Emission Calculation Documentation

Contents

1 Introduction **1**

2 Emission Line Calculations **1**

3 Non-Maxwellian Distributions **3**

4 Simulating a Spectrum **4**

5 Plotting our Simulated Spectrum **5**

6 Code **6**

1 Introduction

The focus of this write-up will be how we use the CHIANTI database version 8.0.7 to calculate the emissivities for a given emitting species. This document is based on commented documentation in emiss_calc.pro, pop_solver.pro, and my own experience with the code.

To create a synthetic spectrum we first calculate the emissivities for each species as a function of the electron density, and electron temperature using the function emiss_calc.pro.

Emiss_calc.pro was written by Peter Young originally on June 28th, 1997.

The code explicitly computes the emissivities of all lines of a specified ion over given ranges of temperature and density.

This routine calculates:

$$\frac{hc}{\lambda} N_j A_{ji} \tag{1}$$

Where $hc = 1.986 \times 10^{-8}$ ergs Å, λ is in angstroms, N_j is the fraction of ions in the upper emitting level j, and A_{ji} is the radiative decay rate for the transition otherwise known as the Einstein A coefficient for spontaneous emission.

Transitions where only theoretical energies are available for at least one of the two levels are assigned negative wavelengths in the .wgfa file. With emiss_calc, the wavelength is set to be positive, but emiss.flag is set to -1 for that transition, so that other routines can keep track of the theoretical wavelengths.

2 Emission Line Calculations

Below I have a sample call to an edited version of emiss_calc.pro I have renamed emiss_calc_EGN.pro.

```
conversion=11604.5221 ; conversion between eV and Kelvin from NIST
; emission line calculation: Sulfur ions
s2em = emiss_calc_egn(16, 2, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)
s3em = emiss_calc_egn(16, 3, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)
s4em = emiss_calc_egn(16, 4, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)
s5em = emiss_calc_egn(16, 5, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)


; emission line calculation: oxygen ions
o2em = emiss_calc_egn(8, 2, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)
o3em = emiss_calc_egn(8, 3, temp = alog10(Tel*conversion), dens = alog10(nel), /no_de, radt = 1., /quiet,proton_ratio=0.1)
```

emiss_calc_EGN.pro has the following inputs:

IZ The atomic number of the ion

ION The spectroscopic number of the ion (e.g., 12 = XII)

Temp The electron temperature assuming Maxwellian (log10 Kelvin)

Dens The electron density (Log10 cm^{-3})

Radt Specify background radiation temperature (I have set this background to 1 in all I do so there is no photo-excitation or stimulated emission and only spontaneous emission)

Proton_ratio the proton to electron ratio.

Relevant Keywords:

NO_DE Drops the hc/λ factor in the computation of the emissivities. Useful for emission measure analyses involving photon fluxes.

QUIET If set, don't list the temperatures and densities at which the emissivities are calculated.

NOPROT If set, then the default setting will be NOT to use proton rates. This can be changed within the routine.

As can be seen above the main inputs are the atomic number of the ion, the spectroscopic number of the ion, the electron temperature, and the electron density.

In this edited version of the code I have added an input to directly specify the proton to electron ratio. If we wish to not use proton rates then we should specify the /noprot keyword.

In the original version of emiss_calc.pro if you do not specify not to use proton emission rates then the proton to electron ratio would be calculated using the function proton_dens.pro based on ion equilibrium and abundance files.

This doesn't change the output considerably for us as we are dominated by electron excitation in the Io Plasma Torus environment.

In the original version of emiss_calc.pro the additional atomic processes of ionization and recombination can be included when calculating the level populations. This assumes ionization equilibrium. We do not include these in our version of the code where we have explicitly set status=-1 which in the population solver tells it no to include these corrections to the populations. We only use proton and electron rates for excitation.

Within emiss_calc.pro first the commonblocks are defined and a call to zion2filename to help locate the relevant CHIANTI database files. Then setup_ion is called on because several of the CHIANTI routines require the atomic data for ions to be read into common blocks, and so this routine performs this task for individual ions. Then the code calls on the routine zion2spectroscopic to provide identification strings. At this point the code defines the input structure for the population solver and calls on the population solver the first time to find the size of the population array which is then used to set the appropriate dimensions for the output emission structure. Then the population solver is called on again to fill the emiss structure

which is then returned as the output of emiss_calc.pro.

OUTPUT of emiss_calc.pro:

The structure that is output has the following tags:

.ion_name string; contains ion name, e.g., 'Fe XIII'

.lambda float; contains wavelength

.level1 integer; contains lower level of transition

.lvl1_desc string; gives config. and term of lower level

.level2 integer; contains upper level of transition

.lvl2_desc string; gives config. and term of upper level

.flag integer; a flag to mark particular transitions

.em fltarr(nt,nd); contains emissivities at particular temperatures and densities.

The population solver solves the level balance equations for the CHIANTI ions. Atomic data is pre-loaded into the COMMON blocks, and so POP_SOLVER can only be called indirectly through other routines.

The matrix equation Ax=b is solved where A contains all the atomic data (electron rate coefficients, radiative decay rates, proton rate coefficients, photoexcitation rates), x are the level populations, and b a vector set to zeros except for the first element which is 1.

To solve the matrix equation, pop_solver calls out to the CHIANTI routine matrix_solver.

The matrix A is populated with the atomic data in the COMMON blocks. Once this matrix is solved and the level populations are determined we return to emiss_calc.pro where these level populations are multiplied by the radiative decay rates to give the emissivities.

To simulate a spectrum we calculate these discrete emissivities using emiss_calc_EGN.pro for each emitting ion. Then we multiply by the ion density, 10^{-6}, and integrate this over the line of sight to get Rayleighs. Finally we model each discrete emission as a normalized Gaussian at the fwhm of the instrument we are trying to compare with for analysis.

