

! Version 1.4

! This file collects all the various FM routines from the "More sample programs" page on the
! FM web site. See that page for the programs that call these routines and illustrate their
! use. Here is a list of the user-callable routines in this file (see the documentation at
! the top of each routine for explanation of each of the arguments to the routines).

! 1. Find a minimum or maximum function value of a real function of one variable.

! SUBROUTINE FM_FIND_MIN(MIN_OR_MAX,AX,BX,TOL,XVAL,FVAL,F,NF,KPRT,KW)

! 2. Nth derivative of a real function of one variable.

! FUNCTION FM_FPRIME(N,A,F,NF)

! 3. Nth derivative of a complex function of one variable.

! FUNCTION ZM_FPRIME(N,A,F,NF)

! 4. Definite integral for a real function of one variable.

! SUBROUTINE FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)

! 5. Inverse matrix for a real NxN matrix.

! SUBROUTINE FM_INVERSE(A,N,B,DET)

! 6. Inverse matrix for a complex NxN matrix.

! SUBROUTINE ZM_INVERSE(A,N,B,DET)

! 7. Generate the real linear system of normal equations for a least square fit.

! SUBROUTINE FM_GENEQ(F,A,B,K,X,Y,N)

! 8. Solve a real NxN linear system of equations.

! SUBROUTINE FM_LIN_SOLVE(A,X,B,N,DET)

! 9. Solve a complex NxN linear system of equations.

! SUBROUTINE ZM_LIN_SOLVE(A,X,B,N,DET)

! 10. Solve a real differential equation (initial value problem).

! SUBROUTINE FM_RK14(A,B,N_ORDER,N_FUNCTION,S,TOL,S1)

! 11. Find a root of a real function of one variable.

! SUBROUTINE FM_SECANT(AX,BX,F,NF,ROOT,KPRT,KU)

! 12. Find a root of a complex function of one variable.

! SUBROUTINE ZM_SECANT(AX,BX,F,NF,ROOT,KPRT,KU)

! 13. Find NR roots of a complex function of one variable.

```
! SUBROUTINE ZM_ROOTS(NR,F,NF,N_FOUND,LIST_OF_ROOTS,KPRT,KU)
```

```
! SUBROUTINE FM_FIND_MIN(MIN_OR_MAX,AX,BX,TOL,XVAL,FVAL,F,NF,KPRT,KW)
```

```
! USE FMZM
```

```
! MIN_OR_MAX having value 1 means minimize the function, otherwise maximize.
! AX, BX define the endpoints of an interval in which the search takes place.
! TOL is the tolerance for the minimum. Usually TOL should be no less than
! sqrt(epsilon(ax)), meaning the x-coordinate XVAL of the extreme point will
! be accurate to only about half the digits carried. The y-coordinate FVAL
! should be accurate to nearly full precision.
! This happens because the typical graph is nearly parabolic near the minimum,
! and within sqrt(epsilon(ax)) of the minimum all the function values are
! essentially identical at the current precision.
! XVAL is returned as the value of X that minimizes (or maximizes) function F(X,NF).
! It is a relative extreme point, and may not be the global extreme point if the
! function has more than one extremum on the interval.
! FVAL is returned as the function value at XVAL.
! F(X,NF) is the function to be minimized. X is the argument and NF is the function
! number, in case several functions are defined within F.
! KPRT controls printing within the routine:
! KPRT = 0 for no output
! KPRT = 1 for the approximation to the root and the function
! value to be printed once at the end of the routine.
! KPRT = 2 for the approximation to the root and the function
! value to be printed each iteration.
! KW is the unit number for output.
! The method used is a combination of golden section search and successive parabolic interpolation.
! Convergence is never much slower than that for a fibonacci search. If f has a continuous second
! derivative which is positive at the minimum (which is not at ax or bx), then convergence is
! superlinear, and usually of the order of about 1.324....
! The function f is never evaluated at two points closer together than eps*abs(FVAL)+(tol/3),
! where eps is approximately the square root of the relative machine precision. If f is a
! unimodal function and the computed values of f are always unimodal when separated by at least
! eps*abs(x)+(tol/3), then FVAL approximates the abscissa of the global minimum of f on the
! interval ax,bx with an error less than 3*eps*abs(FVAL)+tol. If f is not unimodal, then FVAL
! may approximate a local, but perhaps non-global, minimum to the same accuracy.
! This routine is a slightly modified translation of function FVAL from netlib, which was adapted
! from the algol 60 procedure localmin given by Richard Brent in Algorithms For Minimization
! Without Derivatives, Prentice-Hall (1973).
```

```
! IMPLICIT NONE
```

```
! CHARACTER(80) :: ST1,ST2
```

```
! INTEGER :: J,MINV,MIN_OR_MAX,NF,KPRT,KW
```

```
! TYPE (FM) :: AX,BX,TOL,XVAL,FVAL
```

```
! TYPE (FM), EXTERNAL :: F
```

```
! TYPE (FM), SAVE :: A, B, C, D, E, EPS, XM, P, Q, R, T2, U, V, W, FU, FV, FW, FX, X, TOL1, TOL3
```

```
! MINV = 1
```

```
! IF (MIN_OR_MAX /= 1) MINV = -1
```

```
! C is the squared inverse of the golden ratio.
```

```
C = (3-SQRT(TO_FM('5.0D0')))/2
```

