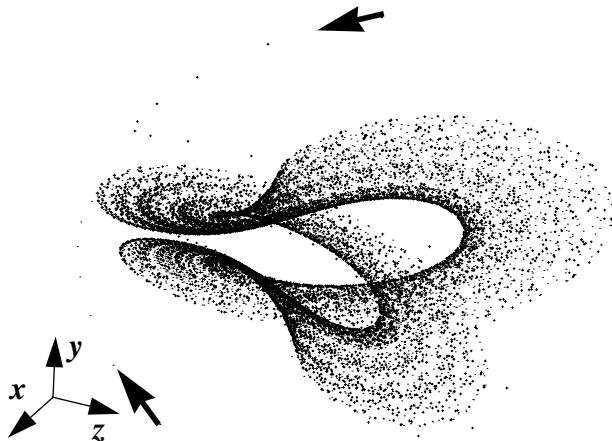


(usually all that is changed is ρ), find the equilibrium points, search for periodic orbits, and observe how small changes in the initial data evolve into large changes in state, even though most orbits soon approach the attractor.



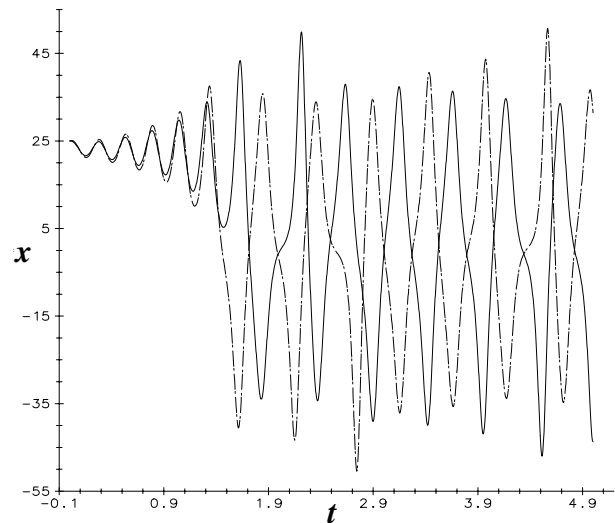
The figure above shows the orbits that start at $(40, -70, 150)$ and $(-40, 110, 220)$ as they approach the attractor. The final figure shows that small changes in initial data (from $(1, 10, 100)$ to $(1.01, 10, 100)$) evolve into markedly different system states.

In a more advanced class, students can check that the equilibrium points destabilize as the parameter ρ increases, verify the appearance of Hopf bifurcations, and observe the first stages of a period-doubling route to chaos.

Sources

W.E. Boyce and R.C. DiPrima, **Elementary Differential Equations and Boundary Value Problems**, John Wiley and Sons, 5th Edition, 1992. (This edition of the "classic" has a section on the Lorenz system.)

R.L. Borrelli, C.S. Coleman, and W.E. Boyce, **Differential Equations Laboratory Workbook**, John Wiley and Sons, 1992. (Two of the experiments are devoted to the Lorenz system.)



J.M.T. Thompson and H.B. Stewart, **Nonlinear Dynamics and Chaos**, John Wiley and Sons, 1986. (An informal presentation of many chaotic systems.)

J. Gleick, **CHAOS, Making a New Science**, Viking, 1987. (A delightful look at the ideas of chaos and the people who started it all.) □

The Savvy Solver III

Larry Shampine

Southern Methodist University
Dallas, TX 75275

h5sr1001@smuvm1.bitnet

(Editor's Note: A function $y(x)$ may be a solution of several differential equations that have no obvious similarity. If a computer differential equation solver is applied to one of these equations in order to generate $y(x)$, the actual output may depend very much on the form of the equation. Here is an example of this and an explanation of how to get around the problem.)

A discrete variable method for the numerical solution of an initial value prob-

lem for $y' = F(x, y)$ starts with the given value $y_0 = y(a) = A$ and successively produces approximations $y_j \approx y(x_j)$ on a mesh $a = x_0 < x_1 < \dots < x_N = b$ that spans the interval $[a, b]$ of interest. It is plausible that the behavior of the numerical method depends solely on the solution $y(x)$. Discussions of numerical methods for the initial value problem generally start with Euler's method: $y_{j+1} = y_j + hF(x_j, y_j)$ where $x_{j+1} = x_j + h$. This formula is often motivated by the Taylor series expansion

