SSA_sample

September 3, 2015

1 Stellar spectra A. Basic Line Formation

This notebook converts the coding parts of the exercise statement, written in IDL, into Python programming language to help you if using it. You may still need to follow the text of the statement in order to understand the instructions and/or add & modify parts of the code. WARNING: Some of the functions or parts of the code may contain typos, bugs or mistakes. Please, take them as a reference but recheck with the statement idl version and write it in your own way :)

2.4 Saha-Boltzmann populations of schadeenium

You just need to create a text file with the extension .py, e.g. SSA2.py, and save it in the directory you want. In order to run it later, you just need to write python SSA.py in the command line window, once you are in the directory where the code was saved. The basic structure of the code should be something similar to the following:

```
In [ ]: # importing useful libraries (you may need more)
        import numpy as np                  # numerical package
        import matplotlib.pyplot as plt     # plotting package
        from matplotlib import rc
        rc('font',**{'family':'serif'})     # This is for Latex writing

        # definition of constants and variables
        k = 8.61734e-5                      # Boltzmann constant
        # add all constants here so you do not need to put them in every function


        # definition of functions
        def name_function(parameters):
            statements
            statements

            return output_parameters

        def another_function(another_params):
            statements

            return another_output


        # Main part calling the functions to do things
        name_function(parameters)
        another_function(another_params)
```

Remember to use the command 'print name_of_variable' in several places to see results on the terminal when looking for errors in the code. Thus, you will be able to detect where the error is and fix it.

compute the partition functions Ur of the Schadee element:

```
In [ ]: chiion = np.array([7, 16, 31, 51])      # Schadee ionization energies into numpy array
        k = 8.61734e-5                          # Boltzmann constant in eV/deg
        temp = 5000.                            # the decimal point here also makes it a float
        u = np.zeros(4)                         # declare a 4 zero-element array
        for r in range(4):
            for s in range(chiion[r]):
                u[r] = u[r] + np.exp(-s / k / temp)
        print u                                 # prints all the values of u array (now is not zeros)

        # Notice that in Python we always start counting at zero.
```

The same made a function:

```
In [ ]: def partfunc_E(temp):
            chiion = np.array([7, 16, 31, 51])      # Schadee ionization energies into numpy array
            k = 8.61734e-5                          # Boltzmann constant in eV/deg
            u = np.zeros(4)                         # declare a 4 zero-element array
            for r in range(4):
                for s in range(chiion[r]):
                    u[r] = u[r] + np.exp(-s / k / temp)

            return u                                 # returns all the values of u array

        # Notice that the variable temp is not inside the function since it will be called when calling
        # the function using the command
        partfunc_E(temp)
        # So, the variable temp has to be defined before executing this command (outside the function).
```

Then write a Boltzmann routine which computes n_r,s/Nr

```
In [ ]: def boltz_E(temp, r, s):
            u = partfunc_E(temp)
            KeV = 8.61734e-5        # This constant does need to be defined here again if it was before
            relnrs = 1. / u[r - 1] * np.exp(-(s - 1) / (KeV * temp))

            return relnrs
```

Check this is working by printing the second Schadee table on page 12 for three temperatures

```
In [ ]: for s in range(1,11):      # now the loop starts at 1 and finishes at 10
            print boltz_E(5000., 1., s)
```

Then write the Saha routine

```
In [ ]: def saha_E(temp, elpress, ionstage):
            kerg = 1.380658e-16
            kev =
            h =
            elmass =
            kevT = kev * temp
            kergT = kerg * temp
            eldens = elpress / kergT
            chiion = np.array([7, 16, 31, 51 ])
```

```
    u = partfunc_E(temp)
    u = np.append(u, 2)        # With this command we are adding a new element to the array
    sahaconst = (2. * np.pi * elmass * kergT / (h**2))**1.5 * 2. / eldens
    nstage = np.zeros(5)
    nstage[0] = 1.             # We set the first element of the array to a value 1
    for r in range(4):
        nstage[r + 1] = nstage[r] * sahaconst * u[r + 1] / u[r] * np.exp(-chiion[r] / keVT)
    ntotal = np.sum(nstage)
    nstagerel = nstage / ntotal

    return nstagerel[ionstage - 1]
```

