A SMART INTERFACE FOR NUMERICAL SOFTWARE

Elisha Sacks

CS-TR-341-91

Ausust 1991

A Smart Interface for Numerical Software

Elisha Sacks*
Department of Computer Science
Princeton University
Princeton, NJ 08544, USA

August 21, 1991

Abstract

The paper describes a smart interface that makes numerical libraries easy to use and reliable by exploiting mathematical theory, symbolic algebra, and descriptions of the input/output formats of numerical subroutines. The interface accepts a high-level problem description, selects the appropriate numerical subroutine, programs the problem in subroutine format, runs the program, corrects for numerical errors and special conditions, and returns the output in a high-level format. The current interface manages a root finder, a continuation package, an ordinary differential equation integrator, and a Lyapunov exponent calculator. I describe these functions and illustrate their use in a program that analyzes ordinary differential equations.

*This research is supported by the National Science Foundation under grant No. IRI–9008527 and by an IBM grant.

1 Introduction

Current numerical software places many burdens on the user, making it hard to use and error prone. For example, suppose someone wishes to find the roots of a polynomial using a typical numerical library. He must find the appropriate subroutine in the index, learn its input/output format, and write a program that passes the polynomial to the subroutine in its format and returns the roots in his format. He must then compile, link, and run the program. Finally, he must assess the output. If the roots seem unreasonable, he must look for programming and interface errors. If the program appears correct, he must look for numerical errors or for pecularities in the polynomial that violate the assumptions of the subroutine. For example, if two roots are very close together, he must decide if the second root is legitimate, is an artifact of the convergence criterion, or indicates a double root.

In this paper, I describe a smart interface that makes numerical libraries easy to use and reliable by automating the relevant mathematical theory, symbolic algebra, and programming formats. The interface accepts a high-level problem description, selects the appropriate numerical subroutine, programs the problem in subroutine format, runs the program, corrects for numerical errors and special conditions, and returns the output in a high-level format. For example, it accepts a system of equations in symbolic format, programs the equations and the Jacobian matrix, runs a Newton-Rhapson subroutine, eliminates spurious roots, and returns the remaining roots. The interface benefits human users and other programs. Humans benefit primarily from the ease of use. Programs benefit primarily from the reliability of the output, since they often misbehave badly when given incorrect data.

The current interface manages a root finder, a continuation package, an ordinary differential equation integrator, and a Lyapunov exponent calculator. I describe these functions in the following four sections. I then illustrate their use in a program that analyzes ordinary differential equations. I conclude with plans for future work.

2 Root finding

The interface finds the roots of a system of equations

$$f_i(x_1, \ldots, x_n) = 0; \quad i = 1, \ldots, n$$

($f(x) = 0$ in vector notation) that lie in a bounding box

$$\{(x_1, \ldots, x_n) | l_i \leq x_i \leq u_i\}.$$

The standard solution is to pick a point r_0 and a tolerance ϵ then iterate the Newton-Rhapson formula

$$r_{i+1} = r_i - D_x f(r_i)^{-1} f(r_i)$$

until $|f(r_i)| < \epsilon$. Here $D_x f$ denotes the Jacobian matrix $[\partial f_i / \partial x_j]$. If r_0 is sufficiently near a root, the iteration quickly (normally quadratically) converges to it. Every numerical library contains a Newton-Raphson subroutine. The user must derive $D_x f$, program f and $D_x f$, and link them with the subroutine. He must then repeatedly pick r_0, run the program, and assess the output.

The interface automates the entire process. The inputs are the equations in symbolic notation, the bounding box, and the tolerance ϵ. The outputs are the roots in the box and the eigenvalues of the Jacobian at each root. The interface programs the equations and the Jacobian and links them to the Newton-Rhapson subroutine from Press [8], calculating the derivatives with the BOUNDER package [9]. The resulting program takes an initial point and a tolerance as input and returns a root. It aborts if the number of iterations exceeds a bound (10 by default) or if the iterates leave the bounding box. The interface runs the program many times (100 by default) with tolerance ϵ and with random initial points in the bounding box. It double-checks roots that are less than 10ϵ apart by continuing the iteration to tolerance 0.01ϵ. If the distance between the roots drops beneath ϵ, it collapses them into a single root. It repeatedly generates and tests sets of initial points until many sets (20 by default) yield no new roots. It calculates the eigenvalues of the roots with the EIGRF routine from IMSL [4].

