```
import numpy as np
import os
import copy

from constants_lineRT import *

class LineData:

    def __init__(self, species, debug=False):

        filename = self.get_filename(species)
        mu, num_lvls, E,g,freq, A,B,C_all, num_partners, temps_all =
         self._read_data(filename, debug)

        self.mu = mu                        # molecular weight
        self.num_lvls = num_lvls            # number of energy levels
        self.E = E                          # energy of each level [eV]
        self.g = g                          # weight of each level
        self.freq = freq                    # Frequency freq[i][j] of
         transition i<->j [Hz]
        self.A = A                          # Einstein A-coeff A[i][j] for
         transition i->j [1/s]
        self.B = B                          # Einstein B-coeff B[i][j] for
         transition i->j [m2/(eV*s)]
        self.C_all = C_all                  # collision coeffs C_all[p][T] for
         each coll partner [m3/s]
        self.num_partners = num_partners    # number of collision partners in
         the data file
        self.temps_all = temps_all          # T at which C is given for each
         coll partner [K]

    def get_filename(self, species):

        # filename is already given
        if (species[-4:]=='.dat') or (species[-4:]=='.txt'):
            return species

        # molecule is chosen
        THIS_FOLDER = os.path.dirname(os.path.abspath(__file__))
        database = os.path.join(THIS_FOLDER, 'LAMDA')
        if species=='HCO+':
            filename = os.path.join(database, 'HCO+.dat')
        elif species=='H13CO+':
            filename = os.path.join(database, 'H13CO+.dat')
        elif species=='N2H+':
            filename = os.path.join(database, 'N2H+.dat')
        elif species=='SiO':
            filename = os.path.join(database, 'SiO.dat')
        elif species=='HNC':
            filename = os.path.join(database, 'HNC.dat')
        elif species=='HCN':
            filename = os.path.join(database, 'HCN.dat')
        elif species=='CO':
            filename = os.path.join(database, 'CO.dat')
        else:
```

```
            print 'Unknow species. Chose from HCO+, H13CO+, N2H+, SiO, HNC,
             HCN, CO'
            print 'or provide a LAMDA datafile.'
            exit()
        return filename

    ''' read LAMBA data file with atom/molecule info
        transistions between i----
                             j----
        PARAMS:
          filename = name of the LAMBDA file
        RETURNS:
          all fields listed in __init__ as single values or np.arrays
        '''
    def _read_data(self, filename, debug):
        f = open(filename,'r')
        f.readline()
        species = f.readline()
        print 'Reading data for', species
        # molecular weight
        f.readline()
        mu = float(f.readline())
        # number of energy levels
        f.readline()
        num_lvls = int(f.readline())
        print 'number of energy levels', num_lvls
        # read energy levels: energy E, statistical weight g
        f.readline()
        E = []
        g = []
        for l in range(num_lvls):
            words =  f.readline().split()
            E.append(float(words[1]) * 100.*c_si*h_ev) # cm^-1 -> eV
            g.append(float(words[2]))
            if debug:
                print words[1], words[2]
            else:
                print "energy",E[l], "g", g[l]

        # number of radiative transistions
        f.readline()
        num_trans = int(f.readline())
        print 'number of radiative transitions', num_trans
        # read transistions: upper lvl, lower lvl, A-coefficient, frequency
        f.readline()
        A = np.zeros((num_lvls,num_lvls))
        freq = np.zeros((num_lvls,num_lvls))
        for t in range(num_trans):
            words = f.readline().split()
            i = int(words[1]) - 1
            j = int(words[2]) - 1
            A[i][j] = float(words[3]) # s^-1
            freq[i][j] = float(words[4]) * 1e9 # GHz -> Hz
            freq[j][i] = freq[i][j]
            if debug:
```

```
            print words[1], words[2], words[3], words[4]
        else:
            print 'up', i, 'low', j, 'A', A[i][j], 'freq', freq[i][j]

    # compute B-coefficient via Einstein relations
    # Bij = coeff for stimulated emission, Bji = coeff for extinction
     (j<i)
    B = np.zeros((num_lvls,num_lvls))
    for i in range(0,num_lvls):
        for j in range(0,i):
            if A[i][j] != 0:
                B[i][j] = A[i][j] * (c_si**2) / (2*h_ev * (freq[i]
                 [j])**3) # m2/(eV*s)
                B[j][i] = B[i][j] * g[i]/g[j]
                if debug:
                    print 'up', i, 'low', j, 'B_ij', B[i][j], 'B_ji',
                     B[j][i]

    # number of collision partners in the data file
    f.readline()
    num_partners = int(f.readline())
    print 'number of collision partners', num_partners
    C_all = []
    temps_all = []
    for partner in range(num_partners):
        # reference
        f.readline()
        line = f.readline()
        print 'data for collisions:', line[2:-1]
        # number of collisional transitions
        f.readline()
        num_collis = int(f.readline())
        print 'number of collisional transitions', num_collis

        # number of temperatures in the table
        f.readline()
        num_temps = int(f.readline())
        print "number of temperatures", num_temps
