```
   module fmvals

!     Version 1.4                       David M. Smith

!  These are the global and saved variables used by the FM package.
!  See the FM_User_Manual.txt file for further description of some of these variables.

!  They are initialized assuming the program will run on a 32-bit computer with variables in
!  fm.f95 having names beginning with 'm' being declared as having 64-bit representations
!  (double precision).

!  For a machine with a different architecture, or for setting the precision level to a different
!  value, call fmset(nprec) before doing any multiple precision operations.  fmset tries to
!  initialize the variables to the best values for the given machine.  To have the values chosen
!  by fmset written on unit kw, call fmvars.

!  Base and precision will be set to give slightly more than 50 decimal digits of precision, giving
!  the user 50 significant digits of precision along with several base ten guard digits.

!  mbase is set to 10**7.
!  jform1 and jform2 are set to es format displaying 50 significant digits.

!  The trace option is set off.
!  The mode for angles in trig functions is set to radians.
!  The rounding mode is set to symmetric rounding (to nearest).
!  Warning error message level is set to 1.
!  Cancellation error monitor is set off.
!  Screen width for output is set to 80 columns.
!  The exponent character for FM output is set to 'm'.
!  Debug error checking is set off.

!  kw, the unit number for all FM output, is set to 6.

      private aint, ceiling, digits, epsilon, huge, log, max, min, sqrt

      type multi
         real (kind(1.0d0)), allocatable :: mp(:)
      end type

      type fm
         type(multi) :: mfm
      end type

      type im
         type(multi) :: mim
      end type

      type zm
         type(multi) :: mzm(2)
      end type


      real (kind(1.0d0)), parameter :: m_two = 2
      double precision, parameter :: dp_two = 2
      integer, parameter :: i_two = 2
      real, parameter :: r_two = 2
```

```
!             kw is the unit number for standard output from the FM package.
!                This includes trace output and error messages.

      integer, save :: kw = 6

!             The min below is needed when m-variables have more precision than double,
!             as with 64-bit integer m-variables and 64-bit doubles (53-bit precision).

      real (kind(1.0d0)), parameter :: max_representable_m_var =  &
                           ( (m_two ** (min(digits(m_two), digits(dp_two))-1)) - 1 ) * 2 + 1

!             maxint should be set to a very large integer, possibly the largest representable
!                    integer for the current machine.  For most 32-bit machines, maxint is set
!                    to  2**53 - 1 = 9.007d+15  when double precision arithmetic is used for
!                    m-variables.  Using integer m-variables usually gives
!                    maxint = 2**31 - 1 = 2147483647.
!                    Setting maxint to a smaller number is ok, but this unnecessarily restricts
!                    the permissible range of mbase and mxexp.

      real (kind(1.0d0)), save :: maxint = max_representable_m_var

!             intmax is a large value close to the overflow threshold for integer variables.
!                    It is usually 2**31 - 1 for machines with 32-bit integer arithmetic.

      integer, save :: intmax = huge(i_two)

!             dpmax should be set to a value near the machine's double precision overflow threshold,
!                   so that dpmax and 1.0d0/dpmax are both representable in double precision.

      double precision, save :: dpmax = huge(dp_two)/5

!             spmax should be set to a value near the machine's single precision overflow threshold,
!                   so that 1.01*spmax and 1.0/spmax are both representable in single precision.

      real, save :: spmax = huge(r_two)/5

!             mxbase is the maximum value for mbase.

      real (kind(1.0d0)), parameter :: max_base = aint(sqrt(max_representable_m_var + 1.0d-9))

      real (kind(1.0d0)), save :: mxbase = max_base

!             mbase is the currently used base for arithmetic.

      real (kind(1.0d0)), parameter :: m_ten = 10

      real (kind(1.0d0)), save :: mbase = m_ten ** aint(log(max_base/4.0d0) / log(10.0d0))

!             ndig is the number of digits currently being carried.

      integer, save :: ndig = ceiling( 52.0d0 / aint(log(max_base/4.0d0)/log(10.0d0)) ) + 1

!             kflag is the flag for error conditions.

      integer, save :: kflag = 0

!             ntrace is the trace switch.  Default is no printing.
```

```
      integer, save :: ntrace = 0

!             lvltrc is the trace level.  Default is to trace only routines called directly
!                    by the user.

      integer, save :: lvltrc = 1

!             ncall is the call stack pointer.

      integer, save :: ncall = 0

!             raise_ndig is set to 1 when one FM routine calls another and the second one needs
!                        to use more guard digits.

