Journal of Numerical Mathematics and Stochastics, **2** (1) : 54-63, 2010 © ***JNM@S***
http://www.jnmas.org/jnmas2-5.pdf Euclidean Press, LLC
Online: ISSN 2151-2302

Jacobian Computation-free Newton's Method for Systems of Nonlinear Equations

M. Y. WAZIRI , W. J. LEONG, M. A. HASSAN, and M. MONSI

Department of Mathematics, Faculty of Science, Universiti Putra, Malaysia 43400 Serdang, Malaysia.
E-mail: waziri@math.upm.edu.my

Abstract. *We propose a modification to Newton's method for solving nonlinear equations, namely a Jacobian Computation-free Newton's Method . Unlike the classical Newton's method, the proposed modification neither requires to compute and store the Jacobian matrix, nor to solve a system of linear equations in each iteration. This is made possible by approximating the Jacobian inverse to a diagonal matrix without computing the Jacobian. The proposed method turns out to be significantly cheaper than Newton's method, much faster than fixed Newton and is suitable for small, medium or large scale nonlinear equations with a dense or sparse Jacobian. After proving the convergence of the reported algorithm, numerical experiments are reported to illustrate the promise of this method*.

Key words : Nonlinear Equations, Large Scale Systems, Newton's Method, Diagonal Matrix Updating, Jacobian Approximation.

AMS Subject Classifications : 65H11, 65K05

1. Introduction

Consider the problem of finding a solution to a nonlinear system

$$F(x) = 0, \tag{1}$$

with the mapping $F : R^n \to R^n$ assumed to satisfy the following assumptions:
1. F is continuously differentiable in an open convex set Ω.
2. There exist a solution vector x^* of (1) in Ω such that $F(x^*) = 0$ and $F'(x^*) \neq 0$.
3. The Jacobian $F'(x)$ is Lipschitz continuous at x^*.

The standard method for finding the solution to (1) is due to Newton. The method generates an iterative sequence $\{x_k\}$ from a given initial guess vector x_0 in the neighborhood of x^*, according to the following procedure.

Algorithm CN (Newton)
For $k = 0,1,2,\ldots$ of $F'(x_k)$, the Jacobian matrix of F,
Step 1: solve $F'(x_k)s_k = -F(x_k)$,
Step 2: Update $x_{k+1} = x_k + s_k$,
where s_k is the Newton correction and the equation of step 2 is the Newton system. When the Jacobian matrix $F'(x^*)$ is nonsingular at a solution of (1) the convergence is guaranteed with a quadratic rate from any initial point x_0 in the neighborhood of x^* [4,10], i.e.,

$$\|x_{k+1} - x^*\| \leq h\|x_k - x^*\|^2 , \tag{2}$$

for some h. However, an iteration of the CN algorithm turns out to be expensive, because it requires to compute and store the Jacobian matrix, as well as solving a Newton's system in each iteration. Despite its simplicity and general reliability, Newton's method has some major widely known shortcomings [3]. Several strategies have been developed to overcome these shortcomings . The simplest strategy is incorporated in the fixed Newton method, which lets $F'(x_k) \equiv F'(x_0)$ for $k > 0$. It generates an iterative sequence $\{x_k\}$ from a given initial guess x_0 according to the algorithm that follows.

Algorithm FN (Fixed Newton)
Step 1: Solve $F'(x_0)s_k = -F(x_k)$,
Step 2: Set $x_{k+1} = x_k + s_k , \forall\ k = 0,1,2\ldots.$
This method avoids both the computation of the Jacobian (except for the first iteration), as well as solving the system of n linear equations in each iteration but is significantly slower [8] than the CN algorithm.The second strategy encampasses the inexact Newton method, which finds the approximate solution of the Newton system by some iterations, see, e.g. [3],viz.

Algorithm IN (Inexact Newton)
Let x_0 be given.
Step 1: Find some s_k which satisfies
$F'(x_k)s_k = -F(x_k) + r_k$,
where $\|r_k\| \leq \eta_k\|F(x_k)\|$, and η_k is explained in [3].
Step 2: Set $x_{k+1} = x_k + s_k$.

