Question 1

The problem provides us to consider $L \sim N(2, 9)$, to compute the theoretical value at risk and expected shortfall for a given α of L, we use the following formulas:

$$\text{VaR}_\alpha(L) = \mu + \sigma\Phi^{-1}(\alpha) \tag{1.}$$

$$\text{ES}_\alpha(L) = \mu + \sigma\frac{\Phi(\Phi^{-1}(\alpha))}{1-\alpha} \tag{2.}$$

1)

The theoretical value of $\text{VaR}_{0.99(L)}$ using Equation 1 where we are given $\mu = 2$, $\sigma = 3$ and $\alpha = 0.99$:

$$\begin{aligned}\text{VaR}_{0.99}(L) &= 2 + 3\Phi^{-1}(0.99) \\ &= 8.98\end{aligned} \tag{3.}$$

The theorical value of $\text{ES}_{0.99(L)}$ using Equation 2 with the mentioned given values is:

$$\begin{aligned}\text{ES}_{0.99}(L) &= 2 + 3\frac{\Phi(\Phi^{-1}(0.99))}{1-0.99} \\ &= 10.00\end{aligned} \tag{4.}$$

2-3) (a-c)

All of these parts are done at the same time in our Python program, which will have comments to show where each part is completed. The following content is `q1.py`

```python
from scipy.stats import norm
import numpy as np
from scipy.stats import shapiro, kstest
import matplotlib.pyplot as plt

# Given parameters
mu = 2
sigma = 3
alpha = 0.99
# we will loop through all sample sizes provided 2-3)
sample_sizes = [10**6, 10**7, 10**8]

# Theoretical values as calculated in 1)
VaR99_theo = mu + sigma * norm.ppf(alpha)
ES99_theo = mu + sigma * (norm.pdf(norm.ppf(alpha)) / (1 - alpha))

# Function to compute empirical VaR and ES
# As defined in Lecture 5, the empiricial Var
# is defined by the 99th percentile of the sample
# The empirical ES is defined by the average of the values
# greater than or equal to the empirical Var
def empirical_var_es(data, alpha=0.99):
    var = np.percentile(data, alpha * 100)
    es = np.mean(data[data >= var])
    return var, es

# Run simulations
results = {}
for N in sample_sizes:
    np.random.seed(42)  # For reproducibility
    sample = np.random.normal(mu, sigma, N)

    # a) Standardizing the data and Testing for normality
    # Standardize the data
    mean = np.mean(sample)
    std = np.std(sample)
    standardized_sample = (sample - mean) / std

    # Test for normality
    stat1, p_value1 = kstest(standardized_sample, 'norm')
    stat2, p_value2 = shapiro(standardized_sample[:5000])  # Using subset due to Shapiro limit

    is_normal = p_value1 > 0.05 and p_value2 > 0.05
    # Parametric estimates (if normal)
    if is_normal:
        VaR99_param = mean + std * norm.ppf(alpha)
        ES99_param = mean + std * (norm.pdf(norm.ppf(alpha)) / (1 - alpha))
    else:
        VaR99_param = None
        ES99_param = None

    # b) Empirical estimates
    # Empirical estimates
    VaR99_emp, ES99_emp = empirical_var_es(sample, alpha)

    # c) Compute the absolute errors and compare with true values in the results
    # Compute absolute errors
    abs_error_parametric = (abs(VaR99_param - VaR99_theo), abs(ES99_param - ES99_theo)) if is_normal
else (None, None)
    abs_error_empirical = (abs(VaR99_emp - VaR99_theo), abs(ES99_emp - ES99_theo))

    # Store results
    results[N] = {
        "data": sample,
        "VaR99_param": VaR99_param,
        "ES99_param": ES99_param,
        "VaR99_emp": VaR99_emp,
        "ES99_emp": ES99_emp,
        "abs_error_parametric": abs_error_parametric,
        "abs_error_empirical": abs_error_empirical,
        "is_normal": is_normal
    }

# Print results
for N, res in results.items():
    print(f"N = {N}")
    print(f"  Normality confirmed: {res['is_normal']}")
    print(f"  VaR_99 Parametric: {res['VaR99_param']}, Abs Error: {res['abs_error_parametric'][0]}")
    print(f"  ES_99 Parametric: {res['ES99_param']}, Abs Error: {res['abs_error_parametric'][1]}")
    print(f"  VaR_99 Empirical: {res['VaR99_emp']}, Abs Error: {res['abs_error_empirical'][0]}")
    print(f"  ES_99 Empirical: {res['ES99_emp']}, Abs Error: {res['abs_error_empirical'][1]}")
    print("-")

    # Normality check with histograms
    sample = results[N]['data']
    # Create the plot
    plt.figure(figsize=(8, 4))
    plt.hist(sample, bins=30, density=True, alpha=0.6, color='skyblue')
    plt.title(f'Normality Check for N={N}')
    plt.xlabel('Value')
    plt.ylabel('Density')
    plt.grid(True)
    plt.tight_layout()
    # Save the plot first
    plt.savefig(f'normality_check_{N}.png', dpi=300, bbox_inches='tight')
    # Show the plot
    plt.show()
    plt.close()


# Create figure with subplots
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 12))
# Plot 1: VaR Comparison
var_empirical = [results[N]['VaR99_emp'] for N in sample_sizes]
var_parametric = [results[N]['VaR99_param'] for N in sample_sizes]
var_theoretical = [VaR99_theo] * len(sample_sizes)
x_pos = np.arange(len(sample_sizes))
width = 0.25
ax1.bar(x_pos - width, var_empirical, width, label='Empirical VaR', color='skyblue')
ax1.bar(x_pos, var_parametric, width, label='Parametric VaR', color='lightcoral')
ax1.bar(x_pos + width, var_theoretical, width, label='Theoretical VaR', color='lightgreen')
```