! EPS is approximately the square root of the relative machine precision.

```
EPS = EPSILON(AX)
```

```
TOL1 = EPS + 1
```

```
EPS = SQRT(EPS)
```

```
A = MIN(AX, BX)
```

```
B = MAX(AX, BX)
```

```
V = A + C*(B-A)
```

```
W = V
```

```
X = V
```

```
E = 0
```

```
FX = F(X, NF)*MINV
```

```
FV = FX
```

```
FW = FX
```

```
TOL3 = TOL/3
```

```
J = 1
```

```
IF (KPRT == 2) THEN
```

```
  WRITE (KW,*) ' '
```

```
  IF (MIN_OR_MAX == 1) THEN
```

```
    WRITE (KW,*) ' FM_FIND_MIN.  Begin trace of all iterations.'
```

```
    WRITE (KW,*) ' Search for a relative minimum on the interval'
```

```
    WRITE (KW, "(13X,ES20.10,' to ',ES20.10/)" TO_DP(AX),TO_DP(BX)
```

```
  ELSE
```

```
    WRITE (KW,*) ' FM_FIND_MIN.  Begin trace of all iterations.'
```

```
    WRITE (KW,*) ' Search for a relative maximum on the interval'
```

```
    WRITE (KW, "(13X,ES20.10,' to ',ES20.10/)" TO_DP(AX),TO_DP(BX)
```

```
  ENDIF
```

```
  ST1 = FM_FORMAT('ES35.25',X)
```

```
  ST2 = FM_FORMAT('ES35.25',FX*MINV)
```

```
  WRITE (KW, "( ' J =',I3,4X,' x = ',A)" J,TRIM(ST1)
```

```
  WRITE (KW, "( ' ',3X,4X,' f(x) = ',A/)" TRIM(ST2)
```

```
ENDIF
```

! The main loop starts here.

```
110 XM = (A+B)/2
```

```
TOL1 = EPS*ABS(X) + TOL3
```

```
T2 = 2*TOL1
```

! Check the stopping criterion.

```
IF (ABS(X-XM) <= (T2-(B-A)/2)) GO TO 160
```

```
P = 0
```

```
Q = 0
```

```
R = 0
```

```
IF (ABS(E) > TOL1) THEN
```

```
  R = (X-W)*(FX-FV) ! Fit a parabola.
```

```
  Q = (X-V)*(FX-FW)
```

```
  P = (X-V)*Q-(X-W)*R
```

```
  Q = 2*(Q-R)
```

```
  IF (Q > 0) THEN
```

```
    P = -P
```

```
  ELSE
```

```

        Q = -Q
    ENDIF
    R = E
    E = D
ENDIF

```

```

IF ((ABS(P) >= ABS(Q*R/2)) .OR. (P <= Q*(A-X)) .OR. (P >= Q*(B-X))) GO TO 120

```

```

!           Make a parabolic-interpolation step.

```

```

D = P/Q
U = X + D

```

```

!           f must not be evaluated too close to ax or bx.

```

```

IF (((U-A) >= T2) .AND. ((B-U) >= T2)) GO TO 130
D = TOL1
IF (X >= XM) D = -D
GO TO 130

```

```

!           Make a golden-section step.

```

```

120 IF (X < XM) THEN
    E = B - X
ELSE
    E = A - X
ENDIF
D = C*E

```

```

!           f must not be evaluated too close to x.

```

```

130 IF (ABS(D) >= TOL1) THEN
    U = X + D
ELSE
    IF (D > 0) THEN
        U = X + TOL1
    ELSE
        U = X - TOL1
    ENDIF
ENDIF
FU = F(U,NF)*MINV

```

```

J = J + 1

```

```

IF (KPRT == 2) THEN
    ST1 = FM_FORMAT('ES35.25',U)
    ST2 = FM_FORMAT('ES35.25',FU*MINV)
    WRITE (KW,"('          J =',I3,4X,' x = ',A)") J,TRIM(ST1)
    WRITE (KW,"('          ',3X,4X,' f(x) = ',A/)" TRIM(ST2)
ENDIF

```

```

!           update a, b, v, w, and x.

```

```

IF (FX <= FU) THEN
    IF (U < X) THEN
        A = U
    ELSE
        B = U
    ENDIF

```

```

ENDIF
IF (FU > FX) GO TO 140
IF (U < X) THEN
    B = X
ELSE
    A = X
ENDIF
V = W
FV = FW
W = X
FW = FX
X = U
FX = FU
GO TO 110

```

```

140 IF ((FU > FW) .AND. (W /= X)) GO TO 150
    V = W
    FV = FW
    W = U
    FW = FU
    GO TO 110

```

```

150 IF ((FU > FV) .AND. (V /= X) .AND. (V /= W)) GO TO 110
    V = U
    FV = FU
    GO TO 110

```

! end of main loop

```

160 XVAL = X
    FVAL = FX*MINV

```

```

IF (KPRT >= 1) THEN
    IF (KPRT == 1) WRITE (KW,*) ' '
    IF (MIN_OR_MAX == 1) THEN
        WRITE (KW, "(' FM_FIND_MIN.  Function ',I3,I6,' iterations.  A relative minimum'// &
            " on interval'/13X,ES20.10,' to ',ES20.10,' is')") &
            NF,J,TO_DP(AX),TO_DP(BX)
        ST1 = FM_FORMAT('ES35.25',XVAL)
        ST2 = FM_FORMAT('ES35.25',FVAL)
        WRITE (KW,"(15X,' x = ',A)") TRIM(ST1)
        WRITE (KW,"(15X,' f(x) = ',A)") TRIM(ST2)
    ELSE
        WRITE (KW, "(' FM_FIND_MIN.  Function ',I3,I6,' iterations.  A relative maximum'// &
            " on interval'/13X,ES20.10,' to ',ES20.10,' is')") &
            NF,J,TO_DP(AX),TO_DP(BX)
        ST1 = FM_FORMAT('ES35.25',XVAL)
        ST2 = FM_FORMAT('ES35.25',FVAL)
        WRITE (KW,"(15X,' x = ',A)") TRIM(ST1)
        WRITE (KW,"(15X,' f(x) = ',A)") TRIM(ST2)
    ENDIF
    WRITE (KW,*) ' '
ENDIF

