```
      PROGRAM TEST

!   This is a sample program using version 1.4 of the FMZM and FM_RATIONAL_ARITHMETIC modules
!   for doing exact rational arithmetic using the FM_RATIONAL derived type.

!   The program's output to the screen is also saved in file SampleFMrational.out.
!   The program checks all the results and the last line of the output file should be
!   "All results were ok."

      USE FMVALS
      USE FMZM
      USE FM_RATIONAL_ARITHMETIC

      IMPLICIT NONE

!             Declare the multiple precision variables.  The three types used in this program are:
!             (FM) for multiple precision real
!             (IM) for multiple precision integer
!             (FM_RATIONAL) for multiple precision rational

      TYPE (FM), SAVE                        :: DET_FM, ERROR, MAX_REL_ERROR
      TYPE (FM), SAVE, ALLOCATABLE           :: A_FM(:,:), B_FM(:), C_FM(:,:), X_FM(:)
      TYPE (IM), SAVE                        :: C1, C2
      TYPE (FM_RATIONAL), SAVE               :: CHECK, DET_RM, F_RM, T_RM
      TYPE (FM_RATIONAL), SAVE, ALLOCATABLE :: A_RM(:,:), B_RM(:), C_RM(:,:), X_RM(:)

!             Declare the other variables (not multiple precision).

      CHARACTER(80)  :: ST1
      CHARACTER(175) :: FMT
      INTEGER :: I, I_MAX, J, J_MAX, K, KOUT, KW_SAVE, N, KERROR, NERROR, P(4), Q(4),  &
                 ROOTS_FOUND
      REAL :: T1, T2
      DOUBLE PRECISION :: VALUE

!             Write output to the screen (unit *), and also to the file SampleFMrational.out.

      KOUT = 18
      OPEN (KOUT,FILE='SampleFMrational.out')

!             KW is the unit used for all automatically generated output from the FM routines.
!             This includes calls to the various print routines, as well as error messages.
!             KW should also default to screen output.

      KW_SAVE = KW

      CALL FM_SET(50)
      CALL FM_SETVAR(' KSWIDE = 100 ')
      NERROR = 0


!             1.  Find all rational roots of the equation
!                 f(t) = 21*t**5 + 43*t**4 - 113*t**3 - 46*t**2 + 49*t + 10 = 0.

!                 The rational root theorem says that if a polynomial with integer
!                 coefficients has rational roots, they must be of the form p/q where
!                 p divides the constant term (10 here) and q divides the high-order
```

```
!                 coefficient (21 here).

!                 This gives a short list of possibilities that can be quickly checked.
!                 p could be + or - { 1, 2, 5, 10 }, and q could be { 1, 3, 7, 21 }.
!                 That gives 2*4*4 = 32 possible roots to check.

!                 TO_FM_RATIONAL is a conversion function for creating FM_RATIONAL numbers.
!                 There are several versions, including 1 or 2 integer arguments, 1 or 2
!                 string arguments, etc.  See the user manual for all the options.

      FMT = "(///' Sample 1.  Find all rational roots: " //  &
            " f(t) = 21*t**5 + 43*t**4 - 113*t**3 - 46*t**2 + 49*t + 10 = 0'/)"
      WRITE (*   ,FMT)
      WRITE (KOUT,FMT)

      P = (/ 1, 2, 5, 10 /)
      Q = (/ 1, 3, 7, 21 /)
      ROOTS_FOUND = 0
      DO I = 2, 1, -1
         DO J = 1, 4
            DO K = 1, 4
               T_RM = (-1)**I * TO_FM_RATIONAL( P(J), Q(K) )
               F_RM = 21*T_RM**5 + 43*T_RM**4 - 113*T_RM**3 - 46*T_RM**2 + 49*T_RM + 10
               IF (F_RM == 0) THEN
                   WRITE (*   ,"(A)") "    Exact rational root found:"
                   CALL FM_PRINT_RATIONAL( T_RM )
                   WRITE (KOUT,"(A)") "    Exact rational root found:"
                   KW = KOUT
                   CALL FM_PRINT_RATIONAL( T_RM )
                   KW = KW_SAVE

!                  Check the results.

