Draft version May 7, 2020
Typeset using LaTeX default style in AASTeX63

Internal Memo: An Unofficial Introduction to the Flexible Atomic Code and its Applications

Steven Bromley

(Dated: May 7, 2020)

1. INTRODUCTION

The ‘Flexible Atomic Code’ (FAC) is an integrated software package used “to calculate various atomic radiative and collisional processes, including energy levels, radiative transition rates, collisional excitation and ionization by electron impact, photoionization, autoionization, radiative recombination and dielectronic capture” (FAC). The FAC also contains a collisional-radiative (CR) code able to construct synthetic spectra under a variety of physical conditions. Depending on the size and amount of atomic data incorporated into the CR model, the author of the FAC noted that it may be beneficial for very large calculations to export the FAC atomic data and incorporate them into a different collisional-radiative code for faster execution.

This document is intended as an unofficial user’s guide and/or introduction to the Flexible Atomic Code. This document is intended to be supplementary to official FAC documentation, and as I have not written the FAC, may not be 100% accurate. I have written this document as I worked through the available FAC materials, i.e. the official manual, published papers, and source code, as I tried to generate atomic data relevant to EBIT plasmas. Throughout this document, I have tried to provide additional documentation for various functions that I thought was lacking or under-explained in the official FAC materials. If you find parts of this text unhelpful or questionable, I recommend double-checking the official materials to confirm the true purpose/meaning/use/units of the questionable part(s). Lastly, I have no problem sharing this document with any interested parties (students, faculty, or otherwise) that find this document helpful or useful. Please feel free to share as need be. I hope you find this document helpful.

In the following, I have detailed the installation of the code on a fresh “Windows Subsystem for Linux” (WSL) environment on Windows 10, which mimics the commands and capabilities of any Ubuntu or similar distribution on a Windows machine. By using WSL, the Linux utilities and codes have full access to your hardware, which may not be the case when running in a virtual machine. I want to note that it is not advised to move/add/remove files to/from the Ubuntu file system from any Windows program, e.g. Notepad++, as the file permissions may be disrupted and/or you may lose data. I would recommend saving the code on your machine twice: once in your Windows environment for readability purposes in something like Notepad++, and another in your WSL environment for execution. This prevents corrupting or introducing errors to the source code you are executing.

In the remainder of this document, the installation of the FAC and a sample of its applications are discussed. In Sec. 2, I have written a guide for installing the FAC and the KRONOS charge exchange database in WSL. In Sec. 3, the basics of electronic structure calculations with FAC are discussed. Sample scripts for producing energy levels and transition energies/rates are provided for a $\mathrm{Ne}^{(9-10)+}$ test case. Sec. 4 introduces FAC’s collisional-radiative code *crm*, and FAC/KRONOS data is used to generate synthetic charge exchange spectra involving the $\mathrm{Ne}^{(9-10)+}$ test case and neutral H_2. Sec. 5 discusses moving files out of the WSL environment and provides a sample script for convolving spectra files with gaussian functions.

In Sec. 6, using the FAC to calculate transition rates/cross sections for a number of other atomic processes relevant to EBIT experiments is discussed, including electron impact excitation (Sec. 6.1), electron impact ionization (Sec. 6.2), radiative recombination and photoionization (Sec. 6.3), and dielectronic recombination/autoionization (Sec. 6.4). Finally, in Sec. 7, I discuss the application of the FAC to produce atomic data necessary to model EBIT spectra. The capabilities of the FAC *crm* module are discussed in context of simulating EBIT spectra.

2. INSTALLATION

In this guide, we assume a fresh install of the Windows Subsystem for Linux (WSL) without modifications. Be advised that this method is not guaranteed to work for other systems e.g. Cygwin, MAC OS, or native Ubuntu. These instructions have been confirmed to work on 2 (to date) WSL environments. When first running WSL, be aware that the initial directory is inside the Windows filesystem. To move to the main WSL linux directory, type ‘cd’ and press enter. In the following, I detail the download, installation, and common problems encountered when installing the FAC. When installing, copy and run the commands encased *inside* the <> characters only. Comments on some commands are provided following a # character. The required packages are:

- <sudo apt install build-essential> # This installs the C compiler *gcc* and some other utilities
- <sudo apt install gfortran> # This is the Fortran compiler
- <sudo apt install python-dev> # This is required to run the FAC Python interface *pfac* discussed later.
- <sudo apt install git> # The *git* package is used to pull the FAC code from the source repository.

To copy the FAC code onto your file system, run the following in your home directory:

- <git clone https://www.github.com/flexible-atomic-code/fac/ >

Your FAC code should now be in a folder called ‘fac.’ At this point, one can follow the installation instructions noted in the git repository and the manual. However, I have run into some seemingly common issues with installing FAC in WSL. These did not appear to be an issue when installing on a Ubuntu Virtual Machine, so you may be able to skip these additional steps.

First, we need to add the $PATH variable to the .bashrc file. This will enable your install to search in the correct directories for the packages installed above. We will do this with the text editor ‘nano’ (<sudo apt install nano>) though you can use whichever text editor you prefer. In your home directory, run the following:

- <nano ~/.bashrc>

At the end of the file, add the following line and run the ‘source’ command to incorporate these changes into your current WSL environment:

- <export PATH="$HOME/bin:$PATH"> # add this to the .bashrc file
- <source ~/.bashrc> #Execute this on command line after closing the .bashrc file

There seems to be a missing symbolic link in WSL. We can simply add it via the following

- <sudo ln -s /usr/lib/gcc/x86_64-linux-gnu/7/libgcc_s.so /usr/lib/x86_64-linux-gnu/>

Lastly, it seems to be an issue on WSL that the Makefiles for the FAC code do not know which fortran interpreter to use. To set the fortran interpreter and install the FAC, execute the following commands:

- <export F77=gfortran> #This sets gfortran as the fortran compiler. This step does not seem necessary on my Ubuntu VM, but was required to successfully install in WSL.

We can now run the install instructions in the FAC READ_ME file:

- <./configure>
- <sudo make>
- <sudo make install>
- <sudo make pfac> #This prepares the python interface *pfac*
- <sudo make install-pfac>

If no errors are printed to the terminal: Congratulations! You have successfully installed the Flexible Atomic Code. If not, most errors arise from either (a) missing packages, in which case the terminal will inform you of the missing package(s) to install, or (b) a package is installed but the code is looking in the wrong directories for the packages. This will have to be handled on a case-by-case basis, and Google is most helpful in this case.

Lastly, for some projects it may be of interest to incorporate charge exchange into the model. The FAC is equipped with functions for loading charge exchange cross sections from the KRONOS charge exchange database and calculating spectra resulting from ion-neutral collisions. The KRONOS database currently (as of 16-04-2020) contains cross sections for single electron capture of many H-like and He-like ions. I have heard that multi-electron transfer and other methods are being applied to expand the number of cross sections in the database. Currently, some cases have cross sections produced from multiple methods, but most ions are restricted to cross sections calculated with the Multichannel Landau-Zener (MCLZ) approach.

To incorporate charge exchange into the collisional-radiative model down the line, we will need a copy of the KRONOS charge exchange database run by Dr. Phillip Stancil at UGA. To avoid issues with Windows/Linux file permissions, we will pull the database files directly with the *wget* package. Alternatively, one could download the .zip to the Windows Download folder and copy them to the Linux filesystem with <cp /mnt/c/Users/XXX/Downloads/Kronos_v3.1 .> (with XXX changed to your Windows username). In the home directory, run the following to install *wget*, download the KRONOS files, and un-zip them:

- <sudo apt install wget> #The *wget* package allows you to download files directly from a website via command line
- <wget https://www.physast.uga.edu/owncloud/index.php/s/2WiLZ6Y0oKr467m>
- <sudo apt install unzip>
- <unzip download> #Note: The .zip file was saved with the name 'download' in WSL, but saves as 'Kronos_v3.1.zip' in Windows 10. The file extension (not shown in WSL) is .zip.

You now have all of the charge exchange cross section data in the KRONOS database. The path to these files e.g. '/home/steve/Kronos_v3.1/' will be needed when running the *crm* code later.

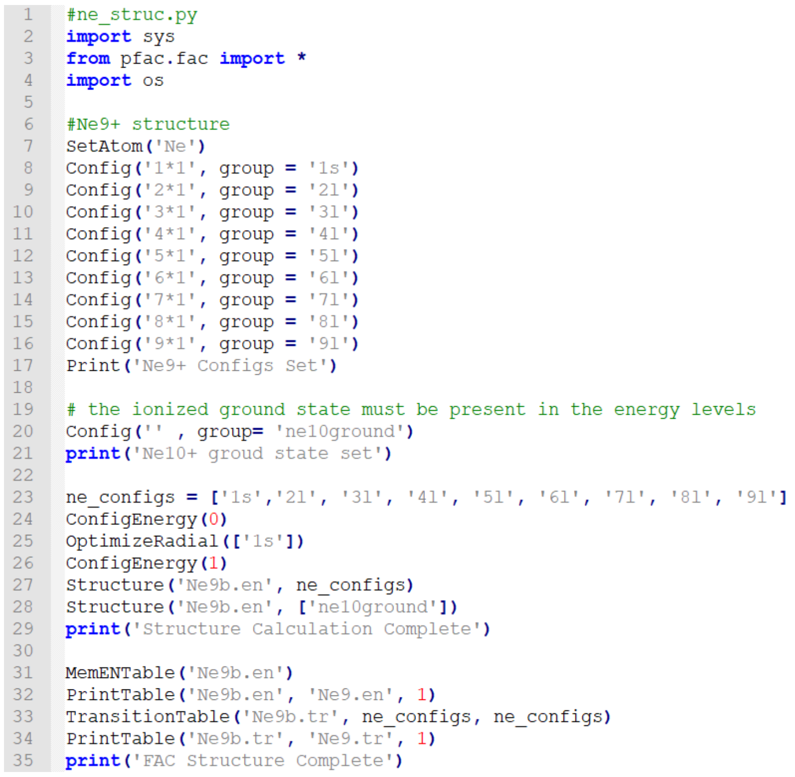
3. CALCULATING ELECTRONIC STRUCTURE WITH THE FLEXIBLE ATOMIC CODE

No matter the process of interest, one of the first steps is to calculate the electronic structure of the system(s) involved. I have included a script above (Fig. 1) for calculating the electronic structure of Ne^{9+} which we will use to generate synthetic spectra following charge exchange between Ne^{10+} and neutral H_2. Note that the scripts discussed below are written and executed in Python to take advantage of the scripting capabilities within the *pfac* interface. In this way, Python scripting/syntax can be used to set up and run the code, and C/Fortran provides fast computation behind the scenes. To run these scripts, execute with e.g. <python ne_struc.py>.

Following collisions of Ne^{10+} and a neutral target, the initial electron capture states are dependent on the cross sections in the KRONOS database. The photon cascade following this initial capture will depend on the electronic structure of Ne^{9+}, including the allowed transitions, energy levels, and transition rates A. Fig. 1 shows a simple script to calculate these levels and transitions. I have commented below on some of the things to understand about each function call.

- Line 3: Import the FAC functions from the *pfac* library
- Line 7: Set the Ion/Atom. Ions are selected by their atomic symbol, e.g. Ne, Fe, ... The number of electrons is inferred from the configurations, and the SetAtom() command simply sets the nuclear charge.
- Lines 8-16: Each call to fac.Config() adds an electronic configuration to the calculation. For example, line 8 <Config('1*1', group = '1s')> adds all configurations in the $n = 1$ shell. The group of levels within this configuration are denoted by the string label '1s.' Note that line 10, which adds the configurations '3*1,' adds all $n = 3$ levels, i.e. $3s, 3p$, and $3d$. If specific orbitals or configurations need to be specified, it is possible to do something such as <Config('3s1 3p1', group = '3s3p')>.