1.0.1 2.5 Payne curves for shadeenium

Write a function Sahabolt_E

In [ ]:
```
def sahabolt_E(temp, elpress, ion, level):

    return saha_E(temp, elpress, ion) * boltz_E(temp, ion, level)
```

In [ ]:
```
temp = np.arange(0,30001,1000)
#print temp
pop = np.zeros((5,31))
for T in np.arange(1,31):
    for r in np.arange(1,5):
        pop[r,T] = sahabolt_E(temp[T],131.,r,1)

labellst = ['ground stage', 'first ion stage', 'second ion stage', 'third ion stage']


#print pop
plt.figure(0)
# ground-state plot
for i in range(1,5):
    plt.plot(temp,pop[i,:], label=labellst[i-1])

plt.xlabel('temperature', size=14)
plt.ylabel('population', size=14)
plt.yscale('log')
plt.ylim([1e-3, 1.1])
plt.legend(loc='best')
plt.show()
```

1.0.2 2.7 Saha-Boltzmann populations of hydrogen

Write a function Sahabolt_H

In [ ]:
```
def sahabolt_H(temp,elpress,level):
    keVT = keV*temp
    kergT = kerg*temp
    eldens = elpress/kergT

    # energy levels and weights for hydrogen
    nrlevels = 100                          # reasonable partition function cut-off value
    g = np.zeros((2,nrlevels))              # declarations weights (too many for proton)
```

```
    chiexc = np.zeros((2,nrlevels))             # declaration excitation energies (idem)

    for s in range(nrlevels):
        g[0,s] = 2.*(s+1.)**2.                  # statistical weights
        chiexc[0,s] = 13.598*(1.-1./(s+1.)**2.) # excitation weights

    g[1,0] = 1.                                 # statistical weights free proton
    chiexc[1,0] = 0.

    # partition functions
    u = np.zeros([2])
    for s in range(nrlevels):
        u[0] = u[0] + g[0,s]*exp(-chiexc[0,s]/keVT)
    u[1] = g[1,0]

    # Saha
    sahaconst = (2*np.pi*elmass*kergT /(h*h))**(1.5)*2./eldens
    nstage = np.zeros(2)
    nstage[0] = 1.
    nstage[1] = nstage[0] * sahaconst * u[1]/u[0] * np.exp(-13.598/keVT)
    ntotal = np.sum(nstage)                     # sum both stages = total hydrogen density

    # Boltzmann
    nlevel = nstage[0]*g[0,level-1]/u[0]*np.exp(-chiexc[0,level-1]/keVT)
    nlevelrel = nlevel/ntotal                   # fraction of total hydrogen density

    return nlevelrel

print sahabolt_H(6000,1e2,1)

for s in range(6):
    print s+1, g[0,s], chiexc[0,s], g[0,s]*np.exp(-chiexc[0,s]/keVT)

#print
for s in range(0,nrlevels,10):
    print s+1, g[0,s], chiexc[0,s], g[0,s]*np.exp(-chiexc[0,s]/keVT)
```

1.0.3 2.8 Solar Ca+K versus Ha: line strength

In [ ]:
```
temp = np.arange(1000,20001,100)
CaH = np.zeros(temp.shape)
Caabund = 2.0e-6
for i in range(0,191):
    NCa = sahabolt_E(temp[i],1e2,2,1)         # is equal to sahabolt_Ca
    NH = sahabolt_H(temp[i],1e2,2)
    CaH[i] = NCa*Caabund/NH

plt.plot(temp,CaH, label=r'strength ratio Ca$^+$K / H$\alpha$')
plt.yscale('log')
plt.xlabel(r'temperature $T / K$', size=14)
plt.ylabel(r'Ca II K / H$\alpha$', size=14)
plt.legend(fontsize=14)
plt.show()
```

```
print 'Ca/H ratio at 5000 K = ', CaH[np.argwhere(temp==5000)][0][0]
```

