```
      Program dwave
      parameter(neps=180,nang=180,nfreq=100,nphys=1000)
      double precision del0,alpha,scr,minfreq,maxfreq,trel,tc,wc
      real e(nphys),sigma(nphys)
      real pi,tc0,d0,ep,gam
     * ,sigma0,sg,cut,w,wmin,wmax,rr,l,es,sum,sumn,epsinf
      complex eps,epsx,cn
      integer delflag,i,nw,it,nt
      open(21,file='dwave.log')
      pi=4.*atan(1.)
c      write(*,*) 'T, and Tc0 in units of Kelvin '
      read(*,*) t,Tc0
      trel=t/tc0
      d0=tc0*0.695*1.73
      del0=d0/(tc0 * 0.695)
c      write(*,*) 'give epsinfty, scattering cross section,
c     * and plasma frequency, decay rate in 1/cm '
      read(*,*) epsinf,scr,ep,gam
      alpha=gam/(2.* pi * tc0 * 0.695)
c      write(*,*) 'give lower and upper frequency
c     * in 1/cm and no of points'
      read(*,*) wmin,wmax,nw
      minfreq=0.05
      maxfreq=8.05
      wc=40.d0
      delflag=1
      nt=4
c      do 900 it=1,nt+1
c       trel=real(it)/real(nt)
****** calculate the real part of the conductivity using palumba's routines
      call rcond(del0,alpha,scr,minfreq,maxfreq,nfreq,neps,nang,
     *  delflag,trel,tc,wc,e,sigma)
       write(21,*) 'trel,tc0,tc,del0 ',trel,tc0,tc,del0
c      **convert t and tc to absolute units of K
c       t=trel*tc0
       tc=tc*tc0
       write(21,*) 'trel,tc0,tc,del0 ',trel,tc0,tc,del0
       sigma0=ep**2 / (2. * gam )
****** convert back to absolut units: 1/cm for frequency and sigma.
****** and calculate the superfluid response part of epsilon using the
****** f-sum rule (integration using trapezium method, same was used in
****** the Kramers-Kronig transformation routine Hilber)
       sum=0.0
       do 100 i=1,nfreq
      e(i)=real(e(i))*del0*tc0*0.695
        sigma(i)=real(sigma(i))*sigma0
        sum=sum+sigma(i)
100    continue
       de=e(2)-e(1)
       sum=(sum-sigma(1)/2.-sigma(nfreq/2.))*4.*de/pi
       sumn=ep*ep*(2./pi)*atan(e(nfreq)/gam)
       write(*,*) 'sums,sumn ',sum,sumn
       if (sumn.gt.sum) then
      es=sqrt(sumn-sum)
       else
      es=0.
       endif
       write(21,*) 'superfluid fraction = ', (es/ep)**2
```

```
****** Make a Kramers-Kronig transform of the real part of sigma, assuming
****** that for w > e(nfreq) the behaviour is purely Drude like. This part
****** of the Kramers-Kronig transform is calculated using an analytical
****** expression. Also the f-sum part is added.
       cut=e(nfreq)
       do 200 i=0,nw
       w=wmin+real(i)*(wmax-wmin)/real(nw)
       if (w.ne.cut) then
        CALL HILBER(sigma,e,nfreq,w,eps,l)
        call extrap(ep,gam,l,cut,w,epsx)
        eps=epsinf-(es/w)**2+eps+epsx
        write(*,*) w/8.0657,real(eps),aimag(eps)
       endif
200    continue
c900   continue
      close(21)
      end
*************************************************************************
*************************************************************************
      subroutine rcond(del0, alpha, sigma, wfreq0, maxfreq
     * , Nfreq, Neps, Nang, delflag, T, tc, wc, xout, yout)
*************************************************************************
c     **del0: Initial value for the gap amplitude in units of Tc0.
c     **alpha: 1/2*pi*tau (tau=scattering rate) in units of Tc0.
c     **sigma: Scattering cross-section:  0.0 = Born limit; 1.0 = Unitarity
limit.
c     **wfreq: Initial frequency in units of gap amplitude (del0).
c     **maxfreq: Maximum frequency in units of gap amplitude (del0).
c     **Nfreq: Number of points at which to calculate the conductivity.
c     **Neps: Number of points used in the epsilon integral for the
conductivity.
c     **Nang: Number of points used in the angular integral for the
conductivity.
c     **delflag: If 0 -> Use the value given for "del0" as the gap amplitude.
c     **         If 1 -> Solve self-consistently for the gap amplitude.
c     **T: temperature in units of the "actual" Tc (only used if delflag=1).
c     **wc: Energy cutoff for the Matsubara sums in units of Tc0
c     **    (only used if delflag=1; 40 is a good number).
c     **
c     **Note: Tc0 = Transition temperature for the clean (alpha=0) system.
c     **      Tc = "Actual" transition temperature for the given alpha.
c     **
c     **Calculates the real part of the optical conductivity for a simple
c     **d-wave model.
c     **On output : xout = the Nfreq frequency values in units of Tc0
c     **             yout = the Nfreq conductivity values in units of sigma-0
c     **On output T is given in units of the actual tc
c     **on output del0 is the new gap in units of Tc0
c     **on output Tc is given in units of tc0
c     **Original Version : 05-94
c     **Last revised : 23-09-94
c
c     **Written by:
c     **  Mario Palumbo, Matthias Graf, and Dierk Rainer
c     **  Theoretische Physik III
c     **  Universitaet Bayreuth
c     **  D-95440 Bayreuth, GERMANY
c     **  converted to subroutine format by D. van der Marel
```

```
*************************************************************************
      implicit none
      integer Nfreq, Neps, Nang, delflag, i, units
      complex*16 csmall, cint1, cint2, cint3, cint4, cond, eps_int
      double precision del0, alpha, sigma, T, wc
      double precision wfreq0, maxfreq, delfreq, wfreq, Tc, pairbrk
      double precision delta, epsilon, err, pi, range, llim, ulim, tmp
      double precision zero, one, two, three, four, five
      real xout(1000),yout(1000)
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, three=3.0d0)
      parameter (four=4.0d0, five=5.0d0)
c     **  Set some constants:
c     **  delta : Offset for the finite-differenced derivatives.
c     **  epsilon : Maximum residual for termination of the Newton iterations.
c     **  err : Integrate up to err from singularities.
c     **  csmall : +/- with epsilon to get retarded/advanced propagators.
c     **  pi : 3.14...
      delta = 1.0d-6
      epsilon = 1.0d-10
      err = 1.0d-8
      csmall = (zero,1.0d-8)
      pi = acos(-one)
c     **Calculate the "actual" Tc
      Tc = pairbrk(alpha)
c     **Calculate self-consistent gap if desired.
      if (delflag .gt. 0) then
        call get_del0(del0,T*Tc,alpha,sigma,wc,delta,epsilon,1)
      end if
c     **Determine the frequency (omega) step size in units of Tc0.
      if (Nfreq .gt. 1) then
        delfreq = (maxfreq-wfreq0)*del0 / dble(Nfreq-1)
      else
        delfreq = zero
      end if
      T = T*Tc
      do 100 i = 1, Nfreq, 1
        wfreq = wfreq0*del0 + dble(i-1)*delfreq
        range = wfreq/two + 7.0d0*T
        cint1 = (zero,zero)
        cint2 = (zero,zero)
        cint3 = (zero,zero)
        cint4 = (zero,zero)
c       **For each frequency (wfreq) we must do an energy integral.  This
c       **integral contains, in general, singularities (1/sqrt type).  In
c       **fact we cannot locate them exactly since we integrate on epsilon
c       **while the singularities occur at:
c       **  Re( epsilon-tilde(epsilon) ) = (+/-) delta_0
c       **  Re( epsilon-tilde(epsilon+omega) ) = delta_0
c       **where epsilon-tilde is the renormalized or "dressed" energy.  On
c       **the other hand the sigularities are only a problem when alpha->0,
c       **since here epsilon-tilde has almost no imaginary component; but
c       **as alpha->0, epsilon-tilde->epsilon so we can do a pretty good
c       **job of locating the sigularities in practice.  The following if
c       **statement breaks the integral into pieces (around the "critcal"
c       **points) based on the following three cases repectively:
c       **  delta_0 < omega/2           (needs 3/4 regions)
c       **  delta_0 > omega             (needs 4 regions)
c       **  omega/2 < delta_0 < omega   (needs 3/4 regions)
```

```
c       **Note that epsilon=0 is always a special point since the integrand
c       **has a sharp jump there at low temperatures (coming from a tanh).
        if ( del0 .le. (wfreq/two) ) then
c         **Do integral from -omega/2 to -del0 (if del0 != omega/2).
          if ( (wfreq/two - del0) .gt. two*err ) then
            llim = -(wfreq / two) + err
            ulim = -del0 - err
            cint1 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                      Nang,Neps,Nfreq)
          end if
c         **Do integral from -del0 to 0 (if del0 != omega/2).
          llim = -del0 + err
          ulim = -err
          cint2 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
c         **Do integral from 0 to (range-omega/2)
          llim = err
          ulim = range - (wfreq / two)
          cint3 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
        else if ( del0 .gt. wfreq ) then
c         **Do integral from -omega/2 to 0.
          llim = -(wfreq / two) + err
          ulim = -err
          cint1 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
c         **Do integral from 0 to (range-omega/2)
          llim = err
          ulim = range - (wfreq / two)
          cint2 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
        else
c         **Do integral from -omega/2 to del0-wfreq.
          llim = -(wfreq / two) + err
          ulim = (del0 - wfreq) - err
          cint1 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
c         **Do integral from del0-wfreq to 0 (if del0-omega != 0).
          if ( (wfreq-del0) .gt. two*err) then
            llim = (del0 - wfreq) + err
            ulim = -err
            cint2 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                      Nang,Neps,Nfreq)
          end if
c         **Do integral from 0 to (range-omega/2)
          llim = err
          ulim = range - (wfreq / two)
          cint3 = eps_int(llim,ulim,range,wfreq,del0,alpha,sigma,T,
     #                    Nang,Neps,Nfreq)
        end if
c       **In our units we multipy the resulting integral by alpha and
c       **divide by omega (plus a small imaginary piece to prevent an
c       **explosion as omega->0) to get the conductivity.
        cond = -alpha*(cint1 + cint2 + cint3) / (wfreq + csmall)
        xout(i) = wfreq/del0
        yout(i) = real(cond)
       write(21,*) 'rcond : ',xout(i),yout(i)
100    continue
```

