```
!  This program finds the eigenvalues and eigenvectors for a random 20x20 real matrix.
!  The peculiar algorithm used is not a very good way to do this, but is a test for several
!  of the FM sample subroutines.

!  Some trace output is written to the screen as the program runs, and the eigenvalues and
!  eigenvectors are written to file EigenSystem.out at the end.

      module eigen_variables
      use fmzm

!  coeffs holds the coefficients for the characteristic polynomial of a matrix.
!  n_coeffs is the number of coefficients, and ndig_of_coeffs is their precision.

      type (fm), allocatable :: coeffs(:)
      integer, save :: n_coeffs, ndig_of_coeffs

      end module eigen_variables


      program test
      use fmzm
      implicit none

      type (fm), allocatable :: a(:,:)
      type (zm), allocatable :: eigenvectors(:,:), eigenvalues(:)
      type (fm) :: err
      integer :: seed(7)
      integer :: j, k, ku, kprt, n
      double precision :: xval
      real :: time1, time2

      n = 20
      k = 40
      kprt = 1
      ku = 6
      call fm_set(k)

      allocate(a(n, n), eigenvectors(n, n), eigenvalues(n), stat=j)
      if (j /= 0) stop
      call fm_setvar(" kswide = 110 ")

!             Generate an n x n matrix of random integers between -100 and +100.

      call fm_random_seed_size(j)
      seed(1:j) = 13
      seed(1) = n
      seed(2) = k
      call fm_random_seed_put(seed(1:j))
      do j = 1, n
         do k = 1, n
            call fm_random_number(xval)
            a(j, k) = int(200*xval - 100)
         enddo
      enddo

!             Find the Eigenvalues and Eigenvectors.
```

```
      call cpu_time(time1)

      call fm_eigensystem(a, n, eigenvectors, eigenvalues, kprt, ku)

      call cpu_time(time2)

!             Write the solution.

      open(22, file='EigenSystem.out')
      call fm_setvar(" kw = 22 ")
      do j = 1, n
         write (22,*) ' '
         write (22, "(a, i3, a)") ' Eigenvalue  ', j, ':'
         call zm_print(eigenvalues(j))
         write (22,*) ' '
         write (22, "(a, i3, a)") ' Eigenvector ', j, ':'
         do k = 1, n
            call zm_print(eigenvectors(k, j))
         enddo
      enddo
      write (22,*) ' '
      write (22,*) ' '

!             Find the norm of the residual,  a x - c x  for each eigenvalue c and eigenvector x.

      err = 0
      do j = 1, n
         err = err + sum(abs(matmul(to_zm(a(1:n, 1:n)), eigenvectors(1:n, j)) -  &
                         eigenvalues(j)*eigenvectors(1:n, j)))
      enddo
      write (*,*) ' '
      write (*, "(a, es16.7)") ' Norm of the residual = ', to_dp(err/n**2)
      write (*,*) ' '

      write (*,*) ' '
      write (*, "(a, f9.2, a)") '  Time to solve the system: ', time2-time1, ' seconds.'
      write (*,*) ' '
      write (*,*) ' The eigenvalues and eigenvectors are in file EigenSystem.out.'
      write (*,*) ' '

      close(22)
      stop

      end program test

      subroutine fm_eigensystem(a, n, eigenvectors, eigenvalues, kprt, ku)
      use fmvals
      use fmzm
      use eigen_variables
      implicit none

!  Find the eigensystem for real n x n matrix a.

!  eigenvalues(1:n) is returned with the (complex) eigenvalues.

!  eigenvectors(1:n, j) is returned with the (complex) eigenvector corresponding to eigenvalues(j),
!                       for j = 1, 2, ..., n.
```

```
!  a is a type (fm) multiprecision real array,
!  eigenvalues and eigenvectors are type (zm) multiprecision complex arrays.

!  kprt  controls printing within the routine:
!        kprt = 0 for no output
!        kprt = 1 for some trace output.

!  ku    is the unit number for output.

