COMBINED SYMBOLIC AND NUMERICAL METHODS FOR SOLVING EQUATIONAL SYSTEMS

Vishaka Nanayakkara and Gihan Dias
Department of Computer Science & Engineering
University of Moratuwa, Moratuwa.
Email: vishaka@cse.mrt.ac.Ik, gihan@cse.mrt.ac.lk

COMBINED SYMBOLIC AND NUMERICAL METHODS FOR SOLVING EQUATIONAL SYSTEMS

Vishaka Nanayakkara and Gihan Dias
Department of Computer Science & Engineering
University of Moratuwa, Moratuwa.
Email: vishaka@cse.mrt.ac.lk, gihan@cse.mrt.ac.lk

ABSTRACT

In this paper we discuss systems developed to assist scientists and engineers to solve their problems based on mathematical models using numerical and symbolic methods. Numerical library routines for numerical algorithms and Computer Algebra Systems (CASs) for symbolic computation are both now well established areas. The recent research interest in these areas is oriented towards combining these solvers, rather than on improving the individual solvers. We analyse the state of research in numerical and symbolic solvers as well as the area of these combining methods or coupled (or combined) systems as they are commonly referred to. We evaluate these coupled systems to identify their essential features and various methods of integration. Based on our analysis we then propose a new strategy for integration which can enhance the facilities provided by the existing symbolic and numerical systems.

INTRODUCTION

Engineers and scientists use equations to model the problems arising in their work, and then try to solve them. If the solution cannot be obtained by symbolic simplifications alone, the equations are transformed to a form in which they can be solved using numerical algorithms. Before computers became available both tasks were done manually. The desire to mechanise the tedious, often repetitive numerical calculations and to increase speed and accuracy was one of the driving forces behind the development of computers.

To exploit the advantages of computers for numerical calculations, users started developing new algorithms; which could make more efficient use of their power. Many algorithms were developed to solve the most common types of numerical problems. Collections of such algorithms, coded in various high level programming languages, have long been available. These numerical programs have frequently been refined to increase their efficiency and accuracy and hence are usually superior to any code a non-expert user might write for the same purpose. In order to use such a standard algorithm users need to transform their original problem to an instance of the one solved by the algorithm.

Symbolic computation can be used in the transformation of the initial mathematical description into an equivalent form that is either itself the solution or an instance of a problem for which a numerical solution algorithm exists. This process requires mathematical knowledge about transformation rules. Computer Algebra Systems are increasingly becoming capable to carry out these transformations without the risk of error and at far greater speed than would be possible if the same transformations were done by hand.

Given that there are tools to perform both numerical and symbolic calculations, the users have developed many solution algorithms which can efficiently be implemented using these solvers from both numerical and symbolic domains. However the absence of an environment to use these solvers interactively may hinder the efficient implementation of these algorithms. Coupled systems, systems which allow the

interactive use of symbolic and numerical solvers, can overcome this problem. Such systems allow the users to work using equations, formulae, symbols and numerical computations, evaluate results using graphical facilities provided by CASs and automatic or semi-automatic selection of numerical algorithms.

Design and development of these systems is still at an early stage. During the last two decades many researchers have approached this problem with different methods and many coupled systems have been designed and implemented. However one common characteristic of these attempts are that the systems are being developed only for particular applications. These application domain specific combined solvers have some drawbacks. The models, which cannot be fitted into one of these application areas or the ones, which require, more than one mathematical domain cannot benefit from such environments. The cost involved in developing custom built environments prohibits users from developing combined solvers specifically for an application. Therefore the requirement is to have an environment where both symbolic and numerical systems are used interactively without restrictions on the problems it can be used for. We propose a method to build an environment for using symbolic and numerical solvers for various application areas.

NUMERICAL LIBRARIES

The solvers available for numerical calculations largely consist of high level programming language subroutines implemented using the algorithms developed by various mathematicians. These subroutines have been organised into various libraries available as commercial or public domain software. In addition to the libraries there are systems developed with easy to use interfaces to similar types of subroutines. Numerical library routines in general are built using subprograms collected over the years. The available libraries are either general purpose or specific for some area of mathematics. The criteria used in classification of the subroutines are their scope, numerical stability, accuracy and speed.

