IMPLEMENTING ALGORITHMS FOR SOLVING SPARSE NONLINEAR SYSTEMS OF EQUATIONS

Márcia A. Gomes-Ruggiero
José Mario Martínez
and
Antonio Carlos Moretti

RELATÓRIO TÉCNICO Nº 19/89

Abstract. In this paper we describe implementations of seven algorithms of Newton and Quasi–Newton type for solving large sparse systems of nonlinear equations. For linear algebra calculations we use a symbolic manipulation and a static data structure introduced recently by George and Ng, which allow a partial pivoting strategy for solving linear systems. We present an example of a numerical comparison of the implemented methods.

Universidade Estadual de Campinas
Instituto de Matemática, Estatística e Ciência da Computação
Caixa Postal 6065
13.081 – Campinas – SP
BRASIL

Maio – 1989

Implementing Algorithms for Solving Sparse Nonlinear Systems of Equations

Márcia A. Gomes–Ruggiero
José Mario Martínez
and
Antonio Carlos Moretti[1]

Abstract. In this paper we describe implementations of seven algorithms of Newton and Quasi–Newton type for solving large sparse systems of nonlinear equations. For linear algebra calculations we use a symbolic manipulation and a static data structure introduced recently by George and Ng, which allow a partial pivoting strategy for solving linear systems. We present an example of a numerical comparison of the implemented methods.

Key Words: Nonlinear systems of equations, sparse matrices, L-U factorizations, Newton's method, Quasi–Newton methods.

AMS: 65H10.

[1]Applied mathematics Lab., IMECC – UNICAMP, CP 6065, 13081 - Campinas, SP, Brazil

1. Introduction

Many real–life problems require the solution of large systems of nonlinear equations:

$$F(x) = 0$$

$$F = (f_1, \ldots, f_n)^T, \tag{1.1}$$

where $F : \mathbb{R}^n \to \mathbb{R}^n$ is a nonlinear C^1–function, and its Jacobian matrix $J(x)$ is sparse (see [10, 27, 30]). The best known method for solving this type of problems is Newton's method. This is an iterative method, where the successive approximations to the solution of (1.1) are calculated according to the following formula:

$$x^{k+1} = x^k - J(x^k)^{-1} F(x^k). \tag{1.2}$$

Hence, at each iteration of Newton's method, the derivatives $\dfrac{\partial f_i}{\partial x_j}$ must be calculated, and the linear $n \times n$ system

$$J(x^k)s = -F(x^k) \tag{1.3}$$

must be solved, in order to obtain x^{k+1}. When analytic derivatives are not available, they may be estimated using finite differences (see [5]).

Quasi–Newton methods [1–4, 7–10, 16, 21–24, 26, 28, 32] were also introduced to deal with situations where analytic derivatives are not available, or are very expensive to calculate. They obey the formulae:

$$B_k s = -F(x^k), \tag{1.4}$$

$$x^{k+1} = x^k + s. \tag{1.5}$$

At each iteration of a Quasi–Newton method, only the function values $F(x^k)$ are calculated and the linear system (1.4) is solved. The new matrix B_{k+1} is obtained from B_k using recurrence relations which only involve x^k, x^{k+1}, $F(x^k)$ and $F(x^{k+1})$ (see [10]). Usually, B_{k+1} is chosen as one of the matrices which satisfy the "secant equation":

$$B_{k+1}s = y = F(x^{k+1}) - F(x^k). \tag{1.6}$$

The best known Quasi–Newton method for small dense problems is Broyden's first method [1, 3, 9, 10]. This method uses a rank–one correction matrix to obtain B_{k+1} from B_k:

$$B_{k+1} = B_k + \frac{(y - B_k s)s^T}{s^T s}. \tag{1.7}$$

Using (1.7) and the Sherman–Morrison formula [15, page 3], B_{k+1}^{-1} may be obtained from B_k^{-1} using $O(n^2)$ flops. Moreover, a Q-R factorization of B_{k+1} may be obtained from a Q-R factorization of B_k using $O(n^2)$ flops, (see [26]). Hence, if (1.4), (1.5), (1.7) are used, not only the time for computing derivatives is saved, but also the computational work for solving linear systems may be considerably reduced, in relation to the computational work needed to solve (1.3) ($O(n^3)$ flops). For this reason, in many cases, Broyden's first method may be more efficient than Newton's method, even when derivatives are easily available, in spite of its lower speed of convergence.

The situation in the large sparse case is somewhat different. In fact, if B_k is a sparse matrix, and (1.7) is used, B_{k+1} generally turns out to be a dense matrix, which may have little relation to true Jacobian matrices. Broyden [2] and Schubert [28, 24] developed a variant of Broyden's first method where matrices B_k keep the same pattern of sparsity as $J(x^k)$, satisfying the secant equation (1.6), and using a minimum variation principle. However, the difference $B_{k+1} - B_k$ is no longer a rank–one matrix as in Broyden's first method, and so, no easy relationship between factorizations of B_k and B_{k+1} seems to be possible. Therefore, when Schubert's method is used to generate the B_k's, the resolution of (1.4) is as expensive as the resolution of (1.3).

These observations motivated Dennis and Marwil [7] to develop the first Quasi–Newton method where an L-U factorization of B_{k+1} is obtained directly from an L-U factorization of B_k, giving a substantially lower cost in the resolution of (1.4) in relation to the resolution of (1.3). Basically, the Dennis– Marwil method keeps the L–factor fixed and modifies the U–factor from one iteration to the following, preserving the sparsity pattern of U and using a Schubert–type formula. Unhappily, convergence properties of the Dennis - Marwil method are not quite satisfactory. In fact, local convergence is only obtained if the algorithm is restarted with $B_k = J(x^k)$ when k is multiple of a fixed integer q (see [7]). Martínez [21] introduced a method where the LDM^T factorization of B_k (see [15]) is stored and only the factor D is modified from one iteration to another, in order to obtain the factorization of B_{k+1}. This method belongs to a larger family introduced later in [22]. Unlike the Dennis–Marwil method, the methods in this family have local convergence properties without restarts, but superlinear convergence is only obtained using restart procedures (see [22]). Chadee [4], generalizing a method of Johnson and Austria [16], introduced a locally superlinear method where the L-U factorization of B_{k+1} is obtained simultaneously modifying the L-U factors of B_k. Unfortunately, the inverses of the triangular matrices L must have a definite sparsity pattern for Chadee's method to be useful, and so, its applicability seems to be limited to special structures of the Jacobian matrices. Martínez [23] introduced a very large family which includes most

superlinearly convergent methods for solving systems of nonlinear equations. The Dennis–Marwil method does not belong to this family, but it may be interpreted as a limit case ($\theta \to 1$) of a parametric subfamily where the case $\theta = \frac{1}{2}$ is Chadee's method.

In this paper, we compare Newton's method, the Modified Newton Method, Schubert's Method, the Dennis–Marwill Method, and three methods in family [22]. The implementation of methods for solving sparse nonlinear systems of equations requires a decision about the algorithm which is going to be used for linear algebra calculations. V. R. Lopes [17, 18] called our attention to some ill–conditioned banded linear systems, derived from approximation of diffusion problems using variational principles (see [18, 33]). According to [17], the resolution of these systems using a general purpose sparse linear system solver [11] with a default tolerance parameter $a = 0.1$ was completely unreliable, but good results were obtained using a very strict tolerance $a = 0.999$. This choice is equivalent to the use of L-U factorizations with partial pivoting (see [12, 15]) and represents the most stable way of solving linear systems using L-U factorizations. Another reason, of a more theoretical nature, led us to the decision of using L-U factorizations with partial pivoting. Namely, the local convergence theorems for the methods introduced in [7, 21, 22] impose that the pivoting rule which allows the L-U factorization of $J(x^0)$ will also allow the L-U factorization of $J(x^*)$, the Jacobian matrix at the solution. This objective is most likely attained if the partial pivoting rule is used, since it is intuitively evident that, the larger the chosen pivots are for the factorization of a given matrix, the greater is the distance between that matrix and the set of matrices for which a zero pivot appears using the same permutation rule. According to these observations, we decided to use the George–Ng [14] factorization algorithm, which uses partial pivoting, a static data structure and a symbolic factorization scheme to predict fill–in in calculations, for all the linear algebra manipulations of our algorithms. The numerical results are not independent from this decision: superiority of a particular method for a given problem may be challenged if the algorithms for factorizations, data structures, resolution of triangular systems, etc., are changed. This paper is organized as follows: In Section 2 we describe the set of algorithms used for our comparison. Features like step control, restart procedures and stopping criteria are discussed. In Section 3 we survey the theoretical properties of the methods involved in this study. In Section 4 we describe our numerical experiments, criteria for comparison are discussed, and results are commented.

Finally, in Section 5 we state some conclusions and we suggest some lines for future research.

2. The Algorithms

As we mentioned in the Introduction, we selected seven algorithms for our comparison: Newton's method, the Modified Newton method, Schubert's method, the Dennis–Marwil method, and three methods of Martínez's family [21, 22]. We call them the Diagonal method, the Row–Scaling method and the Column–Scaling method. All these methods use a small tolerance parameter $TOL > 0$, to detect and modify singularity of the matrices involved, and a step control Δ, to inhibit very large steps $x^{k+1} - x^k$. Moreover, a symbolic phase preceeds the first iteration of all the algorithms, and even the first iteration is common to all of them. The symbolic phase corresponds to the symbolic manipulation of the George–Ng algorithm [14], and the first iteration corresponds to the following algorithm with $k = 0$.

Algorithm 2.1 (Newton Iteration)

Step 1. Compute $F(x^k)$, $J(x^k)$. Set $B_k = J(x^k)$.

