Computer Assisted Mechanics and Engineering Sciences, 7: 493–505, 2000.

Recent advances in solvers for nonlinear alegebraic equations

Deborah Dent, Marcin Paprzycki
School of Mathematical Sciences, University of Southern Mississippi
Hattiesburg, MS 39406-5106, USA

Anna Kucaba-Piętal
Department of Fluid Mechanics and Aerodynamics, Technical University of Rzeszów
ul. W. Pola 2, 35-959 Rzeszów

(Received November 3, 1999)

In this paper the performance of four solvers for systems of nonlinear algebraic equations applied to a number of test problems with up to 250 equations is discussed. These problems have been collected from research papers and from the Internet and are often recognized as "standard" tests. Solver quality is assessed by studying their convergence and sensitivity to simple starting vectors. Experimental data is also used to categorize the test problems themselves. Future research directions are summarized.

1. Introduction

Increasing power of desktop computers helps engineer's attempt at finding answers to problems that were not solvable in the past e.g. large systems of nonlinear algebraic equations [8, 17]. This increased interest resulted in a number of new algorithms being implemented and made available on the Internet [15]. However, while each of the solvers has been tested on its own, little is known how their performance compares when they are applied to the same problem. This may be contrasted with the solution of systems of linear equations, where a substantial literature comparing the performance of the state of the art solvers is readily available (e.g. compare reports of the LAPACK project summarized in [2] with the second edition of the only comprehensive book devoted to solution of systems of nonlinear equations [17] and references cited there). Situation is similarly fuzzy when test problems are considered. Even though some problems are relatively popular, still different researchers use different problems to test their algorithms. In addition, typical test problems are mostly for systems of 2–4 equations and only very few reach 10 equations. In our literature and Internet search we have not located tests corresponding to the real-life engineering problems of 100+ equations (similar to the avionics problem studied in [9, 10]). Separately, in engineering computing (e.g. in electrical engineering) problems arise which involve non-smooth functions (e.g. functions with the absolute value). These problems are also not represented in the test sets we encountered (however, since interest is directed primarily toward solving large systems of equations, the non-smooth cases will be omitted from our considerations).

In our earlier work [5, 6, 7] we have reported on our initial attempts at comparing the performance of the nonlinear system solvers applied to the popular test problems. We have found that the simple algorithms (which work well in the case of a single nonlinear equation) like bisection or Newton's method and its modifications perform poorly when applied to systems of equations. Second, we have established that the in-house developed simple implementations of known algorithms are only slightly less efficient than state of the art library codes. Finally, we were able to determine that out of 22 popular test problems, five are easily solvable by all solvers and thus their usability as test cases (establishing and differentiating quality of algorithms and/or their implementations) is minimal.

The aim of this paper is to summarize the results of our experimental studies comparing the performance of four "advanced" solvers for systems of nonlinear algebraic equations. These solvers were applied to the test problems in a way that is to resemble an engineer approaching a solution to a real-life problem. It is thus assumed that the potential user is not a highly trained numerical analyst and/or programmer who is able and willing to invest time into studying intricacies of methods and their implementations (e.g. working toward finding a proper homotopy map). Rather, we envision the user as someone who will be applying the codes as more or less black-box solvers, likely to follow the default settings and aiming at finding a verifiable solution to the problem. While the focus of the presentation is on the tests as they are stated in their simple "default" formulation, we also report our initial findings for the case when the size of the problem is increased (up to 250 equations).

The remaining parts of the paper are organized as follows. Section 2 briefly describes selected algorithms used to solve systems of nonlinear algebraic equations (only algorithms pertinent to the solvers used in our experiments are mentioned). In Section 3 we present the solvers (summarizing important details of experimental set-up) and the test problems. Section 4 summarizes the results of numerical experiments. Description of future research directions concludes the paper.

2. ALGORITHMS FOR THE SOLUTION OF SYSTEMS OF NONLINEAR ALGEBRAIC EQUATIONS

This section contains a brief summary of algorithms behind nonlinear solvers used in our experiments (in all cases the references cited and [17] should be consulted for the details). We assume that a system of n nonlinear algebraic equations $f(x) = 0$ is to be solved where x is n-dimensional vector and 0 is the zero vector.

2.1. Newton's method

The *Newton's* method for a system of nonlinear algebraic equations is a natural extension of the *Newton's* method for a single equation [8, 20]. It is the basis for many subsequent methods that will be discussed below. Let us assume that the function G is defined by

$$G(x) = x - J(x)^{-1} f(x) \tag{1}$$

and the functional iteration procedure is: select starting vector x_0 and generate a series of vectors

$$x_k = G(x_{k-1}) = x_{k-1} - J(x_{k-1})^{-1} f(x_{k-1}) \tag{2}$$

where $J(x)$ is the Jacobian matrix. The convergence rate for this method is fast, but the success of the method depends on a good starting vector x_0 .

