```fortran
      SUBROUTINE ZM_ROOTS(NR,F,NF,N_FOUND,LIST_OF_ROOTS,KPRT,KU)
      USE FMVALS
      USE FMZM
      IMPLICIT NONE

!  This routine searches for NR roots of F(X,NF) = 0.
!  NF is the function number in case roots to several functions are needed.

!  N_FOUND is returned as the number of roots found.
!  LIST_OF_ROOTS is an array returned with the roots found.  They are complex type (zm) numbers,
!                even when the actual root is real.

!  KPRT  controls printing within the routine:
!        KPRT = 0 for no output
!        KPRT = 1 for the approximation to each root to be printed as they are found.

!  KU    is the unit number for output.

!  The search for roots begins with fairly small magnitude complex values, so small roots are
!  often found before larger roots, but there is no guarantee of this, and the order in which
!  the roots are found is fairly random.  The user can sort LIST_OF_ROOTS and print them after
!  all have been found.

!  The secant method often fails to converge to any root for a given pair of starting points.
!  This routine may call ZM_ROOT1 many more than NR times before NR roots are found.  It can
!  also happen that ZM_ROOTS eventually gives up and returns N_FOUND < NR roots.

!  The user's function F is divided by the product of (X - LIST_OF_ROOTS(j)) over the roots that
!  have been found so far.  This tries keep the ZM_ROOT1 routine from returning to a root that is
!  already on the list (unless it is a root of multiplicity M > 1).

      TYPE (ZM), EXTERNAL :: F
      INTEGER :: J, KU, KPRT, KWARN_SAVE, NDIG_OF_ROOTS, NDSAVE, NF, NR, N_FOUND
      DOUBLE PRECISION :: VALUE
      LOGICAL :: REMOVE_PREVIOUS_ROOTS, RETRY
      TYPE (ZM) :: LIST_OF_ROOTS(NR)
      TYPE (ZM), SAVE :: AX, BX, X1

      CALL FM_ENTER_USER_ROUTINE

!             Raise precision slightly.

      NDSAVE = NDIG
      NDIG = NDIG + NGRD52
      KWARN_SAVE = KWARN
      KWARN = 0
      RETRY = .FALSE.

      N_FOUND = 0
      LIST_OF_ROOTS = TO_ZM(' UNKNOWN + UNKNOWN i ')
      NDIG_OF_ROOTS = NDIG

      DO J = 1, 10*NR
         IF (RETRY) THEN
             CALL FM_RANDOM_NUMBER(VALUE)
```

```
            IF (MOD(J,4) == 0) THEN
                AX = TO_ZM(' 1.1 + 1.2 i ')*((2+J)*VALUE+1)
            ELSE IF (MOD(J,4) == 1) THEN
                AX = TO_ZM(' 1.1 - 0.8 i ')*((2+J)*VALUE+1)
            ELSE IF (MOD(J,4) == 2) THEN
                AX = TO_ZM(' -0.8 - 1.2 i ')*((2+J)*VALUE+1)
            ELSE IF (MOD(J,4) == 3) THEN
                AX = TO_ZM(' -1.1 + 0.8 i ')*((2+J)*VALUE+1)
            ENDIF
            BX = TO_ZM(' 0.87 + 0.5 i ')*AX
        ELSE
            AX = TO_ZM(' 1.1 + 1.2 i ')
            BX = TO_ZM(' 3.4 + 4.5 i ')
        ENDIF
        REMOVE_PREVIOUS_ROOTS = .TRUE.
        CALL ZM_ROOT1(AX,BX,NR,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS,  &
                      X1,-1,KU)
        IF (.NOT. (IS_UNKNOWN(ABS(X1)) .OR. IS_OVERFLOW(ABS(X1))) ) THEN
            N_FOUND = N_FOUND + 1
            LIST_OF_ROOTS(N_FOUND) = X1

!             Some roots, primarily multiple roots, may have lost some accuracy due to the
!             divisions by previously found roots.  Refine them using F without dividing.