This structure calculation will be used to calculate synthetic spectra of Ne^{9+} ions following charge exchange between Ne^{10+} and neutral H_2. It is assumed that the initial ion is in the ground state, so only 1 level/configuration must be included for that ion. For example, if we were studying the CX spectra of He^{+} ions colliding with neutral H, our structure calculation would need the ground state of He^{+} (1s) and the full structure of neutral He. In our current case, our projectile is a bare neon ion, and thus has 0 electrons. However, the structure calculation still needs to know



```
 1  #ne_struc.py
 2  import sys
 3  from pfac.fac import *
 4  import os
 5
 6  #Ne9+ structure
 7  SetAtom('Ne')
 8  Config('1*1', group = '1s')
 9  Config('2*1', group = '2l')
10  Config('3*1', group = '3l')
11  Config('4*1', group = '4l')
12  Config('5*1', group = '5l')
13  Config('6*1', group = '6l')
14  Config('7*1', group = '7l')
15  Config('8*1', group = '8l')
16  Config('9*1', group = '9l')
17  Print('Ne9+ Configs Set')
18
19  # the ionized ground state must be present in the energy levels
20  Config('' , group= 'ne10ground')
21  print('Ne10+ groud state set')
22
23  ne_configs = ['1s','2l', '3l', '4l', '5l', '6l', '7l', '8l', '9l']
24  ConfigEnergy(0)
25  OptimizeRadial(['1s'])
26  ConfigEnergy(1)
27  Structure('Ne9b.en', ne_configs)
28  Structure('Ne9b.en', ['ne10ground'])
29  print('Structure Calculation Complete')
30
31  MemENTable('Ne9b.en')
32  PrintTable('Ne9b.en', 'Ne9.en', 1)
33  TransitionTable('Ne9b.tr', ne_configs, ne_configs)
34  PrintTable('Ne9b.tr', 'Ne9.tr', 1)
35  print('FAC Structure Complete')
```

Figure 1. FAC script for calculating the electronic structure of Ne^{9+}. All electron configurations up to $n = 9$ are included. Note the ground state of Ne^{10+} (0 electrons) is included (see text).

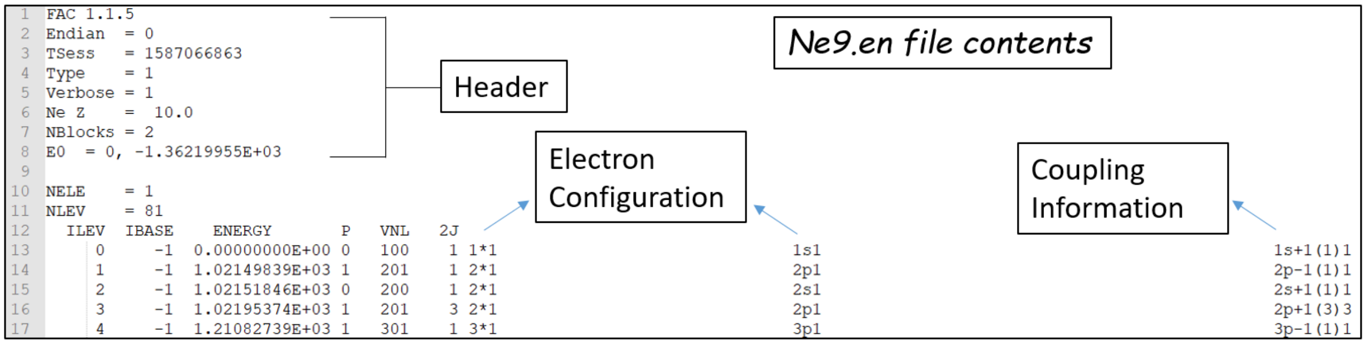
the electronic state, which is stored as some type of NULL value inside the code. The ground state of our projectile Ne^{10+} is incorporated on Line 20 with an empty string in place of a proper configuration (Note: this is discussed in the FAQ inside the FAC manual):

- Config(‘’, group=’ne10ground’)

Line 23 is an array storing all configuration labels. We will use this later to calculate the transitions. Lines 24 - 26 involve calculating/optimizing the single central potential used to treat the wavefunctions. The call <ConfigEnergy(0)> (line 25) first optimizes each configuration group separately. The call <OptimizeRadial([‘1s’])> on line 25 then optimizes the wavefunction of only the ‘1s’ (ground) level in the presence of this potential. On line 26, the second call to ConfigEnergy(1) will then “recalculate the average energy of the configuration groups under the potential taking into account all configuration groups” (FAC).

Lines 27-28 calculate the electronic structure of our system and saves it to the binary files “Ne9b.en.” The file name here is very important: **file extensions and naming conventions must follow those outlined in the manual**. There are several key conventions here. Files containing the structure **must** end with a .en extention. The ‘b’ before the file extension signifies that the data is stored in binary format. Line 31 saves this data to memory, and the PrintTable() call on line 32 takes the data in file ‘Ne9b.en’, converts it to ASCII format, and saves it to the file ‘Ne9.en’ in verbose (human read-able) mode which is specified by ‘1’ in the last parameter of the function call. Line 33 constructs a table in memory of all possible transitions between *all* levels in array ‘ne_configs’ and stores them in the binary file ‘Ne9b.tr’. The function call PrintTable() on line 34 then converts the binary file to ASCII and stores them in human-readable form in file ‘Ne9.tr’.

Figures 2 and 3 shows a sample of the output of this calculation. In the levels file ‘Ne9.en’, the contents stored from left to right (after the header) are: numerical level label, ibase = -1 (ignore), energy (in eV), parity (0 = even, 1 = odd), the level’s J value times 2, two columns detailing the electron configuration, and coupling information (last column). For our purposes, the most interesting/useful columns are the numerical label ILEV, the energy, parity, J value, and configuration(s).



```
FAC 1.1.5
Endian   = 0
TSess    = 1587066863
Type     = 1
Verbose = 1
Ne Z     =  10.0
NBlocks = 2
E0  = 0, -1.36219955E+03

NELE     = 1
NLEV     = 81
  ILEV  IBASE    ENERGY       P   VNL   2J
     0     -1  0.00000000E+00 0   100    1 1*1                    1s1                       1s+1(1)1
     1     -1  1.02149839E+03 1   201    1 2*1                    2p1                       2p-1(1)1
     2     -1  1.02151846E+03 0   200    1 2*1                    2s1                       2s+1(1)1
     3     -1  1.02195374E+03 1   201    3 2*1                    2p1                       2p+1(3)3
     4     -1  1.21082739E+03 1   301    1 3*1                    3p1                       3p-1(1)1
```

Figure 2. Sample of the data stored in the file 'Ne9.en' produced by the script in Fig. 1

.

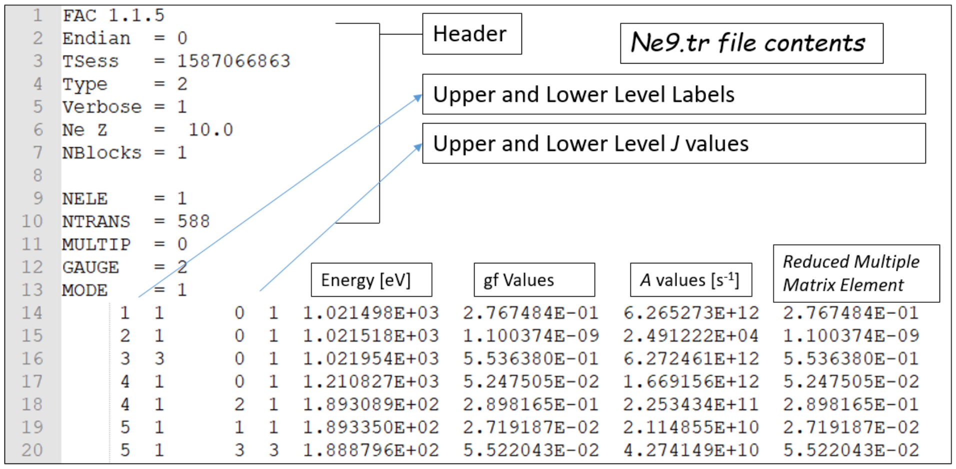
In the transitions file 'Ne9.tr', the quantities stored after the 13-line header are (from left to right): upper and lower level numerical labels, upper and lower level J values, the transition energy in eV, the gf value of the transition, the transition A value in s^{-1}, and the reduced multiple matrix element. See the FAC manual for more details.

These scripts and outputs are meant to highlight and show some of the capabilities of the FAC. Depending on the process of interest, you may need to utilize other functions provided in the FAC documentation. To see examples of these, I would recommend checking out some of the sample scripts in the 'demo' folder inside your FAC directory.

4. SIMULATING CHARGE EXCHANGE WITH FAC AND THE *CRM* MODULE

We are now equipped to generate synthetic spectra of Ne^{9+} following single electron capture in Ne^{10+}-H_2 collisions. We will carry out this calculation with functions from FAC's collisional-radiative module *crm*. The FAC CR model will be discussed in detail in Sec. 7. Here I provide a simple example for quickly generating charge exchange spectra and discuss aspects of the *crm* module relevant to charge exchange. The script "ne_cxs.py", shown in Fig. 4, calculates synthetic spectra using our FAC levels and transitions and CX cross sections from the KRONOS database. In the following, the details of each function call are described in detail.

Line 5 sets the directory for the KRONOS database as discussed in Section 2. Line 6 pulls cross section data from the KRONOS database with the ReadKronos() function call. The variables here are, from left to right: nuclear charge (10), initial electronic charge (0), atom name as string ('Ne'), target name as string ('H2'), method (optional; options are per source code in 'rates.c': 'rcmd', 'qmocc', 'mocc', 'aocc', 'ctmc', 'mclz', 'faclz'), l-distribution (optional), s ("indicates if interpolation of cross sections to be performed in logarithmic scale" FAC). It is important to note that the KRONOS cross sections are n-resolved. Acquiring nl resolved cross sections requires spreading the n resolved cross



