Записки научных
семинаров ПОМИ
Том 258, 1999 г.

**Stefan Măruşter, Viorel Negru,
Dana Petcu, and Călin Sandru**

INTELLIGENT FRONT-END FOR SOLVING DIFFERENTIAL AND NON-LINEAR EQUATIONS SYSTEMS

ABSTRACT. `INTENSE` (INTElligent Non-linear Systems of Equations solver) is a problem-solving environment that provides the computational facilities necessary to find a numerical solution for two classes of problems: ordinary differential equations solving and non-linear algebraic simultaneous equations. `INTENSE` proposes facilities like a friendly user interface for problem specification, the automatic detection of the problem properties, a friendly user interface for the methods specification, the automatic detection of the method properties, a matching mechanism between the problem properties and the method properties, apply the numerical method and supervise the error, some lists or graphics of approximate solution values, solve the problem without the specification of a particular method.

1. INTRODUCTION

The volume and diversity of the numerical software available for solving numerical mathematical problems has become today a problem. When confronted with many software products, it is important for the user to have some knowledges of numerical software and to know where this software can be optimally applied in order to be able to choose the most appropriate software for a specific problem [4, 1].

In this paper, we restrict our area of interest at solving *ordinary differential equations (ODE)* and *non-linear equations systems (NES)*. The basic concepts beyond INTENSE will be shortly presented.

`INTENSE` (INTElligent Non-linear System Equations solver) is a problem-solving environment that provides the computational facilities needed to find numerical solutions for two classes of problems:

1. ordinary differential equations systems solving (ODE):

$$\begin{cases} y'(t) = f(t, y(t)), \\ y(t_0) = y_0 \end{cases} \tag{1}$$

where $t \in [t_0, t_0 + T],\ y_0 \in \mathbb{R}^n,\ f : [t_0, t_0 + T] \times \mathbb{R}^n \to \mathbb{R}^n$.

2. Non-linear algebraic simultaneous equations:

$$F(x) = 0, \tag{2}$$

where $F : D \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}^n$.

Unfortunately, there is not a unique recipe in solving these kind of equations systems. To choose a particular method one should take into account several aspects like the form of the system, structural characteristics, the precision, the memory space, the convergence speed, etc. All these factors must be considered when constructing a numerical software to solve such problems [5, 7].

2. The Available Mathematical Software

We can classify the existing software into several major classes.

General purpose tools (for example, the Computer Algebra Systems like `Mathematica, Maple, Magma, Reduce, Matlab, Octave, Hi-Q, Mafia, Mlab, SymbMath`) have included a mechanism for generating an analytical or numerical solution of mathematical problems , which, unfortunately, are not usefully for any kind of ODEs or NESs. On another hand, the number of equations cannot be too large. To design a CAS for any mathematical problem is a very hard task. The base idea of an actual CAS is to construct a kernel which solve any simple problem and to use special library functions and procedures for more complicated problems.

These packages of dedicated function can be seen as problem dedicated software.

Dedicated programs (for example `Biograph, Chaotic Dynamics Workbench, Chaotic Mapper, Depad, Differential Systems, Mdep, Ode, Ode Workbench, Phaser`) are almost all designed with a friendly user interface. Unfortunately, these tools have been constructed with some fixed method databases which are properly for specific classes of problems with a small number of equations.

Method libraries (like `Odepack` or `Desir`) can be used in the integration of a wide variety of large systems of equations if the user knows which method must be selected for his problem. The method database can be extended, but the user must have some experience with the programming languages.

Problem solving environments (like `Odexpert` [5] project and also the software presented in this paper) have the advantage of exploiting a knowledge database in order to select an appropriate method for a specific problem. Large systems of equations of different kinds can be integrated using these tools. The user interface is generally designed in such a way that it is possible to define a problem within a certain scientific or technical context. After the problem has been specified with sufficient accuracy, the program decides which subsystem or subprogram is to be used to solve it. The selection mechanism ranges from simple decision trees to complicated expert systems whose knowledge base come from specialists in the particular field. When the internal solution mechanism has produced results, the problem solving environment put them into a form which allows the user to interpret and use them. Some support in the form of help functions is offered if the user needs assistance.

