On the use of rootfinding ODE software for the solution of a common problem in nonlinear dynamical systems

D.K. KAHAHNER  
Center for Computational & Applied Mathematics, National Institute of Standards and Technology, Gaithersburg, MD 20899, U.S.A.

W.F. LAWKINS  
Mathematical Sciences Research Section, Oak Ridge National Laboratory, Oak Ridge, TN 37830, U.S.A.

S. THOMPSON  
Mathematics Department, Radford University, Radford, VA 24142, U.S.A.

Received 27 July 1988  
Revised 10 November 1988

Abstract: We discuss how rootfinding, which is built into some ODE software, can be used to generate Poincaré sections. An interactive program is also described.

Keywords: Ordinary differential equations, dynamical systems, automatic integration, Poincaré sections.

1. Introduction

Research concerning the basic mechanisms governing nonlinear phenomena has grown rapidly in recent years. One of the principal reasons for that growth has been the discovery that relatively simple systems of ordinary differential equations (ODEs) can display a remarkable variety of qualitative and quantitative behavior, usually related to the variation of some system parameter. Two examples are the Lorentz model [14], a system of ODEs which can be related to buoyancy driven flow in a fluid-filled tube and the Franceschini model [4], a system of ODEs approximating an ideal experiment of flow on the surface of a two-dimensional surface with a spatially periodic forcing term. It would be difficult to over-estimate the contribution to the field of nonlinear dynamics that has derived from the study of these as well as many other nonlinear models. Indeed, empirical investigations using such models have provided the inspiration for a number of important new concepts to explain the complex behavior of dynamical systems in, or evolving toward, a state of chaos.

Most nonlinear models must be studied numerically. Current software for solving ODEs automatically is sufficiently robust to permit reliable integration over long time intervals. A
useful feature of some automatic ODE software is the ability to simultaneously locate roots of auxiliary functions \( g(t, y) \) that depend on the solution. A task that arises frequently in nonlinear dynamical problems is the necessity to calculate and plot Poincaré sections for the solution. This task is ideally suited to software with the ability to do rootfinding. The typical calculation of Poincaré sections (which involves finding intersections of certain solution components with planes) is sometimes performed by integrating and saving the solution, manually inspecting the results to bracket points of intersection, and then using linear interpolation to approximate the points at which the trajectories pierce the given plane. One of our first experiences with problems in this area occurred because a physicist came to us with just such data asking for a more accurate method. These tasks can be performed automatically using ODE software that can do rootfinding.

This paper first reviews briefly the importance of rootfinding in continuous simulation and describes the manner in which problems with rootfinding requirements have affected the evolution of software for ODES. With respect to the problem of computing Poincaré sections, we next describe how a standard FORTRAN ODE solver can be used in this context. We then describe how the continuous simulation solver PLOD can be used to solve the problem. In addition to retaining the desirable features of the FORTRAN solution, PLOD requires no specialized knowledge of ODE software or graphics software and is convenient and easy to use for scientists and engineers. We demonstrate the versatility of this approach by using PLOD to calculate Poincaré sections for a nonlinear dynamics model of plane incompressible flow, and for a model of wind loading.

2. The importance of rootfinding in continuous simulation

The basic problem of interest in this paper is the following:

\[
\frac{dy}{dt} = f(t, y), \quad y(t_0) = y_0, \quad 0 = g(t, y).
\]

Here, \( y \) is the solution to the standard ordinary differential equation initial-value problem. In addition to solving the ODE, we are also interested in finding roots of the \( g \)-equation. \( y \) is generally a vector and there may be multiple \( g \)-functions. Roots of such functions typically correspond to times at which special events occur (e.g., derivative discontinuities or times at which special output processing is required). A simple example that clearly illustrates the ideas involved is the bouncing ball problem [16]:

\[
\frac{dh}{dt} = v, \quad h(0) = h_0, \quad \frac{dv}{dt} = -G, \quad v(0) = 0, \\
0 = g(t, h, v) = h,
\]

