```fortran
!   This is a sample program using the FMZM and FM_INTERVAL_ARITHMETIC modules for doing
!   interval arithmetic using the FM_INTERVAL derived type.

!   The output is saved in file SampleFMinterval.out.  A comparison file, SampleFMinterval.chk,
!   is provided showing the expected output from machines using 64-bit double precision and IEEE
!   arithmetic.  This would give about 16 significant digit accuracy for a stable calculation.
!   When run on other computers, all the multiple precision results should be the same, and the
!   results from the machine precision (d.p.) calculations will be different.
!   The program checks all the results and the last line of the output file should be
!   "All results were ok."

!  Sample 3 below uses an array-valued function of type FM_INTERVAL.
!  The function is defined here in a module with an explicit interface.

  MODULE EXP_SUM_MOD

   INTERFACE EXP_SUM
      MODULE PROCEDURE EXP_SUM3
   END INTERFACE

   CONTAINS

      FUNCTION EXP_SUM3(R_FM)

!  Sample function usage for type FM_INTERVAL.

!  The test function is exp(x) = 1 + x + x**2/2! + x**3/3! + ...
!  summed for the three values of x in array R_FM

!  Note that functions returning an FM_INTERVAL variable need a call to FM_ENTER_USER_FUNCTION
!  upon entry to the routine and one to FM_EXIT_USER_FUNCTION upon exit, where the argument in
!  each case is the function name (EXP_SUM3 here).

!  To keep from wasting memory, local variables like S should have the SAVE attribute.

      USE FMZM
      USE FM_INTERVAL_ARITHMETIC
      IMPLICIT NONE
      TYPE (FM) :: R_FM(3)
      TYPE (FM_INTERVAL) :: EXP_SUM3(3)
      TYPE (FM_INTERVAL), SAVE :: S, T, X
      INTEGER :: J, K

      CALL FM_ENTER_USER_FUNCTION(EXP_SUM3)

      DO J = 1, 3
         S = 1
         T = 1
         X = R_FM(J)
         DO K = 1, 1000
            T = T * X / K
            S = S + T
            IF (ABS(T) < TO_FM('1.0E-75')) EXIT
         ENDDO
         EXP_SUM3(J) = S
```

```
      ENDDO
      CALL FM_EXIT_USER_FUNCTION(EXP_SUM3)

      END FUNCTION EXP_SUM3

  END MODULE EXP_SUM_MOD



      PROGRAM SAMPLE_INTERVAL
      USE EXP_SUM_MOD
      USE FM_INTERVAL_ARITHMETIC
      IMPLICIT NONE

!             Declare the multiple precision variables.
!             (FM) for multiple precision reals
!             (FM_INTERVAL) for multiple precision real intervals

      TYPE (FM), SAVE :: DIGITS_LOST_FM, ERROR_FM, FACT_FM, R_FM(3), S_FM, S2_FM, T_FM, X_FM, X2_FM
      TYPE (FM_INTERVAL), SAVE :: FACT_FM_INTERVAL, S_FM_INTERVAL, T_FM_INTERVAL,  &
                                  X_FM_INTERVAL, X2_FM_INTERVAL, EXP_SUM_INTERVAL(3)
!      TYPE (FM_INTERVAL), EXTERNAL :: EXP_SUM

!             Declare the other variables (not multiple precision).

      CHARACTER(80)  :: ST1, STF
      INTEGER :: E(3), J, K, KOUT, NERROR
      DOUBLE PRECISION :: FACT, S, T, X, X2

!             Write output to the file SampleFMinterval.out.

      KOUT = 18
      OPEN (KOUT,FILE='SampleFMinterval.out')

      NERROR = 0

!  ----------------------------------------------------------------------------------------Sample 1

!             One of the common uses for multiple precision and also interval arithmetic is to
!             test the accuracy and stability of an algorithm.

!             Here is a sum that theoretically converges to the Bessel function
!             J(1,x) for x = 35.

!             1.  Try it using double precision.