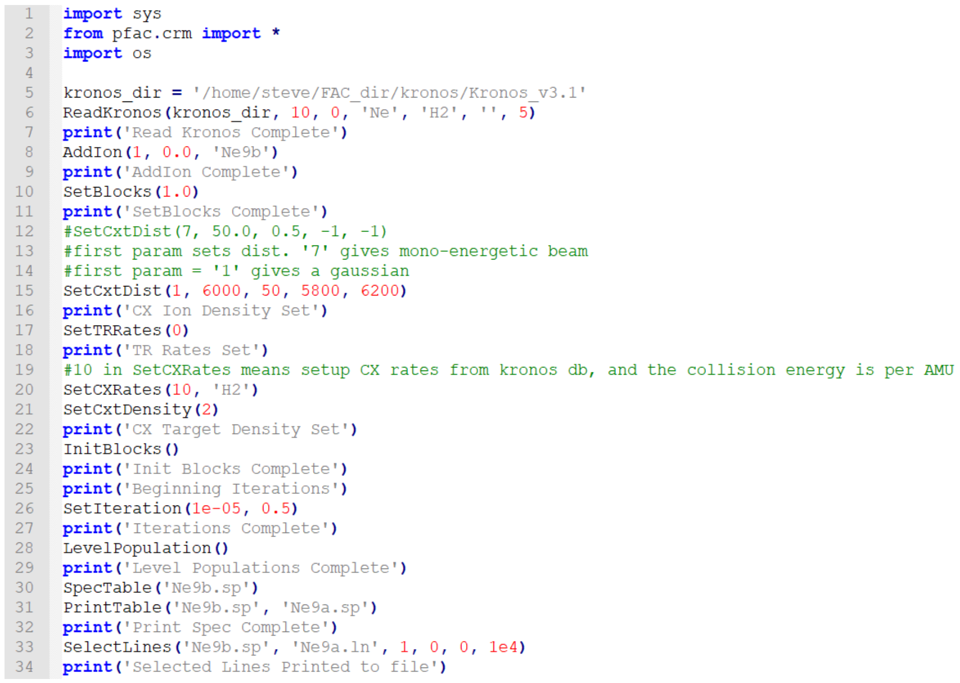
```
FAC 1.1.5
Endian   = 0
TSess    = 1587066863
Type     = 2
Verbose = 1
Ne Z     =  10.0
NBlocks = 1

NELE     = 1
NTRANS   = 588
MULTIP   = 0
GAUGE    = 2
MODE     = 1
     1  1      0  1  1.021498E+03  2.767484E-01  6.265273E+12  2.767484E-01
     2  1      0  1  1.021518E+03  1.100374E-09  2.491222E+04  1.100374E-09
     3  3      0  1  1.021954E+03  5.536380E-01  6.272461E+12  5.536380E-01
     4  1      0  1  1.210827E+03  5.247505E-02  1.669156E+12  5.247505E-02
     4  1      2  1  1.893089E+02  2.898165E-01  2.253434E+11  2.898165E-01
     5  1      1  1  1.893350E+02  2.719187E-02  2.114855E+10  2.719187E-02
     5  1      3  3  1.888796E+02  5.522043E-02  4.274149E+10  5.522043E-02
```

Header
Ne9.tr file contents
Upper and Lower Level Labels
Upper and Lower Level J values
Energy [eV]
gf Values
A values [s-1]
Reduced Multiple Matrix Element

Figure 3. Sample of the data stored in the file 'Ne9.tr' produced by the script in Fig. 1

.



```
 1  import sys
 2  from pfac.crm import *
 3  import os
 4
 5  kronos_dir = '/home/steve/FAC_dir/kronos/Kronos_v3.1'
 6  ReadKronos(kronos_dir, 10, 0, 'Ne', 'H2', '', 5)
 7  print('Read Kronos Complete')
 8  AddIon(1, 0.0, 'Ne9b')
 9  print('AddIon Complete')
10  SetBlocks(1.0)
11  print('SetBlocks Complete')
12  #SetCxtDist(7, 50.0, 0.5, -1, -1)
13  #first param sets dist. '7' gives mono-energetic beam
14  #first param = '1' gives a gaussian
15  SetCxtDist(1, 6000, 50, 5800, 6200)
16  print('CX Ion Density Set')
17  SetTRRates(0)
18  print('TR Rates Set')
19  #10 in SetCXRates means setup CX rates from kronos db, and the collision energy is per AMU
20  SetCXRates(10, 'H2')
21  SetCxtDensity(2)
22  print('CX Target Density Set')
23  InitBlocks()
24  print('Init Blocks Complete')
25  print('Beginning Iterations')
26  SetIteration(1e-05, 0.5)
27  print('Iterations Complete')
28  LevelPopulation()
29  print('Level Populations Complete')
30  SpecTable('Ne9b.sp')
31  PrintTable('Ne9b.sp', 'Ne9a.sp')
32  print('Print Spec Complete')
33  SelectLines('Ne9b.sp', 'Ne9a.ln', 1, 0, 0, 1e4)
34  print('Selected Lines Printed to file')
```

Figure 4. Python script for generating synthetic CX spectra following collisions of Ne^{10+} and neutral H$_2$.

.

sections over a number of l levels by specifying an l-distribution. The options for the l distributions (variable '5' in the ReadKronos() call in Fig. 4) are taken from the Kronos documentation as follows:

- 1) $m = 1$: Low Energy Distribution. This distribution is favored for collisions at energies $< 10-100$eV/u (Mullen *et al.* 2016). The weighting factor for a particular nl level goes as

$$W_{nl}^{le} = (2l+1)\frac{[(n-1)!]^2}{(n+1)!(n-1-l)!} \tag{1}$$

- 2) $m = 2$: Statistical Distribution. This distribution is recommended for collisions at energies > 10keV/u (Mullen *et al.* 2016). The weighting factor for each nl level goes as

$$W_{nl}^{st} = \frac{2l+1}{n^2} \tag{2}$$

- 3) $m = 3$: Modified Low Energy Distribution. The modified LE distribution is similar to the low energy distribution above, but is modified to produce 0 populations of $l = 0(s)$ states:

$$W_{nl} = l(l+1)(2l+1)\frac{(n-1)!(n-2)!}{(n+l)!(n-l-1)!} \tag{3}$$

- 4) $m = 4$: Separable Distribution. Each level is weighted by (Smith *et al.* 2014):

$$W(l) = (\frac{2l+1}{q})\exp[-\frac{l(l+1)}{q}] \tag{4}$$

- 5) $m = 5$: Shifted Low Energy Distribution. This is discussed briefly in Cumbee *et al.* (2017), and is intended to enhance the $l = 1(p)$ populations. Further documentation pending.

Line 8 adds Ne^{9+} into the model. The function takes in 3 variables: (1) the number of electrons present in the ion (1), (2) the initial density of the ion (0), and the file extension for the structure of this ion ('Ne9b'). **Without the 'b' at the end of the file extension, the model will produce no output. It is imperative that the 'b' is present at the end of the file names.** The SetBlocks() function call on line 10 performs an initial setup of the spectral model.

Line 15 sets the energy distribution of the ion beam. In the function call SetCxtDist(i, ...) the parameter i sets the choice of energy distribution of the projectile ions (Ne^{10+}). The choices of parameter i do not follow the same convention as those outlined in the manual. The distributions on page 32 of the manual are for electrons, not ions. The choices of ion distributions are (from the source code file 'rates.c'):

- $i = 0$: Maxwellian.
- $i = 1$: Gaussian
- $i = 2$: "MaxPower"
- $i = 4$: "Shifted Maxwellian"
- $i = 5$: "Hybrid Maxwellian; Powerlaw and Maxwellian continuously matched"
- $i = 6$: Double Maxwellian
- $i = 7$: Mono-energetic

From line 15 in Fig. 4, we have chosen a Gaussian distribution with energy 6000 eV, FWHM 50 eV, minimum energy 5800 eV, maximum energy 6200 eV.

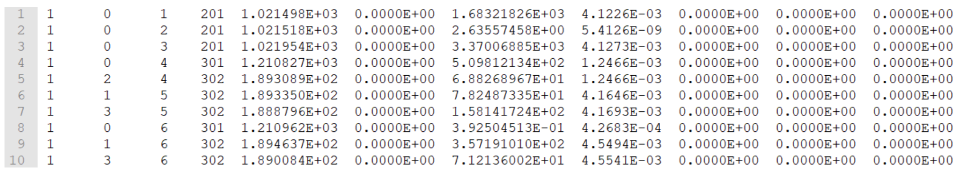
Line 17 loads the transitions rates (the A values) of all transitions in the file 'Ne9b.tr.' The function call SetCXRates(m, t) on line 20 pulls in CX cross sections from the KRONOS database. This function takes in two variables: m, and t. The parameter m tells the function how/where to look for the cross sections. This function was set up by Dr. Gu (author of FAC) when I asked him for help in running these calculations, and I am unsure of how to modify this. The second parameter t is simply a string containing the name of the neutral target. On line 21, the density of the neutral target is set in units of 10^{10} cm^{-3}. In this case, we have set the neutral density to 2×10^{10} cm^{-3}.

Line 23 initializes the model, and the SetIteration(1e-05, 0.5) call on line 26 iterates the model solution. The first parameter is a convergence criteria, and the latter is a "stabilizer" (FAC). The LevelPopulation() call on line 28 solves for the steady-state solution of the level populations. In our case, this does not seem so important but is (I think) a necessary step in running the codes and producing useful output.

Lastly, lines 30-33 involve saving and storing the resulting spectra. The SpecTable() call on line 30 saves the spectra in memory and stores them to the binary file 'Ne9b.sp.' The following PrintTable() converts this binary file to ASCII and stores it in the file 'Ne9a.sp.' **Again, it is imperative that the file format retains the letters 'b' or 'a' before the extension.** Lastly, the function call on line 33, SelectLines('Ne9b.sp', 'Ne9a.ln', 1, 0, 0, 1e4), saves the lines to the ASCII value 'Ne9a.ln.' The additional parameters are detailed in the FAC manual in the SelectLines() function on page 77. I have summarized them below for function SelectLines(ifn, ofn, n, t, $e0$, $e1$, s):

- ifn: binary database file
- ofn: ASCII-formatted output file
- n: number of electrons of the ion of interest
- t: transition type. If $t = 0$, all transition types (e.g. E1, M1, M2, ...) are included.
- $e0$: minimum of energy range
- $e1$: maximum of energy range
- s: optional parameter which allows you to save lines based on level indices (see FAC manual).

Looking at our SelectLines() call, we have chosen to save all lines (of all types) between 0 - 1e4 eV to the file 'Ne9a.ln.' A sample of the .ln file is shown in Fig. 5. From left to right, the columns are enumerated in the SelectLines() source code in 'crm.c': (0) number of electrons, (1) lower level index, (2) upper level index, (3) transition type, (4) energy (eV), (5) sdev, (6) strength, (7) wtr (?), (8) wtt0 (?), (9) wtt (?), (10) wd (?). Parameters marked with a (?) are things I am uncertain of.



```
1      0      1    201  1.021498E+03  0.0000E+00  1.68321826E+03  4.1226E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      0      2    201  1.021518E+03  0.0000E+00  2.63557458E+00  5.4126E-09  0.0000E+00  0.0000E+00  0.0000E+00
1      0      3    201  1.021954E+03  0.0000E+00  3.37006885E+03  4.1273E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      0      4    301  1.210827E+03  0.0000E+00  5.09812134E+02  1.2466E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      2      4    302  1.893089E+02  0.0000E+00  6.88268967E+01  1.2466E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      1      5    302  1.893350E+02  0.0000E+00  7.82487335E+01  4.1646E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      3      5    302  1.888796E+02  0.0000E+00  1.58141724E+02  4.1693E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      0      6    301  1.210962E+03  0.0000E+00  3.92504513E-01  4.2683E-04  0.0000E+00  0.0000E+00  0.0000E+00
1      1      6    302  1.894637E+02  0.0000E+00  3.57191010E+02  4.5494E-03  0.0000E+00  0.0000E+00  0.0000E+00
1      3      6    302  1.890084E+02  0.0000E+00  7.12136002E+01  4.5541E-03  0.0000E+00  0.0000E+00  0.0000E+00
```

Figure 5. Sample of the contents of the ASCII file ‘Ne9a.ln’ produced by the CX spectra script in Fig. 4

.

