```
      program PiN
c
c%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
c     updated 20/6/97
c     Main program for Lippman-Schwinger equation desciption
c     of  coupled (piN-piDelta) system. Calculates piN partial wave phase
c     shifts from  threshold to  Delta resonance. Also produces the
c     scattering lengths and differential cross sections.
c     PiN takes input form factors from chiral quark models
c     (e.g., chiral color-dielectric model) and searches for best model
c     parameters (bare masses or glueball mass and coulping constant)
c     by fitting the nucleon and Delta poles in P11 and P33 channels.
c     PiN was developed by Dinghui Lu and Rubin Landau in 1994-1995. It
c     inherits features from the KbarN code (developed by Guangliang
c     He and Rubin Landau).
c     Work supported by US DOE, Oregon State University.
c************************************************************************
      implicit none
c        Local variables
      integer mclock, nch, l, clmax
      integer  ich, jch, ichpt, jchpt, ichpt0, jchpt0
      double precision  hundred
c        Shared variables
      integer Nifty, npts, iopt, n, m, nprint, info, ldfjac
      integer n_fit, ner, nei, Fidx, Isheet
      integer Ipvt(3), njev, nfev, mode, maxfev
      double precision P, W, er1, er2, ei1, ei2
      double precision X(3), Fvec(450), Xpole(2)
      double precision fjac(450, 3), Diag(3), Qtf(3)
      double precision Wa1(3), Wa2(3), Wa3(3), Wa4(450)
      double precision factor, ftol, xtol, gtol, epsfcn
      double precision f0, mgb, alpha, c1, c2, Mbare, mpion, Mass, rcbm
      double complex zi, Vsuper(128,128,6)
      double complex Knew(2), enew
      character*14 reout,imout
      logical first

      parameter (zi=(0.0d0, 1.0d0))
      parameter (hundred = 1.0d2)

      common /control/ Nifty(15)
c     grid.h, gaussian grid points and weights
      common /grid/ P(64), W(64), npts
c     bag.h Version 1.4  contains the CCDM parameters. first 3 independent
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
c     mass.h Version 1.1 masses of pion, nucleon and delta
      common /mass/ mpion, Mass(2)
c     sheet.h: parameters to decide which sheet the energy is on.
      common /sheet/ Isheet(2)

      external Chi2, Det1Mvg, mclock
      data first /.true./

c************************************************************************
c
c start the clock
c      tic1 = mclock()
      open(unit=8, file='output',status='unknown')
      open(unit=22,file='debug', status='unknown')

c     print out the Banner
      write(8, 900)
      write(8, *) 'IBM XL FORTRAN, xlf'

c     kpinit does the necessary initialization, including open input files
      call KpInit(er1, er2, ner, ei1, ei2, nei, n_fit,
     &     ftol, xtol, epsfcn)

      nch = Nifty(2)
      clmax = Nifty(6)

c1111111
      if (Nifty(4) .eq. 1) then
c     calculating the potential only
         enew = er1+ zi * ei1
         write(8, 902) enew
         do ich = 1, nch
            Knew(ich) = P(npts)
         enddo
         call VBag(npts,P,enew,Knew,Vsuper)
c        write the potential
         do 10030 l =1, clmax
            do jch = 1, nch
               jchpt0 = (jch-1)*(npts+1) + 1
               do ich = jch, nch
                  write(reout, 910) l,ich,jch
                  write(imout, 911) l,ich,jch
                  open(9, file = reout, status='unknown')
                  open(10, file = imout, status='unknown')
                  write(8, 912) l, reout, imout
                  ichpt0 = (ich-1)*(npts+1) + 1
                  write(9, 916) ((dble(Vsuper(ichpt,jchpt,l)),
     &                  ichpt = ichpt0, ichpt0+npts),
     &                  jchpt = jchpt0, jchpt0+npts)
                  write(10, 916) ((imag(Vsuper(ichpt,jchpt,l)),
     &                  ichpt = ichpt0, ichpt0+npts),
     &                  jchpt = jchpt0, jchpt0+npts)
                  close(9)
                  close(10)
               enddo
            enddo
10030    continue

c222222222
      else if (Nifty(4) .eq. 2) then
c     calculate scattering amplitudes,
c     phase shifts and scattering lengths once the parameters are fixed
         if (Nifty(8) .eq. 1) then
            call Amp(er1, er2, ner, ei1, ei2, nei)
         else if(Nifty(8) .eq. 2) then
            call Obs(er1, er2, ner)
         endif

c333333333
      else if (Nifty(4) .eq. 3) then
c     run chi-square fit
         write(8, *) 'Start to prepare for snls1'
         iopt = 1
         n_fit = 5
         m = n_fit + 1
         call SetFit(n, X)
         ldfjac = 450
         gtol = 0.0d0
         maxfev = 100*(n+1)
         mode = 1
         factor = 1.0d2
         nprint = 1
         write(*,*)'epsfcn=',epsfcn
         call snls1(chi2, iopt, m, n, x, Fvec, fjac, ldfjac, ftol,
     &        xtol, gtol, maxfev, epsfcn, Diag, mode, factor, nprint,
```

```
     &        info, nfev, njev, Ipvt, Qtf, Wa1, Wa2, Wa3, Wa4)
          write(8, 920)
          call PrtPot()
          write(8, 925)
          write(8, *) 'INFO', info
c         Extra run for phashi, PiNNff and p11
c         isheet(1) = -1
c         isheet(2) = 1
          Nifty(4) = 2
          Nifty(6) = 6
          call Obs(er1,er2,ner)
c
          write(8,*) 'PAREMETERS for POLE search'
          call PrtPot()
          Nifty(4) = 4
          Nifty(6) = 3
          Xpole(1) = 940.-mpion-Mass(1)
          Xpole(2) = 0.
          call GetPole(Xpole,xtol,epsfcn)

c         calc coupling constant
          er1 = Xpole(1)+mpion+Mass(1)
          ei1 = Xpole(2)
          er2 = Mbare(1)
          ei2 = 0.
          Nifty(4) = 2
          Nifty(6) = 6
          Nifty(8) = 1
          Nifty(11) = 0
          call Amp(er1, er2, ner, ei1, ei2, nei)

c444444444
      else if (Nifty(4) .eq. 4) then
c     call snsq searching zero of det(1-VG) for one particular channel
c      iopt = 2
c         n = 2
          Xpole(1) = er1-mpion-Mass(1)
          Xpole(2) = ei1
          call GetPole(Xpole,xtol,epsfcn)
      endif


c        stop the clock
c         tic2 = mclock()
c        on 1 Fortran, one second is 100 tics, not 60 tics as said in manual.
c         seconds = (tic2 - tic1) * 0.01d0
c      write(8, 990) seconds
      close(8)

  900 format(1x, 'PiN --- A Package Calculates PiN-PiDelta system')
  902 format('0', 'Energy == ', 2(2x, e10.4))
  910 format('vl+.r',3i1)
  911 format('vl+.i',3i1)
  912 format('0', 'writing part of Vsuper( , , ', i1,
     &                 ') to ', a14, 'and ', a14)
  916 format(6(1x,e12.6))
 920  format(/'0',10('='), ' Final Paras + EXTRA OBS and POLE', 10('='))
 925  format(1x, 38('='))
 930  format('0', 10('='), ' Final BS (POLE) Results ', 10('='), /,
     &     1x, 'Info = ', i4, /,
     &     1x, 'Re(E)   = ', e12.6, 3x, 'Im(E)   = ', e12.6, /,
     &     1x, 'Re(Det) = ', e12.6, 3x, 'Im(Det) = ', e12.6, /,
     &     1x, 47('='))
 935  format('0', 'Zero not fount. Info = ', i3, /,
     &     1x, 'See doc of snsq for more details')
  990 format('0', 'Total time used: ', f10.3, ' seconds.')
      end
c*********************************************************************

      subroutine Amp(er1, er2, ner, ei1, ei2, nei)
c
c*********************************************************************
c     Amp calculates the scattering amplitudes (3D Tmatrix pole)
c     for given complex energy range er1, er2, ei1, ei2
c     if nif11=0, calc PiNN cpling const and ff, possibly for PiDD later
c              1, 3D diagonal tmatrix vs E
c              2, 3D inverse det(1-vg) vs E (incorrect)
c*********************************************************************
c
      integer cl, i, istep, rstep,  c
      integer first, iflag, ii, ileft,iright
      double precision zero, hbarc,  de, de1, de2, tmp, rmd
      double precision fpi1, ff2ren1(64),ff2ren2(64)
      double precision c0, s1, s2, Iprop, ff2bar1(64), ff2bar2(64)
c
      integer Nifty, nch, ner, nei, npts
      double precision er1, er2, ei1, ei2, rsize, isize, er, ei, P, W
      double precision mpion, Mass, fact, pi
      double precision Evec(2), Detvec(2)
      double complex enew, eold, Knew(2), Knew2(2), Dk2de(2), Wgf(128)
      double complex Vsuper(128,128,6), Tonshl(2,2,6)
      double complex Thalf(64), Tdiag(64), Te(64,240), Ve(64,240)
      double complex epole, ebare
      common /half/ Thalf, Tdiag
      common /grid/ P(64), W(64), npts
      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)

      parameter (zero=0.0d0, hbarc=1.9732858d2, pi=3.1415926d0)
      data first/1/
c*****************
c
      nch = Nifty(2)
      cl = Nifty(6)
      if (cl.eq.3) then
        rmd = 1.
        else if (cl.eq.6) then
        rmd = (2.*1.414/5.0)**2
        endif
      call PrtPot

      if (Nifty(11) .eq. 0) then
        open(9, file='PiNNee', status='unknown')
        write(9, 2000) er1, ei1
2000    format('# Energy dependnece of ff**2(upto a const):Resonance
     &E =', 2e12.5/
     &  '#    E',4x,'~Tonshl(1,1,3)',2x,'~Tdiag(1)',2x,'~Vbare(1,1,3)')
c       plot diagonal t matrix, start 10 MeV below resonance and bare pole
c       use (er1,ei1) as physical pole, use (er2,ei2) as bare mass
c       epole and ebare is 0.001MeV away from exact
        epole = dcmplx(er1, ei1) + 0.001
        ebare = dcmplx(er2, ei2) + 0.001
        ner = 40

c    renormalized cpling const
        write(*,*) 'Calculate Renomalized Cpling and FF'
        er = er1 - 10.0
        eold = dcmplx(er, ei1)
        do    rstep = 1, ner
          er = er + 0.5
```

```
          enew = dcmplx(er, ei1)
          call OnShl(enew, eold, Knew, Knew2, Dk2de)
          call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)
          call VBag(npts, P, enew, Knew, Vsuper)
          call TMtrx(npts, Dk2de, Wgf, Vsuper, Tonshl)
c          write(*,1005) er,Tdiag(1),Vsuper(1,1,3)
          if (first.eq.1 .and. Nifty(10).eq.1) then
c           calc extra factor c0 in cpling const due to mass renormalization
            s1 = Iprop(er1-1.0, P, W, npts)
            s2 = Iprop(er1+1.0, P, W, npts)
            c0 = 1. + (s2 - s1)/2.0
            write(*,*) '*** c0,s1,s2 ***', c0, s1, s2
            first = 0
          else
            c0 = 1.0
          endif
c
          fact = 3.*rmd*(P(1)/mpion)**2/(2.*sqrt(mpion**2+P(1)**2))
          fpi1 = Tdiag(1)*(enew - epole)/fact
          write(9,2002) er, fpi1/c0, fpi1
          do i = 1, npts
            Te(i,rstep) = Tdiag(i)
          enddo
        enddo

c       subtract the ff(q)**2 at 7 MeV away from pole (both side)
        ii = 4
        ileft = 2
        de = er1 - 10.0 + ileft*0.5 - er1
        de1 = 1./de - 1./(de+0.5*ii)
        iright = 34
        de = er1 - 10.0 + iright*0.5 - er1
        de2 = 1./de - 1./(de+0.5*ii)

        do i = 1, npts
          fact = 3.*rmd*(P(i)/mpion)**2/(2.*sqrt(mpion**2+P(i)**2))
          ff2ren1(i) = (Te(i,ileft)-Te(i,ileft+ii))/de1/fact
          ff2ren2(i) = (Te(i,iright)-Te(i,iright+ii))/de2/fact
        enddo

c   bare cpling const
        write(*,*) 'Calculate Bare Cpling and FF'
        write(9,2003)
        er = er2 - 10.0
        eold = dcmplx(er, 0.0d0)
        do    rstep = 1, ner
          er = er + 0.500001
          enew = dcmplx(er, 0.0d0)
          call OnShl(enew, eold, Knew, Knew2, Dk2de)
          call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)
          call VBag(npts, P, enew, Knew, Vsuper)
          fact = 3.*rmd*(P(1)/mpion)**2/(2.*sqrt(mpion**2+P(1)**2))
          fpi1 = Vsuper(1,1,cl)*(enew - ebare)/fact
          write(*,1005) er, Vsuper(1,1,cl), fpi1
          write(9,2002) er, fpi1
          do i = 1, npts
            Ve(i,rstep) = Vsuper(i,i,cl)
          enddo
        enddo
        open(10, file='PiNNff', status='unknown')
        de = er2 - 10.0 + ileft*0.5 - er2
        de1 = 1./de - 1./(de+0.5*ii)
        de = er2 - 10.0 + iright*0.5 - er2
        de2 = 1./de - 1./(de+0.5*ii)
```

```
        do i = 1, npts
          fact = 3.*rmd*(P(i)/mpion)**2/(2.*sqrt(mpion**2+P(i)**2))
          ff2bar1(i) = (Ve(i,ileft)-Ve(i,ileft+ii))/de1/fact
          ff2bar2(i) = (Ve(i,iright)-Ve(i,iright+ii))/de2/fact
          write(10,2002) P(i),ff2ren1(i),ff2ren2(i),
     &ff2bar1(i),ff2bar2(i),ff2ren1(i)/ff2ren1(1),ff2bar1(i)/ff2bar1(1)
        enddo
2002      format(1x,e12.5,9e11.4)
2003      format(/)
1005      format('Amp:er ',e12.4,6e11.3)
c*******t********2*********3*********4*********5*********6*********7
      else if (Nifty(11) .ge. 1) then
        open(9, file='Amp.re', status='unknown')
        open(10, file='Amp.im', status='unknown')
        c = Nifty(6)

c        calculating step size
        if (ner .ne. 1) then
          rsize = (er2-er1)/(ner-1)
          else
            rsize = zero
            endif
        if (nei .ne. 1) then
          isize = (ei2-ei1)/(nei-1)
          else
            isize = zero
            endif

        eold = dcmplx(er1, ei1)
        do istep = 1, nei
          ei = ei1 + isize*(istep-1)
          do    rstep = 1, ner
            er = er1 + rsize*(rstep-1)
          if (Nifty(11) .eq. 1) then
            enew = dcmplx(er, ei)
            call OnShl(enew, eold, Knew, Knew2, Dk2de)
            call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)
            call VBag(npts, P, enew, Knew, Vsuper)
            call TMtrx(npts, Dk2de, Wgf, Vsuper, Tonshl)

c           print out scattering amplitudes as TRIUMF's convention
c           show pole using 1:2:4 3D plot
            call PrtKin(enew, Knew)
            write(9,1001) er, ei, -hbarc*dble(Dk2de(1)*Tonshl(1,1,c)),
     &      -hbarc*dble(Dk2de(1)*Tdiag(1)),dble(Tdiag(1)),dble(Tdiag(2))
            write(10,1001) er, ei, -hbarc*imag(Dk2de(1)*Tonshl(1,1,c)),
     &      -hbarc*imag(Dk2de(1)*Tdiag(1)),imag(Tdiag(1)),imag(Tdiag(2))
          else if (Nifty(11) .eq. 2) then
c           Detvec(2) return det(1-vg)
            iflag = 1
            Evec(1) = er
            Evec(2) = ei
            call Det1Mvg(2, Evec, Detvec, iflag)
            tmp = Detvec(1)**2 + Detvec(2)**2
            write(9,1001) er, ei, Detvec(1)/tmp, 1.0/Detvec(1)
            write(10,1001) er, ei, -Detvec(2)/tmp, 1.0/Detvec(2)
          endif
c
            eold = enew
          enddo
           write(9,1002)
           write(10,1002)
        enddo
      endif
1001    format(8e12.4)
```

```
1002    format(/)

      return
      end

c*********************************************************************
c     propogator integral for subtract the extra factor for cpling
        double precision function Iprop(ener, P, W, npts)
        integer npts, ipt, Fidx
        double precision Uborn(64), Eg(64), ener
        double precision mpion, Mass, fact, pi, P(64), W(64)
        double precision f0, mgb, alpha, c1, c2, Mbare, rcbm, sum

        common /mass/ mpion, Mass(2)
        common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
        common /cpl/ Uborn
        parameter (pi=3.1415926d0)

c     relativistic kinematics
        do ipt = 1, npts
          Eg(ipt) = sqrt(P(ipt)**2 + mpion**2)
     &                  +sqrt(P(ipt)**2 + Mass(1)**2)
        enddo
c
        sum = 0.0
        do ipt = 1, npts
          fact = P(ipt)**4/sqrt(P(ipt)**2 + mpion**2)
          sum=sum + fact*Uborn(ipt)**2*W(ipt)/(ener-Eg(ipt))
        enddo
        Iprop = sum * 25.0/(12.*pi*f0**2)/4./pi
c       the extra 1/4pi factor is necessary to cancel the denominator of reson
        write(*,*) ener,Iprop,sum
        return
        end
c*********************************************************************

      subroutine Getpole(X, xtol, epsfcn)
      implicit none
      integer iopt, nprint, info, ldfjac, n
      integer njev, nfev, mode, maxfev, ml, mu, lr
      double precision fjac(450, 3), Diag(3), Qtf(3)
      double precision Wa1(3), Wa2(3), Wa3(3), Wa4(450)
      double precision X(3), Fvec(450), factor, xtol, epsfcn, r(3)
      double precision mpion, Mass

      common /mass/ mpion, Mass(2)

      external Det1Mvg,jac

c*********************************************************************
c
c     call snsq searching zero of det(1-VG) for one particular channel
         n = 2
         iopt = 2
         ldfjac = 2
         maxfev = 100*(n+1)*2
         ml = n - 1
         mu = n - 1
         mode = 1
         factor = 100.
         nprint = 1
         lr = (n*(n+1))/2
         call snsq(Det1Mvg, jac, iopt, n, X, Fvec, Fjac,
     &        ldfjac, xtol, maxfev, ml, mu, epsfcn, Diag, mode,
     &        factor, nprint, info, nfev, njev, r, lr, Qtf,
     &        Wa1, Wa2, Wa3, Wa4)
         if (info .eq. 1) then
            write(8, 930) info, X(1)+mpion+Mass(1), X(2),
     &                       Fvec(1), Fvec(2)
         else
            write(8, 935) info
         endif
c      endif

 930  format('0', 10('='), ' Final BS (POLE) Results ', 10('='), /,
     &     1x, 'Info = ', i4, /,
     &     1x, 'Re(E)   = ', e12.6, 3x, 'Im(E)   = ', e12.6, /,
     &     1x, 'Re(Det) = ', e12.6, 3x, 'Im(Det) = ', e12.6, /,
     &     1x, 47('='))
 935  format('0', 'Zero not fount. Info = ', i3, /,
     &     1x, 'See doc of snsq for more details')
        return
        end
c
c*********************************************************************
      subroutine KpInit(er1, er2, ner, ei1, ei2, nei, n_fit,
     &    ftol, xtol, epsfcn)
c
c*********************************************************************
c     Kpinit does the necessary initialization for the kpfit program
c*********************************************************************
c     LOCAL variables
      integer i, nch
c     SHARED variables
      integer ner, nei, n_fit, Nifty, Isheet, Fidx
      integer npts, nnuc, ncoul
      double precision er1, er2, ei1, ei2, P, W, pcoul, pnuc, zero
      double precision ftol, xtol, epsfcn, rcbm
      double precision f0, mgb, alpha, Mbare, mpion, Mass, c1, c2
c
      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
      common /grid/ P(64), W(64), npts
      common /sheet/ Isheet(2)
c
      parameter (zero = 0.0d0)
c
c*********************************************************************
c
      open(unit = 7, file = 'input', status = 'old')

c     read Nifty parameters and print out their meaning
      read (7, *) (Nifty(i), i = 1, 15)
      nch = Nifty(2)
      call PrtNif

c     initialize the mass vectors, and coupling const
      call SetUp

c     initial potential paramenters, only first 3 adjustable
      read(7, *) f0, Mbare(1), Mbare(2), rcbm, mpion
      read(7, *) (Fidx(i), i = 1, 3)

c     read in more inputs
      read(7, *) er1, er2, ner
      read(7, *) ei1, ei2, nei
c      if (Nifty(4) .eq. 2) er1 = mpion + Mass(1)
      read(7, *) pcoul, pnuc, ncoul, nnuc
      npts = ncoul + nnuc
```

```
c     check if number of grid points exceeds the capacity of the code
      if (npts+1 .gt. 64) then
         write(8, 900) npts+1
         stop 'kpinit, npts too big'
      endif

c     read in initial Isheet array
      read(7, *) (Isheet(i), i = 1, nch)
      write(8, 910) (Isheet(i), i = 1, nch)

c     reading parameters for snls1
      read(7, *) ftol, xtol, epsfcn

c     calculate grid points for either potential or T-matrix calculation
      call Gauss(ncoul, 0, zero, pcoul, P, W)
      call Gauss(nnuc, 2, pcoul, pnuc, P(ncoul+1), W(ncoul+1))

      write(8, 920) pcoul, pnuc, ncoul, nnuc

c     print out grid points and weights only if we are not fitting
      if (Nifty(4) .ne. 3) then
         write(8, *) 'K-SPACE GRID POINTS:'
         write(8, 925) (P(i), i = 1, npts)
         write(8, *) 'Weights:'
         write(8, 925) (W(i), i = 1, npts)
      endif
c
      close(7)
c
      n_fit = 0
c
 900  format('0', 'Total, i5, 1x, grid points for one channel',
     &      1x, 'exceeds the capacity of the program.')
 910  format('0', 'Initial Isheet Values:', 5i4)
 920  format('0', 'PCOUL = ', e12.5, 3x, 'PNUC = ', e12.5, 5x,
     &            'NCOUL = ', i4, 3x, 'NNUC = ', i4)
 925  format(5(1x, e15.8))
 926  format(5x, 2e15.8)
c
      return
      end

c*********************************************************************
      subroutine Obs(er1, er2, ner)

c*********************************************************************
c     Obs calculates the observables, including phase shifts, cross section
c     and scattering length etc.
c*********************************************************************
c     LOCAL variables
      integer i, nch, clmax, cl
      double precision hbarc, pi, zero, two, three
      double precision x1, x2, stmp, etmp, rho, psdrop, psold(6)
      double precision SL(6), Phase(6), Eta(6)
      double precision dm, p11, p13, p31, p33
c     SHARED variables
      integer Nifty, npts, ner, nfit, first
      double precision er1, er2, P, W, mpion, Mass, tlab
      double precision Efit(20), exp_xs, th_xs
      double complex enew, eold, Knew(2), Knew2(2), Dk2de(2), Wgf(128)
      double complex Vsuper(128,128,6), Tonshl(2,2,6)

      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)
      common /grid/ P(64), W(64), npts
      common /data/ exp_xs(20, 6), th_xs(20, 6), nfit

      parameter (hbarc=1.9732858d2, pi=3.1415926d0)
      parameter (zero=0.0d0, two=2.0d0, three=3.0d0)
      data first/1/
c****************************************************

      nch = Nifty(2)
      clmax = Nifty(6)
      eold = dcmplx(er1, zero)

      if (Nifty(4) .eq. 2) then
c        calc phase shift from energy er1 to er2
c        pi-N threshold 1078.9 MeV
         if (er1 .lt. 1078.0) then
            write(*,*)'starting energy (er1) too small, try threshold'
         endif

         do cl = 1, 6
           psold(cl) = 0.
         enddo

         open(11, file='o.on-T', status='unknown')
         open(2,  file='o.ps-s', status='unknown')
         open(12, file='o.ps-p', status='unknown')
         open(3,  file='o.sc-len', status='unknown')

         write(11,100)
100      format('# Partial wave amplitude (onshl Tmatrix)'/,
     &   '# 1st col is energy, followed by Re/Im tmatrix for eigenchannels
     &  S11,S31,P11,P31,P13,P33')

         etmp = er1 - (er2-er1)/(ner-1)
         do 10010 i = 1, ner
            etmp = etmp + (er2-er1)/(ner-1)
            enew = dcmplx(etmp, zero)

            call OnShl(enew, eold, Knew, Knew2, Dk2de)

            call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)

            call VBag(npts, P, enew, Knew, Vsuper)

            call TMtrx(npts, Dk2de, Wgf, Vsuper, Tonshl)

c           Devide by relativistic density of state (real), PiN->PiN
            do cl = 1, clmax
              rho = Knew(1)*Dk2de(1)
              Tonshl(1,1,cl) = -Tonshl(1,1,cl) * rho
            enddo

c           print out for ploting, S11, S31, P11, P31, P13, P33 Re/Im
c           with phase shift anlysis: Arndt,Ford and Roper, PRD32(85)1085

            write(11,101) etmp, (Tonshl(1,1,cl),cl=1,clmax)
101         format(f6.1,12e11.3)

            do cl = 1, clmax
c             Compute phase shift and inelastic coef.
              x1 = 2.d0*real(Tonshl(1,1,cl))
              x2 = 1.d0-2.d0*imag(Tonshl(1,1,cl))
              Phase(cl) = 0.5d0 * atan2(x1,x2) * 180.d0/pi
              Eta(cl) = sqrt(x1*x1 + x2*x2)
              psdrop = Phase(cl) - psold(cl)
```

```
            if (psdrop .lt. -20.) then
               Phase(cl)=Phase(cl)+180.d0
            endif
            psold(cl) = Phase(cl)

c           Compute scattering length
            stmp = tan(Phase(cl)*pi/180.0)
            if (cl .gt. 2) then
              if (stmp.lt.0.0d0) then
                stmp = -(-stmp)**(0.333333d0)
                else
                  stmp = stmp**(0.333333d0)
                  endif
            endif
            SL(cl) = stmp*hbarc/real(Knew(1))
          enddo
          write(2,201) etmp, (Phase(cl),Eta(cl),cl=1,2)
          write(12,201)etmp, (Phase(cl),Eta(cl),cl=3,6),
     &                dsin(Phase(6)*pi/180.0)**2
          tlab = 0.5*(etmp+mpion+Mass(1))*(etmp-mpion-Mass(1))/Mass(1)
          write(3,301)tlab, real(Knew(1)), (SL(cl),cl=1,clmax)
201       format(f6.1,6(e12.3,f6.2),e10.3)
301       format(f6.1,e10.3,6e12.4)

c          if (Nifty(9) .eq. 1) then
c          calc total cross sections sigma(nch)?
c             call Xsctn(enew, Knew, Dk2de, Tonshl, tlm, sigma)
c
c          else if (Nifty(9) .eq. 2) then
c          calc diff cross sections and polarizations
c             write(8, *) 'X and pola'
c          endif

10010    continue
c        end loop for energy axis
        close(11)
        close(2)
        close(3)

        else

c         calc phase shifts for selected energy points for fitting
c         p11,p13,p31,p33 are phase shifts in units of degree
c         dm is a useless var only for convenience (data format)
c         parameters er1, er2, and ner are irrelevant in this case

          if (first .eq. 1) then
            nfit = 5

            open(4,file='datafit',status='unknown')

            do i = 1, nfit
             read(4,*) dm, p11,dm, p13,dm, p31,dm, p33,dm, Efit(i)
             exp_xs(i,3) = dsin(p11*pi/180.)**2
             exp_xs(i,6) = dsin(p33*pi/180.)**2
             write(8,900) Efit(i), p11,exp_xs(i,3), p33,exp_xs(i,6)
            enddo
900         format('Fit XS:', 5e13.5)
            close(4)
            first = 0
          endif

          do 10020 i = 1, nfit
            etmp = Efit(i)
            enew = dcmplx(etmp, zero)

            call OnShl(enew, eold, Knew, Knew2, Dk2de)
            call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)
            call VBag(npts, P, enew, Knew, Vsuper)
            call TMtrx(npts, Dk2de, Wgf, Vsuper, Tonshl)

c           Devide by relativistic density of state (real), PiN->PiN
            do cl = 3, clmax, 3
              rho = Knew(1)*Dk2de(1)
              Tonshl(1,1,cl) = -Tonshl(1,1,cl) * rho

c             Compute phase shift and inelastic coef.
              x1 = 2.d0*real(Tonshl(1,1,cl))
              x2 = 1.d0-2.d0*imag(Tonshl(1,1,cl))
              th_xs(i,cl) = dsin(0.5d0 * atan2(x1,x2))**2
            enddo
10020     continue
        endif

c       print out the potential parameters
c        call PrtPot()

        return
        end

c*******************************************************************

      subroutine PrtPot

c*******************************************************************
c     prtpot prints the current potential parameters
c*******************************************************************

      integer Fidx
      double precision f0, mgb, alpha, c1, c2, Mbare, rcbm

c     bag.h Version 1.4  contains the CCDM parameters. first 3 independent
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)

      write(8, 900) f0, alpha, mgb, Mbare(1), Mbare(2), Mbare(3), rcbm

  900 format('PARAMETERS:',/
     &     5x, 'Pion decay const:   ', f14.4, ' Mev', /,
     &     5x, 'Strg cpling const:  ', f14.4, '    ', /,
     &     5x, 'Glueball mass:      ', f14.4, ' MeV', /,
     &     5x, 'Nucleon bare mass:  ', f14.4, ' MeV', /,
     &     5x, 'Delta bare mass:    ', f14.4, ' Mev', /,
     &     5x, 'Roper bare mass:    ', f14.4, ' MeV', /,
     &     5x, 'CBM bag radius:     ', f14.4, '  fm')

      return
      end
c*******************************************************************

      subroutine SetFit(n, X)

c*******************************************************************
c     SetFit set up the fitting parameter array X() and number of
c     fitting parameters n, according to the fitting index array
c     Fidx() and potential parameters f0,  mgb, alpha.
c     Note that if fidx(?)=0 then the parameter x(?) do not changed.
c*******************************************************************

      integer Fidx, n, i
      double precision f0, mgb, alpha, c1, c2, Mbare, rcbm
```

```
      double precision X(3)

c     bag.h Version 1.4  contains the CCDM parameters. first 3 independent
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)

      do 10000 i = 1, 3
         if ((Fidx(i) .ne. 0) .and. (Fidx(i) .ne. 1)) then
            write(8, *) 'bad fidx value'
            stop 'setfit'
         endif
10000 continue

      n = 0

      if (Fidx(1) .eq. 1) then
         n = n + 1
         X(n) = f0
      endif

      if (Fidx(2) .eq. 1) then
         n = n + 1
         X(n) = Mbare(1)
      endif

      if (Fidx(3) .eq. 1) then
         n = n + 1
         X(n) = Mbare(2)
      endif
c
      return
      end
c
c***********************************************************************
      subroutine VBag(npts, P, ze, Zk, Vsuper)
c
c***********************************************************************
c     VBag calculates super potential matrix.
c        Vsuper_ba(k', k, l), where l stands for eigen-channel (up to 6)
c     Cplv, Cplct and Cplcs are SU(N) coupling constants in isospin base.
c***********************************************************************
c     LOCAL variables
      integer first, nch, npts, npts1, clmin, clmax, cl
      integer ich, jch, ipt, jpt, ichpt, jchpt
      double precision zero, two
c     SHARED variables
      integer Nifty, Fidx
      double precision P(64), Wm(64), Wmst(64), rcbm
      double precision f0, mgb, alpha, c1, c2, Mbare, mpion, Mass
      double precision Uborn(64), Uct(64,64,0:1), Ucs(64,64,0:1)
      double precision Urop(64)
      double complex Wm0(2), Wm0st(2), Vsuper(128,128,6)
      double complex ze, Zk(2), zUborn(2),zUrop(2)
      double complex zUct1(2,64,0:1),zUcs1(2,64,0:1)
      double complex zUct2(64,2,0:1),zUcs2(64,2,0:1)
      double complex zUct0(2,2,0:1),zUcs0(2,2,0:1)

      parameter (zero=0.0d0, two=2.0d0)
      common /control/ Nifty(15)
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
      common /mass/ mpion, Mass(2)
      common /cpl/Uborn
      data first /1/
c*******************

      npts1 = npts+1

c     do following only at first time or after model parameters change
      if (first .eq. 1) then
         nch = Nifty(2)
         clmax = Nifty(6)
         clmin = 1
         if (Nifty(4) .eq. 4) clmin = clmax

c        calc meson energy for each grid point using relat kinematics
         do ipt = 1, npts
              Wm(ipt)=sqrt(mpion**2+P(ipt)*P(ipt))
              Wmst(ipt)=sqrt(two*Wm(ipt))
         enddo
         first = 0
        endif
c     end if (first)

      call FfInt(npts,P,Zk,Uborn,Urop,Uct,Ucs,
     & zUborn,zUrop,zUct1,zUct2,zUct0,zUcs1,zUcs2,zUcs0)

c     Compute meson onshell point energy and its square root
c     Zk(nch) is on-shell channel momentum, complex, used in LS equ.
      do ich = 1, nch
            Wm0(ich) = sqrt(Zk(ich)**2+mpion**2)
            Wm0st(ich) = sqrt(two*Wm0(ich))
      enddo

c*********************************************************************
c       The pot matrix Vsuper is of nch*nch block. Each block (i,j)
c     is a npts1*npts1 complex matrix corresp'g to the trans'n pot
c     from j- to i-chan'l. The 1st npts points are gaussian grids
c     and the last one is on the shell needed for calc of T-matrix.
c     For calc pot only, the on-shell momentum is set to P(npts)
c       By sym, only lower half (inc diagnal) of Vsuper are computed.
c     Upper half are set to lower half after all diag h'n included.
c
c     initialize the whole potential matrix
      do cl = 1, clmax
         do jchpt = 1, nch*npts1
            do ichpt = 1, nch*npts1
               Vsuper(ichpt, jchpt, cl) = zero
            enddo
         enddo
      enddo

c     The order of 6 partial waves: l=S11,S31,P11,P31,P13,P33
c     start to call Diag [s, u, t]

      call DiagSU(npts,P,ze,Zk,Wm,Wmst,Wm0,Wm0st
     &             ,Mbare,f0,Uborn,zUborn,Urop,zUrop,Vsuper)
       if (Nifty(10) .ge. 3) then
           call DiagT(npts,P,ze,Zk,Wm,Wmst,Wm0,Wm0st,f0,Uct,Ucs
     &             ,zUct1,zUct2,zUct0,zUcs1,zUcs2,zUcs0,Vsuper)
       endif


c     now, we get another half of potential matrix
      do cl = clmin, clmax
c         do j = 1, nch*npts1
c            do i = 1, j-1
c       donot use symmetry of potntial
c                Vsuper(i, j, cl) = Vsuper(j, i, cl)
c           enddo
c         enddo
```

```
        do 103 jch = 1, nch
        do 104 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Vsuper(jchpt,ichpt,cl) = Vsuper(ichpt,jchpt,cl)
               enddo
               ichpt = ich*npts1
               Vsuper(jchpt,ichpt,cl) = Vsuper(ichpt,jchpt,cl)
            enddo

            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Vsuper(jchpt,ichpt,cl) = Vsuper(ichpt,jchpt,cl)
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Vsuper(jchpt,ichpt,cl) = Vsuper(ichpt,jchpt,cl)
  104    continue
  103    continue
        enddo
      return
      end

c
c*********************************************************************

      subroutine Det1Mvg(n, Evec, Detvec, iflag)
c
c*********************************************************************
c     Det1mvg calculates the det(1-VG)
c     the argument list is designed around the routine sqsqe()
c
c     n:         should be always equal to 2
c     Evec(1)    real part of energy
c     Evec(2)    imaginary part of energy
c     Detvec(1)  real part of det(1-vg)
c     Detvec(2)  imaginary part of det(1-vg)
c     iflag      not changed in this routine.
c*********************************************************************
c
      implicit none
      integer Nifty, n, iflag, info,num
      integer nch, cl, job
      integer npts, Indx(128), ichpt, jchpt, nchpt
      double precision P, W, zero, one, ten, pt1
      double precision Evec(2), Detvec(2), zabs1, vc(64, 64)
      double precision mpion, Mass, mtotal
      double complex work(128)
      double complex Knew(2), Knew2(2), enew, eold, Dk2de(2)
      double complex Wgf(128), Fmtrx(128, 128), zdet(2), zdumy
      double complex Vsuper(128, 128, 6)
      logical first

      common /control/ Nifty(15)
      common /grid/ P(64), W(64), npts
      common /mass/ mpion, Mass(2)
c
      parameter (zero = 0.0d0, one = 1.0d0, ten = 1.0d1, pt1 = 0.1d0)
      data first /.true./, num/0/
      zabs1(zdumy) = abs(dble(zdumy)) + abs(imag(zdumy))
c
c*********************************************************************
c
      write(8,*) 'iflag=', iflag
      if (iflag .eq. 0) then
         write(8, 900)
c         call PrtKin(enew, Knew)
         write(8, 905) Evec(1)+mtotal, Evec(2)
 900  format('0', 10('-'), ' DET(1-VG) Print Mode ', 10('-'))
 905  format(1x, 'Pole position:', e10.3, 1x, e10.3)
 910  format(1x, 'Det(1-VG) =', e10.3, 1x, e10.3)
         return
      else
c
c       cl denotes the particular channel for pole searching
c
         cl = Nifty(6)
         mtotal = mpion + Mass(1)
c
         if (first) then
            eold = dcmplx(Evec(1)+mtotal, Evec(2))
c           D. Lu turn the coulomb off
c            if (Nifty(13) .ne. 0)
c     &           call VCoul(1, -1, cl, npts, p, w, 64, vc)
         endif
c
         nch = Nifty(2)
         nchpt = nch*(npts+1)
c
         enew = dcmplx(Evec(1)+mtotal, Evec(2))
         call OnShl(enew, eold, Knew, Knew2, Dk2de)
         call WtgFcn(npts, P, W, enew, Knew, Knew2, Dk2de, Wgf)
         call VBag(npts, P, enew, Knew, Vsuper)
         if (Nifty(13) .ne. 0) then
               do ichpt = 1, npts
                  do jchpt = 1, npts
                     Vsuper(jchpt, ichpt, cl) =
     &                    Vsuper(jchpt, ichpt, cl) + vc(jchpt, ichpt)
                  enddo
               enddo
         endif
c     set up matrix 1/G-V
        do jchpt = 1, nchpt
               if (zabs1(Wgf(jchpt)) .ne. zero) then
                  do ichpt = 1, nchpt
                     Fmtrx(ichpt, jchpt) = -Vsuper(ichpt, jchpt, cl)
                  enddo
                  Fmtrx(jchpt, jchpt) = 1/Wgf(jchpt)
     &                 + Fmtrx(jchpt, jchpt)
               else
                  do ichpt = 1, nchpt
                     Fmtrx(ichpt, jchpt) = zero
                  enddo
                  Fmtrx(jchpt, jchpt) = one
               endif
         enddo
c
c     computing det(1/G-V)
         call zsifa(Fmtrx, 128, nchpt, Indx, info)
         job = 10
         call zsidi(Fmtrx, 128, nchpt, Indx, zdet, work, job)
c
c     computing det(G)*det(1/G-V)
         do ichpt = 1, nchpt
            if (zabs1(Wgf(ichpt)) .ne. zero) then
               zdet(1) = zdet(1)*Wgf(ichpt)
               if ((zabs1(zdet(1)) .lt. one) .and.
