

# libapecnei: the APEC Non-equilibrium Ionization Library

---

for version 0.2, 7 August 2015

Adam R. Foster ([afoster@cfa.harvard.edu](mailto:afoster@cfa.harvard.edu))

---

This manual is for the APEC NEI Library (libapecnei) (version 0.2, 7 August 2015),  
Copyright © 2013-2015 Smithsonian Institution.

Permission is granted to use, copy, modify, and distribute this software and its documentation for educational, research and non-profit purposes, without fee and without a signed licensing agreement, provided that this notice, including the following two paragraphs, appear in all copies, modifications and distributions. For commercial licensing, contact the Office of the Chief Information Officer, Smithsonian Institution, 380 Herndon Parkway, MRC 1010, Herndon, VA. 20170, 202-633-5256.

This software and accompanying documentation is supplied "as is" without warranty of any kind. The copyright holder and the Smithsonian Institution: (1) expressly disclaim any warranties, express or implied, including but not limited to any implied warranties of merchantability, fitness for a particular purpose, title or non-infringement; (2) do not assume any legal liability or responsibility for the accuracy, completeness, or usefulness of the software; (3) do not represent that use of the software would not infringe privately owned rights; (4) do not warrant that the software is error-free or will be maintained, supported, updated or enhanced; (5) will not be liable for any indirect, incidental, consequential special or punitive damages of any kind or nature, including but not limited to lost profits or loss of data, on any basis arising from contract, tort or otherwise, even if any of the parties has been warned of the possibility of such loss or damage.

# Table of Contents

<b>1</b>	<b>Background .....</b>	<b>1</b>
<b>2</b>	<b>Installation .....</b>	<b>2</b>
2.1	Make Check.....	2
<b>3</b>	<b>Using the Library .....</b>	<b>3</b>
3.1	Basic Usage.....	3
3.2	Intermediate Usage.....	3
3.3	Advanced Usage .....	3
3.4	Memory Efficient Usage .....	4
3.5	IDL Interface .....	4
3.6	Python Interface .....	5
<b>4</b>	<b>Library Contents .....</b>	<b>8</b>
4.1	Library Functions .....	8
4.1.1	load_ion_rec_data .....	8
4.1.2	calc_elem_ion_rec_rate.....	8
4.1.3	calc_elem_ionbal_nei.....	9
4.1.4	calc_elem_ionbal.....	10
4.1.5	calc_elem_nei.....	10
4.1.6	calc_timescale_nei .....	11
4.1.7	calc_multielem_nei .....	12
4.1.8	calc_all.....	13
	<b>Index .....</b>	<b>15</b>

# 1 Background

This library provides routines to calculate the ionization balance for a plasma in collisional ionization equilibrium (CIE) and in non-equilibrium ionization (NEI) state. As the latter is the more useful feature of the library, this is the name of library.

This library is part of the AtomDB project [www.atomdb.org](http://www.atomdb.org), and is based upon the Astrophysical Plasma Emission Code (APEC) and the ionization and recombination data in AtomDB.

This library solves the NEI plasma populations using the eigenvector method as outlined in *Smith, R.K. and Hughes, J.P., the Astrophysical Journal, Volume 718, Issue 1, pp. 583-585 (2010)*.

## 2 Installation

Code Prerequisites:

- `liblapack` Linear algebra maths library, required for matrix manipulation.
- `libblas` More linear algebra, required for matrix manipulation.
- `libcfitsio` Allows reading in of FITS files, i.e. the AtomDB database.
- The AtomDB database, from [www.atomdb.org](http://www.atomdb.org).
- The environment variable `ATOMDB` set to the directory containing the AtomDB database.

Beyond that, the standard `./configure; make; make install` procedure should work to install `libapecnei`. Note that both shared libraries and statically linked libraries will be generated.

### 2.1 Make Check

The `make check` command should run as you would expect, and hopefully passes all tests. Ensure that the `ATOMDB` environment variable is correctly set before running the tests otherwise they will fail.

