

Dave Boore's notes on program wholsp\_ptsrc.for for computing the complete wave field in a whole space for a point-source dislocation.

The program evaluates the following integral (slightly modified from equation 4.30 in Aki and Richards, 2002):

$$\begin{aligned}
 u_n = & \left( \frac{30\gamma_n\gamma_p\gamma_qV_q - 6v_n\gamma_p - 6\delta_{np}\gamma_qV_q}{4\pi\rho r^4} \right) M_0 \int_{r/\alpha}^{r/\beta} \tau h_p(t - \tau) d\tau \\
 & + \left( \frac{12\gamma_n\gamma_p\gamma_qV_q - 2v_n\gamma_p - 2\delta_{np}\gamma_qV_q}{4\pi\rho\alpha^2 r^2} \right) M_0 h_p\left(t - \frac{r}{\alpha}\right) \\
 & - \left( \frac{12\gamma_n\gamma_p\gamma_qV_q - 3v_n\gamma_p - 3\delta_{np}\gamma_qV_q}{4\pi\rho\beta^2 r^2} \right) M_0 h_p\left(t - \frac{r}{\beta}\right) \\
 & + \frac{2\gamma_n\gamma_p\gamma_qV_q}{4\pi\rho\alpha^3 r} M_0 \dot{h}_p\left(t - \frac{r}{\alpha}\right) - \left( \frac{2\gamma_n\gamma_p\gamma_qV_q - v_n\gamma_p - \delta_{np}\gamma_qV_q}{4\pi\rho\beta^3 r} \right) M_0 \dot{h}_p\left(t - \frac{r}{\beta}\right)
 \end{aligned}$$

where  $u_n$  is the  $n$ th component of the displacement vector, and repeated subscripts imply summation over the three components of each vector.  $\gamma$  and  $v$  are unit vectors in the direction of the ray leaving the source and the fault normal, respectively.  $h$  is the dislocation slip vector, normalized to have unit maximum amplitude.  $M_0$  is the seismic moment. Assuming that the time dependence is the same for each component, the normalized slip vector can be written as

$$h = h(t)i_h$$

Following Joyner and Spudich's (1994) notation, the vector displacement  $u$  equation above can be written as

$$u = u^N + u^{IP} + u^{IS} + u^{FP} + u^{FS}$$

where the individual terms represent the near-field, intermediate-field  $P$ -wave, intermediate-field  $S$ -wave, far-field  $P$ -wave, and far-field  $S$ -wave terms, as indicated by the superscripts.

Using a Cartesian coordinate system with  $x_1, x_2, x_3 =$  north, east, and down, the unit vectors are given by the equations implied by these Fortran subroutines:

```

* -----
  subroutine unit_vector_p(i_p, az, toa)
* Dates: 04/20/04 - Written by D. Boore

  real i_p(3)

  pi = 4.0*atan(1.0)
  dtor = pi/180.0

  azr = az*dtor
  toar = toa*dtor

  cos_az = cos(azr)
  sin_az = sin(azr)

  cos_toa = cos(toar)
  sin_toa = sin(toar)

  i_p(1) = + sin_toa*cos_az
  i_p(2) = + sin_toa*sin_az
  i_p(3) = + cos_toa

  return
  end
* -----

```

The subroutine above gives the direction unit vector of the ray as it leaves the source with an azimuth of az clockwise from north and a take-off angle toa, measured from the downward vertical direction.

```

* -----
  subroutine unit_vector_slip(i_slip, sa, da, ra)
* Dates: 04/20/04 - Written by D. Boore

  real i_slip(3)

  pi = 4.0*atan(1.0)
  dtor = pi/180.0

  sar = sa*dtor
  dar = da*dtor
  rar = ra*dtor

  cos_sa = cos(sar)
  sin_sa = sin(sar)

  cos_da = cos(dar)
  sin_da = sin(dar)

  cos_ra = cos(rar)
  sin_ra = sin(rar)

  i_slip(1) = cos_ra*cos_sa + cos_da*sin_ra*sin_sa

```

```

        i_slip(2) = cos_ra*sin_sa - cos_da*sin_ra*cos_sa
        i_slip(3) =                - sin_da*sin_ra

        return
    end
* -----
* -----
    subroutine unit_vector_fn(i_fn, sa, da, ra)
* Dates: 04/20/04 - Written by D. Boore

    real i_fn(3)

    pi = 4.0*atan(1.0)
    dtor = pi/180.0

    sar = sa*dtor
    dar = da*dtor
    rar = ra*dtor

    cos_sa = cos(sar)
    sin_sa = sin(sar)

    cos_da = cos(dar)
    sin_da = sin(dar)

    cos_ra = cos(rar)
    sin_ra = sin(rar)

    i_fn(1) = - sin_da*sin_sa
    i_fn(2) = + sin_da*cos_sa
    i_fn(3) = - cos_da

    return
end
* -----

```

In these subroutines, the strike azimuth (sa), dip angle (da), and rake angle (ra) are as defined in Figure 4.20 of Aki and Richards (2002)

I use Joyner and Spudich's trick for evaluating the near-field term. Representing the integral in the near-field terms as

$$I = \int_{r/\alpha}^{r/\beta} \tau h(t - \tau) d\tau$$

the acceleration requires  $\ddot{i}$ . This is

$$\ddot{i} = h(t - t_\alpha) - h(t - t_\beta) + t_\alpha \dot{h}(t - t_\alpha) - t_\beta \dot{h}(t - t_\beta),$$