```

```
     &              (zabs1(zdet(1)) .ne. zero)) then
c     incress zdet(1) and decress zdet(2)
10000             zdet(1) = zdet(1)*ten
                  zdet(2) = zdet(2) - one
                  if (zabs1(zdet(1)) .lt. one) goto 10000
               else if (zabs1(zdet(1)) .gt. ten) then
10010             zdet(1) = zdet(1)*pt1
                  zdet(2) = zdet(2) + one
                  if (zabs1(zdet(1)) .gt. ten) goto 10010
               endif
            endif
         enddo
         Detvec(1) = dble(zdet(1)*ten**zdet(2))
         Detvec(2) = imag(zdet(1)*ten**zdet(2))
c     endif (iflag .eq. 0)
      endif
c
        num=num +1
        write(*,999) num, Evec(1)+mtotal, Evec(2), detvec
999     format(i5,2x,'E:',2e14.5,2x,'DET:',2e14.5)
c
      return
      end

c***********************************************************************


      subroutine OnShl(enew, eold, Knew, Knew2, Dk2de)

c***********************************************************************
c     Onshl calc the on-shell momenta for every channel.
c     The sign of the momenta (or which sheet of energies) is also
c          decided in this roution by looking the relative position
c          of the current and previous momentum on the complex plane.
c     NOTE: Generally, Knew is complex because of complex physical mass
c           of Delta. To show 3D tmatrix struct, even the energy becomes
c           complex which causes complex onshell Knew.
c***********************************************************************

      implicit none

      integer  nch, i
      double precision mtotal, mdiff, xold, yold, xnew, ynew, rek, imk
      double precision zero, two, three, four

      integer  Nifty, Isheet
      double precision mpion, Mass
      double complex enew, eold, Knew(2), Knew2(2), kold2(2), Dk2de(2)

      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)
      common /sheet/ Isheet(2)

      parameter (zero = 0.0d0, two = 2.0d0, three = 3.0d0, four = 4.0d0)

      nch = Nifty(2)
      if (eold .eq. 0.0d0) eold = mpion + Mass(1)

c     this if-statement serves as initialization for nifty(4)=3
c     relativistic kimematics
         do 10000 i = 1, nch
            mtotal = mpion + Mass(i)
            mdiff = mpion - Mass(i)
            Kold2(i) = (eold-mtotal)*(eold-mdiff)
     &                *(eold+mtotal)*(eold+mdiff)/(four*eold**2)
            Knew2(i) = (enew-mtotal)*(enew-mdiff)
     &                *(enew+mtotal)*(enew+mdiff)/(four*enew**2)
            Dk2de(i) = (enew**four - mtotal**two*mdiff**two)
     &                /(two*enew**three)
10000    continue

c the mometum k = sqrt(k**2), but the sign is undecided.
c we are try to decide the sign relative the the old k by looking where
c the straight line connecting Kold2 and Knew2 cross the 'cut'
c The 'cut' is choosen as from k=0 to +infinity alone positive real axis
c The real axis is belong to upper half plane

      do 10020 i = 1, nch
         xold = dble(Kold2(i))
         yold = imag(Kold2(i))
         xnew = dble(Knew2(i))
         ynew = imag(Knew2(i))
         if (yold*ynew .lt. zero) then
            if ((xold*ynew-xnew*yold)/(ynew-yold) .gt. zero)
     &         Isheet(i) = -Isheet(i)
         else if ((yold .eq. zero) .and. (xold .gt. zero)) then
            if (ynew .lt. zero) Isheet(i) = -Isheet(i)
         else if ((ynew .eq. zero) .and. (xnew .gt. zero)) then
            if (yold .lt. zero) Isheet(i) = -Isheet(i)
         endif
10020 continue

c compute channel momenta Knew(i)
      do 10030 i = 1, nch
         Knew(i) = sqrt(Knew2(i))
         rek = dble(Knew(i))
         imk = imag(Knew(i))
         if (Isheet(i) .gt. 0) then
c force 0 <= arg(Knew(i) < Pi
            if (imk .lt. zero) then
               Knew(i) = -Knew(i)
            else if ((imk .eq. zero) .and. (rek .lt. zero)) then
               Knew(i) = -Knew(i)
            endif
         else
c force Pi <= arg(Knew(i) < 2Pi
            if (imk .gt. zero) then
               Knew(i) = -Knew(i)
            else if ((imk .eq. zero) .and. (rek .gt. zero)) then
               Knew(i) = -Knew(i)
            endif
         endif
10030 continue

      eold = enew

      return
      end

c***********************************************************************

      subroutine PrtKin(e, K)

c***********************************************************************
c     subroutine PrtKin print out the kinematic variables such as
c        on-shell momentum for every channel and the total energy
c        and Isheet variable
c***********************************************************************

      implicit none
```

```
      integer Nifty, Isheet, nch, i
      double complex e, K(2)

c     sheet.h: parameters to decide which sheet the energy is on.
      common /sheet/ Isheet(2)
      common /control/ Nifty(15)

c*********************************************************
c
      nch = Nifty(2)
c write out the channel momenta and Isheet
      write(8, 900) e
      write(8, 910) (Isheet(i), i = 1, nch)
      write(8, 920)
      do 10000 i = 1, nch
         write(8, 930) i, K(i)
10000 continue

  900 format('0', 'Energy = ', 2(f8.3, 1x), 'MeV')
  910 format(1x, 'ISHEET: ', 5i2)
  920 format(1x, 'Channel momemta (MeV/c):')
  930 format(1x, i2, 1x, 2f12.5)

      return
      end
c
c**********************************************************************

      subroutine TMtrx(npts, Dk2de, Wgf, Vsuper, Tonshl)
c
c**********************************************************************
c     Tmtrx calculates scattering length flj(ich, jch) from a Wgf
c        array and a symmetric vlj potential matrix by formular
c       T = V + VGT ===> T = 1/(1-VG)*V ===> T = 1/G*1/(1/G-V)*V
c       input Super V matrix, output on-shell channel T matrix
c**********************************************************************
c
      implicit none
c     LOCAL variables
      integer first
      integer npts1, nchpt, nch, lmax
      integer ichpt, jchpt, l, ich, jch, id, j
      double precision one, zero
      double complex Fmtrx(128, 128), temp
c     SHARED variables
      integer npts, Nifty, Kpvt(128)
      double complex Dk2de(2), Wgf(128), B(128)
      double complex Vsuper(128,128,6), Tonshl(2,2,6)
      double complex Thalf(64),Tdiag(64)
      common /half/ Thalf,Tdiag
      common /control/ Nifty(15)

      parameter (zero = 0.0d0, one = 1.0d0)
      data first/1/
c      zabs1(zdumy) = abs(dble(zdumy)) + abs(imag(zdumy))

c**********************************************************************
c
      nch = nifty(2)
      lmax = nifty(6)
      npts1 = npts + 1
      nchpt = nch*npts1

c     solve Matrix Lippmann-Schwinger Eq.
      if (nifty(7) .eq. 1) then
```

```
c  #undef 1
      if (Nifty(11) .eq. 1) then
c        write(8,*) '@tmatrx, nifty(11)=1, not implemented'
c       stop
      endif

      do 10050 l = 1, lmax
         do jchpt = 1, nchpt
            do ichpt = 1, nchpt
               Fmtrx(ichpt, jchpt) = -Vsuper(ichpt, jchpt, l)*Wgf(jchpt)
            enddo
            Fmtrx(jchpt, jchpt) = one + Fmtrx(jchpt, jchpt)
         enddo

         call ludcmp(Fmtrx, nchpt, 128, Kpvt, temp, id)

         do ich = 1, nch
            do ichpt = 1, nchpt
               B(ichpt) = Vsuper(ichpt, ich*npts1, l)
            enddo
            call lubksb(Fmtrx, nchpt, 128, Kpvt, B)
            do jch = 1, nch
               Tonshl(jch, ich, l) = B(jch*npts1)
            enddo
         enddo

c        calc diagonal t matrix
             if (Nifty(8).eq.1 .and. l.eq.nifty(6)) then
                do j = 1, npts
                  do ichpt = 1, nchpt
                    B(ichpt) = Vsuper(ichpt, j, l)
                  enddo
                  call lubksb(Fmtrx, nchpt, 128, Kpvt, B)
                  Tdiag(j) = B(j)
                enddo
             endif
10050 continue

      else

c     using born approximation
         do 10200 l = 1, lmax
            do jch = 1, nch
               jchpt = jch*npts1
               do ich = 1, nch
                  ichpt = ich*npts1
                  Tonshl(ich, jch, l) = Vsuper(ichpt, jchpt, l)
               enddo
            enddo
10200    continue
      endif

      return
      end

c**********************************************************************

      subroutine WtgFcn(npts, P, W, ze, Zk, Zk2, Dk2de, Wgf)

c**********************************************************************
c     subroutine WtgFcn calculates the WeighTed Green's FunCtioN at
c        each grid points
c     See G. He's note 'From Integral Equation to Matrix Equation',
c        Nov. 13, 14, 1990.
c     In folder 'Notes on Scattering Theory'
```

```
c     wgf (weighted green's function) is same as D_{i,gamma} in
c        the notes
c**********************************************************************

      implicit none

      integer npts1,ipt,nch,ich,ichpt
      double precision Egama(64,2), zero, two, pi

      integer Nifty,npts
      double precision P(64), W(64), mpion, Mass
      double complex zi, ze, Zk(2), Zk2(2), Dk2de(2), Wgf(128)

      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)

      parameter (zero = 0.0d0, two = 2.0d0)
      parameter (pi=3.14159265358979323846d0)
      parameter (zi = (0.0d0, 1.0d0))
c***********************************

      nch = Nifty(2)
      npts1 = npts + 1
c     relativistic kinematics
         do 10010 ich = 1, nch
            do ipt = 1, npts
               Egama(ipt, ich) = sqrt(P(ipt)**2 + mpion**2)
     &                       +sqrt(P(ipt)**2 + Mass(ich)**2)
            enddo
10010    continue

c calculate wgf on grid points
      do 10050 ich = 1, nch
         do ipt = 1, npts
            ichpt = (ich-1)*npts1+ipt
            Wgf(ichpt) = two*W(ipt)*P(ipt)*P(ipt)
     &                  /(pi*(ze-Egama(ipt,ich)))
         enddo
10050 continue

c calculate wgf on on-shell points
      do 10070 ich = 1, nch
         ichpt = ich*npts1
         Wgf(ichpt) = zero
c         if ( ((dble(ze)-mpion-Mass(ich)).gt.zero) .or.
c     &   (imag(Zk(ich)).lt.zero) ) then
        if ( (dble(ze)-mpion-Mass(ich)).gt.zero ) then

c       3/9 lu change .le. to .lt. to cure delta channel below md-mpi.
c       i.e. for above threshold or decaying resonant state
            do ipt = 1, npts
               Wgf(ichpt) = Wgf(ichpt) + W(ipt)/(Zk2(ich)-P(ipt)**2)
            enddo
            Wgf(ichpt) = -Wgf(ichpt)*two*Zk2(ich)*Dk2de(ich)/pi
c     &            -zi*Zk(ich)*Dk2de(ich)
c           if im(k) !=0 iepslon prescription irrelevent
            if (imag(Zk(ich)) .le. zero) then
               Wgf(ichpt) = Wgf(ichpt) - zi*Zk(ich)*Dk2de(ich)
            else
               Wgf(ichpt) = Wgf(ichpt) + zi*Zk(ich)*Dk2de(ich)
            endif
         endif
10070 continue

      return
      end

c**********************************************************************


      subroutine Chi2(iflag, m, n, X, Fvec, Fjac, ldFjac)

c**********************************************************************
c     Chi2 is designed to be called by snls1. It returns a vector
c     of deviation Fvec(m) which would be minimized in snls1.
c
c     iflag is assighed by snls1. if iflag = 0, Chi2 only prints,
c                iflag != 0, chi2 only calculates, doesn't print.
c**********************************************************************
c
      integer i, ner
      integer n, m, ldfjac
      double precision er1, er2, ep11, xtol
c      double precision Diag(3), Qtf(3)
c      double precision Wa1(3), Wa2(3), Wa3(3), Wa4(450)
c
      integer Nifty, nfit, iflag, Fidx
      double precision X(n), Xpole(2), Fvec(m), Fjac(ldFjac,n)
      double precision exp_xs, th_xs, zero, Scale(6)
      double precision f0,mgb,alpha,c1,c2,Mbare,mpion,Mass,rcbm

      common /control/ Nifty(15)
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
      common /mass/ mpion, Mass(2)

      common /data/ exp_xs(20, 6), th_xs(20, 6), nfit
      external Det1Mvg
      parameter (zero = 0.0d0)
      data xtol /1.0d-2/, epsfcn /1.0d-6/, ep11 /940.d0/
      data Scale /1.d5,1.d6,1.d6,1.d5,1.d5,1.d4/

c******************************************************

      call GetFit(x)

c     print x Fvec if iflag == 0
      if (iflag .eq. 0) then
         write(8, 900)
         call PrtPot()

      else
c     calculate chi-square if iflag != 0
c       chi-square from scattering data
c       nfit is assgined in obs (to be 5)

        call Obs(er1, er2, ner)

        do i = 1, nfit
        Fvec(i) = ((exp_xs(i,6) - th_xs(i,6))/exp_xs(i,6))**2
        enddo

c       constraints from P11 phase shift
        Fvec(4) = abs((exp_xs(2,3) - th_xs(2,3))/exp_xs(2,3))
        Fvec(5) = ((exp_xs(5,3) - th_xs(5,3))/exp_xs(5,3))**2

c       chi-square from p11 pole (nucleon mass)
         Xpole(1) = ep11 - mpion - Mass(1)
         Xpole(2) = 0.
         Nifty(6) = 3
```

```
         call GetPole(Xpole,xtol,epsfcn)
         Nifty(6) = 6

         Fvec(nfit+1) = (Xpole(1)+mpion+Mass(1)-ep11)**2
c        Fvec(nfit+2) = (X(2)/ep11)**2
         do i = 1, nfit+1
                Fvec(i) = Scale(i)*Fvec(i)
         enddo
         write(*,999) f0,Mbare(1),Mbare(2),
     &          Xpole(1)+mpion+Mass(1),Fvec(3),Fvec(6)
        endif

 999  format('CHI2::f0,Mbare=',3e12.5,2x,'POLE=',3e12.4)
 900  format('0', 10('-'), 'Chi2, Beginning of printing mode ', 10('-'))
 940  format('0', 'Chi-Square = ', e13.7,
     &     3x, 'Chi-Square/per-freedom = ', e13.7)
 945  format('0', 'Chi-Square = ', e13.7)

      return
      end

c***********************************************************************


      subroutine GetFit(X)

c***********************************************************************
c     Getfit gets the potential parameters from fitting parameter
c     array X(), according to the fitting index array Fidx()
c***********************************************************************

      integer Fidx, n
      double precision f0, mgb, alpha, c1, c2, Mbare, rcbm
      double precision X(3)
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)

      n = 0
      if (Fidx(1) .eq. 1) then
         n = n + 1
         f0 = X(n)
      endif

      if (Fidx(2) .eq. 1) then
         n = n + 1
         Mbare(1) = X(n)
      endif

      if (Fidx(3) .eq. 1) then
         n = n + 1
         Mbare(2) = X(n)
      endif

      return
      end

c***********************************************************************

      subroutine Getpole(X, xtol, epsfcn)
      implicit none
      integer iopt, nprint, info, ldfjac, n
      integer njev, nfev, mode, maxfev, ml, mu, lr
      double precision fjac(450, 3), Diag(3), Qtf(3)
      double precision Wa1(3), Wa2(3), Wa3(3), Wa4(450)
      double precision X(3), Fvec(450), factor, xtol, epsfcn, r(3)
      double precision mpion, Mass

      common /mass/ mpion, Mass(2)

      external Det1Mvg,jac

c***********************************************************************
c
c     call snsq searching zero of det(1-VG) for one particular channel
         n = 2
         iopt = 2
         ldfjac = 2
         maxfev = 100*(n+1)*2
         ml = n - 1
         mu = n - 1
         mode = 1
         factor = 100.
         nprint = 1
         lr = (n*(n+1))/2
         call snsq(Det1Mvg, jac, iopt, n, X, Fvec, Fjac,
     &        ldfjac, xtol, maxfev, ml, mu, epsfcn, Diag, mode,
     &        factor, nprint, info, nfev, njev, r, lr, Qtf,
     &        Wa1, Wa2, Wa3, Wa4)
         if (info .eq. 1) then
            write(8, 930) info, X(1)+mpion+Mass(1), X(2),
     &                        Fvec(1), Fvec(2)
         else
            write(8, 935) info
         endif
c      endif

 930  format('0', 10('='), ' Final BS (POLE) Results ', 10('='), /,
     &     1x, 'Info = ', i4, /,
     &     1x, 'Re(E)   = ', e12.6, 3x, 'Im(E)   = ', e12.6, /,
     &     1x, 'Re(Det) = ', e12.6, 3x, 'Im(Det) = ', e12.6, /,
     &     1x, 47('='))
 935  format('0', 'Zero not fount. Info = ', i3, /,
     &     1x, 'See doc of snsq for more details')
        return
        end

c
c*****************************************************************
c
         subroutine Jac(n,x,fvec,fjac,ldfjac,iflag)
c
c       jac is the name of the user-supplied subroutine which calculates
c         the jacobian.  if iopt=1, then jac must be declared in an
c         external statement in the user calling program, and should be
c         written as follows. We never actually use this!!! (use iopt = 2)
         integer n,ldfjac,iflag
         real x(n),fvec(n),fjac(ldfjac,n)
c         ----------
c         calculate the jacobian at x and return this
c         matrix in fjac.  fvec contains the function
c         values at x and should not be altered.
c         ----------
         return
         end
c     Minimization routines
c     mainly two codes:  snsl1 and snsq
c     revised by Oregon State Univ theory group.
c
c*****************************************************************
c
c*****************************************************************
```

```
      subroutine chkder(m,n,x,fvec,fjac,ldfjac,xp,fvecp,mode,err)
c***begin prologue  chkder
c***date written   800301   (yymmdd)
c***revision date  830608   (yymmdd)
c***category no.  f3,g4c
c***keywords  gradients,jacobian,minpack,nonlinear
c***author  hiebert k.l. (snla)
c***purpose  checks the gradients of m nonlinear functions in n
c            variables, evaluated at a point x, for consistency
c            with the functions themselves.
c***description
c
c   this subroutine is a companion routine to snls1,snls1e,snsq,and
c   snsqe which may be used to check the calculation of the jacobian.
c
c     subroutine chkder
c
c     this subroutine checks the gradients of m nonlinear functions
c     in n variables, evaluated at a point x, for consistency with
c     the functions themselves. the user must call ckder twice,
c     first with mode = 1 and then with mode = 2.
c
c     mode = 1. on input, x must contain the point of evaluation.
c               on output, xp is set to a neighboring point.
c
c     mode = 2. on input, fvec must contain the functions and the
c                         rows of fjac must contain the gradients
c                         of the respective functions each evaluated
c                         at x, and fvecp must contain the functions
c                         evaluated at xp.
c               on output, err contains measures of correctness of
c                          the respective gradients.
c
c     the subroutine does not perform reliably if cancellation or
c     rounding errors cause a severe loss of significance in the
c     evaluation of a function. therefore, none of the components
c     of x should be unusually small (in particular, zero) or any
c     other value which may cause loss of significance.
c
c     the subroutine statement is
c
c       subroutine chkder(m,n,x,fvec,fjac,ldfjac,xp,fvecp,mode,err)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of functions.
c
c       n is a positive integer input variable set to the number
c         of variables.
c
c       x is an input array of length n.
c
c       fvec is an array of length m. on input when mode = 2,
c         fvec must contain the functions evaluated at x.
c
c       fjac is an m by n array. on input when mode = 2,
c         the rows of fjac must contain the gradients of
c         the respective functions evaluated at x.
c
c       ldfjac is a positive integer input parameter not less than m
c         which specifies the leading dimension of the array fjac.
c
c       xp is an array of length n. on output when mode = 1,
c         xp is set to a neighboring point of x.
c
c       fvecp is an array of length m. on input when mode = 2,
c         fvecp must contain the functions evaluated at xp.
c
c       mode is an integer input variable set to 1 on the first call
c         and 2 on the second. other values of mode are equivalent
c         to mode = 1.
c
c       err is an array of length m. on output when mode = 2,
c         err contains measures of correctness of the respective
c         gradients. if there is no severe loss of significance,
c         then if err(i) is 1.0 the i-th gradient is correct,
c         while if err(i) is 0.0 the i-th gradient is incorrect.
c         for values of err between 0.0 and 1.0, the categorization
c         is less certain. in general, a value of err(i) greater
c         than 0.5 indicates that the i-th gradient is probably
c         correct, while a value of err(i) less than 0.5 indicates
c         that the i-th gradient is probably incorrect.
c
c     subprograms called
c
c       slatec supplied ... r1mach
c
c       fortran supplied ... abs,log10,sqrt
c
c     argonne national laboratory. minpack project. march 1980.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c***references  powell, m. j. d., *a hybrid method for nonlinear
c                 equations*, 'numerical methods for nonlinear
c                 algebraic equations', p. rabinowitz, editor,
c                 gordon and breach, 1970.
c***routines called  r1mach
c***end prologue  chkder
      implicit none
      integer m,n,ldfjac,mode
      double precision x(n),fvec(m),fjac(ldfjac,n),xp(n),fvecp(m),err(m)
      integer i,j
      double precision eps,epsf,epslog,epsmch,factor,one,temp,zero
      double precision r1mach
      data factor,one,zero /1.0d2,1.0d0,0.0d0/
c
c***first executable statement  chkder
      epsmch = r1mach(4)

      eps = sqrt(epsmch)
c
      if (mode .eq. 2) go to 20
c
c        mode = 1.
c
         do 10 j = 1, n
            temp = eps*abs(x(j))
            if (temp .eq. zero) temp = eps
            xp(j) = x(j) + temp
   10       continue
         go to 70
   20 continue
c
c        mode = 2.
c
         epsf = factor*epsmch
         epslog = log10(eps)
         do 30 i = 1, m
            err(i) = zero
```

```
   30       continue
         do 50 j = 1, n
            temp = abs(x(j))
            if (temp .eq. zero) temp = one
            do 40 i = 1, m
               err(i) = err(i) + temp*fjac(i,j)
   40          continue
   50       continue
         do 60 i = 1, m
            temp = one
            if (fvec(i) .ne. zero .and. fvecp(i) .ne. zero
     1          .and. abs(fvecp(i)-fvec(i)) .ge. epsf*abs(fvec(i)))
     2         temp = eps*abs((fvecp(i)-fvec(i))/eps-err(i))
     3                /(abs(fvec(i)) + abs(fvecp(i)))
            err(i) = one
            if (temp .gt. epsmch .and. temp .lt. eps)
     1         err(i) = (log10(temp) - epslog)/epslog
            if (temp .ge. eps) err(i) = zero
   60       continue
   70 continue
c
      return
c
c     last card of subroutine chkder.
c
      end


c***********************************************************
c
      subroutine dogleg(n,r,lr,diag,qtb,delta,x,wa1,wa2)
c***begin prologue  dogleg
c***refer to  snsq,snsqe
c
c     **********
c     subroutine dogleg
c
c     given an m by n matrix a, an n by n nonsingular diagonal
c     matrix d, an m-vector b, and a positive number delta, the
c     problem is to determine the convex combination x of the
c     gauss-newton and scaled gradient directions that minimizes
c     (a*x - b) in the least squares sense, subject to the
c     restriction that the euclidean norm of d*x be at most delta.
c
c     this subroutine completes the solution of the problem
c     if it is provided with the necessary information from the
c     qr factorization of a. that is, if a = q*r, where q has
c     orthogonal columns and r is an upper triangular matrix,
c     then dogleg expects the full upper triangle of r and
c     the first n components of (q transpose)*b.
c
c     the subroutine statement is
c
c       subroutine dogleg(n,r,lr,diag,qtb,delta,x,wa1,wa2)
c
c     where
c
c       n is a positive integer input variable set to the order of r.
c
c       r is an input array of length lr which must contain the upper
c         triangular matrix r stored by rows.
c
c       lr is a positive integer input variable not less than
c         (n*(n+1))/2.
c
c       diag is an input array of length n which must contain the
c         diagonal elements of the matrix d.
c
c       qtb is an input array of length n which must contain the first
c         n elements of the vector (q transpose)*b.
c
c       delta is a positive input variable which specifies an upper
c         bound on the euclidean norm of d*x.
c
c       x is an output array of length n which contains the desired
c         convex combination of the gauss-newton direction and the
c         scaled gradient direction.
c
c       wa1 and wa2 are work arrays of length n.
c
c     subprograms called
c
c       minpack-supplied ... r1mach,enorm
c
c       fortran-supplied ... abs,amax1,amin1,sqrt
c
c     minpack. version of july 1978.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  enorm,r1mach
c***end prologue  dogleg
      integer n,lr
      double precision delta
      double precision r(lr),diag(n),qtb(n),x(n),wa1(n),wa2(n)
      integer i,j,jj,jp1,k,l
      double precision alpha,bnorm,epsmch,gnorm,one,qnorm
      double precision sgnorm,sum,temp,zero
      double precision r1mach,enorm
      data one,zero /1.0d0,0.0d0/
c***first executable statement  dogleg
      epsmch = r1mach(4)
c
c     first, calculate the gauss-newton direction.
c
      jj = (n*(n + 1))/2 + 1
      do 50 k = 1, n
         j = n - k + 1
         jp1 = j + 1
         jj = jj - k
         l = jj + 1
         sum = zero
         if (n .lt. jp1) go to 20
         do 10 i = jp1, n
            sum = sum + r(l)*x(i)
            l = l + 1
   10       continue
   20    continue
         temp = r(jj)
         if (temp .ne. zero) go to 40
         l = j
         do 30 i = 1, j
            temp = max(temp,abs(r(l)))
            l = l + n - i
   30       continue
         temp = epsmch*temp
         if (temp .eq. zero) temp = epsmch
   40    continue
         x(j) = (qtb(j) - sum)/temp
   50    continue
c
```

```
c     test whether the gauss-newton direction is acceptable.
c
      do 60 j = 1, n
         wa1(j) = zero
         wa2(j) = diag(j)*x(j)
   60    continue
      qnorm = enorm(n,wa2)
      if (qnorm .le. delta) go to 140
c
c     the gauss-newton direction is not acceptable.
c     next, calculate the scaled gradient direction.
c
      l = 1
      do 80 j = 1, n
         temp = qtb(j)
         do 70 i = j, n
            wa1(i) = wa1(i) + r(l)*temp
            l = l + 1
   70       continue
         wa1(j) = wa1(j)/diag(j)
   80    continue
c
c     calculate the norm of the scaled gradient direction,
c     normalize, and rescale the gradient.
c
      gnorm = enorm(n,wa1)
      sgnorm = zero
      alpha = delta/qnorm
      if (gnorm .eq. zero) go to 120
      do 90 j = 1, n
         wa1(j) = (wa1(j)/gnorm)/diag(j)
   90    continue
c
c     calculate the point along the scaled gradient
c     at which the quadratic is minimized.
c
      l = 1
      do 110 j = 1, n
         sum = zero
         do 100 i = j, n
            sum = sum + r(l)*wa1(i)
            l = l + 1
  100       continue
         wa2(j) = sum
  110    continue
      temp = enorm(n,wa2)
      sgnorm = (gnorm/temp)/temp
c
c     test whether the scaled gradient direction is acceptable.
c
      alpha = zero
      if (sgnorm .ge. delta) go to 120
c
c     the scaled gradient direction is not acceptable.
c     finally, calculate the point along the dogleg
c     at which the quadratic is minimized.
c
      bnorm = enorm(n,qtb)
      temp = (bnorm/gnorm)*(bnorm/qnorm)*(sgnorm/delta)
      temp = temp - (delta/qnorm)*(sgnorm/delta)**2
     1       + sqrt((temp-(delta/qnorm))**2
     2              +(one-(delta/qnorm)**2)*(one-(sgnorm/delta)**2))
      alpha = ((delta/qnorm)*(one - (sgnorm/delta)**2))/temp
  120 continue
c
```

```
c     form appropriate convex combination of the gauss-newton
c     direction and the scaled gradient direction.
c
      temp = (one - alpha)*min(sgnorm,delta)
      do 130 j = 1, n
         x(j) = temp*wa1(j) + alpha*x(j)
  130    continue
  140 continue
      return
c
c     last card of subroutine dogleg.
c
      end


c****************************************************************
c
      function enorm(n,x)
c***begin prologue  enorm
c***refer to  snls1,snls1e,snsq,snsqe
c
c     **********
c     function enorm
c
c     given an n-vector x, this function calculates the
c     euclidean norm of x.
c
c     the euclidean norm is computed by accumulating the sum of
c     squares in three different sums. the sums of squares for the
c     small and large components are scaled so that no overflows
c     occur. non-destructive underflows are permitted. underflows
c     and overflows do not occur in the computation of the unscaled
c     sum of squares for the intermediate components.
c     the definitions of small, intermediate and large components
c     depend on two constants, rdwarf and rgiant. the main
c     restrictions on these constants are that rdwarf**2 not
c     underflow and rgiant**2 not overflow. the constants
c     given here are suitable for every known computer.
c
c     the function statement is
c
c       real function enorm(n,x)
c
c     where
c
c       n is a positive integer input variable.
c
c       x is an input array of length n.
c
c     subprograms called
c
c       fortran-supplied ... abs,sqrt
c
c     minpack. version of october 1979.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  (none)
c***end prologue  enorm
      implicit none
      integer n, i
      double precision x(n), enorm
      double precision agiant,floatn,one,rdwarf,rgiant,s1,s2,s3,
     1     xabs,x1max,x3max,zero
      data one,zero,rdwarf,rgiant /1.0d0,0.0d0,3.834d-20,1.304d19/
c***first executable statement  enorm
```

```
      s1 = zero
      s2 = zero
      s3 = zero
      x1max = zero
      x3max = zero
      floatn = n
      agiant = rgiant/floatn
      do 90 i = 1, n
         xabs = abs(x(i))
         if (xabs .gt. rdwarf .and. xabs .lt. agiant) go to 70
            if (xabs .le. rdwarf) go to 30
c
c              sum for large components.
c
               if (xabs .le. x1max) go to 10
                  s1 = one + s1*(x1max/xabs)**2
                  x1max = xabs
                  go to 20
   10          continue
                  s1 = s1 + (xabs/x1max)**2
   20          continue
               go to 60
   30       continue
c
c              sum for small components.
c
               if (xabs .le. x3max) go to 40
                  s3 = one + s3*(x3max/xabs)**2
                  x3max = xabs
                  go to 50
   40          continue
                  if (xabs .ne. zero) s3 = s3 + (xabs/x3max)**2
   50          continue
   60       continue
            go to 80
   70    continue
c
c           sum for intermediate components.
c
            s2 = s2 + xabs**2
   80    continue
   90    continue
c
c     calculation of norm.
c
      if (s1 .eq. zero) go to 100
         enorm = x1max*sqrt(s1+(s2/x1max)/x1max)
         go to 130
  100 continue
         if (s2 .eq. zero) go to 110
            if (s2 .ge. x3max)
     1         enorm = sqrt(s2*(one+(x3max/s2)*(x3max*s3)))
            if (s2 .lt. x3max)
     1         enorm = sqrt(x3max*((s2/x3max)+(x3max*s3)))
            go to 120
  110    continue
            enorm = x3max*sqrt(s3)
  120    continue
  130 continue
      return
c
c     last card of function enorm.
c
      end
```

```
c******************************************************************
c
      subroutine fdjac1(fcn,n,x,fvec,fjac,ldfjac,iflag,ml,mu,epsfcn,
     1   wa1,wa2)
c***begin prologue  fdjac1
c***refer to  snsq,snsqe
c
c               subroutine fdjac1
c
c     this subroutine computes a forward-difference approximation
c     to the n by n jacobian matrix associated with a specified
c     problem of n functions in n variables. if the jacobian has
c     a banded form, then function evaluations are saved by only
c     approximating the nonzero terms.
c
c     the subroutine statement is
c
c       subroutine fdjac1(fcn,n,x,fvec,fjac,ldfjac,iflag,ml,mu,epsfcn,
c                         wa1,wa2)
c
c     where
c
c       fcn is the name of the user-supplied subroutine which
c         calculates the functions. fcn must be declared
c         in an external statement in the user calling
c         program, and should be written as follows.
c
c         subroutine fcn(n,x,fvec,iflag)
c         integer n,iflag
c         real x(n),fvec(n)
c         ----------
c         calculate the functions at x and
c         return this vector in fvec.
c         ----------
c         return
c         end
c
c         the value of iflag should not be changed by fcn unless
c         the user wants to terminate execution of fdjac1.
c         in this case set iflag to a negative integer.
c
c       n is a positive integer input variable set to the number
c         of functions and variables.
c
c       x is an input array of length n.
c
c       fvec is an input array of length n which must contain the
c         functions evaluated at x.
c
c       fjac is an output n by n array which contains the
c         approximation to the jacobian matrix evaluated at x.
c
c       ldfjac is a positive integer input variable not less than n
c         which specifies the leading dimension of the array fjac.
c
c       iflag is an integer variable which can be used to terminate
c         the execution of fdjac1. see description of fcn.
c
c       ml is a nonnegative integer input variable which specifies
c         the number of subdiagonals within the band of the
c         jacobian matrix. if the jacobian is not banded, set
c         ml to at least n - 1.
c
```

```
c       epsfcn is an input variable used in determining a suitable
c         step length for the forward-difference approximation. this
c         approximation assumes that the relative errors in the
c         functions are of the order of epsfcn. if epsfcn is less
c         than the machine precision, it is assumed that the relative
c         errors in the functions are of the order of the machine
c         precision.
c
c       mu is a nonnegative integer input variable which specifies
c         the number of superdiagonals within the band of the
c         jacobian matrix. if the jacobian is not banded, set
c         mu to at least n - 1.
c
c       wa1 and wa2 are work arrays of length n. if ml + mu + 1 is at
c         least n, then the jacobian is considered dense, and wa2 is
c         not referenced.
c
c     subprograms called
c
c       minpack-supplied ... r1mach
c
c       fortran-supplied ... abs,amax1,sqrt
c
c     minpack. version of june 1979.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  r1mach
c***end prologue  fdjac1
      integer n,ldfjac,iflag,ml,mu
      double precision epsfcn
      double precision x(n),fvec(n),fjac(ldfjac,n),wa1(n),wa2(n)
      integer i,j,k,msum
      double precision eps,epsmch,h,temp,zero
      double precision r1mach, tmp
      external fcn
      data zero /0.0d0/
c***first executable statement  fdjac1
      epsmch = r1mach(4)
c
c      tmp = max(epsfcn,epsmch)
c     ??? epsfcn(never appear in LHS) is changed in pinCDM code
c         (4.436652884672037E-315), why ? through argument ??
c     ??? since epsmch=e-16, tmp should not be zero, but eps turns out
c         to be zero, even max works ok.

      eps = 1.0e-4

      msum = ml + mu + 1
      if (msum .lt. n) go to 40
c
c        computation of dense approximate jacobian.
c
         do 20 j = 1, n
            temp = x(j)
            h = eps*abs(temp)
            if (h .eq. zero) h = eps
            x(j) = temp + h
            call fcn(n,x,wa1,iflag)
            if (iflag .lt. 0) go to 30
            x(j) = temp
            do 10 i = 1, n
               fjac(i,j) = (wa1(i) - fvec(i))/h
   10          continue
   20       continue
   30    continue
         go to 110
   40 continue
c
c        computation of banded approximate jacobian.
c
         do 90 k = 1, msum
            do 60 j = k, n, msum
               wa2(j) = x(j)
               h = eps*abs(wa2(j))
               if (h .eq. zero) h = eps
               x(j) = wa2(j) + h
   60          continue
            call fcn(n,x,wa1,iflag)
            if (iflag .lt. 0) go to 100
            do 80 j = k, n, msum
               x(j) = wa2(j)
               h = eps*abs(wa2(j))
               if (h .eq. zero) h = eps
               do 70 i = 1, n
                  fjac(i,j) = zero
                  if (i .ge. j - mu .and. i .le. j + ml)
     1               fjac(i,j) = (wa1(i) - fvec(i))/h
   70             continue
   80          continue
   90       continue
  100    continue
  110 continue
      return
c
c     last card of subroutine fdjac1.
c
      end


c*******************************************************************
c
      subroutine fdjac3(fcn,m,n,x,fvec,fjac,ldfjac,iflag,epsfcn,wa)
c***begin prologue  fdjac3
c***refer to  snls1,snls1e
c
c     **********
c     subroutine fdjac3
c
c     this subroutine computes a forward-difference approximation
c     to the m by n jacobian matrix associated with a specified
c     problem of m functions in n variables.
c
c     the subroutine statement is
c
c       subroutine fdjac3(fcn,m,n,x,fvec,fjac,ldfjac,iflag,epsfcn,wa)
c
c     where
c
c       fcn is the name of the user-supplied subroutine which
c         calculates the functions. fcn must be declared
c         in an external statement in the user calling
c         program, and should be written as follows.
c
c         subroutine fcn(iflag,m,n,x,fvec,fjac,ldfjac)
c         integer ldfjac,m,n,iflag
c         real x(n),fvec(m),fjac(ldfjac,n)
c         ----------
c         when iflag.eq.1 calculate the functions at x and
c         return this vector in fvec.
c         ----------
```

```
c         return
c         end
c
c         the value of iflag should not be changed by fcn unless
c         the user wants to terminate execution of fdjac3.
c         in this case set iflag to a negative integer.
c
c       m is a positive integer input variable set to the number
c         of functions.
c
c       n is a positive integer input variable set to the number
c         of variables. n must not exceed m.
c
c       x is an input array of length n.
c
c       fvec is an input array of length m which must contain the
c         functions evaluated at x.
c
c       fjac is an output m by n array which contains the
c         approximation to the jacobian matrix evaluated at x.
c
c       ldfjac is a positive integer input variable not less than m
c         which specifies the leading dimension of the array fjac.
c
c       iflag is an integer variable which can be used to terminate
c         the execution of fdjac3. see description of fcn.
c
c       epsfcn is an input variable used in determining a suitable
c         step length for the forward-difference approximation. this
c         approximation assumes that the relative errors in the
c         functions are of the order of epsfcn. if epsfcn is less
c         than the machine precision, it is assumed that the relative
c         errors in the functions are of the order of the machine
c         precision.
c
c       wa is a work array of length m.
c
c     subprograms called
c
c       user-supplied ...... fcn
c
c       minpack-supplied ... r1mach
c
c       fortran-supplied ... abs,max,sqrt
c
c     minpack. version of june 1979.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  r1mach
c***end prologue  fdjac3
      implicit none
      integer m,n,ldfjac,iflag
      double precision epsfcn, r1mach
      double precision x(n),fvec(m),fjac(ldfjac,n),wa(m)
      integer i,j
      double precision eps,epsmch,h,temp,zero
      external fcn
      data zero /0.0d0/
c***first executable statement  fdjac3
      epsmch = r1mach(4)
c
      eps = sqrt(max(epsfcn,epsmch))
c      set iflag=1 to indicate that function values
c           are to be returned by fcn.
      iflag = 1
      do 20 j = 1, n
         temp = x(j)
         h = eps*abs(temp)
         if (h .eq. zero) h = eps
         x(j) = temp + h
         call fcn(iflag,m,n,x,wa,fjac,ldfjac)
         if (iflag .lt. 0) go to 30
         x(j) = temp
         do 10 i = 1, m
            fjac(i,j) = (wa(i) - fvec(i))/h
   10       continue
   20    continue
   30 continue
      return
c
c     last card of subroutine fdjac3.
c
      end


c************************************************************************
c
      subroutine fdump
c***begin prologue  fdump
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  z
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  symbolic dump (should be locally written).
c***description
c        ***note*** machine dependent routine
c        fdump is intended to be replaced by a locally written
c        version which produces a symbolic dump.  failing this,
c        it should be replaced by a version which prints the
c        subprogram nesting list.  note that this dump must be
c        printed on each of up to five files, as indicated by the
c        xgetua routine.  see xsetua and xgetua for details.
c
c     written by ron jones, with slatec common math library subcommittee
c     latest revision ---  23 may 1979
c***routines called  (none)
c***end prologue  fdump
c***first executable statement  fdump
      implicit none
      write(8, *) 'FDUMP CALLED'
      return
      end


c********************************************************
c
      function i1mach(i)
c***begin prologue  i1mach
c***date written   750101   (yymmdd)
c***revision date  840405   (yymmdd)
c***category no.  r1
c***keywords  machine constants
c***author  fox, p. a., (bell labs)
c           hall, a. d., (bell labs)
c           schryer, n. l., (bell labs)
c***purpose  returns integer machine dependent constants
c***description
c
c * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
c   these machine constant routines must be activated for
c   a particular environment.
c * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
c
c     i1mach can be used to obtain machine-dependent parameters
c     for the local machine environment.  it is a function
c     subroutine with one (input) argument, and can be called
c     as follows, for example
c
c          k = i1mach(i)
c
c     where i=1,...,16.  the (output) value of k above is
c     determined by the (input) value of i.  the results for
c     various values of i are discussed below.
c
c  i/o unit numbers.
c    i1mach( 1) = the standard input unit.
c    i1mach( 2) = the standard output unit.
c    i1mach( 3) = the standard punch unit.
c    i1mach( 4) = the standard error message unit.
c
c  words.
c    i1mach( 5) = the number of bits per integer storage unit.
c    i1mach( 6) = the number of characters per integer storage unit.
c
c  integers.
c    assume integers are represented in the s-digit, base-a form
c
c               sign ( x(s-1)*a**(s-1) + ... + x(1)*a + x(0) )
c
c               where 0 .le. x(i) .lt. a for i=0,...,s-1.
c    i1mach( 7) = a, the base.
c    i1mach( 8) = s, the number of base-a digits.
c    i1mach( 9) = a**s - 1, the largest magnitude.
c
c  floating-point numbers.
