CREATIVE MATH.
13 (2004), 97 - 110

Software issues in solving initial value problems for ordinary differential equations

Dana Petcu

Abstract. A vast collection of mathematical software, representing a significant source of mathematical expertise, is available now for use by scientists and engineers in their efforts for modelling the evolution of real systems. Unfortunately, the heterogeneity of this collection makes difficult the task to determine what software is available to solve a given problem.

We discuss the state-of-the art in the field of software for ordinary differential equations (ODEs) and several issues which are of importance both from the point of view of software design and software or method evaluation. The survey is based on the analysis of current software: computer algebra systems including ODE solving facilities, dedicated problem solving environments, and specialized free or commercial packages. A special attention is paid to stiff and large ODE systems and parallel solvers. The requirements of an expert system for solving initial value problems (IVPs) for ODEs are also discussed.

1. Introduction

The volume and diversity of the numerical software available for ordinary differential equations has become, today, a problem. When confronted with many software products, it is important for the user to have some knowledge 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. A difficult problem is also the selection of an adequate method from the method database of a specific software product. In the case of an IVP for ODEs, a correct selection can be done only knowing the problem properties.

In this paper we review the current available ODE software packages (Section 2) and describe what it is expected today from a ODE solver (Section 3).

2. Current available ODE software

2.1. **Special codes and search engines.** Early knowledge about the numerical solution of ODEs was transposed in a lot of codes. The most used language for ODE solvers is Fortran due to the period in which these codes where developed. One can find many such codes in the public domain by using current web searching engines or classification schemes like GAMS [17]. We review here the most known codes which are still used nowadays as they are or as black boxes behind sophisticated interfaces.

ODEPACK [23], developed in 1983 and last updated in 2003, is a collection of Fortran solvers for the IVP for ODEs. It consists of 9 solvers, namely a basic solver called LSODE and 8 variants of it – LSODES, LSODA, LSODAR, LSODPK,

Received: 24.09.2004. In revised form: 10.11.2004.
1991 *Mathematics Subject Classification.* 65L05, 65L12, 68W10.
Key words and phrases. *Initial value problems, mathematical software, parallel computing.*

LSODKR, LSODI, LSOIBT, and LSODIS. The collection is suitable for both stiff and nonstiff systems. It includes solvers for systems given in explicit form $y\prime = f(t, y)$, and also solvers for systems given in linearly implicit form, $M(t, y)y\prime = g(t, y)$ (differential-algebraic equations, DAEs). Two of the solvers use sparse matrix direct solvers for the linear systems that arise and two others use iterative (preconditioned Krylov) methods. The most recent addition is LSODIS solving implicit problems with general sparse treatment of all matrices involved.

LSODE, the Livermore solver for ODEs [43] (for explicit forms), the successor of the package GEAR [7], treats in the stiff case the Jacobian matrix f'_y as either a dense or a banded matrix, and as either user-supplied or internally approximated by difference quotients. It uses Adams methods (predictor-corrector) in the nonstiff case, and Backward Differentiation Formula (BDF) methods in the stiff case. The linear systems that arise are solved by direct methods (LU solver). VODE [4] is similar to LSODE, but it uses variable-coefficient methods instead of the fixed-step-interpolate methods in LSODE. VODPK [6] is a variant of VODE which uses iterative preconditioned Krylov methods for the linear systems that arise, instead of the direct methods in VODE. GEARBI [24] is a variant of the older GEAR package which solves stiff and nonstiff systems, using BDF and Adams methods. In the case of stiff systems, it uses a block-iterative method, Block-SOR, to solve the linear systems that arise at each time step. GEARBI is designed for use on problems that arise from the spatial discretization of PDE systems, such that the resulting ODE system has a regular block structure. The Fortran code BiM [2], based on blended implicit methods, implements a variable order-variable stepsize method for (stiff) initial value problems for ODEs. The order of the method varies from 4 to 14, according to a suitable order variation strategy. J.F. Cash [9] proposed a suite of codes (last updated in 2000, Table 1) written in Fortran and implementing Modified Extended BDFs (MEBDF).

KRYSI [25] is another solver for stiff systems, and is a variant of an implicit Runge-Kutta solver called SIMPLE. Both KRYSI and SIMPLE use the same 3-stage third order SDIRK method. But where SIMPLE uses a direct (dense) solver for the associated linear systems, KRYSI uses a preconditioned Krylov method (preconditioned GMRES iteration).

Netlib [27] collects and distributes the most used free codes for the numerical solution of ODES. A short list of them is reproduced in Tables 2 and 3.

Several codes were developed to solve large systems of ODEs on parallel computers. A good example is the package developed at the University of Halle (Table 4) which uses the standard Basic Linear Algebra Subroutines (BLAS) and LAPACK.