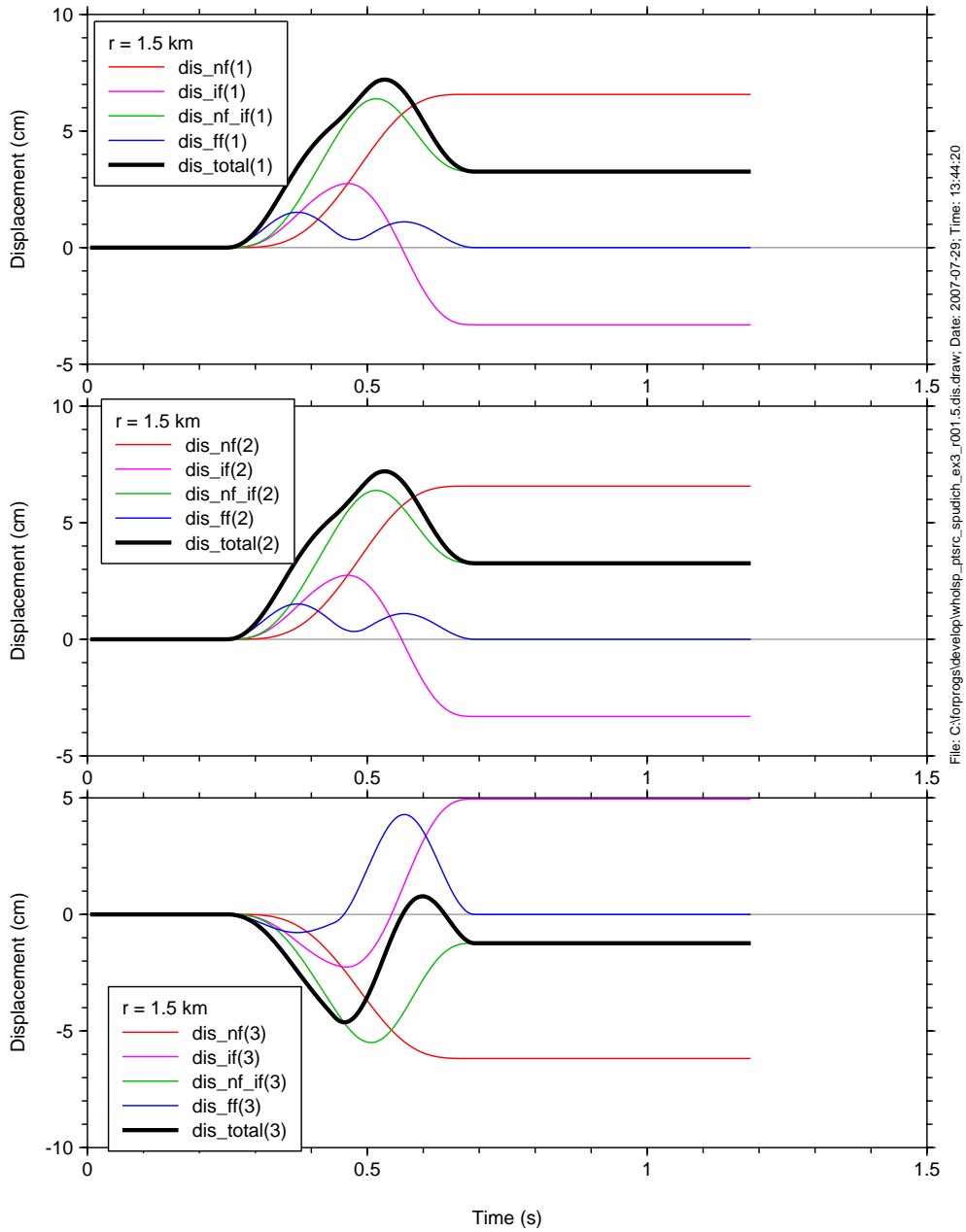
where  $t_\alpha$  and  $t_\beta$  are travel times for  $P$  waves and  $S$  waves.

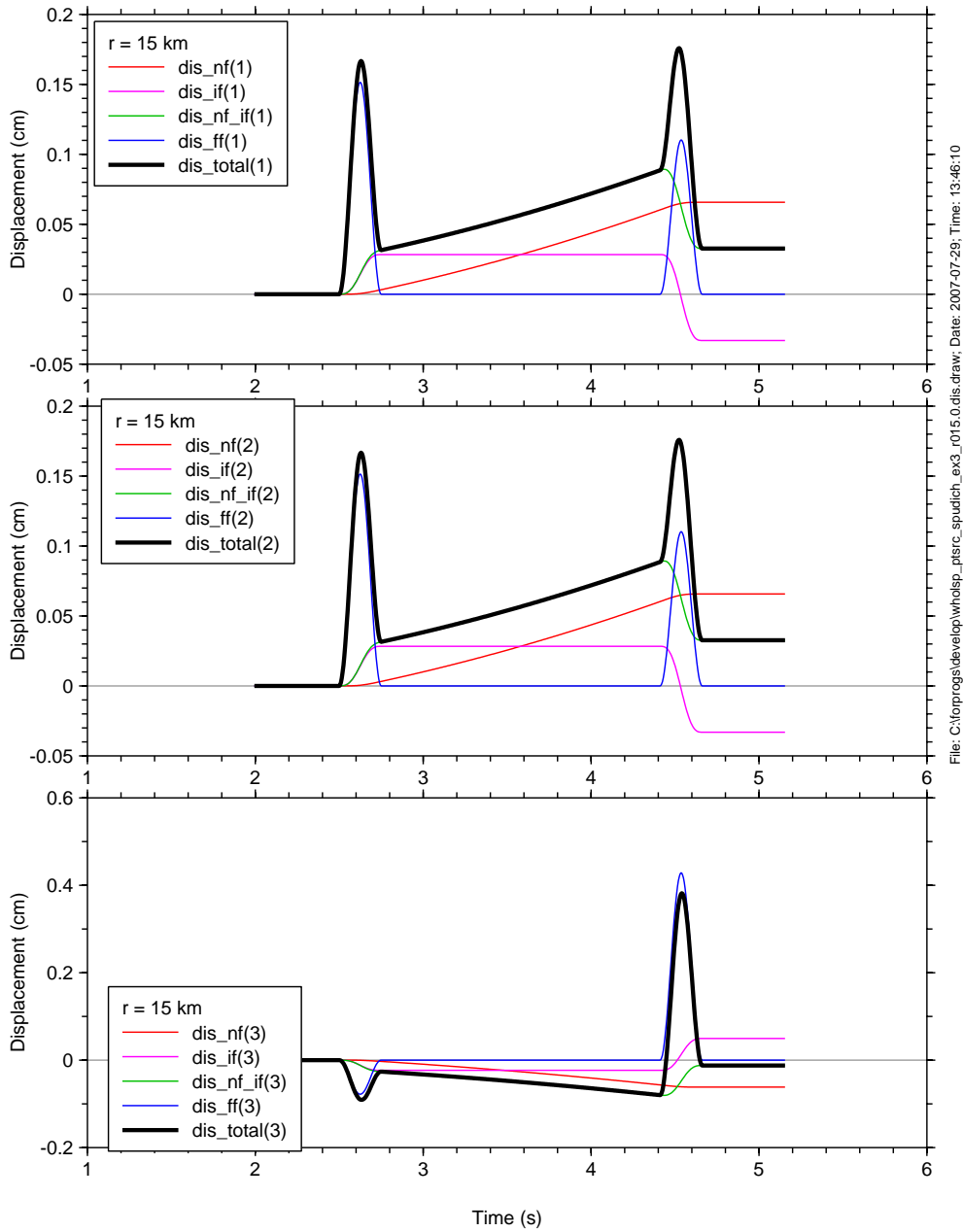
Thus, ground acceleration requires  $h(t)$ ,  $\dot{h}(t)$ ,  $\ddot{h}(t)$ , and  $\ddot{\ddot{h}}(t)$ . In addition, ground velocity and ground displacement require the first and second integrals of  $h(t)$ . For convenience in deriving analytical expressions for the derivatives and integrals of  $h(t)$ , I use the following equation for  $h(t)$ :

