Intelligent system for non-linear simultaneous equations solving

V. Negru, St. Maruster, C. Sandru

West University
Faculty of Mathematics
Computer Science Department
1900 Timişoara, ROMANIA
email: {vnegru,stmaru,sandru}@info.uvt.ro

December 05, 1998

Abstract

The goal of this study is to develop an intelligent system able to assist the user in solving non-linear equation systems. Modeling the human expertise, the system must be able to choose automatically the most adequate numerical methods or to combine numerical methods for solving a non-linear equations system.

Our study refers to: the acquisition and formalization of human expert knowledge; the task-oriented formalization of reasoning; the design of the numerical routines library and of the methods base.

For this purpose we are using a knowledge based system (rules and objects based) with the possibility to combine symbolic and numerical computing. We are using classical numerical methods (Newton, Secant etc.), and also recent methods (conjugated gradient, Broyden etc.).

A prototype was implemented using the expert system shell Clips.

Keywords: expert systems, task-oriented model, coupling symbolic and numerical computing, numerical analysis, non-linear equation systems.

CR Classification: I.2, I.2.4, I.2.5, G.1, G.1.5

1 Introduction

There is an increasing need to develop knowledge-based systems for scientific computing [12, 24, 10]. The quality of these programs is proportional to the number of methods that may be used. Lack of information or the necessary expertise in making the correct choice can often be observed: no guidance and control in choosing the right method, in the interpretation of the results and in the sequencing of the execution stages is performed. Thus, programs are privileged in the detriment of methods.

A solution we propose is *to step from the program encapsulating the method to the method encapsulating the program.* Besides knowledge about `how?` a method is realized by a program, knowledge about `when?` and `why?` should also be considered.

Combining *symbolic methods* derived from AI with (traditional) *numerical methods* derived from Computational Mathematics leads to increased power and more opportunities to solve the problems. Employing AI techniques in order to link numerical methods enables the use - on a higher level - of symbolic techniques capable of reasoning and explanation upon the lower level of numerical methods.

A system which combing both symbolic and numerical methods, which knows the available numerical methods and which reasons about applying, executing and about the results of numerical methods is said to be a *coupled system* [12, 9]. A favorable medium to obtain this coupling is made by Expert Systems, which allow to integrate the possibilities of solving problems and explain of the systems based on knowledges with the precision of the traditional numerical calculus.

An expert system for solving non-linear equations systems was developed at Carnegie-Mellon (Talukdar, 1983) [13]. The expert system supervise the application of three different numerical programs for solve some non-linear algebraically equation systems which cannot be solved with only one method. The choice is made in function of the consequence concerning convergence.

Generally, there is not a unique recipe in solving the non-linear equations systems (non-linear simultaneous equations) [14]. To choose a particular method one should take into account several external aspects: the form of the system, knowledges about a "good" iteration, structural characteristics (the quality of the Jacobian, condition number), etc. Other elements who influence the solving process are: the precision, the memory space, the convergence speed.

The goal of this study is to develop an intelligent system able to assist the user in solving a non-linear equations systems. Modeling the human expertise, the system must be able to choose automatically the most adequate numerical methods or to combine two or more numerical methods for solving a non-linear equations systems. The system assume coupling the symbolic and numeric calculus [25, 1].

We have considered the following goals:

- the acquisition and the formalization of the expert knowledge;
- the development of the methods base and of the corresponding routines library;
- the design of a task oriented formalism for modeling of problem solving reasoning;

A prototype NESS (Non-linear Equations System Solver)[19, 18] was implemented in CLIPS 6.0 [3], expert system shell. The graphics interface was developed in WxWindows [28].

The intelligent system can be used by experts (specialists in numerical analysis) for a better knowledge of the non-linear systems behavior on one hand, and to solve non-linear equation systems as model of some phenomena, by users having not deep knowledge of numerical analysis, on the other hand.

2 Solving non-linear equations systems

2.1 Problem definition

The methods for solving nonlinear systems of equations have some certain intrinsic characteristics (which do not generally depend on the particular system which has to be solved); such characteristics are crucial elements of the general solving strategy. There have also been developed special methods for certain classes of problems, having in view to use some particularities of the problems and to get enhanced performances by these means.

In the following, we shall refer to four characteristics.

2.1.1 The convergence rate and the computing effort per one iteration

Generally, these two characteristics exclude each other. A method with a superior convergence rate (quadratic,super-quadratic etc.) -e.g. the methods of Newton, Chebychev, Steffensen and others generally entail a great computing effort per one iteration (in the case of the considered methods of Newton and Steffensen the number of operations per one iteration has the order n^3). In the literature of numerical calculus there have been many attempts for getting methods with high rates of convergence and a low computing effort. Some of the better known results are the quasi-Newton methods [6]; the convergence rate is super-linear and the number of operations per one iteration has the order n^2.

The computing may also be influenced by the number of function evaluations per one iteration. In general, the methods that use the Jacobian (Newton, gradient and others) need a number of function evaluations of the oder n^2. Even from this point of view, the methods of quasi-Newton type are profitable because they do not need the computation of the Jacobian. Instead of this, such methods need a matrix that approximates at the very beginning the reverse of the Jacobian computed in solution [5].

2.1.2 The possibilities to exploit the sparsity

Sometimes the nonlinear systems of great dimensions are sparse, that is in every equation there are only few unknown variables. The methods which involve the inversion of the Jacobian or the solving of a nonlinear system in which the matrix is the Jacobian can not exploit the sparsity. From this point of view are useful the methods of gradient type.

Fridman's variant [7] has the form:

$$x_{k+1} = x_k - \frac{||F(x_k)||^2}{||F'(x_k)^T \cdot F(x_k)||^2} F'(x_k)^T F(x_k).$$

We considered above the system $F(x) = 0$, $F : D \subseteq R^n \to R^n$, $F'(x)$ the Jacobian and $||.||$ the Euclidean norm. The most important processing of each iteration is the product $F'(x_k)^T F(x_k)$. When $F'(x_k)$ is sparse, one can only store the nonzero elements together with additional information regarding the position of the elements in the matrix and therefore the processing becomes inexpensive. Unfortunately, the convergence is only nonlinear; this means that, for a given precision, we need a big number of iterations, that is a great overall computing effort. There appear two problems:

1. Are there situations in which this method (Fridman's variant) has a super-linear convergence? A result is given in the paper [15]; the gradient method has super-linear convergence if the condition number of the Jacobian computed in solution is $c = 1$ ($c = ||F'(x^*)^{-1}|| \cdot ||F'(x^*)||$, where x^* is the solution of the system).

2. Getting new variants which keep the advantages of the gradient method (relatively low computing effort per one iteration and possibility to exploit the sparsity) and also have a super-linear convergence. We mention here the method of the conjugate gradient, specially developed for minimizing nonlinear functionals but also used for nonlinear systems.

