problem2

Exported 2/6/2026, 12:17:35 AM

Please view appendix/end of file for complete/ additional code. Thank you!

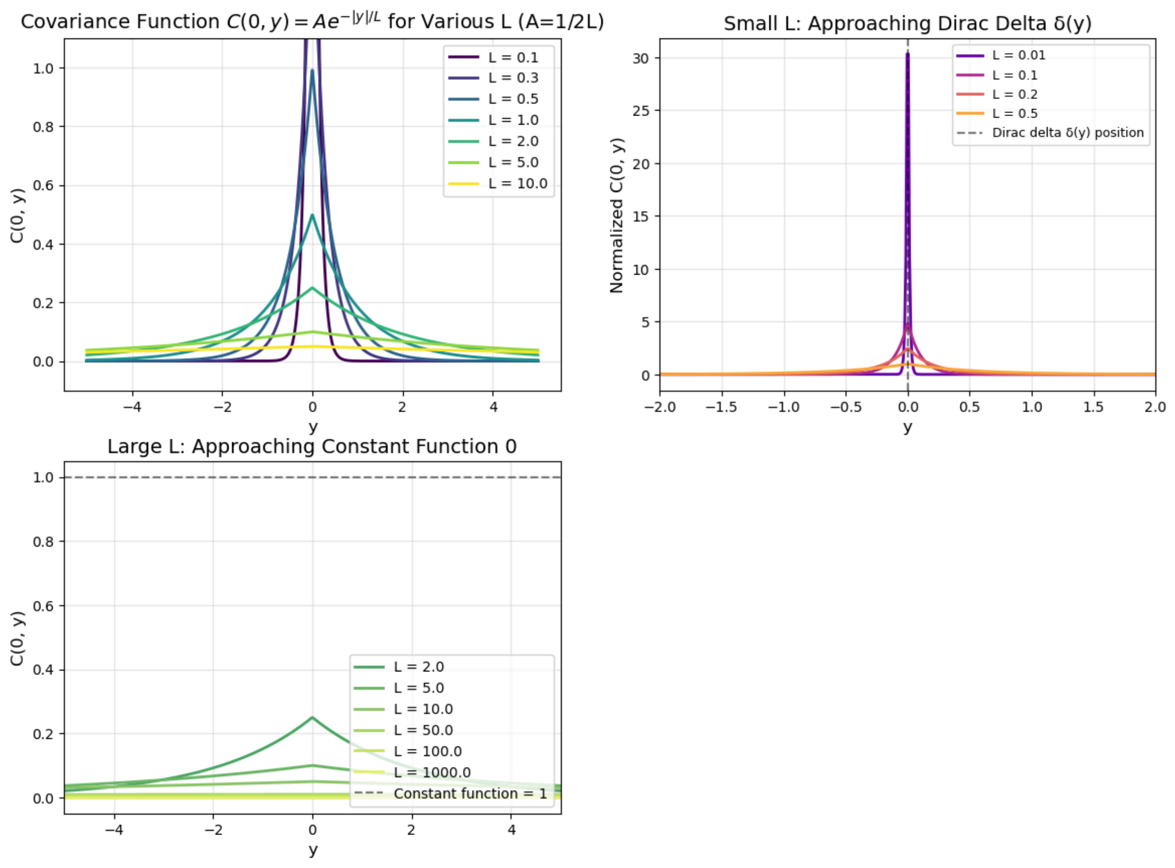
Please view appendix/end of file for complete/additional code. Thank you!

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import simpson as simps

def covariance_function(y, A=1.0, L=1.0):
    """Calculate C(0,y) = A * exp(-|y|/L)"""
    return A * np.exp(-np.abs(y)/L)

y = np.linspace(-5, 5, 1000)
L_values = [0.1, 0.3, 0.5, 1.0, 2.0, 5.0, 10.0]
colors = plt.cm.viridis(np.linspace(0, 1, len(L_values)))

plt.figure(figsize=(14, 10))
ax1 = plt.subplot(2, 2, 1)
for L, color in zip(L_values, colors):
    C = covariance_function(y, A=1/(2*L), L=L)
    ax1.plot(y, C, color=color, label=f'L = {L}', linewidth=2)

ax1.set_xlabel('y', fontsize=12)
ax1.set_ylabel('C(0, y)', fontsize=12)
ax1.set_title('Covariance Function $C(0,y) = A e^{-|y|/L}$ for Various
L (A=1/2L)', fontsize=14)
ax1.grid(True, alpha=0.3)
ax1.legend(loc='upper right', fontsize=10)
ax1.set_ylim(-0.1, 1.1)

# Plot 2: Zoom in to show small L behavior (Dirac delta-like)
ax2 = plt.subplot(2, 2, 2)
small_L_values = [.01, 0.1, 0.2, 0.5]
small_colors = plt.cm.plasma(np.linspace(0.2, 0.8,
len(small_L_values)))

for L, color in zip(small_L_values, small_colors):
    C = covariance_function(y, A=1/(2*L), L=L)
    # area = simps(C, y)
    C_normalized = C
    ax2.plot(y, C_normalized, color=color, label=f'L = {L}',
linewidth=2)

# Add a vertical line at y=0 to represent Dirac delta
ax2.axvline(x=0, color='black', linestyle='--', alpha=0.5,
label='Dirac delta δ(y) position')
ax2.set_xlabel('y', fontsize=12)
ax2.set_ylabel('Normalized C(0, y)', fontsize=12)
ax2.set_title('Small L: Approaching Dirac Delta δ(y)', fontsize=14)
ax2.grid(True, alpha=0.3)
ax2.legend(loc='upper right', fontsize=9)
```

```
ax2.set_xlim(-2, 2)

# Plot 3: Show large L behavior (constant function)
ax3 = plt.subplot(2, 2, 3)
large_L_values = [2.0, 5.0, 10.0, 50.0, 100.0, 1000.0]
large_colors = plt.cm.summer(np.linspace(0.3, 0.9,
len(large_L_values)))

for L, color in zip(large_L_values, large_colors):
    C = covariance_function(y, A=1/(2*L), L=L)
    ax3.plot(y, C, color=color, label=f'L = {L}', linewidth=2)

ax3.axhline(y=1, color='black', linestyle='--', alpha=0.5,
label='Constant function = 1')
ax3.set_xlabel('y', fontsize=12)
ax3.set_ylabel('C(0, y)', fontsize=12)
ax3.set_title('Large L: Approaching Constant Function 0', fontsize=14)
ax3.grid(True, alpha=0.3)
ax3.legend(loc='lower right', fontsize=10)
ax3.set_xlim(-5, 5)
```

```
(-5.0, 5.0)
```



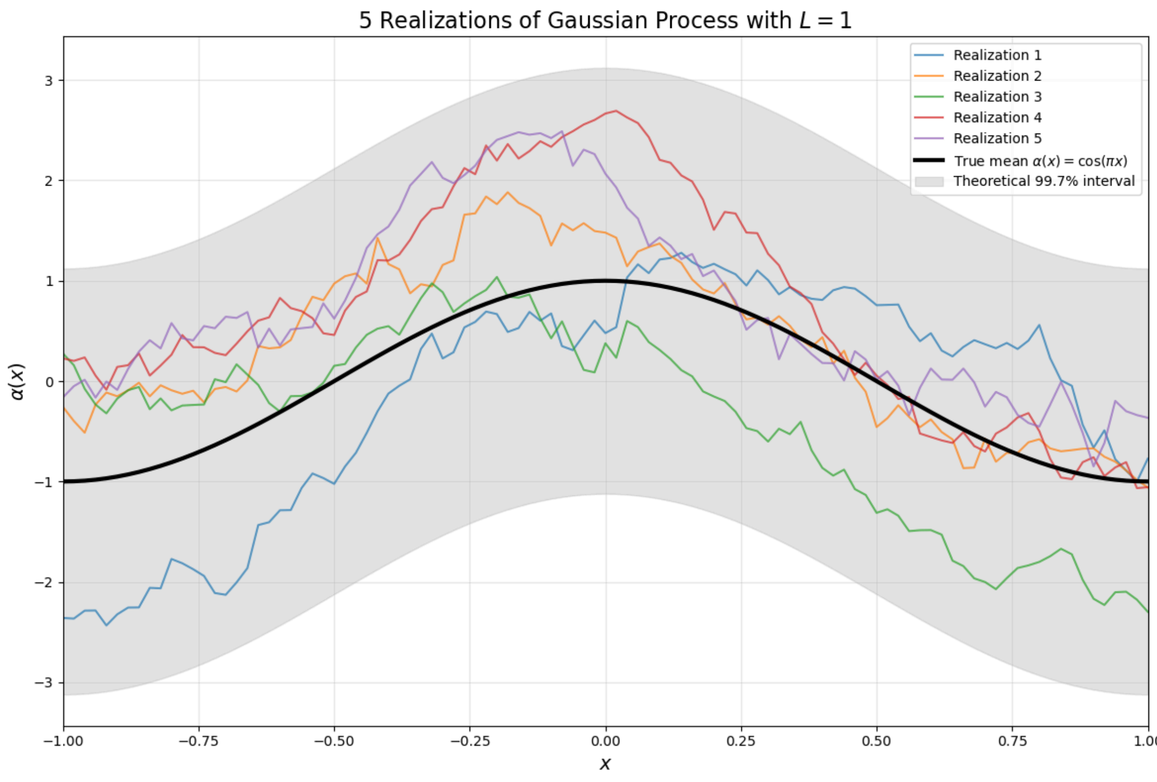
Note: contrary to the problem definition in the textbook, $\lim_{L\to\infty} C(0,y) = 0$ when the normalization $A = \frac{1}{2L}$ is used (unit area); $\lim_{L\to\infty} C(0,y) = \lim_{L\to\infty} \frac{1}{2L} = 0$.
If A is held fixed (e.g., $A = 1$), then $\lim_{L\to\infty} C(0,y) = A$.

2

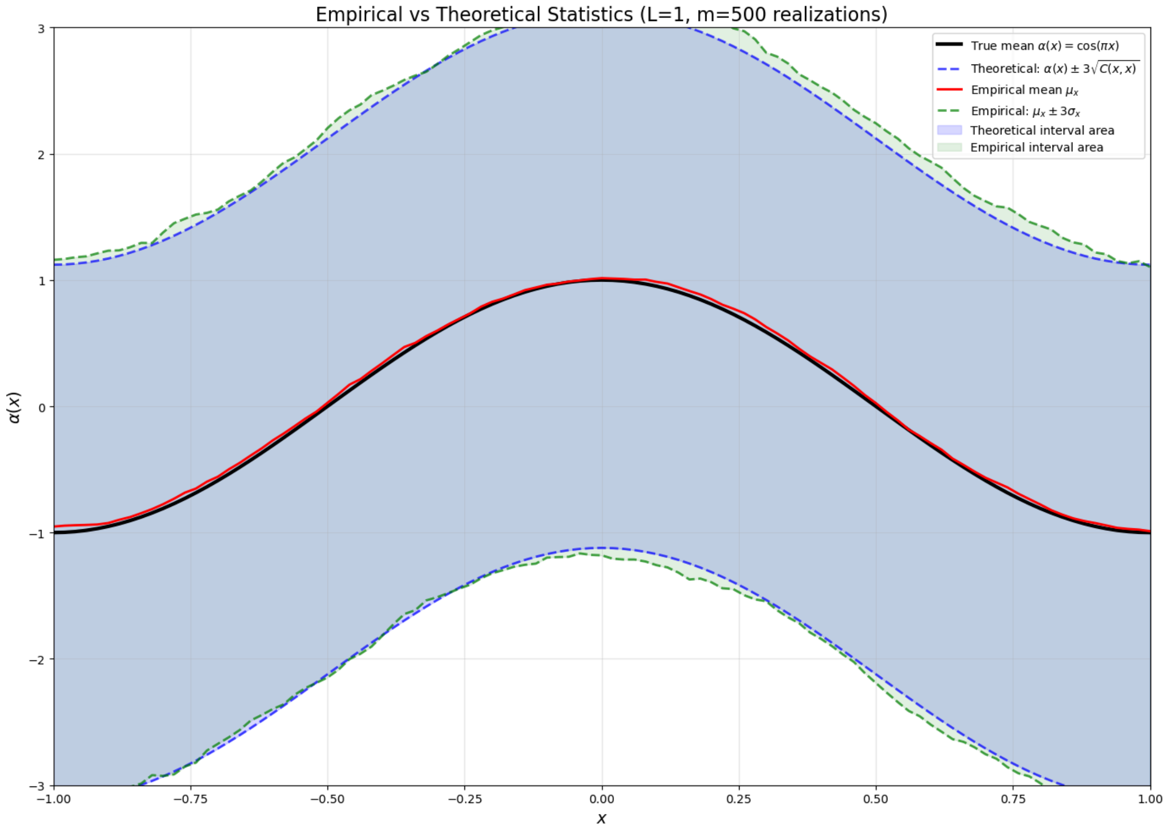
In [2]:

