From Formulas to Fortran to Results: The AUTOMATED PROGRAMMER System

CHARLES B. ENGLE, JR.

Computers have expanded the scope of problems that scientists and engineers are willing to tackle. A complex system of equations that may defy solution in closed form can often be put on a computer and solved numerically. Similarly, a more tractable problem that might have been solved for only a few cases can be examined in detail.

However, one may not feel that computers save a lot of work when the task of converting equations into computer code is actually confronted. For example, suppose you have developed a complex series of equations to model some system. You would like to be able to provide values for the variables in your model and to execute the model with enough data to get a reasonable feel for the results.

Now consider what you must do. You must either turn your efforts over to a computer programmer to translate them into the commands that the computer can use, or you can do the computer programming yourself. In the former case, you may work with the programmer, but unless you are conversant with programming terms and the language being used, you must effectively trust that the translation being accomplished is correct, that is, that the programmer cannot only program, but also understand enough mathematics to properly translate your formulas.

In the latter case, you spend time and effort programming (and, inevitably, debugging) that could be spent on doing work in your own field. You may also have to learn more about a language or applications program than you really care to know. In either case, "small" errors may creep in that may not be noticed until after many sets of results have been generated. Clearly, neither of these choices is optimal.

What is needed is some automated mechanism to convert the "language" of the profession (in this case, mathematics) into something that can be understood by the computer. This idea is not new; in fact, it has a name—executable specifications. Unfortunately, there have been few successes in this endeavor to date.

This is precisely the reason that applications programs were created. They allow the user who lacks a detailed knowledge of computer programming to use the computer. The tool (the computer) can be used without having to be an expert in creating tools (programming). Applications programs that most people are familiar with include spreadsheets, word processors, database managers, and such; examples in mathematics include equation-solving and symbolic-manipulation programs. A successful applications program should interact with a user knowledgeable in the problem domain without requiring that the user also be a computer scientist.

It can be argued that many applications programs do not meet this goal. Many users have dealt with an applications program that failed and then generated a cryptic message such as "error 32," or some other similarly meaningless remark, or have used a tool that required more customizing than anticipated and that required taking a crash course in computer programming just to use the tool. However, it is in precisely this area that the AUTOMATED PROGRAMMER (*1*) makes some significant progress. As the documentation for the AUTOMATED PROGRAMMER system states,

> The AUTOMATED PROGRAMMER System attempts to utilize, wherever possible, language forms for problem specification which are traditional, and are directly executable, thus minimizing the need for specialized programming.

System Description

The AUTOMATED PROGRAMMER is a complete system for entering, interpreting, compiling (*2*), and running applications of mathematics typically encountered in science and engineering. The AUTOMATED PROGRAMMER approaches the ideal of an executable specification in the sense that it allows the user to write mathematical equations in their familiar form and then automatically translates them into a machine-executable form. In order to do this, the AUTOMATED PROGRAMMER automatically interprets the two-dimensional lines on the computer screen; for example, integration symbols and limits are interpreted directly (*3*). An example of input is shown in Fig. 1. It then converts such input into a programming language for which a compiler exists (*4*). This intermediate form (programming language) is then compiled. At the user's option, the generated object module can be stored in a library (from which it can be accessible to a link editor) or it can be immediately linked and executed. In the latter case, all of the intermediate steps are hidden from the user and so it appears that the two-dimensional mathematical text produces output.

The intermediate form mentioned above was not designed to be accessible to the user as it represents the initial translation of the user input into a programming language. However, a user may request that the intermediate step be saved. I did this for the formula in Fig. 1. and examined the Fortran code produced from the screen image (Fig. 2). I was surprised at how "tightly" the automatically generated code was written; for example, no use is made of the infamous GOTO statement so characteristic of a quickly written Fortran program (*5*). For routines such as integration and differentiation, there are default options; however, this process can be overridden by the user and other routines supplied by the user can be called.