We may also list here the famous Quasi-Newton's method that replaces the Jacobian or its inverse with an approximation which can be updated at each iteration [2], viz.

Algorithm QN (Quasi-Newton)
Step 1: solve $B_k s_k = -F(x_k)$,
Step 2: Update $x_{k+1} = x_k + s_k$,
B_k being an approximation to the Jacobian.

The rationale behind the quasi-Newton method is to reduce the evaluation cost of the Jacobian matrix, especially when the function evaluations are very expensive. Despite their increased relative storage requirements, the solution cost by quasi-Newton methods could be much lower than with inexact Newton methods [5]. Many efforts have recently been made by a number of authors [5, 8, 7, 9] to overcome the shortcomings of various Newton methods. The most critical idea common to all these efforts is forming and storing a full-matrix approximation to the Jacobian (directly or indirectly), which can be a very expensive task for large scale problems. In fact the limitations/shortcomings of Newton methods on large scale

problems are what motivate this work.

In this paper, we design an alternative approximation to the Jacobian inverse by a diagonal matrix. This allows us to bypass the need to compute or store the Jacobian as well as to solve the pertaining Newton's system of n linear equations in each iteration. The anticipation has been to reduce the computational costs, storage requirements and processing time (CPU time).The rest of this paper is organized as follows. In the next section , we present the proposed method. Convergence results are presented in section 3. Some numerical results are reported in Section 4, and section 5 contains some conclusions. The quite reliable numerical results of section 4 illustrate not only the promise of this method but also that it is cheaper than the CN algorithm and faster than the fixed Newton method.

2. The Jacobian Computation-free Newton's Method

Consider Taylor′s expansion of $F(x)$ about x_k

$$F(x) = F(x_k) + F'(x_k)(x - x_k) + \bigcirc(\|x - x_k\|^2). \tag{3}$$

A first-order incomplete version of this expansion is

$$\hat{F}(x) = F(x_k) + F'(x_k)(x - x_k), \tag{4}$$

where $F'(x_k)$ is the Jacobian of $F(x)$ at x_k .

In order to incorporate more information on the Jacobian in the updating matrix, we impose on (4) the condition

$$\hat{F}(x_{k+1}) = F(x_{k+1}), \tag{5}$$

where $\hat{F}(x_{k+1})$ is an approximation to F evaluated at x_{k+1}. The reader is referred to [1] for full details. Consequently relation (4) becomes

$$F(x_{k+1}) \approx F(x_k) + F'(x_k)(x_{k+1} - x_k), \tag{6}$$

and

$$F'(x_k)(x_{k+1} - x_k) \approx F(x_{k+1}) - F(x_k). \tag{7}$$

Multiplication of (7) by $F'(x_k)^{-1}$ leads to

$$(x_{k+1} - x_k) \approx F'(x_k)^{-1}(F(x_{k+1}) - F(x_k)). \tag{8}$$

Let us consider an approximation of the Jacobian inverse $F'(x_k)^{-1}$ by a certain diagonal matrix D_k,

$$F'(x_k)^{-1} \approx D_k, \tag{9}$$

for which $D = diag(d^1, d^2, \ldots, d^n)$, to be updated at each iteration. This would transform (8) to

$$x_{k+1} - x_k \approx D_k(F(x_{k+1}) - F(x_k)). \tag{10}$$

Since we require D_k to be a diagonal matrix, then satisfaction of (9) in (10) allows for

$$d_{k+1}^{(i)} = \frac{x_{k+1}^{(i)} - x_k^{(i)}}{F_i(x_{k+1}) - F_i(x_k)}. \tag{11}$$

Hence,

$$D_{k+1} = diag(d_{k+1}^{(i)}), \tag{12}$$

$\forall\ i = 1,2,\ldots,n$ and $\forall\ k = 0,1,2,\ldots,n$, where $F_i(x_{k+1})$ is the i^{th} component of the vector $F(x_{k+1})$, $F_i(x_k)$ is the i^{th} component of the vector $F(x_k)$, $x_{k+1}^{(i)}$ is the i^{th} component of the vector x_{k+1}, $x_k^{(i)}$ is the i^{th} component of the vector x_k and $d_{k+1}^{(i)}$ is the i^{th} diagonal element of D_{k+1}.

