```fortran
      program test
      use fmzm
      implicit none

!  If the integrand is highly (or infinitely) oscillatory, standard numerical integration methods
!  often take too long when used directly.

!  In this program, we indirectly integrate 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 we use an extrapolation method 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.

!  To compute this integral to 50 significant digits, use 50 for the precision and 70 for the
!  number of roots.

      type (fm), save :: b, c, d, pi, r1, r2, results(100), s, terms(100), tol
      integer :: j, k, kprt, kw, n, nroots
      character(80) :: st1
      type (fm), external :: f

      kprt = 1
      kw = 12
      open(12, file='OscillateFM.out')
      call fm_setvar(' kw = 12 ')

      n = 50
      call fmset(n)
      nroots = 70

!             Integrate between pairs of roots.

      write (*, "(a)") ' '
      write (*, "(a)") ' Integrals between roots:'
      write (12, "(a)") ' '
      write (12, "(a)") ' Integrals between roots:'

      call fm_pi(pi)
      tol = to_fm(10)**(-n)

      do j = 1, nroots
         kprt = 1
         if (j == 1) then
             r1 = 1/pi
             r2 = 1
         else
             r1 = 1/(j*pi)
             r2 = 1/((j-1)*pi)
         endif
         call fm_integrate(f, 1, r1, r2, tol, results(j), kprt, kw)
         if (mod(j, 10) == 0) write (*, "(a, i4)") ' j = ', j
      enddo
```

```fortran
!             Form the sequence of partial sums.

      write (12, "(a)") ' '
      write (12, "(a)") ' Partial sums:'
      terms(1) = results(1)
      do j = 2, nroots
         terms(j) = results(j) + terms(j-1)
         call fm_form('f56.50', terms(j), st1)
         write (12, "(7x, a)") st1
      enddo

!             Use Aitken extrapolation on the sequence of partial sums.

      k = nroots

      write (12, "(a)") ' '
      write (12, "(a)") ' Aitken extrapolation of the partial sums:'
      kprt = 0
      r1 = abs(terms(k) - terms(k-1))
      do j = 3, nroots, 2
         call aitken(k, terms, kprt, kw)
         k = k - 2
         r2 = abs(terms(k) - terms(k-1))
         call fm_form('es12.4', r2, st1)
         write (12, "(i4, a, a)") j/2, ' extrapolations.  Estimated error =', trim(st1)
         if (r2 > r1 .or. j >= nroots-1) then
             write (12, "(a)") ' '
             write (12, "(a, i4, a)") ' The last two estimates after ', j/2-1,  &
                                      ' Aitken extrapolations ='
             write ( *, "(a)") ' '
             write ( *, "(a, i4, a)") ' The last two estimates after ', j/2-1,  &
                                      ' Aitken extrapolations ='
             call fm_form('f56.50', terms(k+1), st1)
             write (12, "(7x, a)") st1
             write ( *, "(7x, a)") st1
             call fm_form('f56.50', terms(k+2), st1)
             write (12, "(7x, a)") st1
             write ( *, "(7x, a)") st1
             exit
         endif
         r1 = r2
      enddo

!             Compare Cohen's alternating series extrapolation method.

!             This method applies to alternating series where the first term is positive and the
!             sequence of partial sums a(k) is totally monotonic.  This means that for each fixed k,
!             the sequence of the k-th forward differences of a(k) consists of all positive values
!             or all negative values.  Negate the result when the first term is negative.

      write (*, "(a)") ' '
      write (*, "(a)") " Cohen's alternating series extrapolation method:"
      write (*, "(a)") ' '
      write (12, "(a)") ' '
      write (12, "(a)") " Cohen's alternating series extrapolation method:"
      write (12, "(a)") ' '
      do n = nroots-1, nroots
```

```fortran
         d = (3 + sqrt(to_fm(8)))**n
         d = (d + 1/d)/2
         b = -1
         c = -d
         s = 0
         do k = 0, n-1
            c = b - c
            s = s + c*abs(results(k+1))
            b = (k+n)*(k-n)*b / ((k+to_fm('0.5'))*(k+1))
         enddo
         s = s/d
         write (12, "(1x, a, i2, a)") ' n = ', n, '.    Extrapolated value ='
         if (results(1) < 0) s = -s
         call fm_form('f56.50', s, st1)
         write (12, "(7x, a)") st1
         write ( *, "(1x, a, i2, a)") ' n = ', n, '.    Extrapolated value ='
         write ( *, "(7x, a)") st1
      enddo

!              For this example problem, there is a closed-form answer in terms
!              of the cosine integral and the sine.  Print it as a check.

      r1 = sin(to_fm(1)) - cos_integral(to_fm(1))
      write (12, "(a)") ' '
      write (12, "(a)") ' For this example problem, there is a closed-form answer: Sin(1) - Ci(1) ='
      call fm_form('f56.50', r1, st1)
      write (12, "(7x, a)") st1
      write ( *, "(a)") ' '
      write ( *, "(a)") ' For this example problem, there is a closed-form answer: Sin(1) - Ci(1) ='
      write ( *, "(7x, a)") st1
      write ( *, "(a)") ' '
      write ( *, "(a)") ' Intermediate results from this calculation are in file Oscillate.out'
      write ( *, "(a)") ' '

      close(12)
      stop

      end program test

      subroutine aitken(k, results, kprt, kw)
      use fmzm
      implicit none

!  Aitken extrapolation.
!  Extrapolate results(1), ..., results(k).  The Aitken values are returned in
!  results(1), ..., results(k-2)
!  kprt = 1 means write the new values in results on unit kw
!       = 0 means no output is written.

      type (fm) :: results(100)
      integer :: j, k, kprt, kw
      intent (in) :: k, kprt, kw
      intent (inout) :: results

      if (kprt == 1) then
          write (kw, "(a)") ' '
          write (kw, "(a)") ' Aitken extrapolation.'
      endif
```

```
do j = 1, k-2
   if (results(j+2) - 2*results(j+1) + results(j) == 0) then
       results(j) = results(j+2)
   else
       results(j) = results(j+2) - (results(j+2)-results(j+1))**2 /  &
                    (results(j+2) - 2*results(j+1) + results(j))
   endif
   if (kprt == 1) then
       call fm_print(results(j))
   endif
enddo

end subroutine aitken

function f(x, n)     result (return_value)
use fmzm
implicit none

type (fm) :: return_value, x
integer :: n
intent (in) :: x, n

return_value = x
if (n == 1) then
    return_value = sin(1/x)
endif

end function f
```