3. ODE Problem Description

The formal description of the problem is done by (1). We consider the problem of numerically solving the differential equation system, determining the values of the solution function $y : [t_0, t_0 + T] \rightarrow \mathbb{R}^n$ at a number of points from the interval $[t_0, t_0 + T]$.

There are various methods ranging from more general to system fitted methods. No method can be declared the best, in the sense that it solve any differential system in the shortest time at the highest level of solution accuracy.

The most used methods for the numerical solution of (1) are the `Runge-Kutta` methods and the linear multi-step methods (in our software some other methods have been take into account – like nonlinear methods, second derivative linear multi-step methods hybrid methods etc.). The difference between particular methods is due to the method properties (dictating, finally, the applicability of a method to a problem). We mention here some of the main properties.

Method type.

First of all, a numerical method for ordinary differential equations can be explicit or implicit. An explicit method is like an iterative process in which on one side of the iterative equation is the unknown variable and on the other side are the known variable. An implicit method can be applied only in conjunction with an implicit equation solver (which kind of such solver can be used depends of the type of the differential system).

Another classification make distinction between the multistage or onestage methods. An multistage method describes an iterative process in which there are some intermediate values of the approximate solution which are used only for computing a special value of the approximate solution and which must not be stored.

One-step or multi-step method is another problem. A multi-step method needs some starting procedure (for example an one-step method) for the first iterative steps. The starting procedure has a great influence on the convergence of the approximate solution, computed by a multi-step method, to the exact solution.

An multi-derivative method, in opposite to an one-derivative method, needs an special procedure to compute the exact or approximate derivatives of the function specific for the given problem.

Method order.

The approximate solution accuracy can be estimated using the method order and the method error constant. Knowing the desired accuracy of the numerical solution, we can compute the running parameters of the computer procedure associated with the numerical method. Examples of such parameters are the step size, the estimated number of steps, the estimated computational time and so on.

The error constant make a distinction between the methods with the same order. The method with the lowest error constant is preferred.

Method stability.

The convergence of the approximate solution to the exact solution when the stepsize of the method decreases to zero depends on the method order (which is an expression of the rate of convergence) and on the method stability. The problem of method stability was very much discussed in the literature. Methods have been found which are zero-stable, A-stable, B-stable, stiff-stable, A_0-stable, algebraic-stable and so on. The most important remains the zero-stability, a property which must be encountered to each numerical method for differential equations. Unlike the method order which is independent of the problem, the property of zero-stability make a link between the method and the problem properties via the eigenvalues of a linear system associated with the initial differential equation system. A stability region must be detected and if each product between the stepsize and a such eigenvalue is inside in this region we expect that we get an approximate solution. Otherwise, more probably that the error produced between the approximate and the exact solution will grow unlimited sometime in the computational process.

Applicability to special kind of ordinary differential equations.

Unfortunately, the stability and the degree of accuracy of a method are insufficient informations in order to be sure that the computational process of the approximate solution will run without any inconvenient. For example, if the method has a bounded stability region, and the differential system is so that the above mentioned eigenvalues are very large, the estimated computational time will be unreasonable.

Our expert system was designed to recognize some special problem properties like that of *stiffness*. In this case only implicit methods are recommended to be applied. Special methods have been constructed to solve stiff systems (see, for example, the methods from [33]).

Separability is another problem property which can be fruitfully exploited. Different methods can be applied to distinct subsystems function of the subsystem properties.

Solving *large systems* of differential equations is another big problem. Special computer techniques like using parallel computers or workstation networks have been developed in the last years (for example, see [34]). Compared to parallel methods in the field of numerical linear algebra, there exist only few parallel procedures for the practical solution of initial value problems (IVPs) of systems of ordinary differential equations (ODEs). This fact can be explained primarily by the inherently sequential nature of any forward step method for the numerical solution of such problems.