            AX = LIST_OF_ROOTS(N_FOUND)*(1+1.0D-10)
            BX = LIST_OF_ROOTS(N_FOUND)*(1+1.0D-15)
            REMOVE_PREVIOUS_ROOTS = .FALSE.
            CALL ZM_ROOT1(AX,BX,NR,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,  &
                          NDIG_OF_ROOTS,X1,-1,KU)
            IF (ABS(REAL(X1)) < EPSILON(TO_FM(1))*ABS(X1)) X1 = CMPLX( TO_FM(0) , AIMAG(X1) )
            IF (ABS(AIMAG(X1)) < EPSILON(TO_FM(1))*ABS(X1)) X1 = CMPLX( REAL(X1) , TO_FM(0) )
            LIST_OF_ROOTS(N_FOUND) = X1

            IF (KPRT > 0) THEN
                WRITE (*,"(A,I9,A,I6,A)") ' ZM_ROOTS.  Function ',NF,' Root ',N_FOUND,' ='
                CALL ZM_PRINT(X1)
            ENDIF
            IF (N_FOUND == NR) EXIT

!             Check to see if the conjugate of this root is also a root.

            IF (ABS(AIMAG(X1)) < 100*EPSILON(TO_FM(1))) CYCLE
            AX = CONJG(AX)
            BX = CONJG(BX)
            REMOVE_PREVIOUS_ROOTS = .TRUE.
            CALL ZM_ROOT1(AX,BX,NR,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,  &
                          NDIG_OF_ROOTS,X1,-1,KU)
            IF (.NOT. (IS_UNKNOWN(ABS(X1)) .OR. IS_OVERFLOW(ABS(X1))) ) THEN
                N_FOUND = N_FOUND + 1
                LIST_OF_ROOTS(N_FOUND) = X1
                AX = LIST_OF_ROOTS(N_FOUND)*(1+1.0D-10)
                BX = LIST_OF_ROOTS(N_FOUND)*(1+1.0D-15)
                REMOVE_PREVIOUS_ROOTS = .FALSE.
                CALL ZM_ROOT1(AX,BX,NR,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,  &
                              NDIG_OF_ROOTS,X1,-1,KU)
                IF (ABS(REAL(X1)) < EPSILON(TO_FM(1))*ABS(X1)) X1 = CMPLX( TO_FM(0) , AIMAG(X1) )
```

```
                IF (ABS(AIMAG(X1)) < EPSILON(TO_FM(1))*ABS(X1)) X1 = CMPLX( REAL(X1) , TO_FM(0) )
                LIST_OF_ROOTS(N_FOUND) = X1
                IF (KPRT > 0) THEN
                    WRITE (*,"(A,I9,A,I6,A)") ' ZM_ROOTS.  Function ',NF,' Root ',N_FOUND,' ='
                    CALL ZM_PRINT(X1)
                ENDIF
                IF (N_FOUND == NR) EXIT
            ENDIF
            RETRY = .FALSE.
        ELSE
            RETRY = .TRUE.
        ENDIF
     ENDDO

!             Round the roots to the user's precision.

     DO J = 1, N_FOUND
        X1 = LIST_OF_ROOTS(J)
        CALL ZM_EQU(X1,LIST_OF_ROOTS(J),NDIG,NDSAVE)
     ENDDO

     NDIG = NDSAVE
     KWARN = KWARN_SAVE
     CALL FM_EXIT_USER_ROUTINE
     END SUBROUTINE ZM_ROOTS

     SUBROUTINE ZM_ROOT1(AX,BX,NR,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,  &
                         NDIG_OF_ROOTS,ROOT,KPRT,KU)
     USE FMVALS
     USE FMZM
     IMPLICIT NONE

!  This is a special version of ZM_SECANT, modified to work with ZM_ROOTS so that some calls
!  will use F and others will use F divided by all the (x - r) terms of the roots found so far.