!  This routine does not handle cases with eigenvalues of multiplicity greater than 1,  since
!  some of those cases do not have a full set of n eigenvectors.


!  The algorithm used here is not the best way to compute eigensystems.  It is not even a good
!  way to do it.  But it was easy to code using existing FM routines for finding determinants
!  and all the roots of a polynomial, and it has a certain Rube Goldberg charm.

!  Algorithm:   1.  Generate the n+1 matrices  a - lambda*i  for lambda = 1, 2, ..., n+1
!               2.  Use fm_factor_lu to find the n+1 determinants for these matrices
!               3.  Use fm_lin_solve to find the n+1 coefficients of the Nth degree characteristic
!                   polynomial for a, using the points on the curve found in step 2.
!               4.  Use zm_roots to find the n (complex) roots of this polynomial.
!                   These roots are the eigenvalues.
!               5.  For each eigenvalue, do a few iterations of the inverse power method to find
!                   the corresponding eigenvector.

      integer :: i, j, jmax, k, kl, kprt, ku, kwarn_save, lambda, n, ndsave, n_found
      type (fm) :: a(n, n)
      type (zm) :: eigenvalues(n), eigenvectors(n, n)
      type (fm), save :: det, p, tol
      type (fm), allocatable :: a1(:,:), a2(:,:), eqn(:,:), identity(:,:), rhs(:)
      type (zm), allocatable :: z1(:,:), z2(:,:), zx(:), zt(:), list_of_roots(:)
      type (zm), save :: t, zdet
      integer, allocatable :: kswap(:)
      intent (in) :: a, n, kprt, ku
      intent (inout) :: eigenvectors, eigenvalues
      type (zm), external :: eigen_poly

!             Raise precision slightly.

      ndsave = ndig
      ndig = ndig + ngrd52
      kwarn_save = kwarn
      kwarn = 0

      allocate(a1(n, n), a2(n, n), eqn(n+1, n+1), identity(n, n), z1(n, n), z2(n, n),  &
               zx(n), zt(n), rhs(n+1), coeffs(n+1), list_of_roots(n), kswap(n), stat=j)
      if (j /= 0) then
          write (ku, "(/' Error in fm_eigensystem.  Unable to allocate arrays with n = ', i8/)") n
          stop
      endif

!             Copy a to a1 with higher precision, and generate an identity matrix.

      kl = 1
      do while (kl == 1)
         kl = 0
         tol = to_fm(mbase)**(-ndsave)
```

```
      identity = 0
      do i = 1, n
         do j = 1, n
            call fm_equ(a(i, j), a1(i, j), ndsave, ndig)
         enddo
         identity(i, i) = 1
      enddo

!           Generate the n+1 (real) n x n matrices  a - lambda*i  for lambda = 1, 2, ..., n+1,
!           and find the n+1 determinants for these matrices

      do lambda = 1, n+1
         a2 = a1 - lambda*identity
         call fm_factor_lu(a2, n, det, kswap)
         p = 1
         do j = n+1, 1, -1
            eqn(lambda, j) = p
            p = lambda*p
         enddo
         rhs(lambda) = det
      enddo
      n_coeffs = n + 1

!           Find the (real) coefficients of the characteristic polynomial for a.

      call fm_lin_solve(eqn, coeffs, rhs, n_coeffs, det)
      ndig_of_coeffs = ndig
      if (det == 0) then
          write (ku, "(/a, a/)") ' Error in fm_eigensystem.  Zero determinant -- no unique',  &
                                 ' characteristic polynomial.'
          stop
      endif

!           Use zm_roots to find all the (complex) roots of this polynomial.
!           These roots (in list_of_roots) are the eigenvalues.

      call zm_roots(n, eigen_poly, 1, n_found, list_of_roots, kprt, ku)

      if (n_found /= n) then
          if (kprt > 0) then
              write (ku, "(/a, i3, a, i3/)") ' In fm_eigensystem, n_found /= n.  n = ', n,  &
                                             '.  n_found =', n_found
              write (ku, "(/' Increase precision and try again.'/)")
          endif
          ndig = 2*ndig
          kl = 1
          cycle
      endif

!           Sort the eigenvalues so they have decreasing magnitude, and check that there were
!           no multiple roots.

