

```
PROGRAM TEST
USE FMZM
USE FM_INTERVAL_ARITHMETIC
```

```
! This program contains the code for the FM interval arithmetic examples that are discussed in
! the paper "A Multiple-Precision Interval Arithmetic Package".
```

```
IMPLICIT NONE
```

```
INTEGER :: I, J, J_MAX, J_MIN, K, N, NF, K_FUNCTION, N_ORDER, N_STEPS, KPRT, KW, K_GRAPH
TYPE (FM_INTERVAL) :: A, B, RESULT, S, TERM, PI, C, X, Y, V(5), DET
TYPE (FM_INTERVAL), ALLOCATABLE :: MATRIX(:,,:), RHS(:), SOLN(:)
INTEGER, ALLOCATABLE :: KSWAP(:)
TYPE (FM) :: A_FM, B_FM, S_FM, PI_FM, C_FM, X_FM, Y_FM
TYPE (FM) :: ERROR_FM, WIDTH_FM, LEAST_WIDTH_FM
DOUBLE PRECISION :: RAND
```

```
! Set the FM precision to 50 digits.
```

```
CALL FM_SET(50)
```

```
! Write output to file IntervalExamplesFM.out.
! The FM_SETVAR call directs all output from within the FM package to unit 22 also.
```

```
OPEN(22,FILE='IntervalExamplesFM.out')
CALL FM_SETVAR(' KW = 22 ')
```

```
! Unit 31 will be used to write several files that give the interval width
! for each step of the different examples.
```

```
K_GRAPH = 31
```

```
! Example 1. Ignoring variable correlation in formulas causes interval arithmetic
! to get too wide a resulting interval.
```

```
WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) " Example 1. Compute  $f(x) = x^2 - x + 3$  using different formulas."
WRITE (22,*) ' '
```

```
X = TO_FM_INTERVAL( " -0.5 ", " 1.0 " )
```

```
WRITE (22,*) ' '
WRITE (22,*) ' X is the interval [ -0.5 , 1.0 ]'
WRITE (22,*) ' '
```

```
X_FM = 0.5
A_FM = X_FM**2 - X_FM + 3
X_FM = LEFT_ENDPOINT(X)
B_FM = X_FM**2 - X_FM + 3
LEAST_WIDTH_FM = B_FM - A_FM
```

```
Y = X**2
```

```
C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
```

```
WRITE (22,"(A,F6.3,A,F6.3,A)") ' X**2 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]'
```

Y = X\*X

C\_FM = (RIGHT\_ENDPOINT(Y) - LEFT\_ENDPOINT(Y)) / LEAST\_WIDTH\_FM

```
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A)") ' X*X gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]'
```

```
WRITE (22,*) ' '
WRITE (22,*) ' '
```

Y = X\*\*2 - X + 3

C\_FM = (RIGHT\_ENDPOINT(Y) - LEFT\_ENDPOINT(Y)) / LEAST\_WIDTH\_FM

```
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X**2 - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)
```

Y = X\*X - X + 3

C\_FM = (RIGHT\_ENDPOINT(Y) - LEFT\_ENDPOINT(Y)) / LEAST\_WIDTH\_FM

```
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*X - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)
```

Y = X\*(X - 1) + 3

C\_FM = (RIGHT\_ENDPOINT(Y) - LEFT\_ENDPOINT(Y)) / LEAST\_WIDTH\_FM

```
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*(X - 1) + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)
```

Y = (X - 0.5)\*\*2 + 2.75

C\_FM = (RIGHT\_ENDPOINT(Y) - LEFT\_ENDPOINT(Y)) / LEAST\_WIDTH\_FM

```
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' (X - 0.5)**2 + 2.75 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)
```

```
WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) ' '
```

! Results are not as bad when the interval doesn't include zero.

X = TO\_FM\_INTERVAL( " 0.1 ", " 1.0 " )

```
WRITE (22,*) ' '
WRITE (22,*) ' X is the interval [ 0.1 , 1.0 ]'
WRITE (22,*) ' '
```

```

X_FM = 0.5
A_FM = X_FM**2 - X_FM + 3
X_FM = RIGHT_ENDPOINT(X)
B_FM = X_FM**2 - X_FM + 3
LEAST_WIDTH_FM = B_FM - A_FM

```

```
Y = X**2 - X + 3
```

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X**2 - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

```
Y = X*X - X + 3
```

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*X - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

```
Y = X*(X - 1) + 3
```

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*(X - 1) + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

```
Y = (X - 0.5)**2 + 2.75
```

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' (X - 0.5)**2 + 2.75 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

```

WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) ' '

```

! Results are better when the interval doesn't include a local minimum.

```
X = TO_FM_INTERVAL( " 0.9 ", " 1.0 " )
```

```

WRITE (22,*) ' '
WRITE (22,*) ' X is the interval [ 0.9 , 1.0 ]'
WRITE (22,*) ' '

```

```

X_FM = LEFT_ENDPOINT(X)
A_FM = X_FM**2 - X_FM + 3
X_FM = RIGHT_ENDPOINT(X)
B_FM = X_FM**2 - X_FM + 3
LEAST_WIDTH_FM = B_FM - A_FM

