dy[j]=yest[j];
}
else {
    for (k=1;k<iest;k++)
        fx[k+1]=x[iest-k]/xest;
    for (j=1;j<nv;j++) {
        Evaluate next diagonal in tableau.
        v=d[j][1];
        d[j][1]=yy=c=yest[j];
        for (k=2;k<iest;k++) {
            b1=fx[k]*v;
            b=b1-c;
            if (b) {
                b=(c-v)/b;
                ddy=c*b;
                c=b1*b;
            } else 
                Care needed to avoid division by 0.
            ddy=v;
            if (k +i est) v=d[j][k];
            d[j][k]=ddy;
            yy += ddy;
        }
        dy[j]=ddy;
        yz[j]=yy;
    }
    free_vector(fx,1,iest);
}

CITED REFERENCES AND FURTHER READING:

16.5 Second-Order Conservative Equations

Usually when you have a system of high-order differential equations to solve it is best to reformulate them as a system of first-order equations, as discussed in §16.0. There is a particular class of equations that occurs quite frequently in practice where you can gain about a factor of two in efficiency by differencing the equations directly. The equations are second-order systems where the derivative does not appear on the right-hand side:

\[ y'' = f(x, y), \quad y(x_0) = y_0, \quad y'(x_0) = z_0 \quad (16.5.1) \]

As usual, \( y \) can denote a vector of values.

Stoermer’s rule, dating back to 1907, has been a popular method for discretizing such systems. With \( h = H/m \) we have

\[
\begin{align*}
y_k &= y_0 + \frac{1}{2} hf(x_0, y_0) \\
y_{k+1} &= 2y_k + y_{k-1} - \frac{1}{2} h^2 f(x_0 + kh, y_k), \quad k = 1, \ldots, m - 1 \\
z_m &= (y_m - y_{m-1})/h + \frac{1}{2} hf(x_0 + H, y_m)
\end{align*}
\quad (16.5.2)
\]
Here \( z_m \) is \( y'(x_0 + H) \). Henrici showed how to rewrite equations (16.5.2) to reduce roundoff error by using the quantities \( \Delta_k \equiv y_{k+1} - y_k \). Start with

\[
\Delta_0 = h[z_0 + \frac{1}{2}hf(x_0, y_0)] \\
y_1 = y_0 + \Delta_0 \tag{16.5.3}
\]

Then for \( k = 1, \ldots, m-1 \), set

\[
\Delta_k = \Delta_{k-1} + h^2 f(x_0 + kh, y_k) \\
y_{k+1} = y_k + \Delta_k \tag{16.5.4}
\]

Finally compute the derivative from

\[
z_m = \frac{\Delta_{m-1}}{h} + \frac{1}{2}hf(x_0 + H, y_m) \tag{16.5.5}
\]

Gragg again showed that the error series for equations (16.5.3)±(16.5.5) contains only even powers of \( h \), and so the method is a logical candidate for extrapolation à la Bulirsch-Stoer.

We replace \texttt{mmid} by the following routine \texttt{stoerm}:

```c
#include "nrutil.h"

void stoerm(float y[], float d2y[], int nv, float xs, float htot, int nstep, 
            float yout[], void (*derivs)(float, float [], float []))

Stoermer's rule for integrating \( y'' = f(x, y) \) for a system of \( n = \frac{nv}{2} \) equations. On input \( y[1..nv] \) contains \( y \) in its first \( n \) elements and \( y' \) in its second \( n \) elements, all evaluated at \( xs \). \( d2y[1..nv] \) contains the right-hand side function \( f \) (also evaluated at \( xs \)) in its first \( n \) elements. Its second \( n \) elements are not referenced. Also input is \( htot \), the total step to be taken, and \( nstep \), the number of substeps to be used. The output is returned as \( yout[1..nv] \), with the same storage arrangement as \( y \). \( derivs \) is the user-supplied routine that calculates \( f \).
{
    int i,n,neqns,nn;
    float h,h2,halfh,x,*ytemp;
    ytemp=vector(1,nv);
    h=htot/nstep; 
    halfh=0.5*h; 
    neqns=nv/2; 
    for (i=1;i<=neqns;i++) {
        n=neqns+i; 
        ytemp[i]=y[i]+(ytemp[n]=h*(y[n]+halfh*d2y[i])); 
    }
    x=xs+h;
    (*derivs)(x,ytemp,yout);
    h2=h*h;
    for (nn=2;nn<=nstep;nn++) {
        General step.
        for (i=1;i<=neqns;i++) 
            ytemp[i] += (ytemp[n=neqns+i]) += h2*yout[i]); 
        x += h;
        (*derivs)(x,ytemp,yout);
    }
    for (i=1;i<=neqns;i++) {
        Last step.
        n=neqns+i;
        yout[n]=ytemp[n]/h+halfh*yout[i];
        yout[i]=ytemp[i];
    }
    free_vector(ytemp,1,nv);
}
```
Note that for compatibility with \texttt{bsstep} the arrays \texttt{y} and \texttt{d2y} are of length \(2n\) for a system of \(n\) second-order equations. The values of \(y\) are stored in the first \(n\) elements of \(y\), while the first derivatives are stored in the second \(n\) elements. The right-hand side \(f\) is stored in the first \(n\) elements of the array \texttt{d2y}; the second \(n\) elements are unused. With this storage arrangement you can use \texttt{bsstep} simply by replacing the call to \texttt{mmid} with one to \texttt{stoerm} using the same arguments; just be sure that the argument \texttt{nv} of \texttt{bsstep} is set to \(2n\). You should also use the more efficient sequence of step sizes suggested by Deuflhard:

\[
n = 1, 2, 3, 4, 5, \ldots
\] (16.5.6)

and set \( \texttt{KMAXX} = 12 \) in \texttt{bsstep}.

CITED REFERENCES AND FURTHER READING:

16.6 Stiff Sets of Equations

As soon as one deals with more than one first-order differential equation, the possibility of a \textit{stiff} set of equations arises. Stiffness occurs in a problem where there are two or more very different scales of the independent variable on which the dependent variables are changing. For example, consider the following set of equations [1]:

\[
\begin{align*}
  u' &= 998u + 1998v \\
  v' &= -999u - 1999v
\end{align*}
\] (16.6.1)

with boundary conditions

\[
  u(0) = 1 \quad v(0) = 0
\] (16.6.2)

By means of the transformation

\[
  u = 2y - z \quad v = -y + z
\] (16.6.3)

we find the solution

\[
\begin{align*}
  u &= 2e^{-x} - e^{-1000x} \\
  v &= -e^{-x} + e^{-1000x}
\end{align*}
\] (16.6.4)

If we integrated the system (16.6.1) with any of the methods given so far in this chapter, the presence of the \(e^{-1000x}\) term would require a stepsize \(h \ll 1/1000\) for the method to be stable (the reason for this is explained below). This is so even