2.2. Trust-region method

The *trust-region* method is a version of the *Newton's* method that reduces the full step size when it moves too far for the quadratic approximation to be valid. Given the function f and a step size of s, we can say that this method takes the view that the linear model $f(x_k) + f'(x_k)s$ of $f(x_k + s)$ is valid only when s is not too large (and thus a restriction on the size of the step is introduced). In the *trust-region* method, we replace the Jacobian matrix that is used by the *Newton's* method with an approximation [17]. Then the step [14] is obtained as an approximate solution of the sub-problem

$$\min \|f(x_k) + B_k s\| : \|D_k s\|_2 \leq \Delta_k \tag{3}$$

where B_k is either the exact Jacobian or a close approximation, D_k is a scaling matrix, s is the step size and Δ_k is the trust region radius. Stopping criteria is met if the ratio

$$\rho_k = \|f(x_k)\| - \frac{\|f(x_k + s_k)\|}{\|f(x_k\|} - \|f(x_k + B_k s_k\| \tag{4}$$

of the actual-to-predicted decrease in $\|f(x)\|$ is greater than some constant σ_0 (typically 0.0001). Otherwise, the radius of the trust region is decreased and the ratio re-computed. The radius may also be updated between iterations depending on how close the ratio is to the ideal value of 1. The convergence rate of this method is slow, but it can use an arbitrary starting solution vector and still converge.

2.3. Continuation method

The *continuation* method is designed to target more complicated problems and is the subject of current research efforts [1, 20, 14]. The method defines an easy problem for which the solution is known along with a path between the easy problem and the hard problem that is to be solved. The solution of the easy problem is gradually transformed to the solution of the hard problem by tracing this path. The path may be defined by introducing an addition scalar parameter λ into the problem and defining a function

$$h(x, \lambda) = f(x) - (1 - \lambda) f(x_0) \tag{5}$$

where x_0 is a given point in $\mathbf{R}^n$. The problem $h(x, \lambda) = 0$ is then solved for values of λ between 0 and 1. When $\lambda = 0$, the solution is clearly $x = x_0$. When $\lambda = 1$, we have $h(x, 1) = f(x)$, and the solution of $h(x, \lambda)$ coincides with the solution of the original problem $f(x) = 0$. The convergence rate of the *continuation method* varies, but the method does not require a good choice of the initial vector.

2.4. Homotopy method

The *homotopy* [4] and *continuation* methods are closely related. In the *homotopy* method, a given problem $f(x) = 0$ is embedded in a one-parameter family of problems using a parameter λ assuming values in [0,1]. Like the *continuation* method, the solution of an easy problem is gradually transformed to the solution of the hard problem by tracing a path. There are three basic path-tracking algorithms for this method: ordinary differential equation based, normal flow, and quasi Newton augmented Jacobian matrix. The original problem corresponds to $\lambda = 1$ and a problem with a known solution corresponds to $\lambda = 0$. For example, the set of problems

$$G(x, \lambda) = f(x) + (1 - \lambda) f(x_0) = 0, \qquad 0 \leq \lambda \leq 1, \tag{6}$$

for fixed $x_0 \in \mathbf{R}^n$ forms a homotopy. When $\lambda = 0$, the solution is $x(\lambda = 0) = x_0$. The solution to the original problem corresponds to $x(\lambda = 1)$. Similarly to the *continuation* method, the convergence rate of the *homotopy* method varies. The *homotopy* method does not require a good choice of the initial vector but proper implementation of this method involves defining the homotopy $h(z, t)$ and finding a numerical method for tracking the paths defined by $h(z, t) = 0$.

2.5. Tensor method

The *tensor* method [19] goes a step beyond *Newton's* method by including second-order derivative information from *f* into its model function [14]. This method converges more rapidly than *Newton's* method, particularly when $f'(x)$ is singular at the solution x_0. The *tensor* method [3] assumes that we have a system of nonlinear equations of the form

$$f : R_n \to R_m , \qquad m \geq n, \tag{7}$$

where f is assumed to be at least once differentiable. When m is equal n, the algorithm solves the nonlinear equation problem, $f(x) = 0$. If m is greater than n, it will solve the nonlinear least-squares problem. The *tensor* method approximates $f(x)$ by a quadratic model. This method bases each iteration on a quadratic model of the nonlinear function

$$M(x_c + d) = f(x_c) + f'(x_c d + \tfrac{1}{2}T_c dd) \tag{8}$$

where x_c is the current iterate, d is the direction, and T_c is a three-dimensional object referred to as a tensor and the second-order term is chosen so that the model is hardly more expensive to form, store, or solve than the standard linear model.