The Fortran codes from the successful ODE books like those of Hairer, [19] and [20], are available on the net [21] and for some of them a Matlab interface was already designed. Ode [3] (last update in 2001) is a Unix command-line ODE solver which uses a slightly modified version of the 8th-order Runge-Kutta code DOP853 proposed by Hairer in [19].

The recent code GAM [16] (last update in 2003) numerically solves IVPs of first order ODEs using boundary value methods, namely the Generalized Adams Methods (GAMs) of order 3,5,7,9 with step size control.

2.2. **Testing the software.** The testing of ODE solvers has been, until recently, limited to comparing code performances using for example the DETEST set [14] from 1975. More recent test sets are nowadays available on the web, e.g., the Geneva test set [22], the IVPtestset [8], the ODElab [35], CWI test set[13], NSDTST and STDTST [15], and PADETEST [1].

The development of adaptive software from mathematically methods entails selecting termination criteria for iterative methods, constructing control structures

TABLE 1. Cash codes based on MEBDFs [9]

Code	Type of problem and method specifics
MEBDFDAE	stiff IVPs of ODEs in explicit form or DAEs with M a constant matrix.
MEBDFV	stiff IVPs for systems of linearly implicit DAEs $M(y)y\prime = f(t, y)$.
MEBDFI	IVPs for systems of implicit DAEs of the general form $g(t, y, y\prime) = 0$.
MEBDFSO	stiff IVPs for very large sparse systems of ODEs of the explicit form; the linear equation solver is the sparse solver YSMP; the code is particularly useful in the solution of time dependent PDEs using the method of lines.
MEBDFSD	stiff IVPs for very large sparse systems of DAEs with M a constant matrix; the sparse solver used is MA28; the code has applications in the method of lines solution of time dependent PDEs.

TABLE 2. ODE software on Netlib [27]

Code	Type of problem/Method
composition	ODE/composition methods, mostly palindromic schemes size
cvode	large nonstiff or stiff ODE IVP/combines earlier vode and vodpk
dresol	stiff and nonstiff matrix differential Riccati eqs/Adams' formulae and BDSs
dverk	ODE, global error control/Verner's 5th and 6th order Runge-Kutta pair
epsode	stiff ODE/BDFs (variable coefficient formulae)
mebdfdae	stiff ODE and linearly implicit DAE IVPs/extended BDFs
mebdfso	large sparse stiff ODE IVPs/extended BDFs
ode	ODE IVP/Adam's methods
odeToJava	stiff, nonstiff ODEs/explicit Runge-Kutta, linearly implicit-explicit IMEX
parsodes	large systems of stiff ODEs/multiimplicit Runge-Kutta with "across the method" parallelization in MPI
rkc	parabolic PDEs/2nd-order explicit Runge-Kutta-Chebyshev formulae
rkf45	ODEs/Runge-Kutta Fehlberg 4th-5th order
rksuite	ODEs/a suite of codes, choice of RK methods; includes an error assessment facility and a sophisticated, stiffness checker
sderoot	ODE,with root stopping/Adam's methods
sode	ODE/Adam's methods
srkf45	ODE/Runge-Kutta Fehlberg 4th-5th order
svode	nonstiff or stiff ODEs/BDFs (variable coefficient formulae)
svodpk	large nonstiff or stiff ODEs/BDFs (variable coefficient formulae) with GMRES with user-supplied preconditioner
vode	nonstiff or stiff ODEs/BDFs (variable coefficient formulae)
vodpk	large nonstiff or stiff ODEs/BDFs(variable coefficient formulae) with GMRES with user-supplied preconditioner

Table 3. DAE software on Netlib [27]

Code	Type of problem/Method
coldae	semi-explicit DAEs,index<=2/collocation,projection on constraint manifold
daspk	DAE/BDFs with direct & preconditioned Krylov linear solvers
daskr	DAE, with rootfinding/BDFs, direct & preconditioned Krylov linear solvers
ddassl	stiff DAE/BDFs
ddasrt	stiff DAE with root stopping/BDFs
dgelda	general linear DAEs/package including a computation of all the local invariants of the system, a regularization procedure and an index reduction scheme; implements BDFs and Runge-Kutta schemes
mebdfi	general implicit DAE IVPs, index<=3/extended BDFs
sdasrt	stiff DAE, with root stopping/BDFs
sdassl	DAE/BDFs

Table 4. Codes for large ODE systems [30]

