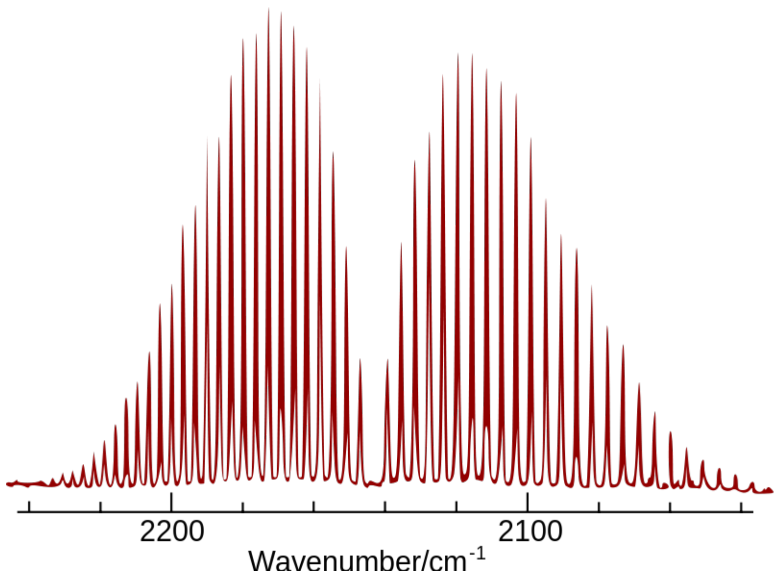

Figure 1: The vibration-rotation line spectrum of CO.

Problem Set 5

1 Good Rovibrations

Let's see if we can cook up a simulation that reproduces the rovibrational spectrum of CO pictured in Figure 1. To get some of the answers, you'll need to write code to work out the numerical details.

1.1

First, use the center wavenumber to estimate which vibrational transition we are talking about for CO. Assume we end in the $n = 0$ state.

1.2

Assuming that the populations of each J state are set by Boltzmann statistics and assuming that line intensity is proportional to the population of the upper J state, see if you can work estimate the excitation temperature of CO.
(Hint: each line corresponds to a $\Delta J = \pm 1$ transition and the spin degeneracy of each J state is $g = 2J + 1$. If you can find where the spectrum turns over, that indicates the exponential in the Boltzmann distribution has taken over from the increase in degeneracy with J.)
What value does this imply for T? Don't forget that the total energy is the sum of the vibrational and rotational energies.

1.3

Estimate the Einstein A coefficients for this Δn transition as a function of J. Is the variation sufficient that it needs to be included in predictions of line strength?

1.4

Now, use a simulation to reproduce this plot. Determine the moment of inertia I for CO as a function of n and J, as we worked on in class. Populate the n and J states according to Boltzmann statistics using the T estimated above. Finally, use the population of each state and the transition strength to determine the relative intensity of each transition as a function of frequency. Plot it versus wavenumber and compare to Figure 1.
Here's some skeleton code:

```
mu_CO # reduced mass of CO
k_CO # effective vibrational spring constant, from class, g/s^2
w0_CO # angular frequency of vibration, from mu_CO, k_CO
x_CO # atomic separation, cm

def E_vib_CO(n):
    '''Return the energy of the nth vibrational state.'''
    return E_vib

def I_CO(n, J):
    '''Return (optionally n,J-dependent) CO moment of inertia, in g*cm^2.'''
    return mom_inertia

def E_rot_CO(J):
    '''Return the energy of the Jth rotational state of CO.'''
    return E_rot

def E_rovib_CO(n, J):
    '''Return the energy of the n,Jth rovibrational state of CO.'''
    return E_rovib

def g(J):
    '''Return the degeneracy of the Jth rotational state.'''
    return degeneracy

def population(n, J, T):
    '''Fractional population in J,n rovib state state of CO.'''
    return population

def omega(J1, J0, delta_n=1):
    '''Return angular frequency of J1->J0, delta_n transition.'''
    return ang_freq

def A_CO(J1, J0, delta_n=1):
    '''Return Einstein A coefficient for J1->J0 delta_n=1 transition.'''
    return A

ns = np.array([[0],[1]]) # (2,1) array of vibrational states
```

```
Js = np.arange(20); Js.shape = (1,-1) # (1,10) array of rotational states
I_COs # (2,10) array of moments of inertia
E_COs # (2,10) array of energies
gs # (1,10) array of degeneracies
populations # (2,10) array of population fractions relative to n=0, J=0

transitions = []
for index,j in enumerate(Js.flat):
    # for each of Delta J = 1, -1
    dE # energy difference between states
    A # Einstein A of transition
    pop # population engaging in this transition
    transitions.append((w, intensity_of_line))
transitions = np.array(transitions)
nus,intensities = transitions[:,0], transitions[:,1]
```

1.5

Are there remaining discrepancies? What effects haven't you included that would improve your simulation?