To safeguard against the possibility of small $F_i(x_{k+1}) - F_i(x_k)$ we use (12) only when $|F_i(x_{k+1}) - F_i(x_k)| > 10^{-8}$ or otherwise we set $d_k^{(i)} = d_{k-1}^{(i)}$. The update for this Jacobian computation-free Newton's method (JCFN) is

$$x_{k+1} = x_k - D_k F(x_k),\ \ k = 1,2,\ldots, \tag{13}$$

This paves the way towards the following first result of this work.

Algorithm JCFN (Jacobian computation-free Newton)
Consider $F(x) : \Re^n \to \Re^n$ with the same properties as in (1).
Step 1 : Given x_0 and $D_0 = I_n$, set $k = 0$,
Step 2 : Compute $F(x_k)$,
Step 3 : Compute $x_{k+1} = x_k - D_k F(x_k)$ where D_k is defined by (12), when $|F_i(x_{k+1}) - F_i(x_k)| > 10^{-8}$. Otherwise set $d_k^{(i)} = d_{k-1}^{(i)}$,$\forall\ k = 1,2,\ldots,$
Step 4 : If $\|x_{k+1} - x_k\| + \|F(x_k)\| \leq 10^{-8}$ stop. Otherwise set $k = k + 1$ and go back to step 2.

3. Convergence Analysis

Here we report on a result for $F : (f_1, f_2, \ldots, f_n)$ of a lemma that follows, which provides a condition under which the JCFN algorithm is linearly convergent to x^*.

Lemma 3.1. *Let $F(x) : \Re^n \to \Re^n$ be continuously differentiable in an open convex set $\Omega \in R^n$. If D_k* defined by (12) *and $D_0 = I_n$, then $\{D_k\}$ is bounded for each $k > 0$..*

Proof. Since $D_0 = I_n$ then $\|D_0\|_F \leq \sqrt{n}$, and $D_{k+1} = diag(d_{k+1}^{(i)})$ for $i = 1,2,\ldots,n$. By letting $F_{(i)}(x_{k+1}) - F_{(i)}(x_k) = \Delta F_k^{(i)}$ and $x_{k+1}^{(i)} - x_k^{(i)} = \Delta x_k^{(i)}$ we have

$$D_{k+1} = diag(\frac{\Delta x_k^{(i)}}{\Delta F_k^{(i)}}). \tag{14}$$

Continue by induction, viz. $k = 0$ generates

$$D_1 = diag(\frac{\Delta x_0^{(i)}}{\Delta F_0^{(i)}}), \tag{15}$$

whose norm is

$$\|D_1\| \leq \sqrt{\sum_{i=1}^{n} (\frac{\Delta x_0^{(i)}}{\Delta F_0^{(i)}})^2} = \beta_0, \tag{16}$$

$k = 1$ generates

$$D_2 = diag(\frac{\Delta x_1^{(i)}}{\Delta F_1^{(i)}}), \tag{17}$$

with

$$\|D_2\| \leq \sqrt{\sum_{i=1}^{n} \left(\frac{\Delta x_1^{(i)}}{\Delta F_1^{(i)}}\right)^2} = \beta_1, \tag{18}$$

and so on for $k = 2,3,\ldots\ldots$ etc. We then let $max[\sqrt{n}, \beta_0, \beta_1, \ldots \beta_n] = \alpha$ to realize that $\forall\ k$, $\|D_k\| \leq max[\sqrt{n}, \beta_0, \beta_1, \ldots \beta_n]$, i.e.

$$\|D_k\| \leq \alpha. \tag{19}$$

Here the proof ends. ■

Theorem 3.1. *Let Ω be an open convex subset of R^n and $F(x) : \Omega \to \Re^n$ be a continuously differentiable mapping . Assume that $\exists$ (i) $x^* \in \Omega$, with $F(x^*) = 0$ and $F'(x^*) \neq 0$ and (ii) constants $\gamma_1 \leq \gamma_2$ such that $\gamma_1\|\xi\|^2 \leq \xi^T F'(x)\xi \leq \gamma_2\|\xi\|^2$ for all x and $y \in \Re^n$. Then the sequence $\{x_k\}_{k\geq 0}$ generated by the JCFN algorithm converges linearly to x^* .*