Why is it important to have a tool like this? The fairly complex mathematical expression in Fig. 1 can be compared with its Fortran implementation in Fig. 2. Although these two expressions of the mathematical formula are in a sense equivalent, the first one is easily recognizable and understandable by anyone with minimal mathematical sophistication, whereas the second one is more difficult to decipher unless the user is well trained in Fortran.

This small example illustrates another reason for using the AUTOMATED PROGRAMMER; it is self-documenting. The technical English and mathematical symbols used in the AUTOMATED PROGRAMMER are directly understandable without external documentation. In contrast, the Fortran implementation would require a large set of documentation files to explicate its purpose and use. Additionally, for those few cases when explanatory information is needed, the AUTOMATED PROGRAMMER allows comments to be inserted.

This useful tool has some rather distinctive and useful features, a few of which I list here:

- Implied multiplication.
- Automatic dimensioning for arrays.
- Two-dimensional "image" output formatting.
- Integrated two-dimensional text editor.

Florida Institute of Technology, Department of Computer Science, Melbourne, FL 32901.

- Implicit variable declaration.
- Subscripts and superscripts indicated normally.
- Unparenthesized function arguments.
- Standard mathematical symbols (sum, product, integral, and so forth).
- Double-precision floating point arithmetic.
- Menu-driven commands (may be overridden by keyboard commands).
- Keyboard template for special-symbol generation.

Comments and Evaluations

In an earlier review of this product, LoSacco (*6*) mentioned the need for a security block to be placed on the serial port. The purpose of the hardware copy-protection key (security block) was to prevent unauthorized copying of the AUTOMATED PROGRAMMER. Since that time the developers of this product have removed the requirement for the protection key and no longer ship it with the product.

However, two other capabilities that LoSacco laments as not being present in this otherwise excellent system are still in the development stage and have not been implemented. These are an internal facility for graphing output automatically and a mechanism for inputting complex variables. The first of these perceived shortcomings is not an actual problem, however, because the AUTOMATED PROGRAMMER easily can call external graphing routines that the user provides. The second problem has been recognized by the developers and is being addressed in a future release.

The language used in the AUTOMATED PROGRAMMER is very powerful and flexible, yet remains simple. There is a very short learning curve for the AUTOMATED PROGRAMMER, since the language that you use is basically guided by the "WYSIWYG" principle, that is, "what you see is what you get." If the mathematical formula is presented correctly on the terminal screen, then it is right in the language of the AUTOMATED PROGRAMMER.

One of the features of the AUTOMATED PROGRAMMER System that I found most useful was the ability to describe the output format with an image mechanism. This capability is best illustrated by an example; consider Fig. 3A, which is a program to calculate some current flow in an electrical circuit, given the voltage and resistance of each resistor. Note that the print command

