```
import numpy as np
from constants_lineRT import *
from scipy.special import erf

#------background
 spectra-------------------------------------------------------------------
 -------
''' Planck spectrum for a black body
    PARAMS:
      nu = freq in Hz
      T = temperature in K
    RETURN:
      spectral radiance in W/sr/m2/Hz '''
def B_nu_si(nu, T):
    x = h_si*nu/(kb_si*T)
    # units J/Hz * Hz3 * s2/m2 = J/Hz * Hz * m-2 = J/s /Hz /m2 = W/(m2*Hz)
    return 2.0*h_si*(nu**3)/(c**2) / (np.exp(x)-1.0)

def B_nu_ev(nu, T):
    if nu==0.:
        return 0.
    else:
        x = h_si*nu/(kb_si*T)
        # units eV*s * Hz3 / (m2/s2) = eV * s3 * Hz3 * m-2 = eV/s/Hz/m2
        return 2.0*h_ev*(nu**3)/(c**2) / (np.exp(x)-1.0)

#-----
 Definitions---------------------------------------------------------------
 --------------------
''' specific emissivity of the line i-j at frequency nu in W/m3/sr/Hz
    phi = a function with the line profile '''
def emissivity(nu, nu_ij, x_i, n_molec, A_ij, T, molecMass, phi, v_los=0.):
    # units: J*s * Hz * sr-1 * m-3 * s-1 * Hz-1 = J/s/Hz/m3/sr = W/(sr*m3)
    return h_si*nu_ij/(4*np.pi) * x_i * n_molec * A_ij * phi(nu, nu_ij, T,
     molecMass, v_los=v_los)

''' emissivity of the line i-j in the frequency range [nu_a,nu_b] in W/m3/
 sr/Hz
    integr_phi = a function with the integral over the line profile in some
     range '''
def emissivity_part(nu_a, nu_b, nu_ij, x_i, n_molec, A_ij, T, molecMass,
 integr_phi, v_los=0.):
    # units: J*s * Hz * sr-1 * m-3 * s-1 = eV/s/m3/sr
    therm_a = np.sqrt(2.*kb_si*T/(molecMass*MH))
    total = h_si*nu_ij/(4*np.pi) * x_i * n_molec * A_ij\
           * integr_phi(nu_a, nu_b, nu_ij, T, molecMass, v_los=v_los)
    return total / (nu_b - nu_a)

''' total emissivity of transition i-j in W/m3/sr
    J = integral j dv = integral cst phi(v) dv = cst integral phi dv = cst *
     1
    Remark that v_ij is constant! '''
def integrated_emissivity(nu_ij, x_i, n_molec, A_ij):
    # units: J*s * Hz * sr-1 * m-3 * s-1 = J/s/m3/sr
    return h_si*nu_ij/(4*np.pi) * x_i * n_molec * A_ij
```

```
''' specific extinction of the line i-j at frequency nu in 1/m/sr/Hz/s
    B in m2/(eV*s) '''
def extinction(nu, nu_ij, x_i, x_j, n_molec, B_ij, B_ji, T, molecMass, phi,
 v_los=None):
    # units: eV*s * Hz * sr-1 * m2/(eV*s) * m-3 * Hz-1 = 1/(sr*m*s*Hz)
    a = h_ev*nu_ij/(4*np.pi) * (x_j*B_ji - x_i*B_ij) * n_molec\
        * phi(nu, nu_ij, T, molecMass, v_los=v_los)
    if a<0.:
        print 'WARNING: negative extinction!'