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

*************************************************************************
      SUBROUTINE HILBER(sigma,e,n,w,eps,l)
*************************************************************************
******HILBERT TRANSFORM of the imaginary part of epsilon to
******the complex epsilon. The program takes on input only
******positive values of frequency, but uses the fact that
******Im(eps) is odd.
* An exact linear interpolation is performed in the intervals between the
* energies where the imaginary part of g is defined on input. The integration
* is 'smoothed' with an energy value l, also defined on input. This
* value must be made much smaller than the energy increments de of the input
* array. The output contains 'kinks' at the energies where Re sigma was
* defined on input.
* From input it reads
*(1) sigma: the real part of the dielectric conductivity function
*(2) E(i): the energy of the array element no i of the array sigma
*(3) n: the number of energy values contained in array sigma
*(4) w: The value of the energy for which epsilon is calculated
* To output it writes:
*(5) eps: The complex dielectric function.
*     IT IS VERY IMPORTANT THAT BOTH REAL AND
*     IMAGINARY PARTS OF THIS ARRAY ARE USED TOGETHER, because together
*     they satisfy Kramers Kronig relations EXACTLY. Note that the imaginary
*     part differs slightly from the input array, because of the broadening
*     through l!!
*************************************************************************
      parameter(nphys=1000)
      REAL sigma(nphys),sa,sb,e(nphys),A,B,l,w,rho,gam,pi,rcd,icd
      COMPLEX eps,s,cw
      INTEGER N,I,J
       pi=4.*atan(1.)
*******l is the 'smoothing' parameter, chosen much smaller than delta-e
       l=(e(2)-e(1))*1.e-5
       cw=cmplx(w,l)
       S=(0.,0.)
       DO 40 J=1,N-1
         a=e(j)
         b=e(j+1)
         sa=sigma(J)
         sb=sigma(J+1)
         rho=(sa*b-sb*a)/(b-a)
         gam=(sb-sa)/(b-a)
         S=S+rho*(clog(b-cw)-clog(b+cw)+clog(a+cw)-clog(a-cw))/cw
     *      +gam*(clog(b**2-cw**2)-clog(a**2-cw**2))
40     CONTINUE
       eps=S*2./pi
c      write(*,*) i,w,eps
c      WRITE(*,*) 'Hilberttransform ready.'
      RETURN
      END
*************************************************************************

*************************************************************************
      subroutine extrap(ep,gam,l,cut,w,epsx)
```

```
**************************************************************************
      real g,ep,gam,l,cut,pi,w
      complex epsx,cw
      INTEGER I
      cw=cmplx(w,l)
      pi=4.*atan(1.)
      epsx=((ep**2)/(pi*(cw**2+gam**2))) *
     *         ( (gam/cw)*(clog(cut+cw)-clog(cut-cw))
     *         + 2*atan(cut/gam) - pi )
c      write(*,*) w,ep,gam,cut,epsx
      return
      end
**************************************************************************



c**********************************************************************
c
      complex*16 function eps_int(llim,ulim,range,wfreq,del0,alpha,
     #                            sigma,T,Nang,Neps,iflag)
c
      implicit none
      double precision llim, ulim, range, wfreq
      double precision del0, alpha, sigma, T
      integer Nang, Neps, iflag
c
c   **This function uses Simpson's rule to do an energy integral from
c   **llim to ulim.  If iflag=1 then the integrand will be printed
c   **(in the file fort.20) showing which points where were used.
c
      integer j, Nc
      complex*16 ct, cint, phi_int, phi_int2
      double precision eps, deleps, pi, acos, delta, epsilon
      double precision zero, one, two, three, four
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, three=3.0d0)
      parameter (four=4.0d0)
c
      delta = 1.0d-6
      epsilon = 1.0d-10
      pi = acos(-one)
c
c   **Nc is calculated to keep the density of points roughly the same
c   **in each region.  However there is a lower bound put in by hand to
c   **be sure each region has enough points.  Since we use Simpson's rule,
c   **Nc MUST be positive.
      Nc = 2*(( int((ulim-llim)*dble(Neps)/range) + 1 ) / 2)
      if (Nc .lt. 20)  Nc = 20
      deleps = (ulim - llim) / dble(Nc)
      eps = llim
      cint = phi_int(eps,wfreq,del0,alpha,sigma,delta,epsilon,T,
     #               Nang,Neps)
      if (iflag .eq. 1)  write(20,*) eps, real(cint), dimag(cint)
      do j = 1, Nc-1, 1
        eps = eps + deleps
        ct = phi_int(eps,wfreq,del0,alpha,sigma,delta,epsilon,T,
     #               Nang,Neps)
        if (iflag .eq. 1)  write(20,*) eps, real(ct), dimag(ct)
        if (mod(j,2) .eq. 1) then
```

```
          cint = cint + four*ct
        else
          cint = cint + two*ct
        end if
      end do
      eps = eps + deleps
      ct = phi_int(eps,wfreq,del0,alpha,sigma,delta,epsilon,T,
     #             Nang,Neps)
      if (iflag .eq. 1)  write(20,*) eps, real(ct), dimag(ct)
      cint = cint + ct
      cint = deleps*cint/three
c
      eps_int = cint
c
      return
      end

c
c**File : integrals.f
c
c**A set of routines for doing the angular integrals in the conductivity.
c
c**Original Version : 05-94
c**Last revised : 11-10-94
c
c**Written by:
c**  Mario Palumbo, Matthias Graf, and Dierk Rainer
c**  Theoretische Physik III
c**  Universitaet Bayreuth
c**  D-95440 Bayreuth, GERMANY
c

c      complex*16 function sqrt(x)
c      complex*16 x
c      sqrt=dcmplx(csqrt(cmplx(x)))
c      return
c      end
c
c      complex*16 function log(x)
c      complex*16 x
c      log=dcmplx(clog(cmplx(x)))
c      return
c      end
c
c      double precision function imag(x)
c      complex*16 x
c      imag=aimag(cmplx(x))
c      return
c      end
c-----------------------------------------------------------------------
c
      complex*16 function phi_int(eps,wfreq,del0,alpha,sigma,delta,
     #                            epsilon,T,Nang,iflag)
c
      double precision eps, wfreq, del0, alpha, sigma, delta, epsilon, T
      integer Nang, iflag
c
c   **Performs the angular integration over the conductivity integrand,
```