!                 Printing the partial sums each 5 terms shows that this formula is unstable for
!                 x = 35, since some of the partial sums are more than 1.0e+14 times larger than
!                 the final sum.  This makes it seem that we have lost at least 14 significant
!                 digits to cancellation.
!                 For this example it is fairly clear from the double precision output that the
!                 final value of S is not accurate, but that might not be easy to see for a more
!                 complicated calculation.

      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' Sample 1.  Unstable summation.'
```

```
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' 1.  Do the sum in double precision.'
      WRITE (KOUT,*) ' '
      S = 0
      X = 35.0D0/2
      X2 = -(X**2)
      FACT = 1
      DO K = 0, 70
         T = X / ( (K+1) * FACT**2 )
         IF ( ABS(T) < EPSILON(S)*ABS(S) ) THEN
             WRITE (KOUT,"(A,I3,A,ES25.15)") '      K = ',K,'    S = ',S
             EXIT
         ENDIF
         S = S + T
         X = X * X2
         FACT = FACT * (K+1)
         IF (MOD(K,5) == 0) THEN
             WRITE (KOUT,"(A,I3,A,ES25.15)") '      K = ',K,'    S = ',S
         ENDIF
      ENDDO
      IF (ABS(S-4.399D-2) > 3.0D-3) THEN
          NERROR = NERROR + 1
          WRITE (KOUT,*) ' '
          WRITE (KOUT,*) ' Error in case 1 (or double precision accuracy is not 53 bits).'
          WRITE (KOUT,*) ' '
      ENDIF

!             2.  Try it using multiple precision, with 20, 30, 40, and 50 significant digits.
!                 To measure the error each time, compute it first with 100 digits.

!                 The error is measured in ulps (units in the last place), since the actual
!                 accuracy is slightly more than the number of digits requested.

!                 When FM uses the default large base (10^7 is typical for 64-bit double precision)
!                 this test will show about 18 (base 10) digits lost during the calculation.

!                 The reason for the "at least" in the descriptions below is that if the base is
!                 10^7, then the first word of the multiple precision number can have from 1 to 7
!                 base 10 digits.  So asking for 20 digit precision with CALL FM_SET(20) gives
!                 5 digits base 10^7, since we want a few guard digits past 20, and using 4 digits
!                 base 10^7 would guarantee only 1 + 3*7 = 22 decimal digits.  With 5 digits every
!                 intermediate value in the computation will have from 29 to 35 significant digits.

      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' 2.  Use FM with increasing precision.'
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) '     Setting precision to J digits via CALL FM_SET(J) will actually set the'
      WRITE (KOUT,*) '     equivalent number of decimal significant digits slightly higher than J.'
      WRITE (KOUT,*) '     For example, if the base used internally in FM is 10**7, then asking for'
      WRITE (KOUT,*) '     20 digits with CALL FM_SET(20) gives at least 29 significant digits.'
      WRITE (KOUT,*) '     CALL FM_SET(30) gives at least 36 significant digits.'
      WRITE (KOUT,*) '     CALL FM_SET(40) gives at least 50 significant digits.'
      WRITE (KOUT,*) '     CALL FM_SET(50) gives at least 57 significant digits.'
      WRITE (KOUT,*) ' '
      CALL FM_SET(100)
```

```
      S2_FM = 0
      X_FM = 35.0D0/2
      X2_FM = -(X_FM**2)
      FACT_FM = 1
      DO K = 0, 700
         T_FM = X_FM / ( (K+1) * FACT_FM**2 )
         IF ( ABS(T_FM) < EPSILON(S2_FM)*ABS(S2_FM) ) EXIT
         S2_FM = S2_FM + T_FM
         X_FM = X_FM * X2_FM
         FACT_FM = FACT_FM * (K+1)
      ENDDO

      DO J = 20, 50, 10
         CALL FM_SET(J)
         S_FM = 0
         X_FM = 35.0D0/2
         X2_FM = -(X_FM**2)
         FACT_FM = 1
         DO K = 0, 700
            T_FM = X_FM / ( (K+1) * FACT_FM**2 )
            S_FM = S_FM + T_FM
            IF ( ABS(T_FM) < EPSILON(S_FM)*ABS(S_FM) ) THEN
                WRITE (STF,*) ' F',J+3,'.',J
                CALL FM_FORM(TRIM(STF),S_FM,ST1)
                WRITE (KOUT,"(5X,I3,A,I3,A,A)") J,' digits,',K,' terms gave   S_FM = ',TRIM(ST1)
                CALL FM_ULP(S_FM,T_FM)

!                  Since S2_FM was computed at a different precision than S_FM, we should
!                  round it to the current precision.  If we knew the two values of the FM
!                  internal variable NDIG that were used in computing S2_FM and S_FM, the
!                  standard FM rounding routine FM_EQU could be used.  Here we used FM_SET
!                  to ask for slightly more than 100 and J decimal digits for S2_FM and S_FM,
!                  so the routine ROUND_FM in this program gets those values of NDIG and does
!                  the rounding.