Step 2. Compute a permutation P, a lower triangular matrix $L = (\ell_{ij})$, an upper-tringular matrix $U = (u_{ij})$, and $2n$ sets I_i^L, I_i^U, $i = 1, \ldots, n$ such that:

$$PB_k = LU, \tag{2.1}$$

$$\ell_{ii} = 1, \quad i = 1, \ldots, n. \tag{2.2}$$

$$|\ell_{ij}| \leq 1 \quad \text{for all} \quad j = 1, \ldots, i, \quad i = 1, \ldots, n. \tag{2.3}$$

$$I_i^L \subset \{1, \ldots, n\}, \tag{2.4}$$

$$I_i^U \subset \{1, \ldots, n\},$$

$$i = 1, \ldots n.$$

$$\ell_{ij} = 0 \quad \text{for all} \quad j \notin I_i^L, \quad j \neq i, \quad i = 1, \ldots, n \tag{2.5}$$

$$u_{ij} = 0 \quad \text{for all} \quad j \notin I_i^U, \quad j \neq i, \quad i = 1, \ldots n. \tag{2.6}$$

Step 3. If $|u_{ii}| < TOL$, replace u_{ii} by

$$sg(u_{ii})TOL, \quad i = 1, \ldots, n.$$

Step 4. Solve

$$Lw = -PF(x^k), \quad \text{and} \tag{2.7}$$

$$Us = w. \tag{2.8}$$

Step 5. If $\|s\|_\infty > \Delta$, replace s by $s\Delta/\|s\|_\infty$.

Step 6. $x^{k+1} = x^k + s$.

Matrices L, U, and the sets I_i^L, I_i^U which satisfy (2.1) – (2.6) are computed using the George–Ng algorithm. ℓ_{ij} may be equal to zero for some $j < i$, $j \in I_i^L$, and u_{ij} may be equal to zero for some $j > i$, $j \in I_i^U$. In fact, the sets I_i^L, I_i^U represent the structural nonzero elements of any pair of matrices L, U when the partial pivoting algorithm is performed on a matrix with the nonzero structure of PB_k.

Algorithm 2.1 is a Newton iteration, with the safeguards against singularity and large steps given by steps 3 and 5. Therefore, our Newton Method may be described by the following algorithm:

Algorithm 2.2 (Newton's Method)
Given an arbitrary initial point x^0, compute x^{k+1}, $k = 0, 1, 2, \ldots$ using Algorithm 2.1.

The Modified Newton method is described by the following algorithm.

Algorithm 2.3 (Modified Newton Method)
Given x^0, compute x^1 using Algorithm 2.1. For $k = 1, 2, \ldots$, compute x^{k+1} performing Steps 4 to 6 of Algorithm 2.1.

Our implementation of Schubert's method requires the definition of n additional sets of indexes $I_i \subset \{1, \ldots, n\}$, $i = 1, \ldots n$. We define:

$$I_i = \{j \in \{1, \ldots, n\} \mid \frac{\partial f_i}{\partial x_j}(x) \neq 0 \quad \text{for some } x \text{ in the domain of } f\}. \tag{2.9}$$

With this definition Schubert's method is described by Algorithm 2.4.

Algorithm 2.4 (Schubert's method)

Given x^0, compute x^1 using Algorithm 2.1. For $k = 1,2,\ldots$, compute x^{k+1} performing the following steps:

Step 1. Compute $y = F(x^k) - F(x^{k-1})$.

Step 2. Solve the optimization problem

$$\begin{aligned} &\text{Minimize} \quad ||B - B_{k-1}||_F \\ &\text{s.t.} \quad B = (b_{ij}), \quad b_{ij} = 0 \quad \text{if} \quad j \notin I_i, \quad i = 1,\ldots,n. \\ &\qquad Bs = y \end{aligned} \tag{2.10}$$

Compact formulae for the resolution of (2.10) are given in [10, 28].

Let us call B_k the solution of (2.10).

Step 3. Perform steps 2 to 6 of Algorithm 2.1.

Algorithm 2.5, 2.6, 2.7 and 2.8 describe the Dennis–Marwil method, the Diagonal method, the Row–Scaling method and the Column–Scaling method respectively.

Algorithm 2.5 (Dennis–Marwil method)

Given x^0 compute x^1 using algorithm 2.1. For $k = 1,2,\ldots$ compute x^{k+1} performing the following steps:

Step 1. Compute $y = F(x^k) - F(x^{k-1})$.

Step 2. Solve $Lw = Py$.

Step 3. Define $v = Us$.

Step 4. For $i = 1,\ldots,n$ perform Step 5.

Step 5. If

$$\Big(\sum_{j \in I_i^U} s_j^2\Big)^{1/2} \geq 10^{-4} \cdot ||s||_\infty$$

perform Step 5.1. Otherwise, increment i and repeat Step 5.

Step 5.1. For each j, such that, $u_{ij} \neq 0$, compute:

$$u_{ij} \longleftarrow u_{ij} + s_j \cdot [w_i - v_i] \Big/ \sum_{j \in J_i^U} s_j^2.$$

Step 6. Perform Steps 3 to 6 of algorithm 2.1.

Algorithm 2.6 (Diagonal Scaling method)

Given x^0, compute x^1 using Algorithm 2.1. Set $D = \text{Diag}(d_{ii}) = \text{Diag}(u_{ii})$ and replace U by $D^{-1}U$. For $k = 1, 2, \ldots$ compute x^{k+1} performing the following steps:

Step 1. Compute $y = F(x^k) - F(x^{k-1})$.

Step 2. Solve

$$Lw = Py.$$

Step 3. Define

(2.11) $$v = Us.$$

Step 4. For $i = 1, \ldots, n$, execute Step 5.

Step 5. If

$$|v_i| \geq 10^{-4} \|s\|_\infty$$

set $d_{ii} = w_i / v_i$, if not, then increment i and repeat this step.

Step 6. For $i = 1, \ldots, n$, if $|d_{ii}| < TOL$, replace

$$d_{ii} \longleftarrow sg(d_{ii}) TOL.$$

Step 7. Solve

$$LDUs = -PF(x^k).$$

Step 8. Perform Steps 5 and 6 of Algorithm 2.1.

Algorithm 2.7 (Row–Scaling method)

Given x^0, compute x^1 using Algorithm 2.1. Define $D = \text{Diag}(d_{ii})$. $d_{ii} = 1$, $i = 1, \ldots, n$. Compute x^{k+1} performing the following steps:

Step 1. Compute $y = F(x^k) - F(x^{k-1})$.

Step 2. Define

(2.12) $$v = LUs.$$

Step 3. Define $w = Py$.

Step 4. For $i = 1, \ldots, n$, execute Step 5.

Step 5. If

$$|v_i| \geq 10^{-4} ||F(x^k)||_\infty$$

set $d_{ii} = w_i / v_i$, otherwise increment i and repeat Step 5.

Step 6. For $i = 1, \ldots, n$, if $|d_{ii}| < TOL$, replace

$$d_{ii} \longleftarrow sg(d_{ii}) TOL.$$

Step 7. Solve

$$DLUs = -PF(x^k). \tag{2.13}$$

Step 8. Execute Steps 5 and 6 of Algorithm 2.1.

Algorithm 2.8 (Column–Scaling method)

Given x^0, compute x^1 using Algorithm 2.1. Define $D = \text{Diag}(d_{ii})$, $d_{ii} = 1$, $i = 1, \ldots, n$. For $k = 1, 2, \ldots$, compute x^{k+1} performing the following steps:

Step 1. Define z the solution of

$$LUz = PF(x^{k-1}). \tag{2.14}$$

Step 2. Solve

$$LUw = PF(x^k). \tag{2.15}$$

Step 3. Set

$$v = w - z.$$

Step 4. For $i = 1, \ldots, n$, execute Step 5.

Step 5. If

$$|s_i| \geq 10^{-4} ||s||_\infty$$

set $d_{ii} = v_i / s_i$ otherwise increment i and repeat Step 5.

Step 6. For $i = 1, \ldots, n$, if $|d_{ii}| < TOL$, replace

$$d_{ii} \longleftarrow sg(d_{ii}) TOL.$$

Step 7. Solve

$$Ds = -w.$$

Step 8. Execute Steps 5 and 6 of Algorithm 2.1.

Remarks.

We used the words "solve" or "compute" to indicate that some calculation must effectively be performed, and the word "define" to indicate that the result of the calculation may be obtained from previous computations.

Singularity and Step Control

Step 3 of Algorithm 2.1 and Step 6 of Algorithm 2.6, 2.7 and 2.8 correct the L-U factorization when a nearly singular matrix B_k appears. However, a very ill-conditioned B_k may still occur, provoking very large steps $-B_k^{-1}F(x^k)$. Therefore, in practical implementation of methods for solving nonlinear systems of equations some control in the size of increments $x^{k+1} - x^k$ is recommended (see [20, 25]). In our codes, we adopted a very simple way of controlling the stepsize: assuming that Δ is given by the user as an estimate of the distance between the initial guess and the solution, we simply test if $||s||_\infty = ||-B_k^{-1}F(x^k)||_\infty$ is less than Δ or not. If it is, then the increment s is accepted. Otherwise, it is replaced by $s\,\Delta/||s||_\infty$. This is done at Steps 5 and 6 of Algorithm 2.1.

Stopping Criteria

A natural stopping criterion for algorithms which solve nonlinear systems is

$$||F(x^k)||_\infty \leq \varepsilon_1 \tag{2.16}$$

where ε_1 is a small positive number given by the user. When (2.16) occurs, we declare "convergence of type 0".

However, sometimes criterion (2.16) is very difficult or impossible to achieve, maybe because of a large Jacobian at the solution. So, we incorporate another convergence test:

$$||s||_\infty < \varepsilon_2. \tag{2.17}$$

When (2.17) holds, we declare "covergence of type 1".