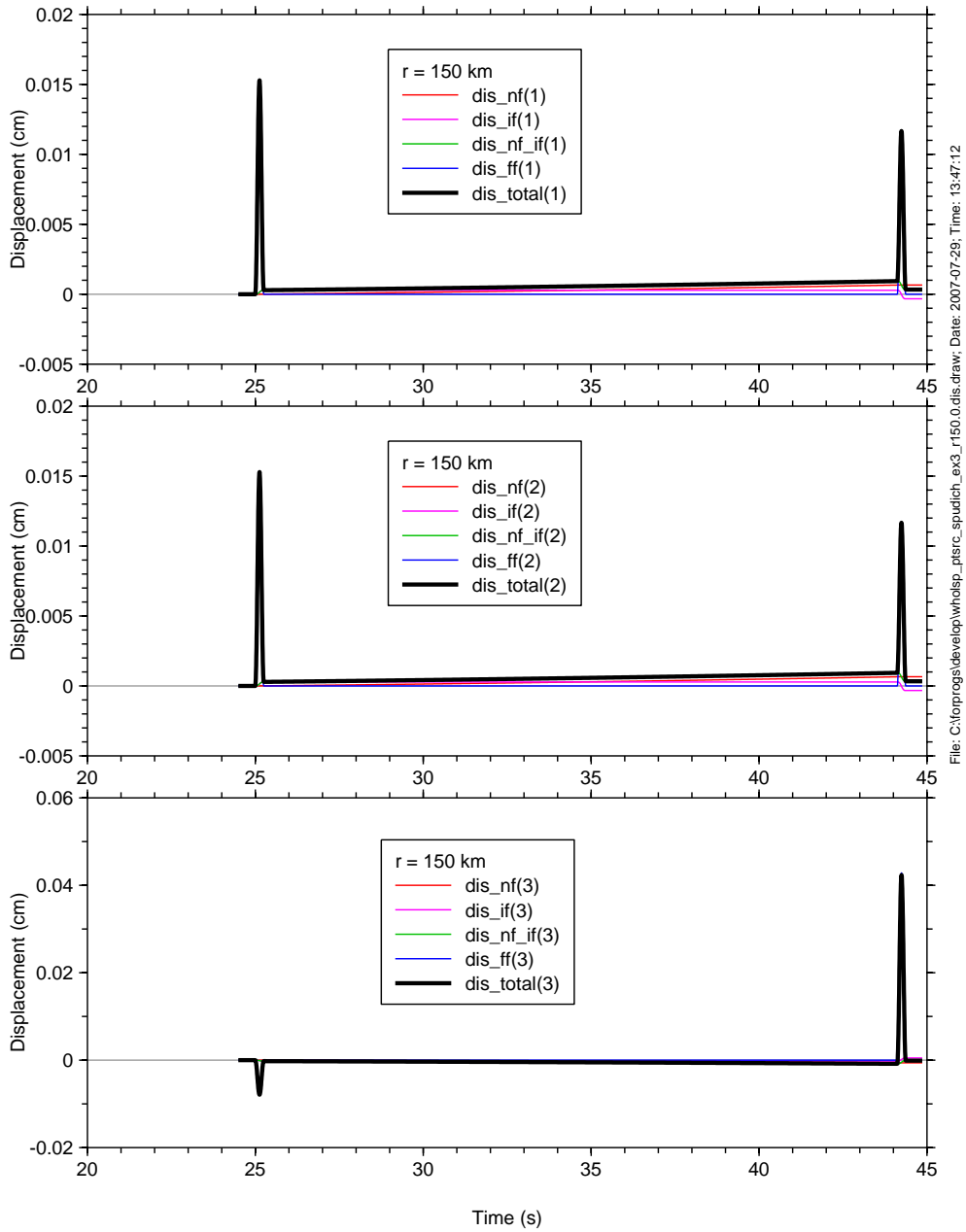
$$h(t) = \begin{cases} 0, & t < 0 \\ \frac{1}{T} \left[ t - \frac{T}{2\pi} \sin(2\pi t / T) \right], & 0 \leq t < T \\ 1, & T \leq t \end{cases}$$