```

```
Y = X**2 - X + 3
```

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X**2 - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = X^2 - X + 3$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*X - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = X*(X - 1) + 3$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*(X - 1) + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = (X - 0.5)**2 + 2.75$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' (X - 0.5)**2 + 2.75 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

```

WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) ' '

```

! Look at an even smaller interval

```

X = TO_FM_INTERVAL( " 0.99 ", " 1.0 " )

```

```

WRITE (22,*) ' '
WRITE (22,*) ' X is the interval [ 0.99 , 1.0 ]'
WRITE (22,*) ' '

```

```

X_FM = LEFT_ENDPOINT(X)
A_FM = X_FM**2 - X_FM + 3
X_FM = RIGHT_ENDPOINT(X)
B_FM = X_FM**2 - X_FM + 3
LEAST_WIDTH_FM = B_FM - A_FM

```

$$Y = X^2 - X + 3$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X**2 - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = X^2 - X + 3$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*X - X + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = X*(X - 1) + 3$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' X*(X - 1) + 3 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)

```

$$Y = (X - 0.5)**2 + 2.75$$

```

C_FM = (RIGHT_ENDPOINT(Y) - LEFT_ENDPOINT(Y)) / LEAST_WIDTH_FM
WRITE (22,*) ' '
WRITE (22,"(A,F6.3,A,F6.3,A,F6.3)") ' (X - 0.5)**2 + 2.75 gives [ ', &
      TO_DP(LEFT_ENDPOINT(Y)), ' , ', TO_DP(RIGHT_ENDPOINT(Y)), &
      ' ]. Magnification = ', TO_DP(C_FM)
WRITE (22,*) ' '

```

! Example 2. Sum  $1 / n^7$  from 1 to 100,000.

```

OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Sum')

```

```

WRITE (K_GRAPH,"(//A//)") 'Example 2. Sum  $1 / n^7$  from 1 to 100,000.'
```

```

N = 10**5

```

```

S = 0

```

```

DO J = 1, N

```

```

    TERM = J

```

```

    S = S + 1 / TERM**7

```

```

    IF (MOD(J,N/1000) == 1) THEN

```

```

        WIDTH_FM = MAX( RIGHT_ENDPOINT(S)-LEFT_ENDPOINT(S) , EPSILON(LEFT_ENDPOINT(S)) )

```

```

        WRITE (K_GRAPH,"(A,I7,A,F8.3,A)") '{', J, ', ', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '},'
```

```

    ENDIF

```

```

ENDDO

```

```

WRITE (22,*) ' '

```

```

WRITE (22,*) " Example 2. Sum  $1 / n^7$  from 1 to 100,000 ="
```

```

CALL FMPRINT_INTERVAL(S)

```

```

WRITE (22,*) ' '

```

```

ERROR_FM = ABS( (RIGHT_ENDPOINT(S)-LEFT_ENDPOINT(S)) / RIGHT_ENDPOINT(S) )

```

```

J = -NINT(LOG10(ERROR_FM))

```

```

WRITE (22,"(A,I3,A)") ' The two endpoints agree to about',J,' decimal digits.'
```

```

WRITE (22,*) ' '

```

```

CLOSE(K_GRAPH)

```

!            Example 3. Use composite 9-point Gauss quadrature with 1,000 subintervals  
 !            to integrate  $\sin(t) / t$  from  $t = 0$  to  $t = 20$ .

```

OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Integration')

WRITE (K_GRAPH,"(//A//)") &
  'Example 3. Use composite 9-point Gauss quadrature with 1,000 subintervals'

NF = 1
N = 1000
A = 0
B = 20
CALL GAUSS_9(NF, A, B, N, RESULT)

WRITE (22,*) ' '
WRITE (22,*) " Example 3. Integrate  $\sin(t) / t$  from  $t = 0$  to  $t = 20$ :"
CALL FMPRINT_INTERVAL(RESULT)
WRITE (22,*) ' '
ERROR_FM = ABS( (RIGHT_ENDPOINT(RESULT)-LEFT_ENDPOINT(RESULT)) / RIGHT_ENDPOINT(RESULT) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(A,I3,A)") '        The two endpoints agree to about',J,' decimal digits.'
WRITE (22,*) ' '
CLOSE(K_GRAPH)

```

!            Example 4. Start with  $(x, y) = (1, 0)$  and step around the unit circle  
 !            using the recurrence  $(x, y) \leftarrow (x*c - y*d, y*c + x*d)$ ,  
 !             $c = \cos(2*\pi/n)$  and  $s = \sin(2*\pi/n)$

!            After 23 trips around the circle with 100 steps per trip,  
 !             $x = (-58.6, 60.6)$  and  $y = (-59.6, 59.6)$

!            After 21 trips around the circle with 1000 steps per trip,  
 !             $x = (0.49, 1.51)$  and  $y = (-0.51, 0.51)$

```

OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Circle Recurrence')

WRITE (K_GRAPH,"(//A//)") &
  'Example 4. Start with  $(x, y) = (1, 0)$  and step around the unit circle'