3 Non-Maxwellian Distributions

To model the electron velocity distribution as a sum of Maxwellians instead of a single Maxwellian you can use the optional input:

SUM_MWL_COEFFS:

An array of coefficients of the same length as the array of temperatures. Electron and proton rate coefficients will be calculated at each temperature and then a weighted sum of the coefficients is performed using SUM_MWL_COEFFS. This allows non-Maxwellian energy distributions to be incorporated into the level balance equations.

An example of using the code in this way is shown below:

```
conversion=11604.5221 ; conversion between eV and Kelvin from NIST
; emission line calculation: sulfur ions
s2em = emiss_calc_egn(16, 2, temp = [alog10(Tel*conversion),alog10(Tehot*conversion)],sum_mwl_coeff=[fec,feh], dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
s3em = emiss_calc_egn(16, 3, temp = [alog10(Tel*conversion),alog10(Tehot*conversion)], sum_mwl_coeff=[fec,feh],dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
s4em = emiss_calc_egn(16, 4, temp = [alog10(Tel*conversion),alog10(Tehot*conversion)], sum_mwl_coeff=[fec,feh],dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
s5em = emiss_calc_egn(16,5,temp = [alog10(Tel*conversion),alog10(Tehot*conversion)], sum_mwl_coeff=[fec,feh],dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
; emission line calculation: oxygen ions
o2em = emiss_calc_egn(8, 2, temp = [alog10(Tel*conversion),alog10(Tehot*conversion)],sum_mwl_coeff=[fec,feh], dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
o3em = emiss_calc_egn(8, 3, temp = [alog10(Tel*conversion),alog10(Tehot*conversion)], sum_mwl_coeff=[fec,feh],dens = alog10(nel), /no_de, radt = 1., /quiet, proton_ratio=0.1)
```

Here fec is the fraction of core electrons, feh is the fraction of hot electrons, the core electrons have temperature Tel, and the hot electrons have temp Tehot. In this case nel is actually the total electron density counting both the core and hot fraction.

4 Simulating a Spectrum

Now I will show an example of how to use the code to calculate the emissivities assuming a Maxwellian distribution and plot the simulated spectrum.

There are several steps to follow. After you have done the emission line calculation for each ion as I described above you must integrate over the line of sight. For the warm Io Plasma torus we are biased towards the peak emission and because the density falls off so quickly we can assume the distribution is constant along the line of sight and rewrite the line integral and multiplying by the local ion density as just multiplying by the ion column density along the line of sight. This is done by the following code:

```
;puts in rayleighs, nsp etc. in column density units #/cm^2
s2emiss = reform(s2em.em)*Nsp*(10.^(-6.))
s3emiss = reform(s3em.em)*Ns2p*(10.^(-6.))
s4emiss = reform(s4em.em)*Ns3p*(10.^(-6.))
s5emiss=  reform(s5em.em)*Ns4p*(10.^(-6.))
o2emiss = reform(o2em.em)*Nop*(10.^(-6.))
o3emiss = reform(o3em.em)*No2p*(10.^(-6.))
```

This can be found in the attached IDL function simulate_IPT_spectrum_rayleighs.pro.

Now we must put all of the emission into one array and sort it by wavelength then model it as a Gaussian at the input fwhm. This is done by the following code in simulate_IPT_spectrum_rayleighs.pro:

```
  ; discrete
  xwavi = [s2em.lambda, s3em.lambda, s4em.lambda,s5em.lambda, o2em.lambda, o3em.lambda]
  wsort = sort(xwavi) ; sort by wavelength
  xwavi = xwavi[wsort]

  yptsi = [s2emiss, s3emiss, s4emiss, s5emiss, o2emiss, o3emiss]
  yptsi = yptsi[wsort]

  yname = [s2em.ion_name, s3em.ion_name, s4em.ion_name, s5em.ion_name, o2em.ion_name, o3em.ion_name]
  yname = yname[wsort]

  ; cut down arrays to necessary range
  avgwav = (minwav + maxwav)/2.
  wrange = where(abs(xwavi - avgwav) le avgwav - minwav)
  xwavi = xwavi[wrange]
  yptsi = yptsi[wrange]
  yname = yname[wrange]

  ; non discrete
  xwav=x
  numpoints=n_elements(xwav)
  ;xwav = (maxwav - minwav)*dindgen(numpoints)/(numpoints - 1.) + minwav
  ypts = dblarr(numpoints)

  ; relation between sigma and fwhm:
  ; => fwhm = 2*sigma*sqrt(2*ln(2))
  sigma = fwhm/(2d*sqrt(2d*alog(2d)))
  sigma2 = sigma*sigma
  a = 1d/(sigma*sqrt(2d*!dpi))
  for ii = 0, numpoints - 1 do begin
    ; treat every emission as a normalized gaussian, add up all gaussians to create simulated spectra
    ; f(x) = (1/sigma*sqrt(2pi)) * exp( -.5(x - x0)^2 / sigma^2 )
    y = a*yptsi[0]*exp( (-.5*(xwav[ii]-xwavi[0])*(xwav[ii]-xwavi[0]))/sigma2 )
    for jj = 1, n_elements(xwavi)-1 do y += a*yptsi[jj]*exp( (-.5*(xwav[ii]-xwavi[jj])*(xwav[ii]-xwavi[jj]))/sigma2 )

    ypts[ii] = y
  endfor

  return, ypts

  ; note: xwav is also an output, placed as a keyword

end
```

To call on this function and plot the simulated spectrum I use the procedure simulate_and_plot_ipt_spectrum.pro

like so:

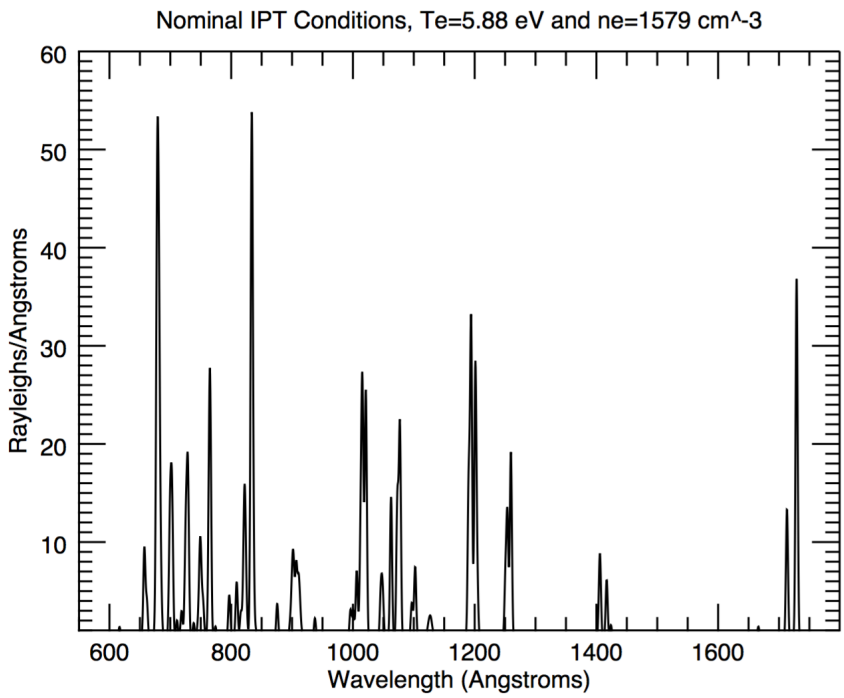
```
Pro simulate_and_plot_ipt_spectrum2

x=findgen(3000)*(1./2.4)+550.
;print,min(x),max(x),x
  tel=5.88013d
  nel=1578.87d
  euv_bin_size=0.6049d ; for euv Cassini UVIS (550-1150A)
  fuv_bin_size=0.7794d; for fuv Cassini UVIS  (1150-1800A)



  ;stop

mixingNSp=0.0639787 ;
  mixingNS2p=0.210333 ;
  mixingNS3p=0.0371900 ;
  mixingNS4p=0.0
  mixingNOp=0.257333 ;
  mixingNO2p=0.0232268 ;
  Ne_column=1.42376e+14 ;
  Ns4p=0.


  nsp=mixingNSp* Ne_column
  ns2p=mixingNS2p* Ne_column
  ns3p=mixingNS3p* Ne_column
  nop=mixingNop* Ne_column
  no2p=mixingNo2p* Ne_column
  min=min(x)
  max=max(x)
  fwhm=4.

;stop
  spec1=simulate_IPT_spectrum_rayleighs( x, Tel, nel, Nsp, Ns2p, Ns3p,Ns4p, Nop, No2p, $
    min = min, max = max, fwhm = fwhm)
    ylog=0

    spec1[0:1440]=spec1[0:1440]/euv_bin_size
    spec1[1441:2999]=spec1[1441:2999]/fuv_bin_size

   ; stop

    p1=plot(x,spec1,title='Nominal IPT Conditions, Te=5.88 eV and ne=1579 cm^-3',xtitle='Wavelength (Angstroms)',ytitle='Rayleighs/Angstroms',xrange=[min(x),max(x)],ylog=ylog,yrange=[1.,60.])
    stop

end
```

To set the path of the main CHIANTI directory, create a string of all directories under CHIANTI folder, compile the code, and run simulate_and_plot_ipt_spectrum.pro I use the procedure runner_simulate_and_plot_ipt_spectrum.pro like so:

```
pro runner_simulate_and_plot_ipt_spectrum2

  ; set variable to path of main chianti directory, assuming chianti is in the current working directory
  cd, current = current
  chiantipath = current + '/chianti/'

  ; or, if chianti is located elsewhere, edit the following variable
  ;chiantipath = '/Users/shinn/pro/chianti/' ; <---- user specific path

  ; use 'exists' keyword to determine whether to execute the following
  defsysv, '!chianti', exists = compile
  if ~compile then defsysv, '!chianti', 1

  if ~compile then defsysv, '!rjkm', 71492. ; RJ in km
  if ~compile then defsysv, '!kev', 8.617385d-5 ; Boltzmann's constant in eV
  if ~compile then defsysv, '!rootpi', sqrt(!dpi)

  ; create a string of all directories under chianti folder separated by ':'s
  if ~compile then chiantipros = ':' + expand_path('+' + chiantipath + 'idl/')

  ; if compile script has not been run before, add chianti path
  if ~compile then !path = !path + chiantipros
  if ~compile then use_chianti, chiantipath + 'dbase'

  ; compile the code
  resolve_routine, 'simulate_and_plot_ipt_spectrum2'

  ; run the code
simulate_and_plot_ipt_spectrum2

end
```

5 Plotting our Simulated Spectrum

Running this code by typing runner_simulate_and_plot_ipt_spectrum in the command line with runner_simulate_and_plot_ipt and other files in your current working directory produces the following plot:



6 Code

All the necessary IDL code to produce this plot is attached to this document on the following pages. With the exception of the CHIANTI database version 8.0.7 and accompanying routines which can be found at: http://www.chiantidatabase.org/chianti_download.html

