```
    free_vector(ysav,1,nv);
    free_vector(yerr,1,nv);
    free_vector(x,1,KMAXX);
    free_vector(err,1,KMAXX);
    free_matrix(dfdy,1,nv,1,nv);
    free_vector(dfdx,1,nv);
    free_matrix(d,1,nv,1,KMAXX);
}
```

The routine stifbs is an excellent routine for all stiff problems, competitive with the best Gear-type routines. stiff is comparable in execution time for moderate N and $\epsilon \lesssim 10^{-4}$. By the time $\epsilon \sim 10^{-8}$, stifbs is roughly an order of magnitude faster. There are further improvements that could be applied to stifbs to make it even more robust. For example, very occasionally ludcmp in simpr will encounter a singular matrix. You could arrange for the stepsize to be reduced, say by a factor of the current nseq[k]. There are also certain stability restrictions on the stepsize that come into play on some problems. For a discussion of how to implement these automatically, see [6].

CITED REFERENCES AND FURTHER READING:

Gear, C.W. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, NJ: Prentice-Hall). [1]

Kaps, P., and Rentrop, P. 1979, *Numerische Mathematik*, vol. 33, pp. 55–68. [2]

Shampine, L.F. 1982, *ACM Transactions on Mathematical Software*, vol. 8, pp. 93–113. [3]

Enright, W.H., and Pryce, J.D. 1987, *ACM Transactions on Mathematical Software*, vol. 13, pp. 1–27. [4]

Bader, G., and Deuflhard, P. 1983, *Numerische Mathematik*, vol. 41, pp. 373–398. [5]

Deuflhard, P. 1983, *Numerische Mathematik*, vol. 41, pp. 399–422.

Deuflhard, P. 1985, *SIAM Review*, vol. 27, pp. 505–535.

Deuflhard, P. 1987, "Uniqueness Theorems for Stiff ODE Initial Value Problems," *Preprint SC-87-3* (Berlin: Konrad Zuse Zentrum für Informationstechnik). [6]

Enright, W.H., Hull, T.E., and Lindberg, B. 1975, *BIT*, vol. 15, pp. 10–48.

Wanner, G. 1988, in *Numerical Analysis 1987*, Pitman Research Notes in Mathematics, vol. 170, D.F. Griffiths and G.A. Watson, eds. (Harlow, Essex, U.K.: Longman Scientific and Technical).

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag).

16.7 Multistep, Multivalue, and Predictor-Corrector Methods

The terms multistep and multivalue describe two different ways of implementing essentially the same integration technique for ODEs. Predictor-corrector is a particular subcategrory of these methods — in fact, the most widely used. Accordingly, the name predictor-corrector is often loosely used to denote all these methods.

We suspect that predictor-corrector integrators have had their day, and that they are no longer the method of choice for most problems in ODEs. For high-precision applications, or applications where evaluations of the right-hand sides are expensive, Bulirsch-Stoer dominates. For convenience, or for low precision, adaptive-stepsize Runge-Kutta dominates. Predictor-corrector methods have been, we think, squeezed out in the middle. There is possibly only one exceptional case: high-precision solution of very smooth equations with very complicated right-hand sides, as we will describe later.

Nevertheless, these methods have had a long historical run. Textbooks are full of information on them, and there are a lot of standard ODE programs around that are based on predictor-corrector methods. Many capable researchers have a lot of experience with predictor-corrector routines, and they see no reason to make a precipitous change of habit. It is not a bad idea for you to be familiar with the principles involved, and even with the sorts of bookkeeping details that are the bane of these methods. Otherwise there will be a big surprise in store when you first have to fix a problem in a predictor-corrector routine.