where \( G \) is the force of gravity. The ODEs and the initial conditions describe the height and velocity of a ball initially at rest and dropped from a height \( h_0 \). Each time the ball bounces, special action must be taken in order to reverse the motion of the ball. For example, a typical action is to replace the velocity \( v \) of the ball at the instant of impact by \(-Kv\) where \( K \) is a constant between 0 and 1. In order to know when to change the value of \( \frac{dh}{dt} \) we must determine the instant of bounce.
One way to try and do this is to add some coding to the routine that defines the right-hand side of ODEs, such as

\[
\text{IF}(h \leq 0.0) h' = -K \cdot v
\]

and then let the ODE solver fend for itself. However, this induces a discontinuity in \( dh/dt \), and most modern solvers will perform inefficiently near the bounce times and can completely fail to make any further progress. In many physically realistic problems characterized by the presence of multiple special events, it is also possible for a solver to miss one of the events and subsequently generate an incorrect solution.

A better way to deal with this problem is not to alter \( dh/dt \) until after the bounce time \( t_b \) has been found, and then restart the integration at \( t_b \) with a new value of the velocity. We can do this by locating the roots of the event function \( g = h \). The initial stepsize that is selected by an automatic adaptive ODE solver is usually much smaller than the physical scale of the problem, but subsequent stepsizes increase rapidly. Thus the overhead associated with restarting at each bounce point is less than forcing the solver to integrate through the discontinuities.

Several problems of this type which motivate our interest in rootfinding for ODEs are described in [17] (see also [16]). One such problem models the performance of a reactor during the refill and reflood stages following a postulated loss-of-coolant accident. During the refill phase, prior to the onset of system recovery, the underlying initial-value problem is relatively easy to solve. However, when the water level (which is determined by the solution of the initial-value problem) reaches the bottom of the exposed reactor core, the defining equations change drastically due to the introduction of additional equations and correlations. For a typical set of conditions, many components grow by several orders of magnitude almost instantaneously. It is virtually impossible to solve such problems both accurately and efficiently unless rootfinding is used to locate the time the coolant strikes the core and the integration is restarted at this point. Another problem models a reactor system pressurizer that maintains a roughly constant system pressure through the use of heaters to raise the pressure, and the use of sprays and relief valves to reduce the pressure. For this model, there are actually more special event functions than ODEs. Due to the manner in which the various special events in the model interact, it is necessary to locate accurately all event times and to execute the events in the correct order. A similar three-region steam generator model requires the solution of a set of steady-state flow equations. The steam generator contains three distinct regions consisting of a subcooled liquid region, a second saturated steam and liquid region, and a third region containing superheated steam. In this model rootfinding is required to locate the boundaries of the various regions. Rootfinding is also required to create and delete regions as they appear or disappear in the model. Another problem models the growth of vapor bubbles in a time-dependent pressure field. The underlying problem is a moving boundary problem in which the motion equation is coupled to a spatially discretized heat equation at the interface of the vapor bubble and the surrounding liquid. Rootfinding is required to determine the onset of nucleation (which is determined by the time at which the system pressure drops to a prescribed value). In this model as well as the above ones, rootfinding is also useful for determining times at which to switch between nonstiff and stiff solution methods depending on the phase of the problem. For descriptions of other problems that benefit from the use of rootfinding to handle system-derivative discontinuities, the reader is referred to [2,16,17].