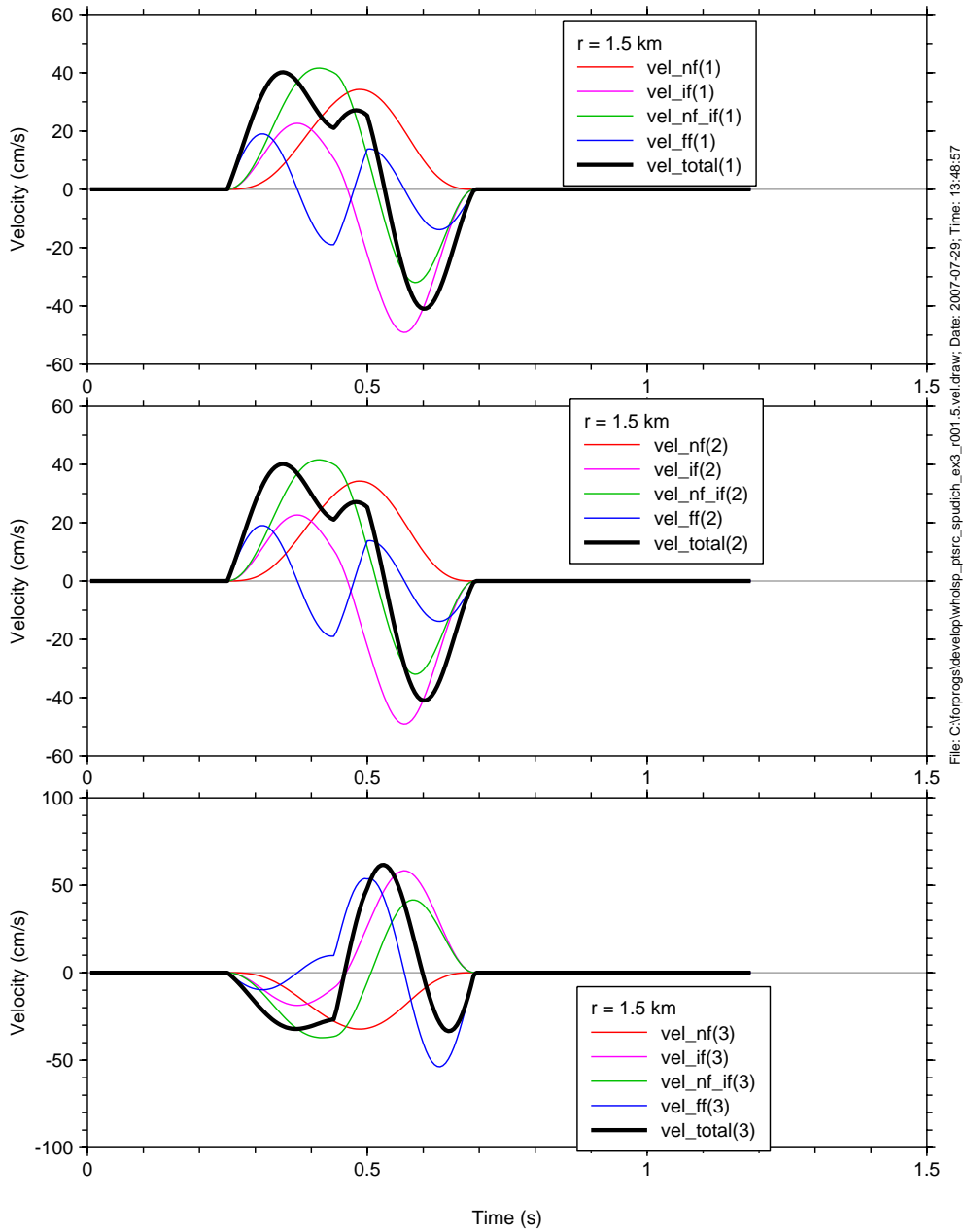
This form of the displacement was used by Makris and Chang (2000). For acceleration it corresponds to a single cycle of a sine wave. It is easy to differentiate and integrate this function analytically, and thus it is easy to write a program to evaluate the complete wavefield for a point source dislocation in a whole space. My program for doing this is `wholsp_ptsrc`; a listing of the program follows the references. Here are examples of the waveform for a particular case, for three components and displacements and velocities at three distances. The individual contributions of the near-, intermediate, and far-field terms are shown (as well as the sum of the near- and intermediate field terms). The control file used for these calculations is given here (for a distance of 1.5 km):

```
! Control file for program WholSp_PtSrc
!
!Enter Vs, Vp, dens:
  3.4 6.0 2.7
!Enter sa,da,ra(degrees):
  0 90 0
!Enter az2sta, r2sta, toa, tp_minus_tstart,
tend_minus_ts_plus_pw, dt:
  45.0 1.5 110.0 0.5 0.5 0.005
!Enter M, pulse width
  5.0 0.25
!Enter stem name of output file:
  wholsp_ptsrc_spudich_ex3
```