                   ROOTS_FOUND = ROOTS_FOUND + 1
                   IF (ROOTS_FOUND == 1) THEN
                       IF (.NOT.(T_RM == TO_FM_RATIONAL(' 2/3 '))) THEN
                           NERROR = NERROR + 1
                       ENDIF
                   ELSE IF (ROOTS_FOUND == 2) THEN
                       IF (.NOT.(T_RM == TO_FM_RATIONAL(' -5 / 7 '))) THEN
                           NERROR = NERROR + 1
                       ENDIF
                   ELSE IF (ROOTS_FOUND > 2) THEN
                           NERROR = NERROR + 1
                   ENDIF
               ENDIF
            ENDDO
         ENDDO
      ENDDO

      IF (NERROR > 0 .OR. ROOTS_FOUND /= 2) THEN
          WRITE (*   ,"(/' Error in sample case number 1.'/)")
          WRITE (KOUT,"(/' Error in sample case number 1.'/)")
      ENDIF


!             2.  Exact solution of linear systems (integer coefficients)
```

```
!                 Many linear systems of equations have integer coefficients, making the solutions
!                 rational.  Others have rational coefficients, also giving rational solutions.

!                 Generate several systems of the type that come from some kinds of least-squares
!                 problems.

!                 The coefficient matrix is NxN for N = 25, 50, 75, 100.

!                 Compare the accuracy and speed of the floating-point routine FM_LIN_SOLVE
!                 with the exact rational routine RM_LIN_SOLVE.

!                 Note that for systems with integer coefficients, it can be faster to find
!                 the exact rational solution than to find a 50-digit approximate solution,
!                 even though in the 100x100 case the numerators and denominators have over
!                 200 digits each.

      FMT = "(///' Sample 2.  Solve four NxN linear systems having small integer coefficients.')"
      WRITE (*   ,FMT)
      WRITE (KOUT,FMT)
      KERROR = 0

      DO N = 25, 100, 25
         WRITE (*   ,*) ' '
         WRITE (KOUT,*) ' '
         ALLOCATE(A_FM(N,N), B_FM(N), X_FM(N), A_RM(N,N), B_RM(N), X_RM(N))

         A_FM = 0
         B_FM = 0
         X_FM = 0
         A_RM = 0
         B_RM = 0
         X_RM = 0
         DO J = 1, N*N
            CALL FM_RANDOM_NUMBER(VALUE)
            I = N*VALUE + 1
            CALL FM_RANDOM_NUMBER(VALUE)
            K = N*VALUE + 1
            A_RM(I,I) = A_RM(I,I) + 1
            A_RM(I,K) = A_RM(I,K) - 1
            A_RM(K,K) = A_RM(K,K) + 1
            A_RM(K,I) = A_RM(K,I) - 1
            CALL FM_RANDOM_NUMBER(VALUE)
            B_RM(I) = B_RM(I) + (I-K) + INT(12*VALUE - 6)
            B_RM(K) = B_RM(K) - (I-K) + INT(12*VALUE - 6)
         ENDDO
         A_RM(N,1:N) = 0
         A_RM(N,N) = 1
         B_RM(N) = N
         A_FM = TO_FM( A_RM )
         B_FM = TO_FM( B_RM )

!             Solve the system with floating-point 50 significant arithmetic.

!             Routines with names like FM_LIN_SOLVE_RM are copies of the corresponding routines
!             (FM_LIN_SOLVE here) from the standard floating-point FM sample routine file.  The
!             ones with names ending "_RM" are available in the FM_RATIONAL_ARITHMETIC module.

         CALL CPU_TIME(T1)
```

```fortran
         CALL FM_LIN_SOLVE_RM(A_FM, X_FM, B_FM, N, DET_FM)
         CALL CPU_TIME(T2)

         FMT = "(/'    FM_LIN_SOLVE approximate solution for ',I4,' x',I4,' system in',"//  &
               "F12.2,' seconds.')"
         WRITE (*    ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*    ,*) '              Determinant ='
         WRITE (KOUT,*) '              Determinant ='
         CALL FM_PRINT(DET_FM)
         KW = KOUT
         CALL FM_PRINT(DET_FM)
         KW = KW_SAVE
         WRITE (*    ,*) '              X(1) ='
         WRITE (KOUT,*) '              X(1) ='
         CALL FM_PRINT(X_FM(1))
         KW = KOUT
         CALL FM_PRINT(X_FM(1))
         KW = KW_SAVE

!             Solve the system with exact rational arithmetic.