Parallelism can be exploited in many ways. For example, if solutions are needed in real time, if there is a large period of integration, if parameters fitting is to be performed – which requires repeated integrations, or if function evaluations are expensive. Another source of parallelism arises if the problem to be solved is stiff. If a problem is stiff and nonlinear, the explicit methods are generally not suited because of unnatural restrictions on the stepsize and so implicit methods must be used, that means a lot of computational time. In particular, the method of lines applied to parabolic partial differential equations usually results in a stiff system if the spatial grid is moderately fine-grained.

Massive parallelism in the field of differential equations either requires a priori knowledge of a special structure of the respective problem or a great number of redundant calculations. One-step and straight multistep methods have little potential for parallelism and multi-value methods have more to offer (as it was demonstrated in [34]). Most of the work to date on the parallel solution of IVPs for ODEs concentrates on developing potentially useful numerical schemes, rather than their effective implementation, comparisons of methods, or the development of reliable, robust, and (it is hoped) portable mathematical software.

The means of archiving parallelism in IVP solvers can be classified into three main categories:

(1) *parallelism across the system* – the possibility of partitioning the system of ODEs by assigning one single equation, or a block of them, to each processor for concurrent integration;

(2) *parallelism across the method* – the possibility of distributing the computational effort of each single integration step, or block of steps, among the various processors;

(3) *parallelism across the time* – even through it contradicts the intrinsic sequentially of the problem, the possibility of concurrently executing the integration over a certain number of successive time steps. All these three categories are represented in our software product.

Moving from multiprocessor super-computers to a network of workstations is a new trend in parallel computing. Using some software products, like `PVM (Parallel Virtual Machine)`, the designer of a tool for ODEs can easily adapt his code to the parallel or distributed computing system.

Method selection. Many software applications were constructed in the idea to help the practicians, like engineers, in solving real problems without consulting a mathematician (instead an "expert" application). The problems which are very simple can be easily solved with these applications. In the case of initial value problems of ordinary differential equations, there are two main categories: for which we can construct an analytic solution and those for which we do not know how to construct an analytic solution. In the first case, we can obtain a function which represent the exact solution, using one of the actual Computer Algebra Systems. A such software application does symbolic computations in order to solve a given problem. If the problem is simple, probably, the system will recognize very fast the exact solution. Otherwise, we must apply some transformations to the equation system, or we must use dedicated library procedures which are not in the system kernel. If we failed in obtaining an analytic solution (using the computer or a human expert), we must try to approximate the exact solution. The number of *numerical methods* which we can select in order to due this task is very large. The main question is *which one is adequate to the given problem.* The answer depends on what *quality level* we request for the approximation, the *time* in which we want to obtain the approximation, and the *knowledge* of the solving method class. There are many schemes to decide the better way to solve a given problem. The simplest way is to use a solving scheme with an arbitrary constant method, an arbitrary constant stepsize, and no control on the computation time (choose a method, select an arbitrary stepsize, test if the computation process reaches the interval end, otherwise take a smaller step, stop the computation process and take a greater step if the computation process is too slow, after a number of failures, try another method). One of the hardest task in constructing an automatic expert is to simulate the human capacity to learn from practical experience and to add new knowledge – in particular, to know for a specific problem which method will be better to use.

Most of the numerical software products used for ordinary differential equations are constructed with a fixed method database including the most used numerical methods, like *Euler*'s method or standard *Runge-Kutta* method. Simple problems can be solved using these applications. Troubles come when we want to solve problems with many number of equations or which cannot be solved with the classical methods. In the case of a software which is open to new methods, the user must pay attention to the application language in order to describe correctly the numerical method and the control procedure of the computational process. The application for describing the new method must be closed to the mathematical language in which are usually described the numerical methods.

