A Scientific Programming Environment For a Class of Inverse Problems

Bart Childs **Tim McGuire**
Department of Computer Science
Texas A&M University
College Station, TX 77843-3112
bart@cs.tamu.edu mcguire@cs.tamu.edu

Abstract

A prototype scientific programming environment for solving boundary value problems in ordinary differential and integro-differential equations is presented. This environment is designed to assist scientific programmers in overcoming the semantic gap between the formulation of the mathematical model of a physical system and the code for solving that model. The environment includes the following concepts:

- Knuth's Literate Programming where the user can document the mathematical model in the TEX or LATEX system.
- The use of a Taylor series based integrator which gives efficiency and stability at the cost of higher overhead in code preparation.
- Symbolic computation to reduce the overhead in the previous item.

Most of the environment is based upon the use of a public domain and portable editor (*gnu-emacs*), [STALLMAN]. This paper specifically addresses the part of conversion of a portion of the TEX code to higher level language (HLL) code such as C or FORTRAN. This conversion is tedious and people are not well suited to it.

Automating these procedures enables the use of codes that solve these problems with a fraction ($\approx$ 3 to 20%) of the usual CPU requirements. The higher accuracy requirements gave the best speed improvements.

The Taylor series has a characteristic of covering a larger range of the independent variable than the usual Runge-Kutta or multistep methods. This obviously requires a larger fraction of the computation in the generation of the series. This increased computation to communication ratio should be advantageous for parallel computers.

Boundary Value Problems in ODEs

We will pose three problems that can be formulated in a manner appropriate for solution as a multipoint boundary value problem.

- The aerodynamic drag and rolling friction of an automobile is critical to its efficiency. Many would suggest that the placement of the vehicle in a wind tunnel and on a treadmill is a means of determining these parameters. We propose that gathering a simple set of data from one coasting run can yield a more accurate estimation. This would use a second order differential equation with several parameters.
- Satellite orbits are affected by the reality that nature's gravitational bodies are not perfect spheres. The use of a lumped mass model of the bodies can make calculations of corrective propulsions quite convenient. This would use a sixth order differential equation with many unknown parameters.
- The eco-systems of the earth are now being modeled for performance and critical status. The model of a one acre stand of trees could be modeled as a first or second order differential equation. It can also be argued that such an equation should be written for each tree and feedback from neighboring trees be included. For this small size, we would use hundreds of differential equations and parameters.

In all these cases, we would probably use many more observations of the phenomena that we have equations. These are typical applications.

Basic Theory

Multi-point boundary value problem in ordinary differential equations can also be described in the familiar terms of "regression analysis" where the model is the solution of a set of (non)linear ordinary differential equations. There are usually many more observations (boundary conditions) than the number needed to specify a unique solution. Further, these observations are naturally imprecise. The model is known but in addition to just solving the differential equation there is also the need to determine the constant parameters in the differential equation.

The solution of the problem is achieved by the use of superposition methods (often called shooting methods.) A sequence of initial value problems with *assumed* initial values is generated and these solutions are *superimposed* to satisfy the boundary conditions.

The superposition of the solutions results in a system of algebraic equations. Solving this gives a "better" estimate of the initial values for another iteration. These iterations are repeated until there is no significant change in the initial values. One might consider this to be *Regression Analysis* where the model is a differential equation.

The following problem is offered as a simple but nontrivial example. It is a model of a *spring mass dashpot, electrical circuit, or other physical systems.* This is a linear problem if the parameters are known.

$$\ddot{x} + \mu\dot{x} + \xi x = \lambda sin(t) \tag{1}$$

We realize that these parameters will not be a linear function of data like:

t_i	1.0	2.0	3.0	4.0	5.0	6.0	$\cdots$
$\ddot{x}(t_i)$	-0.220	0.035	-0.474	-0.589	0.393	1.597	$\cdots$

The *constants* μ, ξ, and λ are unknown and their estimation is a primary goal. The values of $x(0)$ and $\dot{x}(0)$) are also not linearly dependent upon the data.