N = 100
X = 1
Y = 0
PI = ACOS(TO_FM(-1))
C = COS(2*PI/N)
S = SIN(2*PI/N)

WRITE (22,*) ' '
WRITE (22,*) " Example 4. Step around the unit circle multiple times using a recurrence"
WRITE (22,*) ' '

DO K = 1, 23

```

```

DO J = 1, N
  A = X*C - Y*S
  B = Y*C + X*S
  X = A
  Y = B
  WIDTH_FM = MAX( RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X) , EPSILON(LEFT_ENDPOINT(X))/10**7 )
  WRITE (K_GRAPH,"(A,I7,A,F8.3,A)" '{', N*(K-1)+J, &
        ', ', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '}', '
ENDDO
ERROR_FM = ABS( (RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X)) / RIGHT_ENDPOINT(X) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(I4,A,I3,A)" K,' trips. The two endpoints for x agree to about', &
      J,' decimal digits.'
ENDDO

```

```

WRITE (22,*) ' '
WRITE (22,*) ' X ='
CALL FMPRINT_INTERVAL(X)
WRITE (22,*) ' '
WRITE (22,*) ' Y ='
CALL FMPRINT_INTERVAL(Y)
WRITE (22,*) ' '
ERROR_FM = ABS( (RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X)) / RIGHT_ENDPOINT(X) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(A,I3,A)" ' The two endpoints for x agree to about',J,' decimal digits.'
WRITE (22,*) ' '

```

! Compare the same calculation in ordinary FM arithmetic.

! After 23 trips around the circle with 100 steps per trip,  
! x = 1.0 to 60 digits and y = -2.3e-55

! After 23 trips around the circle with 1000 steps per trip,  
! x = 1.0 exactly and y = -2.3e-55

```

WRITE (22,*) ' '
WRITE (22,*) ' Compare the same calculation in ordinary FM arithmetic.'
WRITE (22,*) ' '
N = 100
X_FM = 1
Y_FM = 0
PI_FM = ACOS(TO_FM(-1))
C_FM = COS(2*PI_FM/N)
S_FM = SIN(2*PI_FM/N)

```

```

DO K = 1, 23
  DO J = 1, N
    A_FM = X_FM*C_FM - Y_FM*S_FM
    B_FM = Y_FM*C_FM + X_FM*S_FM
    X_FM = A_FM
    Y_FM = B_FM
  ENDDO
  ERROR_FM = ABS( X_FM - 1 ) + TO_FM(' 1.0e-99 ')
  J = -NINT(LOG10(ERROR_FM))
  WRITE (22,"(I4,A,I3,A)" K,' trips. x agrees with 1.0 to about', &
        J,' decimal digits.'
ENDDO

```

ENDDO

```

WRITE (22,*) ' '
WRITE (22,*) ' X ='
CALL FM_PRINT(X_FM)
WRITE (22,*) ' '
WRITE (22,*) ' Y ='
CALL FM_PRINT(Y_FM)
WRITE (22,*) ' '
CLOSE(K_GRAPH)

```

!            Test to see if it is x and y oscillating between positive and negative values  
!            that causes the exponential growth of interval width, or if oscillation through  
!            positive values alone can cause it.

!            Use the recurrence it oscillate back and forth between step number 8 and  
!            step number 18 of the original recurrence.  
!            Once back and forth takes 20 steps, so  $5 \times 23 = 115$  trips back and forth will  
!            take the same number of steps as the 23 trips around the circle above.

```

OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Back and Forth Recurrence')

```

```

WRITE (K_GRAPH,"(//A/A//)") &
'Example 4b. Start with ( x, y ) = ( cos( 8* 2*pi/n), sin( 8* 2*pi/n) ) ', &
' and step around the unit circle to ( cos(18* 2*pi/n), sin(18* 2*pi/n) ) and back'

```

```

WRITE (22,*) ' '
WRITE (22,*) " Example 4b. Step back and forth around the unit circle using a recurrence"
WRITE (22,*) ' '

```

```

N = 100
PI = ACOS(TO_FM(-1))
X = COS(8* 2*PI/N)
Y = SIN(8* 2*PI/N)
C = COS(2*PI/N)
S = SIN(2*PI/N)

```

```

WRITE (22,*) ' '
WRITE (22,*) ' Starting point for X ='
CALL FMPRINT_INTERVAL(X)
WRITE (22,*) ' '
WRITE (22,*) ' Starting point for Y ='
CALL FMPRINT_INTERVAL(Y)
WRITE (22,*) ' '

```

!            5 back and forth trips equals 100 steps, the same as one circle trip above.