Code	Type of problem and method specifics
ROWMAP	a ROW-code of order 4 with Krylov techniques for large stiff ODEs
EPTRK	explicit pseudo two-step RK methods of order 5 and 8 for parallel computers
ROWMBS4	a 4th order partitioned Rosenbrock method for multibody systems, 8 stages
EPTRKN	two explicit pseudo Runge-Kutta-Nyström methods of order 6 and 10
NYRA	a Runge-Kutta-Nyström-type block predictor-corrector method for parallel computers with shared memory

Table 5. Codes from the books [19] and [20] available on the net [21]

Code	Type of problem and method used
DOPRI5	explicit Runge-Kutta method of order 5(4) for problems in explicit form
DOP853	explicit Runge-Kutta method of order 8(5,3) for problems in explicit form
ODEX	extrapolation method (GBS) for problems in explicit form
ODEX2	extrapolation method (Stömers rule) for second order DEs $y\prime\prime = f(x, y)$
RADAU5	implicit Radau IIA method of order 5 for DAE problems with constant and possibly singular matrix M; concerning the linear solvers the user has the choice to link the program with LAPACK and other solvers.
RADAU	implicit Radau IIA method of variable order (switches automatically between orders 5, 9, and 13) for DAE problems with constant and possibly singular M
RODAS	Rosenbrock method of order 4(3), for DAE problems with constant and possibly singular matrix M; algebraic order conditions verified; linear solvers – the user can choose to link the program with LAPACK and other solvers.
SEULEX	extrapolation method based on linearly implicit Euler for DAE problems with constant and possibly singular matrix M; linear solvers – same as for RODAS
SDIRK4	diagonally-implicit Runge-Kutta method of order 4 for DAE problems with constant and possibly singular matrix M
ROS4	classical Rosenbrock methods of order 4(3), for DAE problems with constant and possibly singular matrix M
SODEX	extrapolation method based on linearly implicit mid-point rule for DAE problems with constant matrix
RADAUP	implicit Runge-Kutta method of order 5, 9, or 13 (Radau IIA) for DAE problems of the form with constant and possibly singular matrix M

and objectives, and many more decisions. Most software constructors have taken a heuristic approach to these design choices, invalidating the process of deducing from software comparisons that one method is better than another [44].

2.3. **Symbolic computations.** When solving stiff IVPs, analytical Jacobians are quite advantageous. Current CAS have an automated differentiation capability that appeared to offer an exciting possibility of combining symbolic and numerical methods. When a numerical solution is requested, the solution is provided for discrete values; the accuracy is constrained by hardware and the problem itself. The solution can be obtained fast with the current hardware and one can solve large and complex ODE systems. When a symbolic solution is requested, the solution is provided as a function and one can get more-or-less arbitrary accuracy; the constraints of speed and complexity cause emphasis on a few equations.

RKF45 is the Fortran code most widely used in CAS to numerically solve IVPs. It is the foundation of the original default IVP solvers of the CASs (Maple rkf45, Matlab ode45, Mathematica NDSolve). The dsolve command in Maple used with the numeric option applies the (4,5) explicit Runge-Kutta. It works well if the system is nonstiff. Maple procedures detects the stiffness as follows. For efficiency reasons the step size must be as big as possible. However, the step size must be small enough that the error at each step is less than a given tolerance the computation is stable. If accuracy determines the step size, the IVP is nonstiff. If stability restricts the step size severely, the IVP is stiff. In the stiff case a (3,4) Rosenbrock method is used and the solver forms a procedure internally for evaluating the Jacobian analytically. Supplementary options are available in Maple when the user loads the ODEtools package [37]. Similarly, Odesolve ([36], last update in 2003) is a MATLAB program for solving arbitrary systems of ODEs; it uses the programs dfield and pplane which are described in some detail in [42]. It follows the ODE Suite developed in 1997 [46].

Several comparisons of performances between different CAS were performed on thousands of ODE systems and help to improve their facilities. Significant results were obtained using the Kamke [32] and Wester tests [48].

Specialized software was also produced. For example Taylor [47] produces a numerical solver (C program) for a given set of ODEs in symbolic form.

CATHODE and CATHODE 2 were two European ESPRIT projects (finalized in 2000), concerning Computer Algebra Tools for Handling ODEs. They produce a set of cooperating computer algebra tools for the manipulation and solution of ordinary differential equations and systems. New algorithms were developed, and implementations created (Table 6). NODES (Nonlinear ODEs Solver, [11]) for Maple (1997) was developed of the first project CATHODE. Its goal was to develop tools for handling dynamical systems. Its original aspect lies in the use of a matrix notation for representing DEs (called the Quasimonomial Formalism). One matrix is used for the coefficients and an other one for the exponents. Thanks to this notation, usual techniques may be reformulated using only basic linear algebra and become suitable for an efficient computer algebra implementation. The program is the MAPLE implementation of the techniques which were developed. Note that