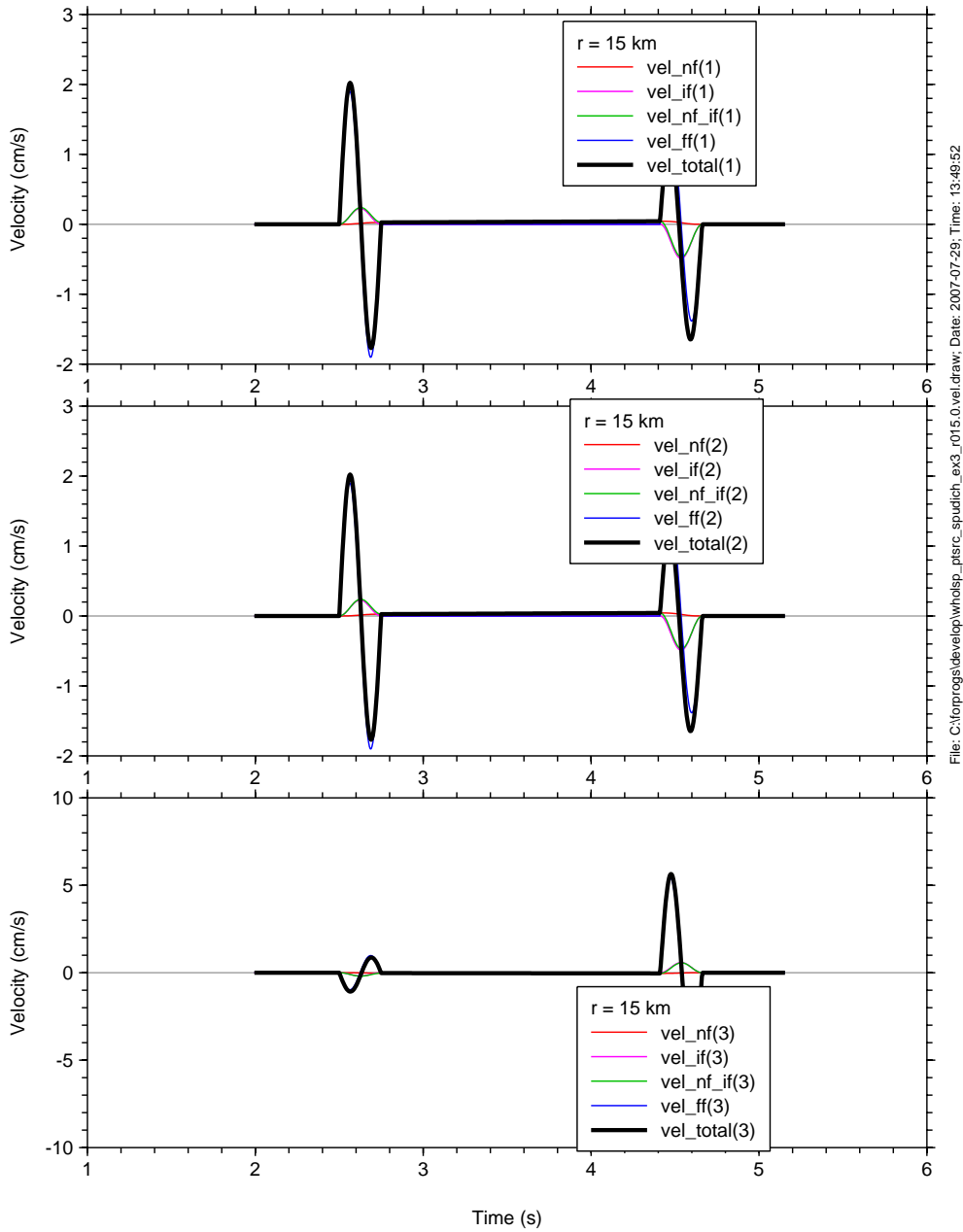


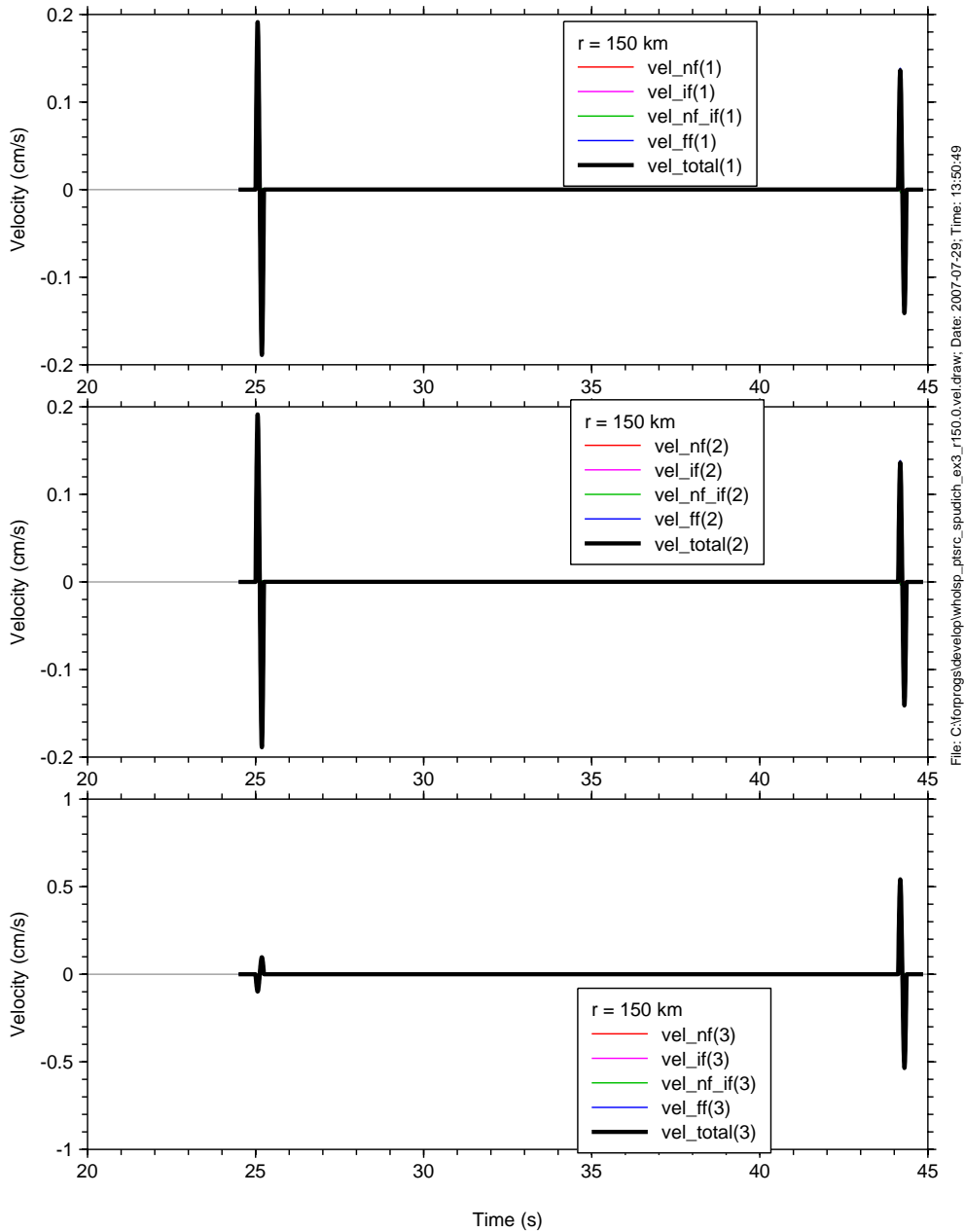












## References

Aki, K. and P. G. Richards (2002). *Quantitative Seismology: Second Edition*, 700 p., University Science Books, Sausalito, California.

Joyner, W. B., and P. Spudich (1994). Including near-field terms in the isochrone integration method for application to finite-fault or Kirchoff boundary integral problems, *Bull. Seism. Soc. Am.* **84**, 1260—1265.

Makris, N., and S.-P. Chang (2000). Effect of viscous, viscoplastic, and friction damping on the response of seismic isolated structures, *Earthquake Engineering and Structural Dynamics* **29**, 85--107.

```

* -----
      Program WholSp_PtSrc

* Compute seismograms for a point source in a whole space,
including
* near-, intermediate-, and far-field terms. The assumed
slip is
* equivalent to a single cycle of a sine wave.

*! Control file for program WholSp_PtSrc
*!
*!Enter Vs, Vp, dens:
*   3.5 6.1 2.7
*!Enter sa,da,ra(degrees):
*   0.0 90.0 0.0
*!Enter az2sta, r2sta, toa, tp_minus_tstart,
tend_minus_ts_plus_pw, dt:
*       20.0   10.0 90.0       5.0   5.0 0.05
*!Enter M, pulse width
*   7.0 10.0
*!Enter stem name of output file:
*   test_wholsp_ptsrc

* Dates: 07/20/07 - Written by D. Boore, using equations in
*               Aki and Richards (text) and Joyner and
Spudich (BSSA)

!       implicit real*8 (a - h, o - z)

!       real*4 acc(*), omega, damp, dt, rd, rv,aa

      character  ctl_cmmnts(100)*79
      character  f_ctl*100, f_stem*100,
:              f_acc*100, f_vel*100, f_dis*100

      logical f_ctl_exist
      real i_slip(3), i_fn(3), i_p(3)
      real radpat_nf(3), radpat_if_p(3), radpat_if_s(3),
:              radpat_ff_p(3), radpat_ff_s(3)
      real acc_nf(3), acc_if(3), acc_nf_if(3), acc_ff(3),
acc_total(3)
      real vel_nf(3), vel_if(3), vel_nf_if(3), vel_ff(3),
vel_total(3)