The linear multipoint boundary value problem is to find the solution of the differential equation:

$$\dot{y} = Ly + f \tag{2}$$

subject to the m boundary conditions:

$$q_i(y(t_i)) = b_i, \quad for \quad i = 1, 2, ..., m \tag{3}$$

where:

y is the state vector of n elements,

L is a linear operator that is a variable or constant coefficient matrix,

t is the independent variable, often *time.*

f is an n element vector function of the independent variable t, and

$(\dot{})$ denotes differentiation with respect to t.

q_i is an operator that defines a linear combination of the elements of the state vector, y, that is equal to the boundary value b_i at $t = t_i$.

Since the constant coefficients of the differential equation are unknown, the vector differential equation is nonlinear. Thus, we rewrite equation 2 as:

$$\dot{y} = g(y, t) \tag{4}$$

It is of order five, because x, $\dot{x}$, and three parameters are not known (at $t = 0$). The elements y_3, y_4, and y_5 are μ, ξ and λ, respectively. The five elements of equation 4 for the example of equation (1) are:

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= -y_1 y_4 - y_2 y_3 + y_5 \sin(t) \\ \dot{y}_3 &= 0 \\ \dot{y}_4 &= 0 \\ \dot{y}_5 &= 0 \end{aligned} \tag{5}$$

If $w(0)$ is the best available estimate for the initial values of $\dot{y} = g(y, t)$, then we can formulate a "linearized" approximation $z(t)$ of $y(t)$. The truncated Taylor series about w is:

$$\dot{z} = g(w, t) + \frac{\partial g}{\partial w}(z - w) \tag{6}$$

Superposition Methods

In the usual superposition method, the solution of a linear differential equation is expressed as the weighted sum of n linearly independent homogeneous solutions upon a particular solution.

The solution can also be written as the sum of $n+1$ particular solutions, given that a certain condition (or constraint) is met. We write:

$$y = \sum_{j=0}^{n} P_{(\cdot,j)}\beta_j = P\beta \tag{7}$$

where:

β_j are superposition coefficients and

P is a matrix whose j^{th} column is denoted by $P_{(\cdot,j)}$.

The jth column of P is a solution of the ODE:

$$\dot{P}_{(\cdot,j)} = LP_{(\cdot,j)} + f \qquad j = 0, 1, 2, \ldots, n \tag{8}$$

We multiply each of these equations by the appropriate superposition coefficient, β_j, and sum over the indicated range which yields:

$$\sum_{j=0}^{n} \dot{P}_{(\cdot,j)}\,\beta_j = L\Big(\sum_{j=0}^{n} P_{(\cdot,j)}\,\beta_j\Big) + f\sum_{j=0}^{n}(\beta_j) \tag{9}$$

Comparing this with equation (3) and its derivative with respect to t, we see that:

$$\sum_{j=0}^{n} \beta_j \equiv 1 \tag{10}$$

Boundary Condition Specification

The superposition coefficients for this definition are found by substituting into the boundary conditions. The system of equations which must be solved to determine the superposition coefficients are:

$$C\beta = d \tag{11}$$

where the elements of these arrays are:

$$\begin{aligned} C_{(0,j)} &= 1 \quad for\ j = 0, 1, 2, \ldots, n \\ C_{(i,j)} &= q_i\big(P_{(\cdot,j)}(t_i)\big) \quad for \begin{cases} i = 1, 2, \ldots, m \\ j = 0, 1, \ldots, n \end{cases} \\ d_0 &= 1 \\ d_i &= b_i \quad for\ i = 1, 2, \ldots, m \end{aligned} \tag{12}$$

Constrained Least Squares

This is a system of over-determined equations 12, but the first is to be met exactly. Thus, this is a *constrained least squares* problem. The overdetermined set of equations is well know to be **non-singular** and **ill-conditioned**. Problems of low order can be solved by forming the usual normal equations of regression analysis.