Proof. The Taylor series expansion of $F(x)$ about (x_k) is

$$F(x) = F(x_k) + F'(x_k)(x - x_k) + o(\|x - x_k\|^2)\ . \tag{20}$$

When $x = x^*$, (20) becomes

$$F(x^*) = F(x_k) + F'(x_k)(x^* - x_k) + o(\|x^* - x_k\|^2)\ . \tag{21}$$

But $F(x^*) = 0$, then we have

$$-F(x_k) = F'(x_k)(x^* - x_k) + o(\|x^* - x_k\|^2)\ . \tag{22}$$

Subtract then x^* from both sides of (13) to write,

$$x_{k+1} - x^* = x_k - x^* - D_k\, F(x_k)\ , \tag{23}$$

which upon substitution of (22) in it yields

$$x_{k+1} - x^* = x_k - x^* + D_k\,[F'(x_k)(x^* - x_k) + o(\|(x^* - x_k)\|^2)]\ . \tag{24}$$

It follows from (24) that

$$x_{k+1} - x^* = x_k - x^* - D_k\, F'(x_k)(x_k - x^*) + o(\|x_k - x^*\|^2), \tag{25}$$

which is the same as

$$x_{k+1} - x^* = (x_k - x^*)[E - D_k F'(x_k)], \tag{26}$$

where E is an identity matrix.
Take now the norm of both sides of (26) to arrive at

$$\|x_{k+1} - x^*\| \leq \|E - D_k F'(x_k)\|\,\|x_k - x^*\|. \tag{27}$$

Assume further the validity of Lemma 1 and boundedness of the Jacobian when $\delta = max[\gamma_1, \gamma_2]$, to write

$$\|x_{k+1} - x^*\| \leq (\sqrt{n} - \alpha\delta)\|x_k - x^*\|.$$

Clearly if $\theta = \sqrt{n} - \alpha\delta$, then

$$\|x_{k+1} - x^*\| \leq \theta \|x_k - x^*\|, \tag{28}$$

which means that the sequence $\{x_k\}_{k\geq 0}$ generated by the JCFN algorithm converges linearly to x^*. ■

4. Numerical Results

In this section we present numerical tests for the JCFN algorithm in application to some benchmark problems from the literature. A comparison of of the results of this algorithm is made with those of the following four alternative prominent methods.
(1) The Newton method (CN).
(2) The fixed Newton method (FN).
(3) The Incomplete Jacobian Newton method(IJN)
(4) The MSVF, denoting Newton -like method with the modification of right-hand side vector.
The criteria for this comparison are namely : (i) the number of needed iterations, (ii) the CPU time in seconds and (iii) the storage requirement. It should be noted that the MSVF and IJN methods were respectively proposed in [8] and [7]. The computational tests were made using MATLAB 7.4 with a double precision computer. The stopping condition used is:

$$\|x_{k+1} - x_k\| + \|F(x_k)\| \leq 10^{-8}.$$

Here is a listing of these benchmark problems, where each problem is accompanied by a table summarizing its comparative computational results. The symbol "−" in the tables indicates a failure due to memory shortages or / and when the number of iterations exceeds 250.

Problem 4.1. Solve the system of five nonlinear equations, [7].

$$F(x) = \left\{ \begin{array}{l} (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + 1)(x_1 - 1) + x_1(x_2 + x_3 + x_4) - 4 \\ (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + 1)(x_2 - 1) + x_2(x_1 + x_3 + x_4) - 4 \\ (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + 1)(x_3 - 1) + x_3(x_1 + x_2 + x_4) - 4 \\ (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + 1)(x_4 - 1) + x_4(x_1 + x_2 + x_3) - 4 \\ (x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + 1), (x_5 - 1), \end{array} \right\} = \left(\begin{array}{c} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{array} \right)$$

$x_0 = (-1.5, 3.5, -1.5, 3.5, -1.5)$

Problem 4.2. Extended Rosenbrock, [7].