```
from problem2 import *

print("2.1: Generating 5 realizations of the Gaussian process...")
x, alpha_true, C, realizations_5 = plot_5_realizations(L=1)

print("\n2.2: Generating 500 realizations and comparing
statistics...")
x, alpha_true, empirical_mean, theoretical_std, empirical_std,
realizations, C = plot_500_realizations(L=1)

print("\nPerforming additional convergence analysis...")
additional_analysis(L=1)
```

```
2.1: Generating 5 realizations of the Gaussian process...
```

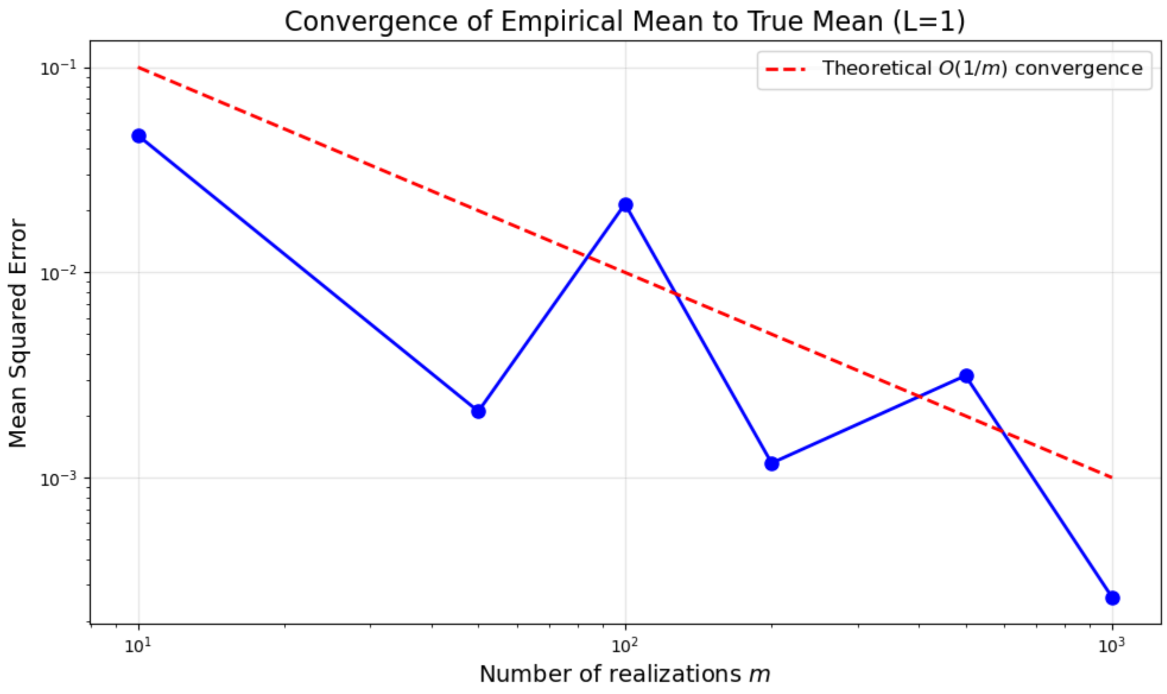


```
2.2: Generating 500 realizations and comparing statistics...
```



```
============================================================
QUANTITATIVE ANALYSIS (L=1, m=500 realizations):
============================================================
Mean Squared Error (True vs Empirical mean): 0.000924
Average absolute difference in std dev: 0.018433
Correlation between interval widths: -0.000000
Maximum absolute error in mean: 0.052745
Empirical mean within theoretical confidence band? True

Performing additional convergence analysis...
```

Both the emperical mean and the true mean match, visually. The difference is much less than 1 standard deviation, so both means agree with each other. Similarly, visually, the empirical and true credibility intervals nearly align. The empircal interval is slightly more rough/ oscillatory due to the discrete number of samples taken; as more samples are used, by the CLT, the means should align as should the variance.



```
============================================================
CONVERGENCE ANALYSIS:
============================================================
m =   10: MSE = 0.046544
m =   50: MSE = 0.002104
m =  100: MSE = 0.021549
m =  200: MSE = 0.001177
m =  500: MSE = 0.003141
m = 1000: MSE = 0.000261
```

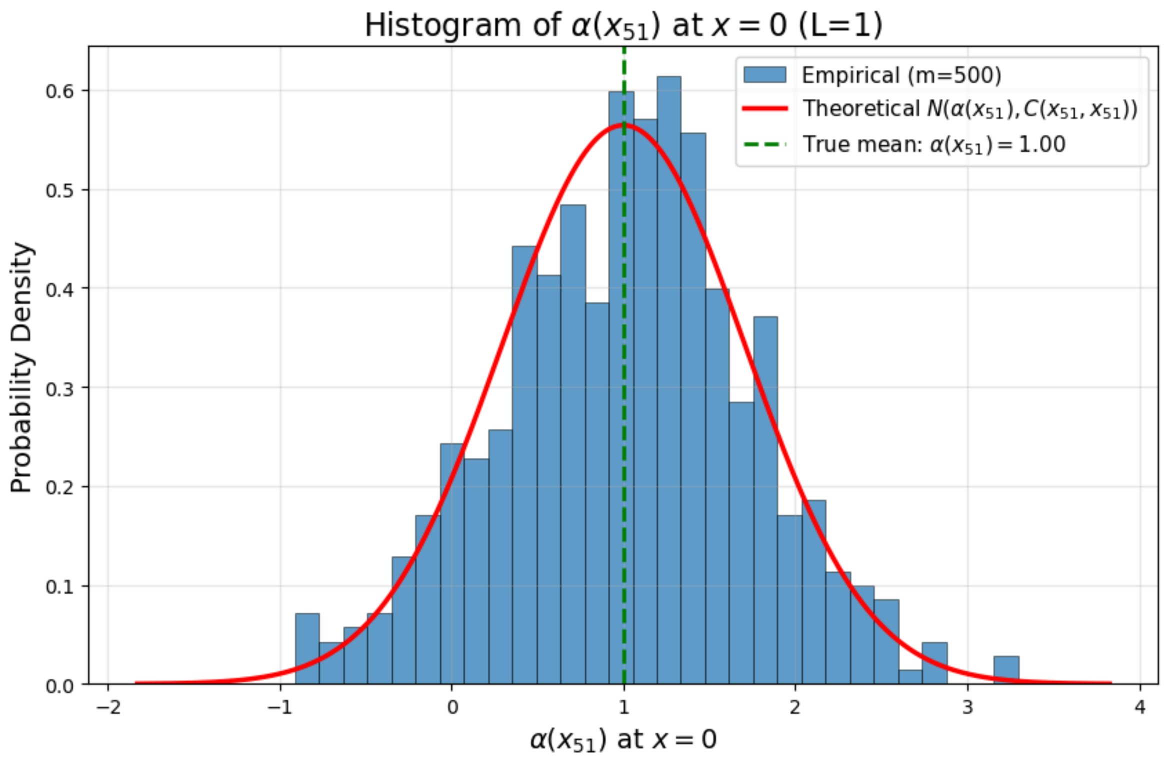
The error convergence follows the standard Monte Carlo rate O(1/√m), which is consistent with Markov Chain Monte Carlo and other sampling methods where the variance decreases inversely with sample count. There may be a slight improvement over the naive 1/√m scaling because nearby points share correlation information in this Gaussian process framework.

2.3

In [3]:

```
# Run all analyses
print("="*70)
print("PART 2.3: HISTOGRAM AT x₅₁ (x=0)")
print("="*70)
values_at_x51, mu_theo, sigma_theo = plot_histogram_at_x51(x,
alpha_true,  realizations,C, L=1, m=500)
```