```

```

END SUBROUTINE FM_FIND_MIN

```

```

FUNCTION FM_FPRIME(N,A,F,NF)      RESULT (RETURN_VALUE)
USE FMVALS

```

```
USE FMZM
IMPLICIT NONE
```

```
! This routine finds the Nth derivative of F(X,NF), evaluated at A.
! NF is passed on to function F to indicate which function to use in cases where several
! different functions may be defined there.

! F must be defined in an interval containing A, so that F can be sampled on both sides of A.

! N may be zero, so that in cases where F suffers cancellation error at A, an accurate
! function value is returned.

! FM_FPRIME tries to return full accuracy for the derivative, by raising precision above
! the user's level and using difference formulas.
```

```
TYPE (FM)          :: RETURN_VALUE, A
TYPE (FM), EXTERNAL :: F
INTEGER :: J, K, KWARN_SAVE, NDSAVE, N, NF
TYPE (FM), SAVE :: D1, D2, F1, F2, H, TOL, TOL2, X1
```

```
!           Raise precision slightly.
```

```
NDSAVE = NDIG
NDIG = NDIG + NGRD52
CALL FM_EQU(A,X1,NDSAVE,NDIG)
KWARN_SAVE = KWARN
KWARN = 0
```

```
D2 = 0
F1 = F(X1,NF)
IF (F1 /= 0) THEN
    CALL FM_ULP(F1,TOL)
ELSE
    TOL = EPSILON(TO_FM(1))
ENDIF
TOL = ABS(TOL)
```

```
!           Check for a legal function value.
```

```
IF (IS_UNKNOWN(F1) .OR. IS_OVERFLOW(F1) .OR. IS_UNDERFLOW(F1) .OR. N < 0) THEN
    D2 = TO_FM(' UNKNOWN ')
    GO TO 110
ENDIF
F2 = F1
```

```
!           Loop at increasing precision until the difference formula is accurate.
```

```
DO J = 1, 100

    NDIG = 2*NDIG
```

```
!           Define the variables used below at the new higher precision.
```

```
CALL FM_EQU(D2,D1,NDIG/2,NDIG)
CALL FM_EQU(F2,F1,NDIG/2,NDIG)
CALL FM_EQU(TOL,TOL2,NDSAVE,NDIG)
CALL FM_EQU(A,X1,NDSAVE,NDIG)
```

! Special case for N = 0.

```
IF (N == 0) THEN
  F2 = F(X1,NF)
  D2 = F2
  IF (ABS(F2-F1) < TOL2) GO TO 110
  CYCLE
ENDIF
F2 = F1
```

! Special case for N = 1.

```
IF (N == 1) THEN
  IF (X1 /= 0) THEN
    CALL FM_ULP(X1,H)
  ELSE
    H = EPSILON(TO_FM(1))
  ENDIF
  H = SQRT(ABS(H))
  D2 = ( F(X1+H,NF) - F(X1-H,NF) ) / (2*H)
  IF (ABS(D2-D1) < TOL2 .AND. J > 1) GO TO 110
  CYCLE
ENDIF
```

! General case for even N > 1.

```
IF (MOD(N,2) == 0) THEN
  IF (X1 /= 0) THEN
    CALL FM_ULP(X1,H)
  ELSE
    H = EPSILON(TO_FM(1))
  ENDIF
  H = ABS(H)**(TO_FM(1)/(N+2))
  D2 = (-1)**(N/2) * BINOMIAL(TO_FM(N),TO_FM(N/2)) * F(X1,NF)
  DO K = 0, N/2-1
    D2 = D2 + (-1)**K * BINOMIAL(TO_FM(N),TO_FM(K)) * &
      ( F(X1+(N/2-K)*H,NF) + F(X1-(N/2-K)*H,NF) )
  ENDDO
  D2 = D2 / H**N
  IF (ABS(D2-D1) < TOL2 .AND. J > 1) GO TO 110
  CYCLE
ENDIF
```

! General case for odd N > 1.

```
IF (MOD(N,2) == 1) THEN
  IF (X1 /= 0) THEN
    CALL FM_ULP(X1,H)
  ELSE
    H = EPSILON(TO_FM(1))
  ENDIF
  H = ABS(H)**(TO_FM(1)/(N+2))
  D2 = 0
  DO K = 0, N/2
    D2 = D2 + (-1)**K * BINOMIAL(TO_FM(N-1),TO_FM(K)) * &
      ( F(X1+(N/2-K+1)*H,NF) - F(X1-(N/2-K+1)*H,NF) ) * &
      TO_FM(N*(N+1-2*K)) / ((N-K)*(N+1-K))
  ENDDO
```

```

        D2 = D2 / (2*H**N)
        IF (ABS(D2-D1) < TOL2 .AND. J > 1) GO TO 110
        CYCLE
    ENDIF

```

```

ENDDO

```

```

!           Round and return.

```

```

110 CALL FM_EQU(D2,RETURN_VALUE,NDIG,NDSAVE)
    NDIG = NDSAVE
    KWARN = KWARN_SAVE
    END FUNCTION FM_FPRIME

```

```

SUBROUTINE FM_GENEQ(F,A,B,K,X,Y,N)
    USE FMZM
    IMPLICIT NONE

```

```

! Generate the KxK matrix A and Kx1 vector B of normal equations for the least square
! fit of the K-parameter model

```

```

!     Y = C(1)*F(1,X) + ... + C(K)*F(K,X)

```

```

! to the data points (X(J),Y(J)), J = 1, 2, ..., N.

```

```

! A and B are returned, and then the coefficients C can be found by solving the
! linear system  A * C = B.

```

```

! Function L in the model evaluated at X is referenced by F(L,X) in this routine,
! and F should be supplied as an external function subprogram by the user.

```

```

INTEGER :: K, N
TYPE (FM), EXTERNAL :: F
TYPE (FM) :: A(K,K), B(K), X(N), Y(N)
TYPE (FM), ALLOCATABLE :: FXI(:)
INTEGER :: I, J, L
TYPE (FM) :: XI, YI, FXIL

```

```

IF (N <= 0 .OR. K <= 0) THEN
    WRITE (*, "(/' Error in FM_GENEQ. K,N=',2I8/)" ) K,N
    STOP
ENDIF

```

```

ALLOCATE(FXI(K),STAT=J)
IF (J /= 0) THEN
    WRITE (*, "(/' Error in FM_GENEQ. Unable to allocate FXI with size ',I8/)" ) K
    STOP
ENDIF

```

```

!           Initialize the upper triangle of A.