5. MOVING FILES OUT OF WSL AND PLOTTING

With the .ln file in hand, we can now plot a meaningful spectra. Rather than set up some type of third party GUI for WSL, I have opted to pull my output files out of the WSL environment for plotting. One could automate this saving with the ‘sys’ package to automatically save the output locally in WSL and to some folder in the regular windows file system that mimics the same directory structure as the WSL environment. To move the .ln file from within WSL to a directory in my regular Windows filesystem, I would use something like the following:

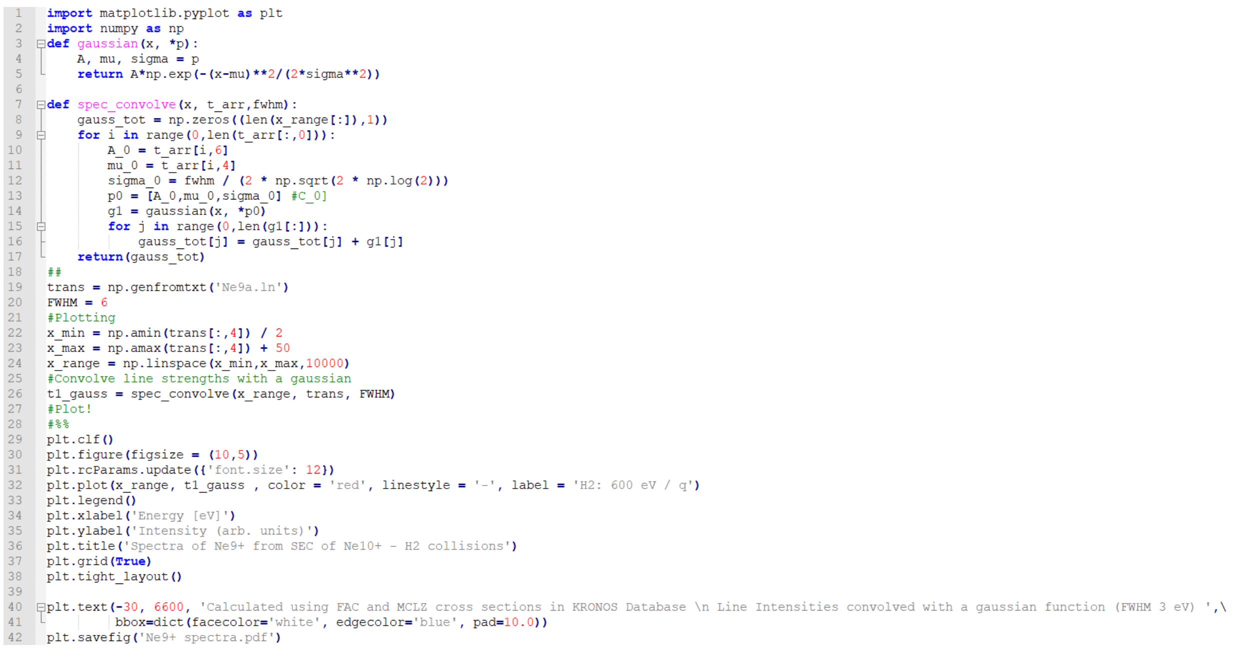
- <cp /home/steve/fac/test/Ne9a.ln /mnt/c/Users/steve_000/Desktop/fac_plotting/>

or from within the current directory (/home/steve/fac/test/):

- < cp ./Ne9a.ln /mnt/c/Users/steve_000/Desktop/fac_plotting>

Here I am using the ‘cp’ command to copy the file ‘Ne9a.ln’ out of my WSL environment and into my Windows filesystem. The directory ‘/mnt/c’ allows one to access Windows files from within the WSL environment. Here I have copied the ‘Ne9a.ln’ file into the folder ‘fac_plotting’ on my Windows desktop.

One could simply plot column (4) versus column (6) to produce a stick plot of each transition strength versus energy. To produce a more meaningful spectra, I have written a quick script to convolve each peak in the .ln file with a gaussian. The script “ne10_plot.py” in Fig. 6 should produce the output ‘Ne9+ spectra.pdf’ in Fig. 7. Note that the label ‘600 eV per q’ indicates the total energy (6000 eV) divided by the charge state of the initial ion (+10).



```python
import matplotlib.pyplot as plt
import numpy as np
def gaussian(x, *p):
    A, mu, sigma = p
    return A*np.exp(-(x-mu)**2/(2*sigma**2))

def spec_convolve(x, t_arr,fwhm):
    gauss_tot = np.zeros((len(x_range[:]),1))
    for i in range(0,len(t_arr[:,0])):
        A_0 = t_arr[i,6]
        mu_0 = t_arr[i,4]
        sigma_0 = fwhm / (2 * np.sqrt(2 * np.log(2)))
        p0 = [A_0,mu_0,sigma_0] #C_0]
        g1 = gaussian(x, *p0)
        for j in range(0,len(g1[:])):
            gauss_tot[j] = gauss_tot[j] + g1[j]
    return(gauss_tot)
##
trans = np.genfromtxt('Ne9a.ln')
FWHM = 6
#Plotting
x_min = np.amin(trans[:,4]) / 2
x_max = np.amax(trans[:,4]) + 50
x_range = np.linspace(x_min,x_max,10000)
#Convolve line strengths with a gaussian
t1_gauss = spec_convolve(x_range, trans, FWHM)
#Plot!
#%%
plt.clf()
plt.figure(figsize = (10,5))
plt.rcParams.update({'font.size': 12})
plt.plot(x_range, t1_gauss , color = 'red', linestyle = '-', label = 'H2: 600 eV / q')
plt.legend()
plt.xlabel('Energy [eV]')
plt.ylabel('Intensity (arb. units)')
plt.title('Spectra of Ne9+ from SEC of Ne10+ - H2 collisions')
plt.grid(True)
plt.tight_layout()

plt.text(-30, 6600, 'Calculated using FAC and MCLZ cross sections in KRONOS Database \n Line Intensities convolved with a gaussian function (FWHM 3 eV) ',\
         bbox=dict(facecolor='white', edgecolor='blue', pad=10.0))
plt.savefig('Ne9+ spectra.pdf')
```

Figure 6. Plotting script "ne10_plot.py". The script plots the line intensities (convolved with a gaussian) from the file 'Ne9a.ln' produced by the CX spectra script in Fig. 4. The script produces the figure 'Ne9+ spectra.pdf' shown in Fig. 7.

.

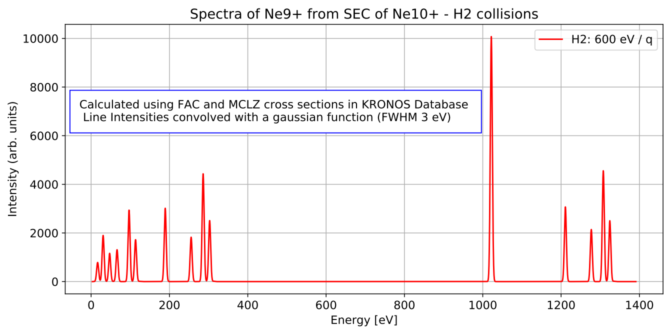


Figure 7. Output plot from plotting script "ne10_plot.py" in Fig. 6.

6. GENERATING ATOMIC DATA FOR COLLISION PROCESSES WITH THE FAC

As noted in Sec. 1, the FAC is capable of producing cross sections / rates for a variety of processes. These processes include electron impact excitation (EIE), electron impact ionization (EII), radiative recombination (RR), dielectronic recombination (DR), autoionization (AI), and photoionization (PI). In the following, I provide explanations and sample scripts to calculate atomic data for each process. Each process here (save for photoionization) is incorporated into a CR model of an EBIT plasma in Sec. 7.

6.1. *Electron Impact Excitation*

Electron impact excitation (EIE) is the process in which an electron collides with an atom or ion A and promotes an electron to an excited state of the atom or ion. The decay of this excited state is dictated by the transition rate (the 'A' value), and in some environments where the electron density is large enough, the cross section for collisional de-excitation. The EIE process may be written

$$e^- + A \rightarrow A^* + e^- \rightarrow A + e^- + \gamma \tag{5}$$

where the photon γ carries the energy lost by the electron. In the FAC, calculation of the collision strength, i.e. the probability that a given electronic state of A is populated by electron impact, is calculated in the distorted wave (DW) formalism. The details of this process are available in Gu (2008). Simply, the impacting electron is represented by a series of "continuum orbitals" as the electron is unbound. These continuum orbitals couple to the electronic states of the ion/atom A, and the net result of a collision is the transfer of some of the impacting electron's kinetic energy into one or more of the electrons bound to A. Calculation of the EIE rates first requires knowledge of the electronic structure, which may be calculated as in e.g. Fig. 1.

In FAC, the electron impact excitation rates are calculated with a single command. By default, the electron energy grid consists of 6 energy values based on the transition energies present in the .tr file generated by the structure calculation. Alternatively, one can set the electron energy grid manually (see FAC for variations) with a python grid, e.g.

- <e_grid = [1,10,1000,5000]> # energies are listed in eV
- <SetCEGrid(e_grid)>

For our purposes here, I have let FAC construct the energy grid via the default options. Later, when calculating cross sections at a number of energies, the FAC uses interpolation/extrapolation methods to calculate the cross section(s) at energies absent from the default grid.

A sample script for calculating electron impact excitation cross sections is shown in Fig. 8. The script calculates electron impact excitation cross sections for all possible combinations of upper and lower levels. In the following, the purpose and use of each line is described.

- Lines 43 - 44: Define exponents used for the np.logspace() function. The np.logspace function defines a logarithmic grid of points from $10^{\mathrm{e\_low}}$ to $10^{\mathrm{e\_high}}$ with *n_points* number of points.
- Lines 45: Define the number of points in the np.logspace() grid. This will be used as the grid over which to print the *output* cross sections. The energy grid used to calculate the cross sections for the DW calculation is set by SetCEGrid() discussed previously.
- Lines 46 - 47: Define the np.logspace() grid and convert it to a list.
- Lines 48 - 49: Define lower and upper levels for CECross() (discussed later)
- Line 50: (Optional) Re-define the energy for the output (*not* input energy grid for the DW calculation)

Maintaining the file naming convention from the manual, calculating the electron impact excitation rates of Ne^{9+} would proceed as follows with line 51:

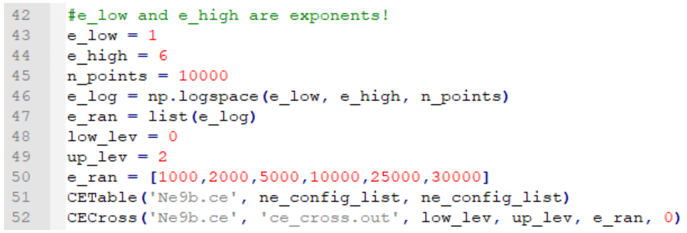
- <fac.CETable('Ne9b.ce', ne_config_list, ne_config_list)>

Note that this calculation takes *significantly* longer than the structure calculation above. The CETable() call above calculates electron impact excitation rates for all possible transitions between all levels. The FAC must first construct the continuum orbitals, then handle the coupling between all continuum orbitals and the levels in ne_config_list. When running an EIE calculation for only two shells, i.e. $n = 2$ to $n = 3$ excitations, the CETable() call took my machine 27 s. Including all configurations listed in the structure script took 489 minutes (∼8 hours). Luckily, these values (in theory) only need to be calculated once. Alternatively, if trying to modify the calculations, it would be best to run these jobs on a computer cluster (e.g. Palmetto) as FAC can be operated with MPI across multiple computing nodes.

