```python
'''
Problem 18
Quantum Bouncing Ball
-----------------------
a) Find the average energy <E> in units of mgl_g
(Plot <E>/mgl_g vs. kbT/mgl_g)

b) Find the average position of the particle above the floor, <z>,
(Plot <z>/l_g vs. kbT/mgl_g

c) Find the probability distribution P(z') for finding the particle
between z' and z'+dz' vs. z'=z/l_g. For kbT/mgl_g = 10, compare
result to the expected classical distribution.
'''

import numpy as np
from scipy.special import airy, ai_zeros
from scipy.integrate import quad_vec
from scipy.integrate import quad
import matplotlib.pyplot as plt

class problem_18:
    def __init__(self, x):

        # let x = kbT/mgl_g
        self.x = x

        # Finds first 1000 z_n terms
        self.zn = ai_zeros(100)[0]

        # En in units of mgl_g is simply -zn
        self.En = -1.0 * self.zn

        # Normalization Constant
        self.Nn = self._normalization()

    def _Pn(self, x):
        '''
        This function find the probability exp(zn/x)/Q
        given x=kbT/mgl_g

        Returns an array of Pn
        '''

        # Find the partition function
        Q = np.sum(np.exp(-self.En / x))
```

```python
        # An array of probability Pn
        Pn = np.exp(-self.En / x) / Q

        # Make sure probabilities add up to 1
        assert np.isclose(np.sum(Pn), 1.0)

        return Pn

    def _average_energy(self, x):
        '''
        This function finds the average energy in the unit mgl_g
        <E>/mgl_g = Sum Pn * En/mgl_g
                  = Sum Pn * -zn
        '''

        average_energy = np.sum(self._Pn(x) * self.En)
        return average_energy

    def _normalization(self):
        # Find Normalization constant using sympy

        Ai2_func = lambda z : airy(z - np.abs(self.zn))[0]**2
        invNn2 = quad_vec(Ai2_func, 0, np.inf)[0]
        Nn = 1.0/np.sqrt(invNn2)

        return Nn

    def _average_z(self, x):
        '''
        Finds the <z>, given x.
        Returns a single value <z>
        '''

        z_n_func = lambda z : z * (airy(z - np.abs(self.zn))[0])**2
        average_z = np.sum(self._Pn(x) * self.Nn**2 * quad_vec(z_n_func, 0, np.inf)[0])

        return average_z

    def _quantum_prob_dist(self, z, x):
        '''
        This function finds the probability distribution
        Returns a single point of probability distribution given z and x.
        '''

        P_dist = np.sum(self._Pn(x) * (self.Nn*airy(z - np.abs(self.zn))[0])**2)
```

```python
        return P_dist

    def _classical_prob_dist(self, z, x):
        # Classical distribution

        func = lambda z, x: np.exp(-z/x)
        Q = quad(func, 0, np.inf, args=(x))[0]
        P_dist = np.exp(-z / x) / Q

        return P_dist

    def _plot(self, x, y, xlabel, ylabel, label, dotsize=25, style="scatter"):
        '''
        This function does scatter plot given:
        y = data points for y-axis
        ylabel: label for the y-axis
        '''
        fig, ax1 = plt.subplots()

        for i in range(len(y)):
            if style == "scatter":
                ax1.scatter(x, y[i], label=label[i], s=dotsize)

            elif style == "linear":
                ax1.plot(x, y[i], label=label[i])

        ax1.legend(loc="upper left")
        ax1.set_xlabel(xlabel)
        ax1.set_ylabel(ylabel)
        plt.show()

        return fig

    def do_part_a(self):
        # Do part a of the problem, plot <E>

        average_energies = []

        # Looop over different x

        for i in range(len(self.x)):
            average_energies.append(self._average_energy(self.x[i]))

        y = [average_energies]
```

```python
        xlabel = r"$\frac{k_BT}{mgl_g}$"
        ylabel = r"$\frac{<E>}{mgl_g}$"
        label = ["Part A"]

        self._plot(self.x, y, xlabel, ylabel, label)

    def do_part_b(self):
        # Do part a of the problem, plot <z> = Pn * integral(z * (N*Ai(z-|zn|))**2)

        average_zs = []

        # Looop over different x
        for i in range(len(self.x)):
            average_zs.append(self._average_z(self.x[i]))

        y = [average_zs]

        xlabel = r"$\frac{k_BT}{mgl_g}$"
        ylabel = r"$\frac{<z>}{l_g}$"
        label = ["Part B"]

        self._plot(self.x, y, xlabel, ylabel, label)

    def do_part_c(self, x):
        # Do part c of the problem, plot probability distribution with given x

        quantum_ps = []
        zs = np.linspace(1e-5, 50, 1000)

        # Looop over different x
        for i in range(len(zs)):
            quantum_ps.append(self._quantum_prob_dist(zs[i], x))

        classical_ps = self._classical_prob_dist(zs, x)
        y = [quantum_ps, classical_ps]

        xlabel = "z'"
        ylabel = r"$P(z')$"
        label = [f"Quantum", f"Classical"]
        self._plot(zs, y, xlabel, ylabel, label, dotsize=5, style="linear")

prob_18 = problem_18(np.array([0.5, 1.0, 3.0, 5.0, 10.0]))
prob_18.do_part_a()
prob_18.do_part_b()
prob_18.do_part_c(10.0)
```