2.1.3 Numerical stability

The topic is mainly the influence of errors on the qualitative properties of the methods, specially on the convergence speed. Methods involving the gradient inversion, usually having good properties of convergence may be substantially depreciated while having badly conditioned systems (e.g. when $||F'(x^*)|| \approx 0$). In this case the computing of the Jacobian and its inversion must be accomplished with superior precision, fact that considerably increases the computing effort. Instead, the methods of successive relaxation, the methods of gradient type (including here the conjugate gradient) etc. are numerically stable. E.g. the qualitative behavior of the Fridman's variant in not influenced of the computing errors in the evaluation of the Jacobian (This is obviously true if the errors remain between certain limits).

The decision about which method is best fitted for solving a given system must be made considering these characteristics of the methods. It is obvious that -for instance- for a sparse system should be tried a method that can exploit the sparsity of the Jacobian. Some other times, the problem is more difficult; for instance, for a badly conditioned system is not recommended a method which needs the inversion of the Jacobian, but even finding out that a system is badly conditioned is a problem not so easily to be solved.

Beside these characteristics of general order (convergence speed, computing effort, possibilities to exploit the sparsity) there are many methods with special characteristics, appropriate for particular nonlinear systems. This is the case of the nonlinear systems proceeded from the discretization of nonlinear differential systems. For example, the problem of heat conduction with nonlinear limit conditions leads by discretization with finite differences to a nonlinear system shared in the next form:

$$U(x) + V(x) = 0,\ U, V : D \subset R^n \to R^n,$$

the operators U and V having some special properties (for instance, U is linear and V is diagonal) For such systems (the linear case) Peaceman and Rachford proposed a method named ADI (Alternating Direction Iteration) which is able to use this particular form. The method was generalized for the nonlinear case by Kellog [11]. Another significant example is the weakly nonlinear two point problem

$$u" = f(t, u),\ t \in [0, 1], u(0) = \alpha,\ u(1) = \beta,$$

where $f : [0, 1] \times R^n \to R^n$ and $\alpha, \beta \in R^n$. By discretization with finite limits one gets the nonlinear system

$$Ax + \phi(x) = 0,$$

where

$$A = \begin{pmatrix} 2 & -1 & 0 & \dots & 0 & 0 & 0 \\ -1 & 2 & -1 & \dots & 0 & 0 & 0 \\ 0 & -1 & 2 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & -1 & 2 & -1 \\ 0 & 0 & 0 & \dots & 0 & -1 & 2 \end{pmatrix}$$

$$\phi(x) = \begin{pmatrix} h^2 f(t_1, x_1) - \alpha \\ h^2 f(t_2, x_2) \\ \dots \\ h^2 f(t_{n-1}, x_{n-1}) \\ h^2 f(t_n, x_n) - \beta \end{pmatrix}$$

The constant h is the discretization step, $h = \frac{1}{n+1}$.

This system can be solved using the next simple method

$$x_{k+1} = -A^{-1}\phi(x_k),$$

the matrix A^{-1} being possible to make explicit [14].

This method is remarkable for satisfying the principle of independence of the discretization step in the sense that for a given precision the number of iterations does not depend on discretization step.

Solving nonlinear problems also involve a series of collateral extremely important problems; for example, choosing the initial iteration, error evaluation, possibility of parallelization, the problem of finding all solutions in a given domain, etc.

- *Finding the initial iteration.* In essence, it comes to the estimation of convergence domain [22], a difficult and less studied problem. One can have in mind one of the following strategies:

1. Exploiting as much as possible the technical information of the problem. For instance, if the solutions represent concrete technical data (lengths, pressures, etc.) one can make reasonable estimates of the initial iteration.
2. Heuristic determinations (by tests). For instance, the systematic examination of the knots of an appropriate net. If having large dimensions ($n > 100$) such procedure can lead to a huge amount of computing.
3. Estimating a portion of the convergence domain's boundary and testing two points placed on both sides of the boundary (for example, on the normal). In some cases, the estimation of the convergence domain can confine to the estimation of the domain of asymptotic stability for ordinary nonlinear differential systems for which are known numerical procedure.

- *Error evaluation.* It is necessary both for the decision to stop the computing process and for additional information needed by the user. Often the simple fact that the left side of the system is zero by an unknown precision is not enough. In some of the cases, the theorems of convergence provide such errors. There are also fine evaluations of the errors in some particular cases (Newton's method) [8].

- *Possibility of parallelization.* When having large dimensions, solving nonlinear systems involves a huge volume of computing. Specially the methods that need to compute and sometimes to invert the Jacobian lead to a great number of function evaluations. A possible strategy is to compute them in parallel and to adequately organize the computing that occur afterwards -the product between Jacobian and a vector or solving a nonlinear system.

- *Getting all solutions in given a rectangular domain.* Referring to complexity of this problem, finding all solutions from a given domain is generally uncertain. First, for some nonlinear system, determining the number of solutions is a difficult problem (the effective computing of the topologic gradient is not simple). Probably, the only strategy that may be taken into account is the examination of some net of points in the given domain.

2.2 Mathematical model

The solving non-linear equations systems is a classical problem of numerical analysis and there are numerous theoretical and computing results, as well as a rich literature in this domain [14]. The numerical methods allow the possibility of elaborating some efficient programs, and also the solving some specific situations, degenerating cases, sparse systems, etc. Because in some concrete problems non-linear systems had big dimensions (up them 100 equations), the researchers efforts were concerning towards obtaining some methods with rapid convergence and a reduced calculus effort per iteration.

In the next, we present the mathematical definition of a non-linear equations systems and same characteristics definitions.

Let be a non-linear system:

$$F(x) = 0, \quad where \quad F : D \subseteq \Re^n \longrightarrow \Re^n \tag{1}$$

We consider the problem of solving numerically this system, in the sense of determinate $x^* \in D$ such as $F(x^*) = 0$. The elements of the $\Re^n$ are columns vectors, thus if $x \in \Re^n$ then:

$$\mathbf{X} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

We can write also $x = (x_1, \cdots, x_n)^T$.

We note the components of F by $f_i, i = 1, \cdots, n$, therefore:

$$\mathbf{F}(\mathrm{x}) = \begin{pmatrix} f_1(x_1, \cdots, x_n) \\ \vdots \\ f_n(x_1, \cdots, x_n) \end{pmatrix} \tag{2}$$

The used norm is the Euclidean norm in $\Re^n$, thus if $x \in \Re^n$ then:

$$\| x \| = (\sum_{i=1}^{n} x_i^2)^{1/2} \tag{3}$$

Many methods involve the Jacobian calculus F which we note $F'(x)$, where:

$$\mathbf{F}'(\mathrm{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & ,\cdots, & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_n}{\partial x_1} & ,\cdots, & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{4}$$

We say that the system $F(x) = 0$ is *sparse* if the matrix $F'(x)$ is sparse for $\forall x \in D$. This mean that every component f_i of F depends of a small number of unknowns $x_1, \cdots, x_n$. Such a situation must be exploited in the case of systems with big dimensions, for example by stoking in the computer memory only the nonzero elements. Some methods need the Jacobian to be a nonsingular matrix in the point x^*, therefore to exist $F'(x^*)^{-1}$.

