```
      PROGRAM DIFF_EQ

!   This is a sample program using 14th order Runge Kutta to solve ordinary differential
!   equations (initial value problems) to high precision.

!   Subroutine FM_RK14 uses a starting point A, stopping point B, and tolerance TOL.
!   FM_RK14 calls FM_RK14_STEP for the individual steps, adjusting the step size along the
!   way to try to keep the distance between the approximate solution vector S1 and the true
!   solution less than TOL.  TOL should be no smaller than 1.0e-75.

!   Subroutine FM_RK14_COEFFS is called by FM_RK14_STEP to initialize the many coefficients
!   used to define the 14th order Runge Kutta formula.  They are defined with 85-digit
!   accuracy, so accuracy up to about 75 digits can be achieved by these routines.

!   The speed of FM_RK14 drops quickly as the requested precision increases, since more
!   steps (with smaller stepsize) are needed to get from A to B, and also because higher
!   FM precision must be used.

!   For a typical 2014 computer, here are the times for the third-order equation in case 3:

!   TOL     FM precision     time (seconds)
!   1e-20   30                0.18
!   1e-30   40                1.13
!   1e-40   50                7.06
!   1e-50   60               43.91

!   Since RK14 has error O(h^14), the stepsize h needed for a given tol is of order tol^(1/14).
!   That gives a total number of steps proportional to tol^(-1/14) and means that decreasing
!   tol by a factor of 1e+10 will multiply the total number of steps required by about
!   1e-10^(-1/14) = 5.2.

!   The actual time ratios in the table above are 6.3, 6.2, 6.2, slightly more than 5.2,
!   because the time for each step increases as FM precision goes up.


      USE FMZM
      IMPLICIT NONE

!             Set MAXIMUM_ORDER here and in the subroutines to the highest order
!             differential equation to be solved.

      INTEGER, PARAMETER :: MAXIMUM_ORDER = 3
      INTEGER :: N_FUNCTION, N_ORDER
      TYPE (FM) :: A, B, ERR, S(MAXIMUM_ORDER), S1(MAXIMUM_ORDER), TOL
      EXTERNAL :: FM_RK14_F
      REAL :: T1, T2

!             Set the FM precision level to 40 significant digits for cases 1 through 3.

      CALL FM_SET(40)


!             We will use differential equations with known analytic solutions so we can check
!             the accuracy of the result.
```

```
!             1.  First-order equation.

!                 y' = -y + 2*sin(x),   y(0) = 0

!                 The right-hand-side function is defined as function number 1
!                 in subroutine FM_RK14_F (at the end of this file).

!                 Since this is a first-order equation, the "state" vector S is just y.

!                 Set tol = 1e-30 and find y(5).


      CALL CPU_TIME(T1)

      N_ORDER = 1
      N_FUNCTION = 1
      A = 0
      B = 5
      S(1) = 0
      TOL = TO_FM(' 1.0e-30 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 1.  y(5) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) ' '
      ERR = ABS( (SIN(B) - COS(B) + EXP(-B))  - S1(1) )
      WRITE (*,"(A,ES16.7)") '     Error in the computed solution = ',TO_DP(ERR)


      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '



!             2.  Second-order equation.

!                 y'' = -y' - exp(x)*y + sin(x) - exp(-x)*(sin(x) + cos(x)),
!                 y(0) = 0,  y'(0) = 1.

!                 The right-hand-side function is defined as function number 2
!                 in subroutine FM_RK14_F (at the end of this file).

!                 First reduce this equation to a system of first-order equations.

!                 Let u = y'.  Then u' = y''.  Now for s = ( y, u ) the vector
!                 differential equation is

!                 s' = ( y', u' ) = ( u, -u + exp(x)*y + sin(x) - exp(-x)*(sin(x) + cos(x)) )
!                 s(0) = ( y(0), u(0) ) = ( 0, 1 ).
```

```
!                Find y(2).


      CALL CPU_TIME(T1)

      N_ORDER = 2
      N_FUNCTION = 2
      A = 0
      B = 2
      S(1:2) = (/ 0, 1 /)
      TOL = TO_FM(' 1.0e-30 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 2.  y(2) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) "          y'(2) ="
      CALL FM_PRINT(S1(2))
      WRITE (*,*) ' '
      ERR = ABS( (SIN(B)*EXP(-B)) - S1(1) )
      WRITE (*,"(A,ES16.7)") '     Error in the computed y(2) solution = ',TO_DP(ERR)


      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '


!             3.  Third-order equation.