Algorithm like 2.2 – 2.8 are local in nature, and, therefore, divergence may occur for arbitray nonlinear systems if x^0 is far from the solution. We declare "Divergence" when, for some large positive number BIG, given by the user, the following inequality holds:

$$||F(x^k)||_\infty > BIG. \tag{2.18}$$

Finally, the execution of the programs is interrupted when either a previously defined computer time or some large number of iterations is exceeded.

Restarting Criteria

Sometimes, rather than executing Algorithm 2.3 – 2.8 in their original version, it is more efficient to restart the iterative process, performing a Newton iteration (Algorithm 2.1) instead of the original iteration, at certain steps k. The most natural restarting criterion is to execute Algorithm 2.1 for obtaining x^{k+1} when k is a multiple of a fixed integer q. For some algorithms, and considering only the computational work involved in function evaluations, an optimal q may be determined using Ostrowski's efficiency index (see [19, 31]). However, in our case, the computational work involved in linear algebra calculations is not negligible at all, so, we need restarting criteria which take this work into account. We developed a "local efficiency restarting criterion" ($LERC$) based on the following arguments:

Let us call t_k the computer time used by some algorithm at iteration k. Assume, further, that

$$\|F(x^{k+1})\| = \theta_k \|F(x^k)\|. \tag{2.19}$$

Therefore, if relation (2.19) is mantained throughout the calculation, with $\theta_k < 1$ and the same type of iteration is used, the computer time to achieve (2.16) should be proportional to $-t_k/\log \theta_k$. This justifies the defining of E_k, the efficiency of iteration k, as

$$\begin{aligned} E_k &= \frac{-\log \theta_k}{t_k}, \quad \text{if} \quad \theta_k < 1, \quad \text{or} \\ E_k &= 0 \qquad\qquad \text{otherwise.} \end{aligned} \tag{2.20}$$

Assume now that ℓ is the last index of a Newton iteration previous to iteration k. We adopt the following criterion for deciding whether to use a "normal" iteration or a Newton type one at iteration $k+1$:

(i) If $\theta_k > 1$, iteration $k+1$ must be a Newton iteration.

(ii) If $E_k > E_\ell$, use a "normal" iteration for obtaining x^{k+1}, and if $E_k \leq E_\ell$ use a Newton iteration.

A similar criterion is used in [6] for minimization problems, with good numerical results.

3. Theoretical Properties

Let us assume that $F : \Omega \subset \mathbb{R}^n \to \mathbb{R}^n$, Ω an open and convex set, $F \in C^1(\Omega)$, $F(x^*) = 0$, $J(x^*)$ nonsingular, and, for all $x \in \Omega$,

$$||J(x) - J(x^*)|| \leq L\,||x - x^*||^p$$

for some L, $p > 0$.

For a local convergence analysis, consider the algorithms described in Section 2, without correction of singularity, and with no control of the stepsize. That is, eliminate Steps 3 and 5 of Algorithm 2.1, Step 4 of Algorithm 2.5, and Step 6 of Algorithms 2.6, 2.7 and 2.8. In fact, local convergence theorems show that these steps are not necessary with the hypotheses above, if x^0 is near enough x^*. Let us survey here the convergence results related to Algorithms 2.2 - 2.8. The first one concerns local convergence of Newton's method and the Modified Newton method.

Theorem 3.1. Given $r \in (0, 1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that, if $||x^0 - x^*|| \leq \varepsilon$, the sequences (x^k) generated by Algorithms 2.2 or 2.3 converge to x^* and satisfy

$$||x^{k+1} - x^*|| \leq r\,||x^k - x^*|| \tag{3.1}$$

for all $k = 0, 1, 2, \ldots$. Moreover, for Algorithm 2.2 (Newton's method), there exists $c > 0$ such that:

$$||x^{k+1} - x^*|| \leq c\,||x^k - x^*||^{p+1} \tag{3.2}$$

for all $k = 0, 1, 2, \ldots$.

Proof. See [10, 27, 30].

Like many Quasi–Newton algorithms, Schubert's method satisfies not only the linear convergence result (3.1), but a stronger (superlinear) convergence result which, on the other hand, is weaker than (3.2).

Teorem 3.2. Given $r \in (0, 1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that, if $||x^0 - x^*|| \leq \varepsilon$, the sequence (x^k) generated by Algorithm 2.4 converges to x^* and satisfies (3.1). Moreover, the speed of convergence is Q–superlinear. That is,

$$\lim_{k \to \infty} ||x^{k+1} - x^*||/||x^k - x^*|| = 0. \tag{3.3}$$

Proof. See [10, 24].

Algorithms 2.6, 2.7 and 2.8 have the same type of convergence result as the Modified Newton method:

Theorem 3.3. Given $r \in (0,1)$, there exists $\varepsilon = \varepsilon(r) > 0$ such that, if $\|x^0 - x^*\| \leq \varepsilon$, the sequences (x^k) generated by Algorithms 2.6, 2.7 or 2.8 converge to x^* and satisfies (3.1).

Proof. See [21] for the convergence of Algorithm 2.6. For proving the convergence of the Algorithms 2.7 and 2.8 we need to interpret them as particular cases of the family introduced in [22]. This may be easily done: for Algorithm 2.7, $C(B) = I$, $D(B) = I$, $E(B) = B$ and for Algorithm 2.8, $C(B) = B$, $D(B) = I$, $E(B) = I$. Since these functions are trivially continuous, Theorem 2.1 of [22] may be applied.

We are almost sure that the Dennis–Marwil method, without restarts, is not locally convergent. However, the members of a closely related family of methods introduced recently in [23], have local and superlinear convergence. We call the members of this family "Quasi–Dennis–Marwil" methods. Given $\alpha \in (0,1)$, the Quasi–Dennis–Marwill method defined by α may be described by the following algorithm:

Algorithm 3.1.

Given x^0, obtain x^1 using Algorithm 2.1. For computing x^k, $k = 1, 2, \ldots$ use the recurrence:

$$x^{k+1} = x^k - (L_k U_k)^{-1} P F(x^k), \tag{3.4}$$

computing, at each iteration, L_{k+1}^{-1}, U_{k+1} as the solution of the following optimization problem

$$\begin{aligned} \text{Minimize} \quad & \alpha \|M - L_k^{-1}\|_F + (1-\alpha)\|U - U_k\|_F \\ \text{s.t.} \quad & Us = My \\ & s = x^{k+1} - x^k \\ & y = P[F(x^{k+1}) - F(x^k)] \\ & U = (u_{ij}) \mid u_{ij} = 0 \quad \text{if} \quad j \notin I_i^U. \end{aligned} \tag{3.5}$$

The relation between the Dennis–Marwil method and the method described above is given in the following theorem:

Theorem 3.4. Suppose x^k, L_k, U_k are given, and let us call L_{k+1}, U_{k+1} the matrices obtained using the Dennis–Marwil method. Of course, $L_{k+1} = L_k$. Let us call $L_{k+1}(\alpha)$, $U_{k+1}(\alpha)$ the ones obtained using Algorithm 3.1. Then,

$$\lim_{\alpha \to 1} L_{k+1}(\alpha) = L_{k+1} \tag{3.6}$$

$$\lim_{\alpha \to 1} U_{k+1}(\alpha) = U_{k+1}. \tag{3.7}$$

Proof. See [23].

The Quasi–Dennis–Marwil methods are not practical for solving sparse nonlinear systems because sparsity of L_k is not preserved from one iteration to another. However, they have the same local convergence properties as Schubert's method (see [23]). Therefore, although the Dennis–Marwil method seems to have the poorest convergence properties among the algorithms described in Section 2, the fact that "very analogous" methods in the sense of (3.6) – (3.7) have good local convergence properties makes us to feel that "some part" of these properties is inherited by the Dennis–Marwil method.

Up to now, we have considered the convergence properties of the "pure" algorithms of Section 2. If restarts are incorporated, the convergence results for Algorithms 2.3 and 2.8 look very similar:

Theorem 3.5. Assume that (x^k) is obtained using Algorithm 2.3, 2.6, 2.7 or 2.8, except that, for $k \in K_0$, an infinite set of indexes, x^{k+1} is calculated using Algorithm 2.1. Then, there exists $\varepsilon > 0$ such that, if $\|x^0 - x^*\| \leq \varepsilon$, x^k converges to x^* and (3.3) also holds.

Proof. See [22, 27], or use Theorems 3.1 and 3.3.

For the Dennis–Marwil method the following slightly weaker result holds:

Theorem 3.6. Assume that (x^k) is obtained using Algorithm 2.5, except that, for $k \in K_0$, an infinite set of indexes, x^{k+1} is computed using Algorithm 2.1. Assume, further, that the difference between any pair of consecutive indexes of K_0 is never grater than a fixed integer q. Then the thesis of Theorem 3.5 holds for the sequence (x^k).

Proof. See [10].

4. Numerical Experiments

We wrote a FORTRAN code which implements the methods described in Section 2. All the reported tests were run in a VAX11/785 at the State University at Campinas, using the FORTRAN 77 compiler and the VMS Operational System. Single precision was used for real variables in all our tests. A compatible IBM–PC version of the code was also written using a Microsoft Fortran compiler. The results for this version were consistent with the results of the VAX version of the code.

Problem 1 (Broyden Tridiagonal) See [1, 2].

$$f_1(x) = (3 - kx_1)x_1 - 2x_2 + 1$$

$$f_i(x) = (3 - kx_i)x_i - x_{i-1} - 2x_{i+1} + 1 \quad i = 2, \ldots, n-1$$

$$f_n(x) = (3 - kx_n)x_n - x_{n-1} + 1.$$

$$x^0 = (-1, \ldots, -1)^T.$$

Algorithmic parameters: $\varepsilon_1 = \varepsilon_2 = 10^{-4}$, $TOL = 10^{-7}$,

$$\Delta = 10, \; BIG = 10^{10}.$$

Figure 1 shows the structure of the Jacobian matrix, and the date structure for L-U factorization with partial pivoting computed by the George–Ng method (with $n = 40$).