The condition number of a matrix A is by definition:

$$c = \| A \| \| A^{-1} \| \tag{5}$$

This number is very important concerning the convergence speed.

The methods considered in our work are iterative methods with one step, having the form:

$$x_{k+1} = \Phi(x_k), \quad x_0 - given \tag{6}$$

The only one method which make exception is the secant method, a method with two steps:

$$x_{k+1} = \Phi(x_k, x_{k-1}), \quad x_0, x_1 - given \tag{7}$$

The function Φ is call *function of iteration*; it depends on F and essentially this function define the calculus method. For example, for the Newton method, Φ has the form:

$$\Phi(x) = x - F'(x)^{-1} F(x) \tag{8}$$

Generally, such a methods generate a string $\{x_k\}$ which will approximate the solution x^* of the system. (Let notice that will not be any confusion between the terms of the string and of the components of a term; therefore $x_k \in \Re^n$ and $x_k = (x_{k1}, \ldots, x_{kn})^T$). The sense of this approximation is that $x_k \to x^*, \quad k \to \infty$; different methods are building such as this condition has to be fulfilled. Generally, the convergence conditions assure only a locale convergence, in the sense that if x_0 is sufficiently closed of the solution, then the generated string $\{x_k\}$ of the considered method converge to x^*.

From the point of view of the calculus performances, it is interesting the string convergence speed, characterized by the convergence rate and the asymptotic error. By definition, *the convergence rate* r is a positive number, such as:

$$\lim_{k \to \infty} \frac{\| x_{k+1} - x^* \|}{\| x_k - x^* \|^r} = \rho, \quad 0 < \rho < \infty. \tag{9}$$

Let notice that, because we can consider $\| x_k - x^* \| < 1$, the bigger r, must be the closer x_{k+1} of the x^*, that is the convergence speed is bigger. If $r = 2$, the convergence is call *squarely*; if $r > 1$, we say that we have *super-linear* convergence. The number ρ is call *asymptotically error*. For fixed r, it is better that ρ be smaller.

From the numerical solving methods, we selected some classical ones with high performances (*Classic Newton, Discretized Newton, Secant, Steffensen, Parallel Lines*, etc.) or others relative recently (*Broyden, Friedman, Conjugated Gradient, Successive Overrelaxation, Alternating Directions Iteration*, etc.), as well as special methods, generally useful for solving equations with differences resulted from continuous problems discretization (for example, the equations with differences corresponding to nonlinear weak bilocale problem).

2.3 Some numerical methods description

In the following we present some of the numerical methods used to solve nonlinear equations systems and we will insist on implementations issues and on characteristics concerning the convergence, calculus effort, programming particularities, memory requirements, etc. These characteristics are fundamental for decision making in choosing a particular method for the solving process.

Among the discussed methods we selected some classical methods with high solving performances (Newton, Secant Method, Steffensen, Parallel Lines Method, etc.) as well as more recently developed methods (Broyden, Friedman, Conjugated Gradient with Reeves Balanced Factor, Alternate Directions, etc.). We also discuss some special methods, used for solving equations with differences by discretization of continuous problems (for example, equations with differences corresponding to bilocal nonlinear problem) .

2.3.1 Newton's method

The calculus formula for the Newton's method is:

$$x_{k+1} = x_k - F'(x_k)^{-1} F(x_k) \tag{10}$$

For this method one computes the actual values of the function, the derivative and the solution of the linear system by making the operations below:

- multiply the above equality with $F'(x_k)$, the value $x_{k+1} - x_k$ will be denoted by b and we will solve the system:

  $$F'(x_k)b = -F(x_k),$$

  next b will be $b = x_{k+1} - x_k$ and the iterative process will be evaluated by $x_{k+1} = b + x_k$ in the main program.

- solve the linear system by making a classical Gauss elimination

- compute the Jacobian by programming the derivatives

We point out some of the method's characteristics

1. A very good convergence with a convergence rate of $r = 2$ if $F'(x^*)$ is unsure.

2. The calculus effort per iteration is great. At every step we must solve the system $F'(x_k)b = -F(x_k)$, which means about $n^3/3$ operations.

3. This method is not recommended for systems having a great number of equations.

4. It is not possible to use the system sparsity because from the fact that the matrix $\mathbf{A}$ of the derivatives is sparse does not result that $\mathbf{A}^{-1}$ is also sparse.

2.3.2 Secant method and Steffensen method

These methods resemble the Newton Method. The Jacobian $F'(x)$ from the general formula in the Newton's Method will be approximated in both the methods we discuss with $J(x,h)$, where $h \in \Re^n$. The following formula results:

$$x_{k+1} = x_k - J(x_k, h_k)^{-1}F(x_k) \tag{11}$$

The form of the method is identical with Newton, so the calculus procedure rests the same.

In order to approximate $F'(x) \approx J(x,h)$, let be $h = (h_1, \ldots, h_n)^T$, then:

$$J(x,h) = (d_{ij}),$$

$$d_{ij} = \frac{f_i(x_1, \ldots, x_j + h_j, \ldots, x_n) - f_i(x_1, \ldots, x_n)}{h_j} \tag{12}$$

Observation: If $h_j \to 0$, then $d_{ij} \approx \frac{\partial f_i}{\partial x_j}$, and this justifies the approximation.

Function of the choice of the h_j values, it is possible to distinguish variants:

- **Approximative Newton method**:

  The values h_j , where $j = \{1, \ldots, n\}$ are chosen equals with a value h, where h must be "good enough"; it is impossible to choose choice criteria.

- **Secant method**:

  The values h_j are chosen $h_j = y_j - x_j$, where:

  $x = (x_1, \ldots, x_n)^T$ - represents the old iteration;

  $y = (y_1, \ldots, y_n)^T$ - represents the new iteration (the one just computed). In this case the convergence rate is $r = \frac{1+\sqrt{5}}{2} \approx 1.6$.

  Note that both iterations must be retained in memory.

- **Steffensen method**:

  The values h_j are taken $h_j = f_j(x)$. In this case the convergence rate is $r = 3$.

Observation: In the case of the Secant and Steffensen methods $h_{ij} \to 0$, because the derivatives are computed dividing very small numbers resulted form subtraction of not very small numbers and they no more have a good precision. This raise problems about numbers precision.

The main characteristics are:

1. All variants have good convergence.

2. They presents the advantage that it is no need for programming the calculus formulas for partial derivatives. When the system has a medium dimension ($n = 100$), the programming effort is substantially reduced (there is no need to program 10,000 formulas for partial derivatives).

