Phys 600: HW 4

Yarone Tokayer

October 29, 2023

In [1]:
```
# Install packages

#from astropy.cosmology import FlatLambdaCDM
from astropy import units as u

import numpy as np
from scipy.special import zeta

import matplotlib.pyplot as plt

from tqdm import tqdm
```

In [2]:
```
plt.rcParams.update({
    "text.usetex": True,
    "font.family": "Helvetica"
})
```

Problem 1

1.2: Figure for $X_e(z)$

In [3]:
```
# Constants

Tcmb0 = 0.235 * u.meV # CMB temperature today
eta = 6e-10 # Baryons-to-photon fraction (Baumann)
E_I = 13.6 * u.eV # Ionization energy of hydrogen
m_e = 0.510998950 * u.MeV # Mass of electron (Wikipedia)
```

In [4]:
```
def x_e(z):
    '''
    Function to compute the electron fraction as a function of redshift

    Inputs:
    z - Redshift

    Returns:
    x_e - free electron fraction at that redshift
    '''

    T = Tcmb0 * ( 1 + z )

    a = (2 * zeta(3) / np.pi ** 2) * eta * ((2 * np.pi * T / m_e) ** (3/2)) * np.exp(E_I / T)

    x_e = ((-1 + np.sqrt(1 + 4*a)) / (2 * a)).to(u.dimensionless_unscaled).value

    return x_e
```

1.3: Invert to find $z(X_e)$

In [5]:
```
def x_e_to_z(x_e_target, tolerance=1e-6):
    '''
    Function to find redshift (z) given the free electron fraction (x_e) using a bisection method.

    Inputs:
    x_e_target - Free electron fraction
    tolerance - Tolerance for the bisection method convergence

    Returns:
    z - Redshift corresponding to the given lookback time
    '''

    # Initialize the search interval for z
    z_low, z_high = 1e2, 1e4

    # Perform the bisection search
    while z_high - z_low > tolerance:
        z_mid = (z_low + z_high) / 2
        x_e_mid = x_e(z_mid)
        if x_e_mid > x_e_target:
            z_high = z_mid
        else:
            z_low = z_mid

    # The bisection method has converged; return the redshift
    return z_mid
```

Our two points of interest:

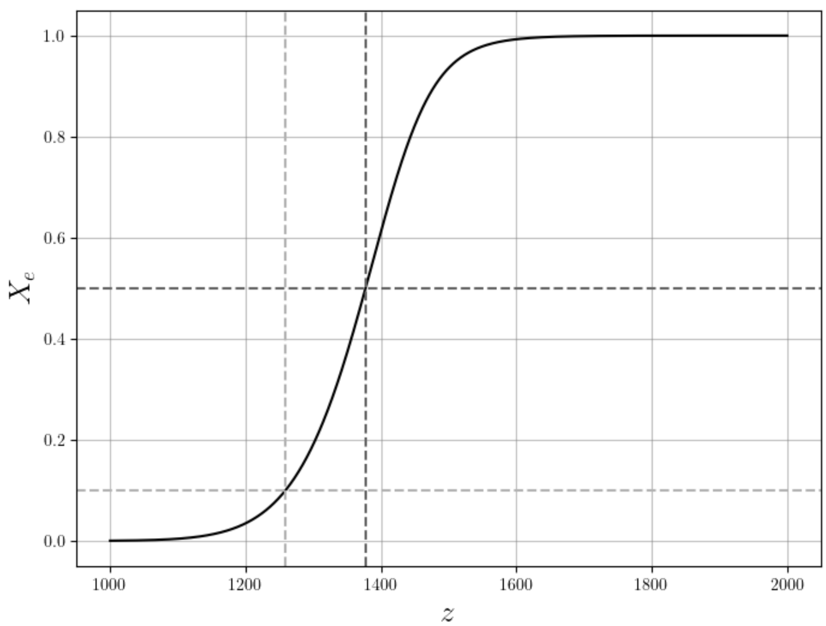
In [6]:
```
x_e_to_z(0.1), x_e_to_z(0.5)
```

Out[6]: (1258.9799694891553, 1377.2434625017922)

Now we can plot everything:

In [7]:
```
# Redshift range to calculate

z = np.linspace(1000, 2000, 1000)
```

In [8]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,6))

ax.plot(z, x_e(z), color='black')
ax.axhline(y=0.5, color=(0.4,0.4,0.4), ls='--')
ax.axhline(y=0.1, color=(0.7,0.7,0.7), ls='--')
ax.axvline(x=x_e_to_z(0.5), color=(0.4,0.4,0.4), ls='--')
ax.axvline(x=x_e_to_z(0.1), color=(0.7,0.7,0.7), ls='--')

ax.set_axisbelow(True)
ax.xaxis.grid(color='gray', alpha=0.5, linestyle='-')
ax.yaxis.grid(color='gray', alpha=0.5, linestyle='-')

ax.set_xlabel('$z$', fontsize=18)
ax.set_ylabel(r'$ X_e$', fontsize=18)

# ax.legend(fontsize=14)

fig.savefig('/Users/yaronetokayer/Yale Drive/Classes/PHYS 600/phys600 hw/phys600 hw 4/x_e_plot.png',
          dpi=300, bbox_inches='tight')
```



1.4: Age of the universe

In [9]:
```
h_0 = 70 * u.km / u.s / u.Mpc
```

In [10]:
```
((1 / h_0) * (2/3) * (1 / (1 + x_e_to_z(0.1)))**(3/2)).to(u.kyr)
```

Out[10]: 208.21491 kyr

1.5: z at decoupling

Evaluate the target value for $X_e(z)(1+z)^{3/2}$ (see write-up)

In [11]:
```
# Constants
Om0 = 0.3; sigma_T = 2e-3 * u.MeV**-2
```

In [12]:
```
# Need to convert h_0 to natural units
h_0_nat = h_0 * u.GeV * 6.5821e-25 * u.s
```

In [13]:
```
target = (h_0_nat * np.pi**2 * np.sqrt(Om0) / (2 * eta * zeta(3) * sigma_T * Tcmb0**3)).to(u.dimensionless_unscaled)
target
```

Out[13]: 215.59192

Use a bisection method to solve for z:

In [14]:
```
tolerance=1e-6

# Initialize the search interval for z
z_low, z_high = 1e2, 1e4

# Perform the bisection search
while z_high - z_low > tolerance:
    z_dec = (z_low + z_high) / 2
    x_e_mid = x_e(z_dec) * (1 + z_dec)**(3/2)
    if x_e_mid > target:
        z_high = z_dec
    else:
        z_low = z_dec

# The bisection method has converged; return the redshift
z_dec
```

Out[14]:
```
1113.8222924259026
```

The age of the universe at this redshift:

In [15]:
```
((1 / h_0) * (2/3) * (1 / (1 + z_dec))**(3/2)).to(u.kyr)
```

Out[15]: $250.17782\ \mathrm{kyr}$

The free electron fraction at this redshift:

In [16]:
```
x_e(z_dec)
```

Out[16]:
```
0.005791939562935705
```

Problem 2

Assuming the "freezeout" temperature is 0.8 MeV, estimate the freeze out abundance of neutrons (X_n). We do this using our formula for X_n:

In [17]:
```
def x_n(t, q=1.3 * u.MeV):
    '''
    Function to compute the neutron fraction as a function of temperature

    Inputs:
    t - Temperatute (astropy natural (energy) units)
    q - Neutron mass minus the proton mass (astropy natural (energy) units)

    Returns:
    x_n - neutron fraction at that temperature
    '''

    ratio = (q / t).to(u.dimensionless_unscaled).value

    x_n = np.exp( -ratio ) / (1 + np.exp( -ratio ) )

    return x_n
```

In [18]:
```
x_n_fo = x_n(0.8 * u.MeV)
x_n_fo
```

Out[18]:
```
0.1645164628965632
```

Mass fraction of helium-4 at this abundance:

In [19]:
```
2 * x_n_fo / ( 1 - x_n_fo )
```

Out[19]:
```
0.3938233504083882
```

Now for $Q = 2.6$ MeV:

In [20]:
```
x_n(0.8 * u.MeV, q=2.6 * u.MeV), 2 * x_n(0.8 * u.MeV, q=2.6 * u.MeV) / ( 1 - x_n(0.8 * u.MeV, q=2.6 * u.MeV) )
```

Out[20]:
```
(0.037326887344129464, 0.07754841566344403)
```

Problem 3

Plot $Y_{\rm eq}$ and Y

Functions:

In [21]:
```
def y_eq(x):
    '''
    Function to calculate Y_eq as a function of dimensionless time x=n/T^3