```

DO K = 1, 23 * 5
  DO J = 1, 20
    A = X*C - Y*S
    B = Y*C + X*S
    X = A
    Y = B
    WIDTH_FM = MAX( RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X) , EPSILON(LEFT_ENDPOINT(X))/10**7 )
    WRITE (K_GRAPH,"(A,I7,A,F8.3,A)") '{', (20*(K-1)+J), &
      ', ', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '}', '
  
```



```

    IF (MOD(J,10) == 0) THEN
        S = -S
    ENDIF
ENDDO
IF (MOD(K,5) == 0) THEN
    ERROR_FM = ABS( (RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X)) / RIGHT_ENDPOINT(X) )
    J = -NINT(LOG10(ERROR_FM))
    WRITE (22,"(I4,A,I3,A)") K, ' trips. The two endpoints for x agree to about', &
        J, ' decimal digits.'
ENDIF
ENDDO

WRITE (22,*) ' '
WRITE (22,*) ' X ='
CALL FMPRINT_INTERVAL(X)
WRITE (22,*) ' '
WRITE (22,*) ' Y ='
CALL FMPRINT_INTERVAL(Y)
WRITE (22,*) ' '
ERROR_FM = ABS( (RIGHT_ENDPOINT(X)-LEFT_ENDPOINT(X)) / RIGHT_ENDPOINT(X) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(A,I3,A)") ' ' The two endpoints for x agree to about',J,' decimal digits.'
WRITE (22,*) ' '
CLOSE(K_GRAPH)

```

! Example 5. 2 \* Product  $4^n / (4^n - 1)$  from 1 to 10,000.

```

OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Product')

WRITE (K_GRAPH,"(//A//)") &
    'Example 5. 2 * Product  $4^n / (4^n - 1)$  from 1 to 10,000.'

N = 10**4
S = 2
DO J = 1, N
    K = 4 * J * J
    TERM = K
    TERM = TERM / ( K - 1 )
    S = S * TERM
    IF (MOD(J,N/1000) == 1) THEN
        WIDTH_FM = MAX( RIGHT_ENDPOINT(S)-LEFT_ENDPOINT(S) , EPSILON(LEFT_ENDPOINT(S)) )
        WRITE (K_GRAPH,"(A,I7,A,F8.3,A)") '{', J, ',', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '}',
    ENDIF
ENDDO

WRITE (22,*) ' '
WRITE (22,*) " Example 5. 2 * Product  $4^n / (4^n - 1)$  from 1 to 10,000. ="
CALL FMPRINT_INTERVAL(S)
WRITE (22,*) ' '
ERROR_FM = ABS( (RIGHT_ENDPOINT(S)-LEFT_ENDPOINT(S)) / RIGHT_ENDPOINT(S) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(A,I3,A)") ' ' The two endpoints of y(30) agree to about',J,' decimal digits.'
WRITE (22,*) ' '

```

! Example 6. Differential equation.  $y'' = -y' / 10 - 2 * y / (x+2)$ ,  
 !  $y(0) = 0, \quad y'(0) = 1$

! This solution has 2 roots between 0 and 30.

```
OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Diff Eq 1')
```

```
WRITE (K_GRAPH,"(//A//)") &  
"Example 6. Differential equation.  $y'' = -y' / 10 - 2 * y / (x+2)$ ,"
```

```
WRITE (22,*) ' '  
WRITE (22,*) " Example 6. Differential equation.  $y'' = -y' / 10 - 2 * y / (x+2)$ "  
WRITE (22,*) ' '  
N_ORDER = 2  
A = 0  
B = 30  
N_STEPS = 10000  
V(1) = 0  
V(2) = 1  
K_FUNCTION = 1  
KPRT = N_STEPS / 10  
KW = 22  
CALL RK4(N_ORDER, A, B, N_STEPS, V, K_FUNCTION, KPRT, KW)  
WRITE (22,*) ' '  
WRITE (22,*) ' y(30) = '  
WRITE (22,*) ' '  
CALL FMPRINT_INTERVAL(V(1))  
WRITE (22,*) ' '  
WRITE (22,*) " y'(30) = "  
WRITE (22,*) ' '  
CALL FMPRINT_INTERVAL(V(2))  
WRITE (22,*) ' '  
ERROR_FM = ABS( (RIGHT_ENDPOINT(V(1))-LEFT_ENDPOINT(V(1))) / RIGHT_ENDPOINT(V(1)) )  
J = -NINT(LOG10(ERROR_FM))  
WRITE (22,"(A,I3,A)") ' The two endpoints of y(30) agree to about',J,' decimal digits.'  
WRITE (22,*) ' '
```

! Example 7. Differential equation.  $y'' = -y' / 10 - 200 * y / (x+2)$ ,  
 !  $y(0) = 0, \quad y'(0) = 1$

! This solution has 38 roots between 0 and 30.  
 ! The rapid oscillation causes the interval calculation to be unstable  
 ! and lose accuracy in a manner similar to example 4, the stepping  
 ! recurrence making multiple trips around the unit circle.