```
c   **and multipies in the "tanh" factors at the end.
c
      complex*16 csmall, ceps, cepst, cepsw, cepstw, get_cepst
      complex*16 cint1, cint2, ctmp1, ctmp2
      complex*16 ccepst, ctmpae, ctmpre, ctmprw, Dmr, Dmn, cg3bar
      double precision err, epsw, tmp, phi0, phi0w
      double precision zero, one, two, pi
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0)
c
      pi = acos(-one)
      err = 1.0d-6
      csmall = (zero,1.0d-6)
c
      epsw = eps + wfreq
c
c   **Calculate epsilon_tilde_retarded at epsilon and epsilon+omega.
      ceps = eps + csmall
      cepst = get_cepst(cepst,ceps,del0,alpha,sigma,delta,epsilon,0)
      cepsw = epsw + csmall
      cepstw = get_cepst(cepstw,cepsw,del0,alpha,sigma,delta,epsilon,0)
c
c   **Calculate the "D_minus" factors which appear in the conductivity
c   **integrand.
      ccepst = conjg(cepst)
      tmp = alpha*sqrt(sigma)*sqrt(one-sigma)
      if (tmp .gt. 1.0d-8) then
        ctmpae = one + (cg3bar(ccepst,del0,0)/pi)**2
        ctmpre = one + (cg3bar(cepst,del0,0)/pi)**2
        ctmprw = one + (cg3bar(cepstw,del0,0)/pi)**2
        Dmr = ((tmp*ctmpre) / (one - sigma*ctmpre))
     #      - ((tmp*ctmprw) / (one - sigma*ctmprw))
        Dmn = ((tmp*ctmpae) / (one - sigma*ctmpae))
     #      - ((tmp*ctmprw) / (one - sigma*ctmprw))
      else
        Dmr = zero
        Dmn = zero
      end if
c
c   **Determine how many sqrt-singularities our phi-integrand has.
      phi0 = -one
      phi0w = -one
      tmp = real(cepst**2)
      if (tmp .ge. zero)  tmp = sqrt(tmp) / del0
      if ((tmp .le. one) .and. (tmp .ge. zero))  phi0 = acos(tmp)
      tmp = real(cepstw**2)
      if (tmp .ge. zero)  tmp = sqrt(tmp) / del0
      if ((tmp .le. one) .and. (tmp .ge. zero))  phi0w = acos(tmp)
c
c   **Do the angular integral...
      if ( (phi0 .lt. zero) .and. (phi0w .lt. zero) )then
c     **This is the case of no sqrt_singularities.  Do a simple Simpson's
c     **rule from 0 to pi/2.
        call simpson0(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                Dmr,Dmn,Nang,iflag)
        if (iflag .eq. 1)  write(21,*) eps, '   0.0'
      else if ( (phi0w .lt. zero) .and. (phi0 .gt. zero) )then
c     **This is the case of one sqrt_singularity at phi0.  Here we do a
c     **Simpson's rule integral from 0 to phi0-err and from phi0+err to
```

```
c     **pi/2 with the singular part of the integral subtracted out.  The
c     **finite contribution of the sqrt-singularity is then added back
c     **by hand from an analytic expression.
        call simpson1(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                Dmr,Dmn,zero,phi0-err,phi0,1,Nang,iflag)
        call simpson1(cepst,cepstw,ctmp1,ctmp2,del0,alpha,sigma,T,
     #                Dmr,Dmn,phi0+err,pi/two,phi0,1,Nang,iflag)
        cint1 = cint1 + ctmp1
        cint2 = cint2 + ctmp2
        if (iflag .eq. 1)  write(21,*) eps, '   1.0'
      else if ( (phi0w .gt. zero) .and. (phi0 .lt. zero) )then
c     **This is the case of one sqrt_singularity at phi0w.  Here we do a
c     **Simpson's rule integral from 0 to phi0w-err and from phi0w+err to
c     **pi/2 with the singular part of the integral subtracted out.  The
c     **finite contribution of the sqrt-singularity is then added back
c     **by hand from an analytic expression.
        call simpson1(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                Dmr,Dmn,zero,phi0w-err,phi0w,2,Nang,iflag)
        call simpson1(cepst,cepstw,ctmp1,ctmp2,del0,alpha,sigma,T,
     #                Dmr,Dmn,phi0w+err,pi/two,phi0w,2,Nang,iflag)
        cint1 = cint1 + ctmp1
        cint2 = cint2 + ctmp2
        if (iflag .eq. 1)  write(21,*) eps, '   1.5'
      else
c     **This is the case of two sqrt_singularities at phi0 and phi0w.
c     **Here we divide the integration region (0 to pi/2) into 4 regions:
c     **0 to phi0w-err, phi0w+err to (phi0+phi0w)/2, (phi0+phi0w)/2 to
c     **phi0-err, and phi0+err to pi/2.  Here we have assumed that phi0w
c     **will always be less than phi0!  In the first 2 regions we subtract
c     **out the term which diverges at phi0w, while in the second 2 regions
c     **we subtract out the terms which diverges at phi0.  The finite
c     **contributions of these 2 singular terms are added back by hand
c     **using an analytic expression.
        if (phi0 .lt. phi0w)
     #    write(6,*) 'Warning: phi0 < phi0w! (eps,omega)= ', eps, wfreq
        call simpson1(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                Dmr,Dmn,zero,phi0w-err,phi0w,2,Nang,iflag)
        call simpson1(cepst,cepstw,ctmp1,ctmp2,del0,alpha,sigma,T,
     #                Dmr,Dmn,phi0w+err,(phi0+phi0w)/two,phi0w,2,
     #                Nang,iflag)
        cint1 = cint1 + ctmp1
        cint2 = cint2 + ctmp2
        call simpson1(cepst,cepstw,ctmp1,ctmp2,del0,alpha,sigma,T,
     #                Dmr,Dmn,(phi0+phi0w)/two,phi0-err,phi0,1,
     #                Nang,iflag)
        cint1 = cint1 + ctmp1
        cint2 = cint2 + ctmp2
        call simpson1(cepst,cepstw,ctmp1,ctmp2,del0,alpha,sigma,T,
     #                Dmr,Dmn,phi0+err,pi/two,phi0,1,Nang,iflag)
        cint1 = cint1 + ctmp1
        cint2 = cint2 + ctmp2
        if (iflag .eq. 1)  write(21,*) eps, '   2.0'
      end if
c
c   **Multiply in the "tanh" factors and normalize.
      phi_int = tanh(eps/(two*T))*cint1
     #        - tanh(epsw/(two*T))*conjg(cint1)
     #        + (tanh(epsw/(two*T)) - tanh(eps/(two*T)))*cint2
```

```
      phi_int = two*phi_int/pi
c
      return
      end
c
c*********************************************************************
c
      subroutine cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,Dmr,Dmn,
     #                      cterm1,cterm2,cterm3,cterm4,iflag)
c
      complex*16 cepst, cepstw, cterm1, cterm2, cterm3, cterm4, Dmr, Dmn
      double precision Gap2, del0, alpha, sigma
      integer iflag
c
      complex*16 Cpr, Cpn, ccepst, ctmpae, ctmpre, ctmprw
      double precision pi, acos, zero, one
      parameter (zero=0.0d0, one=1.0d0)
c
      pi = acos(-one)
      ccepst = conjg(cepst)
c
c   **Calculate the "C_plus" factors.
      ctmpae = sqrt( Gap2 - ccepst**2 )
      ctmpre = sqrt( Gap2 - cepst**2 )
      ctmprw = sqrt( Gap2 - cepstw**2 )
      Cpr = -(ctmprw + ctmpre) / pi
      Cpn = -(ctmprw + ctmpae) / pi
c
c   **Remove singular denominator if desired.
      if (iflag .eq. 1) then
        ctmpre = (one,zero)
        ctmpae = (one,zero)
      end if
      if (iflag .eq. 2) then
        ctmprw = (one,zero)
      end if
c
c   **Calculate the four independent terms of the integrand.
      cterm1 = (pi**2)*Cpr / ((pi*Cpr)**2 + Dmr**2)
      cterm2 = cterm1*(cepst*cepstw + Gap2)/(ctmpre*ctmprw)
      cterm3 = (pi**2)*Cpn / ((pi*Cpn)**2 + Dmn**2)
      cterm4 = cterm3*(ccepst*cepstw + Gap2)/(ctmpae*ctmprw)
c
      return
      end
c
c*********************************************************************
c
      subroutine simpson0(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                    Dmr,Dmn,Nang,iflag)
c
      complex*16 cepst, cepstw, cint1, cint2, Dmr, Dmn
      double precision del0, alpha, sigma, T
      integer Nang, iflag
c
      integer i
      complex*16 ct1, ct2, ct3, ct4
      double precision phi, dphi, Gap2
```