Problems such as the ones described above led to considerable interest in the solution of
ODEs with rootfinding requirements. There are troublesome issues that must be considered when rootfinding techniques are implemented in an ODE solver. Several of these issues are discussed in [5,15,16]. As a result of the continued interest in rootfinding, several high quality ODE solvers contain provisions for rootfinding of one type or another. One of the earliest implementations was in the program DVDQ [11] where it was called "G-stop." Another early implementation was in the FORSIM [3] simulation language. Rootfinding has also been implemented in several modern high-quality FORTRAN callable integrators. For example, stiff solvers with the facility to do rootfinding include DSTPGT [17], LSODAR [6], RDEBD [18] and SDRIV [10]. Each of the solvers is a descendant of the celebrated Gear solver. Each uses the Nordsieck scaled Taylor series used to represent the underlying polynomial approximations to the solution for the purposes of interpolation. Nonstiff solvers with rootfinding capabilities include the Adams solver RDEABM [19] and the Runge–Kutta–Sarafyan solver RKST35 [15]. Both LSODAR [6] and SDRIV [10] also allow use of Adams methods. Interestingly, most users of FORTRAN callable ODE software are unaware that rootfinding is available. Even those that know the capability exists may not be aware of an important fact about the way these codes operate. To do rootfinding in addition to integrating the differential equations \( y' = f(t, y) \), they use only evaluations of the stopping function \( g \) and of the polynomial representing the solution \( y(t) \). They do not do additional evaluations of the function \( f \) (which might be much more costly) or other integration operations. This makes the additional cost minimal, and contrasts with what a naive user might do to solve such a problem using an ODE solver that does not have a built-in rootfinding ability.

The SDRIV/DDRIV solver is implemented in the PLOD [1] simulation package. It is the use of this solver which will be the primary focus of this paper.

3. Nonlinear dynamical systems and Poincaré sections

The problem of analyzing time-dependent solutions of any sort is nontrivial and the formal mathematical tools available to do so are very limited. There is, for example, the Floquet theory for analyzing the stability of a single-frequency limit cycle solution of a dynamical system [7]. However, nonlinear dynamical systems offer a rich variety of possible solutions—ranging from stable steady-state solutions, to stable multiple frequency quasi-periodic solutions on a multidimensional torus, to chaotic solutions living on so-called strange attractors. Thus it is unrealistic to presume that theoretical tools will be available anytime soon for analyzing any but the simplest systems. Consequently, there is significant interest in the development of numerical tools which lend themselves to visual and graphical approaches for empirically investigating complicated solutions. Foremost among these tools, particularly with regard to nonlinear dynamics, is the Poincaré section. See [9] for a discussion of the growing number of numerical tools being reported in the literature for analyzing Poincaré sections to gain quantitative information about the solutions to dynamical systems.

A broad class of nonlinear dynamical problems receiving a great deal of attention today can be generically represented by a system of ODEs

\[
\frac{dy}{dt} = f(y, R),
\]
where $R$ is a positive parameter and $f(0, R) = 0$ for all $R > 0$. For values of the parameter $R$ below some critical value, the zero solution is a stable, steady-state solution. At the first critical value of $R$, the zero solution becomes unstable and very often a Hopf bifurcation then takes place. That is, a single-frequency limit cycle emerges from the zero solution in state space. As $R$ is increased from this critical value to a second critical value, the limit cycle becomes unstable. A solution with two frequencies (whose ratio is a rational number—thereby making the solution itself periodic) may appear. As $R$ is increased further, more critical values are reached and very complicated solutions are likely to appear. It is of great interest to investigate the critical values of $R$ at which solutions become unstable, to understand the mechanics associated with the bifurcation processes taking place at those critical values, and to quantitatively analyze the sets (or attractors) on which the solutions reside in state space. For these reasons, Poincaré sections serve as a means to probe answers to these questions. A Poincaré section of such a system for a fixed value of $R$ results from first finding intersections of the solution $y(t)$ with some hyperplane determined by constraining one component of the solution, say $y_1$, and then projecting that intersection point onto a two-dimensional plane corresponding to two of the remaining components, say the $y_2,y_3$-plane.

Since Poincaré sections correspond to the intersection of various components of the solution with fixed planes, the real problem that is of interest is equivalent to locating the times at which a function of the solution attains a value of 0. In order to obtain an accurate graph of the resulting Poincaré section, it may be necessary to locate hundreds or even thousands of such intersections. The result can then be saved and later post-processed in order to graph the section. This task is amenable to solution by a good adaptive ODE solver. For example, if the problem in question is to calculate the Poincaré section corresponding to, say $y_1 = 0$, the solver can be used by having it integrate the solution and return control to the user each time it locates a root of the solution $g(t, y) = y_1 = 0$. Each time such a return is made, the solution can be processed and the necessary information saved for later plotting purposes. Control is then returned to the solver which continues to integrate the problem until the next root is located.

