```
pro modelstreamlines, inputfiles, dt, npackets, seed

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Driver to determine particle streamlines.
;;
;; Method = 0 -> Satellite positions at end of model time are given
;; Method = 1 -> Satellite positions at begining of model time are given
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;Load in the common blocks
common constants
common ratecoefs
common plasma

tstart = systime(1)
tittot = 0.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Determine program version
readfmt, 'version.dat', /silent, 'A100', version
version = strtrim(version, 2)
if (n_elements(version) EQ 1) then stop
ntot = 0L

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Loop over each inputfile
ninputs = n_elements(inputfiles)
for iii=0,ninputs-1 do begin
  trun0 = systime(1)
  strstart = 'Inputfile #' + strint(iii) + ': '

  inputfile = inputfiles[iii]
  print, '**********************'
  print, strstart + 'Starting ' + inputfile
  print, strstart + systime(0)

  input = inputs_restore(inputfile)
  outputfile = output_filename(input) + '.streamline'

  ;; Set up the stuff structure
  stuff = {s:0, aplanet:0., vrplanet:0., radpres_v:ptr_new(0), $
    radpres_const:ptr_new(0), datapath:''}

  ;; Read in the constants
  SystemConstants, input.geometry.Planet, SystemConsts, DipoleConsts
  stuff.s = (where(strlowcase(*SystemConsts.Objects) EQ $
    strlowcase(input.geometry.StartPoint)))[0]

  ;; Determine distance and radial velocity of planet relative to the sun
```

```
52    planet_dist, input.geometry.taa, SystemConsts, distance=dd, velocity=vv
53    stuff.aplanet = dd
54    stuff.vrplanet = vv/SystemConsts.rplan
55
56    ;; Set up the paths to necessary data
57    testdir = ['/Users/mburger/Data/AtomicData/', $
58      '/Users/burger/Data/AtomicData/', $
59      '$HOME/NeutralModel/AtomicData/']
60    w = (where(file_test(testdir)))[0]
61    if (w EQ -1) $
62      then stop $
63      else stuff.datapath = testdir[w]
64
65    ;; find the default reactions and datasets
66    if (input.options.lifetime EQ 0) $
67      then loss_info = lifetime_setup(input) $
68      else loss_info = !null
69
70    ;; Set up the radiation pressure
71    if (input.forces.radpres) then begin
72      q = get_gvalue(stuff.aplanet, input.options.atom, path=stuff.datapath+'g-values/')
73      q /= SystemConsts.rplan ;; v in rplan/s, a in rplan/s^2
74      *stuff.radpres_v = q[*,0]
75      *stuff.radpres_const = q[*,1]
76    endif else begin
77      *stuff.radpres_v = 0.
78      *stuff.radpres_const = 0.
79    endelse
80
81    ;; For streamlines, set at_once = 1
82    input.options.at_once = 1
83
84    ;; there are two ways to do the streamlines:
85    ;;   a) Time given is the location at the end of the model period
86    ;;   b) Time given is the location at the begining of the model period
87
88    ;; Only doing method A
89    ;; Determine the initial source distirbution
90    input.options.trackloss = 0 ;; for now
91    source_distribution, input, npackets, seed, output=output
92    if (input.options.lifetime EQ 0) then output.loss_info = $
93      {reactions:ptr_new(loss_info.reaction), files:ptr_new(loss_info.file), $
94      type:ptr_new(loss_info.type)}
95
96    ;; Number of time steps
97    nt = fix(input.options.endtime/dt)+1
98
99    xx = dblarr(npackets,nt) & xx[*,0] = *output.x
100   yy = dblarr(npackets,nt) & yy[*,0] = *output.y
101   zz = dblarr(npackets,nt) & zz[*,0] = *output.z
102   ff = dblarr(npackets,nt) & ff[*,0] = *output.frac
```

```
103     vx = dblarr(npackets,nt) & vx[*,0] = *output.vx
104     vy = dblarr(npackets,nt) & vy[*,0] = *output.vy
105     vz = dblarr(npackets,nt) & vz[*,0] = *output.vz
106
107     loc = {t:ptr_new(0), x:ptr_new(0), v:ptr_new(0), frac:ptr_new(0)}
108     *loc.x = [[*output.x], [*output.y], [*output.z]]
109     *loc.v = [[*output.vx], [*output.vy], [*output.vz]]
110     *loc.frac = *output.frac
111     *loc.t = *output.time ;; This is how much time before present and works up to zero
112
113     h = replicate(dt, npackets)
114     which = where(*input.geometry.include, nw)
115     pp = replicate(1.,n_elements(*SystemConsts.objects))
116     for i=1,nt-1 do begin
117       todo = where(*loc.frac GT 0, ntodo)
118       if (ntodo GT 0) then begin
119         ;; extract necessary packets
120         loc1 = {t:ptr_new((*loc.t)[todo]), x:ptr_new((*loc.x)[todo,*]), $
121           v:ptr_new((*loc.v)[todo,*]), frac:ptr_new((*loc.frac)[todo])}
122
123         ;; advance the step
124         rk_5, loc1, h, input, which, delta
125
126         ;; Check to see if we hit anything
127         jj = replicate(1., ntodo)
128
129         ;; 1) Did the packets hit anything?
