Mathematical Software: Plod

Is user-easy software for "stiff" equations merely a fond dream? Not with the Plod approach.

Elvira Agron
National Institutes of Health

I-Lok Chang
The American University

Gamini Gunaratna

David K. Kahaner
National Bureau of Standards

Martin A. Reed
IBM Corporation

Physical models—even those described as "routine"—can pose some interesting challenges. First of all, they can involve hundreds, or thousands, of equations. These equations range from simple algebraic statements of conservation or symmetry to relations between variables at discrete time units. They include derivatives, integrals, or complicated functionals of the unknowns. Consequently, describing the model mathematically can become tedious. Another challenge is that an approximate solution, generated numerically, must eventually take a form that is palatable to users. The achievement of such a form inevitably changes the values of constants, approximations, or even the model itself.

By a *model*, we mean an initial value problem[1] given by a set of ordinary differential equations. Many models are modest, involving only a few dozen ODEs and a similar number of parameters. These models occur frequently in dynamics, such as Josephson junctions.[2] Although the problems in this group may be small, they are among the most taxing to solve numerically because of instability with respect to initial conditions, or "stiffness." (See box on the next page.) One can rarely solve models in closed form, but some of them are easy to integrate numerically. This type of model often occurs as an example in classroom situations.

Although a number of simulation software packages are available on the market, their cost, scope, capability, and quality vary tremendously. The journal *Simulation*, a good source for information on available software, has published a compendium of almost 50 such packages exclusively for microcomputers.[3] Among them, the Advanced Continuous Simulation Language package is typical and well known.[4] Some packages have good integrators, but many use explicit Runge-Kutta integration, which is inadequate for stiff ODEs.

In view of the foregoing, we designed Plod (plotted solutions of ordinary differential equations) mathematical software to address the difficulties under discussion. In addition, we wanted to provide a problem-solving tool for the expanding number of micro users unfamiliar with programming techniques. Consequently, we proceeded with the following criteria in mind:

- The physical problem can be described by 25 ODEs and 10 parameters.
- The problem is solvable while the user is at the terminal (Plod is entirely interactive).

- Plod is easy to use; no manual is required.
- The numerical methods are of high quality.
- Rapid, flexible, and attractive graphics (not publication quality) are included.
- Plod is mostly portable.
- Plod is in the public domain.

In discussion of these criteria, Plod's 25 ODE/10 parameter description addresses some application and prototype problems.[5] However, Plod is inappropriate for studying, say, the complete suspension system of an automobile. It also excludes any problems that require large amounts of CPU time. The latter conditions can occur in "chaotic" systems[2] with many abrupt changes, or when the user wants to integrate for long periods of simulated time. Plod also excludes problems that need to be repeated automatically to generate an average as a parameter is changed. We designed Plod for "what if" studies and as a prototype for more ambitious simulations.

Plod includes two files with installation and tutorial information. Users should read them, but once Plod is installed, even first-time users can proceed directly. Keyboard blunders do not prevent the package from working. It is not possible to anticipate all situations, but error recovery was a major design goal. Thus Plod is also suited for classroom use.

In regard to numerical methods, Plod's integrator, DDRIV,[6] is a modern double-precision implementation of an algorithm described by Gear.[1] This well-known algorithm is in widespread use. DDRIV implements variable-step, variable-order Adams and Gear formulas, with the ability to solve stiff ODEs. Plod's results are as accurate and reliable as current technology permits.

After reading installation and tutorial information, even first-time users can proceed directly.

To promote portability, we did not use special characteristics of a particular computer unless it was unavoidable. Plod is not entirely portable because of graphics and screen control, but we isolated these requirements and kept them to a minimum. The current target environment is an IBM PC XT/AT (discussed later), although versions have been moved to a Sun workstation and a Univac/Tektronix combination.

Instability and Stiffness

The solution curve of a specific initial value problem is insufficient in itself to determine whether the problem will be difficult to solve numerically. More important is the solution family of the ODE, that is, the set of solutions (curves) for all possible initial values. A numerical method tries to follow one member of the family (for example, the one) through the given initial point. This process can never be exact: The calculated values can be thought of as exact points on another family member. If curves in the family separate rapidly from the exact solution of the initial value problem, then no numerical method will give accurate results. Such a problem is said to be unstable with respect to initial conditions; small changes in the initial values result in a very different solution curve. Most engineering problems are, at worst, very mildly unstable in this sense.