TABLE 6. CATHODE-2 software[10]

Acronym	Type of problem and method
BERNINA	stand-alone interactive program for computing rational solutions, symmetric and exterior powers, Darboux curves and invariants of linear ODEs
BOLD	MAPLE package for computing error bounds for IVPs with linear ODEs.
CONLAW	REDUCE package for determining conservation laws for PDEs or ODEs.
CRACK	REDUCE package for solving overdetermined systems of PDEs.
DIFFGROB2	Maple package for computing differential Gröbner bases.
EXPSOL	MAPLE package for computing exponential solutions of linear ODEs.
ISOLDE	MAPLE package for computing formal invariants and local and global symbolic solutions of systems of linear ODEs.
ODESOLVE	REDUCE package for solving simple ODEs in closed form.
OMWS/OMD	OpenMath Worksheet/Dispatcher is a client/server systems for offering easy access to OpenMath - based computation over the internet.

NODES (Numerical ODEs) is also the name of another package [45] which was partially added in the default solver in Maple 7.

2.4. **ODE software for education.** Traditional introductory courses in ODEs have concentrated on teaching a repertoire of techniques for finding formula solutions of various classes of DEs. The fundamental of stability, asymptotics, dependence on parameters, and numerical methods are difficult to teach because they have a great deal of geometrical content and, especially in the case of numerical methods, involve a great deal of computation. Modern mathematical software systems can help to overcome these difficulties.

The ease with which numerical routines could be implemented in a programming language made the software a very useful pedagogical device. However, computer technology distracted students from learning about differential equations. The scene changed dramatically with the advent of symbolic manipulation programs which made symbolic as well as numerical solutions of differential equations possible via computers. In this context several books have appear in the last decade dealing with using computing to teach ODEs. The book [18] for example provide a traditional treatment of elementary ODEs while introducing the computer-assisted methods that are available with Mathematica. The book [12] uses Maple to introduce numerical methods, geometric interpretation, symbolic computation, and qualitative analysis into an ODE course.

A Consortium of ODE Experiments (C*ODE*E [34]) was establish in 1992 (last update in 1998) with the goal to share the rapidly growing wealth of computational instruction techniques with as many teachers of differential equations as is possible. ODE Architect was developed in the frame of CODEE in 1996. It combines rich multimedia application with powerful yet easy-to-use custom mathematical tools. The software is intended to provide a highly interactive environment for students to examine the properties of linear and nonlinear systems of DEs, and to explore and construct ODE modes of real-world situations, as well as self-designed models. It combines mathematical simulation, graphic animations, hypermedia, and numerical solvers to offer a complete multimedia learning environment, and a friendly, efficient, and interesting way to study and explore ODEs.

TABLE 7. A Windows and MS-DOS software collection for ODEs [26]

Type	Software
Freeware or public domain	Mafia, ODE, ODE's slide show, Linear ODEs, Populus, Slopes, Transmath, VisualMethods, Graphing separable equations, Ready for ODEs, Calculus and DEs, Midshipman DE program
Shareware, all features enabled	Graphmatica
Shareware, some features disabled	Formula wizard, ODEcalc, SymbMath
Commercial software	Biograph, Logistic, MicroCalc, MLab, Models, ODE workbench, Phaser, Chaos Simulations, Chaotic Dynamic Workbench, Chaotic Mapper, DEGraph
Demo of commercial software	tour of Maple, Chaos demonstrations

A Windows and MS-DOS software collection for ODEs (last update in 2001) is available from 1996 [26] and includes description of several packages (Table 7) which were described also in [39].

IDEA [28] is another effort to provide students and teachers around the world with computer based activities for DEs in a wide variety of disciplines. IDEA contains a database of computer activities illustrating both mathematical concepts and the application of these concepts in a wide variety of disciplines. It provides the software DynaSys (1997) that can be used to implement many of the activities.

Infinity (released in 2004, [29]) is a non-linear math application that allows to use complex mathematical expressions within equations to describe the problem which requires solution. Once the model is described using the common math language one can see the result immediately.

2.5. **Software classification.** The actual software applications for ODEs can be classified using some criteria as (a) the solving method – symbolic computations or numerical computations, (b) the covered domain – general problem solvers or problem dedicated solvers, (c) the opening to new methods – build-in solving methods or with some mechanisms for constructing new problem solvers.

