

```

1 pro speed_distribution, input, output, npack, seed
2
3 #####
4 ;;
5 ;; Version History:
6 ;;   3.2: 12/13/2010
7 ;;   * changes to random deviates methods
8 ;;   3.1: 9/15/2010
9 ;;   * added option for "user defined" speed distribution
10 ;;   3.0: 7/19/2010
11 ;;   * Revised with new structure architecture
12 ;;   2.3: 1/12/10
13 ;;   * Added thermal distribution where velocity distribution depends on local
14 ;;     surface temperature
15 ;;   2.2: 3/3/09
16 ;;   * Added support for Weibull distribution [removed 1/20/10]
17 ;;   2.1 --
18 ;;   * broke velocity_distribution_2.0 into separate speed, angular, and perturbation
19 ;;     components
20 ;;
21 ;; Returns either an array of speeds in *output.vx0 or the full velocity
22 ;; in *output.vx0, *output.vy0, *output.vz0
23 ;; -- Right now only returns full velocity if "circular orbits" speed distribution is
24 ;;    chosen
25 ;;
26 ;; All speeds are returned in units of Rplan
27 ;;
28 #####
29
30 common Constants
31
32 SpeedDist = input.SpeedDist
33 case strlowercase(SpeedDist.type) of
34 'gaussian': begin
35   if (SpeedDist.sigma EQ 0) $
36     then *output.vx0 = replicate(SpeedDist.vprob, npack) $
37     else *output.vx0 = RandomGaussian(npack, SpeedDist.vprob, SpeedDist.sigma)
38   end
39 'trigaussian': begin
40   if (SpeedDist.vxsigma EQ 0) $
41     then *output.vx0 = replicate(SpeedDist.vxprob, npack) $
42     else *output.vx0 = RandomGaussian(npack, SpeedDist.vxprob, SpeedDist.vxsigma)
43
44   if (SpeedDist.vysigma EQ 0) $
45     then *output.vy0 = replicate(SpeedDist.vyprob, npack) $
46     else *output.vy0 = RandomGaussian(npack, SpeedDist.vyprob, SpeedDist.vysigma)
47
48   if (SpeedDist.vzsigma EQ 0) $
49     then *output.vz0 = replicate(SpeedDist.vzprob, npack) $
50     else *output.vz0 = RandomGaussian(npack, SpeedDist.vzprob, SpeedDist.vzsigma)
51   end

```

```

52 'dolsfunction': begin
53   stop
54   velocity = findgen(1001.)/100.
55   f_v = dolsdist(velocity, SpeedDist.dols0, SpeedDist.dols1, input.options.atom)
56   *output.vx0 = RandomDeviates_ld(velocity, f_v, npack) ;; km/s
57 end
58 'sputtering': begin
59   velocity = findgen(5000)/100.+1
60   f_v = sputdist(velocity, SpeedDist.U, SpeedDist.alpha, SpeedDist.beta,$
61     input.options.atom)
62   *output.vx0 = RandomDeviates_ld(velocity, f_v, npack) ;; km/s
63 end
64 'maxwellian': begin
65   if (SpeedDist.temperature NE 0) then begin
66     ;; Use a constant surface temperature
67     v_th = sqrt(2*SpeedDist.temperature*!const.kb/atomicmass(input.options.atom)) /1e5
68     velocity = findgen(1001)/1000 * v_th*5 & velocity = velocity[1:*]
69     f_v = MaxwelllianDist(velocity, SpeedDist.temperature, input.options.atom)
70     *output.vx0 = RandomDeviates_ld(velocity, f_v, npack) ;; km/s
71   endif else begin
72     ;; Use a surface temperature map
73     rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
74     SZA = acos(-*output.y0/rr) ;; cos(SZA) = [0,-1,0]·[x,y,z]/r = -y/r
75     q = where(finite(SZA) EQ 0, nq) & if (nq NE 0) then stio
76     surftemp = surface_temperature(input.geometry, SZA)
77
78     nt = 101 & np = 1001
79     temperature = dindgen(nt)/(nt-1)*(max(surftemp)-min(surftemp)) + min(surftemp)
80     v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
81     prob = dindgen(np)/(np-1)
82     vgrid = dblarr(nt,np)
83     for i=0,nt-1 do begin
84       ;; Produces the velocity as fn of T and cumulative value.
85       ;; Given T and random P, can get v
86       vrange = dindgen(np)/(np-1)*v_temp[i]*3.
87       f_v = MaxwelllianDist(vrange, temperature[i], input.options.atom)
88       sumdist = f_v
89       for j=1,np-1 do sumdist[j] += sumdist[j-1]
90       sumdist /= max(sumdist)
91       vgrid[i,*] = interpol(vrange, sumdist, prob)
92     endfor
93     p = random_nr(seed=seed, npack)
94     *output.vx0 = interpolate_xy(vgrid, temperature, prob, surftemp, p)
95   endelse
96 end
97 'maxwellian2': begin
98   if (SpeedDist.temperature NE 0) then begin
99     ;; Use a constant surface temperature
100     v_th = sqrt(2*SpeedDist.temperature*!const.kb/atomicmass(input.options.atom)) /1e5
101     velocity = findgen(1001)/1000 * v_th*5 & velocity = velocity[1:*]
102     f_v = MaxwelllianDist2(velocity, SpeedDist.temperature, input.options.atom)