Efficient integration of some "stiff" ODEs is a major advance. We use this term generically to describe a system that requires many small integration steps, even when the solution is slowly changing. Actually, whether a problem is stiff or not depends on a combination of factors such as the ODE, the initial conditions, the interval on which the solution is required, the numerical method employed, and seemingly minor details of implementation.

For many engineering applications, a workable notion is that a problem is stiff if it contains time constants whose values vary, at any given time, by four or more orders of magnitude. Almost every area of application leads to such problems, but one of the most common areas is chemical kinetics. Reaction rates of different species can easily vary by 10 or 20 orders. Mathematically, time constants of the model are associated with eigenvalues of the Jacobian matrix, $\delta f/\delta y$, and large differences in the magnitude of these eigenvalues are often symptomatic of stiffness. Readers can find many useful expository articles, papers, and references in *Stiff Computation*.[1]

References

1. R.C. Aiken, ed., *Stiff Computation*, Oxford, England, 1985.

Overview of Plod

Plod is a package for the solution of a set of ODEs subject to specified initial conditions,

$$\begin{aligned} y_1' &= f_1(t, y_1, y_2, \ldots, y_n) & y_1(a) &= Y_1 \\ y_2' &= f_2(t, y_1, y_2, \ldots, y_n) & y_2(a) &= Y_2 \\ &\vdots \\ y_n' &= f_n(t, y_1, y_2, \ldots, y_n) & y_n(a) &= Y_n. \end{aligned}$$

The solution is desired on an interval $[a, b]$. Most problems also involve physical parameters whose values determine the character of the solution. The previous equation does not show these explicitly.

We designed Plod to be used in two interactive steps: Plod0 and Plod1. Plod, a batch program, runs Plod0 and optionally does the processing to pass to Plod1. During Plod0 the user enters the ODEs. No integration occurs during this step, and numerical values are not requested for any of the variables or parameters. Plod0 is a preprocessor with a Fortran program as output. The output program includes the ODEs and has a mechanism to communicate needed information to the Plod1 step.

Plod automatically compiles the output Fortran program and links it to precompiled modules supplied with the package. The result is an executable program that we call the Plod1 step. Plod1 prompts for parameter and initial values, and for the interval on which the integration is to take place. After the integration, you can generate graphs and listings. You can also change parameter values, initial conditions, and the integration interval, for example. You may experiment with the problem, examining results under different conditions. A sophisticated user can also alter the integration method and make other changes of interest to a specialist in numerical integration. During Plod1, you cannot alter the functional form of the model by adding terms, or by adding or removing equations. Those actions require returning to the Plod0 step and recompiling.

If Plod did not use a compiler, you could easily flip between changing the model and integrating the equations. The ODEs would be interpreted. We have developed a compiler-free version for our own use. Not much is lost in simple problems, but the integration of difficult problems slows by a factor of 10.

What do you need to run Plod?

Plod requires an IBM PC XT/AT (running MS-DOS) with a math coprocessor and a memory of at least 360 Kbytes. An executable file in the Plod1 step averages 270 to 300 Kbytes for five to 10 equations. Plod also runs on an IBM PC, but a hard disk is more convenient. Teachers could run Plod0, compile and link steps on an XT or AT, and distribute copies of the executable output for PC use.

Graphics necessitate an IBM enhanced graphics adapter, an IBM color graphics adapter, or a Hercules graphics card and associated monitor. Plod runs automatically at highest resolution on an EGA, CGA, or HGA, and also runs on an IBM professional graphics adapter in CGA mode. Because the HGA does not have any color capabilities and because some IBM compatibles do not support color, Plod does not utilize color in any way. An IBM graphics printer, a Proprinter, or Epson graphics printers such as models FX 80 and 100 produce screen quality plots. Printer output is optional.