The IVPs of ODEs can be classified in two categories: for which one can construct an analytic solution and those for which one do not know how to construct an analytic solution. In the first case, one can obtain a function which represent the exact solution, using one of the actual computer algebra systems (CAS) which allows symbolic computations. If the problem is simple, probably, the system will recognize very fast the exact solution. Otherwise, one must apply some transformations to the ODEs, or must use dedicated library procedures which are not in the system kernel. Unfortunately, an overall mathematical knowledge is not translated in databases for computer applications. If the try to obtain an analytic solution (using the computer or an human expert) fails, one must try to approximate the exact solution. The number of numerical methods which can be selected 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 is requested for the approximation, the time constrains, the knowledge of the solving method class.

General purpose tools (for example, the CASs) have included a mechanism for generating an analytical or numerical solution of an IVP for ODEs, which, unfortunately, is not usefully for any kind of ODEs. 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) are almost all designed with a friendly user interface and have been build with some fixed method databases which are properly for specific classes of problems with a small number of equations.

Method libraries (like Odepack) can be used in the integration of a wide variety of large systems of equations if the user know 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 [31], Godess [38] and EpODE [40]) 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.

3. Constructing an expert system for IVPs of ODEs

The main requirements of an expert system for ODEs are the following ones: friendly user interface for problem specification, automatic detection of the problem properties like linearity, separability, the extreme eigenvalues of the Jacobian matrix at the initial value (symbolic computation of the system Jacobian), stiffness ratio, estimated time for the function evaluation, friendly user interface for a difference method specification, automatic detection of the method properties like explicit or implicit schema, method accuracy, stability properties, one-step or multistep, one-stage or multistage, one-derivative or multiderivative types. It must establish the matching between the problem properties and the method properties, adequate integration step, approximate computation time, estimated error, and then apply the numerical method and supervise the error. It must be possible to create a list or a graphic of approximate solution values. It must be also possible to solve the problem without the specification of a particular method: (i) choosing an appropriate method from a database of classical methods (the selection is based on a classification of the methods, and the database can be enlarged introducing new methods using the above mentioned front-end); (ii) select a new method when the previous one generates unreasonable errors; (iii) in the case of a large number of equations the expert recommends the use of the numerical codes combined with a message passing interface (like PVM and MPI) in the idea to distribute the computations on some processors of a local network. The automatic selection must be based on the problem properties and the user must control the maximum computation time and the maximum level of the accumulated error of the approximate solution relative to the exact solution. In order to interpret the results, the approximate solution can be analyzed reading the table with the computed values, or looking to some graphics in two or three dimensional space. The solvers for IVPs must be implemented in a uniform way for several methods – an unique calling sequence for all methods, so that all solvers behave in a coherent way. New methods must be easy to be added, tested, and verified under similar conditions.

3.1. **Problem's and method's properties identification.** The special properties of the problem must be identified, but unlike in the application with fixed method set, some theoretical results about the characteristics of an difference method can be checked. For the solution computational process very important are the method order (which can be practically influenced by the starting procedure, in the case of multistep methods) or the method stability properties (for example, the boundless of the stability region can be influenced by the implicit equation solver – these aspect is not always underlined when a new implicit method for ODEs is described). The method order can be estimated applying the method to some test equations and finding the accuracy of the approximate solutions. The domain of A_0-stability of each iterative method can be establish. A method which is A_0-stable can be a candidate for stiff-stability or A-stability (properties which are more complicated to be establish numerically for any kind of iterative method, and which are adequate for a stiff problem solver).

3.2. **Unique generic solving procedure.** Although it seems not possible to express all difference methods for ODEs in a same form (think to different representation of multistep, Runge-Kutta schemes, block, nonlinear, multiderivative, (A,B)-methods, schemes with variable stepsize, etc.), from the programming point of view it is possible to think to all of these methods as special cases of an generic iterative procedure which must be used for any problem (some concessions have been make, for example, it is very complicate to describe any possibility to approximate Jacobian for an arbitrary function, and, therefore, it was more convenient to construct a procedure for symbolic derivation of an arbitrary mathematical expression). One of the multiple advantage of using a single generic solver procedure is the possibility to compare different methods solving the same problem, especially using criteria like approximate solution accuracy, computing time or effort.

3.3. **Mechanisms for problem and method splitting.** These tasks can be accomplished constructing dependency graphs and identifying the independent parts of them – direct applications are the possibility of distributing the solution computation on distinct processes running on different processors, and optimizing the solution computation time and information storing (especially for sparse systems).

3.4. **User interface for describing new iterative methods.** A method analyzer must be behind these task which take each entry given by the user and interpret. To describe an iterative method the user must specify the method variables which represent the entries for one iterative step, the outputs of the same step, the intermediate variables requested to describe one iterative step, the output variables which are considered as control variables in the error estimation procedure, the iterative equations (relations between the entries, outputs and intermediate variables), and how the outputs of one step are to be changed into entries to the next iterative step. According to the properties detected by the method analyzer, supplementary information must be provided by the user – for example, the implicit equation solver in the case of an implicit method.