      do i = 2, n
         jmax = i-1
         do j = i, n
            if (abs(list_of_roots(j)) > abs(list_of_roots(jmax))) jmax = j
         enddo
         t = list_of_roots(i-1)
```

```
            list_of_roots(i-1) = list_of_roots(jmax)
            list_of_roots(jmax) = t
         enddo
         do i = 2, n
            if (list_of_roots(i-1) == list_of_roots(i)) then
                write (ku, "(/a, a/)") ' Error in fm_eigensystem.',  &
                                       '  eigenvalues of multiplicity more than 1'
                call zm_print(list_of_roots(i))
                stop
            endif
         enddo

!             For each eigenvalue, do a few iterations of the inverse power method to find
!             the corresponding eigenvector.
!             Since the a matrix is real, conjugate eigenvalues have conjugate eigenvectors,
!             so some eigenvectors may not need the inverse power method.

         do i = 1, n
            if (i > 1) then
                if (list_of_roots(i) == conjg(list_of_roots(i-1))) then
                    do k = 1, n
                       call zm_equ(eigenvectors(k, i-1), t, ndsave, ndig)
                       call zm_equ(conjg(t), eigenvectors(k, i), ndig, ndsave)
                    enddo
                    call zm_equ(eigenvalues(i-1), t, ndsave, ndig)
                    call zm_equ(conjg(t), eigenvalues(i), ndig, ndsave)
                    cycle
                endif
            endif
            p = (1 + epsilon(to_fm(1))**0.3d0)
            z1 = a1
            do j = 1, n
               z1(j, j) = z1(j, j) - p*list_of_roots(i)
            enddo
            call zm_inverse(z1, n, z2, zdet)
            zx = 1
            do j = 1, 6
               zx = matmul(z2, zx)
               zx = zx / sqrt(abs(dot_product(zx, zx)))
               zt = matmul(to_zm(a1), zx) - list_of_roots(i)*zx
               p = sqrt(abs(dot_product(zt, zt)))
               if (p < tol) then
                   do k = 1, n
                      call zm_equ(zx(k), eigenvectors(k, i), ndig, ndsave)
                   enddo
                   call zm_equ(list_of_roots(i), eigenvalues(i), ndig, ndsave)
                   exit
               endif
               if (j == 6) then
                   if (kprt > 0) then
                       write (ku,"(/a, a, i3, a)")                    &
                                   ' In fm_eigensystem inverse iteration did not converge',  &
                                   ' for i = ', i, '.'
                       write (ku,"(a, es15.7, a, es15.7/)")  &
                                   ' Error =', to_dp(p), ' was not less than tol =', to_dp(tol)
                       write (ku, "(/' Increase precision and try again.'/)")
                   endif
                   ndig = 2*ndig
```

```
                kl = 1
                exit
             endif
          enddo
          if (kl == 1) exit
       enddo
    enddo

    deallocate(a1, a2, eqn, identity, z1, z2, zx, zt, rhs, coeffs, list_of_roots, kswap)

    ndig = ndsave
    kwarn = kwarn_save
    end subroutine fm_eigensystem

    function eigen_poly(x, nf)     result (return_value)
    use eigen_variables
    use fmvals
    use fmzm
    implicit none

!  x  is the argument to the function.
!  nf is the function number.

    integer :: j, nf
    type (zm) :: return_value, x
    type (fm), save :: d
    intent (in) x, nf

    if (nf == 1) then
        call fm_equ(coeffs(1), d, ndig_of_coeffs, ndig)
        return_value = d
        do j = 2, n_coeffs
           call fm_equ(coeffs(j), d, ndig_of_coeffs, ndig)
           return_value = return_value*x + d
        enddo
    else
        return_value = 3*x - 2
    endif

    end function eigen_poly
```