```
import numpy as np
import matplotlib.pyplot as plt
from scipy.spatial.distance import cosine

# Set random seed for reproducibility
np.random.seed(42)

#
================================================================================
====
# IDENTITY STABILITY (Δ_I) IMPLEMENTATION
#
================================================================================
====

def compute_delta_I(z_nominal, z_perturbed_list, lambda_weight=1.0):
    """
    Compute Identity Stability metric Δ_I

    Parameters:
    -----------
    z_nominal : np.ndarray
        Nominal internal representation (embedding vector)
    z_perturbed_list : list of np.ndarray
        List of perturbed internal representations
    lambda_weight : float
        Weight for variance term (default 1.0)

    Returns:
    --------
    delta_I : float
        Identity stability metric
    delta_I_mu : float
        Mean drift component
    delta_I_sigma : float
        Instability (variance) component
    """
    # Compute cosine distances for all perturbations
    distances = []
    for z_pert in z_perturbed_list:
        # d(x') = 1 - cos(z_nominal, z_perturbed)
        cos_sim = np.dot(z_nominal, z_pert) / (np.linalg.norm(z_nominal) * np.linalg.norm(z_pert))
        d = 1 - cos_sim
        distances.append(d)
```

```
    distances = np.array(distances)

    # Compute mean drift and variance
    delta_I_mu = np.mean(distances)
    delta_I_sigma = np.var(distances)

    # Combined metric
    delta_I = delta_I_mu + lambda_weight * delta_I_sigma

    return delta_I, delta_I_mu, delta_I_sigma

#
================================================================================
====
# SYNTHETIC AGENT SIMULATION
#
================================================================================
====

class SyntheticAgent:
    """
    Simple synthetic agent with internal belief state
    Simulates stable vs unstable identity regimes
    """
    def __init__(self, embedding_dim=64, stability_mode='stable'):
        self.embedding_dim = embedding_dim
        self.stability_mode = stability_mode

        # Initialize base belief state (random unit vector)
        self.base_state = np.random.randn(embedding_dim)
        self.base_state = self.base_state / np.linalg.norm(self.base_state)

    def get_representation(self, input_signal, perturbation_strength=0.0):
        """
        Get internal representation for given input

        Parameters:
        -----------
        input_signal : np.ndarray
            Input signal (not used in this toy model, but kept for interface)
        perturbation_strength : float
            Strength of perturbation applied
```

```
        Returns:
        --------
        z : np.ndarray
            Internal representation (belief state embedding)
        """
        if self.stability_mode == 'stable':
            # Stable regime: small Gaussian noise
            noise = np.random.randn(self.embedding_dim) * perturbation_strength * 0.1
            z = self.base_state + noise

        elif self.stability_mode == 'unstable':
            # Unstable regime: large random drift
            noise = np.random.randn(self.embedding_dim) * perturbation_strength * 0.8
            z = self.base_state + noise

        # Normalize to unit sphere
        z = z / np.linalg.norm(z)
        return z

#
================================================================================
====
# EXPERIMENT: MEASURE Δ_I FOR STABLE VS UNSTABLE AGENTS
#
================================================================================
====

def run_experiment(n_agents=50, n_perturbations=20, perturbation_strength=1.0):
    """
    Run experiment measuring Δ_I for stable vs unstable agents
    """
    results = {
        'stable': {'delta_I': [], 'delta_I_mu': [], 'delta_I_sigma': []},
        'unstable': {'delta_I': [], 'delta_I_mu': [], 'delta_I_sigma': []}
    }

    for mode in ['stable', 'unstable']:
        for _ in range(n_agents):
            # Create agent
            agent = SyntheticAgent(embedding_dim=64, stability_mode=mode)

            # Get nominal representation (no perturbation)
            nominal_input = np.random.randn(10)  # Dummy input
            z_nominal = agent.get_representation(nominal_input, perturbation_strength=0.0)
```

```
            # Generate perturbed representations
            z_perturbed_list = []
            for _ in range(n_perturbations):
                z_pert = agent.get_representation(nominal_input,
perturbation_strength=perturbation_strength)
                z_perturbed_list.append(z_pert)

            # Compute Δ_I
            delta_I, delta_I_mu, delta_I_sigma = compute_delta_I(z_nominal, z_perturbed_list,
lambda_weight=1.0)

            results[mode]['delta_I'].append(delta_I)
            results[mode]['delta_I_mu'].append(delta_I_mu)
            results[mode]['delta_I_sigma'].append(delta_I_sigma)

    return results

# Run experiment
print("Running Identity Stability (Δ_I) experiment...")