130         ;Get object positions
131         if (input.options.motion) $
132           then locmoon, *loc1.t, *input.geometry.phi, *SystemConsts.a, $
133             *SystemConsts.orbrate, x=xSat, y=ySat, z=zSat, ang=ang $
134           else locmoon, fltarr(ntodo), *input.geometry.phi, *SystemConsts.a, $
135             *SystemConsts.orbrate, x=xSat, y=ySat, z=zsat, ang=ang
136
137         ;; Distance of packets from each object
138         tempR = sqrt(((*loc1.x)[*,0]#pp - xSat)^2 + ((*loc1.x)[*,1]#pp - ySat)^2 + $
139           ((*loc1.x)[*,2]#pp - zSat)^2)
140
141         ;Is r < satellite radius?
142         eps = 0.
143         satrad = jj # (*SystemConsts.radius)[which]*(1-eps)
144         hhh = where((tempR-satrad) LT 0, nhits)
145         ;;print, nhits
146         if (nhits NE 0) then begin
147           ;;w = where((*loc1.t)[hhh mod nq] EQ 0, nw) & if (nw NE 0) then stop
148           hx = hhh mod ntodo & hy = hhh/ntodo
149
150           ;; adject the frac values
151           if (input.sticking_info.stickcoef EQ 1) $
152             then (*loc1.frac)[hx] = 0 $
153             else (*loc1.frac)[hx] = (*loc1.frac)[hx] * (1.-input.sticking_info.stickcoef)
```

```
154
155         ;; Figure out exactly where things hit the surface
156         srad = satrad[hhh]
157         r0 = tempR[hhh]  ;; R_plan
158         x0 = (*loc1.x)[hx,0]  ;; R_plan
159         y0 = (*loc1.x)[hx,1]  ;; R_plan
160         z0 = (*loc1.x)[hx,2]  ;; R_plan
161         r0 = sqrt(x0^2 + y0^2 + z0^2)
162
163         ;; Position of the satellites
164         xcent = xSat[hx,hy]
165         ycent = ySat[hx,hy]
166         zcent = zSat[hx,hy]
167
168         ;; Vector from center of satellite to packet
169         ;;   -- packet positions relative to satellite
170         x1 = (x0-xcent)/srad  ;; rsat
171         y1 = (y0-ycent)/srad  ;; rsat
172         z1 = (z0-zcent)/srad  ;; rsat
173
174         ;; Velocity - orbital vel = vel relative to satellite
175         vxsat = -(*SystemConsts.orbvel)[hy]*cos(ang[hx,hy])*input.options.motion/$
176           SystemConsts.rplan
177         vysat = -(*SystemConsts.orbvel)[hy]*sin(ang[hx,hy])*input.options.motion/$
178           SystemConsts.rplan
179
180         vx0 = (*loc1.v)[hx,0] - vxsat  ;; rplan/s
181         vy0 = (*loc1.v)[hx,1] - vysat  ;; rplan/s
182         vz0 = (*loc1.v)[hx,2]  ;; rplan/s
183
184         ;; Find where the packet hit the surface
185         ;;    |x + vt| = 1 -- see ResearchNotes from 4/28/08
186         a = vx0^2 + vy0^2 + vz0^2
187         b = 2*(x1*vx0 + y1*vy0 + z1*vz0)
188         c = x1^2 + y1^2 + z1^2 - 1
189
190         dd = b^2 - 4*a*c
191         q = where(dd LT 0, nq) & if (nq NE 0) then stop
192         t0 = (-b - sqrt(b^2-4*a*c))/(2*a)
193         t1 = (-b + sqrt(b^2-4*a*c))/(2*a)
194         t = (t0 LE 0)*t0 + (t1 LT 0)*t1
195
196         ;; Point where packet hit the surface
197         x2 = x1 + vx0*t
198         y2 = y1 + vy0*t
199         z2 = z1 + vz0*t
200         ;; r2 = sqrt(x2^2 + y2^2 + z2^2)  ;; -- this should be = 1.