2.6. Line-search method

The *line-search* method [14], similarly to the *trust-region* method, when implemented together with the *Newton's* method, attempts to overcome its disadvantages by using basic strategies to improve global convergence behavior. Sometimes taking the full Newton step $p = \Delta x$ may cause us to move far for the quadratic approximation to be valid, so the goal is to move to a new point x_{new} along the direction for the Newton step p, but not necessarily all the way were $x_{\text{new}} = x_{\text{old}} + dp$, $0 < d \leq 1$. The aim is to find d so the $f(x_{\text{old}} + dp)$ has decreased sufficiently. The *line-search* method uses a linear model to achieve this. Given an approximation to the Jacobian matrix or the exact Jacobian, B_k, the *line-search* method obtains a search direction, d_k, by solving a system of linear equations

$$B_k d_k = -f(x_k). \tag{9}$$

The next iterate is then defined as

$$x_k = x_k + a_k d_k \tag{10}$$

where the line-search parameter $a_k > 0$ is chosen by the line-search procedure so that

$$\|f(x_{k+1}\| < \|f(x_k\| \tag{11}$$

When the "approximate" Jacobian is "exact", as in Newton's method, d_k is a downhill direction 2-norm, so there is certain to be an $a_k > 0$ such that

$$\|f(x_{k+1})\|_2 < \|f(x_k\|_2 . \tag{12}$$

This descent property does not necessarily hold for other choices of the approximate Jacobian, so *line-search* method is used only when B_k is either the exact Jacobian or a close approximation to it.

3. Experimental Setup

In our numerical experiments we have used four solvers [15]:

- HYBRD1 – combination of *trust-region* and *Powell's* (modified *Newton's*) method
- CONTIN – *continuation* method
- HOMPACK – *homotopy* method
- TENSOLVE – *tensor* method combined with the *trust-region* and *line-search* methods.

The algorithms described in Section 2 are the basis of these four solvers. We have earlier experimented with a number of variants of *Newton's* method and with the *bisection* algorithm (library based and in-house developed implementations) and found their performance rather unsatisfactory [5, 6, 7]. The results of these experiments will thus be omitted. Second, we have experimented with an in-house *hybrid* method based on [16]. Even though its performance was only slightly weaker than that of HYBRD1, to keep the presentation focused, these results will be left out as well.

3.1. HYBRD1

HYBRD1 is part of the MINPACK-1 suite of codes [13]. HYBRD1's design is based on a combination of the trust region and steepest descent concepts. It finds a zero of a system of n nonlinear functions in m variables by a modification of the *Powell-hybrid* method (a modified *Newton's* method) [16]. The code requires the user to provide subroutines to compute the function $f(x)$ while the Jacobian is computed internally by a forward-difference approximation. Termination occurs when the estimated relative error less than or equal the tolerance that is defined by the user. We used the same tolerance as is given as a default for other codes: square root of machine precision.

3.2. CONTIN

CONTIN [14], also know as PITCON [18], implements a continuation algorithm with an adaptive choice of a local coordinate. The *continuation* method is designed to be able to target more complicated problems and is the subject of various research efforts [1, 20, 14]. This method is expected to be slower than *line-search* and the *trust-region* methods, but it is to be useful on difficult problems for which a good starting point is difficult to establish.

This solver requires a user subroutine for the evaluation of the function $f(x)$ and either internally computes a finite-difference approximation of the Jacobian or utilizes a user-provided subroutine for its direct computation. In our experiments we have used the Jacobian approximation provided by the program (which is consistent across all solvers).

The user must also provide the initial starting vector and may provide target points. The code includes features for computation of target points and turning points. We followed an example provided with the code and choose as out target points the nth value of a known solution set. CONTIN provides a function to perform acceptance test, which checks if the solution is drifting away from the solution curve. The test uses a tolerance provided by the user (we followed the examples again and used 0.0001). The check is performed at the end of each iteration. If the relative error test is satisfied, the solution is accepted. If an acceptable solution is not found the code will terminate after exceeding the iteration limit.

3.3. HOMPACK

HOMPACK [23] is a suite of subroutines for solving nonlinear systems of equations by *homotopy* methods. The *homotopy* method is carried out via three qualitatively different algorithms: ODE-based (code FIXPDF), normal flow (code FIXPNF), and augmented Jacobian (code FIXPQF). The code is modular and arranged hierarchically to allow the user to easily supply information. For better results, users familiar with the code can call the tracking routines directly with complete control. The code is available in both Fortran 77 and Fortran 90 [22]. The Fortran 77 version was used in our test.