c    assume floating-point numbers are represented in the t-digit,
c    base-b form
c               sign (b**e)*( (x(1)/b) + ... + (x(t)/b**t) )
c
c               where 0 .le. x(i) .lt. b for i=1,...,t,
c               0 .lt. x(1), and emin .le. e .le. emax.
c    i1mach(10) = b, the base.
c
c  single-precision
c    i1mach(11) = t, the number of base-b digits.
c    i1mach(12) = emin, the smallest exponent e.
c    i1mach(13) = emax, the largest exponent e.
c
c  double-precision
c    i1mach(14) = t, the number of base-b digits.
c    i1mach(15) = emin, the smallest exponent e.
c    i1mach(16) = emax, the largest exponent e.
c
c  to alter this function for a particular environment,
c  the desired set of data statements should be activated by
c  removing the c from column 1.  also, the values of
c  i1mach(1) - i1mach(4) should be checked for consistency
c  with the local operating system.
c***references  fox p.a., hall a.d., schryer n.l.,*framework for a
c                 portable library*, acm transactions on mathematical
c                 software, vol. 4, no. 2, june 1978, pp. 177-188.
c***routines called  (none)
c***end prologue  i1mach
c
      implicit none

      integer imach(16),output,i1mach,i
      equivalence (imach(4),output)
c
c machine constants for Ridge RX/V and IBM AIX/370
c
      data imach( 1) /    5 /
      data imach( 2) /    6 /
      data imach( 3) /    6 /
      data imach( 4) /    6 /
      data imach( 5) /   32 /
      data imach( 6) /    4 /
      data imach( 7) /    2 /
      data imach( 8) /   31 /
      data imach( 9) / 2147483647 /
      data imach(10) /    2 /
      data imach(11) /   23 /
      data imach(12) / -127 /
      data imach(13) /  128 /
      data imach(14) /   53 /
      data imach(15) / -1023 /
      data imach(16) /  1024 /
c
c     machine constants for the burroughs 1700 system.
c
c     data imach( 1) /    7 /
c     data imach( 2) /    2 /
c     data imach( 3) /    2 /
c     data imach( 4) /    2 /
c     data imach( 5) /   36 /
c     data imach( 6) /    4 /
c     data imach( 7) /    2 /
c     data imach( 8) /   33 /
c     data imach( 9) / z1ffffffff /
c     data imach(10) /    2 /
c     data imach(11) /   24 /
c     data imach(12) / -256 /
c     data imach(13) /  255 /
c     data imach(14) /   60 /
c     data imach(15) / -256 /
c     data imach(16) /  255 /
c
c     machine constants for the burroughs 5700 system.
c
c     data imach( 1) /   5 /
c     data imach( 2) /   6 /
c     data imach( 3) /   7 /
c     data imach( 4) /   6 /
c     data imach( 5) /  48 /
c     data imach( 6) /   6 /
c     data imach( 7) /   2 /
c     data imach( 8) /  39 /
c     data imach( 9) / o0007777777777777 /
c     data imach(10) /   8 /
c     data imach(11) /  13 /
c     data imach(12) / -50 /
c     data imach(13) /  76 /
c     data imach(14) /  26 /
c     data imach(15) / -50 /
c     data imach(16) /  76 /
c
c     machine constants for the burroughs 6700/7700 systems.
c
c     data imach( 1) /   5 /
```

```
c     data imach( 2) /   6 /
c     data imach( 3) /   7 /
c     data imach( 4) /   6 /
c     data imach( 5) /  48 /
c     data imach( 6) /   6 /
c     data imach( 7) /   2 /
c     data imach( 8) /  39 /
c     data imach( 9) / o0007777777777777 /
c     data imach(10) /   8 /
c     data imach(11) /  13 /
c     data imach(12) / -50 /
c     data imach(13) /  76 /
c     data imach(14) /  26 /
c     data imach(15) / -32754 /
c     data imach(16) /  32780 /
c
c     machine constants for the convex c-120 (native mode)
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    0 /
c     data imach( 4) /    6 /
c     data imach( 5) /   32 /
c     data imach( 6) /    4 /
c     data imach( 7) /    2 /
c     data imach( 8) /   31 /
c     data imach( 9) / 2147483647 /
c     data imach(10) /    2 /
c     data imach(11) /   24 /
c     data imach(12) / -127 /
c     data imach(13) /  127 /
c     data imach(14) /   53 /
c     data imach(15) / -1023 /
c     data imach(16) /  1023 /
c
c     machine constants for the convex (native mode)
c     with -r8 option
c
c     data imach( 1) /     5 /
c     data imach( 2) /     6 /
c     data imach( 3) /     0 /
c     data imach( 4) /     6 /
c     data imach( 5) /    32 /
c     data imach( 6) /     4 /
c     data imach( 7) /     2 /
c     data imach( 8) /    31 /
c     data imach( 9) / 2147483647 /
c     data imach(10) /     2 /
c     data imach(11) /    53 /
c     data imach(12) / -1023 /
c     data imach(13) /  1023 /
c     data imach(14) /    53 /
c     data imach(15) / -1023 /
c     data imach(16) /  1023 /
c
c     machine constants for the convex c-120 (ieee mode)
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    0 /
c     data imach( 4) /    6 /
c     data imach( 5) /   32 /
c     data imach( 6) /    4 /
c     data imach( 7) /    2 /
c     data imach( 8) /   31 /
c     data imach( 9) / 2147483647 /
c     data imach(10) /    2 /
c     data imach(11) /   24 /
c     data imach(12) / -125 /
c     data imach(13) /  128 /
c     data imach(14) /   53 /
c     data imach(15) / -1021 /
c     data imach(16) /  1024 /
c
c     machine constants for the convex (ieee mode)
c     with -r8 option
c
c     data imach( 1) /     5 /
c     data imach( 2) /     6 /
c     data imach( 3) /     0 /
c     data imach( 4) /     6 /
c     data imach( 5) /    32 /
c     data imach( 6) /     4 /
c     data imach( 7) /     2 /
c     data imach( 8) /    31 /
c     data imach( 9) / 2147483647 /
c     data imach(10) /     2 /
c     data imach(11) /    53 /
c     data imach(12) / -1021 /
c     data imach(13) /  1024 /
c     data imach(14) /    53 /
c     data imach(15) / -1021 /
c     data imach(16) /  1024 /
c
c     machine constants for the cdc cyber 170 series (ftn5).
c
c      data imach( 1) /    5 /
c      data imach( 2) /    6 /
c      data imach( 3) /    7 /
c      data imach( 4) /    6 /
c      data imach( 5) /   60 /
c      data imach( 6) /   10 /
c      data imach( 7) /    2 /
c      data imach( 8) /   48 /
c      data imach( 9) / o"00007777777777777777" /
c      data imach(10) /    2 /
c      data imach(11) /   48 /
c      data imach(12) / -974 /
c      data imach(13) / 1070 /
c      data imach(14) /   96 /
c      data imach(15) / -927 /
c      data imach(16) / 1070 /
c
c     machine constants for the cdc 170/180 series using nos/ve
c
c     data imach( 1) /     5 /
c     data imach( 2) /     6 /
c     data imach( 3) /     7 /
c     data imach( 4) /     6 /
c     data imach( 5) /    64 /
c     data imach( 6) /     8 /
c     data imach( 7) /     2 /
c     data imach( 8) /    63 /
c     data imach( 9) / 9223372036854775807 /
c     data imach(10) /     2 /
c     data imach(11) /    47 /
c     data imach(12) / -4095 /
c     data imach(13) /  4094 /
c     data imach(14) /    94 /
c     data imach(15) / -4095 /
```

```
c     data imach(16) /  4094 /
c
c     machine constants for the cdc cyber 205
c
c     data imach( 1) /      5 /
c     data imach( 2) /      6 /
c     data imach( 3) /      7 /
c     data imach( 4) /      6 /
c     data imach( 5) /     64 /
c     data imach( 6) /      8 /
c     data imach( 7) /      2 /
c     data imach( 8) /     47 /
c     data imach( 9) / x'00007fffffffffff' /
c     data imach(10) /      2 /
c     data imach(11) /     47 /
c     data imach(12) / -28625 /
c     data imach(13) /  28718 /
c     data imach(14) /     94 /
c     data imach(15) / -28625 /
c     data imach(16) /  28718 /
c
c
c     machine constants for the cdc 6000/7000 series.
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    7 /
c     data imach( 4) /6loutput/
c     data imach( 5) /   60 /
c     data imach( 6) /   10 /
c     data imach( 7) /    2 /
c     data imach( 8) /   48 /
c     data imach( 9) / 00007777777777777777b /
c     data imach(10) /    2 /
c     data imach(11) /   47 /
c     data imach(12) / -929 /
c     data imach(13) / 1070 /
c     data imach(14) /   94 /
c     data imach(15) / -929 /
c     data imach(16) / 1069 /
c
c     machine constants for the cray 1
c
c     data imach( 1) /   100 /
c     data imach( 2) /   101 /
c     data imach( 3) /   102 /
c     data imach( 4) /   101 /
c     data imach( 5) /    64 /
c     data imach( 6) /     8 /
c     data imach( 7) /     2 /
c     data imach( 8) /    63 /
c     data imach( 9) /  777777777777777777777b /
c     data imach(10) /     2 /
c     data imach(11) /    47 /
c     data imach(12) / -8189 /
c     data imach(13) /  8190 /
c     data imach(14) /    94 /
c     data imach(15) / -8099 /
c     data imach(16) /  8190 /
c
c     machine constants for the data general eclipse s/200
c
c     data imach( 1) /   11 /
c     data imach( 2) /   12 /
c     data imach( 3) /    8 /
c     data imach( 4) /   10 /
c     data imach( 5) /   16 /
c     data imach( 6) /    2 /
c     data imach( 7) /    2 /
c     data imach( 8) /   15 /
c     data imach( 9) /32767 /
c     data imach(10) /   16 /
c     data imach(11) /    6 /
c     data imach(12) /  -64 /
c     data imach(13) /   63 /
c     data imach(14) /   14 /
c     data imach(15) /  -64 /
c     data imach(16) /   63 /
c
c     machine constants for the harris 220
c
c     data imach( 1) /       5 /
c     data imach( 2) /       6 /
c     data imach( 3) /       0 /
c     data imach( 4) /       6 /
c     data imach( 5) /      24 /
c     data imach( 6) /       3 /
c     data imach( 7) /       2 /
c     data imach( 8) /      23 /
c     data imach( 9) / 8388607 /
c     data imach(10) /       2 /
c     data imach(11) /      23 /
c     data imach(12) /    -127 /
c     data imach(13) /     127 /
c     data imach(14) /      38 /
c     data imach(15) /    -127 /
c     data imach(16) /     127 /
c
c     machine constants for the honeywell 600/6000 series.
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /   43 /
c     data imach( 4) /    6 /
c     data imach( 5) /   36 /
c     data imach( 6) /    6 /
c     data imach( 7) /    2 /
c     data imach( 8) /   35 /
c     data imach( 9) / o377777777777 /
c     data imach(10) /    2 /
c     data imach(11) /   27 /
c     data imach(12) / -127 /
c     data imach(13) /  127 /
c     data imach(14) /   63 /
c     data imach(15) / -127 /
c     data imach(16) /  127 /
c
c     machine constants for the hp 2100
c     3 word double precision option with ftn4
c
c     data imach(1) /      5/
c     data imach(2) /      6 /
c     data imach(3) /      4 /
c     data imach(4) /      1 /
c     data imach(5) /     16 /
c     data imach(6) /      2 /
c     data imach(7) /      2 /
c     data imach(8) /     15 /
c     data imach(9) /  32767 /
c     data imach(10)/      2 /
```

```
c     data imach(11)/     23 /
c     data imach(12)/   -128 /
c     data imach(13)/    127 /
c     data imach(14)/     39 /
c     data imach(15)/   -128 /
c     data imach(16)/    127 /
c
c     machine constants for the hp 2100
c     4 word double precision option with ftn4
c
c     data imach(1) /      5 /
c     data imach(2) /      6 /
c     data imach(3) /      4 /
c     data imach(4) /      1 /
c     data imach(5) /     16 /
c     data imach(6) /      2 /
c     data imach(7) /      2 /
c     data imach(8) /     15 /
c     data imach(9) /  32767 /
c     data imach(10)/      2 /
c     data imach(11)/     23 /
c     data imach(12)/   -128 /
c     data imach(13)/    127 /
c     data imach(14)/     55 /
c     data imach(15)/   -128 /
c     data imach(16)/    127 /
c
c     machine constants for the ibm 360/370 series,
c     the xerox sigma 5/7/9, the sel systems 85/86, and
c     the perkin elmer (interdata) 7/32.
c
c     data imach( 1) /   5 /
c     data imach( 2) /   6 /
c     data imach( 3) /   7 /
c     data imach( 4) /   6 /
c     data imach( 5) /  32 /
c     data imach( 6) /   4 /
c     data imach( 7) /  16 /
c     data imach( 8) /  31 /
c     data imach( 9) / z7fffffff /
c     data imach(10) /  16 /
c     data imach(11) /   6 /
c     data imach(12) / -64 /
c     data imach(13) /  63 /
c     data imach(14) /  14 /
c     data imach(15) / -64 /
c     data imach(16) /  63 /
c
c     machine constants for the pdp-10 (ka processor).
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    5 /
c     data imach( 4) /    6 /
c     data imach( 5) /   36 /
c     data imach( 6) /    5 /
c     data imach( 7) /    2 /
c     data imach( 8) /   35 /
c     data imach( 9) / "377777777777 /
c     data imach(10) /    2 /
c     data imach(11) /   27 /
c     data imach(12) / -128 /
c     data imach(13) /  127 /
c     data imach(14) /   54 /
c     data imach(15) / -101 /
c     data imach(16) /  127 /
c
c     machine constants for the pdp-10 (ki processor).
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    5 /
c     data imach( 4) /    6 /
c     data imach( 5) /   36 /
c     data imach( 6) /    5 /
c     data imach( 7) /    2 /
c     data imach( 8) /   35 /
c     data imach( 9) / "377777777777 /
c     data imach(10) /    2 /
c     data imach(11) /   27 /
c     data imach(12) / -128 /
c     data imach(13) /  127 /
c     data imach(14) /   62 /
c     data imach(15) / -128 /
c     data imach(16) /  127 /
c
c     machine constants for pdp-11 fortran supporting
c     32-bit integer arithmetic.
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    5 /
c     data imach( 4) /    6 /
c     data imach( 5) /   32 /
c     data imach( 6) /    4 /
c     data imach( 7) /    2 /
c     data imach( 8) /   31 /
c     data imach( 9) / 2147483647 /
c     data imach(10) /    2 /
c     data imach(11) /   24 /
c     data imach(12) / -127 /
c     data imach(13) /  127 /
c     data imach(14) /   56 /
c     data imach(15) / -127 /
c     data imach(16) /  127 /
c
c     machine constants for pdp-11 fortran supporting
c     16-bit integer arithmetic.
c
c     data imach( 1) /    5 /
c     data imach( 2) /    6 /
c     data imach( 3) /    5 /
c     data imach( 4) /    6 /
c     data imach( 5) /   16 /
c     data imach( 6) /    2 /
c     data imach( 7) /    2 /
c     data imach( 8) /   15 /
c     data imach( 9) / 32767 /
c     data imach(10) /    2 /
c     data imach(11) /   24 /
c     data imach(12) / -127 /
c     data imach(13) /  127 /
c     data imach(14) /   56 /
c     data imach(15) / -127 /
c     data imach(16) /  127 /
c
c     machine constants for the sun 3 (68881 or fpa)
c
c      data imach( 1) /    5 /
c      data imach( 2) /    6 /
```

```
c       data imach( 3) /    6 /
c       data imach( 4) /    0 /
c       data imach( 5) /   32 /
c       data imach( 6) /    4 /
c       data imach( 7) /    2 /
c       data imach( 8) /   31 /
c       data imach( 9) / 2147483647 /
c       data imach(10) /    2 /
c       data imach(11) /   24 /
c       data imach(12) / -125 /
c       data imach(13) /  128 /
c       data imach(14) /   53 /
c       data imach(15) / -1021 /
c       data imach(16) /  1024 /
c
c      machine constants for the univac 1100 series. ftn compiler
c
c      data imach( 1) /    5 /
c      data imach( 2) /    6 /
c      data imach( 3) /    1 /
c      data imach( 4) /    6 /
c      data imach( 5) /   36 /
c      data imach( 6) /    4 /
c      data imach( 7) /    2 /
c      data imach( 8) /   35 /
c      data imach( 9) / o377777777777 /
c      data imach(10) /    2 /
c      data imach(11) /   27 /
c      data imach(12) / -128 /
c      data imach(13) /  127 /
c      data imach(14) /   60 /
c      data imach(15) /-1024 /
c      data imach(16) / 1023 /
c
c      machine constants for the vax 11/780
c
c      data imach(1) /    5 /
c      data imach(2) /    6 /
c      data imach(3) /    5 /
c      data imach(4) /    6 /
c      data imach(5) /   32 /
c      data imach(6) /    4 /
c      data imach(7) /    2 /
c      data imach(8) /   31 /
c      data imach(9) /2147483647 /
c      data imach(10)/    2 /
c      data imach(11)/   24 /
c      data imach(12)/ -127 /
c      data imach(13)/  127 /
c      data imach(14)/   56 /
c      data imach(15)/ -127 /
c      data imach(16)/  127 /
c
c      machine constants for the z80 microprocessor
c
c      data imach( 1) /     1/
c      data imach( 2) /     1/
c      data imach( 3) /     0/
c      data imach( 4) /     1/
c      data imach( 5) /    16/
c      data imach( 6) /     2/
c      data imach( 7) /     2/
c      data imach( 8) /    15/
c      data imach( 9) / 32767/
c      data imach(10) /     2/
c      data imach(11) /    24/
c      data imach(12) /  -127/
c      data imach(13) /   127/
c      data imach(14) /    56/
c      data imach(15) /  -127/
c      data imach(16) /   127/
c
c
c***first executable statement  i1mach
c      if (i .lt. 1  .or.  i .gt. 16) go to 10
c
       i1mach=imach(i)
       return
c
c   10 continue
c       write(output,9000)
c9000 format('1error    1 in i1mach - i out of bounds ')
c
c      call fdump
c
c
c      stop
       end


c*********************************************
c
      function icamax(n,cx,incx)
c***begin prologue  icamax
c***date written   791001   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d1a2
c***keywords  blas,complex,linear algebra,maximum component,vector
c***author  lawson, c. l., (jpl)
c           hanson, r. j., (snla)
c           kincaid, d. r., (u. of texas)
c           krogh, f. t., (jpl)
c***purpose  find largest component of complex vector
c***description
c
c                b l a s  subprogram
c    description of parameters
c
c     --input--
c        n  number of elements in input vector(s)
c       cx  complex vector with n elements
c     incx  storage spacing between elements of cx
c
c     --output--
c   icamax  smallest index (zero if n .le. 0)
c
c      returns the index of the component of cx having the
c      largest sum of magnitudes of real and imaginary parts.
c     icamax = first i, i = 1 to n, to minimize
c        abs(real(cx(1-incx+i*incx))) + abs(imag(cx(1-incx+i*incx)))
c***references  lawson c.l., hanson r.j., kincaid d.r., krogh f.t.,
c                 *basic linear algebra subprograms for fortran usage*,
c                 algorithm no. 539, transactions on mathematical
c                 software, volume 5, number 3, september 1979, 308-323
c***routines called  (none)
c***end prologue  icamax
c
      implicit none
      integer icamax,i,ii,incx,n,ns
      double precision summax, sumri
```

```
      double complex cx(1)
c***first executable statement  icamax
      icamax = 0
      if(n.le.0) return
      icamax = 1
      if(n .le. 1) return
      ns = n*incx
      ii = 1
      summax = abs(dble(cx(1))) + abs(imag(cx(1)))
          do 20 i=1,ns,incx
          sumri = abs(dble(cx(i))) + abs(imag(cx(i)))
          if(summax.ge.sumri) go to 10
          summax = sumri
          icamax = ii
   10     ii = ii + 1
   20     continue
      return
      end


c***********************************************************************
c
      function j4save(iwhich,ivalue,iset)
c***begin prologue  j4save
c***refer to  xerror
c     abstract
c        j4save saves and recalls several global variables needed
c        by the library error handling routines.
c
c     description of parameters
c      --input--
c        iwhich - index of item desired.
c                = 1 refers to current error number.
c                = 2 refers to current error control flag.
c                 = 3 refers to current unit number to which error
c                    messages are to be sent.  (0 means use standard.)
c                 = 4 refers to the maximum number of times any
c                     message is to be printed (as set by xermax).
c                 = 5 refers to the total number of units to which
c                     each error message is to be written.
c                 = 6 refers to the 2nd unit for error messages
c                 = 7 refers to the 3rd unit for error messages
c                 = 8 refers to the 4th unit for error messages
c                 = 9 refers to the 5th unit for error messages
c        ivalue - the value to be set for the iwhich-th parameter,
c                 if iset is .true. .
c        iset   - if iset=.true., the iwhich-th parameter will be
c                 given the value, ivalue.  if iset=.false., the
c                 iwhich-th parameter will be unchanged, and ivalue
c                 is a dummy parameter.
c      --output--
c        the (old) value of the iwhich-th parameter will be returned
c        in the function value, j4save.
c
c     written by ron jones, with slatec common math library subcommittee
c    adapted from bell laboratories port library error handler
c     latest revision ---  23 may 1979
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  (none)
c***end prologue  j4save
      implicit none
      logical iset
      integer iparam(9)
      integer j4save, iwhich, ivalue
      save iparam
      data iparam(1),iparam(2),iparam(3),iparam(4)/0,2,0,10/
      data iparam(5)/1/
      data iparam(6),iparam(7),iparam(8),iparam(9)/0,0,0,0/
c***first executable statement  j4save
      j4save = iparam(iwhich)
      if (iset) iparam(iwhich) = ivalue
      return
      end


c***********************************************************************
c
      subroutine lmpar(n,r,ldr,ipvt,diag,qtb,delta,par,x,sigma,wa1,wa2)
c***begin prologue  lmpar
c***refer to  snls1,snls1e
c
c     **********
c     subroutine lmpar
c
c     given an m by n matrix a, an n by n nonsingular diagonal
c     matrix d, an m-vector b, and a positive number delta,
c     the problem is to determine a value for the parameter
c     par such that if x solves the system
c
c           a*x = b ,     sqrt(par)*d*x = 0 ,
c
c     in the least squares sense, and dxnorm is the euclidean
c     norm of d*x, then either par is zero and
c
c           (dxnorm-delta) .le. 0.1*delta ,
c
c     or par is positive and
c
c           abs(dxnorm-delta) .le. 0.1*delta .
c
c     this subroutine completes the solution of the problem
c     if it is provided with the necessary information from the
c     qr factorization, with column pivoting, of a. that is, if
c     a*p = q*r, where p is a permutation matrix, q has orthogonal
c     columns, and r is an upper triangular matrix with diagonal
c     elements of nonincreasing magnitude, then lmpar expects
c     the full upper triangle of r, the permutation matrix p,
c     and the first n components of (q transpose)*b. on output
c     lmpar also provides an upper triangular matrix s such that
c
c            t   t                   t
c           p *(a *a + par*d*d)*p = s *s .
c
c     s is employed within lmpar and may be of separate interest.
c
c     only a few iterations are generally needed for convergence
c     of the algorithm. if, however, the limit of 10 iterations
c     is reached, then the output par will contain the best
c     value obtained so far.
c
c     the subroutine statement is
c
c       subroutine lmpar(n,r,ldr,ipvt,diag,qtb,delta,par,x,sigma,
c                        wa1,wa2)
c
c     where
c
c       n is a positive integer input variable set to the order of r.
```

```
c
c       r is an n by n array. on input the full upper triangle
c         must contain the full upper triangle of the matrix r.
c         on output the full upper triangle is unaltered, and the
c         strict lower triangle contains the strict upper triangle
c         (transposed) of the upper triangular matrix s.
c
c       ldr is a positive integer input variable not less than n
c         which specifies the leading dimension of the array r.
c
c       ipvt is an integer input array of length n which defines the
c         permutation matrix p such that a*p = q*r. column j of p
c         is column ipvt(j) of the identity matrix.
c
c       diag is an input array of length n which must contain the
c         diagonal elements of the matrix d.
c
c       qtb is an input array of length n which must contain the first
c         n elements of the vector (q transpose)*b.
c
c       delta is a positive input variable which specifies an upper
c         bound on the euclidean norm of d*x.
c
c       par is a nonnegative variable. on input par contains an
c         initial estimate of the levenberg-marquardt parameter.
c         on output par contains the final estimate.
c
c       x is an output array of length n which contains the least
c         squares solution of the system a*x = b, sqrt(par)*d*x = 0,
c         for the output par.
c
c       sigma is an output array of length n which contains the
c         diagonal elements of the upper triangular matrix s.
c
c       wa1 and wa2 are work arrays of length n.
c
c     subprograms called
c
c       minpack-supplied ... r1mach,enorm,qrsolv
c
c       fortran-supplied ... abs,max,min,sqrt
c
c     minpack. version of june 1979.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  enorm,qrsolv,r1mach
c***end prologue  lmpar
      implicit none
      integer n,ldr
      integer ipvt(n)
      double precision delta,par
      double precision r(ldr,n),diag(n),qtb(n)
      double precision x(n),sigma(n),wa1(n),wa2(n)
      integer i,iter,j,jm1,jp1,k,l,nsing
      double precision dxnorm,dwarf,fp,gnorm,parc,parl
      double precision paru,p1,p001,sum,temp,zero
      double precision r1mach,enorm
      data p1,p001,zero /1.0d-1,1.0d-3,0.0d0/
c***first executable statement  lmpar
      dwarf = r1mach(1)
c
c     compute and store in x the gauss-newton direction. if the
c     jacobian is rank-deficient, obtain a least squares solution.
c
      nsing = n
```

```
      do 10 j = 1, n
         wa1(j) = qtb(j)
         if (r(j,j) .eq. zero .and. nsing .eq. n) nsing = j - 1
         if (nsing .lt. n) wa1(j) = zero
   10    continue
      if (nsing .lt. 1) go to 50
      do 40 k = 1, nsing
         j = nsing - k + 1
         wa1(j) = wa1(j)/r(j,j)
         temp = wa1(j)
         jm1 = j - 1
         if (jm1 .lt. 1) go to 30
         do 20 i = 1, jm1
            wa1(i) = wa1(i) - r(i,j)*temp
   20       continue
   30    continue
   40    continue
   50 continue
      do 60 j = 1, n
         l = ipvt(j)
         x(l) = wa1(j)
   60    continue
c
c     initialize the iteration counter.
c     evaluate the function at the origin, and test
c     for acceptance of the gauss-newton direction.
c
      iter = 0
      do 70 j = 1, n
         wa2(j) = diag(j)*x(j)
   70    continue
      dxnorm = enorm(n,wa2)
      fp = dxnorm - delta
      if (fp .le. p1*delta) go to 220
c
c     if the jacobian is not rank deficient, the newton
c     step provides a lower bound, parl, for the zero of
c     the function. otherwise set this bound to zero.
c
      parl = zero
      if (nsing .lt. n) go to 120
      do 80 j = 1, n
         l = ipvt(j)
         wa1(j) = diag(l)*(wa2(l)/dxnorm)
   80    continue
      do 110 j = 1, n
         sum = zero
         jm1 = j - 1
         if (jm1 .lt. 1) go to 100
         do 90 i = 1, jm1
            sum = sum + r(i,j)*wa1(i)
   90       continue
  100    continue
         wa1(j) = (wa1(j) - sum)/r(j,j)
  110    continue
      temp = enorm(n,wa1)
      parl = ((fp/delta)/temp)/temp
  120 continue
c
c     calculate an upper bound, paru, for the zero of the function.
c
      do 140 j = 1, n
         sum = zero
         do 130 i = 1, j
            sum = sum + r(i,j)*qtb(i)
```

```
  130       continue
         l = ipvt(j)
         wa1(j) = sum/diag(l)
  140    continue
      gnorm = enorm(n,wa1)
      paru = gnorm/delta
      if (paru .eq. zero) paru = dwarf/min(delta,p1)
c
c     if the input par lies outside of the interval (parl,paru),
c     set par to the closer endpoint.
c
      par = max(par,parl)
      par = min(par,paru)
      if (par .eq. zero) par = gnorm/dxnorm
c
c     beginning of an iteration.
c
  150 continue
         iter = iter + 1
c
c        evaluate the function at the current value of par.
c
         if (par .eq. zero) par = max(dwarf,p001*paru)
         temp = sqrt(par)
         do 160 j = 1, n
            wa1(j) = temp*diag(j)
  160       continue
         call qrsolv(n,r,ldr,ipvt,wa1,qtb,x,sigma,wa2)
         do 170 j = 1, n
            wa2(j) = diag(j)*x(j)
  170       continue
         dxnorm = enorm(n,wa2)
         temp = fp
         fp = dxnorm - delta
c
c        if the function is small enough, accept the current value
c        of par. also test for the exceptional cases where parl
c        is zero or the number of iterations has reached 10.
c
         if (abs(fp) .le. p1*delta
     1       .or. parl .eq. zero .and. fp .le. temp
     2            .and. temp .lt. zero .or. iter .eq. 10) go to 220
c
c        compute the newton correction.
c
         do 180 j = 1, n
            l = ipvt(j)
            wa1(j) = diag(l)*(wa2(l)/dxnorm)
  180       continue
         do 210 j = 1, n
            wa1(j) = wa1(j)/sigma(j)
            temp = wa1(j)
            jp1 = j + 1
            if (n .lt. jp1) go to 200
            do 190 i = jp1, n
               wa1(i) = wa1(i) - r(i,j)*temp
  190          continue
  200       continue
  210       continue
         temp = enorm(n,wa1)
         parc = ((fp/delta)/temp)/temp
c
c        depending on the sign of the function, update parl or paru.
c
         if (fp .gt. zero) parl = max(parl,par)
         if (fp .lt. zero) paru = min(paru,par)
c
c        compute an improved estimate for par.
c
         par = max(parl,par+parc)
c
c        end of an iteration.
c
         go to 150
  220 continue
c
c     termination.
c
      if (iter .eq. 0) par = zero
      return
c
c     last card of subroutine lmpar.
c
      end
      subroutine lubksb(a,n,np,indx,b)
        integer ii,i,n,np,indx(n),ll,j
        double complex a(np,np),sum,b(n)
c      dimension a(np,np),indx(n),b(n)
      ii=0
      do 12 i=1,n
        ll=indx(i)
        sum=b(ll)
        b(ll)=b(i)
        if (ii.ne.0)then
          do 11 j=ii,i-1
            sum=sum-a(i,j)*b(j)
11        continue
        else if (sum.ne.0.) then
          ii=i
        endif
        b(i)=sum
12    continue
      do 14 i=n,1,-1
        sum=b(i)
        if(i.lt.n)then
          do 13 j=i+1,n
            sum=sum-a(i,j)*b(j)
13        continue
        endif
        b(i)=sum/a(i,i)
14    continue
      return
      end
c*****************************************************
c     This routine is taken from numerical recipe
c     and modified for complex matrix by D. Lu
c
      subroutine ludcmp(a,n,np,indx,d,id)
        integer i,j,n,np,indx(n),nmax,k,id,imax
        parameter (nmax=100)
        double precision aamax,vv(nmax),d
        double complex a(np,np),sum,tiny,dum
      parameter (tiny=(1.0d-20,0.d0))
c      dimension a(np,np),indx(n),vv(nmax)
      d=1.
      do 12 i=1,n
        aamax=0.
        do 11 j=1,n
          if (abs(a(i,j)).gt.aamax) aamax=abs(a(i,j))
11      continue
```

```
        if (aamax.eq.0.) pause 'singular matrix.'
        vv(i)=1./aamax
12    continue
      do 19 j=1,n
        if (j.gt.1) then
          do 14 i=1,j-1
            sum=a(i,j)
            if (i.gt.1)then
              do 13 k=1,i-1
                sum=sum-a(i,k)*a(k,j)
13            continue
              a(i,j)=sum
            endif
14        continue
        endif
        aamax=0.
        do 16 i=j,n
          sum=a(i,j)
          if (j.gt.1)then
            do 15 k=1,j-1
              sum=sum-a(i,k)*a(k,j)
15          continue
            a(i,j)=sum
          endif
          dum=vv(i)*abs(sum)
          if (abs(dum).ge.aamax) then
            imax=i
            aamax=abs(dum)
          endif
16      continue
        if (j.ne.imax)then
          do 17 k=1,n
            dum=a(imax,k)
            a(imax,k)=a(j,k)
            a(j,k)=dum
17        continue
          d=-d
          vv(imax)=vv(j)
        endif
        indx(j)=imax
        if(j.ne.n)then
          if(a(j,j).eq.0.)a(j,j)=tiny
          dum=1./a(j,j)
          do 18 i=j+1,n
            a(i,j)=a(i,j)*dum
18        continue
        endif
19    continue
      if(a(n,n).eq.0.)a(n,n)=tiny
      return
      end


c********************************************
c
      subroutine qform(m,n,q,ldq,wa)
c***begin prologue  qform
c***refer to  snsq,snsqe
c
c     **********
c     subroutine qform
c
c     this subroutine proceeds from the computed qr factorization of
c     an m by n matrix a to accumulate the m by m orthogonal matrix
c     q from its factored form.
c
c     the subroutine statement is
c
c       subroutine qform(m,n,q,ldq,wa)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of a and the order of q.
c
c       n is a positive integer input variable set to the number
c         of columns of a.
c
c       q is an m by m array. on input the full lower trapezoid in
c         the first min(m,n) columns of q contains the factored form.
c         on output q has been accumulated into a square matrix.
c
c       ldq is a positive integer input variable not less than m
c         which specifies the leading dimension of the array q.
c
c       wa is a work array of length m.
c
c     subprograms called
c
c       fortran-supplied ... min0
c
c     minpack. version of january 1979.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  (none)
c***end prologue  qform
      integer m,n,ldq
      double precision q(ldq,m),wa(m)
      integer i,j,jm1,k,l,minmn,np1
      double precision one,sum,temp,zero
      data one,zero /1.0d0,0.0d0/
c***first executable statement  qform
      minmn = min(m,n)
      if (minmn .lt. 2) go to 30
      do 20 j = 2, minmn
         jm1 = j - 1
         do 10 i = 1, jm1
            q(i,j) = zero
   10       continue
   20    continue
   30 continue
c
c     initialize remaining columns to those of the identity matrix.
c
      np1 = n + 1
      if (m .lt. np1) go to 60
      do 50 j = np1, m
         do 40 i = 1, m
            q(i,j) = zero
   40       continue
         q(j,j) = one
   50    continue
   60 continue
c
c     accumulate q from its factored form.
c
      do 120 l = 1, minmn
         k = minmn - l + 1
         do 70 i = k, m
            wa(i) = q(i,k)
```

```
            q(i,k) = zero
   70       continue
         q(k,k) = one
         if (wa(k) .eq. zero) go to 110
         do 100 j = k, m
            sum = zero
            do 80 i = k, m
               sum = sum + q(i,j)*wa(i)
   80          continue
            temp = sum/wa(k)
            do 90 i = k, m
               q(i,j) = q(i,j) - temp*wa(i)
   90          continue
  100       continue
  110    continue
  120    continue
      return
c
c     last card of subroutine qform.
c
      end

c*****************************************************************
c
      subroutine qrfac(m,n,a,lda,pivot,ipvt,lipvt,sigma,acnorm,wa)
c***begin prologue  qrfac
c***refer to  snls1,snls1e,snsq,snsqe
c
c     **********
c     subroutine qrfac
c
c     this subroutine uses householder transformations with column
c     pivoting (optional) to compute a qr factorization of the
c     m by n matrix a. that is, qrfac determines an orthogonal
c     matrix q, a permutation matrix p, and an upper trapezoidal
c     matrix r with diagonal elements of nonincreasing magnitude,
c     such that a*p = q*r. the householder transformation for
c     column k, k = 1,2,...,min(m,n), is of the form
c
c                           t
c           i - (1/u(k))*u*u
c
c     where u has zeros in the first k-1 positions. the form of
c     this transformation and the method of pivoting first
c     appeared in the corresponding linpack subroutine.
c
c     the subroutine statement is
c
c       subroutine qrfac(m,n,a,lda,pivot,ipvt,lipvt,sigma,acnorm,wa)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of a.
c
c       n is a positive integer input variable set to the number
c         of columns of a.
c
c       a is an m by n array. on input a contains the matrix for
c         which the qr factorization is to be computed. on output
c         the strict upper trapezoidal part of a contains the strict
c         upper trapezoidal part of r, and the lower trapezoidal
c         part of a contains a factored form of q (the non-trivial
c         elements of the u vectors described above).
c
```

```
c       lda is a positive integer input variable not less than m
c         which specifies the leading dimension of the array a.
c
c       pivot is a logical input variable. if pivot is set .true.,
c         then column pivoting is enforced. if pivot is set .false.,
c         then no column pivoting is done.
c
c       ipvt is an integer output array of length lipvt. ipvt
c         defines the permutation matrix p such that a*p = q*r.
c         column j of p is column ipvt(j) of the identity matrix.
c         if pivot is .false., ipvt is not referenced.
c
c       lipvt is a positive integer input variable. if pivot is
c             .false., then lipvt may be as small as 1. if pivot is
c             .true., then lipvt must be at least n.
c
c       sigma is an output array of length n which contains the
c         diagonal elements of r.
c
c       acnorm is an output array of length n which contains the
c         norms of the corresponding columns of the input matrix a.
c         if this information is not needed, then acnorm can coincide
c         with sigma.
c
c       wa is a work array of length n. if pivot is .false., then wa
c         can coincide with sigma.
c
c     subprograms called
c
c       minpack-supplied ... r1mach,enorm
c       fortran-supplied ... max,sqrt,min
c
c     minpack. version of december 1978.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  enorm,r1mach
c***end prologue  qrfac
      implicit none
      integer m,n,lda,lipvt
      integer ipvt(lipvt)
      logical pivot
      double precision a(lda,n),sigma(n),acnorm(n),wa(n)
      integer i,j,jp1,k,kmax,minmn
      double precision ajnorm,epsmch,one,p05,sum,temp,zero
      double precision r1mach,enorm
      data one,p05,zero /1.0d0,5.0d-2,0.0d0/
c***first executable statement  qrfac
      epsmch = r1mach(4)
c
c     compute the initial column norms and initialize several arrays.
c
      do 10 j = 1, n
         acnorm(j) = enorm(m,a(1,j))
         sigma(j) = acnorm(j)
         wa(j) = sigma(j)
         if (pivot) ipvt(j) = j
   10    continue
c
c     reduce a to r with householder transformations.
c
      minmn = min(m,n)
      do 110 j = 1, minmn
         if (.not.pivot) go to 40
c
c        bring the column of largest norm into the pivot position.
