```fortran
 module fm_rational_arithmetic_1


!  FM_rational 1.4                         David M. Smith                          Rational Arithmetic

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

!  Each FM rational number a/b consists of two values, with a and b being type(im) integer multiple
!  precision numbers.  Negative rationals are represented with a being negative.

!  This module supports assignment, arithmetic, comparison, and functions involving fm_rational
!  numbers.

!  Mixed-mode operations, such as adding fm_rational to IM or machine integer types, are supported.
!  In general, operations where both the arguments and results are mathematically rational (machine
!  precision integers, type(im), or type (fm_rational)) are supported, such as a = 1, a = b - 3, or
!  a = b / x_im, where a and b are fm_rational, and x_im is type IM.

!  Array operations are also supported, so a, b, and x_im could be 1- or 2-dimensional arrays in the
!  examples above.

!  Mixed-mode comparisons are also supported, as with if (a == 1), if (a <= b - 3), or
!  if (a > b / x_im).

!  to_fm_rational is a function for creating a number of type fm_rational.
!  This function can have one argument, for the common case of creating a rational number with an
!  integer value.  For to_fm_rational(a), a can be a machine integer or array of integers, a type
!  IM value or array, or a character string.

!  There is also a two argument form, to_fm_rational(a,b), that can be used to create the fraction
!  a/b when a and b are both machine precision integers, type(im) multiple precision integers, or
!  character strings representing integers.

!  The one argument character form can be used with a single input string having both parts of the
!  fraction present and separated by '/', as in to_fm_rational(' 41 / 314 ').  This might be more
!  readable than the equivalent forms to_fm_rational( 41, 314 ) or to_fm_rational( '41', '314' ).

!  The to_fm function from the basic FM package has been extended to convert a type fm_rational to
!  a type FM number.  The result is an approximation accurate to the current FM precision.

!  rational_approx(a, digits) is a function that converts an FM number a to a rational approximation
!  of type fm_rational that has no more than digits decimal digits in the top and bottom.
!  Ex:  a = pi, digits = 2   returns the fm_rational result      22 /      7
!       a = pi, digits = 3   returns the fm_rational result     355 /    113
!       a = pi, digits = 6   returns the fm_rational result  833719 / 265381
!  The rational result usually approximates a to about 2*digits significant digits, so
!  digits should not be more than about half the precision carried for a.
!  Ex:  833719 / 265381 = 3.14159265358107777120441930 6..., and agrees with pi to about 11 s.d.


!  The standard Fortran functions that are available for fm_rational arguments are the ones that
!  give exact rational results.  So if a and b are type fm_rational variables, a**b, exp(a), etc.,
!  are not provided since the results are not generally exact rational numbers.
```

```
!  But int(a), floor(a), max(a,b), mod(a,b), etc., do give rational result and are provided.

!  AVAILABLE OPERATONS:

!     =
!     +
!     -
!     *
!     /
!     **                    a ** j is provided for fm_rational a and machine integer j.
!     ==
!     /=
!     <
!     <=
!     >
!     >=
!     abs(a)
!     ceiling(a)
!     dim(a,b)              Positive difference.  Returns a - b if a > b, zero if not.
!     floor(a)
!     int(a)
!     is_unknown(a)         Returns true if a is unknown.
!     max(a,b,...)          Can have from 2 to 10 arguments.
!     min(a,b,...)          Can have from 2 to 10 arguments.
!     mod(a,b)              Result is a - int(a/b) * b
!     modulo(a,b)           Result is a - floor(a/b) * b
!     nint(a)

!  Array operations for functions.

!     abs, ceiling, floor, int, and nint can each have a 1- or 2-dimensional array argument.
!     They will return a vector or matrix with the function applied to each element of the
!     argument.

!     dim, mod, modulo work the same way, but the two array arguments for these functions must
!     have the same size and shape.

!     is_unknown is a logical function that can be used with an array argument but does not return
!     an array result.  It returns "true" if any element of the input array is fm's special unknown
!     value, which comes from undefined operations such as dividing by zero.

!  Functions that operate only on arrays.

!     dot_product(x,y)      Dot product of rank 1 vectors.
!     matmul(x,y)           Matrix multiplication of arrays
!                           Cases for valid argument shapes:
!                           (1)  (n,m) * (m,k) --> (n,k)
!                           (2)    (m) * (m,k) --> (k)
!                           (3)  (n,m) * (m)   --> (n)
!     maxval(x)             Maximum value in the array
!     minval(x)             Minimum value in the array
!     product(x)            Product of all values in the array
!     sum(x)                Sum of all values in the array
!     transpose(x)          Matrix transposition.  If x is a rank 2 array with shape (n,m), then
!                           y = transpose(x) has shape (m,n) with y(i,j) = x(j,i).

      use fmzm
```

```fortran
     type fm_rational
          type(multi) :: numerator
          type(multi) :: denominator
     end type

!              Work variables for derived type operations.