To properly solve a problem with this software, it is recommended that some knowledge about the solution is available so that the user can define the homotopy map for the continuation process. According to one the authors of the code [21] only when a proper homotopy map is constructed the full strength of this method is utilized and thus using the default homotopy map is not recommended.

Since our assumption is that the user may not be able or willing to spend time constructing the correct homotopy map we have decided to use the default setup and observe the program behavior. Even though this may put the otherwise powerful solution method (for examples of successful application of *homotopy* method see for instance [22, 23]) in a substantial disadvantage, we believe that this is a fair comparison since we used the same general methodology with regards to all solvers. We implemented and executed all of the test problems with each of the three algorithms. FIXPQF has a higher convergence rate and requires less iterations than PIXPDF and FIXPNF. We will report only the results of FIXPQF in our findings.

For the fixed point and zero finding problems, the user must supply a subroutine, $f(x, v)$, which evaluates $f(x)$ at x and returns the vector $f(x)$ in v, and a subroutine $fjac(x, v, k)$ which v the kth column of the Jacobian matrix of $f(x)$ evaluated at x. For the curve tracking problem, the user must supply a subroutine, $rhoa(v, \lambda, x, par, ipar)$, which given (λ, x) returns a parameter vector a in v such that $rho(a, \lambda, x) = 0$, and a subroutine, $rhojac(a, \lambda, x, v, k, par, ipar)$, which returns in v the kth column of the Jacobian matrix. We provided the functions f and $fjac$.

HOMPACK provides a routine that tracks the zero curve of the homotopy map. The user can provide a tolerance for relative error testing during the tracking process or can use the system default (square root of machine precision, which we used). When the curve tracking reaches its target, the code terminates with convergence, otherwise execute until it exceeds the iteration limit.

3.4. TENSOLVE

TENSOLVE [3] is a modular software package for solving systems of nonlinear equations and nonlinear least-square problems using the *tensor* method. It is intended for small to medium-sized problems (up to 100 equations and unknowns) in cases where it is reasonable to calculate the Jacobian matrix or its approximations.

This solver provides two different strategies for global convergence; a line search approach (default) and a two-dimensional trust region approach. Required input to the package includes (1) the dimensions m and n of the problem, where m is the number of nonlinear equations, and n is the number of unknowns; (2) a subroutine to evaluate the function $f(x)$; and (3) an estimate x_0 of the solution x^*. For our test, we used all default options, provided the dimension $m = n$, supplied functions for $f(x)$ and x, and used the internally calculated approximation of the Jacobian (consistently across all experiments). The default strategy for global convergence, line search, was also selected. The stopping criteria is meet when the relative size of $x_{k+1} - x < \epsilon^{1/2}$ where ϵ is the machine precision or if $\|f(x_{k+1})\|_\infty$ is less than $\epsilon^{2/3}$, or the relative size of $J(x_{k+1})^T f(x_{k+1})$ is less than $\epsilon^{2/3}$ and unsuccessfully if the iteration limit is exceeded.

3.5. Experimental set-up and environment

All codes are implemented in Fortran and were run in double precision on a PC with a Pentium Pro 200 MHz processor. For the initial numerical tests we have used 22 problems found in [11, 12, 24]. These tests come from the three collections of test problems for the solution of systems of nonlinear algebraic equations. While some of the problems come from the real life applications, others are artificially generated with properties not typical for real life applications.

In our earlier experiments we have established that test problems: Rosenbrock function, Discrete Boundary Value function, Broyden Tridiagonal function, Broyden Banded function and Freudenstein–Roth function (all from [12]) are easily solvable by all methods, including the simple variants of *Newton* and *bisection*. We have concluded that these problems do not introduce any interesting information about the quality of the algorithm and/or its implementation and should be removed from further considerations. We will thus skip the results obtained while solving them (interested reader may consult our earlier work for the details). Table 1 contains a list of the problems used in current experiments. Detailed descriptions can be found in the Appendix of [7]. Problems:

Table 1. Test problems

1. Powell singular function [12]	10. Semiconductor Boundary Condition [24]
2. Powell badly scaled function [12]	11. Brown Badly Scaled [12]
3. Wood function [12]	12. Powell singular Extended [11]
4. Helical valley function [12]	13. Rosenbrock Extended [12]
5. Watson function [12]	14. Matrix Square Root Problem [11]
6. Chebyquad function [12]	15. Dennis, Gay, Vu Problem [11]
7. Brown almost-linear function [12]	16. Trigonometric function [12]
8. Discrete integral equation function [12]	17. Exponential/Sine function [24]
9. Variably dimensioned function [12]	

5, 6, 7, 8, 14 and 15 allow for varying the number of equations, n. In Sections 4.1 and 4.2 we report the results collected when problems are solved for the minimal default number of equations (the way that they are defined in the test sets and used by other researchers). In Section 4.3 we report on experiments where the number of equations has been increased for three test problems.