Root finding is central to the steady state analysis of ordinary differential equations. The constant solutions of an autonomous system $\dot{x} = f(x)$ are the roots of $f(x) = 0$. We obtain them directly from the interface. The periodic solutions of a periodic equation $\dot{x} = f(x,t)$ with $f(x,t) = f(x, t+T)$ are the roots of $F(x) - x = 0$ where $F(x)$, called the *return map*, equals the value at time T of the solution with initial value x (Fig. 1). The subharmonic periodic trajectories of order k, the initial values to which F returns after k periods, are the roots of $F^k(x) - x = 0$. The notation $F^k(x)$ stands for F iterated k times: $\underbrace{F \circ F \circ \cdots \circ F}_{k}(x)$. We obtain the periodic solutions from the interface by passing it a numerical routine (described below) that calculates F and DF. The interface is especially useful because numerical errors in F often cause the Newton-Raphson subroutine to yield spurious roots. The interface determines the stability of constant and periodic solutions from the eigenvalues of the roots of their defining equations.

3 Continuation

The interface traces the roots of a system of equations $f(x,p) = 0$ as a function of the scalar parameter p. The roots form smooth curves in the (x,p) space, called *branches* (Fig. 2). A single branch goes through points where $D_x f$ is nonsingular, but several branches can meet at a singular point. For example, the equation $x^2 - p = 0$ has two branches, $x = \pm\sqrt{p}$, that meet at $x = p = 0$ where $D_x = 2x = 0$. The standard solution is to pick a parameter value,

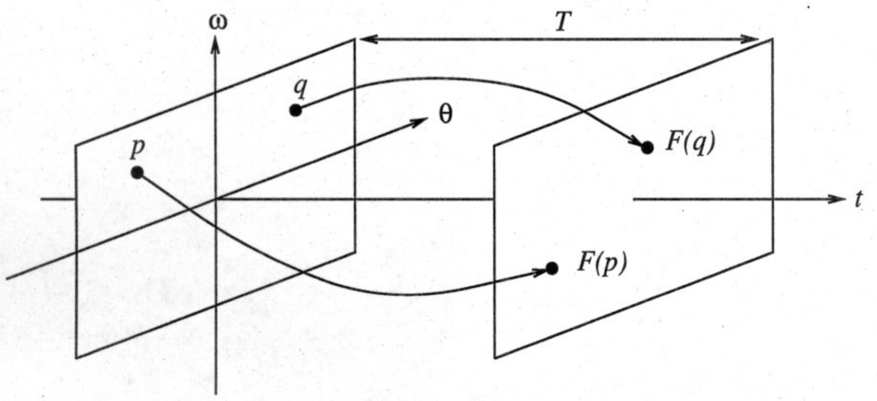

Figure 1: Solutions of a T-periodic planar ODE and their return map F.

find the roots at that value by Newton-Raphson iteration, and trace the roots by *continuation* [7]. The basic continuation step extends a root x_0 at p_0 to a root at p_1 by extrapolating to x_1 along the tangent to $f(x_0, p_0)$ then correcting by Newton-Raphson iteration (Fig. 3). A sequence of basic steps traces the branch through x_0 and any adjacent branches.

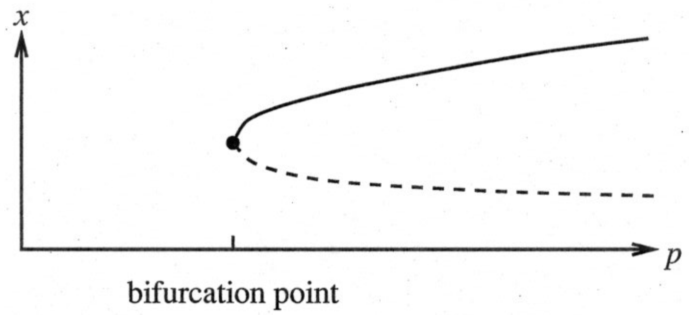

Figure 2: Two branches that meet at a saddle node bifurcation.