                CALL ROUND_FM(S2_FM,X2_FM,100,J)
                ERROR_FM = ABS( (S_FM-X2_FM)/T_FM )
                DIGITS_LOST_FM = NINT(LOG10(ERROR_FM))
                WRITE (KOUT,"(A,I3,A)") '               This calculation lost about ',  &
                                         TO_INT(DIGITS_LOST_FM),' digits.'
                EXIT
            ENDIF
            X_FM = X_FM * X2_FM
            FACT_FM = FACT_FM * (K+1)
         ENDDO
      ENDDO
      IF (ABS(S_FM-TO_FM('.04399094217962563996969897065974247192700503984511')) > 1.0D-35) THEN
          NERROR = NERROR + 1
          WRITE (KOUT,*) ' '
          WRITE (KOUT,*) ' Error in case 2.'
          WRITE (KOUT,*) ' '
      ENDIF

!             3.  Sometimes we want to measure the errors using base 2 arithmetic in FM,
!                 to more accurately reflect what is happening in the d.p. calculation.
!                 Set FM to use base 2, and do the calculation with 53 bits of precision
```

```
!                 (64-bit d.p.), then 73, 93, 113 bits.

!                 We want exact control over the base and precision, so use FM_SETVAR
!                 instead of FM_SET.

!                 This shows a loss of 14 or 15 (base 10) digits when using base 2.

!                 This is typical of the comparison between using FM with the default large base
!                 and base 2.  Normalization error is larger with a large base, but the 30 s.d.
!                 calculation in case 2 gets about the same accuracy as the 113-bit calculation
!                 in case 3, and using base 2 is much slower.

      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' 3.  Use FM with increasing precision in base 2.'
      WRITE (KOUT,*) ' '
      CALL FM_SETVAR(" MBASE = 2 ")
      CALL FM_SETVAR(" NDIG = 150 ")
      S2_FM = 0
      X_FM = 35.0D0/2
      X2_FM = -(X_FM**2)
      FACT_FM = 1
      DO K = 0, 700
         T_FM = X_FM / ( (K+1) * FACT_FM**2 )
         IF ( ABS(T_FM) < EPSILON(S2_FM)*ABS(S2_FM) ) EXIT
         S2_FM = S2_FM + T_FM
         X_FM = X_FM * X2_FM
         FACT_FM = FACT_FM * (K+1)
      ENDDO

      DO J = 53, 113, 20
         WRITE (STF,*) ' NDIG =',J
         CALL FM_SETVAR(TRIM(STF))
         S_FM = 0
         X_FM = 35.0D0/2
         X2_FM = -(X_FM**2)
         FACT_FM = 1
         DO K = 0, 700
            T_FM = X_FM / ( (K+1) * FACT_FM**2 )
            S_FM = S_FM + T_FM
            IF ( ABS(T_FM) < EPSILON(S_FM)*ABS(S_FM) ) THEN
                WRITE (STF,*) ' F',NINT(J*0.301)+3,'.',NINT(J*0.301)+1
                CALL FM_FORM(TRIM(STF),S_FM,ST1)
                WRITE (KOUT,"(A,I3,A,I3,A,A)") '      Using ',J,' bits,',K,  &
                                               ' terms gave   S_FM = ',TRIM(ST1)
                CALL FM_ULP(S_FM,T_FM)

!                  Since S2_FM was computed at a different precision than S_FM, we should
!                  round it to the current precision.  For this case we have explicitly set
!                  NDIG instead of using FM_SET as in case 2 above, so we use FM_EQU to do
!                  the rounding.