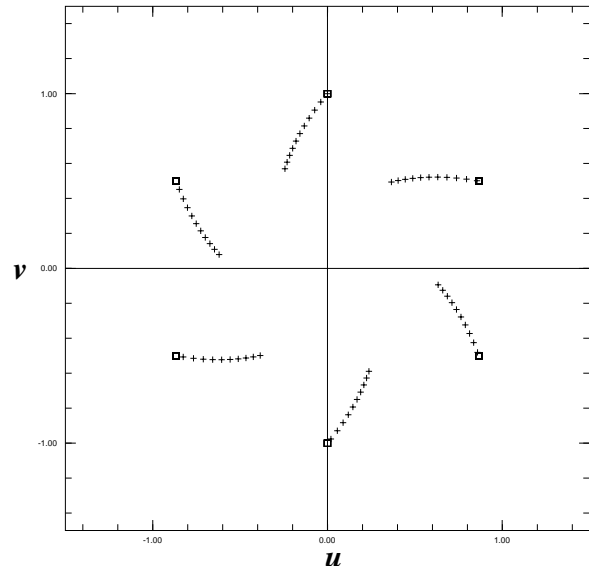
$$y(x_{j+1}) = y(x_j) + hF(x_j, y(x_j)) + \frac{h^2 y''(x_j)}{2} + O(h^3)$$

The expansion shows that the error of the formula depends on the second derivative of the solution and this reinforces an expectation that numerical methods will behave the same when solving problems with the same solution.

Two Examples: Let's test this with some computational experiments. The **quadrature problem** $u' = \cos(x), u(0) = 0, v' = -\sin(x), v(0) = 1$ has the solution $u(x) = \sin(x), v(x) = \cos(x)$. The **linear problem** $u' = v, v' = -u$ with the same initial conditions has the same solution. I used **MDEP** to compute the figure. (Ed. Note: See the Fall 92 issue for a review of **MDEP**).

The figure shows the numerical solutions of the quadrature and linear problems plotted in the phase plane. The boxes correspond to the quadrature problem and the plusses to the linear problem. The problems were integrated over $[0, 20\pi]$ using 60 steps with the classical four stage, fourth order Runge-Kutta formula. The computed solution of the linear problem appears to

spiral clockwise inward with an angular lag. Evidently the results are not even in qualitative agreement.



Accuracy vs Cost: These results show that when using a fixed step size, a difference in F can lead to a difference in the accuracy of the solution. When a code monitors the local accuracy and adjusts the step size so as to meet a tolerance specified by the user, such differences show up in the cost of the integration. To illustrate this I integrated the two problems using the code UT (Usual Task) from **RKSUITE** with relative error tolerances of 10^{-3} and 10^{-5} and thresholds of 10^{-10} . Of the Runge-Kutta formulas of different orders available, I specified the one of order four (a new formula that is much more efficient than the classical formula). It cost 199 and 353 evaluations of f , respectively, to get three digits of local accuracy and 770 and 1161 evaluations to get five. Either way these problems are solved, the numerical results depend not only on $y(x)$, but also on the form of F .

Step Size Constraints: In *The Savvy Solver II* (C·ODE·E, Winter 1993) we talked about how the stability of the differential

equation affects its numerical solution. Now let's talk a little about how the form of the equation can affect its numerical solution. To keep the manipulations simple, we look only at scalar equations of the form

$$y' = \lambda y + g(x)$$

and suppose that $\lambda < 0$ so that the equation is stable. Obviously we must use a step size h small enough that the solution is approximated well by the formula. However, this is not the only constraint on the step size; it must be small enough that the stability of the formula agrees qualitatively with that of the equation. For the simple problems and formulas that we take up, this is easy to analyze.

Suppose that at x_j we have an approximate solution y_j and ask what effect perturbing y_j to u_j would have on subsequent approximations. A little calculation shows that when Euler's method is used,

$(y_{j+1} - u_{j+1}) = (1 + h\lambda)(y_j - u_j)$. The effect of the perturbation grows when the factor $1 + h\lambda$ is larger in magnitude than 1, so if the method is to be stable (like the problem itself), it is necessary that $h \leq 2/|\lambda|$. When stability determines the step size, users are often bewildered because the solution of their problem looks easy to approximate, yet good codes persist in taking absurdly short steps. This issue is an important part of the phenomenon called "stiffness". The point here, however, is that the form of the equation, namely the size of λ , affects the numerical solution by means of a stability constraint on the step size.