```

```

DO I = 1, K
    DO J = I, K
        A(I,J) = 0
    ENDDO
    B(I) = 0
ENDDO

```



```

!           Loop over the data points.

DO I = 1, N
  XI = X(I)
  YI = Y(I)

!           Compute the K function values at X(I).

DO J = 1, K
  FXI(J) = F(J,XI)
ENDDO

!           Multiply the function values and add the products to the matrix.

DO L = 1, K
  FXIL = FXI(L)
DO J = L, K
  A(L,J) = A(L,J) + FXIL*FXI(J)
ENDDO

!           Sum the right-hand-side term.

B(L) = B(L) + YI*FXIL
ENDDO
ENDDO

!           Fill the lower triangle of the A matrix using symmetry.

IF (K >= 2) THEN
  DO L = 2, K
    DO J = 1, L-1
      A(L,J) = A(J,L)
    ENDDO
  ENDDO
ENDIF

DEALLOCATE(FXI)
END SUBROUTINE FM_GENEQ

RECURSIVE SUBROUTINE FM_INTEGRATE(F,N,A,B,TOL,RESULT,KPRT,NW)
USE FMVALS
USE FMZM
IMPLICIT NONE

! High-precision numerical integration.

! Integrate F(X,N) from A to B. N is passed on to function F to indicate which function to use in
! cases where several different functions may be defined there.

! WARNING: If the function F being integrated or one of its derivatives does not exist at one or
! both of the endpoints (A,B), the endpoints should be exactly representable in FM's
! number system. For non-exact numbers like 1/3, sqrt(2), or pi/2, when FM_INTEGRATE
! raises precision to evaluate the integration formula the endpoints are not accurate
! enough at the higher precision.

! Example: Integrate sqrt( tan( x ) ) from 0 to pi/2.
! First, pi/2 is not exact as an FM number. At some precisions it may have rounded up,

```

! making $\tan(x)$ negative and causing an error in `sqrt`. Using `sqrt(abs(tan(x)))`
! is safer.
! Second, $b = \pi/2$ has been computed at the user's precision, so when `FM_INTEGRATE`
! raises precision, the value of b is just zero-padded on the end, which does not give
! enough information about how $f(x)$ behaves near the singularity at $\pi/2$.

! Make the endpoints exact by changing variables. Change the interval to $[0 , 1]$:

! $u = (2/\pi) * x \Rightarrow du = (2/\pi) * dx$
! so

! $x = (\pi/2) * u \Rightarrow dx = (\pi/2) * du$
! new limits

! $x = 0 \Rightarrow u = 0$ and $x = \pi/2 \Rightarrow u = 1$

! New integral: Integrate $(\pi/2) * \text{sqrt}(\text{abs}(\tan(\pi*u/2)))$ from 0 to 1.

! Now the function `F` should declare a local saved `type(fm)` variable `PI`, and then use
! `CALL FM_PI(PI)` each time `F` is called to make sure the value of `PI` is correct at the
! higher precision being used by `FM_INTEGRATE` when `F` is called.

```
TYPE (FM) :: A,B,RESULT,TOL
TYPE (FM), EXTERNAL :: F
INTEGER :: N,KPRT,NW
INTENT (IN) :: N,A,B,TOL,KPRT,NW
INTENT (INOUT) :: RESULT
```

! `A,B,TOL`, and `RESULT` are all `type (FM)` variables, and function `F` returns a `type (FM)` result.

! `RESULT` is returned as the value of the integral, with $\text{ABS}((\text{RESULT}-\text{true})/\text{true})$ less than `TOL`
! (i.e., `TOL` is a relative error tolerance).

! For example, to get 30 significant digits correct for the integral, set `TOL = 1.0E-30`.

! `FM` precision must be set high enough in the calling program (using `FM_SET`) so that $1+\text{TOL} > 1$
! at that precision. Using `TOL = EPSILON(TO_FM(1))` will usually get a full precision result,
! but for some functions this might fail. A better strategy is to set precision higher than
! the accuracy required for the integral. For example, to get the integral to 50 significant
! digits, `CALL FMSET(60)` and then set `TOL = TO_FM(' 1.0E-50 ')` before the call to `FM_INTEGRATE`.

! `KPRT` can be used to show intermediate results on unit `NW`.

! `KPRT = 0` for no output

! `= 1` prints a summary for each call to `FM_INTEGRATE`

! `= 2` prints a trace of all iterations.

! `NW` is the unit number used for `KPRT` output and any error or warning messages.

! No method for numerical integration is foolproof. Since it samples only a finite number of
! function values, any numerical integration algorithm can be made to fail by choosing a
! sufficiently badly-behaved function. Such functions often vary by many orders of magnitude
! over relatively small fractions of the interval (A,B) .

! `F` should be well-behaved in the interior of the interval (A,B) .

! The routine tries to handle any singularities of `F` or `F'` at `A` and/or `B`, so cases with interior
! singularities should be done as separate calls having the singularities as endpoints.

! The routine will try to handle cases where `F` or `F'` has singularities inside (A,B) , but then
! the time will be much slower and the routine might fail.

! For a function with a removable singularity in the interior of the interval, such as

! $f(x) = 1/\ln(x) - 1/(x-1)$, define `F(X,N)` to check for $X = 1$ and return the correct limiting

! value, 0.5 in this case, when X is 1.

! Among functions with no singularities, examples of badly behaved functions are those with one or more extremely narrow tall spikes in their graphs. If possible, identify the peaks of any such spikes first, then make separate calls with the peaks as endpoints of the intervals of integration.

! If the value of the integral is zero or very close to zero, relative error may be undefined, so this routine may fail to converge and then return unknown. For these cases, try breaking the integral into two pieces and calling twice to get two non-zero results. These two results can then be added, often giving the original integral with sufficiently small absolute error even though small relative error could not be attained.

! If the function values are too extreme, it can cause problems. For example, if an exponential in F underflows and then is multiplied by something bigger than one, then F will return unknown. If the result of the integral is much larger than the underflow threshold (TINY(TO_FM(1))), then it is safe to set the underflowed results in F to zero to avoid getting unknown.