     type (im), save :: r_1, r_2, r_3, r_4, r_5, r_6
     type (fm), save :: f_1, f_2
     type (fm_rational), save :: mt_rm, mu_rm
     integer, save :: rational_exp_max = 0, rational_skip_max = 100
     logical, save :: skip_gcd = .false.

    interface to_fm_rational
       module procedure fm_rational_i
       module procedure fm_rational_ii
       module procedure fm_rational_i1
       module procedure fm_rational_i2
       module procedure fm_rational_im
       module procedure fm_rational_imim
       module procedure fm_rational_im1
       module procedure fm_rational_im2
       module procedure fm_rational_st
       module procedure fm_rational_stst
    end interface

    interface fm_undef_inp
       module procedure fm_undef_inp_rational_rm0
       module procedure fm_undef_inp_rational_rm1
       module procedure fm_undef_inp_rational_rm2
    end interface

    interface rational_numerator
       module procedure rational_numerator_im
    end interface

    interface rational_denominator
       module procedure rational_denominator_im
    end interface

    interface assignment (=)
        module procedure fmeq_rational_rmrm
        module procedure fmeq_rational_rmi
        module procedure fmeq_rational_rmim

        module procedure fmeq_rational_rm1rm
        module procedure fmeq_rational_rm1rm1
        module procedure fmeq_rational_rm1i
        module procedure fmeq_rational_rm1i1
        module procedure fmeq_rational_rm1im
        module procedure fmeq_rational_rm1im1

        module procedure fmeq_rational_rm2rm
        module procedure fmeq_rational_rm2rm2
        module procedure fmeq_rational_rm2i
        module procedure fmeq_rational_rm2i2
        module procedure fmeq_rational_rm2im
        module procedure fmeq_rational_rm2im2
```

```
   end interface

   contains


!                                                      to_fm_rational

   function fm_rational_i(top)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: return_value
      integer :: top, n1, n2
      intent (in) :: top
      if (top == 0) then
          call imst2m('0', return_value%numerator)
          call imst2m('1', return_value%denominator)
          return
      endif
      n1 = abs(top)
      n2 = 1
      if (top > 0) then
          call imi2m(n1, return_value%numerator)
      else
          call imi2m(-n1, return_value%numerator)
      endif
      call imi2m(n2, return_value%denominator)
      call fm_max_exp_rm(return_value)
   end function fm_rational_i

   function fm_rational_ii(top, bot)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: return_value
      integer :: top, bot, n1, n2
      intent (in) :: top, bot
      if (bot == 0) then
          call imunknown(return_value%numerator)
          call imunknown(return_value%denominator)
          return
      endif
      if (top == 0) then
          call imst2m('0', return_value%numerator)
          call imst2m('1', return_value%denominator)
          return
      endif
      n1 = abs(top)
      n2 = abs(bot)
      call fmgcdi(n1, n2)
      if ((top > 0 .and. bot > 0) .or. (top < 0 .and. bot < 0)) then
          call imi2m(n1, return_value%numerator)
      else
          call imi2m(-n1, return_value%numerator)
      endif
      call imi2m(n2, return_value%denominator)
      call fm_max_exp_rm(return_value)
   end function fm_rational_ii

   function fm_rational_i1(ival)     result (return_value)
```

```fortran
   use fmvals
   implicit none
   integer, dimension(:) :: ival
   intent (in) :: ival
   type (fm_rational), dimension(size(ival)) :: return_value
   integer :: j
   do j = 1, size(ival)
      call imi2m(ival(j), return_value(j)%numerator)
      call imi2m(1, return_value(j)%denominator)
      call fm_max_exp_rm(return_value(j))
   enddo
end function fm_rational_i1

function fm_rational_i2(ival)     result (return_value)
   use fmvals
   implicit none
   integer, dimension(:,:) :: ival
   intent (in) :: ival
   type (fm_rational), dimension(size(ival, dim=1), size(ival, dim=2)) :: return_value
   integer :: j, k
   do j = 1, size(ival, dim=1)
      do k = 1, size(ival, dim=2)
         call imi2m(ival(j, k), return_value(j, k)%numerator)
         call imi2m(1, return_value(j, k)%denominator)
         call fm_max_exp_rm(return_value(j, k))
      enddo
   enddo
end function fm_rational_i2

function fm_rational_im(top)     result (return_value)
   use fmvals
   implicit none
   type (fm_rational) :: return_value
   integer :: r_sign
   type (im) :: top
   intent (in) :: top
   call fm_undef_inp(top)
   call imeq(top%mim, r_1%mim)
   call imi2m(1, r_2%mim)
   if (is_unknown(r_1) .or. is_overflow(r_1)) then
       call imunknown(return_value%numerator)
       call imunknown(return_value%denominator)