!  REMOVE_PREVIOUS_ROOTS is a logical input variable telling this routine whether or not to
!  divide F by the product of (X - LIST_OF_ROOTS(j)) over the roots that have been found so far.
!  This tries keep the ZM_ROOT1 routine from returning to a root that is already on the list
!  (unless it is a root of multiplicity M > 1).

!  This routine searches for a root of F(X,NF) = 0 using AX and BX as starting points.
!  AX and BX are complex, and the search can fail if AX and BX are not close enough to any roots
!  or if the function is badly behaved.

!  When a root is found, ZM_ROOT1 tries to return full accuracy even in the case of multiple
!  or closely-spaced roots, by raising precision above the user's level.

!  ROOT  is the value returned as the approximate root of the equation.

!  KPRT  controls printing within the routine:
!        KPRT = -1 for no output
!        KPRT =  0 for no output except warning and error messages.
!        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.
```

```
!  KU    is the unit number for output.

      TYPE (ZM)           :: AX, BX, ROOT
      TYPE (ZM), EXTERNAL :: F, ZM_FPRIME, ZM_ROOT_F
      CHARACTER (80) :: STR
      DOUBLE PRECISION :: VALUE
      INTEGER :: J, JSET, K, KU, KPRT, KWARN_SAVE, MAXIT, N_FOUND, NDIG_OF_ROOTS, NDSAVE, NF, NR
      LOGICAL :: REMOVE_PREVIOUS_ROOTS, USE_F_OVER_FP
      TYPE (ZM) :: LIST_OF_ROOTS(NR)
      TYPE (ZM), SAVE :: F1, F1OLD, F2, FP0, FP1, FS, S, X1, X1OLD, X2, X3
      TYPE (FM), SAVE :: ERR, ERR1, TOL

      CALL FM_ENTER_USER_ROUTINE
      IF (KPRT == 2) THEN
          WRITE (KU,"(A)") ' '
          WRITE (KU,"(A)") ' ZM_ROOT1.  Begin trace of all iterations.'
      ENDIF

!             Raise precision slightly.

      NDSAVE = NDIG
      NDIG = NDIG + NGRD52
      CALL ZM_EQU(AX,X1,NDSAVE,NDIG)
      CALL ZM_EQU(BX,X2,NDSAVE,NDIG)
      KWARN_SAVE = KWARN
      KWARN = 0

      MAXIT = 1000
      JSET = 50
      ERR = 1
      TOL = 100*EPSILON(ERR)
      USE_F_OVER_FP = .FALSE.
      F1 = ZM_ROOT_F(X1,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
      F2 = ZM_ROOT_F(X2,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)

!             Check for legal function values.

      IF (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1))) THEN
          DO J = 1, 3
             X3 = ((TO_FM(4) - J)/4)*X1 + (1-(TO_FM(4) - J)/4)*X2
             F1 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
             IF (.NOT. (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1)))) THEN
                 X1 = X3
                 EXIT
             ENDIF
          ENDDO
          IF (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1))) THEN
              DO J = 1, 3
                 X3 = ((TO_FM(4) + J)/4)*X1 + (1-(TO_FM(4) + J)/4)*X2
                 F1 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
                 IF (.NOT. (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1)))) THEN
                     X1 = X3
                     EXIT
                 ENDIF
                 X3 = (1-(TO_FM(4) + J)/4)*X1 + ((TO_FM(4) + J)/4)*X2
```

```
            F1 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
            IF (.NOT. (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1)))) THEN
                X1 = X3
                EXIT
            ENDIF
         ENDDO
     ENDIF
 ENDIF
 IF (IS_UNKNOWN(ABS(F1)) .OR. IS_OVERFLOW(ABS(F1))) THEN
     IF (KPRT >= 0) THEN