Table 1 shows the computer *CPU* times used by the symbolic phase of the methods (previous to the first iteration) for different n.

n	time/seconds
1000	0.23
2000	0.49
3000	0.73
4000	0.99
5000	1.24
6000	1.54

Table 1: *CPU* time of the symbolic phase for Problem 1.

Table 2 shows the mean computer CPU times used by a Newton iteration (Algorithm 2.1) for different n.

n	time/seconds
1000	0.24
2000	0.52
3000	0.77
4000	1.01
5000	1.30
6000	1.53

Table 2: Mean CPU time of a Newton iteration for Problem 1.

Table 3 shows the mean computer CPU times of one ordinary iteration of Algorithms 2.3 - 2.8. We use the following notation:

N : Newton's Method (Algorithm 2.2)
MN : Modified Newton Method (Algorithm 2.3)
S : Schubert's Method (Algorithm 2.4)
DM : Dennis–Marwil Method (Algorithm 2.5)
DS : Diagonal Scaling Method (Algorithm 2.6)
RS : Row–Scaling Method (Algorithm 2.7)
CS : Column–Scaling Method (Algorithm 2.8)

n	MN	S	DM	DS	RS	CS
1000	0.09	0.29	0.16	0.11	0.11	0.10
2000	0.17	0.63	0.33	0.22	0.23	0.22
3000	0.26	0.89	0.50	0.33	0.34	0.32
4000	0.36	1.17	0.68	0.46	0.43	0.43
5000	0.45	1.48	0.84	0.59	0.56	0.53
6000	0.54	1.77	1.03	0.68	0.68	0.65

Table 3: Mean CPU time of a typical iteration for Problem 1.

Observe that the computer CPU time used by different methods is a linear function of n. As expected, iterations of Newton's and Schubert's methods are the most expensive ones. The Jacobian matrix is easily available in this problem, so

Schubert's iteration is even more expensive than Newton's iteration.

In tables 4.1 to 4.4, we describe the overall numerical performance of the algorithms for problem 1. We write (IER, k, k_1, k_2, t) to indicate that a particular algorithm stopped with error code IER, using k iterations, k_1 Newton iterations, k_2 quasi–Newton iterations and t seconds CPU time. IER may assume five values: 0 for convergence of type 0, 1 for convergence of type 1, 2 for divergence, 3 for exceeded number of iterations and 4 for exceed CPU time. We tested problem 1 with $k = 0.5$ and $k = 2$. The Algorithms 2.3 to 2.8 were tested without restarts, and with restarts determined by the efficiency criterion described in the previons section.

Remarks: Looking at tables 4.1 to 4.4 we may observe the following:

1) None of the algorithms had difficulties in solving this problem. All of then used a few iterations and little CPU time for the whole process. Moreover, the overall performance did not depend on n.

2) No significant differences were detected between the versions "with restarts" and "without restarts" in terms of CPU times for algorithms DM, DS, RS and CS. In many cases, the behavior was exactly the same due to the fact that "normal" iterations were always more efficient than the first Newton iteration.

3) Schubert's method is clearly outperformed by Newton's method. This was expected, since a Schubert iteration is slightly more expensive than a Newton iteration, and the expected progress of a Newton iteration is greater than that of a Schubert iteration. A consequence of this fact is that, for the restarted version,

	n = 1000		n = 2000		n = 3000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts
N	(0,3,3,0,0.78)	-	(0,3,3,0,1.61)	-	(0,3,3,0,2.37)	-
MN	(0,10,1,9,1.16)	(0,4,2,2,0.74)	(0,10,1,9,2.19)	(0,4.2,2,1.42)	(0,10,1,9,3.23)	(0,4,2,2,2.06)
S	(0,4,1,3,1.17)	(0,4,2,2,1.15)	(0,4,1,3,2.47)	(0,4.2,2,2.38)	(0,4,1,3,3.48)	(0,4,2,2,3.41)
DM	(0,4,1,3,0.80)	(0,4,1,3,0.77)	(0,4,1,3,1.53)	(0,4,1,3,1.55)	(0,4,1,3,2.33)	(0,4,1,3,2.4)
DS	(0,4,1,3,0.63)	(0,4,1,3,0.60)	(0,4,1,3,1.23)	(0,4,1,3,1.29)	(0,4,1,3,1.80)	(0,4,1,3,1.83)
RS	(0,4,1,3,0.63)	(0,4,1,3,0.69)	(0,4,1,3,1.27)	(0,4,1,3,1.20)	(0,4,1,3,1.80)	(0,4,1,3,1.82)
CS	(1,6,1,5,0.89)	(0,5,3,2,1.04)	(1,6,1,5,1.73)	(0,5.3,2,2.0)	(1,6,1,5,2.48)	(0,5,3,2,3.04)

Table 4.1: Performance of the Methods for Problem 1 ($k = 0.5$)

	n = 4000		n = 5000		n = 6000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts
N	(0,3,3,0,3.09)	-	(0,3,3,0,3.94)	-	(0,3,3,0,4.63)	-
MN	(0,10,1,9,4.42)	(0,4,2,2,2.81)	(0,10,1,9,5.47)	(0,4,2,2,3.49)	(0,10,1,9,6.54)	(0,4,2,2,4.15)
S	(0,4,1,3,4.65)	(0,4,2,2,4.59)	(0,4,1,3,5.85)	(0,4,2,2,5.84)	(0,4,1,3,6.98)	(0,4,2,2,6.79)
DM	(0,4,1,3,3.16)	(0,4,1,3,3.07)	(0,4,1,3,3.83)	(0,4,1,3,3.96)	(0,4,1,3,4.68)	(0,4,1,3,4.58)
DS	(0,4,1,3,2.45)	(0,4,1,3,2.44)	(0,4,1,3,3.05)	(0,4,1,3,3.08)	(0,4,1,3,3.59)	(0,4,1,3,3.62)
RS	(0,4,1,3,2.38)	(0,4,1,3,2.40)	(0,4,1,3,2.99)	(0,4,1,3,3.13)	(0,4,1,3,3.66)	(0,4,1,3,3.69)
CS	(1.6,1,5,2.38)	(0,5,3,2,3.98)	(1,6,1,5,4.0)	(0,5,3,2,4.96)	(1,6,1,5,4.93)	(0,5,3,2,5.92)

Table 4.2: Performance of the Methods for Problem 1 ($k = 0.5$)

	n = 1000		n = 2000		n = 3000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts
N	(0,3,3,0,0.85)	-	(0,3,3,0,1.56)	-	(0,3,3,0,2.33)	-
MN	(1,9,1,8,1.07)	(1,9,1,8,1.09)	(1,9,1,8,2.06)	(1,9,1,8,2.01)	(1,9,1,8,2.95)	(1,9,1,8,2.98)
S	(1,5,1,4,1.42)	(0,4,2,2,1.15)	(1,5,1,4,2.84)	(0,4,2,2,2.25)	(1,5,1,4,4.17)	(0,4,2,2,3.49)
DM	(0,5,1,4,0.94)	(0,5,1,4,0.99)	(0,5,1,4,1.88)	(0,5,1,4,1.89)	(0,5,1,4,2.79)	(0,5,1,4,2.81)
DS	(1,5,1,4,0.75)	(1,5,1,4,0.76)	(1,5,1,4,1.49)	(1,5,1,4,1.45)	(1,5,1,4,2.21)	(1,5,1,4,2.22)
RS	(1,5,1,4,0.79)	(1,5,1,4,0.74)	(1,5,1,4,1.44)	(1,5,1,4,1.45)	(1,5,1,4,2.11)	(1,5,1,4,2.16)
CS	(1,5,1,4,0.74)	(1,5,1,4,0.76)	(1,5,1,4,1.43)	(1,5,1,4,1.48)	(1,5,1,4,2.17)	(1,5,1,4,2.12)

Table 4.3: Performance of the Methods for Problem 1 ($k = 2.0$)

	n = 4000		n = 5000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts
N	(0,3,3,0,3.08)	-	(0,3,3,0,4.3)	-
MN	(1,9,1,8,3.84)	(1,9,1,8,3.99)	(1,9,1,8,4.98)	(1,9,1,8,5.25)
S	(1,5,1,4,5.65)	(0,4,2,2,4.46)	(1,5,1,4,6.98)	(0,4,2,2,6.8)
DM	(0,5,1,4,3.75)	(0,5,1,4,3.85)	(0,5,1,4,4.67)	(0,5,1,4,5.08)
DS	(1,5,1,4,2.87)	(1,5,1,4,2.88)	(1,5,1,4,3.65)	(1,5,1,4,3.55)
RS	(1,5,1,4,2.82)	(1,5,1,4,2.94)	(1,5,1,4,3.57)	(1,5,1,4,3.60)
CS	(1,5,1,4,2.82)	(1,5,1,4,2.85)	(1,5,1,4,3.52)	(1,5,1,4,3.63)

Table 4.4: Performance of the Methods for Problem 1 ($k = 2.0$)

each Schubert iteration was always followed by a Newton iteration. In fact, we observe that $k_2 = k_3$ or $k_2 = k_3 + 1$ for all restarted runs of Schubert's method.

4) Surprisingly, Schubert's nonrestarted method used about the same number of iterations as the nonsuperlinear Algorithms 2.5 – 2.8.

5) In the mean, *DS* and *RS* were the most efficient methods for this problem. *CS* and *DM* also behaved quite well, both of them with similar *CPU* times as Newton's metho The Modified Newton method behaved clearly worse than *DM*, *DS*, *RS* and *CS*, a remarkable fact, since *MN*, *DS*, *RS* and *CS* have the same local conv gence properties, and the convergence results known for *DM* are even weaker.