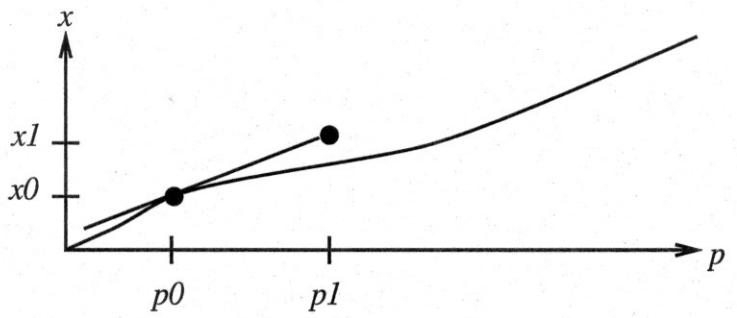

Figure 3: The basic continuation step.

The primary application of branch tracing is to determine the effect of parameter variation on the steady state solutions of differential equations. The branches yield the steady states for a range of parameter values. Values at which branches appear, vanish, or change stability type indicate qualitative changes in the steady state behavior, called *bifurcations*. Two types of bifurcations cover most constant solutions. A *saddle node* bifurcation occurs when two solutions merge and vanish (Fig. 2). A *Hopf* bifurcation occurs when a constant solution gives birth to a periodic solution. Analogous bifurcations occur in periodic solutions. A continuation algorithm can detect the bifurcations by monitoring certain functions of $D_x f$ at each basic step. For example, it checks if $D_x f$ changes sign between basic steps, since $D_x f = 0$ at saddle nodes.

Although less common than root finders, several continuation packages are available. The user must program $D_x f$ and $D_p f$ and link them with the package. He must then repeatedly find initial roots, run the program, and test the output for spurious, duplicate, and missing branches. He must also set tens of program parameters that control tolerances, search bounds, iteration counts, and computation methods.

The interface automates branch tracing and bifurcation detection. The inputs are the equations, a bounding box for the roots, a bounding interval for the parameter, and an error tolerance. The outputs are the branches, their stability types, and the bifurcations. The interface programs the equations and the derivatives, links them to the AUTO continuation package [1], and sets the program parameters. The resulting program takes a parameter value, a root, a parameter interval, and a tolerance as input and returns a list of branches and bifurcations. The interface samples the parameter interval at regular intervals (10 by default) and calculates the roots at each parameter value with the root finder. It checks whether each root lies on a previously detected branch. If not, it constructs a new branch by continuing the root over the parameter interval. It prunes spurious branches whose points are not roots and duplicate bifurcations that lie within the tolerance of each other. It checks if every change in the number or stability of the roots matchs a bifurcation. If not, it finds the missing bifurcations and deletes spurious bifurcations.

4 Integration

The interface integrates a parameterized system of ordinary differential equations $\dot{x} = f(x,t,p)$ from an initial state $x(0) = x_0$ to a final state $x(T)$. The inputs are the equations, the parameters p, T, x_0, a bounding box for x, and a tolerance. The outputs are $x(T)$ and optionally the derivatives of $x(T)$ with respect to x_0 and p. The user can select among Runge-Kutta, implicit Adams, and BDF (for stiff equations) integration methods. The interface programs f, $D_x f$, and $D_p f$, links them to the appropriate subroutine (Press [8] for Runge-Kutta, ODESSA [6] otherwise), sets the program parameters, and runs the program. It terminates the run if the solution leaves the bounding box.

The primary benefit of the interface is convenience, since numerical integration is very reliable and efficient. It saves the user the effort of programming f, $D_x f$, and $D_p f$ and of setting tens of program parameters.

5 Lyapunov exponents

The final interface task is computing Lyapunov exponents of maps, such as the return map of a system of periodic ordinary differential equations. The Lyapunov exponents of a map on $\Re^n$ with initial state x are defined as

$$m_i = \lim_{k\to\infty} |m_i(k)|^{1/k}; \quad i = 1, \ldots, n$$

where the $m_i(k)$ are the eigenvalues of $D_x F^k(x)$. The Lyapunov exponents characterize the steady state behavior of the map under iteration. In particular, the exponents of a return map characterize the steady state behavior of the corresponding solutions. The current implementation calculates the largest Lyapunov exponent, which suffices for the analysis of planar equations [7]. A negative exponent indicates a periodic solution, a zero exponent indicates a *quasi-periodic* solution that contains two incommensurate periodic components, and a positive exponent indicates a *chaotic* solution that wanders erratically through state space.