                CALL FM_EQU(S2_FM,X2_FM,150,J)
                ERROR_FM = ABS( (S_FM-X2_FM)/T_FM )
                DIGITS_LOST_FM = NINT(LOG10(ERROR_FM))
                WRITE (KOUT,"(A,I3,A)") '                This calculation lost about ',  &
                                        TO_INT(DIGITS_LOST_FM),' decimal digits.'
```

```
            EXIT
         ENDIF
         X_FM = X_FM * X2_FM
         FACT_FM = FACT_FM * (K+1)
      ENDDO
   ENDDO
   IF (ABS(S_FM-TO_FM('.04399094217962563996863023518761 96')) > 1.0D-20) THEN
       NERROR = NERROR + 1
       WRITE (KOUT,*) ' '
       WRITE (KOUT,*) ' Error in case 3.'
       WRITE (KOUT,*) ' '
   ENDIF

!            4.  A second way to check an algorithm's stability is to re-do the calculation
!                at the same precision but with different rounding modes.

!                Use base 2 with 53 bits and round down, then round symmetrically, then round up.

!                The results show the three values have no digits of agreement, confirming the
!                loss of about 15 or 16 s.d. in the ones rounded down and up.

   WRITE (KOUT,*) ' '
   WRITE (KOUT,*) ' 4.  Use FM with different rounding modes in base 2.'
   WRITE (KOUT,*) ' '
   CALL FM_SETVAR(" MBASE = 2 ")
   CALL FM_SETVAR(" NDIG = 53 ")

   DO J = 1, 3
      IF (J == 1) THEN
          CALL FM_SETVAR(" KROUND = -1 ")
      ELSE IF (J == 2) THEN
          CALL FM_SETVAR(" KROUND = 1 ")
      ELSE IF (J == 3) THEN
          CALL FM_SETVAR(" KROUND = 2 ")
      ENDIF
      S_FM = 0
      X_FM = 35.0D0/2
      X2_FM = -(X_FM**2)
      FACT_FM = 1
      DO K = 0, 700
         T_FM = X_FM / ( (K+1) * FACT_FM**2 )
         S_FM = S_FM + T_FM
         IF ( ABS(T_FM) < EPSILON(S_FM)*ABS(S_FM) ) THEN
             IF (J == 1) THEN
                 STF = 'rounding left         ,'
             ELSE IF (J == 2) THEN
                 STF = 'rounding symmetrically,'
             ELSE IF (J == 3) THEN
                 STF = 'rounding right        ,'
             ENDIF
             CALL FM_FORM(' F20.17 ',S_FM,ST1)
             WRITE (KOUT,"(A,I4,A,A,I3,A,A)") '      Using',53,' bits, ',TRIM(STF),K,  &
                                              ' terms gave   S_FM = ',TRIM(ST1)
             EXIT
         ENDIF
         X_FM = X_FM * X2_FM
```

```
            FACT_FM = FACT_FM * (K+1)
         ENDDO
         R_FM(J) = S_FM
      ENDDO
      ERROR_FM = MAX( ABS((R_FM(1)-R_FM(2))/R_FM(1)) , ABS((R_FM(1)-R_FM(3))/R_FM(1)) ,  &
                      ABS((R_FM(2)-R_FM(3))/R_FM(2)) )
      J = -NINT(LOG10(ERROR_FM))
      WRITE (KOUT,"(A,I3,A)") '      These agree to about',J,' decimal digits.'
      IF (ABS(S_FM-TO_FM('.0519420065663800')) > 1.0D-4) THEN
          NERROR = NERROR + 1
          WRITE (KOUT,*) ' '
          WRITE (KOUT,*) ' Error in case 4.'
          WRITE (KOUT,*) ' '
      ENDIF

!                 Use base 2 with 113 bits and round down, then round symmetrically, then round up.

