module globals
  double precision::a0,a1,a2,a3,a4
end module globals

program rootfinder
  use globals
  implicit none
  integer::exitflag,maxit,numit
  double precision::epsi1,epsi2,poly,polyp,r,r1,r2
  character (len=1)::ans
  character (len=3)::fmt="(a)"
  external poly,polyp
  interface
    subroutine newton(xnew,xold,f,fp,epsi1,epsi2,maxit,numit,exitflag)
      implicit none
      integer, intent(in)::maxit
      integer, intent(out)::exitflag,numit
      double precision, intent(out)::xnew
      double precision, intent(inout)::xold
      double precision, intent(in)::epsi1,epsi2
      interface
        function f(x)
          double precision::f
          double precision, intent(in)::x
        end function f
        function fp(x)
          double precision::fp
          double precision, intent(in)::x
        end function fp
      end interface
    end subroutine newton
    subroutine secant(xnew,xold,xolder,f,epsi1,epsi2,maxit,numit,exitflag)
      integer, intent(in)::maxit
      integer, intent(out)::exitflag,numit
      double precision, intent(out)::xnew
      double precision, intent(inout)::xold,xolder
      double precision, intent(in)::epsi1,epsi2
      interface
        function f(x)
          double precision::f
          double precision, intent(in)::x
        end function f
      end interface
    end subroutine secant
  end interface
  write(*,fmt,advance="no")"Enter 4th order polynomial coefficients (a0-a4): "
  read(*,*)a0,a1,a2,a3,a4
  write(*,fmt,advance="no")"Epsilon for establishing root found? "
  read(*,*)epsi1
  write(*,fmt,advance="no")"Epsilon for establishing slow progress? "
  read(*,*)epsi2
  write(*,fmt,advance="no")"Maximum number of iterations? "
  read(*,*)maxit
  write(*,fmt,advance="no")"Secant (s) or Newton's (n) method? "

read(*,fmt)ans
if(ans=="s")then
  write(*,fmt,advance="no")"Enter 2 initial root estimates: "
  read(*,*)r1,r2
  call secant(r,r1,r2,\poly,\eps1,\eps2,\maxit,\numit,\exitflag)
else
  write(*,fmt,advance="no")"Enter an initial root estimate: "
  read(*,*)r1
  call newton(r,r1,\poly,\polyp,\eps1,\eps2,\maxit,\numit,\exitflag)
end if
select case (\exitflag)
case (1)
  write(*,*)"Root found within tolerance ",\eps1
  write(*,*)" Root: ",r
  write(*,*)" Number of iterations: ",\numit
case (2)
  write(*,*)"Slow progress tolerance (\eps2) reached"
  write(*,*)" Root estimate at exit: ",r
  write(*,*)" Number of iterations: ",\numit
case (3)
  write(*,*)"Maximum number of iterations reached (\maxit)"
  write(*,*)" Root estimate at exit: ",r
  write(*,]*)" Root estimate at exit: ",r
  write(*,]*)" Root estimate at exit: ",r
end select
case (4)
  write(*,*)"Divergence suspected, iterations halted"
end select
stop
end program rootfinder

subroutine newton(xnew,xold,f,fp,\eps1,\eps2,\maxit,\numit,\exitflag)
  implicit none
  integer, intent(in)::\maxit
  integer, intent(out)::\exitflag,\numit
  double precision, intent(out)::xnew
  double precision, intent(inout)::xold
  !The following statement must NOT be included since it would re-define the
  ! functions that are defined in the interface below
  !double precision, external::f,fp
  double precision, intent(in)::\eps1,\eps2

  !Local variables
  double precision::xolder ! place holder for old root estimate
  double precision::fxold,fp\xold,fxnew ! placeholders for function values

  !Interface to the subprogram used to evaluate the function in question and
  ! it's derivative
  interface
    function f(x)
      double precision::f
      double precision, intent(in)::x
    end function f
    function fp(x)
      double precision::fp
      double precision, intent(in)::x
    end function fp
This routine implements the Newton's method for root finding

Arguments and their types

- **xnew**: final root estimate (double precision, intent out)
- **xold**: initial root estimate (double precision, intent in)
- **f**: name of the user supplied function subprogram containing the function whose root we are trying to determine (double precision, external)
- **fp**: name of the user supplied function subprogram containing the derivative (double precision, external)
- **epsi1**: convergence tolerance - root is assumed to have been found if the function value at xnew is less than epsi1 (double precision, intent in)
- **epsi2**: slow progress tolerance - progress is assumed to be too slow to continue if xnew-xold is less than epsi2 (double precision, intent in)
- **maxit**: maximum number of iterations allowed (integer, intent in)
- **numit**: number of iterations already performed (intent, out)
- **exitflag**: indicates exit condition from this subroutine
  - exitflag=1 indicates solution found with tolerance epsi1
  - exitflag=2 indicates slow progress condition
  - exitflag=3 maximum number of iterations reached
  - exitflag=4 divergence test failed (difference between successive root estimates increasing)

The code goes here

Subroutine secant(xnew, xold, xolder, f, epsi1, epsi2, maxit, numit, exitflag)

Implicit none

integer, intent(in)::maxit
integer, intent(out)::exitflag,numit
double precision, intent(out)::xnew
double precision, intent(inout)::xold,xolder

!The following statement must NOT be included since it would re-define the function that is defined in the interface below
!double precision, external::f
double precision, intent(in)::epsi1, epsi2

!Local variable
double precision::fxold,fxolder,fxnew ! placeholders for function values

!Interface to the subprogram used to evaluate the function in question
interface
  function f(x)
    double precision::f
    double precision, intent(in)::x
  end function f
end interface

This routine implements the secant method for root finding

Arguments and their types

- **xnew**: final root estimate (double precision, intent out)
- **xold**: initial root estimate (double precision, intent in)
166  !  xold: another initial root estimate (double precision, intent in)
167  !  f: name of the user supplied function subprogram containing the function
168  !  whose root we are trying to determine (double precision, external)
169  !  epsi1: convergence tolerance - root is assumed to have been found if the
170  !  function value at xnew is less then epsi1 (double precision, intent in)
171  !  epsi2: slow progress tolerance - progress is assumed to be too slow to
172  !  continue if xnew-xold is less than epsi2 (double precision, intent in)
173  !  maxit: maximum number of iterations allowed (integer, intent in)
174  !  numit: number of iterations already performed (intent, out)
175  !  exitflag: indicates exit condition from this subroutine
176  !   exitflag=1 indicates solution found with tolerance epsi1
177  !   exitflag=2 indicates slow progress condition
178  !   exitflag=3 maximum number of iterations reached
179  !   exitflag=4 divergence test failed (difference between successive
180  !     root estimates increasing)
181
182  ! CODE GOES HERE
183
184  return
185end subroutine secant
186
187function poly(x)
188  use globals
189  implicit none
190  double precision::poly
191  double precision, intent(in)::x
192  poly=a4*x**4+a3*x**3+a2*x**2+a1*x+a0
193  return
194end function poly
195
196function polyp(x)
197  use globals
198  implicit none
199  double precision::polyp
200  double precision, intent(in)::x
201  polyp=4d0*a4*x**3+3d0*a3*x**2+2d0*a2*x+a1
202  return
203end function polyp