{A}\right\|_2 \leq \sqrt{\left\|\mathbf{A}\right\|_1 \left\|\mathbf{A}\right\|_\infty} \tag{38}$$