```
      double precision zero, one, two, three, four, pi
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, three=3.0d0)
      parameter (four=4.0d0)
c
      pi = acos(-one)
c
      dphi = pi / dble(2*Nang)
      phi = zero
      Gap2 = (del0*cos(phi))**2
      call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                Dmr,Dmn,ct1,ct2,ct3,ct4,0)
      if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
      if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
      if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
      if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
      cint1 = ct1 + ct2
      cint2 = ct3 + ct4
      do i = 1, Nang-1, 1
        phi = phi + dphi
        Gap2 = (del0*cos(phi))**2
        call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                  Dmr,Dmn,ct1,ct2,ct3,ct4,0)
        if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
        if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
        if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
        if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
        if (mod(i,2) .eq. 1) then
          cint1 = cint1 + four*(ct1+ct2)
          cint2 = cint2 + four*(ct3+ct4)
        else
          cint1 = cint1 + two*(ct1+ct2)
          cint2 = cint2 + two*(ct3+ct4)
        end if
      end do
      phi = phi + dphi
      Gap2 = (del0*cos(phi))**2
      call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                Dmr,Dmn,ct1,ct2,ct3,ct4,0)
      if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
      if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
      if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
      if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
      cint1 = cint1 + ct1 + ct2
      cint2 = cint2 + ct3 + ct4
      cint1 = cint1*dphi/three
      cint2 = cint2*dphi/three
c
      return
      end
c
c*********************************************************************
c
      subroutine simpson1(cepst,cepstw,cint1,cint2,del0,alpha,sigma,T,
     #                    Dmr,Dmn,llim,ulim,phi0,sflag,Nang,iflag)
c
      complex*16 cepst, cepstw, cint1, cint2, Dmr, Dmn
      double precision del0, alpha, sigma, T
      double precision llim, ulim, phi0
```

```
      integer sflag, Nang, iflag
c
      integer i, Nc
      complex*16 ci, tmp1, tmp2, ctmp, carg, csum
      complex*16 ct1, ct2, ct3, ct4
      complex*16 ct2_0, ct4_0, ceps2, ceps4, sqrt
      double precision phi, dphi, Gap2, diff
      double precision reps2, ieps2, reps4, ieps4
      double precision zero, one, two, three, four, pi
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, three=3.0d0)
      parameter (four=4.0d0)
c
      pi = acos(-one)
      ci = (zero,one)
c
      if (sflag .eq. 1) then
        ceps2 = cepst
        ceps4 = conjg(cepst)
      else if (sflag .eq. 2) then
        ceps2 = cepstw
        ceps4 = cepstw
      else
        write(*,*) 'Bad value for sflag in simpson1: sflag = ', sflag
        stop
      end if
      reps2 = dreal(ceps2**2)
      ieps2 = dimag(ceps2**2)
      reps4 = dreal(ceps4**2)
      ieps4 = dimag(ceps4**2)
      tmp1 = del0**2 - two*reps2
      tmp2 = sqrt(dcmplx(reps2))*sqrt(dcmplx(del0**2 - reps2))
      call cond_terms(cepst,cepstw,reps2,del0,alpha,sigma,
     #                Dmr,Dmn,ct1,ct2_0,ct3,ct4_0,sflag)
c
      Nc = 2*((int(dble(2*Nang)*(ulim-llim)/pi) + 1) / 2)
      if (Nc .lt. 10)  Nc = 10
      dphi = (ulim-llim) / dble(Nc)
      phi = llim
      diff = phi - phi0
      Gap2 = (del0*cos(phi))**2
      call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                Dmr,Dmn,ct1,ct2,ct3,ct4,0)
      ct2 = ct2 - ct2_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps2)
      ct4 = ct4 - ct4_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps4)
      if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
      if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
      if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
      if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
      cint1 = ct1 + ct2
      cint2 = ct3 + ct4
      do i = 1, Nc-1, 1
        phi = phi + dphi
        diff = phi - phi0
        Gap2 = (del0*cos(phi))**2
        call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                  Dmr,Dmn,ct1,ct2,ct3,ct4,0)
        ct2 = ct2 - ct2_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps2)
        ct4 = ct4 - ct4_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps4)
```

```
        if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
        if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
        if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
        if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
        if (mod(i,2) .eq. 1) then
          cint1 = cint1 + four*(ct1 + ct2)
          cint2 = cint2 + four*(ct3 + ct4)
        else
          cint1 = cint1 + two*(ct1 + ct2)
          cint2 = cint2 + two*(ct3 + ct4)
        end if
      end do
      phi = phi + dphi
      diff = phi - phi0
      Gap2 = (del0*cos(phi))**2
      call cond_terms(cepst,cepstw,Gap2,del0,alpha,sigma,
     #                Dmr,Dmn,ct1,ct2,ct3,ct4,0)
      ct2 = ct2 - ct2_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps2)
      ct4 = ct4 - ct4_0 / sqrt(tmp1*(diff**2)-two*tmp2*diff-ci*ieps4)
      if (iflag .eq. 1)  write(44,*) phi, real(ct1), dimag(ct1)
      if (iflag .eq. 1)  write(45,*) phi, real(ct2), dimag(ct2)
      if (iflag .eq. 1)  write(46,*) phi, real(ct3), dimag(ct3)
      if (iflag .eq. 1)  write(47,*) phi, real(ct4), dimag(ct4)
      cint1 = cint1 + ct1 + ct2
      cint2 = cint2 + ct3 + ct4
      cint1 = cint1*dphi/three
      cint2 = cint2*dphi/three
c
      if (dreal(tmp1) .lt. zero) then
        ctmp = ci*sqrt(ci*ieps2*tmp1 + tmp2**2)
        carg = (tmp1*(ulim-phi0) - tmp2) / ctmp
        csum = log(carg + sqrt(carg**2 + (one,zero)))
        carg = (tmp1*(llim-phi0) - tmp2) / ctmp
        csum = csum - log(carg + sqrt(carg**2 + (one,zero)))
        cint1 = cint1 + ct2_0*csum/sqrt(tmp1)
        ctmp = ci*sqrt(ci*ieps4*tmp1 + tmp2**2)
        carg = (tmp1*(ulim-phi0) - tmp2) / ctmp
        csum = log(carg + sqrt(carg**2 + (one,zero)))
        carg = (tmp1*(llim-phi0) - tmp2) / ctmp
        csum = csum - log(carg + sqrt(carg**2 + (one,zero)))
        cint2 = cint2 + ct4_0*csum/sqrt(tmp1)
      else
        carg= two*sqrt(tmp1*(tmp1*(ulim-phi0)**2 - two*tmp2*(ulim-phi0)
     #       - ci*ieps2)) + two*tmp1*(ulim-phi0) - two*tmp2
        csum = log(carg)
        carg= two*sqrt(tmp1*(tmp1*(llim-phi0)**2 - two*tmp2*(llim-phi0)
     #       - ci*ieps2)) + two*tmp1*(llim-phi0) - two*tmp2
        csum = csum - log(carg)
        cint1 = cint1 + ct2_0*csum/sqrt(tmp1)
        carg= two*sqrt(tmp1*(tmp1*(ulim-phi0)**2 - two*tmp2*(ulim-phi0)
     #       - ci*ieps4)) + two*tmp1*(ulim-phi0) - two*tmp2
        csum = log(carg)
        carg= two*sqrt(tmp1*(tmp1*(llim-phi0)**2 - two*tmp2*(llim-phi0)
     #       - ci*ieps4)) + two*tmp1*(llim-phi0) - two*tmp2
        csum = csum - log(carg)
        cint2 = cint2 + ct4_0*csum/sqrt(tmp1)
      end if
c
```