         CALL CPU_TIME(T1)
         CALL RM_LIN_SOLVE(A_RM, X_RM, B_RM, N, DET_RM)
         CALL CPU_TIME(T2)

         FMT = "(/'    RM_LIN_SOLVE       exact solution for ',I4,' x',I4,' system in',"//  &
               "F12.2,' seconds.')"
         WRITE (*    ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*    ,*) '              Determinant ='
         WRITE (KOUT,*) '              Determinant ='
         CALL FM_PRINT_RATIONAL(DET_RM)
         KW = KOUT
         CALL FM_PRINT_RATIONAL(DET_RM)
         KW = KW_SAVE
         WRITE (*    ,*) '              X(1) ='
         WRITE (KOUT,*) '              X(1) ='
         CALL FM_PRINT_RATIONAL(X_RM(1))
         KW = KOUT
         CALL FM_PRINT_RATIONAL(X_RM(1))
         KW = KW_SAVE

!                  Check the results.

         IF (.NOT.(ABS(DET_FM - TO_FM(DET_RM)) <= 1.0D-45*ABS(DET_FM))) THEN
             KERROR = KERROR + 1
         ENDIF
         DO J = 1, N
            IF (.NOT.(ABS(X_FM(J) - TO_FM(X_RM(J))) < 1.0D-45)) THEN
                KERROR = KERROR + 1
            ENDIF
         ENDDO

         DEALLOCATE(A_FM, B_FM, X_FM, A_RM, B_RM, X_RM)
      ENDDO

      IF (KERROR > 0) THEN
```

```
            WRITE (*    ,"(/' Error in sample case number 2.'/)")
            WRITE (KOUT,"(/' Error in sample case number 2.'/)")
            NERROR = NERROR + 1
        ENDIF


!             3.  Exact solution of linear systems (rational coefficients).

!                 Modify sample 2 so that the coefficients are rationals with numerators and
!                 denominators having no more than 2 digits.

!                 This causes the number of digits in the rational solution's numerators and
!                 denominators to get much larger, slowing RM_LIN_SOLVE compared to FM_LIN_SOLVE.

!                 Use smaller N's for the coefficient matrix here: NxN for N = 10, 20, 30, 40.

      FMT = "(///' Sample 3.  Solve four NxN linear systems, this time having non-integer" //  &
            " rational coefficients.'/)"
      WRITE (*    ,FMT)
      WRITE (KOUT,FMT)
      KERROR = 0

      DO N = 10, 40, 10
         WRITE (*    ,*) ' '
         WRITE (KOUT,*) ' '
         ALLOCATE(A_FM(N,N), B_FM(N), X_FM(N), A_RM(N,N), B_RM(N), X_RM(N))

         A_FM = 0
         B_FM = 0
         X_FM = 0
         A_RM = 0
         B_RM = 0
         X_RM = 0
         DO J = 1, N*N
            CALL FM_RANDOM_NUMBER(VALUE)
            I = N*VALUE + 1
            CALL FM_RANDOM_NUMBER(VALUE)
            K = N*VALUE + 1
            A_RM(I,I) = A_RM(I,I) + TO_FM_RATIONAL( I, ABS(K) + 1 )
            A_RM(I,K) = A_RM(I,K) - TO_FM_RATIONAL( I, ABS(K) + 1 )
            A_RM(K,K) = A_RM(K,K) + TO_FM_RATIONAL( I, ABS(K) + 1 )
            A_RM(K,I) = A_RM(K,I) - TO_FM_RATIONAL( I, ABS(K) + 1 )
            CALL FM_RANDOM_NUMBER(VALUE)
            B_RM(I) = B_RM(I) + (I-K) + INT(12*VALUE - 6)
            B_RM(K) = B_RM(K) - (I-K) + INT(12*VALUE - 6)
         ENDDO
         A_RM(N,1:N) = 0
         A_RM(N,N) = 1
         B_RM(N) = N
         A_FM = TO_FM( A_RM )
         B_FM = TO_FM( B_RM )

!             Solve the system with floating-point 50 significant arithmetic.

         CALL CPU_TIME(T1)
         CALL FM_LIN_SOLVE_RM(A_FM, X_FM, B_FM, N, DET_FM)
         CALL CPU_TIME(T2)
```

```
      FMT = "(/'    FM_LIN_SOLVE approximate solution for ',I4,' x',I4,' system in',"//  &
            "F12.2,' seconds.')"