3. The calculus effort per iteration is great. It is necessary to solve a linear system at every step.

4. The method is recommended for medium systems.

2.3.3 Broyden method

The general calculus formula is:

$$\begin{aligned} x_{k+1} &= x_k - H_k F(x_k) \\ H_{k+1} &= H_k - \frac{(H_k s_k - y_k) s_k^T}{\| s_k \|^2}, \qquad x_0, H_0 - done \end{aligned} \tag{13}$$

where $s_k = x_{k+1} - x_k, y_k = F(x_{k+1}) - f(x_k)$.
Observation: The method determines two strings $\{x_k\}$ and $\{H_k\}$ so that

$$x_k \to x^*, k \to \infty, H_k \approx F'(x^*)^{-1}$$

Among characteristics we show:

1. A super-linear convergence rate. From tests (start matrix is unity matrix), it results a behavior resembling the Newton method.

2. A reduced calculus effort per iteration. The main operations are the products Hs, HF and $(Hs-y)s^T$, i.e. $3n^2$ operations. The Broyden method does not need to inverse matrix or nonlinear systems solving.

3. The method needs a start matrix H_0 which must approximate $F'(x^*)^{-1}$. Obviously, determining such a matrix is very difficult. When there is no way to evaluate H_0, one can try the unity matrix.

4. The memory requirements are great because it must be retained the non-sparse H matrix. Consequently, the method is appropriate for medium dimension systems.

5. One cannot exploit the sparsity.

2.3.4 Newton - SOR method

The general formula is:

$$x_{k+1} = x_k - \omega[D(x_k) - \omega R(x_k)]^{-1}F(x_k), \quad x_0 - dat \tag{14}$$

where ω is a *relaxing factor*, $0 < \omega < 1$, and D and R are respectively the diagonal and the triangular inferior matrix resulting from decomposing of $F'(x_x)$:

$$F'(x) = D(x) - R(x) - P(x)$$

So,

$$\mathbf{D}'(\mathrm{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & ,\cdots, & 0 \\ \vdots & \ddots & \vdots \\ 0 & ,\cdots, & \frac{\partial f_n}{\partial x_n} \end{pmatrix}, \quad -\mathbf{R}(\mathrm{x}) = \begin{pmatrix} 0 & & \cdots & & 0 \\ \frac{\partial f_2}{\partial x_1} & 0 & \cdots & & 0 \\ \frac{\partial f_3}{\partial x_1} & \frac{\partial f_3}{\partial x_2} & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & ,\cdots, & \frac{\partial f_n}{\partial x_1} & 0 \end{pmatrix}$$

Likewise the Newton method, the inversion of $D - \omega R$ matrix is replaced with the solving of the linear system:

$$\omega(D - \omega R)z = -F$$

then $x_{k+1} = x_k + z$. Note that for the matrix $\omega(D - \omega R)$ being triangular inferior, the solving process is simple. However it is necessary that $\frac{\partial f_i}{\partial x_i} \neq 0$.

The main characteristics are:

1. A good convergence rate, perhaps a super-linear one.

2. The calculus effort per iteration is very reduced. The main operation is to solve the linear system $\omega(D - \omega R)z = -F$, which, having a triangular inferior matrix, is directly solved (no Gauss elimination) and needs $n(n+1)/2$ operations.

3. The relaxing factor ω has a certain influence on the convergence; we can take $\omega = 1$.

4. The disadvantage of the method resides in the condition: $\frac{\partial f_i}{\partial x_i} \neq 0$ which is not always naturally achieved. If the system's equations are explicit (a new restriction) we generally can re-number the equations so that this condition becomes true.

5. When the method may be applied it can be used successfully for medium and great systems.

2.3.5 SOR method - of successive supra-relaxation

One presume that the system must be put in the form: (explicit-diagonal):

$$\begin{cases} x_1 &= f(x_2, x_3, \ldots, x_n) \\ x_2 &= f(x_1, x_3, \ldots, x_n) \\ \vdots & \qquad \vdots \\ x_n &= f(x_1, x_2, \ldots, x_n) \end{cases}$$

In the vectorial form is $x = F(x)$. The algorithm has two steps:

1. $x = F(x_k)$,

2. $x_{k+1} = x_k + \omega(x - x_k)$,

where $\omega \in \Re$ and is called *relaxing factor*. Particularly, for $\omega = 1$ we obtain *the simple iterations method*:

$$x_{k+1} = F(x_k) \tag{15}$$

Observation: The method may be also applied in the case when the system has not a diagonal explicit form but at any step must be solved n equations with a single unknown

$$f_i(x_{k+1}, \ldots, x_{k+1,k-1}, x, x_{k,k+1}, \ldots, x_{kn}) = 0$$

in order to determine the vector x from the step (2), making the computation extremely complicated.

The method characteristics are:

1. The convergence is linear; the convergence conditions are restrictive, the method is convergent only in particular cases.

2. The required explicit form is generally hard to obtain and only in particular situations.

3. On the other side, the iteration is very simple and needs at every step $2n$ additions and n multiplications (excepting the f calculus). The method may be used for high systems having the required form.

4. The relaxing factor ω, cannot be recommended a priori; It has the scope to increase the convergence speed. The user must supply the value and to observe his influence on convergence speed, adjusting it accordingly.

2.3.6 ADI method - the alternate directions

The system has the following form:

$$U(x) + V(x) = 0 \tag{16}$$

where U is a linear application, and V a particular form application so that $V(x) = 0$ is relatively easy to solve.

For example (n=3):

$$\begin{array}{rcl} 2 * x_1 - x_2 + x_3 + & \sin x_1 - 1 & = & 0 \\ x_1 + x_2 - 3 * x_3 & -x_2^2 + 1 & = & 0 \\ \underbrace{x_1 + 2 * x_2 + 2 * x_3}_{U} & + \underbrace{x_3^3 - 1}_{V} & = & 0 \end{array}$$

The current iteration is determined with the next two formulas:

$$rx_{\frac{k+1}{2}} + Ux_{\frac{k+1}{2}} = rx_k - V(x_k), rx_{k+1} + V(x - k + 1) = rx_{\frac{k+1}{2}} - Ux_{\frac{k+1}{2}}$$

The algorithm has two steps:

1. determining an intermediary iteration $x_{\frac{k+1}{2}}$ with the first formula, involving the solving of a linear system $Ax = b$, where $A = rI + U$ and $b = rx_k - V(x_k)$;
2. solving the nonlinear system $F(x) = c$, where $F = rI + V$, and $c = rx - Ux$ (x is determined in the first step).

We will presume that V has a diagonal form and the equation i depends only on the x_i component of x, like in the example above.