The use of an adaptive solver for the problem minimizes the possibility that the solver will step over roots due to the stepsize being too large. Most rootfinding ODE solvers integrate forward using a variable step determined by an error control mechanism while they also look for a sign change in (an) event function(s). Detection of a sign change signifies that a root exists between two successive integration points. The accurate location of the root is determined by a combination of bisection, interpolation and the secant method. Two programs that locate roots not associated with a sign change are described in [19]. Using any of these programs relieves the user of the burden to experiment with multiple stepsize solutions in order to calculate Poincaré sections accurately. Finding Poincaré sections is a particular case of rootfinding, as $g(t, y) = y_1$. Reliable ODE software that is specialized to provide Poincaré sections has not been developed. Instead one normally makes use of one of the excellent general purpose routines. For Poincaré sections, the times $t_i$ at which $y_1 = 0$ are recorded along with the associated values of the other solution components, but the form of the equations does not change there. Thus, unlike the bouncing ball problem, the integration need not be restarted but can continue without alteration.

For many problems in dynamics a Poincaré section plot exhibits symmetry with respect to one or more of the plotting variables. In that case it is unnecessary to plot the entire section. What is preferred is that portion of the section in a particular quadrant, say when both of the plotting
variables are positive. Sometimes the best way to generate this is to find all the intersections of
the solution with \( y_1 = 0 \) and then plot \( y_2 y_3 \) using a window specified by \( y_2 > 0 \) and \( y_3 > 0 \). If a
great many points are to be generated a better alternative is only to record those points that
satisfy both \( y_1 = 0 \) and \( y_2 > 0, y_3 > 0 \).

4. Examples of typical dynamical systems requiring analysis

In this section we illustrate three dynamical systems that require the type of analysis that we
have been referring to.

(1) In [12] mathematical analysis was performed on a model involving an infinite bar of
square cross section that is suspended by springs at top and bottom in an air flow. More
recently the model has been generalized to two spring coupled bars. Applications include
wind galloping of bridges, hydraulic stresses on anchored oil rigs, etc. A wind-tunnel was
built and the engineers then wanted to relate their experimental data to the solutions of the
model equations for a wide range of eight input parameters, including parameters scaled
for steel bars in a water flow. The equations for the displacement and velocity of each of
the bars are

\[
x_1' = x_2, \quad Z = \frac{x_1'}{u},
\]

\[
x_2' = -x_1 + a_1 x_3 + n_1 u^2 Z \left(2.69 - \frac{2 B_1}{n_1 u} + Z^2 \left(168 + Z^2 (6270.0 - Z^2 59900)\right)\right),
\]

\[
x_3' = x_4, \quad W = \frac{x_3'}{u},
\]

\[
x_4' = a_1 x_1 m - a_2 x_3 m
\]

\[
+ n_2 u^2 W \left(2.69 - \frac{2 B_1}{n_2 u} + W^2 \left(168 + W^2 (6270.0 - W^2 59900)\right)\right).
\]

The parameters \( m \) and \( n_i \) are the dimensionless mass and the mass coefficients, \( B_i \) and \( a_i \)
are the damping and stiffness of the springs, and \( u \) is the velocity of the fluid flowing past
the bars.

(2) In [13] the second order equation

\[
\beta \phi'' + (1 + \gamma \cos \phi) \phi' + \sin \phi = \alpha + k \omega \sin \omega t
\]

is considered as a model for simulating the behavior of the circuit consisting of a series
connection of an inductor, a resistor and a Josephson junction (used as a noise thermome-
ter). The variable \( \phi \) is the time-dependent quantum-mechanical phase difference across the
Josephson barrier. The quantities \( \alpha, \beta, \gamma, k, \) and \( \omega \) represent combinations of circuit
parameters, as well as the amplitude and frequency of the applied signal. To make use of
most of the ODE software available this equation must be converted to a pair of
first-order ODEs by the standard transformation \( \phi_p = \phi' \). This leads to the system

\[
\phi' = \phi_p,
\]

\[
\beta \phi_p' = -(1 + \gamma \cos \phi) \phi_p - \sin \phi + \alpha + k \omega \sin \omega t.
\]
(3) The Franceschini model considered in [4] is related to the Navier–Stokes equations for an incompressible fluid on the surface of a torus \( T^2 = [0, 2\pi] \times [0, 2\pi] \),

\[
\frac{\partial u}{\partial t} + (u \cdot \nabla) u = -\nabla p + f + \nu \Delta u, \quad \text{div } u = 0, \quad \int_{T^2} u \, dx = 0.
\]