! If the function is nearly divergent FM_INTEGRATE may fail. $1/x$ from 0 to b is divergent. $1/x^{0.99}$ converges, but so slowly that FM_INTEGRATE may run a long time and then might fail. $1/x^{0.9999}$ converges even more slowly and FM_INTEGRATE may fail by declaring that the integral seems divergent.

! When the integrand is highly (or infinitely) oscillatory, FM_INTEGRATE may fail. If F has more than about 100 oscillations on the interval (A,B), it may be necessary to break the interval into smaller intervals and call FM_INTEGRATE several times. For infinitely many oscillations, like $\sin(1/x)$ from 0 to 1, first turn the integral into an infinite series by calling FM_INTEGRATE to integrate each separate loop between roots of $\sin(1/x)$. The function is well-behaved for each call, so FM_INTEGRATE can get high precision quickly for each. Next form a sequence of k partial sums for this series. The series converges slowly, with 50 or 100 terms giving only 3 or 4 significant digits of the sum, so an extrapolation method can be used to get a more accurate value of the sum of this series from its first k terms. For an alternating series like this, the extrapolation method of Cohen, Villegas, and Zagier often works very well. Repeated Aitken extrapolation could be used instead -- it is a more widely known method. Sample program Oscillate.f95 computes this integral.

! M is the maximum level for the integration algorithm. The number of function evaluations roughly doubles for each successive level until the tolerance is met. Using M = 12 allows up to about 5,000 digits for most integrals, but the upper limit for a given M depends on the function. Raising M further will approximately double the maximum precision for each extra level, but will also double the memory usage for each extra level.

```

INTEGER, PARAMETER :: M = 12
INTEGER, PARAMETER :: NT = 20*2**M
TYPE (FM), SAVE :: ST_SAVE(0:NT)
INTEGER, SAVE :: PRECISION_OF(0:NT) = 0
INTEGER :: ABSIGN,I,ISTEP,K,KWSAVE,NDS,NDSAVE,NRETRY
INTEGER, SAVE :: R_LEVEL = 0, NUM_F = 0

```

```

TYPE (FM) :: A1,AB2,B1,C1,C2,CI,CT,D,EPS,ERR1,ERR2,FMAX,FMAX2,H,HF, LAST_H,PI,PRIOR_HS, &
S,S1,S2,SI,ST,T,TOL1,TOL2,X,XF,XMAX,V,W

```

```

CHARACTER(80) :: ST1,ST2
REAL :: TIME1,TIME2
LOGICAL :: SPIKE_FOUND,ST_IS_SAVED

```

! Iterative tanh-sinh integration is used, increasing the order until convergence
! is obtained, or M levels have been done.

RESULT = 0

NCALL = NCALL + 1

NAMEST(NCALL) = 'INTEGRATE'

KWSAVE = KW

KW = NW

R_LEVEL = R_LEVEL + 1

NRETRY = 0

IF (KPRT >= 2) THEN

WRITE (NW, "(A)") ' '

WRITE (NW, "(A,I9,A)") ' Input to FM_INTEGRATE. Function N = ',N,'. A, B ='

CALL FM_PRINT(A)

CALL FM_PRINT(B)

CALL FM_FORM('E20.8',TOL,ST1)

WRITE (NW, "(A,A)") ' TOL =',TRIM(ST1)

ENDIF

CALL CPU_TIME(TIME1)

! Check for special cases.

IF (A == B) THEN

A1 = F(A,N)

IF (R_LEVEL <= 1) THEN

NUM_F = 1

ELSE

NUM_F = NUM_F + 1

ENDIF

IF (A1%MFMP(2) == MUNKNO) THEN

CALL FM_F_FAIL(A,N,NW)

PRIOR_HS = A1

GO TO 120

ELSE

PRIOR_HS = 0

GO TO 120

ENDIF

ENDIF

! Check to make sure the user has set precision high enough for the value of TOL chosen.

TOL1 = TOL

IF (TOL < ABS(EPSILON(A))) THEN

WRITE (NW, "(A)") ' '

WRITE (NW, "(A)") ' Error in FM_INTEGRATE. TOL is '

CALL FM_PRINT(TOL)

WRITE (NW, "(A)") ' This is too small for the current precision level. Current epsilon ='

TOL1 = ABS(EPSILON(A))

CALL FM_PRINT(TOL1)

WRITE (NW, "(A)") ' This larger value will be used. TOL ='

CALL FM_PRINT(TOL1)

WRITE (NW, "(A)") ' Use FM_SET to set a higher precision before the call to'

WRITE (NW, "(A)") ' FM_INTEGRATE if the smaller TOL is needed.'

ENDIF

! Raise the precision.

!
! Check for an integrable singularity at either endpoint, and increase precision
! if it seems that a retry would be needed at the first precision.

```
NDSAVE = NDIG
A1 = LOG10(F(A+(B-A)*TO_FM(' 1.0E-10 '),N)/F(A+(B-A)*TO_FM(' 1.0E-20 '),N))/10
B1 = LOG10(F(B-(B-A)*TO_FM(' 1.0E-10 '),N)/F(B-(B-A)*TO_FM(' 1.0E-20 '),N))/10
IF (R_LEVEL <= 1) THEN
  NUM_F = 4
ELSE
  NUM_F = NUM_F + 4
ENDIF
IF (A1 < -0.999 .OR. B1 < -0.999) THEN
  PRIOR_HS = TO_FM(' UNKNOWN ')
  WRITE (NW, "(A)") ' '
  WRITE (NW, "(A,I9,A)") ' FM_INTEGRATE failed -- F(X,N) for N = ',N, &
    ' seems to have a non-integrable singularity'
  WRITE (NW, "(A)") ' at an endpoint. A,B ='
  CALL FM_PRINT(A)
  CALL FM_PRINT(B)
  WRITE (NW, "(A)") ' Check the limits of integration, function number (N), and' // &
    ' function definition.'
  WRITE (NW, "(A)") ' '
  GO TO 120
ENDIF
NDIG = NDIG+INT(30/ALOGMT)
CALL FM_EQU_R1(A1,NDSAVE,NDIG)
CALL FM_EQU_R1(B1,NDSAVE,NDIG)
IF (A1 < -0.2 .OR. B1 < -0.2) NDIG = 2*NDIG

CALL CPU_TIME(TIME1)
```