1.0.4 2.9 Solar Ca+K versus Ha: temperature sensitivity

In [ ]:
```
temp = np.arange(2000,12001,100)
dNCadT = np.zeros(temp.shape)
dNHdT = np.zeros(temp.shape)
dT = 1.
for i in range(101):
    NCa = sahabolt_E(temp[i],1e2,2,1)
    NCa2 = sahabolt_E(temp[i]-dT,1e2,2,1)
    dNCadT[i] = (NCa - NCa2)/(dT*NCa)
    NH = sahabolt_H(temp[i],1e2,2)
    NH2 = sahabolt_H(temp[i]-dT,1e2,2)
    dNHdT[i] = (NH-NH2)/(dT*NH)

plt.figure()
plt.plot(temp,np.absolute(dNHdT), label=r'H')
plt.plot(temp,np.absolute(dNCadT), label=r'Ca$^+$K')

plt.yscale('log')
#plt.ylim(1e-9,1)
plt.xlabel(r'temperature $T/K$', size=14)
plt.ylabel(r"$\left| \left( \Delta n(r,s) / \Delta T \right) /  n(r,s) \right|$", size=20)
plt.legend(loc=4, fontsize=12)
```

```
NCa = np.zeros(temp.shape)
NH = np.zeros(temp.shape)
for i in range(101):
    NCa[i] = sahabolt_E(temp[i],1e2,2,1)
    NH[i] = sahabolt_H(temp[i],1e2,2)

ax[1].plot(temp,NH/np.amax(NH), ls='--',  label = 'rel. pop. H')
ax[1].plot(temp,NCa/np.amax(NCa), ls='--', label = r'rel. pop. Ca$^+$')
plt.show()
```

1.0.5 2.10 Hot stars versus cool stars

In [ ]:
```
for T in np.arange(2e3,2e4+1,2e3):
    print T, sahabolt_H(T,1e2,1)

temp = np.arange(1e3,2e4+1,1e2)
nH = np.zeros(temp.shape)
for i in range(191):
    nH[i] = sahabolt_H(temp[i],1e2,1)

plt.plot(temp,nH)
plt.xlabel('temperature $T/K$', size=14)
plt.ylabel('neutral hydrogen fraction', size=14)
```

```
plt.legend()
plt.show()
```

2 3. Fraunhofer line strengths and the curve of growth

Use the previously computed Planck function to plot Planck curves agains wavelength in the visible.

```
In [ ]: wav = np.arange(1000,20801,200)
        b = np.zeros(wav.shape)

        plt.xlabel(r'wavelength $\lambda / \AA$', size=14)
        plt.ylabel(r'Planck function', size=14)
        plt.xlim(0,20800)

        for T in range(8000,5000-1,-200):
            b[:] = planck(T, wav[:]*1e-8)
            plt.plot(wav,b,'-')
```

2.0.6 3.2 Radiation through an isothermal layer

Make plots of the emergent intensity I for given values of B and I(0) against 'tau'

```
In [ ]: B = 2.
        tau = np.arange(0.01,10.01, 0.01)
        intensity = np.zeros(tau.shape)
        for I0 in range(4,-1,-1):
            intensity[:] = I0 * np.exp(-tau[:]) + B*(1-np.exp(-tau[:]))
            plt.plot(tau, intensity, label = 'intensity I0 = ' + str(I0))

        plt.xlabel(r'optical depth $\tau$', size=14)
        plt.ylabel('intensity', size=14)
        plt.legend(fontsize=12)
        plt.show()
```