```
      return
      end



c
c**File : cepst_funcs.f
c
c**Routines for self-consistently calculating "epsilon_tilde" (the
c**renormalized quasiparticle excitation energy) for a given "bare"
c**excitation energy (ceps), gap amplitude (del0), scattering rate,
c**and scattering cross-section.
c
c**Original Version : 05-94
c**Last revised : 23-09-94
c
c**Written by:
c**  Mario Palumbo, Matthias Graf, and Dierk Rainer
c**  Theoretische Physik III
c**  Universitaet Bayreuth
c**  D-95440 Bayreuth, GERMANY
c
c-------------------------------------------------------------------------
c
      complex*16 function get_cepst(cepst,ceps,del0,alpha,sigma,
     #                              delta,epsilon,iflag)
c
      implicit none
      complex*16 cepst, ceps
      double precision del0, alpha, sigma, delta, epsilon
      integer iflag
c
c   **Calculates the retarded epsilon_tilde (cepst) in the "real-energy"
c   **representation as a function of epsilon (ceps) using a simple Newton
c   **iteration technique.
c
      integer i
      complex*16 ci, cfunc, dcfunc, cepst_func
      double precision  zero, one, pi
      parameter (zero=0.0d0, one=1.0d0)
c
      ci = (zero,one)
      pi = acos(-one)
c
c   **If "iflag"=0 we use the normal state value for "cepst" as an initial,
c   **guess, otherwise we use the passed value.
      if (iflag .eq. 0)  cepst = ceps + ci*pi*alpha
      cfunc = cepst_func(cepst,ceps,del0,alpha,sigma)
      dcfunc = cepst_func(cepst+delta,ceps,del0,alpha,sigma) - cfunc
c
      do i = 1, 50, 1
        cepst = cepst - delta*cfunc/dcfunc
        cfunc = cepst_func(cepst,ceps,del0,alpha,sigma)
        if (abs(cfunc) .lt. epsilon) goto 10
        dcfunc = cepst_func(cepst+delta,ceps,del0,alpha,sigma) - cfunc
      end do
      write(*,*) 'Warning: cepst iteration did not converge at: '
      write(*,*) '    ceps = ', ceps
```

```
      write(*,*) '    cepst = ', cepst
      write(*,*) '    cfunc = ', cfunc
10    continue
c
      get_cepst = cepst
c
      return
      end
c
c*********************************************************************
c
      complex*16 function cepst_func(cepst,ceps,del0,alpha,sigma)
c
      implicit none
      complex*16 cepst, ceps
      double precision del0, alpha, sigma
c
c   **Evaluates the "epsilon_tilde equation" in the real-energy rep.
c   **for a given epsilon_tilde (cepst) and a given gap (del0).
c
      complex*16 cg3, cg3bar, ctmp
      double precision one, pi
      parameter (one=1.0d0)
c
      pi = acos(-one)
c
      cg3 = cg3bar(cepst,del0,0)
c
      ctmp = one - sigma*(one + (cg3/pi)**2)
      cepst_func = ceps - (alpha*cg3/ctmp) - cepst
c
      return
      end
c
c*********************************************************************
c
      complex*16 function cg3bar(cepst,del0,flag)
c
      implicit none
      complex*16 cepst
      double precision del0
      integer flag
c
c   **Calculates the retarted g3_bar in the "real-energy" representation
c   **by doing a numerical integration from 0 to pi/2.  First we determine
c   **if the integrand is singular in the interval (0,pi/2), and then we
c   **employ the appropriate integration technique.
c
      integer n, Nc0, Nc
      complex*16 ctmp, ctmp2, csum, csum2, cg1, cg2, ci
      double precision err, tmp, reps, ieps, phi0, phi, dphi
      double precision zero, one, two, three, four, pi
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, three=3.0d0)
      parameter (four=4.0d0)
c
      pi = acos(-one)
      ci = (zero,one)
c
```

```
      err = 1.0d-4
      Nc = 40
      csum = (zero,zero)
      csum2 = (zero,zero)
      reps = dreal(cepst**2)
      ieps = dimag(cepst**2)
      tmp = reps
      if (tmp .gt. zero)  tmp = sqrt(tmp) / del0
c
      if ( (tmp .gt. one) .or. (tmp .lt. zero) ) then
c     **This is the case of no singularities.  Do a simple Simpson's rule
c     **from 0 to pi/2.
        dphi = pi / (two*dble(Nc))
        phi = zero
        csum = one / sqrt( (del0*cos(phi))**2 - cepst**2 )
        if (flag .eq. 1)  write(11,*) phi, real(csum), dimag(csum)
        do n = 1, Nc-1, 1
          phi = phi + dphi
          ctmp = one / sqrt( (del0*cos(phi))**2 - cepst**2 )
          if (flag .eq. 1)  write(11,*) phi, real(ctmp), dimag(ctmp)
          if (mod(n,2) .eq. 1) then
            csum = csum + four*ctmp
          else
            csum = csum + two*ctmp
          end if
        end do
        phi = phi + dphi
        ctmp = one / sqrt( (del0*cos(phi))**2 - cepst**2 )
        if (flag .eq. 1)  write(11,*) phi, real(ctmp), dimag(ctmp)
        csum = csum + ctmp
        csum = dphi*csum/three
      else
c     **This is the case of one sqrt-singularity located at the angle phi0.
c     **First evaluate phi0, then break the integral into two parts for
c     **phi less than and phi greater than phi0.  The contribution due to
c     **the singularity (which is integrable!) will be removed from the
c     **numerical integral by a subtraction trick, and then added in
c     **analytically at the end.
        phi0 = acos(tmp)
        cg1 = del0**2 - reps
        cg2 = cg1 / ( two*sqrt(reps)*sqrt(del0**2-reps) )
        Nc0 = 100
c
c     **Do the first half of of the integral
        if (phi0 .lt. err)  goto 10
        Nc = 2*((int((phi0-two*err)*dble(2*Nc0)/pi) + 1) / 2)
        if (Nc .lt. 20)  Nc = 20
        dphi = (phi0-two*err) / dble(Nc)
        phi = -phi0 + err
        csum = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #       - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
        if (flag .eq. 1) write(11,*) phi+phi0, real(csum), dimag(csum)
        do n = 1, Nc-1, 1
          phi = phi + dphi
          ctmp = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #        - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
          if (flag .eq. 1) write(11,*) phi+phi0, real(ctmp), dimag(ctmp)
          if (mod(n,2) .eq. 1) then
```

```
            csum = csum + four*ctmp
          else
            csum = csum + two*ctmp
          end if
        end do
        phi = phi + dphi
        ctmp = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #       - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
        if (flag .eq. 1) write(11,*) phi+phi0, real(ctmp), dimag(ctmp)
        csum = csum + ctmp
        csum = dphi*csum/three
c
 10     continue
c
c     **Do the second half of of the integral
        if (pi/two-phi0 .lt. err)  goto 20
        Nc = 2*((int((pi/two-phi0-two*err)*dble(2*Nc0)/pi) + 1) / 2)
        if (Nc .lt. 20)  Nc = 20
        dphi = (pi/two-phi0-two*err) / dble(Nc)
        phi = zero + err
        ctmp = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #       - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
        if (flag .eq. 1) write(11,*) phi+phi0, real(ctmp), dimag(ctmp)
        csum2 = csum2 + ctmp
        do n = 1, Nc-1, 1
          phi = phi + dphi
          ctmp = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #        - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
          if (flag .eq. 1) write(11,*) phi+phi0, real(ctmp), dimag(ctmp)
          if (mod(n,2) .eq. 1) then
            csum2 = csum2 + four*ctmp
          else
            csum2 = csum2 + two*ctmp
          end if
        end do
        phi = phi + dphi
        ctmp = one / sqrt( (del0*cos(phi+phi0))**2 - cepst**2 )
     #       - one / sqrt( cg1*(phi**2-(one/cg2)*phi - (ci/cg1)*ieps) )
        if (flag .eq. 1) write(11,*) phi+phi0, real(ctmp), dimag(ctmp)
        csum2 = csum2 + ctmp
        csum2 = dphi*csum2/three
c
 20     continue
c
c     **Add the contribution from the singular part.
        ctmp = two*sqrt( (pi/two-phi0)**2 - (one/cg2)*(pi/two-phi0)
     #                  - (ci/cg1)*ieps )
     #       + two*(pi/two-phi0) - (one/cg2)
        ctmp2 = two*sqrt( phi0**2 + (one/cg2)*phi0 - (ci/cg1)*ieps )
     #        - two*phi0 - (one/cg2)
        ctmp = (one/sqrt(cg1))*(log(ctmp) - log(ctmp2))
        csum = csum + csum2 + ctmp
      end if
c
      cg3bar = -two*cepst*csum
c
      return
      end
```