3.5. **Method selection mechanism.** Using a problem properties analyzer, an expert in ODE numerical solvers can estimate which class of methods are theoretically indicated for the current problem – accomplishing this task automatically is not very simple, and the simulation of the human expert remains an open problem. The software can be designed to recognize stiff or nonstiff character of an IVP for ODEs, and to consider, respectively, numerical methods for stiff or nonstiff problems (inside a such class which method will be applied for the current problem depends on the solution accuracy requirements and on the computational time restrictions). Just before the approximate solution computation, the maximum stepsize, for which a given level of solution accuracy is attained, must be automatically computed using the method order and the current stability restrictions according the eigenvalues of the Jacobian matrix.

3.6. **Large set of problems and methods.** In order to extend as much as possible the expert applicability, several problems must been tested from the point of view of their known properties, and thereafter integrated with different methods. After the approximate solution computation, different classical statistics must be reported: function evaluation, matrix inversions, Jacobian evaluations, computing time, number of integration steps. These statistics recorded for a large set of problems can be used to compare different methods.

3.7. **Parallel computations.** The means of achieving parallelism in IVP solvers can be classified into three main categories [5]: (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 (or across steps) – 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.

The most natural way to apply the parallelism across system is to decompose the ODE system into several independent ODE subsystems. This decomposition is not possible for any system. The software must be capable to detect the separability of the ODE system into independent subsystems (a natural parallelism); a number of processes equal with the number of subproblems will be created. A such process works on his associated subsystem and must communicate only with the main process which collects the results and display them.

Parallelism across the method is the employ of the parallelism inherently available within the method. It can be achieved, for example, by computing the stages of a Runge-Kutta method on different processor. A similar technique with the digraph method, used for Runge-Kutta methods, can be adopted to determine the degree of the parallelism of a given method. Each equation of the iterative procedure associated with the numerical method is analyzed and a data flow graph is determined. This graph is divided into stages and processes in a similar manner with the level-process partitioning proposed for Runge-Kutta methods. An appropriate number of processes will be created. At a particular stage, each process is responsible for the solution of one or more method equations. These equations are to be changed from each stage to another, but, at a particular stage number, the equations distributed to a process are the same at each integration step.

In parallelism across the time a number of steps is performed simultaneously, yielding numerical approximations in many points on the time-axis in parallel. For example in the case of a Jacobi waveform relaxation method, at the first stage, the system of n differential equations is decoupled into n independent equations which can be computed in n separated processes. At the next step, some n new equations are integrated separately, and the difference between the old and the new solutions must be computed. If this difference in a given norm is under the admissible error level, the solution computation main process is stopped.

The implementation questions that need to be addressed in the development of a parallel production code for the numerical solution of ODEs are considerably more complex than in the sequential case. Such facets that must be considered include the automatic load balancing of the work amongst processors, the avoidance of non-determinacy and deadlock, whether synchronous or asynchronous communication should be utilized and the avoidance of communication bottleneck, and finally the gathering of appropriate performance statistics and suitable test problems. Comparing the proposed algorithms by measuring their performance on well-known parameters such as efficiency can really become a very difficult task. The comparison should be made between the execution time of the parallel algorithms and the fastest existent sequential algorithm running on the same machine. One can speak about efficiency of the parallel algorithm implementation when there is a large number of differential equations or a large integration interval (especially in the case of stiff systems). In the case of parallelism across system, load balancing is obtained when the subsystems are of the same dimension and difficulty. In the case of parallelism across method the computations must be balanced between the processes on each stage. The application of waveform relaxation methods allows the reduction to some systems of algebraic equations of smaller dimension and correspondingly leads to a greatly reduced computational effort; to achieve load balancing these new systems must be of similar complexity.

3.8. **Available prototype.** The first version of EpODE (expert system for ODEs) was build in 1997-1999 with respect to the above requirements. It was designed especially for the numerical solution of stiff ODEs and it is independent from any other computer application. No supplementary code is generated when a new method will be added. A large database of problems and methods was tested (roughly 100 problems and 100 methods). The prototype is available on Win32, Linux and Unix (free download from http://web.info.uvt.ro/ petcu/software.html). Tests have been reported in [39] and [41].

3.9. **Adapting to the new technologies.** Grid computing enables the development of large scientific applications on an international scale. Grid-aware applications make use of coupled computational resources that cannot be replicated at a single site. Solving larger problems is possible by pooling together resources that could not be coupled easily before grids. In this context current international efforts are dedicated to build computational grids devoted to solve problems described in mathematical terms.