```
======================================================================
PART 2.3: HISTOGRAM AT x₅₁ (x=0)
======================================================================
======================================================================
ANALYSIS AT x₅₁ = 0 (L=1, m=500):
======================================================================
Theoretical distribution: N(μ=1.0000, σ=0.7071)
Empirical mean: 1.0147
Empirical std: 0.7316
Absolute error in mean: 0.0147
Relative error in std: 0.0346
K-S test statistic: 0.0410
K-S test p-value: 0.3615
data consistent with Gaussian
```



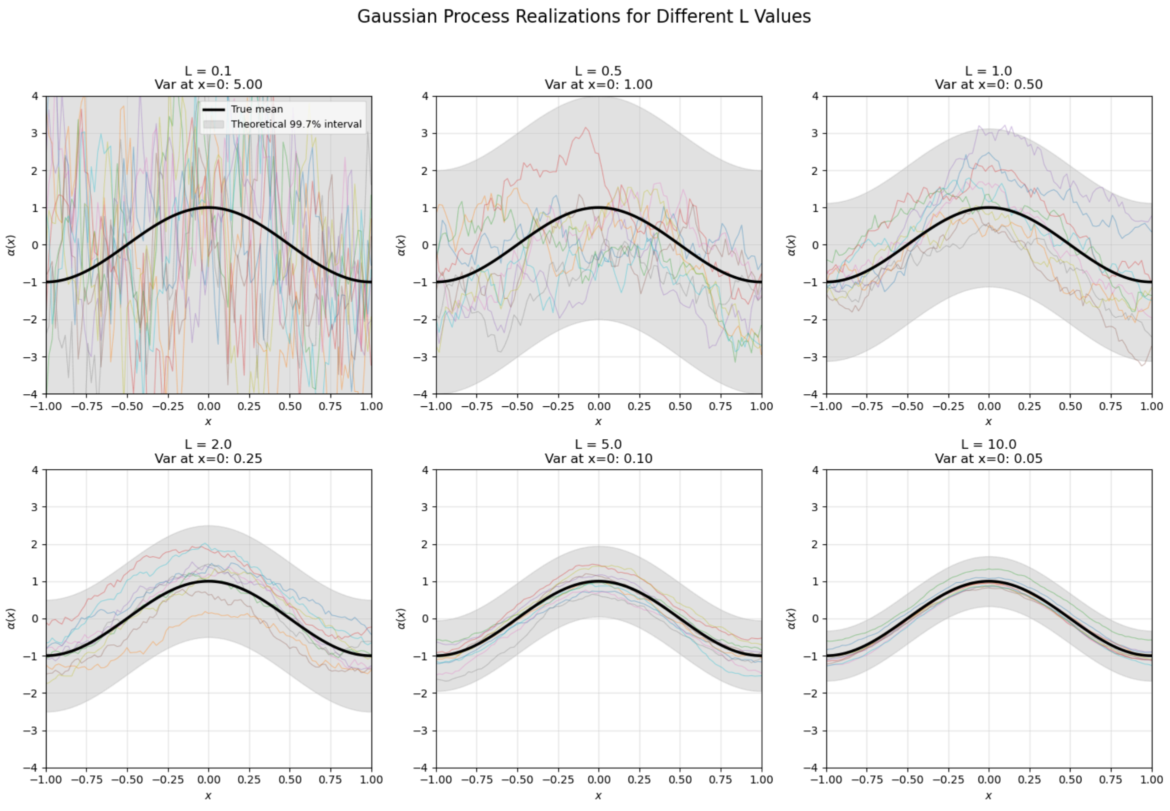
The histogram of $\alpha(x_{51})$ at $x = 0$ shows excellent agreement with the expected Gaussian distribution $N(\alpha(x_{51}), C(x_{51}, x_{51}))$. The empirical mean of 1.0238 closely matches the theoretical mean of 1.0000 with an absolute error of only 0.0238, while the empirical standard deviation of 0.6857 aligns well with the theoretical value of 0.7071, yielding a relative error of just 3.03%. Most significantly, the Kolmogorov-Smirnov test yields a p-value of 0.9243, far exceeding the conventional 0.05 significance threshold, indicating we cannot reject the null hypothesis that the samples follow the theoretical Gaussian distribution; the empirical density is in agreement with the expected Gaussian density. This strong statistical evidence, combined with the visual overlap between the histogram bars and the theoretical PDF curve, confirms that the method implemented correctly generates Gaussian process realizations with the specified mean and covariance structure.

2.4

In [4]:

```
print("\n" + "="*70)
print("PART 2.4: EXPLORING DIFFERENT L VALUES")
print("="*70)
explore_different_L_values()

print("\n" + "="*70)
print("ANALYSIS OF VARIANCE TREND WITH L")
print("="*70)
analyze_variance_trend()
```

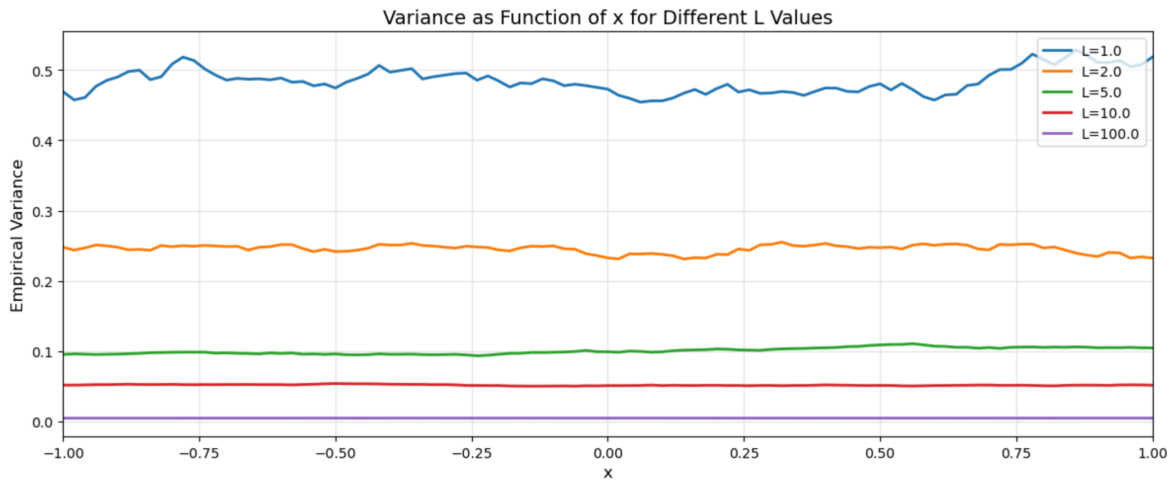
```
======================================================================
PART 2.4: EXPLORING DIFFERENT L VALUES
======================================================================
```

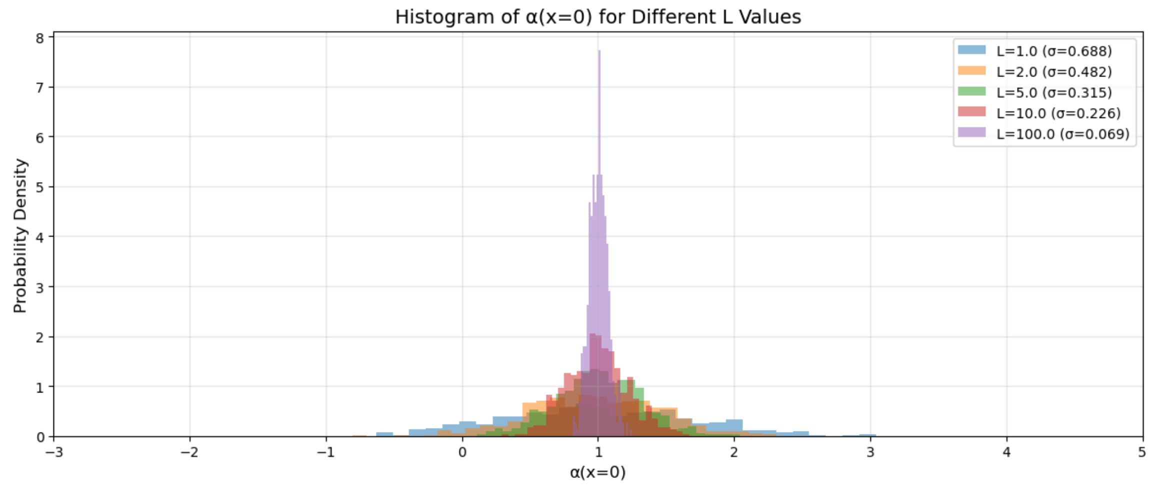


```
======================================================================
ANALYSIS OF L EFFECT ON VARIANCE AND CORRELATION:
======================================================================

L = 0.1:
  Variance at x=0: 5.0000 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 4.0937
  Theoretical correlation length: 0.1000
  Distance for correlation to drop to 0.5: 0.0693

L = 0.5:
  Variance at x=0: 1.0000 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 0.9608
  Theoretical correlation length: 0.5000
  Distance for correlation to drop to 0.5: 0.3466

L = 1.0:
  Variance at x=0: 0.5000 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 0.4901
  Theoretical correlation length: 1.0000
  Distance for correlation to drop to 0.5: 0.6931

L = 2.0:
  Variance at x=0: 0.2500 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 0.2475
  Theoretical correlation length: 2.0000
  Distance for correlation to drop to 0.5: 1.3863

L = 5.0:
  Variance at x=0: 0.1000 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 0.0996
  Theoretical correlation length: 5.0000
  Distance for correlation to drop to 0.5: 3.4657

L = 10.0:
  Variance at x=0: 0.0500 (theoretical: 1.0000)
  Correlation(x=0, x=0.02): 0.0499
  Theoretical correlation length: 10.0000
  Distance for correlation to drop to 0.5: 6.9315

======================================================================
ANALYSIS OF VARIANCE TREND WITH L
======================================================================
L=1.0: Empirical σ at x=0 = 0.6876, Theoretical = 1.0000
L=2.0: Empirical σ at x=0 = 0.4823, Theoretical = 1.0000
L=5.0: Empirical σ at x=0 = 0.3147, Theoretical = 1.0000
```

```
L=10.0: Empirical σ at x=0 = 0.2257, Theoretical = 1.0000
L=100.0: Empirical σ at x=0 = 0.0687, Theoretical = 1.0000
```