2.3.7 The conjugated gradient method

The general computing formula is:

$$\begin{aligned} x_k &= x - aF'(x)^T p, \quad p_0 = F(x) \\ & p_k = F(x_k + bp) \\ & a = \frac{< F(x), p >}{\| F'(x)^T p \|^2} \qquad b = \frac{\| F(x_k) \|^2}{\| F(x) \|^2} \end{aligned} \tag{17}$$

Observation: The method may be used reseting p after every n iterations, i.e. if $k = Mp$ (multiple of p), then p value is forced $f(x_k)$. This reset may substantially improve the convergence speed.

The method characteristics are:

1. The convergence rate is perhaps $r = 1$, but the method has a very good asymptotic error, i.e.

$$\lim_{k \to \infty} \frac{\| x_{k+1} - x^k \|}{\| x_k - x \|} = \rho,$$

then $\quad o < \rho \ll 1$. This fact implies a very good convergence.

When used with reset (i.e. the calculus is remade having as initial iteration the value resulting after a first run), the numerical tests indicates a square convergence if only the reseted iterations are considered, i.e. the string:

$$x_0, x_n, x_{2n}, x_{3n}, \ldots$$

2. The main operation is the product $F'(x_k)^T F(x_k)$, i.e. n^2 operations per iteration. Consequently the calculus effort is reduced.

3. When F is sparse the Jacobian is a sparse matrix, so we can store only the non-zero elements and a column index; in this case the product $F'(x_k)^T F(x_k)$ will need a more reduced calculus effort.

4. The memory requirements are reduced.

2.3.8 The parallel lines method

The general formula is:

$$x_{k+1} = x_k - aF(x_k) \tag{18}$$

Among characteristics we can enumerate:

1. The convergence conditions are restrictive; the methods converge if $F(x^*)$ is a positive defined matrix i.e. the following condition exists:

$$< F'(x^*)x, x > \geq \| x \|^2, \forall x \in \Re^n$$

2. The relaxing factor must satisfy the double condition:

$$a < \| F'(x^*) - I \|^{-2} \qquad a \leq 1$$

   In practice it can be small enough $(0.1, 0.2, etc.)$. The convergence speed is influenced by this factor.

3. The iteration is very simple and the calculus effort is minimum. The method may be used for great system dimensions, eventually to find the initial iteration for another method.

Observation: If $F(x^*)$ if negative defined $< F'(x^*)x, x > \leq \| x \|^2$ the considered relaxing factor must be negative.

2.3.9 Method to solve the bilocal problem

The bilocal weak nonlinear problem is the following: We consider the equation:

$$\begin{aligned} u'' &= f(t,u), \quad t \in [0,1] \times \Re \longrightarrow \Re \\ & u(0) = \alpha, \quad u(1) = \beta \end{aligned} \tag{19}$$

When discretized with finite differences we obtain:

$$Ax + \phi(x) = 0 \tag{20}$$

where

$$\mathbf{A} = \begin{pmatrix} 2 & -1 & \cdots & 0 \\ -1 & & & 0 \\ \vdots & \ddots & \ddots & \vdots \\ 0 & & & -1 \\ 0 & ,\cdots, & -1 & 2 \end{pmatrix} \qquad \phi(\mathrm{x}) = \begin{pmatrix} h^2 f(t_1, x^1) - \alpha \\ h^2 f(t_2, x_2) \\ \vdots \\ h^2 f(t_{n-1}, x_{n-1}) \\ h^2 f(t_n, x_n) - \beta \end{pmatrix}$$

$$h = \frac{1}{n+1} \quad , t_i = ih \quad , x^T = (x_1, \cdots, x_n)$$

To solve the 19 we use the iteration:

$$x^{k+1} = -A^{-1}\phi(x^k) \tag{21}$$

where $x^0 = (x_1^0, \cdots, x_n^0)$ is the initial iteration.
A^{-1} must be make explicit as: $A^{-1} = (a_{ij}^{-1})$ where

$$a_{ij}^{-1} = \frac{1}{n+1} \begin{cases} i(n-j+1) & , i \leq j \\ j(n-i+1) & , i \geq j \end{cases}$$

The method characteristics are:

1. The main operation is the product $A^{-1}\phi(x_k)$.Considering that A^{-1} may be make explicit and that the elements may be make explicit function of indices, the calculus effort per iteration is extremely small.

2. The numeric tests show strong enough convergence conditions, so the method has no great applicability.

3. The method satisfies the principle of independence on discretization step, i.e. for a done precision and for the same initial iteration the number of iterations does not depend on discretization step.

3 NESS

3.1 The expert working mode

The expert activity for solve a non-linear equation system cannot be reduced to a predefined scheme of solving, this process involving complex knowledges in domain and a lot of experience [14].

For solving a non-linear equations system, the human expert follow generally the next steps:

1. the problem analyze for determinate the system properties;
2. the choose of the most suitable numerical method according with the properties already determinated and with the characteristics of the known methods;
3. after the execution of the numerical method, the expert analyze the intermediary and final results;
4. in the case of failure, he try a diagnosis and a rework of solving, by changing some conditions or by changing the method.

From the equations system features we mention:

- the system form: general, sparse, sum between a linear and a non-linear and a diagonal part, explicitly diagonal, special, etc.;
- variational features: linearity degree (non-specific variation, weak non-linear), Jacobian (nonsingular, positive defined, singular);
- dimension: small ($n \leq 10$), medium ($10 < n \leq 50$), big ($50 < n \leq 500$), very big ($500 < n$).

The numerical methods are characterized of: rate of convergence (linear, super-linear, quadratic), domain of convergence (very small, small, sufficiently big, hard to specify), computing effort per iteration (reduced, big), used memory (reduced, big, very big), asymptotically error), initial data (initial iteration, starting matrix, discretization step), the sparse conservation, the dependency of the discretization step, etc.

Starting from the features of the system and of the numerical methods, human expert use domain knowledges (numerical analysis) and heuristics for: choose the most suitable method, interpret the results (intermediary and final), rework the solve in the case of failure.

3.2 The task oriented model

3.2.1 The task definition

A solution for an intelligent combination between different models and methods for problems solving is given by the description of reasoning at a higher level of abstraction.

Newell [21] proposes a *knowledge level*, corresponding to the rationalization of the expert's behavior and a *symbol level* , corresponding to the mechanization of the expert's behavior. Therefore, in conceiving an expert system the accent falls more on *how to solve the problem* and less an *how to implement the system*.

Between approaches concerning the reasoning abstraction we mention: the inference structures (Clancey); KADS (Wielinga); generic tasks and task structure (Chandrasekaran), problem solving methods (McDermott), expertise components (Steels) [31, 2, 29, 4].

A generic task is a structured unit that regroup common features of the tasks which belong to miscellaneous application domains, those domains being related by the same solving process and using the same type of knowledge. A generic task can be regarded as a group : problem - method - knowledges - inference.

Further Chandrasekaran reconsiders the generic task notion introducing the notion of *task structure*[2]. A task models the problem to be solved and it is characterized by an initial context, a goal and a set of methods.

