```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Some functions to help out computing the results
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_loadfile, file, pts_sun, vels_sun, frac, keepall=keepall

;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Load results file and convert to proper reference frame
;;
;; Input:
;;   file = output file to restore
;;
;; Outputs:
;;   pts_sun = x,y,z in the solar frame with (0,0,0)=object center and units=R_obj
;;   vels_sun = vx,vy,vz in the solar frame, units=km/s
;;   frac = packet fraction remaining
;;
;;;;;;;;;;;;;;;;;;;;;;

common constants
common results

if (keepall EQ !null) then keepall=0

;; Determine the image origin
s = (where(strcmp(*SystemConsts.objects, format.geometry.origin, /fold), ns))[0]
if (ns NE 1) then stop

if (s NE 0) then begin
  ;; Will need to translate packets to satellite frame
  origin = (*SystemConsts.a)[s]*[-sin((*input.geometry.phi)[s]), $
    cos((*input.geometry.phi)[s]), 0.]   ;; location of satellite
  sc = 1./(*SystemConsts.radius)[s]      ;; scale factor
endif else begin
  origin = [0., 0., 0.]
  sc = 1.
endelse

;; Reuseable script to load the output file and get the packets to use
ofile = obj_new('IDL_savefile', file)
ofile.restore, 'output'
obj_destroy, ofile

;; Extract packets to use
touse = (keepall) ? lindgen(n_elements(*output.frac)) : where(*output.frac NE 0, npack)

;; Determine position relative to origin -- not rotated
pts_sun = [[(*output.x)[touse]-origin[0]], $
  [(*output.y)[touse]-origin[1]], $
  [(*output.z)[touse]-origin[2]]]
pts_sun *= sc  ;; Units = R_obj

;; Velocities not adjusted -- still includes orbital motion
vels_sun = [[(*output.vx)[touse]], [(*output.vy)[touse]], [(*output.vz)[touse]]]
vels_sun *= SystemConsts.rplan

frac = (*output.frac)[touse]
destroy_structure, output

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

pro results_intensity_setup

common constants
common results

if (max(strcmp(format.emission.mechanism, 'resscat', /fold))) then begin
  ;; get g-values
  gvalue = get_gvalue(input.options.atom, stuff.aplanet)
endif

if (max(strcmp(format.emission.mechanism, 'eimp', /fold))) then begin
  stop
  ;; load plasma info
endif

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function slit_solidangle, data

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;; Determine the solid angle subtended by the slit
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

temp = [[[*data.xcorner]], [[*data.ycorner]], [[*data.zcorner]]]
c0 = reform(temp[0,*,*]) & c1 = reform(temp[1,*,*])
c2 = reform(temp[2,*,*]) & c3 = reform(temp[3,*,*]) & temp = 0

xxx = c0[*,1]*c2[*,2] - c0[*,2]*c2[*,1]
yyy = -c0[*,0]*c2[*,2] + c0[*,2]*c2[*,0]
zzz = c0[*,0]*c2[*,1] - c0[*,1]*c2[*,0]
ccc = total(c0*c2,2)
```

```

q0 = abs(c1[*,0]*xxx + c1[*,1]*yyy + c1[*,2]*zzz)
q1 = 1 + ccc + total(c1*c0,2) + total(c1*c2,2)
omega0 = atan(q0,q1)
q = where(omega0 LT 0, nq) & if (nq NE 0) then omega0[q] += !pi

q0 = abs(c3[*,0]*xxx + c3[*,1]*yyy + c3[*,2]*zzz)
q1 = 1 + ccc + total(c3*c0,2) + total(c3*c2,2)
omega1 = atan(q0,q1)
q = where(omega1 LT 0, nq) & if (nq NE 0) then omega1[q] += !pi

omega = 2*(omega0+omega1) ;; slit solid angle for each spectrum

return, omega

end

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

function results_find_intersection_points, data, input

nn = n_elements(*data.x)
tt = dblarr(2,nn)

oedge = (input.options.outeredge*1.25)^2 ;; give 25% leeway
dist_from_plan = sqrt(*data.x^2 + *data.y^2 + *data.z^2)
for i=0,nn-1 do begin
  r0 = dist_from_plan[i]
  t = findgen(1001)/1000. * (dist_from_plan[i]+input.options.outeredge*1.5)

  p0x = (*data.x)[i] + t*(*data.xbore)[i]
  p0y = (*data.y)[i] + t*(*data.ybore)[i]
  p0z = (*data.z)[i] + t*(*data.zbore)[i]
  r2 = p0x^2 + p0y^2 + p0z^2
  if (dist_from_plan[i] LT input.options.outeredge) then begin
    tt[0,i] = 0.
    tt[1,i] = interpol(t, r2, oedge)
  endif else begin
    q = (where(r2 EQ min(r2)))[0]
    tt[0,i] = interpol(t[0:q], r2[0:q], oedge)
    tt[1,i] = interpol(t[q:*], r2[q:*], oedge)
  endelse
endfor

return, tt

end
```