A hard problem is the *selection of an adequate method* from a method database. A correct selection can be done only knowing the *problem properties*. The method database must be organized in method classes depending on the properties of the problems for which the methods are working.

4. NES Problem Description

The formal description of the problem is done by (2).

We consider the problem of numerically solving the nonlinear equation system, determinating $x^* \in D$ such as $F(x^*) = 0$.

There are various methods to do that ranging from more general to more particular methods. No method can solve any system and there are systems which cannot be solved using known methods.

From the numerical solving methods, we have selected some classical ones with high performances (*Classic Newton, Discretizated Newton, Secant, Steffensen, Parallel Lines*, etc.) or others relative recently developed(*Broyden, Friedman, Conjugated Gradient with Reeves Bal-*

anced Factor, Successive Over-relaxation, 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 the weakly nonlinear two point boundary problem).

We will present bellow several issues we have considered in the solving process.

In the following we denote by $F'(x)$ the Jacobian of F in the point x and x^* denote a solution of the system in D.

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, using some specific particularities of the problems and to get enhanced performances by these means.

In the next sections we will refer to four characteristics of the NES problem.

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, 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 [24]; 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 iteration. In general, the methods that use the Jacobian (Newton, gradient and others) need a number of function evaluations of the order 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 inverse 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 [25].

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, for example, the methods of gradient type.

The most important processing of each iteration of these methods is the product $F'(x_k)^T F(x_k)$ where x_k is the current iterate. 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 linear; this means that, for a given precision, we need a big number of iterations, that is a great overall computing effort.

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 $detF'(x^*) \approx 0$). 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) 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 it 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 boundary conditions, the weakly nonlinear two point boundary value problem, and so on.

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 given a rectangular domain, etc.

- **Finding the initial iteration**. In essence, it comes to the estimation of convergence domain [29], 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) [30].
- **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 degree 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.

The methods considered in our work are iterative methods with one step (only one method which make exception is the secant method, a method with two steps), having the form:

$$x_{k+1} = \Phi(x_k), \quad x_0 - \text{given}, \Phi - \text{function of iteration} \tag{3}$$

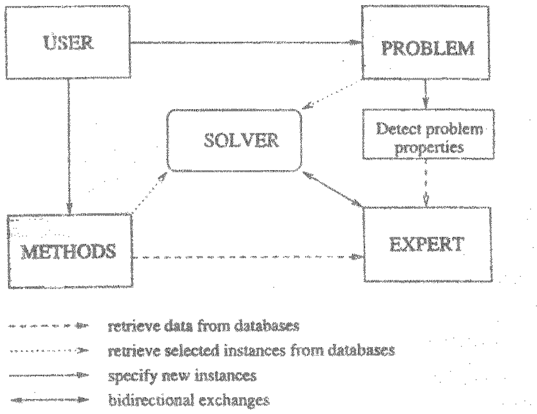
To choose a particular method one should take into account several external aspects: the form of the system, knowledges about a "good" iteration, structural characteristics (the quality of the Jacobian, condition number), etc. Other elements who influence the solving process are: the precision, the memory space, the convergence speed [11].

A correspondence between the system properties and methods is illustrated in the Table 1.

Methods	System form	Linearity degree	Jacobian	Dimension
Classic Newton	general	certain	$\neq 0$	small
Discretizated 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
ClasicADI	sum	certain	certain	medium, big
ADI Kellog	sum	certain	certain	medium
Specials				

5. Intelligent Front-End

The figure 1 offers a general view over INTENSE environment.



The arrows specify diffrent types of dependencies and interactions between modules.

The user specifies the problem and eventually a method to solve the problem. Then the problem is analyzed and its properties are discovered. From the methods database and based on problem properties, the expert is able to select the appropiate methods to solve the problem. One of this methods is activated in the solver module and the expert will supervise the method execution. A more detailed view over the expert is provided in the section.