         WRITE (KU,"(A)") ' '
         WRITE (KU,"(A,A)") ' Invalid input for ZM_ROOT1. ',  &
                            ' Unknown or overflowed function value for AX ='
         CALL ZM_PRINT(X1)
         WRITE (KU,"(A)") ' '
     ENDIF
     J = 0
     X2 = TO_ZM(' UNKNOWN + UNKNOWN i ')
     ERR = TO_ZM(' UNKNOWN + UNKNOWN i ')
     GO TO 110
 ENDIF

 IF (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2))) THEN
     DO J = 1, 3
        X3 = ((TO_FM(4) - J)/4)*X1 + (1-(TO_FM(4) - J)/4)*X2
        F2 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
        IF (.NOT. (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2)))) THEN
            X2 = X3
            EXIT
        ENDIF
     ENDDO
     IF (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2))) THEN
         DO J = 1, 3
            X3 = ((TO_FM(4) + J)/4)*X1 + (1-(TO_FM(4) + J)/4)*X2
            F2 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
            IF (.NOT. (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2)))) THEN
                X2 = X3
                EXIT
            ENDIF
            X3 = (1-(TO_FM(4) + J)/4)*X1 + ((TO_FM(4) + J)/4)*X2
            F2 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
            IF (.NOT. (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2)))) THEN
                X2 = X3
                EXIT
            ENDIF
         ENDDO
     ENDIF
 ENDIF
 IF (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2))) THEN
     IF (KPRT >= 0) THEN
         WRITE (KU,"(A)") ' '
         WRITE (KU,"(A,A)") ' Invalid input for ZM_ROOT1. ',  &
                            ' Unknown or overflowed function value for BX ='
         CALL ZM_PRINT(X2)
         WRITE (KU,"(A)") ' '
     ENDIF
```

```
          J = 0
          X2 = TO_ZM(' UNKNOWN + UNKNOWN i ')
          ERR = TO_ZM(' UNKNOWN + UNKNOWN i ')
          GO TO 110
      ENDIF

!              Secant does not do well if the magnitudes of the two starting function values differ
!              by too much.  Adjust if necessary.

      DO J = 1, 10
         IF (ABS(F2/F1) > 10) THEN
             X2 = (X1 + X2)/2
             F2 = ZM_ROOT_F(X2,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
         ELSE IF (ABS(F1/F2) > 10) THEN
             X1 = (X1 + X2)/2
             F1 = ZM_ROOT_F(X1,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
         ELSE
             EXIT
         ENDIF
      ENDDO

      IF (KPRT == 2) THEN
          STR = ZM_FORMAT('ES20.10','ES20.10',F1)
          WRITE (KU,"('  J =',I3,3X,'f(AX) = ',A,'    x:')") 0,TRIM(STR)
          CALL ZM_PRINT(X1)
          STR = ZM_FORMAT('ES20.10','ES20.10',F2)
          WRITE (KU,"('  J =',I3,3X,'f(BX) = ',A,'    x:')") 0,TRIM(STR)
          CALL ZM_PRINT(X2)
      ENDIF

!              This loop does the iteration.

      DO J = 1, MAXIT

         IF (F2-F1 /= 0.0) THEN
             X3 = X2 - F2*(X2-X1)/(F2-F1)
         ELSE
             X3 = X2 + 1
         ENDIF

!              Multiple roots cause very slow convergence and loss of accuracy.
!              If the slope is very small, try to improve convergence and accuracy by using
!              the (slower) function  f(x)/f'(x)  which has no multiple roots.

         X1OLD = X1
         F1OLD = F1
         IF ( (ABS((F2-F1)/(X2-X1)) < 1.0D-2 .AND. ABS(F2) < 1.0D-4 .AND.  &
               ABS(F2*(X2-X1)/(F2-F1)) < 1.0D-4*ABS(X2)) .OR. USE_F_OVER_FP) THEN
             USE_F_OVER_FP = .TRUE.