Emiss_Calc_EGN.pro

```
;
; PROJECT:  CHIANTI
;
;       CHIANTI is an Atomic Database Package for Spectroscopic Diagnostics of
;       Astrophysical Plasmas. It is a collaborative project involving the Naval
;       Research Laboratory (USA), the University of Florence (Italy), the
;       University of Cambridge and the Rutherford Appleton Laboratory (UK).
;
; NAME: EMISS_CALC()
;
; PURPOSE:
;
;       To compute the emissivities of all lines of a specified ion over
;      given ranges of temperature and density.
;
; CATEGORY:
;
;       Scientific analysis
;
; EXPLANATION:
;
;      This routine calculates:
;
;             hc
;             --  * N_j  * A_ji
;            lamb
;
;      where hc = 1.986 * 10^-8 erg AA, lamb is in angstroms, N_j is the
;      fraction of ions in the upper emitting level j, and A_ji is the
;      radiative decay rate for the transition.
;
;      The emissivities are stored in a structure called EMISS that also
;      holds the wavelength of the transition, the level numbers i and j
;      and also a 'flag', which is set to -1 if the wavelength is negative.
;
;      The temperature and density ranges can be specified directly using
;      the TEMP and DENS keywords. Setting TMAX to the log T_max of the
;      ion, gives emissivities for 3 temperatures: log T_max +- 0.15.
;      If DENS is not set, then it is set to 8 to 12 in 0.5 dex intervals.
;      STDENS allows the start density (of 8) to be changed to some other
;      value; ND allows the number of densities to be varied (default 9);
;      DINT allows the density interval to be varied (default 0.5).
;
; CALLING SEQUENCE:
;
;       EMISS=EMISS_CALC (IZ, ION, [ TEMP=TEMP, DENS=DENS, RADT=RADT, $
;                            DIL=DIL, PATH=PATH, /NO_DE, /PROTON, $
;                            QUIET, PRESSURE=PRESSURE)
;
; EXAMPLES:
;
;      EMISS=EMISS_CALC(26,13)
;      EMISS=EMISS_CALC(26,13,temp=[6.2],dens=findgen(5)+8)
;      EMISS=EMISS_CALC(26,13,temp=findgen(11)/100.+5.5,press=10.^15)
;
; INPUTS:
;
;      IZ      The atomic number of the ion
```

```
;
;       ION     The spectroscopic number of the ion (e.g., 12 = XII)
;
; OPTIONAL INPUTS:
;
;       TEMP    Direct specification of the temperature range (log T)
;
;       DENS    Direct specification of the density range (log Ne)
;
;       RADTEMP Specify background radiation temperature (default: 6000 K)
;
;       RPHOT   Distance from the centre of the star in stellar radius units.
;               I.e., RPHOT=1 corresponds to the star's surface. (Default is
;               infinity, i.e., no photoexcitation.)
;
;       PATH    If specified, the routine will look for the atomic data in
;               the PATH directory, rather than in the CHIANTI database
;
;       PRESSURE        If a temperature array is given, and PRESSURE set,
;                       then the emissivities will be evaluated at the
;                       specified temperatures, but for densities =
;                       pressure/temperature. If DENS is set, then it will
;                       be ignored. The pressure is assumed to be in units
;                       K * cm^-3.
;
;       ABUND_FILE  The name of a CHIANTI abundance file. This is used for
;               calculating the proton to electron ratio. Default is
;               !abund_file.
;
;       IONEQ_FILE  The name of a CHIANTI ion balance file. This is used for
;               calculating the proton to electron ratio and evaluating
;               the T_max of the ion. Default is !ioneq_file.
;
;       SUM_MWL_COEFFS  An array of coefficients of the same length as
;                       the array of temperatures. Electron and proton rate
;                       coefficients will be calculated at each temperature
;                       and then a weighted sum of the coefficients is
;                       performed using SUM_MWL_COEFFS. This allows
;                       non-Maxwellian energy distributions to be
;                       incorporated into the level balance equations.
;                       This keyword is not compatible with the PRESSURE
;                       keyword.
;
;       RADFUNC         The name of a user-defined function that will generate
;                       a radiation spectrum as a function of temperature.
;                       This radiation field will replace the black-body that
;                       is assumed when using the RADTEMP keyword in the call
;                       to pop_solver.
;
; KEYWORDS:
;
;       NO_DE   Drops the hc/lambda factor in the computation of the
;               emissivities. Useful for emission measure analyses involving
;               photon fluxes
;
;       NOPROT  If set, then the default setting will be NOT to use
;               proton rates. This can be changed within the routine.
;
;       QUIET   If set, don't list the temperatures and densities at which
;               the emissivities are caculated.
```

```
;
;       DIEL   If the dielectronic recombination files exist for the ion,
;               then these are used to derive the emissivities.
;
;       NO_SETUP  If emiss_calc is called from a routine where the ELEMENTS
;                 common block has already been set up, then this keyword
;                 stops emiss_calc loading up the common block.
;
;       NO_CALC  If set, then the emissivities are not calculated but
;                everything else in the output structure is populated.
;
; OUTPUT:
;
;      The structure that is output has the following tags:
;
;      .ion_name      string; contains ion name, e.g., 'Fe XIII'
;      .lambda        float; contains wavelength
;      .level1        integer; contains lower level of transition
;       .lvl1_desc      string; gives config. and term of lower level
;      .level2        integer; contains upper level of transition
;       .lvl2_desc      string; gives config. and term of upper level
;      .flag          integer; a flag to mark particular transitions
;      .em            fltarr(nt,nd); contains emissivities at particular
;                       temperatures and densities.
;
; PROGRAMMING NOTES:
;
;      Transitions where only theoretical energies are available for at
;      least one of the two levels are assigned negative wavelengths in
;      the .wgfa file. With emiss_calc, the wavelength is set to be
;      positive, but emiss.flag is set to -1 for that transition, so
;      that other routines can keep track of the theoretical wavelengths.
;
; COMMON BLOCKS:
;
;      ELVLC, WGFA, UPSILON, RADIATIVE, PROTON, ELEMENTS, IONREC
;
; CALLS:
;
;      SETUP_ION, POP_SOLVER
;       ZION2FILENAME, ZION2SPECTROSCOPIC
;      READ_PROT, PROTON_DENS
;
; HISTORY:
;
;      Ver 1, PRY 28-Jun-97
;      Ver 2, PRY 26-Jul-97  - corrected problem with size of emiss
;      Ver 3, PRY 22-Sep-97  - allowed photo-excitation to be included
;      Ver 4, PRY 6-Jul-98   - added PATH
;      Ver 5, PRY 5-Sep-98   - added call to choose_ioneq
;       Ver 6, PRY 3-Dec-98   - dosen't crash if no params given
;      Ver 7, PRY 9-Jan-99   - allowed proton rates to be added through
;                              /PROTON keyword.
;      Ver 8, PRY 10-Feb-99  - added /QUIET keyword
;      Ver 9, PRY 8-Oct-99   - for H-like ions, there's a 2-photon
;                              transition with a non-zero A-value that is
;                              assigned a zero wavelength (as it does not
;                              produce an emission line). This caused
;                              problems for dens_plotter, so emiss_calc
;                              now removes this transition from the emiss
```

```
;                           structure.
;       Ver 10, PRY 15-Dec-99 -       added deu to the upsilon common block in
;                           order to be consistent with the main Chianti
;                           routines.
;       Ver 11, PRY 8-May-00  - added PRESSURE
;       Ver 12, PRY 17-Aug-00 - changed elvlc common block to match new
;                               version of pop_solver
;       Ver 13, PRY 10-Oct-00 - now calls setup_ion to read ion data
;       Ver 14, PRY 1-Jun-00  - removed call to choose_ioneq; now makes use
;                               of the !ioneq_file system variable.
;       Ver 15, PRY 25-Sep-01 - modified for 9-point splines and proton rates
;       Ver 16, PRY 9-Dec-01  - completed changes for v.4
;
;       V. 17, 29-May-2002, Giulio Del Zanna (GDZ)
;                   generalized directory concatenation to work for
;                   Unix, Windows  and VMS.
;                   Now we only call zion2filename, corrected the call to
;                   zion2spectroscopic (dielectronic cases were not handled
;                   correctly).
;
;       V.18, 6-Aug-2002, Peter Young
;                   Theoretical wavelengths weren't being flagged so this
;                   has now been corrected.
;
;       V.19, 7-Aug-2002, Peter Young
;                   Stopped "lines" with zero wavelength being included in
;                   the structure.
;                   Changed the use of CHIANTI system variables.
;
;       V.20 14-Aug-2002, GDZ
;                   Modified the use of Version, to make it compatible with the
;                   other CHIANTI v.4 routines.
;
;       V.21 10-Sep-2002, Peter Young
;                   Allowed a density of 1 cm-3 to be input.
;
;       V.22 7-Aug-2003, Peter Young
;                   Added keyword /NO_SETUP
;
;       V.23  4-Oct-2003, GDZ
;                  modified the input to POP_SOLVER, now it is dealt with an
;                  input structure.
;
;       V.24  5-Mar-2004, Enrico Landi (EL)
;                  included ionization and recombination as level population
;                  processes
;
;       V.25  6-May-2005, EL
;                  corrected a minor incompatibility with IDL 5.6
;
;       V.26  5-Jul-2005, Peter Young
;                  added RADFUNC= and SUM_MWL_COEFFS= keywords
;
;       V.27  1-Aug-2005, Peter Young
;                  re-ordered code for setup to prevent crash
;
;       V.28  3-Aug-2005,GDZ
;                  fixed bug, only define ioneq_ionrec when files are
;                  there (it was failing with neutrals)
;
```

```
;       V.29 , 7-Dec-07, GDZ changed a for loop to long to allow long
