```
 module fm_interval_arithmetic_1


!  FM_interval 1.4                        David M. Smith                        Interval Arithmetic

!  This module extends the definition of the basic Fortran arithmetic and function operations so
!  they also apply to multiple precision intervals, using version 1.4 of FM.
!  The multiple precision interval data type is called
!    type (fm_interval)

!  Each FM interval consists of two endpoints, with each endpoint being a type(fm) multiple
!  precision number.  The first of these endpoints defines the left endpoint of an interval,
!  and the second defines the right endpoint of the interval.

!  Most of the functions defined in this module are multiple precision interval versions of standard
!  Fortran functions.  In addition, there are functions for direct conversion, formatting, and some
!  mathematical special functions.

!  to_fm_interval is a function for converting other types of numbers to type fm_interval.
!  Like the to_fm function in module fmzm, to_fm_interval(3.12) converts the real constant
!  to an FM interval, but it is accurate only to single precision.  to_fm_interval(3.12d0)
!  agrees with 3.12 to double precision accuracy, and to_fm_interval('3.12') or
!  to_fm_interval(312)/to_fm_interval(100) agrees to full FM accuracy.


    use fmzm


!  For all comparisons except == and /=, the order is not well defined if intervals overlap.
!  In those cases, the midpoints of the intervals are compared.

    type fm_interval
         type(multi) :: left
         type(multi) :: right
    end type

!            Work variables for derived type operations.

    type (fm_interval), save :: mtfm_i, mufm_i, mvfm_i, mwfm_i, m0fm_i, m1fm_i, m2fm_i, m3fm_i
    type (fm_interval), save :: m4fm_i, m5fm_i, m6fm_i, m7fm_i, m8fm_i, m9fm_i
    type (fm), save :: m_1, m_2, m_3, m_4, m_5, m_6, m_7, m_8, m_9, m_10, m_11, m_12,  &
                       x_edge, y_edge, xy_edge, f_left, f_right
    type (zm), save :: mz_1
    type(multi), save :: mtim_i, mtzm_i(2)
    integer, parameter :: n_prev = 10
    integer, save :: ndig_xy_edge, kxy_edge, k_routine_edge, kround_prev(0:n_prev-1),  &
                     routine_prev(0:n_prev-1), num_prev = 0
    type (fm), save :: m1_prev(0:n_prev-1), m2_prev(0:n_prev-1),  &
                       m3_prev(0:n_prev-1)

   interface to_fm_interval

!            Create an interval by giving both endpoints.

      module procedure interval_fm_i
      module procedure interval_fm_r
      module procedure interval_fm_d
```

```
      module procedure interval_fm_z
      module procedure interval_fm_zd
      module procedure interval_fm_fm
      module procedure interval_fm_im
      module procedure interval_fm_zm
      module procedure interval_fm_st

!             Convert single values to intervals with both endpoints the same.

      module procedure fm_interval_i
      module procedure fm_interval_r
      module procedure fm_interval_d
      module procedure fm_interval_z
      module procedure fm_interval_zd
      module procedure fm_interval_fm
      module procedure fm_interval_fma
      module procedure fm_interval_im
      module procedure fm_interval_zm
      module procedure fm_interval_st
      module procedure fm_interval_i1
      module procedure fm_interval_r1
      module procedure fm_interval_d1
      module procedure fm_interval_z1
      module procedure fm_interval_zd1
      module procedure fm_interval_fm1
      module procedure fm_interval_fma1
      module procedure fm_interval_im1
      module procedure fm_interval_zm1
      module procedure fm_interval_st1
      module procedure fm_interval_i2
      module procedure fm_interval_r2
      module procedure fm_interval_d2
      module procedure fm_interval_z2
      module procedure fm_interval_zd2
      module procedure fm_interval_fm2
      module procedure fm_interval_fma2
      module procedure fm_interval_im2
      module procedure fm_interval_zm2
      module procedure fm_interval_st2
   end interface

!             Return the left or right endpoint of an interval as a type (fm) number.

