```python
from q1_b import *

import matplotlib.pyplot as plt
from numpy import sqrt, pi, log


def l(g_w):
    '''

    Parameters
    ----------
    g_w : Coupling Constant

    Returns
    -------

    Lambda constant in ODE Equation

    '''
    m_x = 500
    g_s = 106.75

    def sigma_v():
        m_x = 500
        m_w = 80.4

        return (m_x ** 2 / (16 * pi ** 2)) * ((g_w) / (m_w)) ** 4

    def H():
        M_pl = 2.4e18
        m_x = 500
        g_s = 106.75

        return ((pi / 3) * sqrt(g_s / 10) * m_x ** 2) / (M_pl)

    return ((2 * pi ** 2) / 45) * g_s * m_x ** 3 * sigma_v() / H()


def dfX(logx, logN, g_w):
    '''

    Parameters
    ----------
    logx : specific X point
    logN : Specific N point
    g_w : Coupling Constant

    Returns
    -------
    dF/dX at a given point.

    '''
    return l(g_w) * (exp(logN) - exp(-logN + 2 * log(N_eq(exp(logx))))) * exp(-logx)


def dfN(logx, logN, g_w):
```

```python
    '''

    Parameters
    ----------
    logx : specific X point
    logN : Specific N point
    g_w : Coupling Constant

    Returns
    -------
    dF/dN at a given point.

    '''
    return - l(g_w) * (exp(logN) + exp(-logN + 2 * log(N_eq(exp(logx))))) * exp(-logx)


def f(logx, logN, g_w):
    '''

    Parameters
    ----------
    logx : specific X point
    logN : Specific N point
    g_w : Coupling Constant

    Returns
    -------
    F(X,N) written in log transformation

    '''
    return -l(g_w) * (exp(logN) - exp(-logN + 2 * log(N_eq(exp(logx))))) * exp(-logx)


def dN(logx, logN, dlogx, g_w):
    '''

    Parameters
    ----------
    logx : specific X point
    logN : Specific N point
    g_w : Coupling Constant

    Returns
    -------
    Delta N Taylor Transformation to first order

    '''
    return (f(logx, logN, g_w) * dlogx + dfX(logx, logN, g_w) * (dlogx ** 2)) / (1 - dfN(logx, logN, g_w) * dlogx)


# Set-up constants
g_w_array = [1e-4, 1e-3, 1e-2, 1e-1, 1]
g_w_colors = ['b', 'g', 'r', 'c', 'm']

```

```python
# Set-up x space
N = 1000
x_min = 0.1
x_max = 1000.
x_evals = np.linspace(log(x_min), log(x_max), N)
dlogx = x_evals[1] - x_evals[0]

# Calculate Nx(x)
for i in range(len(g_w_array)):
    log_N_x = []
    val = log(N_eq(exp(x_min)))
    for j in range(len(x_evals)):
        if j == 0:
            log_N_x.append([val])
        else:
            old_val = val
            val += dN(x_evals[j], old_val, dlogx, g_w_array[i])
            log_N_x.append([val])
    if g_w_array[i] >= 1e-3:
        print("Experimental N(oo) = {} @ gw = {}".format(exp(log_N_x[-1]), g_w_array[i]))
        plt.hlines(6e-17 * g_w_array[i] ** (-3.8), x_min, x_max, linestyles='dashed', colors='{}'.format(g_w_colors[i]),
                   linewidth=1)
        print("Analyic N(oo) = {}".format(6e-17 * g_w_array[i] ** (-3.8)))
    plt.loglog(exp(x_evals), exp(log_N_x), '{}'.format(g_w_colors[i] + '-'), linewidth=1.5,
               label=r'$g_w$ = {}'.format(g_w_array[i]))

# Plot
plt.loglog(x, N_eq(x), 'k--', linewidth=1, label=r'$N_{eq}(x)$')
plt.title(r'$N_x(x)$')
plt.ylabel(r'$N_x$')
plt.xlabel(r'$x$')
plt.xlim(x_min, x_max)
plt.legend(loc='best')
plt.savefig('g_w_transitions', dpi=300)
plt.show()
```