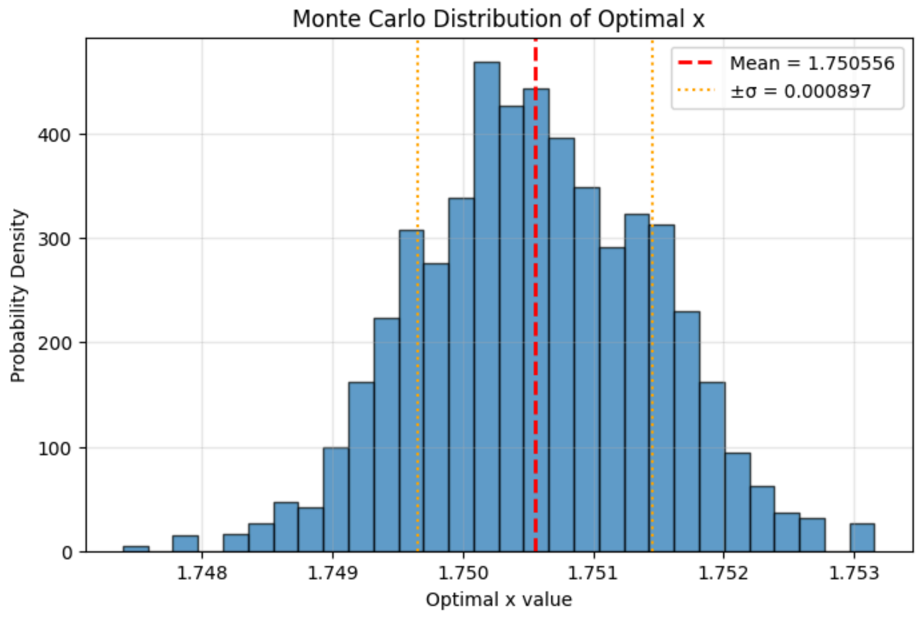
```
"""monte carlo simulation doing in this:
  its finding the mean optimum value across the N samples of the 9 planet dataset and then we can find the standard devi

"""For each iteration (1000 times):
    ->Take the 9 planet distances
    ->Add random noise to each of the 9 values (simulating measurement error)
    ->This creates one perturbed dataset with 9 noisy planet distances
    ->Find the optimal x for this single perturbed dataset
    ->Store this one optimal x value
    -> We have 1000 different optimal x values (one from each perturbed dataset)
    -> Calculate the standard deviation of these 1000 x values
    -> This standard deviation IS the uncertainty"""

import numpy as np
import matplotlib.pyplot as plt

# Planet data in AU
a = np.array([0.3871, 0.7233, 1.0000, 1.5236, 2.7692, 5.2034, 9.5371, 19.165, 30.178])
n = np.array([-2, -1, 0, 1, 2, 3, 4, 5, 6])

def error_derivative(x, a_data, n_values):
    """Derivative of error function"""
    predicted = x ** n_values
    relative_errors = (predicted - a_data) / a_data
    return np.sum(2 * relative_errors * n_values * (x ** (n_values - 1)) / a_data)

def find_optimal_x(a_data, n_values):
    """Find optimal x using bisection, Assume the optimal x is between 1.5 and 2.5
    and bisection method will converge in 50 iterations"""
    x_left, x_right = 1.5, 2.5
    for _ in range(50):
        x_mid = (x_left + x_right) / 2.0
        deriv_mid = error_derivative(x_mid, a_data, n_values)
        if abs(deriv_mid) < 1e-6:
            return x_mid
        deriv_left = error_derivative(x_left, a_data, n_values)
        if deriv_mid * deriv_left < 0:
            x_right = x_mid
        else:
            x_left = x_mid
    return (x_left + x_right) / 2.0


num_simulations = 1000
sigma = 0.005 * a  # 0.5% uncertainty
x_values = np.zeros(num_simulations)

print(f"Running {num_simulations} Monte Carlo simulations...")
for i in range(num_simulations):
    a_perturbed = np.random.normal(a, sigma)
    x_values[i] = find_optimal_x(a_perturbed, n)
    if (i + 1) % 100 == 0:
        print(f"  {i + 1}/{num_simulations} completed")

# Results
mean_x = np.mean(x_values)
std_x = np.std(x_values, ddof=1)

print(f"\nResult: x = {mean_x:.6f} ± {std_x:.6f}")
print(f"Range: [{np.min(x_values):.6f}, {np.max(x_values):.6f}]")

# Plot histogram
plt.figure(figsize=(8, 5))
plt.hist(x_values, bins=30, density=True, alpha=0.7, edgecolor='black')
plt.axvline(mean_x, color='red', linestyle='--', linewidth=2, label=f'Mean = {mean_x:.6f}')
plt.axvline(mean_x - std_x, color='orange', linestyle=':', linewidth=1.5, label=f'±σ = {std_x:.6f}')
plt.axvline(mean_x + std_x, color='orange', linestyle=':', linewidth=1.5)
plt.xlabel('Optimal x value')
plt.ylabel('Probability Density')
plt.title('Monte Carlo Distribution of Optimal x')
plt.legend()
plt.grid(alpha=0.3)
plt.savefig('monte_carlo_histogram.png', dpi=200)
plt.show()

print("\nPlot saved: monte_carlo_histogram.png")
```

```
Running 1000 Monte Carlo simulations...
  100/1000 completed
  200/1000 completed
  300/1000 completed
  400/1000 completed
  500/1000 completed
  600/1000 completed
  700/1000 completed
  800/1000 completed
  900/1000 completed
  1000/1000 completed

Result: x = 1.750556 ± 0.000897
Range: [1.747396, 1.753154]
```



```
Plot saved: monte_carlo_histogram.png
```