```fortran
      program test

!   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 2021 computer, here are the times for the third-order equation in case 3:

!   tol      FM precision     time (seconds)
!   1e-20    30                0.07
!   1e-30    40                0.46
!   1e-40    50                2.90
!   1e-50    60               19.00

!   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 6.6, 6.3, 6.6, 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).
```

```fortran
      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
```

```fortran
      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 test


      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.
!          (n_order is unused in this sample version)

!  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
      intent (in) :: n_order, n_function, x, s
      intent (inout) :: rhs

      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 0.48 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 0.27 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:  0.61 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:  0.50 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:  0.47 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

      end subroutine fm_rk14_f
```