print("=" * 60)
results = run_experiment(n_agents=50, n_perturbations=20, perturbation_strength=1.0)

#
================================================================================
====
# ANALYSIS AND VISUALIZATION
#
================================================================================
====

# Convert to arrays
stable_delta_I = np.array(results['stable']['delta_I'])
unstable_delta_I = np.array(results['unstable']['delta_I'])

# Compute statistics
print("\nRESULTS:")
print("-" * 60)
print(f"Stable agents:")
print(f"  Mean Δ_I: {np.mean(stable_delta_I):.4f} ± {np.std(stable_delta_I):.4f}")
print(f"  Range: [{np.min(stable_delta_I):.4f}, {np.max(stable_delta_I):.4f}]")
print()
print(f"Unstable agents:")
print(f"  Mean Δ_I: {np.mean(unstable_delta_I):.4f} ± {np.std(unstable_delta_I):.4f}")
```

```
print(f"  Range: [{np.min(unstable_delta_I):.4f}, {np.max(unstable_delta_I):.4f}]")
print()
print(f"Separation: {np.mean(unstable_delta_I) - np.mean(stable_delta_I):.4f}")
print("=" * 60)

# Create visualization
fig, axes = plt.subplots(1, 2, figsize=(14, 5))

# Plot 1: Distributions
ax1 = axes[0]
ax1.hist(stable_delta_I, bins=15, alpha=0.7, label='Stable', color='#2ecc71', edgecolor='black')
ax1.hist(unstable_delta_I, bins=15, alpha=0.7, label='Unstable', color='#e74c3c',
edgecolor='black')
ax1.axvline(np.mean(stable_delta_I), color='#27ae60', linestyle='--', linewidth=2, label=f'Stable
mean: {np.mean(stable_delta_I):.3f}')
ax1.axvline(np.mean(unstable_delta_I), color='#c0392b', linestyle='--', linewidth=2,
label=f'Unstable mean: {np.mean(unstable_delta_I):.3f}')
ax1.set_xlabel('Identity Stability (Δ_I)', fontsize=12, fontweight='bold')
ax1.set_ylabel('Frequency', fontsize=12, fontweight='bold')
ax1.set_title('Distribution of Δ_I: Stable vs Unstable Agents', fontsize=14, fontweight='bold')
ax1.legend(fontsize=10)
ax1.grid(alpha=0.3)
# Plot 2: Box plot comparison
ax2 = axes[1]
bp = ax2.boxplot([stable_delta_I, unstable_delta_I],
                 tick_labels=['Stable', 'Unstable'],
                 patch_artist=True,
                 widths=0.6)
bp['boxes'][0].set_facecolor('#2ecc71')
bp['boxes'][1].set_facecolor('#e74c3c')
for element in ['whiskers', 'fliers', 'means', 'medians', 'caps']:
    plt.setp(bp[element], color='black', linewidth=1.5)
ax2.set_ylabel('Identity Stability (Δ_I)', fontsize=12, fontweight='bold')
ax2.set_title('Δ_I Comparison: Clear Separation', fontsize=14, fontweight='bold')
ax2.grid(axis='y', alpha=0.3)

plt.tight_layout()
plt.savefig('delta_I_experiment.png', dpi=300, bbox_inches='tight')
print("\nPlot saved to: delta_I_experiment.png")

#
==========================================================================
====
# DECOMPOSITION: MEAN DRIFT vs VARIANCE
```

```
#
================================================================================
====

fig2, axes2 = plt.subplots(1, 2, figsize=(14, 5))

# Plot mean drift component
ax1 = axes2[0]
stable_mu = np.array(results['stable']['delta_I_mu'])
unstable_mu = np.array(results['unstable']['delta_I_mu'])
ax1.scatter(range(len(stable_mu)), stable_mu, alpha=0.6, s=50, color='#2ecc71', label='Stable',
edgecolor='black')
ax1.scatter(range(len(unstable_mu)), unstable_mu, alpha=0.6, s=50, color='#e74c3c',
label='Unstable', edgecolor='black')
ax1.axhline(np.mean(stable_mu), color='#27ae60', linestyle='--', linewidth=2)
ax1.axhline(np.mean(unstable_mu), color='#c0392b', linestyle='--', linewidth=2)
ax1.set_xlabel('Agent Index', fontsize=12, fontweight='bold')
ax1.set_ylabel('Mean Drift (Δ_I^μ)', fontsize=12, fontweight='bold')
ax1.set_title('Mean Drift Component', fontsize=14, fontweight='bold')
ax1.legend(fontsize=10)
ax1.grid(alpha=0.3)

# Plot variance component
ax2 = axes2[1]
stable_sigma = np.array(results['stable']['delta_I_sigma'])
unstable_sigma = np.array(results['unstable']['delta_I_sigma'])
ax2.scatter(range(len(stable_sigma)), stable_sigma, alpha=0.6, s=50, color='#2ecc71',
label='Stable', edgecolor='black')
ax2.scatter(range(len(unstable_sigma)), unstable_sigma, alpha=0.6, s=50, color='#e74c3c',
label='Unstable', edgecolor='black')
ax2.axhline(np.mean(stable_sigma), color='#27ae60', linestyle='--', linewidth=2)
ax2.axhline(np.mean(unstable_sigma), color='#c0392b', linestyle='--', linewidth=2)
ax2.set_xlabel('Agent Index', fontsize=12, fontweight='bold')
ax2.set_ylabel('Instability (Δ_I^σ)', fontsize=12, fontweight='bold')
ax2.set_title('Variance Component', fontsize=14, fontweight='bold')
ax2.legend(fontsize=10)
ax2.grid(alpha=0.3)

plt.tight_layout()
plt.savefig('delta_I_decomposition.png', dpi=300, bbox_inches='tight')
print("Decomposition plot saved to: delta_I_decomposition.png")

print("\n" + "=" * 60)
print("EXPERIMENT COMPLETE")
```

```
print("=" * 60)
```