## 3 Using the Library

`Libapecnei` is designed to solve the question “Given an initial ionization balance of  $X$ , in a plasma with electron temperature  $T_e$ , density  $N_e$ , what will the ionization balance be after time  $t$ ?”. Solving this requires loading the ionization and recombination data, finding the correct ionization recombination and ionization rates for the  $T_e$  in question, and then solving for the new values.

The full interface for each function in `libapecnei` is described in [Chapter 4 \[LibraryContents\]](#), [page 8](#). The library contains several overlapping functions, which give different levels of access to the calculations: basic use can be a single call to a wrapper program to get the ionization balance, while “power users” can play with individual routines to speed up the calculations which must be made frequently.

Remember that for the library to function, a copy of the AtomDB database must be installed and the environment variable `ATOMDB` must be set to point to it.

### 3.1 Basic Usage

The `calc_all` routine is the simplest access method. Use this when performance is not an issue. This code will load up the relevant atomic data, then calculate the relevant numbers and exit. As it reads in the atomic data every time, it is rather inefficient, but it is simple to code and it works.

Note that the initial ion population can be specified explicitly, or it can be set to the value at another temperature. So for example, in an ionizing plasma you could start with an ionization balance specified for cold electrons ( $T_e = 10^4\text{K}$ ), which would be largely at or near neutral for most elements, and then watch it ionize in a hotter electron plasma.

### 3.2 Intermediate Usage

If you wish to avoid reloading the basic ionization and recombination rate information repeatedly, call the following codes in order:

1. `load_ion_rec_data`: loads the basic ionization and recombination data for the specified elements. Call only once, store results.
2. ... and then, for each different case...
  1. (optional) `calc_elem_ionbal`: calculate steady state ionization balance at temperature  $T$ , perhaps to use as initial ionization balance
  2. `calc_multielem_nei`: calculate ionization balance in each case

### 3.3 Advanced Usage

If you really want to fiddle around under the hood, here is the full order in which the call to `calc_all` calls various subroutines. You can therefore recreate this yourself, modifying values as and where you need to. Possible situations would be if you have different ionization/recombination rates you wish to use, or if you wish to do many runs at the same temperature with different timescales and wish to avoid some computation overhead.

1. `load_ion_rec_data`: loads the basic ionization and recombination data for the specified elements.

2. `calc_elem_ionbal`: set initial ionization balance (if required)
3. `calc_multielem_nei`: which calls for each element:
  1. `calc_elem_nei`: wrapper to do the nei calculation for each ion, by...
  2. `calc_elem_ion_rec_rate`: calculate the ionization and recombination rates at electron temperature  $T_e$
  3. `calc_elem_ionbal_nei`: calculate the ionization balance for each ion at  $T_e = T_e, N_e = N_e, \text{time} = t$

### 3.4 Memory Efficient Usage

In particular for accessing the library multiple times using external routines, there are routines to read the raw ionization and recombination data in to a global variable, allowing repeated calculations of the ionization balance without rereading the files. This save both time (less file reading) and memory (not filling up memory with the same stuff over and over again). To do this, call the `apecneiInitialize` function, followed by `calc_all_external` as many times as is necessary. Note that is possible to make the calculations even more efficient, but that is probably unnecessary for the majority of uses (the code does not take long to run).

### 3.5 IDL Interface

There is a sample script in the `idl` directory, which can be used to call the library from IDL. Note that this has been tested and works on 32 bit IDL machines, and on 64 bit machines with GDL, but we have had no success with 64 bit IDL. If anyone can help with this, any help would be appreciated.