The output of the electron impact excitation calculation above is in a difficult-to-read format, even when saved in verbose mode. The function

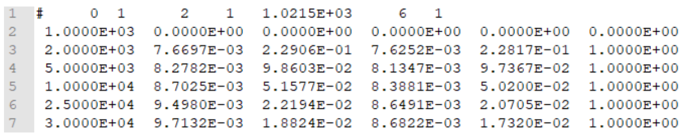
- <fac.CECross(ifn, ofn, *low*, *up*, e, m (opt.))>

on line 52 can be used to convert the .ce file into a more useful format. This function pulls the collision strengths from the .ce file (ifn), calculates the cross sections for EIE from level *low* to level *up* at energies listed in e, and saves the output to file ofn. Note that the level designations *low* and *up* must be integer values, i.e. the integer label 'ILEV' in Fig. 2. The output file ofn can be any file name. The energies e may be different than those used for the 6-point internal grid of the DW calculation. The values in this internal grid are used to extrapolate or interpolate to the values



```
42  #e_low and e_high are exponents!
43  e_low = 1
44  e_high = 6
45  n_points = 10000
46  e_log = np.logspace(e_low, e_high, n_points)
47  e_ran = list(e_log)
48  low_lev = 0
49  up_lev = 2
50  e_ran = [1000,2000,5000,10000,25000,30000]
51  CETable('Ne9b.ce', ne_config_list, ne_config_list)
52  CECross('Ne9b.ce', 'ce_cross.out', low_lev, up_lev, e_ran, 0)
```

Figure 8. Sample script showing the application of the fac.CECross() function.



```
#      0   1       2     1    1.0215E+03      6    1
 1.0000E+03  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00  0.0000E+00
 2.0000E+03  7.6697E-03  2.2906E-01  7.6252E-03  2.2817E-01  1.0000E+00
 5.0000E+03  8.2782E-03  9.8603E-02  8.1347E-03  9.7367E-02  1.0000E+00
 1.0000E+04  8.7025E-03  5.1577E-02  8.3881E-03  5.0200E-02  1.0000E+00
 2.5000E+04  9.4980E-03  2.2194E-02  8.6491E-03  2.0705E-02  1.0000E+00
 3.0000E+04  9.7132E-03  1.8824E-02  8.6822E-03  1.7320E-02  1.0000E+00
```

Figure 9. Output of the fac.CECross() call in Fig. 8.

at the energies in *e_ran*. Alternatively, one could define the electron energy grid for the CETable() calculation with the SetCEGrid() function discussed previously. The last argument ('m' in FAC) controls whether the cross sections are output as a function of incident ($m = 0$) or scattering energy ($m = 1$). Here, $m = 0$ will lead the CECross() call to produce cross sections as a function of incident electron energy.

Note that in the EIE script in Fig. 8, the logarithmic grid (line 46) is unnecessary to run the script as we have redefined the array of energy values 'e_ran' on line 50. However, I have included it here for educational purposes as it is an easy way to automate the energy grid. The re-defined list for this example (line 50) contains energies at 1, 2, 5, 10, 25, and 30 keV respectively.

On line 52, I show an example of the CECross() call. The resulting cross sections are saved in 'ce_cross.out' which is shown in Fig. 9. Note that this the CECross() calculation took ~2.5 ms. Assuming the same timescale for each pair of levels, this step should only take $(\binom{80}{2} = 3160) * 2.5 \times 10^{-3} = 8$ seconds for the entirety of our system.

From left to right, the information contained in the header is: (1) lower level label, (2) upper level label, (3) lower level J, (4) upper level J, (5) e (?), (6) # of energy points, and (7) number of magnetic sublevels. In the main body, the columns contain (from left to right): (1) electron energy [eV], (2) cross section [10^{-20} cm^2] (units are unstated, but 10^{-20} cm^2 is consistent with the documentation for other processes in the manual), and (3 - 6) parameters regarding the calculation and/or interpolation of the cross section. I am unsure of what these values are exactly. For more information, see the source code in 'interpolation.c.'

6.2. *Electron Impact Ionization*

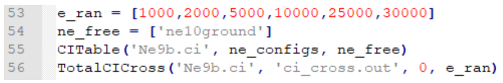
Electron impact ionization (EII) is the process in which an impacting electron e_i^- collides with an atom or ion A, and excites/removes an electron e_e^- from the atom or ion:

$$e_i^- + A \rightarrow A^+ + e_i^- + e_e^- \tag{6}$$

FAC's treatment of EII proceeds in a similar manner to the electron impact excitation calculation above. Details are available in the paper Gu (2008). To summarize, the impacting electron is represented by a continuum wave function. The liberation of an electron is handled by replacing a bound orbital in the final state(s) of the EIE calculation by the orbital of an unbound electron. Just as in EIE, the energy grid used in the EII calculation consists of 6 points constructed internally by FAC. This grid is based on the transition energies in the .tr file resulting from the structure calculation. In a similar manner to the SetCEGrid() function discussed in Sec. 6.1, the electron energy grid for collisional ionization e.g.

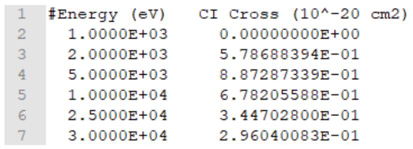
- <e_grid = [1,10,1000,5000]> # energies are listed in eV
- <SetCIGrid(e_grid)>

may be custom-defined. Cross sections at energies absent from the list of energies (either the default or user-constructed grid) are calculated via interpolation/extrapolation from the values stored in the energy grid.



```
53  e_ran = [1000,2000,5000,10000,25000,30000]
54  ne_free = ['ne10ground']
55  CITable('Ne9b.ci', ne_configs, ne_free)
56  TotalCICross('Ne9b.ci', 'ci_cross.out', 0, e_ran)
```

Figure 10. Sample script for calculation electron impact ionization cross sections (see text in Sec. 6.2).



```
#Energy (eV)    CI Cross (10^-20 cm2)
  1.0000E+03     0.00000000E+00
  2.0000E+03     5.78688394E-01
  5.0000E+03     8.87287339E-01
  1.0000E+04     6.78205588E-01
  2.5000E+04     3.44702800E-01
  3.0000E+04     2.96040083E-01
```

Figure 11. Output of the fac.TotalCICross() call (see text in Sec. 6.2).

Fig. 10 shows a sample script for calculating electron impact ionization cross sections of Ne^{9+}. Just an in EIE, this script must be implemented following the electronic structure calculation. The functions of this short script are discussed below. First, on line 55 the function

- <fac.CITable(fn, b, f)>

calculates the electron impact ionization rates between bound states b and the 'free' configurations f. The results of this calculation are saved in the file fn. For example, if we were studying the electron impact ionization of neutral helium, the 'free' states would be the ground state of He^+ and the bound state(s) would be the structure of neutral helium (He^0). In our case, the 'free' states f are the electronic states of the ion Ne^{10+}. We can calculate the cross sections for electron impact ionization of Ne^{9+} to Ne^{10+} via the following command

- <fac.CITable('Ne9b.ci', 'ne_configs', 'ne10ground')>

where arrays 'ne_configs' and 'ne10ground' are previously defined (see Fig. 1.)

After computing the necessary collisional ionization data and storing it in file 'Ne9b.ci', we use another FAC function to convert it to a more user-friendly format. The function 'TotalCICross()' will compute the collisional ionization cross section for a specified level. The function fac.TotalCICross(ifn, ofn, $ilev$, $energy$) takes in several familiar values. Parameters ifn and ofn signify the input ('Ne9b.ci') and output files (e.g. 'ci_cross.out', may be any name). The parameter ILEV is the numerical label of the level of interest. Parameter *energy* is an array of energy values at which the cross section is printed/extrapolated/interpolated. Using the same energy range as the EIE example (Fig. 8), the function call

- <fac.TotalCICross('Ne9b.ci', 'ci_cross.out', 0, e_ran)>

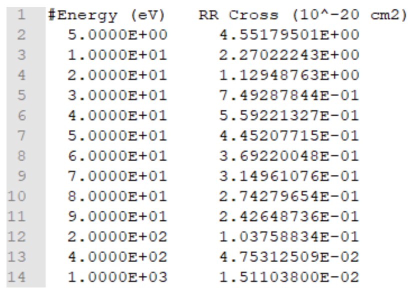
produces the two-column output file 'ci_cross.out' shown in Fig. 11. Here the energy [eV] and cross section [10^{-20} cm^2] are listed. Note that the EII calculation is significantly faster than the EIE calculations. Running the CITable() function for the electron impact ionization of the 1s level took my machine ~100 ms. The full calculation of *all* levels took only 1.44 seconds.

6.3. *Radiative Recombination and Photoionization*

We will now divert our attention to electron capture processes, e.g. Radiative Recombination (RR), sometimes called radiative electron capture. This process is expected to be absent in HCI-neutral collisions in e.g. COLTRIMS-type measurements. However, inside the trap region of an EBIT, it is expected that the plasma will contain many electron-ion collision processes, and processes such as RR will contribute to the observed x-ray spectra. In these collisions, it is possible a free electron is captured into an excited state of the ion. We may write this radiative recombination (RR) process as (Drake 2006):

$$e^-(E,l') + A^{q+}(c) \to A^{(q-1)+}(c,nl) + h\nu \quad (7)$$

where the electron carrying energy E and angular momentum l' is captured into an excited state (n,l) of the ion described by the electronic state c. The photon $h\nu$ carries away the excess energy. Note that in RR, the electron is captured directly to the final state, and the photon $h\nu$ carries away the energy $(E - |I_{nl}|)$ where I_{nl} is the binding energy of the final ion state (n,l). As the free electron energy is continuous, the spectra of pure RR will show a peak around the threshold energy $(E - |I_{nl}|)$ and a continuum blueward of the threshold energy.



```
#Energy (eV)   RR Cross (10^-20 cm2)
  5.0000E+00     4.55179501E+00
  1.0000E+01     2.27022243E+00
  2.0000E+01     1.12948763E+00
  3.0000E+01     7.49287844E-01
  4.0000E+01     5.59221327E-01
  5.0000E+01     4.45207715E-01
  6.0000E+01     3.69220048E-01
  7.0000E+01     3.14961076E-01
  8.0000E+01     2.74279654E-01
  9.0000E+01     2.42648736E-01
  2.0000E+02     1.03758834E-01
  4.0000E+02     4.75312509E-02
  1.0000E+03     1.51103800E-02
```

Figure 12. Sample of the fac.TotalRRCross() output file 'rr_cross.out' (see text in Sec. 6.3).

In a similar manner to both electron impact excitation and ionization, the RR cross sections are calculated in FAC by coupling continuum electrons to bound states of the ion. Just as in EIE and EII, the incident electron energy grid is based on the transition energies in the .tr file from the structure calculation. Alternatively, one can define a custom electron energy grid via

- <e_ran = [1,10,50,1000]> # energies are in units of eV
- <fac.SetPEGrid(e_ran)> # Note that the electron energy grid is a different function than EIE and EII

For this example, I have let FAC auto-generate the default grid. Using the $pfac$ interface, the RR rates may be calculated with a single function call:

- <fac.RRTable(fn, b, f, m)

The RRTable() function calculates the bound-free oscillator strengths between the bound groups b and free groups f. The results are saved to file fn, which follows similar guidelines for file name conventions discussed before (e.g. 'Ne9b.rr'). The last option m sets the multipole type, which is -1 (E1) by default. This parameter may be omitted from all function calls.