A version of Plod is available to support Ryan-McFarland Fortran, version 2.14, or Lahey Fortran (F77l), version 2.20.

Linking the libraries into an executable program requires either IBM Link (version 2.3 or higher) or Phoenix Software's Plink86.

Because of the high degree of text processing, we wrote Plod0 in Pascal. However, we wrote Plod1 (mostly) in Fortran, which allowed us to make the easiest use of the high-quality ODE software currently available. Also, most scientific users already have a Fortran compiler. The need for full ANSI Fortran 77 compatibility determined our specific choices.

Please note that since we compiled the libraries supplied with Plod by using one of our supported Fortran compilers, these libraries may not link correctly with other compilers.

Portability issues. The Fortran source for Plod1 compiles with any compiler supporting the full Fortran 77 standard. The sections dealing directly with graphics are in Intel 8088 Assembler and Lattice C languages. The conventions of calling assembly language from Fortran differ for each compiler. Thus, if you request a copy of Plod you *must* specify which Fortran you are using. To generate a working copy of Plod1 from source, you must have Fortran and C compilers, as well as Assembler. To compile Plod0 requires Turbo Pascal. (To *use* Plod, you need have only a Fortran compiler.)

An example of Plod0

Plod0 has a built-in editor for generating models, which are written onto a user-specified file with extension MOD. A companion file with extension FOR for the Plod1 step is also produced. Figure 1 displays an example MOD file. MOD files use a Fortran-like syntax except that the prime symbol can denote derivatives and a percent sign can denote comments that can begin anywhere in a line. Although Figure 1 does not show it, MOD files can be quite complicated and can include IF tests. Once you have created a MOD file, you can use it as direct input to a Plod run. Thus it is only necessary for you to create the model once.

```
T
% HBRO2: Scaled concentration of Bromous Acid
% BRION: Scaled concentration of Bromide ion
% CEIV:  Scaled concentration of Cesium IV
%  T  :  Scaled seconds
% Reasonable initial conditions are
%        HBRO2=4, BRION=1.1, CEIV=4
% Reasonable parameter values are
%        S=77.270
%        Q=0.8375E-05
%        W=0.161
% A full cycle is on interval [0, 350],
%    with lots of action by T=10.
% Plot all variables, log scaling vertically
HBRO2'=S*(BRION-HBRO2*BRION+HBRO2-Q*HBRO2**2)
BRION'=(CEIV-BRION-HBRO2*BRION)/S
CEIV'=W*(HBRO2-CEIV)
```

Figure 1. Example of a MOD file generated by Plod.

An example of Plod1

A user normally selects automatic compilation and linking to Plod1 from a menu. A typical Plod1 session involves data input, integration, plot or list of results, modification of some aspect of the problem, or reexamination of the output.

You can type in initial and parameter values or read them from a file. These values include simple constants or expressions involving variables or parameters. A parser catches syntax errors or square roots of negative numbers, for example. You can also set the integration-stopping conditions. This normally involves specifying the interval on which the integration is to be performed, but it can also be stated in terms of any of the dependent variables, such as BRION = 2*HBR02.

You need not specify the method used to integrate and the accuracy of integration, but can set them in an Expert menu, which is easily accessed from the main menu. During the integration, intermediate output is displayed one screen at a time, so it is possible to verify whether things are behaving properly. The integration can also be performed "quietly." Plod takes a maximum of 400 internal steps before asking "what-next?" (The number of steps can be changed to suit specific problem needs in the Expert menu.) Even in Quiet mode, you can interrupt the calculation by pressing the space bar, and if all is well, continue the interaction. The integration step size is variable, which makes it impossible to tell in advance how many steps will be needed.

In graphics mode, up to nine curves can be drawn on one screen involving any combination of the variables, parameters, and derivatives. In practice, the most-often used plots are the dependent variables plotted against t (the default) or phase-plane plots, y_1 versus y_2. Plots can be linear or logarithmic, and can be generated on an attached printer. It is possible to move around the screen and read out the numerical values, as well as to "zoom." The first-time user can get default graphs by responding with a carriage return to every question.