```
OPEN(K_GRAPH,FILE='INTERVAL_EXAMPLES Diff Eq 2')
```

```

WRITE (K_GRAPH, "(//A//)") &
    "Example 7. Differential equation.  $y' = -y' / 10 - 200 * y / (x+2),$ "

WRITE (22,*) ' '
WRITE (22,*) " Example 7. Differential equation.  $y' = -y' / 10 - 200 * y / (x+2)$ "
WRITE (22,*) ' '
N_ORDER = 2
A = 0
B = 30
N_STEPS = 10000
V(1) = 0
V(2) = 1
K_FUNCTION = 2
KPRT = N_STEPS / 10
KW = 22
CALL RK4(N_ORDER, A, B, N_STEPS, V, K_FUNCTION, KPRT, KW)
WRITE (22,*) ' '
WRITE (22,*) ' y(30) = '
WRITE (22,*) ' '
CALL FMPRINT_INTERVAL(V(1))
WRITE (22,*) ' '
WRITE (22,*) " y'(30) = "
WRITE (22,*) ' '
CALL FMPRINT_INTERVAL(V(2))
WRITE (22,*) ' '
ERROR_FM = ABS( (RIGHT_ENDPOINT(V(1))-LEFT_ENDPOINT(V(1))) / RIGHT_ENDPOINT(V(1)) )
J = -NINT(LOG10(ERROR_FM))
WRITE (22,"(A,I3,A)") ' The two endpoints of y(30) agree to about',J,' decimal digits.'
WRITE (22,*) ' '
CLOSE(K_GRAPH)

```

!            Example 8. Solve a "random" NxN linear system.

```

WRITE (22,*) ' '
WRITE (22,*) ' Example 8. Solve a "random" NxN linear system.'
WRITE (22,*) ' '

DO N = 10, 100, 10

    ALLOCATE( MATRIX(N,N), RHS(N), SOLN(N), KSWAP(N) )
    DO I = 1, N
        DO J = 1, N
            CALL FM_RANDOM_NUMBER(RAND)
            MATRIX(I,J) = RAND
        ENDDO
        RHS(I) = I
    ENDDO

    CALL FM_INTERVAL_FACTOR_LU(MATRIX, N, DET, KSWAP)
    CALL FM_INTERVAL_SOLVE_LU (MATRIX, N, RHS, SOLN, KSWAP)

```

```

WRITE (22,*) ' '
J_MIN = 99
J_MAX = 0
DO I = 1, N
  ERROR_FM = ABS( (RIGHT_ENDPOINT(SOLN(I))-LEFT_ENDPOINT(SOLN(I))) / &
                  RIGHT_ENDPOINT(SOLN(I)) )
  J = -NINT(LOG10(ERROR_FM))
  J_MIN = MIN( J_MIN, J )
  J_MAX = MAX( J_MAX, J )
ENDDO
WRITE (22, "(A,I3,A,I3,A,I3,A)") ' For N = ', N, &
  ' the solution elements agree to between ', &
  J_MIN, ' and ', J_MAX, ' significant digits.'

DEALLOCATE( MATRIX, RHS, SOLN, KSWAP )
ENDDO

```

!            Example 9. Newton's method starting with a single point. Solve  $x \cdot \exp(x) - 2 = 0$ .

```

WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) " Example 9. Newton's method starting with a single point."
WRITE (22,*) ' '

X = 1
DO J = 1, 10
  A = X*EXP(X) - 2
  B = (X+1)*EXP(X)
  X = X - A/B
  WRITE (22,*) ' '
  WRITE (22, "(A,I2,A)") " Iteration ",J, ". X ="
  CALL FMPRINT_INTERVAL(X)
ENDDO

```

!            Example 10. Newton's method starting with an interval containing a root.  
!            Solve  $x \cdot \exp(x) - 2 = 0$ .  
!            The next interval is  $x_t - f(x_t)/f'(x_t)$  intersected with  $x$ .  
!             $x_t$  is a single point from the interval  $x$  (the midpoint is used here).  
!            The idea is to have a contracting sequence of intervals that  
!            each contain a root.  
!            Refs: Hansen-Greenberg 83, Baker Kearfott 95-97,  
!            Mayer 95, van Hentenryck et al. 97

```

WRITE (22,*) ' '
WRITE (22,*) ' '
WRITE (22,*) " Example 10. Newton's method starting with an interval containing a root"
WRITE (22,*) ' '

```

```

X = TO_FM_INTERVAL( " 0.5 ", " 1.0 " )
DO J = 1, 9
  X_FM = LEFT_ENDPOINT(X)/2 + RIGHT_ENDPOINT(X)/2
  Y_FM = X_FM*EXP(X_FM) - 2
  B = (X+1)*EXP(X)
  Y = X_FM - Y_FM/B

  X = TO_FM_INTERVAL( MAX( LEFT_ENDPOINT(X),LEFT_ENDPOINT(Y) ) , &
                     MIN( RIGHT_ENDPOINT(X),RIGHT_ENDPOINT(Y) ) )
  WRITE (22,*) ' '
  WRITE (22,"(A,I2,A)") " Iteration ",J,'. X ='
  CALL FMPRINT_INTERVAL(X)
ENDDO

```

```

WRITE (22,*) ' '
WRITE (22,*) ' '

```

```

END PROGRAM TEST

```

```

FUNCTION F(NF, X)      RESULT (RETURN_VALUE)

```

```

USE FMZM
USE FM_INTERVAL_ARITHMETIC
IMPLICIT NONE
INTEGER :: NF
TYPE (FM_INTERVAL) :: RETURN_VALUE, X

```

```

IF (NF == 1) THEN
  RETURN_VALUE = SIN(X) / X
ENDIF

```

```

END FUNCTION F

```

```

SUBROUTINE GAUSS_9(NF, A, B, N, RESULT)

```

! Sample subroutine usage for FM.

! Integrate F(NF, X) from A to B using N subintervals, and return the answer in RESULT.

! This does numerical integration using a 9-point Gauss quadrature rule.

! It is not a very good way to do high-precision integration, but it is a short routine

! and can often get 50 digits if f(x) is well-behaved and the interval of integration

! is not too big.