$f_1(x) = -400x_1(x_2 - x_1^2) - 2(1 - x_1) + x_1(\sum_{j=2}^{n} x_j) - n + 1,$

$f_j(x) = 200(x_i - x_{i-1}^2) - 400x_i(x_{i+1} - x_i^2) - 2(1 - x_i) + x_i(\sum_{j\neq 1}^{n} x_j) - n + 1, i = 2, \ldots, n-1,$

$f_n = 200(x_n - x_{n-1}^2) + x_n(\sum_{j\neq i}^{n} x_j) - n + 1,$

$x_0 = (1.2, 1, 1.2, 1, 1.2, \ldots)^T.$

Table 1. Results for problem 4.1
(Number of iterations / CPU time)

CN	FN	MSVF	IJN	JCFN
6/ 0.0008	59/ 0.0003	$-(\alpha=-.08)$	–	8/ 0.0002

Table 2. Results for problem 4.2
(Number of iterations / CPU time)

n	CN	FN	MSVF	IJN	JCFN
			$(\alpha=-.08)$		
25	6/ 0.0240	45/ 0.0230	42/ 0.0240	7/ 0.0013	12/ 0.0011
50	6/ 0.0310	69/ 0.0280	47/ 0.0300	9/ 0.0020	14/ 0.0015
80	6/ 0.0497	73/ 0.0310	50/ 0.0354	18/ 0.0024	17/ 0.0018
100	6/ 1.0431	73/ 0.9801	51/ 1.0009	22/ 0.0041	18/ 0.0026
200	6/ 54.4550	78/ 21.8710	51/ 34.7210	26/ 0.0064	22/ 0.0035
500	6/ 106.7143	78/ 55.0377	64/ 87.5410	43/ 0.0176	28/ 0.0056
1000	6/ 109.614	90/ 68.6521	87/ 95.9642	84/ 0.0487	49/ 0.0109
5000	–	–	–	91/ 0.1348	58/ 0.0512
10000	–	–	–	92/ 0.9610	67/ 0.3137

Problem 4.3. System of n nonlinear equations, [7].

$F_j(x) = (\sum_{i=1}^{n} x_i^2 + 1)(x_j - 1) + x_j \sum_{i \neq j} x_i - n + 1, j = 1,2,\dots,n-1,$

$F_n(x) = (\sum_{i=1}^{n} x_i^2 + 1)(x_n - 1) \quad ; \qquad x_0 = (-1.5, 3.5, -1.5, 3.5, \dots)^T.$

Table 3. Results for problem 4.3
(Number of iterations / CPU time)

n	CN	FN	MSVF	IJN	JCFN
			$(\alpha = -.01)$		
25	8/ 0.0034	43/ 0.0029	58/ 0.0031	26/ 0.0014	24/ 0.0011
50	9/ 0.0054	49 / 0.0052	62/ 0.0050	28 / 0.0018	26/ 0.0012
80	9/ 0.0672	53/ 0.0583	79/ 0.0602	30/ 0.0024	29/ 0.0018
100	9/ 5.6500	56/ 5.1280	86/ 5.4070	30/ 0.0099	29/ 0.0043
200	9/ 8.6590	60/ 7.6420	–	31/ 0.0157	30/ 0.0085
500	9/ 19.2550	62/ 16.8163	–	32/ 0.0263	30/ 0.0089
1000	9/ 196.1002	69/ 188.5926	–	34/ 0.0481	31/ 0.0157
5000	–	–	–	34/ 0.0319	32/ 0.0148
10000	–	–	–	34/ 0.2564	32/ 0.0897

Table 4. Results for problem 4.4
(Number of iterations / CPU time)

n	CN	FN	MSVF	IJN	JCFN
			$(\alpha = -.01)$		
25	4/ 0.0434	–	10/ 0.0342	41/ 0.0215	18/ 0.0162
50	4/ 0.0480	–	10/ 0.0040	17/ 0.0362	18/ 0.0195
80	4/ 0.0973	–	12 / 0.0830	17/ 0.0749	20/ 0.0412
100	4/ 0.1516	–	12 / 0.0967	20/ 0.0841	20/ 0.0491
200	5/ 0.7898	–	12 / 0.4712–	23/ 0.2654	22/ 0.0724
500	5/ 1.30985	–	14 / 0.7823–	25/ 0.5134	23/ 0.0940
1000	5/ 6.1698	–	15 / 1.3514–	28/ 0.8135	26/ 0.1078
5000	–	–	–	32/ 0.9104	28/ 0.2196
10000	–	–	–	33/ 0.9671	30/ 0.3270