! Start here when doing a retry.

```
110 NRETRY = NRETRY + 1
CALL FM_EQU(A,A1,NDSAVE,NDIG)
CALL FM_EQU(B,B1,NDSAVE,NDIG)
ABSIGN = 1
IF (A1 > B1) THEN
  CALL FM_EQU(B,A1,NDSAVE,NDIG)
  CALL FM_EQU(A,B1,NDSAVE,NDIG)
  ABSIGN = -1
ELSE IF (A1 == B1) THEN
  PRIOR_HS = 0
  GO TO 120
ENDIF
CALL FM_EQU(TOL1,TOL2,NDSAVE,NDIG)

IF (KPRT >= 2) THEN
  WRITE (NW, "(A)") ' '
  WRITE (NW, "(A,I9,A,I5)") ' Begin FM_INTEGRATE. NDIG = ',NDIG,' Recursion level = ', &
    R_LEVEL
ENDIF

S = 0
PRIOR_HS = 0
ERR2 = 1
EPS = EPSILON(TOL2)
D = ABS(B1-A1)/100
```

```

FMAX = 0
FMAX2 = 0
XMAX = A1
H = 1
LAST_H = H/2**M
CALL FM_PI(PI)

```

```

HF = (B1-A1)/2
AB2 = A1 + HF

```

```

DO K = 1, M
  H = H/2
  ISTEP = 2**(M-K)
  IF (K > 1) THEN
    T = ISTEP*LAST_H
    CALL FM_CHSH(T,C1,S1)
    T = 2*S1*C1
    C2 = C1**2 + S1**2
    S2 = T
  ENDIF
DO I = 0, NT, ISTEP
  IF (MOD(I,2*ISTEP) /= 0 .OR. K == 1) THEN

```

! The + or -X values are the abscissas for interval (-1,1).
! XF translates these to the interval (A,B).

```

IF (I == 0) THEN
  X = 0
  W = PI/2
  XF = HF*X + AB2
  T = F(XF,N)
  NUM_F = NUM_F + 1
  IF (T%MFM%MP(2) == MUNKNO) THEN
    CALL FM_F_FAIL(XF,N,NW)
    PRIOR_HS = T
    GO TO 120
  ENDIF
  FMAX2 = MAX(FMAX2,ABS(T))
  IF (ABS(T) > FMAX .AND. XF > A1+D .AND. XF < B1-D) THEN
    FMAX = ABS(T)
    XMAX = XF
  ENDIF
  S = S + W*HF*T
ELSE
  IF (K == 1) THEN
    T = I*LAST_H
    CALL FM_CHSH(T,CI,SI)
  ELSE

```

! Use the hyperbolic addition formulas to get the next cosh and sinh
! quickly when evaluated at I*LAST_H.

```

IF (I == ISTEP) THEN
  CI = C1
  SI = S1
ELSE
  T = SI*C2 + CI*S2
  CI = CI*C2 + SI*S2
  SI = T

```

```

        C1 = CI
        S1 = SI
    ENDIF
ENDIF
ST_IS_SAVED = .FALSE.
IF (ALLOCATED(ST_SAVE(I)%MFM%MP)) THEN
    IF (PRECISION_OF(I) >= NDIG) ST_IS_SAVED = .TRUE.
ENDIF
IF (ST_IS_SAVED) THEN
    ST = ST_SAVE(I)
    CT = SQRT(1+ST**2)
ELSE
    T = PI*SI/2
    CALL FM_CHSH(T,CT,ST)
    ST_SAVE(I) = ST
    PRECISION_OF(I) = NDIG
ENDIF
W = (PI/2)*CI/CT**2
IF (W < EPS) EXIT
X = ST/CT
XF = HF*(-X) + AB2
IF (XF > A1) THEN
    T = F(XF,N)
    NUM_F = NUM_F + 1
    IF (T%MFM%MP(2) == MUNKNO) THEN
        CALL FM_F_FAIL(XF,N,NW)
        PRIOR_HS = T
        GO TO 120
    ENDIF
    FMAX2 = MAX(FMAX2,ABS(T))
    IF (ABS(T) > FMAX .AND. XF > A1+D .AND. XF < B1-D) THEN
        FMAX = ABS(T)
        XMAX = XF
    ENDIF
    S = S + W*HF*T
ENDIF
XF = HF*(X) + AB2
IF (XF < B1) THEN
    T = F(XF,N)
    NUM_F = NUM_F + 1
    IF (T%MFM%MP(2) == MUNKNO) THEN
        CALL FM_F_FAIL(XF,N,NW)
        PRIOR_HS = T
        GO TO 120
    ENDIF
    FMAX2 = MAX(FMAX2,ABS(T))
    IF (ABS(T) > FMAX .AND. XF > A1+D .AND. XF < B1-D) THEN
        FMAX = ABS(T)
        XMAX = XF
    ENDIF
    S = S + W*HF*T
ENDIF
ENDIF
ENDIF
ENDDO
IF (KPRT >= 2) THEN
    WRITE (NW,"(A)") ' '
    WRITE (NW,"(A,I9,A,I9,A)") ' K = ',K,' ',NUM_F, &

```

' function calls so far. Integral approximation ='

V = H*S

CALL FM_PRINT(V)

ENDIF

IF (K > 1) THEN

ERR1 = ERR2

IF (S /= 0) THEN

ERR2 = ABS((PRIOR_HS - H*S)/(H*S))

ELSE

ERR2 = ABS((PRIOR_HS - H*S))

ENDIF

IF (KPRT >= 2) THEN

CALL FM_FORM('ES15.3',ERR2,ST1)

WRITE (NW,"(A,A)") ' relative error of the last two approximations = ', &
TRIM(ST1)

ENDIF

! Check for convergence.

IF (K > 3 .AND. ERR2 > 0 .AND. ERR2 < TOL2/10.0) EXIT

IF (K > 5 .AND. ERR2 == 0) EXIT

! If the errors do not decrease fast enough, raise precision and try again.