!                 Now the three values agree to about 18 decimal digits, which is again consistent
!                 with a loss of about 15 or 16 s.d. in the ones rounded down and up.

      WRITE (KOUT,*) ' '
      CALL FM_SETVAR(" MBASE = 2 ")
      CALL FM_SETVAR(" NDIG = 113 ")

      DO J = 1, 3
         IF (J == 1) THEN
             CALL FM_SETVAR(" KROUND = -1 ")
         ELSE IF (J == 2) THEN
             CALL FM_SETVAR(" KROUND = 1 ")
         ELSE IF (J == 3) THEN
             CALL FM_SETVAR(" KROUND = 2 ")
         ENDIF
         S_FM = 0
         X_FM = 35.0D0/2
         X2_FM = -(X_FM**2)
         FACT_FM = 1
         DO K = 0, 700
            T_FM = X_FM / ( (K+1) * FACT_FM**2 )
            S_FM = S_FM + T_FM
            IF ( ABS(T_FM) < EPSILON(S_FM)*ABS(S_FM) ) THEN
                IF (J == 1) THEN
                    STF = 'rounding left         ,'
                ELSE IF (J == 2) THEN
                    STF = 'rounding symmetrically,'
                ELSE IF (J == 3) THEN
                    STF = 'rounding right        ,'
                ENDIF
                CALL FM_FORM(' F20.17 ',S_FM,ST1)
                WRITE (KOUT,"(A,I4,A,A,I3,A,A)") '      Using',113,' bits, ',TRIM(STF),K,  &
                                                  ' terms gave   S_FM = ',TRIM(ST1)
                EXIT
            ENDIF
            X_FM = X_FM * X2_FM
            FACT_FM = FACT_FM * (K+1)
         ENDDO
         R_FM(J) = S_FM
```

```
      ENDDO
      ERROR_FM = MAX( ABS((R_FM(1)-R_FM(2))/R_FM(1)) , ABS((R_FM(1)-R_FM(3))/R_FM(1)) ,  &
                      ABS((R_FM(2)-R_FM(3))/R_FM(2)) )
      J = -NINT(LOG10(ERROR_FM))
      WRITE (KOUT,"(A,I3,A)") '       These agree to about',J,' decimal digits.'
      IF (ABS(S_FM-TO_FM('.0439909421796257')) > 1.0D-4) THEN
          NERROR = NERROR + 1
          WRITE (KOUT,*) ' '
          WRITE (KOUT,*) ' Error in case 4.'
          WRITE (KOUT,*) ' '
      ENDIF


!             5.  A third way to check an algorithm's stability is to re-do the calculation
!                 at the same precision but using interval arithmetic.

!                 Use base 2 with 53 bits.

!                 The result shows the two endpoints of the interval having opposite signs
!                 indicating a worst-case loss of all 16 s.d. during the calculation.

      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' 5.  Use FM with interval arithmetic in base 2.'
      WRITE (KOUT,*) ' '
      CALL FM_SETVAR(" MBASE = 2 ")
      CALL FM_SETVAR(" NDIG = 53 ")

      S_FM_INTERVAL = 0
      X_FM_INTERVAL = 35.0D0/2
      X2_FM_INTERVAL = -(X_FM_INTERVAL**2)
      FACT_FM_INTERVAL = 1
      DO K = 0, 700
         T_FM_INTERVAL = X_FM_INTERVAL / ( (K+1) * FACT_FM_INTERVAL**2 )
         S_FM_INTERVAL = S_FM_INTERVAL + T_FM_INTERVAL
         IF ( ABS(T_FM_INTERVAL) < EPSILON(S_FM_INTERVAL)*ABS(S_FM_INTERVAL) ) THEN
             CALL FM_INTERVAL_FORM(' F20.16 ',S_FM_INTERVAL,ST1)
             WRITE (KOUT,"(A,I4,A,I3,A,A)") '       Using',53,' bits, ',K,  &
                                            ' terms gave  S_FM_INTERVAL = ',TRIM(ST1)
             EXIT
         ENDIF
         X_FM_INTERVAL = X_FM_INTERVAL * X2_FM_INTERVAL
         FACT_FM_INTERVAL = FACT_FM_INTERVAL * (K+1)
      ENDDO
      ERROR_FM = ABS((RIGHT_ENDPOINT(S_FM_INTERVAL)-LEFT_ENDPOINT(S_FM_INTERVAL)) /  &
                      RIGHT_ENDPOINT(S_FM_INTERVAL))
      J = -NINT(LOG10(ERROR_FM))
      WRITE (KOUT,"(A,I3,A)") '       The two endpoints agree to about',J,' decimal digits.'