NAG library routines [*http: www.nag.co.uk*] and Numerical Recipes [1] are examples for general-purpose numerical libraries. These libraries consist of subroutines developed using the most commonly used algorithms in mathematics, and covering the most frequently occurring mathematical problems. However these routines may not in certain situations achieve the same efficiency as some of the more specific ones. There are some libraries available to provide numerical solutions to problems that can be grouped into particular areas of Mathematics. The areas are selected to allow a large cross section of problems to be treated under the same area but are specific in such a way that most of the algorithms in any area can be programmed under one library. Examples of these area specific libraries are BLAS (Basic Linear Algebra Subprograms) [2,3], LAPACK (Linear Algebra PACKage) [4,5] and Templates [6] for Linear Algebra and IMSL (International Mathematical and Statistical Library) for Statistics.

Methods for Selecting Routines

The large number of algorithms and implementations available can make it difficult for users find the one best suited for a given problem. Especially non-expert users need assistance in the selection and the use of the available software. The traditional method of written documentation can sometimes be difficult to use for inexperienced

users. If these libraries are to be used in automated solution processes there must be methods to search for algorithms to match the problem requirements using keywords.

GAMS (Guide to Available Mathematical Software) developed by the NIST (National Institute of Standards and Technology, USA) provides a large tree-structured database of mathematical and statistical software which is available on-line. GAMS allows users to search based on library and subroutine names and keywords. Users can access GAMS using the web browsers at the URL http://gams.nist.gov.

Using an expert system oriented approach to subroutine selection is another possibility. NAXPERT [7] claims to be a prototype expert system, which gives advice to users about a small mathematical library based on Fortran for IBM personal computers. Users need to select the mathematical domain of the problem at hand and then either give the keywords describing the problem or respond to the keywords suggested by NAXPERT.

A more recent approach to select the subroutines is the use of Computer Algebra Systems. ARC (Automatic Routine Chooser) [8] is a package developed to exploit this idea. This system which is implemented in REDUCE [9] chooses the best routines to match the user's description of the problem. It is implemented for a part of the NAG library.

SYMBOLIC SOLVERS

With symbolic solvers users from Science and Engineering work with computers in their natural working environments, (i.e. using equations and algebraic terms). Apart from the long tedious sequence of mathematical manipulations, inappropriateness of approximate numerical solutions (for example, in computing real roots of polynomials), easy interpretation of partial results in symbolic form are also reasons for the need for symbolic computing. Most modern day Engineers and Scientists use symbolic preprocessing in order to ensure the efficiency and accuracy in carrying out the algebraic simplifications in their work.

The concept of symbolic computing (or non-numerical computing) is not restricted only to manipulation of algebraic equations. Program compiling, logic programming, word processing, expert systems and other artificial intelligence applications are some examples. But in the context of Computer Algebra (Symbolic Computing) in this paper we discuss only the algebraic manipulation of mathematical terms. A Computer Algebra System (CAS) can be defined as a collection of an algorithm with a data structure for representing non-numerical data, a language making it possible to manipulate them, and a library of effective functions for carrying out the necessary symbolic operations.

The two major requirements expected from CASs are to provide a set of basic pre-programmed commands which instruct the computer to carry out the algebraic calculations and to offer a programming language which enables the users to define higher level commands or procedures to enlarge the original set of commands. In order to fulfill these requirements CASs usually provide the following functions.

- Operations on integers, rational, real and complex numbers to (in principle) any desired accuracy.

- Simple analysis such as differentiation, expansion of series, etc and manipulation of formulae and operations on polynomials.
- Matrix algebra with numerical and/or symbolic elements.