Computational grids are relying on a pool of mathematical software codes and on a collection of personal computers and dedicated clusters of workstations. To adapt an existing tool to a computational grid, grid-enabled version of the available software must be written. EpODE can be wrapped to be seen by a grid user as grid software resource (a wrapper is needed). On other hand, in order to construct a grid-aware version of EpODE, we must add to it at least the following facilities like multiple threads for different external solvers, search for hardware resources and code transfer, links to proprietary and free codes. Since different components of EpODE can be used not only for solving initial value problems, the codes are rewritten now in Java instead C++, allowing an easier use of Java CoG and Globus toolkit. The parallel ODE solvers from EpODE have been designed for dedicated homogeneous cluster environments. In order to use heterogeneous grid resources a dynamic load balancing scheme must be added. Moreover, the message passing interface PVM must be replaced with the MPICH-G which works both on cluster and grid environments. A grid-aware version of EpODE is now under construction.

4. Conclusions

A survey was proposed based on the analysis of current ODE software (computer algebra systems, dedicated problem solving environments, specialized free or commercial packages) with a special attention to stiff and large ODE systems and parallel solvers. Current requirements on ODE solvers are identified and an ODE expert architecture is proposed. New technologies like grid computing must be taken in consideration when a new generation of ODE solvers will emerge.

References

[1] Bellen, *A., PADETEST - a set of real-life test differential equations for parallel computing*, Technical Report 103, Universita di Trieste (1992)

[2] BiM, *Numerical solution of stiff ODE-IVPs*, http://www.math.unifi.it/~/ BiM/index.html

[3] Briggs, K., *ODE (2001)*, http://www.btexact.com/people/briggsk2/Ode.html

[4] Brown, P.N., Byrne, G.D., Hindmarsh, A.C., VODE, *A Variable-Coefficient ODE Solver*, SIAM J. Sci. Stat. Comput. **10** (1989), 1038-1051

[5] Burrage, K., *Parallel and Sequential Methods for Ordinary Differential Equations*, Numerical Mathematics and Science Publication Series, Oxford Publishers Claredon Press (1995)

[6] Byrne, G.D., *Pragmatic Experiments with Krylov Methods in the Stiff ODE Setting, in Computational Ordinary Differential Equations* (J. Cash and I. Gladwell, Eds.), Oxford Univ. Press, Oxford, 1992, 323-356

[7] Byrne, G.D., Hindmarsh, A.C., Jackson, K.R., Brown, H.G., *Comparative Test Results for Two ODE Solvers - EPISODE and GEAR*, ANL-77-19 (1977)

[8] Cash, J.F., *IVPtestset*, http://www.ma.ic.ac.uk/˜ jcash

[9] Cash, J.F., *Software for initial value problems* (2000), http://www.ma.ic.ac.uk/˜ jcash/IVP_software/readme.php

[10] Cathode2, *Software*, http://cathode.maths.qmw.ac.uk/demos.html

[11] Codutti, M., Ferier, L., Fliegans, O., *NODES: Nonlinear Ordinary Differential Equations Solver*, (1997), http://cso.ulb.ac.be/˜ mcodutti/nodes/

[12] Coombes, K., Hunt, B., Lipsman, R., Osborn, J., Stuck, G., *Differential Equations with Maple*, 2nd Edition (1998) John Wiley and Sons, Inc

[13] CWI, *Test Set for Initial Value Problem Solvers*, http://pitagora.dm.uniba.it/˜ testset/struct.htm

[14] Enright, W.H., Hull, T.E., Lindberg, B., *Comparing numerical methods for stiff systems of ODEs*, *BIT* **15** (1975), 10-48

[15] Enright, W.H., Pryce, J.D., *NSDTST and STDTST: routines for assessing the performance of IV solvers*, ACM Transactions on Mathematical Software, **13** (1), 1987, 28-34

[16] GAM, *Code for Stiff Initial Value Problems* (2003), http://www.dm.uniba.it/˜ mazzia/ode/readme.html

[17] *GAMS*, http://math.nist.gov/cgi-bin/gams-serve/class/I1.html

[18] Gray, A., Mezzino, M., Pinsky, M.A., *Introduction to Ordinary Differential Equations with Mathematica - An Integrated Multimedia Approach*, http://math.cl.uh.edu/ode/ode.html

[19] Hairer, E., Norsett, S.P., Wanner, G., *Solving Ordinary Differential Equations. Nonstiff Problems*, 2nd edition (1993), Springer Series in Comput. Math., vol. 8

[20] Hairer, E., Wanner, G., *Solving Ordinary Differential Equations. Stiff and Differential-Algebraic Problems*, 2nd edition (1996), Springer Series in Comput. Math., vol. 14