```

```
c
         kmax = j
         do 20 k = j, n
            if (sigma(k) .gt. sigma(kmax)) kmax = k
   20       continue
         if (kmax .eq. j) go to 40
         do 30 i = 1, m
            temp = a(i,j)
            a(i,j) = a(i,kmax)
            a(i,kmax) = temp
   30       continue
         sigma(kmax) = sigma(j)
         wa(kmax) = wa(j)
         k = ipvt(j)
         ipvt(j) = ipvt(kmax)
         ipvt(kmax) = k
   40    continue
c
c        compute the householder transformation to reduce the
c        j-th column of a to a multiple of the j-th unit vector.
c
         ajnorm = enorm(m-j+1,a(j,j))
         if (ajnorm .eq. zero) go to 100
         if (a(j,j) .lt. zero) ajnorm = -ajnorm
         do 50 i = j, m
            a(i,j) = a(i,j)/ajnorm
   50       continue
         a(j,j) = a(j,j) + one
c
c        apply the transformation to the remaining columns
c        and update the norms.
c
         jp1 = j + 1
         if (n .lt. jp1) go to 100
         do 90 k = jp1, n
            sum = zero
            do 60 i = j, m
               sum = sum + a(i,j)*a(i,k)
   60          continue
            temp = sum/a(j,j)
            do 70 i = j, m
               a(i,k) = a(i,k) - temp*a(i,j)
   70          continue
            if (.not.pivot .or. sigma(k) .eq. zero) go to 80
            temp = a(j,k)/sigma(k)
            sigma(k) = sigma(k)*sqrt(max(zero,one-temp**2))
            if (p05*(sigma(k)/wa(k))**2 .gt. epsmch) go to 80
            sigma(k) = enorm(m-j,a(jp1,k))
            wa(k) = sigma(k)
   80       continue
   90       continue
  100    continue
         sigma(j) = -ajnorm
  110    continue
      return
c
c     last card of subroutine qrfac.
c
      end


c**********************************************************
c
      subroutine qrsolv(n,r,ldr,ipvt,diag,qtb,x,sigma,wa)
c***begin prologue  qrsolv
c***refer to  snls1,snls1e
c
c     **********
c     subroutine qrsolv
c
c     given an m by n matrix a, an n by n diagonal matrix d,
c     and an m-vector b, the problem is to determine an x which
c     solves the system
c
c           a*x = b ,     d*x = 0 ,
c
c     in the least squares sense.
c
c     this subroutine completes the solution of the problem
c     if it is provided with the necessary information from the
c     qr factorization, with column pivoting, of a. that is, if
c     a*p = q*r, where p is a permutation matrix, q has orthogonal
c     columns, and r is an upper triangular matrix with diagonal
c     elements of nonincreasing magnitude, then qrsolv expects
c     the full upper triangle of r, the permutation matrix p,
c     and the first n components of (q transpose)*b. the system
c     a*x = b, d*x = 0, is then equivalent to
c
c                  t       t
c           r*z = q *b ,  p *d*p*z = 0 ,
c
c     where x = p*z. if this system does not have full rank,
c     then a least squares solution is obtained. on output qrsolv
c     also provides an upper triangular matrix s such that
c
c            t   t               t
c           p *(a *a + d*d)*p = s *s .
c
c     s is computed within qrsolv and may be of separate interest.
c
c     the subroutine statement is
c
c       subroutine qrsolv(n,r,ldr,ipvt,diag,qtb,x,sigma,wa)
c
c     where
c
c       n is a positive integer input variable set to the order of r.
c
c       r is an n by n array. on input the full upper triangle
c         must contain the full upper triangle of the matrix r.
c         on output the full upper triangle is unaltered, and the
c         strict lower triangle contains the strict upper triangle
c         (transposed) of the upper triangular matrix s.
c
c       ldr is a positive integer input variable not less than n
c         which specifies the leading dimension of the array r.
c
c       ipvt is an integer input array of length n which defines the
c         permutation matrix p such that a*p = q*r. column j of p
c         is column ipvt(j) of the identity matrix.
c
c       diag is an input array of length n which must contain the
c         diagonal elements of the matrix d.
c
c       qtb is an input array of length n which must contain the first
c         n elements of the vector (q transpose)*b.
c
c       x is an output array of length n which contains the least
c         squares solution of the system a*x = b, d*x = 0.
c
```

```
c       sigma is an output array of length n which contains the
c         diagonal elements of the upper triangular matrix s.
c
c       wa is a work array of length n.
c
c     subprograms called
c
c       fortran-supplied ... abs,sqrt
c
c     minpack. version of december 1978.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  (none)
c***end prologue  qrsolv
      implicit none
      integer n,ldr,ipvt(n)
      double precision r(ldr,n),diag(n),qtb(n),x(n),sigma(n),wa(n)
      integer i,j,jp1,k,kp1,l,nsing
      double precision cos,cotan,p5,p25,qtbpj,sin,sum,tan,temp,zero
      data p5,p25,zero /5.0d-1,2.5d-1,0.0d0/
c***first executable statement  qrsolv
      do 20 j = 1, n
         do 10 i = j, n
            r(i,j) = r(j,i)
   10       continue
         x(j) = r(j,j)
         wa(j) = qtb(j)
   20    continue
c
c     eliminate the diagonal matrix d using a givens rotation.
c
      do 100 j = 1, n
c
c        prepare the row of d to be eliminated, locating the
c        diagonal element using p from the qr factorization.
c
         l = ipvt(j)
         if (diag(l) .eq. zero) go to 90
         do 30 k = j, n
            sigma(k) = zero
   30       continue
         sigma(j) = diag(l)
c
c        the transformations to eliminate the row of d
c        modify only a single element of (q transpose)*b
c        beyond the first n, which is initially zero.
c
         qtbpj = zero
         do 80 k = j, n
c
c           determine a givens rotation which eliminates the
c           appropriate element in the current row of d.
c
            if (sigma(k) .eq. zero) go to 70
            if (abs(r(k,k)) .ge. abs(sigma(k))) go to 40
               cotan = r(k,k)/sigma(k)
               sin = p5/sqrt(p25+p25*cotan**2)
               cos = sin*cotan
               go to 50
   40       continue
               tan = sigma(k)/r(k,k)
               cos = p5/sqrt(p25+p25*tan**2)
               sin = cos*tan
   50       continue
c
c           compute the modified diagonal element of r and
c           the modified element of ((q transpose)*b,0).
c
            r(k,k) = cos*r(k,k) + sin*sigma(k)
            temp = cos*wa(k) + sin*qtbpj
            qtbpj = -sin*wa(k) + cos*qtbpj
            wa(k) = temp
c
c           accumulate the tranformation in the row of s.
c
            kp1 = k + 1
            if (n .lt. kp1) go to 70
            do 60 i = kp1, n
               temp = cos*r(i,k) + sin*sigma(i)
               sigma(i) = -sin*r(i,k) + cos*sigma(i)
               r(i,k) = temp
   60          continue
   70       continue
   80       continue
   90    continue
c
c        store the diagonal element of s and restore
c        the corresponding diagonal element of r.
c
         sigma(j) = r(j,j)
         r(j,j) = x(j)
  100    continue
c
c     solve the triangular system for z. if the system is
c     singular, then obtain a least squares solution.
c
      nsing = n
      do 110 j = 1, n
         if (sigma(j) .eq. zero .and. nsing .eq. n) nsing = j - 1
         if (nsing .lt. n) wa(j) = zero
  110    continue
      if (nsing .lt. 1) go to 150
      do 140 k = 1, nsing
         j = nsing - k + 1
         sum = zero
         jp1 = j + 1
         if (nsing .lt. jp1) go to 130
         do 120 i = jp1, nsing
            sum = sum + r(i,j)*wa(i)
  120       continue
  130    continue
         wa(j) = (wa(j) - sum)/sigma(j)
  140    continue
  150 continue
c
c     permute the components of z back to components of x.
c
      do 160 j = 1, n
         l = ipvt(j)
         x(l) = wa(j)
  160    continue
      return
c
c     last card of subroutine qrsolv.
c
      end

c**********************************************
c
      function r1mach(i)
```

```
c***begin prologue  r1mach
c***date written   790101   (yymmdd)
c***revision date  860825   (yymmdd)
c***category no.  r1
c***keywords  machine constants
c***author  fox, p. a., (bell labs)
c           hall, a. d., (bell labs)
c           schryer, n. l., (bell labs)
c***purpose  returns single precision machine dependent constants
c***description
c
c     r1mach can be used to obtain machine-dependent parameters
c     for the local machine environment.  it is a function
c     subroutine with one (input) argument, and can be called
c     as follows, for example
c
c          a = r1mach(i)
c
c     where i=1,...,5.  the (output) value of a above is
c     determined by the (input) value of i.  the results for
c     various values of i are discussed below.
c
c  single-precision machine constants
c  r1mach(1) = b**(emin-1), the smallest positive magnitude.
c  r1mach(2) = b**emax*(1 - b**(-t)), the largest magnitude.
c  r1mach(3) = b**(-t), the smallest relative spacing.
c  r1mach(4) = b**(1-t), the largest relative spacing.
c  r1mach(5) = log10(b)
c***references  fox, p.a., hall, a.d., schryer, n.l, *framework for
c                 a portable library*, acm transactions on mathe-
c                 matical software, vol. 4, no. 2, june 1978,
c                 pp. 177-188.
c***routines called  xerror
c***end prologue  r1mach
c
      implicit none
      integer i, small(2), large(2), right(2), diver(2), log10(2)
      double precision rmach(5)
      double precision r1mach
c
      equivalence (rmach(1),small(1))
      equivalence (rmach(2),large(1))
      equivalence (rmach(3),right(1))
      equivalence (rmach(4),diver(1))
      equivalence (rmach(5),log10(1))
c
c This works for Ridge RX/V, MPW/Language Fortran
      data rmach(1) / 2.225073858507202d-308 /
      data rmach(2) / 1.797693134862315d+308 /
      data rmach(3) / 1.110223024625157d-016 /
      data rmach(4) / 2.220446049250313d-016 /
      data rmach(5) / 3.010299956639812d-001 /
c
c     machine constants for the burroughs 1700 system.
c
c     data rmach(1) / z400800000 /
c     data rmach(2) / z5ffffffff /
c     data rmach(3) / z4e9800000 /
c     data rmach(4) / z4ea800000 /
c     data rmach(5) / z500e730e8 /
c
c     machine constants for the burroughs 5700/6700/7700 systems.
c
c     data rmach(1) / o1771000000000000 /
c     data rmach(2) / o0777777777777777 /
c     data rmach(3) / o1311000000000000 /
c     data rmach(4) / o1301000000000000 /
c     data rmach(5) / o1157163034761675 /
c
c     machine constants for the convex c-120 (native mode)
c
c     data rmach(1) / 2.9387360e-39 /
c     data rmach(2) / 1.7014117e+38 /
c     data rmach(3) / 5.9604645e-08 /
c     data rmach(4) / 1.1920929e-07 /
c     data rmach(5) / 3.0102999e-01 /
c
c     machine constants for the convex c-120 (native mode)
c     with -r8 option
c
c     data rmach(1) / 5.562684646268007d-309 /
c     data rmach(2) / 8.988465674311577d+307 /
c     data rmach(3) / 1.110223024625157d-016 /
c     data rmach(4) / 2.220446049250313d-016 /
c     data rmach(5) / 3.010299956639812d-001 /
c
c     machine constants for the convex c-120 (ieee mode)
c
c     data rmach(1) / 1.1754945e-38 /
c     data rmach(2) / 3.4028234e+38 /
c     data rmach(3) / 5.9604645e-08 /
c     data rmach(4) / 1.1920929e-07 /
c     data rmach(5) / 3.0102999e-01 /
c
c     machine constants for the convex c-120 (ieee mode)
c     with -r8 option
c
c      data rmach(1) / 2.225073858507202d-308 /
c      data rmach(2) / 1.797693134862315d+308 /
c      data rmach(3) / 1.110223024625157d-016 /
c      data rmach(4) / 2.220446049250313d-016 /
c      data rmach(5) / 3.010299956639812d-001 /
c
c     machine constants for the cdc cyber 170 series (ftn5).
c
c      data rmach(1) / o"00014000000000000000" /
c      data rmach(2) / o"37767777777777777777" /
c      data rmach(3) / o"16404000000000000000" /
c      data rmach(4) / o"16414000000000000000" /
c      data rmach(5) / o"17164642023241175720" /
c
c     machine constants for the cdc 170/180 series using nos/ve
c
c     data rmach(1) / z"3001800000000000" /
c     data rmach(2) / z"4ffefffffffffffe" /
c     data rmach(3) / z"3fd2800000000000" /
c     data rmach(4) / z"3fd3800000000000" /
c     data rmach(5) / z"3fff9a209a84fbcf" /
c
c     machine constants for the cdc cyber 205
c
c     data rmach(1) / x'9000400000000000' /
c     data rmach(2) / x'6fff7ffffffffffe' /
c     data rmach(3) / x'ffa3400000000000' /
c     data rmach(4) / x'ffa4400000000000' /
c     data rmach(5) / x'ffd04d104d427de8' /
c
c     machine constants for the cdc 6000/7000 series.
c
c     data rmach(1) / 00564000000000000000b /
```

```
c     data rmach(2) / 377677777777777777776b /
c     data rmach(3) / 164140000000000000000b /
c     data rmach(4) / 164240000000000000000b /
c     data rmach(5) / 171646420232411757206b /
c
c     machine constants for the cray 1
c
c     data rmach(1) / 200034000000000000000b /
c     data rmach(2) / 577767777777777777776b /
c     data rmach(3) / 377224000000000000000b /
c     data rmach(4) / 377234000000000000000b /
c     data rmach(5) / 377774642023241175720b /
c
c     machine constants for the data general eclipse s/200
c
c     note - it may be appropriate to include the following card -
c     static rmach(5)
c
c     data small/20k,0/,large/77777k,177777k/
c     data right/35420k,0/,diver/36020k,0/
c     data log10/40423k,42023k/
c
c     machine constants for the harris 220
c
c     data small(1),small(2) / '20000000, '00000201 /
c     data large(1),large(2) / '37777777, '00000177 /
c     data right(1),right(2) / '20000000, '00000352 /
c     data diver(1),diver(2) / '20000000, '00000353 /
c     data log10(1),log10(2) / '23210115, '00000377 /
c
c     machine constants for the honeywell 600/6000 series.
c
c     data rmach(1) / o402400000000 /
c     data rmach(2) / o376777777777 /
c     data rmach(3) / o714400000000 /
c     data rmach(4) / o716400000000 /
c     data rmach(5) / o776464202324 /
c
c     machine constants for the hp 2100
c     3 word double precision with ftn4
c
c     data small(1), small(2) / 40000b,       1 /
c     data large(1), large(2) / 77777b, 177776b /
c     data right(1), right(2) / 40000b,    325b /
c     data diver(1), diver(2) / 40000b,    327b /
c     data log10(1), log10(2) / 46420b,  46777b /
c
c     machine constants for the hp 2100
c     4 word double precision with ftn4
c
c     data small(1), small(2) / 40000b,       1 /
c     data large91), large(2) / 77777b, 177776b /
c     data right(1), right(2) / 40000b,    325b /
c     data diver(1), diver(2) / 40000b,    327b /
c     data log10(1), log10(2) / 46420b,  46777b /
c
c     machine constants for the ibm 360/370 series,
c     the xerox sigma 5/7/9, the sel systems 85/86  and
c     the perkin elmer (interdata) 7/32.
c
c     data rmach(1) / z00100000 /
c     data rmach(2) / z7fffffff /
c     data rmach(3) / z3b100000 /
c     data rmach(4) / z3c100000 /
c     data rmach(5) / z41134413 /
c
c     machine constants for the pdp-10 (ka or ki processor).
c
c     data rmach(1) / "000400000000 /
c     data rmach(2) / "377777777777 /
c     data rmach(3) / "146400000000 /
c     data rmach(4) / "147400000000 /
c     data rmach(5) / "177464202324 /
c
c     machine constants for pdp-11 fortran supporting
c     32-bit integers (expressed in integer and octal).
c
c     data small(1) /    8388608 /
c     data large(1) / 2147483647 /
c     data right(1) /  880803840 /
c     data diver(1) /  889192448 /
c     data log10(1) / 1067065499 /
c
c     data rmach(1) / o00040000000 /
c     data rmach(2) / o17777777777 /
c     data rmach(3) / o06440000000 /
c     data rmach(4) / o06500000000 /
c     data rmach(5) / o07746420233 /
c
c     machine constants for pdp-11 fortran supporting
c     16-bit integers  (expressed in integer and octal).
c
c     data small(1),small(2) /   128,      0 /
c     data large(1),large(2) / 32767,     -1 /
c     data right(1),right(2) / 13440,      0 /
c     data diver(1),diver(2) / 13568,      0 /
c     data log10(1),log10(2) / 16282,   8347 /
c
c     data small(1),small(2) / o000200, o000000 /
c     data large(1),large(2) / o077777, o177777 /
c     data right(1),right(2) / o032200, o000000 /
c     data diver(1),diver(2) / o032400, o000000 /
c     data log10(1),log10(2) / o037632, o020233 /
c
c     machine constants for the sun 3 (68881 or fpa)
c
c     data small(1) / x'00800000' /
c     data large(1) / x'7f7fffff' /
c     data right(1) / x'33800000' /
c     data diver(1) / x'34000000' /
c     data log10(1) / x'3e9a209b' /
c
c     machine constants for the univac 1100 series.
c
c     data rmach(1) / o000400000000 /
c     data rmach(2) / o377777777777 /
c     data rmach(3) / o146400000000 /
c     data rmach(4) / o147400000000 /
c     data rmach(5) / o177464202324 /
c
c     machine constants for the vax 11/780
c    (expressed in integer and hexadecimal)
c  ***the hex format below may not be suitable for unix systems***
c  *** the integer format should be ok for unix systems***
c
c     data small(1) /       128 /
c     data large(1) /    -32769 /
c     data right(1) /     13440 /
c     data diver(1) /     13568 /
c     data log10(1) / 547045274 /
```

```
c
c     data small(1) / z00000080 /
c     data large(1) / zffff7fff /
c     data right(1) / z00003480 /
c     data diver(1) / z00003500 /
c     data log10(1) / z209b3f9a /
c
c     machine constants for the z80 microprocessor
c
c     data small(1),small(2) /     0,    256/
c     data large(1),large(2) /    -1,   -129/
c     data right(1),right(2) /     0,  26880/
c     data diver(1),diver(2) /     0,  27136/
c     data log10(1),log10(2) /  8347,  32538/
c
c
c***first executable statement  r1mach
      if (i .lt. 1  .or.  i .gt. 5)
     1   call xerror ( 'r1mach -- i out of bounds',25,1,2)
c
      r1mach = rmach(i)
      return
c
      end

c*******************************************
c
      subroutine r1mpyq(m,n,a,lda,v,w)
c***begin prologue  r1mpyq
c***refer to  snsq,snsqe
c
c     **********
c     subroutine r1mpyq
c
c     given an m by n matrix a, this subroutine computes a*q where
c     q is the product of 2*(n - 1) transformations
c
c           gv(n-1)*...*gv(1)*gw(1)*...*gw(n-1)
c
c     and gv(i), gw(i) are givens rotations in the (i,n) plane which
c     eliminate elements in the i-th and n-th planes, respectively.
c     q itself is not given, rather the information to recover the
c     gv, gw rotations is supplied.
c
c     the subroutine statement is
c
c       subroutine r1mpyq(m,n,a,lda,v,w)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of a.
c
c       n is a positive integer input variable set to the number
c         of columns of a.
c
c       a is an m by n array. on input a must contain the matrix
c         to be postmultiplied by the orthogonal matrix q
c         described above. on output a*q has replaced a.
c
c       lda is a positive integer input variable not less than m
c         which specifies the leading dimension of the array a.
c
c       v is an input array of length n. v(i) must contain the
c         information necessary to recover the givens rotation gv(i)
```

```
c         described above.
c
c       w is an input array of length n. w(i) must contain the
c         information necessary to recover the givens rotation gw(i)
c         described above.
c
c     subroutines called
c
c       fortran-supplied ... abs,sqrt
c
c     minpack. version of december 1978.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more
c     **********
c***routines called  (none)
c***end prologue  r1mpyq
      integer m,n,lda
      double precision a(lda,n),v(n),w(n)
      integer i,j,nmj,nm1
      double precision cos,one,sin,temp
      data one /1.0d0/
c***first executable statement  r1mpyq
      nm1 = n - 1
      if (nm1 .lt. 1) go to 50
      do 20 nmj = 1, nm1
         j = n - nmj
         if (abs(v(j)) .gt. one) cos = one/v(j)
         if (abs(v(j)) .gt. one) sin = sqrt(one-cos**2)
         if (abs(v(j)) .le. one) sin = v(j)
         if (abs(v(j)) .le. one) cos = sqrt(one-sin**2)
         do 10 i = 1, m
            temp = cos*a(i,j) - sin*a(i,n)
            a(i,n) = sin*a(i,j) + cos*a(i,n)
            a(i,j) = temp
   10       continue
   20    continue
c
c     apply the second set of givens rotations to a.
c
      do 40 j = 1, nm1
         if (abs(w(j)) .gt. one) cos = one/w(j)
         if (abs(w(j)) .gt. one) sin = sqrt(one-cos**2)
         if (abs(w(j)) .le. one) sin = w(j)
         if (abs(w(j)) .le. one) cos = sqrt(one-sin**2)
         do 30 i = 1, m
            temp = cos*a(i,j) + sin*a(i,n)
            a(i,n) = -sin*a(i,j) + cos*a(i,n)
            a(i,j) = temp
   30       continue
   40    continue
   50 continue
      return
c
c     last card of subroutine r1mpyq.
c
      end

c**********************************************
c
      subroutine r1updt(m,n,s,ls,u,v,w,sing)
c***begin prologue  r1updt
c***refer to  snsq,snsqe
c
c     **********
c     subroutine r1updt
c
```

```
c     given an m by n lower trapezoidal matrix s, an m-vector u,
c     and an n-vector v, the problem is to determine an
c     orthogonal matrix q such that
c
c                     t
c          (s + u*v )*q
c
c     is again lower trapezoidal.
c
c     this subroutine determines q as the product of 2*(n - 1)
c     transformations
c
c          gv(n-1)*...*gv(1)*gw(1)*...*gw(n-1)
c
c     where gv(i), gw(i) are givens rotations in the (i,n) plane
c     which eliminate elements in the i-th and n-th planes,
c     respectively. q itself is not accumulated, rather the
c     information to recover the gv, gw rotations is returned.
c
c     the subroutine statement is
c
c       subroutine r1updt(m,n,s,ls,u,v,w,sing)
c
c     where
c
c       m is a positive integer input variable set to the number
c         of rows of s.
c
c       n is a positive integer input variable set to the number
c         of columns of s. n must not exceed m.
c
c       s is an array of length ls. on input s must contain the lower
c         trapezoidal matrix s stored by columns. on output s contains
c         the lower trapezoidal matrix produced as described above.
c
c       ls is a positive integer input variable not less than
c         (n*(2*m-n+1))/2.
c
c       u is an input array of length m which must contain the
c         vector u.
c
c       v is an array of length n. on input v must contain the vector
c         v. on output v(i) contains the information necessary to
c         recover the givens rotation gv(i) described above.
c
c       w is an output array of length m. w(i) contains information
c         necessary to recover the givens rotation gw(i) described
c         above.
c
c       sing is a logical output variable. sing is set .true. if any
c         of the diagonal elements of the output s are zero. otherwise
c         sing is set .false.
c
c     subprograms called
c
c       minpack-supplied ... r1mach
c       fortran-supplied ... abs,sqrt
c
c     minpack. version of december 1978.
c     burton s. garbow, kenneth e. hillstrom, jorge j. more,
c     john l. nazareth
c     **********
c***routines called  r1mach
c***end prologue  r1updt
      integer m,n,ls
      logical sing
      double precision s(ls),u(m),v(n),w(m)
      integer i,j,jj,l,nmj,nm1
      double precision cos,cotan,giant,one,p5,p25
      double precision sin,tan,tau,temp,zero
      double precision r1mach
      data one,p5,p25,zero /1.0d0,5.0d-1,2.5d-1,0.0d0/
c***first executable statement  r1updt
      giant = r1mach(2)
c
c     initialize the diagonal element pointer.
c
      jj = (n*(2*m - n + 1))/2 - (m - n)
c
c     move the nontrivial part of the last column of s into w.
c
      l = jj
      do 10 i = n, m
         w(i) = s(l)
         l = l + 1
   10    continue
c
c     rotate the vector v into a multiple of the n-th unit vector
c     in such a way that a spike is introduced into w.
c
      nm1 = n - 1
      if (nm1 .lt. 1) go to 70
      do 60 nmj = 1, nm1
         j = n - nmj
         jj = jj - (m - j + 1)
         w(j) = zero
         if (v(j) .eq. zero) go to 50
c
c        determine a givens rotation which eliminates the
c        j-th element of v.
c
         if (abs(v(n)) .ge. abs(v(j))) go to 20
            cotan = v(n)/v(j)
            sin = p5/sqrt(p25+p25*cotan**2)
            cos = sin*cotan
            tau = one
            if (abs(cos)*giant .gt. one) tau = one/cos
            go to 30
   20    continue
            tan = v(j)/v(n)
            cos = p5/sqrt(p25+p25*tan**2)
            sin = cos*tan
            tau = sin
   30    continue
c
c        apply the transformation to v and store the information
c        necessary to recover the givens rotation.
c
         v(n) = sin*v(j) + cos*v(n)
         v(j) = tau
c
c        apply the transformation to s and extend the spike in w.
c
         l = jj
         do 40 i = j, m
            temp = cos*s(l) - sin*w(i)
            w(i) = sin*s(l) + cos*w(i)
            s(l) = temp
            l = l + 1
   40       continue
```

```
   50    continue
   60    continue
   70 continue
c
c     add the spike from the rank 1 update to w.
c
      do 80 i = 1, m
         w(i) = w(i) + v(n)*u(i)
   80    continue
c
c     eliminate the spike.
c
      sing = .false.
      if (nm1 .lt. 1) go to 140
      do 130 j = 1, nm1
         if (w(j) .eq. zero) go to 120
c
c        determine a givens rotation which eliminates the
c        j-th element of the spike.
c
         if (abs(s(jj)) .ge. abs(w(j))) go to 90
            cotan = s(jj)/w(j)
            sin = p5/sqrt(p25+p25*cotan**2)
            cos = sin*cotan
            tau = one
            if (abs(cos)*giant .gt. one) tau = one/cos
            go to 100
   90    continue
            tan = w(j)/s(jj)
            cos = p5/sqrt(p25+p25*tan**2)
            sin = cos*tan
            tau = sin
  100    continue
c
c        apply the transformation to s and reduce the spike in w.
c
         l = jj
         do 110 i = j, m
            temp = cos*s(l) + sin*w(i)
            w(i) = -sin*s(l) + cos*w(i)
            s(l) = temp
            l = l + 1
  110       continue
c
c        store the information necessary to recover the
c        givens rotation.
c
         w(j) = tau
  120    continue
c
c        test for zero diagonal elements in the output s.
c
         if (s(jj) .eq. zero) sing = .true.
         jj = jj + (m - j + 1)
  130    continue
  140 continue
c
c     move w back into the last column of the output s.
c
      l = jj
      do 150 i = n, m
         s(l) = w(i)
         l = l + 1
  150    continue
      if (s(jj) .eq. zero) sing = .true.
      return
c
c     last card of subroutine r1updt.
c
      end


c*****************************************************
c
      subroutine rwupdt(n,r,ldr,w,b,alpha,cos,sin)
c***begin prologue  rwupdt
c***refer to  snls1,snls1e
c
c     **********
c     subroutine rwupdt
c
c     given an n by n upper triangular matrix r, this subroutine
c     computes the qr decomposition of the matrix formed when a row
c     is added to r. if the row is specified by the vector w, then
c     rwupdt determines an orthogonal matrix q such that when the
c     n+1 by n matrix composed of r augmented by w is premultiplied
c     by (q transpose), the resulting matrix is upper trapezoidal.
c     the orthogonal matrix q is the product of n transformations
c
c           g(1)*g(2)* ... *g(n)
c
c     where g(i) is a givens rotation in the (i,n+1) plane which
c     eliminates elements in the i-th plane. rwupdt also
c     computes the product (q transpose)*c where c is the
c     (n+1)-vector (b,alpha). q itself is not accumulated, rather
c     the information to recover the g rotations is supplied.
c
c     the subroutine statement is
c
c       subroutine rwupdt(n,r,ldr,w,b,alpha,cos,sin)
c
c     where
c
c       n is a positive integer input variable set to the order of r.
c
c       r is an n by n array. on input the upper triangular part of
c         r must contain the matrix to be updated. on output r
c         contains the updated triangular matrix.
c
c       ldr is a positive integer input variable not less than n
c         which specifies the leading dimension of the array r.
c
c       w is an input array of length n which must contain the row
c         vector to be added to r.
c
c       b is an array of length n. on input b must contain the
c         first n elements of the vector c. on output b contains
c         the first n elements of the vector (q transpose)*c.
c
c       alpha is a variable. on input alpha must contain the
c         (n+1)-st element of the vector c. on output alpha contains
c         the (n+1)-st element of the vector (q transpose)*c.
c
c       cos is an output array of length n which contains the
c         cosines of the transforming givens rotations.
c
c       sin is an output array of length n which contains the
c         sines of the transforming givens rotations.
c
c     subprograms called
```

```
c
c       fortran-supplied ... abs,sqrt
c
c     minpack. version of december 1978.
c     burton s. garbow, dudley v. goetschel, kenneth e. hillstrom,
c     jorge j. more
c     **********
c***routines called  (none)
c***end prologue  rwupdt
      implicit none
      integer n,ldr
      double precision alpha
      double precision r(ldr,n),w(n),b(n),cos(n),sin(n)
      integer i,j,jm1
      double precision cotan,one,p5,p25,rowj,tan,temp,zero
      data one,p5,p25,zero /1.0d0,5.0d-1,2.5d-1,0.0d0/
c***first executable statement  rwupdt
      do 60 j = 1, n
         rowj = w(j)
         jm1 = j - 1
c
c        apply the previous transformations to
c        r(i,j), i=1,2,...,j-1, and to w(j).
c
         if (jm1 .lt. 1) go to 20
         do 10 i = 1, jm1
            temp = cos(i)*r(i,j) + sin(i)*rowj
            rowj = -sin(i)*r(i,j) + cos(i)*rowj
            r(i,j) = temp
   10       continue
   20    continue
c
c        determine a givens rotation which eliminates w(j).
c
         cos(j) = one
         sin(j) = zero
         if (rowj .eq. zero) go to 50
         if (abs(r(j,j)) .ge. abs(rowj)) go to 30
            cotan = r(j,j)/rowj
            sin(j) = p5/sqrt(p25+p25*cotan**2)
            cos(j) = sin(j)*cotan
            go to 40
   30    continue
            tan = rowj/r(j,j)
            cos(j) = p5/sqrt(p25+p25*tan**2)
            sin(j) = cos(j)*tan
   40    continue
c
c        apply the current transformation to r(j,j), b(j), and alpha.
c
         r(j,j) = cos(j)*r(j,j) + sin(j)*rowj
         temp = cos(j)*b(j) + sin(j)*alpha
         alpha = -sin(j)*b(j) + cos(j)*alpha
         b(j) = temp
   50    continue
   60    continue
      return
c
c     last card of subroutine rwupdt.
c
      end

c***********************************************************************
c
      subroutine snls1(fcn,iopt,m,n,x,fvec,fjac,ldfjac,ftol,xtol,gtol,
     1   maxfev,epsfcn,diag,mode,factor,nprint,info,nfev,njev,ipvt,qtf,
     2   wa1,wa2,wa3,wa4)
c***begin prologue  snls1
c***date written   800301   (yymmdd)
c***revision date  840405   (yymmdd)
c***category no.  k1b1a1,k1b1a2
c***keywords  levenberg-marquardt,nonlinear data fitting,
c             nonlinear least squares
c***author  hiebert, k. l., (snla)
c***purpose  snls1 minimizes the sum of the squares of m nonlinear
c            functions in n variables by a modification of the
c            levenberg-marquardt algorithm.  this code is a combination
c            of the minpack codes (argonne) lmder, lmdif, and lmstr.
c***description
c
c 1. purpose.
c
c       the purpose of snls1 is to minimize the sum of the squares of m
c       nonlinear functions in n variables by a modification of the
c       levenberg-marquardt algorithm.  the user must provide a subrou-
c       tine which calculates the functions.  the user has the option
c       of how the jacobian will be supplied.  the user can supply the
c       full jacobian, or the rows of the jacobian (to avoid storing
c       the full jacobian), or let the code approximate the jacobian by
c       forward-differencing.   this code is the combination of the
c       minpack codes (argonne) lmder, lmdif, and lmstr.
c
c
c 2. subroutine and type statements.
c
c       subroutine snls1(fcn,iopt,m,n,x,fvec,fjac,ldfjac,ftol,xtol,
c      *                 gtol,maxfev,epsfcn,diag,mode,factor,nprint,info
c      *                 ,nfev,njev,ipvt,qtf,wa1,wa2,wa3,wa4)
c       integer iopt,m,n,ldfjac,maxfev,mode,nprint,info,nfev,njev
c       integer ipvt(n)
c       real ftol,xtol,gtol,epsfcn,factor
c       real x(n),fvec(m),fjac(ldfjac,n),diag(n),qtf(n),
c      *     wa1(n),wa2(n),wa3(n),wa4(m)
c
c
c 3. parameters.
c
c       parameters designated as input parameters must be specified on
c       entry to snls1 and are not changed on exit, while parameters
c       designated as output parameters need not be specified on entry
c       and are set to appropriate values on exit from snls1.
c
c       fcn is the name of the user-supplied subroutine which calculates
c         the functions.  if the user wants to supply the jacobian
c         (iopt=2 or 3), then fcn must be written to calculate the
c         jacobian, as well as the functions.  see the explanation
c         of the iopt argument below.
c         if the user wants the iterates printed (nprint positive), then
c         fcn must do the printing.  see the explanation of nprint
c         below.  fcn must be declared in an external statement in the
c         calling program and should be written as follows.
c
c
c         subroutine fcn(iflag,m,n,x,fvec,fjac,ldfjac)
c         integer iflag,ldfjac,m,n
c         real x(n),fvec(m)
c         ----------
c         fjac and ldfjac may be ignored     , if iopt=1.
c         real fjac(ldfjac,n)                , if iopt=2.
c         real fjac(n)                       , if iopt=3.
```

```
c         ----------
c           if iflag=0, the values in x and fvec are available
c           for printing.  see the explanation of nprint below.
c           iflag will never be zero unless nprint is positive.
c           the values of x and fvec must not be changed.
c         return
c         ----------
c           if iflag=1, calculate the functions at x and return
c           this vector in fvec.
c         return
c         ----------
c           if iflag=2, calculate the full jacobian at x and return
c           this matrix in fjac.  note that iflag will never be 2 unless
c           iopt=2.  fvec contains the function values at x and must
c           not be altered.  fjac(i,j) must be set to the derivative
c           of fvec(i) with respect to x(j).
c         return
c         ----------
c           if iflag=3, calculate the ldfjac-th row of the jacobian
c           and return this vector in fjac.  note that iflag will
c           never be 3 unless iopt=3.  fvec contains the function
c           values at x and must not be altered.  fjac(j) must be
c           set to the derivative of fvec(ldfjac) with respect to x(j).
c         return
c         ----------
c         end
c
c
c         the value of iflag should not be changed by fcn unless the
c         user wants to terminate execution of snls1.  in this case, set
c         iflag to a negative integer.
c
c
c       iopt is an input variable which specifies how the jacobian will
c         be calculated.  if iopt=2 or 3, then the user must supply the
c         jacobian, as well as the function values, through the
c         subroutine fcn.  if iopt=2, the user supplies the full
c         jacobian with one call to fcn.  if iopt=3, the user supplies
c         one row of the jacobian with each call.  (in this manner,
c         storage can be saved because the full jacobian is not stored.)
c         if iopt=1, the code will approximate the jacobian by forward
c         differencing.
c
c       m is a positive integer input variable set to the number of
c         functions.
c
c       n is a positive integer input variable set to the number of
c         variables.  n must not exceed m.
c
c       x is an array of length n.  on input, x must contain an initial
c         estimate of the solution vector.  on output, x contains the
c         final estimate of the solution vector.
c
c       fvec is an output array of length m which contains the functions
c         evaluated at the output x.
c
c       fjac is an output array.  for iopt=1 and 2, fjac is an m by n
c         array.  for iopt=3, fjac is an n by n array.  the upper n by n
c         submatrix of fjac contains an upper triangular matrix r with
c         diagonal elements of nonincreasing magnitude such that
c
c                t     t           t
c               p *(jac *jac)*p = r *r,
c
c         where p is a permutation matrix and jac is the final calcu-
c         lated jacobian.  column j of p is column ipvt(j) (see below)
c         of the identity matrix.  the lower part of fjac contains
c         information generated during the computation of r.
c
c       ldfjac is a positive integer input variable which specifies
c         the leading dimension of the array fjac.  for iopt=1 and 2,
c         ldfjac must not be less than m.  for iopt=3, ldfjac must not
c         be less than n.
c
c       ftol is a non-negative input variable.  termination occurs when
c         both the actual and predicted relative reductions in the sum
c         of squares are at most ftol.  therefore, ftol measures the
c         relative error desired in the sum of squares.  section 4 con-
c         tains more details about ftol.
c
c       xtol is a non-negative input variable.  termination occurs when
c         the relative error between two consecutive iterates is at most
c         xtol.  therefore, xtol measures the relative error desired in
c         the approximate solution.  section 4 contains more details
c         about xtol.
c
c       gtol is a non-negative input variable.  termination occurs when
c         the cosine of the angle between fvec and any column of the
c         jacobian is at most gtol in absolute value.  therefore, gtol
c         measures the orthogonality desired between the function vector
c         and the columns of the jacobian.  section 4 contains more
c         details about gtol.
c
c       maxfev is a positive integer input variable.  termination occurs
c         when the number of calls to fcn to evaluate the functions
c         has reached maxfev.
c
c       epsfcn is an input variable used in determining a suitable step
c         for the forward-difference approximation.  this approximation
c         assumes that the relative errors in the functions are of the
c         order of epsfcn.  if epsfcn is less than the machine preci-
c         sion, it is assumed that the relative errors in the functions
c         are of the order of the machine precision.  if iopt=2 or 3,
c         then epsfcn can be ignored (treat it as a dummy argument).
c
c       diag is an array of length n.  if mode = 1 (see below), diag is
c         internally set.  if mode = 2, diag must contain positive
c         entries that serve as implicit (multiplicative) scale factors
c         for the variables.
c
c       mode is an integer input variable.  if mode = 1, the variables
c         will be scaled internally.  if mode = 2, the scaling is speci-
c         fied by the input diag.  other values of mode are equivalent
c         to mode = 1.
c
c       factor is a positive input variable used in determining the ini-
c         tial step bound.  this bound is set to the product of factor
c         and the euclidean norm of diag*x if nonzero, or else to factor
c         itself.  in most cases factor should lie in the interval
c         (.1,100.).  100. is a generally recommended value.
c
c       nprint is an integer input variable that enables controlled
c         printing of iterates if it is positive.  in this case, fcn is
c         called with iflag = 0 at the beginning of the first iteration
c         and every nprint iterations thereafter and immediately prior
c         to return, with x and fvec available for printing. appropriate
c         print statements must be added to fcn (see example) and
c         fvec should not be altered.  if nprint is not positive, no
c         special calls to fcn with iflag = 0 are made.
c
```

```
c       info is an integer output variable.  if the user has terminated
c         execution, info is set to the (negative) value of iflag.  see
c         description of fcn and jac. otherwise, info is set as follows.
c
c         info = 0  improper input parameters.
c
c         info = 1  both actual and predicted relative reductions in the
c                   sum of squares are at most ftol.
c
c         info = 2  relative error between two consecutive iterates is
c                   at most xtol.
c
c         info = 3  conditions for info = 1 and info = 2 both hold.
c
c         info = 4  the cosine of the angle between fvec and any column
c                   of the jacobian is at most gtol in absolute value.
c
c         info = 5  number of calls to fcn for function evaluation
c                   has reached maxfev.
c
c         info = 6  ftol is too small.  no further reduction in the sum
c                   of squares is possible.
c
c         info = 7  xtol is too small.  no further improvement in the
c                   approximate solution x is possible.
c
c         info = 8  gtol is too small.  fvec is orthogonal to the
c                   columns of the jacobian to machine precision.
c
c         sections 4 and 5 contain more details about info.
c
c       nfev is an integer output variable set to the number of calls to
c         fcn for function evaluation.
c
c       njev is an integer output variable set to the number of
c         evaluations of the full jacobian.  if iopt=2, only one call to
c         fcn is required for each evaluation of the full jacobian.
c         if iopt=3, the m calls to fcn are required.
c         if iopt=1, then njev is set to zero.
c
c       ipvt is an integer output array of length n.  ipvt defines a
c         permutation matrix p such that jac*p = q*r, where jac is the
c         final calculated jacobian, q is orthogonal (not stored), and r
c         is upper triangular with diagonal elements of nonincreasing
c         magnitude.  column j of p is column ipvt(j) of the identity
c         matrix.
c
c       qtf is an output array of length n which contains the first n
c         elements of the vector (q transpose)*fvec.
c
c       wa1, wa2, and wa3 are work arrays of length n.
c
c       wa4 is a work array of length m.
c
c
c 4. successful completion.
c
c       the accuracy of snls1 is controlled by the convergence parame-
c       ters ftol, xtol, and gtol.  these parameters are used in tests
c       which make three types of comparisons between the approximation
c       x and a solution xsol.  snls1 terminates when any of the tests
c       is satisfied.  if any of the convergence parameters is less than
c       the machine precision (as defined by the function r1mach(4)),
c       then snls1 only attempts to satisfy the test defined by the
c       machine precision.  further progress is not usually possible.
c
c       the tests assume that the functions are reasonably well behaved,
c       and, if the jacobian is supplied by the user, that the functions
c       and the jacobian are coded consistently.  if these conditions
c       are not satisfied, then snls1 may incorrectly indicate conver-
c       gence.  if the jacobian is coded correctly or iopt=1,
c       then the validity of the answer can be checked, for example, by
c       rerunning snls1 with tighter tolerances.
c
c       first convergence test.  if enorm(z) denotes the euclidean norm
c         of a vector z, then this test attempts to guarantee that
c
c               enorm(fvec) .le. (1+ftol)*enorm(fvecs),
c
c         where fvecs denotes the functions evaluated at xsol.  if this
c         condition is satisfied with ftol = 10**(-k), then the final
c         residual norm enorm(fvec) has k significant decimal digits and
c         info is set to 1 (or to 3 if the second test is also satis-
c         fied).  unless high precision solutions are required, the
c         recommended value for ftol is the square root of the machine
c         precision.
c
c       second convergence test.  if d is the diagonal matrix whose
c         entries are defined by the array diag, then this test attempts
c         to guarantee that
c
c               enorm(d*(x-xsol)) .le. xtol*enorm(d*xsol).
c
c         if this condition is satisfied with xtol = 10**(-k), then the
c         larger components of d*x have k significant decimal digits and
c         info is set to 2 (or to 3 if the first test is also satis-
c         fied).  there is a danger that the smaller components of d*x
c         may have large relative errors, but if mode = 1, then the
c         accuracy of the components of x is usually related to their
c         sensitivity.  unless high precision solutions are required,
c         the recommended value for xtol is the square root of the
c         machine precision.
c
c       third convergence test.  this test is satisfied when the cosine
c         of the angle between fvec and any column of the jacobian at x
c         is at most gtol in absolute value.  there is no clear rela-
c         tionship between this test and the accuracy of snls1, and
c         furthermore, the test is equally well satisfied at other crit-
c         ical points, namely maximizers and saddle points.  therefore,
c         termination caused by this test (info = 4) should be examined
c         carefully.  the recommended value for gtol is zero.
c
c
c 5. unsuccessful completion.
c
c       unsuccessful termination of snls1 can be due to improper input
c       parameters, arithmetic interrupts, or an excessive number of
c       function evaluations.
c
c       improper input parameters.  info is set to 0 if iopt .lt. 1
c         or iopt .gt. 3, or n .le. 0, or m .lt. n, or for iopt=1 or 2
c         ldfjac .lt. m, or for iopt=3 ldfjac .lt. n, or ftol .lt. 0.e0,
c         or xtol .lt. 0.e0, or gtol .lt. 0.e0, or maxfev .le. 0, or
c         factor .le. 0.e0.
c
c       arithmetic interrupts.  if these interrupts occur in the fcn
c         subroutine during an early stage of the computation, they may
c         be caused by an unacceptable choice of x by snls1.  in this
c         case, it may be possible to remedy the situation by rerunning
c         snls1 with a smaller value of factor.