!                 y''' = -y'' -y' - y + ( ( -35x**3 + 2x**2 + 111x + 68 )*cos(6x) +
!                                       ( 210x**3 + 642x**2 + 618x + 186 )*sin(6x) ) / (1+x)**4
!                 y(0) = 1,  y'(0) = -1,  y''(0) = -34.

!                 The right-hand-side function is defined as function number 3
!                 in subroutine FM_RK14_F (at the end of this file).

!                 First reduce this equation to a system of first-order equations.

!                 let u = y' and v = y''.  Then v' = y'''.  Now for s = ( y, u, v ) the vector
!                 differential equation is

!                 s' = ( y', u', v' ) =
!                      ( u , v , -v - u - y +
!                                ( ( -35x**3 + 2x**2 + 111x + 68 )*cos(6x) +
!                                  ( 210x**3 + 642x**2 + 618x + 186 )*sin(6x) ) / (1+x)**4 )
!                 s(0) = ( y(0), u(0), v(0) ) = ( 1, -1, -34 ).

!                 Find y(2).


      CALL CPU_TIME(T1)
```

```
      N_ORDER = 3
      N_FUNCTION = 3
      A = 0
      B = 2
      S(1:3) = (/ 1, -1, -34 /)
      TOL = TO_FM(' 1.0e-30 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 3.  y(2) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) "          y'(2) ="
      CALL FM_PRINT(S1(2))
      WRITE (*,*) "          y''(2) ="
      CALL FM_PRINT(S1(3))
      WRITE (*,*) ' '
      ERR = ABS( (COS(6*B)/(B+1)) - S1(1) )
      WRITE (*,"(A,ES16.7)") '     Error in the computed y(2) solution = ',TO_DP(ERR)


      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '




!             4.  Solve case 3 again, this time asking for 20 digit accuracy.
!                 For this case we can lower the FM precision level to 30 digits.

      CALL FM_SET(30)

      CALL CPU_TIME(T1)

      N_ORDER = 3
      N_FUNCTION = 3
      A = 0
      B = 2
      S(1:3) = (/ 1, -1, -34 /)
      TOL = TO_FM(' 1.0e-20 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 4.  y(2) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) "          y'(2) ="
      CALL FM_PRINT(S1(2))
      WRITE (*,*) "          y''(2) ="
      CALL FM_PRINT(S1(3))
      WRITE (*,*) ' '
      ERR = ABS( (COS(6*B)/(B+1)) - S1(1) )
```

```
      WRITE (*,"(A,ES16.7)") '     Error in the computed y(2) solution = ',TO_DP(ERR)


      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '



!             5.  Same as case 4, but use TOL = 1.0e-40.
!                 The FM precision level should be set to at least 10 digits more than TOL.

      CALL FM_SET(50)

      CALL CPU_TIME(T1)

      N_ORDER = 3
      N_FUNCTION = 3
      A = 0
      B = 2
      S(1:3) = (/ 1, -1, -34 /)
      TOL = TO_FM(' 1.0e-40 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 5.  y(2) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) "          y'(2) ="
      CALL FM_PRINT(S1(2))
      WRITE (*,*) "          y''(2) ="
      CALL FM_PRINT(S1(3))
      WRITE (*,*) ' '
      ERR = ABS( (COS(6*B)/(B+1)) - S1(1) )
      WRITE (*,"(A,ES16.7)") '     Error in the computed y(2) solution = ',TO_DP(ERR)


      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '



!             6.  Same as case 4, but use TOL = 1.0e-50.

!                 The FM precision level should be set to at least 10 digits more than TOL.


      CALL FM_SET(60)

      CALL CPU_TIME(T1)
```

```
      N_ORDER = 3
      N_FUNCTION = 3
      A = 0
      B = 2
      S(1:3) = (/ 1, -1, -34 /)
      TOL = TO_FM(' 1.0e-50 ')

      CALL FM_RK14( A, B, N_ORDER, FM_RK14_F, N_FUNCTION, S, TOL, S1 )

      WRITE (*,*) ' '
      WRITE (*,*) ' Case 6.  y(2) ='
      CALL FM_PRINT(S1(1))
      WRITE (*,*) "          y'(2) ="
      CALL FM_PRINT(S1(2))
      WRITE (*,*) "          y''(2) ="
      CALL FM_PRINT(S1(3))
      WRITE (*,*) ' '
      ERR = ABS( (COS(6*B)/(B+1)) - S1(1) )
      WRITE (*,"(A,ES16.7)") '     Error in the computed y(2) solution = ',TO_DP(ERR)

      CALL CPU_TIME(T2)
      WRITE (*,*) ' '
      WRITE (*,"(5X,A,ES12.4,A,F8.2,A)") ' For tolerance = ',TO_DP(TOL),'   time = ',T2-T1,' sec.'