Backward Euler: The backward Euler method is the recipe:

$$y_{j+1} = y_j + hF(x_{j+1}, y_{j+1}) .$$

This implicit method can also be motivated by a Taylor series expansion, now about x_{j+1} instead of x_j . Although the backward Euler method has the same accuracy as the forward Euler method, its stability properties are profoundly different. An easy argument like that for the forward Euler method shows that there is no restriction on the step size for stability when integrating these model problems with $\lambda < 0$. We don't, however, get something for nothing -- the bill is presented when we evaluate the implicit formula. Implicit methods are evaluated by "predicting" the solution at x_{j+1} by means of an explicit formula. For example, we might use the forward Euler method to determine y_{j+1} and then set

$$y_{j+1}^{(0)} = y_j + hF(x_{j+1}, y_{j+1}) .$$

This value is improved by simple iteration:

$$y_{j+1}^{(m+1)} = y_j + hF(x_{j+1}, y_{j+1}^{(m)}) .$$

A little calculation shows that

$$y_{j+1} - y_{j+1}^{(m+1)} = h\lambda(y_{j+1} - y_{j+1}^{(m)})$$

hence that we must have $h < 1/|\lambda|$ for convergence. In practice the step size must be rather smaller than this bound because we need rapid convergence. This constraint on the step size is more severe than the stability constraint on the forward Euler method! To exploit the superb stability properties of the backward Euler method, it is necessary to employ a more effective scheme for evaluating the formula. Unfortunately, "more effective" comes at a pretty high price -- another part of stiffness. Here the point is that F plays a role in the cost of evaluating an implicit formula.

Predictor-Corrector: An obvious difficulty when evaluating an implicit formula is deciding when to quit iterating. "Predictor-corrector" methods dodge this difficulty.

These methods are the result of a fixed number of iterations. Many do not appreciate that the behavior of these methods can be very different from that of the underlying implicit, "corrector" formulas. Of course the restriction on the step size for the convergence of the scheme for evaluating the implicit formula falls away for a predictor-corrector pair because it is explicit. More fundamentally, the predictor affects how quickly the implicit formula is evaluated, but not the result of the step.

In the case of a predictor-corrector pair, the predictor affects the result itself. One consequence is that the stability properties can be profoundly different. For example, if we form a predictor-corrector pair of the forward and backward Euler formulas with exactly one correction, we obtain the formula,

$$p_{j+1} = y_j + hF(x_j, y_j)$$

$$y_{j+1} = y_j + hF(x_{j+1}, p_{j+1}) \quad .$$

This formula is stable for the model only when $|1 + h\lambda + h^2\lambda^2| < 1$. In contrast to the backward Euler formula, the predictor-corrector pair is stable only when h is constrained in a way that depends on F .

Truncation Error: The predictor-corrector pair just derived illustrates another way that the behavior of a formula can depend on the form of the equation. To study the error of a formula in a single step, we apply it to a solution $y(x)$ of the model problem and see how big the discrepancy is, the local truncation error. In the case of the error of the forward Euler method, we have

$$y(x_{j+1}) = y(x_j) + h[\lambda y(x_j) + g(x_j)]$$

$$+ \frac{h^2 y''(x_j)}{2} + O(h^3),$$

and in the case of the backward Euler method,

$$y(x_{j+1}) = y(x_j) + h[\lambda y(x_{j+1}) + g(x_{j+1})]$$

$$+ \frac{h^2 y''(x_j)}{2} + O(h^3) \quad .$$

To do this for the predictor-corrector pair, we observe that the expression for the forward Euler method says that

$$y(x_{j+1}) = p_{j+1} + \frac{h^2 y''(x_j)}{2} + O(h^3)$$

and on substituting into the expression for the backward Euler method, we find that

$$y(x_{j+1}) = y(x_j) + h[\lambda p_{j+1} + g(x_{j+1})]$$

$$+ \frac{h^2(1 + \lambda)y''(x_j)}{2} + O(h^3) \quad .$$

Notice that the form of the equation now influences the local truncation error by means of the factor λ . The predictor-corrector pair happens to be a Runge-Kutta formula, so the example shows that both kinds of methods can have a truncation error that depends on F .

Step Size & F: Through simple examples we have seen that the accuracy of a step can depend on F as well as $y(x)$, and for popular predictor-corrector and Runge-Kutta formulas, it normally does. The step size must be small enough for the formula to be accurate, but in addition it must be small enough for the stability of the method to imitate the stability of the problem. For this reason all the explicit methods have constraints on the step size that depend on F . When non-stiff problems are solved with implicit methods, the convergence of the iteration used for evaluating the formula also imposes a constraint on the step size that depends on F . □