IF (K > 3*NRETRY .AND. ERR1 > 0 .AND. ERR2 > 0) THEN

IF (LOG(ERR2)/LOG(ERR1) < 1.2 .AND. ERR1 < 1.0D-6) THEN

NDIG = 2*NDIG

IF (KPRT >= 2) THEN

WRITE (NW,"(A,I9,A,I9)") ' FM_INTEGRATE Retry. So far, NUM_F = ', NUM_F, &
' New NDIG = ', NDIG

ENDIF

IF (NRETRY <= 3) GO TO 110

NDIG = NDIG/2

ENDIF

ENDIF

ENDIF

PRIOR_HS = H*S

! No convergence in M iterations.

! Before giving up, look for an interior singularity or tall spike. If one is found,
! split (A,B) into two intervals with the interior singularity as an endpoint, and try
! again as two integrals.

IF (K == M .OR. (K >= 9 .AND. ERR2 > 1.0D-7 .AND. ABS(TOL2) < 1.0D-16)) THEN

IF (KPRT >= 2) THEN

WRITE (NW,"(A)") ' '

WRITE (NW,"(A,I6,A)") ' No convergence in ', M, &
' iterations. Look for an interior singularity.'

CALL FM_FORM('ES25.6',XMAX,ST1)

CALL FM_FORM('ES25.6',FMAX,ST2)

WRITE (NW,"(I9,A,A,A)") NUM_F, ' function calls so far. XMAX, FMAX = ', &
TRIM(ST1), TRIM(ST2)

ENDIF

CALL FM_SPIKE(F, N, A1, B1, XMAX, FMAX, NUM_F, SPIKE_FOUND, KPRT, NW)

CALL FMEQU_R1(A1%MFM, NDIG, NDSAVE)

CALL FMEQU_R1(B1%MFM, NDIG, NDSAVE)

CALL FMEQU_R1(XMAX%MFM, NDIG, NDSAVE)

NDS = NDIG


```

NDIG = NDSAVE
IF (SPIKE_FOUND) THEN
  IF (MIN(ABS(A-XMAX),ABS(B-XMAX)) < 1.01*D) THEN
    NDIG = 2*NDS
    IF (NRETRY <= 5) GO TO 110
    NDIG = NDSAVE
  ENDIF
  IF (KPRT >= 2) THEN
    WRITE (NW,"(A)") ' '
    WRITE (NW,"(A)") ' Split the integral. First half: A,B = '
    CALL FM_PRINT(A1)
    CALL FM_PRINT(XMAX)
  ENDIF
  CALL FM_INTEGRATE(F,N,A1,XMAX,TOL,C1,KPRT,NW)
  IF (C1%MFMP%MP(2) == MUNKNO) THEN
    PRIOR_HS = C1
    GO TO 120
  ENDIF
  IF (KPRT >= 2) THEN
    WRITE (NW,"(A)") ' '
    WRITE (NW,"(A)") ' Split the integral. Second half: A,B = '
    CALL FM_PRINT(XMAX)
    CALL FM_PRINT(B1)
  ENDIF
  CALL FM_INTEGRATE(F,N,XMAX,B1,TOL,C2,KPRT,NW)
  PRIOR_HS = C1 + C2
  GO TO 120
ENDIF
CALL FM_INT_FAIL(N,A,B,TOL,M,ERR2,PRIOR_HS,NW)
PRIOR_HS = TO_FM(' UNKNOWN ')
GO TO 120
ENDIF
ENDDO

```

```

PRIOR_HS = ABSIGN*H*S

```

! Round the result and return.

```

120 CALL FM_EQU(PRIOR_HS,RESULT,NDIG,NDSAVE)

```

```

NDIG = NDSAVE
NCALL = NCALL - 1
CALL CPU_TIME(TIME2)

```

```

IF (KPRT >= 2 .OR. ( R_LEVEL <= 1 .AND. KPRT == 1 ) ) THEN
  WRITE (NW,"(A)") ' '
  WRITE (NW,"(A,I9,A)") ' Return from FM_INTEGRATE. Function N = ',N,'. A, B = '
  CALL FM_PRINT(A)
  CALL FM_PRINT(B)
  CALL FM_FORM('ES20.8',TOL,ST1)
  WRITE (NW,"(A,A)") ' TOL =',TRIM(ST1)
  IF (ABS(TIME2-TIME1) > 0.0001 .AND. ABS(TIME2-TIME1) < 1000.0) THEN
    WRITE (NW,"(1X,I9,A,F9.5,A)") NUM_F,' function calls were made in ',TIME2-TIME1, &
      ' seconds.'
    WRITE (NW,"(A)") ' RESULT = '
  ELSE
    WRITE (NW,"(1X,I9,A,ES14.5,A)") NUM_F,' function calls were made in ',TIME2-TIME1, &
      ' seconds.'
  ENDIF
ENDIF