       return
   endif
   if (r_1 == 0) then
       call imst2m('0', return_value%numerator)
       call imst2m('1', return_value%denominator)
       return
   endif
   r_sign = 1
   if (r_1 < 0) then
       r_sign = -1
   endif
   call im_abs(r_1, r_4)
   call im_abs(r_2, r_5)
   call fm_max_exp_im(r_4, r_5)
   if (skip_gcd .and. max(r_4%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
       if (r_sign == -1) r_4%mim%mp(1) = -1
```

```
         call imeq(r_4%mim, return_value%numerator)
         call imeq(r_5%mim, return_value%denominator)
     else
         call im_gcd(r_4, r_5, r_3)
         call im_div(r_4, r_3, r_1)
         call im_div(r_5, r_3, r_2)
         if (r_sign == -1) r_1%mim%mp(1) = -1
         call imeq(r_1%mim, return_value%numerator)
         call imeq(r_2%mim, return_value%denominator)
     endif
  end function fm_rational_im

  function fm_rational_imim(top, bot)     result (return_value)
     use fmvals
     implicit none
     type (fm_rational) :: return_value
     integer :: r_sign
     type (im) :: top, bot
     intent (in) :: top, bot
     call fm_undef_inp(top)
     call fm_undef_inp(bot)
     call imeq(top%mim, r_1%mim)
     call imeq(bot%mim, r_2%mim)
     if (r_2 == 0 .or. is_unknown(r_1) .or. is_overflow(r_1) .or.  &
                       is_unknown(r_2) .or. is_overflow(r_2) ) then
         call imunknown(return_value%numerator)
         call imunknown(return_value%denominator)
         return
     endif
     if (r_1 == 0) then
         call imst2m('0', return_value%numerator)
         call imst2m('1', return_value%denominator)
         return
     endif
     r_sign = 1
     if ((r_1 > 0 .and. r_2 < 0) .or. (r_1 < 0 .and. r_2 > 0)) then
         r_sign = -1
     endif
     call im_abs(r_1, r_4)
     call im_abs(r_2, r_5)
     call fm_max_exp_im(r_4, r_5)
     if (skip_gcd .and. max(r_4%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
         if (r_sign == -1) r_4%mim%mp(1) = -1
         call imeq(r_4%mim, return_value%numerator)
         call imeq(r_5%mim, return_value%denominator)
     else
         call im_gcd(r_4, r_5, r_3)
         call im_div(r_4, r_3, r_1)
         call im_div(r_5, r_3, r_2)
         if (r_sign == -1) r_1%mim%mp(1) = -1
         call imeq(r_1%mim, return_value%numerator)
         call imeq(r_2%mim, return_value%denominator)
     endif
  end function fm_rational_imim

  function fm_rational_im1(top)     result (return_value)
     use fmvals
     implicit none
```

```fortran
   type (im), dimension(:) :: top
   intent (in) :: top
   type (fm_rational), dimension(size(top)) :: return_value
   integer :: j
   do j = 1, size(top)
      call imeq(top(j)%mim, return_value(j)%numerator)
      call imi2m(1, return_value(j)%denominator)
      call fm_max_exp_rm(return_value(j))
   enddo
end function fm_rational_im1

function fm_rational_im2(top)     result (return_value)
   use fmvals
   implicit none
   type (im), dimension(:,:) :: top
   intent (in) :: top
   type (fm_rational), dimension(size(top, dim=1), size(top, dim=2)) :: return_value
   integer :: j, k
   do j = 1, size(top, dim=1)
      do k = 1, size(top, dim=2)
         call imeq(top(j, k)%mim, return_value(j, k)%numerator)
         call imi2m(1, return_value(j, k)%denominator)
         call fm_max_exp_rm(return_value(j, k))
      enddo
   enddo
end function fm_rational_im2

function fm_rational_st(top)     result (return_value)
   use fmvals
   implicit none
   type (fm_rational) :: return_value
   integer :: k, r_sign
   character(*) :: top
   intent (in) :: top
   k = index(top, '/')
   if (k > 0) then
       call imst2m(top(1:k-1), r_1%mim)
       call imst2m(top(k+1:len(top)), r_2%mim)
   else
       call imst2m(top, r_1%mim)
       call imi2m(1, r_2%mim)
   endif
   if (r_2 == 0 .or. is_unknown(r_1) .or. is_overflow(r_1) .or.  &
                    is_unknown(r_2) .or. is_overflow(r_2) ) then
       call imunknown(return_value%numerator)
       call imunknown(return_value%denominator)
       return
   endif
   if (r_1 == 0) then
       call imst2m('0', return_value%numerator)
       call imst2m('1', return_value%denominator)
       return
   endif
   r_sign = 1
   if ((r_1 > 0 .and. r_2 < 0) .or. (r_1 < 0 .and. r_2 > 0)) then
       r_sign = -1
   endif
   call im_abs(r_1, r_4)
```

```
      call im_abs(r_2, r_5)
      call fm_max_exp_im(r_4, r_5)
      if (skip_gcd .and. max(r_4%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
          if (r_sign == -1) r_4%mim%mp(1) = -1
          call imeq(r_4%mim, return_value%numerator)
          call imeq(r_5%mim, return_value%denominator)
      else
          call im_gcd(r_4, r_5, r_3)
          call im_div(r_4, r_3, r_1)
          call im_div(r_5, r_3, r_2)