    Inputs:
```

```
    x - A single value or a NumPy array of dimensionless time values

    Returns:
    y_eq - A single value or a NumPy array of Y_eq values
    '''

    # Ensure that x is a NumPy array for vectorized calculations
    x = np.asarray(x)

    xi = np.linspace(0, 500, 10000)  # xi values to integrate over

    if x.size == 1:  # Scalar input
        integrand = xi**2 / (np.exp(np.sqrt(xi**2 + x**2)) + 1)
        y_eq_value = (1 / (2 * np.pi**2)) * np.trapz(integrand, x=xi)
        return y_eq_value

    else:  # Array input
        # Expand dimensions of x and xi to allow broadcasting
        x_expanded = x[:, np.newaxis]
        xi_expanded = xi[np.newaxis, :]

        integrand = xi_expanded**2 / (np.exp(np.sqrt(xi_expanded**2 + x_expanded**2)) + 1)
        y_eq_values = (1 / (2 * np.pi**2)) * np.trapz(integrand, x=xi, axis=1)

        return y_eq_values
```

In [22]:
```
def y(x, lam=1e-6, x0=0.01, y0=1e-20):
    '''
    Function to compute Y for freeze in DM

    Inputs:
    x - A single value or a NumPy array of dimensionless time values
    lam - Lambda parameter (default: 1e-6)
    x0 - Initial value of x (default: 0.01)
    y0 - Initial value of Y (default: 1e-20)

    Returns:
    y - A single value or a NumPy array of Y values
    '''

    # Ensure that x is a NumPy array for vectorized calculations
    x = np.asarray(x)

    if x.size == 1:
        x_array = np.linspace(x0, x, 1000)
        h = x_array / (x_array + 2)
        integrand = lam * x_array * h * y_eq(x_array)
        delta_y = np.trapz(integrand, x=x_array)
        return y0 + delta_y
    else:
        y_values = np.empty(len(x))
        for i, x_value in tqdm(enumerate(x), total=len(x)):
            x_array = np.linspace(x0, x_value, 1000)
            h = x_array / (x_array + 2)
            integrand = lam * x_array * h * y_eq(x_array)
            delta_y = np.trapz(integrand, x=x_array)
            y_value = y0 + delta_y
            y_values[i] = y_value
        return np.array(y_values)
```

Compute:

In [23]:
```
x0 = 0.01
xf = 100
x = np.geomspace(x0, xf, 1000)
lams = [1e-6, 1e-8, 1e-10]
ys = []

for lam in lams:
    ys.append( y(x, lam=lam, x0=x0) )
```

```
100%|████████████████████████████████████████| 1000/1000 [01:09<00:00, 14.34it/s]
100%|████████████████████████████████████████| 1000/1000 [01:10<00:00, 14.24it/s]
100%|████████████████████████████████████████| 1000/1000 [01:08<00:00, 14.66it/s]
```

In [24]:
```
y_eq_plot = y_eq(x)
```

Truncate arrays for plotting:

In [25]:
```
mask = x >= 0.1
```

In [26]:
```
fig, ax = plt.subplots(1, 1, figsize=(8,6))

for i, y in enumerate(ys):
    ax.plot(x[mask], y[mask], color='black',
            label=r'$\lambda_1$ = ' + str(lams[i]), alpha = 1 - i / len(ys))

ax.plot(x[mask], y_eq_plot[mask], color='black', ls='--', label=r'$Y_\mathrm{eq}$')

ax.set_axisbelow(True)
ax.xaxis.grid(color='gray', alpha=0.5, linestyle='-')
ax.yaxis.grid(color='gray', alpha=0.5, linestyle='-')
```

```
ax.set_xscale('log')
ax.set_yscale('log')

ax.set_ylim([1e-12,1])

ax.set_xlabel(r'$\log x$', fontsize=18)
ax.set_ylabel(r'$Y=n/T^3$', fontsize=18)

ax.legend(fontsize=14)

fig.savefig('/Users/yaronetokayer/Yale Drive/Classes/PHYS 600/phys600 hw/phys600 hw 4/y_plot.png',
          dpi=300, bbox_inches='tight')
```