Many problems will require the use of techniques such as singular value decomposition or Gram-Schmidt orthogonalization. These procedures also yield the matrices that are needed to calculate covariences and confidence estimates. Many problems also have a mixture of boundary conditions that are observations and constraints (such as continuity equations). The observations are to be met in a best fit sense, but continuity implies exactness. This dictates the need for some generality in the constrained least squares routines.

Taylor Series Integrators

The Taylor series is one of the most common and important tools in applied mathematics. The solution of ordinary differential equations can be expressed in this form. The usual integration formulae, Runge-Kutta and multistep methods, are based upon the approximation of the first few terms of Taylor series.

One disadvantage of the Taylor series as an integrator is that the formal manipulation of these power series can often be quite complex and require detailed manual preparation. The use of symbolic computation to develop the recurrence relations largely overcomes the problems. This is worth considering since computational timings generally indicate an 80 percent reduction in the CPU requirements with these integrators. Our studies indicate that with current systems this reduction is more likely to be in the range of 93 to 97 percent, [McGuire, Childs]. These comparisons used standard IMSL libraries and the problems discussed in the classic paper, [Hull].

The resultant series will be valid in a region of convergence. This finite region can be extended by a process similar to analytic continuation.

Fröbenius Recurrence Equations

The method of Fröbenius is an economical means of calculating these coefficients through the use of recurrence equations rather than using the repeated differentiation. We will describe various routines which will expedite some of the formal manipulations necessary to solve a fairly broad subset of the common differential equations. For the differential equation 4, we assume that the solution y and the right-hand-side can be expanded in power series. Since y is a vector, the coefficient of t^k is a vector and is denoted by $y_{(k,\cdot)}$. Thus:

$$y(t) = \sum_{k=0}^{\infty} y_{(k,\cdot)} t^k \tag{13}$$

$$g(t) = \sum_{k=0}^{\infty} g_{(k,\cdot)} t^k \tag{14}$$

$$\dot{y}(t) = \sum_{k=0}^{\infty} (k+1) y_{(k+1,\cdot)} t^k \tag{15}$$

The upper limit of these summations is theoretically ∞, but the practical use requires these be finite, say m. Tests of some years ago indicated a value of $m = 12$ was the best to give economical precision.

Substituting the second and third of equations 14 and 15 into 4 yields:

$$\sum_{k=0}^{m} (k+1) y_{(k+1,\cdot)} t^k = \sum_{k=0}^{m} g_{(k,\cdot)} t^k \tag{16}$$

This equality requires that coefficients of like powers of t must be equal. Selecting the $(k-1)^{\text{th}}$ term, we get a Fröbenius recurrence:

$$y_{(k,\cdot)} = \frac{1}{k} g_{(k-1,\cdot)} \tag{17}$$

The identification process will require linearized equations which will be based upon the base or approximate solution, w. Equations 6 for this model are:

$$\begin{aligned} \dot{y}_1 &= y_2 \\ \dot{y}_2 &= w_4 w_1 + w_3 w_2 + y_5 \sin(t) \\ &\quad - w_4 y_1 - w_3 y_2 - w_2 y_3 - w_1 y_4 \end{aligned} \tag{18}$$

Equations 17 for this model are:

$$\begin{aligned} y_{(k,1)} &= (y_{(k-1,2)})/k \\ y_{(k,2)} &= \{(w_4 w_{(k-1,1)} + w_3 w_{(k-1,2)} \\ &\quad - w_4 y_{(k-1,1)} - w_3 y_{(k-1,2)} \\ &\quad - w_{(k-1,2)} y_3 - w_{(k-1,1)} y_4 + y_5 sn_{(k-1)})\}/k \end{aligned} \tag{19}$$

Literate Programming

Knuth's *literate programming* is a development that should affect scientific programming. His thesis was that it should be just as important to communicate with the other persons who read the program as it is to communicate with the computer [Knuth].

Literate programming is the integration of two important concepts. One is that of *structured programming* [Dahl]. Structured programming is the creation of a program by joining together or nesting logical units that are either structured programming units themselves, or are in the form of a well-defined control structure. A second concept, sometimes referred to as *pretty-printing*, deals with the presentation of code on the printed page in a readable fashion and reflect its structure.