```

```

      real dis_nf(3), dis_if(3), dis_nf_if(3), dis_ff(3),
dis_total(3)

      pi = 4.0*atan(1.)

      f_ctl_exist = .false.
      do while (.not. f_ctl_exist)
        f_ctl = ' '
        write(*, '(a\)\')
      :   ' Enter name of control file (cr =
wholsp_ptsrc.ctl): '
        read(*, '(a)\') f_ctl
        if (f_ctl .eq. ' ') f_ctl = 'wholsp_ptsrc.ctl'
        call trim_c(f_ctl,nc_f_ctl)
        inquire(file=f_ctl(1:nc_f_ctl), exist=f_ctl_exist)
        if (.not. f_ctl_exist) then
          write(*,'(a)\') ' ***** FILE DOES NOT EXIST
***** '
        end if
      end do

      call get_lun(nu_ctl)

      open(unit=nu_ctl,file=f_ctl(1:nc_f_ctl),status='unknown')

      call skipcmnt(nu_ctl,ctl_cmmnts, nc_ctl_cmmnts)
      read(nu_ctl,*) vs, vp, dens

      call skipcmnt(nu_ctl,ctl_cmmnts, nc_ctl_cmmnts)
      read(nu_ctl,*) sa, da, ra

      dd = sa + 90.0

      call skipcmnt(nu_ctl,ctl_cmmnts, nc_ctl_cmmnts)
      read(nu_ctl,*) az2sta, r2sta, toa,
:      tp_minus_tstart, tend_minus_ts_plus_pw, dt

      call skipcmnt(nu_ctl,ctl_cmmnts, nc_ctl_cmmnts)
      read(nu_ctl,*) amag, pw

      amom = 10.0**(1.5*amag+16.05)

      call skipcmnt(nu_ctl,ctl_cmmnts, nc_ctl_cmmnts)
      f_stem = ' '
      read(nu_ctl,'(a)\') f_stem

      r2stacm = r2sta * 1.0e+05
      vpcgs = vp * 1.0e+05
      vscgs = vs * 1.0e+05

      tp = r2sta/vp
      ts = r2sta/vs

```

```

        afctr_nf = amom/(4.0 * pi * dens * r2stacm**4)
        afctr_if_p = amom/(4.0 * pi * dens * vpcgs**2 *
r2stacm**2)
        afctr_if_s = amom/(4.0 * pi * dens * vscgs**2 *
r2stacm**2)
        afctr_ff_p = amom/(4.0 * pi * dens * vpcgs**3 *
r2stacm)
        afctr_ff_s = amom/(4.0 * pi * dens * vscgs**3 *
r2stacm)

        call unit_vector_slip(i_slip, sa, da, ra)
        call unit_vector_fn(i_fn, sa, da, ra)
        call unit_vector_p(i_p, az2sta, toa)

        do i = 1, 3

            radpat_nf(i) =
:           30 *      i_p(i) * dot(i_p,i_fn,3) *
dot(i_p,i_slip,3)
:           - 6 *      i_fn(i) * dot(i_p,i_slip,3)
:           - 6 * i_slip(i) * dot(i_p,i_fn,3)

            radpat_if_p(i) =
:           12 *      i_p(i) * dot(i_p,i_fn,3) *
dot(i_p,i_slip,3)
:           - 2 *      i_fn(i) * dot(i_p,i_slip,3)
:           - 2 * i_slip(i) * dot(i_p,i_fn,3)

            radpat_if_s(i) =
:           -12 *      i_p(i) * dot(i_p,i_fn,3) *
dot(i_p,i_slip,3)
:           + 3 *      i_fn(i) * dot(i_p,i_slip,3)
:           + 3 * i_slip(i) * dot(i_p,i_fn,3)

            radpat_ff_p(i) =
:           2 *      i_p(i) * dot(i_p,i_fn,3) *
dot(i_p,i_slip,3)

            radpat_ff_s(i) =
:           - 2 *      i_p(i) * dot(i_p,i_fn,3) *
dot(i_p,i_slip,3)
:           +          i_fn(i) * dot(i_p,i_slip,3)
:           +          i_slip(i) * dot(i_p,i_fn,3)

        end do

        call trim_c(f_stem,nc_f_stem)

        f_acc = ' '
        f_acc = f_stem(1:nc_f_stem)//'.acc.out'
        call trim_c(f_acc, nc_f_acc)

```

```

        call get_lun(nu_acc)

open(unit=nu_acc,file=f_acc(1:nc_f_acc),status='unknown')

        write(nu_acc,'(5x,a, 3(4x,a, 4x,a, 1x,a, 4x,a,
1x,a))')
        :
        : 'time',
        : 'acc_nf(1)', 'acc_if(1)', 'acc_nf_if(1)',
        : 'acc_ff(1)', 'acc_total(1)',
        : 'acc_nf(2)', 'acc_if(2)', 'acc_nf_if(2)',
        : 'acc_ff(2)', 'acc_total(2)',
        : 'acc_nf(3)', 'acc_if(3)', 'acc_nf_if(3)',
        : 'acc_ff(3)', 'acc_total(3)'

        f_vel = ' '
        f_vel = f_stem(1:nc_f_stem)//'.vel.out'
        call trim_c(f_vel, nc_f_vel)

        call get_lun(nu_vel)

open(unit=nu_vel,file=f_vel(1:nc_f_vel),status='unknown')

        write(nu_vel,'(5x,a, 3(4x,a, 4x,a, 1x,a, 4x,a,
1x,a))')
        :
        : 'time',
        : 'vel_nf(1)', 'vel_if(1)', 'vel_nf_if(1)',
        : 'vel_ff(1)', 'vel_total(1)',
        : 'vel_nf(2)', 'vel_if(2)', 'vel_nf_if(2)',
        : 'vel_ff(2)', 'vel_total(2)',
        : 'vel_nf(3)', 'vel_if(3)', 'vel_nf_if(3)',
        : 'vel_ff(3)', 'vel_total(3)'

        f_dis = ' '
        f_dis = f_stem(1:nc_f_stem)//'.dis.out'
        call trim_c(f_dis, nc_f_dis)

        call get_lun(nu_dis)

open(unit=nu_dis,file=f_dis(1:nc_f_dis),status='unknown')

        write(nu_dis,'(5x,a, 3(4x,a, 4x,a, 1x,a, 4x,a,
1x,a))')
        :
        : 'time',
        : 'dis_nf(1)', 'dis_if(1)', 'dis_nf_if(1)',
        : 'dis_ff(1)', 'dis_total(1)',
        : 'dis_nf(2)', 'dis_if(2)', 'dis_nf_if(2)',
        : 'dis_ff(2)', 'dis_total(2)',
        : 'dis_nf(3)', 'dis_if(3)', 'dis_nf_if(3)',
        : 'dis_ff(3)', 'dis_total(3)'

