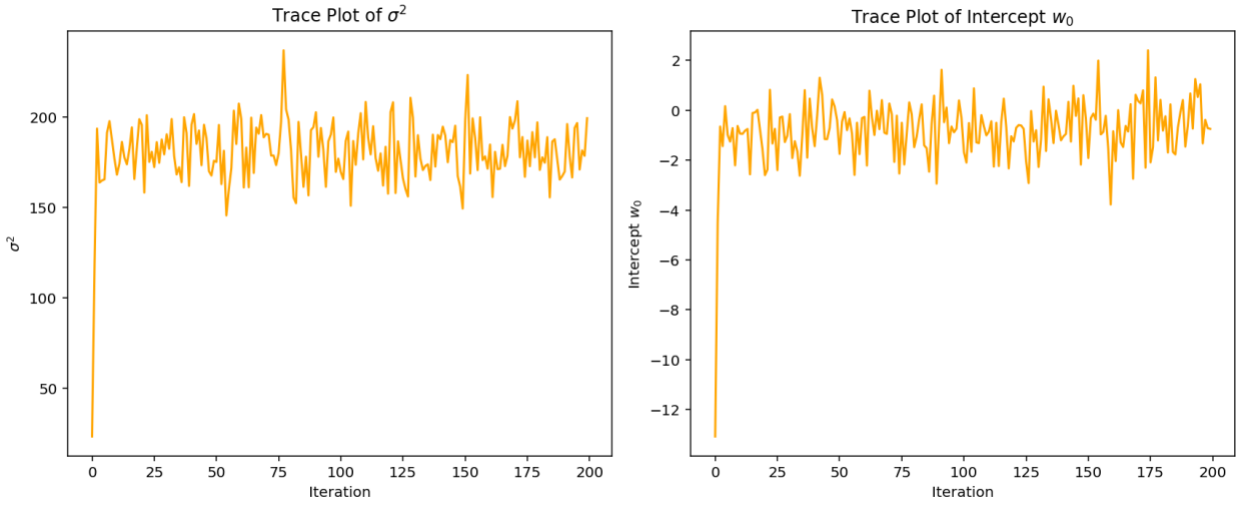
Problem 2

```
import numpy as np
import matplotlib.pyplot as plt
from numpy.linalg import inv
from scipy.stats import invgamma
from numpy.random import default_rng

file_path = "LAozone.data"

with open(file_path, "r") as f:
    lines = f.readlines()

data = []
for line in lines[1:]:
    if line.strip():
        parts = line.strip().split(",")
        if len(parts) == 10:
            data.append([float(x) for x in parts])



data = np.array(data)

y = data[:, 0]  # ozone
wind = data[:, 2]
humidity = data[:, 3]
```

```
temp = data[:, 4]
vis = data[:, 8]

X = np.column_stack((np.ones(len(y)), wind, humidity, temp, vis))

n, d = X.shape
XtX = X.T @ X
XtX_inv = inv(XtX)
prior_mean = np.zeros(d)
prior_cov = 10 * XtX_inv

a0, b0 = 3, 1

n_iter = 200
rng = default_rng()
w_samples = np.zeros((n_iter, d))
sigma2_samples = np.zeros(n_iter)
sigma2 = 1.0

for i in range(n_iter):
   Vn = inv(inv(prior_cov) + XtX / sigma2)
   wn = Vn @ (X.T @ y / sigma2)
   w = rng.multivariate_normal(wn, Vn)

   resid = y - X @ w
   an = a0 + n / 2
```

```
    bn = b0 + 0.5 * resid.T @ resid
    sigma2 = invgamma.rvs(a=an, scale=bn, random_state=rng)

    w_samples[i, :] = w
    sigma2_samples[i] = sigma2

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

plt.subplot(1, 2, 1)
plt.plot(sigma2_samples, color="orange")
plt.title("Trace Plot of $\\sigma^2$")
plt.xlabel("Iteration")
plt.ylabel("$\\sigma^2$")

plt.subplot(1, 2, 2)
plt.plot(w_samples[:, 0], color="orange")
plt.title("Trace Plot of Intercept $w_0$")
plt.xlabel("Iteration")
plt.ylabel("Intercept $w_0$")

plt.tight_layout()
plt.show()

w_mean = np.mean(w_samples, axis=0)
sigma2_mean = np.mean(sigma2_samples)
```

```
print("Bayes Estimator of w (posterior mean):")
for i in range(d):
    print(f"w[{i}] = {w_mean[i]:.4f}")

print(f"\nBayes Estimator of σ² (posterior mean): {sigma2_mean:.4f}")
```