```

```
c
c       excessive number of function evaluations.  a reasonable value
c         for maxfev is 100*(n+1) for iopt=2 or 3 and 200*(n+1) for
c         iopt=1.  if the number of calls to fcn reaches maxfev, then
c         this indicates that the routine is converging very slowly
c         as measured by the progress of fvec, and info is set to 5.
c         in this case, it may be helpful to restart snls1 with mode
c         set to 1.
c
c
c 6. characteristics of the algorithm.
c
c       snls1 is a modification of the levenberg-marquardt algorithm.
c       two of its main characteristics involve the proper use of
c       implicitly scaled variables (if mode = 1) and an optimal choice
c       for the correction.  the use of implicitly scaled variables
c       achieves scale invariance of snls1 and limits the size of the
c       correction in any direction where the functions are changing
c       rapidly.  the optimal choice of the correction guarantees (under
c       reasonable conditions) global convergence from starting points
c       far from the solution and a fast rate of convergence for
c       problems with small residuals.
c
c       timing.  the time required by snls1 to solve a given problem
c         depends on m and n, the behavior of the functions, the accu-
c         racy requested, and the starting point.  the number of arith-
c         metic operations needed by snls1 is about n**3 to process each
c         evaluation of the functions (call to fcn) and to process each
c         evaluation of the jacobian it takes m*n**2 for iopt=2 (one
c         call to fcn), m*n**2 for iopt=1 (n calls to fcn) and
c         1.5*m*n**2 for iopt=3 (m calls to fcn).  unless fcn
c         can be evaluated quickly, the timing of snls1 will be
c         strongly influenced by the time spent in fcn.
c
c       storage.  snls1 requires (m*n + 2*m + 6*n) for iopt=1 or 2 and
c         (n**2 + 2*m + 6*n) for iopt=3 single precision storage
c         locations and n integer storage locations, in addition to
c         the storage required by the program.  there are no internally
c         declared storage arrays.
c
c    ***see the long description for an example problem***
c***long description
c
c 7. example.
c
c       the problem is to determine the values of x(1), x(2), and x(3)
c       which provide the best fit (in the least squares sense) of
c
c             x(1) + u(i)/(v(i)*x(2) + w(i)*x(3)),  i = 1, 15
c
c       to the data
c
c             y = (0.14,0.18,0.22,0.25,0.29,0.32,0.35,0.39,
c                  0.37,0.58,0.73,0.96,1.34,2.10,4.39),
c
c       where u(i) = i, v(i) = 16 - i, and w(i) = min(u(i),v(i)).  the
c       i-th component of fvec is thus defined by
c
c             y(i) - (x(1) + u(i)/(v(i)*x(2) + w(i)*x(3))).
c
c       **********
c
c       program test(input,output,tape6=output)
c c
c c     driver for snls1 example.
c c
c       integer j,iopt,m,n,ldfjac,maxfev,mode,nprint,info,nfev,njev,
c      *        nwrite
c       integer ipvt(3)
c       real ftol,xtol,gtol,factor,fnorm,epsfcn
c       real x(3),fvec(15),fjac(15,3),diag(3),qtf(3),
c      *     wa1(3),wa2(3),wa3(3),wa4(15)
c       real enorm,r1mach
c       external fcn
c       data nwrite /6/
c c
c       iopt = 1
c       m = 15
c       n = 3
c c
c c     the following starting values provide a rough fit.
c c
c       x(1) = 1.e0
c       x(2) = 1.e0
c       x(3) = 1.e0
c c
c       ldfjac = 15
c c
c c     set ftol and xtol to the square root of the machine precision
c c     and gtol to zero.  unless high precision solutions are
c c     required, these are the recommended settings.
c c
c       ftol = sqrt(r1mach(4))
c       xtol = sqrt(r1mach(4))
c       gtol = 0.e0
c c
c       maxfev = 400
c       epsfcn = 0.0
c       mode = 1
c       factor = 1.e2
c       nprint = 0
c c
c       call snls1(fcn,iopt,m,n,x,fvec,fjac,ldfjac,ftol,xtol,
c      *           gtol,maxfev,epsfcn,diag,mode,factor,nprint,
c      *           info,nfev,njev,ipvt,qtf,wa1,wa2,wa3,wa4)
c       fnorm = enorm(m,fvec)
c       write (nwrite,1000) fnorm,nfev,njev,info,(x(j),j=1,n)
c       stop
c  1000 format (5x,' final l2 norm of the residuals',e15.7 //
c      *        5x,' number of function evaluations',i10 //
c      *        5x,' number of jacobian evaluations',i10 //
c      *        5x,' exit parameter',16x,i10 //
c      *        5x,' final approximate solution' // 5x,3e15.7)
c       end
c       subroutine fcn(iflag,m,n,x,fvec,dum,idum)
c c     this is the form of the fcn routine if iopt=1,
c c     that is, if the user does not calculate the jacobian.
c       integer m,n,iflag
c       real x(n),fvec(m)
c       integer i
c       real tmp1,tmp2,tmp3,tmp4
c       real y(15)
c       data y(1),y(2),y(3),y(4),y(5),y(6),y(7),y(8),
c      *     y(9),y(10),y(11),y(12),y(13),y(14),y(15)
c      *     /1.4e-1,1.8e-1,2.2e-1,2.5e-1,2.9e-1,3.2e-1,3.5e-1,3.9e-1,
c      *      3.7e-1,5.8e-1,7.3e-1,9.6e-1,1.34e0,2.1e0,4.39e0/
c c
c       if (iflag .ne. 0) go to 5
c c
c c     insert print statements here when nprint is positive.
```

```
c c
c       return
c     5 continue
c       do 10 i = 1, m
c          tmp1 = i
c          tmp2 = 16 - i
c          tmp3 = tmp1
c          if (i .gt. 8) tmp3 = tmp2
c          fvec(i) = y(i) - (x(1) + tmp1/(x(2)*tmp2 + x(3)*tmp3))
c    10    continue
c       return
c       end
c
c
c       results obtained with different compilers or machines
c       may be slightly different.
c
c       final l2 norm of the residuals  0.9063596e-01
c
c       number of function evaluations        25
c
c       number of jacobian evaluations         0
c
c       exit parameter                         1
c
c       final approximate solution
c
c        0.8241058e-01  0.1133037e+01  0.2343695e+01
c
c
c       for iopt=2, fcn would be modified as follows to also
c       calculate the full jacobian when iflag=2.
c
c       subroutine fcn(iflag,m,n,x,fvec,fjac,ldfjac)
c c
c c     this is the form of the fcn routine if iopt=2,
c c     that is, if the user calculates the full jacobian.
c c
c       integer ldfjac,m,n,iflag
c       real x(n),fvec(m)
c       real fjac(ldfjac,n)
c       integer i
c       real tmp1,tmp2,tmp3,tmp4
c       real y(15)
c       data y(1),y(2),y(3),y(4),y(5),y(6),y(7),y(8),
c      *     y(9),y(10),y(11),y(12),y(13),y(14),y(15)
c      *     /1.4e-1,1.8e-1,2.2e-1,2.5e-1,2.9e-1,3.2e-1,3.5e-1,3.9e-1,
c      *      3.7e-1,5.8e-1,7.3e-1,9.6e-1,1.34e0,2.1e0,4.39e0/
c c
c       if (iflag .ne. 0) go to 5
c c
c c     insert print statements here when nprint is positive.
c c
c       return
c     5 continue
c       if(iflag.ne.1) go to 20
c       do 10 i = 1, m
c          tmp1 = i
c          tmp2 = 16 - i
c          tmp3 = tmp1
c          if (i .gt. 8) tmp3 = tmp2
c          fvec(i) = y(i) - (x(1) + tmp1/(x(2)*tmp2 + x(3)*tmp3))
c    10    continue
c       return
c c
```

```
c c     below, calculate the full jacobian.
c c
c    20    continue
c c
c       do 30 i = 1, m
c          tmp1 = i
c          tmp2 = 16 - i
c          tmp3 = tmp1
c          if (i .gt. 8) tmp3 = tmp2
c          tmp4 = (x(2)*tmp2 + x(3)*tmp3)**2
c          fjac(i,1) = -1.e0
c          fjac(i,2) = tmp1*tmp2/tmp4
c          fjac(i,3) = tmp1*tmp3/tmp4
c    30    continue
c       return
c       end
c
c
c       for iopt = 3, fjac would be dimensioned as fjac(3,3),
c         ldfjac would be set to 3, and fcn would be written as
c         follows to calculate a row of the jacobian when iflag=3.
c
c       subroutine fcn(iflag,m,n,x,fvec,fjac,ldfjac)
c c     this is the form of the fcn routine if iopt=3,
c c     that is, if the user calculates the jacobian row by row.
c       integer m,n,iflag
c       real x(n),fvec(m)
c       real fjac(n)
c       integer i
c       real tmp1,tmp2,tmp3,tmp4
c       real y(15)
c       data y(1),y(2),y(3),y(4),y(5),y(6),y(7),y(8),
c      *     y(9),y(10),y(11),y(12),y(13),y(14),y(15)
c      *     /1.4e-1,1.8e-1,2.2e-1,2.5e-1,2.9e-1,3.2e-1,3.5e-1,3.9e-1,
c      *      3.7e-1,5.8e-1,7.3e-1,9.6e-1,1.34e0,2.1e0,4.39e0/
c c
c       if (iflag .ne. 0) go to 5
c c
c c     insert print statements here when nprint is positive.
c c
c       return
c     5 continue
c       if( iflag.ne.1) go to 20
c       do 10 i = 1, m
c          tmp1 = i
c          tmp2 = 16 - i
c          tmp3 = tmp1
c          if (i .gt. 8) tmp3 = tmp2
c          fvec(i) = y(i) - (x(1) + tmp1/(x(2)*tmp2 + x(3)*tmp3))
c    10    continue
c       return
c c
c c     below, calculate the ldfjac-th row of the jacobian.
c c
c    20 continue
c
c       i = ldfjac
c          tmp1 = i
c          tmp2 = 16 - i
c          tmp3 = tmp1
c          if (i .gt. 8) tmp3 = tmp2
c          tmp4 = (x(2)*tmp2 + x(3)*tmp3)**2
c          fjac(1) = -1.e0
c          fjac(2) = tmp1*tmp2/tmp4
c          fjac(3) = tmp1*tmp3/tmp4
```

```
c       return
c       end
c***references  more, jorge j.
c                 the levenberg-marquardt algorithm, implementation and
c                 theory.  numerical analysis, g. a. watson, editor.
c                 lecture notes in mathematics 630, springer-verlag,
c                 1977.
c***routines called  chkder,enorm,fdjac3,lmpar,qrfac,r1mach,rwupdt,
c                    xerror,xerrwv
c***end prologue  snls1
      implicit none
      integer iopt,m,n,ldfjac,maxfev,mode,nprint,info,nfev,njev
      integer ijunk,nrow,ipvt(n)
      double precision ftol,xtol,gtol,factor,epsfcn
      double precision x(n),fvec(m),fjac(ldfjac,n),diag(n),qtf(n)
      double precision wa1(n),wa2(n),wa3(n),wa4(m)
      logical sing
      external fcn
      integer i,iflag,iter,j,l,modech
      double precision actred,delta,dirder,epsmch,fnorm,fnorm1,gnorm,
     1     one,par,pnorm,prered,p1,p5,p25,p75,p0001,ratio,
     2     sum,temp,temp1,temp2,xnorm,zero
      double precision r1mach,enorm,err,chklim
      data chklim/.1d0/
      data one,p1,p5,p25,p75,p0001,zero
     1     /1.0d0,1.0d-1,5.0d-1,2.5d-1,7.5d-1,1.0d-4,0.0d0/
c
c***first executable statement  snls1
      epsmch = r1mach(4)
c
      info = 0
      iflag = 0
      nfev = 0
      njev = 0
c
c     check the input parameters for errors.
c
      if (iopt .lt. 1 .or. iopt .gt. 3 .or. n .le. 0 .or.
     1    m .lt. n .or. ldfjac .lt. n .or. ftol .lt. zero
     2    .or. xtol .lt. zero .or. gtol .lt. zero
     3    .or. maxfev .le. 0 .or. factor .le. zero) go to 300
      if (iopt .lt. 3 .and. ldfjac .lt. m) go to 300
      if (mode .ne. 2) go to 20
      do 10 j = 1, n
         if (diag(j) .le. zero) go to 300
   10    continue
   20 continue
c
c     evaluate the function at the starting point
c     and calculate its norm.
c
      iflag = 1
      ijunk = 1
      call fcn(iflag,m,n,x,fvec,fjac,ijunk)
      nfev = 1
      if (iflag .lt. 0) go to 300
      fnorm = enorm(m,fvec)
c
c     initialize levenberg-marquardt parameter and iteration counter.
c
      par = zero
      iter = 1
c
c     beginning of the outer loop.
c
   30 continue
c
c        if requested, call fcn to enable printing of iterates.
c
         if (nprint .le. 0) go to 40
         iflag = 0
         if (mod(iter-1,nprint) .eq. 0)
     1      call fcn(iflag,m,n,x,fvec,fjac,ijunk)
         if (iflag .lt. 0) go to 300
   40    continue
c
c        calculate the jacobian matrix.
c
      if (iopt .eq. 3) go to 475
c
c     store the full jacobian using m*n storage
c
      if (iopt .eq. 1) go to 410
c
c     the user supplies the jacobian
c
         iflag = 2
         call fcn(iflag,m,n,x,fvec,fjac,ldfjac)
         njev = njev + 1
c
c             on the first iteration, check the user supplied jacobian
c
         if(iter .gt. 1) go to 355
         if(iflag .lt. 0) go to 300
c
c             get the incremented x-values into wa1(*).
c
         modech = 1
         call chkder(m,n,x,fvec,fjac,ldfjac,wa1,wa4,modech,err)
c
c             evaluate function at incremented value and put in wa4(*).
c
         iflag = 1
         call fcn(iflag,m,n,wa1,wa4,fjac,ldfjac)
         nfev = nfev + 1
         if(iflag .lt. 0) go to 300
         do 350 i = 1, m
           modech = 2
           call chkder(1,n,x,fvec(i),fjac(i,1),ldfjac,wa1,
     1          wa4(i),modech,err)
           if(err .ge. chklim) go to 350
           call xerrwv( ' snls1--derivative of function i1 may be wrong,
     1 err=r1 too close to 0.' ,70,7,0,1,i,0,1,err,err)
350      continue
355      continue
c
         go to 420
c
c     the code approximates the jacobian
c
410      iflag = 1
         call fdjac3(fcn,m,n,x,fvec,fjac,ldfjac,iflag,epsfcn,wa4)
         nfev = nfev + n
  420    if (iflag .lt. 0) go to 300
c
c        compute the qr factorization of the jacobian.
c
         call qrfac(m,n,fjac,ldfjac,.true.,ipvt,n,wa1,wa2,wa3)
c
c        form (q transpose)*fvec and store the first n components in
```

```
c        qtf.
c
         do 430 i = 1, m
            wa4(i) = fvec(i)
  430         continue
         do 470 j = 1, n
            if (fjac(j,j) .eq. zero) go to 460
            sum = zero
            do 440 i = j, m
               sum = sum + fjac(i,j)*wa4(i)
  440          continue
            temp = -sum/fjac(j,j)
            do 450 i = j, m
               wa4(i) = wa4(i) + fjac(i,j)*temp
  450          continue
  460       continue
            fjac(j,j) = wa1(j)
            qtf(j) = wa4(j)
  470       continue
         go to 560
c
c        accumulate the jacobian by rows in order to save storage.
c        compute the qr factorization of the jacobian matrix
c        calculated one row at a time, while simultaneously
c        forming (q transpose)*fvec and storing the first
c        n components in qtf.
c
  475    do 490 j = 1, n
            qtf(j) = zero
            do 480 i = 1, n
               fjac(i,j) = zero
  480          continue
  490        continue
         do 500 i = 1, m
            nrow = i
            iflag = 3
            call fcn(iflag,m,n,x,fvec,wa3,nrow)
            if (iflag .lt. 0) go to 300
c
c            on the first iteration, check the user supplied jacobian.
c
            if(iter .gt. 1) go to 498
c
c            get the incremented x-values into wa1(*).
c
            modech = 1
            call chkder(m,n,x,fvec,fjac,ldfjac,wa1,wa4,modech,err)
c
c            evaluate at incremented values, if not already evaluated.
c
            if(i .ne. 1) go to 495
c
c            evaluate function at incremented value and put into wa4(*).
c
            iflag = 1
            call fcn(iflag,m,n,wa1,wa4,fjac,nrow)
            nfev = nfev + 1
            if(iflag .lt. 0) go to 300
495         continue
            modech = 2
            call chkder(1,n,x,fvec(i),wa3,1,wa1,wa4(i),modech,err)
            if(err .ge. chklim) go to 498
            call xerrwv( ' snls1--derivative of function i1 may be wrong
     1, err=r1 too close to 0.' ,70,7,0,1,i,0,1,err,err)
498         continue
c
            temp = fvec(i)
            call rwupdt(n,fjac,ldfjac,wa3,qtf,temp,wa1,wa2)
  500       continue
         njev = njev + 1
c
c        if the jacobian is rank deficient, call qrfac to
c        reorder its columns and update the components of qtf.
c
         sing = .false.
         do 510 j = 1, n
            if (fjac(j,j) .eq. zero) sing = .true.
            ipvt(j) = j
            wa2(j) = enorm(j,fjac(1,j))
  510       continue
         if (.not.sing) go to 560
         call qrfac(n,n,fjac,ldfjac,.true.,ipvt,n,wa1,wa2,wa3)
         do 550 j = 1, n
            if (fjac(j,j) .eq. zero) go to 540
            sum = zero
            do 520 i = j, n
               sum = sum + fjac(i,j)*qtf(i)
  520         continue
            temp = -sum/fjac(j,j)
            do 530 i = j, n
               qtf(i) = qtf(i) + fjac(i,j)*temp
  530          continue
  540       continue
            fjac(j,j) = wa1(j)
  550       continue
  560    continue
c
c        on the first iteration and if mode is 1, scale according
c        to the norms of the columns of the initial jacobian.
c
         if (iter .ne. 1) go to 80
         if (mode .eq. 2) go to 60
         do 50 j = 1, n
            diag(j) = wa2(j)
            if (wa2(j) .eq. zero) diag(j) = one
   50       continue
   60    continue
c
c        on the first iteration, calculate the norm of the scaled x
c        and initialize the step bound delta.
c
         do 70 j = 1, n
            wa3(j) = diag(j)*x(j)
   70       continue
         xnorm = enorm(n,wa3)
         delta = factor*xnorm
         if (delta .eq. zero) delta = factor
   80    continue
c
c        compute the norm of the scaled gradient.
c
         gnorm = zero
         if (fnorm .eq. zero) go to 170
         do 160 j = 1, n
            l = ipvt(j)
            if (wa2(l) .eq. zero) go to 150
            sum = zero
            do 140 i = 1, j
               sum = sum + fjac(i,j)*(qtf(i)/fnorm)
  140          continue
```

```
            gnorm = max(gnorm,abs(sum/wa2(l)))
  150       continue
  160       continue
  170    continue
c
c        test for convergence of the gradient norm.
c
         if (gnorm .le. gtol) info = 4
         if (info .ne. 0) go to 300
c
c        rescale if necessary.
c
         if (mode .eq. 2) go to 190
         do 180 j = 1, n
            diag(j) = max(diag(j),wa2(j))
  180       continue
  190    continue
c
c        beginning of the inner loop.
c
  200    continue
c
c           determine the levenberg-marquardt parameter.
c
            call lmpar(n,fjac,ldfjac,ipvt,diag,qtf,delta,par,wa1,wa2,
     1                 wa3,wa4)
c
c           store the direction p and x + p. calculate the norm of p.
c
            do 210 j = 1, n
               wa1(j) = -wa1(j)
               wa2(j) = x(j) + wa1(j)
               wa3(j) = diag(j)*wa1(j)
  210          continue
            pnorm = enorm(n,wa3)
c
c           on the first iteration, adjust the initial step bound.
c
            if (iter .eq. 1) delta = min(delta,pnorm)
c
c           evaluate the function at x + p and calculate its norm.
c
            iflag = 1
            call fcn(iflag,m,n,wa2,wa4,fjac,ijunk)
            nfev = nfev + 1
            if (iflag .lt. 0) go to 300
            fnorm1 = enorm(m,wa4)
c
c           compute the scaled actual reduction.
c
            actred = -one
            if (p1*fnorm1 .lt. fnorm) actred = one - (fnorm1/fnorm)**2
c
c           compute the scaled predicted reduction and
c           the scaled directional derivative.
c
            do 230 j = 1, n
               wa3(j) = zero
               l = ipvt(j)
               temp = wa1(l)
               do 220 i = 1, j
                  wa3(i) = wa3(i) + fjac(i,j)*temp
  220             continue
  230          continue
            temp1 = enorm(n,wa3)/fnorm
            temp2 = (sqrt(par)*pnorm)/fnorm
            prered = temp1**2 + temp2**2/p5
            dirder = -(temp1**2 + temp2**2)
c
c           compute the ratio of the actual to the predicted
c           reduction.
c
            ratio = zero
            if (prered .ne. zero) ratio = actred/prered
c
c           update the step bound.
c
            if (ratio .gt. p25) go to 240
               if (actred .ge. zero) temp = p5
               if (actred .lt. zero)
     1            temp = p5*dirder/(dirder + p5*actred)
               if (p1*fnorm1 .ge. fnorm .or. temp .lt. p1) temp = p1
               delta = temp*min(delta,pnorm/p1)
               par = par/temp
               go to 260
  240       continue
               if (par .ne. zero .and. ratio .lt. p75) go to 250
               delta = pnorm/p5
               par = p5*par
  250          continue
  260       continue
c
c           test for successful iteration.
c
            if (ratio .lt. p0001) go to 290
c
c           successful iteration. update x, fvec, and their norms.
c
            do 270 j = 1, n
               x(j) = wa2(j)
               wa2(j) = diag(j)*x(j)
  270          continue
            do 280 i = 1, m
               fvec(i) = wa4(i)
  280          continue
            xnorm = enorm(n,wa2)
            fnorm = fnorm1
            iter = iter + 1
  290       continue
c
c           tests for convergence.
c
            if (abs(actred) .le. ftol .and. prered .le. ftol
     1          .and. p5*ratio .le. one) info = 1
            if (delta .le. xtol*xnorm) info = 2
            if (abs(actred) .le. ftol .and. prered .le. ftol
     1          .and. p5*ratio .le. one .and. info .eq. 2) info = 3
            if (info .ne. 0) go to 300
c
c           tests for termination and stringent tolerances.
c
            if (nfev .ge. maxfev) info = 5
            if (abs(actred) .le. epsmch .and. prered .le. epsmch
     1          .and. p5*ratio .le. one) info = 6
            if (delta .le. epsmch*xnorm) info = 7
            if (gnorm .le. epsmch) info = 8
            if (info .ne. 0) go to 300
c
c           end of the inner loop. repeat if iteration unsuccessful.
c
```

```
           if (ratio .lt. p0001) go to 200
c
c        end of the outer loop.
c
         go to 30
  300 continue
c
c     termination, either normal or user imposed.
c
      if (iflag .lt. 0) info = iflag
      iflag = 0
      if (nprint .gt. 0) call fcn(iflag,m,n,x,fvec,fjac,ijunk)
      if (info .lt. 0) call xerror( 'snls1  -- execution terminated beca
     1use user set iflag negative.',63,1,1)
      if (info .eq. 0) call xerror( 'snls1  -- invalid input parameter.'
     1,34,2,1)
      if (info .eq. 4) call xerror( 'snls1  -- third convergence conditi
     1on, check results before accepting.',70,1,1)
      if (info .eq. 5) call xerror( 'snls1  -- too many function evaluat
     1ions.',40,9,1)
      if (info .ge. 6) call xerror(  'snls1  -- tolerances too small, no
     1 further improvement possible.',64,3,1)
      return
c
c     last card of subroutine snls1.
c
      end

c***********************************************************************
c
      subroutine snsq(fcn,jac,iopt,n,x,fvec,fjac,ldfjac,xtol,maxfev,ml,
     1   mu,epsfcn,diag,mode,factor,nprint,info,nfev,njev,r,lr,qtf,wa1,
     2   wa2,wa3,wa4)
c***begin prologue  snsq
c***date written   800301   (yymmdd)
c***revision date  840405   (yymmdd)
c***category no.  f2a
c***keywords  nonlinear square system,powell hybrid method,zero
c***author  hiebert, k. l., (snla)
c***purpose  snsq finds to find a zero of a system of n nonlinear
c            functions in n variables by a modification of the powell
c            hybrid method.  this code is the combination of the minpack
c            codes (argonne) hybrd and hybrdj.
c***description
c
c 1. purpose.
c
c       the purpose of snsq is to find a zero of a system of n non-
c       linear functions in n variables by a modification of the powell
c       hybrid method.  the user must provide a subroutine which calcu-
c       lates the functions.  the user has the option of either to
c       provide a subroutine which calculates the jacobian or to let the
c       code calculate it by a forward-difference approximation.
c       this code is the combination of the minpack codes (argonne)
c       hybrd and hybrdj.
c
c
c 2. subroutine and type statements.
c
c       subroutine snsq(fcn,jac,iopt,n,x,fvec,fjac,ldfjac,xtol,maxfev,
c      *                 ml,mu,epsfcn,diag,mode,factor,nprint,info,nfev,
c      *                 njev,r,lr,qtf,wa1,wa2,wa3,wa4)
c       integer iopt,n,maxfev,ml,mu,mode,nprint,info,nfev,ldfjac,njev,lr
c       real xtol,epsfcn,factor
c       real x(n),fvec(n),diag(n),fjac(ldfjac,n),r(lr),qtf(n),
c      *     wa1(n),wa2(n),wa3(n),wa4(n)
c       external fcn,jac
c
c
c 3. parameters.
c
c       parameters designated as input parameters must be specified on
c       entry to snsq and are not changed on exit, while parameters
c       designated as output parameters need not be specified on entry
c       and are set to appropriate values on exit from snsq.
c
c       fcn is the name of the user-supplied subroutine which calculates
c         the functions.  fcn must be declared in an external statement
c         in the user calling program, and should be written as follows.
c
c         subroutine fcn(n,x,fvec,iflag)
c         integer n,iflag
c         real x(n),fvec(n)
c         ----------
c         calculate the functions at x and
c         return this vector in fvec.
c         ----------
c         return
c         end
c
c         the value of iflag should not be changed by fcn unless the
c         user wants to terminate execution of snsq.  in this case, set
c         iflag to a negative integer.
c
c       jac is the name of the user-supplied subroutine which calculates
c         the jacobian.  if iopt=1, then jac must be declared in an
c         external statement in the user calling program, and should be
c         written as follows.
c
c         subroutine jac(n,x,fvec,fjac,ldfjac,iflag)
c         integer n,ldfjac,iflag
c         real x(n),fvec(n),fjac(ldfjac,n)
c         ----------
c         calculate the jacobian at x and return this
c         matrix in fjac.  fvec contains the function
c         values at x and should not be altered.
c         ----------
c         return
c         end
c
c         the value of iflag should not be changed by jac unless the
c         user wants to terminate execution of snsq.  in this case, set
c         iflag to a negative integer.
c
c         if iopt=2, jac can be ignored (treat it as a dummy argument).
c
c       iopt is an input variable which specifies how the jacobian will
c         be calculated.  if iopt=1, then the user must supply the
c         jacobian through the subroutine jac.  if iopt=2, then the
c         code will approximate the jacobian by forward-differencing.
c
c       n is a positive integer input variable set to the number of
c         functions and variables.
c
c       x is an array of length n.  on input, x must contain an initial
c         estimate of the solution vector.  on output, x contains the
c         final estimate of the solution vector.
c
c       fvec is an output array of length n which contains the functions
c         evaluated at the output x.
```

```
c
c       fjac is an output n by n array which contains the orthogonal
c         matrix q produced by the qr factorization of the final approx-
c         imate jacobian.
c
c       ldfjac is a positive integer input variable not less than n
c         which specifies the leading dimension of the array fjac.
c
c       xtol is a non-negative input variable.  termination occurs when
c         the relative error between two consecutive iterates is at most
c         xtol.  therefore, xtol measures the relative error desired in
c         the approximate solution.  section 4 contains more details
c         about xtol.
c
c       maxfev is a positive integer input variable.  termination occurs
c         when the number of calls to fcn is at least maxfev by the end
c         of an iteration.
c
c       ml is a non-negative integer input variable which specifies the
c         number of subdiagonals within the band of the jacobian matrix.
c         if the jacobian is not banded or iopt=1, set ml to at
c         least n - 1.
c
c       mu is a non-negative integer input variable which specifies the
c         number of superdiagonals within the band of the jacobian
c         matrix.  if the jacobian is not banded or iopt=1, set mu to at
c         least n - 1.
c
c       epsfcn is an input variable used in determining a suitable step
c         for the forward-difference approximation.  this approximation
c         assumes that the relative errors in the functions are of the
c         order of epsfcn.  if epsfcn is less than the machine preci-
c         sion, it is assumed that the relative errors in the functions
c         are of the order of the machine precision.  if iopt=1, then
c         epsfcn can be ignored (treat it as a dummy argument).
c
c       diag is an array of length n.  if mode = 1 (see below), diag is
c         internally set.  if mode = 2, diag must contain positive
c         entries that serve as implicit (multiplicative) scale factors
c         for the variables.
c
c       mode is an integer input variable.  if mode = 1, the variables
c         will be scaled internally.  if mode = 2, the scaling is speci-
c         fied by the input diag.  other values of mode are equivalent
c         to mode = 1.
c
c       factor is a positive input variable used in determining the ini-
c         tial step bound.  this bound is set to the product of factor
c         and the euclidean norm of diag*x if nonzero, or else to factor
c         itself.  in most cases factor should lie in the interval
c         (.1,100.).  100. is a generally recommended value.
c
c       nprint is an integer input variable that enables controlled
c         printing of iterates if it is positive.  in this case, fcn is
c         called with iflag = 0 at the beginning of the first iteration
c         and every nprint iteration thereafter and immediately prior
c         to return, with x and fvec available for printing. appropriate
c         print statements must be added to fcn(see example).  if nprint
c         is not positive, no special calls of fcn with iflag = 0 are
c         made.
c
c       info is an integer output variable.  if the user has terminated
c         execution, info is set to the (negative) value of iflag.  see
c         description of fcn and jac. otherwise, info is set as follows.
c
c         info = 0  improper input parameters.
c
c         info = 1  relative error between two consecutive iterates is
c                   at most xtol.
c
c         info = 2  number of calls to fcn has reached or exceeded
c                   maxfev.
c
c         info = 3  xtol is too small.  no further improvement in the
c                   approximate solution x is possible.
c
c         info = 4  iteration is not making good progress, as measured
c                   by the improvement from the last five jacobian eval-
c                   uations.
c
c         info = 5  iteration is not making good progress, as measured
c                   by the improvement from the last ten iterations.
c
c         sections 4 and 5 contain more details about info.
c
c       nfev is an integer output variable set to the number of calls to
c         fcn.
c
c       njev is an integer output variable set to the number of calls to
c         jac. (if iopt=2, then njev is set to zero.)
c
c       r is an output array of length lr which contains the upper
c         triangular matrix produced by the qr factorization of the
c         final approximate jacobian, stored rowwise.
c
c       lr is a positive integer input variable not less than
c         (n*(n+1))/2.
c
c       qtf is an output array of length n which contains the vector
c         (q transpose)*fvec.
c
c       wa1, wa2, wa3, and wa4 are work arrays of length n.
c
c
c 4. successful completion.
c
c       the accuracy of snsq is controlled by the convergence parameter
c       xtol.  this parameter is used in a test which makes a comparison
c       between the approximation x and a solution xsol.  snsq termi-
c       nates when the test is satisfied.  if the convergence parameter
c       is less than the machine precision (as defined by the function
c       r1mach(4)), then snsq only attempts to satisfy the test
c       defined by the machine precision.  further progress is not
c       usually possible.
c
c       the test assumes that the functions are reasonably well behaved,
c       and, if the jacobian is supplied by the user, that the functions
c       and the jacobian are coded consistently.  if these conditions
c       are not satisfied, then snsq may incorrectly indicate conver-
c       gence.  the coding of the jacobian can be checked by the
c       subroutine chkder. if the jacobian is coded correctly or iopt=2,
c       then the validity of the answer can be checked, for example, by
c       rerunning snsq with a tighter tolerance.
c
c       convergence test.  if enorm(z) denotes the euclidean norm of a
c         vector z and d is the diagonal matrix whose entries are
c         defined by the array diag, then this test attempts to guaran-
c         tee that
c
c               enorm(d*(x-xsol)) .le. xtol*enorm(d*xsol).
```

```
c
c         if this condition is satisfied with xtol = 10**(-k), then the
c         larger components of d*x have k significant decimal digits and
c         info is set to 1.  there is a danger that the smaller compo-
c         nents of d*x may have large relative errors, but the fast rate
c         of convergence of snsq usually avoids this possibility.
c         unless high precision solutions are required, the recommended
c         value for xtol is the square root of the machine precision.
c
c
c 5. unsuccessful completion.
c
c       unsuccessful termination of snsq can be due to improper input
c       parameters, arithmetic interrupts, an excessive number of func-
c       tion evaluations, or lack of good progress.
c
c       improper input parameters.  info is set to 0 if iopt .lt. 1,
c         or iopt .gt. 2, or n .le. 0, or ldfjac .lt. n, or
c         xtol .lt. 0.e0, or maxfev .le. 0, or ml .lt. 0, or mu .lt. 0,
c         or factor .le. 0.e0, or lr .lt. (n*(n+1))/2.
c
c       arithmetic interrupts.  if these interrupts occur in the fcn
c         subroutine during an early stage of the computation, they may
c         be caused by an unacceptable choice of x by snsq.  in this
c         case, it may be possible to remedy the situation by rerunning
c         snsq with a smaller value of factor.
c
c       excessive number of function evaluations.  a reasonable value
c         for maxfev is 100*(n+1) for iopt=1 and 200*(n+1) for iopt=2.
c         if the number of calls to fcn reaches maxfev, then this
c         indicates that the routine is converging very slowly as
c         measured by the progress of fvec, and info is set to 2.  this
c         situation should be unusual because, as indicated below, lack
c         of good progress is usually diagnosed earlier by snsq,
c         causing termination with info = 4 or info = 5.
c
c       lack of good progress.  snsq searches for a zero of the system
c         by minimizing the sum of the squares of the functions.  in so
c         doing, it can become trapped in a region where the minimum
c         does not correspond to a zero of the system and, in this situ-
c         ation, the iteration eventually fails to make good progress.
c         in particular, this will happen if the system does not have a
c         zero.  if the system has a zero, rerunning snsq from a dif-
c         ferent starting point may be helpful.
c
c
c 6. characteristics of the algorithm.
c
c       snsq is a modification of the powell hybrid method.  two of its
c       main characteristics involve the choice of the correction as a
c       convex combination of the newton and scaled gradient directions,
c       and the updating of the jacobian by the rank-1 method of broy-
c       den.  the choice of the correction guarantees (under reasonable
c       conditions) global convergence for starting points far from the
c       solution and a fast rate of convergence.  the jacobian is
c       calculated at the starting point by either the user-supplied
c       subroutine or a forward-difference approximation, but it is not
c       recalculated until the rank-1 method fails to produce satis-
c       factory progress.
c
c       timing.  the time required by snsq to solve a given problem
c         depends on n, the behavior of the functions, the accuracy
c         requested, and the starting point.  the number of arithmetic
c         operations needed by snsq is about 11.5*(n**2) to process
c         each evaluation of the functions (call to fcn) and 1.3*(n**3)
c         to process each evaluation of the jacobian (call to jac,
c         if iopt = 1).  unless fcn and jac can be evaluated quickly,
c         the timing of snsq will be strongly influenced by the time
c         spent in fcn and jac.
c
c       storage.  snsq requires (3*n**2 + 17*n)/2 single precision
c         storage locations, in addition to the storage required by the
c         program.  there are no internally declared storage arrays.
c
c
c 7. example.
c
c       the problem is to determine the values of x(1), x(2), ..., x(9),
c       which solve the system of tridiagonal equations
c
c       (3-2*x(1))*x(1)           -2*x(2)                   = -1
c               -x(i-1) + (3-2*x(i))*x(i)         -2*x(i+1) = -1, i=2-8
c                                   -x(8) + (3-2*x(9))*x(9) = -1
c c     **********
c
c       program test(input,output,tape6=output)
c c
c c     driver for snsq example.
c c
c       integer j,iopt,n,maxfev,ml,mu,mode,nprint,info,nfev,ldfjac,lr,
c      *        nwrite
c       real xtol,epsfcn,factor,fnorm
c       real x(9),fvec(9),diag(9),fjac(9,9),r(45),qtf(9),
c      *     wa1(9),wa2(9),wa3(9),wa4(9)
c       real enorm,r1mach
c       external fcn
c       data nwrite /6/
c c
c       iopt = 2
c       n = 9
c c
c c     the following starting values provide a rough solution.
c c
c       do 10 j = 1, 9
c          x(j) = -1.e0
c    10    continue
c c
c       ldfjac = 9
c       lr = 45
c c
c c     set xtol to the square root of the machine precision.
c c     unless high precision solutions are required,
c c     this is the recommended setting.
c c
c       xtol = sqrt(r1mach(4))
c c
c       maxfev = 2000
c       ml = 1
c       mu = 1
c       epsfcn = 0.e0
c       mode = 2
c       do 20 j = 1, 9
c          diag(j) = 1.e0
c    20    continue
c       factor = 1.e2
c       nprint = 0
c c
c       call snsq(fcn,jac,iopt,n,x,fvec,fjac,ldfjac,xtol,maxfev,ml,mu,
c      *           epsfcn,diag,mode,factor,nprint,info,nfev,njev,
c      *           r,lr,qtf,wa1,wa2,wa3,wa4)
```

```
c       fnorm = enorm(n,fvec)
c       write (nwrite,1000) fnorm,nfev,info,(x(j),j=1,n)
c       stop
c  1000 format (5x,' final l2 norm of the residuals',e15.7 //
c      *        5x,' number of function evaluations',i10 //
c      *        5x,' exit parameter',16x,i10 //
c      *        5x,' final approximate solution' // (5x,3e15.7))
c       end
c       subroutine fcn(n,x,fvec,iflag)
c       integer n,iflag
c       real x(n),fvec(n)
c       integer k
c       real one,temp,temp1,temp2,three,two,zero
c       data zero,one,two,three /0.e0,1.e0,2.e0,3.e0/
c c
c       if (iflag .ne. 0) go to 5
c c
c c     insert print statements here when nprint is positive.
c c
c       return
c     5 continue
c       do 10 k = 1, n
c          temp = (three - two*x(k))*x(k)
c          temp1 = zero
c          if (k .ne. 1) temp1 = x(k-1)
c          temp2 = zero
c          if (k .ne. n) temp2 = x(k+1)
c          fvec(k) = temp - temp1 - two*temp2 + one
c    10    continue
c       return
c       end
c
c       results obtained with different compilers or machines
c       may be slightly different.
c
c       final l2 norm of the residuals  0.1192636e-07
c
c       number of function evaluations        14
c
c       exit parameter                         1
c
c       final approximate solution
c
c       -0.5706545e+00 -0.6816283e+00 -0.7017325e+00
c       -0.7042129e+00 -0.7013690e+00 -0.6918656e+00
c       -0.6657920e+00 -0.5960342e+00 -0.4164121e+00
c***references  powell, m. j. d.
c                 a hybrid method for nonlinear equations.
c                 numerical methods for nonlinear algebraic equations,
c                 p. rabinowitz, editor.  gordon and breach, 1970.
c***routines called  dogleg,enorm,fdjac1,qform,qrfac,r1mach,r1mpyq,
c                    r1updt,xerror
c***end prologue  snsq
      integer iopt,n,maxfev,ml,mu,mode,nprint,info,nfev,ldfjac,lr,njev
      double precision xtol,epsfcn,factor
      double precision x(n),fvec(n),diag(n),fjac(ldfjac,n),r(lr),
     1     qtf(n),wa1(n),wa2(n),wa3(n),wa4(n)
      external fcn,jac
      integer i,iflag,iter,j,jm1,l,ncfail,ncsuc,nslow1,nslow2
      integer iwa(1)
      logical jeval,sing
      double precision actred,delta,epsmch,fnorm,fnorm1,
     1     p1,p5,p001,p0001,ratio,sum,temp,xnorm,zero
      double precision one,pnorm,prered
      double precision r1mach,enorm
      data one,p1,p5,p001,p0001,zero
     1     /1.0d0,1.0d-1,5.0d-1,1.0d-3,1.0d-4,0.0d0/
c
c***first executable statement  snsq
      epsmch = r1mach(4)
c
      info = 0
      iflag = 0
      nfev = 0
      njev = 0
c
c     check the input parameters for errors.
c
      if (iopt .lt. 1 .or. iopt .gt. 2 .or.