```

USE FMVALS
USE FMZM
USE FM_INTERVAL_ARITHMETIC
IMPLICIT NONE
TYPE (FM_INTERVAL) :: A, B, RESULT
TYPE (FM_INTERVAL), SAVE :: AJ, H2, XJ, XI(9), WI(9)
TYPE (FM_INTERVAL), EXTERNAL :: F
TYPE (FM), SAVE :: WIDTH_FM
INTEGER :: NF, N, J, K
INTEGER, SAVE :: COEFF_BASE = 0, COEFF_PRECISION = 0
INTENT (IN) :: N, A, B
INTENT (INOUT) :: RESULT

```

```

IF (COEFF_BASE /= MBASE .OR. COEFF_PRECISION < NDIG) THEN
  COEFF_BASE = MBASE
  COEFF_PRECISION = NDIG
  XI(9) = TO_FM( ' 0.968160239507626089835576202903672870049404800491925329550023 ' )
  XI(1) = -XI(9)
  XI(8) = TO_FM( ' 0.836031107326635794299429788069734876544106718124675996104372 ' )
  XI(2) = -XI(8)
  XI(7) = TO_FM( ' 0.613371432700590397308702039341474184785720604940564692872813 ' )
  XI(3) = -XI(7)
  XI(6) = TO_FM( ' 0.324253423403808929038538014643336608571956260736973088827047 ' )
  XI(4) = -XI(6)
  XI(5) = 0

  WI(9) = TO_FM( ' 0.081274388361574411971892158110523650675661720782410750711108 ' )
  WI(1) = WI(9)
  WI(8) = TO_FM( ' 0.180648160694857404058472031242912809514337821732040484498336 ' )
  WI(2) = WI(8)
  WI(7) = TO_FM( ' 0.260610696402935462318742869418632849771840204437299951939997 ' )
  WI(3) = WI(7)
  WI(6) = TO_FM( ' 0.3123470770400028400686304065844436655987548612619046465554011 ' )
  WI(4) = WI(6)
  WI(5) = TO_FM( ' 32768 ' ) / 99225
ENDIF

RESULT = 0
H2 = ( B - A ) / ( 2 * N )
DO J = 1, N
  AJ = A + (J-1) * 2 * H2
  DO K = 1, 9
    XJ = AJ + H2 + H2 * XI(K)
    RESULT = RESULT + F( NF, XJ ) * WI(K)
  ENDDO
  IF (MOD(J,N/1000) == 0) THEN
    WIDTH_FM = MAX( RIGHT_ENDPOINT(RERESULT)-LEFT_ENDPOINT(RERESULT) , &
      EPSILON(LEFT_ENDPOINT(RERESULT)) )
    WRITE (31, "(A,I7,A,F8.3,A)" ' {', J, ',', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '}', '
  ENDF
ENDDO

RESULT = RESULT * H2

END SUBROUTINE GAUSS_9

```

```

MODULE RK_FUNCT

```

! For a function to return an array as its function value, there must be an explicit interface.

! F is the generic name of the function, F\_RK is the specific version for 1-dimensional arrays.

```

INTERFACE F
  MODULE PROCEDURE F_RK
END INTERFACE

```

```

CONTAINS

```

```
FUNCTION F_RK( X, V, K_FUNCTION )      RESULT (RETURN_VALUE)
```

```
USE FMZM
```

```
USE FM_INTERVAL_ARITHMETIC
```

```
IMPLICIT NONE
```

```
INTEGER :: K_FUNCTION
```

```
TYPE (FM_INTERVAL) :: X, V(5), RETURN_VALUE(5)
```

```
RETURN_VALUE = 0
```

```
IF (K_FUNCTION == 1) THEN
```

```
!           Function 1.   $y'' = -y' / 10 - 2 * y / (x+2),$   
!                        $y(0) = 0, \quad y'(0) = 1$ 
```

```
!                        $v' = \{ y', y'' \} = \{ u, -u / 10 - 2 y / (x+2) \} = F(x,v)$ 
```

```
RETURN_VALUE(1) = V(2)
```

```
RETURN_VALUE(2) = -V(2) / 10 - 2 * V(1) / ( X + 2 )
```

```
ELSE IF (K_FUNCTION == 2) THEN
```

```
!           Function 2.   $y'' = -y' / 10 - 200 * y / (x+2),$   
!                        $y(0) = 0, \quad y'(0) = 1$ 
```

```
!                        $v' = \{ y', y'' \} = \{ u, -u / 10 - 2 y / (x+2) \} = F(x,v)$ 
```

