ex9_numerical_solution

December 5, 2025

[1]:
```
from IPython.display import Markdown
import numpy as np
import matplotlib.pyplot as plt
# Electron & proton masses, Boltzmann & Planck constants
from astropy.constants import m_e, m_p, k_B, h
# Useful units
from astropy.units import eV, kg, m, Kelvin

# Ionization energies
I1 = 13.598 * eV
I2 = 24.587 * eV
I3 = 54.416 * eV

# Partition function ratios
omega1 = 1
omega2 = 4
omega3 = 1

# Species Masses
m_H = m_p + m_e
m_He = 4 * m_H

# Number abundances
nu_1 = 0.9
nu_2 = 0.1

# Density and temperature of the medium
rho = 1e-7 * kg / m**3
T = 12500 * Kelvin

# And finally, beta (=1 since we neglect radiation pressure)
beta = 1
```

[2]:
```
# We give mu0 in atomic mass units
mu_0 = (nu_1 * m_H + nu_2 * m_He) / m_H
Markdown(rf"The value of $\mu_0$ is approximately {mu_0:.04f}")
```

[2]:
The value of μ_0 is approximately 1.3000

[3]:
```
def P(E):
    """ Eq (8) of the question sheet """
    return k_B / m_H * (1 + E) / mu_0 * rho * T

def K(E, I_i, omega_i):
    """ Eq (6) of the question sheet """
    numerator = (2 * np.pi * m_e)**(3/2) * (k_B * T)**(5/2)
    denominator = beta * P(E) * h**3
    return numerator / denominator * omega_i * np.exp(-I_i / (k_B*T))
```

[4]:
```
def getE(X1, X2, X3):
    """ Eq (1) of the solution sheet"""
    return nu_1 * X1 + nu_2 * (X2 + 2 * X3)

def getalpha(E):
    """ alpha defined as in the solution sheet """
    return (1 + E) / E

def getXs(X1, X2, X3):
    """ Here, we update X1, X2 and X3 based on the current guess """
    # Compute E:
    E = getE(X1, X2, X3)

    # Obtain the {Ki}:
    K1 = K(E, I1, omega1)
    K2 = K(E, I2, omega2)
    K3 = K(E, I3, omega3)
    alpha = getalpha(E)
    X1new = alpha * K1 / (1 + alpha*K1)
    X2new = alpha * K2 / (1 + alpha*K2*(1 + alpha*K3))
    X3new = alpha**2 * K2 * K3 / (1 + alpha*K2*(1 + alpha*K3))

    return X1new.value, X2new.value, X3new.value
```

[5]:
```
# Starting guess
X1 = 1
X2 = 0.0
X3 = 0.0
```

[6]:
```
X1, X2, X3 = getXs(X1, X2, X3)
Markdown(f"After one iteration: $X_1={X1:.05f}$, $X_2={X2:.05f}$, $X_3={X3:.
↪05f}$")
```

[6]:
After one iteration: $X_1 = 0.99629$, $X_2 = 0.03834$, $X_3 = 0.00000$

[7]:
```
X1, X2, X3 = getXs(X1, X2, X3)
Markdown(f"After two iterations: $X_1={X1:.05f}$, $X_2={X2:.05f}$, $X_3={X3:.
↪05f}$")
```

[7]:
After two iterations: $X_1 = 0.99629$, $X_2 = 0.03832$, $X_3 = 0.00000$

[8]:
```
X1, X2, X3 = getXs(X1, X2, X3)
Markdown(f"After three iterations: $X_1={X1:.05f}$, $X_2={X2:.05f}$, $X_3={X3:.
↪05f}$")
```