Bayes Estimator of w (posterior mean):

w[0] = -0.8159

w[1] = -0.0026

w[2] = 0.0039

w[3] = 0.0221

w[4] = -0.0006

Bayes Estimator of σ^2 (posterior mean): 180.6160

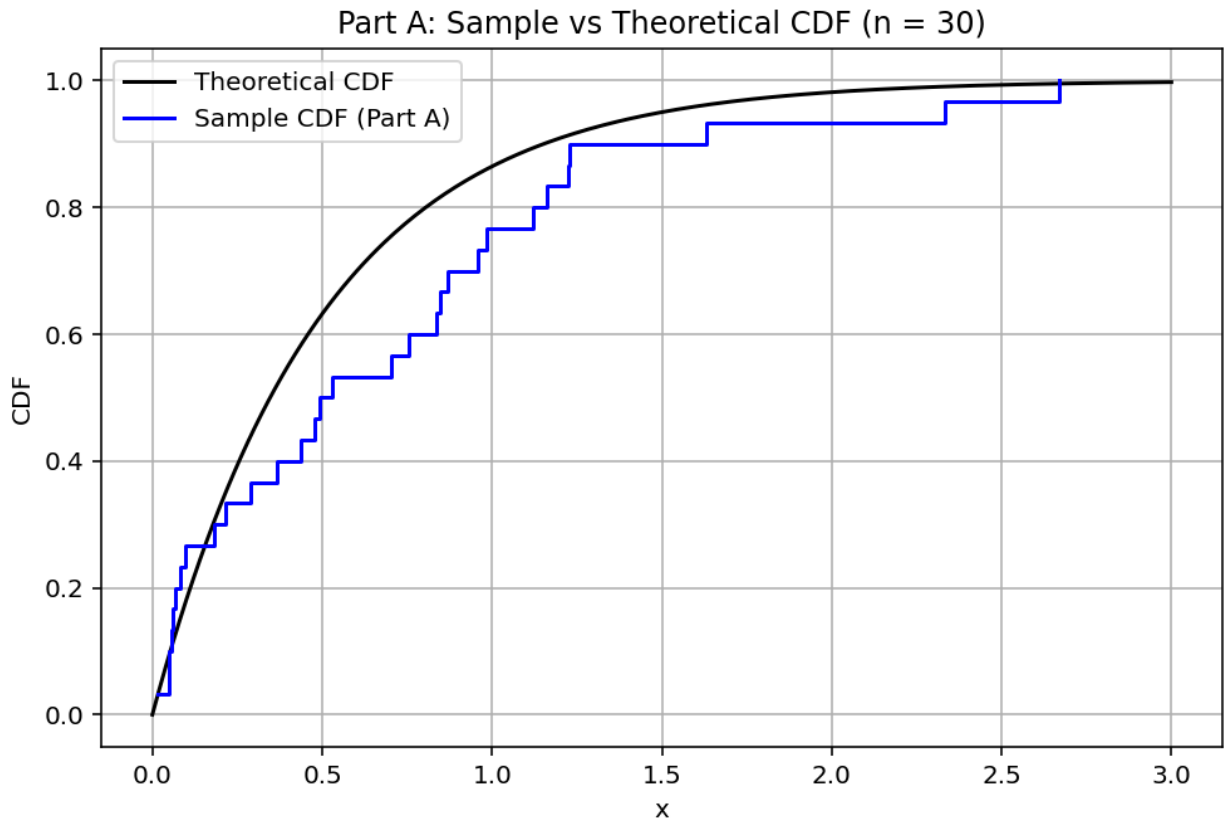
Problem 3

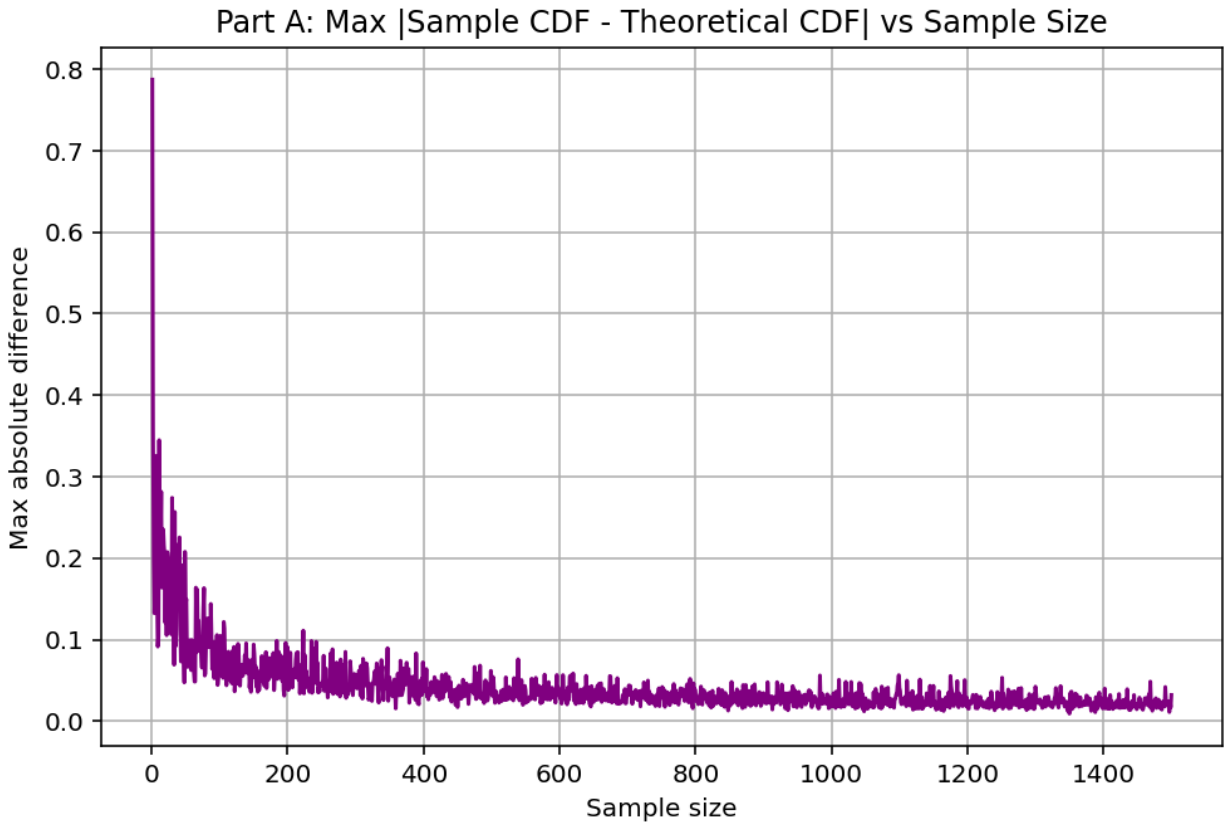
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import expon

rng = np.random.default_rng()
lambda_param = 2
scale_param = 1 / lambda_param

# ------------------------
# Part A – NumPy's .exponential()
# ------------------------

sample_30_a = rng.exponential(scale=scale_param, size=30)
sorted_a = np.sort(sample_30_a)
sample_cdf_a = np.arange(1, 31) / 30
x_vals = np.linspace(0, 3, 300)
theoretical_cdf = expon.cdf(x_vals, scale=scale_param)