```
C-----------------------------------------------------------------------
      Double Precision Function PAIRBRK(ALPHA)
C-----------------------------------------------------------------------
C***BEGIN PROLOGUE  PAIRBRK
C***DATE WRITTEN   940419   (YYMMDD)
C***REVISION DATE  940425   (YYMMDD)
C***CATEGORY NO.
C***KEYWORDS  LIBRARY=PRIVATE, PAIRBREAKING PARAMTER ALPHA AND
C           TEMPERATURE T_C(ALPHA)/T_C(0) .
C              TYPE=DOUBLE PRECISION (PAIRBRK-D)
C***AUTHOR  M. J. GRAF, UNIVERSITY OF BAYREUTH
C***PURPOSE  Store  critical temperature for given pairbreaking
C          parameter ALPHA.
C
C On entry:
C     ALPHA -     Pairbreaking parameter (double precision)
C On Return:
C     PAIRBRK     -     Pairbreaking temperature (double precision)
C
C           Standard Fortran Subroutine
C***REFERENCES  (NONE)
C***ROUTINES CALLED  (NONE)
C
C***END PROLOGUE  PAIRBRK
C-----------------------------------------------------------------------
      DOUBLE PRECISION ALPHA, ALFA(148), TC(148)
      INTEGER i
      DOUBLE PRECISION step
      DATA ALFA/
     C       0.00000000000000D+00,  2.00000000000000D-03,
     C       4.00000000000000D-03,  6.00000000000000D-03,
     C       8.00000000000000D-03,  1.00000000000000D-02,
     C       1.20000000000000D-02,  1.40000000000000D-02,
     C       1.60000000000000D-02,  1.80000000000000D-02,
     C       2.00000000000000D-02,  2.20000000000000D-02,
     C       2.40000000000000D-02,  2.60000000000000D-02,
     C       2.80000000000000D-02,  3.00000000000000D-02,
     C       3.20000000000000D-02,  3.40000000000000D-02,
     C       3.60000000000000D-02,  3.80000000000000D-02,
     C       4.00000000000000D-02,  4.20000000000000D-02,
     C       4.40000000000000D-02,  4.60000000000000D-02,
     C       4.80000000000000D-02,  5.00000000000000D-02,
     C       5.20000000000000D-02,  5.40000000000000D-02,
     C       5.60000000000000D-02,  5.80000000000000D-02,
     C       6.00000000000000D-02,  6.20000000000000D-02,
     C       6.40000000000000D-02,  6.60000000000000D-02,
     C       6.80000000000000D-02,  7.00000000000000D-02,
     C       7.20000000000000D-02,  7.40000000000000D-02,
     C       7.60000000000000D-02,  7.80000000000001D-02,
     C       8.00000000000001D-02,  8.20000000000001D-02,
     C       8.40000000000001D-02,  8.60000000000001D-02,
     C       8.80000000000001D-02,  9.00000000000001D-02,
     C       9.20000000000001D-02,  9.40000000000001D-02,
     C       9.60000000000001D-02,  9.80000000000001D-02,
     C       1.00000000000000D-01,  1.02000000000000D-01,
     C       1.04000000000000D-01,  1.06000000000000D-01,
     C       1.08000000000000D-01,  1.10000000000000D-01,
```

```
      C       1.12000000000000D-01,  1.14000000000000D-01,
      C       1.16000000000000D-01,  1.18000000000000D-01,
      C       1.20000000000000D-01,  1.22000000000000D-01,
      C       1.24000000000000D-01,  1.26000000000000D-01,
      C       1.28000000000000D-01,  1.30000000000000D-01,
      C       1.32000000000000D-01,  1.34000000000000D-01,
      C       1.36000000000000D-01,  1.38000000000000D-01,
      C       1.40000000000000D-01,  1.42000000000000D-01,
      C       1.44000000000000D-01,  1.46000000000000D-01,
      C       1.48000000000000D-01,  1.50000000000000D-01,
      C       1.52000000000000D-01,  1.54000000000000D-01,
      C       1.56000000000000D-01,  1.58000000000000D-01,
      C       1.60000000000000D-01,  1.62000000000000D-01,
      C       1.64000000000000D-01,  1.66000000000000D-01,
      C       1.68000000000000D-01,  1.70000000000000D-01,
      C       1.72000000000000D-01,  1.74000000000000D-01,
      C       1.76000000000000D-01,  1.78000000000000D-01,
      C       1.80000000000000D-01,  1.82000000000000D-01,
      C       1.84000000000000D-01,  1.86000000000000D-01,
      C       1.88000000000000D-01,  1.90000000000000D-01,
      C       1.92000000000000D-01,  1.94000000000000D-01,
      C       1.96000000000000D-01,  1.98000000000000D-01,
      C       2.00000000000000D-01,  2.02000000000000D-01,
      C       2.04000000000000D-01,  2.06000000000000D-01,
      C       2.08000000000000D-01,  2.10000000000000D-01,
      C       2.12000000000000D-01,  2.14000000000000D-01,
      C       2.16000000000000D-01,  2.18000000000000D-01,
      C       2.20000000000000D-01,  2.22000000000000D-01,
      C       2.24000000000000D-01,  2.26000000000000D-01,
      C       2.28000000000000D-01,  2.30000000000000D-01,
      C       2.32000000000000D-01,  2.34000000000000D-01,
      C       2.36000000000000D-01,  2.38000000000000D-01,
      C       2.40000000000000D-01,  2.42000000000000D-01,
      C       2.44000000000000D-01,  2.46000000000000D-01,
      C       2.48000000000000D-01,  2.50000000000000D-01,
      C       2.52000000000000D-01,  2.54000000000000D-01,
      C       2.56000000000000D-01,  2.58000000000000D-01,
      C       2.60000000000000D-01,  2.62000000000000D-01,
      C       2.64000000000000D-01,  2.66000000000000D-01,
      C       2.68000000000000D-01,  2.70000000000000D-01,
      C       2.71000000000000D-01,  2.72000000000000D-01,
      C       2.73000000000000D-01,  2.74000000000000D-01,
      C       2.75000000000000D-01,  2.76000000000000D-01,
      C       2.77000000000000D-01,  2.78000000000000D-01,
      C       2.79000000000000D-01,  2.80000000000000D-01,
      C       2.81000000000000D-01,  2.82000000000000D-01/
C
       DATA   TC/
      C       1.00000000000000D+00,  9.94979985780569D-01,
      C       9.90168211444149D-01,  9.85254826826387D-01,
      C       9.80269824854986D-01,  9.75273290321524D-01,
      C       9.70285176811678D-01,  9.65307020706046D-01,
      C       9.60266689230268D-01,  9.55335334055061D-01,
      C       9.50230656758989D-01,  9.45258666385887D-01,
      C       9.40124878419510D-01,  9.35206234857973D-01,
      C       9.30013743037869D-01,  9.24997229038698D-01,
      C       9.19956199101609D-01,  9.14952268836332D-01,
      C       9.09902595219130D-01,  9.04822495918116D-01,
```

```
C       8.99734266155204D-01,  8.94548354115810D-01,
C       8.89359501182762D-01,  8.84253369012316D-01,
C       8.79192325250694D-01,  8.74031693990246D-01,
C       8.68798856714218D-01,  8.63643441647420D-01,
C       8.58502452054537D-01,  8.53418908248721D-01,
C       8.48221321046611D-01,  8.42902405896712D-01,
C       8.37743834052559D-01,  8.32516860116987D-01,
C       8.27316377681085D-01,  8.21985741058676D-01,
C       8.16685376854979D-01,  8.11440692831239D-01,
C       8.06049828197337D-01,  8.00882381307412D-01,
C       7.95494556184380D-01,  7.90252180968870D-01,
C       7.84876573270205D-01,  7.79566288487801D-01,
C       7.74124290358790D-01,  7.68789008065455D-01,
C       7.63518570327797D-01,  7.58040750804586D-01,
C       7.52520135287419D-01,  7.47198991064242D-01,
C       7.41769455240884D-01,  7.36319086411256D-01,
C       7.30789145165074D-01,  7.25254532222533D-01,
C       7.19683568017091D-01,  7.14149685664602D-01,
C       7.08725228042168D-01,  7.03101828375263D-01,
C       6.97646455067538D-01,  6.91970931564599D-01,
C       6.86353417631351D-01,  6.80695172913838D-01,
C       6.75146034868489D-01,  6.69474029046236D-01,
C       6.63780153177413D-01,  6.58026030349122D-01,
C       6.52410657498903D-01,  6.46700337482505D-01,
C       6.40972677168680D-01,  6.35117953426164D-01,
C       6.29328930008711D-01,  6.23513875789632D-01,
C       6.17648313904561D-01,  6.11853896974146D-01,
C       6.05892188829178D-01,  6.00006359723133D-01,
C       5.94119111875201D-01,  5.88173537369706D-01,
C       5.82238615607424D-01,  5.76170013114152D-01,
C       5.70101015785300D-01,  5.64058797884797D-01,
C       5.58041546988724D-01,  5.51850865743404D-01,
C       5.45718626185905D-01,  5.39609150227492D-01,
C       5.33387951947750D-01,  5.27137801596926D-01,
C       5.20896896141043D-01,  5.14627762489749D-01,
C       5.08215757178552D-01,  5.01920351581060D-01,
C       4.95475666214391D-01,  4.89049034380746D-01,
C       4.82571580981771D-01,  4.76048611955350D-01,
C       4.69465271664764D-01,  4.62856162669029D-01,
C       4.56304114272036D-01,  4.49547022131376D-01,
C       4.42837420316735D-01,  4.36024140574640D-01,
C       4.29232665942347D-01,  4.22346204791249D-01,
C       4.15389083781882D-01,  4.08404788612145D-01,
C       4.01240326312244D-01,  3.94167742688480D-01,
C       3.86956483499255D-01,  3.79751416274201D-01,
C       3.72355145242825D-01,  3.64901791259141D-01,
C       3.57501548000570D-01,  3.49891880923231D-01,
C       3.42210542152577D-01,  3.34472586529999D-01,
C       3.26578870601291D-01,  3.18630173373065D-01,
C       3.10555789957023D-01,  3.02336141795134D-01,
C       2.94008192119330D-01,  2.85553607926441D-01,
C       2.77000508140849D-01,  2.68181677524817D-01,
C       2.59196124354300D-01,  2.49971067669985D-01,
C       2.40591452087387D-01,  2.31042331853926D-01,
C       2.21189970855487D-01,  2.10965309404162D-01,
C       2.00335369242877D-01,  1.89455118266312D-01,
C       1.78107339472908D-01,  1.66150851355199D-01,
C       1.53636815771606D-01,  1.40062226622720D-01,
```