The IDL routine `call_libapecnei.pro` contains the interface to the library. So far it only interfaces with `calc_all`, via some wrapper routines. Finer grained interaction with the library is not available. There is a sample routine, `test.pro` in the `idl` folder which shows examples of calling the routine.

```

PRO call_libapecnei, filemap_file,      $
                                Te,      $
                                dens,      $
                                time,      $
                                list_Z,      $
                                init_all_Ion_pop, $
                                all_Ion_pop,      $
                                steady_state_init=steady_state_init,$
                                Te_init=Te_init
                                ;
; Program CALL_LIBAPECNEI.PRO
;
; This routine is designed for accessing libapecnei
; It takes an initial ionization balance for a list of elements and

```

```

; calculates the new ionization balance after time
;
; filemap_file =      filemap to use
; Te =              electron temperature (K)
; dens =            electron density (cm-3)
; time =            time (s)
; list_Z =          list of elements to do calculation for, given by
;                  nuclear charge Z
; init_all_Ion_pop = 2D double array, first index is the element index
;                  in list_Z, second index is the ion stage
;                  (0=neutral)
; all_Ion_pop =      returned ion population after time t
;
; KEYWORDS
; steady_state_init = if you want to start with a CIE plasma at a
;                  different temperature, set this keyword. It will
;                  overwrite "init_all_Ion_pop" with the initial
;                  ionization balance and the code will run from
;                  there
; Te_init =          if using steady_state_init, this should be set to
;                  the initial ionization balance temperature, in K.
;
; This code then converts these inputs to be of the correct form for
; wrap_libapecnei, which calls the C code.
;

```

### 3.6 Python Interface

There is a sample python script in the `python` directory, which can be used to call the library from python using the `ctypes` interface. This has been tested with python 2.7 only. We'll get to python 3 one day.

The interface consists of two important routines: `initialize`, which reads in the data to memory, and then `make_ionbal_ctypes` which calculates the ionization balance based upon this.

```

def initialize(libapecnei, filemap_file, list_Z):
    Initialization routine for libapecnei. Call this once (and only once)
    to read all the ionization and recombination data in to memory.
    Once this is done, subsequent calls to make_ionbal_ctypes will
    not require re-reading the data files, making the code faster and
    preventing excessive memory use

```



```

INPUTS
libapecnei    : the shared object library, as opened with
                ctypes.cdll.LoadLibrary
filemap_file  : the filemap, usually in $ATOMDB/filemap
list_Z        : list of elements to load the data for

Version 0.1 Jun 5th 2015
Adam Foster
"""
def make_ionbal_ctypes(libapecnei, \
                        kT, \
                        time_in, \
                        Z_list, \
                        init_ion_pop=False, \
                        kTin=False, \
                        units="K", \
                        force_extrap=False):

    wrapper for C library libapecnei.
    remember to change the location of libapecnei in the source code
    inputs:
    kT (float)          electron temperature. K by default or keV
                        (see "units")
    time_in (float)     Ne * time, in cm-3 s
    Z_list (list, integer) list of the elements to use,
                        e.g. [1,2,6] = H, He, C
    init_ion_pop (dict, int, float list)
        e.g.
        init_ion_pop={}
        init_ion_pop[1]=[1.0, 0.0]
        init_ion_pop[2]=[1.0, 0.0, 0.0]
        init_ion_pop[6]=[1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
        would give you neutral H, He and C
    kTin (float)        use a temperature to specify the initial ion
                        population instead of init_ion_pop
    units              by default K, if "keV" then kT & kTin should be
                        in keV

```

Note that one of `init_ion_pop` or `kTin` must be provided.

Returns:

ion population, normalized to 1 for each element. structure is the same as `init_ion_pop` above

## 4 Library Contents

The library contains many functions, as well as several fuctions which call many of the other functions sequentially. This enables either fine grained and more computationally efficient access for high power use, or for the user to simply ask for the ionization balance in a single call.

### 4.1 Library Functions

#### 4.1.1 load\_ion\_rec\_data

```
ElementP *load_ion_rec_data(int N_Elements,
                           int *list_Z,
                           char *filemap_file);

/* This function loads in the initial ionization and recombination data
 * for the elements, given by their nuclear charge Z.
 *
 * INPUTS:
 * N_Elements:      number of elements to be used
 * list_Z:          array of element nuclear charges of interest
 * filemap_file:    location of filemap listing where are the data
 *                  are kept.
 *
 * RETURNS:
 * All of the raw atomic data from the AtomDB FITS files for all the
 * elements in list_Z, in an ElementP structure
 *
 * NOTE: the ATOMDB environment variable must be set, and pointing to
 *       the directory which has the APED subdirectory in it.
 */
```