Here \( u \) is the velocity field, \( p \) is the pressure, \( \nu \) is viscosity and \( f \) is a periodic volume force. If we assume that \( u \) can be expanded in a seven-node truncated Fourier expansion [4] the equations become

\[
\begin{align*}
y_1' &= -2y_1 + 4\sqrt{5} y_2 y_3 + 4\sqrt{5} y_4 y_5, \\
y_2' &= -9y_2 + 3\sqrt{5} y_1 y_3 + 3\sqrt{5} y_6 y_7, \\
y_3' &= -5y_3 + 9y_1 y_7 - 7\sqrt{5} y_1 y_2 + R, \\
y_4' &= -5y_4 - \sqrt{5} y_1 y_5, \\
y_5' &= -y_5 - 3\sqrt{5} y_1 y_4, \\
y_6' &= -8y_6 - 4\sqrt{5} y_2 y_7, \\
y_7' &= -5y_7 + \sqrt{5} y_2 y_6 - 9y_1 y_3.
\end{align*}
\]

The parameter \( R \) in this system, called a Reynold's number in [4], is related to the force \( f \). As \( R \) is varied, significant qualitative changes in the solution take place. These changes are represented by the corresponding Poincaré sections. In fact, [4] contains a good example of a post-solution analysis using Poincaré section data. Briefly, the Poincaré section data is used to generate a real-valued quasi-time series using one of the coordinates of that section. A discrete Fourier analysis of that series produces what [4] calls the spectrum of the Poincaré map. The quantitative as well as qualitative change to that spectrum as \( R \) varies (for example, from an interval where the solution is quasi-periodic to one where it is chaotic) provides a very useful tool for analyzing the nature of the bifurcation that takes place. The desire to perform this type of analysis necessitates the ability to calculate Poincaré sections and to save the relevant data for later analysis. It is this question on which the remainder of this paper will concentrate.

5. PLOD

A typical dynamical system analysis consists of several stages.
- Model formulation.
- Parameter formulation.
- Experimentation.
- Production.

Early on, the analyst is confronted with the task of deciding what equations to study. In most nonlinear systems the equations will contain several embedded parameters as we have illustrated in the preceding section. The behavior of the system can depend strongly on the values of these parameters, and the range of interesting values is rarely known in advance. More often it is
necessary to solve the system with trial parameter values and even try variations in the form of
the model. Once final decisions have been made a production run can be undertaken on a large
and fast computer. The production software that is needed includes a differential equation
solver, graphics routines, and a driver program that particularizes the output to the needs of each
specific problem. As mentioned in Section 2, good ODE software contains root finding
capability, and graphics packages, such as Disspla [8] can be used to produce publication quality
plots.

PLOD is a program for IBM PC type computers that is designed to assist analysts in the first
three stages. It has been designed to satisfy the following seven criteria.

(1) The physical problem can be described by \( \leq 25 \) ODES and \( \leq 25 \) parameters.
(2) The problem is solvable while the user is at the terminal. PLOD is entirely interactive.

Criterion (1) addresses some “application”, and prototype problems. PLOD is inappropriate
for studying, say, the complete suspension system of an automobile. Interestingly, most users
who have requested expansion of the package cite the restriction on the number of parameters.
Number (2) excludes any problems using large amounts of CPU-time. It also excludes problems
that need to be repeated automatically to generate an average as a parameter is changed. PLOD
is designed for “what if” studies, and as a prototype for more ambitious simulations. It is not
designed for the fourth, or production step listed above.