plt.figure(figsize=(8, 5))
plt.plot(x_vals, theoretical_cdf, label="Theoretical CDF", color="black")
plt.step(sorted_a, sample_cdf_a, label="Sample CDF (Part A)", color="blue", where="post")
plt.title("Part A: Sample vs Theoretical CDF (n = 30)")
plt.xlabel("x")
plt.ylabel("CDF")
```

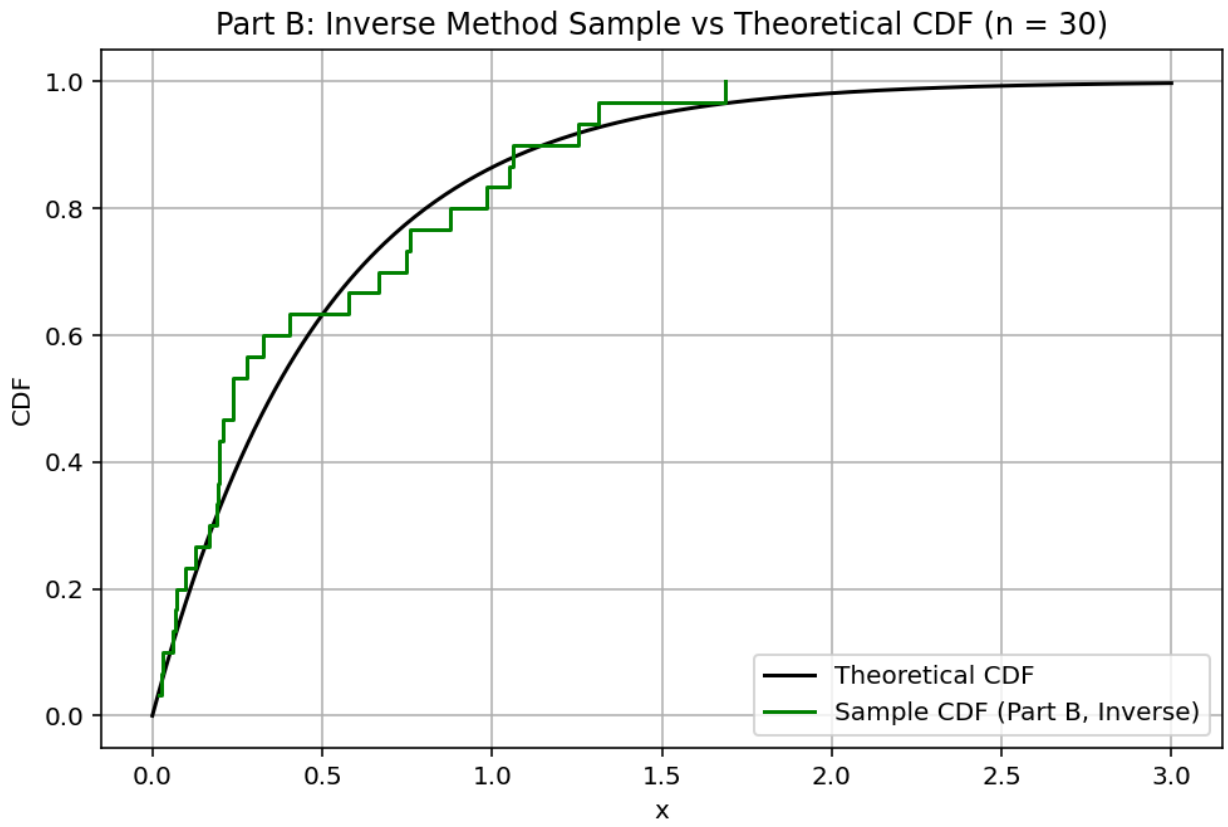
```
plt.legend()
plt.grid(True)
plt.show()

sample_sizes = np.arange(1, 1501)
max_diffs_a = []
for n in sample_sizes:
    s = rng.exponential(scale=scale_param, size=n)
    sorted_s = np.sort(s)
    sample_cdf = np.arange(1, n+1) / n
    true_cdf = expon.cdf(sorted_s, scale=scale_param)
    max_diffs_a.append(np.max(np.abs(sample_cdf - true_cdf)))

plt.figure(figsize=(8, 5))
plt.plot(sample_sizes, max_diffs_a, color="purple")
plt.title("Part A: Max |Sample CDF - Theoretical CDF| vs Sample Size")
plt.xlabel("Sample size")
plt.ylabel("Max absolute difference")
plt.grid(True)
plt.show()



