```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import fsolve
from scipy.integrate import quad


class problem_28:

    def __init__(self):

        self.T = np.logspace(-4, 1.0, 500)

    def chemical_potential(self):
        return 1.0 - self.T**2*np.pi**2/6.0

    def mu_integral(self, eps, T, u):
        # z=eps/e_f,  y=u/e_f,  x=kbT/e_f
        return 2.0*eps/(np.exp((eps-u)/T) + 1.0)

    def numerical_chemical_potential(self):

        def zero_func(u, T):
            ans = quad(self.mu_integral, 0.0, 100.0, args=(T, u),
                       epsrel=1.e-11, epsabs=1.e-11)[0]
            return 1.0 - ans

        u = []
        u0 = self.chemical_potential()

        for n, T in enumerate(self.T):
            u.append(fsolve(zero_func, u0[n], args=(T), xtol=1e-11)[0])

        return u

    def classical_chemical_potential(self):
        return -self.T*(np.log(2) + 2.0*np.log(self.T))

    def heat_capacity(self):
        u = self.chemical_potential()
        return 4.0/3.0*np.pi**2*u*self.T

    def classical_heat_capacity(self):
        return np.full(np.shape(self.T), 2.0)

    def heat_capacity_integral(self, eps, T, u):
        # z=eps/e_f,  y=u/e_f,  x=kbT/e_f
```

```
        return 2.0*eps*eps/(np.exp((eps-u)/T) + 1.0)

    def numerical_heat_capacity(self, mu):

        C_A = []
        for n, T in enumerate(self.T):
            C_A.append(quad(self.heat_capacity_integral, 0.0, 130.0, args=(T, mu[n]),
                            epsrel=1.e-11, epsabs=1.e-11)[0])

        C_A = np.array(C_A)
        C_A = np.diff(C_A) / np.diff(self.T)

        return C_A

    def plot(self):

        fig = plt.figure(figsize=(16, 10))
        ax1 = fig.add_subplot(121)
        ax2 = fig.add_subplot(122)

        chemical_potential = self.chemical_potential()
        classical_chemical_potential = self.classical_chemical_potential()
        numerical_chemical_potential = self.numerical_chemical_potential()

        heat_capacity = self.heat_capacity()
        classical_heat_capacity = self.classical_heat_capacity()
        numerical_heat_capacity = self.numerical_heat_capacity(numerical_chemical_potential)

        ax1.plot(self.T[chemical_potential > numerical_chemical_potential[-1]],
                 chemical_potential[chemical_potential > numerical_chemical_potential[-1]],
                 "-",
        # ax1.plot(self.T, chemical_potential, "-",
                 label="Sommerfeld Chemical Potential")
        ax1.plot(self.T, classical_chemical_potential, "--",
                 label="Classical Chemical Potential")
        ax1.plot(self.T, numerical_chemical_potential, "-.",
                 label="Numerical Chemical Potential")

        ax2.plot(self.T[self.T < 1.0], heat_capacity[self.T < 1.0], "-",
                 label="Sommerfeld Heat Capacity")
        ax2.plot(self.T, classical_heat_capacity, "--",
                 label="Classical Heat Capacity")
        ax2.plot(self.T[1:][numerical_heat_capacity > 0.0],
                numerical_heat_capacity[numerical_heat_capacity > 0.0],
                 "-.", label="Numerical Heat Capacity")
```

```
        ax1.set_xscale("log")
        ax2.set_xscale("log")

        ax1.legend()
        ax2.legend()

        ax1.set_xlabel(r"$\frac{k_BT}{\epsilon_F}$")
        ax2.set_xlabel(r"$\frac{k_BT}{\epsilon_F}$")
        ax1.set_ylabel(r"$\frac{\mu}{\epsilon_F}$")
        ax2.set_ylabel(r"$\frac{C_A}{Nk_B}$")
        plt.show()
        return fig


prob28 = problem_28()
prob28.plot()
```