INTENSE proposes the following facilities:

1. friendly user interface for problem specification; Such a interface comprise a modality to enter the problem input in a natural manner so that a non-specialist user may enter his problems without knowing special languages.

2. automatic detection of the problem properties; This is done by implementing a matching mechanism between the methods properties and the system properties. The expert can then suggest a list of adequate methods to be tried by the user or to be automatically executed.

3. friendly user interface for the methods specification; Apart from the problem specification, the user is allowed to specify a method to be used in the solving process. The new method is optionally saved in the method database.

4. automatic detection of the method properties; The method properties for ODE are: (a) explicit/implicit schema, (b) method accuracy, (c) stability properties, (d) one-step/multi-step, one-stage/multistage, one-derivative / multi-derivative method, (e) estimated time for solving the test problem $y' = \lambda y$ with $\mathrm{Re}\lambda < 0$; The methods for non-linear equations system are characterized by: (a) rate of convergence (linear, super-linear, quadratic), (b) domain of convergence, (c) computing effort per iteration, (d) used memory, (d) asymptotically error, (e) initial data (initial iteration, starting matrix, discretization step), (f) the sparse conservation, (g) the dependency of the discretization step, etc.

5. establish: (a) matching between the problem properties and the method properties, (b) adequate integration step, (c) approximate computation time, (d) estimated error;

6. apply the numerical method and supervise the error;

7. create a list/graphic of approximate solution values (two or three-dimensional graphs);

8. solve the problem without the specification of a particular method: (a) choosing an appropriate method from a database of classical methods (the selection is based on a classification of the method, and the database can be enlarged introducing new methods using the above mentioned front-end); (b) select a new method when the previous one generates unreasonable errors; (c) in the case of a large number of equations use some special numerical codes [34] with PVM procedures (Parallel Virtual Machine) in the idea to distribute the computations on some processors of a local network. 9. if a method fails, the system (INTENSE) redo 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).

6. The Expert

The expert is the component which copes with the intelligent aspects of the solving process rather than the mathematical aspects. INTENSE is intended to assist the user in solving ODEs or NESs, so there is a need for a module able to perform some kind of special tasks as illustrated bellow:

1. The user describes the problem in a mathematical language while, internally, the equations are represented using the polish inverse notation (for symbolic transformations and derivations) and C is used to construct executable code. 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). It is the expert's task to *couple* the symbolic data with numeric data in order to represent knowledges and reason about them.

2. Being done an equations system, the expert must be able to select a method for solving it from the methods database using the system properties and the properties of methods. A matching mechanism is provided in order to accomplish this task.

As we said above, the "keyword" here are the *properties*. A **property** is an information about the system being solved or about one method. The methods properties may have the same values as the system properties and additionally they may refer to some other kind of requirements like speed requirements, the use of the symbolic calculus, numerical precision and so on.

For ODE problems we mention several properties: (a) linearity, (b) separable, (c) symbolic computation of the Jacobian matrix, (d) the greatest and the lowest eigenvalues of the Jacobian matrix at the initial value; (e) stiffness ratio, (f) estimated time for the evaluation of the right side of (1);

For the non-linear equations systems we mention: (a) the system form: general, sparse, sum between a linear and a non-linear and a diagonal part, explicitly diagonal, special, etc.; (b) variational features: linearity degree (non-specific variation, weak non-linear), Jacobian (nonsingular, positive defined, singular); (c) the dimension.

The methods characteristics and properties are hard coded in the expert source. For every new system some auxiliary routines compute the system properties and inform the expert about their values.

3. The expert is able to update methods base with user's new methods. This feature assures a kind of learning mechanism and in this way the system becomes more attractive.
4. Once a method selected, the expert must provide it all the initial data required for that particular method. We include here an initial iteration and eventually the needed auxiliary parameters. Additional computing steps may be required here.
5. The expert is intended to supervise the execution steps of the selected method. This means that some static data must be maintained into the expert which allow the expert to reason about the current state

of solution. This way, it can stop the current method, decide if the current solution is good enough or start a new method. A choice here could be to choose new initial data for the same method and restart it (in the NES case).