4. Experimental results

In our experiments we have used various sets of initial values (which correspond to what an engineer could try to use in cases where the solution is unknown). First, for each of the test problems (as they were described in the literature) a starting vector was provided and we have utilized this data (results denoted DEFAULT). We have used starting vectors of zero (denoted ZERO) and one (denoted ONE) and, finally, vector of random numbers (denoted RANDOM).

In the test results, lack of convergence is denoted as *nc*. For TENSOLVE the code *nc* indicates that the execution was terminated before convergence due to exceeding maximum number of iterations. For HYBRD1 and HOMPACK *nc* indicates that the execution terminated before convergence due to iterations not making good progress as measured by the improvement from the last five Jacobian evaluations. The codes indicating abnormal termination for CONTIN are *nc*, and *ne*. Here *nc* means that the code did not reach a point of interest (convergence criteria not met before iteration limit met) and *ne* specifies the problem terminated because numerically singular matrix was encountered.

4.1. Medium difficult problems

In our experiments we have found that three problems (2, 5 and 16 from Table 1) have been solved easily by all four methods. Solved easily means that starting at least from one of the four starting vectors solver converged to the solution. Considering that five problems have already been deemed easy, these three test cases will be named medium difficult. The results are summarized in Table 2. Number of function evaluations is reported.

It can be observed that the set of values associated as defaults for the test problems are not always the best for finding the solution. Out of the codes the *continuation* and *homotopy* methods converge less often than the *hybrid* and the *tensor* method solvers. The lack of success of the *homotopy* method can be associated with the fact that the default homotopy map was used. It seems that problem 16 is the most interesting out of the group leading to the largest variation of the number of function evaluations. In all cases the number of function evaluations is so small that, on the PC used, the solution required almost no time to be completed.

Table 2. Results of medium difficult problems using DEFAULT, ZEROS, ONES and RANDOM initial value sets

#	Solver	N	Initial Value			
			DEFAULT	Zeros	Ones	random
2	CONTIN	2	43	*nc*	*nc*	*nc*
	HYBRD1	2	181	4	4	4
	TENSOLVE	2	259	6	6	6
	HOMPACK	2	8	*nc*	*nc*	*nc*
5	CONTIN	6	259	*nc*	*nc*	*nc*
	HYBRD1	6	96	8	8	8
	TENSOLVE	6	208	14	14	14
	HOMPACK	6	23	60	60	60
16	CONTIN	10	*nc*	588	*nc*	*nc*
	HYBRD1	10	84	84	84	108
	TENSOLVE	10	113	11	503	347
	HOMPACK	10	*nc*	4	4	4

Table 3. Results of difficult problems 1,3,6,7,8 using DEFAULT, ZEROS, ONES and RANDOM initial value sets

		N	Initial Value Type			
			DEFAULT	Zeros	Ones	Random
1	CONTIN	4	446	*nc*	*nc*	*nc*
	HYBRD1	4	*nc*	*nc*	*nc*	*nc*
	TENSOLVE	4	20	15	15	15
	HOMPACK	4	*nc*	*nc*	*nc*	*nc*
3	CONTIN	4	189	36	*nc*	*nc*
	HYBRD1	4	94	6	6	6
	TENSOLVE	4	50	10	10	10
	HOMPACK	4	*nc*	*nc*	*nc*	*nc*
4	CONTIN	3	68	400	15	102
	HYBRD1	3	27	5	5	5
	TENSOLVE	3	42	8	8	8
	HOMPACK	3	*nc*	*nc*	*nc*	*nc*
6	CONTIN	5	*ne*	*ne*	*ne*	*ne*
	HYBRD1	5	17	7	7	7
	TENSOLVE	5	31	12	12	12
	HOMPACK	5	10	6	6	6
7	CONTIN	10	50	*nc*	*nc*	*nc*
	HYBRD1	10	31	12	12	12
	TENSOLVE	10	93	22	22	22
	HOMPACK	10	*nc*	*nc*	*nc*	*nc*
8	CONTIN	10	*nc*	*nc*	*nc*	*nc*
	HYBRD1	10	16	12	12	12
	TENSOLVE	10	33	22	22	22
	HOMPACK	10	4	*nc*	*nc*	*nc*

4.2. Difficult problems

Tables 3 and 4 summarize the results of test cases for which one or more codes were unable to meet the convergence criteria. Again, in case of convergence, the number of function evaluations is reported.