          if (r_sign == -1) r_1%mim%mp(1) = -1
          call imeq(r_1%mim, return_value%numerator)
          call imeq(r_2%mim, return_value%denominator)
      endif
   end function fm_rational_st

   function fm_rational_stst(top, bot)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: return_value
      integer :: r_sign
      character(*) :: top, bot
      intent (in) :: top, bot
      call imst2m(top, r_1%mim)
      call imst2m(bot, r_2%mim)
      if (r_2 == 0 .or. is_unknown(r_1) .or. is_overflow(r_1) .or.  &
                        is_unknown(r_2) .or. is_overflow(r_2) ) then
          call imunknown(return_value%numerator)
          call imunknown(return_value%denominator)
          return
      endif
      if (r_1 == 0) then
          call imst2m('0', return_value%numerator)
          call imst2m('1', return_value%denominator)
          return
      endif
      r_sign = 1
      if ((r_1 > 0 .and. r_2 < 0) .or. (r_1 < 0 .and. r_2 > 0)) then
          r_sign = -1
      endif
      call im_abs(r_1, r_4)
      call im_abs(r_2, r_5)
      call fm_max_exp_im(r_4, r_5)
      if (skip_gcd .and. max(r_4%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
          if (r_sign == -1) r_4%mim%mp(1) = -1
          call imeq(r_4%mim, return_value%numerator)
          call imeq(r_5%mim, return_value%denominator)
      else
          call im_gcd(r_4, r_5, r_3)
          call im_div(r_4, r_3, r_1)
          call im_div(r_5, r_3, r_2)
          if (r_sign == -1) r_1%mim%mp(1) = -1
          call imeq(r_1%mim, return_value%numerator)
          call imeq(r_2%mim, return_value%denominator)
      endif
   end function fm_rational_stst

!                                                               rational_numerator
```

```fortran
   function rational_numerator_im(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      type (im) :: return_value
      intent (in) :: ma
      call imeq(ma%numerator, return_value%mim)
   end function rational_numerator_im

!                                                                  rational_denominator

   function rational_denominator_im(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      type (im) :: return_value
      intent (in) :: ma
      call imeq(ma%denominator, return_value%mim)
   end function rational_denominator_im

   subroutine fm_max_exp_rm(ma)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      intent (in) :: ma
      integer :: nt, nb
      nt = ma%numerator%mp(2)
      nb = ma%denominator%mp(2)
      if (nt < mexpov .and. nt > rational_exp_max) rational_exp_max = nt
      if (nb < mexpov .and. nb > rational_exp_max) rational_exp_max = nb
      end subroutine fm_max_exp_rm

   subroutine fm_max_exp_im(ma, mb)
      use fmvals
      implicit none
      type (im) :: ma, mb
      intent (in) :: ma, mb
      integer :: nt
      nt = ma%mim%mp(2)
      if (nt < mexpov .and. nt > rational_exp_max) rational_exp_max = nt
      nt = mb%mim%mp(2)
      if (nt < mexpov .and. nt > rational_exp_max) rational_exp_max = nt
      end subroutine fm_max_exp_im

!                                                                  fm_print_rational

   subroutine fm_print_rational(ma)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      character(100) :: st1, st2
      character(203) :: str
      intent (in) :: ma
      integer :: j, kpt

!             If the top and bottom integers can be printed on one line, as  12 / 7
!             in fewer than kswide characters, do it.  Otherwise call imprint twice.
```

```fortran
   call imabs(ma%numerator, r_1%mim)
   call imabs(ma%denominator, r_2%mim)
   call immpy(r_1%mim, r_2%mim, r_3%mim)

   if (to_im(10)**(kswide-11) > r_3 .and. r_1 < to_im('1e+99') .and. r_2 < to_im('1e+99')) then
       call imform('i100', ma%numerator, st1)
       call imform('i100', ma%denominator, st2)
       str = ' '
       kpt = 0
       do j = 1, 100
          if (st1(j:j) /= ' ') then
              kpt = kpt + 1
              str(kpt:kpt) = st1(j:j)
          endif
       enddo
       str(kpt+1:kpt+3) = ' / '
       kpt = kpt + 3
       do j = 1, 100
          if (st2(j:j) /= ' ') then
              kpt = kpt + 1
              str(kpt:kpt) = st2(j:j)
          endif
       enddo
       if (ma%numerator%mp(1) < 0) then
           write (kw, "(6x, a)") str(1:kpt)
       else
           write (kw, "(7x, a)") str(1:kpt)
       endif
   else
       call imprint(ma%numerator)
       write (kw, "(a)") '    /'
       call imprint(ma%denominator)
   endif
end subroutine fm_print_rational


subroutine fm_undef_inp_rational_rm0(ma)
   use fmvals
   implicit none
   type (fm_rational) :: ma
   intent (in) :: ma
   if (.not. allocated(ma%numerator%mp)) call fm_input_error
   if (.not. allocated(ma%denominator%mp)) call fm_input_error