Plod's integration output points drive its graphics. You need not specify the plotting interval. Choosing the output points adaptively solves an aliasing problem that is common in generating plots of dynamic variables; plots are guaranteed to be smooth and not miss structural details. To prevent too much data from being plotted, we use a sieving algorithm that only plots points that affect the appearance of the graphs. Figure 2 on the next page shows two typical plots from the model discussed.

Graphics capabilities

We examined several Fortran-callable libraries for IBM micros, but the Plod user needs to compile and link programs. Hence, the graphics library must be available to the linker. Distributing a proprietary library would have violated each of the vendors' purchase agreements. Since we were committed to development of public domain software, we saw simple graphics, or plotting, as an important need. Consequently, we wrote a small set of screen plotting subroutines.

Extensibility

Plod is a menu-driven, interactive program built on Fortran subroutines for solving ODEs. As such, it cannot be completely flexible. At the beginning of this article, we mentioned several situations where Plod is inappropriate. Sometimes the best bet is to return to the original Fortran to program a specific application. In other cases, one would like to have just a bit more flexibility than Plod allows. You can modify the Plod source, but please do not hesitate to contact us first (address at the end of this article) to see whether a more general solution could be reached that would also be useful to others.

One of the important results of the rapid spread of microprocessors is that people with less computer-oriented training are using computers to help in their jobs. This has pressured software developers into finding clever and interesting methods of generating problem-solving tools that do not require programming. Plod is one step in this direction. It is being used at several hundred sites for solving continuous time-