```

```

        WRITE (NW,"(A)") ' RESULT ='
    ENDIF
    CALL FM_PRINT(RESULT)
ENDIF
KW = KWSAVE
R_LEVEL = R_LEVEL - 1

END SUBROUTINE FM_INTEGRATE

SUBROUTINE FM_INT_FAIL(N,A,B,TOL,M,ERR,VAL,NW)
USE FMZM
IMPLICIT NONE
INTEGER :: N,M,NW
TYPE (FM) :: A,B,TOL,ERR,VAL

WRITE (NW,*) ' '
WRITE (NW,*) ' FM_INTEGRATE failed -- no convergence in ',M,' iterations.'
WRITE (NW,*) ' UNKNOWN has been returned in RESULT.'
WRITE (NW,*) ' Possible causes: (1) highly oscillatory integrand'
WRITE (NW,*) '                (2) non-convergent integral'
WRITE (NW,*) '                (3) integrable singularity in the interior of interval (A,B)'
WRITE (NW,*) '                (4) narrow spike in the interior of interval (A,B)'
WRITE (NW,*) '                (5) integral too close to zero'
WRITE (NW,*) ' A possible remedy for the last 3 is to split the integral into two pieces,'
WRITE (NW,*) ' making two calls to FM_INTEGRATE and then adding the two results.'
WRITE (NW,*) ' Put singularities or spikes at the endpoints of the intervals of integration.'
WRITE (NW,*) ' '
WRITE (NW,*) ' Function N = ',N,'.    A, B ='
CALL FM_PRINT(A)
CALL FM_PRINT(B)
WRITE (NW,*) ' TOL ='
CALL FM_PRINT(TOL)
WRITE (NW,*) ' The last integral approximation ='
CALL FM_PRINT(VAL)
WRITE (NW,*) ' The estimated relative error in the last integral approximation ='
CALL FM_PRINT(ERR)
WRITE (NW,*) ' '

END SUBROUTINE FM_INT_FAIL

SUBROUTINE FM_F_FAIL(X,N,NW)
USE FMZM
IMPLICIT NONE
INTEGER :: N,NW
TYPE (FM) :: X

WRITE (NW,*) ' '
WRITE (NW,*) ' FM_INTEGRATE failed -- F(X,N) gave UNKNOWN for N = ',N,' and X ='
CALL FM_PRINT(X)
WRITE (NW,*) ' Check the limits of integration, function number (N), and' // &
' function definition.'
WRITE (NW,*) ' Be careful of rounding at an irrational endpoint producing an' // &
' illegal function argument.'
WRITE (NW,*) ' Example: Integrate log( cos( t ) ) from 0 to pi/2'
WRITE (NW,*) ' At some precisions the computed value of pi/2 rounds up, giving' // &
' a small negative cos(t),'
WRITE (NW,*) ' which then causes the log function to return UNKNOWN.'
WRITE (NW,*) ' Possible fixes:'

```

```

WRITE (NW,*) '      Change variables to get an integral from 0 to 1'
WRITE (NW,*) '      Change the function to log( abs( cos( t ) ) )'
WRITE (NW,*) '      Compute pi/2 with rounding toward -infinity'
WRITE (NW,*) ' '

```

```

END SUBROUTINE FM_F_FAIL

```

```

SUBROUTINE FM_SPIKE(F,N,A,B,XMAX,FMAX,NUM_F,SPIKE_FOUND,KPRT,NW)
USE FMVALS
USE FMZM
IMPLICIT NONE

```

```

! look for an interior singularity or tall spike in F(X,N).
! After getting no convergence in M iterations in FM_INTEGRATE, FMAX = ABS(F(XMAX,N)) was
! the largest magnitude found for F on the interval (A+D,B-D), where D = ABS(B-A)/100.

```

```

TYPE (FM) :: A,B,XMAX,FMAX
TYPE (FM), EXTERNAL :: F
INTEGER :: N,NUM_F,KPRT,NW
LOGICAL :: SPIKE_FOUND
TYPE (FM), SAVE :: AB,AVERAGE,D,DX,FPMAX,EPS,F1,F2,H,T,X1,XPMAX
INTEGER :: J

```

```

SPIKE_FOUND = .FALSE.
IF (KPRT >= 2) THEN
  WRITE (NW,"(A)") ' Enter FM_SPIKE'
ENDIF

```

```

DX = ABS(B-A)/100
H = MIN(XMAX-A,B-XMAX)/2

```

```

AVERAGE = 0
DO J = 1, 99
  X1 = A + (J*(B-A))/100
  F1 = F(X1,N)
  NUM_F = NUM_F + 1
  IF (F1%MFM%MP(2) == MUNKNO) THEN
    XMAX = X1
    FMAX = ABS(F1)
    SPIKE_FOUND = .TRUE.
    RETURN
  ELSE
    AVERAGE = AVERAGE + ABS(F1)
  ENDIF
ENDDO
AVERAGE = AVERAGE/99

```

```

! Search for a singularity in F.

```

```

DO
  X1 = XMAX + H
  IF (X1 > A + DX .AND. X1 < B - DX) THEN
    F1 = F(X1,N)
    NUM_F = NUM_F + 1
    IF (F1%MFM%MP(2) == MUNKNO) THEN
      XMAX = X1
      FMAX = ABS(F1)
      SPIKE_FOUND = .TRUE.
    ENDIF
  ENDIF

```

```

        RETURN
    ENDF
    IF (ABS(F1) > FMAX) THEN
        XMAX = X1
        FMAX = ABS(F1)
        H = -(H*14)/10
    ENDF
ENDF
H = -(H*10)/14
AB = MAX(ABS(A),ABS(B))
CALL FM_ULP(AB, EPS)
IF (ABS(H) < ABS(EPS)) THEN
    IF (FMAX > 50*AVERAGE) THEN
        SPIKE_FOUND = .TRUE.
        RETURN
    ELSE
        EXIT
    ENDF
ENDF
ENDDO

```

! Search for a singularity in F'.

```

FPMAX = -1
AVERAGE = 0
AB = MAX(ABS(A),ABS(B))
CALL FM_ULP(AB, EPS)
EPS = SQRT(EPS)
DO J = 1, 99
    X1 = A + (J*(B-A))/100
    IF (X1-EPS <= A) CYCLE
    F1 = F(X1-EPS,N)
    IF (X1+EPS >= B) CYCLE
    F2 = F(X1+EPS,N)
    NUM_F = NUM_F + 2
    IF (F1%MFM%MP(2) == MUNKNO .OR. F2%MFM%MP(2) == MUNKNO) THEN
        IF (F1%MFM%MP(2) == MUNKNO) THEN
            XMAX = X1 - EPS
            FMAX = ABS(F1)
        ELSE
            XMAX = X1 + EPS
            FMAX = ABS(F2)
        ENDF
        SPIKE_FOUND = .TRUE.
        RETURN
    ELSE
        D = ABS((F2-F1)/(2*EPS))
        AVERAGE = AVERAGE + D
        IF (D > FPMAX) THEN
            XPMAX = X1
            FPMAX = D
        ENDF
    ENDF
ENDDO
AVERAGE = AVERAGE/99

H = MIN(XPMAX-A, B-XPMAX)/2
DO

```