201
202         lonhit = (atan(x2, -y2) + 2*!pi) mod (2*!pi)
203         lathit = asin(z2)
204
```

```
205           ;; Put new coordinates into the array
206           x_final = xcent + x2*srad
207           y_final = ycent + y2*srad
208           z_final = zcent + z2*srad
209
210           q = where(finite(x_final) EQ 0, nq) & if (nq NE 0) then stop
211           q = where(finite(y_final) EQ 0, nq) & if (nq NE 0) then stop
212           q = where(finite(z_final) EQ 0, nq) & if (nq NE 0) then stop
213           (*loc1.x)[hx,0] = x_final
214           (*loc1.x)[hx,1] = y_final
215           (*loc1.x)[hx,2] = z_final
216
217           if (input.sticking_info.stickcoef LT 1) then begin
218             ;; Figure out rebound velocity
219             vv02 = vx0^2 + vy0^2 + vz0^2  ;; rplan/s
220             PE = 2*(*SystemConsts.GM)[hy]*(1./r0-1./srad)
221             vv02 += PE
222             q = where(vv02 LT 0, nq) & if (nq NE 0) then vv02[q] = 0.
223             q = where(finite(vv02) EQ 0, nq) & if (nq NE 0) then stop
224
225             case strlowcase(input.sticking_info.emitfn) of
226               'maxwellian': begin ;; Re-emit the packets with a thermal distribution
227                 if (input.sticking_info.Tsurf EQ 0) then begin
228                   surftemp = temp0 + (temp1*(abs(cos(lonhit)*cos(lathit)))^nn)*$
229                     (abs(lonhit) LT !pi/2)
230                   vv_new = interpolate_xy(vgrid, temperature, prob, surftemp, $
231                     random_nr(seed=seed, nhits))/SystemConsts.rplan
232                 endif else vv_new = interpol(vrange, sumdist, $
233                   random_nr(seed=seed, nhits))/SystemConsts.rplan ;; rplan/s
234
235                 vv2 = sqrt(input.Sticking_info.accom_factor*vv_new^2 + $
236                   (1-input.Sticking_info.accom_factor)*vv02)
237                 end
238               'elastic scattering': vv2 = sqrt(vv02)
239             endcase
240
241             ;; Determine new direction with F(v) ~ cos(theta)
242             alt = acos(random_nr(seed=seed, nhits))
243             az = 2*!pi * random_nr(seed=seed, nhits)
244
245             v_rad = sin(alt)          ;; Radial component of velocity
246             v_east = -cos(alt) * sin(az) ;; Component along latitude line (points east)
247             v_north = cos(alt) * cos(az) ;; Component along longitude line (points to NP)
248
249             vx2 = v_rad*x2
250             vy2 = v_rad*y2
251             vz2 = v_rad*z2
252
253             lat = asin(z2)
254             lon = atan(x2, y2)
255
```

```
256             vx2 = dblarr(nhits) & vy2 = dblarr(nhits) & vz2 = dblarr(nhits)
257             for i=0L,nhits-1 do begin
258               M = transpose([ $
259                 [cos(lat[i])*sin(lon[i]), cos(lat[i])*cos(lon[i]), sin(lat[i])], $
260                 [-cos(lon[i]), sin(lon[i]), 0], $
261                 [-sin(lat[i])*sin(lon[i]), -sin(lat[i])*cos(lon[i]), cos(lat[i])] ])
262               v_ren = [v_rad[i], v_east[i], v_north[i]]