Problem 4.4. Trigonometric-exponential system, [9].
$f_1(x) = 3x_1^2 + 2x_2 - 5 + sin(x_1 - x_2)sin(x_1 + x_2)$,
$f_j(x) = 3x_i^2 + 2x_{i+1} - 5 + sin(x_i - x_{i+1})sin(x_i + x_{i+1}) + 4x_i - x_{i-1}exp(x_{i-1} - x_i) - 3$,
$f_n(x) = 4x_n - x_{n-1}exp(x_{i-1} - x_i) - 3$; $x_i^0 = 0, i = 2, \ldots, n-1$.

Problem 4.5. Singular Broyden, [6] .
$f_1(x) = ((3 - hx_1)x_1 - 2x_2 + 1)^2$,
$f_i(x) = ((3 - hx_i)x_i - x_{i-1} - 2x_{i+1} + 1)^2$,
$f_n(x) = ((3 - hx_n)x_n - x_{n-1} + 1)^2$,
$x_i^0 = -1$ and $h = 2$.

Problem 4.6. System of n nonlinear equations, [7].
$F_j(x) = (\sum_{i=1}^{n} x_i^2 + j)(x_j - 1) + x_j \sum_{i \neq j} x_i - n + 1$,
$x_0 = (-3, 3, -3, 3, -3 \ldots)^T$, $j = 1, 2, \ldots, n$.

Table 5. Results for problem 4.5
(Number of iteration/CPU time)

n	CN	FN	MSVF	IJN	JCFN
			($\alpha = -.01$)		
25	10/ 0.0045	609/ 6.716	398/ 3.1265	21/ 0.0041	12/ 0.0026
50	10/ 0.0078	864/ 15.9725	467/ 6.9861	21/ 0.0046	12/ 0.0031
80	10/ 0.0162	968/ 24.813	608/ 16.4521	22/ 0.0083	13/ 0.0039
100	13/ 0.0290	–	789/ 20.9741	23/ 0.0093	14/ 0.0072
200	13/ 0.0310	–	912/ 25.3170	23/ 0.0120	16/ 0.0094
500	13/ 0.0456	–	978/ 28.8672	32/ 0.0169	20/ 0.0108
1000	13/ 0.1981	–	–	35/ 0.0328	24/ 0.0142
5000	–	–	–	35/ 0.0526	24/ 0.0266
10000	–	–	–	35/ 0.0919	25/ 0.0695

Table 6. Results for problem 4.6
(Number of iteration/CPU time)

n	CN	FN	MSVF	IJN	JCFN
			($\alpha = -.01$)		
25	8/ 0.0023	42/ 0.0027	60/ 0.0038	21/ 0.0013	23/ 0.0012
50	8/ 0.0058	46/ 0.0056	67/ 0.0083	24/ 0.0049	23/ 0.0012
80	8/ 0.0661	49/ 0.0597	85/ 0.0772	25/ 0.0182	28/ 0.0021
100	8/ 5.5726	55/ 5.6836	98/ 6.6209	25/ 0.0233	28/ 0.0053
200	8/ 7.9367	60/ 7.5109	–	26/ 0.0247	30/ 0.0094
500	8/ 17.7592	66/ 15.9856	–	28/ 0.0259	31/ 0.0101
1000	8/ 180.1409	70/ 178.0479	–	30/ 0.0572	33/ 0.0119
5000	–	–	–	32/ 0.1745	36/ 0.0139
10000	–	–	–	32/ 0.2564	36/ 0.0703

The robustness index [9] is $V_j = \frac{t_j}{n_j}$, where t_j is the number of successes by method j and n_j is the number of problems attempted by method j . Table 7, illustrates how the the JCFN method considerably outperforms in robustness the CN, FN, MRVF and IJN for all tested problems.

