Intro to Numerical Computations by Yakowitz & Szidarovszky

*7.7 ADAPTIVE STEP-SIZE SELECTION AND ERROR CONTROL

Up to this point we have not discussed how the step size h of the preceding methods is to be chosen. Obviously, there is a trade-off to be made: If the step size is too small, then computer time is needlessly wasted and accumulation of arithmetic roundoff errors can become a hazard. A large step size invites large truncation error associated with higher-order terms neglected in the construction of the methods. For simplicity, our developments will be concerned only with Runge–Kutta rules.

Techniques for automatic step-size selection are based on estimating the local error at each step and then choosing the step size to keep this estimated error within some tolerance bound. Thus step-size selection hinges on estimation of the *local error*, which at the jth step is defined to be

$$\hat{y}(x_{j+1}) - y_{j+1}.$$

Here y_{j+1} is, of course, the computed approximation of $y(x_{j+1})$, and $\hat{y}(x_{j+1})$ we define to be the exact value at x_{j+1} of the differential equation solution that passes through the point (x_j, y_j). That is, $\hat{y}(x)$ solves the initial-value problem

$$\hat{y}' = f(x, \hat{y}), \qquad \hat{y}(x_j) = y_j.$$

In contrast to local errors, the *global error* at x_{j+1} is defined to be

$$y(x_{j+1}) - y_{j+1},$$

where $y(x)$ is the exact solution of the original initial-value problem (7.3). Figure 7.7 illustrates the relationships between $y(x)$, $\hat{y}(x)$, and local and global errors. Intuitively, the local error is the additional truncation error arising from inexact solution at a given step. The global error gives the accumulated total error propagating from the entire sequence of steps.

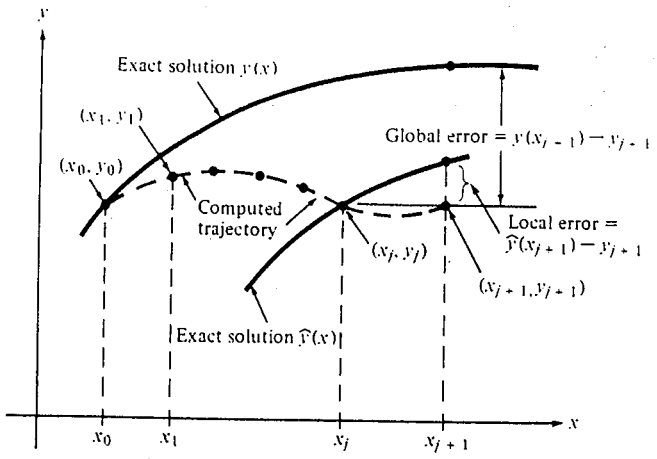

FIGURE 7–7 Relationship Between $y(x)$, $\hat{y}(x)$, Local and Global Errors

Assume that some Runge–Kutta procedure has been selected. We let y_0, y_1, y_2, . . . denote the computed solution values at the arguments x_0, x_1, x_2, ... The local error estimation techniques at each stage apply a higher-order technique to compute an additional approximation, say z_{j+1}, of $y(x_{j+1})$. Since a higher-order technique is used, if the solution is "well behaved" and the step size h is small enough that neglected terms really are negligible, then one may anticipate that the local error of the higher-order method is much less than that of the selected Runge–Kutta procedure. That is,

$$|\hat{y}(x_{j+1}) - z_{j+1}| << |\hat{y}(x_{j+1}) - y_{j+1}|. \tag{7.51}$$

If the approximation above indeed holds, then

$$z_{j+1} - y_{j+1} \simeq \hat{y}(x_{j+1}) - y_{j+1}. \tag{7.52}$$

and we take $z_{j+1} - y_{j+1}$ as the estimate of local error.

Of course, computation of z_{j+1} is typically more expensive than that of y_{j+1} itself, since z_{j+1} must be more accurate. Here, as in other walks of life, information must be paid for. A popular idea toward making this expense as small as possible has been offered by Fehlberg (1964). For a given order, say $p + 1$, the corresponding member of the Fehlberg family computes z_{j+1} with a minimum number of function calls, according to the limitations in Table 7.10, and then provides the pth-order estimate y_{j+1} without any additional function calls. A particularly popular Fehlberg rule is given in Table 7.21, which gives a fifth-order estimate z_{j+1} for a fourth-order rule y_{j+1}.