      integer, save :: raise_ndig = 0

!             namest is the call stack.

      integer :: i_fmvals = 0
      character(9), save :: namest(0:50) = (/ ('MAIN     ' , i_fmvals = 0, 50) /)

!             Some constants that are often needed are stored with the maximum precision to which
!             they have been computed in the currently used base.  This speeds up the trig, log,
!             power, and exponential functions.

!             ndigpi is the number of digits available in the currently stored value of pi (mpisav).

      integer, save :: ndigpi = 0

!             mbspi is the value of mbase for the currently stored value of pi.

      real (kind(1.0d0)), save :: mbspi = 0

!             ndige is the number of digits available in the currently stored value of e (mesav).

      integer, save :: ndige = 0

!             mbse is the value of mbase for the currently stored value of e.

      real (kind(1.0d0)), save :: mbse = 0

!             ndiglb is the number of digits available in the currently stored value of ln(mbase)
!                    (mlbsav).

      integer, save :: ndiglb = 0

!             mbslb is the value of mbase for the currently stored value of ln(mbase).

      real (kind(1.0d0)), save :: mbslb = 0

!             ndigli is the number of digits available in the currently stored values of the four
!                    logarithms used by fmlni:  mln2, mln3, mln5, mln7.

      integer, save :: ndigli = 0

!             mbsli is the value of mbase for the currently stored values of mln2, mln3, mln5, mln7.

      real (kind(1.0d0)), save :: mbsli = 0
```

```
!             mxexp  is the current maximum exponent.
!             mxexp2 is the internal maximum exponent.  This is used to define the overflow and
!                    underflow thresholds.
!
!             These values are chosen so that FM routines can raise the overflow/underflow limit
!             temporarily while computing intermediate results.  mxexp2 satisfies these conditions:
!             1. exp(intmax) > mxbase**(mxexp2+1)
!             2. mxexp2 < maxint/20
!
!             The overflow threshold is mbase**(mxexp+1), and the underflow threshold is
!             mbase**(-mxexp-1).  This means the valid exponents in the first word of an FM
!             number can range from -mxexp to mxexp+1 (inclusive).

      real (kind(1.0d0)), parameter :: max_exponent = aint( min(                                &
                                       max(huge(intmax) / log(max_base+1.0d-9) , 117496405.0d0),  &
                                       max_representable_m_var / 20.0d0) )

      real (kind(1.0d0)), save :: mxexp = aint( max_exponent / 2.01d0 + 0.5d0 )

      real (kind(1.0d0)), save :: mxexp0 = aint( max_exponent / 2.01d0 + 0.5d0 )

      real (kind(1.0d0)), save :: mxexp2 = max_exponent

!             mexpun is the exponent used as a special symbol for underflowed results.

      real (kind(1.0d0)), save :: mexpun = aint( -max_exponent * 1.01d0 )

!             mexpov is the exponent used as a special symbol for overflowed results.

      real (kind(1.0d0)), save :: mexpov = aint( max_exponent * 1.01d0 )

!             munkno is the exponent used as a special symbol for unknown FM results
!                    (1/0, sqrt(-3.0), ...).

      real (kind(1.0d0)), save :: munkno = aint( max_exponent * 1.0201d0 )

!             runkno is returned from FM to real or double conversion routines when no valid result
!                    can be expressed in real or double precision.  On systems that provide a value
!                    for undefined results (e.g., Not a Number) setting runkno to that value is
!                    reasonable.  On other systems set it to a value that is likely to make any
!                    subsequent results obviously wrong that use it.  In either case a kflag = -4
!                    condition is also returned.

      real, save :: runkno = -1.01*(huge(r_two)/3.0)

!             iunkno is returned from FM to integer conversion routines when no valid result can be
!                    expressed as a one word integer.  kflag = -4 is also set.

      integer, save :: iunkno = -huge(i_two)/18

!             jform1 indicates the format used by fmout.

      integer, save :: jform1 = 1

!             jform2 indicates the number of digits used in fmout.

      integer, save :: jform2 = 50
```

```
!             krad = 1 indicates that trig functions use radians,
!                  = 0 means use degrees.

      integer, save :: krad = 1

!             kwarn = 0 indicates that no warning message is printed and execution continues when
!                       unknown or another exception is produced.
!                   = 1 means print a warning message and continue.
!                   = 2 means print a warning message and stop.

      integer, save :: kwarn = 1

!             kround =  1  causes all results to be rounded to the nearest FM number, or to the
!                          value with an even last digit if the result is halfway between two
!                          FM numbers.