Symbolic Algebra Systems

Some of the general purpose symbolic algebra programs in current use are MACSYMA, Maple, *Mathematica*, DERIVE, REDUCE, and ScratchPad II.

One problem plaguing symbolic computation systems is that they do not generally interact conveniently with other software systems. Most are stand-alone systems using their own unique syntax. Systems often do not provide adequate access to commonly used numeric libraries. Furthermore, most systems cannot be used as a "symbolic engine" by other software. Symbolic computation systems needs better software to enable interface software with other codes.

It is illustrative of the limitations of current symbolic systems that the symbolic manipulations performed for this research were performed using a program written specifically for that purpose. The systems available were not flexible enough to accept input directly in the standard form chosen, nor to manipulate the expressions to produce the required output format. For example, in *Mathematica* it is possible to generate output for certain types of expressions in TEX, FORTRAN, or C form but it does not accept input in any of these forms.

A *Mathematica* script was written to derive the Frobenius recurrence relations from the translated TEX form. The method chosen worked well with constant coefficient ODEs. Variable coefficient ODEs led to unacceptably complicated recurrence relations when expanding about a center other than the origin.

No acceptable resolution of this difficulty was found. It was decided to develop a parser system for translating the TEX description into HLL code using the *unix* tools *lex* and *yacc*.

Thus, *Mathematica* is not part of the current prototype, but it may be a part of future versions. It might be helpful in simplifying complex expressions on the right hand side of the ODE system or in translating ODEs of arbitrary order into first order systems.

A Prototype Environment

Issues in Programming Environments

A literate programming environment specifically for scientific programming has been proposed by one of the authors [McGuire]. Programming environments which do not fit the literate paradigm have been proposed for scientific programming: these include PROTRAN [Rice], Interactive ELLPACK [Dyksen], and R^n [Carle]. An environment for literate scientific programming should be cognizant of the following characteristics of scientific programming:

1. Scientific programming is becoming *multi-lingual*. Most "number crunching" is still performed in FORTRAN but other languages are perhaps better suited for specifying input, output, and graphical interpretation. A significant amount of scientific computation is now being done in C and C++.
2. The "native tongue" of scientific thought is mathematics [Dantzig]. It is not always straightforward to translate from one language to another, and a literate scientific programming environment should aid this translation. There is a 'semantic chasm' between $Ax = b$ and subprogram calls with perhaps seven arguments.
3. It is critical that scientific codes be efficient, since the majority of the processing is CPU-bound. It is common for scientific code execution times to be measured in days even on the fastest available computers.
4. Scientists and engineers now have access to a wide spectrum of systems, ranging from personal workstations to supercomputers. Any task to be performed should be run on a system appropriate to that task.

We will describe a prototype of an environment for scientific programming. This prototype is a symbolic processor interface with an extensible editor, *gnu-emacs* [Motl], for use with FWEB [Krommes].

Design of the Environment

Previous systems have required the user to learn a new input mechanism for describing the ODE system. The decision was to use a TEX-based input medium, since

- many scientists and engineers use TEX for writing technical documents,
- TEX possesses a structured mechanism for describing mathematics, and
- the major system for implementing literate programming is WEB.

The environment should be able to linearize systems of non-linear ODEs. This linearization should be incorporated in this environment because this is the most tedious and error-prone part of the solution process.

The environment will generate a power series solution using the power series operators described in [McGuire, Childs]. The easy transformation of an ODE system to a form suitable for a Runge-Kutta integrator such as DVERK disappears when we use the method of Frobenius. This method is relatively straightforward to perform by hand if the system is simple. If the system is more complex, especially if it involves boundary value problems requiring a linearization technique, some form of automated assistance would be extremely helpful.

The power series operators implement either binary or unary operations. For this reason it is advantageous that the ODE system be translated into a canonical system of algebraic differential equations with one operator on the RHS of each equations. This necessitates the introduction of temporary variables. An advantage of this intermediate step is that it can be typeset in TEX format if desired and included in the documentation. In addition, it allows for hand optimization if desired. Once the canonical system has been generated, the second phase translates it into HLL code.