The results in Tables 3 and 4 illustrate that, for the reasons specified above, the *homotopy* method is significantly less efficient than the remaining three methods. Overall, out of 14 tests, CONTIN was able to solve 8 problems, HYBRD1 11 problems, TENSOLVE 14 problems and HOMPACK 3 problems. Our results indicate that problems 12 and 14 are particularly hard as only TENSOLVE is capable of solving them for the four starting vectors used, while the remaining solvers have not converged in a single case.

Table 4. Results of difficult problems 9, 10, 11, 12, 13, 14, 15, 17 DEFAULT, ZEROS, ONES and RANDOM initial value sets

#	Solver	n	Initial Value Type			
			DEFAULT	Zeros	Ones	Random
9	CONTIN	10	254	*nc*	*nc*	*nc*
	HYBRD1	10	60	12	12	12
	TENSOLVE	10	165	22	22	22
	HOMPACK	10	*nc*	43	43	43
10	CONTIN	6	*nc*	*nc*	*nc*	*nc*
	HYBRD1	6	9	11	13	*nc*
	TENSOLVE	6	49	49	49	*nc*
	HOMPACK	6	*nc*	*nc*	*nc*	*nc*
11	CONTIN	2	82	*nc*	*nc*	*nc*
	HYBRD1	2	18	18	*nc*	*nc*
	TENSOLVE	2	3	3	3	3
	HOMPACK	2	*nc*	*nc*	*nc*	*nc*
12	CONTIN	12	*nc*	*nc*	*nc*	*nc*
	HYBRD1	12	*nc*	*nc*	*nc*	*nc*
	TENSOLVE	12	52	39	39	39
	HOMPACK	12	*nc*	*nc*	*nc*	*nc*
13	CONTIN	10	99	*nc*	*nc*	*nc*
	HYBRD1	10	47	12	12	12
	TENSOLVE	10	98	22	22	22
	HOMPACK	10	*nc*	*nc*	*nc*	*nc*
14	CONTIN	9	*ne*	*ne*	*ne*	*ne*
	HYBRD1	9	*nc*	*nc*	*nc*	*nc*
	TENSOLVE	9	10	10	10	10
	HOMPACK	9	*nc*	*nc*	*nc*	*nc*
15	CONTIN	6	*nc*	*nc*	*nc*	*nc*
	HYBRD1	6	125	8	8	8
	TENSOLVE	6	542	14	14	14
	HOMPACK	6	*nc*	*nc*	*nc*	*nc*
17	CONTIN	8	679	ne	110	87
	HYBRD1	8	11	*nc*	*nc*	15
	TENSOLVE	8	*nc*	3	33	23
	HOMPACK	8	*nc*	*nc*	*nc*	*nc*

In most cases the number of function evaluations remains small – which is related to the fact that the size of the system is small as well. When the convergence occurs no pattern can be observed that would allow us to predict which solver will use the smallest number of function evaluations.

4.3. Large problems

In an effort to compare the methods for larger values of n, this section explores the test problems for n greater than 50. The solvers that we are using are designed for small to medium size problems (up to 100 equations) but we have been able to execute problems with up to $n = 250$ equations. We tested all of the problems that allowed variable sizes for n with values 50 and greater (2 problems from the group designated as easy along with problems 5, 6, 7, 8, 14 and 15 from our test set). From this group of problems, for the Broyden banded function, Broyden tridiagonal function (both from the easy problems) and the Brown almost linear function (problem 7 in our test set) we were able to obtain convergence for $n \geq 50$. Table 5 contains the results of these test problems (with n equal to 50, 100, 150, 200, and 250) using the DEFAULT set of initial values. The DEFAULT set defined in Section 4 can be used for any value of n due to the fact that they are generated automatically (for details see Appendix in [7]). As previously, in case of convergence, the number of function evaluations is reported.