!                    =  0  causes all results to be rounded toward zero (chopped).
!                    = -1  causes all results to be rounded toward minus infinity.
!                    =  2  causes all results to be rounded toward plus infinity.

      integer, save :: kround = 1

!             kround1 saves the rounding mode at the user's level, while kround may sometimes
!                     be changed within FM during evaluation of a function.

      integer, save :: kround1 = 1

!             krperf = 1  causes more guard digits to be used, to get perfect rounding in the mode
!                         set by kround.
!                    = 0  causes a smaller number of guard digits used, to give nearly perfect
!                         rounding.  This number is chosen so that the last intermediate result
!                         should have error less than 0.001 unit in the last place of the final
!                         rounded result.
!             Beginning with version 1.3 krperf is not used, since perfect rounding is always done.
!             The variable has been left in the package for compatibility with earlier versions.

      integer, save :: krperf = 0

!             kswide defines the maximum screen width to be used for all unit kw output.

      integer, save :: kswide = 80

!             keswch = 1   causes input to fminp with no digits before the exponent letter to be
!                          treated as if there were a leading '1'.  This is sometimes better for
!                          interactive input:  'e7' converts to 10.0**7.
!                    = 0   causes a leading zero to be assumed.  This gives compatibility with
!                          Fortran:  'e7' converts to 0.0.

      integer, save :: keswch = 1

!             cmchar defines the exponent letter to be used for FM variable output from fmout,
!                    as in 1.2345m+678.
!                    Change it to 'e' for output to be read by a non-FM program.

      character, save :: cmchar = 'M'

!             kround_retry is an internal flag controlling cases where the result was close to
!                          1/2 ulp of error and the operation should be done again with more
```

```
!                        guard digits to insure perfect rounding.

      integer, save :: kround_retry = 0

!             ksub is an internal flag set during subtraction so that the addition routine will
!                  negate its second argument.

      integer, save :: ksub = 0

!             ksqr is an internal flag set during squaring so that at high precision the
!                  multiplication routine will not need to compute the fft of its second argument.

      integer, save :: ksqr = 0

!             krem is an internal flag set during high precision integer division operations to
!                  indicate that the remainder in imdivr need not be computed.

      integer, save :: krem = 1

!             jrsign is an internal flag set during arithmetic operations so that the rounding
!                    routine will know the sign of the final result.

      integer, save :: jrsign = 0

!             These arrays are used by the fft routines for very high precision operations.

      complex (kind(0.0d0)), save, dimension(:), allocatable :: cx, cy, cz, roots_of_unity

!             n_roots_of_unity is the number of roots of unity stored in the roots_of_unity array,
!                              used by the fft array.

      integer, save :: n_roots_of_unity = 0

!             lhash is a flag variable used to indicate when to initialize two hash tables that are
!                   used for character look-up during input conversion.
!                   lhash = 1 means that the tables have been built.
!             lhash1 and lhash2 are the array dimensions of the tables.
!             khasht and khashv are the two tables.

      integer, save :: lhash = 0
      integer, parameter :: lhash1 =   0
      integer, parameter :: lhash2 = 256
      integer, save :: khasht(lhash1:lhash2), khashv(lhash1:lhash2)

!             dpeps is the approximate machine precision.

      double precision, save :: dpeps = epsilon(dp_two)

!             in_fact is 1 when fmgam is called from fmfact, to avoid raising precision twice.

      integer, save :: in_fact = 0

!             ljsums is the maximum number of concurrent sums to use in function evaluation.

      integer, parameter :: ljsums = 1000

!             Saved constants that depend on mbase.
```

```
      real (kind(1.0d0)), save :: mblogs = 0
!             (Setting mblogs to zero here will cause the other variables that depend on mbase
!              to automatically be defined when the first FM operation is done.)

      real, save :: alogmb = 1.611810e+1
      real, save :: alogm2 = 2.325350e+1
      real, save :: alogmx = 3.673680e+1
      real, save :: alogmt = 7.0e0

      integer, save :: ngrd21 = 1
      integer, save :: ngrd52 = 2
      integer, save :: ngrd22 = 2
      real (kind(1.0d0)), save :: mexpab = aint(max_exponent / 5.0d0)

      double precision, save :: dlogmb = 1.611809565095832d+1
      double precision, save :: dlogtn = 2.302585092994046d+0
      double precision, save :: dlogtw = 6.931471805599453d-1
      double precision, save :: dppi   = 3.141592653589793d+0
      double precision, save :: dlogtp = 1.837877066409345d+0
      double precision, save :: dlogpi = 1.144729885849400d+0
      double precision, save :: dlogeb = 2.236222824932432d+0

      real (kind(1.0d0)), save :: mbasel = 0
      real (kind(1.0d0)), save :: mbasen = 0

      integer, save :: ndigl = 0
      integer, save :: ndign = 0
      integer, save :: nguarl = 0
      integer, save :: n21
      integer, save :: ngrdn

!             These variables are used by fm_random_number.