Our *task-oriented model* is based on task structure and allows to make explicit reasoning and control knowledge at different levels of abstraction [17, 16]. A task is an elementary component of reasoning and represents a knowledge granularity level.

A task is a structured entity gathering all the information involved in a problem resolution, whatever the abstraction level or structural complexity, in the form of a *goal/context/processing* association [2, 23, 18].

The *context* is referring to different abstraction levels (general knowledge, specific knowledges, particular cases). *Processing* includes all the methods which allow to realize the task. A method represents a solving strategy and can be a computing routine, an action plan, a knowledge base (a rule base), a task chaining, a task base with an own control mechanism, an user, etc. The methods are represented as well at different levels of abstraction.

A task is characterized by a goal or a subproblem and of one or several solving methods (figure 1).

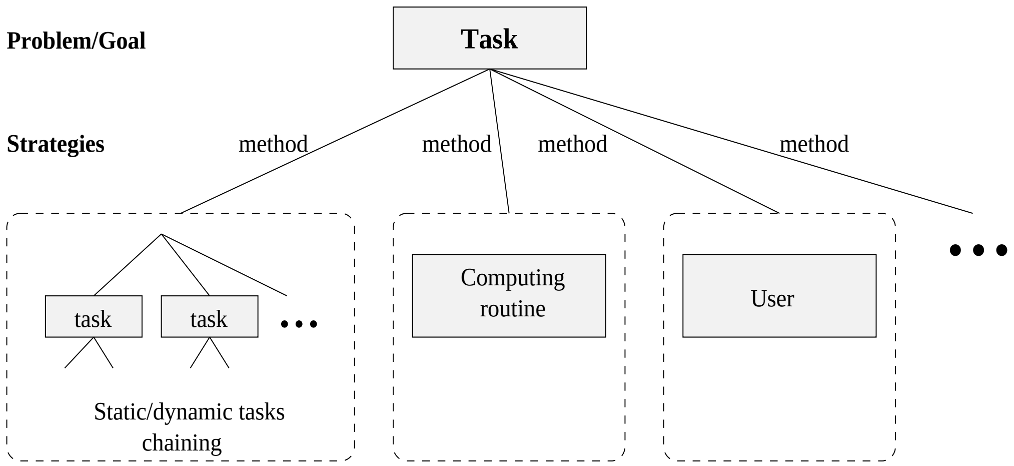

Figure 1: Task - methods relationship

Tasks are organized on several levels according to the problem degree of abstraction or the problem complexity. A high level task is more supple (modeling, analyzing, simulating, identifying, classifying), while a low level task is more rigid (Jacobian calculus, reversing matrix, symbolical derivative). The refinement is done from high levels to low levels.

The tasks chaining can be static (chaining defined using elementary control structure) or dynamic (chaining computed during the resolution, according to a specific context or strategy).

Choosing a suitable method (strategy) to accomplish a task is not an easy operation. It depends on objects involved in the task and on conditions for the good work of the method. For example, the choice of a method to realize the task "reverse a matrix" can be done through a classifying operation on the tree structure of the possible methods depending on the implied matrix properties.

We will use task-graphs (in particular: task-trees) to represent a reasoning scenario. A task-graph is a static model of solving the problem (usually described in a language of subproblems) and it is a first approximation of the reasoning.

The root node has the role of an initial task and the leaves correspond to elementary tasks (tasks associated to methods like: calculus routines or rules bases). Methods associated to intermediary tasks may be sequences of tasks or selection structure of alternative tasks. Tasks management is done by means of tasks agenda. An interpreter (inference engine) dynamically controls the agenda.

3.2.2 Clips extension of task model

We tried to map the concept of tasks as described above on the CLIPS rule based knowledge system and it's modularization and OOP concepts [18].

A first CLIPS concept used was the notion of *module*. A module is a denominated separate context which can be used to implement some of the task's components. We mean the precondition, postcondition and task body. One CLIPS module can have its own classes and instances which enter in the inference mechanism with the module's rules. These rules are the decision structures which determine the actions to be performed. The structures in a module can use local templates (and facts) in order to perform communication between various task components, as well as global templates in order to pass data to other tasks (or task components).

The global templates (or exported templates by a module) may be used to implement the procedural aspects if needed.

Using the CLIPS COOL facilities (OOP part of CLIPS), we have designed a task class, which includes: precondition, postcondition, taskbody and subtasks together with a method to activate the task.

The *precondition* part consists of a set of rules which may decide over the execution of the task. Some tests can be made here in order to establish the opportunity of running the task. The *postcondition* part may be another set of rules which decides over the results of the task. The results may be considered as valid or not, depending on the reasoning procedures located here. This model of task allows to simulate a backward chaining reasoning in the frame of forward chaining CLIPS inference engine.

We also designed a *task-monitor* entity whose purpose is to be a manager of all the existing tasks. This manager is on object of a dedicated class which has as members a list with all the tasks and a stack used by the monitor to activate in the right order the tasks. The *stack* will contain in every moment the name of tasks and will activate the task in the top of the stack. The activation of a task involves to put in the stack all the subtasks and to change the focus in the modules stack (CLIPS internal) accordingly to the characteristics of the task being activated. For example, when a task is activated the postcondition module (if it exists) is placed in the focus stack. Then the taskbody module is placed and then the precondition module.

If the subtask list exists, all the tasks here will be placed in the *task-monitor*'s stack.

A task-monitor object has several methods used to manipulate tasks (make-task, pop-task, push-task etc).

The *taskmonitor* structure and the task classes and objects are created inside a dedicated CLIPS module. This module get the focus after any activation of a task. This is done by placing the module in the focus stack at the right place. This way we are sure that the tasks machine will have continuity and the tasks stack in the *taskmonitor* is checked in order to activate the next task. A simple loop rule which activate the task in the top of the taskmonitor's stack can do this.

3.3 NESS implementation

NESS was implemented in CLIPS, a programming environment for developing expert systems, created at NASA [3]. CLIPS allows for multiple knowledge types: rule-based, procedural and object knowledge.

The graphics interface was developed in WxWindows [28] a GUI C++ class library, available in both UNIX and MSWindows environments so it is an attractive solution for a GUI portability. The provided class hierarchy allows the programmer to build a graphical interface with a minimum of effort.

To solve a non-linear system of equations, we will use a model which simulates a human expert's reasoning. The model id similar to the solving paradigms *Generate, Test, and Debug* (GTD) [27]. In our case we will use a meta-reasoning level which suppose the following stages: construction, instantiation and diagnosis [17].

Construction the reasoning means finding a chaining of tasks and their associated methods, which - at the level of elementary tasks - is nothing else than a sequence of calls to calculus routines or inference mechanisms (classification, inductive or deductive rule-based mechanisms, etc.). As result of this stage a graph of elementary methods is obtained.