6. Sometimes, the user may provide useful informations about the system characteristics and, eventually a kind of indications about how these can be used. The expert must be able to benefit from this data and to integrate new facts in its knowledge base.
7. Another expert's goal is to offer details to the user about the motivations to select a particular action. The explications are useful for advanced users which can decide if the expert reasoning is right and if it needs some adjustments.

7. Conclusions

The INTENSE problem solving environment is intended to assist the user in the process of solving ODE and NES problems. We propose a system having the ability to process the input data and to generate some knowledges about the problem maintained at the level of an expert module. Based on these knowledges the solving process is directed by the user or by the expert. All the system capabilities are designed to make INTENSE an easy to use and user friendly environment.

In the actual state of the project implementation the ODE solver (EPODE) [5] and NES solver [11] are separately implemented. The solvers are implemented in C/C++ on the Windows OS. CLIPS [22] is the expert system shell we have chosen to implement the expert, although the ODE expert is made in C for now.

The next step is to integrate the two existing environments into a fully functional one. The parallel solving approach must be also extended to the NES part since it is already implemented for the ODE solver. More refinements are to be done at the expert level and research to identify new aspects allowing the expert to generate an improved reasoning.

We also think at another direction of Intense expandability concerning the reusing of already existing CAS procedures. There are a lot of CAS which implements solving procedures for ODE and NES problems. The idea is to benefit from this systems using their methods. This job involves passing mathematical objects between systems and making correct interogations in the appropiate language. It will results in a client-server environment with many CAS servers able to offer services to the Intense client.

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** (1989), No. 5, 546–562.
2. B. Chandrasekaran, *Design problem solving. A task analysis.* AI Magazine (1990), 59–71.
3. 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**, No. 4 (1995) 435–471.
4. N. Jacobstein, C. T. Kitzmiller, and J. S. Kowalik, *Integrating symbolic and numerical methods in knowledge-based systems: current status, future prospects, driving events. Coupling Symbolic and Numeric Computing in Expert Systems.* Elsevier Science Pbs., 1988, 3–11.
5. 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**, No. 1 (1993), 44–61.
6. C. T. Kitzmiller, J. S. Kowalik, *Symbolic and numerical computing in knowledge-based systems.* Coupling symbolic and numeric computing in expert systems, Elsevier Science Pbs., 1986, 3–15.
7. J. S. Kowalik, *Coupling symbolic and numeric computing in expert systems.* Elsevier Science Pbs., 1986.
8. S. Maruster, *Numerical methods in nonlinear equations solving.* Editura Tehnică, 1981.
9. V. Negru, *Polymers characteristics determination using a blackboard model.* Artificial intelligence: Methodology, Systems, Applications, World Scientific, September, 1994, 291–297.
10. V. Negru, S. Maruster, L. Maruster, M. Marin, and C. Sandru,, *Intelligent system for solving simultaneuos nonlinear equations. A CLIPS implementation.* Theoretical Informatics Department, Timişoara West University, 1995, Research Report.
11. V. Negru, L. Maruster, *Non-linear equations systems solver.* Artificial Intelligence: Methodology, Systems, Applications; Frontiers in Arificial Intelligence and Applications series, IOS Press, June 15-18, 1997, 330–339.
12. C. Sandru, V. Negru, *A distributed task-oriented problem solving system.* International Workshop "Distributed Artificial Intelligence and Multi-Agent Systems" (W-DAIMAS´ 97), June 15-18, 1997, 193–203.
13. A. Newell, *The knowledge level.* Artificial Intelligence year, No. 18 (1981), 1–20.
14. C. Pierret-Golbreich, *Vers une nouvelle architecture pour l'explicitation du raisonnement au niveau connaissances: task.* Université Paris-Sud, 1991, Rapport de recherche, No. 09/674, Orsay.
15. F. Rechenmann, *Modeling tasks and methods in knowledge based PDE solver.* 1991.
16. M. F. Russo, R. L. Peskin, A. D. Kowalski, *Using symbolic computation for the automatic development of numerical programs.* Coupling symbolic and numeric computing in expert systems, Elsevier Science Pbs., 1988, 77–89.
17. R. Simmons, *Generate, Test and Debug: a paradigm for combining associational and cauzal reasoning.* Second generation expert systems, Springer Verlag, 1993.