      WRITE (*    ,FMT) N, N, T2-T1
      WRITE (KOUT,FMT) N, N, T2-T1
      WRITE (*    ,*) '             Determinant ='
      WRITE (KOUT,*) '             Determinant ='
      CALL FM_PRINT(DET_FM)
      KW = KOUT
      CALL FM_PRINT(DET_FM)
      KW = KW_SAVE
      WRITE (*    ,*) '             X(1) ='
      WRITE (KOUT,*) '             X(1) ='
      CALL FM_PRINT(X_FM(1))
      KW = KOUT
      CALL FM_PRINT(X_FM(1))
      KW = KW_SAVE

!            Solve the system with exact rational arithmetic.

      CALL CPU_TIME(T1)
      CALL RM_LIN_SOLVE(A_RM, X_RM, B_RM, N, DET_RM)
      CALL CPU_TIME(T2)

      FMT = "(/'    RM_LIN_SOLVE       exact solution for ',I4,' x',I4,' system in',"//  &
            "F12.2,' seconds.')"
      WRITE (*    ,FMT) N, N, T2-T1
      WRITE (KOUT,FMT) N, N, T2-T1
      WRITE (*    ,*) '             Determinant ='
      WRITE (KOUT,*) '             Determinant ='
      CALL FM_PRINT_RATIONAL(DET_RM)
      KW = KOUT
      CALL FM_PRINT_RATIONAL(DET_RM)
      KW = KW_SAVE
      WRITE (*    ,*) '             X(1) ='
      WRITE (KOUT,*) '             X(1) ='
      CALL FM_PRINT_RATIONAL(X_RM(1))
      KW = KOUT
      CALL FM_PRINT_RATIONAL(X_RM(1))
      KW = KW_SAVE

!                 Check the results.

      IF (.NOT.(ABS(DET_FM - TO_FM(DET_RM)) <= 1.0D-45*ABS(DET_FM))) THEN
          KERROR = KERROR + 1
      ENDIF
      DO J = 1, N
         IF (.NOT.(ABS(X_FM(J) - TO_FM(X_RM(J))) < 1.0D-45)) THEN
             KERROR = KERROR + 1
         ENDIF
      ENDDO

      DEALLOCATE(A_FM, B_FM, X_FM, A_RM, B_RM, X_RM)
   ENDDO

   IF (KERROR > 0) THEN
       WRITE (*    ,"(/' Error in sample case number 3.'/)")
       WRITE (KOUT,"(/' Error in sample case number 3.'/)")
       NERROR = NERROR + 1
```

```
      ENDIF


!             4.  Exact matrix inverse.

!                 One possible use for exact rational arithmetic is in looking for patterns
!                 in the answers.

!                 For an example, there is a formula for the determinant of the Hilbert matrix,
!                     a(j,k) = 1 / ( j + k - 1 ).
!                 We might have a similar matrix where no formula is known and we could try
!                 to discover one by examining factorizations of numerator and denominator.

!                 Try this for the Hilbert matrix with N = 1, 2, ..., 5

!                 N =              1         2             3                 4                    5
!                 det = 1 /        1        12          2160           6048000         266716800000
!                 factorization:   1     2^2 3     2^4 3^3 5     2^8 3^3 5^3 7     2^10 3^5 5^5 7^3

!                 There are some clues that might help us guess a formula, but the first thing
!                 to try is the On-line Encyclopedia of Integer Sequences, https://oeis.org/
!                 entering   1, 12, 2160, 6048000, 266716800000   produces several references
!                 to the inverse Hilbert matrix, where we can find a formula.

      FMT = "(///' Sample 4.  Examine determinants of several small Hilbert matrices.'/)"
      WRITE (*   ,FMT)
      WRITE (KOUT,FMT)
      KERROR = 0

      DO N = 1, 5
         WRITE (*   ,*) ' '
         WRITE (KOUT,*) ' '
         ALLOCATE(A_FM(N,N), C_FM(N,N), A_RM(N,N), C_RM(N,N))

         A_FM = 0
         C_FM = 0
         A_RM = 0
         C_RM = 0
         DO J = 1, N
            DO K = 1, N
               A_RM(J,K) = TO_FM_RATIONAL( 1, J+K-1 )
            ENDDO
         ENDDO
         A_FM = TO_FM( A_RM )

!             Invert the matrix with floating-point 50 significant arithmetic.