Window based environments and graphical outputs are now becoming standard features of CASs. The field of CASs has now grown such that there are a large number of general purpose CASs available. Maple [10] and Mathematica [11] can be considered as the leaders among the commercial packages. The various news groups formed on the Internet for different CASs are examples for widespread user communities of these packages. In addition there are many special purpose CASs, which were developed for specific problem domains. CAYLEY [12] and Macaulay [13] are examples for such special purpose CASs.

MOTIVATION FOR COUPLED SYSTEMS

Efficient solutions of many problems encountered in Engineering and Science require combination of methods from numerical analysis and symbolic computation. Computers have long handled numerical calculations and as a result large libraries of numerical routines are available. As an alternative to the traditional approach of often tedious and error prone symbolic calculations by hand, CASs have become widely used in the last 20 years. However, in applications, which require a high level of both symbolic and numerical processing, the user is still forced to alternate between both types of solvers. Instead it could be desirable to have both capabilities combined. This has motivated various attempts at combined systems. In this section we will discuss features of such systems, existing research work towards integrating different solvers and problems related to producing efficient coupled systems.

Coupled systems can be defined as systems which provide facilities to implement solutions to a given domain of applications using both symbolic and numerical methods interactively, and without the necessity for the user to have specialised knowledge of the underlying systems. These systems can help the user to solve problems that need specialised knowledge and expertise. The merits of coupled systems can be summarised as:

- Computational efficiency: Where appropriate symbolic computation can reduce the numerical workload, for instance by pre-processing symbolic input to a form which is numerically tractable.
- Reduction of errors: Limits and approximations can be handled better symbolically than numerically.
- Automation: Simplified programming with automated solution process (e.g. automated code generation).
- User guidance: Systems can be developed which allow even the non-expert users to use them.
- Resource efficiency: Developing interfaces between existing numerical and symbolic solvers costs less human and other resources than an entirely new system.
- Reuseability of results: Combined systems allow recursive refinement of the problem solving process without leaving the system.

Existing Coupled Systems

The realization of the need for combined the symbolic and numerical solvers has led to a growing research effort in this direction and to the development of number of systems. ELLPACK [14, 15], Sinapse [16, 17], ObjectMath [18, 19], FRISCO [20], CAS/PI [21, 22] and OpenMath [23] are examples of some of the existing systems which are presently available as complete systems or systems in the research stage.

Depending on the type and the area covered the existing work can broadly be classified into two categories as *Software Environments* and *Protocols for Data Interchange*.

Software: Driven by the necessity of having systems which can solve specific applications using both symbolic and numerical solvers many people have designed systems using ad-hoc integration methods to communicate with solvers. But in spite of the fact that there is no general method for communication among solvers such systems can be considered as coupled systems. ELLPACK is an example for this category. Sinapse, ObjectMath and FRISCO also exhibit similar features.

The systems in this area show two different trends in the domains they cover. Some systems are developed to solve applications having similar mathematical problems. For example ELLPACK can be used only to solve problems of PDEs. The aim of the developers of FRISCO is to provide a solver for polynomials. On the other hand some solvers are developed to solve problems in a certain application area. ObjectMath is used to generate programs to design components for a certain industrial partner. SciComp using SciNapse develops solvers customised for certain applications.

Protocols: Realising the lack of an efficient general method to transform data types and handle other communication related issues in combining the solvers, recently there have been several attempts to remedy this. The earlier attempts of ASAP [24] and MP [25] have been followed by a more general method in the OpenMath project.

Functions that can be provided by coupled systems:

- Interactive access to both CASs and numerical library routines.
- Facilities for the users to work at various levels of abstraction, without leaving the system, using equations, formulae, symbols and numerical computation.
- Facilities to evaluate results using extensive graphical facilities provided by CASs.
- Automatic or semi-automatic selection of suitable numerical algorithms.

Efficient integration of systems from different origins involves dealing with many complex issues. Some of these are

- Hiding command language variations between the different solvers by developing a common interface.
- Transparent management of remote computations.
- Automated programming environments to interactively communicate with solvers providing easy and fast communication among solvers.

The existing approaches use different methods in addressing these issues. Also when developing application-specific coupled systems the extent to which these issues has to be met vary with the applications and the requirements of the users.