#### 4.1.2 calc\_elem\_ion\_rec\_rate

```
void calc_elem_ion_rec_rate(struct ELEMENT_ION_STATE *state,
                           double Te,
                           double Ne,
                           double *ioniz,
                           double *recomb,
                           struct EMISSION_LINES **ionrec_line);

/* This routine calculates the ionization and recombination rates
 * for 1 element only. It assumes to have been passed in the correct
```

```

* "state"
*
* INPUTS:
* state:          raw ionization and recombination data from the
*                  AtomDB database for this element
* Te:             Electron temperature (K)
* Ne:             Electron density (cm-3)
*
* OUTPUTS:
* ioniz:          array of ionization rates
* recomb:         array of recombination rates
* ionrec_line:    this is included only to make other routines run.
*                  It will not return useful data and should be
*                  ignored.
*/

```

#### 4.1.3 calc\_elem\_ionbal\_nei

```

void calc_elem_ionbal_nei(double *ioniz,
                        double *recomb,
                        double *init_Ion_pop,
                        double Te,
                        double Ne,
                        double time,
                        int Z,
                        double *Ion_pop);

/* This routine calculates the ionization balance for all the ions
* of an element.
* INPUTS:
* ioniz:          array of ionization rates
* recomb:         array of recombination rates
* init_Ion_pop:   input ionization balance (normalized to 1).
* Te:            electron temperature (K)
* Ne:            electron density (cm-3)
* time:          time step (s)
* Z:            element atomic number
*
* OUTPUTS
* Ion_pop:       ionization balance after time (normalized to 1).
*/

```

#### 4.1.4 calc\_elem\_ionbal

```
void calc_elem_ionbal(struct ELEMENT_ION_STATE *state,
                    double Te,
                    double Ne,
                    int Z,
                    double Ion_pop[]));
/* this routine calculates the steady-state ionization balance
 * INPUTS:
 * state:          raw ionization and recombination data from
 *                  AtomDB database for this element
 * Te:             Electron temperature (K)
 * Ne:             Electron density (cm-3)
 * Z:              element atomic number
 *
 * OUTPUTS
 * Ion_pop:        steady state ionization balance (normalized to 1).
 */
```

#### 4.1.5 calc\_elem\_nei

```
void calc_elem_nei(struct ELEMENT_ION_STATE *state,
                  double Te,
                  double Ne,
                  double time,
                  int Z,
                  double *init_Ion_pop,
                  double *Ion_pop);
/* This routine takes a state object (i.e. all the atomic data for
 * an element), calculates the ionization & recombination rates at
 * Te, Ne, and then calculates the evolution of init_Ion_pop after
 * a specified time. This is expected to be the most heavily used call
 * in the library
 *
 * INPUTS:
 * state:          raw ionization and recombination data from
 *                  AtomDB database for this element
 * Te:             Electron temperature (K)
 * Ne:             Electron density (cm-3)
 * time:           time step (s)
 * Z:              element atomic number
```

```

* init_Ion_pop:      input ionization balance (normalized to 1).
*
* OUTPUTS:
* Ion_pop:           ionization balance after time (normalized to 1).
*/

```

#### 4.1.6 calc\_timescale\_nei

```

void calc_timescale_nei(ElementP *Atomic_params,
                        double Te,
                        double Ne,
                        int N_Elements,
                        int *list_Z,
                        double *mineig,
                        double *maxeig);

/* This routine calculates the suitable timescales for NEI
 * calculations, using the Smith & Hughes 2010 paper.
 * (ApJ 718:583, 2010)
 *
 * It goes through all the elements in list_Z, and calculates
 * the minimum and maximum eigenvalues in units of cm-3 s.
 * It return the lowest of the low and the highest of the high
 * in mineig and maxeig respectively. Your timestep for an
 * NEI calculation should be somewhere in between those 2 values.
 *
 * INPUTS:
 * Atomic_params:    the full set of atomic data for all elements
                    from load_ion_rec_data
 * Te:               Electron temperature (K)
 * Ne:               Electron density (cm-3)
 * N_Elements:       number of elements to be used
 * list_Z:           array of element nuclear charges of interest
 *
 * OUTPUTS:
 * mineig:           minimum Eigenvalue (cm-3 s) for all the elements
 * maxeig:           maximum Eigenvalue (cm-3 s) for all the elements
 */