# ------------------------
# Pat B – Manual Inverse CDF Sampling
# ------------------------
```

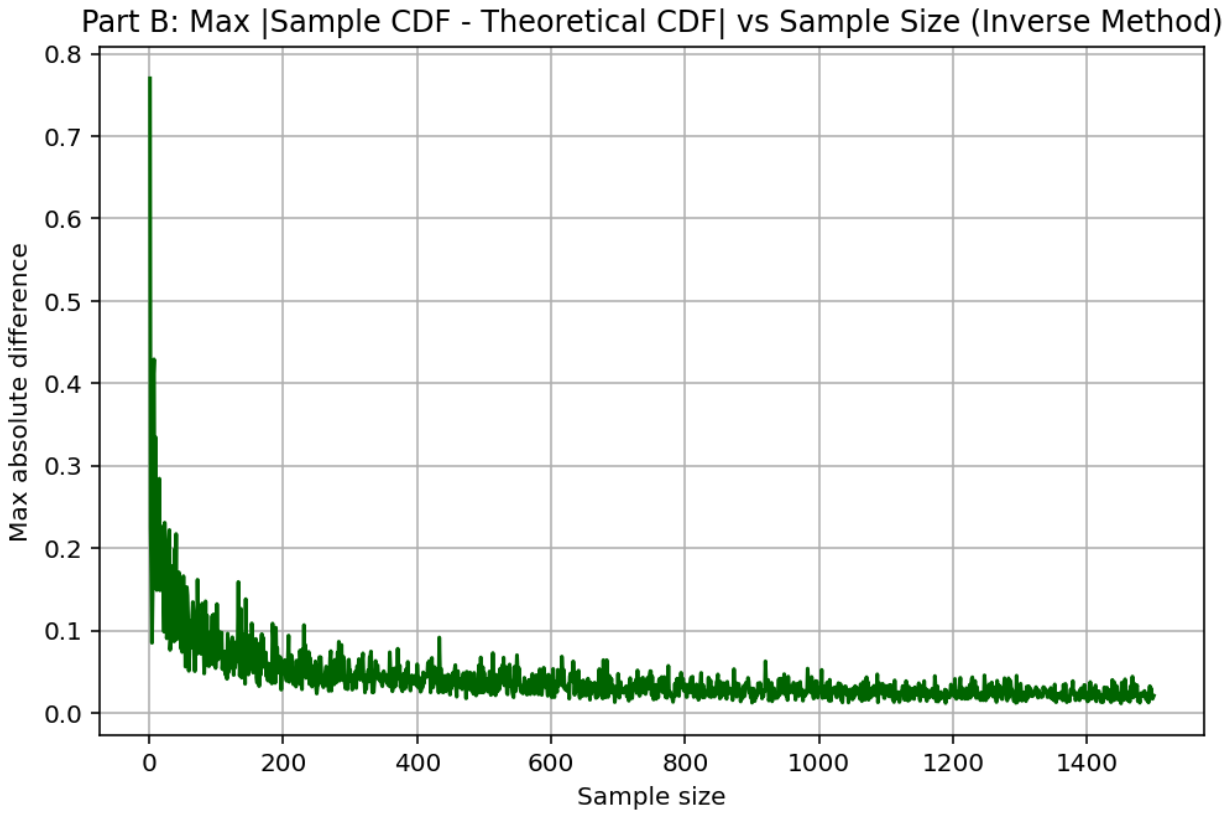
```
def inverse_exponential(u, lam):
    return -np.log(1 - u) / lam

u_30 = rng.random(30)
sample_30_b = inverse_exponential(u_30, lambda_param)
sorted_b = np.sort(sample_30_b)
sample_cdf_b = np.arange(1, 31) / 30

plt.figure(figsize=(8, 5))
plt.plot(x_vals, theoretical_cdf, label="Theoretical CDF", color="black")
plt.step(sorted_b, sample_cdf_b, label="Sample CDF (Part B, Inverse)", color="green",
where="post")
plt.title("Part B: Inverse Method Sample vs Theoretical CDF (n = 30)")
plt.xlabel("x")
plt.ylabel("CDF")
plt.legend()
plt.grid(True)
plt.show()

max_diffs_b = []
for n in sample_sizes:
    u = rng.random(n)
    samp = inverse_exponential(u, lambda_param)
    sorted_s = np.sort(samp)
    sample_cdf = np.arange(1, n+1) / n
    true_cdf = expon.cdf(sorted_s, scale=scale_param)
```

```
    max_diffs_b.append(np.max(np.abs(sample_cdf - true_cdf)))


plt.figure(figsize=(8, 5))
plt.plot(sample_sizes, max_diffs_b, color="darkgreen")
plt.title("Part B: Max |Sample CDF - Theoretical CDF| vs Sample Size (Inverse Method)")
plt.xlabel("Sample size")
plt.ylabel("Max absolute difference")
plt.grid(True)
plt.show()
```




Part A: Max |Sample CDF - Theoretical CDF| vs Sample Size
Max absolute difference
Sample size
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
200
400
600
800
1000
1200
1400


Part B: Inverse Method Sample vs Theoretical CDF (n = 30)
CDF
x
Theoretical CDF
Sample CDF (Part B, Inverse)
0.0
0.2
0.4
0.6
0.8
1.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0



In Part A, 30 values were sampled from an Exponential(0.5) distribution using NumPy's built-in function. The resulting sample cumulative distribution function (CDF) was plotted alongside the theoretical CDF. While the sample curve showed some fluctuation, it generally followed the expected shape.

To examine how accuracy improves with more data, the maximum difference between the sample and theoretical CDFs was measured across sample sizes from 1 to 1500. The difference steadily decreased as the sample size grew, showing that the sample distribution gradually aligned with the true one.

Part B repeated the same process, but instead of using a built-in function, the samples were created manually using the inverse transform method. This approach produced similar results: the sample CDF resembled the theoretical CDF more closely as the sample size increased, and the maximum difference between them also became smaller over time.

Both parts demonstrate that with enough data, the sampled values begin to accurately reflect the behavior of the Exponential(0.5) distribution. This matches expectations based on general principles of probability and sampling.