2.0.7 3.3 Spectral lines from a solar reversing layer

Voigt profile

```
In [ ]: u = np.arange(-10,10.1,0.1)
        a = np.array([0.001,0.01,0.1,1])
        vau = np.zeros((a.shape[0],u.shape[0]))

        for i in range(4):
            vau[i,:] = voigt(a[i],u[:])
            plt.plot(u[:],vau[i,:], label = 'a = ' + np.str(a[i]))

        plt.set_ylim(0,1)
        plt.set_xlim(-10,10)
        plt.legend(fontsize=12)
        plt.set_ylabel('voigt profile', size=12)

        for i in range(4):
            vau[i,:] = voigt(a[i],u[:])
            plt.plot(u[:],vau[i,:], label = 'a = ' + np.str(a[i]))
```

```
plt.set_yscale('log')
plt.legend(fontsize=12, loc = 8)
plt.set_xlabel('u', size=14)
plt.set_ylabel('logarithmic voigt profile', size=12)
```

Schuster-Schwarzchild line profile

In [ ]:
```
Ts = 5700.                    # solar surface temperature
Tl = 4200.                    # solar T-min temperature = 'reversing layer'
a = 0.1                       # damping parameter
wav = 5000.0e-8               # wavelength in cm
tau0 = 1.                     # reversing layer thickness at line center
u = np.arange(-10,10.1,0.1)
intensity = np.zeros(u.shape)

for i in range(201):
    tau = tau0 * voigt(a, u[i])
    intensity[i] = planck(Ts,wav) * np.exp(-tau) + planck(Tl,wav)*(1.-np.exp(-tau))

plt.plot(u,intensity)
plt.show()
```

```
logtau0 = np.arange(-2,2.1,0.5)

for itau in range(9):
    for i in range(201):
        tau = 10.**(logtau0[itau]) * voigt(a, u[i])
        intensity[i] = planck(Ts,wav) * np.exp(-tau) + planck(Tl,wav)*(1.-np.exp(-tau))
    plt.plot(u,intensity, label = r'$\log{(\tau_0)} = $' + np.str(logtau0[itau]))

plt.legend(loc=3, fontsize=12)
plt.show()
```

```
for iwav in range(1,4):
    wav = (iwav**2+1.)*1.0e-5          # wav = 2000, 5000, 10000 angstrom
    for itau in range(8):
        for i in range(201):
            tau = 10.**(logtau0[itau]) * voigt(a,u[i])
            intensity[i] = planck(Ts,wav) * exp(-tau) + planck(Tl,wav)*(1.-exp(-tau))
        intensity = intensity / intensity[0]
        plt.plot(u,intensity[:],  linewidth=1.)
plt.show()
```

2.0.8 3.4 The equivalent width of spectral lines

```
In [ ]: def profile(a,tau0,u):
            Ts = 5700.
            Tl = 4200.
            wav = 5000.0e-8
            intensity = np.zeros(u.size)
            usize = u.size
            for i in range(usize):
                tau = tau0 * voigt(a, u[i])
                intensity[i] = planck(Ts,wav)*np.exp(-tau) + planck(Tl,wav)*(1.-np.exp(-tau))

            return intensity



        # Checking the profile
        u = np.arange(-200,200.4,0.4)
        a = 0.1
        tau0 = 1.0e2
        intensity = profile(a,tau0,u)

        plt.plot(u,intensity)
        plt.show()



        # relative
        reldepth = (intensity[0]-intensity)/intensity[0]
        plt.plot(u,reldepth)
        eqw = sum(reldepth)*0.4
        print eqw
```

2.0.9 3.5 The curve of growth

```
In [ ]: tau0 = np.logspace(-2, 4, 61)
        eqw = np.zeros(tau0.size)

        for i in range(61):
            intensity = profile(a,tau0[i],u)
            reldepth = (intensity[0] - intensity) / intensity[0]
            eqw[i] = sum(reldepth)*0.4

        plt.plot(tau0,eqw)
        plt.xlabel(r'$\tau_0$', size=18)
        plt.ylabel(r'equivalent width $W_{\lambda}$', size=14)
        plt.xscale('log')
        plt.yscale('log')
        plt.show()
```