end subroutine fm_undef_inp_rational_rm0

subroutine fm_undef_inp_rational_rm1(ma)
   use fmvals
   implicit none
   type (fm_rational), dimension(:) :: ma
   integer :: j
   intent (in) :: ma
   do j = 1, size(ma)
      if (.not. allocated(ma(j)%numerator%mp)) call fm_input_error1(j)
      if (.not. allocated(ma(j)%denominator%mp)) call fm_input_error1(j)
   enddo
end subroutine fm_undef_inp_rational_rm1
```

```fortran
   subroutine fm_undef_inp_rational_rm2(ma)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      integer :: j, k
      intent (in) :: ma
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            if (.not. allocated(ma(j, k)%numerator%mp)) call fm_input_error2(j, k)
            if (.not. allocated(ma(j, k)%denominator%mp)) call fm_input_error2(j, k)
         enddo
      enddo
   end subroutine fm_undef_inp_rational_rm2

   subroutine fmeq_rational(ma, mb)
      use fmvals
      implicit none
      type (fm_rational) :: ma, mb
      intent (in) :: ma
      intent (inout) :: mb
      call imeq(ma%numerator, mb%numerator)
      call imeq(ma%denominator, mb%denominator)
   end subroutine fmeq_rational

!                                                                                         =

   subroutine fmeq_rational_rmrm(ma, mb)
      use fmvals
      implicit none
      type (fm_rational) :: ma, mb
      intent (inout) :: ma
      intent (in) :: mb
      call fm_undef_inp(mb)
      call fmeq_rational(mb, ma)
      call fm_max_exp_rm(ma)
   end subroutine fmeq_rational_rmrm

   subroutine fmeq_rational_rmi(ma, ival)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      integer :: ival
      intent (inout) :: ma
      intent (in) :: ival
      call imi2m(ival, ma%numerator)
      call imi2m(1, ma%denominator)
      call fm_max_exp_rm(ma)
   end subroutine fmeq_rational_rmi

   subroutine fmeq_rational_rmim(ma, mb)
      use fmvals
      implicit none
      type (fm_rational) :: ma
      type (im) :: mb
      intent (inout) :: ma
      intent (in) :: mb
      call fm_undef_inp(mb)
      call imeq(mb%mim, ma%numerator)
```

```fortran
      call imi2m(1, ma%denominator)
      call fm_max_exp_rm(ma)
   end subroutine fmeq_rational_rmim

!              Array equal assignments for RM.

!              (1) rank 1  =  rank 0

   subroutine fmeq_rational_rm1i(ma, ival)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      integer :: ival, j
      intent (inout) :: ma
      intent (in) :: ival
      do j = 1, size(ma)
         call fmeq_rational_rmi(ma(j), ival)
      enddo
   end subroutine fmeq_rational_rm1i

   subroutine fmeq_rational_rm1rm(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      type (fm_rational) :: mb
      integer :: j
      intent (inout) :: ma
      intent (in) :: mb
      call fmeq_rational_rmrm(mt_rm, mb)
      do j = 1, size(ma)
         call fmeq_rational_rmrm(ma(j), mt_rm)
      enddo
   end subroutine fmeq_rational_rm1rm

   subroutine fmeq_rational_rm1im(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      type (im) :: mb
      integer :: j
      intent (inout) :: ma
      intent (in) :: mb
      call imeq(mb%mim, r_1%mim)
      do j = 1, size(ma)
         call fmeq_rational_rmim(ma(j), r_1)
      enddo
   end subroutine fmeq_rational_rm1im

!              (2) rank 1  =  rank 1

   subroutine fmeq_rational_rm1i1(ma, ival)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      integer, dimension(:) :: ival
      integer :: j
      intent (inout) :: ma
      intent (in) :: ival
```

```fortran
      if (size(ma) /= size(ival)) then
          call imunknown(mt_rm%numerator)
          call imunknown(mt_rm%denominator)
          do j = 1, size(ma)
             call imeq(mt_rm%numerator, ma(j)%numerator)
             call imeq(mt_rm%denominator, ma(j)%denominator)
          enddo
          return
      endif
      do j = 1, size(ma)
         call fmeq_rational_rmi(ma(j), ival(j))
      enddo
   end subroutine fmeq_rational_rm1i1

   subroutine fmeq_rational_rm1rm1(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      type (fm_rational), dimension(:) :: mb
      type (fm_rational), allocatable, dimension(:) :: temp
      integer :: j, n
      intent (inout) :: ma
      intent (in) :: mb
      if (size(ma) /= size(mb)) then
          call imunknown(mt_rm%numerator)
          call imunknown(mt_rm%denominator)
          do j = 1, size(ma)
             call imeq(mt_rm%numerator, ma(j)%numerator)
             call imeq(mt_rm%denominator, ma(j)%denominator)
          enddo
          return
      endif
      n = size(ma)

!             To avoid problems when lhs and rhs are overlapping parts of the same array, move mb
!             to a temporary array before re-defining any of ma.