Instantiation means the realization of the formerly obtained tasks chaining (more exactly: the execution of a path in the graph of elementary methods).

Diagnosis refers to an analysis of the intermediary and final results. It also means - in the case of failure - establishing the causes of the failure (by use of causal knowledge) and reworking the reasoning.

Automatic call of calculus routine is performed at the level of elementary methods. An elementary method comprises several nested levels: description of the type of problem, of the input data, of the constraints on the input data, of the expected results and constraints on them, side effects, cost; conversion of the expert system's data into data required by the calculus routine, of the results and handling errors; dispatch of calculus

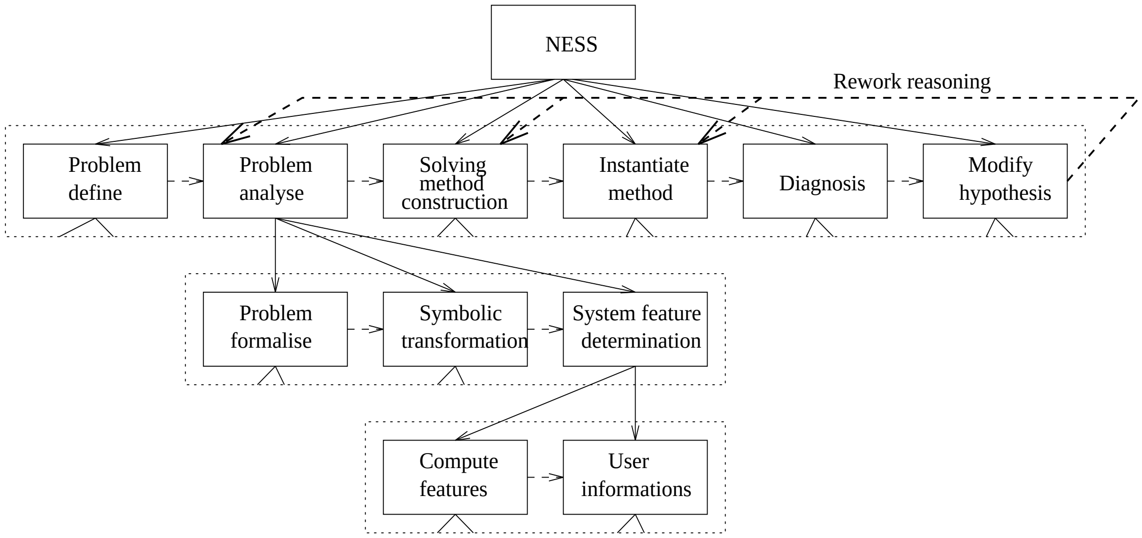

Figure 2: A partial tasks tree of NESS application

routine (dynamically loaded and execution code, fork process or batch process).

Domain knowledge (methods; mathematical entities: equations, variables, operators, functions, matrices) and control knowledge (tasks) are represented as hierarchies of classes (in Clips object oriented language - COOL)), while the human expert's knowledge are represented using rules (in Clips rule based language).

The (partial) tree of tasks is shown in figure 2.

The user describes the problem in a mathematical language while, internally, the equations are represented using the polish inverse notation (for transformation and symbolic derivations) and C is used to construct executable code.

NESS [19, 18] is able to determine the properties of the system of equations, based on the problem's description, and to propose candidate numerical methods based on the characteristics of about twenty (which variants) implemented numerical methods.

If a method fails, the system (NESS) reworks the reasoning with (some) modified parameters (discretization step, number of iterations, precision), a modified initial iteration or another numerical method (or combination of numerical methods). For example, if the conjugated gradient method fails, NESS can try to solve the problem with the Steffensen method using the last iteration of conjugated gradient method as initial iteration.

The symbolical transformations refer to: the reducing of the terms of equations, the reducing of the number of equations, derivative computing and the explicitly diagonal form obtaining. Also it is possible to obtain the sum form (a linear and a non-linear part).

We can find the problem of solving non-linear equations systems within some mathematical software packages : Maple, Mathematica, etc. As a remark, there are used one or two general solving methods, methods which fail in some special cases (bad conditioned systems, degeneration, etc.).

Tests were carried out on systems of ill-conditioned equations (degenerate Jacobian of $x = x^*$) and on sparse systems. Small and medium sized systems were considered. In the most cases the solution could not be reached by Maple, while Mathematica encounters

Methods	System form	Linearity degree	Jacobian	Dimension
Classic Newton	general	certain	$\neq 0$	small
Discretized Newton	general	certain	$\neq 0$	small
Gen. Simpl. Newton	general	certain	$\neq 0$	medium
Parallel Lines	general	certain	$>> 0$	big
Secant	general	certain	$\neq 0$	small
Steffensen	general	certain	$\neq 0$	small
Broyden	general	certain	$\neq 0$	medium, big
Friedman	sparse	certain	certain	big
Conjugated Gradient	sparse	certain	certain	big
Simple SOR	sparse, explicitly diagonal	weakly non-linear	$>> 0$	medium, big
General SOR	sparse	weakly non-linear	certain	big
ClassicADI	sum	certain	certain	medium, big
ADI Kellog	sum	certain	certain	medium
Specials				

Figure 3: Correspondence: system properties - methods

Methods	Rate of conv.	Domain of conv.	Comp. effort/ iteration	Used memory	Initial data
Classic Newton	$r = 2$	small	big	big	initial iter. (it.)
Discretized Newton	$r \approx 1.6$	small	big	big	it., discret. step
Gen. Simpl. Newton	$r = 1$	hard to specify	medium	medium	it.
Parallel lines	$r = 1$	hard to specify	reduced	very small	it.
Secant	$r = 1.6$	small	big	big	it.
Steffensen	$r = 3$	very small	big	big	it.
Broyden	$r > 1$	small	reduced	medium	it., start matrix
Friedman	$r = 1$	sufficiently big	reduced	small	it., discret. step
Conjugated Gradient	$r > 1$	sufficiently big	reduced	small	it., discret. step
Simple SOR	$r = 1$	hard to specify	reduced	small	it., relax. factor
General SOR	$r = 1$	hard to specify	reduced	small	it., relax. factor
Classic ADI	$r = 1$	hard to specify	medium	small	it.
ADI Kellog	$r = 1$	hard to specify	medium	small	it.
Specials	$r > 1$	hard to specify	reduced	small	it.

Figure 4: Numerical methods characteristics

the same difficulty for large systems.

Our expert system NESS finds the solution in all the above mentioned examples using Newton or gradient type methods. When both Mathematica and NESS find the solution the elapsed times are comparable [19].

4 Conclusions

NESS is not an exhaustive system from the point of view of the methods and procedures it offers. Effort should be made to enrich its methods base by adding new high performance methods.