         CALL CPU_TIME(T1)
         CALL FM_INVERSE_RM(A_FM, N, C_FM, DET_FM)
         CALL CPU_TIME(T2)

         FMT = "(/'    FM_INVERSE approximate inverse for ',I4,' x',I4,' matrix in',"//  &
               "F12.2,' seconds.')"
         WRITE (*   ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*   ,*) '               Determinant ='
         WRITE (KOUT,*) '               Determinant ='
         CALL FM_PRINT(DET_FM)
```

```
         KW = KOUT
         CALL FM_PRINT(DET_FM)
         KW = KW_SAVE

!             Invert the matrix with exact rational arithmetic.

         CALL CPU_TIME(T1)
         CALL RM_INVERSE(A_RM, N, C_RM, DET_RM)
         CALL CPU_TIME(T2)

         FMT = "(/'    RM_INVERSE       exact inverse for ',I4,' x',I4,' matrix in',"//  &
               "F12.2,' seconds.')"
         WRITE (*    ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*    ,*) '               Determinant ='
         WRITE (KOUT,*) '               Determinant ='
         CALL FM_PRINT_RATIONAL(DET_RM)
         KW = KOUT
         CALL FM_PRINT_RATIONAL(DET_RM)
         KW = KW_SAVE

!                  Check the results.

         IF (.NOT.(ABS(DET_FM - TO_FM(DET_RM)) <= 1.0D-45*ABS(DET_FM))) THEN
             KERROR = KERROR + 1
         ENDIF
         DO J = 1, N
            DO K = 1, N
               IF (.NOT.(ABS(C_FM(J,K) - TO_FM(C_RM(J,K))) < 1.0D-45)) THEN
                   KERROR = KERROR + 1
               ENDIF
            ENDDO
         ENDDO

         DEALLOCATE(A_FM, C_FM, A_RM, C_RM)
      ENDDO

      IF (KERROR > 0) THEN
          WRITE (*    ,"(/' Error in sample case number 4.'/)")
          WRITE (KOUT,"(/' Error in sample case number 4.'/)")
          NERROR = NERROR + 1
      ENDIF


!             5.  Exact matrix inverse.

!                 Use the Hilbert matrix with some larger values for N, and compare times with FM.

!                 There are two things to notice about this case:
!                 (1) The Hilbert matrix becomes so ill-conditioned as N increases that even
!                     carrying over 50 digits with floating-point arithmetic in FM_INVERSE
!                     is not enough.  The maximum relative error for elements of C_FM are:
!                         N =        10          20          30          40
!                         error =  1.09e-50    7.10e-36    1.15e-20    2.19e-5
!                     If we wanted 50-digit accuracy from FM_INVERSE for N=40, we would need
!                     to set precision to at least 100 digits.
!                 (2) The numerators and denominators in the Hilbert matrix are all fairly
!                     small, so the modular method is faster than FM_INVERSE, even though
```

```
!                         the exact numerators and denominators have more than 50 digits.
!                         Timing will vary, but a typical result is for RM_INVERSE to run in
!                         less than half the time of FM_INVERSE.  The determinant for N = 40
!                         has over 900 digits in the denominator, but the largest element in
!                         the (integer-valued) inverse matrix has only 58 digits.

      FMT = "(///' Sample 5.  Examine determinants of several larger Hilbert matrices.'/)"
      WRITE (*   ,FMT)
      WRITE (KOUT,FMT)
      KERROR = 0

      DO N = 10, 40, 10
         WRITE (*   ,*) ' '
         WRITE (KOUT,*) ' '
         ALLOCATE(A_FM(N,N), C_FM(N,N), A_RM(N,N), C_RM(N,N))

         A_FM = 0
         C_FM = 0
         A_RM = 0
         C_RM = 0
         DO J = 1, N
            DO K = 1, N
               A_RM(J,K) = TO_FM_RATIONAL( 1, J+K-1 )
            ENDDO
         ENDDO
         A_FM = TO_FM( A_RM )

!             Invert the matrix with floating-point 50 significant arithmetic.

         CALL CPU_TIME(T1)
         CALL FM_INVERSE_RM(A_FM, N, C_FM, DET_FM)
         CALL CPU_TIME(T2)

         FMT = "(/'    FM_INVERSE approximate inverse for ',I4,' x',I4,' matrix in',"//  &
               "F12.2,' seconds.')"