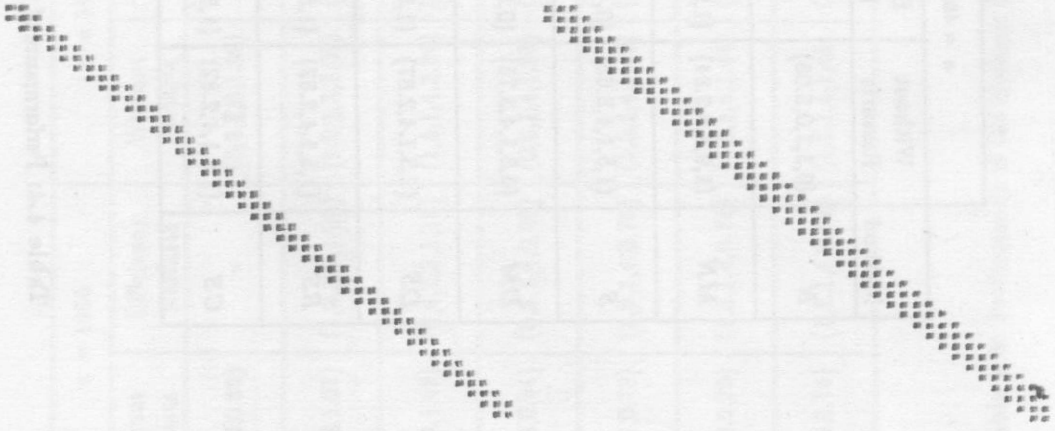

Figure 1. Structure of the Jacobian matrix and data structure for the L-U factorization.

Problem 2. (Band Broyden [2])

$$f_i(x) = (3 + 5x_i^2)x_i + 1 - \sum_{j \in I_i}(x_j + x_j^2) \quad i = 1, \ldots, n$$

where

$$I_i = \{i_1, \ldots, i_2\} - \{i\},$$

$$i_1 = \max\{1, i - 5\}, \quad i_2 = \min\{n, i + 5\}.$$

$$x^0 = (-1, \ldots -1)^T, \quad \varepsilon_1 = \varepsilon_2 = 10^{-4}, \quad TOL = 10^{-7}, \quad \Delta = 10, \quad BIG = 10^{10}.$$

Figure 2 shows the structure of the Jacobian matrix and the structure for the L-U factorization calculated by the George–Ng algorithm with $n = 40$.

Table 5 shows the computer CPU times used by the symbolic phase of the methods

n	time/seconds
1000	0.80
2000	1.62
3000	2.63
4000	3.56
5000	4.61

Table 5: CPU time of the Symbolic phase for Problem 2.

Table 6 shows the mean computer CPU time used by a Newton iteration

n	time/seconds
1000	0.97
2000	1.97
3000	2.99
4000	4.02
5000	5.53

Table 6: Mean CPU time of a Newton iteration for Problem 2.

Table 7 shows the mean computer CPU time of one ordinary iteration of Algorithms 2.3 – 2.8. We use the same notation as in Problem 1 for the algorithms.

n	MN	S	DM	DS	RS	CS
1000	0.22	1.04	0.39	0.25	0.25	0.23
2000	0.45	2.10	0.77	0.51	0.48	0.48
3000	0.67	2.93	1.19	0.74	0.73	0.72
4000	0.89	4.48	1.57	1.05	1.01	1.01
5000	1.11	5.88	1.94	1.28	1.23	1.21

Table 7: Mean CPU time of a typical iteration for Problem 2.

We observe that the CPU times of symbolic factorizations and of Newton and Schubert iterations are about four times the CPU times of those of the systems with the same dimension as in Problem 1. On the other hand, the CPU times of other Quasi–Newton iterations, and NM are about twice the times of the corresponding ones in Problem 1.

In tables 8.1 and 8.2 we show the numerical performance of Algorithms 2.2 to 2.8 for Problem 2. We use the same conventions as in Problem 1

Remarks: Most of the observations drawn for Problem 1 are also valid for this problem: This is also an easy problem for all the tested algorithms, the behavior with and without restarts coincide, and Schubert's method is not competitive with the rest of the algorithms. In addition:

1) Surprisingly, methods DS, RS and CS used less iterations than Schubert's method in the nonrestarted case. This behavior is not predictable by local convergence properties.

2) Algorithms DS, CS and RS are clearly the most efficient ones for this problem, followed by Newton's method the, Dennis–Marwil method and the Modified Newton method.

	n = 1000		n = 2000		n = 3000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficient Restarts	Without Restarts	Efficient Restarts
N	(0,4,4,0,3.96)	—	(0,4,4,0,7.93)	—	(0,4,4,0,12.0)	—
MN	(1,15,1,14,4.29)	(1,15,1,14,4.21)	(1,15,1,14,8.63)	(1.15,1,14,8.42)	(1,15,1,14,12.5)	(1,15,1,14,12.6)
S	(1,8,1,7,7.75)	(0,5,3,2,5.12)	(1,8,1,7,15.8)	(0,5,3,2,10.7)	(1,8,1,7,23.2)	(0,6,2,4,18.4)
DM	(1,10,1,9,4.64)	(0,8,2,6,4.38)	(1,10,1,9,9.01)	(0,8,2,6,8.66)	(1,10,1,9,13.4)	(0,8,2,6,13.1)
DS	(0,6,1,5,2.41)	(0,6,1,5,2.31)	(0,6,1,5,4.66)	(0,6,1,5,4.6)	(0,6,1,5,6.74)	(0,6,1,5,6.73)
RS	(0,6,1,5,2.20)	(0,6,1,5,2.29)	(0,6,1,5,4.51)	(0,6,1,5,4.5)	(0,6,1,5,6.61)	(0,6,1,5,6.66)
CS	(0,6,1,5,2.25)	(0,6,1,5,2.24)	(0,6,1,5,4.39)	(0,6,1,5,4.55)	(0,6,1,5,6.34)	(0,6,1,5,6.62)

Table 8.1: Performance of the Methods for Problem 2.

	n = 4000		n = 5000	
Method	Without Restarts	Efficiency Restarts	Without Restarts	Efficiency Restarts
N	(0,4,4,0,16.1)	—	(0,4,4,0,22.3)	—
MN	(1,15,1,14,16.9)	(1,15,1,14,16.8)	(1,15,1,14,21.0)	(1,15,1,14,21.2)
S	(1,8,1,7,35.5)	(0,5,3,2,23.2)	(1,8,7,1,44.1)	(0,5,3,2,29.7)
DM	(1,10,1,9,17.9)	(0,8,2,6,17.7)	(1,10,1,9,23.1)	(0,8,2,6,22.7)
DS	(0,6,1,5,9.35)	(0,6,1,5,9.53)	(0,6,1,5,11.8)	(0,6,1,5,12.0)
RS	(0,6,1,5,9.06)	(0,6,1,5,9.33)	(0,6,1,5,11.8)	(0,6,1,5,11.8)
CS	(0,6,1,5,9.19)	(0,6,1,5,9.29)	(0,6,1,5,11.6)	(0,6,1,5,11.8)

Table 8.2: Performance of the Methods for Problem 2.

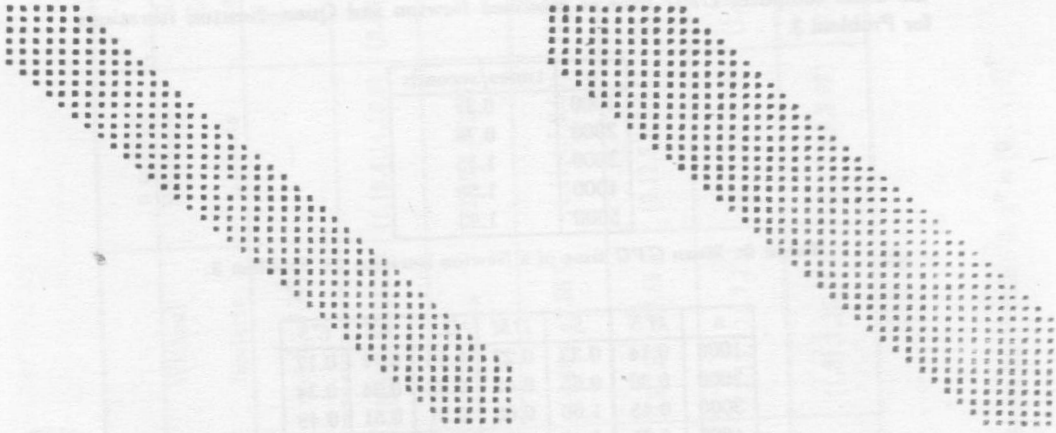

Figure 2: Structure of the Jacobian matrix and data structure for the L-U factorization.

Problem 3 (Trigexp [32])

$$f_1(x) = 3x_1^3 + 2x_2 - 5 + \sin(x_1 - x_2)\sin(x_1 + x_2)$$

$$f_i(x) = -x_{i-1}e^{(x_{i-1}-x_i)} + x_i(4 + 3x_i^2) + 2x_{i+1} +$$

$$+ \sin(x_i - x_{i+1})\sin(x_i + x_{i+1}) - 8 \qquad i = 2, \ldots, n-1.$$

$$f_n(x) = -x_{n-1}e^{(x_{n-1}-x_n)} + 4x_n - 3.$$

Initial points: $(0, \ldots, 0)^T$ and $(0.3, \ldots, 0.3)^T$.

Algorithmic parameters: $\epsilon_1 = \epsilon_2 = 10^{-4}$, $TOL = 10^{-7}$, $\Delta = 10$, $BIG = 10^{10}$.