   interface left_endpoint
      module procedure left_endpoint_interval_fm
   end interface

   interface right_endpoint
      module procedure right_endpoint_interval_fm
   end interface

   interface to_fm
      module procedure fm_fm_interval
      module procedure fm_fm_interval1
      module procedure fm_fm_interval2
   end interface

   interface to_im
```

```
   module procedure im_fm_interval
   module procedure im_fm_interval1
   module procedure im_fm_interval2
end interface

interface to_zm
   module procedure zm_fm_interval
   module procedure zm_fm_interval1
   module procedure zm_fm_interval2
end interface

interface to_int
   module procedure fm_interval_2int
   module procedure fm_interval_2int1
   module procedure fm_interval_2int2
end interface

interface to_sp
   module procedure fm_interval_2sp
   module procedure fm_interval_2sp1
   module procedure fm_interval_2sp2
end interface

interface to_dp
   module procedure fm_interval_2dp
   module procedure fm_interval_2dp1
   module procedure fm_interval_2dp2
end interface

interface to_spz
   module procedure fm_interval_2spz
   module procedure fm_interval_2spz1
   module procedure fm_interval_2spz2
end interface

interface to_dpz
   module procedure fm_interval_2dpz
   module procedure fm_interval_2dpz1
   module procedure fm_interval_2dpz2
end interface

interface is_overflow
   module procedure fm_interval_is_overflow
   module procedure fm_interval_is_overflow1
   module procedure fm_interval_is_overflow2
end interface

interface is_underflow
   module procedure fm_interval_is_underflow
   module procedure fm_interval_is_underflow1
   module procedure fm_interval_is_underflow2
end interface

interface is_unknown
   module procedure fm_interval_is_unknown
   module procedure fm_interval_is_unknown1
   module procedure fm_interval_is_unknown2
end interface
```

```
   interface fm_interval_undef_inp
      module procedure fm_undef_inp_interval_fm0
      module procedure fm_undef_inp_interval_fm1
      module procedure fm_undef_inp_interval_fm2
   end interface

 contains

!                                                        to_fm_interval

   function fm_interval_i(ival)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      integer :: ival
      intent (in) :: ival
      call fmi2m_interval(ival, return_value)
   end function fm_interval_i

   function fm_interval_r(r)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      real :: r
      intent (in) :: r
      call fmsp2m_interval(r, return_value)
   end function fm_interval_r

   function fm_interval_d(d)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      double precision :: d
      intent (in) :: d
      call fmdp2m_interval(d, return_value)
   end function fm_interval_d

   function fm_interval_z(z)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      complex :: z
      intent (in) :: z
      call fmsp2m_interval(real(z), return_value)
   end function fm_interval_z

   function fm_interval_zd(c)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      complex (kind(0.0d0)) :: c
      intent (in) :: c
      call fmdp2m_interval(real(c, kind(0.0d0)), return_value)
   end function fm_interval_zd

   function fm_interval_fm(ma)     result (return_value)
      use fmvals
```

```
   implicit none
   type (fm_interval) :: ma, return_value
   intent (in) :: ma
   call fm_interval_undef_inp(ma)
   call fmmin(ma%left, ma%right, return_value%left)
   call fmmax(ma%left, ma%right, return_value%right)
end function fm_interval_fm

function fm_interval_fma(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   type (fm) :: ma
   intent (in) :: ma
   call fm_undef_inp(ma)
   call fmeq(ma%mfm, return_value%left)
   call fmeq(ma%mfm, return_value%right)
end function fm_interval_fma

function fm_interval_im(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   type (im) :: ma
   intent (in) :: ma
   call fm_undef_inp(ma)
   call imi2fm(ma%mim, return_value%left)
   call imi2fm(ma%mim, return_value%right)
end function fm_interval_im

function fm_interval_zm(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   type (zm) :: ma