Most of the coupled systems have an aim of capturing the application knowledge necessary in the field in which they are applicable. This makes reuse of knowledge in those domains feasible without much trouble. Also the interactive use of solvers in one environment facilitates the reuse of partial results generated for the same problem.

In many coupled symbolic-numerical systems, the symbolic analysis of a problem is used to determine which of a given menu of numerical procedures needs to be performed. The symbolic routines then call the chosen numerical procedures with the appropriate numerical parameters. The output of the numerical computation is either the desired results themselves, or is passed back to the symbolic routines for further analysis and action. The link between symbolic and numerical components is static in the sense that the numerical routines are chosen from a pre-selected collection. The only degree of freedom is the choice of input parameters to them.

A more flexible way to couple symbolic and numerical computation is to use a CASs to generate numerical code automatically, on demand, from the symbolic component. The automatically generated code could be a complete computation itself, or could include calls to existing numerical libraries. The symbolic component then compiles, loads and executes the code. The numerical results could either be given to the user, or used as part of further symbolic analysis.

A FRAMEWORK FOR GENERAL INTEGRATION

As discussed in the previous section the existing approaches towards integrated solvers do not provide a solution to the requirement of a general integration environment. The aim of the proposed approach is to design a framework for integrating solvers in a general way without restricting them to specific application areas. We are proposing to build a common environment for using existing CASs and numerical library routines to enable the users to access the functionality of both interactively.

Most of the CASs available today are self-contained systems for general requirements. But a problem arises when an application requires extensive mathematical calculations, which are not provided by the embedded mathematical routines. We believe that the best way to overcome this restriction is to build an interface from which users can use the functions provided by both symbolic and numerical solvers interactively.

The most general solution would have many CASs and numerical libraries working under a common command level. With this in mind, the aim of this section is to show how such an environment can be developed using a single CAS and some numerical library routines.

A perfect automated programming environment would automatically transform the algebraic problem into efficient symbolic and numerical programs. It would select the required CAS functions, guide the CAS to do the simplifications, select the necessary

numerical routines and do the required transformations. Though such an automated system is the ideal, it is more realistic to assume some user interaction, with the user supplying information to decide which functions are needed in CASs, the extent to which these functions can be applied, and the type of transformations required. The knowledge given by an expert user can to a certain extent be replaced by a rule-base.

Among the reasons for not totally excluding the human interaction are the following:

- Accuracy versus speed tradeoff: the choice of algorithms may depend on the desired accuracy of the results, or on a user-imposed execution time limit.
- Indeterminacy of the rule-base: the situation where more than one algorithm is applicable, leaving the choice to the user is certainly preferable to imposing a selection rule not suggested by the problem at hand.
- Flexibility: new algorithms can be added and tested.

The self-contained nature of CASs requires efficient and effective techniques of transformation methods from problems to CASs and of partial results from these to numerical library routines. The common features of various application domains will be evaluated and then transformation rules to use the required symbolic and numerical methods will be defined. Figure 1 shows a diagrammatic representation of the proposed system.

The proposed system has to make use of the defined functions of CASs to suit the requirements of the given task, decide the extent to which symbolic simplification can be applied, make a correspondence between the output of CAS and the input to the numerical library routines; and also to provide for human interaction, where necessary.

The functionality of such a system will be illustrated using a case study from linear optimisation. Existing work by Manocha and Canny [30] on using symbolic and numerical calculations in solving the problem of inverse kinematics in robotics is another good example. Since the proposed method is general, it can be extended to handle problems from any number of application areas.

Description of Functions:

The various steps need to be performed by the proposed system are described below.

Problem Specification: This is the phase in which the user specifies the problem to be solved in mathematical terms.

Example: Solve ***apde***
Subject to ***boundary and initial conditions***
Where ***apde and boundary and initial conditions*** are valid mathematical expressions.

Pre-processing: The system simplifies the equations given where possible and transforms them to a standard type. This is required to ensure the necessary degree of uniformity in the problem specification in order to find solution methods.