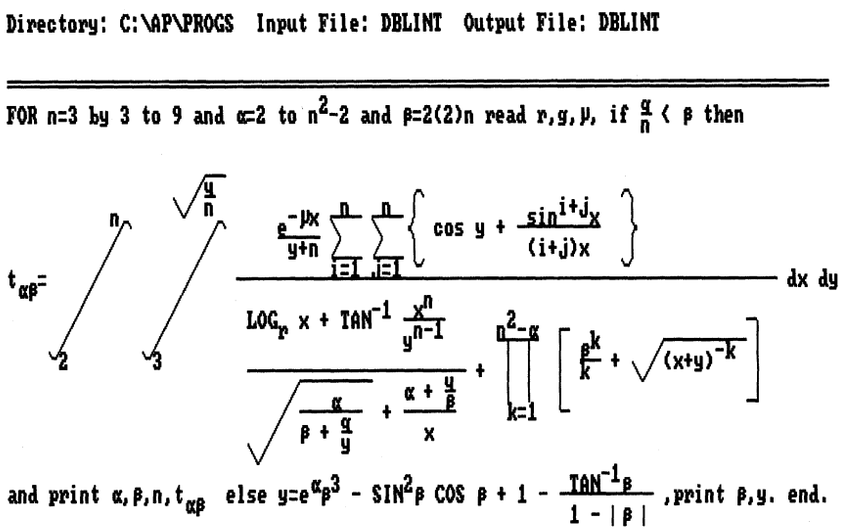

Fig. 1. Double integral to be evaluated by the AUTOMATED PROGRAMMER (supplied as demonstration program DBLINT).

```
      IMPLICIT REAL*8(T)
      COMMON /APABWDKI/ IMAGEL,V0051,V0058,V0059,V0064,V0065,V0066,V0068
     x,V0069,V0070,V0071,V0072,V0073,V0074,V0075,V0076,V0067
      EXTERNAL AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,F0078,F0080
      REAL*8 AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,V0051,V0058,V0059,V
     x0064,V0065,V0066,V0067(0:79,0:9),V0068,V0069,V0070,V0071,V0072,V00
     x73,V0074,V0075,V0076,F0078,F0080
      CALL AP$INIT(-1)
      IF (IMAGEL .EQ. 0) THEN
      IMAGEL = 1
      CALL AP$RINIT(V0067,800)
      ENDIF
      DO 12009 V0071 = 0.3000D1,0.9000D1,0.3000D1
      TL2015 = (AP$POW(V0071,0.2000D1)-0.2000D1)
      TI2015 = 0.2000D1
      DO 12015 V0072 = TI2015,TL2015,SIGN(1.0D0,TL2015-TI2015)
      DO 12018 V0073 = 0.2000D1,V0071,0.2000D1
      WRITE (0,'(1X,/,1A,\)')'Enter 3 Values -->'
      READ (INT(5),*)V0064,V0065,V0066
      IF (((V0065/V0071) .LT. V0073)) THEN
      V0067(NINT(V0072),NINT(V0073)) = AP$INTGR(V0059,0.2000D1,V0071,F00
     x80)
      WRITE (6,12087)V0072,V0073,V0071,V0067(NINT(V0072),NINT(V0073))
12087 FORMAT (1X,G15.8,1X,G15.8,1X,G15.8,1X,G15.8,1X)
      ELSE
      V0070 = ((((AP$POW(V0051,V0072)*AP$POW(V0073,0.3000D1))-(AP$POW(SI
     xN(V0073),0.2000D1)*COS(V0073)))+0.1000D1)-(ARCTAN(V0073)/(0.1000D1
     x-ABS(V0073))))
      WRITE (6,12108)V0073,V0070
12108 FORMAT (1X,G15.8,1X,G15.8,1X)
      ENDIF
12018 CONTINUE
12015 CONTINUE
12009 CONTINUE
      CALL AP$ERR(0)
99    STOP
      END
      REAL*8 FUNCTION F0078(V0077)
      IMPLICIT REAL*8(T)
      REAL*8 V0077
      REAL*8 V0078
      COMMON /APABWDKI/ IMAGEL,V0051,V0058,V0059,V0064,V0065,V0066,V0068
     x,V0069,V0070,V0071,V0072,V0073,V0074,V0075,V0076,V0067
      EXTERNAL AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,F0080
      REAL*8 AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,V0051,V0058,V0059,V