         WRITE (*   ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*   ,*) '               1 / Determinant ='
         WRITE (KOUT,*) '               1 / Determinant ='
         CALL FM_PRINT(1/DET_FM)
         KW = KOUT
         CALL FM_PRINT(1/DET_FM)
         KW = KW_SAVE

!             Invert the matrix with exact rational arithmetic.

         CALL CPU_TIME(T1)
         CALL RM_INVERSE(A_RM, N, C_RM, DET_RM)
         CALL CPU_TIME(T2)

         FMT = "(/'    RM_INVERSE       exact inverse for ',I4,' x',I4,' matrix in',"//  &
               "F12.2,' seconds.')"
         WRITE (*   ,FMT) N, N, T2-T1
         WRITE (KOUT,FMT) N, N, T2-T1
         WRITE (*   ,*) '               Determinant ='
         WRITE (KOUT,*) '               Determinant ='
         CALL FM_PRINT_RATIONAL(DET_RM)
         KW = KOUT
```

```
      CALL FM_PRINT_RATIONAL(DET_RM)
      KW = KW_SAVE

!                Check the results.

!                Because the Hilbert matrix is pathologically ill-conditioned, even using
!                50 digits for the input to FM_INVERSE can give little accuracy in the
!                solution.  Use the mathematically exact values to check the results
!                from RM_INVERSE.

!                The correct determinant of the Hilbert matrix is always 1 / integer

!                = 1 / ( c(2n) / c(n)^4 ),  where c(n) = product( j^(n-j) ; j=1,n-1 )

      C1 = 1
      DO J = 1, N-1
         C1 = C1 * TO_IM(J)**TO_IM(N-J)
      ENDDO
      C2 = 1
      DO J = 1, 2*N-1
         C2 = C2 * TO_IM(J)**TO_IM(2*N-J)
      ENDDO
      C2 = C2 / C1**4
      IF (.NOT.(DET_RM == TO_FM_RATIONAL( TO_IM(1), C2 ))) THEN
          KERROR = KERROR + 1
          IF (KERROR == 1) THEN
              WRITE (*   ,"(/' Error in determinant for sample case number 5.'/)")
              WRITE (KOUT,"(/' Error in determinant for sample case number 5.'/)")
          ENDIF
      ENDIF

!                The correct elements of the inverse Hilbert matrix are:

!                c_rm(i,j) = (-1)^(i+j) * (i+j-1) * binomial(n+i-1,n-j) * binomial(n+j-1,n-i) *
!                                                   binomial(i+j-2,i-1)^2

      DO I = 1, N
         DO J = 1, N
            C1 = BINOMIAL(TO_FM(N+I-1),TO_FM(N-J)) * BINOMIAL(TO_FM(N+J-1),TO_FM(N-I))
            C2 = BINOMIAL(TO_FM(I+J-2),TO_FM(I-1))**2
            CHECK = (-1)**(I+J) * (I+J-1) * C1 * C2
            IF (.NOT.(C_RM(I,J) == CHECK)) THEN
                KERROR = KERROR + 1
                IF (KERROR == 1) THEN
                    WRITE (*   ,"(/' Error in inverse element for sample case number 5.'/)")
                    WRITE (KOUT,"(/' Error in inverse element for sample case number 5.'/)")
                ENDIF
            ENDIF
         ENDDO
      ENDDO

!           Check how badly conditioned each matrix is by finding the least accurate element
!           in the computed inverse matrix from FM_INVERSE.
!           Use the relative error between C_FM and C_RM, since the numbers are large.

      MAX_REL_ERROR = -1
      DO I = 1, N
         DO J = 1, N
```

```
            ERROR = ABS( ( C_FM(I,J) - TO_FM(C_RM(I,J)) ) / TO_FM(C_RM(I,J)) )
            IF (ERROR > MAX_REL_ERROR) THEN
                MAX_REL_ERROR = ERROR
                I_MAX = I
                J_MAX = J
            ENDIF
         ENDDO
      ENDDO

      FMT = "(/'     FM_INVERSE inverse matrix largest relative error"//   &
                 " was in row',I3,' column',I3,'.  Error =',A)"
      CALL FM_FORM('ES14.5',MAX_REL_ERROR,ST1)
      WRITE (*    ,FMT) I_MAX, J_MAX, ST1
      WRITE (KOUT,FMT) I_MAX, J_MAX, ST1

      DEALLOCATE(A_FM, C_FM, A_RM, C_RM)
   ENDDO

   IF (KERROR > 0) THEN
       WRITE (*    ,"(/' Error in sample case number 5.'/)")
       WRITE (KOUT,"(/' Error in sample case number 5.'/)")
       NERROR = NERROR + 1
   ENDIF


!             6.  Exact matrix inverse.