!                 Use base 2 with 113 bits.

!                 The result shows the two endpoints of the interval agree to about 18 s.d.
!                 indicating a worst-case loss of about 16 s.d. during the calculation.

      WRITE (KOUT,*) ' '
      CALL FM_SETVAR(" MBASE = 2 ")
      CALL FM_SETVAR(" NDIG = 113 ")
```

```
      S_FM_INTERVAL = 0
      X_FM_INTERVAL = 35.0D0/2
      X2_FM_INTERVAL = -(X_FM_INTERVAL**2)
      FACT_FM_INTERVAL = 1
      DO K = 0, 700
         T_FM_INTERVAL = X_FM_INTERVAL / ( (K+1) * FACT_FM_INTERVAL**2 )
         S_FM_INTERVAL = S_FM_INTERVAL + T_FM_INTERVAL
         IF ( ABS(T_FM_INTERVAL) < EPSILON(S_FM_INTERVAL)*ABS(S_FM_INTERVAL) ) THEN
             CALL FM_INTERVAL_FORM(' F20.16 ',S_FM_INTERVAL,ST1)
             WRITE (KOUT,"(A,I4,A,I3,A,A)") '      Using',113,' bits, ',K,  &
                                            ' terms gave  S_FM_INTERVAL = ',TRIM(ST1)
             EXIT
         ENDIF
         X_FM_INTERVAL = X_FM_INTERVAL * X2_FM_INTERVAL
         FACT_FM_INTERVAL = FACT_FM_INTERVAL * (K+1)
      ENDDO
      ERROR_FM = ABS((RIGHT_ENDPOINT(S_FM_INTERVAL)-LEFT_ENDPOINT(S_FM_INTERVAL)) /  &
                      RIGHT_ENDPOINT(S_FM_INTERVAL))
      J = -NINT(LOG10(ERROR_FM))
      WRITE (KOUT,"(A,I3,A)") '      The two endpoints agree to about',J,' decimal digits.'
      IF (ABS(S_FM_INTERVAL-TO_FM('.0439909421796257')) > 1.0D-4) THEN
          NERROR = NERROR + 1
          WRITE (KOUT,*) ' '
          WRITE (KOUT,*) ' Error in case 5.'
          WRITE (KOUT,*) ' '
      ENDIF


      IF (NERROR == 0) THEN
          WRITE (KOUT,"(//A)") ' Summary:'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' For this calculation all four methods for measuring the degree of'
          WRITE (KOUT,"(A)") ' instability worked well.  When done with double precision carrying'
          WRITE (KOUT,"(A)") ' 16 significant digits and using the default symmetric rounding,'
          WRITE (KOUT,"(A)") ' only 1 digit remained correct at the end of the sum.'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' Interval arithmetic is probably the strongest of these checks, and'
          WRITE (KOUT,"(A)") ' using FM arithmetic with 30 digits and a large base (method 2) is'
          WRITE (KOUT,"(A)") ' the fastest of these methods for getting the sum correct to full'
          WRITE (KOUT,"(A)") ' double precision accuracy.'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' Comparing the last value of S in method 1 with the first value of'
          WRITE (KOUT,"(A)") ' S_FM in method 3 should show whether this compiler carries extra'
          WRITE (KOUT,"(A)") ' digits while evaluating expressions like X / ( (K+1) * FACT**2 ).'
          WRITE (KOUT,"(A)") ' If the two values are the same, no extra digits are carried in d.p.'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' '
      ENDIF

!  ------------------------------------------------------------------------------------Sample 2

!             Here is a recurrence that is seriously unstable.