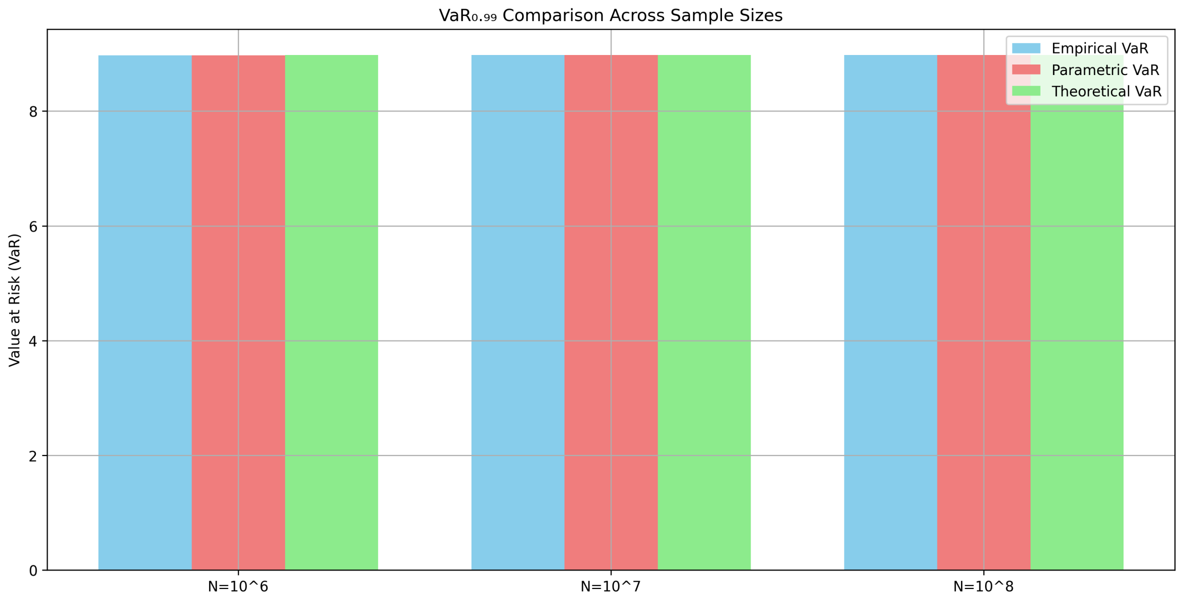
```python
ax1.set_xticks(x_pos)
ax1.set_xticklabels([f'N=10^{int(np.log10(N))}' for N in sample_sizes])
ax1.set_title('VaR₀.₉₉ Comparison Across Sample Sizes')
ax1.set_ylabel('Value at Risk (VaR)')
ax1.legend()
ax1.grid(True)

# Plot 2: Absolute Errors
# This will help us visualize the effect on absolute errors as sample size increases
var_errors_empirical = [results[N]['abs_error_empirical'][0] for N in sample_sizes]
var_errors_parametric = [results[N]['abs_error_parametric'][0] for N in sample_sizes]
es_errors_empirical = [results[N]['abs_error_empirical'][1] for N in sample_sizes]
es_errors_parametric = [results[N]['abs_error_parametric'][1] for N in sample_sizes]

x_pos = np.arange(len(sample_sizes))
width = 0.2

ax2.bar(x_pos - 1.5*width, var_errors_empirical, width, label='VaR Empirical Error', color='skyblue')
ax2.bar(x_pos - 0.5*width, var_errors_parametric, width, label='VaR Parametric Error',
color='lightcoral')
ax2.bar(x_pos + 0.5*width, es_errors_empirical, width, label='ES Empirical Error', color='lightgreen')
ax2.bar(x_pos + 1.5*width, es_errors_parametric, width, label='ES Parametric Error', color='lightpink')
ax2.set_xticks(x_pos)
ax2.set_xticklabels([f'N=10^{int(np.log10(N))}' for N in sample_sizes])
ax2.set_title('Absolute Errors Comparison')
ax2.set_ylabel('Absolute Error')
ax2.legend()
ax2.grid(True)
ax2.set_yscale('log')  # Using log scale to better show error differences
plt.tight_layout()
# Save the figure
filename = 'var_es_comparison.png'
plt.savefig(filename, dpi=300, bbox_inches='tight')
print(f"Plot saved as: {filename}")
```