;              lists of lines.
;
;       V.30 , 12-Jun-2009, Enrico Landi
;               Changed the definition of the temperature array for ion fractions
;               in the IONREC variable, now taken directly from the output of
;               READ_IONEQ.PRO
;
;       V.31, 28-Apr-2013, Peter Young
;               The first call to pop_solver (which is only used to
;               get the size of the population array) is now made with
;               the /no_calc keyword in order to speed it up.
;
;       V.32, 25-Feb-2014, Peter Young
;               Added /no_calc to emiss_calc.
;
;        V.33, 13-Feb-2018, Edward G. Nerney
;               Added proton_ratio input to emiss_calc to explicitly
;               define the proton to electron ratio. Also explicitly set Status=-1
;               so no ionization or recombination corrections to the populations are
used.
;
;
;
;
; VERSION     :   32, 25-Feb-2014
;
;-

FUNCTION  EMISS_CALC_EGN, IZ, ION, TEMP=TEMP, DENS=DENS, RADTEMP=RADTEMP, $
                     RPHOT=RPHOT, $
                     PATH=PATH, NO_DE=NO_DE, NOPROT=NOPROT, $
                     QUIET=QUIET, PRESSURE=PRESSURE, DIEL=DIEL, $
                     IONEQ_FILE=IONEQ_FILE, ABUND_FILE=ABUND_FILE, $
                     NO_SETUP=NO_SETUP, SUM_MWL_COEFFS=SUM_MWL_COEFFS, $
                     RADFUNC=RADFUNC, NO_CALC=NO_CALC, proton_ratio=proton_ratio



COMMON elvlc,l1,term,conf,ss,ll,jj,ecm,eryd,ecmth,erydth,eref
COMMON wgfa, wvl,gf,a_value
COMMON upsilon,splstr
COMMON radiative, radt,dilute
COMMON proton, pstr, pe_ratio
COMMON elements,abund,abund_ref,ioneq,temp_all,ioneq_ref
COMMON ionrec,rec_rate,ci_rate,temp_ionrec,luprec,lupci,status


;explicitly set Status=-1
;so no ionization or recombination corrections to the populations were used.

Status=-1 ;

; Added proton_ratio input to emiss_calc to
; explicitly define the proton to electron ratio.
pe_ratio=proton_ratio

IF N_PARAMS() LT 2 THEN BEGIN
  PRINT,'Use:  IDL> emiss=emiss_calc(iz,ion [, temp=temp, dens=dens, '
  PRINT,'                              radtemp=radtemp, rphot=rphot, /no_de,'
```

```
  PRINT,'                                path=path, /noprot, /quiet, $ '
  PRINT,'                                pressure=pressure, ioneq_file=ioneq_file, $'
  print,'                                abund_file=abund_file ] )'
  RETURN,0
ENDIF

IF n_elements(pressure) NE 0 AND n_elements(sum_mwl_coeffs) NE 0 THEN BEGIN
  print,'% EMISS_CALC: the PRESSURE and SUM_MWL_COEFFS keywords are incompatible with'
  print,'              each other. Please use one or the other.'
  return,0
ENDIF

;
; The following extracts the names of the files to be read. I want to allow
; a different path to be chosen, and I extract only the information I need
; from filename.
;
; zion2filename is called on to help locate the relevant CHIANTI database files.
;
; zion2spectroscopic is called on to provide identification strings
;
IF keyword_set(diel) THEN diel = 1 ELSE diel = 0

zion2filename,iz,ion,filename, name=name, diel=diel
gname = name

setup_ion,name,-1,-1,wvltst,lvl1,lvl2,wvl1,gf1,a_value1,path=path, $
     noprot=noprot

IF NOT keyword_set(no_setup) OR n_elements(abund) EQ 0 OR $
     n_elements(ioneq) EQ 0 THEN BEGIN
  IF n_elements(ioneq_file) EQ 0 THEN ioneq_file=!ioneq_file
  read_ioneq,ioneq_file,temp_all,ioneq,ioneq_ref
  IF n_elements(abund_file) EQ 0 THEN abund_file=!abund_file
  read_abund,abund_file,abund,abund_ref
ENDIF

t_ioneq=temp_all             ; Defines the temperature array of ionization equilibrium
for the ionrec structure


;-----------------------------------+
; If TEMP not specified, then set TEMP=[log(T_max),log(T_max)+-0.15]
; using the ion balance data in CHIANTI.
;
IF N_ELEMENTS(temp) EQ 0 THEN BEGIN
  f_all=ioneq(*,iz-1,ion-1)
  ind=where(f_all EQ max(f_all))
  tmax=temp_all(ind) & tmax=tmax(0)
  temp=[tmax-0.15,tmax,tmax+0.15]
ENDIF
nt=N_ELEMENTS(temp)
;-----------------------------------+


;---------------X
sumtst=0
IF n_elements(sum_mwl_coeffs) NE 0 THEN BEGIN
  IF n_elements(temp) NE n_elements(sum_mwl_coeffs) THEN BEGIN
    print,'% EMISS_CALC: SUM_MWL_COEFFS must have the same size as TEMP. Returning...'
```

```
    return,0
  ENDIF
 ;
 ; normalize sum_mwl_coeffs
 ;
  smc=double(sum_mwl_coeffs)
  IF total(smc) NE 1. THEN BEGIN
    print,'% EMISS_CALC: normalizing SUM_MWL_COEFFS'
    smc=smc/total(smc)
  ENDIF
  sumtst=1
ENDIF
;---------------X


;---------------O
zion2spectroscopic,iz,ion,spectname
IF status GT 0 AND n_elements(sum_mwl_coeffs) NE 0 THEN BEGIN
  print,'% EMISS_CALC: this ion ('+spectname+') has contributions from ionization'
  print,'              and recombination that can not be included when the
SUM_MWL_COEFFS'
  print,'              keyword is set. The ionization and recombination processes'
  print,'              will be switched off.'
  status=-1   ; this switches off ioniz/rec
ENDIF
;---------------O


IF n_elements(rphot) EQ 0 THEN dilute=0d0 ELSE dilute=r2w(rphot)
IF n_elements(radtemp) EQ 0 THEN radt=6d3 ELSE radt=double(radtemp)



;;-----------------<>
; Default density range is 8 to 12 in .5 intervals:
;
IF n_elements(dens) EQ 0 THEN dens=findgen(9)/2. + 8.
nd=n_elements(dens)
;;-----------------<>


; If you use pressure and temp as inputs instead of temp and density then
N_ELEMENTS(pressure) NE 0
; In this case the code will find the density using the ideal gas law:



IF N_ELEMENTS(pressure) NE 0 THEN BEGIN
  nd=1
  dens=ALOG10(pressure/10.^temp)
ENDIF

;
; Create the structure with the ion/rec data, including ion fractions for
; the ion and the two adjacent ones
;

IF status lt 0 THEN ionrec = {rec:0., ci:0., temp:0., lev_up_rec:0., $
                              lev_up_ci:0., status:-1, ioneq:0., temp_ioneq:0.}
IF status gt 0 THEN begin
```

```
  IF ion gt 1 THEN ioneq_ionrec=reform(ioneq(*,iz-1,ion-2:ion))
  IF ion eq 1 THEN ioneq_ionrec=reform(ioneq(*,iz-1,ion-1:ion)) ; No coll.ionization to
neutral ions!


ionrec = {rec:rec_rate, ci:ci_rate, temp:temp_ionrec, $
                              lev_up_rec:luprec, lev_up_ci:lupci, $
                              status:status, ioneq:ioneq_ionrec, $
                              temp_ioneq:t_ioneq}
endif

;----------X
; Find the size of the population array
; This first call is only used to find the dimensions of the output structure.
;pe_ratio=proton_dens(temp[0])

input = {gname:gname, jj:jj, ecm:ecm,ecmth:ecmth, $
 wvl:wvl, a_value:a_value, splstr:splstr, $
 pe_ratio:pe_ratio,prot_struc:pstr, dilute:dilute, radtemp:radt, ionrec:ionrec}

pop_solver,input, 10.^temp[0],10.^9,pop, /no_calc, n_lev=n_lev

nlvls2=n_lev
;----------X

;
; remove lines with zero wavelengths
;
ind=where(wvl1 NE 0.)
IF ind[0] NE -1 THEN BEGIN
  lvl1=lvl1[ind]
  lvl2=lvl2[ind]
  wvl1=wvl1[ind]
  gf1=gf1[ind]
  a_value1=a_value1[ind]
ENDIF

;
; remove lines involving levels not in pop_solver model
;
ind=where((lvl1 LE nlvls2) AND (lvl2 LE nlvls2))
IF ind[0] NE -1 THEN BEGIN
  lvl1=lvl1[ind]
  lvl2=lvl2[ind]
  wvl1=wvl1[ind]
  gf1=gf1[ind]
  a_value1=a_value1[ind]
ENDIF

ntrans2=n_elements(wvl1)

;----------------------------------o
; Create the EMISS structure, using ntrans2 to set size
;
fname=concat_dir(!xuvtop,'VERSION')
str1=' '
IF file_exist(fname) THEN BEGIN
  openr,lun,fname,/get_lun
  readf,lun,str1
```

```
  free_lun,lun
;  str1='CHIANTI '+str1
ENDIF ELSE BEGIN
  str1=' '
print, 'Please update your version of CHIANTI to v.4 or later'
ENDELSE
;
zion2spectroscopic,iz,ion,species, diel=diel
;
IF keyword_set(sumtst) THEN em=dblarr(1,nd) ELSE em=dblarr(nt,nd)
;
str={   ion_name: species, $
        lambda: 0., $
        level1: 0, $
        lvl1_desc: '', $
        level2: 0, $
        lvl2_desc: '', $
        flag:   0, $
        em: em, $
        version: str1 $
        }
;
emiss=replicate(str,ntrans2)
;---------------------------------o

emiss.lambda=abs(wvl1)
emiss.level1=lvl1
emiss.level2=lvl2
emiss.lvl1_desc=term[lvl1-1]
emiss.lvl2_desc=term[lvl2-1]
ind=where(wvl1 LT 0.)
IF ind[0] NE -1 THEN emiss[ind].flag=-1


IF NOT KEYWORD_SET(quiet) THEN BEGIN
  PRINT,''
  PRINT,'Log_10 temperatures...'
  PRINT,FORMAT='("   ",9f6.2)',temp
  PRINT,''
  PRINT,'Log_10 densities...'
  PRINT,FORMAT='("   ",9f6.2)',dens
ENDIF

;The following sets the multiplicative factor to multiply N_j by. Either hc/lamb A_ji or
just A_ji

IF NOT keyword_set(no_de) THEN mult_factor=1.986d-8/emiss.lambda*a_value1 $
ELSE mult_factor=a_value1
;
;pe_ratio=proton_dens(temp)

; The code below calls on pop_solver to solve the level balance equations and return N_j

; N_ELEMENTS(pressure) NE 0 then you are using the pressure instead of the density as an
input


; N_ELEMENTS(pressure) = 0 then you are using the density instead of the pressure as an
input
```

```
IF NOT keyword_set(no_calc) THEN BEGIN
IF N_ELEMENTS(pressure) NE 0 THEN BEGIN

  input = {gname:gname, jj:jj, ecm:ecm,ecmth:ecmth, $
           wvl:wvl, a_value:a_value, splstr:splstr, $
           pe_ratio:pe_ratio,prot_struc:pstr, dilute:dilute, $
           radtemp:radt, ionrec:ionrec}
 ;
  pop_solver, input, 10.^temp,10.^dens,pop, radfunc=radfunc, /pressure
 ;
  emiss.em=pop[*,emiss.level2-1]
  FOR i=0,nt-1 DO emiss.em[i]=emiss.em[i]*mult_factor
ENDIF ELSE BEGIN
 ;
  input = {gname:gname, jj:jj, ecm:ecm,ecmth:ecmth, $
           wvl:wvl, a_value:a_value, splstr:splstr, $
           pe_ratio:pe_ratio,prot_struc:pstr, dilute:dilute, $
           radtemp:radt, ionrec:ionrec}
 ;
;

  pop_solver,input, 10.^temp,10.^dens,pop, sum_mwl_coeffs=smc, $
       radfunc=radfunc
 ;
  IF sumtst EQ 1 THEN BEGIN
   ;
    IF nd NE 1 THEN emiss.em[0,*]=pop[*,*,emiss.level2-1]
    IF nd EQ 1 THEN emiss.em[0,*]=reform(pop[*,*,emiss.level2-1],1, $
                                         n_elements(emiss.level2))
   ;
  ENDIF ELSE BEGIN
   ;
    IF nt EQ 1 THEN BEGIN
      IF nd NE 1 THEN emiss.em=pop[*,*,emiss.level2-1] ELSE $
           emiss.em=reform(pop[*,*,emiss.level2-1],1,n_elements(emiss.level2))
    ENDIF ELSE emiss.em=reform(pop[*,*,emiss.level2-1])
   ;
  ENDELSE
 ;
  FOR i=0L,ntrans2-1 DO emiss[i].em=emiss[i].em*mult_factor[i]
ENDELSE
ENDIF

;pop_solver above finds N_j and then these are multiplied by hc/lamb A_ji
; or just A_ji depending on no_de keyword
;These emissivities are stored in emiss.em

ind=sort(emiss.lambda)
emiss=emiss[ind]

RETURN,emiss

END
```