!             mbrand is the base used for the random number arithmetic.
!                    It needs to remain the same even if the user changes mbase.

      real (kind(1.0d0)), save :: mbrand = m_ten ** aint(log(max_base/4.0d0) / log(10.0d0))

      type(multi), save :: mrnx
      type(multi), save :: mrna
      type(multi), save :: mrnm
      type(multi), save :: mrnc
      integer, save :: start_random_sequence = -1
      integer, save :: last_digit_of_m_m1
      double precision, save :: dpm

!             Work area for FM numbers, and related variables.

      type(multi), save :: mwa
      type(multi), save :: mwd
      type(multi), save :: mwe
      type(multi), save :: mpa
      type(multi), save :: mpb
      type(multi), save :: mpc
      type(multi), save :: mpd
      type(multi), save :: mwi
      type(multi), save :: mpma
      type(multi), save :: mpmb
      type(multi), save :: mpx(2)
```

```
      type(multi), save :: mpy(2)
      type(multi), save :: mpz(2)

!             Variables related to input/output and formatting.

      integer, save :: lmbuff = 0
      integer, save :: lmbufz = 0
      character, save, dimension(:), allocatable :: cmbuff, cmbufz, move_cmbuff

!             Saved FM constants.

      type(multi), save :: mpisav
      type(multi), save :: mesav
      type(multi), save :: mlbsav
      type(multi), save :: mln2
      type(multi), save :: mln3
      type(multi), save :: mln5
      type(multi), save :: mln7

!             Set the default value of jformz to give ' 1.23 + 4.56 i ' style format for output
!             of complex variables.

      integer, save :: jformz = 1

!             Set the default value of jprntz to print real and imaginary parts on one line
!             whenever possible.

      integer, save :: jprntz = 1

!             k_stat give status of allocation of multiple-precision variables.

      integer, save :: k_stat

!             mbern  is the array used to save Bernoulli numbers so they do not have to be
!                    re-computed on subsequent calls.
!             ndbern is the array used to save the number of digits in the current base for
!                    each of the saved Bernoulli numbers.

!             mbsbrn is the value of mbase for the currently saved Bernoulli numbers.

      real (kind(1.0d0)), save :: mbsbrn = 0

!             numbrn is the number of the largest Bernoulli number saved using base mbsbrn.

      integer, save :: numbrn = 0

!             lmbern is the size of the arrays mbern and ndbern.

      integer, parameter :: lmbern = 60000
      type(multi), save, dimension(lmbern) :: mbern
      integer, save, dimension(lmbern) :: ndbern = 0

!             b(2n) is stored in mbern(n) for 2n >= 28.

!             m_euler     is the saved value of Euler's constant.
!             m_gamma_ma  is the last input value to fmgam, and
!             m_gamma_mb  is the corresponding output value.
!             m_ln_2pi    holds the saved value of ln(2*pi).
```

```
      type(multi), save :: m_euler
      type(multi), save :: m_gamma_ma
      type(multi), save :: m_gamma_mb
      type(multi), save :: m_ln_2pi

!             mbsgam is the value of mbase used in the currently stored value of
!                    m_gamma_ma and m_gamma_mb.
!             ndggam is the maximum ndig used in the currently stored value of
!                    m_gamma_ma and m_gamma_mb.

      real (kind(1.0d0)), save :: mbsgam = 0

      integer, save :: ndggam = 0

!             mbs2pi is the value of mbase used in the currently stored value of ln(2*pi).
!             ndg2pi is the maximum ndig used in the currently stored value of ln(2*pi).

      real (kind(1.0d0)), save :: mbs2pi = 0

      integer, save :: ndg2pi = 0

!             mbseul is the value of mbase used in the currently stored value of m_euler.
!             ndgeul is the maximum ndig used in the currently stored value of m_euler.

      real (kind(1.0d0)), save :: mbseul = 0

      integer, save :: ndgeul = 0

   end module fmvals
```