Let us first consider the multistep approach. Think about how integrating an ODE is different from finding the integral of a function: For a function, the integrand has a known dependence on the independent variable x, and can be evaluated at will. For an ODE, the "integrand" is the right-hand side, which depends both on x and on the dependent variables y. Thus to advance the solution of $y' = f(x, y)$ from x_n to x, we have

$$y(x) = y_n + \int_{x_n}^{x} f(x', y)\, dx' \tag{16.7.1}$$

In a single-step method like Runge-Kutta or Bulirsch-Stoer, the value y_{n+1} at x_{n+1} depends only on y_n. In a multistep method, we approximate $f(x, y)$ by a polynomial passing through *several* previous points $x_n, x_{n-1}, \ldots$ and possibly also through x_{n+1}. The result of evaluating the integral (16.7.1) at $x = x_{n+1}$ is then of the form

$$y_{n+1} = y_n + h(\beta_0 y'_{n+1} + \beta_1 y'_n + \beta_2 y'_{n-1} + \beta_3 y'_{n-2} + \cdots) \tag{16.7.2}$$

where y'_n denotes $f(x_n, y_n)$, and so on. If $\beta_0 = 0$, the method is explicit; otherwise it is implicit. The order of the method depends on how many previous steps we use to get each new value of y.

Consider how we might solve an implicit formula of the form (16.7.2) for y_{n+1}. Two methods suggest themselves: *functional iteration* and *Newton's method*. In functional iteration, we take some initial guess for y_{n+1}, insert it into the right-hand side of (16.7.2) to get an updated value of y_{n+1}, insert this updated value back into the right-hand side, and continue iterating. But how are we to get an initial guess for y_{n+1}? Easy! Just use some *explicit* formula of the same form as (16.7.2). This is called the *predictor step*. In the predictor step we are essentially *extrapolating* the polynomial fit to the derivative from the previous points to the new point x_{n+1} and then doing the integral (16.7.1) in a Simpson-like manner from x_n to x_{n+1}. The subsequent Simpson-like integration, using the prediction step's value of y_{n+1} to *interpolate* the derivative, is called the *corrector step*. The difference between the predicted and corrected function values supplies information on the local truncation error that can be used to control accuracy and to adjust stepsize.

If one corrector step is good, aren't many better? Why not use each corrector as an improved predictor and iterate to convergence on each step? Answer: Even if you had a *perfect* predictor, the step would still be accurate only to the finite order of the corrector. This incurable error term is on the same order as that which your iteration is supposed to cure, so you are at best changing only the coefficient in front of the error term by a fractional amount. So dubious an improvement is certainly not worth the effort. Your extra effort would be better spent in taking a smaller stepsize.

As described so far, you might think it desirable or necessary to predict several intervals ahead at each step, then to use all these intervals, with various weights, in a Simpson-like corrector step. That is not a good idea. Extrapolation is the least stable part of the procedure, and it is desirable to minimize its effect. Therefore, the integration steps of a predictor-corrector method are overlapping, each one involving several stepsize intervals h, but extending just one such interval farther than the previous ones. Only that one extended interval is extrapolated by each predictor step.

The most popular predictor-corrector methods are probably the Adams-Bashforth-Moulton schemes, which have good stability properties. The Adams-Bashforth part is the predictor. For example, the third-order case is

$$\text{predictor:} \qquad y_{n+1} = y_n + \frac{h}{12}(23y'_n - 16y'_{n-1} + 5y'_{n-2}) + O(h^4) \qquad (16.7.3)$$

Here information at the current point x_n, together with the two previous points x_{n-1} and x_{n-2} (assumed equally spaced), is used to predict the value y_{n+1} at the next point, x_{n+1}. The Adams-Moulton part is the corrector. The third-order case is

$$\text{corrector:} \qquad y_{n+1} = y_n + \frac{h}{12}(5y'_{n+1} + 8y'_n - y'_{n-1}) + O(h^4) \qquad (16.7.4)$$

Without the trial value of y_{n+1} from the predictor step to insert on the right-hand side, the corrector would be a nasty implicit equation for y_{n+1}.

There are actually three separate processes occurring in a predictor-corrector method: the predictor step, which we call P, the evaluation of the derivative y'_{n+1} from the latest value of y, which we call E, and the corrector step, which we call C. In this notation, iterating m times with the corrector (a practice we inveighed against earlier) would be written P(EC)^m. One also has the choice of finishing with a C or an E step. The lore is that a final E is superior, so the strategy usually recommended is PECE.