     1    n .le. 0 .or. xtol .lt. zero .or. maxfev .le. 0
     2    .or. ml .lt. 0 .or. mu .lt. 0 .or. factor .le. zero
     3    .or. ldfjac .lt. n .or. lr .lt. (n*(n + 1))/2) go to 300
      if (mode .ne. 2) go to 20
      do 10 j = 1, n
         if (diag(j) .le. zero) go to 300
   10    continue
   20 continue
c
c     evaluate the function at the starting point
c     and calculate its norm.
c
      iflag = 1
      call fcn(n,x,fvec,iflag)
      nfev = 1
      if (iflag .lt. 0) go to 300
      fnorm = enorm(n,fvec)
c
c     initialize iteration counter and monitors.
c
      iter = 1
      ncsuc = 0
      ncfail = 0
      nslow1 = 0
      nslow2 = 0
c
c     beginning of the outer loop.
c
   30 continue
         jeval = .true.
c
c        calculate the jacobian matrix.
c
         if (iopt .eq. 2) go to 31
c
c        user supplies jacobian
c
            call jac(n,x,fvec,fjac,ldfjac,iflag)
            njev = njev+1
            go to 32
c
c        code approximates the jacobian
c
   31       iflag = 2
            call fdjac1(fcn,n,x,fvec,fjac,ldfjac,iflag,ml,mu,epsfcn,wa1,
     1                  wa2)
            nfev = nfev + min(ml+mu+1,n)
c
   32    if (iflag .lt. 0) go to 300
c
c        compute the qr factorization of the jacobian.
```

```
c
         call qrfac(n,n,fjac,ldfjac,.false.,iwa,1,wa1,wa2,wa3)
c
c        on the first iteration and if mode is 1, scale according
c        to the norms of the columns of the initial jacobian.
c
         if (iter .ne. 1) go to 70
         if (mode .eq. 2) go to 50
         do 40 j = 1, n
            diag(j) = wa2(j)
            if (wa2(j) .eq. zero) diag(j) = one
   40       continue
   50    continue
c
c        on the first iteration, calculate the norm of the scaled x
c        and initialize the step bound delta.
c
         do 60 j = 1, n
            wa3(j) = diag(j)*x(j)
   60       continue
         xnorm = enorm(n,wa3)
         delta = factor*xnorm
         if (delta .eq. zero) delta = factor
   70    continue
c
c        form (q transpose)*fvec and store in qtf.
c
         do 80 i = 1, n
            qtf(i) = fvec(i)
   80       continue
         do 120 j = 1, n
            if (fjac(j,j) .eq. zero) go to 110
            sum = zero
            do 90 i = j, n
               sum = sum + fjac(i,j)*qtf(i)
   90          continue
            temp = -sum/fjac(j,j)
            do 100 i = j, n
               qtf(i) = qtf(i) + fjac(i,j)*temp
  100          continue
  110       continue
  120       continue
c
c        copy the triangular factor of the qr factorization into r.
c
         sing = .false.
         do 150 j = 1, n
            l = j
            jm1 = j - 1
            if (jm1 .lt. 1) go to 140
            do 130 i = 1, jm1
               r(l) = fjac(i,j)
               l = l + n - i
  130          continue
  140       continue
            r(l) = wa1(j)
            if (wa1(j) .eq. zero) sing = .true.
  150       continue
c
c        accumulate the orthogonal factor in fjac.
c
         call qform(n,n,fjac,ldfjac,wa1)
c
c        rescale if necessary.
c
         if (mode .eq. 2) go to 170
         do 160 j = 1, n
            diag(j) = max(diag(j),wa2(j))
  160       continue
  170    continue
c
c        beginning of the inner loop.
c
  180    continue
c
c           if requested, call fcn to enable printing of iterates.
c
            if (nprint .le. 0) go to 190
            iflag = 0
            if (mod(iter-1,nprint) .eq. 0) call fcn(n,x,fvec,iflag)
            if (iflag .lt. 0) go to 300
  190       continue
c
c           determine the direction p.
c
            call dogleg(n,r,lr,diag,qtf,delta,wa1,wa2,wa3)
c
c           store the direction p and x + p. calculate the norm of p.
c
            do 200 j = 1, n
               wa1(j) = -wa1(j)
               wa2(j) = x(j) + wa1(j)
               wa3(j) = diag(j)*wa1(j)
  200          continue
            pnorm = enorm(n,wa3)
c
c           on the first iteration, adjust the initial step bound.
c
            if (iter .eq. 1) delta = min(delta,pnorm)
c
c           evaluate the function at x + p and calculate its norm.
c
            iflag = 1
            call fcn(n,wa2,wa4,iflag)
            nfev = nfev + 1
            if (iflag .lt. 0) go to 300
            fnorm1 = enorm(n,wa4)
c
c           compute the scaled actual reduction.
c
            actred = -one
            if (fnorm1 .lt. fnorm) actred = one - (fnorm1/fnorm)**2
c
c           compute the scaled predicted reduction.
c
            l = 1
            do 220 i = 1, n
               sum = zero
               do 210 j = i, n
                  sum = sum + r(l)*wa1(j)
                  l = l + 1
  210             continue
               wa3(i) = qtf(i) + sum
  220          continue
            temp = enorm(n,wa3)
            prered = zero
            if (temp .lt. fnorm) prered = one - (temp/fnorm)**2
c
c           compute the ratio of the actual to the predicted
c           reduction.
```

```
c
            ratio = zero
            if (prered .gt. zero) ratio = actred/prered
c
c           update the step bound.
c
            if (ratio .ge. p1) go to 230
               ncsuc = 0
               ncfail = ncfail + 1
               delta = p5*delta
               go to 240
  230       continue
               ncfail = 0
               ncsuc = ncsuc + 1
               if (ratio .ge. p5 .or. ncsuc .gt. 1)
     1            delta = max(delta,pnorm/p5)
               if (abs(ratio-one) .le. p1) delta = pnorm/p5
  240       continue
c
c           test for successful iteration.
c
            if (ratio .lt. p0001) go to 260
c
c           successful iteration. update x, fvec, and their norms.
c
            do 250 j = 1, n
               x(j) = wa2(j)
               wa2(j) = diag(j)*x(j)
               fvec(j) = wa4(j)
  250          continue
            xnorm = enorm(n,wa2)
            fnorm = fnorm1
            iter = iter + 1
  260       continue
c
c           determine the progress of the iteration.
c
            nslow1 = nslow1 + 1
            if (actred .ge. p001) nslow1 = 0
            if (jeval) nslow2 = nslow2 + 1
            if (actred .ge. p1) nslow2 = 0
c
c           test for convergence.
c
            if (delta .le. xtol*xnorm .or. fnorm .eq. zero) info = 1
            if (info .ne. 0) go to 300
c
c           tests for termination and stringent tolerances.
c
            if (nfev .ge. maxfev) info = 2
            if (p1*max(p1*delta,pnorm) .le. epsmch*xnorm) info = 3
            if (nslow2 .eq. 5) info = 4
            if (nslow1 .eq. 10) info = 5
            if (info .ne. 0) go to 300
c
c           criterion for recalculating jacobian
c
            if (ncfail .eq. 2) go to 290
c
c           calculate the rank one modification to the jacobian
c           and update qtf if necessary.
c
            do 280 j = 1, n
               sum = zero
               do 270 i = 1, n
                  sum = sum + fjac(i,j)*wa4(i)
  270             continue
               wa2(j) = (sum - wa3(j))/pnorm
               wa1(j) = diag(j)*((diag(j)*wa1(j))/pnorm)
               if (ratio .ge. p0001) qtf(j) = sum
  280          continue
c
c           compute the qr factorization of the updated jacobian.
c
            call r1updt(n,n,r,lr,wa1,wa2,wa3,sing)
            call r1mpyq(n,n,fjac,ldfjac,wa2,wa3)
            call r1mpyq(1,n,qtf,1,wa2,wa3)
c
c           end of the inner loop.
c
            jeval = .false.
            go to 180
  290    continue
c
c        end of the outer loop.
c
         go to 30
  300 continue
c
c     termination, either normal or user imposed.
c
      if (iflag .lt. 0) info = iflag
      iflag = 0
      if (nprint .gt. 0) call fcn(n,x,fvec,iflag)
      if (info .lt. 0) call xerror( 'snsq   -- execution terminated beca
     1use user set iflag negative.',63,1,1)
      if (info .eq. 0) call xerror( 'snsq   -- invalid input parameter.'
     1 ,34,2,1)
      if (info .eq. 2) call xerror( 'snsq   -- too many function evaluat
     1ions.',40,9,1)
      if (info .eq. 3) call xerror( 'snsq   -- xtol too small. no furthe
     1r improvement possible.',58,3,1)
      if (info .gt. 4) call xerror( 'snsq   -- iteration not making good
     1 progress.',45,1,1)
      return
c
c     last card of subroutine snsq.
c
      end

c****************************************
c
      subroutine xerabt(messg,nmessg)
c***begin prologue  xerabt
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  r3c
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  aborts program execution and prints error message.
c***description
c     abstract
c        ***note*** machine dependent routine
c        xerabt aborts the execution of the program.
c        the error message causing the abort is given in the calling
c        sequence, in case one needs it for printing on a dayfile,
c        for example.
c
c     description of parameters
c        messg and nmessg are as in xerror, except that nmessg may
```

```
c        be zero, in which case no message is being supplied.
c
c     written by ron jones, with slatec common math library subcommittee
c     latest revision ---  19 mar 1980
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  (none)
c***end prologue  xerabt
      implicit none
      integer nmessg
      character*(*) messg
c***first executable statement  xerabt
      write(8, *) 'XERABT CALLED, ABORT'
      stop
      end


c***********************************************************
c
      subroutine xerctl(messg1,nmessg,nerr,level,kontrl)
c***begin prologue  xerctl
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  r3c
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  allows user control over handling of individual errors.
c***description
c     abstract
c        allows user control over handling of individual errors.
c        just after each message is recorded, but before it is
c        processed any further (i.e., before it is printed or
c        a decision to abort is made), a call is made to xerctl.
c        if the user has provided his own version of xerctl, he
c        can then override the value of kontrol used in processing
c        this message by redefining its value.
c        kontrl may be set to any value from -2 to 2.
c        the meanings for kontrl are the same as in xsetf, except
c        that the value of kontrl changes only for this message.
c        if kontrl is set to a value outside the range from -2 to 2,
c        it will be moved back into that range.
c
c     description of parameters
c
c      --input--
c        messg1 - the first word (only) of the error message.
c        nmessg - same as in the call to xerror or xerrwv.
c        nerr   - same as in the call to xerror or xerrwv.
c        level  - same as in the call to xerror or xerrwv.
c        kontrl - the current value of the control flag as set
c                 by a call to xsetf.
c
c      --output--
c        kontrl - the new value of kontrl.  if kontrl is not
c                 defined, it will remain at its original value.
c                 this changed value of control affects only
c                 the current occurrence of the current message.
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  (none)
c***end prologue  xerctl
      implicit none
      integer nmessg, nerr, level, kontrl
      character*20 messg1
c***first executable statement  xerctl
      return
      end
c     head.h Version 1.1
c*************************************************************************
c
c       Oregon State University Nuclear Theory Group
c       Multi-Channel Anti Kaon -- Nucleon Code
c       Guangliang He & Rubin H. Landau
c
c*************************************************************************
c     fortran.h Version 1.4
c*************************************************************************
c     xerprt.F Version 1.1
c     Latest Revision Date 23:17:54 11/12/90
c*************************************************************************
c
      subroutine xerprt(messg,nmessg)
c***begin prologue  xerprt
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  z
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  prints error messages.
c***description
c     abstract
c        print the hollerith message in messg, of length nmessg,
c        on each file indicated by xgetua.
c     latest revision ---  19 mar 1980
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  i1mach,s88fmt,xgetua
c***end prologue  xerprt
      implicit none
      integer nmessg,nunit,lenmes,kunit,iunit,ichar,last
      integer i1mach
      integer lun(5)
      character*(*) messg
c     obtain unit numbers and write line to each unit
c***first executable statement  xerprt
      call xgetua(lun,nunit)
      lenmes = len(messg)
      do 20 kunit=1,nunit
         iunit = lun(kunit)
         if (iunit.eq.0) iunit = i1mach(4)
         do 10 ichar=1,lenmes,72
            last = min(ichar+71 , lenmes)
            write (iunit,'(1x,a)') messg(ichar:last)
   10    continue
   20 continue
      return
      end
c
c*************************************************************************
c
      subroutine xerror(messg,nmessg,nerr,level)
c***begin prologue  xerror
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  r3c
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
```

```
c***purpose  processes an error (diagnostic) message.
c***description
c     abstract
c        xerror processes a diagnostic message, in a manner
c        determined by the value of level and the current value
c        of the library error control flag, kontrl.
c        (see subroutine xsetf for details.)
c
c     description of parameters
c      --input--
c        messg - the hollerith message to be processed, containing
c                no more than 72 characters.
c        nmessg- the actual number of characters in messg.
c        nerr  - the error number associated with this message.
c                nerr must not be zero.
c        level - error category.
c                =2 means this is an unconditionally fatal error.
c                =1 means this is a recoverable error.  (i.e., it is
c                   non-fatal if xsetf has been appropriately called.)
c                =0 means this is a warning message only.
c                =-1 means this is a warning message which is to be
c                   printed at most once, regardless of how many
c                   times this call is executed.
c
c     examples
c        call xerror('smooth -- num was zero.',23,1,2)
c        call xerror('integ  -- less than full accuracy achieved.',
c                     43,2,1)
c        call xerror('rooter -- actual zero of f found before interval f
c    1ully collapsed.',65,3,0)
c        call xerror('exp    -- underflows being set to zero.',39,1,-1)
c
c     latest revision ---  19 mar 1980
c     written by ron jones, with slatec common math library subcommittee
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  xerrwv
c***end prologue  xerror
      implicit none
      integer nmessg, nerr, level
      character*(*) messg
c***first executable statement  xerror
      call xerrwv(messg,nmessg,nerr,level,0,0,0,0,0.,0.)
      return
      end

c*****************************************************************
c
      subroutine xerrwv(messg,nmessg,nerr,level,ni,i1,i2,nr,r1,r2)
c***begin prologue  xerrwv
c***date written   800319   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  r3c
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  processes error message allowing 2 integer and two real
c            values to be included in the message.
c***description
c     abstract
c        xerrwv processes a diagnostic message, in a manner
c        determined by the value of level and the current value
c        of the library error control flag, kontrl.
c        (see subroutine xsetf for details.)
c        in addition, up to two integer values and two real
c        values may be printed along with the message.
c
c     description of parameters
c      --input--
c        messg - the hollerith message to be processed.
c        nmessg- the actual number of characters in messg.
c        nerr  - the error number associated with this message.
c                nerr must not be zero.
c        level - error category.
c                =2 means this is an unconditionally fatal error.
c                =1 means this is a recoverable error.  (i.e., it is
c                   non-fatal if xsetf has been appropriately called.)
c                =0 means this is a warning message only.
c                =-1 means this is a warning message which is to be
c                   printed at most once, regardless of how many
c                   times this call is executed.
c        ni    - number of integer values to be printed. (0 to 2)
c        i1    - first integer value.
c        i2    - second integer value.
c        nr    - number of real values to be printed. (0 to 2)
c        r1    - first real value.
c        r2    - second real value.
c
c     examples
c        call xerrwv('smooth -- num (=i1) was zero.',29,1,2,
c    1   1,num,0,0,0.,0.)
c        call xerrwv('quadxy -- requested error (r1) less than minimum (
c    1r2).,54,77,1,0,0,0,2,errreq,errmin)
c
c     latest revision ---  19 mar 1980
c     written by ron jones, with slatec common math library subcommittee
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  fdump,i1mach,j4save,xerabt,xerctl,xerprt,xersav,
c                    xgetua
c***end prologue  xerrwv
      implicit none
      integer lun(5), nmessg, nerr, level, ni, i1, i2, nr
      integer lkntrl, maxmes, kdummy, junk, kount, lmessg
      integer lerr, llevel, mkntrl, nunit, isizei, isizef
      integer kunit, iunit, i, ifatal
      integer j4save, i1mach
      double precision r1, r2
      character*(*) messg
      character*20 lfirst
      character*37 form
c     get flags
c***first executable statement  xerrwv
      lkntrl = j4save(2,0,.false.)
      maxmes = j4save(4,0,.false.)
c     check for valid input
      if ((nmessg.gt.0).and.(nerr.ne.0).and.
     1    (level.ge.(-1)).and.(level.le.2)) go to 10
         if (lkntrl.gt.0) call xerprt('fatal error in...',17)
         call xerprt('xerror -- invalid input',23)
         if (lkntrl.gt.0) call fdump
         if (lkntrl.gt.0) call xerprt('job abort due to fatal error.',
     1  29)
         if (lkntrl.gt.0) call xersav(' ',0,0,0,kdummy)
         call xerabt('xerror -- invalid input',23)
         return
   10 continue
c     record message
      junk = j4save(1,nerr,.true.)
```

```
      call xersav(messg,nmessg,nerr,level,kount)
c     let user override
      lfirst = messg
      lmessg = nmessg
      lerr = nerr
      llevel = level
      call xerctl(lfirst,lmessg,lerr,llevel,lkntrl)
c     reset to original values
      lmessg = nmessg
      lerr = nerr
      llevel = level
      lkntrl = max0(-2,min0(2,lkntrl))
      mkntrl = iabs(lkntrl)
c     decide whether to print message
      if ((llevel.lt.2).and.(lkntrl.eq.0)) go to 100
      if (((llevel.eq.(-1)).and.(kount.gt.min0(1,maxmes)))
     1.or.((llevel.eq.0)   .and.(kount.gt.maxmes))
     2.or.((llevel.eq.1)   .and.(kount.gt.maxmes).and.(mkntrl.eq.1))
     3.or.((llevel.eq.2)   .and.(kount.gt.max0(1,maxmes)))) go to 100
         if (lkntrl.le.0) go to 20
            call xerprt(' ',1)
c           introduction
            if (llevel.eq.(-1)) call xerprt
     1('warning message...this message will only be printed once.',57)
            if (llevel.eq.0) call xerprt('warning in...',13)
            if (llevel.eq.1) call xerprt
     1      ('recoverable error in...',23)
            if (llevel.eq.2) call xerprt('fatal error in...',17)
   20    continue
c        message
         call xerprt(messg,lmessg)
         call xgetua(lun,nunit)
         isizei = log10(float(i1mach(9))) + 1.0
         isizef = log10(float(i1mach(10))**i1mach(11)) + 1.0
         do 50 kunit=1,nunit
            iunit = lun(kunit)
            if (iunit.eq.0) iunit = i1mach(4)
            do 22 i=1,min(ni,2)
               write (form,21) i,isizei
   21          format ('(11x,21hin above message, i',i1,'=,i',i2,')   ')
               if (i.eq.1) write (iunit,form) i1
               if (i.eq.2) write (iunit,form) i2
   22       continue
            do 24 i=1,min(nr,2)
               write (form,23) i,isizef+10,isizef
   23          format ('(11x,21hin above message, r',i1,'=,e',
     1         i2,'.',i2,')')
               if (i.eq.1) write (iunit,form) r1
               if (i.eq.2) write (iunit,form) r2
   24       continue
            if (lkntrl.le.0) go to 40
c              error number
               write (iunit,30) lerr
   30          format (15h error number =,i10)
   40       continue
   50    continue
c        trace-back
         if (lkntrl.gt.0) call fdump
  100 continue
      ifatal = 0
      if ((llevel.eq.2).or.((llevel.eq.1).and.(mkntrl.eq.2)))
     1ifatal = 1
c     quit here if message is not fatal
      if (ifatal.le.0) return
      if ((lkntrl.le.0).or.(kount.gt.max0(1,maxmes))) go to 120
c        print reason for abort
         if (llevel.eq.1) call xerprt
     1   ('job abort due to unrecovered error.',35)
         if (llevel.eq.2) call xerprt
     1   ('job abort due to fatal error.',29)
c        print error summary
         call xersav(' ',-1,0,0,kdummy)
  120 continue
c     abort
      if ((llevel.eq.2).and.(kount.gt.max0(1,maxmes))) lmessg = 0
      call xerabt(messg,lmessg)
      return
      end


c*******************************************************
c
      subroutine xersav(messg,nmessg,nerr,level,icount)
c***begin prologue  xersav
c***date written   800319   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  z
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  records that an error occurred.
c***description
c     abstract
c        record that this error occurred.
c
c     description of parameters
c     --input--
c       messg, nmessg, nerr, level are as in xerror,
c       except that when nmessg=0 the tables will be
c       dumped and cleared, and when nmessg is less than zero the
c       tables will be dumped and not cleared.
c     --output--
c       icount will be the number of times this message has
c       been seen, or zero if the table has overflowed and
c       does not contain this message specifically.
c       when nmessg=0, icount will not be altered.
c
c     written by ron jones, with slatec common math library subcommittee
c     latest revision ---  19 mar 1980
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  i1mach,s88fmt,xgetua
c***end prologue  xersav
      implicit none
      integer lun(5)
      character*(*) messg
      character*20 mestab(10),mes
      integer nertab(10),levtab(10),kount(10),i,ii
      integer kountx,nmessg,nerr,level,icount,nunit,kunit,iunit
      integer i1mach
      save mestab,nertab,levtab,kount,kountx
c     next two data statements are necessary to provide a blank
c     error table initially
      data kount(1),kount(2),kount(3),kount(4),kount(5),
     1     kount(6),kount(7),kount(8),kount(9),kount(10)
     2     /0,0,0,0,0,0,0,0,0,0/
      data kountx/0/
c***first executable statement  xersav
      if (nmessg.gt.0) go to 80
c     dump the table
```

```
         if (kount(1).eq.0) return
c        print to each unit
         call xgetua(lun,nunit)
         do 60 kunit=1,nunit
            iunit = lun(kunit)
            if (iunit.eq.0) iunit = i1mach(4)
c           print table header
            write (iunit,10)
   10       format (32h0          error message summary/
     1      51h message start             nerr     level     count)
c           print body of table
            do 20 i=1,10
               if (kount(i).eq.0) go to 30
               write (iunit,15) mestab(i),nertab(i),levtab(i),kount(i)
   15          format (1x,a20,3i10)
   20       continue
   30       continue
c           print number of other errors
            if (kountx.ne.0) write (iunit,40) kountx
   40       format (41h0other errors not individually tabulated=,i10)
            write (iunit,50)
   50       format (1x)
   60    continue
         if (nmessg.lt.0) return
c        clear the error tables
         do 70 i=1,10
   70       kount(i) = 0
         kountx = 0
         return
   80 continue
c     process a message...
c     search for this messg, or else an empty slot for this messg,
c     or else determine that the error table is full.
      mes = messg
      do 90 i=1,10
         ii = i
         if (kount(i).eq.0) go to 110
         if (mes.ne.mestab(i)) go to 90
         if (nerr.ne.nertab(i)) go to 90
         if (level.ne.levtab(i)) go to 90
         go to 100
   90 continue
c     three possible cases...
c     table is full
         kountx = kountx+1
         icount = 1
         return
c     message found in table
  100    kount(ii) = kount(ii) + 1
         icount = kount(ii)
         return
c     empty slot found for new message
  110    mestab(ii) = mes
         nertab(ii) = nerr
         levtab(ii) = level
         kount(ii)  = 1
         icount = 1
         return
      end

c***********************************
c
      subroutine xgetua(iunita,n)
c***begin prologue  xgetua
c***date written   790801   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  r3c
c***keywords  error,xerror package
c***author  jones, r. e., (snla)
c***purpose  returns unit number(s) to which error messages are being
c            sent.
c***description
c     abstract
c        xgetua may be called to determine the unit number or numbers
c        to which error messages are being sent.
c        these unit numbers may have been set by a call to xsetun,
c        or a call to xsetua, or may be a default value.
c
c     description of parameters
c      --output--
c        iunit - an array of one to five unit numbers, depending
c                on the value of n.  a value of zero refers to the
c                default unit, as defined by the i1mach machine
c                constant routine.  only iunit(1),...,iunit(n) are
c                defined by xgetua.  the values of iunit(n+1),...,
c                iunit(5) are not defined (for n .lt. 5) or altered
c                in any way by xgetua.
c        n     - the number of units to which copies of the
c                error messages are being sent.  n will be in the
c                range from 1 to 5.
c
c     latest revision ---  19 mar 1980
c     written by ron jones, with slatec common math library subcommittee
c***references  jones r.e., kahaner d.k., "xerror, the slatec error-
c                 handling package", sand82-0800, sandia laboratories,
c                 1982.
c***routines called  j4save
c***end prologue  xgetua
      implicit none
      integer iunita(5), n, i, index
      integer j4save
c***first executable statement  xgetua
      n = j4save(5,0,.false.)
      do 30 i=1,n
         index = i+4
         if (i.eq.1) index = 3
         iunita(i) = j4save(index,0,.false.)
   30 continue
      return
      end

c********************************************
c
      subroutine zaxpy(n,ca,cx,incx,cy,incy)
c***begin prologue  zaxpy
c***date written   791001   (yymmdd)
c***revision date  840425   (yymmdd)
c***category no.  d1a7
c***keywords  blas,complex,linear algebra,triad,vector
c***author  lawson, c. l., (jpl)
c           hanson, r. j., (snla)
c           kincaid, d. r., (u. of texas)
c           krogh, f. t., (jpl)
c***purpose  complex computation y = a*x + y
c***description
c
c                b l a s  subprogram
c    description of parameters
c
c     --input--
```

```
c        n  number of elements in input vector(s)
c       ca  complex scalar multiplier
c       cx  complex vector with n elements
c     incx  storage spacing between elements of cx
c       cy  complex vector with n elements
c     incy  storage spacing between elements of cy
c
c     --output--
c       cy  complex result (unchanged if n .le. 0)
c
c     overwrite complex cy with complex  ca*cx + cy.
c     for i = 0 to n-1, replace  cy(ly+i*incy) with ca*cx(lx+i*incx) +
c       cy(ly+i*incy), where lx = 1 if incx .ge. 0, else lx = (-incx)*n
c       and ly is defined in a similar way using incy.
c***references  lawson c.l., hanson r.j., kincaid d.r., krogh f.t.,
c                 *basic linear algebra subprograms for fortran usage*,
c                 algorithm no. 539, transactions on mathematical
c                 software, volume 5, number 3, september 1979, 308-323
c***routines called  (none)
c***end prologue  zaxpy
c
      implicit none
      integer i,incx,incy,kx,ky,n,ns
      double precision canorm
      double complex cx(1),cy(1),ca
c***first executable statement  caxpy
      canorm = abs(dble(ca)) + abs(imag(ca))
      if(n.le.0.or.canorm.eq.0.e0) return
      if(incx.eq.incy.and.incx.gt.0) go to 20
      kx = 1
      ky = 1
      if(incx.lt.0) kx = 1+(1-n)*incx
      if(incy.lt.0) ky = 1+(1-n)*incy
          do 10 i = 1,n
          cy(ky) = cy(ky) + ca*cx(kx)
          kx = kx + incx
          ky = ky + incy
   10 continue
      return
   20 continue
      ns = n*incx
          do 30 i=1,ns,incx
          cy(i) = ca*cx(i) + cy(i)
   30     continue
      return
      end

c*******************************************
c
      subroutine zcopy(n,cx,incx,cy,incy)
c***begin prologue  zcopy
c***date written   791001   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d1a5
c***keywords  blas,complex,copy,linear algebra,vector
c***author  lawson, c. l., (jpl)
c           hanson, r. j., (snla)
c           kincaid, d. r., (u. of texas)
c           krogh, f. t., (jpl)
c***purpose  complex vector copy y = x
c***description
c
c                b l a s  subprogram
c    description of parameters
c
c     --input--
c        n  number of elements in input vector(s)
c       cx  complex vector with n elements
c     incx  storage spacing between elements of cx
c       cy  complex vector with n elements
c     incy  storage spacing between elements of cy
c
c     --output--
c       cy  copy of vector cx (unchanged if n .le. 0)
c
c     copy complex cx to complex cy.
c     for i = 0 to n-1, copy cx(lx+i*incx) to cy(ly+i*incy),
c     where lx = 1 if incx .ge. 0, else lx = (-incx)*n, and ly is
c     defined in a similar way using incy.
c***references  lawson c.l., hanson r.j., kincaid d.r., krogh f.t.,
c                 *basic linear algebra subprograms for fortran usage*,
c                 algorithm no. 539, transactions on mathematical
c                 software, volume 5, number 3, september 1979, 308-323
c***routines called  (none)
c***end prologue  zcopy
c
      implicit none
      integer kx, ky, i, n, ns, incx, incy
      double complex cx(1),cy(1)
c***first executable statement  zcopy
      if(n .le. 0)return
      if(incx.eq.incy.and.incx.gt.0) go to 20
      kx = 1
      ky = 1
      if(incx.lt.0) kx = 1+(1-n)*incx
      if(incy.lt.0) ky = 1+(1-n)*incy
          do 10 i = 1,n
          cy(ky) = cx(kx)
          kx = kx + incx
          ky = ky + incy
   10 continue
      return
   20 continue
      ns = n*incx
          do 30 i=1,ns,incx
          cy(i) = cx(i)
   30     continue
      return
      end

c****************************************
c
      function zdotu(n,cx,incx,cy,incy)
c***begin prologue  zdotu
c***date written   791001   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d1a4
c***keywords  blas,complex,inner product,linear algebra,vector
c***author  lawson, c. l., (jpl)
c           hanson, r. j., (snla)
c           kincaid, d. r., (u. of texas)
c           krogh, f. t., (jpl)
c***purpose  inner product of complex vectors
c***description
c
c                b l a s  subprogram
c    description of parameters
c
c     --input--
c        n  number of elements in input vector(s)
```

```
c       cx  complex vector with n elements
c     incx  storage spacing between elements of cx
c       cy  complex vector with n elements
c     incy  storage spacing between elements of cy
c
c     --output--
c    zdotu  complex result (zero if n .le. 0)
c
c     returns the dot product for complex cx and cy, no conjugation
c     zdotu = sum for i = 0 to n-1 of  cx(lx+i*incx) * cy(ly+i*incy),
c     where lx = 1 if incx .ge. 0, else lx = (-incx)*n, and ly is
c     defined in a similar way using incy.
c***references  lawson c.l., hanson r.j., kincaid d.r., krogh f.t.,
c                 *basic linear algebra subprograms for fortran usage*,
c                 algorithm no. 539, transactions on mathematical
c                 software, volume 5, number 3, september 1979, 308-323
c***routines called  (none)
c***end prologue  zdotu
c
                implicit none
                integer i, incx, incy, kx, ky, n, ns
      double complex cx(1),cy(1), zdotu
c***first executable statement  zdotu
      zdotu = (0.0d0,0.0d0)
      if(n .le. 0)return
      if(incx.eq.incy.and.incx.gt.0) go to 20
      kx = 1
      ky = 1
      if(incx.lt.0) kx = 1+(1-n)*incx
      if(incy.lt.0) ky = 1+(1-n)*incy
          do 10 i = 1,n
          zdotu = zdotu + cx(kx)*cy(ky)
          kx = kx + incx
          ky = ky + incy
   10     continue
      return
   20 continue
      ns = n*incx
          do 30 i=1,ns,incx
          zdotu = zdotu + cx(i)*cy(i)
   30     continue
      return
      end

c******************************************************
c
      subroutine zsidi(a,lda,n,kpvt,det,work,job)
c***begin prologue  zsidi
c***date written   780814   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d2d1a,d3d1a
c***keywords  complex,determinant,factor,inverse,linear algebra,linpack,
c             matrix,symmetric
c***author  bunch, j., (ucsd)
c***purpose  computes the determinant and inverse of a complex symmetric
c            matrix using the factors from zsifa.
c***description
c
c     zsidi computes the determinant and inverse
c     of a complex symmetric matrix using the factors from zsifa.
c
c     on entry
c
c        a       complex(lda,n)
c                the output from zsifa.
c
c        lda     integer
c                the leading dimension of the array a .
c
c        n       integer
c                the order of the matrix a .
c
c        kvpt    integer(n)
c                the pivot vector from zsifa.
c
c        work    complex(n)
c                work vector.  contents destroyed.
c
c        job     integer
c                job has the decimal expansion  ab  where
c                   if  b .ne. 0, the inverse is computed,
c                   if  a .ne. 0, the determinant is computed,
c
c                for example, job = 11  gives both.
c
c     on return
c
c        variables not requested by job are not used.
c
c        a      contains the upper triangle of the inverse of
c               the original matrix.  the strict lower triangle
c               is never referenced.
c
c        det    complex(2)
c               determinant of original matrix.
c               determinant = det(1) * 10.0**det(2)
c               with 1.0 .le. abs(det(1)) .lt. 10.0
c               or det(1) = 0.0.
c
c     error condition
c
c        a division by zero may occur if the inverse is requested
c        and  zsico  has set rcond .eq. 0.0
c        or  zsifa  has set  info .ne. 0 .
c
c     linpack.  this version dated 08/14/78 .
c     james bunch, univ. calif. san diego, argonne nat. lab.
c
c     subroutines and functions
c
c     blas zaxpy,zcopy,zdotu,zswap
c     fortran abs,cmplx,iabs,mod,real
c***references  dongarra j.j., bunch j.r., moler c.b., stewart g.w.,
c                 *linpack users  guide*, siam, 1979.
c***routines called  zaxpy,zcopy,zdotu,zswap
c***end prologue  zsidi
      implicit none
      integer lda,n,job
      double complex a(lda,1),det(2),work(1)
      integer kpvt(1)
c
      double complex ak,akp1,akkp1,zdotu,d,t,temp
      double precision ten
      integer j,jb,k,km1,ks,kstep
      logical noinv,nodet
      double complex zdum
      double precision zabs1
      zabs1(zdum) = abs(dble(zdum)) + abs(imag(zdum))
c
c***first executable statement  zsidi
```

```
      noinv = mod(job,10) .eq. 0
      nodet = mod(job,100)/10 .eq. 0
c
      if (nodet) go to 100
         det(1) = (1.0d0,0.0d0)
         det(2) = (0.0d0,0.0d0)
         ten = 10.0d0
         t = (0.0d0,0.0d0)
         do 90 k = 1, n
            d = a(k,k)
c
c           check if 1 by 1
c
            if (kpvt(k) .gt. 0) go to 30
c
c              2 by 2 block
c              use det (d  t)  =  (d/t * c - t) * t
c                      (t  c)
c              to avoid underflow/overflow troubles.
c              take two passes through scaling.  use  t  for flag.
c
               if (zabs1(t) .ne. 0.0d0) go to 10
                  t = a(k,k+1)
                  d = (d/t)*a(k+1,k+1) - t
               go to 20
   10          continue
                  d = t
                  t = (0.0d0,0.0d0)
   20          continue
   30       continue
c
            det(1) = d*det(1)
            if (zabs1(det(1)) .eq. 0.0d0) go to 80
   40          if (zabs1(det(1)) .ge. 1.0d0) go to 50
                  det(1) = dcmplx(ten, 0.0d0)*det(1)
                  det(2) = det(2) - (1.0d0,0.0d0)
               go to 40
   50          continue
   60          if (zabs1(det(1)) .lt. ten) go to 70
                  det(1) = det(1)/dcmplx(ten, 0.0d0)
                  det(2) = det(2) + (1.0d0,0.0d0)
               go to 60
   70          continue
   80       continue
   90    continue
  100 continue
c
c     compute inverse(a)
c
      if (noinv) go to 230
         k = 1
  110    if (k .gt. n) go to 220
            km1 = k - 1
            if (kpvt(k) .lt. 0) go to 140
c
c              1 by 1
c
               a(k,k) = (1.0d0,0.0d0)/a(k,k)
               if (km1 .lt. 1) go to 130
                  call zcopy(km1,a(1,k),1,work,1)
                  do 120 j = 1, km1
                     a(j,k) = zdotu(j,a(1,j),1,work,1)
                     call zaxpy(j-1,work(j),a(1,j),1,a(1,k),1)
  120             continue
                  a(k,k) = a(k,k) + zdotu(km1,work,1,a(1,k),1)
  130          continue
               kstep = 1
            go to 180
  140       continue
c
c              2 by 2
c
               t = a(k,k+1)
               ak = a(k,k)/t
               akp1 = a(k+1,k+1)/t
               akkp1 = a(k,k+1)/t
               d = t*(ak*akp1 - (1.0d0,0.0d0))
               a(k,k) = akp1/d
               a(k+1,k+1) = ak/d
               a(k,k+1) = -akkp1/d
               if (km1 .lt. 1) go to 170
                  call zcopy(km1,a(1,k+1),1,work,1)
                  do 150 j = 1, km1
                     a(j,k+1) = zdotu(j,a(1,j),1,work,1)
                     call zaxpy(j-1,work(j),a(1,j),1,a(1,k+1),1)
  150             continue
                  a(k+1,k+1) = a(k+1,k+1)
     1                         + zdotu(km1,work,1,a(1,k+1),1)
                  a(k,k+1) = a(k,k+1) + zdotu(km1,a(1,k),1,a(1,k+1),1)
                  call zcopy(km1,a(1,k),1,work,1)
                  do 160 j = 1, km1
                     a(j,k) = zdotu(j,a(1,j),1,work,1)
                     call zaxpy(j-1,work(j),a(1,j),1,a(1,k),1)
  160             continue
                  a(k,k) = a(k,k) + zdotu(km1,work,1,a(1,k),1)
  170          continue
               kstep = 2
  180       continue
c
c           swap
c
            ks = iabs(kpvt(k))
            if (ks .eq. k) go to 210
               call zswap(ks,a(1,ks),1,a(1,k),1)
               do 190 jb = ks, k
                  j = k + ks - jb
                  temp = a(j,k)
                  a(j,k) = a(ks,j)
                  a(ks,j) = temp
  190          continue
               if (kstep .eq. 1) go to 200
                  temp = a(ks,k+1)
                  a(ks,k+1) = a(k,k+1)
                  a(k,k+1) = temp
  200          continue
  210       continue
            k = k + kstep
         go to 110
  220    continue
  230 continue
      return
      end

c*********************************************
c
      subroutine zsifa(a,lda,n,kpvt,info)
c***begin prologue  zsifa
c***date written   780814   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d2d1a
```

```
c***keywords  complex,factor,linear algebra,linpack,matrix,symmetric
c***author  bunch, j., (ucsd)
c***purpose  factors a complex symmetric matrix by elimination
c            (symmetric pivoting)
c***description
c
c     zsifa factors a complex symmetric matrix by elimination
c     with symmetric pivoting.
c
c     to solve  a*x = b , follow zsifa by zsisl.
c     to compute  inverse(a)*c , follow zsifa by zsisl.
c     to compute  determinant(a) , follow zsifa by zsidi.
c     to compute  inverse(a) , follow zsifa by zsidi.
c
c     on entry
c
c        a       complex(lda,n)
c                the symmetric matrix to be factored.
c                only the diagonal and upper triangle are used.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
c                the order of the matrix  a .
c
c     on return
c
c        a       a block diagonal matrix and the multipliers which
c                were used to obtain it.
c                the factorization can be written  a = u*d*trans(u)
c                where  u  is a product of permutation and unit
c                upper triangular matrices , trans(u) is the
c                transpose of  u , and  d  is block diagonal
c                with 1 by 1 and 2 by 2 blocks.
c
c        kvpt    integer(n)
c                an integer vector of pivot indices.
c
c        info    integer
c                = 0  normal value.
c                = k  if the k-th pivot block is singular.  this is
c                     not an error condition for this subroutine,
c                     but it does indicate that csisl or csidi may
c                     divide by zero if called.
c
c     linpack.  this version dated 08/14/78 .
c     james bunch, univ. calif. san diego, argonne nat. lab.
c
c     subroutines and functions
c
c     blas zaxpy,zswap,icamax
c     fortran abs,imag,max,real,sqrt
c***references  dongarra j.j., bunch j.r., moler c.b., stewart g.w.,
c                 *linpack users  guide*, siam, 1979.