Example: Cancel identical factors from the coefficient and right hand side of a PDE.

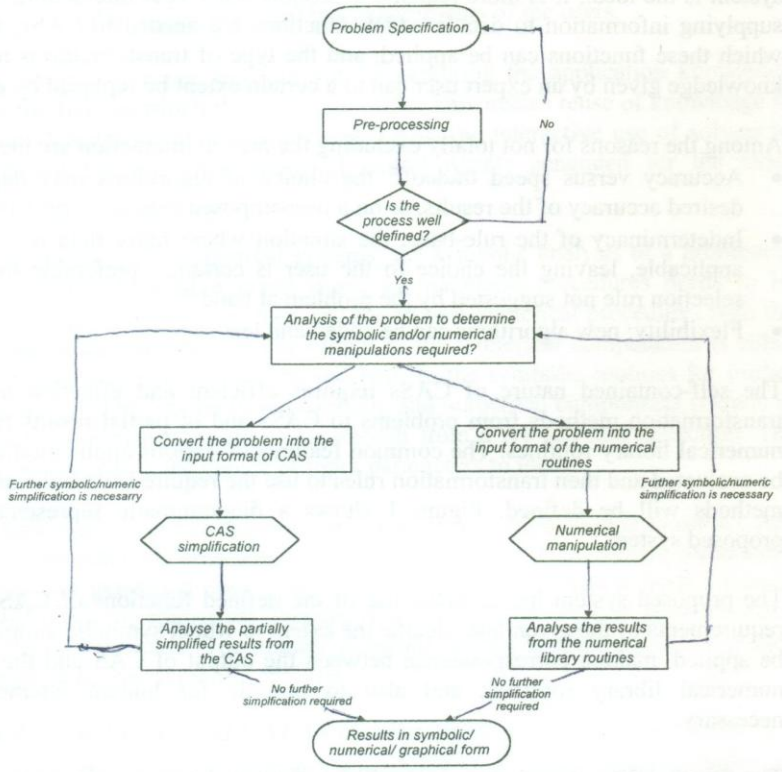


Figure 1 – Flow diagram of the proposed system

Problem Analysis and Algorithm Selection:

Problem Analysis

The system analyses the problem to identify the "type" of the problem. (E.g. In PDEs whether the given PDE is hyperbolic, parabolic or elliptic by analysing the discriminant.) Also the system must check whether the problem is well defined, i.e. whether it has a unique solution. Inappropriate specifications should be detected at this stage. As far as possible the analysis should be done using symbolic methods rather than resolving to numerical calculations.

Example: The possible geometry of the boundary conditions depends on the type of PDE; inappropriate boundary conditions should be rejected at this stage.

Algorithm Selection

The complete problem specification given by the user should then be passed to a knowledge base in order to find a suitable solution algorithm. Whenever the problem specification is incomplete the system ought to prompt the user to give more information. Also if there are more than one solution algorithm available the system should seek assistance from the users in order to make the decision.

If the system is to be useful for the users unfamiliar with CASs and numerical libraries there has to be a technique to decide the method of solution suitable for the given problem. We propose to achieve this by building up a knowledge base. The mathematical solutions that can be used depend very much on the nature of the parameters in the given equations.

For example the methods that can be used to solve a system of equations given by $\boldsymbol{A}\underline{x} = \underline{\boldsymbol{b}}$ depend on the size and the characteristics of $\boldsymbol{A}$. The database will have the solution methods needed for all the cases of $\boldsymbol{A}$. The user need not be aware of the different solution methods. These decisions can be handled by CASs. The advantage of using CASs in selecting the algorithm is that the analysis of the parameters can be done efficiently using symbolic manipulation.

In order to implement the solution the algorithm has to be properly analysed to identify the sections that can be implemented symbolically. One disadvantage of using only one CAS is that the level of symbolic manipulation that can be achieved is restricted to the functions of that CAS. Providing an interface that can include more than one CAS would help to overcome this restriction.