Simulate_IPT_spectrum_Rayleighs.pro

```
function simulate_IPT_spectrum_Rayleighs, x, Tel, nel, Nsp, Ns2p, Ns3p,Ns4p, Nop,
No2p, $
  min = min, max = max, fwhm = fwhm, $
  xwav = xwav
  ;Tel in eV, nel in #/cm^3, Nsp etc in #/cm^2, max/min/fwhm in angstroms
  ; sample spectra
  if keyword_set(max) then maxwav = max else maxwav = 1150d
  if keyword_set(min) then minwav = min else minwav = 550d
  if keyword_set(fwhm) then fwhm = fwhm else fwhm = 3.


  conversion=11604.5221 ; conversion between eV and Kelvin from NIST
  ; emission line calculation: Sulfur ions
  s2em = emiss_calc_egn(16, 2, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)
  s3em = emiss_calc_egn(16, 3, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)
  s4em = emiss_calc_egn(16, 4, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)
  s5em = emiss_calc_egn(16, 5, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)


  ; emission line calculation: oxygen ions
  o2em = emiss_calc_egn(8, 2, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)
  o3em = emiss_calc_egn(8, 3, temp = alog10(Tel*conversion), dens = alog10(nel),
/no_de, radt = 1., /quiet,proton_ratio=0.1)




  ;puts in rayleighs, nsp etc. in column density units #/cm^2 ; Assuming distribution is
uniform over line of sight which works because we are biased by peak emission bc density
falls off sharply with increasing RJ
  s2emiss = reform(s2em.em)*Nsp*(10.^(-6.))
  s3emiss = reform(s3em.em)*Ns2p*(10.^(-6.))
  s4emiss = reform(s4em.em)*Ns3p*(10.^(-6.))
  s5emiss=  reform(s5em.em)*Ns4p*(10.^(-6.))
  o2emiss = reform(o2em.em)*Nop*(10.^(-6.))
  o3emiss = reform(o3em.em)*No2p*(10.^(-6.))

  ; discrete
  xwavi = [s2em.lambda, s3em.lambda, s4em.lambda,s5em.lambda, o2em.lambda, o3em.lambda]
  wsort = sort(xwavi) ; sort by wavelength
  xwavi = xwavi[wsort]

  yptsi = [s2emiss, s3emiss, s4emiss, s5emiss, o2emiss, o3emiss]
  yptsi = yptsi[wsort]

  yname = [s2em.ion_name, s3em.ion_name, s4em.ion_name, s5em.ion_name, o2em.ion_name,
o3em.ion_name]
  yname = yname[wsort]

  ; cut down arrays to necessary range
  avgwav = (minwav + maxwav)/2.
  wrange = where(abs(xwavi - avgwav) le avgwav - minwav)
```

```
  xwavi = xwavi[wrange]
  yptsi = yptsi[wrange]
  yname = yname[wrange]

  ; non discrete
  xwav=x
  numpoints=n_elements(xwav)
  ;xwav = (maxwav - minwav)*dindgen(numpoints)/(numpoints - 1.) + minwav
  ypts = dblarr(numpoints)

  ; relation between sigma and fwhm:
  ; => fwhm = 2*sigma*sqrt(2*ln(2))
  sigma = fwhm/(2d*sqrt(2d*alog(2d)))
  sigma2 = sigma*sigma
  a = 1d/(sigma*sqrt(2d*!dpi))
  for ii = 0, numpoints - 1 do begin
    ; treat every emission as a normalized gaussian, add up all gaussians to create
simulated spectra
    ; f(x) = (1/sigma*sqrt(2pi)) * exp( -.5(x - x0)^2 / sigma^2 )
    y = a*yptsi[0]*exp( (-.5*(xwav[ii]-xwavi[0])*(xwav[ii]-xwavi[0]))/sigma2 )
    for jj = 1, n_elements(xwavi)-1 do y += a*yptsi[jj]*exp( (-.5*(xwav[ii]-
xwavi[jj])*(xwav[ii]-xwavi[jj]))/sigma2 )

    ypts[ii] = y
  endfor

  return, ypts

  ; note: xwav is also an output, placed as a keyword

end
```

