```python
import numpy as np
import matplotlib.pyplot as plt

# Problem parameters
m, n = 500, 1000
A = np.random.randn(m, n)
b = np.ones(m)
alpha = 0.001
rho = 270
max_iters = 1000


def dual_ascent(A, b, alpha, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = -0.5 * A.T @ y
        y = y + alpha * (A @ x - b)
        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

def method_of_multipliers(A, b, rho, max_iters):
    x_vals = []
    infeasibility = []

    y = np.zeros(m)
    x = np.zeros(n)

    for _ in range(max_iters):
        x = (rho * (A.T @ b) - A.T @ y) / (2 + (np.linalg.norm(A)**2))
        y = y + rho * (A @ x - b)

        x_vals.append(x.T @ x)
        infeasibility.append(np.linalg.norm(A @ x - b))

    return x_vals, infeasibility

x_vals_dual, infeasibility_dual = dual_ascent(A, b, alpha, max_iters)
x_vals_mm, infeasibility_mm = method_of_multipliers(A, b, rho, max_iters)

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
plt.plot(x_vals_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(x_vals_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$x^T x$")
plt.title("Objective Function Value")
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(infeasibility_dual, label='Dual Ascent', linestyle='dashed')
plt.plot(infeasibility_mm, label='Method of Multipliers')
plt.xlabel("Iteration")
plt.ylabel(r"$\|Ax - b\|$")
plt.yscale("log")
plt.title("Constraint Violation (log scale)")
plt.legend()

plt.tight_layout()
plt.show()
```