The structure of the Jacobian matrix and, hence, the structure of the data structure for the L-U factorization are the same as those of Problem 1. See Figure

1. The CPU times of the symbolic phase also coincide, obviously with the CPU times of the symbolic phase of Problem 1.

Table 9 shows the mean computer time of a Newton iteration, and table 10 shows the mean computer CPU time of Modified Newton and Quasi–Newton iterations for Problem 3.

n	times/seconds
1000	0.37
2000	0.76
3000	1.15
4000	1.50
5000	1.93

Table 9: Mean CPU time of a Newton iteration for Problem 3.

n	MN	S	DM	DS	RS	CS
1000	0.14	0.33	0.22	0.16	0.17	0.17
2000	0.30	0.65	0.45	0.33	0.34	0.34
3000	0.45	1.00	0.66	0.51	0.51	0.49
4000	0.61	1.40	0.90	0.69	0.69	0.69
5000	0.75	1.72	1.10	0.86	0.86	0.84

Table 10. Mean CPU time of a typical iteration for Problem 3.

Observe that, unlike Problem 1, the CPU time of a Newton iteration is greater than the CPU time of a Schubert iteration due to the transcendental functions involved in the calculation of the Jacobian matrix. However, comparison between these times shows that L-U factorization still dominates the calculation. In Tables 11.1 and 11.2 we show the numerical performance of Algorithms 2.2 to 2.8 for Problem 1, using the same conventions as in the previous problems, for $x^0 = (0, \ldots 0)^T$. The same is done in Tables 12.1 and 12.2 for $x^0 = (0{,}3, \ldots, 0.3)^T$.

Remarks:

1) With the exception of Newton's method only CS is succesful in all trials, for the two initial points.

2) In both tests RS and DS both failed for the no restarts versions; S failed in the trial runs with no restarts and in those with a restart at every 5 iterations for the initial point $x = (0, \ldots 0)^T$ due to the ocorrence of overflow.

		$n = 1000$			$n = 3000$	
Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations
N	(0,8,8,0,3.24)	—	—	(0,8,8,0,9.26)	—	—
MN	(3,50,1,49,8.2)	(1,13,5,8,3.27)	(3,50, 9,41,9.59)	(3,50,1,49,25.2)	(1,18,4,14,10.9)	(3,50,1,49,29.6)
S	overflow	(0,10,5,5,3.71)	overflow	overflow	(0,10,5,5,10.9)	overflow
DM	(2,46,1,45,9.64)	(0,11,4,7,3.0)	(0,22,4,18,5.13)	(2,46,1,45,28.7)	(1,11,4,7,9.00)	(0,22,4,18,15.8)
DS	(3,100,1,99,16.1)	(0,12,3,9,2.68)	(0,26,5,21,5.34)	(3,100,1,99,49.3)	(0,12,3,9,8.04)	(0,26,5,21,16.3)
RS	(2,13,1,12,2.36)	(0,11,3,8,2.51)	(1,15,3,12,3.15)	(2,13,1,12,7.13)	(0,11,3,8,7.58)	(1,15,3,12,9.40)
CS	(1,31,1,30,5.33)	(0,12,3,9,2.68)	(1,17,3,14,3.39)	(1,31,1,30,15.9)	(0,12,3,9,8.02)	(1,17,3,14,10.4)

Table 11.1: Performance of the methods for Problem 3, $x^0 = (0, \ldots, 0)^T$

		$n = 5000$	
Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations
N	(0,8,8,0,15.0)	—	—
MN	(3,50,1,49,40.4)	(0,13,5,8,15.3)	(3,50,9,41,47.6)
S	overflow	(0,10,5,5,18.2)	overflow
DM	(2,46,1,45,48.6)	(1,11,4,7,15.2)	(0,22,4,18,26.3)
DS	(3,100,1,99,83.2)	(0,12,3,9,13.0)	(0,26,5,21,27.4)
RS	(2,13,1,12,12.0)	(0,11,3,8,12.5)	(1,15,3,12,16.0)
CS	(1,31,1,30,26.7)	(0,12,3,9,13.2)	(1,17,3,14,17.2)

Table 11.2: Performance of the methods for Problem 3, $x^0 = (0, \ldots, 0)^T$

		$n = 1000$			$n = 3000$	
Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations
N	(0,6,6,0,2.33)	—	—	(0,6,6,0,7.0)	—	—
MN	(3,50,1,49,8.66)	(1,10,3,7,2.21)	(0,50,9,41,10.1)	(3,50,1,49,23.5)	(1,10,3,7,6.66)	(0,50,9,41,30.0)
S	(0,11,1,10,4.10)	(0,7,4,3,2.71)	(0,9,2,7,3.37)	(0,11,1,10,12.1)	(0,7,4,3,7.97)	(0,9,2,7,9.77)
DM	(1,11,1,10,2.86)	(1,8,2,6,2.22)	(0,8,2,6,2.27)	(1,11,1,10,8.31)	(1,8,2,6,6.36)	(0,8,2,6,6.62)
DS	(1,51,1,50,9.04)	(0,9,3,6,2.32)	(0,9,2,7,2.01)	(1,51,1,50,25.4)	(0,9,3,6,6.32)	(0,9,2,7,5.78)
RS	(3,100,1,99,18.4)	(0,8,3,5,2.1)	(0,19,4,15,4.18)	(3,100,1,99,50.8)	(0,8,3,5,6.09)	(0,19,4,15,12.1)
CS	(1,12,1,11,2.39)	(0,9,3,6,2.30)	(0,9,2,7,2.05)	(1,12,1,11,6.67)	(0,9,3,6,6.60)	(0,9,2,7,6.0)

Table 12.1: Performance of the methods for Problem 3, $x^0 = (0.3, \ldots, 0.3)^T$

		$n = 5000$	
Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations
N	(0,6,6,0,11.7)	—	—
MN	(3,50,1,49,40.2)	(1,10,3,7,11.0)	(0,50,9,41,48.1)
S	(0,11,1,10,19.8)	(0,7,4,3,13.3)	(0,9,2,7,16.9)
DM	(1,11,1,10,13.6)	(1,8,2,6,10.9)	(0,8,2,6,11.2)
DS	(1,51,1,50,42.0)	(0,9,3,6,11.0)	(0,9,2,7,9.68)
RS	(3,100,1,99,84.9)	(0,8,3,5,9.86)	(0,19,4,15,20.1)
CS	(1,12,1,11,11.0)	(0,9,3,6,10.9)	(0,9,2,7,9.65)

Table 12.2: Performance of the methods for Problem 3, $x^0 = (0.3, \ldots, 0.3)^T$

Problem 4 (Poisson Problem [29])

This problem is the nonlinear system of equation arising from finite difference discretization of the Poisson boundary value problem

$$\Delta u = \frac{u^3}{1+s^2+t^2}, \quad 0 \le s \le 1, \quad 0 \le t \le 1$$

$$u(0,t) = 1,$$

$$u(1,t) = 2 - e^t, \quad t \in [0,1]$$

$$u(s,0) = 1,$$

$$u(s,1) = 2 - e^s, \quad s \in [0,1]$$

We use an L^2 grid with $L = 15$ and $L = 31$. Therefore $n = 225$ and $n = 961$ respectively.

We ran the algorithms with $x^0 = (-1,\ldots,-1)^T$, $\varepsilon_1 = \varepsilon_2 = 10^{-4}$, BIG $= 10^{10}$, $\Delta = 5$.

The structure of the Jacobian matrix and the data structure for the L-U factorization (for $n = 36$) is given in Figure 3. The CPU times of the symbolic phase were the following:

$n = 225$, time $= 0.83$ sec.
$n = 961$, time $= 7.70$ sec.

The mean computer times of Newton iterations are the following:

$n = 225$, time $= 1.19$ sec.
$n = 961$, time $= 20.8$ sec.

Table 13 shows the mean computer CPU time of one ordinary iteration of Algorithms 2.3 - 2.8.

L	n	MN	S	DM	DS	RS	CS
15	225	0.10	1.27	0.17	0.13	0.10	0.11
31	961	0.67	23.0	1.23	0.74	0.53	0.88

Table 13: Mean CPU time of a typical iteration for Problem 4.

As expected for the band structure of this problem, the computer time of L-U factorizations (Newton and Schubert) is proportional to L^2n. On the other hand, the

computer time of other iterations is proportional to Ln.

In Tables 14.1 and 14.2 we show the numerical performance of Algorithms 2.2 to 2.8 for Problem 4:

Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iterations
N	(1,5,5,0,6.04)	—	—
MN	(4,1393,1,1392,123.)	(1,17,4,13,6.1)	(1,57,10,47,16.0)
S	(0,9,1,8,11.0)	(0,9,2,7,10.8)	(0,8,2,6,9.64)
DM	(4,844,1,843,122.)	(0,15,5,10,7.68)	(0,27,5,22,9.31)
DS	(2,853,1,852,79.3)	(4,184,92,92,121.)	(4,433,73,360,122.)
RS	(1,329,1,328,86.1)	(0,19,10,9,12.8)	(4,441,74,367,121.)
CS	(1,861,1,860,80.0)	(0,19,10,9,12.8)	(4,437,73,364,121.)

Table 14.1: Performance of the methods for Problem 4.
($L = 15,\ n = 225$)

Method	Without Restarts	Efficiency Restarts	Restart Every 5 Iteration
N	(1,5,5,0,104.)	—	—
MN	(4,412,1,411,301.)	(1,18,4,14,92.6)	(4,73,13,60,309.)
S	(1,7,1,6,150.)	(1,7,1,6,151.)	(1,7,2,5,157.)
DM	(4,266,1,265,301.)	(1,13,6,7,136.)	(1,26,5,21,133.)
DS	(4,364,1,363,302.)	(4,27,14,13,314.)	(4,67,12,55,302.)
RS	(4,380,1,379,301.)	(4,28,14,14,306.)	(4,71,12,59,300.)
CS	(4,372,1,371,302.)	(1,20,10,10,217.)	(4,73,13,60,318.)

Table 14.2: Performance of the methods for Problem 4.
($L = 31,\ n = 961$)

Remarks:

1) Only Newton's method and Schubert's method seem to be reliable for this problem. DM, DS, RS, CS and MN failed in all cases without restart, and DS failed in many cases even with restarts.