Notice that a PC method with a fixed number of iterations (say, one) is an explicit method! When we fix the number of iterations in advance, then the final value of y_{n+1} can be written as some complicated function of known quantities. Thus fixed iteration PC methods lose the strong stability properties of implicit methods and *should only be used for nonstiff problems*.

For stiff problems we *must* use an implicit method if we want to avoid having tiny stepsizes. (Not all implicit methods are good for stiff problems, but fortunately some good ones such as the Gear formulas are known.) We then appear to have two choices for solving the implicit equations: functional iteration to convergence, or Newton iteration. However, it turns out that for stiff problems functional iteration will not even converge unless we use tiny stepsizes, no matter how close our prediction is! Thus Newton iteration is usually an essential part of a multistep stiff solver. For convergence, Newton's method doesn't particularly care what the stepsize is, as long as the prediction is accurate enough.

Multistep methods, as we have described them so far, suffer from two serious difficulties when one tries to implement them:

- Since the formulas require results from equally spaced steps, adjusting the stepsize is difficult.

- Starting and stopping present problems. For starting, we need the initial values plus several previous steps to prime the pump. Stopping is a problem because equal steps are unlikely to land directly on the desired termination point.

Older implementations of PC methods have various cumbersome ways of dealing with these problems. For example, they might use Runge-Kutta to start and stop. Changing the stepsize requires considerable bookkeeping to do some kind of interpolation procedure. Fortunately both these drawbacks disappear with the multivalue approach.

For multivalue methods the basic data available to the integrator are the first few terms of the Taylor series expansion of the solution at the current point x_n. The aim is to advance the solution and obtain the expansion coefficients at the next point x_{n+1}. This is in contrast to multistep methods, where the data are the values of the solution at $x_n, x_{n-1}, \ldots$. We'll illustrate the idea by considering a four-value method, for which the basic data are

$$\mathbf{y}_n \equiv \begin{pmatrix} y_n \\ hy'_n \\ (h^2/2)y''_n \\ (h^3/6)y'''_n \end{pmatrix} \tag{16.7.5}$$

It is also conventional to scale the derivatives with the powers of $h = x_{n+1} - x_n$ as shown. Note that here we use the vector notation $\mathbf{y}$ to denote the solution and its first few derivatives at a point, not the fact that we are solving a system of equations with many components y.

In terms of the data in (16.7.5), we can approximate the value of the solution y at some point x:

$$y(x) = y_n + (x - x_n)y'_n + \frac{(x - x_n)^2}{2}y''_n + \frac{(x - x_n)^3}{6}y'''_n \tag{16.7.6}$$

Set $x = x_{n+1}$ in equation (16.7.6) to get an approximation to y_{n+1}. Differentiate equation (16.7.6) and set $x = x_{n+1}$ to get an approximation to y'_{n+1}, and similarly for y''_{n+1} and y'''_{n+1}. Call the resulting approximation $\widetilde{\mathbf{y}}_{n+1}$, where the tilde is a reminder that all we have done so far is a polynomial extrapolation of the solution and its derivatives; we have not yet used the differential equation. You can easily verify that

$$\widetilde{\mathbf{y}}_{n+1} = \mathbf{B} \cdot \mathbf{y}_n \tag{16.7.7}$$

where the matrix $\mathbf{B}$ is

$$\mathbf{B} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 2 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{16.7.8}$$

We now write the actual approximation to $\mathbf{y}_{n+1}$ that we will use by adding a correction to $\widetilde{\mathbf{y}}_{n+1}$:

$$\mathbf{y}_{n+1} = \widetilde{\mathbf{y}}_{n+1} + \alpha\mathbf{r} \tag{16.7.9}$$

Here **r** will be a fixed vector of numbers, in the same way that **B** is a fixed matrix. We fix α by requiring that the differential equation

$$y'_{n+1} = f(x_{n+1}, y_{n+1}) \tag{16.7.10}$$

be satisfied. The second of the equations in (16.7.9) is

$$hy'_{n+1} = h\widetilde{y}'_{n+1} + \alpha r_2 \tag{16.7.11}$$

and this will be consistent with (16.7.10) provided

$$r_2 = 1, \qquad \alpha = hf(x_{n+1}, y_{n+1}) - h\widetilde{y}'_{n+1} \tag{16.7.12}$$

The values of r_1, r_3, and r_4 are free for the inventor of a given four-value method to choose. Different choices give different orders of method (i.e., through what order in h the final expression 16.7.9 actually approximates the solution), and different stability properties.

