```
pro angular_distribution, input, output, npack, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Version History
;;  3.1: 1/5/2011
;;   * Changing the way the altitude is chosen. sin(alt) is evenly distributed between
;;     minimum and maximum angles.
;;  3.0: 7/19/2010
;;   * revise for new structure architecture
;;  2.2: 17 November 2009
;;    * changed the way the costheta distrubution works.
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

common Constants

AngularDist = input.AngularDist
vv = *output.vx0
case strlowcase(input.AngularDist.type) of
  'none':
  'radial': begin
    rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
    alt = replicate(!pi/2., npack) ;; set all packets going directly up
    az = fltarr(npack)
    end
  'isotropic': begin
    ;; Choose the altitude -- f(alt) = cos(alt)
    aa = sin(AngularDist.altitude)
    sinalt = random_nr(seed=seed, npack) * (aa[1]-aa[0]) + aa[0]
    alt = asin(sinalt)

    ;; Choose the longitude -- f(lon) = 1 / (lonmax-lonmin)
    if ((AngularDist.azimuth)[0] GT (AngularDist.azimuth)[1]) $
      then m = [(AngularDist.azimuth)[0], (AngularDist.azimuth)[1]+2*!pi] $
      else m = AngularDist.azimuth
    az = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)
    end
  'costheta': begin
    aa = sin(AngularDist.altitude)
    sinalt = dindgen(1001)/1000. * (aa[1]-aa[0]) + aa[0]
    f_sinalt = sinalt^AngularDist.n
    sinalt = RandomDeviates_1d(sinalt, f_sinalt, npack)
    alt = asin(sinalt)

    if ((AngularDist.azimuth)[0] GT (AngularDist.azimuth)[1]) $
      then m = [(AngularDist.azimuth)[0], (AngularDist.azimuth)[1]+2*!pi] $
      else m = AngularDist.azimuth
    az = (m[0] + (m[1]-m[0]) * random_nr(seed=seed, npack)) mod (2*!pi)
    end
endcase
```

```
52
53  ;; Find the velocity components in coordinate system centered on the packet
54  v_rad = sin(alt)              ;; Radial component of velocity
55  v_tan0 = cos(alt) * cos(az)  ;; Component along latitude line (points east)
56  v_tan1 = cos(alt) * sin(az)   ;; Component along longitude line (points to NP)
57  ;; Now rotate to the proper surface point
58  ;; v_ren = M # v_xyz => v_xyz = invert(M) # v_ren
59
60  rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
61  x0 = *output.x0/rr & y0 = *output.y0/rr & z0 = *output.z0/rr
62
63  *output.vx0 = dblarr(npack)
64  *output.vy0 = dblarr(npack)
65  *output.vz0 = dblarr(npack)
66  for i=0L,npack-1 do begin
67    rad = [x0[i], y0[i], z0[i]]
68    east = [y0[i], -x0[i], 0]
69    north = [-z0[i]*x0[i], -z0[i]*y0[i], x0[i]^2+y0[i]^2]
70
71    east /= sqrt(total(east*east))
72    north /= sqrt(total(north*north))
73
74    v0 = v_tan0[i]*north + v_tan1[i]*east + v_rad[i]*rad
75    if (abs(total(v0*v0))-1 GT 1e-3) then stop
76    (*output.vx0)[i] = v0[0] * vv[i]
77    (*output.vy0)[i] = v0[1] * vv[i]
78    (*output.vz0)[i] = v0[2] * vv[i]
79  endfor
80
81  end
```