Table 5. Selection problem for larger values of n

Problem	Solver	$n = 50$	$n = 100$	$n = 150$	$n = 200$	$n = 250$
Brown almost-linear	CONTIN	210	410	610	810	1010
	HYBRD1	*nc*	*nc*	*nc*	*nc*	*ne*
	TENSOLVE	153	202	302	402	502
	HOMPACK	2	*nc*	*nc*	*nc*	*nc*
Broyden tridiagonal	CONTIN	626	1226	1826	2426	3026
	HYBRD1	61	111	161	211	*ne*
	TENSOLVE	204	404	604	804	14
	HOMPACK	207	583	*nc*	*nc*	*nc*
Broyden banded	CONTIN	886	1736	2586	3436	4286
	HYBRD1	69	119	169	219	*ne*
	TENSOLVE	255	505	755	1005	1255
	HOMPACK	209	590	*nc*	*nc*	*nc*

TENSOLVE and CONTIN were easily modified (increase in the array sizes) to handle larger numbers of equations. The same modifications were made to HYBRD1 and HOMPACK. We expect that additional code modifications, such as more array size increases and modification of constants, are required to HYBRD1 to allow it to handle n greater than 200 and to HOMPACK for n greater than 100. We plan to investigate this in the near future.

As expected, as the number of equations is increasing, the number of function evaluations increases as well. However, there does not seem to be a substantial increase of computational effort. For instance, for the Broyden Banded problem, the number of function evaluations remains approximately 5 per equation for TENSOLVE and 17 for CONTIN.

It can be noticed, that while the HYBRD1 solver seems to be running into convergence problems, when it converges it uses the smaller number of function evaluations. Surprisingly, while for the other problems the CONTIN solver was relatively inefficient, here is converges in all these cases that the TENSOLVE does. However, it uses a much larger number of function evaluations.

5. Conclusions and Future Work

In this paper we have reported on our experiments comparing performance of solvers for systems of nonlinear algebraic equations on a number of test problems. We have found that

- solvability of the test examples depend on the algorithm and the initial vector,
- without special effort devoted to finding the proper homotopy map the *homotopy* method is relatively inefficient,
- problems, which are not solvable using one method when treated by another method can be solvable,
- currently popular test problems can be divided into three groups:
  - "easy problems" – where even the least powerful methods are capable of converging and which should be discarded from further considerations,
  - "medium difficult problems" – where only the more advanced solvers can be expected to solve them,
  - "hard problems" – where convergence is difficult to obtain and which are the most interesting for testing of new methods,
- even though the existing solvers can cope with large systems of equations, to be able to achieve this their implementations have to be modified,
- test problems' number of equations are too small to add a timing component into the evaluation method.

Out of the methods tested, the *tensor* method appeared to be most robust and capable of solving largest number of problems. When examining the solution sets we noticed a slight difference in the reported results. We have thus, we have compared the second norm of the differences between the solution vectors for a few problems. Table 6 contains the results.

Table 6. Comparisons of standard deviations from easy problems

	Standard Deviation				
	Rosenbrock	Discrete Bond	Broyden Tri-diagonal	Broyden Banded	Freudenstein-Roth
TENSOLVE vs. HYBRD1	6.65357E−07	1.068147639	0.004502123	0.000768171	0.002175014
TENSOLVE vs. CONTIN	6.65357E−07	1.068147656	0.004502051	0	0.002175014
TENSOLVE vs. HOMPACK	6.65357E−07	1.068147586	0.004502172	0.000768172	0.002175014
HYBRD1 vs. CONTIN	0	5.88955E−08	9.25758E−08	3.75472E−08	2.1142164E−07
HYBRD1 vs. HOMPACK	0	9.02943E−08	1.5143E−07	7.67779E−09	0
CONTIN vs. HOMPACK	0	1.2169E−07	1.9892E−07	3.81884E−08	2.1142164E−07

These results indicate that the results reported by the TENSOLVE relatively different that these reported by the remaining solvers. Taking into account that in all cases the square root of machine epsilon was used as the required tolerance and that codes were running in double precision we can conclude that the results reported by HYBRD1, HOMPACK and CONTIN are numerically "identical." These observations agree with the documentation of TENSOLVE, which indicates that the results obtained by it are the best approximations to the solution [3]. This may also suggest that TENSOLVE should be used as a "predictor" and followed by a different code (a "corrector") to obtain the final solution.

It should be stressed that even though the tests used cover a wide spectrum of functions they clearly do not exhaust the possibilities arising in practical engineering applications. First, such applications can result in systems of 100's of nonlinear algebraic equations and none of the test cases used in the current research seems to belong to this category. In addition, none of the examples belongs to the class of non-smooth functions (e.g. functions with an absolute value).

In the near future we plan to proceed as follows. We will adjust the test set by removing the easy problems and adding real-life problems with large number of equations (our literature and Internet searches have located additional potential test cases). We will use these new tests to compare the performance of the existing solvers. We plan to complete our investigations in two directions. First, to build a complete picture describing the sensitivity of the test problems and solution methods to the selection of the starting vector. Second, we will investigate solution of systems with large number of equations. We will search for codes that handle larger numbers of equations and/or modify the existing codes. We expect that with really large nonlinear systems we will be able to introduce the wall-clock time as an additional performance measure. Finally, an attempt at solving large, engineering based systems will be made.