```

```

103 *output.vx0 = RandomDeviate_ld(velocity, f_v, npack) ;; km/s
104 endif else begin
105 ;; Use a surface temperature map
106 rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
107 SZA = acos(-*output.y0/rr) ;; cos(SZA) = [0,-1,0]·[x,y,z]/r = -y/r
108 q = where(finite(SZA) EQ 0, nq) & if (nq NE 0) then stio
109 surftemp = surface_temperature(input.geometry, SZA)
110
111 nt = 101 & np = 1001
112 temperature = dindgen(nt)/(nt-1)*(max(surftemp)-min(surftemp)) + min(surftemp)
113 v_temp = sqrt(2*temperature*!const.kb/atomicmass(input.options.atom)) /1e5
114 prob = dindgen(np)/(np-1)
115 vgrid = dblarr(nt,np)
116 for i=0,nt-1 do begin
117 ;; Produces the velocity as fn of T and cumulative value.
118 ;; Given T and random P, can get v
119 vrange = dindgen(np)/(np-1)*v_temp[i]*3.
120 f_v = MaxwellianDist2(vrange, temperature[i], input.options.atom)
121 sumdist = f_v
122 for j=1,np-1 do sumdist[j] += sumdist[j-1]
123 sumdist /= max(sumdist)
124 vgrid[i,*] = interpol(vrange, sumdist, prob)
125 endfor
126 p = random_nr(seed=seed, npack)
127 *output.vx0 = interpolate_xy(vgrid, temperature, prob, surftemp, p)
128 endelse
129 end
130 'flat': *output.vx0 = random_nr(npack)*(2*SpeedDist.delv) + SpeedDist.vprob - $
131 SpeedDist.delv
132 'circular orbits': begin
133 ;; Determine the Keplerian velocity
134 rr = sqrt(*output.x0^2 + *output.y0^2 + *output.z0^2)
135 velocity = sqrt(abs((SystemConsts.GM)[0]/rr))*SystemConsts.rPlan ;; Kepler vel.
136
137 ;; Determine the plane of the orbit
138 ;; All orbits are in the z x r direction
139 xhat = *output.x0/rr & yhat = *output.y0/rr & zhat = *output.z0/rr
140 zaxis = [0., 0., 1]
141 vhat = fltarr(npack, 3)
142 for i=0L,npack-1 do begin
143 vhat[i,*] = crossp(zaxis, [xhat[i], yhat[i], zhat[i]])
144 vhat[i,*] = vhat[i,*]/sqrt(total(vhat[i,*]*vhat[i,*]))
145 endfor
146
147 ;; Starting velocity
148 *output.vx0 = velocity * vhat[:,0]
149 *output.vy0 = velocity * vhat[:,1]
150 *output.vz0 = velocity * vhat[:,2]
151 end
152 'user defined': begin
153 restore, speeddist.distfile

```

```
154 *output.vx0 = RandomDeviates_ld(*speeddistribution.v, *speeddistribution.fv, npack)
155 destroy_structure, speeddistribution
156 end
157 else: stop
158 endcase
159
160 *output.vx0 /= SystemConsts.rplan
161 *output.vy0 /= SystemConsts.rplan
162 *output.vz0 /= SystemConsts.rplan
163
164 end
```