Simulate_and_plot_ipt_spectrum.pro

```
Pro simulate_and_plot_ipt_spectrum

x=findgen(3000)*(1./2.4)+550.
;print,min(x),max(x),x
  tel=5.88013d
  nel=1578.87d
  euv_bin_size=0.6049d ; for euv Cassini UVIS (550-1150A)
  fuv_bin_size=0.7794d; for fuv Cassini UVIS  (1150-1800A)



  ;stop

mixingNSp=0.0639787 ;
  mixingNS2p=0.210333 ;
  mixingNS3p=0.0371900 ;
  mixingNS4p=0.0
  mixingNOp=0.257333 ;
  mixingNO2p=0.0232268 ;
  Ne_column=1.42376e+14 ;
  Ns4p=0.


  nsp=mixingNSp* Ne_column
```

```
  ns2p=mixingNS2p* Ne_column
  ns3p=mixingNS3p* Ne_column
  nop=mixingNop* Ne_column
  no2p=mixingNo2p* Ne_column
  min=min(x)
  max=max(x)
  fwhm=4.

;stop
  spec1=simulate_IPT_spectrum_rayleighs( x, Tel, nel, Nsp, Ns2p, Ns3p,Ns4p, Nop,
No2p, $
    min = min, max = max, fwhm = fwhm)
    ylog=0

    spec1[0:1440]=spec1[0:1440]/euv_bin_size
    spec1[1441:2999]=spec1[1441:2999]/fuv_bin_size

   ; stop

    p1=plot(x,spec1,title='Nominal IPT Conditions, Te=5.88 eV and ne=1579 cm^-
3',xtitle='Wavelength
(Angstroms)',ytitle='Rayleighs/Angstroms',xrange=[min(x),max(x)],ylog=ylog,yrange=[1.,60
.])
    stop

end
```