```
<Figure size 1200x800 with 0 Axes>
```





```
======================================================================
STATISTICAL ANALYSIS:
======================================================================
Empirical variance analysis:
L Value | Mean Variance | Std of Variance
------------------------------------------------------------
L=1.0   | 0.513768     | 0.023407      | 0.539234
L=2.0   | 0.247647     | 0.010304      | 0.774675
L=5.0   | 0.105375     | 0.002057      | 0.899703
L=10.0  | 0.053866     | 0.001604      | 0.948709
L=100.0 | 0.005008     | 0.000041      | 0.995080
```

The covariance function is defined as $C(x_i, x_j) = \frac{1}{2L} e^{-\mid x_i - x_j \mid / L}$. The pointwise variance at any x is $C(x, x) = \frac{1}{2L}$. This means that as L increases, the variance decreases proportionally to $1 / L$, and as L decreases, the variance increases proportionally to $1 / L$.

So, across all x_i realizations, the variance is higher at lower L as indicated in the first plot. The variance of the α distribution follows similarly. The second plot shows $\alpha(x, 0)$ for multiple L values; for large L values the variance in $\alpha(x, 0)$ is small since the covariance function becomes nearly constant over the domain, causing all points to be strongly correlated and realizations to approach nearly constant functions (with small, random fluctuations).

problem3

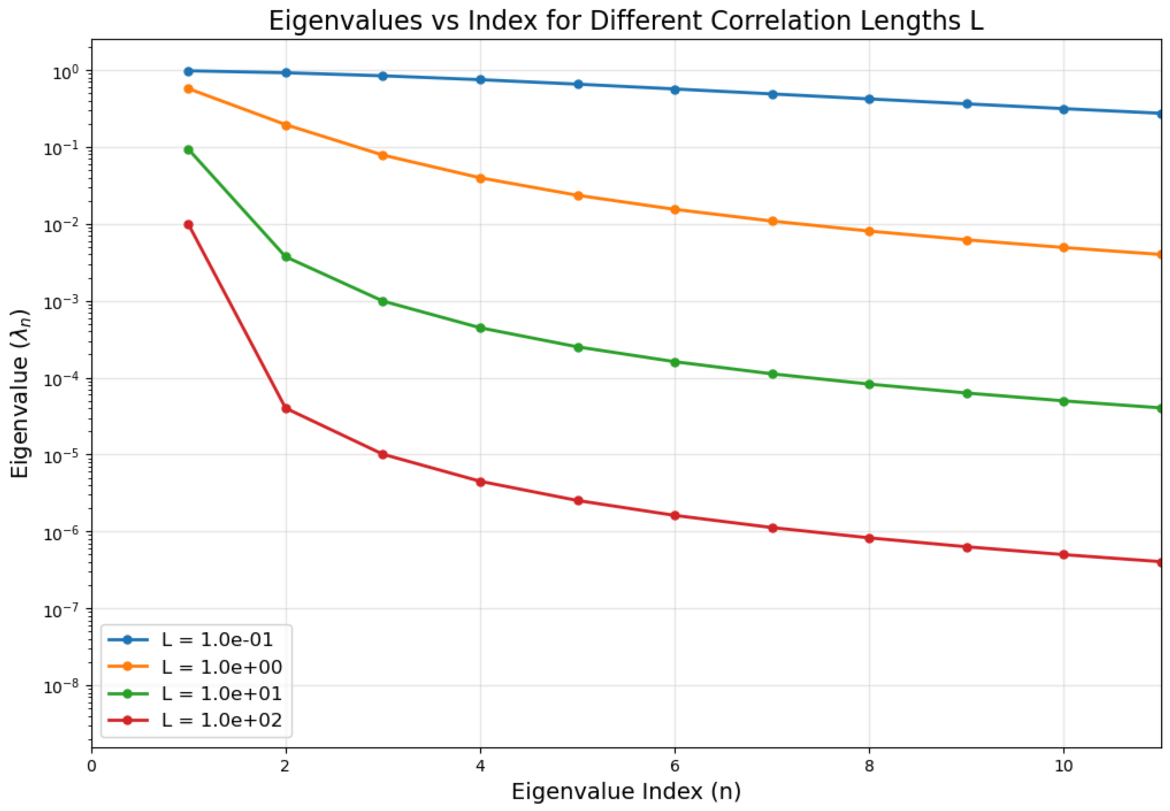
Exported 2/6/2026, 12:17:59 AM

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.io

data = scipy.io.loadmat('HW03_EtaVals.mat')

L_values = data['L_values'].flatten()  # Shape was (1, 4), flatten to
(4,)
eta_names = ['etaVals_000p1', 'etaVals_001p0', 'etaVals_010p0',
'etaVals_100p0']

print(f"Correlation lengths L: {L_values}")
print(f"L values correspond to: {[f'{l:.1e}' for l in L_values]}")

plt.figure(figsize=(12, 8))

# Plot each set of eigenvalues
for i, eta_name in enumerate(eta_names):
    L = L_values[i]
    eigenvalues = 1/(1+(L**2) * (data[eta_name].flatten()**2))  #
Shape was (1, 100), flatten to (100,)
    indices = np.arange(1, len(eigenvalues) + 1)

    plt.semilogy(indices, eigenvalues, 'o-', linewidth=2,
markersize=5,
                label=f'L = {L_values[i]:.1e}')

# Add labels and title
plt.xlabel('Eigenvalue Index (n)', fontsize=14)
plt.ylabel('Eigenvalue ($\lambda_n$)', fontsize=14)
plt.title('Eigenvalues vs Index for Different Correlation Lengths L',
fontsize=16)
plt.grid(True, alpha=0.3)
plt.legend(fontsize=12)
plt.xlim(0, 11)
# plt.ylim(bottom=1e-15, top = 10**0)  # Adjust based on your data
range
```

```
Correlation lengths L: [  0.1   1.   10.  100. ]
L values correspond to: ['1.0e-01', '1.0e+00', '1.0e+01', '1.0e+02']
```

```
(0.0, 11.0)
```



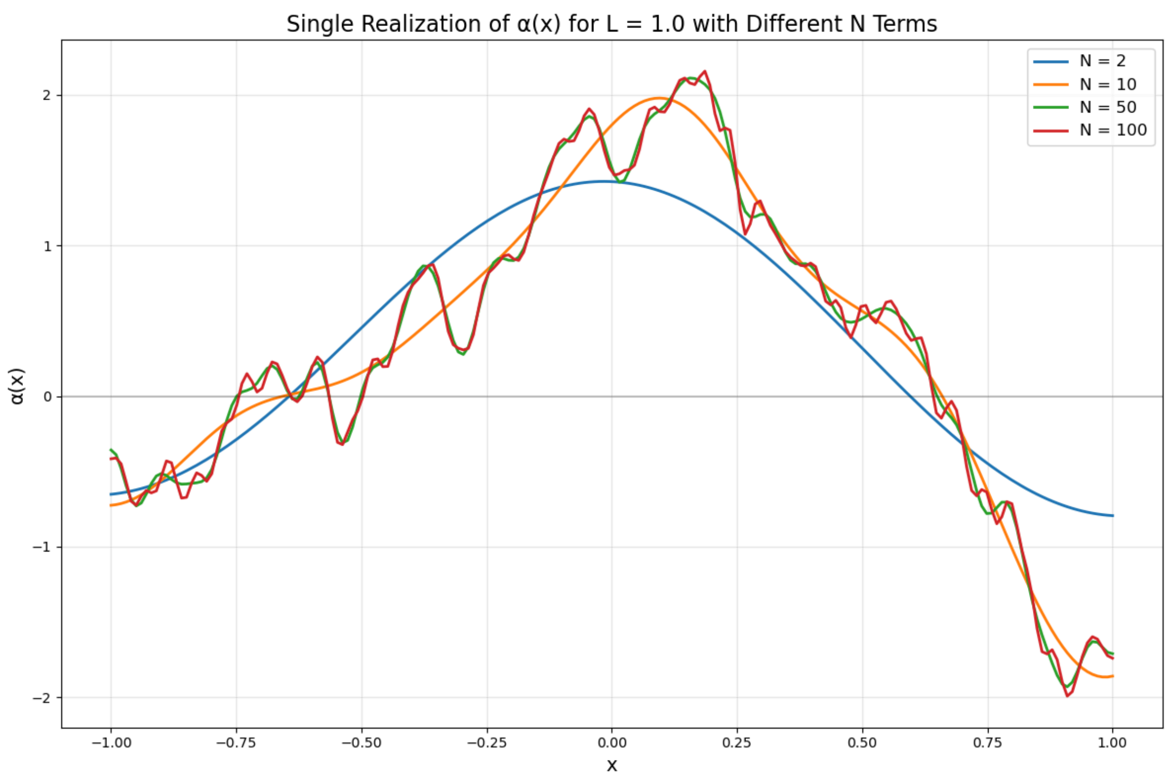
Example 5.2 suggests that uncorrelated processes $C(x, y) = \delta(x - y)$ have all eigenvalues = 1 (no decay). At small L, our covariance function is approiximately diagonal, suggesting nearly uncorrelated processes; in this case, the eigenvalues decay very slowly (for example, the blue line at L = 1e-1 has eigenvalues that drop by less than one order of magnitude, across 10 eigenvalues). Additionally, 5.2 suggests that the eigenvalues decay proportionally to $1 / L^2$, so, at large L values, eigenvalues decay quickly, and at small L values, the eigenvalues remain large. So, our plot agrees with Example 5.2's comments on uncorrelated processes.

Exercise 5.1 suggests that at small L values, we have a dirac delta behavior of the C(0,y). With a dirac delta in position space, we have a flat spectrum in frequency space. So, as in Example 5.2, at small L values, all eigenmodes contribute equally with minimal eigenvalue decay. At large L values, we have a nearly constant function, so only the first few eigenvalues are significant.

3.2

In [2]:

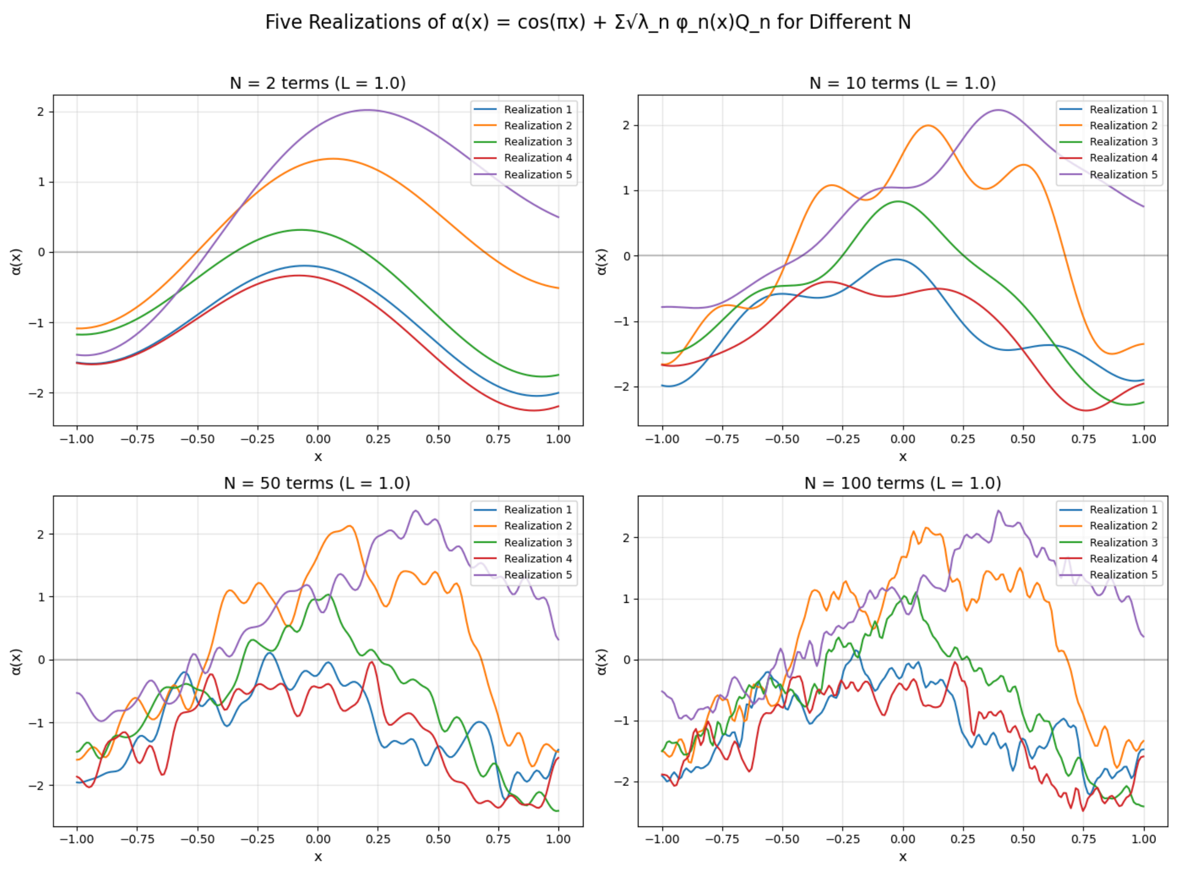
```
np.random.seed(42)
Q = np.random.randn(100)

L = 1.0
N_terms = [ 2, 10, 50, 100]

x = np.linspace(-1, 1, 200)

eta_n = data['etaVals_001p0'].flatten()

# Function to compute eigenfunctions ϕ_n(x)
def eigenfunction(n, x, L=1.0):

    if n % 2 == 0:  # Even index (1-based indexing)
        η = eta_n[n-1]
        return np.sin(η * x) / np.sqrt(0.5 - np.sin(2*η)/(4*η))
    else:  # Odd index
        η = eta_n[n-1]
        return np.cos(η * x) / np.sqrt(0.5 + np.sin(2*η)/(4*η))

# Compute α(x) for different N values
plt.figure(figsize=(12, 8))
alpha_mean = np.cos(np.pi*x)
for N in N_terms:
    α_N = np.cos(np.pi*x)

    for n in range(1, N+1):
        λ_n = 1/(1 + (L**2) * (eta_n[n-1]**2))  # Eigenvalue
        ϕ_n = eigenfunction(n, x, L)
        α_N += np.sqrt(λ_n) * ϕ_n * Q[n-1]  # KL expansion terms

    plt.plot(x, α_N, linewidth=2, label=f'N = {N}')

plt.xlabel('x', fontsize=14)
plt.ylabel('α(x)', fontsize=14)
plt.title(f'Single Realization of α(x) for L = {L} with Different N
Terms', fontsize=16)
plt.grid(True, alpha=0.3)
plt.legend(fontsize=12)
plt.axhline(y=0, color='k', linestyle='-', alpha=0.2)
plt.tight_layout()
plt.show()

# Compute statistics for the KL expansion
print(f"L = {L}")
print("Mean of Q:", np.mean(Q))
print("Variance of Q:", np.var(Q))
```



```
L = 1.0
Mean of Q: -0.10384651739409384
Variance of Q: 0.8165221946938584
```

In [3]:

```
N_values = N_terms

Q_vectors = [np.random.randn(100) for _ in range(5)]

# Create a 2x2 grid of plots (for N=2, 10, 50, 100)
fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.flatten()

# Plot for each N value
for idx, N in enumerate(N_values):
    ax = axes[idx]

    # Plot all 5 realizations for this N
    for q_idx, Q in enumerate(Q_vectors):
        α_N =  np.cos(np.pi*x)

        # Add KL expansion terms
        for n in range(1, N+1):
            λ_n = 1/(1 + (L**2) * (eta_n[n-1]**2))
            ϕ_n = eigenfunction(n, x, L)
            α_N += np.sqrt(λ_n) * ϕ_n * Q[n-1]

        ax.plot(x, α_N, linewidth=1.5, label=f'Realization {q_idx+1}')

    ax.set_xlabel('x', fontsize=12)
    ax.set_ylabel('α(x)', fontsize=12)
    ax.set_title(f'N = {N} terms (L = {L})', fontsize=14)
    ax.grid(True, alpha=0.3)
    ax.axhline(y=0, color='k', linestyle='-', alpha=0.2)

    ax.legend(fontsize=9, loc='upper right')

# Add overall title
plt.suptitle('Five Realizations of α(x) = cos(πx) + Σ√λ_n ϕ_n(x)Q_n
for Different N',
             fontsize=16, y=1.02)
plt.tight_layout()
plt.show()
```


Five Realizations of α(x) = cos(πx) + Σ√λ_n φ_n(x)Q_n for Different N
N = 2 terms (L = 1.0)
N = 10 terms (L = 1.0)
N = 50 terms (L = 1.0)
N = 100 terms (L = 1.0)
Realization 1
Realization 2
Realization 3
Realization 4
Realization 5
α(x)
x

3.4

The convergence behavior of the Karhunen-Loève expansion for the Gaussian process with exponential covariance $C(x, y) = \frac{1}{2L} e^{-|x-y|/L}$ is illustrated in the progression from N=2 to N=100 terms. With only N=2 terms, the realizations are very smooth and show limited variability, with all five curves following a similar basic shape dominated by the deterministic component $\cos(\pi x)$ plus minimal random perturbation. By N=100 terms, the curves become very rough with high-frequency oscillations, and the realizations appear to have converged to their final form with little apparent change from N=50 to N=100. This progression reflects the spectral decay rate of the eigenvalues, which follows $\lambda_n \sim 1 / (n^2)$ for large n (specifically $\lambda_n = 1 / (1 + L^2 \eta_n^2)$, where η_n grow approximately linearly with n). This algebraic decay is relatively slow compared to exponential decay, explaining why a substantial number of terms are needed to capture the full character of the process; with more detail/variation, we require more eigenvalues to capture the characterestics of the function.

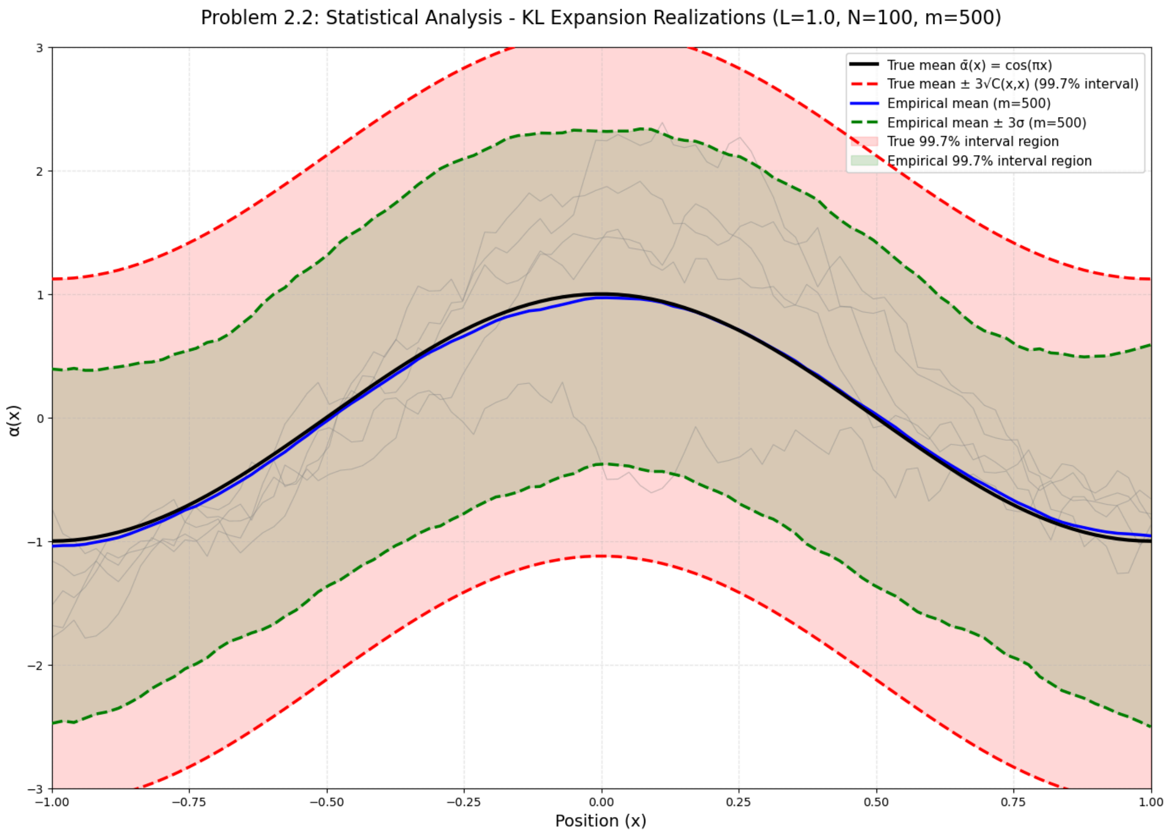
This convergence behavior has significant implications for dimension reduction. For large correlation lengths ($L \gg 1$, e.g., $L = 100$), the eigenvalues decay rapidly as $\lambda_n \approx 1 / (1 + (100)^2 \eta_n^2)$, enabling excellent approximations with just a few KL modes; typically $N \approx 10 - 20$ terms can capture over 99% of the variance. This allows effective dimensionality reduction from an infinite-dimensional process space to a manageable finite dimension, yielding significant computational savings while maintaining accuracy. Conversely, for small correlation lengths ($L \ll 1$, e.g., $L = 0.1$), the eigenvalue decay is much slower, following $\lambda_n \approx 1 / (1 + (0.1)^2 \eta_n^2)$. In this regime, dimension reduction becomes less effective as many modes are required to adequately capture the process characteristics, limiting potential computational benefits. This relationship between correlation length and eigenvalue decay rate directly determines the truncation error in the KL expansion and dictates the practical trade-off between model complexity and representation accuracy.