c***routines called  zaxpy,zswap,icamax
c***end prologue  csifa
      implicit none
      integer lda,n,kpvt(1),info
      double complex a(lda,1)
c
      double complex ak,akm1,bk,bkm1,denom,mulk,mulkm1,t
      double precision absakk,alpha,colmax,rowmax
      integer imax,imaxp1,j,jj,jmax,k,km1,km2,kstep,icamax
      logical swap
      double complex zdum
      double precision zabs1
c
      zabs1(zdum) = abs(dble(zdum)) + abs(imag(zdum))
c
c     initialize
c
c     alpha is used in choosing pivot block size.
c***first executable statement  csifa
      alpha = (1.0d0 + sqrt(17.0d0))/8.0d0
c
      info = 0
c
c     main loop on k, which goes from n to 1.
c
      k = n
   10 continue
c
c        leave the loop if k=0 or k=1.
c
c     ...exit
         if (k .eq. 0) go to 200
         if (k .gt. 1) go to 20
            kpvt(1) = 1
            if (zabs1(a(1,1)) .eq. 0.0d0) info = 1
c     ......exit
            go to 200
   20    continue
c
c        this section of code determines the kind of
c        elimination to be performed.  when it is completed,
c        kstep will be set to the size of the pivot block, and
c        swap will be set to .true. if an interchange is
c        required.
c
         km1 = k - 1
         absakk = zabs1(a(k,k))
c
c        determine the largest off-diagonal element in
c        column k.
c
         imax = icamax(k-1,a(1,k),1)
         colmax = zabs1(a(imax,k))
         if (absakk .lt. alpha*colmax) go to 30
            kstep = 1
            swap = .false.
         go to 90
   30    continue
c
c           determine the largest off-diagonal element in
c           row imax.
c
            rowmax = 0.0d0
            imaxp1 = imax + 1
            do 40 j = imaxp1, k
               rowmax = max(rowmax,zabs1(a(imax,j)))
   40       continue
            if (imax .eq. 1) go to 50
               jmax = icamax(imax-1,a(1,imax),1)
               rowmax = max(rowmax,zabs1(a(jmax,imax)))
   50       continue
            if (zabs1(a(imax,imax)) .lt. alpha*rowmax) go to 60
               kstep = 1
               swap = .true.
            go to 80
```

```
   60       continue
            if (absakk .lt. alpha*colmax*(colmax/rowmax)) go to 70
               kstep = 1
               swap = .false.
            go to 80
   70       continue
               kstep = 2
               swap = imax .ne. km1
   80       continue
   90    continue
         if (max(absakk,colmax) .ne. 0.0d0) go to 100
c
c           column k is zero.  set info and iterate the loop.
c
            kpvt(k) = k
            info = k
         go to 190
  100    continue
         if (kstep .eq. 2) go to 140
c
c           1 x 1 pivot block.
c
            if (.not.swap) go to 120
c
c              perform an interchange.
c
               call zswap(imax,a(1,imax),1,a(1,k),1)
               do 110 jj = imax, k
                  j = k + imax - jj
                  t = a(j,k)
                  a(j,k) = a(imax,j)
                  a(imax,j) = t
  110          continue
  120       continue
c
c           perform the elimination.
c
            do 130 jj = 1, km1
               j = k - jj
               mulk = -a(j,k)/a(k,k)
               t = mulk
               call zaxpy(j,t,a(1,k),1,a(1,j),1)
               a(j,k) = mulk
  130       continue
c
c           set the pivot array.
c
            kpvt(k) = k
            if (swap) kpvt(k) = imax
         go to 190
  140    continue
c
c           2 x 2 pivot block.
c
            if (.not.swap) go to 160
c
c              perform an interchange.
c
               call zswap(imax,a(1,imax),1,a(1,k-1),1)
               do 150 jj = imax, km1
                  j = km1 + imax - jj
                  t = a(j,k-1)
                  a(j,k-1) = a(imax,j)
                  a(imax,j) = t
  150          continue
               t = a(k-1,k)
               a(k-1,k) = a(imax,k)
               a(imax,k) = t
  160       continue
c
c           perform the elimination.
c
            km2 = k - 2
            if (km2 .eq. 0) go to 180
               ak = a(k,k)/a(k-1,k)
               akm1 = a(k-1,k-1)/a(k-1,k)
               denom = 1.0d0 - ak*akm1
               do 170 jj = 1, km2
                  j = km1 - jj
                  bk = a(j,k)/a(k-1,k)
                  bkm1 = a(j,k-1)/a(k-1,k)
                  mulk = (akm1*bk - bkm1)/denom
                  mulkm1 = (ak*bkm1 - bk)/denom
                  t = mulk
                  call zaxpy(j,t,a(1,k),1,a(1,j),1)
                  t = mulkm1
                  call zaxpy(j,t,a(1,k-1),1,a(1,j),1)
                  a(j,k) = mulk
                  a(j,k-1) = mulkm1
  170          continue
  180       continue
c
c           set the pivot array.
c
            kpvt(k) = 1 - k
            if (swap) kpvt(k) = -imax
            kpvt(k-1) = kpvt(k)
  190    continue
         k = k - kstep
      go to 10
  200 continue
      return
      end

c**********************************************
c
      subroutine zsisl(a,lda,n,kpvt,b)
c***begin prologue  zsisl
c***date written   780814   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d2d1a
c***keywords  complex,linear algebra,linpack,matrix,solve,symmetric
c***author  bunch, j., (ucsd)
c***purpose  solves the complex symmetric system  a*x=b  using factors
c            from zsifa.
c***description
c
c     zsisl solves the complex symmetric system
c     a * x = b
c     using the factors computed by zsifa.
c
c     on entry
c
c        a       complex(lda,n)
c                the output from zsifa.
c
c        lda     integer
c                the leading dimension of the array  a .
c
c        n       integer
```

```
c                the order of the matrix  a .
c
c        kvpt    integer(n)
c                the pivot vector from zsifa.
c
c        b       complex(n)
c                the right hand side vector.
c
c     on return
c
c        b       the solution vector  x .
c
c     error condition
c
c        a division by zero may occur if  zsico  has set rcond .eq. 0.0
c        or  zsifa  has set info .ne. 0  .
c
c     to compute  inverse(a) * c  where  c  is a matrix
c     with  p  columns
c           call zsifa(a,lda,n,kvpt,info)
c           if (info .ne. 0) go to ...
c           do 10 j = 1, p
c              call zsisl(a,lda,n,kvpt,c(1,j))
c        10 continue
c
c     linpack.  this version dated 08/14/78 .
c     james bunch, univ. calif. san diego, argonne nat. lab.
c
c     subroutines and functions
c
c     blas zaxpy,zdotu
c     fortran abs
c***references  dongarra j.j., bunch j.r., moler c.b., stewart g.w.,
c                 *linpack users  guide*, siam, 1979.
c***routines called  zaxpy,zdotu
c***end prologue  zsisl
      implicit none
      integer lda,n,kpvt(1)
      double complex a(lda,1),b(1)
c
      double complex ak,akm1,bk,bkm1,zdotu,denom,temp
      integer k,kp
c
c     loop backward applying the transformations and
c     d inverse to b.
c
c***first executable statement  csisl
      k = n
   10 if (k .eq. 0) go to 80
         if (kpvt(k) .lt. 0) go to 40
c
c           1 x 1 pivot block.
c
            if (k .eq. 1) go to 30
               kp = kpvt(k)
               if (kp .eq. k) go to 20
c
c                 interchange.
c
                  temp = b(k)
                  b(k) = b(kp)
                  b(kp) = temp
   20          continue
c
c              apply the transformation.
c
               call zaxpy(k-1,b(k),a(1,k),1,b(1),1)
   30       continue
c
c           apply d inverse.
c
            b(k) = b(k)/a(k,k)
            k = k - 1
         go to 70
   40    continue
c
c           2 x 2 pivot block.
c
            if (k .eq. 2) go to 60
               kp = abs(kpvt(k))
               if (kp .eq. k - 1) go to 50
c
c                 interchange.
c
                  temp = b(k-1)
                  b(k-1) = b(kp)
                  b(kp) = temp
   50          continue
c
c              apply the transformation.
c
               call zaxpy(k-2,b(k),a(1,k),1,b(1),1)
               call zaxpy(k-2,b(k-1),a(1,k-1),1,b(1),1)
   60       continue
c
c           apply d inverse.
c
            ak = a(k,k)/a(k-1,k)
            akm1 = a(k-1,k-1)/a(k-1,k)
            bk = b(k)/a(k-1,k)
            bkm1 = b(k-1)/a(k-1,k)
            denom = ak*akm1 - 1.0d0
            b(k) = (akm1*bk - bkm1)/denom
            b(k-1) = (ak*bkm1 - bk)/denom
            k = k - 2
   70    continue
      go to 10
   80 continue
c
c     loop forward applying the transformations.
c
      k = 1
   90 if (k .gt. n) go to 160
         if (kpvt(k) .lt. 0) go to 120
c
c           1 x 1 pivot block.
c
            if (k .eq. 1) go to 110
c
c              apply the transformation.
c
               b(k) = b(k) + zdotu(k-1,a(1,k),1,b(1),1)
               kp = kpvt(k)
               if (kp .eq. k) go to 100
c
c                 interchange.
c
                  temp = b(k)
                  b(k) = b(kp)
                  b(kp) = temp
```

```
  100         continue
  110       continue
            k = k + 1
        go to 150
  120   continue
c
c           2 x 2 pivot block.
c
            if (k .eq. 1) go to 140
c
c              apply the transformation.
c
               b(k) = b(k) + zdotu(k-1,a(1,k),1,b(1),1)
               b(k+1) = b(k+1) + zdotu(k-1,a(1,k+1),1,b(1),1)
               kp = abs(kpvt(k))
               if (kp .eq. k) go to 130
c
c                 interchange.
c
                  temp = b(k)
                  b(k) = b(kp)
                  b(kp) = temp
  130         continue
  140       continue
            k = k + 2
  150   continue
      go to 90
  160 continue
      return
      end

c***********************************************
c
      subroutine zswap(n,cx,incx,cy,incy)
c***begin prologue  zswap
c***date written   791001   (yymmdd)
c***revision date  820801   (yymmdd)
c***category no.  d1a5
c***keywords  blas,complex,interchange,linear algebra,vector
c***author  lawson, c. l., (jpl)
c           hanson, r. j., (snla)
c           kincaid, d. r., (u. of texas)
c           krogh, f. t., (jpl)
c***purpose  interchange complex vectors
c***description
c
c                b l a s  subprogram
c    description of parameters
c
c     --input--
c        n  number of elements in input vector(s)
c       cx  complex vector with n elements
c     incx  storage spacing between elements of cx
c       cy  complex vector with n elements
c     incy  storage spacing between elements of cy
c
c     --output--
c       cx  input vector cy (unchanged if n .le. 0)
c       cy  input vector cx (unchanged if n .le. 0)
c
c     interchange complex cx and complex cy
c     for i = 0 to n-1, interchange  cx(lx+i*incx) and cy(ly+i*incy),
c     where lx = 1 if incx .gt. 0, else lx = (-incx)*n, and ly is
c     defined in a similar way using incy.
c***references  lawson c.l., hanson r.j., kincaid d.r., krogh f.t.,
c                  *basic linear algebra subprograms for fortran usage*,
c                  algorithm no. 539, transactions on mathematical
c                  software, volume 5, number 3, september 1979, 308-323
c***routines called  (none)
c***end prologue  zswap
c
      implicit none
      integer i,incx,incy,kx,ky,n,ns
      double complex cx(1),cy(1),ctemp
c***first executable statement  zswap
      if(n .le. 0)return
      if(incx.eq.incy.and.incx.gt.0) go to 20
      kx = 1
      ky = 1
      if(incx.lt.0) kx = 1+(1-n)*incx
      if(incy.lt.0) ky = 1+(1-n)*incy
          do 10 i = 1,n
          ctemp = cx(kx)
          cx(kx) = cy(ky)
          cy(ky) = ctemp
          kx = kx + incx
          ky = ky + incy
   10 continue
      return
   20 continue
      ns = n*incx
          do 30 i=1,ns,incx
          ctemp = cx(i)
          cx(i) = cy(i)
          cy(i) = ctemp
   30     continue
      return
      end
c
c***********************************************************************
c
      subroutine OnShl(enew, eold, Knew, Knew2, Dk2de)

c***********************************************************************
c     Onshl calc the on-shell momenta for every channel.
c     The sign of the momenta (or which sheet of energies) is also
c          decided in this roution by looking the relative position
c          of the current and previous momentum on the complex plane.
c     NOTE: Generally, Knew is complex because of complex physical mass
c           of Delta. To show 3D tmatrix struct, even the energy becomes
c           complex which causes complex onshell Knew.
c***********************************************************************

      implicit none

      integer  nch, i
      double precision mtotal, mdiff, xold, yold, xnew, ynew, rek, imk
      double precision zero, two, three, four

      integer  Nifty, Isheet
      double precision mpion, Mass
      double complex enew, eold, Knew(2), Knew2(2), kold2(2), Dk2de(2)

      common /control/ Nifty(15)
      common /mass/ mpion, Mass(2)
      common /sheet/ Isheet(2)

      parameter (zero = 0.0d0, two = 2.0d0, three = 3.0d0, four = 4.0d0)

      nch = Nifty(2)
```

```
      if (eold .eq. 0.0d0) eold = mpion + Mass(1)

c     this if-statement serves as initialization for nifty(4)=3
c     relativistic kimematics
         do 10000 i = 1, nch
            mtotal = mpion + Mass(i)
            mdiff = mpion - Mass(i)
            Kold2(i) = (eold-mtotal)*(eold-mdiff)
     &                 *(eold+mtotal)*(eold+mdiff)/(four*eold**2)
            Knew2(i) = (enew-mtotal)*(enew-mdiff)
     &                 *(enew+mtotal)*(enew+mdiff)/(four*enew**2)
            Dk2de(i) = (enew**four - mtotal**two*mdiff**two)
     &                 /(two*enew**three)
10000    continue

c the mometum k = sqrt(k**2), but the sign is undecided.
c we are try to decide the sign relative the the old k by looking where
c the straight line connecting Kold2 and Knew2 cross the 'cut'
c The 'cut' is choosen as from k=0 to +infinity alone positive real axis
c The real axis is belong to upper half plane

      do 10020 i = 1, nch
         xold = dble(Kold2(i))
         yold = imag(Kold2(i))
         xnew = dble(Knew2(i))
         ynew = imag(Knew2(i))
         if (yold*ynew .lt. zero) then
            if ((xold*ynew-xnew*yold)/(ynew-yold) .gt. zero)
     &         Isheet(i) = -Isheet(i)
         else if ((yold .eq. zero) .and. (xold .gt. zero)) then
            if (ynew .lt. zero) Isheet(i) = -Isheet(i)
         else if ((ynew .eq. zero) .and. (xnew .gt. zero)) then
            if (yold .lt. zero) Isheet(i) = -Isheet(i)
         endif
10020 continue

c compute channel momenta Knew(i)
      do 10030 i = 1, nch
         Knew(i) = sqrt(Knew2(i))
         rek = dble(Knew(i))
         imk = imag(Knew(i))
         if (Isheet(i) .gt. 0) then
c force 0 <= arg(Knew(i) < Pi
            if (imk .lt. zero) then
               Knew(i) = -Knew(i)
            else if ((imk .eq. zero) .and. (rek .lt. zero)) then
               Knew(i) = -Knew(i)
            endif
         else
c force Pi <= arg(Knew(i) < 2Pi
            if (imk .gt. zero) then
               Knew(i) = -Knew(i)
            else if ((imk .eq. zero) .and. (rek .gt. zero)) then
               Knew(i) = -Knew(i)
            endif
         endif
10030 continue

      eold = enew

      return
      end
c
c**********************************************************************
c
```

```
      subroutine VCoul(z1, z2, l, npts, p, w, ldv, vc)
c
c**********************************************************************
c     VCoul calculates coulomb potential matrix from l'th partial wave
c     between two particles z1, z2, in a way suitable for numerical
c     solution of Lippman-Schwinger Equations.
c
c     On input:
c     z1, z2: The charge number of two interaction particles
c     l:      The partial wave number
c     npts:   The number of grid points for Lippman-Schwinger Eq.
c     p, w:   The grid points and weight for LS Eq.
c     ldv:    The leading dimensions of vc
c     On output:
c     vc(ldv, ldv) The Coulomb potential.
c**********************************************************************
c
      integer z1, z2, l, npts, ldv
      double precision p(npts), w(npts), vc(ldv, 1)
      double precision alpha, pi, two, zero, charge2, temp
      integer ipt, jpt
c
      parameter (alpha = 7.29735307639485d-3, pi = 3.141592653589793d0)
      parameter (two = 2.0d0, zero = 0.0d0)
c
c**********************************************************************
c
      charge2 = z1*z2*alpha
c
      if (l .eq. 0) then
c     construct upper half of vc
         do jpt = 1, npts
            do ipt = 1, jpt-1
               vc(ipt, jpt) = charge2/(two*p(ipt)*p(jpt))
     &                *log((p(jpt)+p(ipt))/(p(jpt)-p(ipt)))
            enddo
         enddo
c     lower half
         do jpt = 1, npts
            do ipt = jpt+1, npts
               vc(ipt, jpt) = vc(jpt, ipt)
            enddo
         enddo
c     diagnal elements
         do jpt = 1, npts
            vc(jpt, jpt) = zero
            temp = zero
            do ipt = 1, npts
               temp = temp + w(ipt)*vc(jpt, ipt)
            enddo
            vc(jpt, jpt) = (charge2*pi*pi/(two*two*p(jpt))-temp)/w(jpt)
         enddo
      else
         write(8, *) 'only l = 0 has been implimented'
         stop
      endif
c
      return
      end

c
c**********************************************************************
c
      subroutine FfInt(npts,P,Zk,Uborn,Urop,Uct,Ucs,
     & zUborn,zUrop,zUct1,zUct2,zUct0,zUcs1,zUcs2,zUcs0)
```

```
c
c*************************************************************************
c     FfInt calc form factor for each grid k by num integration.
c     ff at onshell point is calc'ed in 2nd part.
c
c     npts:   the number of grid points
c     rmev:   radius of the bag in unit of 1/Mev
c     Uborn:  vertex form factor array in Born diagrams
c     Uct:  3D array of integral in time component
c     Ucs:  3D array of integral in space component
c     clmax: the maximal channel number
c     lmax: the maximal orbital angular momentum
c     ns2:  the normalization constant for ground state
c     z*: corresponding onshell point matrix
c     CDM and CBM share same matrix var, their actual values diff
c       because of diff inte upper limit
c*************************************************************************
c     LOCAL variables
      integer first,ich,jch,nch,ipt,jpt,l,nmax,lmax
      integer nx,NB,i,nbag,nfor,iq
      parameter (nmax=101,nx=100,nfor=200,lmax=1,NB=1000)
      double precision rmev, vs, ns2, q, tmp, Utmp
      double precision m1, m2, c1r, c2r, hbarc, Urop0(nmax),Vrop0(nmax)
      double precision P0(nmax), Uborn0(nmax), Uct0(nmax), Ucs0(nmax)
      double precision Q0(nmax), Vborn0(nmax), Vct0(nmax), Vcs0(nmax)
      double precision Y2b(nmax), Y2t(nmax), Y2s(nmax), Y2r(nmax)
      double precision j0s(NB), j1s(NB), jk(NB,64,0:2)
      double precision Xs(NB),Ws(NB), Xx(nx),Wx(nx), Xq(nfor),Wq(nfor)
      double precision fb(NB), ft(NB), fs(NB), fr(NB), ak, ak2
      double precision ic0, ic1, is0, is2, twoopi, mid, upper, FFgau
      double complex CFfgau, cjk(NB,2,0:2), Csbj, zq2
      double complex ztmp, zc0, zc1, zs0, zs2
c     SHARED variables
      integer Nifty, npts, Fidx
      double precision Uborn(64), Uct(64,64,0:1), Ucs(64,64,0:1)
      double precision Urop(64), P(64), zero, SBj
      double precision f0, mgb, alpha, c1, c2, Mbare, rcbm
      double complex Zk(2), zUborn(2), zUrop(2)
      double complex zUct1(2,64,0:1),zUcs1(2,64,0:1)
      double complex zUct2(64,2,0:1),zUcs2(64,2,0:1)
      double complex zUct0(2,2,0:1),zUcs0(2,2,0:1)
c
      parameter (zero=0.0d0,vs=2.0427869d0,hbarc=197.3d0)
      parameter (twoopi=0.63661977d0)
c
      common /control/ Nifty(15)
c     bag.h Version 1.4  contains the CCDM parameters. first 3 independent
      common /bag/ rcbm,f0,mgb,alpha,c1,c2,Mbare(3),Fidx(3)
c
      data first /1/
c*************************************************************************
c
      nch = Nifty(2)
      if (first .eq. 1) then
         open (1, file='cdmff', status='unknown')
         if (Nifty(1).ne.0) then
c               include Roper
                read(1,*) alpha, mgb, c1, c2, c1r, c2r
         else
c               donot include Roper
                read(1,*) alpha, mgb, c1, c2
         endif
c       if (nifty(5).eq.1) then
c         Compute hardron bare mass from Phatak's parameterization
c           Mbare(1) = (c1 - c2*alpha)*mgb
c           Mbare(2) = (c1 + c2*alpha)*mgb
c       else
c         use input file to scale m(nucleon) and m(delta) directly
c         for CBM nif14=3, mgb and alpha doesnot mean anything
          mgb   = (Mbare(2) + Mbare(1))/c1/2.0
          alpha = (Mbare(2) - Mbare(1))/c2/2.0/mgb
c       endif
        Mbare(3) = (c1r - c2r*alpha)*mgb
c
c
        if (Nifty(14).eq.1) then
c*****  Read in CDM form factor computed by Dr. Phatak's code. *****
c       Interpolate for gauss grids from 101 equally distributed points.
c       nmax is actual total num of momentum points.
        do ipt = 1, nmax
          if (Nifty(1).ne.0) then
          read (1,*) Q0(ipt),Vct0(ipt),Vborn0(ipt),Vcs0(ipt),Vrop0(ipt)
          else
          read (1,*) Q0(ipt),Vct0(ipt),Vborn0(ipt),Vcs0(ipt)
          endif
c       back to the original ff
          ak = (ipt-1.)/25.
c         ak2 = ak**2/4.
c         ak2 = -ak**2/4.
          ak2 = 0.
          Vct0(ipt)   = Vct0(ipt)   *exp(ak2)
          Vborn0(ipt) = Vborn0(ipt) *exp(ak2)
          Vcs0(ipt)   = Vcs0(ipt)   *exp(ak2)
          Vrop0(ipt)  = Vrop0(ipt)  *exp(ak2)
c         P0(ipt) = mgb * P0(ipt)
c         Ucs0(ipt) = Ucs0(ipt)/mgb
c         divided by hbarc??
        enddo
        close(1)
        endif
      endif
c
      if ((Nifty(14).eq.1) .and.
     & (first.eq.1 .or. m1.ne.Mbare(1) .or. m2.ne.Mbare(2))) then
        first = 0
        m1 = Mbare(1)
        m2 = Mbare(2)
        mgb   = (Mbare(2) + Mbare(1))/c1/2.0
        alpha = (Mbare(2) - Mbare(1))/c2/2.0/mgb
        Mbare(3) = (c1r - c2r*alpha)*mgb
c
c       convert to actual unit, then multiplied by pi ff to each leg
c       here rpi = 0.4fm
        do ipt = 1, nmax
          P0(ipt) = mgb * Q0(ipt)
          Ucs0(ipt) = Vcs0(ipt)/mgb
          ak2 = exp(-P0(ipt)**2/(40.*hbarc**2))
          Uct0(ipt) = Vct0(ipt) * ak2*ak2
          Uborn0(ipt) = Vborn0(ipt) * ak2
          Ucs0(ipt) = Ucs0(ipt) * ak2*ak2
          Urop0(ipt) = Vrop0(ipt) * ak2
c         write to the file 'debug'
          write(22,221) P0(ipt), Uborn0(ipt)
       enddo
 221   format(1x,'dg-ff1',2e12.3)

        call Spline(P0, Uct0, nmax, 0.d0, 0.0d0, Y2t)
        call Spline(P0, Uborn0, nmax, 0.d0, 0.0d0, Y2b)
        call Spline(P0, Ucs0, nmax, 0.d0, 0.0d0, Y2s)
        call Spline(P0, Urop0, nmax, 0.d0, 0.0d0, Y2r)
```

```
c
c       gauss grids for coordinate integration 0-4fm
        nbag = 400
        mid = 1.5/hbarc
        upper = 3.0/hbarc
        call Gauss(nbag, 1, mid, upper, Xs, Ws)

c       call Gauss(nbag, 0, zero, upper, Xs, Ws)
c       gauss grid for fourier transf from p to r space
c       call Gauss(nfor, 2, zero, 252.d0, Xq, Wq)
c       note that maximum Xq(nfor) must be less than maximum P0(nmax)
c       phatak' input range:0 -> 0.451e+4 MeV.
c       diff mgb may result in a smaller P0(nnmax).

        call Gauss(nfor, 0, zero, 4500.d0, Xq, Wq)

c       calc fourier transf of ff, convert p to r space
        write(8,*) 'Fourier transform: i : x(i) : fbi : ft : fs:fr'

        do i = 1, nbag
          fb(i) = zero
          ft(i) = zero
          fs(i) = zero
          fr(i) = zero
          do iq = 1, nfor
            tmp = Xq(iq)**2*Wq(iq)*SBj(0,Xq(iq)*Xs(i))
            call Splint(P0, Uborn0, Y2b, nmax, Xq(iq), Utmp)
                fb(i)=fb(i) + tmp*Utmp
            call Splint(P0, Uct0,   Y2t, nmax, Xq(iq), Utmp)
                ft(i)=ft(i) + tmp*Utmp
            call Splint(P0, Ucs0,   Y2s, nmax, Xq(iq), Utmp)
                fs(i)=fs(i) + tmp*Utmp
            call Splint(P0, Urop0,  Y2r, nmax, Xq(iq), Utmp)
                fr(i)=fr(i) + tmp*Utmp
          enddo
          fb(i) = fb(i)*twoopi
          ft(i) = ft(i)*twoopi
          fs(i) = fs(i)*twoopi
          fr(i) = fr(i)*twoopi
          if (Nifty(4).ne.3 .and. mod(i,10).eq.0)
     &          write(8,901) i, Xs(i),fb(i),ft(i),fs(i),fr(i)
        enddo
901     format(i4, 5e15.5)
c
        do i = 1, nbag
          do l = 0, lmax+1
            do ipt = 1, npts
              jk(i,ipt,l) = SBj(l, P(ipt)*Xs(i))
            enddo
          enddo
        enddo
c
c       Born(N and Delta) and Roper ff
        do ipt = 1, npts
          Uborn(ipt) = zero
          Urop(ipt) = zero
          do i = 1, nbag
            Uborn(ipt) = Uborn(ipt) + Xs(i)**2*Ws(i)
     &                   *fb(i)*jk(i,ipt,0)
            Urop(ipt) = Urop(ipt) + Xs(i)**2*Ws(i)
     &                   *fr(i)*jk(i,ipt,0)
          enddo
        enddo
c       Contact ff
        do jpt = 1, npts
        do ipt = 1, jpt
          ic0 = zero
          ic1 = zero
          is0 = zero
          is2 = zero
          do i = 1, nbag
            ic0=ic0 + Xs(i)**2*Ws(i)*ft(i)*jk(i,ipt,0)*jk(i,jpt,0)
            ic1=ic1 + Xs(i)**2*Ws(i)*ft(i)*jk(i,ipt,1)*jk(i,jpt,1)
c
            is0=is0 + Xs(i)**2*Ws(i)*fs(i)*jk(i,ipt,0)*jk(i,jpt,0)
            is2=is2 + Xs(i)**2*Ws(i)*fs(i)*jk(i,ipt,2)*jk(i,jpt,2)
          enddo
          Uct(ipt,jpt,0) = ic0
          Uct(ipt,jpt,1) = ic1
          Ucs(ipt,jpt,0) = zero
          Ucs(ipt,jpt,1) = P(ipt)*P(jpt)*(is0-is2)/3.0d0
        enddo
        enddo

c        do i = 1,nbag
c           write(22,223) i, xs(i),ws(i),fb(i),jk(i,1,0)
c           enddo
c        do i = 1, nmax
c           write(22,224) i, p0(i), uborn0(i)
c           enddo
c        do i = 1, nfor
c           write(22,225) i, xq(i), wq(i)
c           enddo
 222   format(1x,'dg-ff2',i5, 2e12.3)
 223   format(1x,'dg-ff3',i5, 4e12.3)
 224   format(1x,'dg-ff4',i5, 2e12.3)
 225   format(1x,'dg-ff4',i5, 2e12.3)
c
c
        else if (Nifty(14).eq.2 .and. first.eq.1) then
c***** GAUSS form factor *****
        first = 0
        do ipt = 1, npts
           UBorn(ipt) = Ffgau(Nifty(12), 2, (P(ipt)/mgb)**2)
        enddo
c
c       contact term ff by integrating over Xx=cos(angle)
        call Gauss(nx, 0, -1.0d0, 1.0d0, Xx, Wx)
c
        do jpt = 1, npts
         do ipt = 1, jpt
           Uct(ipt, jpt, 0) = zero
           Uct(ipt, jpt, 1) = zero
           Ucs(ipt, jpt, 0) = zero
           Ucs(ipt, jpt, 1) = zero
           do i = 1, nx
             q = sqrt(P(ipt)**2+P(jpt)**2-2.*P(ipt)*P(jpt)*Xx(i))
             Utmp = Ffgau(Nifty(12), 1, (q/mgb)**2)
             Uct(ipt,jpt,0) = Uct(ipt,jpt,0) + 0.5*Wx(i)*Utmp
             Uct(ipt,jpt,1) = Uct(ipt,jpt,1) + 0.5*Xx(i)*Wx(i)*Utmp
c
             Utmp = Ffgau(Nifty(12), 3, (q/mgb)**2)/mgb
             Ucs(ipt,jpt,1) = Ucs(ipt,jpt,1) + P(ipt)*P(jpt)*
     &                   0.25*(1.-Xx(i)**2)*Wx(i)*Utmp
           enddo
         enddo
        enddo
c
c
      else if (Nifty(14).eq.3 .and. first.eq.1) then
```

```
c***** CBM Born ff (Eq 2.28 of Veit ea,PRD33,1859(1986)) *****
       first = 0
c      calc bessel func at grid for later use.
       nbag = 200
       rmev = rcbm/hbarc
       call Gauss(nbag, 0, 0.0d0, rmev, Xs, Ws)
c
        do i = 1, nbag
         j0s(i) = SBj(0, vs*Xs(i)/rmev)
         j1s(i) = SBj(1, vs*Xs(i)/rmev)
         do l = 0, lmax
            do  ipt = 1, npts
               jk(i, ipt, l) = SBj(l, P(ipt)*Xs(i))
            enddo
         enddo
        enddo
c
c       Born vertex function
        do ipt = 1, npts
           tmp = P(ipt)*rmev
           Uborn(ipt) = SBj(1, tmp)/tmp * vs/(vs-1.d0)
        enddo
c
c       Normalization constant for ground state
        ns2 = 1.d0/(2.d0*SBj(0,vs)**2*rmev**3) * vs/(vs-1.d0)
c
c       contact interaction: time component Uct; space component Ucs
        do  l = 0, lmax
            do jpt = 1, npts
            do ipt = 1, jpt
               Uct(ipt, jpt, l) = zero
               Ucs(ipt, jpt, l) = zero
               do i = 1, nbag
                  Uct(ipt, jpt, l) = Uct(ipt, jpt, l) +
     &                Xs(i)*Xs(i)*(j0s(i)*j0s(i)+j1s(i)*j1s(i))
     &                *jk(i,ipt,l)*jk(i,jpt,l)*Ws(i)*ns2
c
                  Ucs(ipt,jpt,l) = Ucs(ipt,jpt,l)+Ws(i)*Xs(i)
     &               *2.d0*j0s(i)*j1s(i)*jk(i,ipt,l)*jk(i,jpt,l)*ns2
               enddo
            enddo
            enddo
        enddo
      endif
c
c      first = 0
c      endif
c     endif first, E-independent part, except Mbare changes
c
c************************************************
c****        On-Shell point ff matrix        ****
c************************************************
cwrite(*,1001) Zk(1),Zk(2)
1001    format('ffint: onshl momentum',4e12.3)
        Mbare(3) = (c1r - c2r*alpha)*mgb
        if (Nifty(14) .eq. 1) then
c       **** CDM ****
c       Set up onshell complex bessel func
        do i = 1, nbag
          do l = 0, lmax+1
            do ich = 1, nch
              cjk(i,ich,l) = Csbj(l,Zk(ich)*Xs(i))
            enddo
          enddo
        enddo
c
c       BORN(N and Delta) and Roper ff
        do ich = 1, nch
          zUborn(ich) = zero
          zUrop(ich) = zero
          do i = 1, nbag
            zUborn(ich) = zUborn(ich) + Xs(i)**2*Ws(i)
     &                  *fb(i)*cjk(i,ich,0)
            zUrop(ich) = zUrop(ich) + Xs(i)**2*Ws(i)
     &                  *fr(i)*cjk(i,ich,0)
          enddo
        enddo
c
c       CONTACT ff: k'=0, zUct1; k=0, zUct2
        do  ich = 1, nch
         do ipt = 1, npts
            zc0 = zero
            zc1 = zero
            zs0 = zero
            zs2 = zero
            do i = 1, nbag
             zc0=zc0 + Xs(i)**2*Ws(i)*ft(i)*cjk(i,ich,0)*jk(i,ipt,0)
             zc1=zc1 + Xs(i)**2*Ws(i)*ft(i)*cjk(i,ich,1)*jk(i,ipt,1)
c
             zs0=zs0 + Xs(i)**2*Ws(i)*fs(i)*cjk(i,ich,0)*jk(i,ipt,0)
             zs2=zs2 + Xs(i)**2*Ws(i)*fs(i)*cjk(i,ich,2)*jk(i,ipt,2)
            enddo
          zUct1(ich,ipt,0) = zc0
          zUct1(ich,ipt,1) = zc1
          zUct2(ipt,ich,0) = zUct1(ich,ipt,0)
          zUct2(ipt,ich,1) = zUct1(ich,ipt,1)
          zUcs1(ich,ipt,0) = zero
          zUcs2(ipt,ich,0) = zero
          zUcs1(ich,ipt,1) = Zk(ich)*P(ipt)*(zs0-zs2)/3.0d0
          zUcs2(ipt,ich,1) = zUcs1(ich,ipt,1)
         enddo
        enddo
c
c       onshell k'=k=k0, zUct0
        do ich = 1, nch
         do jch = 1, nch
           zc0 = zero
           zc1 = zero
           zs0 = zero
           zs2 = zero
           do i = 1, nbag
            zc0 = zc0 + Xs(i)**2*Ws(i)*ft(i)*cjk(i,ich,0)*cjk(i,jch,0)
            zc1 = zc1 + Xs(i)**2*Ws(i)*ft(i)*cjk(i,ich,1)*cjk(i,jch,1)
            zs0 = zs0 + Xs(i)**2*Ws(i)*fs(i)*cjk(i,ich,0)*cjk(i,jch,0)
            zs2 = zs2 + Xs(i)**2*Ws(i)*fs(i)*cjk(i,ich,2)*cjk(i,jch,2)
           enddo
           zUct0(ich,jch,0) = zc0
           zUct0(ich,jch,1) = zc1
           zUcs0(ich,jch,0) = zero
           zUcs0(ich,jch,1) = Zk(ich)*Zk(jch)*(zs0-zs2)/3.0d0
         enddo
        enddo
c
c*****  GAUSS form factor *****
        else if (Nifty(14) .eq. 2) then
          do ich = 1, nch
             zUborn(ich) = CFfgau(Nifty(12), 2, (Zk(ich)/mgb)**2)
          enddo
c
c        k'=0, zUct1; k=0, zUct2
```

```
          do  ich = 1, nch
           do ipt = 1, npts
             zUct1(ich, ipt, 0) = zero
             zUct1(ich, ipt, 1) = zero
             zUcs1(ich, ipt, 0) = zero
             zUcs1(ich, ipt, 1) = zero
            do i = 1, nx
              zq2 = Zk(ich)**2+P(ipt)**2-2.*Zk(ich)*P(ipt)*Xx(i)
              ztmp = CFfgau(Nifty(12), 1, zq2/mgb**2)
              zUct1(ich,ipt,0) = zUct1(ich,ipt,0) + 0.5*Wx(i)*ztmp
              zUct1(ich,ipt,1) = zUct1(ich,ipt,1) + 0.5*Xx(i)*Wx(i)*ztmp
c
              ztmp = CFfgau(Nifty(12), 3, zq2/mgb**2)/mgb
              zUcs1(ich,ipt,1) = zUcs1(ich,ipt,1) + Zk(ich)*P(ipt)*
     &                            0.25*(1.-Xx(i)**2)*Wx(i)*ztmp
            enddo
            zUct2(ipt,ich,0) = zUct1(ich,ipt,0)
            zUct2(ipt,ich,1) = zUct1(ich,ipt,1)
            zUcs2(ipt,ich,0) = zUcs1(ich,ipt,0)
            zUcs2(ipt,ich,1) = zUcs1(ich,ipt,1)
         enddo
        enddo
c
c       k'=k=k0, zUct0
        do ich = 1, nch
          do jch = 1, nch
            zUct0(ich, jch, 0) = zero
            zUct0(ich, jch, 1) = zero
            zUcs0(ich, jch, 0) = zero
            zUcs0(ich, jch, 1) = zero
           do i = 1, nx
             zq2 = Zk(ich)**2+Zk(jch)**2-2.*Zk(ich)*Zk(jch)*Xx(i)
             ztmp = Cffgau(Nifty(12), 1, zq2/mgb**2)
             zUct0(ich,jch,0) = zUct0(ich,jch,0) + 0.5*Wx(i)*ztmp
             zUct0(ich,jch,1) = zUct0(ich,jch,1) + 0.5*Xx(i)*Wx(i)*ztmp
c
             ztmp = Cffgau(Nifty(12), 3, zq2/mgb**2)/mgb
             zUcs0(ich,jch,1) = zUcs0(ich,jch,1) + Zk(ich)*Zk(jch)*
     &                           0.25*(1.-Xx(i)**2)*Wx(i)*ztmp
           enddo
          enddo
        enddo
c
        else if (Nifty(14) .eq. 3) then
c       **** CBM ****
          do ich = 1, nch
            ztmp = Zk(ich)*rmev
            zUborn(ich) = Csbj(1, ztmp)/ztmp * vs/(vs-1.d0)
          enddo
c
c       Set up onshell complex bessel func
        do l = 0, lmax
            do ich = 1, nch
              do i = 1, nbag
               ztmp = Zk(ich)*Xs(i)
               cjk(i,ich,l) = Csbj(l,ztmp)
              enddo
            enddo
        enddo
c
c        k'=0, zUct1; k=0, zUct2
         do  l = 0, lmax
          do  ich = 1, nch
            do ipt = 1, npts
              zUct1(ich, ipt, l) = zero
              zUct2(ipt, ich, l) = zero
              zUcs1(ich, ipt, l) = zero
              zUcs2(ipt, ich, l) = zero
              do i = 1, nbag
                zUct1(ich,ipt,l) = zUct1(ich,ipt,l)
     &                +Ws(i)*Xs(i)*Xs(i)*(j0s(i)**2+j1s(i)**2)
     &                *cjk(i,ich,l)*jk(i,ipt,l) * ns2
                zUcs1(ich,ipt,l) = zUcs1(ich,ipt,l)
     &                +Ws(i)*Xs(i)*(2.0d0*j0s(i)*j1s(i))
     &                *cjk(i,ich,l)*jk(i,ipt,l) * ns2
              enddo
              zUct2(ipt,ich,l) = zUct1(ich,ipt,l)
              zUcs2(ipt,ich,l) = zUcs1(ich,ipt,l)
            enddo
         enddo
        enddo
c
c       k'=k=k0, zUct0
        do  l = 0, lmax
            do ich = 1, nch
            do jch = 1, nch
              zUct0(ich, jch, l) = zero
              zUcs0(ich, jch, l) = zero
              do i = 1, nbag
               zUct0(ich,jch,l) = zUct0(ich,jch,l)
     &               +Ws(i)*Xs(i)*Xs(i)*(j0s(i)**2+j1s(i)**2)
     &               *cjk(i,ich,l)*cjk(i,jch,l) * ns2
               zUcs0(ich,jch,l) = zUcs0(ich,jch,l)
     &               +Ws(i)*Xs(i)*(2.0d0*j0s(i)*j1s(i))
     &               *cjk(i,ich,l)*cjk(i,jch,l) * ns2
              enddo
            enddo
            enddo
      enddo
c
      endif
      return
      end
c
c***********************************************************************
c
        double precision function Ffgau(nif12, num, p2)
c       Append gaussian form factor, real and complex case2
        integer nif12, num, first
        double precision ut0, ub0, us0, wt, wb, ws, t0, b0, s0, p2
        parameter (ut0=.9999969,ub0=.8580252,us0=1.471414)
        data first/1/
c
        if (first .eq. 1) then
          if (nif12 .eq. 1) then
            wt = 0.0
            wb = 0.0
            ws = 0.0
            t0=5.9528714
            b0=6.3627147
            s0=4.5191895
          else
            wt= 1.0/.7**2
            wb=1.0/.66**2
            ws=1.0/.86**2
            t0= 3.4103130
            b0= 3.4369686
            s0= 2.9414983
          endif
        first = 0
```

```
        endif
c
        if (num .eq. 1) then
          Ffgau = ut0*(1.0-wt*p2)*exp(-t0*p2)
        else if (num .eq. 2) then
          Ffgau = ub0*(1.0-wb*p2)*exp(-b0*p2)
        else
          Ffgau = us0*(1.0-ws*p2)*exp(-s0*p2)
        endif
        return
        end
c
c*********************************************************************
c
        double complex function Cffgau(nif12, num, p2)
        integer nif12, num, first
        double precision ut0, ub0, us0, wt, wb, ws, t0, b0, s0
        double complex p2
        parameter (ut0=.9999969,ub0=.8580252,us0=1.471414)
        data first/1/
c
        if (first .eq. 1) then
          if (nif12 .eq. 1) then
            wt = 0.0
            wb = 0.0
            ws = 0.0
            t0=5.9528714
            b0=6.3627147
            s0=4.5191895
          else
            wt= 1.0/.7**2
            wb=1.0/.66**2
            ws=1.0/.86**2
            t0= 3.4103130
            b0= 3.4369686
            s0= 2.9414983
          endif
        first = 0
        endif
c
        if (num .eq. 1) then
          Cffgau = ut0*(1.0-wt*p2)*exp(-t0*p2)
        else if (num .eq. 2) then
          Cffgau = ub0*(1.0-wb*p2)*exp(-b0*p2)
        else
          Cffgau = us0*(1.0-ws*p2)*exp(-s0*p2)
        endif
        return
        end

c
c*********************************************************************
c
      subroutine Gauss(npts,job,a,b,x,w)
c
c*********************************************************************
c     rescale rescals the Gauss-Legendre grid points and weights
c
c     npts     number of points
c     job = 0  rescalling uniformly between (a,b)
c           1  for integral (0,b) with 50% points inside (0, ab/(a+b))
c           2  for integral (a,inf) with 50% inside (a,b+2a), with
c              rational scaling
c     job = 3  for integral (a, inf) with 50% inside (a, b+2a), with
c              atan() scaling. (a+b) has to great than zero.