!             It is not a realistic problem that would come up in a practical application, but
!             is designed as a counter-example to show that just carrying much higher precision
```

```
!              cannot be proved to always cure numerical instability.
!              Reference:  William Kahan -- (2006)
!                 "How Futile are Mindless Assessments of Roundoff in Floating-Point Computation?"
!                  http://www.cs.berkeley.edu/~wkahan/Mindless.pdf

!              After 60 steps the result without any rounding errors would be very close to 5,
!              but rounding errors cause the result to converge to 100 instead.

      CALL FM_SET(30)
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' Sample 2.  Unstable recurrence.'
      WRITE (KOUT,*) ' '
      WRITE (KOUT,"(A)") '      1.  Use non-interval FM arithmetic with 30 digits.'
      WRITE (KOUT,*) ' '
      X_FM = 4
      X2_FM = TO_FM('4.25')
      DO J = 1, 60
         T_FM = 108 - ( 815 - 1500/X_FM ) / X2_FM
         X_FM = X2_FM
         X2_FM = T_FM
         IF (MOD(J,5) == 0) THEN
             CALL FM_FORM(' F20.14 ',X2_FM,ST1)
             WRITE (KOUT,"(A,I2,A,A)") '          After ',J,  &
                                   ' terms with 30 digit accuracy, the result is',TRIM(ST1)
         ENDIF
      ENDDO
      WRITE (KOUT,*) ' '

      WRITE (KOUT,"(A)") '      2.  Use interval arithmetic with 30 digit accuracy.'
      WRITE (KOUT,*) ' '
      X_FM_INTERVAL = 4
      X2_FM_INTERVAL = TO_FM_INTERVAL('4.25')
      DO J = 1, 60
         T_FM_INTERVAL = 108 - ( 815 - 1500/X_FM_INTERVAL ) / X2_FM_INTERVAL
         X_FM_INTERVAL = X2_FM_INTERVAL
         X2_FM_INTERVAL = T_FM_INTERVAL
         CALL FM_INTERVAL_FORM(' F20.16 ',X2_FM_INTERVAL,ST1)
         WRITE (KOUT,"(A,I3,A,A)") '          After',J,' terms the result is',TRIM(ST1)
         IF (LEFT_ENDPOINT(X_FM_INTERVAL) <= 0 .AND. RIGHT_ENDPOINT(X_FM_INTERVAL) >= 0) EXIT
         IF (J > 10 .AND. J < 25) THEN
             IF (ABS(LEFT_ENDPOINT(X2_FM_INTERVAL)-5) > 0.01 .OR.  &
                 ABS(RIGHT_ENDPOINT(X2_FM_INTERVAL)-5) > 0.01) THEN
                 NERROR = NERROR + 1
                 WRITE (KOUT,*) ' '
                 WRITE (KOUT,*) ' Error in sample 2, case 2.'
                 WRITE (KOUT,*) ' '
                 EXIT
             ENDIF
         ENDIF
      ENDDO

      IF (NERROR == 0) THEN
          WRITE (KOUT,"(//A)") ' Summary:'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' The general solution of this recurrence has a term that causes'
```

```
          WRITE (KOUT,"(A)") ' the values to converge to 100, but for these initial conditions'
          WRITE (KOUT,"(A)") ' 4, 4.25, the specific solution has a coefficient of zero on that'
          WRITE (KOUT,"(A)") ' term, so this sequence converges to 5 mathematically.'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' When rounding errors occur in later x-values, they introduce a'
          WRITE (KOUT,"(A)") ' very small but nonzero amount of the term that causes convergence'
          WRITE (KOUT,"(A)") ' to 100, and that term grows rapidly and soon swamps the rest of'
          WRITE (KOUT,"(A)") ' the solution.'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' Interval arithmetic tracks the growing uncertainty in the x-values,'
          WRITE (KOUT,"(A)") ' and when an interval gets big enough to include zero, dividing by'
          WRITE (KOUT,"(A)") ' that interval is undefined and the result is'
          WRITE (KOUT,"(A)") ' [ -overflow , +overflow ].'