Implementation of the Solution: The outcome of the problem analysis phase is either a solution method, or the system has not been able to find one, in which case the system ought to seek the user's help. Once the algorithm has been selected, the system has to pass the required parameters to these sub routines. When the problem analysis phase has identified the required algorithm, the next step is to implement it using symbolic and numerical solvers. A major problem faced in integrating the solvers is the closed nature of CASs. CASs are developed for end user interaction and therefore it is hard to use them in any other platform to behave as intermediate systems.

The design issues related to calling the symbolic and/or numerical sub routines are not discussed in this paper.

Modularity and Re-usability: An advantage of the proposed method is these sub routines are not specific to any method of solution. The same subroutine can be used by more than one method of solution.

Partially simplified results from the subroutines may be reused for instance when a formula resulting from symbolic manipulation is evaluated repeatedly with different sets of parameters. This saves time which could otherwise have to be spent on redoing the time consuming symbolic preprocessing.

Evaluation of Results: The idea here is to use the extensive graphical capabilities of CASs to present the results in a user friendly and easy to analyse method. Also users may want to rerun or use a different solution method after evaluating the results. We aim to provide facility for re-evaluating partial results where necessary. This enables the user to change the algorithm or reformulate part of the problem while retaining reusable structures as possible.

Implementation Issues

Integrating Solvers

As described earlier the system design involves designing a method for integration of CAS and numerical routines and providing means of user interaction. The integration between different solvers can be achieved in many ways. A very general approach is to build up a common interface to communicate with the solvers. This facilitates a complete integration transparent to users (Figure 2).

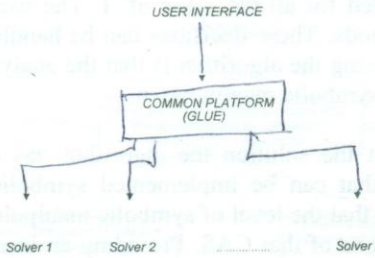


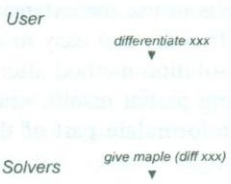
Figure 2: Overview of the system architecture

The common platform can be designed in a modular way to achieve the following functions.

- Communication with the solvers: passing them the sub-problems to be solved, getting the results and passing these results onto other solvers (if necessary), passing information between solvers, etc.
- Synchronisation of the solvers, formulating the output from partial solutions received from solvers.
- User interaction with the user in order to get the necessary user intervention.
- Pattern matching, transformations etc. between the output from one solver and input to another.

These functions can be achieved by implementing the common command level in the way of a submodule architecture. The submodule architecture will facilitate modifications to the system easily.

Users will use a common language to give the instructions to the system and the system will, depending upon the user instructions, issue the commands to solvers. The matching between these two will have to be done by the submodule. For example:



The front end of the system may be implemented either as a part of a knowledge base or the language of the symbolic solver used. Examples for both techniques are among the systems discussed under existing coupled systems.

These implementation issues need to be discussed in detail at implementation stage. Such a detailed description of system design will have to also address the following issues of

- identifying all the required data types
- how to define the abstract data types (ADTs)
- how to implement these data types etc.

Use of a knowledge base

A knowledge base oriented approach can be used to select the appropriate algorithms. The knowledge base that can be used here is based on a database of solution algorithms indicating the symbolic and numerical subroutines and under which conditions each of them is suitable. The knowledge can be extracted from presently available algorithms. Then an expert system can be built to access the knowledge base.

Knowledge bases for the integration of symbolic and numerical methods have been considered by many researchers [26]. Russo, Peskin and Kowalski [27] have shown how to use a knowledge based system to automatically generate the numerical programs for the solution of problems based on continuous partial differential equations. A CAS called PRESS (PRolog Equation Solving System) [28] is used for symbolic manipulations. Re-write rules for symbolic simplifications are written in PROLOG. The problem is analysed by the knowledge based system and then a numerical method is selected based on the problem characteristics. The system uses symbolic methods to discretise and rearrange the set of equations to obtain a form suitable for the numerical method. Mutrie, Char and Bartels [29] have also discussed the use of a knowledge based system in combining symbolic algebra and numerical computations. They have also given attention to expression optimising in such systems. Techniques similar to these can be used in the proposed method to select the algorithms required.