```
     C       1.33072394977902D-01,  1.25606591318740D-01,
     C       1.17713159145716D-01,  1.09491290615641D-01,
     C       1.00616735253519D-01,  9.10322435577660D-02,
     C       8.08045506548403D-02,  6.87554249261647D-02,
     C       5.46492128132853D-02,  3.59672322604878D-02,
     C       3.87897322801049D-03,  0.00000000000000D+00/
C
      SAVE ALFA, TC
C
      If( ALPHA.LT.0d0 )Then
        PAIRBRK=0d0
        Return
      EndIf
C-----------------------------------------------------------------------
C               Locate TALPHA
C-----------------------------------------------------------------------
       If( ALPHA.LT.0.282d0)Then
        If( ALPHA.LE.0.27d0 )Then
          step=2d-03
          i=1+int(ALPHA/step)
        Else
          step=1d-03
          i=int(ALPHA/step)-134
        EndIf
        If( dabs(ALPHA-ALFA(i)).lE. 1d-04 )Then
          PAIRBRK=TC(i)
        Else
          PAIRBRK=TC(i)+( (TC(i+1)-TC(i))/(ALFA(i+1)-ALFA(i))*
     +          (ALPHA-ALFA(i)) )
        EndIf
       Else
      PAIRBRK=0d0
       EndIf
C
CW        Write(*,*)'PAIRBRK=',PAIRBRK
C
      Return
      End

c
c**File : gap_funcs.f
c
c**Routines for self-consistently calculating the gap amplitude for a
c**given temperature, scattering rate, and scattering cross-section.
c
c**Original Version : 05-94
c**Last revised : 23-09-94
c
c**Written by:
c**  Mario Palumbo, Matthias Graf, and Dierk Rainer
c**  Theoretische Physik III
c**  Universitaet Bayreuth
c**  D-95440 Bayreuth, GERMANY
c
c-----------------------------------------------------------------------
c
      double precision function get_del0(del0,T,alpha,sigma,wc,delta,
     #                                   epsilon,iflag)
```

```
c
      implicit none
      double precision del0, T, alpha, sigma, wc, delta, epsilon
      integer iflag
c
c   **Calculates self-consistently the gap amplitude (ie. the maximal
c   **value of the gap over the Fermi surface) using a simple Newton
c   **iteration method.
c
      integer i, N_cut
      double precision gfunc, gfunc1, gap_func
      double precision one, two, pi
      parameter(one=1.0d0, two=2.0d0)
c
      pi = acos(-one)
c
c   **Calculate the maximum N (N_cut) in the Matsubara sum from the
c   **cutoff energy "wc".
      N_cut = int((wc/(pi*T) - one) / two)
c
c   **Solve for del0 by means of Newton iteration.  If "iflag"=0 we set
c   **a default gap value for our starting guess, otherwise we use the
c   **passed value of "del0".
      if (iflag .eq. 0)  del0 = two
      gfunc = gap_func(del0,T,alpha,sigma,delta,epsilon,N_cut)
      gfunc1 = gap_func(del0+delta,T,alpha,sigma,delta,epsilon,N_cut)
c
      do i = 1, 50, 1
        del0 = del0 - delta*gfunc/(gfunc1-gfunc)
        gfunc = gap_func(del0,T,alpha,sigma,delta,epsilon,N_cut)
        if (abs(gfunc) .lt. epsilon) goto 10
        gfunc1 = gap_func(del0+delta,T,alpha,sigma,delta,epsilon,N_cut)
      end do
      write(*,*) 'Warning: del0 iters did not converged at T = ', T
      write(*,*) '         del0, gfunc = ', del0, gfunc
10    continue
c
      get_del0 = del0
c
      return
      end
c
c*********************************************************************
c
      double precision function gap_func(del0,T,alpha,sigma,delta,
     #                                   epsilon,N_cut)
c
      implicit none
      double precision del0, T, alpha, sigma, delta, epsilon
      integer N_cut
c
c   **Evaluates the gap-equation for a given "del0" and "T".  After
c   **the Matsubara sum is truncated (at N_cut), an asymptotic correction
c   **for the "high-epsilon" contribution is added.  The gap equation is
c   **written so that "gap_func=0" at the solution.
c
      integer i, j
      double precision epsn, epst, get_epst
```

```
      double precision sum, g2, arg, K, E, delk, dele
      double precision sterm1, sterm2, eps1, eps2, a, b
      double precision zero, one, two, four, pi
      parameter (zero=0.0d0, one=1.0d0, two=2.0d0, four=4.0d0)
c
      pi = acos(-one)
c
c   **Do the Matsubara sum.  The "K" and "E" are elliptic integrals needed
c   **in the calculation of "g2" (the Fermi surface average of the "tau_2"
c   **component of the quasiclassical propagator "g").
      sum = zero
      epsn = dble(2*N_cut-1)*pi*T
      epst = epsn + pi*alpha
      do i = N_cut, 1, -1
        epst = get_epst(epst,epsn,del0,alpha,sigma,delta,epsilon,1)
        arg = one / sqrt(one + (epst/del0)**2)
        K = delk(arg)
        E = dele(arg)
        g2 = two*((one/arg)*(E-K) + arg*K)
        sum = sum + g2 - (pi*del0)/(two*epsn)
        epsn = dble(2*i-3)*pi*T
      end do
c
c   **Add the asymptotic correction.
      epsn = dble(2*N_cut-1)*pi*T
      eps2 = get_epst(eps2,epsn,del0,alpha,sigma,delta,epsilon,0)
      arg = one / sqrt(one + (eps2/del0)**2)
      K = delk(arg)
      E = dele(arg)
      g2 = two*((one/arg)*(E-K) + arg*K)
      sterm2 = g2 - (pi*del0)/(two*epsn)
c
      epsn = dble(2*N_cut-7)*pi*T
      eps1 = get_epst(eps1,epsn,del0,alpha,sigma,delta,epsilon,0)
      arg = one / sqrt(one + (eps1/del0)**2)
      K = delk(arg)
      E = dele(arg)
      g2 = two*((one/arg)*(E-K) + arg*K)
      sterm1 = g2 - (pi*del0)/(two*epsn)
c
      a = ((eps2**3)*sterm2 - (eps1**3)*sterm1) / (eps2 - eps1)
      b = ((eps2**3)*eps1*sterm2 - (eps1**3)*eps2*sterm1)
     #  / (eps1 - eps2)
      sum = sum + a / (two*pi*T*(eps2 + pi*T))
     #          + b / (four*pi*T*(eps2 + pi*T)**2)
c
      gap_func = log(T)*del0 - four*T*sum
c
      return
      end
c
c*****************************************************************
c
      double precision function get_epst(epst,epsn,del0,alpha,sigma,
     #                                   delta,epsilon,iflag)
c
      implicit none
      double precision epst, epsn, del0, alpha, sigma, delta, epsilon
```