```

```

nstart = int((tp-tp_minus_tstart)/dt) - 1
if (nstart .le. 0) then
  nstart = 1
end if
np = int(tp/dt)
ns = int(ts/dt)
npw = int(pw/dt)
nend = ns + npw + int(tend_minus_ts_plus_pw/dt) - 1

print *, ' tp, tp_minus_tstart, dt, nstart = ',
:      tp, tp_minus_tstart, dt, nstart
print *, ' ts, pw, tend_minus_ts_plus_pw, dt, nend = ',
:      ts, pw, tend_minus_ts_plus_pw, dt, nend

do i = nstart, np - 1

  time = float(i)*dt

  do j = 1, 3
    acc_nf(j) = 0.0
    acc_if(j) = 0.0
    acc_nf_if(j) = 0.0
    acc_ff(j) = 0.0
    acc_total(j) = 0.0
    vel_nf(j) = 0.0
    vel_if(j) = 0.0
    vel_nf_if(j) = 0.0
    vel_ff(j) = 0.0
    vel_total(j) = 0.0
    dis_nf(j) = 0.0
    dis_if(j) = 0.0
    dis_nf_if(j) = 0.0
    dis_ff(j) = 0.0
    dis_total(j) = 0.0
  end do

  write(nu_acc, '(1x,f8.3, 1p, 15(2x,e11.4))')
:      time, (acc_nf(j), acc_if(j), acc_nf_if(j),
acc_ff(j),
:      acc_total(j), j = 1, 3)

  write(nu_vel, '(1x,f8.3, 1p, 15(2x,e11.4))')
:      time, (vel_nf(j), vel_if(j), vel_nf_if(j),
vel_ff(j),
:      vel_total(j), j = 1, 3)

  write(nu_dis, '(1x,f8.3, 1p, 15(2x,e11.4))')
:      time, (dis_nf(j), dis_if(j), dis_nf_if(j),
dis_ff(j),
:      dis_total(j), j = 1, 3)

```

```

end do

do i = np, nend
  time = float(i)*dt
  do j = 1, 3
    acc_nf(j) =
      :      afctr_nf      * radpat_nf(j)      * anf(time, tp,
ts, pw)

    acc_if(j) =
      :      afctr_if_p * radpat_if_p(j) * aif(time, tp,
pw)
      :      + afctr_if_s * radpat_if_s(j) * aif(time, ts,
pw)

    acc_nf_if(j) = acc_nf(j) + acc_if(j)

    acc_ff(j) =
      :      afctr_ff_p * radpat_ff_p(j) * aff(time, tp,
pw)
      :      + afctr_ff_s * radpat_ff_s(j) * aff(time, ts,
pw)

    acc_total(j) = acc_nf_if(j) + acc_ff(j)

    vel_nf(j) =
      :      afctr_nf      * radpat_nf(j)      * vnf(time, tp,
ts, pw)

    vel_if(j) =
      :      afctr_if_p * radpat_if_p(j) * vif(time, tp,
pw)
      :      + afctr_if_s * radpat_if_s(j) * vif(time, ts,
pw)

    vel_nf_if(j) = vel_nf(j) + vel_if(j)

    vel_ff(j) =
      :      afctr_ff_p * radpat_ff_p(j) * vff(time, tp,
pw)
      :      + afctr_ff_s * radpat_ff_s(j) * vff(time, ts,
pw)

    vel_total(j) = vel_nf_if(j) + vel_ff(j)

    dis_nf(j) =
      :      afctr_nf      * radpat_nf(j)      * dnf(time, tp,
ts, pw)

    dis_if(j) =

```



```

pw)      :      afctr_if_p * radpat_if_p(j) * dif(time, tp,
pw)      :      + afctr_if_s * radpat_if_s(j) * dif(time, ts,
      dis_nf_if(j) = dis_nf(j) + dis_if(j)
      dis_ff(j) =
pw)      :      afctr_ff_p * radpat_ff_p(j) * dff(time, tp,
pw)      :      + afctr_ff_s * radpat_ff_s(j) * dff(time, ts,
      dis_total(j) = dis_nf_if(j) + dis_ff(j)
      end do
      write(nu_acc, '(1x,f8.3, 1p, 15(2x,e11.4))')
      :      time, (acc_nf(j), acc_if(j), acc_nf_if(j),
acc_ff(j),
      :      acc_total(j), j = 1, 3)
      write(nu_vel, '(1x,f8.3, 1p, 15(2x,e11.4))')
      :      time, (vel_nf(j), vel_if(j), vel_nf_if(j),
vel_ff(j),
      :      vel_total(j), j = 1, 3)
      write(nu_dis, '(1x,f8.3, 1p, 15(2x,e11.4))')
      :      time, (dis_nf(j), dis_if(j), dis_nf_if(j),
dis_ff(j),
      :      dis_total(j), j = 1, 3)
      end do
      close(nu_acc)
      close(nu_vel)
      close(nu_dis)
      close(nu_ctl)
      stop
      end

```

\* -----

\* -----

```
function anf(t, tp, ts, pw)
```

\* Near-field contribution to acceleration

\* t = time from origin time

\* r = hypocentral distance

\* vs = S-wave velocity

\* vp = P-wave velocity

\* pw = duration of the pulse (pulse width)

```

    pp = t - tp
    ps = t - ts

    anf =          h(pp, pw) -          h(ps, pw)
:      + tp * hdot(pp, pw) - ts * hdot(ps, pw)

    return
end
* -----
* -----
    function aif(t, tps, pw)

* Intermediate-field contribution to acceleration

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    aif = hdotdot(p, pw)

    return
end
* -----
* -----
    function aff(t, tps, pw)

* Far-field contribution to acceleration

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    aff = hdotdotdot(p, pw)

    return
end
* -----
* -----
    function vnf(t, tp, ts, pw)

* Near-field contribution to velocity

* t = time from origin time
* r = hypocentral distance