Table 7. Robustness indecies for problems 4.1- 4.6

	CN	FN	MSVF	IJN	JCFN
V	0.7826	0.5435	0.6087	0.9782	1.

Clearly the JCFN method is the best with 100% of success in comparison with the CN method (having 78% success), the FN method (with 54%), the MRVF (with 61%) and the IJN, reaching 98% success. Moreover, from Tables 1-6 it is evident that the JCFN algorithm is

cheaper (in the sense of having the least CPU time) than the CN, FN, MRVF and IJN methods. This is basically due to the low computational effort associated with forming the approximation of the Jacobian inverse as well as eliminating the requirements of solving a Newtonian equation in each iteration.

Another advantage of the JCFN method over the alternative methods is in its lower storage requirement. Our method has totally eliminated the cost of storing the the Jacobian in each iteration. Unlike in the CN and MSVF methods, where Jacobian storage is required. That is why, as the dimension of a problem increases to $n \geq 5000$, the CN and MSVF fail to converge, due to huge memory requirements of the Jacobians in each iteration and to the accompanying exponential growth of their global costs. In summary it is possible to claim that our proposed JCFN is significantly cheaper than the CN, FN, MSVF and the IJN algorithms and is better than the MRVF and FN methods.

5. Conclusions

In this paper we have advanced a modification to Newton's method for solving systems of nonlinear equations. Our approach is based on approximating the Jacobian inverse to a diagonal matrix. The fact that the JCFN method solves the benchmark problems without the cost of computing and storing the Jacobian is a clear advantage of JCFN method over the CN, FN, MRVF and IJN alternatives. It is worth mentioning that the method is capable of significantly reducing the execution time (CPU time), as compared with the CN, FN , MRVF and IJN while still maintaining a high accuracy of the numerical result. Another fact that makes the JCFN method more attractive, is that in all the benchmark problems, it has never failed to converge. Hence, we may conclude that the JCFN is significantly cheaper than the CN, FN, MRVF and IJN and suitable for small, medium and large scale systems. It appears finally that the JCFN algorithm is a good alternative to CN, FN and MRVF methods especially when the function derivatives are excessively costly to store.

Acknowledgments.
The authors would like to thank the referees for their very useful comments and suggestions.

References

[1] A. Albert, and J. E. Snyman, Incomplete series expansion for function approximation, *Structural and Multidisciplinary Optimization* **34**, (2007), 21-40.

[2] L. Binh, On the convergence of a Quasi-Newton's method for sparse nonlinear systems, *Mathematics of Computation* **32**, (1978), 447-451.

[3] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, Inexact Newton methods, *SIAM Journal of Numerical Analysis* **19**, (1982), 400-408.

[4] J. E. Dennis, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Prince-Hall, Englewood Cliffs, N J, 1983.

[5] H. Drangoslav, and K. Natasa, Quasi-Newton's method with corrections, *Novi Sad Journal of Mathematics* **26**, (1996), 115-127.

[6] M. A. Gomes-Ruggiero, J. M. Martinéz, and A. C. Moretti, Comparing algorithms for solving sparse nonlinear systems of equations , *SIAM Journal on Scientific Computing* **13**, (1982), 459-483.

[7] L. Hao, and N. Qin, Incomplete Jacobian Newton method for nonlinear equation, *Computers and Mathematics with Applications* **56**, (2008), 218-227.

[8] N. Krejić, and Z. Lužanin, Newton-like method with modification of the right-hand-side vector, *Mathematics of Computation* **71** (237), (2002), 237-250.

[9] L. Lukšan, Inexact trust region method for large sparse systems of nonlinear equations, *Journal of Optimization Theory and Applications* **81** (3), (1994), 569-590.

[10] J. J. Tapia, E. Martinéz, and J. R. Torregrosa, Modified Newton's method for systems of nonlinear equations with singular Jacobian, *Journal of Computational and Applied Mathematics* **224** (1), (2009), 77-83.

Article history : Submitted June, 19, 2010 ; Accepted September, 24, 2010.