2) DM behaved, in general, better than DS, RS and CS, but its performance is nowhere comparable to the performance of Newton's method.

3) Even converging, Schubert's method used considerably more iterations than Newton's method and, therefore, it wasted much more CPU time.

Figure 3: Structure of the Jacobian matrix and data structure for the L-U factorization.

Problem 5:

$$f_1(x) = -2x_1^2 + 3x_1 - 2x_2 + 0.5x_{\alpha_1} + 1.0$$

$$f_j(x) = -2x_j^2 + 3x_j - x_{j-1} - 2x_{j+1} + 0.5x_{\alpha_j} + 1.0 \quad j = 2, \ldots, n-1$$

$$f_n(x) = -2x_n^2 + 3x_n - x_{n-1} + 0.5x_{\alpha_n} + 1.0$$

for α_j, $j = 1, 2, \ldots, n$, randomly chosen in the intervals: $\alpha_j \in \{\alpha_{j\min}, \alpha_{j\max}\}$, where $\alpha_{j\min} = \max\{1, j-b\}$ and $\alpha_{j\max} = \min\{n, j+b\}$ for a parameter b which defines the bandwidth.

We used $x^0 = (-1, \ldots, -1)^T$ as a initial point and, as in previous tests,

$\varepsilon_1 = \varepsilon_2 = 10^{-4}$, $TOL = 10^{-7}$, $\Delta = 10$-, $BIG = 10^{10}$. The structure of some typical Jacobian matrices with $n = 40$, for different bandwidths and their correspondent data structures for L-U factorizations are given in Figures 4 and 5.

Table 15 shows the CPU time of the symbolic phase for some tests with $n = 1000$ and $b = 15$, 30, 50 and 100

b	Time
15	1.4
30	2.57
50	4.00
100	8.35

Table 15: CPU time of the Symbolic phase for Problem 5.

Table 16 show the mean computer CPU time of a Newton iteration for the same problem

b	time/seconds
15	1.01
30	2.75
50	6.19
100	22.7

Table 16: Mean CPU time of a Newton iteration for Problem 5.

Table 17 shows the mean computer CPU time of one ordinary iteration of Algorithms 2.3 – 2.8

Method	$b = 15$	$b = 30$	$b = 50$	$b = 100$
MN	0.18	0.26	0.39	0.63
S	1.1	2.74	6.26	24.0
DM	0.34	0.51	0.71	1.33
DS	0.22	0.32	0.44	0.70
RS	0.19	0.30	0.40	0.67
CS	0.19	0.29	0.40	0.67

Table 17: Mean CPU time of a typical iteration for Problem 5.

Method	$b = 15$			$b = 30$		
	Without Restarts	Efficiency Restarts	Restart Every 4 Iterations	Without Restarts	Efficiency Restarts	Restart Every 4 Iterations
N	(0,4,4,0,4.13)	—	—	(0,4,4,0,11.1)	—	—
MN	(1,10.1,9,2.84)	(1,10,1,9.2.81)	(0,6,2,4,2.75)	(1,10,1,9,5.20)	(1,10,1,9.5.2)	(0,6.2,4,7.27)
S	(1,6,1,5,6.44)	(0,4,2,2.4.48)	(0,6,2,4,6.52)	(1,6,1,5,16.0)	(0,4,2,2,11.4)	(0,6,2,4,16.5)
DM	(0,7,1,6,3.21)	(0,7,1,6,3.17)	(0,6,2,4,3.49)	(0,7,1,6,5.72)	(0,7,1,6,5.93)	(0,6,2,4,7.27)
DS	(1,6,1,5,2.16)	(1,6,1,5,2.22)	(0,6,2,4,3.03)	(0,6,1,5,4.18)	(0,6,1,5,4.27)	(0,6,2,4,6.78)
RS	(1,6,1,5,2.10)	(0,6,2,4,2.90)	(0,6,2,4,2.93)	(0,6,1,5,4.14)	(0,6,1,5,4.19)	(0,6,2,4,6.41)
CS	(1,6,1.5,2.79)	(0,6,2,4,2.94)	(0,6,2,4,2.94)	(1,6.1,5,4.04)	(1,6,1,5,4.16)	(0,6,2,4,6.51)

Table 18.1: Performance of the methods for Problem 5.

Method	b = 50			b = 100		
	Without Restarts	Efficiency Restarts	Restart Every 4 Iterations	Without Restarts	Efficiency Restarts	Restart Every 4 Iterations
N	(0,4,4,0,24.9)	—	—	(0,4,4,0,91.0)	—	—
MN	(1,10,1,9,9.60)	(1,10,1,9,9.88)	(0,6,2,4,14.2)	(1,10,1,9,28.01)	(1,10,1,9,29.1)	(0,6,2,4,46.0)
S	(0,6,1,5,36.7)	(0,4,2,2,25.3)	(0,6,2,4,37.5)	(0,6,1,5,134.)	(0,4,2,2,94.6)	(0,6,2,4,134.)
DM	(0,7,1,6,10.23)	(0,7,1,6,10.5)	(0,6,2,4,15.2)	(0,7,1,6,28.9)	(0,7,1,6,31.2)	(0,6,2,4,47.9)
DS	(0,6,1,5,8.24)	(0,6,1,5,8.49)	(0,6,2,4,14.1)	(1,6,1,5,25.4)	(1,6,1,5,26.4)	(0,6,2,4,45.7)
RS	(0,6,1,5,8.10)	(0,6,1,5,8.32)	(0,6,2,4,14.3)	(0,6,1,5,24.8)	(0,6,1,5,26.0)	(0,6,2,4,45.3)
CS	(1,6,1,5,8.02)	(1,6,1,5,8.25)	(0,6,2,4,13.4)	(1,6,1,5,24.9)	(1,6,1,5,26.3)	(0,6,2,4,45.4)

Table 18.2: Performance of the methods for Problem 5.

Observe that the CPU time for L-U factorization behaves in a way which may be described as "worse than linear but better that quadratic". This was expected, due to the random localization of the nontridiagonal entries of the Jacobian matrices.

In Tables 18.1 and 18.2 we show the numerical performance of Algorithms 2.2 to 2.8 for Problem 5.

Remarks:

1) As so happens with Problem 1, this was an easy problem for all the tested methods. Most of the conclusions drawn for that problem are also valid here.

2) Clearly, methods DS, RS and CS are the most efficient ones, followed closely by MN. DM also behaves very well. In spite of the small number of iterations it uses to converge, Newton's method is rather costly (especially for large values of b) due to systems resolution expense.

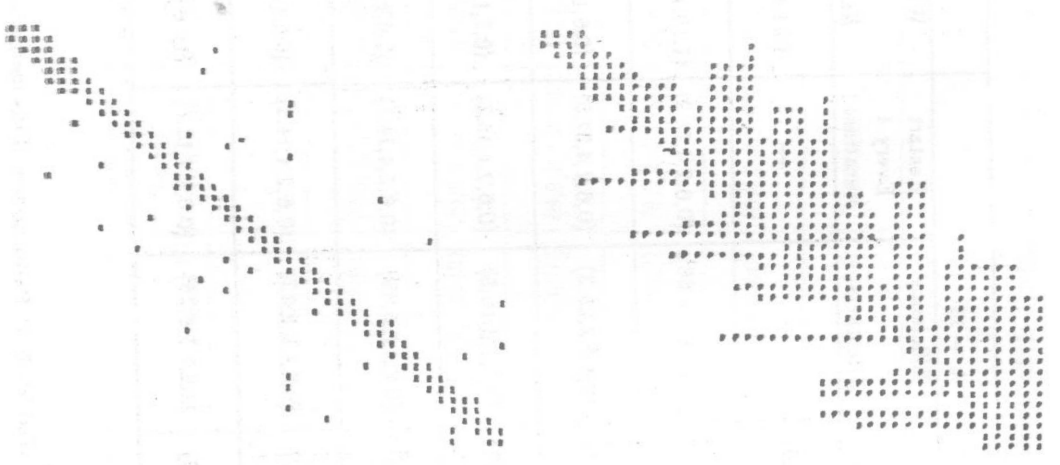

Figure 4: Structure of the Jacobian matriz and data structure for the L-U factorization ($b = 15$)

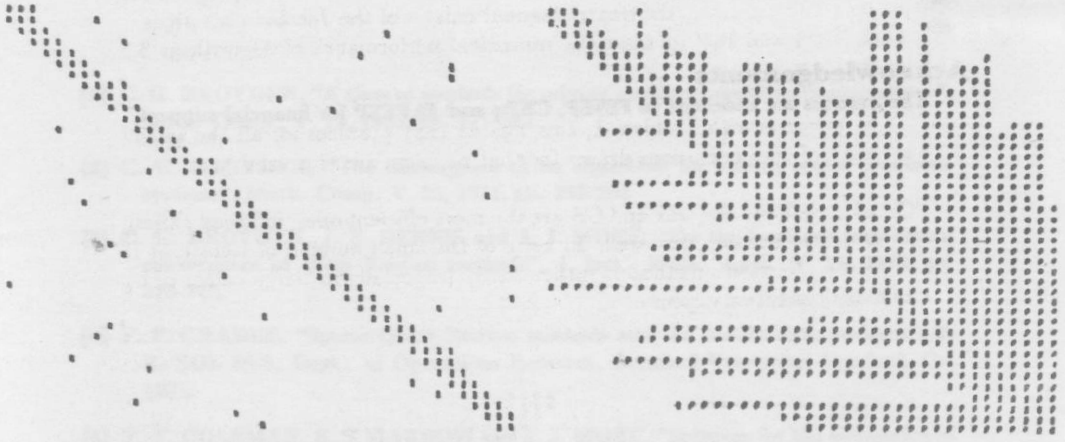

Figure 5: Structure of the Jacobian matriz and data structure for the L-U factorization ($b = 30$)

5. Conclusions

One of the aims of this paper was to verify if some Quasi–Newton methods with direct updates of matrix factorizations [7, 21, 22] are reliable alternatives to Newton's method in situations where derivatives are available. We conclude that, in the cases where they converge, they tend to be more efficient than Newton's method, especially if direct resolution of the linear systems is costly. Unhappily, they are not as robust as Newton's method, or even as Schubert's method. Therefore, development of superlinear and practical Quasi–Newton methods, possibly under the lines of [23], seems to be an interesting challenge.