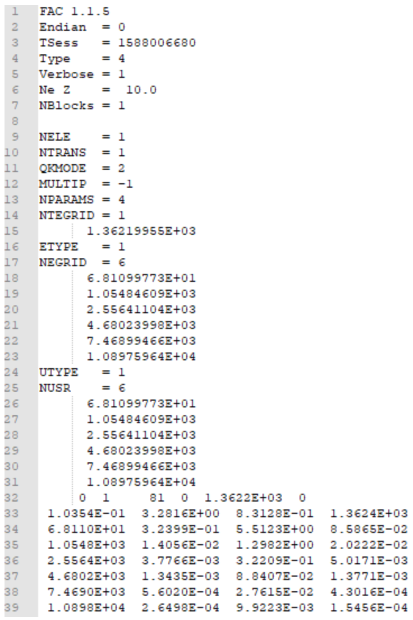
Consider a plasma of Ne inside an EBIT. Naturally, many charge states of Ne will be present. Let us calculate the RR cross sections for electron capture on Ne^{10+} which produces Ne^{9+}. Defining the bound configuration(s) b as the Ne^{9+} levels and the free configuration(s) f as the Ne^{10+} config, the following commands will calculate the RR rate(s) and save them in a more user-friendly format via a PrintTable() call:

- <ne_configs = ...> #Defined previously
- <ne_10 = ['ne10ground']>
- <fac.RRTable('Ne9b.rr', ne_9, ne_10)>
- <fac.PrintTable('Ne9b.rr', 'Ne9a.rr',1)>
- <fac.TotalRRCross('Ne9b.rr', 'rr_cross.out', 81, e_ran)>

First, the RRTable() call calculates all RR processes between the levels in variables ne_9 and ne_10. The PrintTable() call converts the binary file to ASCII format. Lastly, the TotalRRCross() call sums over all captures onto the Ne^{10+} level given by ILEV = 81 in the levels (.en) database file. The cross sections from the TotalRRCross() call are saved to file 'rr_cross.out' (shown in Fig. 12) at each energy in variable e_ran. However, it must be restated that the .rr file contains rates at only the default energies internally generated by FAC. Rates at energies not in this list are extrapolated/interpolated by the FAC. Depending on the range of energies needed, it may be required to set the energy grid manually via the SetPEGrid() function discussed at the beginning of this section. The total time to execute the above (at 6 points) was ∼100 ms. The bulk of this time is spent by the RRTable() call and appears almost unaffected by the number of Ne^{9+} configurations.

In the above, the RRTable() function also produces cross sections for the inverse process, photoionization (PI). We may write the PI process as

$$\gamma + A^q \rightarrow A^{(q-1)+} + e^- \tag{8}$$



```
FAC 1.1.5
Endian  = 0
TSess   = 1588006680
Type    = 4
Verbose = 1
Ne Z    =  10.0
NBlocks = 1

NELE    = 1
NTRANS  = 1
QKMODE  = 2
MULTIP  = -1
NPARAMS = 4
NTEGRID = 1
      1.36219955E+03
ETYPE   = 1
NEGRID  = 6
      6.81099773E+01
      1.05484609E+03
      2.55641104E+03
      4.68023998E+03
      7.46899466E+03
      1.08975964E+04
UTYPE   = 1
NUSR    = 6
      6.81099773E+01
      1.05484609E+03
      2.55641104E+03
      4.68023998E+03
      7.46899466E+03
      1.08975964E+04
     0  1     81  0  1.3622E+03  0
 1.0354E-01  3.2816E+00  8.3128E-01  1.3624E+03
 6.8110E+01  3.2399E-01  5.5123E+00  8.5865E-02
 1.0548E+03  1.4056E-02  1.2982E+00  2.0222E-02
 2.5564E+03  3.7766E-03  3.2209E-01  5.0171E-03
 4.6802E+03  1.3435E-03  8.8407E-02  1.3771E-03
 7.4690E+03  5.6020E-04  2.7615E-02  4.3016E-04
 1.0898E+04  2.6498E-04  9.9223E-03  1.5456E-04
```

Figure 13. Sample of the output file 'Ne9.rr' produced in Sec. 6.3.

where photon γ removes an electron from the ion. For multi-electron sytems, it may be possible that an electron is removed from an inner shell, in which case the ion is left in an excited state, $(A^{(q-1)+})^*$, which subsequently decays to produce one or more photon(s). In Fig. 13, the file 'Ne9a.rr' produced in the RR procedure above is shown. After the header, the data shown from left to right are: (1) electron grid energy, (2) radiative recombination cross section [10^{-20} cm^2], (3) photoionization cross section [10^{-20} cm^2], (4) gf value. Just as the function TotalRRCross() can produce cross sections in two-column format from the .rr file, the function fac.TotalPICross() performs an analogous procedure for photoionization. Running the code

- <TotalPICross('Ne9b.rr', 'pi_cross.out', 0, e_ran)>

will produce photoionization cross sections, as a function of energies *e_ran*, for the process Ne$^{9+}(1s) \rightarrow$ Ne^{10+}. Here the third variable '0' indicates the *initial* level being ionized. It must be stated that the energies in the .rr file are *electron* energies. The energies listed in the output of the TotalPICross() call are *photon* energies. Therefore, it is advised to pull energies/cross sections for specific levels via a TotalPICross() call rather than pulling directly from the .rr file.

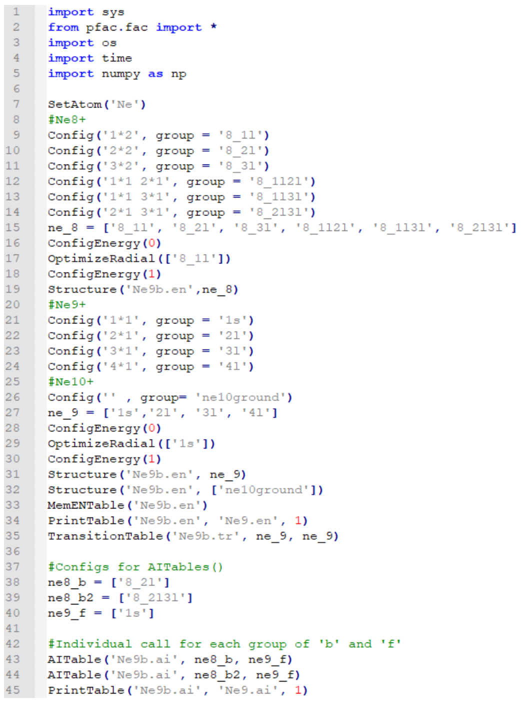
6.4. *Dielectronic Recombination and Autoionization*

Following electron capture by an ion, the electron capture (for non-bare ions) may proceed *dielectronically*. This dielectronic capture (DR) process is similar to RR but includes the simultaneous excitation of a bound electron. We may write this two-step resonant process as

$$e^- + A^{q+} \rightarrow A^{(q-1)+}(n'l'n''l'') \rightarrow A^{(q-1)+}(nl) + h\nu \tag{9}$$

Here the prime notation indicates the excited states. The photon $h\nu$ is sometimes called a *satellite* line as it is close in energy to the $(n'l') \rightarrow (nl)$ transition of the parent ion A^{q+}. The strength of each DR transition is sensitive to both temperature and density, and thus DR lines are commonly used as plasma diagnostics. Note that each DR resonance occurs at a specific energy, and may be revealed in e.g. heat maps of photon energy vs electron beam energy from EBIT x-ray measurements.

When discussing DR, x-ray notation is typically used, e.g. K($l = 1$), L ($l = 2$), M ($L = 3$), ... For example, the KLM DR resonance means the initial electron (K) is in the $n = 1$ state, the excited electron goes from $n = 1$ to $n = 2$ (L), and the captured electron is in an $n = 3$ (M) state. The emission line from this process would be a *satellite* line near the equivalent $n = 2$ to $n = 1$ line in the parent ion A^{q+}.



```
1   import sys
2   from pfac.fac import *
3   import os
4   import time
5   import numpy as np
6
7   SetAtom('Ne')
8   #Ne8+
9   Config('1*2', group = '8_1l')
10  Config('2*2', group = '8_2l')
11  Config('3*2', group = '8_3l')
12  Config('1*1 2*1', group = '8_1l2l')
13  Config('1*1 3*1', group = '8_1l3l')
14  Config('2*1 3*1', group = '8_2l3l')
15  ne_8 = ['8_1l', '8_2l', '8_3l', '8_1l2l', '8_1l3l', '8_2l3l']
16  ConfigEnergy(0)
17  OptimizeRadial(['8_1l'])
18  ConfigEnergy(1)
19  Structure('Ne9b.en',ne_8)
20  #Ne9+
21  Config('1*1', group = '1s')
22  Config('2*1', group = '2l')
23  Config('3*1', group = '3l')
24  Config('4*1', group = '4l')
25  #Ne10+
26  Config('' , group= 'ne10ground')
27  ne_9 = ['1s','2l', '3l', '4l']
28  ConfigEnergy(0)
29  OptimizeRadial(['1s'])
30  ConfigEnergy(1)
31  Structure('Ne9b.en', ne_9)
32  Structure('Ne9b.en', ['ne10ground'])
33  MemENTable('Ne9b.en')
34  PrintTable('Ne9b.en', 'Ne9.en', 1)
35  TransitionTable('Ne9b.tr', ne_9, ne_9)
36
37  #Configs for AITables()
38  ne8_b = ['8_2l']
39  ne8_b2 = ['8_2l3l']
40  ne9_f = ['1s']
41
42  #Individual call for each group of 'b' and 'f'
43  AITable('Ne9b.ai', ne8_b, ne9_f)
44  AITable('Ne9b.ai', ne8_b2, ne9_f)
45  PrintTable('Ne9b.ai', 'Ne9.ai', 1)
```

Figure 14. Sample script for calculating KLL and KLM dielectronic and autoionization rates for $\mathrm{Ne}^{9+} \rightarrow \mathrm{Ne}^{8+}$.

Autoionization (AI) is the inverse process of DR. Following the simultaneous inner-electron excitation and electron capture in DR, AI occurs when an electron is ejected from the ion:

$$e^- + A^{q+} \rightarrow A^{(q-1)+}(n'l'n''l'') \rightarrow A^{q+} + e^- \tag{10}$$

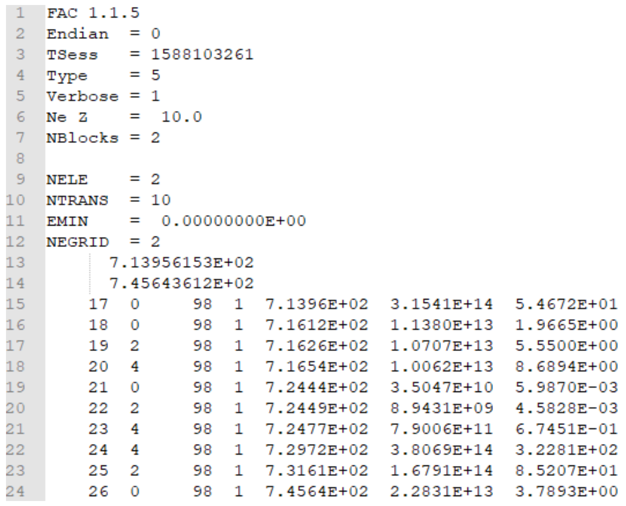
In the FAC, DR is calculated with the same function as AI. The function fac.AITable(fn, b, f) takes in several familiar functions. File fn is the output file, e.g. 'Ne9b.ai', and parameters b and f indicate the bound ('intermediate') levels and f indicates the free ('final') levels. In the AITable() call, the FAC calculates rates for both autoionization and dielectronic recombination. Just as in EIE, EII, and RR, the electron energy grid used internally by the FAC is generated (by default) as a 6-point grid depending on the energies in the .tr file. Alternatively, one can define a custom energy grid via the fac.SetPEGrid() call. For this example, we will use the default grid.

As an example, we will calculate AI and DR rates for electron capture on Ne^{9+} to form Ne^{8+}. As such, we will need to include the structure of Ne^{8+} in our calculation. We will include both KLL and KLM transitions, i.e. the DR intermediate states belonging to the $2l2l'$ and $2l3l'$ configurations following electron capture onto $\mathrm{Ne}^{9+}(1s)$. Our modified structure script, containing both the Ne^{8+} structure and the AI calculation, is shown in Fig. 14.