The interface calculates the largest Lyapunov exponent with the formula

$$\lim_{k\to\infty} \log(\|D_x F^k(x)\|)/k \tag{1}$$

where $\|\ \|$ denotes the standard matrix norm [3]. It performs the calculation directly because the programming cost of using a Lyapunov subroutine outweighs the benefit. The interface iterates F to eliminate transients (1000 iterates by default) then calculates the righthand side of equation (1) for increasing values of k until consecutive iterates differ by less than a tolerance. This stopping condition cannot distinguish zero exponents from exponents with small absolute values. The interfaces detects small negative values by iterating a few more times. If one of the iterates falls near the first iterate, the steady state is periodic. The interface distinguishes zero values from positive values by searching for a quasi-periodic solution.

6 The POINCARE program

I have used the interface in automating the kinematic analysis of mechanisms [5] and the steady state analysis of ordinary differential equations [10, 11]. Here, I sketch the latter project. I have developed a program, called POINCARE, that automates the analysis of one-parameter families of two first-order equations and assists in the analysis of families of three or more equations. The inputs are the family, bounding intervals for the state variables and for the parameter, and an error tolerance. POINCARE traces the branches of constant and periodic solutions in the parameter interval and finds the bifurcations. When the family contains two equations, it can also partition the state space into regions of uniform asymptotic behavior. Each region consists of all the initial states that converge to a common steady state.

POINCARE obtains the branches of constant and periodic solutions by continuing the defining equations, $f(x,p) = 0$ and $F^k(x,p) - x = 0$, over the parameter interval. The interface eliminates 90% of the branches and fills in several bifurcations on a typical periodic planar equation. POINCARE would generate incorrect analyses without this pruning, since it accepts steady states at face value. Alternatively, it would have to incorporate the functionality of the interface, as would every program that uses continuation. POINCARE uses the root finder, integrator, and Lyapunov exponents to partition the state space.

For example, Fallside and Patel [2] model a motor speed control (Fig. 4) with the equations

$$\begin{aligned} \dot{x} &= y \\ \dot{y} &= -x - 6x^2 - 6x^3 - 6x^2y - uy \end{aligned}$$

with u a control parameter. POINCARE fully analyzes the equations, filling in details missing from the original analysis. It finds three branches, one of which contains a Hopf bifurcation at $u = 0$. It partitions the state space at parameter values on either side of the bifurcation, as shown in Fig. 5. The lefthand diagram shows a stable constant solution labeled *ra,* two unstable constant solution labeled *s* and *sr,* and a periodic solution, which appears as a closed curve surrounding *sr*. At the bifurcation, the closed curve collapses onto *sr* and transforms it into the stable solution *sa*.

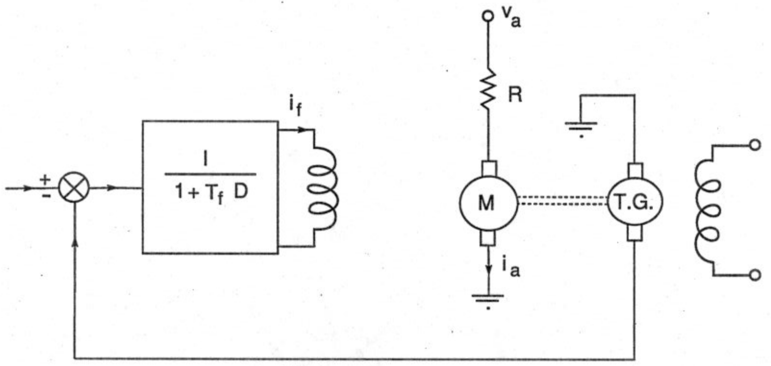

Figure 4: Motor speed control.