          WRITE (KOUT,"(A)") ' '
          WRITE (KOUT,"(A)") ' Interval arithmetic works better than a sequence of increasing'
          WRITE (KOUT,"(A)") ' precision FM results here, since comparing FM results at 30, 40, 50'
          WRITE (KOUT,"(A)") ' digits gives 100 each time.'
          WRITE (KOUT,"(A)") ' '
      ENDIF

!  -----------------------------------------------------------------------------------------Sample 3

!             Sum an unstable series.
!             exp(x) = 1 + x + x**2/2! + x**3/3! + ...
!             converges mathematically for all x, but is unstable for negative x.

      S_FM_INTERVAL = 1
      T_FM_INTERVAL = 1
      R_FM = (/ -25, -30, -35 /)
      EXP_SUM_INTERVAL = EXP_SUM3(R_FM)
      E = (/ -12, -9, -3 /)
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' '
      WRITE (KOUT,*) ' Sample 3.  Unstable sum.'
      DO J = 1, 3
         CALL FM_INTERVAL_FORM(' ES25.16 ',EXP_SUM_INTERVAL(J),ST1)
         CALL FM_FORM(' ES25.16 ',EXP(R_FM(J)),STF)
         WRITE (KOUT,"(/A,F6.2,A,A)") '      For x = ', TO_DP(R_FM(J)), ' The sum gave ', ST1
         WRITE (KOUT,"(20X,A,A)") '      correct = ', STF

!             Check the results.

         X_FM  = LEFT_ENDPOINT(EXP_SUM_INTERVAL(J))
         X2_FM = RIGHT_ENDPOINT(EXP_SUM_INTERVAL(J))
         S_FM = EXP(R_FM(J))
         T_FM = ABS( (X2_FM - X_FM ) / S_FM )
         IF (.NOT.(S_FM > X_FM) .OR. .NOT.(S_FM < X2_FM) .OR. .NOT.(T_FM < TO_FM(10)**E(J))) THEN
             NERROR = NERROR + 1
             EXIT
         ENDIF
      ENDDO

      IF (NERROR == 0) THEN
          WRITE (KOUT,"(//A)") ' Summary:'
          WRITE (KOUT,"(A)") ' '
```

```
        WRITE (KOUT,"(A)") ' As the input x becomes more negative, the instability increases.'
        WRITE (KOUT,"(A)") ' This is shown as the left and right endpoints of the interval'
        WRITE (KOUT,"(A)") ' result agree to fewer digits.'
        WRITE (KOUT,"(A)") ' '
        WRITE (KOUT,"(A)") ' The mathematically correct value of the sum lies within the'
        WRITE (KOUT,"(A)") ' interval in each case.'
        WRITE (KOUT,"(A)") ' '
    ENDIF



    IF (NERROR == 0) THEN
        WRITE (*   ,"(//A,A/)") ' All results were ok.  (The output is in file ',  &
                            'SampleFMinterval.out)'
        WRITE (KOUT,"(//A/)") ' All results were ok.'
    ELSE
        WRITE (*   ,"(//I3,A,A/)") NERROR,' error(s) found.  (The output is in file ',  &
                                    'SampleFMinterval.out)'
        WRITE (KOUT,"(//I3,A/)") NERROR,' error(s) found.'
    ENDIF

    STOP
    END PROGRAM SAMPLE_INTERVAL

    SUBROUTINE ROUND_FM(A,B,J1,J2)
    USE FMVALS
    USE FMZM
    IMPLICIT NONE

!  A was computed with precision defined by CALL FM_SET(J1).
!  B will be returned with the value of A rounded to precision defined by CALL FM_SET(J2).
!  Do not use B in the calling program at any precision higher than J2.

    TYPE (FM) :: A, B
    INTEGER :: J1, J2, NDIG1, NDIG2, NSAVE

    NSAVE = NDIG
    CALL FM_SET(J1)
    NDIG1 = NDIG
    CALL FM_SET(J2)
    NDIG2 = NDIG
    CALL FM_EQU(A,B,NDIG1,NDIG2)
    NDIG = NSAVE

    END SUBROUTINE ROUND_FM
```