On lines 9 - 14, we define the following levels in Ne^{8+}: $1s^2$ (line 9), $2l2l'$ (line 10), $3l3l'$ (line 11), $1s2l$ (line 12), $1s3l$ (line 13), and $2l3l'$ (line 14). On line 15, we define all Ne^{8+} configurations in a single array. On line 16 - 18, we use the $1s^2$ configuration to optimize the central potential and calculate the electronic structure. The same process is repeated for Ne^{9+} on lines 21 - 35 for the configurations belonging to $\mathrm{Ne}^{9+}(nl)$ $(n, l = 1-4)$.

For the autoionization/DR resonance, the FACs calculation method is most reliable when the energy of the ejected electron varies little as a function of the internal (impacting) electron energy grid. As this grid will change for each resonance, the manual recommends a separate AITable() call for each resonance. On lines 38 - 40, we define 3 pairs of configurations: (1) the $2s2l$ configurations for the KLL resonances, (2) the $2l3l'$ configurations for the KLM resonances, and (3) the 'free', i.e. final (in the case of autoionization) configurations of Ne^{9+}. On lines 43 and 44, the AITable() calls calculate the DR and AI rates for the KLL (line 43) and KLM (line 44) resonances. Lastly, line 45 converts the rates from the binary file 'Ne9b.ai' to the verbose ASCII form in file 'Ne9.ai.'

The output file of this AI/DR calculation, 'Ne9.ai', is shown in Fig. 15. After the header, the data columns are: (1) bound state index (Ne^{8+} level index), (2) $2 \times J_{\mathrm{bound}}$, (3) free state index ($\mathrm{Ne}^{9+}$), (4) $2 \times J_{\mathrm{free}}$, (5) photon energy [eV], (6) AI rate [s^{-1}], (7) DC strength. The DC strength is a function of the DR rate A, and is written (in atomic units)



```
FAC 1.1.5
Endian  = 0
TSess   = 1588103261
Type    = 5
Verbose = 1
Ne Z    =  10.0
NBlocks = 2

NELE    = 2
NTRANS  = 10
EMIN    =  0.00000000E+00
NEGRID  = 2
      7.13956153E+02
      7.45643612E+02
    17  0     98  1  7.1396E+02  3.1541E+14  5.4672E+01
    18  0     98  1  7.1612E+02  1.1380E+13  1.9665E+00
    19  2     98  1  7.1626E+02  1.0707E+13  5.5500E+00
    20  4     98  1  7.1654E+02  1.0062E+13  8.6894E+00
    21  0     98  1  7.2444E+02  3.5047E+10  5.9870E-03
    22  2     98  1  7.2449E+02  8.9431E+09  4.5828E-03
    23  4     98  1  7.2477E+02  7.9006E+11  6.7451E-01
    24  4     98  1  7.2972E+02  3.8069E+14  3.2281E+02
    25  2     98  1  7.3161E+02  1.6791E+14  8.5207E+01
    26  0     98  1  7.4564E+02  2.2831E+13  3.7893E+00
```

Figure 15. Sample of the output file 'Ne9.ai' produced by script 'Ne9_dr_ai.py' in Fig. 14.

as (Gu 2008):

$$S_{\mathrm{DC}} = \frac{g_i}{2g_f}\frac{\pi^2}{E_{\mathrm{if}}}A^a \tag{11}$$

where g_i and g_f are the statistical weights of the initial and final states involved in the DR process and E_{if} is the DR resonance energy.

7. SIMULATING AN EBIT PLASMA WITH FAC'S *CRM*

7.1. *Atomic Data Generation*

We are finally equipped to calculate all necessary atomic data for a plasma of Ne$^{(8-10)+}$ inside an EBIT. In Fig. 16, the relevant processes for an EBIT plasma are shown. Presently, some of these processes cannot be included in the FAC CR model. However, I have supplied scripts for generating atomic data for all of these processes. Future versions of this guide will describe the inclusion of this data into an alterative CR model. The underpinnings of the scripts below are the electronic structure of the three ions. I have included the following electronic configurations:

- Ne^{10+}: Ground state (NULL).
- Ne^{9+}: (nl) with $n = 1 - 9$.
- Ne^{8+}: All singly-excited levels of the form $1snl$ $(n = 1-8)$ and doubly-excited levels of the form $2ln'l'$ $(n' = 2-4)$.

In any CR model, spectra are produced by assuming detailed balance between all collision processes. All excitations (either by CX, electron impact, or otherwise) must be balanced by de-excitation by spontaneous emission, electron impact deexcitation, or other processes; CR models then solve for the steady state solution of the level populations and use the various cross sections to produce synthetic spectra.

In FACs CR model, there are a few limitations. Each ion's collision data must be in separate files, e.g. Ne8.ce and Ne9.ce. Bound-free transitions such as radiative recombination and dielectronic recombination are included in the ionization balance, but bound-free transitions do not carry through to the final spectral outputs. I have been advised by Dr. Gu (FAC author) that not more than two charge states should be used in the CR model. I want to note again that the FAC *is* capable of calculating the ionization balance, but it is computationally expensive as all charge changing processes (DR, RR, etc.) need to be included. It is more appropriate in this context to calculate the spectra of individual charge states and then weigh them by an expected ionization balance. Lastly, charge exchange *can* be included, but the SetCXDist() function sets the ion energy distribution for *all* ions, which may not be the case inside an EBIT.

In the future, radiative recombination or dielectronic recombination spectra could be synthesized by convolving the cross sections (produced by RRTable() and AITable() scripts) with an electron energy distribution. This is left as

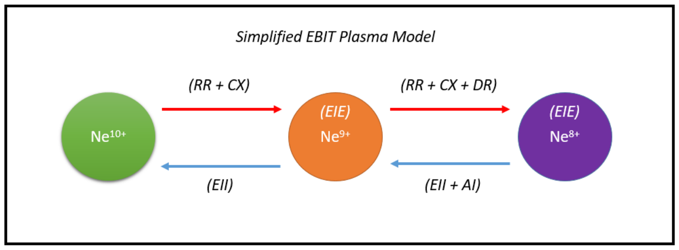


Figure 16. Summary of atomic processes inside an EBIT. Charge-changing processes (RR, CX, EII, AI, and DR) are included. Within the Ne^{8+} and Ne^{9+} systems, EIE is included.

an exercise to the reader at present. Future versions of this guide will feature sample scripts for generating RR/DR spectra.

The following scripts were used to produce atomic data for Ne^{10+}, Ne^{9+}, and Ne^{8+}. It must be stated that as the configuration group labels are not saved in the .en file, our definitions, e.g. $ne9 =$ ['1s', '2l', ...], do not carry over to other scripts if we simply import the energy level table with the MemENTable() function. Therefore, in each of the collision data scripts I have re-done the structure calculations so that I can use the group labels ($ne8 = [...], ne9 = [...]$, etc.) in other function calls. There is likely a solution to this problem, but I have not yet found it. However, the structure calculation only takes a few seconds in our case, and thus has little impact on the data generation process. The atomic data for the collisional processes in Fig. 16 (minus CX) were calculated with the following scripts. In each case, FAC was allowed to construct the default energy grids for the data calculation.

- *Ne8_struc.py/Ne9_struc.py*: Calculates the electronic structure of $Ne^{(8-10)+}$ and stores it in the file 'Neb.en.' Transition energies and A values are calculated and stored in the file 'Neb.tr.' The structure is stored in a single file to ensure that the level indexing is consistent in all collisional data files derived from these structures. In the other scripts below, the relevant structures were re-calculated to preserve the ability to reference groups of configurations with a single label, e.g. 'ne8 = [...].'

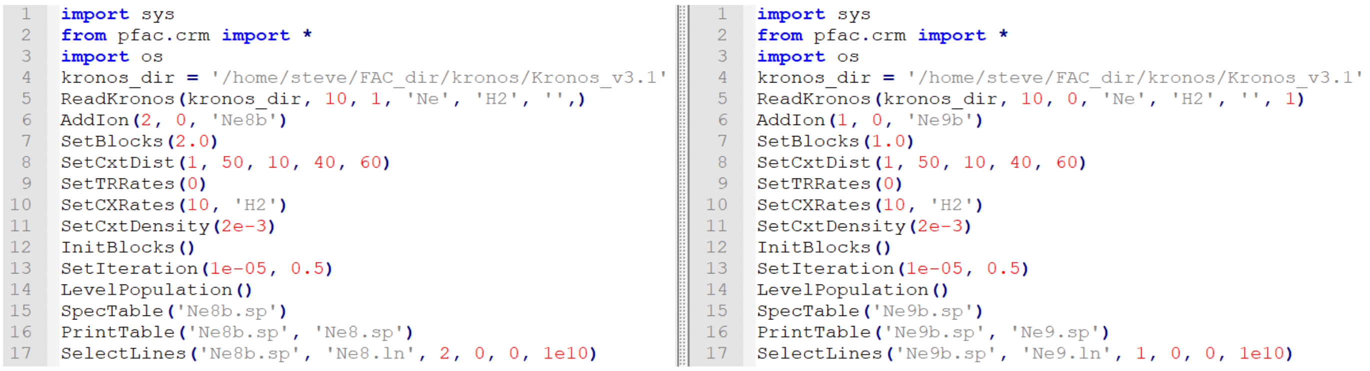
- *Ne8_rr.py/Ne9_rr.py*: Calculate all radiative recombination cross sections of the form $Ne^{10+} \rightarrow Ne^{9+}$ and $Ne^{9+} \rightarrow Ne^{8+}$. Results are saved in the files 'Ne8.rr' and 'Ne9.rr.'

- *Ne8_dr_ai.py*: Calculate all DR and AI rates for processes of the form $(1s) \rightarrow 2ln'l'(n' = 2-4)$. Each configuration group is calculated with a different AITable() call to allow the FAC to properly calculate an impacting electron energy grid for each resonance. Results are saved in the file 'Ne8.ai.' **Note: The .ai file is saved with the 'Ne8' prefix as the transitions, though produced from electron capture on Ne^{9+}, are occuring within the Ne^{8+} levels.**

- *Ne8_eii.py/Ne9_eii.py*: Calculate electron impact ionization cross sections for each energy level of Ne^{9+} and Ne^{8+}. Output from both processes is saved in 'Ne8.ci' and 'Ne9.ci.' For the $Ne^{8+} \rightarrow Ne^{9+}$ and $Ne^{9+} \rightarrow Ne^{10+}$ calculations, the CITable() commands executed in only a few seconds.

- *Ne8_eie.py/Ne9_eie.py*: Calculate electron impact excitation cross sections for each energy level of Ne^{9+} and Ne^{8+}. Output from in files marked by extensions '.ce.' User beware that the EIE calculations are not quick. For the Ne^{8+} and Ne^{9+} systems here, the calculations (with the default 6-point electron energy grid) took 19 and 4.5 hours respectively.