Current Status

The method works for a significant class of ODE systems. It was implemented using standard LR parsing techniques to perform the necessary symbolic manipulations. Standard symbolic computation systems were evaluated for possible use in the prototype, but they are not used at this stage.

There are several issues that need to be addressed in the near future. The prototype system has the following limitations in the current form:

1. There are undue restrictions on the form of the input (names of variables, etc.)
2. Not all the functions we'd like to see are implemented, and there are restrictions on the arguments to these functions.
3. The error diagnostics are extremely limited. Some of this is a limitation of the *yacc* system. The migration to a *gnu-emacs* mode will need to be planned in such a way as to give better error diagnostics.
4. The linearization of non-linear multi-point boundary value problems is not yet implemented.
5. The environment is not seamless; that is, the parsers are run independently of the rest of the environment and their output is included in the user program by a "cut-and-paste" method.

Intelligent Scientific Computation

We now offer a correlation to the *Critical Issues* listed by Chairperson Kant in her "working notes" document.

- We started this work as an extension of a programming environment that has great advantages for scientific computing. (Session 2.)
- We chose the environment because it utilizes the TEX description of the model (ODEs) (Session 3.)
- The speed advantages that the procedures give are reasonable for big problems only if symbolic computations can do the differentiation and translation of the resulting code into its Fröbenius recurrence form (Session 5.)
- The characteristic of the Taylor series integrator (as opposed to a spline form) of avoiding a function minimization is necessary because it is unknown how to weight that functionality with that necessary in the natural specification of the observations (Session 4, Discretization, ...)
- The data provided in typical applications like that of forest eco-systems is a candidate for careful review of the input and correlation with the output. The best of these records span several decades, were done by different people, and have many inconsistencies. (Session 7-8 Data Analysis, Visualization, ... are techniques that need to specific interfaces for 'cleaning' the input and correlation of the output.)

References

A. Carle, K. D. Cooper, R. T. Hood, K. Kennedy, L. Torczon, and S. K. Warren, 1987. "A Practical Environment for Scientific Programming," *Computer*, 20(11): 75–89.

O. J. Dahl, E. Dijkstra, and C. A. R. Hoare, 1972. *Structured Programming*, London: Academic Press.

T. Dantzig, 1930. *Number — The Language of Science*, New York: Macmillan.

W. R. Dyksen and C. J. Ribbens, June 1987. "Interactive ELLPACK: An Interactive Problem-Solving Environment for Elliptic Partial Differential Equations," *ACM Transactions on Mathematical Software*, 13(2): 113-132.

T. E. Hull, W. H. Enright, B. M. Fellen, and A. E. Sedgewick, Dec. 1972 "Comparing Numerical Methods for Ordinary Differential Equations," *SIAM Journal of Numerical Analysis*, 9(4): 603–637.

D. E. Knuth, 1992. *Literate Programming*, Center for the Study of Language and Information, Stanford University.

J. A. Krommes, 1989. "The WEB System of Structured Software Design and Documentation for Fortran, Ratfor, and C," Available for anonymous ftp from `lyman.pppl.gov`.

T. McGuire and B. Childs, 1991. "An Analysis of Power Series Operators," *Applied Mathematics Letters*, 4(2): 45–48.

T. McGuire, August 1991. "The Use of Symbolic Methods in the Development of Efficient Power Series Integrators." Ph.D. dis., Dept. of Computer Science, Texas A&M University.

M. B. Motl, Dec. 1990. "A Literate Programming Environment Based on an Extensible Editor." Ph.D. dis., Dept. of Computer Science, Texas A&M University.

J. R. Rice, 1983. *Numerical Methods, Software, and Analysis, IMSL Reference Edition*, New York, NY: McGraw-Hill.

R. M. Stallman, Oct. 1986. *GNU Emacs Manual*, Fifth Edition, Version 18, Cambridge, MA: Free Software Foundation.