   intent (in) :: ma
   call fm_undef_inp(ma)
   call zmreal_interval(ma%mzm, return_value)
end function fm_interval_zm

function fm_interval_st(st)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   character(*) :: st
   intent (in) :: st
   call fmst2m_interval(st, return_value)
end function fm_interval_st

function fm_interval_i1(ival)     result (return_value)
   use fmvals
   implicit none
   integer, dimension(:) :: ival
   type (fm_interval), dimension(size(ival)) :: return_value
   integer :: j, n
   intent (in) :: ival
   n = size(ival)
   do j = 1, n
```

```
      call fmi2m_interval(ival(j), return_value(j))
   enddo
end function fm_interval_i1

function fm_interval_r1(r)     result (return_value)
   use fmvals
   implicit none
   real, dimension(:) :: r
   type (fm_interval), dimension(size(r)) :: return_value
   integer :: j, n
   intent (in) :: r
   n = size(r)
   do j = 1, n
      call fmsp2m_interval(r(j), return_value(j))
   enddo
end function fm_interval_r1

function fm_interval_d1(d)     result (return_value)
   use fmvals
   implicit none
   double precision, dimension(:) :: d
   type (fm_interval), dimension(size(d)) :: return_value
   integer :: j, n
   intent (in) :: d
   n = size(d)
   do j = 1, n
      call fmdp2m_interval(d(j), return_value(j))
   enddo
end function fm_interval_d1

function fm_interval_z1(z)     result (return_value)
   use fmvals
   implicit none
   complex, dimension(:) :: z
   type (fm_interval), dimension(size(z)) :: return_value
   integer :: j, n
   intent (in) :: z
   n = size(z)
   do j = 1, n
      call fmsp2m_interval(real(z(j)), return_value(j))
   enddo
end function fm_interval_z1

function fm_interval_zd1(c)     result (return_value)
   use fmvals
   implicit none
   complex (kind(0.0d0)), dimension(:) :: c
   type (fm_interval), dimension(size(c)) :: return_value
   integer :: j, n
   intent (in) :: c
   n = size(c)
   do j = 1, n
      call fmdp2m_interval(real(c(j), kind(0.0d0)), return_value(j))
   enddo
end function fm_interval_zd1

function fm_interval_fm1(ma)     result (return_value)
   use fmvals
```

```
   implicit none
   type (fm_interval), dimension(:) :: ma
   type (fm_interval), dimension(size(ma)) :: return_value
   integer :: j, n
   intent (in) :: ma
   call fm_interval_undef_inp(ma)
   n = size(ma)
   do j = 1, n
      call fmeq_interval(ma(j), return_value(j))
   enddo
end function fm_interval_fm1

function fm_interval_fma1(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm), dimension(:) :: ma
   type (fm_interval), dimension(size(ma)) :: return_value
   integer :: j, n
   intent (in) :: ma
   call fm_undef_inp(ma)
   n = size(ma)
   do j = 1, n
      call fmeq(ma(j)%mfm, return_value(j)%left)
      call fmeq(ma(j)%mfm, return_value(j)%right)
   enddo
end function fm_interval_fma1

function fm_interval_im1(ma)     result (return_value)
   use fmvals
   implicit none
   type (im), dimension(:) :: ma
   type (fm_interval), dimension(size(ma)) :: return_value
   integer :: j, n
   intent (in) :: ma
   call fm_undef_inp(ma)
   n = size(ma)
   do j = 1, n
      call imi2fm_interval(ma(j)%mim, return_value(j))
   enddo
end function fm_interval_im1

function fm_interval_zm1(ma)     result (return_value)
   use fmvals
   implicit none
   type (zm), dimension(:) :: ma
   type (fm_interval), dimension(size(ma)) :: return_value
   integer :: j, n
   intent (in) :: ma
   call fm_undef_inp(ma)
   n = size(ma)
   do j = 1, n
      call zmreal_interval(ma(j)%mzm, return_value(j))
   enddo
end function fm_interval_zm1

function fm_interval_st1(st)     result (return_value)
   use fmvals
   implicit none
```

```
   character(*), dimension(:) :: st
   type (fm_interval), dimension(size(st)) :: return_value
   integer :: j, n
   intent (in) :: st
   n = size(st)
   do j = 1, n
      call fmst2m_interval(st(j), return_value(j))
   enddo
end function fm_interval_st1

function fm_interval_i2(ival)     result (return_value)
   use fmvals
   implicit none
   integer, dimension(:,:) :: ival
   type (fm_interval), dimension(size(ival, dim=1), size(ival, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: ival