Runner_simulate_and_plot_ipt_spectrum.pro

```
pro runner_simulate_and_plot_ipt_spectrum

  ; set variable to path of main chianti directory, assuming chianti is in the current
working directory
  cd, current = current
  chiantipath = current + '/chianti/'

  ; or, if chianti is located elsewhere, edit the following variable
  ;chiantipath = '/Users/shinn/pro/chianti/' ; <---- user specific path

  ; use 'exists' keyword to determine whether to execute the following
  defsysv, '!chianti', exists = compile
  if ~compile then defsysv, '!chianti', 1

  if ~compile then defsysv, '!rjkm', 71492. ; RJ in km
  if ~compile then defsysv, '!kev', 8.617385d-5 ; Boltzmann's constant in eV
  if ~compile then defsysv, '!rootpi', sqrt(!dpi)

  ; create a string of all directories under chianti folder separated by ':'s
  if ~compile then chiantipros = ':' + expand_path('+' + chiantipath + 'idl/')
  ; if compile script has not been run before, add chianti path
  if ~compile then !path = !path + chiantipros
  if ~compile then use_chianti, chiantipath + 'dbase'
  ; compile the code
  resolve_routine, 'simulate_and_plot_ipt_spectrum'
  ; run the code
simulate_and_plot_ipt_spectrum

end
```