(3) PLOD is easy to use; no manual is required. Two files are included with installation and
tutorial information. They should be read, 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 is a major design goal. Thus
PLOD is also suited for classroom use.

(4) The numerical methods are of high quality. The integrator, DDRIV [10], is a modern
double precision implementation of the Gear algorithm and 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.

(5) Rapid, flexible (with respect to range and scaling) and attractive graphics suitable for
analytical viewing purposes, that is, for insight, are included.

(6) PLOD is mostly portable. Special characteristics of a particular computer are not used
unless unavoidable. PLOD is not entirely portable because of graphics and screen control,
but this is kept to a minimum and isolated. The current environment is an IBM ‘XT/AT’
although versions have been moved to a Sun workstation and a Univac/Tektronix
combination.

(7) PLOD is in the public domain.

PLOD is designed to be used in two interactive steps, PLOD0 and PLOD1. PLOD, a “batch”
program, runs PLOD0 and optionally will do the processing to pass to PLOD1. During PLOD0
the user enters the ODES. There is no integration 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 communi-
cate needed information to the PLOD1 step.

The PLOD0 step includes a built-in screen editor to allow the input of equations, what we call
the model. Users who prefer working with their own editor can do so if they wish. For example,
the Franceschini model would be written as
Franceschini Bifurcation Model

\[
\begin{align*}
T \\
\text{SQ5} &= \sqrt{5.0} \\
Y_1' &= -2.0 \cdot Y_1 + 4.0 \cdot \text{SQ5} \cdot (Y_2 \cdot Y_3 + Y_4 \cdot Y_5) \\
Y_2' &= -9.0 \cdot Y_2 + 3.0 \cdot \text{SQ5} \cdot (Y_1 \cdot Y_3 + Y_6 \cdot Y_7) \\
Y_3' &= -5.0 \cdot Y_3 + 9.0 \cdot Y_1 \cdot Y_7 - 7.0 \cdot \text{SQ5} \cdot Y_1 \cdot Y_2 + R \text{ % Note parameter R} \\
Y_4' &= -5.0 \cdot Y_4 - \text{SQ5} \cdot Y_1 \cdot Y_5 \\
Y_5' &= -Y_5 - 3.0 \cdot \text{SQ5} \cdot Y_1 \cdot Y_4 \\
Y_6' &= -8.0 \cdot Y_6 - 4.0 \cdot \text{SQ5} \cdot Y_2 \cdot Y_7 \\
Y_7' &= -5.0 \cdot Y_7 + \text{SQ5} \cdot Y_2 \cdot Y_6 - 9.0 \cdot Y_1 \cdot Y_3
\end{align*}
\]

Models can be quite complicated, and can involve IF-THENs, continuations, and comments. Variable names are arbitrary and primed variables can be used on the right side of equations as long as they have been defined previously. Once a model file has been created it can be saved for later use and modification. For the wind loading problem the model file is

Wind Galloping Model

\[
\begin{align*}
T \\
X_1' &= X_2 \\
Z &= X_1' / U \\
Z_2 &= Z \cdot Z \\
X_2' &= -X_1 + A_1 \cdot X_3 + N_1 \cdot U \cdot U \cdot Z \cdot (2.69 - 2 \cdot B_1 / (N_1 \cdot U)) + \\
& \quad Z_2 \cdot (-168. + Z_2 \cdot (6270.0 - Z_2 \cdot 59900))) \\
X_3' &= X_4 \\
W &= X_3' / U \\
W_2 &= W \cdot W \\
X_4' &= M \cdot (A_1 \cdot X_1 - A_2 \cdot X_3) + N_2 \cdot U \cdot U \cdot W \cdot (2.69 - 2 \cdot B_2 / (N_2 \cdot U)) + \\
& \quad W_2 \cdot (-168. + W_2 \cdot (6270.0 - W_2 \cdot 59900)))
\end{align*}
\]

The output FORTRAN program must be compiled and linked to precompiled modules supplied with the package. This is done automatically for PLOD, resulting in an executable program which we call the PLOD1 step. PLOD1 prompts for parameter and initial values, and the interval on which the integration is to take place. These can be directly typed into the program or read from a file. Allowed values are simple constants or expressions involving variables and parameters. A parser catches syntax errors or invalid arithmetic operations. Graphs and listings can be generated after the integration has ended. Changes can be made in the parameter values, initial conditions, integration interval, etc. It is possible to 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 it is not possible to alter the functional form of the model by adding terms, or adding or removing equations. That requires returning to the PLOD0 step, recompiling, etc.