263               v_xyz = invert(M) # v_ren
264               vx2[i] = v_xyz[0] * vv2[i]
265               vy2[i] = v_xyz[1] * vv2[i]
266               vz2[i] = v_xyz[2] * vv2[i]
267             endfor
268
269             ;; The new position in planet-centered coords
270             vx_final = vx2 + vxsat
271             vy_final = vy2 + vysat
272             vz_final = vz2
273
274             q = where(finite(vx_final) EQ 0, nq) & if (nq NE 0) then stop
275             q = where(finite(vy_final) EQ 0, nq) & if (nq NE 0) then stop
276             q = where(finite(vz_final) EQ 0, nq) & if (nq NE 0) then stop
277
278             (*loc1.v)[hx,0] = vx_final
279             (*loc1.v)[hx,1] = vy_final
280             (*loc1.v)[hx,2] = vz_final
281           endif  ;; stickcoef LT 0
282         endif    ;; nhits GT 0
283
284         ;; 2) Have the packets left the corona?
285         if ~(input.options.fullSystem) then begin
286           leftCor = where(tempR[*,stuff.s] GT input.options.OuterEdge * $
287             (*SystemConsts.radius)[stuff.s], hh)
288           if (hh NE 0) then (*loc1.frac)[leftCor] = 0
289         endif
290
291         ;; 3) If Saturn, check to see if anything hit the rings
292         if (input.geometry.planet EQ 'Saturn') then begin
293           ;; Ring region within 2.3 Rs
294           ox = oldx[g,*]
295           cross = ox[*,2] * (*loc1.x)[*,2] ;;if cross is negative, then crossed eq. plane
296           MayHit = where(cross LE 0 , nmay)
297           if (nmay NE 0) then begin
298             orho = sqrt(total(ox[MayHit,0:1]^2, 2))
299             nrho = sqrt(total((*loc1.x)[MayHit,0:1]^2, 2))
300             w = where((orho LT 2.3) or (nrho LT 2.3), nw)
301             for j=0,nw-1 do begin $
302               crosspt = interpol([orho[w[j]],nrho[w[j]]], [ox[MayHit[w[j]],2], $
303                 (*loc1.x)[MayHit[w[j]],2]], 0.)
304               if (crosspt LT 2.3) then (*loc1.frac)[MayHit[w[j]]] = 0.
305             endfor
306           endif
```

```
307         endif
308
309         ;; Save these values and update the full loc structure
310         xx[todo,i] = (*loc1.x)[*,0]
311         yy[todo,i] = (*loc1.x)[*,1]
312         zz[todo,i] = (*loc1.x)[*,2]
313         ff[todo,i] = *loc1.frac
314         vx[todo,i] = (*loc1.v)[*,0]
315         vy[todo,i] = (*loc1.v)[*,1]
316         vz[todo,i] = (*loc1.v)[*,2]
317
318         (*loc.t)[todo] = *loc1.t
319         (*loc.x)[todo,*] = *loc1.x
320         (*loc.v)[todo,*] = *loc1.v
321         (*loc.frac)[todo,*] = *loc1.frac
322         loc1 = 0.
323       endif  ;; nq > 0
324     endfor   ;; loop over timesteps
325     loc = 0
326
327     ;; Save the output structure
328     *output.x = xx
329     *output.y = yy
330     *output.z = zz
331     *output.frac = ff
332     *output.vx = vx
333     *output.vy = vy
334     *output.vz = vz
335     save, output, input, version, file='temp.sav'
336     output = 0 & input = 0 & SystemConsts = 0
337   endfor  ;; inputfiles
338
339   end
340
```