     x0064,V0065,V0066,V0067(0:79,0:9),V0068,V0069,V0070,V0071,V0072,V00
     x73,V0074,V0075,V0076,F0080
      V0078 = 0.0
      V0068 = V0077
      T2044 = 0.0
      DO 12044 V0074 = ANINT(0.1000D1),V0071
      T2043 = 0.0
      DO 12043 V0075 = ANINT(0.1000D1),V0071
      T2043 = T2043+(COS(V0069)+(AP$POW(SIN(V0068),(V0074+V0075))/((V007
     x4+V0075)*V0068)))
12043 CONTINUE
      T2044 = T2044+T2043
12044 CONTINUE
      T2074 = 1.0
      DO 12074 V0076 = ANINT(0.1000D1),(AP$POW(V0071,0.2000D1)-V0072)
      T2074 = T2074*((AP$POW(V0073,V0076)/V0076)+SQRT(AP$POW((V0068+V006
     x9),(-V0076))))
12074 CONTINUE
      V0078 = (((AP$POW(V0051,(-(V0066*V0068)))/(V0069+V0071))*T2044)/((
     x(LOGBASEB(V0064,V0068)+ARCTAN((AP$POW(V0068,V0071)/AP$POW(V0069,(V
     x0071-0.1000D1)))))/(SQRT((V0072/(V0073+(V0065/V0069))))+((V0072+(V
     x0069/V0073))/V0068)))+T2074))
      F0078 = V0078
10078 CONTINUE
      RETURN
      END
      REAL*8 FUNCTION F0080(V0079)
      IMPLICIT REAL*8(T)
      REAL*8 V0079
      REAL*8 V0080
      COMMON /APABWDKI/ IMAGEL,V0051,V0058,V0059,V0064,V0065,V0066,V0068
     x,V0069,V0070,V0071,V0072,V0073,V0074,V0075,V0076,V0067
      EXTERNAL AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,F0078
      REAL*8 AP$POW,AP$INTGR,ARCTAN,SIN,COS,LOGBASEB,V0051,V0058,V0059,V
     x0064,V0065,V0066,V0067(0:79,0:9),V0068,V0069,V0070,V0071,V0072,V00
     x73,V0074,V0075,V0076,F0078
      V0080 = 0.0
      V0069 = V0079
      V0080 = AP$INTGR(V0059,0.3000D1,SQRT((V0069/V0071)),F0078)
      F0080 = V0080
10080 CONTINUE
      RETURN
      END
      BLOCK DATA APBBWDKI
      COMMON /APABWDKI/ IMAGEL,V0051,V0058,V0059,V0064,V0065,V0066,V0068
     x,V0069,V0070,V0071,V0072,V0073,V0074,V0075,V0076,V0067
      REAL*8 V0051,V0058,V0059,V0064,V0065,V0066,V0067(0:79,0:9),V0068,V
     x0069,V0070,V0071,V0072,V0073,V0074,V0075,V0076
      DATA V0051/0.2718281828459045D1/,V0058/0.0000000100000000D0/,V0059
     x/0.0000100000000000D0/
      END
```