Case Studies

Newton's Method for Unconstrained Optimisation

This example shows how manual calculations can be achieved symbolically and how such symbolic solutions can then be used in numerical methods.

Problem:

Find the optimum value of a given expression with n variables without any constraints. This is called unconstrained optimisation.

$$Minimise \{g(x) \mid x \in R^n\}$$

In Newton's method, we start with an initial guess for the set of variables and keep on changing these values in order to minimise the function value. Let $g(x_i)$ be the given function.

Algorithm:

It is assumed that an initial estimate $x^{(0)}$ of $x^{(k)}$ is known.

Step 1: Set $k=0$

Step 2: Compute $g^{(k)}$ and $G^{(k)}$ from

$$g_i^{(k)} = \partial_i g(x^{(k)}) \qquad (i=1,\ldots,n)$$

$$G_{ij}^{(k)} = \partial_i \partial_j g(x^{(k)}) \qquad (i=1,\ldots,n)$$

Step 3: Compute $p^{(k)}$ by solving the system of linear equations:
$G_{ij}{}^{(k)} p^{(k)} = - g_i{}^{(k)}$

Step 4: Compute $x^{(k+1)} = x^{(k)} + p^{(k)}$

Step 5: Check whether $p^{(k)} \leq$ required tolerance, if so stop; otherwise set $k = k+1$ and go to step 2.

Implementation:

The Symbolic computing tools such as Maple [10] can perform Step 2 (or rather the symbolic partial differentiation). So we can have the grad ($g_i{}^{(k)}$) and Hessian ($G_{ij}{}^{(k)}$) and also inverse of the Hessian matrix ($[G_{ij}{}^{(k)}]^{-1}$) in symbolic form as output from a Maple session. Maple allows the users to convert output to a Fortran 77 format. Then we can make use of numerical methods to perform the rest of the algorithm.

CONCLUSION

We have proposed a method that can be used to implement an environment to combine symbolic and numerical methods in solving equational systems. We have implemented the proposed method with some examples. With the help of these examples we have identified the requirements of the different phases of the method. However this paper does not include the extensive implementation details of these phases. The proposed method allows the users to interactively use the features of both symbolic and numerical solvers in many application areas.

REFERENCES:

[1] W Press, B Flannery, S Teukolsky, and W Vetterlin, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, 1989.

[2] C Lawson, R Hanson, D Kincaid, and F Krough, "Basic Linear Algebra Subprograms for Fortran usage", in *ACM transactions on Mathematical Software*, pages 308--323, September 1979.

[3] J Dongarra, J Croz, S Hammarling, and I Duff, "A set of level 3 Basic Linear Algebra Subprograms", in A*CM Transactions on Mathematical Software*, March 1990.

[4] J Dongarra and S Ostrouchov, *LAPACK Working Note 81 - Quick Installation Guide for LAPACk on Unix Systems*, Technical report, University of Tennessee, 1994.

[5] E Anderson and et.al. *LAPACK Users' Guide: 2nd Edition*, SIAM, 1995.

[6] R Barrett, M Berry, T F Chan, J Demmel, J Donato, J Dongarra, V Eijkhout, R Pozo, C Romine, and H Van der Vorst, *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods*, 2nd Edition, SIAM, Philadelphia, PA, 1994.

[7] K Schulze and C Cryer, "NAXPERT: a prototype expert system for numerical software", in *SIAM Journal of Scientific and Statistical Computing*, May 1988.

[8] M Dewar, "Using computer algebra to select numerical algorithms", in *ISSAC 92*, 1992.

[9] G Rayna, *REDUCE: software for algebraic computation*, Springer-Verlag, 1987.

[10] B Char, K Geddes, G Gonnet, B Leong, M Monagan, and S Watt, *Maple V - Library Reference Manual*, Springer-Verlag, 1991.