18. L. Steels,, *Components of expertize.* AI Magazine, 1990.
19. S. N. Talukdar, E. Cardozo, L. Leao, R. Banares, and R. Jacobbani, *A system for distributed problem solving.* Coupling symbolic and numeric computing in expert systems Elsevier Science Pbs., 1986, 59–65.
20. B. Wielinga, G. Schreiber, *KADS: Model based KBS development.* GWAI-90, 14th german workshop on artificial intelligence, 1990 Springer-Verlag, 322–333.
21. L. Winkelbauer, *A structure for the incremental construction of coupled systems: A bridge between shallow and deep coupling.* Coupling symbolic and numeric computing in expert systems Elsevier Science Pbs., 1988, 145–156.
22. C. Culbert, G. Riley, and B. Donnell, *CLIPS reference manual, Volume 1, 2, 3.* Johnson Space Center - NASA, 1993.
23. J. Smart, *Reference manual for wxWindows 1.50: A portable C++ GUI toolkit.* AIAI, University of Edinburgh 1993.
24. J. E. Dennis, J. J. More, *Quassi-Newton methods, motivation and theory.* SIAM J. Numer. Anal., **19**, No. 1 (1977), 46–89.
25. J. E. Dennis, *On the convergence of single-rank quassi-Newton methods.* Math. Comp., **24** (1970), 365–382.
26. V. M. Fridman, *Iterative method with minimal error for non-linear operatorial equations.* Acad. Nauk, Russia, **139** (1961), 1063–1066.
27. St. Maruster, *On the two-step gradient method for nonlinear equations.* Anal. Univ. Timisoara, **25**, No. 3 (1987), 69–84.
28. R. B. Kellog, *A non-linear alternating direction method.* Math. Comp., **23** (1969), 23–27.
29. J. M. Ortega, *Stability of difference equations and convergence of iterative processes.* SIAM J. Numer. Anal., **10** (1973).
30. W. B. Gragg, R. A. Tapia, *Optimal error bounds for the Newton-Kantorovici theorem.* SIAM J. Numer. Anal., **11**, No. 1 (1974), 10–13..
31. S. Weerawarana et al., *PYNTHIA: A knowledge-based system to select scientific algorithms.* ACM Transactions on Mathematical Software, **22**, No. 4 (1986).
32. E. Hairer, P. P. Norsett, and G. Wanner,, *Solving Ordinary Differential Equations I. Nonstiff Problems, 1987 and II. Stiff and Differential-Algebraic Problems, 1991.* Springer Verlag.
33. D. Petcu, *Multistep methods for stiff initial value problems, Mathematical Monographs 50.* Timişoara University Press, (1995).
34. D. Petcu, *Parallel numerical algorithms, Part I. Solving systems of linear, nonlinear or differential equations, Mathematical Monographs 60.* Timişoara University Press, 1996.
35. D. Petcu, *Implementation of some multiprocessor algorithms for ODEs using PVM.* Lecture Notes in Computer Science 1332, Fourth European PVM-MPI Users Group Meeting, Krakow, Poland, 1997, Springer-Verlag, 475–481.

Department of Computer Science
Faculty of Mathematics
West University of Timişoara, Romania
e-mail: {stmaru,vnegru,petcu,sandru}@info.uvt.ro

Поступило 20 июня 1999 г.