```
RETURN_VALUE(1) = V(2)
```

```
RETURN_VALUE(2) = -V(2) / 10 - 200 * V(1) / ( X + 2 )
```

```
ELSE
```

```
RETURN_VALUE(1) = X*V(1)
```

```
ENDIF
```

```
END FUNCTION F_RK
```

```
END MODULE RK_FUNCT
```

```
SUBROUTINE RK4(N_ORDER, A, B, N_STEPS, V, K_FUNCTION, KPRT, KW)
```

```
USE FMZM
```

```
USE FM_INTERVAL_ARITHMETIC
```

```
USE RK_FUNCT
```

```
! N_ORDER is the order of the DE
```

```
! A is the initial x-value
```

```
! B is the final x-value
```

```
! N_STEPS is the number of steps done.
```

```
! V contains the initial values at x = a on input, and is returned with the solution
```

```
! values at x = b.
```

```
! K_FUNCTION is the function number for the RHS.
```

```
! KPRT > 0 causes the solution to be printed on unit KW after each KPRT steps.
```

```
IMPLICIT NONE
```

```
INTEGER :: J, K, K_FUNCTION, KPRT, KW, N_ORDER, N_STEPS
```

```
TYPE (FM_INTERVAL) :: A, B, H, X, V(5), K1(5), K2(5), K3(5), K4(5)
```

```
TYPE (FM) :: ERROR_FM, WIDTH_FM
```

```
SAVE :: H, X, K1, K2, K3, K4
```

```
X = A
```

```
H = (B-A)/N_STEPS
```

```
IF (N_ORDER < 5) V(N_ORDER+1:5) = 0
```

```
DO J = 1, N_STEPS
```

```
  K1 = H * FC( X      , V      , K_FUNCTION )
```

```
  K2 = H * FC( X+H/2 , V+K1/2 , K_FUNCTION )
```

```
  K3 = H * FC( X+H/2 , V+K2/2 , K_FUNCTION )
```

```
  K4 = H * FC( X+H   , V+K3   , K_FUNCTION )
```

```
  X = A + J*H
```

```
  V = V + ( K1 + 2*K2 + 2*K3 + K4 ) / 6
```

```
IF (KPRT > 0) THEN
```

```
  IF (MOD(J,KPRT) == 0) THEN
```

```
    ERROR_FM = ABS( (RIGHT_ENDPOINT(V(1))-LEFT_ENDPOINT(V(1))) / RIGHT_ENDPOINT(V(1)) )
```

```
    K = -NINT(LOG10(ERROR_FM))
```

```
    WRITE (KW,"(A,F15.10,A,2F20.15,A,I3,A)" ' X = ',TO_DP(X), ' V = ', &  
          TO_DP(V(1:N_ORDER)), ' Endpoints of V(1) agree to ', K, ' digits.'
```

```
  ENDIF
```

```
ENDIF
```

```
IF (MOD(J,N_STEPS/1000) == 1) THEN
```

```
  WIDTH_FM = MAX( RIGHT_ENDPOINT(V(1))-LEFT_ENDPOINT(V(1)) , &  
                 EPSILON(LEFT_ENDPOINT(V(1))) )
```

```
  IF (K_FUNCTION == 1) THEN
```

```
    WRITE (31,"(A,I7,A,F8.3,A)" '{', J, ',', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '},'
```

```
  ELSE
```

```
    WRITE (31,"(A,I7,A,F8.3,A)" '{', J, ',', TO_DP(LOG(WIDTH_FM)) / LOG(10.0D0), '},'
```

```
  ENDIF
```

```
ENDIF
```

```
ENDDO
```

```
END SUBROUTINE RK4
```

```
SUBROUTINE FM_INTERVAL_FACTOR_LU(A,N,DET,KSWAP)
```

```
USE FMZM
```

```
USE FM_INTERVAL_ARITHMETIC
```

```
IMPLICIT NONE
```

```
! Gauss elimination to factor the NxN matrix A (LU decomposition).
```

```
! The time is proportional to N**3.
```

```
! Once this factorization has been done, a linear system  $Ax = b$   
! with the same coefficient matrix A and Nx1 vector b can be solved  
! for x using routine FM_INTERVAL_SOLVE_LU in time proportional to N**2.
```

```
! DET is returned as the determinant of A.  
! Nonzero DET means a solution can be found.  
! DET = 0 is returned if the system is singular.
```

```
! KSWAP is a list of row interchanges made by the partial pivoting strategy during the  
! elimination phase.
```



! After returning, the values in matrix A have been replaced by the multipliers  
! used during elimination. This is equivalent to factoring the A matrix into  
! a lower triangular matrix L times an upper triangular matrix U.

```
INTEGER :: N
INTEGER :: JCOL, JDIAG, JMAX, JROW, KSWAP(N)
TYPE (FM_INTERVAL) :: A(N,N), DET
TYPE (FM_INTERVAL), SAVE :: AMAX, AMULT, TEMP
```

```
DET = 1
KSWAP(1:N) = 1
IF (N <= 0) THEN
    DET = 0
    RETURN
ENDIF
IF (N == 1) THEN
    KSWAP(1) = 1
    DET = A(1,1)
    RETURN
ENDIF
```

! Do the elimination phase.  
! JDIAG is the current diagonal element below which the elimination proceeds.