Subroutine RKF (Table 7.22) implements a single step of this Runge–Kutta–Fehlberg formula, outputting y_{j+1} and z_{j+1} as the parameters YOUT and ZOUT. In view of (7.52), the difference of these values provides a local error estimate. Subroutine ARUKU (Table 7.23) utilizes RKF to update the step size as the computation progresses. If the absolute value of ZOUT-YOUT is less than

TABLE 7.21 Runge–Kutta–Fehlberg Formula

$$k_1 = f(x_j, y_j)$$

$$k_2 = f\left(x_j + \frac{1}{4}h,\ y_j + \frac{1}{4}hk_1\right)$$

$$k_3 = f\left(x_j + \frac{3}{8}h,\ y_j + h\left(\frac{3}{32}k_1 + \frac{9}{32}k_2\right)\right)$$

$$k_4 = f\left(x_j + \frac{12}{13}h,\ y_j + h\left(\frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right)\right)$$

$$k_5 = f\left(x_j + h,\ y_j + h\left(\frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right)\right)$$

$$k_6 = f\left(x_j + \frac{1}{2}h,\ y_j + h\left(-\frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right)\right)$$

$$y_{j+1} = y_j + h\left(\frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5\right)$$

$$z_{j+1} = y_j + h\left(\frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6\right)$$

TABLE 7.22 Subroutine RKF for the Runge–Kutta–Fehlberg Formula

```
      SUBROUTINE RKF(XI,YI,H,YOUT,ZOUT)
C
C  ************************************************************
C  *  FUNCTION: A CALL TO THIS SUBROUTINE COMPUTES ONE STEP OF*
C  *            THE SOLUTION AND A GUESS OF THE ERROR FOR A   *
C  *            DIFFERENTIAL EQUATION Y'=F(X,Y) WITH INITIAL  *
C  *            VALUES XI, YI. THIS SOLUTION IS OBTAINED      *
C  *            USING A 4-TH ORDER RUNGE-KUTTA FEHLBERG STEP  *
C  *            METHOD IMBEDDED IN A 5-TH ORDER STEP SOLUTION *
C  *  USAGE:                                                  *
C  *       CALL SEQUENCE: CALL RKF(XI,YI,H,YOUT,ZOUT)         *
C  *       EXTERNAL FUNCTIONS/SUBROUTINES: FUNCTION F(U,V)    *
C  *  PARAMETERS:                                             *
C  *       INPUT:                                             *
C  *          XI=INDEPENDENT VARIABLE INITIAL VALUE           *
C  *          YI=DEPENDENT VARIABLE INITIAL VALUE             *
C  *           H=INTERVAL STEP SIZE                           *
C  *      OUTPUT:                                             *
C  *        YOUT=4-TH ORDER SOLUTION ESTIMATE                 *
C  *        ZOUT=5-TH ORDER SOLUTION ESTIMATE                 *
C  *             (ZOUT-YOUT=LOCAL ERROR ESTIMATE)             *
C  ************************************************************
C
      REAL K1,K2,K3,K4,K5,K6
      K1=F(XI,YI)
      U=XI+0.25*H
      V=YI+0.25*H*K1
      K2=F(U,V)
      U=XI+(3./8.)*H
      V=YI+H*((3./32.)*K1+(9./32.)*K2)
      K3=F(U,V)
      U=XI+H*(12./13.)
      V=YI+(H/2197.)*(1932.*K1-7200.*K2+7296.*K3)
      K4=F(U,V)
      U=XI+H
      V=YI+H*((439./216.)*K1-8.*K2+(3680./513.)*K3-
     1(845./4104.)*K4)
      K5=F(U,V)
      U=XI+0.5*H
      V=-(8./27.)*K1+2.*K2-(3544./2565.)*K3+
     1(1859./4104.)*K4-(11./40.)*K5
      V=YI+H*V
      K6=F(U,V)
      YOUT=(25./216.)*K1+(1408./2565.)*K3+
     1(2197./4104.)*K4-K5/5.
      YOUT=YI+H*YOUT
      ZOUT=(16./135.)*K1+(6656./12825.)*K3+
     1(28561./56430.)*K4-(9./50.)*K5
      ZOUT=ZOUT+(2./55.)*K6
      ZOUT=YI+H*ZOUT
      RETURN
      END
```

the user-specified value TOL (for tolerance), the value ZOUT is accepted for y_{j+1}, and a larger step size (by a factor of 3) is chosen for the next step. Otherwise, h is reduced by a factor of 10, and the computation is repeated from the same condition x_j and y_j. Strictly speaking, YOUT, rather than ZOUT, should be chosen for y_{j+1}, but since in principle the higher-order estimate ZOUT should be more accurate, and since it is available, we adopt the pragmatic viewpoint that it should be used. The reader will note that AKUKU is an obvious modification of subroutine ASIMP for adaptive quadrature (Section 3.8.2).