An interesting result, not obvious from our presentation, is that multivalue and multistep methods are entirely equivalent. In other words, the value y_{n+1} given by a multivalue method with given **B** and **r** is exactly the same value given by some multistep method with given β's in equation (16.7.2). For example, it turns out that the Adams-Bashforth formula (16.7.3) corresponds to a four-value method with $r_1 = 0$, $r_3 = 3/4$, and $r_4 = 1/6$. The method is explicit because $r_1 = 0$. The Adams-Moulton method (16.7.4) corresponds to the implicit four-value method with $r_1 = 5/12$, $r_3 = 3/4$, and $r_4 = 1/6$. Implicit multivalue methods are solved the same way as implicit multistep methods: either by a predictor-corrector approach using an explicit method for the predictor, or by Newton iteration for stiff systems.

Why go to all the trouble of introducing a whole new method that turns out to be equivalent to a method you already knew? The reason is that multivalue methods allow an easy solution to the two difficulties we mentioned above in actually implementing multistep methods.

Consider first the question of stepsize adjustment. To change stepsize from h to h' at some point x_n, simply multiply the components of $\mathbf{y}_n$ in (16.7.5) by the appropriate powers of h'/h, and you are ready to continue to $x_n + h'$.

Multivalue methods also allow a relatively easy change in the *order* of the method: Simply change **r**. The usual strategy for this is first to determine the new stepsize with the current order from the error estimate. Then check what stepsize would be predicted using an order one greater and one smaller than the current order. Choose the order that allows you to take the biggest next step. Being able to change order also allows an easy solution to the starting problem: Simply start with a first-order method and let the order automatically increase to the appropriate level.

For low accuracy requirements, a Runge-Kutta routine like `rkqs` is almost always the most efficient choice. For high accuracy, `bsstep` is both robust and efficient. For very smooth functions, a variable-order PC method can invoke very high orders. If the right-hand side of the equation is relatively complicated, so that the expense of evaluating it outweighs the bookkeeping expense, then the best PC packages can outperform Bulirsch-Stoer on such problems. As you can imagine, however, such a variable-stepsize, variable-order method is not trivial to program. If you suspect that your problem is suitable for this treatment, we recommend use of a canned PC package. For further details consult Gear [1] or Shampine and Gordon [2].

Our prediction, nevertheless, is that, as extrapolation methods like Bulirsch-Stoer continue to gain sophistication, they will eventually beat out PC methods in all applications. We are willing, however, to be corrected.

CITED REFERENCES AND FURTHER READING:

Gear, C.W. 1971, *Numerical Initial Value Problems in Ordinary Differential Equations* (Englewood Cliffs, NJ: Prentice-Hall), Chapter 9. [1]

Shampine, L.F., and Gordon, M.K. 1975, *Computer Solution of Ordinary Differential Equations. The Initial Value Problem.* (San Francisco: W.H Freeman). [2]

Acton, F.S. 1970, *Numerical Methods That Work*; 1990, corrected edition (Washington: Mathematical Association of America), Chapter 5.

Kahaner, D., Moler, C., and Nash, S. 1989, *Numerical Methods and Software* (Englewood Cliffs, NJ: Prentice Hall), Chapter 8.

Hamming, R.W. 1962, *Numerical Methods for Engineers and Scientists*; reprinted 1986 (New York: Dover), Chapters 14–15.

Stoer, J., and Bulirsch, R. 1980, *Introduction to Numerical Analysis* (New York: Springer-Verlag), Chapter 7.