To get the results, we can run the program on the terminal:

Terminal

```
$ python3 q1.py
N = 1000000
  Normality confirmed: True
  VaR_99 Parametric: 8.975556669822536, Abs Error: 0.003486952299985191
  ES_99 Parametric: 9.992346866772403, Abs Error: 0.0032957942650142513
  VaR_99 Empirical: 8.972412832156806, Abs Error: 0.006630789965715422
  ES_99 Empirical: 9.982018640953951, Abs Error: 0.013624020083465638
-
N = 10000000
  Normality confirmed: True
  VaR_99 Parametric: 8.978946535236147, Abs Error: 9.708688637388718e-05
  ES_99 Parametric: 9.995559381058833, Abs Error: 8.327997858437186e-05
  VaR_99 Empirical: 8.977836858606143, Abs Error: 0.0012067635163788282
  ES_99 Empirical: 9.99396136390425, Abs Error: 0.0016812971331674476
-
N = 100000000
  Normality confirmed: True
  VaR_99 Parametric: 8.979114004714269, Abs Error: 7.038259174763084e-05
  ES_99 Parametric: 9.995786749743548, Abs Error: 0.0001440887061310292
  VaR_99 Empirical: 8.978653808056245, Abs Error: 0.0003898140662759175
  ES_99 Empirical: 9.995781538215402, Abs Error: 0.00013887717798510835
-
Plot saved as: var_es_comparison.png
```

To better visualize the results, and how the error changes across sample sizes increasing, consider the following diagram:



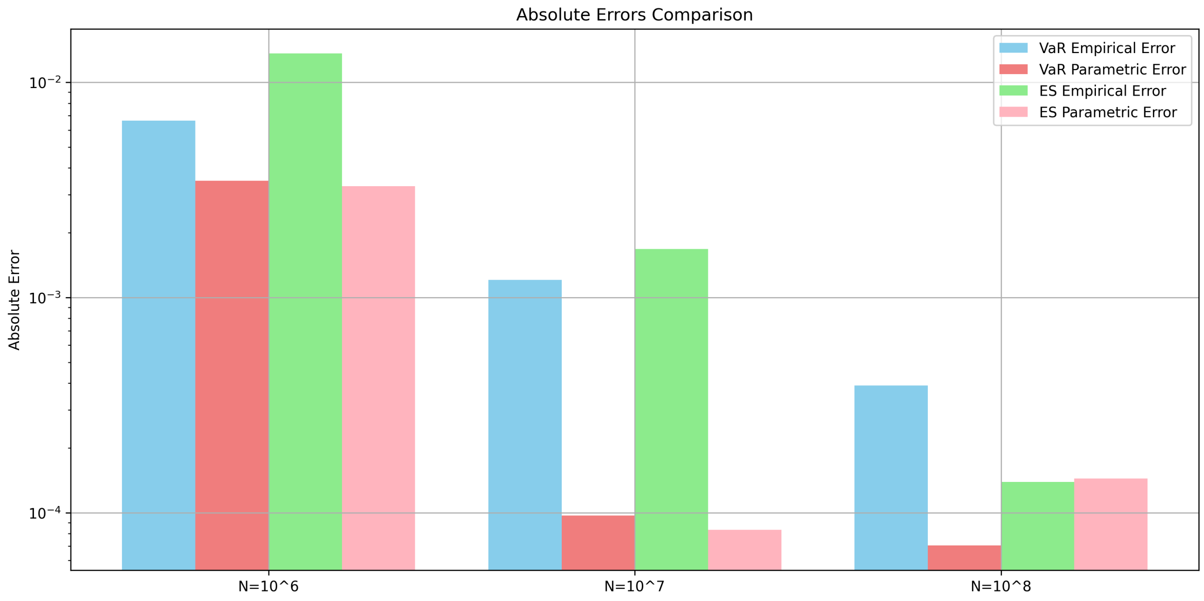


Figure 1: VaR and ES Error Comparison across Sample Sizes

We are able to see in bottom diagram that as the sample sizes increases all errors are decreasing, thus becoming more accurate. We can also notice that for all sizes, the least accurate measurement is the empirical calculation.