        # read the temperature values
        f.readline()
        words = f.readline().split()
        if debug:
            print words
        temps = np.zeros(num_temps)
        for t in range(num_temps):
            temps[t] = float(words[t])
        temps_all.append(temps) #K

        # read collision coeff data: upper lvl, lower lvl, C-coefficient
         for each temp
        C = np.zeros((num_temps,num_lvls,num_lvls))
        f.readline()
        for col in range(num_collis):
            words = f.readline().split()
            i = int(words[1]) - 1
```

```
            j = int(words[2]) - 1
            for t in range(num_temps):
                C[t][i][j] = float(words[3+t]) * 1.e-6 # cm3/s -> m3/s
            if debug:
                print words[1], words[2], C[:,i,j]

        # calculate the inverse coefficient via LTE relation
        for i in range(num_lvls):
            for j in range(i):
                for t in range(num_temps):
                    if C[t][i][j] != 0:
                        C[t][j][i] = C[t][i][j] * np.exp(-(E[i]-E[j])/
                         (kb_ev*temps[t]))*g[i]/g[j]

        # add collision partner data to global array
        C_all.append(C)

    f.close()
    C_all = np.array(C_all)
    temps_all = np.array(temps_all)
    return mu, num_lvls, np.array(E), np.array(g), freq, A, B, C_all,
     num_partners, temps_all

''' Calculate net collision coeff for a gas consisting of different
 components at temp T
    Interpolates table betweem T values
    PARAMS:
      T = temperature (K)
      comp_fractions = fraction of the total density in each component
    RETRUN:
      C = netto collision coeff C[i][j] (m3/s) '''
def calc_total_C(self, T, comp_fractions, debug=False):

    if debug:
        print 'Calculating C_ij for temperature', T, 'and component
         mixture', comp_fractions
    if len(comp_fractions) != self.num_partners:
        print 'ERROR: lenght comp_fractions does not match number of
         partners in data file!'
        exit()

    C = np.zeros((self.num_lvls,self.num_lvls))
    for p in range(self.num_partners):
        max_index = len(self.temps_all[p])
        if T <= self.temps_all[p][0]: # T is lower than lowest value in
         table
            if debug:
                print 'Component',p, ': using C coeff at temperature',
                 self.temps_all[p][0]
            for i in range(self.num_lvls):
                for j in range(self.num_lvls):
                    C[i][j] = C[i][j] + comp_fractions[p] *
                     self.C_all[p][0][i][j]
        elif T >= self.temps_all[p][max_index-1]: # T is higher than
         highest value in table
```

```
                if debug:
                    print 'Component',p, ': using C coeff at temperature',\
                           self.temps_all[p][max_index-1]
                for i in range(self.num_lvls):
                    for j in range(self.num_lvls):
                        C[i][j] = C[i][j] + comp_fractions[p] *
                         self.C_all[p][max_index-1][i][j]
            else: # determine temperature entries needed to interpolate
                t = 1 # T index of upper limit
                while self.temps_all[p][t] < T:
                    t = t+1
                t_frac = (self.temps_all[p][t] - T)/(self.temps_all[p][t] -
                 self.temps_all[p][t-1])
                if debug:
                    print 'Component', p, ': interpolating between
                     temperatures',\
                        self.temps_all[p][t-1], 'and', self.temps_all[p][t],
                         '(t_frac=', t_frac, ')'
                for i in range(self.num_lvls):
                    for j in range(self.num_lvls):
                        interpol = (1-t_frac) * self.C_all[p][t][i][j] + \
                                        t_frac * self.C_all[p][t-1][i][j]
                        C[i][j] = C[i][j] + comp_fractions[p] * interpol
                        if debug:
                            print self.C_all[p][t-1][i][j], self.C_all[p][t]
                             [i][j], '->', C[i][j],\
                                  '(t_frac=', t_frac, ')'
        return C

     #-------------------------------------------------------------------------
     ------------------------

    ''' Reduce the number of levels to take into account based on an LTE
     estimate '''
    def reduce_linedata(self, new_num_lvls):
        ld = copy.deepcopy(self)
        ld.num_lvls = new_num_lvls
        ld.E = ld.E[:new_num_lvls]
        ld.g = ld.g[:new_num_lvls]
        ld.freq = ld.freq[:new_num_lvls,:new_num_lvls]
        ld.A = ld.A[:new_num_lvls,:new_num_lvls]
        ld.B = ld.B[:new_num_lvls,:new_num_lvls]
        ld.C_all = ld.C_all[:,:,:new_num_lvls,:new_num_lvls]
        return ld


#-----------------------------------------------------------------------------
 ------------------------
def test_hco():
    line = LineData('HCO.txt', debug=True)
    line.calc_total_C(12.5, [1.], debug=True)
    line.calc_total_C(125.0, [1.], debug=False)
    line.calc_total_C(2000., [1.], debug=False)
    line.calc_total_C(2500., [1.], debug=False)
    line.calc_total_C(5., [1.], debug=False)
```

```
    line.calc_total_C(10., [1.], debug=False)

def test_co():
    line = LineData('CO.txt', debug=True)
    print line.calc_total_C(10., [.25,0.75], debug=False)

if __name__=='__main__':
    #test_hco()
    test_co()
```