However, it is remarkable that methods like DS, CS or RS, when they converge, use a very small number of iterations, relative to MN, even without restarts, a fact which is not explained by presently available theory.

As expected, Schubert's method is not competitive with Newton's method, for this type of problems. Its applicability is clearly restricted to situations where

derivatives are not available, or are very difficult to calculate.

All the computer codes related to this paper are available under request to the authors.

Acknowledgements

The authors are indebted to FINEP, CNPq and FAPESP for financial support.

References

[1] C. G. BROYDEN, "A class of methods for solving nonlinear simultaneous equations", Math. Comp., V. 19, 1965, pp. 577-593.

[2] C. G. BROYDEN, "The convergence of an algorithm for solving sparse nonlinear systems", Math. Comp. V. 25, 1971, pp. 285-294.

[3] C. G. BROYDEN, J. E. DENNIS and J. J. MORÉ, "On the local and superlinear convergence of quasi–Newton methods", J. Inst. Math. Appl. V. 12, 1973, pp. 223-245.

[4] F. F. CHADEE, "Sparse Quasi–Newton methods and the continuation problem", T. R. SOL 85-8, Dept. of Operations Research, Stanford University, Stanford, Ca, 1985.

[5] T. F. COLEMAN, B. S. GARBOW and J. J. MORÉ, "Software for the estimation of sparse Jacobian matrices", ACM Trans. Math. Software, V. 10, 1984, pp. 329-345.

[6] J. E. DENNIS, N. ECHEBEST, M. T. GUARDARUCCI, J. M. MARTÍNEZ, H. D. SCOLNIK and C. VACCINO, "A curvilinear search using tridiagonal secant updates for unconstrained optimization", Contributed paper to IV Latin–Iberian–American Congress on Operations Research and Systems Engineering, Rio de Janeiro, october 1988.

[7] J. E. DENNIS and E. S. MARWIL, "Direct secant updates of matrix factorizations", Math. Comput., V. 38, 1982, pp. 459-476.

[8] J. E. DENNIS and J. J. MORÉ, "A characterization of superlinear convergence and its application to quasi–Newton methods", Math. Comp., V. 28, 1974, pp. 549-560.

[9] J. E. DENNIS Jr. and J. MORÉ, "Quasi–Newton Methods, motivation and theory", SIAM Review, V. 19, 1977, pp. 46-89.

[10] J. E. DENNIS and R. SCHNABEL, Numerical methods for unconstrained optimization and nonlinear equations", Prentice–Hall Series in Comput. Math., Prentice–Hall, NJ, 1983.

[11] I. S. DUFF, "*MA28* – A set of FORTRAN subroutines for sparse unsymmetric linear equations", Tech. Report AERE R–8730, Harwell, England, 1977.

[12] I. S. DUFF, A. M. ERISMAN and J. K. REID, "Direct Methods for sparse matrices", Clarendon Press, Oxford, 1986.

[13] G. E. FORSYTHE and C. B. MOLER, "Computer solution of linear algebraic systems", Prentice–Hall, Englewood Cliffs, NJ, 1967.

[14] A. GEORGE and E. Ng, "Symbolic factorization for sparse Gaussian elimination with partial pivoting", SIAM Journal on Scientific and Statistical Computing, V. 8, 1987, pp. 877-898.

[15] G. H. GOLUB and CH. F. VAN LOAN, "Matrix Computations", The Johns Hopkins University Press, Baltimore, 1983.

[16] G. W. JOHNSON and N. H. AUSTRIA, "A quasi–Newton method employing direct secant updates of matrix factorizations", SIAM J. Numer. Anal. V. 20, 1983, pp. 315-325.

[17] V. R. LOPES, "Soluções por elementos finitos de equações de difusão lineares via princípios externos duais", PhD Thesis, Dept. of Mathematics, USP S.Carlos, 1988.

[18] V. R. LOPES, "Private communication", 1988.

[19] J. M. MARTÍNEZ, "Generalization of the methods of Brent and Brown for solving nonlinear simultaneous equations", SIAM Journal on Numerical Analysis, V. 16, 1979, pp. 434-448.

[20] J. M. MARTÍNEZ, "Solving nonlinear simultaneous equations with a generalization of Brent's method", BIT, V. 30 1980, pp. 175-186.

[21] J. M. MARTÍNEZ, "A Quasi-Newton method with a new updating for the *LDU* factorization of the approximate Jacobian", Mat., Aplic. e Comput., V. 2, 1983, pp.131-142.

[22] J. M. MARTÍNEZ "Quasi–Newton methods with factorization scaling for solving sparse nonlinear systems of equations", Computing, V. 38, 1987, pp. 131-141.

[23] J. M. MARTÍNEZ, "A family of Quasi–Newton methods with direct secant updates of matrix factorizations", SIAM J. on Numerical Analysis, to appear, 1988.

[24] E. S. MARWIL, "Convergence results for Schubert's method for solving sparse nonlinear equations", SIAM J. Numer.Anal. V. 16, 1979, pp. 588-604.

[25] J. J. MORÉ and M. Y. COSNARD, "On the numerical solution of nonlinear equations", ACM Transactions on Mathematical Software, V. 5, 1979, pp. 64-85.

[26] J. J. MORÉ and J. A. TRANGENSTEIN, "On the global convergence of Broyden's method", Math. Comp., V. 30, 1976, pp. 523-540.

[27] J. M. ORTEGA and W. C. RHEINBOLDT, "Iterative solution of nonlinear equations in several variables". New York, Academic Press 1970.

[28] L. K. SCHUBERT, "Modification of a quasi–Newton method for nonlinear equations with a sparse Jacobian", Math. Compt., V. 24, 1970, pp. 27-30.

[29] H. SCHWANDT, "An interval arithmetic approach for the construction of an almost grobally convergent method for the solution of the nonlinear Poisson equation on the unit square", SIAM J. Sci. Stat. Comput. V. 5, 1984, pp. 427-452.

[30] H. SCHWETLICK, "Numerische Lösung nichtlinearer Gleichungen. Berlin: Deutscher Verlag der Wissenschaften 1978.

[31] V. E. SHAMANSKII, "A modification of Newton's method", Ukrain Mat. Z., V. 19, 1967, pp. 133-138.

[32] Ph. L. TOINT, "Numerical solution of large sets of algebraic nonlinear equations", Mathematics of Computation, V. 16, 1986, pp. 175-189.

[33] J. V. ZAGO, "Approximate solution of generalized Hamiltonian equations with applications", PhD Thesis, Dept. of Mathematics, University of Wisconsin at Madison, Madison, WI, 1976.

RELATÓRIOS TÉCNICOS — 1989

01/89 — **Uniform Approximation of Continuous Functions With Values in** [0, 1] — *João B. Prolla.*

02/89 — **On Some Nonlinear Iterative Relaxation Methods in Remote Sensing** — *A. R. De Pierro.*

03/89 — **A Parallel Iterative Method for Convex Programming with Quadratic Objective** — *Alfredo N. Iusem* and *Alvaro R. De Pierro.*

04/89 — **Fifth Force, Sixth Force, and all that: a Theoretical (Classical) Comment** — *Erasmo Recami* and *Vilson Tonin-Zanchin.*

05/89 — **An Application of Singer's Theorem to Homogeneous Polynomials** — *Raymundo Alencar.*

06/89 — **Summhammer's Experimental Test of the Non-Ergodic Interpretation of Quantum Mechanics** — *Vincent Buonomano.*

07/89 — **Privileged Reference Frames in General Relativity** — *Waldyr A. Rodrigues Jr.* and *Mirian E. F. Scanavini.*

08/89 — **On the Numerical Solution of Bound Constrained Optimization Problems** — *Ana Friedlander* and *José Mario Martínez.*

09/89 — **Dual Extremun Principles for the Heat Equation Solved by Finite Element Methods I** — *Vera Lucia da Rocha Lopes* and *José Vitório Zago.*

10/89 — **Local Convergence Theory of Inexact Newton Methods Based on Structured Least Chance Updates** — *José Mario Martínez*

11/89 — **Real Spin-Clifford Bundle and the Spinor Structure of Space-Time** — *Waldyr A. Rodrigues Jr.* and *Vera L. Figueiredo.*

12/89 — **A Multiplier Theorem on Weighted Orlicz Spaces** — *B. Bordin* and *J. B. Garcia.*

13/89 — **Dual Extremum Principles For The Heat Equation Solved By Finite Element Methods II** — *Vera Lucia da Rocha Lopes* and *José Vitório Zago.*

14/89 — **Dirac and Maxwell Equations in the Clifford and Spin-Clifford Bundels** — *W. A. Rodrigues Jr.* and *E. Capelas de Oliveira.*

15/89 — **Formal Structures, The Concepts of Covariance Invariance, Equivalent Reference Frames, and the Principle of Relativity** — *W. A. Rodrigues Jr., M. E. F. Scanavini* and *L. P. de Alcantara.*

16/89 — **Local Minimizers of a Quadratic Function With a Spherical Constraint** — *José Mario Martínez.*

17/89 — **On Pseudo-Convex Polycircular Domains In Banach Spaces** — *Mário C. Matos.*

18/89 — **On Circular and Special Units of an Abelian Number Field** — *Trajano Nóbrega.*