    return a

''' total extinction of transition i-j in 1/m/sr/s
    B in m2/(eV*s) '''
def integrated_extinction(nu_ij, x_i, x_j, n_molec, B_ij, B_ji):
    # units: eV*s * Hz * sr-1 * m2/(eV*s) * m-3 = 1/m/sr/s
    return h_ev*nu_ij/(4*np.pi) * (x_j*B_ji - x_i*B_ij) * n_molec

''' Source function of transision i-j for optically thick media S=j/a in eV/
 s/Hz/m2 '''
def source_function_thick(nu_ij, x_i, x_j, n_molec, A_ij, B_ij, B_ji):
    j = integrated_emissivity(nu_ij, x_i, n_molec, A_ij) / eV
    a = integrated_extinction(nu_ij, x_i, x_j, n_molec, B_ij, B_ji)
    # units: (eV/s/m3/sr) / (1/m/sr/s) = eV/m2 = eV/s/Hz/m2
    return j/a

#------Line
 profiles-------------------------------------------------------------------
 --------------
# Each line profile should have the parameters:
#    nu, nu_ij, T, molecMass, v_los=optional
# and should have an integral function defined which takes
#    freq_lower, freq_upper, n_ij, T, molecMass, v_los=optional

''' Thermal line profile
    PARAMS:
      freq = frequency (Hz)
      centerFreq = rest frequency of the transition (Hz)
      T = temperature (K)
      moleculeMass = the mass of the particle (atomic units)
      v_los (optional) = the line of sight velocity to calc the doppler
       shift '''
def thermal_profile(freq, freq_center, T, moleculeMass, vlos=0.):
    if (freq==0.) or (freq_center==0.):
        profile = 0.
    else:
        new_center = freq_center * (1. + v_los/c_si)
        # thermal line width in units: sqrt(J/K *K /kg) = sqrt(kg m2/s2 /kg)
         =m/s
        a = np.sqrt(2.*kb_si*T/(moleculeMass*MH))
        f = c_si/(a*new_center*np.sqrt(np.pi))
        # units: m/s * Hz / (m/s *Hz) * m/s /(m/s * Hz) = 1/Hz
        profile = f * np.exp(-(c_si*abs(freq-new_center)/(a*new_center))**2)
    return profile
```

```
''' Integrated line profile over [freq_a, freq_b]
    This is basically the integration of a Gaussian distribution
    C sqrt(pi) int_p^q exp(-(C*x)**2) dx = 0.5 * (erf(q*C) - erf(p*C))
    with C = 1/((sqrt(2)*sigma) and sigma = therm_a * freq_center/
     (sqrt(2)*c)
'''
def integrate_thermal_profile(freq_a, freq_b, freq_center, T, moleculeMass,
 v_los=0.):
    # doppler shift
    new_center = freq_center * (1. + v_los/c_si)
    # shift integration limits so line is centered on around 0
    a = freq_a - new_center
    b = freq_b - new_center
    # calc the thermal width of the line
    therm_a = np.sqrt(2.*kb_si*T/(moleculeMass*MH))
    # calc the integral
    C = c_si / (therm_a * freq_center)
    return 0.5 * (erf(b*C) - erf(a*C))

def test_integration():
    freqs = np.linspace(1.99999e9,2.00001e9,10)
    freq_center = 2.e9
    T = 10.
    mu = 29.
    therm_a = np.sqrt(2.*kb_si*T/(mu*MH))
    tot = 0.
    for f in range(len(freqs)-1):
        part = integrate_thermal_profile(freqs[f], freqs[f+1], freq_center,
         T, mu, 0.)
        print part
        tot = tot + part
    print tot

#-----
 Other-------------------------------------------------------------------
 --------------------
''' Ray tracing along the grid direction '''
def ray_tracing(nx, ny, nz, tau, j, dx, freq):
    image = np.zeros((nx,ny))
    for z in range(nz):
        image = image * np.exp(-tau[z]) + j[z]*dx / tau[z] * (1.0 - np.exp(-
         tau[z]))
    # correct for distance
    D = 1.e6*PC
    image = image * dx**2/D**2
    # convert to dimensionless units nu*F_nu = lambda*F_lambda
    image = image * freq[1][0] # W/(sr*m2)
    image = image + 1e-50
    return image

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

if __name__=='__main__':
    test_integration()
```