   do j = 1, size(ival, dim=1)
      do k = 1, size(ival, dim=2)
         call fmi2m_interval(ival(j, k), return_value(j, k))
      enddo
   enddo
end function fm_interval_i2

function fm_interval_r2(r)     result (return_value)
   use fmvals
   implicit none
   real, dimension(:,:) :: r
   type (fm_interval), dimension(size(r, dim=1), size(r, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: r
   do j = 1, size(r, dim=1)
      do k = 1, size(r, dim=2)
         call fmsp2m_interval(r(j, k), return_value(j, k))
      enddo
   enddo
end function fm_interval_r2

function fm_interval_d2(d)     result (return_value)
   use fmvals
   implicit none
   double precision, dimension(:,:) :: d
   type (fm_interval), dimension(size(d, dim=1), size(d, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: d
   do j = 1, size(d, dim=1)
      do k = 1, size(d, dim=2)
         call fmdp2m_interval(d(j, k), return_value(j, k))
      enddo
   enddo
end function fm_interval_d2

function fm_interval_z2(z)     result (return_value)
   use fmvals
   implicit none
   complex, dimension(:,:) :: z
   type (fm_interval), dimension(size(z, dim=1), size(z, dim=2)) :: return_value
   integer :: j, k
```

```
   intent (in) :: z
   do j = 1, size(z, dim=1)
      do k = 1, size(z, dim=2)
         call fmsp2m_interval(real(z(j, k)), return_value(j, k))
      enddo
   enddo
end function fm_interval_z2

function fm_interval_zd2(c)     result (return_value)
   use fmvals
   implicit none
   complex (kind(0.0d0)), dimension(:,:) :: c
   type (fm_interval), dimension(size(c, dim=1), size(c, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: c
   do j = 1, size(c, dim=1)
      do k = 1, size(c, dim=2)
         call fmdp2m_interval(real(c(j, k), kind(0.0d0)), return_value(j, k))
      enddo
   enddo
end function fm_interval_zd2

function fm_interval_fm2(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval), dimension(:,:) :: ma
   type (fm_interval), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: ma
   call fm_interval_undef_inp(ma)
   do j = 1, size(ma, dim=1)
      do k = 1, size(ma, dim=2)
         call fmeq_interval(ma(j, k), return_value(j, k))
      enddo
   enddo
end function fm_interval_fm2

function fm_interval_fma2(ma)     result (return_value)
   use fmvals
   implicit none
   type (fm), dimension(:,:) :: ma
   type (fm_interval), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: ma
   call fm_undef_inp(ma)
   do j = 1, size(ma, dim=1)
      do k = 1, size(ma, dim=2)
         call fmeq(ma(j, k)%mfm, return_value(j, k)%left)
         call fmeq(ma(j, k)%mfm, return_value(j, k)%right)
      enddo
   enddo
end function fm_interval_fma2

function fm_interval_im2(ma)     result (return_value)
   use fmvals
   implicit none
   type (im), dimension(:,:) :: ma
   type (fm_interval), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
```

```
   integer :: j, k
   intent (in) :: ma
   call fm_undef_inp(ma)
   do j = 1, size(ma, dim=1)
      do k = 1, size(ma, dim=2)
         call imi2fm_interval(ma(j, k)%mim, return_value(j, k))
      enddo
   enddo
end function fm_interval_im2

function fm_interval_zm2(ma)     result (return_value)
   use fmvals
   implicit none
   type (zm), dimension(:,:) :: ma
   type (fm_interval), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: ma
   call fm_undef_inp(ma)
   do j = 1, size(ma, dim=1)
      do k = 1, size(ma, dim=2)
         call zmreal_interval(ma(j, k)%mzm, return_value(j, k))
      enddo
   enddo
end function fm_interval_zm2

function fm_interval_st2(st)     result (return_value)
   use fmvals
   implicit none
   character(*), dimension(:,:) :: st
   type (fm_interval), dimension(size(st, dim=1), size(st, dim=2)) :: return_value
   integer :: j, k
   intent (in) :: st
   do j = 1, size(st, dim=1)
      do k = 1, size(st, dim=2)
         call fmst2m_interval(st(j, k), return_value(j, k))
      enddo
   enddo
end function fm_interval_st2

function interval_fm_i(ival1, ival2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   integer :: ival1, ival2, iv1, iv2