      allocate(temp(n))
      do j = 1, n
         call imeq(mb(j)%numerator, temp(j)%numerator)
         call imeq(mb(j)%denominator, temp(j)%denominator)
      enddo
      do j = 1, n
         call fmeq_rational_rmrm(ma(j), temp(j))
      enddo
      deallocate(temp)
   end subroutine fmeq_rational_rm1rm1

   subroutine fmeq_rational_rm1im1(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      type (im), dimension(:) :: mb
      type (im), allocatable, dimension(:) :: temp
      integer :: j, n
      intent (inout) :: ma
      intent (in) :: mb
      if (size(ma) /= size(mb)) then
```

```
         call imunknown(mt_rm%numerator)
         call imunknown(mt_rm%denominator)
         do j = 1, size(ma)
            call imeq(mt_rm%numerator, ma(j)%numerator)
            call imeq(mt_rm%denominator, ma(j)%denominator)
         enddo
         return
      endif
      n = size(ma)
      allocate(temp(n))
      do j = 1, size(ma)
         call imeq(mb(j)%mim, temp(j)%mim)
      enddo
      do j = 1, size(ma)
         call fmeq_rational_rmim(ma(j), temp(j))
      enddo
      deallocate(temp)
   end subroutine fmeq_rational_rm1im1

!             (3) rank 2  =  rank 0

   subroutine fmeq_rational_rm2i(ma, ival)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      integer :: ival, j, k
      intent (inout) :: ma
      intent (in) :: ival
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmi(ma(j, k), ival)
         enddo
      enddo
   end subroutine fmeq_rational_rm2i

   subroutine fmeq_rational_rm2rm(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      type (fm_rational) :: mb
      integer :: j, k
      intent (inout) :: ma
      intent (in) :: mb
      call fmeq_rational_rmrm(mt_rm, mb)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmrm(ma(j, k), mt_rm)
         enddo
      enddo
   end subroutine fmeq_rational_rm2rm

   subroutine fmeq_rational_rm2im(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      type (im) :: mb
      integer :: j, k
      intent (inout) :: ma
```

```fortran
      intent (in) :: mb
      call imeq(mb%mim, r_1%mim)
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmim(ma(j, k), r_1)
         enddo
      enddo
   end subroutine fmeq_rational_rm2im

!              (4) rank 2  =  rank 2

   subroutine fmeq_rational_rm2i2(ma, ival)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      integer, dimension(:,:) :: ival
      integer :: j, k
      intent (inout) :: ma
      intent (in) :: ival
      if (size(ma, dim=1) /= size(ival, dim=1) .or. size(ma, dim=2) /= size(ival, dim=2)) then
          call imunknown(mt_rm%numerator)
          call imunknown(mt_rm%denominator)
          do j = 1, size(ma, dim=1)
             do k = 1, size(ma, dim=2)
                call imeq(mt_rm%numerator, ma(j, k)%numerator)
                call imeq(mt_rm%denominator, ma(j, k)%denominator)
             enddo
          enddo
          return
      endif
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmi(ma(j, k), ival(j, k))
         enddo
      enddo
   end subroutine fmeq_rational_rm2i2

   subroutine fmeq_rational_rm2rm2(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      type (fm_rational), dimension(:,:) :: mb
      type (fm_rational), allocatable, dimension(:,:) :: temp
      integer :: j, k
      intent (inout) :: ma
      intent (in) :: mb
      call fm_undef_inp(mb)
      if (size(ma, dim=1) /= size(mb, dim=1) .or. size(ma, dim=2) /= size(mb, dim=2)) then
          call imunknown(mt_rm%numerator)
          call imunknown(mt_rm%denominator)
          do j = 1, size(ma, dim=1)
             do k = 1, size(ma, dim=2)
                call imeq(mt_rm%numerator, ma(j, k)%numerator)
                call imeq(mt_rm%denominator, ma(j, k)%denominator)
             enddo
          enddo
          return
      endif
```

```
!              To avoid problems when lhs and rhs are overlapping parts of the same array, move mb
!              to a temporary array before re-defining any of ma.