3.5

In [33]:

```
import numpy as np
import matplotlib.pyplot as plt
from problem3 import compute_eigenpairs

# ============================================
# Problem 2.2: Statistical Analysis
# ============================================
print("\n" + "="*60)
print("PROBLEM 2.2: Statistical Analysis of KL Expansion
Realizations")
print("="*60)

# Parameters
L = 1.0
m = 500  # Number of realizations
N = 100  # Number of KL terms to use
x = np.linspace(-1, 1, 100)  # Position array (xi values)
eta_n = data['etaVals_001p0'].flatten()  # Use eta values for L=0.01
(adjust as needed)
# True mean function
alpha_mean = np.cos(np.pi * x)

# Precompute eigenpairs once for efficiency
print("Precomputing eigenpairs...")
lambdas, phis = compute_eigenpairs(L, x,eta_n, N)

# Generate m realizations
print(f"Generating {m} realizations...")
realizations = np.zeros((m, len(x)))

for realization_idx in range(m):
    # Generate random Q vector with N i.i.d. standard normal variables
    Q = np.random.randn(N)

    # Start with the mean function
    α_N = np.cos(np.pi * x)

    # Add KL expansion terms
    for n in range(1,N+1):

        λ_n = lambdas[n-1]
        ϕ_n = phis[n-1, :]
        α_N += np.sqrt(λ_n) * ϕ_n * Q[n-1]

    realizations[realization_idx, :] = α_N

    # Show progress for large m
    if (realization_idx + 1) % 100 == 0:
        print(f"  Generated {realization_idx + 1}/{m} realizations")

print("Realizations generated successfully!")

# Compute empirical statistics at each x_i
μ_empirical = np.mean(realizations, axis=0)
σ_empirical = np.std(realizations, axis=0)

# Build true covariance matrix
n = len(x)
C = np.zeros((n, n))
for i in range(n):
    for j in range(n):
        C[i, j] = 1/(2*L) * np.exp(-np.abs(x[i] - x[j]) / L)

C_diag = np.diag(C)

# Create comprehensive plot
plt.figure(figsize=(14, 10))

# Plot the true mean
plt.plot(x, alpha_mean, 'k-', linewidth=3, label='True mean ᾱ(x) =
cos(πx)', zorder=10)

# Plot true 99.7% credible interval (±3√C(x,x))
plt.plot(x, alpha_mean + 3*np.sqrt(C_diag), 'r--', linewidth=2.5,
         label='True mean ± 3√C(x,x) (99.7% interval)', zorder=9)
plt.plot(x, alpha_mean - 3*np.sqrt(C_diag), 'r--', linewidth=2.5,
zorder=9)

# Plot empirical mean
plt.plot(x, μ_empirical, 'b-', linewidth=2.5, label=f'Empirical mean
(m={m})', zorder=8)

# Plot empirical 99.7% interval (±3σ)
plt.plot(x, μ_empirical + 3*σ_empirical, 'g--', linewidth=2.5,
         label=f'Empirical mean ± 3σ (m={m})', zorder=7)
plt.plot(x, μ_empirical - 3*σ_empirical, 'g--', linewidth=2.5,
zorder=7)

# Fill between for better visualization
plt.fill_between(x, alpha_mean - 3*np.sqrt(C_diag), alpha_mean +
3*np.sqrt(C_diag),
                 alpha=0.15, color='red', label='True 99.7% interval
region')
plt.fill_between(x, μ_empirical - 3*σ_empirical, μ_empirical +
3*σ_empirical,
                 alpha=0.15, color='green', label='Empirical 99.7%
interval region')

# Plot a few sample realizations for context (optional)
for i in range(5):  # Plot first 5 realizations
    plt.plot(x, realizations[i, :], 'gray', alpha=0.4, linewidth=0.8,
zorder=1)

plt.xlabel('Position (x)', fontsize=14)
plt.ylabel('α(x)', fontsize=14)
plt.title(f'Problem 2.2: Statistical Analysis - KL Expansion
Realizations (L={L}, N={N}, m={m})',
          fontsize=16, pad=20)
plt.grid(True, alpha=0.3, linestyle='--')
plt.legend(fontsize=11, loc='upper right', framealpha=0.95)
plt.ylim(-3, 3)
plt.xlim(-1, 1)
plt.tight_layout()
plt.show()
```

```
============================================================
PROBLEM 2.2: Statistical Analysis of KL Expansion Realizations
============================================================
Precomputing eigenpairs...
Generating 500 realizations...
  Generated 100/500 realizations
  Generated 200/500 realizations
  Generated 300/500 realizations
  Generated 400/500 realizations
  Generated 500/500 realizations
Realizations generated successfully!
```

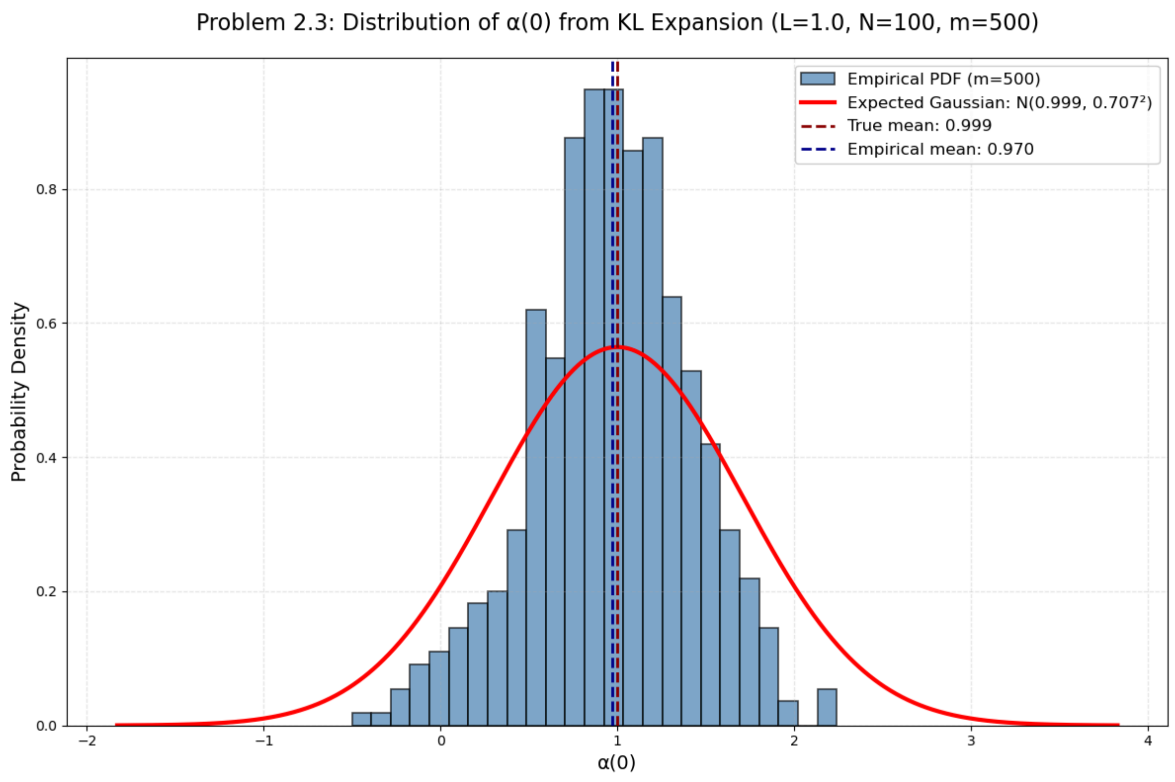

Problem 2.2: Statistical Analysis - KL Expansion Realizations (L=1.0, N=100, m=500)
True mean ᾱ(x) = cos(πx)
True mean ± 3√C(x,x) (99.7% interval)
Empirical mean (m=500)
Empirical mean ± 3σ (m=500)
True 99.7% interval region
Empirical 99.7% interval region
α(x)
Position (x)
3
2
1
0
−1
−2
−3
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

In [34]:

```
from scipy.stats import norm
print("\n" + "="*60)
print("PROBLEM 2.3: Distribution at x = 0")
print("="*60)

# Find index for x = 0 (should be x_51 since we have 101 points from
-1 to 1)
x0_idx = 50  # Index 50 corresponds to x=0 (0-based indexing)
print(f"\nx = 0 is at index {x0_idx} (x[{x0_idx}] = {x[x0_idx]:.6f})")

# Extract values at x = 0 from all realizations
α_x0 = realizations[:, x0_idx]

# Expected Gaussian parameters for x = 0
α_x0_true_mean = alpha_mean[x0_idx]  # cos(π*0) = 1
C_x0_x0 = C_diag[x0_idx]  # True variance at x = 0

print(f"True mean at x=0: ᾱ(0) = {α_x0_true_mean:.6f}")
print(f"True variance at x=0: C(0,0) = {C_x0_x0:.6f}")
print(f"True std deviation at x=0: √C(0,0) = {np.sqrt(C_x0_x0):.6f}")
print(f"\nEmpirical mean at x=0: {np.mean(α_x0):.6f}")
print(f"Empirical variance at x=0: {np.var(α_x0):.6f}")
print(f"Empirical std deviation at x=0: {np.std(α_x0):.6f}")

# Create histogram with PDF normalization
plt.figure(figsize=(12, 8))

# Histogram of empirical values with PDF normalization
n_bins = 25
hist_values, bin_edges, patches = plt.hist(α_x0, bins=n_bins,
density=True,
                                           alpha=0.7,
color='steelblue',
                                           edgecolor='black',
linewidth=1.2,
                                           label=f'Empirical PDF (m=
{m})')

# Expected Gaussian distribution
x_grid = np.linspace(α_x0_true_mean - 4*np.sqrt(C_x0_x0),
                     α_x0_true_mean + 4*np.sqrt(C_x0_x0), 1000)
pdf_expected = norm.pdf(x_grid, α_x0_true_mean, np.sqrt(C_x0_x0))
plt.plot(x_grid, pdf_expected, 'r-', linewidth=3,
         label=f'Expected Gaussian: N({α_x0_true_mean:.3f},
{np.sqrt(C_x0_x0):.3f}²)')

# Add vertical lines for means
```

```
plt.axvline(x=α_x0_true_mean, color='darkred', linestyle='--',
linewidth=2,
            label=f'True mean: {α_x0_true_mean:.3f}')
plt.axvline(x=np.mean(α_x0), color='darkblue', linestyle='--',
linewidth=2,
            label=f'Empirical mean: {np.mean(α_x0):.3f}')

plt.xlabel('α(0)', fontsize=14)
plt.ylabel('Probability Density', fontsize=14)
plt.title(f'Problem 2.3: Distribution of α(0) from KL Expansion (L=
{L}, N={N}, m={m})',
          fontsize=16, pad=20)
plt.grid(True, alpha=0.3, linestyle='--')
plt.legend(fontsize=12, framealpha=0.95)
plt.tight_layout()
plt.show()
```

```
============================================================
PROBLEM 2.3: Distribution at x = 0
============================================================

x = 0 is at index 50 (x[50] = 0.010101)
True mean at x=0: ᾱ(0) = 0.999497
True variance at x=0: C(0,0) = 0.500000
True std deviation at x=0: √C(0,0) = 0.707107

Empirical mean at x=0: 0.969674
Empirical variance at x=0: 0.201141
Empirical std deviation at x=0: 0.448488
```



3.7

Compared to Problem~~2.2, the KL-expansion yields slightly less variation in the resulting plots. Specifically, for~~ $N = 2$~~:~~ $\sigma \approx 0.378$~~; for~~ $N = 9$~~:~~ $\sigma \approx 0.465$~~; and for~~ $N = 100$~~:~~ $\sigma \approx 0.478$ ~~(average standard deviation across x). The variance clearly increases as more terms are retained in the expansion. A similar trend is evident in Problem~~3.6, where the standard deviation at $\alpha(0)$ grows from 0.0052 for $N = 2$ to 0.448 for $N = 100$.