   intent (in) :: ival1, ival2
   iv1 = min(ival1, ival2)
   iv2 = max(ival1, ival2)
   call fmi2m(iv1, return_value%left)
   call fmi2m(iv2, return_value%right)
end function interval_fm_i

function interval_fm_r(r1, r2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   real :: r1, r2, rv1, rv2
   intent (in) :: r1, r2
   rv1 = min(r1, r2)
```

```
   rv2 = max(r1, r2)
   call fmsp2m(rv1, return_value%left)
   call fmsp2m(rv2, return_value%right)
end function interval_fm_r

function interval_fm_d(d1, d2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   double precision :: d1, d2, dv1, dv2
   intent (in) :: d1, d2
   dv1 = min(d1, d2)
   dv2 = max(d1, d2)
   call fmdp2m(dv1, return_value%left)
   call fmdp2m(dv2, return_value%right)
end function interval_fm_d

function interval_fm_z(z1, z2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   complex :: z1, z2
   real :: rv1, rv2
   intent (in) :: z1, z2
   rv1 = min(real(z1), real(z2))
   rv2 = max(real(z1), real(z2))
   call fmsp2m(rv1, return_value%left)
   call fmsp2m(rv2, return_value%right)
end function interval_fm_z

function interval_fm_zd(c1, c2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   complex (kind(0.0d0)) :: c1, c2
   double precision :: dv1, dv2
   intent (in) :: c1, c2
   dv1 = min(real(c1, kind(0.0d0)), real(c2, kind(0.0d0)))
   dv2 = max(real(c1, kind(0.0d0)), real(c2, kind(0.0d0)))
   call fmdp2m(dv1, return_value%left)
   call fmdp2m(dv2, return_value%right)
end function interval_fm_zd

function interval_fm_fm(m1, m2)     result (return_value)
   use fmvals
   implicit none
   type (fm_interval) :: return_value
   type (fm) :: m1, m2
   intent (in) :: m1, m2
   type(multi), save :: mtlvfm, mulvfm
   call fm_undef_inp(m1)
   call fm_undef_inp(m2)
   call fmmin(m1%mfm, m2%mfm, mtlvfm)
   call fmmax(m1%mfm, m2%mfm, mulvfm)
   call fmeq(mtlvfm, return_value%left)
   call fmeq(mulvfm, return_value%right)
end function interval_fm_fm
```

```
   function interval_fm_im(m1, m2)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      type (im) :: m1, m2
      intent (in) :: m1, m2
      type(multi), save :: mtlvim, mulvim
      call fm_undef_inp(m1)
      call fm_undef_inp(m2)
      call immin(m1%mim, m2%mim, mtlvim)
      call immax(m1%mim, m2%mim, mulvim)
      call imi2fm(mtlvim, return_value%left)
      call imi2fm(mulvim, return_value%right)
   end function interval_fm_im

   function interval_fm_zm(m1, m2)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      type (zm) :: m1, m2
      intent (in) :: m1, m2
      type(multi), save :: m1lvfm, m2lvfm
      call fm_undef_inp(m1)
      call fm_undef_inp(m2)
      call zmreal(m1%mzm, m1lvfm)
      call zmreal(m2%mzm, m2lvfm)
      call fmmin(m1lvfm, m2lvfm, return_value%left)
      call fmmax(m1lvfm, m2lvfm, return_value%right)
   end function interval_fm_zm

   function interval_fm_st(s1, s2)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: return_value
      character(*) :: s1, s2
      intent (in) :: s1, s2
      type(multi), save :: m1lvfm, m2lvfm
      call fmst2m(s1, m1lvfm)
      call fmst2m(s2, m2lvfm)
      call fmmin(m1lvfm, m2lvfm, return_value%left)
      call fmmax(m1lvfm, m2lvfm, return_value%right)
   end function interval_fm_st

!                                                                left_endpoint

   function left_endpoint_interval_fm(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      type (fm) :: return_value
      intent (in) :: ma
      call fmeq(ma%left, return_value%mfm)
   end function left_endpoint_interval_fm

!                                                               right_endpoint

   function right_endpoint_interval_fm(ma)     result (return_value)
      use fmvals
```

```
      implicit none
      type (fm_interval) :: ma
      type (fm) :: return_value
      intent (in) :: ma
      call fmeq(ma%right, return_value%mfm)
   end function right_endpoint_interval_fm

!                                                                to_fm

   function fm_fm_interval(ma)     result (return_value)

!  When converting an interval to a non-interval value, use the midpoint.