[21] Hairer, E., *ODE Programs from* [19] *and* [20]*, Springer)*, http://www.unige.ch/math/folks/hairer/prog/nonstiff.tar.gz & stiff.tar.gz, and ftp://ftp.unige.ch/pub/doc/math

[22] Hairer E., *Test set for stiff ODEs*, www.unige.ch/math/folks/hairer/testset/testset.html

[23] Hindmarsh, A.C., *ODEPACK – A Systematized Collection of ODE Solvers*, in Scientific Computing, (R. S. Stepleman et al, Eds.), North-Holland, Amsterdam, 1983, 55-64, http://www.llnl.gov/CASC/odepack/

[24] Hindmarsh, A.C., *Preliminary Documentation of GEARBI: Solution of ODE Systems with Block-Iterative Treatment of the Jacobian*, LLNL report UCID-30149 (1976)

[25] Hindmarsh, A.C., Norsett, S.P., KRYSI, *An ODE Solver Combining a Semi-Implicit Runge-Kutta Method and a Preconditioned Krylov Method*, LLNL report UCID-21422 (1988)

[26] Hush, L., *Windows and MSDos software collection for ODEs* (2001), http://archives.math.utk.edu/software/msdos/diff.equations/

[27] Netlib, *ODE Software*, http://www.netlib.org

[28] *Internet Differential Equations Activities*, http://www.sci.wsu.edu/idea/welcome (1997)

[29] *Infinity*, http://www.mathrevolt.com

[30] *Institut für Numerische Mathematik, Martin-Luther-Universität Halle-Wittenberg, ODE Software* (1997), http://www.mathematik.uni-halle.de/institute/numerik/software/

[31] Kamel, M.S., Ma, K.S. , Enright, W.H., *ODEXPERT: An expert system to select numerical solvers for initial value ODE systems*, ACM Transactions on Mathematical Software **19** (1993), 44-62

[32] Kamke, H., *Comparison of performances between Maple 7 dsolve and Mathematica 4.1 DSolve*, http://lie.uwaterloo.ca/odetools/comparison.html

[33] Lioen, W.M., de Swart, J.J.B., van der Veen, W.A., *Test set for IVP solvers* (1996), http://www.cwi.nl/cwi/projects/IVPtestset.shtml

[34] Math Department of Harvey Mudd College, *Consortium of ODE Experiments, C*ODE*E* (1998), http://www.math.hmc.edu/codee/home.html

[35] Nowak, L., Gebauer, A., *ODElab*, http://newton.zib.de:8001/public/odelab/

[36] *Odesolve*, http://math.rice.edu/˜ dfield/odesolve/odesolve.m

[37] *ODEtools package*, http://lie.uwaterlo.ca/odetools/

[38] Olsson, H., *Short presentation of Godess project* (1996), http://www.dna.lth.se/home/Hans_Olsson/Godess

[39] Petcu, D., *Parallelism in solving ordinary differential equations*, Mathematical Monographs **64**, Tipografia Universităţii de Vest, Timşoara (1998)

[40] Petcu, D., Drăgan, M., *Designing an ODE solving environment, in LNCSE* **10**: Advances in Software Tools for Scientific Computing,(H.P. Langtangen et al, Eds.) Springer-Verlag, Berlin, 2000, 319-338

[41] Petcu, D., *Experiments with an ODE Solver on a Multiprocessor System*, Computers & Mathematics with Applications **42** (8-9), 2001, Pergamon-Elsevier Science, 1189-1199

[42] Polking, J.C., *Ordinary Differential Equations using MATLAB*, 3rd ed., Prentice Hall (2004)

[43] Radhakrishnan, K., Hindmarsh, A.C., *Description and Use of LSODE, the Livermore Solver for Ordinary Differential Equations*, LLNL report UCRL-ID-113855 (1993)

[44] Soderlind, G., *Evaluating numerical ODE/DAE methods, algorithms and software*, in Procs. SCICADE 99, Fraser Island, Australia (1999)

[45] Shampine, L., Corless, R.M., *NODES package*, ftp://cygnus.math.smu.edu:pub/shampine/maplecodes/

[46] Shampine L., Reichelt, M., *The Matlab ODE Suite*, SIAM J. Sci. Comput. **18** (1997), 1–22

[47] Taylor, http://www.scicomp.uni-erlangen.de/letter/v0n12/s1

[48] Wester, M., *A critique of the mathematical abilities of CA systems. In CASs - A Practical Guide* (M.Wester, ed.), John Wiley (1999), math.unm.edu/˜wester/cas_review

Institute e-Austria in Timişoara
and
Western University of Timişoara
Computer Science Department
B-dul Vasile Pârvan 4, 300223 Timişoara, Romania
E-mail address: `petcu@info.uvt.ro`