      allocate(temp(size(ma, dim=1), size(ma, dim=2)))
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call imeq(mb(j, k)%numerator, temp(j, k)%numerator)
            call imeq(mb(j, k)%denominator, temp(j, k)%denominator)
         enddo
      enddo
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmrm(ma(j, k), temp(j, k))
         enddo
      enddo
      deallocate(temp)
   end subroutine fmeq_rational_rm2rm2

   subroutine fmeq_rational_rm2im2(ma, mb)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      type (im), dimension(:,:) :: mb
      type (im), allocatable, dimension(:,:) :: temp
      integer :: j, k
      intent (inout) :: ma
      intent (in) :: mb
      if (size(ma, dim=1) /= size(mb, dim=1) .or. size(ma, dim=2) /= size(mb, dim=2)) then
          call imunknown(mt_rm%numerator)
          call imunknown(mt_rm%denominator)
          do j = 1, size(ma, dim=1)
             do k = 1, size(ma, dim=2)
                call imeq(mt_rm%numerator, ma(j, k)%numerator)
                call imeq(mt_rm%denominator, ma(j, k)%denominator)
             enddo
          enddo
          return
      endif
      allocate(temp(size(ma, dim=1), size(ma, dim=2)))
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call imeq(mb(j, k)%mim, temp(j, k)%mim)
         enddo
      enddo
      do j = 1, size(ma, dim=1)
         do k = 1, size(ma, dim=2)
            call fmeq_rational_rmim(ma(j, k), temp(j, k))
         enddo
      enddo
      deallocate(temp)
   end subroutine fmeq_rational_rm2im2

   subroutine fmadd_rational_rmrm_0(ma, mb, mc)
      use fmvals
      implicit none
      type (fm_rational) :: ma, mb, mc
      intent (in) :: ma, mb
```

```fortran
   intent (inout) :: mc
   call fm_undef_inp(ma)
   call fm_undef_inp(mb)
   call imeq(ma%numerator,  r_1%mim)
   call imeq(ma%denominator, r_2%mim)
   call imeq(mb%numerator,  r_3%mim)
   call imeq(mb%denominator, r_4%mim)
   call im_mpy(r_1, r_4, r_5)
   call im_mpy(r_2, r_3, r_1)
   call im_add(r_1, r_5, r_3)
   call im_mpy(r_2, r_4, r_5)
   call fm_max_exp_im(r_3, r_5)
   if (skip_gcd .and. max(r_3%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
       call imeq(r_3%mim, mc%numerator)
       call imeq(r_5%mim, mc%denominator)
   else
       call im_gcd(r_3, r_5, r_1)
       call imdiv(r_3%mim, r_1%mim, mc%numerator)
       call imdiv(r_5%mim, r_1%mim, mc%denominator)
   endif
   if (mc%denominator%mp(1) < 0) then
       mc%denominator%mp(1) = 1
       mc%numerator%mp(1) = -mc%numerator%mp(1)
   endif
end subroutine fmadd_rational_rmrm_0

subroutine fmsub_rational_rmrm_0(ma, mb, mc)
   use fmvals
   implicit none
   type (fm_rational) :: ma, mb, mc
   intent (in) :: ma, mb
   intent (inout) :: mc
   call fm_undef_inp(ma)
   call fm_undef_inp(mb)
   call imeq(ma%numerator,  r_1%mim)
   call imeq(ma%denominator, r_2%mim)
   call imeq(mb%numerator,  r_3%mim)
   call imeq(mb%denominator, r_4%mim)
   call im_mpy(r_1, r_4, r_5)
   call im_mpy(r_2, r_3, r_1)
   call im_sub(r_5, r_1, r_3)
   call im_mpy(r_2, r_4, r_5)
   call fm_max_exp_im(r_3, r_5)
   if (skip_gcd .and. max(r_3%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
       call imeq(r_3%mim, mc%numerator)
       call imeq(r_5%mim, mc%denominator)
   else
       call im_gcd(r_3, r_5, r_1)
       call imdiv(r_3%mim, r_1%mim, mc%numerator)
       call imdiv(r_5%mim, r_1%mim, mc%denominator)
   endif
   if (mc%denominator%mp(1) < 0) then
       mc%denominator%mp(1) = 1
       mc%numerator%mp(1) = -mc%numerator%mp(1)
   endif
end subroutine fmsub_rational_rmrm_0

subroutine fmmpy_rational_rmrm_0(ma, mb, mc)
```

```
   use fmvals
   implicit none
   type (fm_rational) :: ma, mb, mc
   intent (in) :: ma, mb
   intent (inout) :: mc
   integer :: idiv
   call fm_undef_inp(ma)
   call fm_undef_inp(mb)
   call imeq(ma%numerator,   r_1%mim)
   call imeq(ma%denominator, r_2%mim)
   call imeq(mb%numerator,   r_3%mim)
   call imeq(mb%denominator, r_4%mim)
   call im_mpy(r_1, r_3, r_5)
   call im_mpy(r_2, r_4, r_3)
   call fm_max_exp_im(r_3, r_5)
   if (skip_gcd .and. max(r_3%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
       call imeq(r_5%mim, mc%numerator)
       call imeq(r_3%mim, mc%denominator)
   else
       call im_gcd(r_3, r_5, r_1)
       if (r_1%mim%mp(2) == 1 .and. r_1%mim%mp(3) == 1) then
           call imeq(r_5%mim, mc%numerator)
           call imeq(r_3%mim, mc%denominator)
       else if (r_1%mim%mp(2) == 1 .and. r_1%mim%mp(3) < mxbase) then
           idiv = r_1%mim%mp(3)
           call imdivi(r_5%mim, idiv, mc%numerator)
           call imdivi(r_3%mim, idiv, mc%denominator)
       else
           call imdiv(r_5%mim, r_1%mim, mc%numerator)
           call imdiv(r_3%mim, r_1%mim, mc%denominator)
       endif
   endif
   if (mc%denominator%mp(1) < 0) then
       mc%denominator%mp(1) = 1