```
      integer iflag
c
c   **Calculates epsilon_tilde as a function of epsilon_sub_n (the
c   **Masubara frequency), for a given value of the gap "del0",
c   **using a simple Newton iteration technique.
c
      integer i
      double precision Fepst, dFepst, epst_func
      double precision one, pi
      parameter (one=1.0d0)
c
      pi = acos(-one)
c
c   **If "iflag"=0 we start with the normal state value for "epst",
c   **otherwise we use the value passed in.
      if (iflag .eq. 0)  epst = epsn + pi*alpha
      Fepst = epst_func(epst,epsn,del0,alpha,sigma)
      dFepst = epst_func(epst+delta,epsn,del0,alpha,sigma) - Fepst
c
      do i = 1, 50, 1
        epst = epst - delta*Fepst/dFepst
        Fepst = epst_func(epst,epsn,del0,alpha,sigma)
        if (abs(Fepst) .lt. epsilon) goto 10
        dFepst = epst_func(epst+delta,epsn,del0,alpha,sigma) - Fepst
      end do
      write(*,*) 'Warning: epst iters did not converge at epsn = ', epsn
      write(*,*) '         epst, Fepst = ', epst, Fepst
10    continue
c
      get_epst = epst
c
      return
      end
c
c*********************************************************************
c
      double precision function epst_func(epst,epsn,del0,alpha,sigma)
c
      implicit none
      double precision epst, epsn, del0, alpha, sigma
c
c   **Evaluates the "epsilon_tilde equation" for a given "epsn" and "del0".
c   **The equation is written such that "epst_func=0" at the solution.
c
      double precision tmp1, tmp2, denom, g3bar, K, delk
      double precision one, two, pi
      parameter (one=1.0d0, two=2.0d0)
c
      pi = acos(-one)
c
      tmp1 = epst / del0
      tmp2 = one / sqrt(one + tmp1**2)
      K = delk(tmp2)
      g3bar = two*tmp1*tmp2*K
c
      denom = one - sigma*(one - (g3bar/pi)**2)
      epst_func = epsn + (alpha*g3bar/denom) - epst
c
```

```
      return
      end

c ----------------------------------------------------------------------
      DOUBLE PRECISION FUNCTION DELE (XX)
c ----------------------------------------------------------------------
c***BEGIN PROLOGUE
c***PURPOSE Evaluate the complete elliptic integral of the second
c           kind E(x).
c
c  Usage:  DELE(X)
c
c***REMARK We use the Russian notation (see Gradshteyn/Ryzhik) !!!!
c
c***DESCRIPTION
c  Arguments:
c     XX      - Argument for which the function value is desired.
c               (Input)
c               XX must be greater than or equal to 0 and less than or
c               equal to 1.
c     DELE    - Function value.  (Output)
c
c***ROUTINES CALLED  NONE
c***END PROLOGUE
c ----------------------------------------------------------------------
C                                     SPECIFICATIONS FOR ARGUMENTS
      DOUBLE PRECISION XX
C                                     SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I
      DOUBLE PRECISION X, CAYSQ, EPS, ETA, SUMA, SUMB
C                                     SPECIFICATIONS FOR SAVE VARIABLES
      DOUBLE PRECISION A(10), B(10)
      SAVE       A, B
C                                     SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  DLOG
c      DOUBLE PRECISION DLOG
C
      DATA A(1)/0.1494662175718132D-03/
      DATA A(2)/0.2468503330460722D-02/
      DATA A(3)/0.8638442173604073D-02/
      DATA A(4)/0.1077063503986645D-01/
      DATA A(5)/0.7820404060959553D-02/
      DATA A(6)/0.7595093422559432D-02/
      DATA A(7)/0.115695957452954D-01/
      DATA A(8)/0.2183181167613048D-01/
      DATA A(9)/0.5680519456755915D-01/
      DATA A(10)/0.4431471805608895D00/
      DATA B(1)/0.3185919565550157D-04/
      DATA B(2)/0.9898332846225384D-03/
      DATA B(3)/0.6432146586438302D-02/
      DATA B(4)/0.1680402334636338D-01/
      DATA B(5)/0.2614501470031388D-01/
      DATA B(6)/0.3347894366576162D-01/
      DATA B(7)/0.4271789054738309D-01/
      DATA B(8)/0.5859366125553149D-01/
      DATA B(9)/0.9374999972120313D-01/
      DATA B(10)/0.2499999999999017D00/
c ----------------------------------------------------------------------
```

```
C
      X = XX*XX
      DELE = 1d+100
C
      IF (X.LT.0.0D0 .OR. X.GT.1.0D0)THEN
      Write(*,'(A18,D15.8,A18,A)')
     +     'The argument, X = ',X,', must be greater ',
     +     'than or equal to 0.0 and less than or equal to 1.0.'
      Return
      END IF
C
      EPS = 1d-14
      CAYSQ = X
C
      ETA = 1.0D0 - CAYSQ
C
      IF (ETA .GE. EPS) THEN
         SUMA = 0.0D0
         SUMB = 0.0D0
         DO 10  I=1, 10
            SUMA = (SUMA+A(I))*ETA
            SUMB = (SUMB+B(I))*ETA
   10    CONTINUE
         DELE = SUMA - DLOG(ETA)*SUMB
         DELE = DELE + 1.0D0 + EPS
      ELSE
         DELE = 1.0D0
      END IF
C
      RETURN
      END

c ----------------------------------------------------------------------
      DOUBLE PRECISION FUNCTION DELK (XX)
c ----------------------------------------------------------------------
c***BEGIN PROLOGUE
c***PURPOSE Evaluate the complete elliptic integral of the second
c           kind K(x).
c
c  Usage:  DELK(X)
c
c***REMARK We use the Russian notation (see Gradshteyn/Ryzhik) !!!!
c
c***DESCRIPTION
c  Arguments:
c     XX      - Argument for which the function value is desired.
c               (Input)
c               XX must be greater than or equal to 0 and less than or
c               equal to 1.
c     DELK    - Function value.  (Output)
c
c***ROUTINES CALLED  NONE
c***END PROLOGUE
c ----------------------------------------------------------------------
C                                    SPECIFICATIONS FOR ARGUMENTS
      DOUBLE PRECISION XX
C                                    SPECIFICATIONS FOR LOCAL VARIABLES
      INTEGER    I
```

```
      DOUBLE PRECISION X, CAYSQ, EPS, ETA, SUMA, SUMB
C                                  SPECIFICATIONS FOR SAVE VARIABLES
      DOUBLE PRECISION A(11), B(11)
      SAVE       A, B
C                                  SPECIFICATIONS FOR INTRINSICS
      INTRINSIC  DLOG
C      DOUBLE PRECISION DLOG
C
      DATA A(1)/0.1393087857006646D-03/
      DATA A(2)/0.2296634898396958D-02/
      DATA A(3)/0.8003003980649985D-02/
      DATA A(4)/0.9848929322176892D-02/
      DATA A(5)/0.6847909282624505D-02/
      DATA A(6)/0.6179627446053317D-02/
      DATA A(7)/0.8789801874555064D-02/
      DATA A(8)/0.1493801353268716D-01/
      DATA A(9)/0.3088514627130518D-01/
      DATA A(10)/0.9657359028085625D-01/
      DATA A(11)/0.1386294361119890D01/
      DATA B(1)/0.2970028096655561D-04/
      DATA B(2)/0.9215546349632497D-03/
      DATA B(3)/0.5973904299155429D-02/
      DATA B(4)/0.155309416319772D-01/
      DATA B(5)/0.2393191332311079D-01/
      DATA B(6)/0.3012484901289892D-01/
      DATA B(7)/0.373777397586236D-01/
      DATA B(8)/0.4882804190686239D-01/
      DATA B(9)/0.7031249973903835D-01/
      DATA B(10)/0.1249999999999908D00/
      DATA B(11)/0.5D00/
c -----------------------------------------------------------------------
     X = XX*XX
C
      DELK = 1d+100
C
      IF (X.LT.0.0D0 .OR. X.GT.1.0D0)THEN
        Write(*,'(A18,D15.8,A18,A )')
     +  'The argument, X = ', X, ', must be greater ',
     +  'than or equal to 0.0 and less than or equal to 1.0.'
        Return
      END IF
C
      EPS = 1d-14
      CAYSQ = X
C
      ETA = 1.0D0 - CAYSQ
C
      IF (ETA .GE. EPS) THEN
         SUMA = A(1)
         SUMB = B(1)
         DO 10  I=2, 11
            SUMA = SUMA*ETA + A(I)
            SUMB = SUMB*ETA + B(I)
   10    CONTINUE
         DELK = SUMA - DLOG(ETA)*SUMB
      ELSE
C                                  RETURN FOR SMALL ARGUMENT
         DELK = A(11) - DLOG(ETA)*B(11)
```

```
      END IF
C
      RETURN
      END
```