Fig. 2. Fortran code produced by the AUTOMATED PROGRAMMER for the double integral in Fig. 1.

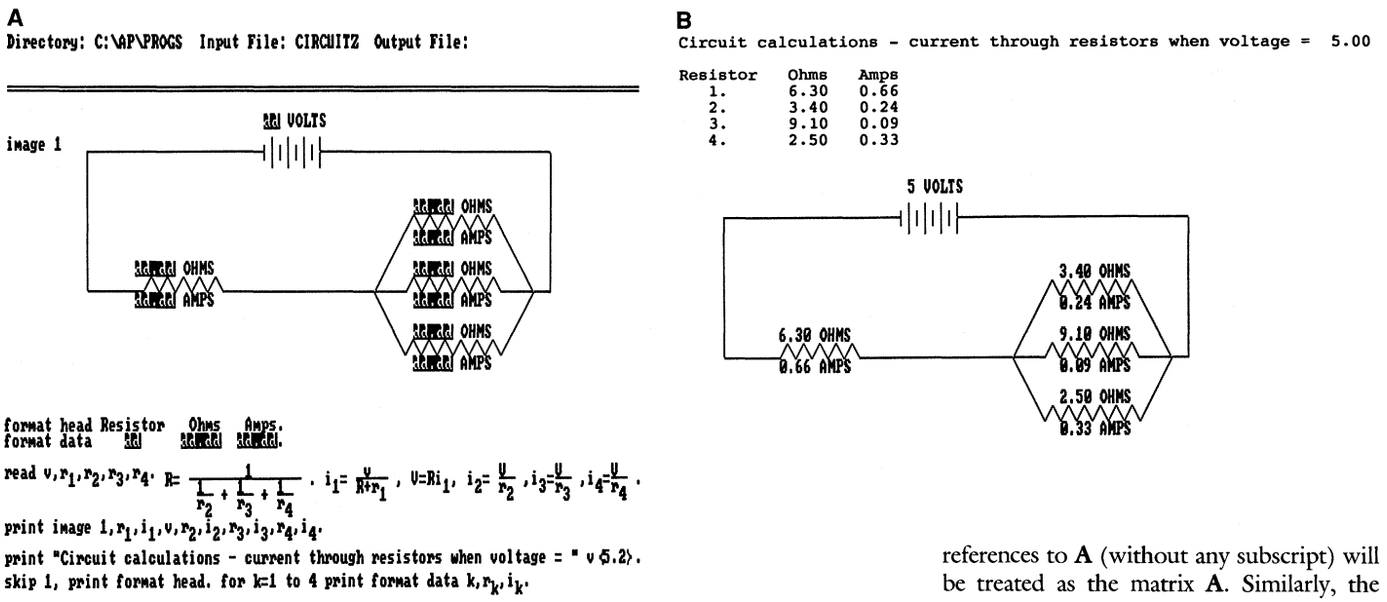

Fig. 3. (**A**) Program for calculating currents in an electrical network. (**B**) Output for a given problem.

includes an image which describes how the output should appear on the screen or printer. It includes a pictorial representation of the circuit diagram. Without this capability, as in an ordinary computational system for a problem of this nature, the output may appear as a list of the resistor numbers showing their value in ohms and their computed amperages in amperes when given the initial value of the voltage.

This tabular form of output is usually neat and well labeled. However, when the output is given in this form, there remains a translation process; namely, associating the resultant values with the proper resistors on the circuit diagram. The output from this program is shown in Fig. 3B. The top portion of this output is the typical display that a user might expect to see from an "ordinary" computation. Compare and contrast this portion of the output with the bottom portion of the same output. It displays the same data computed by the same program, but this time the image feature is used to show precisely which value is associated with which resistor directly on the diagram. It is readily apparent which style of output is more readable and understandable.

The user is free to design any sort of image for any problem to be solved. The only limit is the imagination of the user. This capability is a boon to the people that use this AUTOMATED PROGRAMMER System in that output cannot only be computed, but designed to appeal to the eye of the user, leaving little chance for improper understanding of the meaning of the output or lack of proper association of the output data to the problem domain.

Another example of the power of this feature is shown in Fig. 4A, which shows a simple "program" to compute a discrete range of integrals. In an ordinary computational system, the user might expect to see the output presented in a tabular form with perhaps some label describing what these values mean. An example of the output of the "program" is shown in Fig. 4B. Note that labels are not necessary because the image capability allows the user to see the successive values of the discrete range displayed in the output in their respective positions within the formula as well as the computed value; the output is two-dimensional, visually appealing, and meaningful.

Another powerful feature of the AUTOMATED PROGRAMMER is complete support for vector and matrix arithmetic. Any one-dimensional array can be treated as a vector; any two-dimensional array can be treated as a matrix. For example, if an array is identified as $\mathbf{A}_{i,j}$ in a formula, then the AUTOMATED PROGRAMMER will automatically dimension it as a two-dimensional array as noted earlier. Subsequent references to **A** (without any subscript) will be treated as the matrix **A**. Similarly, the vector $\mathbf{V}_i$ in its unsubscripted form V will be treated as the vector **V**.

This feature allows a user to write such things as **A** + **B** to mean the matrix sum of the matrix **A** and the matrix **B**. Combining this with the implied and alternative forms for multiplication that the AUTOMATED PROGRAMMER supplies, the symbols **AB** or **A*****B** or **A** · **B** all mean matrix multiplication. Furthermore, if *s* is some scalar value, then **A** + **s** adds *s* to each element in **A**. Built-in operations for matrices are similarly powerful, such as $\mathbf{A}^t$, which yields the matrix transpose of **A**, and $\mathbf{A}^{-1}$, which yields the matrix inverse of **A**. Also, writing **det A** yields the determinant of **A**.

I have only mentioned a few of the many very powerful features in the AUTOMATED PROGRAMMER System. The user will find that the transformation of the solution to a problem into something that is machine processable need no longer be a difficult or error-prone operation.