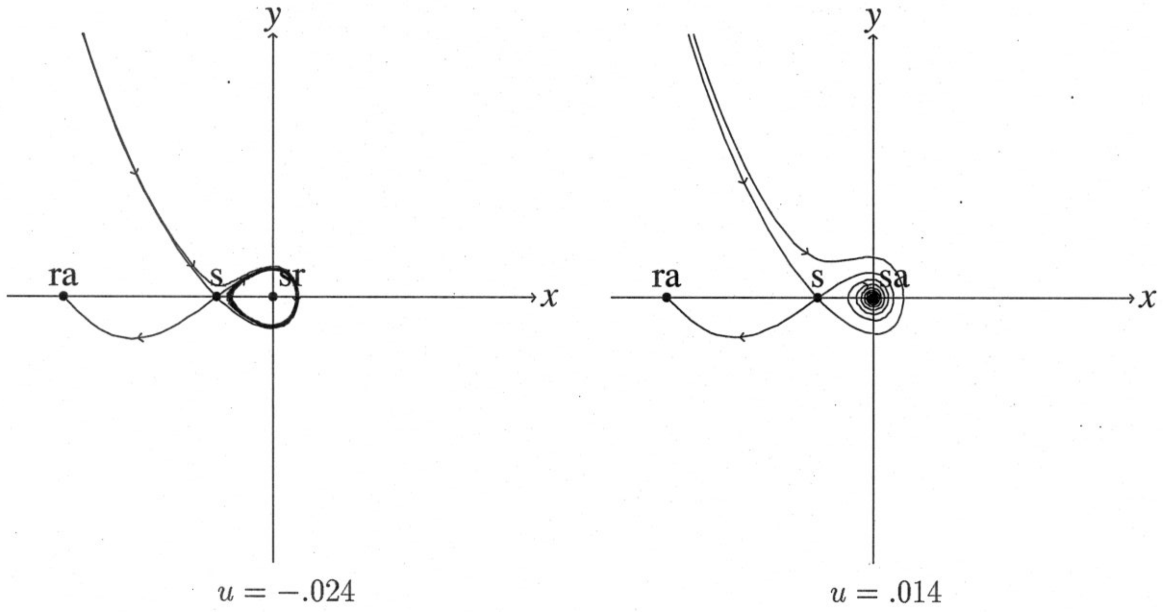

Figure 5: Steady state behavior before and after the Hopf bifurcation at $u = 0$.

7 Conclusions

We have seen that a smart interface can make numerical subroutines easy to use and reliable by automating the relevant mathematical theory, symbolic algebra, and programming formats. The current interface manages four subroutines and exploits a limited amount of knowledge. I plan to extend the interface to domains such as optimization, differential geometry, and partial differential equations. I also plan to extend its domain knowledge. For example, it collapses adjacent roots of algebraic equations based on a proximity heuristic, without trying to prove its conclusions. A smarter program could try to prove that Newton-Raphson iteration is a contraction on a neighborhood of the roots, hence that the neighborhood contains a single root. Extensive knowledge is crucial in the new domains that I hope to cover, for example in picking global optimization strategies and in coping with singular curves and surfaces. Given sufficient breadth and depth, I believe that smart numerical software will prove invaluable in automating scientific and engineering computing.

References

[1] Doedel, E. AUTO 86 user manual: software for continuation and bifurcation problems in ordinary differential equations. Technical report, Princeton University, Feb. 1986.

[2] Fallside, F. and Patel, M. R. Step response behavior of a speed control system. *Proceedings of the IEE* **112** (1965).

[3] Hogg, T. and Huberman, B. A. Generic behavior of coupled oscillators. *Physical Review A* **29** (1984) 275–281.

[4] *IMSL Library Reference Manual.* (IMSL Inc., Houston, 8 edition, 1980).

[5] Joskowicz, L. and Sacks, E. P. Computational kinematics. *Artificial Intelligence* (1991). in press.

[6] Leis, J. R. and Kramer, M. A. The simultaneous solution and sensitivity analysis of systems described by ordinary differential equations. *ACM Transactions on Mathematical Software* **14** (1988) 45–60.

[7] Parker, T. S. and Chua, L. O. *Practical Numerical Algorithms for Chaotic Systems.* (Springer-Verlag, New York, 1989).

[8] Press, W. H., Flannery, B. P., Teukolsky, S. A., et al. *Numerical Recipes in C.* (Cambridge University Press, Cambridge, England, 1990).

[9] Sacks, E. P. Hierarchical reasoning about inequalities. in: *Proceedings of the National Conference on Artificial Intelligence*, 1987. Reprinted in [12].

[10] Sacks, E. P. A progress report on automating the analysis of ordinary differential equations. in: *Proceedings of the 7th Israeli Symposium on Artificial Intelligence and Computer Vision*, 1990.

[11] Sacks, E. P. Automatic analysis of one-parameter planar ordinary differential equations by intelligent numerical simulation. *Artificial Intelligence* **48** (1991) 27–56.

[12] Weld, D. S. and de Kleer, J. (Eds.). *Readings in Qualitative Reasoning about Physical Systems.* (Morgan Kaufman, San Mateo, Ca., 1990).