Acknowledgements

Work of Deborah Dent was sponsored by the US Army Corps of Engineers, Engineering Research and Development Center, Waterways Experiment Station, Vicksburg, Mississippi, and USA.

References

[1] E. Allgowerr, K. George. *Numerical Continuation Methods: An Introduction.* Springer-Verlag, Berlin, 365, 1990.

[2] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, D. Sorensen. *LAPACK Users' Guide.* SIAM, Philadelphia, 1994.

[3] A. Bouaricha, R. Schnabel. Algorithm 768: TENSOLVE: A software package for solving systems of nonlinear equations and nonlinear least-squares problems using tensor methods. *ACM Trans. Math. Software*, **23**(2): 174–195, 1997.

[4] R.L. Burden, J.D. Faries. *Numerical Analysis*, 575–576. PWS-Kent Publishing Company, Boston, 1993.

[5] D. Dent, M. Paprzycki, A. Kucaba-Piętal. Performance of solvers for systems of nonlinear algebraic equations. *Proceedings of 15th Annual Conf. on Applied Math*, Edmond, OK, 67–77, 1999.

[6] D. Dent, M. Paprzycki, A. Kucaba-Piętal. Studying the numerical properties of solvers for systems of nonlinear equations. *Proceedings of the Ninth International Colloquium on Differential Equations* VSP, Utrecht, The Netherlands, 113–118, 1999.

[7] D. Dent, M. Paprzycki, A. Kucaba-Piętal. Testing convergence of nonlinear system solvers. *FSCC*, 1, `http://pax.st.usm.edu/cmi/fscc98_html/processed/`, 1999.

[8] G.H. Hostetter, M.S. Santina, P. D'Capio-Montalvo. *Analytical, Numerical and Computational Methods for Science and Engineering.* Prentice Hall, Englewood Cliffs, 1991.

[9] A. Kucaba-Piętal, L. Laudanski. *Modeling Stationary Gaussian Loads.* Scientific Papers of Silesian Technical University, Mechanics, textbf121, 173–181, 1995.

[10] L. Laudanski. Designing random vibration tests. *Int. J. Non-Linear Mechanics*, **31**(5): 563–572, 1996.

[11] J.J. More. A collection of nonlinear model problems. *Amer. Math. Soc.*, **26**, 723–762, 1990.

[12] J.J. More, B.S. Garbow, K.E. Hillstrom. Algorithm 566. *ACM Trans, Math. Software*, **20**(3): 282–285, 1994.

[13] J.J. More, D.C. Sorensen, K.E. Hillstrom, B.S. Garbow. *The MINPACK Project, in Sources and Development of Mathematical Software.* Prentice-Hall, 1984.

[14] NEOS Guide. `http://www-fp.mcs.anl.gov/otc/Guide/`, 1996.

[15] Netlib Repository. `http://www.netlib.org/liblist.html`, 1999.

[16] M.J.D. Powell. *A Hybrid Method for Nonlinear Algebraic Equations* (in Polish). Gordon and Breach, Rabinowitz, 1979.

[17] W.C. Rheinboldt. *Methods for Solving System of Nonlinear Equations.* SIAM, Philadelphia, 1998.

[18] W.C. Rheinboldt, J. Burkardt. Algorithm 596: A program for a locally parametrized continuation process. *ACM Trans. Math. Software*, **9**: 236–241, 1983.

[19] R.B. Schnabel, P. Frank. Tensor methods for nonlinear equations. *SIAM J. Numer. Anal.*, **21**: 814–843, 1984.

[20] J. Stoer, R. Bulirsh. *Introduction to Numerical Analysis.* Springer, New York, 521, 1993.

[21] L.T. Watson. personal communication.

[22] L.T. Watson, M. Sosonkina, R.C. Melville, A.P. Morgan, H.F. Walker. Algorithm 777: HOMPACK 90: Suite of Fortran 90 codes for globally convergent homotopy algorithms. *ACM Trans. Math. Software*, **23**(4): 514–549, 1997.

[23] L.T. Watson, S.C. Billups, A.P. Morgan. Algorithm 652:HOMPACK: A suite of codes for globally convergent homotopy algorithms. *ACM Trans. Math. Software*, **13**: 281–310, 1987.

[24] U.N. Weimann. A family of Newton codes for systems of highly nonlinear equations. *ZIB Technical Report* TR-91-10. ZIB, Berlin, Germany, 1991.