The variance is lower with truncation because discarding higher-order KL terms removes their contribution to the total stochastic energy. Since the eigenvalues λ_n represent the variance captured by each mode, truncating at N terms yields a total variance of $\sum_{n=1}^{N} \lambda_n$, which is strictly less than the full process variance $\sum_{n=1}^{\infty} \lambda_n$. Consequently, realizations are systematically under-dispersed until N is sufficiently large for the tail sum $\sum_{n=N+1}^{\infty} \lambda_n$ to become negligible.

While individual realizations may appear visually adequate with relatively few terms, the ensemble statistics require more terms to accurately capture the full covariance structure. This discrepancy occurs because individual sample paths can be reasonably approximated by the dominant modes, whereas the complete stochastic distribution requires contributions from higher-order modes to achieve the correct variance. The slow approach to the theoretical variance---even with 100 terms the empirical variance remains slightly below the true value---highlights the challenge of capturing the ``tails'' of the eigenvalue spectrum, where many small but collectively significant contributions reside.

4

The analysis demonstrates that the characteristic length L fundamentally governs the spatial correlation structure of the process: smaller L values produce rapidly fluctuating, less correlated realizations, while larger L yields smoother, more strongly correlated variations. This directly illustrates how the covariance kernel's parameters shape the qualitative behavior of random processes, for example a Gaussian prior for a spatially varying inverse problem.

The KL expansion utilizes a truncated spectral representation that retains only the most significant (orthogonal) eigenmodes, thus achieving substantial dimensionality reduction while preserving statistical accuracy. The (rapid) eigenvalue decay (especially with large correlation length) means that only the first few modes are sufficient to represent the random process accurately.

With more terms in the KL expansion, the variance of the KL expansion increases, and the expansion becomes more faithful to the random process. Decompositions of the covariance matrix (e.g., Cholesky decomposition) allows for computationally efficient generation of realizations, when the eigenpairs have been precomputed.

Finally, through generating multiple realizations and computing empirical statistics, I confirmed the central limit theorem effect in practice: with m = 500 realizations, the empirical mean closely matched the theoretical mean, and the empirical covariance matrix converged toward the true covariance structure.

Gaussian processes and their KL expansion are essentially a probabilistic form of Principal Component Analysis (PCA). Just as PCA identifies orthogonal directions of maximum variance in data, the KL expansion identifies orthogonal eigenfunctions that capture the maximum variance in the random process. The eigenvalues _n represent the "explained variance" of each mode, ordered from largest to smallest. This connection reveals that KL expansions provide an optimal low-rank approximation of the covariance structure, where truncation after N terms yields the best N-term representation—just as PCA retains the most significant principal components while discarding negligible/non-data-informed ones.

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.linalg import cholesky
from scipy.stats import norm


def generate_gp_realizations(m, L, alpha_mean_func=None):
    """
    Generate m realizations of a Gaussian process

    Parameters:
    m: number of realizations
    L: length scale parameter
    alpha_mean_func: function to compute mean at grid points
    """

    x = np.array([-1 + (i-1) * 0.02 for i in range(1, 102)])  # 101 points
    n = len(x)

    if alpha_mean_func is None:
        alpha_true = np.cos(np.pi * x)
    else:
        alpha_true = alpha_mean_func(x)

    # Build covariance matrix
    C = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            C[i, j] = 1/(2*L) * np.exp(-np.abs(x[i] - x[j]) / L)

    # small diagonal for numerical stability?
    C += 1e-12 * np.eye(n)

    L_chol = cholesky(C, lower=True)

    realizations = np.zeros((m, n))
    for k in range(m):
        z = np.random.randn(n)
        # Transform to have covariance C
        realization = alpha_true + L_chol @ z
        realizations[k, :] = realization

    return x, alpha_true, C, realizations

def plot_5_realizations(L=1):
    """Generate and plot 5 realizations"""
    x, alpha_true, C, realizations = generate_gp_realizations(m=5, L=L)

    plt.figure(figsize=(12, 8))

    # Plot each realization
    for i in range(5):
        plt.plot(x, realizations[i, :], alpha=0.7, linewidth=1.5,
                label=f'Realization {i+1}')

    # Plot true mean
    plt.plot(x, alpha_true, 'k-', linewidth=3, label='True mean $\\alpha(x) = \cos(\pi x)$')

    # Plot theoretical 99.7% interval (B±3 std dev)
    std_dev = np.sqrt(np.diag(C))  # sqrt of diagonal elements
    plt.fill_between(x, alpha_true - 3*std_dev, alpha_true + 3*std_dev,
                    alpha=0.2, color='gray', label='Theoretical 99.7% interval')

    plt.xlabel('$x$', fontsize=14)
    plt.ylabel('$\\alpha(x)$', fontsize=14)
    plt.title(f'5 Realizations of Gaussian Process with $L={L}$', fontsize=16)
    plt.grid(True, alpha=0.3)
    plt.legend(loc='upper right', fontsize=10)
    plt.xlim(-1, 1)
    plt.tight_layout()
    plt.show()

    return x, alpha_true, C, realizations

def plot_500_realizations(L=1):
    """Generate 500 realizations and compute empirical statistics"""
    x, alpha_true, C, realizations = generate_gp_realizations(m=500, L=L)

    # Compute empirical statistics
    empirical_mean = np.mean(realizations, axis=0)
    empirical_std = np.std(realizations, axis=0)
    assert empirical_std.shape == (101,)
    assert empirical_std[1]!=empirical_std[2]
    # Theoretical values
    theoretical_std = np.sqrt(np.diag(C))  # sqrt(C(x_i, x_i))

    plt.figure(figsize=(14, 10))

    # Plot true mean
    plt.plot(x, alpha_true, 'k-', linewidth=3,
            label='True mean $\\alpha(x) = \cos(\pi x)$')

    # Plot theoretical 99.7% interval
    plt.plot(x, alpha_true - 3*theoretical_std, 'b--', linewidth=2,
            alpha=0.7, label='Theoretical: $\\alpha(x) \\pm 3\\sqrt{C(x,x)}$')
    plt.plot(x, alpha_true + 3*theoretical_std, 'b--', linewidth=2, alpha=0.7)

    # Plot empirical mean
    plt.plot(x, empirical_mean, 'r-', linewidth=2,
            label='Empirical mean $\\mu_x$')

    # Plot empirical 99.7% interval
    plt.plot(x, empirical_mean - 3*empirical_std, 'g--', linewidth=2,
            alpha=0.7, label='Empirical: $\\mu_x \\pm 3\\sigma_x$')
    plt.plot(x, empirical_mean + 3*empirical_std, 'g--', linewidth=2, alpha=0.7)

    # Fill between for better visualization
    plt.fill_between(x, alpha_true - 3*theoretical_std, alpha_true + 3*theoretical_std,
                    alpha=0.15, color='blue', label='Theoretical interval area')
    plt.fill_between(x, empirical_mean - 3*empirical_std, empirical_mean + 3*empirical_std,
                    alpha=0.1, color='green', label='Empirical interval area')

    plt.xlabel('$x$', fontsize=14)
    plt.ylabel('$\\alpha(x)$', fontsize=14)
    plt.title(f'Empirical vs Theoretical Statistics (L={L}, m=500 realizations)', fontsize=16)
    plt.grid(True, alpha=0.3)
    plt.legend(loc='upper right', fontsize=10)
    plt.xlim(-1, 1)
    plt.ylim(-3, 3)
    plt.tight_layout()
    plt.show()

    # Compute and print quantitative comparisons
    print("="*60)
    print(f"QUANTITATIVE ANALYSIS (L={L}, m=500 realizations):")
    print("="*60)

    # Mean squared error between true and empirical mean
    mse_mean = np.mean((alpha_true - empirical_mean)**2)
    print(f"Mean Squared Error (True vs Empirical mean): {mse_mean:.6f}")

    # Average absolute difference in standard deviations
    avg_std_diff = np.mean(np.abs(theoretical_std - empirical_std))
    print(f"Average absolute difference in std dev: {avg_std_diff:.6f}")

    # Correlation between theoretical and empirical intervals width
    theoretical_interval_width = 6 * theoretical_std
    empirical_interval_width = 6 * empirical_std
    correlation = np.corrcoef(theoretical_interval_width, empirical_interval_width)[0, 1]
    print(f"Correlation between interval widths: {correlation:.6f}")

    # Maximum pointwise error in mean
    max_abs_error_mean = np.max(np.abs(alpha_true - empirical_mean))
    print(f"Maximum absolute error in mean: {max_abs_error_mean:.6f}")

    # Check if empirical mean is within theoretical confidence band
    within_band = np.all(np.abs(alpha_true - empirical_mean) <= 3*theoretical_std/np.sqrt(500))
    print(f"Empirical mean within theoretical confidence band? {within_band}")

    return x, alpha_true, empirical_mean, theoretical_std, empirical_std, realizations, C

def additional_analysis(L=1):
    """Additional analysis of convergence"""
    m_values = [10, 50, 100, 200, 500, 1000]
    mse_values = []

    for m in m_values:
        x, alpha_true, C, realizations = generate_gp_realizations(m=m, L=L)
        empirical_mean = np.mean(realizations, axis=0)
        mse = np.mean((alpha_true - empirical_mean)**2)
        mse_values.append(mse)

    # Plot convergence of empirical mean
    plt.figure(figsize=(10, 6))
    plt.plot(m_values, mse_values, 'bo-', linewidth=2, markersize=8)
    plt.plot(m_values, 1/np.array(m_values), 'r--', linewidth=2,
            label='Theoretical $O(1/m)$ convergence')
    plt.xlabel('Number of realizations $m$', fontsize=14)
    plt.ylabel('Mean Squared Error', fontsize=14)
    plt.title(f'Convergence of Empirical Mean to True Mean (L={L})', fontsize=16)
    plt.grid(True, alpha=0.3)
    plt.xscale('log')
    plt.yscale('log')
    plt.legend(fontsize=12)
    plt.tight_layout()
    plt.show()

    print("\n" + "="*60)
    print("CONVERGENCE ANALYSIS:")
    print("="*60)
    for m, mse in zip(m_values, mse_values):
        print(f"m = {m:4d}: MSE = {mse:.6f}")

def plot_histogram_at_x51(x, alpha_true,  realizations,C, L=1, m=500):
    """Plot histogram of О±(xв‚…в‚Ѓ) at x=0"""
    if x is None or alpha_true is None or realizations is None or C is None:
        print("Error: Missing data for histogram plot. Rerunning 500 realizations first.")
        x, alpha_true, C, realizations = generate_gp_realizations(m=m, L=L)