```
DO JDIAG = 1, N-1
```

! Pivot to put the element with the largest absolute value on the diagonal.

```
AMAX = ABS(A(JDIAG,JDIAG))
JMAX = JDIAG
DO JROW = JDIAG+1, N
    IF (ABS(A(JROW,JDIAG)) > AMAX) THEN
        AMAX = ABS(A(JROW,JDIAG))
        JMAX = JROW
    ENDIF
ENDDO
```

! If AMAX is zero here then the system is singular.

```
IF (AMAX == 0.0) THEN
    DET = 0
    RETURN
ENDIF
```

! Swap rows JDIAG and JMAX unless they are the same row.

```
KSWAP(JDIAG) = JMAX
IF (JMAX /= JDIAG) THEN
    DET = -DET
    DO JCOL = JDIAG, N
        TEMP = A(JDIAG,JCOL)
        A(JDIAG,JCOL) = A(JMAX,JCOL)
        A(JMAX,JCOL) = TEMP
    ENDDO
ENDIF
DET = DET * A(JDIAG,JDIAG)
```

```
!           For JROW = JDIAG+1, ..., N, eliminate A(JROW,JDIAG) by replacing row JROW by
!           row JROW - A(JROW,JDIAG) * row JDIAG / A(JDIAG,JDIAG)
```

```
DO JROW = JDIAG+1, N
  IF (A(JROW,JDIAG) == 0) CYCLE
  AMULT = A(JROW,JDIAG)/A(JDIAG,JDIAG)
```

```
!           Save the multiplier for use later by FM_INTERVAL_SOLVE_LU.
```

```
  A(JROW,JDIAG) = AMULT
  A(JROW,JDIAG+1:N) = A(JROW,JDIAG+1:N) - AMULT*A(JDIAG,JDIAG+1:N)
```

```
ENDDO
```

```
ENDDO
```

```
DET = DET * A(N,N)
```

```
END SUBROUTINE FM_INTERVAL_FACTOR_LU
```

```
SUBROUTINE FM_INTERVAL_SOLVE_LU(A,N,B,X,KSWAP)
```

```
USE FMZM
```

```
USE FM_INTERVAL_ARITHMETIC
```

```
IMPLICIT NONE
```

```
! Solve a linear system  $A x = b$ .
! A is the NxN coefficient matrix, after having been factored by FM_INTERVAL_FACTOR_LU.
! B is the Nx1 right-hand-side vector.
! X is returned with the solution of the linear system.
! KSWAP is a list of row interchanges made by the partial pivoting strategy during the
! elimination phase in FM_INTERVAL_FACTOR_LU.
! Time for this call is proportional to  $N^2$ .
```

```
INTEGER :: N, KSWAP(N)
TYPE (FM_INTERVAL) :: A(N,N), B(N), X(N)
TYPE (FM_INTERVAL), SAVE :: TEMP
INTEGER :: JDIAG, JMAX
```

```
IF (N <= 0) THEN
  RETURN
```

```
ENDIF
```

```
IF (N == 1) THEN
  X(1) = B(1) / A(1,1)
  RETURN
```

```
ENDIF
```

```
X(1:N) = B(1:N)
```

```
!           Do the elimination phase operations only on X.
!           JDIAG is the current diagonal element below which the elimination proceeds.
```

```
DO JDIAG = 1, N-1
```

```
!           Pivot to put the element with the largest absolute value on the diagonal.
```

```
  JMAX = KSWAP(JDIAG)
```

```
!           Swap rows JDIAG and JMAX unless they are the same row.
```

```
IF (JMAX /= JDIAG) THEN
  TEMP = X(JDIAG)
  X(JDIAG) = X(JMAX)
```

```
X(JMAX) = TEMP
```

```
ENDIF
```

```
!           For JROW = JDIAG+1, ..., N, eliminate A(JROW,JDIAG) by replacing row JROW by  
!           row JROW - A(JROW,JDIAG) * row JDIAG / A(JDIAG,JDIAG)  
!           After factoring, A(JROW,JDIAG) is the original A(JROW,JDIAG) / A(JDIAG,JDIAG).
```

```
X(JDIAG+1:N) = X(JDIAG+1:N) - A(JDIAG+1:N,JDIAG)*X(JDIAG)
```

```
ENDDO
```

```
!           Do the back substitution.
```

```
DO JDIAG = N, 1, -1
```

```
!           Divide row JDIAG by the diagonal element.
```

```
X(JDIAG) = X(JDIAG) / A(JDIAG,JDIAG)
```

```
!           Zero above the diagonal in column JDIAG by replacing row JROW by  
!           row JROW - A(JROW,JDIAG) * row JDIAG  
!           For JROW = 1, ..., JDIAG-1.
```

```
IF (JDIAG == 1) EXIT
```

```
X(1:JDIAG-1) = X(1:JDIAG-1) - A(1:JDIAG-1,JDIAG)*X(JDIAG)
```

```
ENDDO
```

```
END SUBROUTINE FM_INTERVAL_SOLVE_LU
```