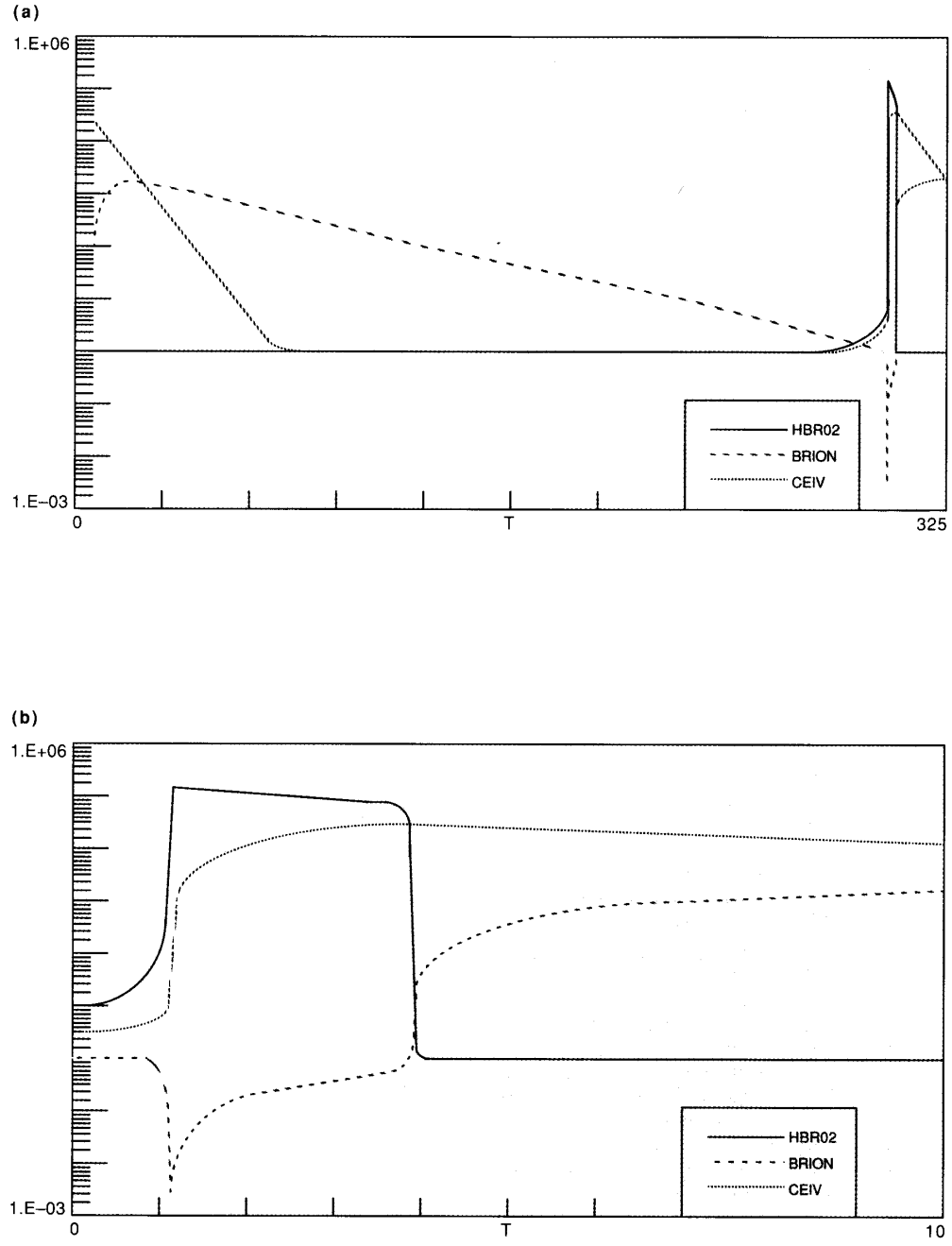

Figure 2. Field-Noyes chemical oscillator (a), and expanded view of its left edge (b).

simulation problems. As our ingenuity and experience improve, we expect to see even better products with this same nonprogramming flavor.

References

1. C.W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations,* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1971.
2. H. Fowler et al., "Wave Form Simulations for Josephson Junction Circuits Used for Noise Thermometry, " *NBS Tech. J.,* 1983.
3. "Catalog of Simulation Software," *Simulation,* Vol. 45, No. 4, 1985, pp. 196-209.
4. *Advanced Continuous Simulation Language (ACSL) Reference Manual,* Mitchell and Gauthier, Assoc., 290 Baker Ave., Concord, Mass., 1981, p. A-19.
5. V. Franceschini, "Bifurcations of Tori and Phase Locking in a Dissipative System of Differential Equations," *Physica,* Vol. 6D, 1983, pp. 285-304.
6. R. Boisvert, S. Howe, and D. Kahaner, "The Guide to Available Mathematical Software (GAMS)," PB 84-17135, Nat'l Tech. Info. Service, Springfield, Va., Feb. 1984.

Elvira Agron is a computer systems analyst at the National Institutes of Health, where she helps researchers utilize statistical program packages and interpret the results.

Agron received her BS degree in mathematics at the University of Puerto Rico and an MA degree in applied mathematics at the University of Maryland.

I-Lok Chang is an associate professor of mathematics at The American University, Washington, DC. His areas of interest include value distribution theory in complex variables, and optimization and approximation in numerical analysis.

Chang received his BS and PhD degrees in mathematics from the California Institute of Technology and Cornell University, respectively.

Gamini Gunaratna holds a master's degree in computer science from The American University. He has previously worked at the National Bureau of Standards and the University of Pittsburgh. His current interests include MS-DOS computers and Turbo Pascal.

David K. Kahaner is the technical group leader for scientific software and microcomputer applications in the Scientific Computing Division, Center for Applied Mathematics at the NBS. His research interests include evaluation of integrals, solution of ordinary differential equations, interpolation, Fourier transforms, and mathematical software in general.

Kahaner received his PhD in applied mathematics from the Stevens Institute of Technology, Hoboken, New Jersey. He then worked at Los Alamos National Laboratory as a numerical analyst. He has been a visiting professor in the Mathematics Department of the University of Michigan, at the ETH-Zurich, at the Technical University-Vienna, and at the Catholic University of America in Washington, DC.

Martin A. Reed is a development manager at IBM, where he has worked for the past 15 years. He received his MS degree in electrical engineering and operations research from Rensselaer Polytechnic Institute in Troy, New York. Reed holds a PhD degree in electrical engineering from the Catholic University of America, where his dissertation research involved interactive computer graphics and modeling. He is a member of the IEEE.

Questions concerning this article may be directed to David K. Kahaner, Center for Applied Mathematics, National Bureau of Standards, Technology Building, Room A161, Gaithersburg, MD 20899.