c
c     x, w     output grid points and weights.
c*********************************************************************
c
      integer npts,job,m,i,j
      double precision x(npts),w(npts),a,b,xi
      double precision t,t1,pp,p1,p2,p3,aj
      double precision eps,pi,zero,two,one,half,quarter,pio4
      parameter (pi = 3.14159265358979323846264338328d0, eps = 3.0d-15)
      parameter (zero=0.0d0,one=1.0d0,two=2.0d0)
      parameter (half=0.5d0,quarter=0.25d0)
c
c *** FIRST EXECTUABLE ***********************************************
c
      m=(npts+1)/2
      do 10020 i=1,m
         t=cos(pi*(i-quarter)/(npts+half))
10000 continue
         p1=one
         p2=zero
         aj=zero
         do 10010 j=1,npts
            p3=p2
            p2=p1
            aj=aj+one
            p1=((two*aj-one)*t*p2-(aj-one)*p3)/aj
10010    continue
         pp=npts*(t*p1-p2)/(t*t-one)
         t1=t
         t=t1-p1/pp
c
         if(abs(t-t1).gt.eps) goto 10000
c
         x(i)=-t
         x(npts+1-i)=t
         w(i)=two/((one-t*t)*pp*pp)
         w(npts+1-i)=w(i)
10020 continue
c
c*********************************************************************
c     rescale the grid points
c*********************************************************************
c
      if (job.eq.0) then
c     scale to (a,b) uniformly
         do 10030 i=1,npts
            x(i)=x(i)*(b-a)/two+(b+a)/two
            w(i)=w(i)*(b-a)/two
10030    continue
c
      elseif (job.eq.1) then
c     scale to (0,b) with 50% points inside (0,ab/(a+b))
         do 10040 i=1,npts
            xi=x(i)
            x(i)=a*b*(one+xi)/(b+a-(b-a)*xi)
            w(i)=w(i)*two*a*b*b/((b+a-(b-a)*xi)*(b+a-(b-a)*xi))
10040    continue
c
      elseif (job.eq.2) then
c     scale to (a,inf) with 50% points inside (a,b+2a)
         do 10050 i=1,npts
            xi=x(i)
            x(i)=(b*xi+b+a+a)/(one-xi)
            w(i)=w(i)*two*(a+b)/((one-xi)*(one-xi))
10050    continue
```

```
      else if (job .eq. 3) then
c     scale to (a, inf) with 50% points inside (a, b+2a)
         pio4 = atan(one)
         if (a+b .le. zero) stop 'Gauss: a+b less than zero'
         do 10060 i = 1, npts
            xi = x(i)
            x(i) = a+(a+b)*tan(pio4*(one+xi))
            w(i) = w(i)*(a+b)*pio4
     &           /(cos(pio4*(one+xi))*cos(pio4*(one+xi)))
10060    continue
      else
         stop 'Gauss: Wrong value of job'
      endif
c
      return
      end
c
c***********************************************************************
c
      subroutine PrtNif
c
c***********************************************************************
c     PrtNif prints out the Nifty's and their meaning.
c***********************************************************************
c
      integer Nifty, Up(15), i
      character*45 Cnif(0:32, 15)
      character*60 error
c
c     control.h Version 1.1
      common /control/ Nifty(15)
c
c***********************************************************************
c
      data error /'XXXXXXXXXXXXXXX BAD VALUE XXXXXXXXXXXXXXXXX'/
      data up /2,2,2,4,1,6,2,2,3,5,2,2,1,3,1/
c
c Nifty(1)
      data Cnif(0, 1) /'Do not include Roper'/
      data Cnif(1, 1) /'Include Roper through d-Born only'/
      data Cnif(2, 1) /'Include Roper through d,x-Born'/
c
c Nifty(2)
      data Cnif(1, 2) /'PiN single channel problem'/
      data Cnif(2, 2) /'PiN + PiDelta coupled channel problem'/
c
c Nifty(3)
      data Cnif(1, 3) /'Relativistic Kinematics'/
      data Cnif(2, 3) /'Non-Relativistic Kinematics'/
c
c Nifty(4)
      data Cnif(1, 4) /'Compute potential on a mesh of k-space'/
      data Cnif(2, 4) /'Compute T-matrix on complex energy plane'/
      data Cnif(3, 4) /'Fit potential parameters to exp. data'/
      data Cnif(4, 4) /'Search zero of det(1-GV) on complex plane'/
c
c Nifty(5)
      data Cnif(0, 5) /'Scaling m(N) and m(Delta)'/
      data Cnif(1, 5) /'Scaling mgb and alpha'/
c
c Nifty(6)
      data Cnif(1, 6) /'Only eigen channel S11'/
      data Cnif(2, 6) /'Only eigen channel S31'/
      data Cnif(3, 6) /'Only eigen channel P11'/
      data Cnif(4, 6) /'Only eigen channel P31'/
      data Cnif(5, 6) /'Only eigen channel P13'/
      data Cnif(6, 6) /'Only P33 or all 6 channels'/
c
c Nifty(7)
      data Cnif(0, 7) /'Born Approximation'/
      data Cnif(1, 7) /'Full Lippman-Schwinger Eq'/
c
c Nifty(8)
      data Cnif(1, 8) /'amp, controled by Nifty(11)'/
      data Cnif(2, 8) /'Comput obs(phase shfts, x-sect, scat length'/
c
c Nifty(9)
      data Cnif(0, 9) /'No cross section'/
      data Cnif(1, 9) /'compute total X-sect'/
      data Cnif(2, 9) /'compute diff. X-sect and analy power'/
c
c Nifty(10)
      data Cnif(0, 10) /'Zero potential'/
      data Cnif(1, 10) /'Including s channel diag only'/
      data Cnif(2, 10) /'Including s + u channel diag'/
      data Cnif(3, 10) /'Including s + u + t channel diag'/
      data Cnif(4, 10) /'Switch off space comp of t chl'/
      data Cnif(5, 10) /'Switch off time comp of t chl'/

c Nifty(11)
      data Cnif(0, 11) /'w nif8=1, calc PiNN Cpling Const'/
      data Cnif(1, 11) /'w nif8=1, 3D Tmatrix vs complex E'/
      data Cnif(2, 11) /'w nif8=1, 3D inverse det(1-vg) vs E'/

c Nifty(12)
      data Cnif(1, 12) /'Only if nif14=2, simple gaussian'/
      data Cnif(2, 12) /'Only if nif14=2, 1-node gaussian'/

c Nifty(13)
      data Cnif(0, 13) /'Without Coulomb'/
      data Cnif(1, 13) /'With Coulomb'/

c Nifty(14)
      data Cnif(1, 14) /'Use CDM form factor'/
      data Cnif(2, 14) /'Use Gaussian form factor'/
      data Cnif(3, 14) /'Use CBM form factor'/
c
c Nifty(15)
      data Cnif(0, 15) /'No debug info'/
      data Cnif(1, 15) /'Print debug info'/
c
c***********************************************************************
c
      write(8, *)
      do 10000 i = 1, 15
         if (Nifty(i) .gt. Up(i)) then
            write(8, 900) i, Nifty(i), error
            stop 'prtnif'
         else
            write(8, 900) i, Nifty(i), Cnif(Nifty(i), i)
         endif
10000 continue
c
 900  format(1x, ' Nifty(', i2, ') = ', i3, 3x, a)
      return
      end
c
c***********************************************************************
c
      subroutine SetUp
```

```
c
c***********************************************************************
c     SetUp sets up the masses for every channels, and the
c     SU(2) coupling constants between the channels.
c***********************************************************************

      double precision Cplv, Cplct, Cplcs
      double precision zero, one, two, three, four, five, sqrt2, sqrt3
      double precision proton, neutron, delta, pip, pi0, pim
      double precision mpion, Mass

c     mass.h Version 1.1 masses of pion, nucleon and delta
      common /mass/ mpion, Mass(2)
      common /couple/ Cplv(3,3),Cplct(2,2,2),Cplcs(2,2,2)

      parameter (zero=0.0d0,one=1.0d0,two=2.0d0,three=3.0d0,four=4.0d0)
      parameter (five=5.0d0,sqrt2=1.41421356237d0,sqrt3=1.73205080757d0)
c     taken from 'Rev of Part Prop', Phys Let B, Vol 239. 12 April 1990
      parameter (proton=938.272d0, neutron=939.565d0, delta=1232.d0)
      parameter (pip=139.5675d0,pi0=134.9739d0,pim=139.5675d0)
c
c***********************************************************************
c
c use isospin avarage masses
c      mpion = (pip + pi0 + pim)/three
c      use mpion (from input in kpinit.f) to test chiral symmetry
c
      Mass(1) = (proton + neutron)/two
      Mass(2) = delta
c
c     now initialize the SU(2) coupling constants
c
c     Cplv(2,3) is for s and u channel vervex
c     1 denotes for N; 2 for Delta; 3 for Roper
      Cplv(1,1) = five
      Cplv(1,2) = two * sqrt2
      Cplv(1,3) = five/sqrt3
      Cplv(2,1) = four * sqrt2
      Cplv(2,2) = five
      Cplv(2,3) = four*sqrt2/sqrt3
      Cplv(3,1) = Cplv(1,3)
      Cplv(3,2) = Cplv(2,3)
      Cplv(3,3) = zero
c
c     Cplct(2,2,2)
c
      Cplct(1,1,1) = -1.d0
      Cplct(1,2,1) = zero
      Cplct(2,1,1) = zero
      Cplct(2,2,1) = -2.5d0
      Cplct(1,1,2) =  0.5d0
      Cplct(1,2,2) = zero
      Cplct(2,1,2) = zero
      Cplct(2,2,2) = -1.d0
c
c     Cplcs(2,2,2)
c
      Cplcs(1,1,1) = -five/three
      Cplcs(1,2,1) = -two*sqrt2/three
      Cplcs(2,1,1) = Cplcs(1,2,1)
      Cplcs(2,2,1) = 5.d0*sqrt(10.d0)/6.d0
      Cplcs(1,1,2) = 5.d0/6.d0
      Cplcs(1,2,2) = -2.d0*sqrt(5.d0)/3.d0
      Cplcs(2,1,2) = Cplcs(1,2,2)
      Cplcs(2,2,2) = sqrt(10.d0)/3.d0
      return
      end
c
c***********************************************************************
c
      subroutine DiagSU(npts,P,ze,Zk,Wm,Wmst,Wm0,Wm0st
     &                 ,Mbare,f0,Uborn,zUborn,Urop,zUrop,Vsuper)

c     DiagSU calc the potential contributed by s and u channel diagrams
c***********************************************************************

      implicit none
c     LOCAL variables
      integer nch, npts1, cl
c     SHARED variables
      integer Nifty, npts, isXbon
      double precision P(64), Wm(64), Wmst(64), Uborn(64), Urop(64)
      double precision Cplv, Cplct, Cplcs, f0, Mbare(3), mpion, Mass
      double complex ze, Zk(2), zUborn(2), zUrop(2)
      double complex Wm0(2), Wm0st(2), Vsuper(128,128,6)

      common /control/ Nifty(15)
      common /couple/ Cplv(3,3),Cplct(2,2,2),Cplcs(2,2,2)
      common /mass/ mpion, Mass(2)

c***********************************************************************
c
      nch = Nifty(2)
      npts1 = npts + 1

c Func(Vsuper) is only initialized to zero once in VBag. Therefore
c everytime the PotFcn is called, Vsuper is updated (added on).
c
      if (Nifty(10) .ge. 1) then
c       s-channel diagram contributes to P11(N), P33(Delta) waves only
c       P11(N):
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.3)) then
          isXbon = 0
          cl = 3
          call PotFcn(isXbon, cl, npts, ze, P, Zk, Wm, Wmst,
     &  Wm0, Wm0st, Mbare, f0, Uborn, zUborn, Urop,zUrop,Vsuper)
        endif
c
c       P33(Delta):
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.6)) then
          isXbon = 0
          cl = 6
          call PotFcn(isXbon, cl, npts, ze, P, Zk, Wm, Wmst,
     &  Wm0, Wm0st, Mbare, f0, Uborn, zUborn, Urop,zUrop,Vsuper)
        endif
      endif

      if (Nifty(10) .ge. 2) then
c       u-channel diagram contributes to all 4 p waves, isXbon = 1
c       P11, cl = 3
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.3)) then
          call PotFcn(1, 3, npts, ze, P, Zk, Wm, Wmst, Wm0, Wm0st,
     &    Mbare, f0, Uborn, zUborn, Urop, zUrop, Vsuper)
        endif

c       P31, cl = 4
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.4)) then
          call PotFcn(1, 4, npts, ze, P, Zk, Wm, Wmst, Wm0, Wm0st,
     &    Mbare, f0, Uborn, zUborn, Urop, zUrop, Vsuper)
        endif
```

```
c       P13, cl = 5
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.5)) then
          call PotFcn(1, 5, npts, ze, P, Zk, Wm, Wmst, Wm0, Wm0st,
     &    Mbare, f0, Uborn, zUborn, Urop, zUrop, Vsuper)
        endif

c       P33, cl = 6
        if ((Nifty(4).ne.4) .or. (Nifty(6).eq.6)) then
          call PotFcn(1, 6, npts, ze, P, Zk, Wm, Wmst, Wm0, Wm0st,
     &    Mbare, f0, Uborn, zUborn, Urop, zUrop, Vsuper)
        endif
      endif
      return
      end
c
c**********************************************************************
c
      subroutine DiagT(npts,P,ze,Zk,Wm,Wmst,Wm0,Wm0st,f0,Uct,Ucs,
     &          zUct1,zUct2,zUct0,zUcs1,zUcs2,zUcs0,Vsuper)
c
c     DiagT calc the potential contributed by the contact diagram
c**********************************************************************
c
      implicit none
c     LOCAL variables
      integer nch, npts1, ich, jch, cl,clmin,clmax, j, iso
      integer first,ichpt,jchpt,ipt,jpt,l
      double precision pi, zero, one, two, eight, norm, jp
c     SHARED variables
      integer Nifty, npts
      double precision P(64), Wm(64), Wmst(64), A(2,2,2)
      double precision Uct(64,64,0:1), Ucs(64,64,0:1)
      double precision Cplv, Cplct, Cplcs, f0, mpion, Mass
      double complex ze, Zk(2)
      double complex zUct1(2,64,0:1),zUcs1(2,64,0:1)
      double complex zUct2(64,2,0:1),zUcs2(64,2,0:1)
      double complex zUct0(2,2,0:1),zUcs0(2,2,0:1)
      double complex Wm0(2), Wm0st(2), Vsuper(128,128,6)

      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, eight=8.0d0)
      parameter (pi=3.14159265358979323846d0)

      common /control/ Nifty(15)
      common /couple/ Cplv(3,3),Cplct(2,2,2),Cplcs(2,2,2)
      common /mass/ mpion, Mass(2)

      data first/1/
c*********************************
      nch = Nifty(2)
      npts1 = npts + 1
      norm = 1.d0/(eight*pi)
c      lmax = 1
      clmax = Nifty(6)
      clmin = 1
      if (Nifty(4) .eq. 4) clmin = clmax
c
c     Order of PW: P11,P31,P13,P33.
c     add a switch. if nif10=4 switch off space component
c                   nif10=5 switch off time component
      if (Nifty(10).eq.3 .or. Nifty(10).eq.4) then

c ****************************************************
c ***  add time component of contact interaction  ***
c     orbital angular momentum
      do cl = clmin, clmax
        if (cl .le. 2) then
          l = 0
        else if (cl .ge. 3) then
          l = 1
        endif
c       isospin
        if (cl.eq.1 .or. cl.eq.3 .or. cl.eq.5) then
          iso = 1
        else
          iso = 2
        endif

c       time component don't contribute to transition elements
         do 101 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (ich-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &            +norm*Cplct(ich,ich,iso)*Uct(ipt,jpt,l)
     &            *(Wm(ipt)+Wm(jpt))/(f0*f0*Wmst(ipt)*Wmst(jpt))
               enddo
               ichpt = ich*npts1
               Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplct(ich,ich,iso)*zUct1(ich,jpt,l)
     &         *(Wm0(ich)+Wm(jpt))/(f0*f0*Wm0st(ich)*Wmst(jpt))
            enddo
c
            jchpt = ich*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplct(ich,ich,iso)*zUct2(ipt,ich,l)
     &         *(Wm(ipt)+Wm0(ich))/(f0*f0*Wmst(ipt)*Wm0st(ich))
            enddo
            ichpt = ich*npts1
            Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplct(ich,ich,iso)*zUct0(ich,ich,l)
     &         *(Wm0(ich)+Wm0(ich))/(f0*f0*Wm0st(ich)*Wm0st(ich))
 101     continue
      enddo

      endif

      if (Nifty(10).eq.3 .or. Nifty(10).eq.5) then
c ******************************************************
c ***  add spatial component of contact interaction  ***
c       first initialize coef A(ich, jch, j)
c       angular momentu must be 1(p); j=1(1/2) and 2(3/2)
        if (first .eq. 1) then
          do j = 1, 2
            jp = j - 0.5d0
            A(1,1,j) =  2.d0*(0.75 + 2.d0 - jp*(jp+1))
            A(1,2,j) =  (jp+0.5d0) * sqrt(0.5d0*(jp+3.5d0)*(2.5d0-jp))
            A(2,1,j) =  A(1,2,j)
            A(2,2,j) =  -sqrt(0.4d0)*(5.75d0-jp*(jp+1))
          enddo
          first = 0
        endif
c

      l = 1
      clmin = 3
      if (nifty(4).eq.4) clmin = clmax
```

```
      do cl = clmin, clmax
c       total angular momentum
        if (cl.eq.3 .or. cl.eq.4) then
          j = 1
        else
          j = 2
        endif
c       isospin
        if (cl.eq.3 .or. cl.eq.5) then
           iso = 1
        else
           iso = 2
        endif
c
c
        do 103 jch = 1, nch
        do 104 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &            +norm*Cplcs(ich,jch,iso)*Ucs(ipt,jpt,l)
     &            *A(ich,jch,j)/(f0*f0*Wmst(ipt)*Wmst(jpt))
               enddo
               ichpt = ich*npts1
               Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplcs(ich,jch,iso)*zUcs1(ich,jpt,l)
     &         *A(ich,jch,j)/(f0*f0*Wm0st(ich)*Wmst(jpt))
            enddo
c
            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplcs(ich,jch,iso)*zUcs2(ipt,jch,l)
     &         *A(ich,jch,j)/(f0*f0*Wmst(ipt)*Wm0st(jch))
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Vsuper(ichpt,jchpt,cl) = Vsuper(ichpt,jchpt,cl)
     &         +norm*Cplcs(ich,jch,iso)*zUcs0(ich,jch,l)
     &         *A(ich,jch,j)/(f0*f0*Wm0st(ich)*Wm0st(jch))
  104    continue
  103    continue
        enddo
c
      endif
      return
      end
c
c*********************************************************************
c
      function Csbj(l, z)
c*********************************************************************
c     Spherical Bessel function for complex z and integer l = 0, 1, 2
c*********************************************************************
c
      implicit none
      integer l
      double precision a1, a3, a5, a7, a9, a11
      double precision a2, a4, a6, a8, a10, a12
      double precision tiny, quarter, one, three, x, y
      double complex z, z2, zero, Csbj
c
      parameter (zero = (0.0d0, 0.0d0))
      parameter (tiny = 0.01d0, quarter = 0.25d0)
      parameter (one = 1.0d0, three = 3.0d0)
      parameter (a1=  0.3333333333333333333333333333d00)
      parameter (a3= -0.3333333333333333333333333333d-1)
      parameter (a5=  0.1190476190476190476190476190d-2)
      parameter (a7= -0.2204585537918871252204585538d-4)
      parameter (a9=  0.2505210838544171877505210839d-6)
      parameter (a11=-0.1927085260418593751927085260d-8)
      parameter (a2 = 0.666666666666666666666666666667d-1)
      parameter (a4 =-0.476190476190476190476190476190d-2)
      parameter (a6 = 0.132275132275132275132275132275d-3)
      parameter (a8 =-0.200416867083533750200416867084d-5)
      parameter (a10= 0.192708526041859375192708526042d-7)
      parameter (a12=-0.128472350694572916795139017361d-9)
c
      if (l .eq. 0) then
         if (z .eq. zero) then
            Csbj = one
         else
            Csbj = sin(z)/z
         endif
      else if (l .eq. 1) then
         x = dble(z)
         y = imag(z)
         z2 = z*z
         if ((x*x+y*y) .lt. tiny) then
            Csbj = z*(a1+z2*(a3+z2*(a5+z2*(a7+z2*(a9+a11*z2)))))
         else
            Csbj = sin(z)/z2-cos(z)/z
         endif
      else if (l .eq. 2) then
         z2=z*z
         x = dble(z)
         y = imag(z)
         if ((x*x+y*y) .lt. quarter) then
            Csbj=z2*(a2+z2*(a4+z2*(a6+z2*(a8+z2*(a10+a12*z2)))))
         else
            Csbj=sin(z)/z*(three/z2-one)-three*cos(z)/z2
         endif
      else
         write(8, 900) l
         stop 'Csbj: l value out of range'
      endif
c
  900 format('0', 'l value = ', i2, ', only 0, 1, 2 allowed')
c
      return
      end
c
c*********************************************************************
c
      subroutine Dummy()
c
c*********************************************************************
c     Dummy is a dummy routine (as the name says). It is used as a
c     place holder and should not be actually called. It sugests an
c     error input on the caller routine when dummy is called
c*********************************************************************
c
      implicit none
c
c*********************************************************************
c
      stop 'Error: Dummy routine being called'
```

```
c
      return
      end
c
c*********************************************************************
c
      function EngCom(job, k, mt, mp)
c
c*********************************************************************
c     function EngCom returns the COM energy for given incident
c     lab momentum k, where mp is the mass of projectile and mt
c     is the mass of target
c
c     if job = 0, relativistic formular is used
c     else  non-relativistic formular.
c
c     Ref. Eq. 1.35 and 1.37, page 13 of QM II by Landau
c*********************************************************************
c
      implicit none
      integer job
      double precision two, EngCom, k, mp, mt, mp2
c
      parameter (two = 2.0d0)
c*********************************************************************
c
      mp2 = mp*mp
      if (job .eq. 0) then
         EngCom = sqrt(mt*mt + mp2 + two*mt*sqrt(mp2+k*k))
      else
         write(8, *) 'not yet implimented'
         stop
      endif
c
      return
      end
c
c*********************************************************************
c
      subroutine LegendreP(l, m, n, x, plm)
c
c*********************************************************************
c     LegendreP computes the associated legendre polynomial p_l^m
c     for the array x(0:n), and return the results in plm(0:n)
c*********************************************************************

      integer l, m, n, i
      double precision x(0:n), plm(0:n)
      double precision half, one, three

      parameter (half = 0.5d0, one = 1.0d0, three = 3.0d0)
c*****************************************************

      if (m .eq. 0) then
         if (l .eq. 0) then
            do 10000 i = 0, n
               plm(i) = one
10000       continue
         else if (l .eq. 1) then
            do 10010 i = 0, n
               plm(i) = x(i)
10010       continue
         else if (l .eq. 2) then
            do 10020 i = 0, n
               plm(i) = half*(three*x(i)*x(i)-one)
10020       continue
         else
            write(8, *)
     &         'LegendreP: l can only be 0, 1, 2 when m = 0, stop'
            stop
         endif
      else if (m .eq. 1) then
         if (l .eq. 1) then
            do 10110 i = 0, n
               plm(i) = -sqrt(one-x(i)*x(i))
10110       continue
         else if (l .eq. 2) then
            do 10120 i = 0, n
               plm(i) = -three*x(i)*sqrt(one-x(i)*x(i))
10120       continue
         else
            write(8, *)
     &         'LegendreP: l can only be 1, 2 when m = 1, stop'
            stop
         endif
      endif

      return
      end
c
c*********************************************************************
c
      subroutine PotFcn(isXbon, cl, npts, ze, P, Zk, Wm, Wmst,
     &Wm0, Wm0st, Mbare, f0, Uborn, zUborn, Urop, zUrop, Func)

c     Func is a pot func for a specific channel and intermediate baryon
c     for born and x-born diagrams
c*********************************************************************
c
      implicit none
c     LOCAL variables
      integer first, nch, npts1
      integer ich, jch, ipt, jpt, ichpt, jchpt, ib, cl
      double precision pi, zero, one, two, norm
c     SHARED variables
      integer Nifty, npts, isXbon
      double precision P(64), Uborn(64), Wm(64), Wmst(64), Urop(64)
      double precision Cplv, Cplct, Cplcs, A(2,2,6,2), f0, Mbare(3)
      double complex ze, Zk(2), Wm0(2), Wm0st(2)
      double complex zUborn(2), zUrop(2), Func(128,128,6)

      parameter (zero=0.0d0, one=1.0d0, two=2.0d0)
      parameter (pi=3.14159265358979323846d0)

      common /control/ Nifty(15)
      common /couple/ Cplv(3,3),Cplct(2,2,2),Cplcs(2,2,2)

      data first /1/
c***************************
c
      norm = one/(48.0d0*pi)
      nch = Nifty(2)
      npts1 = npts+1

c ???    initialize Func
c      do jch = 1, 128
c         do ich = 1, 128
c            Func(ich,jch,cl) = 0.0d0
c         enddo
c      enddo
```

```
      if (isXbon .eq. 0) then
         if (cl .eq. 3) then
             ib = 1
             else if (cl .eq. 6) then
                 ib = 2
                 endif
         do 101 jch = 1, nch
         do 102 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &            +norm*Cplv(ich,ib)*Cplv(jch,ib)*Uborn(ipt)*Uborn(jpt)
     &            *P(ipt)*P(jpt)/(f0*f0*Wmst(ipt)*Wmst(jpt)
     &            *(ze-Mbare(ib)))
               enddo
               ichpt = ich*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*zUborn(ich)*Uborn(jpt)
     &         *Zk(ich)*P(jpt)/(f0*f0*Wm0st(ich)*Wmst(jpt)
     &         *(ze-Mbare(ib)))
            enddo
c
            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*Uborn(ipt)*zUborn(jch)
     &         *P(ipt)*Zk(jch)/(f0*f0*Wmst(ipt)*Wm0st(jch)
     &         *(ze-Mbare(ib)))
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*zUborn(ich)*zUborn(jch)
     &         *Zk(ich)*Zk(jch)/(f0*f0*Wm0st(ich)*Wm0st(jch)
     &         *(ze-Mbare(ib)))
  102    continue
  101    continue
c
        if (Nifty(1).ne.0 .and. cl.eq.3) then
c       including Roper contribution to P11 through direct Born
         ib = 3
         do 201 jch = 1, nch
         do 202 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &            +norm*Cplv(ich,ib)*Cplv(jch,ib)*Urop(ipt)*Urop(jpt)
     &            *P(ipt)*P(jpt)/(f0*f0*Wmst(ipt)*Wmst(jpt)
     &            *(ze-Mbare(ib)))
               enddo
               ichpt = ich*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*zUrop(ich)*Urop(jpt)
     &         *Zk(ich)*P(jpt)/(f0*f0*Wm0st(ich)*Wmst(jpt)
     &         *(ze-Mbare(ib)))
            enddo
c
            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*Urop(ipt)*zUrop(jch)
     &         *P(ipt)*Zk(jch)/(f0*f0*Wmst(ipt)*Wm0st(jch)
     &         *(ze-Mbare(ib)))
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &         +norm*Cplv(ich,ib)*Cplv(jch,ib)*zUrop(ich)*zUrop(jch)
     &         *Zk(ich)*Zk(jch)/(f0*f0*Wm0st(ich)*Wm0st(jch)
     &         *(ze-Mbare(ib)))
  202    continue
  201    continue
        endif

c*************
      else
c       compute spin-isospin coefficient just one time
        if (first .eq. 1) then
c       P11: cl = 3
        call SpIso(0.5d0, 0.5d0, 3, A)
c       P31: cl = 4
        call SpIso(1.5d0, 0.5d0, 4, A)
c       P13: cl = 5
        call SpIso(0.5d0, 1.5d0, 5, A)
c       P33: cl = 6
        call SpIso(1.5d0, 1.5d0, 6, A)
        first = 0
        endif
c
c     sum over intermediate baryon, notice the order change of Cplv's index
      do ib = 1, 2
         do 103 jch = 1, nch
         do 104 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &              + A(ich,jch,cl,ib)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &              * Uborn(ipt)*Uborn(jpt)*P(ipt)*P(jpt)
     &/(Wmst(ipt)*Wmst(jpt)*f0*f0*(ze-Mbare(ib)-Wm(ipt)-Wm(jpt)))
               enddo
               ichpt = ich*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &           + A(ich,jch,cl,ib)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &           * zUborn(ich)*Uborn(jpt)*Zk(ich)*P(jpt)
     &/(Wm0st(ich)*Wmst(jpt)*f0*f0*(ze-Mbare(ib)-Wm0(ich)-Wm(jpt)))
            enddo
c
            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &           + A(ich,jch,cl,ib)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &           * Uborn(ipt)*zUborn(jch)*P(ipt)*Zk(jch)
     &/(Wmst(ipt)*Wm0st(jch)*f0*f0*(ze-Mbare(ib)-Wm(ipt)-Wm0(jch)))
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &        + A(ich,jch,cl,ib)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &        * zUborn(ich)*zUborn(jch)*Zk(ich)*Zk(jch)
```

```
     &/(Wm0st(ich)*Wm0st(jch)*f0*f0*(ze-Mbare(ib)-Wm0(ich)-Wm0(jch)))
  104    continue
  103    continue
      enddo
c     enddo ib

      if (Nifty(1).eq.2) then
c     including roper contribution from x-born which TRIUMF ignored
c     except the diff mass and ff, Roper has the same spin-isospin
c     factor and cpling const as nucleon
      ib = 3
         do 203 jch = 1, nch
         do 204 ich = 1, nch
            do jpt = 1, npts
               jchpt = jpt + (jch-1)*npts1
               do ipt = 1, jpt
                  ichpt = ipt + (ich-1)*npts1
                  Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &              + A(ich,jch,cl,1)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &              * Urop(ipt)*Urop(jpt)*P(ipt)*P(jpt)
     &/(Wmst(ipt)*Wmst(jpt)*f0*f0*(ze-Mbare(ib)-Wm(ipt)-Wm(jpt)))
               enddo
               ichpt = ich*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &           + A(ich,jch,cl,1)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &           * zUrop(ich)*Urop(jpt)*Zk(ich)*P(jpt)
     &/(Wm0st(ich)*Wmst(jpt)*f0*f0*(ze-Mbare(ib)-Wm0(ich)-Wm(jpt)))
            enddo
c
            jchpt = jch*npts1
            do ipt = 1, npts
               ichpt = ipt + (ich-1)*npts1
               Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &           + A(ich,jch,cl,1)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &           * Urop(ipt)*zUrop(jch)*P(ipt)*Zk(jch)
     &/(Wmst(ipt)*Wm0st(jch)*f0*f0*(ze-Mbare(ib)-Wm(ipt)-Wm0(jch)))
            enddo
            ichpt = ich*npts1
            jchpt = jch*npts1
            Func(ichpt,jchpt,cl) = Func(ichpt,jchpt,cl)
     &        + A(ich,jch,cl,1)*norm*Cplv(ib,ich)*Cplv(ib,jch)
     &        * zUrop(ich)*zUrop(jch)*Zk(ich)*Zk(jch)
     &/(Wm0st(ich)*Wm0st(jch)*f0*f0*(ze-Mbare(ib)-Wm0(ich)-Wm0(jch)))
  204    continue
  203    continue
      endif
c       end if roper
      endif
c       end if x-born

      return
      end
c
c***********************************************************************
c
        subroutine SpIso(I, J, cl, A)
c       return spin isospin matrix A(ich,jch,cl,ib) for x-born diag
c       actualy I,J and cl has 1->1 relation, so argv I,J is not necessary.
        integer cl
        double precision I, J, A(2,2,6,2), sign, two, sqrt5
        double precision F11, F12, Fx1, Fx2, F22
        parameter (two=2.0d0, sqrt5=2.2360679775d0)
        F11(I) = (0.5+I)/3.0
        F12(I) = (0.5+I)*(two*(3.5+I)*(1.5-I)-(I-0.5)**2)/12.0
        Fx1(I) = sqrt((3.5+I)*(2.5-I))/3.0
        Fx2(I) = sqrt((3.5+I)*(2.5-I))*
     &           ((4.5+I)*(1.5-I)-two*(I+0.5)*(I-0.5))
        F22(I) = (0.5+I)*((I-0.5d0)**2-two*(4.5+I)*(2.5-I))/30.0

        if (cl.eq.4 .or. cl.eq.5) then
          sign = -1.0d0
        else
          sign = 1.0d0
        endif
        A(1,1,cl,1) = sign* F11(I) * F11(J)
        A(1,1,cl,2) = sign* F12(I) * F12(J)
        A(1,2,cl,1) = sign* Fx1(I) * Fx1(J)
        A(1,2,cl,2) = sign* Fx2(I) * Fx2(J)/360.0
        A(2,1,cl,1) = A(1,2,cl,1)
        A(2,1,cl,2) = A(1,2,cl,2)
        A(2,2,cl,1) = sign* F11(I) * F11(J)
        A(2,2,cl,2) = sign* F22(I) * F22(J)
        return
        end

      function SBj(l, x)
c
      implicit none
      integer l
      double precision x, x2, SBj
      double precision a1, a3, a5, a7, a9, a11
      double precision a2, a4, a6, a8, a10, a12
      double precision zero, tiny, quarter, one, three
c
      parameter (zero = 0.0d0, one = 1.0d0, three = 3.0d0)
      parameter (tiny=0.01d0, quarter = 0.25d0)
      parameter (a1=  0.333333333333333333333333333333d00)
      parameter (a3= -0.333333333333333333333333333333d-1)
      parameter (a5=  0.119047619047619047619047619048d-2)
      parameter (a7= -0.220458553791887125220458553800d-4)
      parameter (a9=  0.250521083854417187750521083900d-6)
      parameter (a11=-0.192708526041859375192708526000d-8)
      parameter (a2 = 0.666666666666666666666666666667d-1)
      parameter (a4 =-0.476190476190476190476190476190d-2)
      parameter (a6 = 0.132275132275132275132275132275d-3)
      parameter (a8 =-0.200416867083533750200416867084d-5)
      parameter (a10= 0.192708526041859375192708526042d-7)
      parameter (a12=-0.128472350694572916795139017361d-9)
c
      if (l .eq. 0) then
         if (x .eq. zero) then
            SBj = one
         else
            SBj = sin(x)/x
         endif
      else if (l .eq. 1) then
         x2 = x*x
         if (x2 .lt. tiny) then
            SBj = x*(a1+x2*(a3+x2*(a5+x2*(a7+x2*(a9+a11*x2)))))
         else
            SBj = sin(x)/x2-cos(x)/x
         endif
      else if (l .eq. 2) then
         x2=x*x
         if (x2 .lt. quarter) then
            SBj=x2*(a2+x2*(a4+x2*(a6+x2*(a8+x2*(a10+a12*x2)))))
         else
            SBj=sin(x)/x*(three/x2-one)-three*cos(x)/x2
         endif
      else
```

```
         write(8, 900) l
         stop 'SBj: l value out of range'
      endif
c
  900 format('0', 'l = ', i2, ', out of range. Only 0, 1, 2 allowed')
c
      return
      end
c
c**********************************************************************
c
      subroutine Splin2(x1a,x2a,ya,y2a,m,n,x1,x2,y)
      parameter (nn=100)
      dimension x1a(m),x2a(n),ya(m,n),y2a(m,n),ytmp(nn),y2tmp(nn),yytmp(
     *nn)
      do 12 j=1,m
        do 11 k=1,n
          ytmp(k)=ya(j,k)
          y2tmp(k)=y2a(j,k)
11      continue
        call Splint(x2a,ytmp,y2tmp,n,x2,yytmp(j))
12    continue
      call Spline(x1a,yytmp,m,1.e30,1.e30,y2tmp)
      call Splint(x1a,yytmp,y2tmp,m,x1,y)
      return
      end
c
c**********************************************************************
c
      subroutine Splint(xa,ya,y2a,n,x,y)
c
c     Spline interpolation  adapted from Numerical Recipes.
c     Give a value y at x. input xa(n) is equally distributed.
c     re-tested on 20/6/97 by Lu.
c     note that x must be smaller xa(n)!!! otherwise set y = 0.

      implicit double precision (a-h,o-z)
      integer n, klo, k
      dimension xa(n),ya(n),y2a(n)

      if (x .gt. xa(n)) then
         y = 0.0d0
         return
         endif

      klo=1
      khi=n
1     if (khi-klo.gt.1) then
        k=(khi+klo)/2
        if(xa(k).gt.x)then
          khi=k
        else
          klo=k
        endif
      goto 1
      endif
      h=xa(khi)-xa(klo)
      if (h.eq.0.) pause 'bad xa input.'
      a=(xa(khi)-x)/h
      b=(x-xa(klo))/h
      y=a*ya(klo)+b*ya(khi)+
     *      ((a**3-a)*y2a(klo)+(b**3-b)*y2a(khi))*(h**2)/6.
      return
      end
      subroutine Spline(x,y,n,yp1,ypn,y2)
      implicit double precision (a-h,o-z)
      integer n, nmax, i, k
      parameter (nmax=101)
      dimension x(n), y(n), y2(n), u(nmax)
      if (yp1.gt..99e30) then
        y2(1)=0.
        u(1)=0.
      else
        y2(1)=-0.5
        u(1)=(3./(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-yp1)
      endif
      do 11 i=2,n-1
        sig=(x(i)-x(i-1))/(x(i+1)-x(i-1))
        p=sig*y2(i-1)+2.
        y2(i)=(sig-1.)/p
        u(i)=(6.*((y(i+1)-y(i))/(x(i+1)-x(i))-(y(i)-y(i-1))
     *      /(x(i)-x(i-1)))/(x(i+1)-x(i-1))-sig*u(i-1))/p
11    continue
      if (ypn.gt..99e30) then
        qn=0.
        un=0.
      else
        qn=0.5
        un=(3./(x(n)-x(n-1)))*(ypn-(y(n)-y(n-1))/(x(n)-x(n-1)))
      endif
      y2(n)=(un-qn*u(n-1))/(qn*y2(n-1)+1.)
      do 12 k=n-1,1,-1
        y2(k)=y2(k)*y2(k+1)+u(k)
12    continue
      return
      end
c
c**********************************************************************
c
```