```

### 4.1.7 calc\_multielem\_nei

```

void calc_multielem_nei(ElementP *Atomic_params,
                        double Te,
                        double Ne,
                        double time,
                        int N_Elements,
                        int *list_Z,
                        double *init_all_Ion_pop,
                        double *all_Ion_pop);

/* This routine calculates the new ionization population for a list
 * of elements at t=time. Note that the all_init_ion_pop and all_ion_pop
 * arrays are the populations for ALL the ions, in a row.
 *
 * E.g. if you have hydrogen, helium and carbon, then the elements of
 * all_init_ion_pop are:
 *
 * all_init_ion_pop[0] = H+0
 * all_init_ion_pop[1] = H+1
 * all_init_ion_pop[2] = He+0
 * all_init_ion_pop[3] = He+1
 * all_init_ion_pop[4] = He+2
 * all_init_ion_pop[5] = C+0
 * all_init_ion_pop[6] = C+1
 * all_init_ion_pop[7] = C+2
 * all_init_ion_pop[8] = C+3
 * all_init_ion_pop[9] = C+4
 * all_init_ion_pop[10] = C+5
 * all_init_ion_pop[11] = C+6
 *
 * The same applies for the return value.
 *
 * INPUTS:
 * Atomic_params:    the full set of atomic data for all elements
 *                   from load_ion_rec_data
 * Te:               Electron temperature (K)
 * Ne:               Electron density (cm-3)
 * time:             time step (s)

```

```

* N_Elements:      number of elements to be used
* list_Z:          array of element nuclear charges of interest
* all_init_ion_pop: array of initial ion populations
*
* OUTPUTS:
* all_ion_pop:      array of final ion populations
*
*/

```

#### 4.1.8 calc\_all

```

void calc_all(char *filemap_file,
              double Te,
              double Ne,
              double time,
              int N_Elements,
              int *list_Z,
              double *init_all_Ion_pop,
              double *all_Ion_pop,
              int steady_state_init,
              double Te_init);

/* Super wrapper routine: can use just the routine for everything
 * if performance is not an issue. It will read in the data,
 * calculate the initial ionization balance if required,
 * calculate the final ionization balance, and return this.
 * None of the intermediate steps are committed to memory so this
 * is slow if you need to do these calculations a lot (e.g.
 * parallel MHD code, etc)
 *
 *
 * INPUTS:
 * filemap_file:      location of filemap listing where are the data
 *                    are kept.
 * Te:                Electron temperature (K)
 * Ne:                Electron density (cm-3)
 * time:              time step (s)
 * N_Elements:        number of elements to be used
 * list_Z:            array of element nuclear charges of interest
 * init_all_Ion_pop:  initial ionization population
 * steady_state_init: If true, populate initial_all_Ion_pop with

```



```
*          an equilibrium ionization balance at Te=Te_init
* Te_init:    initial electron temperature (K), used if
*             steady_state_init is true.
*
*
* OUTPUTS:
* all_ion_pop: array of final ion populations
*
*/
```

# Index

## A

Advanced Usage ..... 3

## B

background ..... 1

Basic Usage ..... 3

## C

calc\_all ..... 13

calc\_elem\_ion\_rec\_rate ..... 8

calc\_elem\_ionbal ..... 10

calc\_elem\_ionbal\_nei ..... 9

calc\_elem\_nei ..... 10

calc\_multielem\_nei ..... 12

calc\_timescale\_nei ..... 11

## I

IDL Interface ..... 4

Intermediate Usage ..... 3

## L

Library Contents ..... 8

load\_ion\_rec\_data ..... 8

## M

Memory Efficient Usage ..... 4

## P

Python Interface ..... 5

## U

Using the Library ..... 3