             X1 = X2
             X2 = X3
             F1 = F2
             IF (REMOVE_PREVIOUS_ROOTS) THEN
                 FP0 = ZM_FPRIME(0,X3,F,NF)
                 FP1 = ZM_FPRIME(1,X3,F,NF)
                 S = 0
```

```
                DO K = 1, N_FOUND
                   S = S + 1/(X3-LIST_OF_ROOTS(K))
                ENDDO
                F2 = FP0 / ( FP1 - FP0*S )
             ELSE
                F2 = ZM_FPRIME(0,X3,F,NF) / ZM_FPRIME(1,X3,F,NF)
             ENDIF
          ELSE
             X1 = X2
             X2 = X3
             F1 = F2
             F2 = ZM_ROOT_F(X3,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)

!             If the function has a large number of roots, like a high-degree polynomial, then
!             from a distance the function looks like it has multiple roots even though once we
!             get closer the roots appear distinct.  This can slow the rate of convergence in
!             the early iterations.  Try an Aitken extrapolation once every few steps to try to
!             speed up this initial phase of convergence.

             IF (MOD(J,5) == 0 .AND. X2-2*X1+X1OLD /= 0) THEN
                S = X2 - (X2-X1)/(X2-2*X1+X1OLD)
                FS = ZM_ROOT_F(S,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
                IF (ABS(FS) < MAX(ABS(F1),ABS(F2))) THEN
                    X1 = X2
                    F1 = F2
                    X2 = S
                    F2 = FS
                ENDIF
             ENDIF
          ENDIF

!             If F2 is one of the FM non-numbers, +-underflow, +-overflow, unknown,
!             then replace it by something representable, so that the next x3 will be
!             closer to x1.  Also swap x1 and x2, making the bad x go away first.

          IF (IS_UNKNOWN(ABS(F2)) .OR. IS_OVERFLOW(ABS(F2))) THEN
             F2 = -2*F1
             X3 = X1
             X1 = X2
             X2 = X3
             X3 = F1
             F1 = F2
             F2 = X3
          ENDIF

!             A common failure mode for secant is to get into a pattern that repeats x1 and x2
!             close together with nearly equal function values and x3 farther away with much
!             larger function value.  Check for this, and re-start the iteration by choosing
!             a different x3.

          IF (ABS(F2) > 100*MAX(ABS(F1OLD),ABS(F1)) .AND. J >= JSET) THEN
             IF (JSET >= 200) THEN
                MAXIT = J
                EXIT
             ENDIF
             JSET = JSET + 50
```

```
                CALL FM_RANDOM_NUMBER(VALUE)
                VALUE = 9*VALUE - 4
                X2 = VALUE*X1 + (1-VALUE)*X2
                F2 = ZM_ROOT_F(X2,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
            ENDIF

            IF (KPRT == 2) THEN
                STR = ZM_FORMAT('ES20.10','ES20.10',F2)
                WRITE (KU,"('  J =',I3,4X,'f(x) = ' ,A,'     x:')") J,TRIM(STR)
                CALL ZM_PRINT(X2)
            ENDIF

            ERR1 = ERR
            IF (X2 /= 0.0) THEN
                ERR = ABS((X2-X1)/X2)
            ELSE
                ERR = ABS(X2-X1)
            ENDIF

!             If the error is less than the tolerance, double check to make sure the previous
!             error was small along with the current function value.  Some divergent iterations
!             can get err < tol without being close to a root.

            IF (ERR < TOL .OR. F2 == 0) THEN
                IF (ERR1 > SQRT(SQRT(TOL)) .AND. ABS(F2) > SQRT(EPSILON(TO_FM(1)))) THEN
                    IF (KPRT >= 0) THEN
                        WRITE (KU,"(/' Possible false convergence in ZM_ROOT1 after',I5,"//  &
                                  "' iterations.  ','Last two approximations =')") J
                        CALL ZM_PRINT(X1)
                        CALL ZM_PRINT(X2)
                        WRITE (KU,"(/' These agree to the convergence tolerance, but the previous"//  &
                                  " iteration was suspiciously far away:')")
                        CALL ZM_PRINT(X1OLD)
                        WRITE (KU,"(/' and the function value of the last iteration was"//  &
                                  " suspiciously far from zero:')")
                        CALL ZM_PRINT(F2)
                        WRITE (KU,"(/' Unknown has been returned.')")