In order to reduce the solving time, besides adding heuristic and domain specific knowledge for choosing the appropriate numerical methods, one should consider: the use of efficient data structures and access primitives in the case of sparse systems as well as parallelization of the existent numerical algorithms.

Among the open numerical problems one may consider: the determination of the initial iteration, which is related to the determination of the domain of convergence for the iterative method; the determination of the number of solution and their separation, which depends on the topological degree of the function on the left side of the system.

The use of same multi-agent distributed models [30, 17, 26] will allow a concurrent cooperation of numerical methods in the process of searching the solution. We have started to develop a multi-agent architecture intended to be used for problem solving (especially complex scientific problems) and to adapt NESS to this architecture [20]. The solution of the problem (the achieving of the goal) is reached by cooperation between agents. Our study will refers to a model of cooperating agents involving expert agents which use a task oriented reasoning.

5 Acknowledgments

This research has been partially supported by Romanian Ministry of Education and by World Bank (grant nr. 21/1998 - D.G.I.S.C.S.U. and grant nr. 12623/1998).

References

[1] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. Sacks, G. J. Sussman, J. Wisdom, and K. Yip. Intelligence in scientific computing. *Communications of the ACM*, 32(5):546–562, may 1989.

[2] B. Chandrasekaran. Design problem solving. A task analysis. *AI Magazine*, pages 59–71, winter 1990.

[3] C. Culbert, G. Riley, and B. Donnell. *CLIPS reference manual, Volume 1, 2, 3*, 1993.

[4] J. M. David. Les systèmes experts de second génération ou de l'importance de la modelisation dans la construction de systèmes à base de connaissances. *Technique et science informatiques*, 14(4):435–471, 1995.

[5] J. E. Dennis. On the convergence of single-rank quassi-Newton methods . *Math. Comp.*, 24:365–382, 1970.

[6] J. E. Dennis and J. J. More. Quassi-Newton methods, motivation and theory . *SIAM J. Numer. Anal.*, 19(1):46–89, 1977.

[7] V. M. Fridman. Iterative method with minimal error for non-linear operatorial equations. *Acad. Nauk, Russia*, 139:1063–1066, 1961.

[8] W. B. Gragg and R. A. Tapia. Optimal error bounds for the Newton-Kantorovici theorem. *SIAM J. Numer. Anal.*, 11(1):10–13, 1974.

[9] N. Jacobstein, C. T. Kitzmiller, and J. S. Kowalik. Integrating symbolic and numerical methods in knowledge-based systems: current status, future prospects, driving events. In J. S. Kowalik and C. T. Kitzmiller, editors, *Coupling symbolic and numeric computing in expert systems*, pages 3–11. Elsevier Science Pbs., 1988.

[10] M. S. Kamel, W. S. Ma, and W. H. Enright. ODEXPERT: An expert system to select numerical solvers for initial value ODE systems. *ACM Transaction on Mathematical Software*, 19(1):44–61, March 1993.

[11] R. B. Kellog. A non-linear alternating direction method. *Math. Comp.*, 23:23–27, 1969.

[12] C. T. Kitzmiller and J. S. Kowalik. Symbolic and numerical computing in knowledge-based systems. In J. S. Kowalik, editor, *Coupling symbolic and numeric computing in expert systems*, pages 3–15. Elsevier Science Pbs., 1986.

[13] J. S. Kowalik, editor. *Coupling symbolic and numeric computing in expert systems*. Elsevier Science Pbs., 1986.

[14] S. Maruster. *Numerical methods in nonlinear equations solving*. Editura Tehnică, 1981.

[15] St. Maruster. On the two-step gradient method for nonlinear equations. *Anal. Univ. Timisoara*, 25(3):69–84, 1987.

[16] St. Maruster, V. Negru, and Sandru. CLIPS - Intelligent front-end for non-linear equations solver. In *International Symposium ROSYCS'98, 11th ROmanian SYmposium on Computer Science*, Iasi, May 28-30, 1998.

[17] V. Negru. Polymers characteristics determination using a blackboard model. In P. Jorrand and V. Sgurev, editors, *ARTIFICIAL INTELLIGENCE: Methodology, Systems, Applications*, pages 291–297. World Scientific, September, 1994.

[18] V. Negru and L. Maruster. Non-linear equations systems solver. In A. M. Ramsay, editor, *Artificial Intelligence: Methodology, Systems, Applications; Frontiers in Arificial Intelligence and Applications series*, pages 330 – 339. IOS Press, 1996.

[19] V. Negru, S. Maruster, L Maruster, M. Marin, and C. Sandru. Intelligent system for solving simultaneuos nonlinear equations. A CLIPS implementation. Research report, Theoretical Informatics Department, Timişoara West University, October 1995.

[20] V. Negru, C. Sandru, and D. Pop. Task-oriented multi-agent system. In *HUNABC 98, 1st Hungarian Conference on Agent Based Computing*, Budapest, May 29-31, 1998.

[21] A. Newell. The knowledge level. *Artificial Intelligence*, (18):1–20, 1981.

[22] J. M. Ortega. Stability of difference equations and convergence of iterative processes. *SIAM J. Numer. Anal.*, 10, 1973.

[23] C. Pierret-Golbreich. Vers une nouvelle architecture pour l'explicitation du raisonnement au niveau connaissances: task. Rapport de recherche 09/674, Université Paris-Sud, Orsay, 1991.

[24] F. Rechenmann. Modeling tasks and methods in knowledge based PDE solver. 1991.

[25] M. F. Russo, R. L. Peskin, and A. D. Kowalski. Using symbolic computation for the automatic development of numerical programs. In J. S. Kowalik and C. T. Kitzmiller, editors, *Coupling symbolic and numeric computing in expert systems*, pages 77–89. Elsevier Science Pbs., 1988.

[26] C. Sandru and V. Negru. A distributed task-oriented problem solving system. In *International Workshop "Distributed Artificial Intelligence and Multi-Agent Systems" (W-DAIMAS' 97)*, pages 193–203, 1997.

[27] R. Simmons. Generate, Test and Debug: a paradigm for combining associational and cauzal reasoning. In J. M. David, J. P. Krivine, and R. Simmons, editors, *Second generation expert systems*. Springer Verlag, 1993.

[28] Julian Smart. *Reference manual for wxWindows 1.50: A portable C++ GUI toolkit*, July 1993.

[29] L. Steels. Components of expertize. *AI Magazine*, summer 1990.

[30] S. N. Talukdar, E. Cardozo, L. Leao, R. Banares, and R. Jacobbani. A system for distributed problem solving. In J. S. Kowalik, editor, *Coupling symbolic and numeric computing in expert systems*, pages 59–65. Elsevier Science Pbs., 1986.

[31] B. Wielinga and G. Schreiber. KADS: Model based KBS development. In H. Marburger, editor, *GWAI-90, 14th german workshop on artificial intelligence*, pages 322–333. Springer-Verlag, 1990.