The scripts above calculate all of the relevant atomic data for the processes shown in Fig. 16. As stated above, FAC's *crm* module cannot produce synthetic spectra of bound-free transitions e.g. RR or DR. These spectra could be produced by convolving the calculated cross sections with the electron energy distribution. After producing synthetic spectra of bound-bound processes such as EIE or CX, the bound-free spectra (RR and DR) could be summed and later re-scaled to match EBIT spectra. Alternatively, one could export the FAC atomic data into some other collisional-radiative code such as NOMAD or Colradpy. In the following, I explore several scripts for producing and comparing CX to EIE spectra.



```
1  import sys
2  from pfac.crm import *
3  import os
4  kronos_dir = '/home/steve/FAC_dir/kronos/Kronos_v3.1'
5  ReadKronos(kronos_dir, 10, 1, 'Ne', 'H2', '',)
6  AddIon(2, 0, 'Ne8b')
7  SetBlocks(2.0)
8  SetCxtDist(1, 50, 10, 40, 60)
9  SetTRRates(0)
10 SetCXRates(10, 'H2')
11 SetCxtDensity(2e-3)
12 InitBlocks()
13 SetIteration(1e-05, 0.5)
14 LevelPopulation()
15 SpecTable('Ne8b.sp')
16 PrintTable('Ne8b.sp', 'Ne8.sp')
17 SelectLines('Ne8b.sp', 'Ne8.ln', 2, 0, 0, 1e10)
```

```
1  import sys
2  from pfac.crm import *
3  import os
4  kronos_dir = '/home/steve/FAC_dir/kronos/Kronos_v3.1'
5  ReadKronos(kronos_dir, 10, 0, 'Ne', 'H2', '', 1)
6  AddIon(1, 0, 'Ne9b')
7  SetBlocks(1.0)
8  SetCxtDist(1, 50, 10, 40, 60)
9  SetTRRates(0)
10 SetCXRates(10, 'H2')
11 SetCxtDensity(2e-3)
12 InitBlocks()
13 SetIteration(1e-05, 0.5)
14 LevelPopulation()
15 SpecTable('Ne9b.sp')
16 PrintTable('Ne9b.sp', 'Ne9.sp')
17 SelectLines('Ne9b.sp', 'Ne9.ln', 1, 0, 0, 1e10)
```

Figure 17. Simple scripts, 'Ne8_CX.py' and 'Ne9_CX.py', for generating synthetic CX spectra of the collisions $\mathrm{Ne}^{10+} + \mathrm{H}_2$ and $\mathrm{Ne}^{9+} + \mathrm{H}_2$. See text for details.

7.2. *A comparison of CX and EIE in an EBIT*

In the following, we have modified the CR script from Sec. 4 (Fig. 4). Synthetic CX spectra are produced for single electron capture in collisions of $\mathrm{Ne}^{10+} + \mathrm{H}_2$ and $\mathrm{Ne}^{9+} + \mathrm{H}_2$. Spectra are also produced for electron impact excitation of Ne^{9+} and Ne^{8+} and compared to the CX spectra.

Our first simple script(s), 'Ne9_CX.py' and 'Ne8_CX.py', are shown in Fig. 17. These scripts generate CX spectra following single electron capture between Ne^{10+} and Ne^{9+} and H_2. Note that prior to running each, the structure must be produced and saved in the appropriate files e.g. 'Ne9b.en' and 'Ne8b.en.' In each structure, the ground state of the parent ion must be specified. In Ne^{9+}'s structure calculation, the ground state of Ne^{10+} ('Config('', group = 'ne10ground')') is included, and in the Ne^{8+} structure script, the ground state of Ne^{9+} ('Config('1*1', group = '1l')') is included.

On line 4, the single charge exchange cross sections are read from the Kronos database for H-like (left) and bare (right) Ne. Note that for the bare ion, the parameter '1' selects the low energy l distribution. In the ReadKronos() call for H-like Ne, the l distribution is unspecified as the KRONOS cross sections are nlS-resolved. On lines 6 - 7, we've specified the ion(s) whose spectra are being generated and initiated the model. On line 8, a gaussian energy distribution centered at 50 eV (FWHM 10 eV) with a min/max energy of 40 and 60 eV respectively is specified. The charge exchange target density on line 11 is set at $2{\times}10^{-3}$ (in units of 10^{10} cm^{-3}), which is the neutral density at $P_{\mathrm{EBIT}} \sim 10^{-9}$ mbar. Lastly, lines 12 - 14 iterate the level populations, and lines 15 - 17 save the output of each ion into a human-readable format. The output of this CR model is shown in Fig. 18 where the intensities are normalized such that the strongest line in each charge state is set to unity.

Note that CX at very low ion-neutral collision energies is expected to yield different spectra than higher energy collisions downstream from CUEBIT. We have chosen a lower energy as it is more in-line with the expected ion energies inside an EBIT. We will now generate spectra from bound-bound electron-ion collisions, specifically those produced from electron impact excitation.

In Fig. 19, a script for generating synthetic spectra of electron impact excitation of Ne^{8+}, 'Ne8_eie_crm.py', is shown. A similar script may be produced for Ne^{9+} by simply adjusting the AddIon() call and modifying all file names accordingly. On line 6, the spectral model is initiated. On line 7, SetTRRates(0) loads the transition rates for all Ne^{8+} transitions from the .tr file. The SetCERates(1) call on line 8 loads in collisional excitation rates from the Ne^{8+} .ce file, and the '1' option includes collisional de-excitation. A value of '0' in this call sets only collisional excitation. On lines 9 - 11, the central and min/max of the electron energy distribution are defined and passed into the SetEleDist() function on line 12. A gaussian around 1000 eV was chosen here. On line 13, the electron density is set to 10^{12} cm^{-3}, which is typical for EBIT electron beams. Note that the input to the SetEleDensity() call is in units of 10^{10} cm^{-3}. Lastly, lines 14 - 19 iterate the CR model and save the Ne^{8+} emission lines to file 'Ne8.ln.'

The spectra produced by electron impact of Ne^{8+} are plotted alongside the CX spectra produced by single electron capture of Ne^{9+} from H_2 in Fig. 20. The CX and EIE spectra are normalized independently such that the strongest transition is set to unity. Aside from the peak at 920.9 eV, most transitions are (relatively) weaker in the EIE spectra compared to those produced by CX.

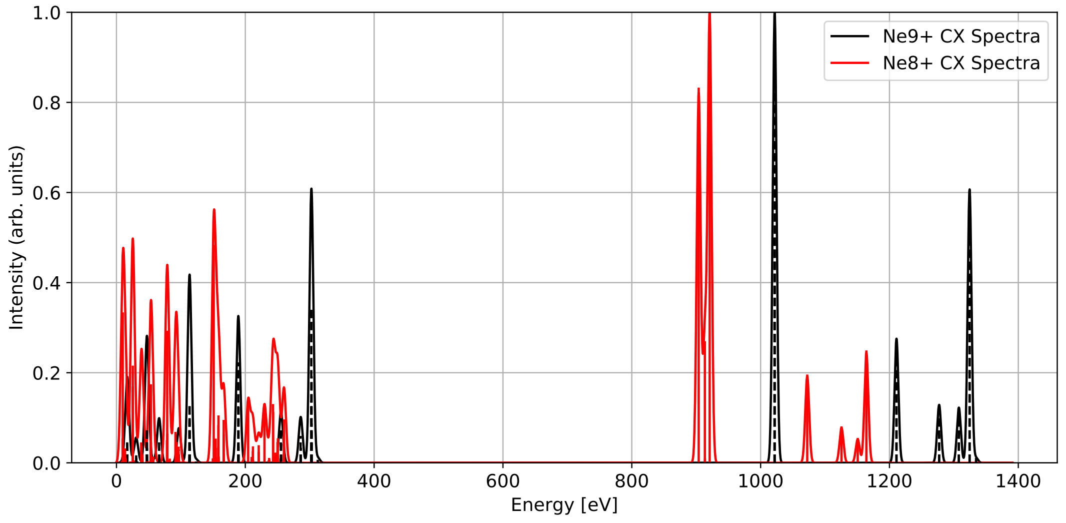


Figure 18. Outfit spectra of the scripts in Fig. 17 for collisions of highly charged Ne^{10+} + H_2 (black) and Ne^{9+} + H_2 (red). Each charge state is normalized such that the strongest transition is set to unity. Each transition is convolved with a gaussian (FWHM 6 eV). Summed spectra are shown as solid curves, and vertical lines indicate the position of strength of individual transitions.

```python
import sys
from pfac.crm import *
import os

AddIon(2, 1.0, 'Ne8b')
SetBlocks()
SetTRRates(0)
SetCERates(1) #1 includes collisional de-excitation
e_cen = 1000
e_min = e_cen - 300
e_max = e_cen + 300
SetEleDist(1,e_cen,40,e_min,e_max)
SetEleDensity(1e2) #in units of 10^10 per cm^3
InitBlocks()
SetIteration(1e-05, 0.5)
LevelPopulation()
SpecTable('Ne8b.sp')
PrintTable('Ne8b.sp', 'Ne8.sp')
SelectLines('Ne8b.sp', 'Ne8.ln', 2, 0, 0, 1e10)
```

Figure 19. Sample script for generating electron impact excitation spectra of Ne^{8+} using FAC's *crm* module. See text for details.

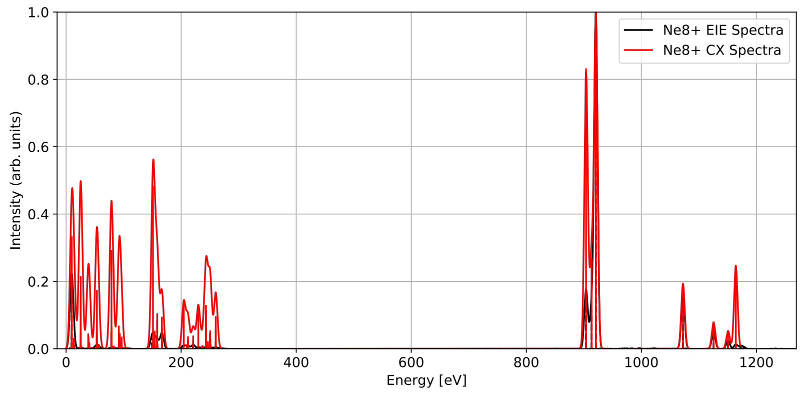


Figure 20. Spectra comparison of Ne^{8+}. CX spectra were produced by FAC's *crm* module for Ne^{9+} interacting with low-density H_2. Electron impact spectra were produced by assuming a gaussian electron beam at 1000 eV with an electron density of $n_e = 10^{12}$ cm^{-3}. Both CX and EIE spectra were convolved with a gaussian of FWHM 6 eV.

8. FUTURE IMPROVEMENTS

I plan to continue learning about the FAC in my spare time. At present, the most likely upcoming change to this document is a set of install intructions for MAC OS. I also plan to write scripts for generating RR / DR spectra from the .ai and .rr files produced by the FAC. Due to the current limitations of the FAC *crm* module, I will be looking into alternative CR codes for simulating EBIT spectra. The most likely candidate for this effort will be ColRadPy (Johnson, Loch, and Ennis 2019).

REFERENCES

Fac 1.1.15 Manual

.

P. D. Mullen, R. S. Cumbee, D. Lyons, and P. C. Stancil, The Astrophysical Journal Supplement Series **224**, 31 (2016)

.

R. K. Smith, A. R. Foster, R. J. Edgar, and N. S. Brickhouse, The Astrophysical Journal **787**, 77 (2014)

.

R. S. Cumbee, P. D. Mullen, D. Lyons, R. L. Shelton, M. Fogle, D. R. Schultz, and P. C. Stancil, The Astrophysical Journal **852**, 7 (2017)

.

M. F. Gu, Canadian Journal of Physics **86**, 675 (2008), https://doi.org/10.1139/p07-197

.

G. Drake, *Springer Handbook of Atomic, Molecular, and Optical Physics* (Springer Science+Business Media, 2006)

.

C. Johnson, S. Loch, and D. Ennis, Nuclear Materials and Energy , 100579 (2019)