System Description

The AUTOMATED PROGRAMMER

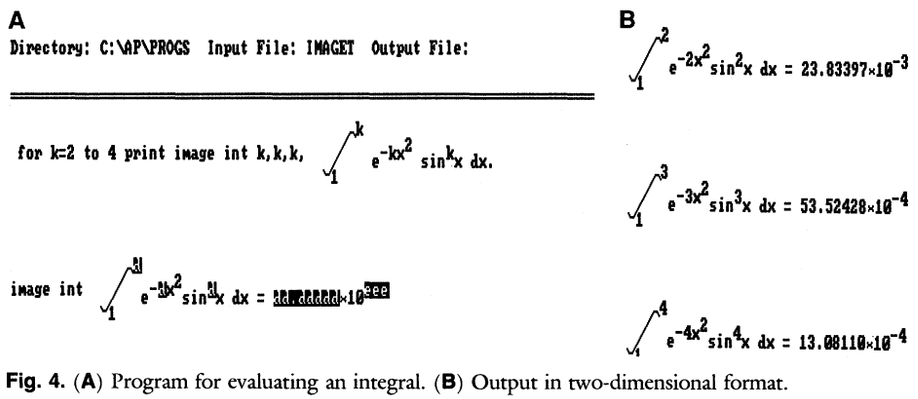

Fig. 4. (**A**) Program for evaluating an integral. (**B**) Output in two-dimensional format.

requires an IBM PC, XT, AT, PS/2, or compatible; a hard disk; at least CGA-compatible graphics; 512-kilobyte memory; PC-DOS; and a Fortran compiler. The math coprocessor for the PC is recommended, although it is not required. The cost of the system is $495.00 (7).

The program is delivered on four floppy disks. It comes with a PC-size binder for the 160-page user's manual. System installation is handled automatically through a .BAT batch file and is quite simple. Telephone consultation is provided for registered users.

The AUTOMATED PROGRAMMER generates Fortran code that is compatible with three Fortran compilers. The user must supply either the Ryan-McFarland Fortran (version 2.10), Lahey Fortran (version 2.22), or Microsoft Fortran (version 4.01). There are some very minor differences in the way each compiler translates the code produced by the AUTOMATED PROGRAMMER, and these differences are explained in the documentation that accompanies the AUTOMATED PROGRAMMER.

Conclusion

As a tool for mathematically sophisticated users in the scientific-engineering-mathematical domain, the AUTOMATED PROGRAMMER is unlike any other product that I have seen. It approaches the ideal of the executable specification, yet is simple and easy to use. Although the developers themselves state that the AUTOMATED PROGRAMMER is not an automatic numerical analyst or an automated problem solver, if a problem can be posed in the standard symbols and technical English of the mathematical domain then it can be solved by the AUTOMATED PROGRAMMER with considerably less effort on the part of the user. The low purchase price of this system is easily recovered by the benefits which it provides. Among these benefits are:

- Increased reliability, attributed to increased readability and understandability.
- Increased maintainability due to the lack of necessity of hand translation from the problem-domain formulation to the solution-domain formulation.
- Increased productivity without the need to become a computer professional as well as a domain expert.
- Savings in time due to the automation of what is now a hand-crafted labor-intensive process.

REFERENCES AND NOTES

1. AUTOMATED PROGRAMMER, KGK Automated Systems, 114 The Colony, Hartsdale, NY 10530.
2. The user must supply the compiler.
3. Actually, in the file created by the AUTOMATED PROGRAMMER editor.
4. Currently only Fortran is supported, but other intermediate forms may be available soon. The C intermediate form is in prototype and an Ada form is planned.
5. Actually, it is the interpretation of the mathematical expression by the AUTOMATED PROGRAMMER and demonstrates the "tightness" of the intermediate code.
6. F. LoSacco, *SIAM News* **1988**, 9 (July 1988).
7. The program was tested on a Zenith 248 system (80286 processor, AT class), with 640 kilobytes of memory, no math coprocessor, a 20-megabyte hard drive, and EGA-compatible graphics.