      WRITE (*,*) ' '
      WRITE (*,*) ' '

      STOP
      END PROGRAM DIFF_EQ


      SUBROUTINE FM_RK14_F(N_ORDER, N_FUNCTION, X, S, RHS)

!  Compute the right-hand-side function for the vector first-order differential equation
!  s' = f(x,s).

!  N_ORDER is the order of the differential equation.  After reducing the equation to
!          a first-order vector D.E., N_ORDER is the length of vectors S and RHS.

!  RHS is returned as the right-hand-side vector function of the differential equation,
!  with S as the input vector:  RHS = F(X,S).

!  N_FUNCTION is the function to be evaluated, for cases where a program may solve
!             several different differential equations.


      USE FMZM
      IMPLICIT NONE

      INTEGER, PARAMETER :: MAXIMUM_ORDER = 3
      INTEGER :: N_ORDER, N_FUNCTION
      TYPE (FM) :: X, S(MAXIMUM_ORDER), RHS(MAXIMUM_ORDER)
      TYPE (FM), SAVE :: T1, T2, T3

      CALL FM_ENTER_USER_ROUTINE
```

```
      IF (N_FUNCTION == 1) THEN

!           y' = -y + 2*sin(x)

          RHS(1) = -S(1) + 2*SIN(X)
      ELSE IF (N_FUNCTION == 2) THEN

!           y'' = -y' - exp(x)*y + sin(x) - exp(-x)*(sin(x) + cos(x))

          RHS(1) = S(2)

!           Note about code-tuning.
!           This is the straight-forward way of coding RHS(2) from the differential equation:

!         RHS(2) = -S(2) - EXP(X)*S(1) + SIN(X) - EXP(-X)*(SIN(X) + COS(X))

!           Using the code above, case 2 in the main program ran in 1.22 seconds.

!           We can speed this up by computing sin(x) once instead of twice for each function
!           evaluation.  Also, doing exp(-x) as 1/exp(x) can save an exponential.
!           More time can be saved by using subroutine FM_COS_SIN, which returns both
!           cos(x) and sin(x) in one call.  FM_COS_SIN computes one of the trig functions,
!           and then gets the other quickly using an identity.

!           Three local variables, T1, T2, T3, are used to save exp(x), cos(x), sin(x).
!           The code below then ran case 2 in 0.75 seconds.

          T1 = EXP(X)
          CALL FM_COS_SIN(X,T2,T3)
          RHS(2) = -S(2) - T1*S(1) + T3 - (T3 + T2) / T1
      ELSE IF (N_FUNCTION == 3) THEN

!           y''' = -y'' -y' - y + ( ( -35x**3 + 2x**2 + 111x + 68 )*cos(6x) +
!                                 ( 210x**3 + 642x**2 + 618x + 186 )*sin(6x) ) / (1+x)**4

          RHS(1) = S(2)
          RHS(2) = S(3)

!           More code-tuning.
!           Original code in case 3:  1.65 seconds.

!         RHS(3) = -S(3) - S(2) - S(1)  +                                          &
!                  ( ( -35*X**3 + 2*X**2 + 111*X + 68 )*COS(6*X) +                 &
!                    ( 210*X**3 + 642*X**2 + 618*X + 186 )*SIN(6*X) )  / (1+X)**4

!           Use FM_COS_SIN as in function 2 above for the trig functions:  1.38 seconds.

!         CALL FM_COS_SIN(6*X,T2,T3)
!         RHS(3) = -S(3) - S(2) - S(1)  +                                        &
!                  ( ( -35*X**3 + 2*X**2 + 111*X + 68 )*T2 +                     &
!                    ( 210*X**3 + 642*X**2 + 618*X + 186 )*T3 )  / (1+X)**4

!           Use Horner's rule for the polynomials:  1.30 seconds.

          CALL FM_COS_SIN(6*X,T2,T3)
          RHS(3) = -S(3) - S(2) - S(1)  +                                        &
```

```
            ( ((( -35*X +   2)*X + 111)*X +  68 )*T2 +                &
              ((( 210*X + 642)*X + 618)*X + 186 )*T3 )  / (1+X)**4

ELSE
    RHS = S(1)
ENDIF

CALL FM_EXIT_USER_ROUTINE
END SUBROUTINE FM_RK14_F
```