[8]:
After three iterations: $X_1 = 0.99629$, $X_2 = 0.03832$, $X_3 = 0.00000$

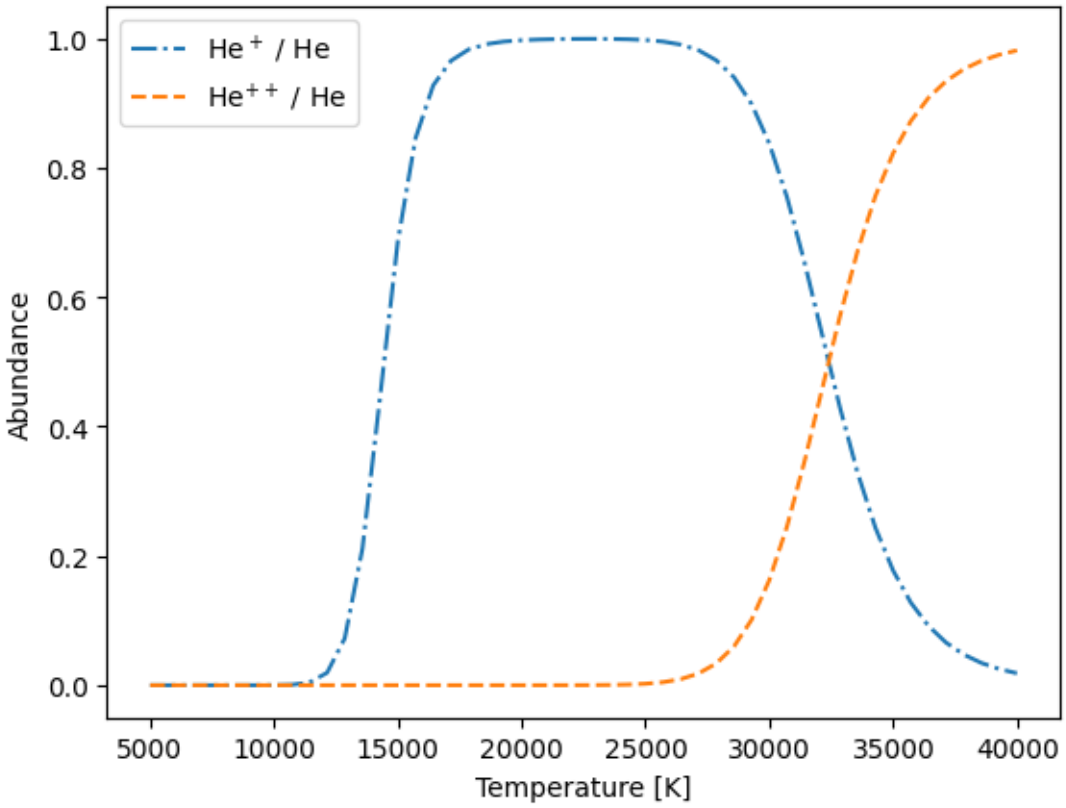
We see that in 3 iterations, we already converge to a level of precision reaching 10^{-5}.

[9]:
```
# We can automate this process using a simple function
def doIterations(niteration=5):
    # Starting guess
    X1 = 1
    X2 = X3 = 0
    E = 0.9

    # Iterate:
    for i in range(niteration):
        X1, X2, X3 = getXs(X1, X2, X3)

    return X1, X2, X3, E
```

[10]:
```
# We are interested in particular in helium, so we
# will store its ionization fraction (both HeII and HeIII)
# over a range of temperatures
X2s, X3s = [], []
Ts = np.linspace(5000, 40000)

# Compute the result for each temperature:
for T in Ts:
    T = T * Kelvin
    X1, X2, X3, E = doIterations()
    X2s.append(X2)
    X3s.append(X3)
```

[11]:
```
plt.plot(Ts, X2s, '-.', label=r"He$^+$ / He")
plt.plot(Ts, X3s, '--', label=r"He$^{++}$ / He")
plt.xlabel('Temperature [K]')
plt.ylabel('Abundance')
plt.legend()
```

[11]:
```
<matplotlib.legend.Legend at 0x109fcb230>
```



Note that it really needs to get hot for helium to be fully ionized !