       mc%numerator%mp(1) = -mc%numerator%mp(1)
   endif
end subroutine fmmpy_rational_rmrm_0

subroutine fmdiv_rational_rmrm_0(ma, mb, mc)
   use fmvals
   implicit none
   type (fm_rational) :: ma, mb, mc
   intent (in) :: ma, mb
   intent (inout) :: mc
   call fm_undef_inp(ma)
   call fm_undef_inp(mb)
   if (ma%numerator%mp(2) == munkno .or. ma%denominator%mp(2) == munkno .or.  &
       mb%numerator%mp(2) == munkno .or. mb%denominator%mp(2) == munkno .or.  &
       mb%numerator%mp(3) == 0 ) then
       call imunknown(mc%numerator)
       call imunknown(mc%denominator)
       return
   endif
   call imeq(ma%numerator,   r_1%mim)
   call imeq(ma%denominator, r_2%mim)
   call imeq(mb%numerator,   r_3%mim)
   call imeq(mb%denominator, r_4%mim)
   call im_mpy(r_1, r_4, r_5)
```

```fortran
      call im_mpy(r_2, r_3, r_4)
      call fm_max_exp_im(r_4, r_5)
      if (skip_gcd .and. max(r_4%mim%mp(2), r_5%mim%mp(2)) < rational_skip_max) then
          call imeq(r_5%mim, mc%numerator)
          call imeq(r_4%mim, mc%denominator)
      else
          call im_gcd(r_4, r_5, r_1)
          call imdiv(r_5%mim, r_1%mim, mc%numerator)
          call imdiv(r_4%mim, r_1%mim, mc%denominator)
      endif
      if (mc%denominator%mp(1) < 0) then
          mc%denominator%mp(1) = 1
          mc%numerator%mp(1) = -mc%numerator%mp(1)
      endif
   end subroutine fmdiv_rational_rmrm_0

 end module fm_rational_arithmetic_1


 module fm_rational_arithmetic_2
    use fm_rational_arithmetic_1

   interface operator (+)
       module procedure fmadd_rational_rm
       module procedure fmadd_rational_rm1
       module procedure fmadd_rational_rm2

       module procedure fmadd_rational_rmrm
       module procedure fmadd_rational_irm
       module procedure fmadd_rational_rmi
       module procedure fmadd_rational_imrm
       module procedure fmadd_rational_rmim

       module procedure fmadd_rational_rm1rm1
       module procedure fmadd_rational_rm1rm
       module procedure fmadd_rational_rmrm1
       module procedure fmadd_rational_i1rm1
       module procedure fmadd_rational_rm1i1
       module procedure fmadd_rational_i1rm
       module procedure fmadd_rational_rmi1
       module procedure fmadd_rational_irm1
       module procedure fmadd_rational_rm1i
       module procedure fmadd_rational_im1rm1
       module procedure fmadd_rational_im1rm
       module procedure fmadd_rational_imrm1
       module procedure fmadd_rational_rm1im1
       module procedure fmadd_rational_rm1im
       module procedure fmadd_rational_rmim1

       module procedure fmadd_rational_rm2rm2
       module procedure fmadd_rational_rm2rm
       module procedure fmadd_rational_rmrm2
       module procedure fmadd_rational_i2rm2
       module procedure fmadd_rational_i2rm
       module procedure fmadd_rational_irm2
       module procedure fmadd_rational_rm2i2
       module procedure fmadd_rational_rm2i
       module procedure fmadd_rational_rmi2
```

```fortran
      module procedure fmadd_rational_im2rm2
      module procedure fmadd_rational_im2rm
      module procedure fmadd_rational_imrm2
      module procedure fmadd_rational_rm2im2
      module procedure fmadd_rational_rm2im
      module procedure fmadd_rational_rmim2
   end interface

   contains

!                                                                           +


   function fmadd_rational_rm(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational) :: ma, return_value
      intent (in) :: ma
      call fm_undef_inp(ma)
      call fmeq_rational(ma, return_value)
      if (return_value%denominator%mp(1) < 0) then
          return_value%denominator%mp(1) = 1
          return_value%numerator%mp(1) = -return_value%numerator%mp(1)
      endif
   end function fmadd_rational_rm

   function fmadd_rational_rm1(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational), dimension(:) :: ma
      type (fm_rational), dimension(size(ma)) :: return_value
      integer :: j
      intent (in) :: ma
      call fm_undef_inp(ma)
      do j = 1, size(ma)
         call fmeq_rational(ma(j), return_value(j))
         if (return_value(j)%denominator%mp(1) < 0) then
             return_value(j)%denominator%mp(1) = 1
             return_value(j)%numerator%mp(1) =  &
             -return_value(j)%numerator%mp(1)
         endif
      enddo
   end function fmadd_rational_rm1

   function fmadd_rational_rm2(ma)     result (return_value)
      use fmvals
      implicit none
      type (fm_rational), dimension(:,:) :: ma
      type (fm_rational), 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_rational(ma(j, k), return_value(j, k))
            if (return_value(j, k)%denominator%mp(1) < 0) then
                return_value(j, k)%denominator%mp(1) = 1
                return_value(j, k)%numerator%mp(1) =  &
                -return_value(j, k)%numerator%mp(1)
```