```

```

* vs = S-wave velocity
* vp = P-wave velocity
* pw = duration of the pulse (pulse width)

    pp = t - tp
    ps = t - ts

    vnf =          hint(pp, pw) -          hint(ps, pw)
:      + tp * h(pp, pw) - ts * h(ps, pw)

    return
end
* -----
* -----
    function vif(t, tps, pw)

* Intermediate-field contribution to velocity

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    vif = hdot(p, pw)

    return
end
* -----
* -----
    function vff(t, tps, pw)

* Far-field contribution to velocity

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    vff = hdotdot(p, pw)

    return
end
* -----
* -----
    function dnf(t, tp, ts, pw)

* Near-field contribution to displacement

```

```

* t = time from origin time
* r = hypocentral distance
* vs = S-wave velocity
* vp = P-wave velocity
* pw = duration of the pulse (pulse width)

    pp = t - tp
    ps = t - ts

    dnf =          hintint(pp, pw) - hintint(ps, pw)
:      + tp * hint(pp, pw) - ts * hint(ps, pw)

    return
end
* -----
* -----
    function dif(t, tps, pw)

* Intermediate-field contribution to displacement

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    dif = h(p, pw)

    return
end
* -----
* -----
    function dff(t, tps, pw)

* Far-field contribution to displacement

* Do not need to know whether this is for the P- or the S-
wave

* "tps" = tp or ts

    p = t - tps

    dff = hdot(p, pw)

    return
end
* -----
* -----

```

```

function h(p, pw)
real h

pi = 4.0*atan(1.0)
twopi = 2.0*pi
twopi_pw = twopi/pw

if (p .lt. 0.0) then
  h = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
  h = (p-sin(twopi_pw*p)/twopi_pw)/pw
else
  h = 1.0
end if

return
end
* -----
* -----
function hdot(p, pw)
real hdot

pi = 4.0*atan(1.0)
twopi = 2.0*pi
twopi_pw = twopi/pw

if (p .lt. 0.0 ) then
  hdot = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
  hdot = (1 - cos(twopi_pw*p))/pw
else
  hdot = 0.0
end if

return
end
* -----
* -----
function hdotdot(p, pw)
real hdot

pi = 4.0*atan(1.0)
twopi = 2.0*pi
twopi_pw = twopi/pw

if (p .lt. 0.0 ) then
  hdotdot = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
  hdotdot = (twopi_pw * sin(twopi_pw*p))/pw
else
  hdotdot = 0.0
end if

```

```

    return
end
* -----
* -----
function hdotdotdot(p, pw)
real hdotdotdot

pi = 4.0*atan(1.0)
twopi = 2.0*pi
twopi_pw = twopi/pw

if (p .lt. 0.0 ) then
    hdotdotdot = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
    hdotdotdot = ((twopi_pw)**2 * cos(twopi_pw*p))/pw
else
    hdotdotdot = 0.0
end if

return
end
* -----
* -----
function hint(p, pw)
real hint

pi = 4.0*atan(1.0)
twopi = 2.0*pi
twopi_pw = twopi/pw

if (p .lt. 0.0 ) then
    hint = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
    hint = (p**2/2.0 - (1 -
cos(twopi_pw*p))/twopi_pw**2)/pw
else
    hint = p - pw/2.0
end if

return
end
* -----
* -----
function hintint(p, pw)
real hintint

pi = 4.0*atan(1.0)
twopi = 2.0*pi

```

```

twopi_pw = twopi/pw

if (p .lt. 0.0 ) then
  hintint = 0.0
else if (p .ge. 0.0 .and. p .le. pw) then
  hintint = (- p/twopi_pw**2 + p**3/6.0
:           + (sin(twopi_pw*p))/twopi_pw**3)/pw
else
  hintint = (1.0/6.0 - 1.0/twopi**2) * pw**2 + p*(p -
pw)/2.0
end if

```

```

return
end

```

\* -----

\* -----

```

function dot(a,b,n)

```

c Computes dot product between a and b

c Written by: Dave Boore  
c Dates: 10/5/88 - Created

```

dimension a(1), b(1)

```

```

dot = 0.0

```

```

do 1 i = 1, n

```

```

1 dot = dot + a(i) * b(i)

```

```

return
end

```

\* -----

\* -----

```

subroutine unit_vector_fn(i_fn, sa, da, ra)

```

\* Dates: 04/20/04 - Written by D. Boore

```

real i_fn(3)

```

```

pi = 4.0*atan(1.0)
dtor = pi/180.0

```

```

sar = sa*dtor
dar = da*dtor
rar = ra*dtor

```

```

cos_sa = cos(sar)
sin_sa = sin(sar)

```

```

cos_da = cos(dar)
sin_da = sin(dar)

cos_ra = cos(rar)
sin_ra = sin(rar)

i_fn(1) = - sin_da*sin_sa
i_fn(2) = + sin_da*cos_sa
i_fn(3) = - cos_da

return
end
* -----
* -----
subroutine unit_vector_p(i_p, az, toa)
* Dates: 04/20/04 - Written by D. Boore

real i_p(3)

pi = 4.0*atan(1.0)
dtor = pi/180.0

azr = az*dtor
toar = toa*dtor

cos_az = cos(azr)
sin_az = sin(azr)

cos_toa = cos(toar)
sin_toa = sin(toar)

i_p(1) = + sin_toa*cos_az
i_p(2) = + sin_toa*sin_az
i_p(3) = + cos_toa

return
end
* -----
* -----
subroutine unit_vector_slip(i_slip, sa, da, ra)
* Dates: 04/20/04 - Written by D. Boore

real i_slip(3)

pi = 4.0*atan(1.0)
dtor = pi/180.0

sar = sa*dtor

```



```

dar = da*dtor
rar = ra*dtor

cos_sa = cos(sar)
sin_sa = sin(sar)

cos_da = cos(dar)
sin_da = sin(dar)

cos_ra = cos(rar)
sin_ra = sin(rar)

i_slip(1) = cos_ra*cos_sa + cos_da*sin_ra*sin_sa
i_slip(2) = cos_ra*sin_sa - cos_da*sin_ra*cos_sa
i_slip(3) =                - sin_da*sin_ra

return
end

```

\* -----

```

include '\forprogs\get_lun.for'
include '\forprogs\trim_c.for'
include '\forprogs\upstr.for'
include '\forprogs\skipcmnt.for'

```