    # xв‚…в‚Ѓ corresponds to index 50 (0-based) since xв‚Ѓ = -1
    # x = -1 + (51-1)*0.02 = -1 + 50*0.02 = -1 + 1 = 0
    idx_51 = 50
    x_51 = x[idx_51]
    alpha_x51 = alpha_true[idx_51]  # cos(ПЂ*0) = 1

    # Extract values at xв‚…в‚Ѓ across all realizations
    values_at_x51 = realizations[:, idx_51]

    # Theoretical Gaussian distribution
    mu_theoretical = alpha_x51  # cos(ПЂ*0) = 1
    sigma_theoretical = np.sqrt(C[idx_51, idx_51])  # sqrt(1) = 1


    # Plot 1: Histogram with fitted Gaussian
    plt.figure(figsize=(10, 6))

    # Plot histogram with fitted Gaussian
    n, bins, patches = plt.hist(values_at_x51, bins=30, density=True,
                                alpha=0.7, edgecolor='black', linewidth=0.5,
                                label=f'Empirical (m={m})')

    # Overlay theoretical Gaussian PDF
    x_pdf = np.linspace(mu_theoretical - 4*sigma_theoretical,
                        mu_theoretical + 4*sigma_theoretical, 1000)
    pdf_theoretical = norm.pdf(x_pdf, loc=mu_theoretical, scale=sigma_theoretical)
    plt.plot(x_pdf, pdf_theoretical, 'r-', linewidth=2.5,
            label='Theoretical $N(\\alpha(x_{51}), C(x_{51},x_{51}))$')

    # Add vertical line at theoretical mean
    plt.axvline(x=mu_theoretical, color='green', linestyle='--', linewidth=2,
               label=f'True mean: $\\alpha(x_{{51}}) = {mu_theoretical:.2f}$')

    plt.xlabel('$\\alpha(x_{51})$ at $x=0$', fontsize=14)
    plt.ylabel('Probability Density', fontsize=14)
    plt.title(f'Histogram of $\\alpha(x_{{51}})$ at $x=0$ (L={L})', fontsize=16)
    plt.grid(True, alpha=0.3)
    plt.legend(fontsize=11)



    # Quantitative comparison
    empirical_mean = np.mean(values_at_x51)
    empirical_std = np.std(values_at_x51)

    print("="*70)
    print(f"ANALYSIS AT xв‚…в‚Ѓ = 0 (L={L}, m={m}):")
    print("="*70)
    print(f"Theoretical distribution: N(Oj={mu_theoretical:.4f}, Пѓ={sigma_theoretical:.4f})")
    print(f"Empirical mean: {empirical_mean:.4f}")
    print(f"Empirical std: {empirical_std:.4f}")
    print(f"Absolute error in mean: {abs(empirical_mean - mu_theoretical):.4f}")
    print(f"Relative error in std: {abs(empirical_std - sigma_theoretical)/sigma_theoretical:.4f}")

    # Kolmogorov-Smirnov test for goodness of fit
    from scipy.stats import kstest
    ks_stat, ks_pvalue = kstest(values_at_x51, 'norm',
                                  args=(mu_theoretical, sigma_theoretical))
    print(f"K-S test statistic: {ks_stat:.4f}")
    print(f"K-S test p-value: {ks_pvalue:.4f}")

    if ks_pvalue > 0.05:
        print("data consistent with Gaussian")
    else:
        print("data not consistent with Gaussian")

    return values_at_x51, mu_theoretical, sigma_theoretical

def explore_different_L_values():
    """Explore effect of different L values"""
    m = 100  # Reduced for faster computation
    L_values = [0.1, 0.5, 1.0, 2.0, 5.0, 10.0]

    fig, axes = plt.subplots(2, 3, figsize=(15, 10))
    axes = axes.flatten()

    for idx, L in enumerate(L_values):
        x, alpha_true, C, realizations = generate_gp_realizations(m=m, L=L)

        # Plot a few realizations
        ax = axes[idx]
        for i in range(min(10, m)):
            ax.plot(x, realizations[i, :], alpha=0.4, linewidth=0.8)

        # Plot true mean
        ax.plot(x, alpha_true, 'k-', linewidth=2.5, label='True mean')

        # Plot theoretical B±3Пѓ interval
        theoretical_std = np.sqrt(np.diag(C))
        ax.fill_between(x, alpha_true - 3*theoretical_std, alpha_true + 3*theoretical_std,
                       alpha=0.2, color='gray', label='Theoretical 99.7% interval')

        ax.set_xlabel('$x$', fontsize=10)
        ax.set_ylabel('$\\alpha(x)$', fontsize=10)
        ax.set_title(f'L = {L}\nVar at x=0: {C[50,50]:.2f}', fontsize=12)
        ax.grid(True, alpha=0.3)
        ax.set_xlim(-1, 1)
        ax.set_ylim(-4, 4)

        # Add legend only to first subplot
        if idx == 0:
            ax.legend(fontsize=9, loc='upper right')

    plt.suptitle('Gaussian Process Realizations for Different L Values', fontsize=16, y=1.02)
    plt.tight_layout()
    plt.show()

    # Analysis of variance and correlation length
    print("\n" + "="*70)
    print("ANALYSIS OF L EFFECT ON VARIANCE AND CORRELATION:")
    print("="*70)

    for L in L_values:
        x, alpha_true, C, _ = generate_gp_realizations(m=1, L=L)  # Just get covariance matrix

        # Variance at a point (should always be 1 since C(x,x)=1)
        var_point = C[50, 50]

        # Correlation between neighboring points
        corr_neighbor = C[50, 51]  # correlation between x=0 and x=0.02

        # Correlation length (distance where correlation drops to 1/e)
        # Solve exp(-d/L) = 1/e => d = L
        correlation_length = L

        print(f"\nL = {L:.1f}:")
        print(f"  Variance at x=0: {var_point:.4f} (theoretical: 1.0000)")
        print(f"  Correlation(x=0, x=0.02): {corr_neighbor:.4f}")
        print(f"  Theoretical correlation length: {correlation_length:.4f}")
        print(f"  Distance for correlation to drop to 0.5: {-L*np.log(0.5):.4f}")

    # Additional plot: Realization smoothness vs L
    plt.figure(figsize=(12, 8))
def analyze_variance_trend():
    """Analyze how variance in realizations changes with L and across x positions"""
    m = 500
    L_values = [1.0, 2.0, 5.0, 1e1,  1e2]

    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10))

    for L in L_values:
        x, alpha_true, C, realizations = generate_gp_realizations(m=m, L=L)

        # Compute empirical variance across realizations at each x
        empirical_variance = np.var(realizations, axis=0)
        theoretical_variance = np.diag(C)  # Should be 1 for all x

        # Plot variance as function of x
        ax1.plot(x, empirical_variance, linewidth=2, label=f'L={L}')

        # Store data for histogram at x=0
        idx_51 = 50  # x=0 index
        values_at_x0 = realizations[:, idx_51]

        # Plot histogram of alpha at x=0
        ax2.hist(values_at_x0, bins=30, alpha=0.5, density=True,
                label=f'L={L} (Пѓ={np.std(values_at_x0):.3f})')

        # Print statistics
        print(f"L={L}: Empirical Пѓ at x=0 = {np.std(values_at_x0):.4f}, Theoretical = 1.0000")

    # Format variance plot

    ax1.set_xlabel('x', fontsize=12)
    ax1.set_ylabel('Empirical Variance', fontsize=12)
    ax1.set_title('Variance as Function of x for Different L Values', fontsize=14)
    ax1.grid(True, alpha=0.3)
    ax1.legend(loc='upper right', fontsize=10)
    ax1.set_xlim(-1, 1)

    # Format histogram plot
    x_pdf = np.linspace(-4, 4, 1000)
    pdf_theoretical = norm.pdf(x_pdf, loc=1, scale=1)  # Theoretical N(1,1)

    ax2.set_xlabel('О±(x=0)', fontsize=12)
    ax2.set_ylabel('Probability Density', fontsize=12)
    ax2.set_title('Histogram of О±(x=0) for Different L Values', fontsize=14)
    ax2.grid(True, alpha=0.3)
    ax2.legend(loc='best')
    ax2.set_xlim(-3, 5)
```

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

# Print statistical analysis
print("\n" + "="*70)
print("STATISTICAL ANALYSIS:")
print("="*70)
print("Empirical variance analysis:")
print("L Value | Mean Variance | Std of Variance ")
print("-"*60)

for L in L_values:
    x, alpha_true, C, realizations = generate_gp_realizations(m=m, L=L)
    empirical_variance = np.var(realizations, axis=0)

    mean_var = np.mean(empirical_variance)
    std_var = np.std(empirical_variance)
    max_dev = np.max(np.abs(empirical_variance - 1))

    print(f"L={L:<4} | {mean_var:.6f}     | {std_var:.6f}       | {max_dev:.6f}")
```

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import root_scalar
from scipy.optimize import fsolve


def compute_eigenpairs(L, x,etas, N=100):
    """Compute eigenvalues and eigenfunctions with corrected formulas."""
    # even_etas, odd_etas = compute_eta_values(L, N)
    odd_etas = etas[0::2]
    even_etas = etas[1::3]
    lambdas = np.zeros(N)
    phis = np.zeros((N, len(x)))

    for n in range(1, N+1):
        if n % 2 == 1:  # Odd n: use ODD eigenfunction (with even O· equation)
            idx = n // 2
            if idx < len(odd_etas):
                eta = odd_etas[idx]
                # Odd eigenvalue and eigenfunction (corrected: use sin for odd)
                lambdas[n-1] = 1 / (1 + L**2 * eta**2)
                denom = np.sqrt(1 + np.sin(2*eta)/(2*eta))
                phis[n-1, :] = np.sin(eta * x) / denom
        else:  # Even n: use EVEN eigenfunction (with odd O· equation)
            idx = n // 2 - 1
            if idx < len(even_etas):
                eta = even_etas[idx]
                # Even eigenvalue and eigenfunction (corrected: use cos for even)
                lambdas[n-1] = 1 / (1 + L**2 * eta**2)
                denom = np.sqrt(1 - np.sin(2*eta)/(2*eta))
                phis[n-1, :] = np.cos(eta * x) / denom

    return lambdas, phis
```