      use fmvals
      implicit none
      type (fm_interval) :: ma
      type (fm) :: return_value
      intent (in) :: ma
      type(multi), save :: mtlvfm
      call fm_interval_undef_inp(ma)
      call fmsub(ma%right, ma%left, mtlvfm)
      call fmdivi_r1(mtlvfm, 2)
      call fmadd(ma%left, mtlvfm, return_value%mfm)
   end function fm_fm_interval

   function fm_fm_interval1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      type (fm), dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      type(multi), save :: mtlvfm
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmsub(ma(j)%right, ma(j)%left, mtlvfm)
         call fmdivi_r1(mtlvfm, 2)
         call fmadd(ma(j)%left, mtlvfm, return_value(j)%mfm)
      enddo
   end function fm_fm_interval1

   function fm_fm_interval2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      type (fm), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      type(multi), save :: mtlvfm
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmsub(ma(j, k)%right, ma(j, k)%left, mtlvfm)
            call fmdivi_r1(mtlvfm, 2)
            call fmadd(ma(j, k)%left, mtlvfm, return_value(j, k)%mfm)
         enddo
      enddo
```

```
   end function fm_fm_interval2

!                                                                 to_im

   function im_fm_interval(ma)     result (return_value)
      use fmvals
      implicit none
      type (im) :: return_value
      type (fm_interval) :: ma
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call imfm2i_interval(ma, return_value%mim)
   end function im_fm_interval

   function im_fm_interval1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      type (im), dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call imfm2i_interval(ma(j), return_value(j)%mim)
      enddo
   end function im_fm_interval1

   function im_fm_interval2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      type (im), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call imfm2i_interval(ma(j, k), return_value(j, k)%mim)
         enddo
      enddo
   end function im_fm_interval2

!                                                                 to_zm

   function zm_fm_interval(ma)     result (return_value)
      use fmvals
      implicit none
      type (zm) :: return_value
      type (fm_interval) :: ma
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmi2m_interval(0, mufm_i)
      call zmcmpx_interval(ma, mufm_i, return_value)
   end function zm_fm_interval

   function zm_fm_interval1(ma)     result (return_value)
      use fmvals
```

```
      implicit none
      type (fm_interval), dimension(:) :: ma
      type (zm), dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      call fmi2m_interval(0, mufm_i)
      do j = 1, n
         call zmcmpx_interval(ma(j), mufm_i, return_value(j))
      enddo
   end function zm_fm_interval1

   function zm_fm_interval2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      type (zm), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmi2m_interval(0, mufm_i)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call zmcmpx_interval(ma(j, k), mufm_i, return_value(j, k))
         enddo
      enddo
   end function zm_fm_interval2

!                                                                to_int

   function fm_interval_2int(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      integer :: return_value
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmm2i_interval(ma, return_value)
   end function fm_interval_2int

   function fm_interval_2int1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      integer, dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmm2i_interval(ma(j), return_value(j))
      enddo
   end function fm_interval_2int1

   function fm_interval_2int2(ma)     result (return_value)
      use fmvals
      implicit none
```

```
      type (fm_interval), dimension(:,:) :: ma
      integer, dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmm2i_interval(ma(j, k), return_value(j, k))
         enddo
      enddo
   end function fm_interval_2int2

!                                                                to_sp

   function fm_interval_2sp(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      real :: return_value
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmm2sp_interval(ma, return_value)
   end function fm_interval_2sp

   function fm_interval_2sp1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      real, dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmm2sp_interval(ma(j), return_value(j))
      enddo
   end function fm_interval_2sp1

   function fm_interval_2sp2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      real, dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmm2sp_interval(ma(j, k), return_value(j, k))
         enddo
      enddo
   end function fm_interval_2sp2

!                                                                to_dp

   function fm_interval_2dp(ma)     result (return_value)
      use fmvals
      implicit none
```

```
      type (fm_interval) :: ma
      double precision :: return_value
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmm2dp_interval(ma, return_value)
   end function fm_interval_2dp

   function fm_interval_2dp1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      double precision, dimension(size(ma)) :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmm2dp_interval(ma(j), return_value(j))
      enddo
   end function fm_interval_2dp1