                    ENDIF
                    X2 = TO_ZM(' UNKNOWN + UNKNOWN i ')
                ENDIF
                GO TO 110
            ENDIF

         ENDDO

!             No convergence after maxit iterations.

         IF (KPRT >= 0) THEN
             WRITE (KU,"(/' No convergence in ZM_ROOT1 after',I5,' iterations.  ',"//  &
                       "'Last two approximations =')") MAXIT
             CALL ZM_PRINT(X1)
             CALL ZM_PRINT(X2)
             WRITE (KU,"(/' Unknown has been returned.')")
         ENDIF
         X2 = TO_ZM(' UNKNOWN + UNKNOWN i ')
```

```
!             The root was found.

  110 CALL ZM_EQU(X2,ROOT,NDIG,NDSAVE)
      NDIG = NDSAVE
      IF (KPRT >= 1) THEN
          CALL FM_ULP(ABS(X2),ERR1)
          IF (.NOT.( IS_UNKNOWN(ERR1) .OR. IS_UNDERFLOW(ERR1) )) THEN
              ERR1 = ABS(ERR1/X2)/2
              IF (ERR < ERR1) ERR = ERR1
          ENDIF
          STR = FM_FORMAT('ES16.6',ERR)
          WRITE (KU,"(A)") ' '
          WRITE (KU,"('  ZM_ROOT1.   Function ',I3,I7,' iterations.'/17X"// &
                    "'Estimated relative error =',A,',    Root:')") NF,J,TRIM(STR)
          CALL ZM_PRINT(ROOT)
          WRITE (KU,"(A)") ' '
      ENDIF

      KWARN = KWARN_SAVE
      CALL FM_EXIT_USER_ROUTINE
      END SUBROUTINE ZM_ROOT1

      FUNCTION ZM_ROOT_F(X,F,NF,REMOVE_PREVIOUS_ROOTS,N_FOUND,LIST_OF_ROOTS,NDIG_OF_ROOTS)
      USE FMVALS
      USE FMZM
      IMPLICIT NONE

!  ZM_ROOT_F is used here to evaluate the user's function F and divide F by the product of
!  (X - LIST_OF_ROOTS(j)) over the roots that have been found so far.  This should keep the
!  ZM_ROOT1 routine from returning to a root that is already on the list (unless it is a
!  root of multiplicity M > 1).

!  When REMOVE_PREVIOUS_ROOTS is false, just evaluate F without doing the division.

!  X  is the argument to the function.
!  NF is the function number.

      INTEGER :: J, NDIG_OF_ROOTS, NF, N_FOUND
      LOGICAL :: REMOVE_PREVIOUS_ROOTS
      TYPE (ZM) :: LIST_OF_ROOTS(N_FOUND)
      TYPE (ZM) :: ZM_ROOT_F, X
      TYPE (ZM), EXTERNAL :: F
      TYPE (ZM), SAVE :: D

      CALL FM_ENTER_USER_FUNCTION(ZM_ROOT_F)

      IF (REMOVE_PREVIOUS_ROOTS) THEN
          ZM_ROOT_F = F(X,NF)
          DO J = 1, N_FOUND
             CALL ZM_EQU(LIST_OF_ROOTS(J),D,NDIG_OF_ROOTS,NDIG)
             IF (X /= D) ZM_ROOT_F = ZM_ROOT_F / (X - D)
          ENDDO
      ELSE
          ZM_ROOT_F = F(X,NF)
      ENDIF
```

```
CALL FM_EXIT_USER_FUNCTION(ZM_ROOT_F)
END FUNCTION ZM_ROOT_F
```