[11] S Wolfram, *Mathematica: A System for Doing Mathematics by Computer*, Addison-Wesley Publishing Company, 1991.

[12] E Engeler and R Mader, "Scientific Computation: The Integration of Symbolic, Numeric and Graphic Computation", in *EUROCAL '85: European Conference on Computer Algebra, Vol.1. Invited lectures, Lecture notes in computer science 203,* B Buchberger, editor, Springer-Verlag, 1984, pages 185-200.

[13] D Bayer and M Stillman, "The design of Macaulay: A system for computing in algebraic geometry and commutative algebra", in *proceedings of the 1986 Symposium on Symbolic and Algebraic Computation: SYMSAC '86*, Bruce W Char, editor, Association for Computing Machinery: New York, 1986.

[14] J Rice and R Boisvert, *Solving Elliptic Problems using ELLPACK*, Springer-Verlag, 1984.

[15] S Weerawarna, E N Houstis, and J R Rice, "An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers", in *Symbolic and Numerical Computation for Artificial Intelligence*, Academic Press, 1992, pages 303-321.

[16] E Kant, "Synthesis of mathematical modeling software", in *IEEE Software*, May 1993.

[17] R Akers, E Kant, C Randall, S Steinberg, and R Young, *Problem solving environments and solution of partial differential equations, Version 1*, Technical report, SciComp Inc., 1997.

[18] P Fritzson, L Viklund, J Herbert, and D Fritzson, "Industrial application of object-oriented mathematical modelling and computer algebra in mechanical analysis", in *TOOLS EUROPE '92*, 1992.

[19] V Engleson, P Fritzson, and L Viklund, "Variant handling, inheritance and composition in the ObjectMath computer algebra environment" in *DISCO '93*, 1993.

[20] P Iglio, *Applying software components technology to computer algebra*, Technical report, NAG/FRISCO Consortium, 1997.
http://extweb.nag.co.uk/projects/FRISCO/reports/component.ps.

[21] N Kajler, "CAS/PI: a portable and extensible interface for computer algebra systems", in *ISSAC '92*, 1992.

[22] N Kajler, "User interfaces for symbolic computation: a case study", in *UIST '93*, 1993.

[23] S Dalmas, M Gaetano, and S Watt, "An Openmath 1.0 implementation", in *ISSAC '97*, 1997.

[24] S Dalmas, M Gaetano, and A Sausse, *ASAP: a protocol for symbolic computation systems*, Technical report, INRIA, Sophia-Antipolis, March 1994.

[25] S Gray, N Kajler, and P Wang, "MP: A protocol for efficient exchange of mathematical expressions", in *ISSAC '94*, 1994.

[26] N Jacobstein, C Kitzmiller, and J Kowalik, "Integarting symbolic and numerical methods in knowledge-based systems: Current status, future prospectus, driving events", in *Coupling Symbolic and Numerical Computing in Expert Systems II*, J Kowalik and C Kitzmiller, editors, Elsevier Science Publishers, 1988.

[27] M Russo, R Peskin, and A Kowalski, "Using symbolic computation for the automatic development of numerical programs", in *Coupling Symbolic and Numerical Computing in Expert Systems II*, J Kowalik and C Kitzmiller, editors, Elsevier Science Publishers, 1988.

[28] A Bundy and B Welham "Using Meta-Level Inference for Selective Application of Multiple Rewrite Rules in Algebraic Manipulation", in *5th Conference on Automated Deduction, Lecture notes in computer science 87*, W Bibeland and R Kowalski, editors, Springer-Verlag, July 1980 pages 24--38.

[29] M Mutrie, B Char, and R Bartels, "Expression optimization in a symbolic-numeric interface", in *Coupling Symbolic and Numerical Computing in Expert Systems II*, J Kowalik and C Kitzmiller, editors, Elsevier Science Publishers, 1988.

[30] D Manocha and J Canny, "*Real time inverse kinematics for general 6R manipulators*", Technical report, University of California, Berkeley.