TABLE 7.23 Subroutine AKUKU for the Adaptive Runge–Kutta Method

```
      SUBROUTINE ARUKU(X,Y,B,M,TOL)
C
C     ***************************************************************
C     *  FUNCTION: THIS SUBROUTINE COMPUTES THE SOLUTION OF A       *
C     *            DIFFERENTIAL EQUATION BY ADAPTIVELY CHOOSING     *
C     *            THE STEP SIZE TO LIMIT THE LOCAL ERROR EST-      *
C     *            IMATE WITHIN A GIVEN TOLERANCE. A 4-TH ORDER     *
C     *            RUNGE-KUTTA-FEHLBERG METHOD IS USED              *
C     *  USAGE:                                                     *
C     *       CALL SEQUENCE: CALL ARUKU(X,Y,B,M,TOL)                *
C     *       EXTERNAL FUNCTIONS/SUBROUTINES:                       *
C     *                      SUBROUTINE RKF(XI,YI,H,YOUT,ZOUT)      *
C     *  PARAMETERS:                                                *
C     *       INPUT:                                                *
C     *        X(1)=INDEPENDENT VARIABLE INITIAL VALUE              *
C     *        Y(1)=DEPENDENT VARIABLE INITIAL VALUE                *
C     *           B=SOLUTION INTERVAL ENDPOINT (LAST X VALUE)       *
C     *           M=MAXIMUM NUMBER OF ITERATIONS                    *
C     *      OUTPUT:                                                *
C     *           X=M BY 1 ARRAY OF INDEPENDENT VARIABLE VALUES     *
C     *           Y=M BY 1 ARRAY OF DEPENDENT VARIABLE SOLUTION     *
C     *             VALUES                                          *
C     ***************************************************************
C
      DIMENSION X(M),Y(M)
C     *** INITIALIZATION ***
      H=.10E-02
      I=1
      N=0
C     *** COMPUTE SOLUTION ITERATIVELY ***
      DO WHILE(X(I).LE.B)
         N=N+1
         CALL RKF(X(I),Y(I),H,YOUT,ZOUT)
         ERR=ZOUT-YOUT
C     *** TEST IF THE NUMBER OF ITERATIONS EXCEEDED ***
         IF(N.GT.M) THEN
            WRITE(6,1)
    1       FORMAT(1X,'PROGRAM STOPPED TOO MANY ITERATIONS')
            STOP
         END IF
C     *** TEST STEP SIZE ***
         IF(ABS(ERR).LT.TOL) THEN
            I=I+1
            X(I)=X(I-1)+H
            H=3.0*H
            Y(I)=ZOUT
         ELSE
            H=H/10.0
         END IF
      END DO
      M=I
      H=B-X(I-1)
      X(M)=X(I-1)+H
      CALL RKF(X(I-1),Y(I-1),H,Y(M),ZOUT)
      RETURN
      END
```

We serve notice that the code ARUKU is intended only to illustrate the principles of automatic error control. It is inefficient and does not have the safeguards of a professional differential equation program package. More will be said about this matter after the following computational example.

EXAMPLE 7.17

By means of the calling program given in Table 7.24, the automatic step-size routine ARUKU is called on to solve the differential equation

$$y' = y, \qquad y(0) = 1 \tag{7.53}$$

over the interval [0, 1]. We chose this over our "usual" differential equation because in the present case it is easy to compute the exact local error $\hat{y}(x_{j+1}) - y_{j+1}$ and thereby see how well the RKF error estimator is doing. Specifically, the solution of (7.53) that passes through points (x_j, y_j) is

$$\hat{y}(x) = y_j \exp(x - x_j),$$

and if h is the current step size, then the exact local error is given by

$$y_j \exp(h) - \text{YOUT}.$$

TABLE 7.24 Calling Program for the Subroutine ARUKU

```
C     PROGRAM RKFMETH
C
C     ***************************************************************
C     THIS PROGRAM WILL SET UP AND SOLVE NUMERICALLY THE DIFFERENTIAL
C     EQUATION Y'=Y WITH THE INITIAL CONDITION Y(0.)=1.
C     THE SOLUTION IS OBTAINED USING THE AUTOMATIC STEPSIZE ROUTINE
C     USING 4-TH ORDER RUNGE-KUTTA FEHLBERG METHOD
C     CALLS:   ARUKU,RKF (BOTH MODIFIED FOR DOUBLE PRECISION)
C     OUTPUT:
C              X(I)=VALUE OF X FOR I=1,...  (MAX=50)
C              Y(I)=APPROXIMATED VALUE OF Y AT X(I)
C     ***************************************************************
C
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      DIMENSION X(50),Y(50)
C
C     *** FIRST, THE INITIAL CONDITION AND ENDPOINT ARE ESTABLISHED***
C     *** THE MAXIMUM NUMBER OF ITERATIONS IS SET TO 50            ***
C
      X(1)=0.D0
      Y(1)=1.D0
      B=1.D0
      MAX=50
      TOL=1.D-4
C
C     *** SUBROUTINE ARUKU WILL APPROXIMATE THE SOLUTION           ***
C     *** RETURNING AT MOST 50 VALUES IN ARRAYS X AND Y            ***
C
      CALL ARUKU(X,Y,B,MAX,TOL)
      WRITE(10,*)(X(I),Y(I),I=1,50)
      STOP
      END
C
C     *** FUNCTION F SPECIFIES THE DIFFERENTIAL EQUATION. IT IS    ***
C     *** CALLED BY SUBROUTINE RKF                                 ***
C
      FUNCTION F(X,Y)
      IMPLICIT DOUBLE PRECISION(A-H,O-Z)
      F=Y
      RETURN
      END
```