Chaos on Computers

Chaos appears in areas as diverse as population dynamics, structural mechanics, and economics. However, describing the chaotic behavior of nonlinear systems is somewhat like describing good food—there is no substitute for first-hand experience. Part of this is intrinsic: the study of chaos is the art of teasing complex and varied responses from simple equations. It is best done with numerical methods, and the personal computer is a good laboratory for exploration.

Chaos Demonstrations (version 1.0), written for the IBM PC by J. C. Sprott, a physics professor at the University of Wisconsin, is designed for users of different backgrounds, from the museum goer on through to the advanced undergraduate level (*1*). The opening menu offers 18 different physical or mathematical systems, each portraying a handful of fundamental concepts. Explanatory windows are provided for each demonstration, along with options for plotting different variables or changing the constants in the equations. The computation can be stopped and restarted, and the **"track/untrack"** function permits observing trajectories either as single moving points or as tracings. Some of the demonstrations convert the dynamics into sound. Colors can be changed to give the best rendition while a calculation is running.

Examples from physics, ecology, and pure mathematics cover a lot of ground here. The nonlinear forced pendulum, for instance, takes us back to the playground swing. A sinusoidal forcing function is like the pumping legs, coupling energy into the motion of the swing. The pumping can either match the natural frequency or work against it or move the bob in more mysterious ways. Normal views of the pendulum are given, but there are also phase-space plots (velocity versus position) and the Poincaré section (velocity at $x = 0$ versus the velocity at the previous $x = 0$ crossing). The right choice of frequency and amplitude of kick offers a concrete view of chaos.

Other entrees introduce new flavors of chaos. The nonlinear oscillator (a variant of Duffing's equation with cubic restoring force) shows a progression from periodic motion to period doubling, and then to chaos—the limit cycles are colorful and sharp on the higher resolution monitor. Another selection illustrates the Van der Pol oscillator, which has found application in the study of electronic circuits, lasers, and the pulsating stars called Cepheids. The motion of planets under the gravitational influence of stars is here too, from simple Keplerian ellipses to wild orbital gyrations. The motion around two fixed stars is arresting: the orderly progression of point masses from a celestial starting gate quickly becomes an orbital free-for-all. Another demonstration shows the frantic dance of charged particles in a magnetic quadrupole trap.

The Lorenz attractor, the predator-prey problem, and the logistic equation are all treated well, as are the Mandelbrot and Julia sets, the Weierstrass function, and the Henon Map. The dessert menu includes some programs that simply demonstrate the beauty of fractal geometry. An odd bestiary of fractal snowflakes, ferns, and Sierpinksi gaskets is provided along with a look at diffusion-limited aggregation. The selections on random-walk diffusion, noise, and Conway's game of life, however, seem to have been added as an afterthought, but they are not unwelcome.

Although the explanatory screens are helpful, and despite the above criticism of cookbooks, the user would do well to refer to one of the available texts (*2*). The software is most effective with the higher resolution afforded by the EGA (enhanced graphics adapter), but it will function with a CGA (computer graphics adapter). A math coprocessor is recommended, and the faster 80286 or 80386 computers are best for handling the combined load of calculation and graphics.

David F. Voss, *Science*, 1333 H Street, NW, Washington, DC 20005.

REFERENCES AND NOTES

1. Chaos Demonstrations, Wisc-Ware, 1210 West Dayton Street, Madison, WI 53706, (800) 543-3201 [$50; also available directly from the author at the University of Wisconsin, (608) 262-3595]. Minimum system requirements: IBM PC/XT/AT/PS-2 or compatible; 256 kilobytes of random-access memory; one floppy drive; CGA or HGA graphics; and MS-DOS (or PC-DOS) 2.0 or higher.
2. J. Thompson and H. Stewart, *Nonlinear Dynamics and Chaos* (Wiley, New York, 1986); H.-O. Peitgen and P. Richter, *The Beauty of Fractals* (Springer-Verlag, New York, 1986).