Normally PLOD integrates along an interval \([t_0 \leq t \leq t_f]\) in the independent variable that is specified by the user. It is also possible to set auxiliary stopping conditions and these are of interest in this paper. PLOD accepts up to five stopping conditions of the form

\[
\begin{align*}
\text{Exp1} &= \text{Exp2} \& \text{ Lexp},
\end{align*}
\]

where \text{Exp1} and \text{Exp2} are arithmetic expressions involving any of the variables and parameters,
for example $Y_1' + \sin(Y_2) = Y_4$, or for the Franceschini model $Y_1 = 0$. $\text{Lexp}$ stands for a logical expression (an expression that evaluates to True or False), for example

$$Y_3 < 0 \text{ } \& \text{ } Y_6 > 0.$$  

The integration will halt when $\text{Exp1} = \text{Exp2}$ if $\text{Lexp}$ is also True. For the Franceschini problem an appropriate stopping condition $Y_1 = 0 \text{ } \& \text{ } Y_3 < 0 \text{ } \& \text{ } Y_6 > 0$. When the integration halts it is possible to stop processing, or to continue integrating until either the endpoint $t_f$ is reached or the stopping condition is satisfied once again. It is also possible to continue integrating to $t_f$ without stopping, while storing the solution at those values of $t$ that satisfy the stopping condition. For Poincaré maps the latter is the relevant option. Eventually we terminate the integration. A plot can then be generated involving any of the variables or combinations of them. For example to plot $Y_6$ versus $Y_3$ one only needs to name these variables. The plot can either be at all time points or only those that satisfied the stopping conditions. Plot points can be either connected or not; usually disconnected points are appropriate for Poincaré maps. In many dynamical system studies a final integration time is not known in advance. PLOD allows for this by permitting a user to continue an integration to a larger value of $t_f$, and will merge the new output with the earlier results, allowing the ensemble to be plotted or listed in a natural way.

Neither the integration method nor integration accuracy need to be specified although they can be set in an Expert menu which is accessed from a main menu. Integration can be performed without any output in the screen (quietly) or with full output during the calculation. Files of complete time history of the integration or of the Poincaré section can be written to a disk. This allows supplementary post-processing if PLOD does not provide some needed function. For example, it might be of interest to perform a Fourier analysis of the Poincaré section.

Figures 1 and 2 were generated by PLOD after integrating the Franceschini model for two particular values of the parameter $R = 326.25$ and $R = 299.50$. They can be compared to [4, Figs. 7 and 4c] which used a specially written program for a mainframe computer. Generating these figures with PLOD involved only entering the model (displayed above), setting the initial
conditions, value of $R$ and the stopping criteria, all via appropriate menus. The integration took about 10 minutes on an IBM PS/2 model 80 for each run. Changing $R$ was done through a "modify" menu. The plot was viewed on the computer screen and then printed (by request) on an attached dot-matrix printer. Users may request copies of PLOD by writing to the first author.

6. Summary and conclusions

As workstations and personal computers become more powerful we expect to see more sophisticated interactive applications developed. Nonlinear dynamics requires reliable, efficient differential equation solvers as well as graphics capabilities. Experimental studies require interactivity, production runs require fast computers. PLOD is designed for problems in the first category. It is well suited to initial experimental analysis because most standard needs are already built in. Others can usually be added as necessary. The program has been evolving over several years in response to user requests, of which there are currently about 250. Most have made valuable suggestions and helped to identify problems. For example, adding Poincaré sections was a new addition this spring. Happily, the design allowed for the evolution fairly easily.

References