   function fm_interval_2dp2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      double precision, dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmm2dp_interval(ma(j, k), return_value(j, k))
         enddo
      enddo
   end function fm_interval_2dp2

!                                                                to_spz

   function fm_interval_2spz(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      complex :: return_value
      real :: r
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmm2sp_interval(ma, r)
      return_value = cmplx( r , 0.0 )
   end function fm_interval_2spz

   function fm_interval_2spz1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      complex, dimension(size(ma)) :: return_value
      integer :: j, n
      real :: r
      intent (in) :: ma
```

```
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmm2sp_interval(ma(j), r)
         return_value(j) = cmplx( r , 0.0 )
      enddo
   end function fm_interval_2spz1

   function fm_interval_2spz2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      complex, dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      real :: r
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmm2sp_interval(ma(j, k), r)
            return_value(j, k) = cmplx( r , 0.0 )
         enddo
      enddo
   end function fm_interval_2spz2

!                                                                to_dpz

   function fm_interval_2dpz(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      complex (kind(0.0d0)) :: return_value
      double precision :: d
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      call fmm2dp_interval(ma, d)
      return_value = cmplx( d , 0.0d0 , kind(0.0d0) )
   end function fm_interval_2dpz

   function fm_interval_2dpz1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      complex (kind(0.0d0)), dimension(size(ma)) :: return_value
      integer :: j, n
      double precision :: d
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      do j = 1, n
         call fmm2dp_interval(ma(j), d)
         return_value(j) = cmplx( d , 0.0d0 , kind(0.0d0) )
      enddo
   end function fm_interval_2dpz1

   function fm_interval_2dpz2(ma)     result (return_value)
      use fmvals
      implicit none
```

```
      type (fm_interval), dimension(:,:) :: ma
      complex (kind(0.0d0)), dimension(size(ma, dim=1), size(ma, dim=2)) :: return_value
      integer :: j, k
      double precision :: d
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmm2dp_interval(ma(j, k), d)
            return_value(j, k) = cmplx( d , 0.0d0 , kind(0.0d0) )
         enddo
      enddo
   end function fm_interval_2dpz2

!                                                          is_overflow

   function fm_interval_is_overflow(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      logical :: return_value
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      return_value = .false.
      if (ma%left%mp(2) == mexpov) return_value = .true.
      if (ma%right%mp(2) == mexpov) return_value = .true.
   end function fm_interval_is_overflow

   function fm_interval_is_overflow1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      logical :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      return_value = .false.
      do j = 1, n
         if (ma(j)%left%mp(2) == mexpov) return_value = .true.
         if (ma(j)%right%mp(2) == mexpov) return_value = .true.
      enddo
   end function fm_interval_is_overflow1

   function fm_interval_is_overflow2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      logical :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      return_value = .false.
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            if (ma(j, k)%left%mp(2) == mexpov) return_value = .true.
            if (ma(j, k)%right%mp(2) == mexpov) return_value = .true.
         enddo
```

```
      enddo
   end function fm_interval_is_overflow2

!                                                          is_underflow

   function fm_interval_is_underflow(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
      logical :: return_value
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      return_value = .false.
      if (ma%left%mp(2) == mexpun) return_value = .true.
      if (ma%right%mp(2) == mexpun) return_value = .true.
   end function fm_interval_is_underflow

!  The integer versions are included for completeness, but type (im) numbers can't underflow.

   function fm_interval_is_underflow1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:) :: ma
      logical :: return_value
      integer :: j, n
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      n = size(ma)
      return_value = .false.
      do j = 1, n
         if (ma(j)%left%mp(2) == mexpun) return_value = .true.
         if (ma(j)%right%mp(2) == mexpun) return_value = .true.
      enddo
   end function fm_interval_is_underflow1

   function fm_interval_is_underflow2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval), dimension(:,:) :: ma
      logical :: return_value
      integer :: j, k
      intent (in) :: ma
      call fm_interval_undef_inp(ma)
      return_value = .false.
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            if (ma(j, k)%left%mp(2) == mexpun) return_value = .true.
            if (ma(j, k)%right%mp(2) == mexpun) return_value = .true.
         enddo
      enddo
   end function fm_interval_is_underflow2

!                                                          is_unknown

   function fm_interval_is_unknown(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_interval) :: ma
```