!                 Like sample 5, except use random A-matrices with numerators and denominators
!                 having no more than 2 digits.

   FMT = "(///' Sample 6.  Find four NxN inverse matrices, having random" //  &
         " 2-digit numerators and denominators.'/)"
   WRITE (*    ,FMT)
   WRITE (KOUT,FMT)
   KERROR = 0

   DO N = 10, 40, 10
      WRITE (*    ,*) ' '
      WRITE (KOUT,*) ' '
      ALLOCATE(A_FM(N,N), C_FM(N,N), A_RM(N,N), C_RM(N,N))

      A_FM = 0
      C_FM = 0
      A_RM = 0
      C_RM = 0
      DO I = 1, N
         DO J = 1, N
            CALL FM_RANDOM_NUMBER(VALUE)
            K = 198*VALUE - 99
            A_RM(I,J) = K
            CALL FM_RANDOM_NUMBER(VALUE)
            K = 99*VALUE + 1
            A_RM(I,J) = A_RM(I,J) / K
            A_FM(I,J) = TO_FM(A_RM(I,J))
         ENDDO
      ENDDO

!             Invert the matrix with floating-point 50 significant arithmetic.
```

```
      CALL CPU_TIME(T1)
      CALL FM_INVERSE_RM(A_FM, N, C_FM, DET_FM)
      CALL CPU_TIME(T2)

      FMT = "(/'    FM_INVERSE approximate solution for ',I4,' x',I4,' system in',"//  &
            "F12.2,' seconds.')"
      WRITE (*    ,FMT) N, N, T2-T1
      WRITE (KOUT,FMT) N, N, T2-T1
      WRITE (*    ,*) '              Determinant ='
      WRITE (KOUT,*) '              Determinant ='
      CALL FM_PRINT(DET_FM)
      KW = KOUT
      CALL FM_PRINT(DET_FM)
      KW = KW_SAVE

!             Invert the matrix with exact rational arithmetic.

      CALL CPU_TIME(T1)
      CALL RM_INVERSE(A_RM, N, C_RM, DET_RM)
      CALL CPU_TIME(T2)

      FMT = "(/'    RM_INVERSE       exact solution for ',I4,' x',I4,' system in',"//  &
            "F12.2,' seconds.')"
      WRITE (*    ,FMT) N, N, T2-T1
      WRITE (KOUT,FMT) N, N, T2-T1
      WRITE (*    ,*) '              Determinant ='
      WRITE (KOUT,*) '              Determinant ='
      CALL FM_PRINT_RATIONAL(DET_RM)
      KW = KOUT
      CALL FM_PRINT_RATIONAL(DET_RM)
      KW = KW_SAVE

!                  Check the results.

!                  These random matrices are not ill-conditioned, so the results can be checked
!                  by comparing the FM and RM inverses.

      IF (.NOT.(ABS(DET_FM - TO_FM(DET_RM)) <= 1.0D-45*ABS(DET_FM))) THEN
          KERROR = KERROR + 1
      ENDIF
      DO I = 1, N
         DO J = 1, N
            IF (.NOT.(ABS(C_FM(I,J) - TO_FM(C_RM(I,J))) < 1.0D-45)) THEN
                KERROR = KERROR + 1
            ENDIF
         ENDDO
      ENDDO

      DEALLOCATE(A_FM, C_FM, A_RM, C_RM)
   ENDDO

   IF (KERROR > 0) THEN
       WRITE (*    ,"(/' Error in sample case number 6.'/)")
       WRITE (KOUT,"(/' Error in sample case number 6.'/)")
       NERROR = NERROR + 1
   ENDIF
```

```
IF (NERROR == 0) THEN
    WRITE (*    ,"(//A/)") ' All results were ok -- no errors were found.'
    WRITE (KOUT,"(//A/)") ' All results were ok -- no errors were found.'
ELSE
    WRITE (*    ,"(//I3,A/)") NERROR,' error(s) found.'
    WRITE (KOUT,"(//I3,A/)") NERROR,' error(s) found.'
ENDIF

STOP
END PROGRAM TEST
```