In [3]:

```
# Import necessary libraries
import numpy as np
from scipy.special import logsumexp
```

In [4]:

```
# Define the Baum-Welch algorithm

# Function to compute the forward probabilities
def forward(A, B, pi, observations):
    N = A.shape[0]  # Number of hidden states
    T = len(observations)  # Length of the observation sequence
    alpha = np.zeros((T, N))

    # Initialize alpha for t=0
    alpha[0, :] = pi * B[:, observations[0]]

    # Compute alpha values for t=1 to T-1
    for t in range(1, T):
        for j in range(N):
            alpha[t, j] = np.sum(alpha[t-1] * A[:, j]) * B[j, observations[t]]

    return alpha
```

In [5]:

```
# Function to compute the backward probabilities
def backward(A, B, observations):
    N = A.shape[0]  # Number of hidden states
    T = len(observations)  # Length of the observation sequence
    beta = np.zeros((T, N))

    # Initialize beta for t=T-1
    beta[-1, :] = 1

    # Compute beta values for t=T-2 to 0
    for t in range(T-2, -1, -1):
        for i in range(N):
            beta[t, i] = np.sum(A[i, :] * B[:, observations[t+1]] * beta[t+1, :])

    return beta
```

In [6]:

```
# Function to compute the Baum-Welch algorithm
def baum_welch(A, B, pi, observations, n_iter=10):
    N = A.shape[0]  # Number of hidden states
    M = B.shape[1]  # Number of observation symbols
    T = len(observations)  # Length of the observation sequence

    for _ in range(n_iter):
        # E-step: Compute forward and backward probabilities
        alpha = forward(A, B, pi, observations)
        beta = backward(A, B, observations)

        # Compute gamma and xi
        gamma = np.zeros((T, N))
        xi = np.zeros((T-1, N, N))
```

```python
        # Compute gamma
        denom = np.sum(alpha[-1, :])
        for t in range(T):
            gamma[t, :] = alpha[t, :] * beta[t, :] / denom

        # Compute xi
        for t in range(T-1):
            denom = np.sum(alpha[t, :] * A * B[:, observations[t+1]] * beta[t+1, :]
            for i in range(N):
                xi[t, i, :] = (alpha[t, i] * A[i, :] * B[:, observations[t+1]] * be

        # M-step: Update the parameters
        pi = gamma[0, :]
        A = np.sum(xi, axis=0) / np.sum(gamma[:-1, :], axis=0)[:, None]
        B = np.zeros((N, M))
        for j in range(M):
            mask = (observations == j)
            B[:, j] = np.sum(gamma[mask, :], axis=0) / np.sum(gamma, axis=0)

    return A, B, pi
```

In [7]:

```python
# Define initial parameters
N = 2  # Number of hidden states
M = 2  # Number of observation symbols

# Initial state probabilities (pi)
pi = np.array([0.5, 0.5])

# Transition probabilities (A)
A = np.array([[0.7, 0.3],
              [0.4, 0.6]])

# Emission probabilities (B)
B = np.array([[0.9, 0.1],
              [0.2, 0.8]])

# Observed sequence: Umbrella (0), No Umbrella (1)
observations = np.array([0, 1, 0, 0, 1])

# Run Baum-Welch algorithm
A_est, B_est, pi_est = baum_welch(A, B, pi, observations)
```

In [8]:

```python
# Print the results
print("Estimated Initial State Probabilities (π):")
print(pi_est)

print("\nEstimated Transition Probabilities (A):")
print(A_est)

print("\nEstimated Emission Probabilities (B):")
print(B_est)
```

```
Estimated Initial State Probabilities (π):
[1.0000000e+000 9.1917645e-107]

Estimated Transition Probabilities (A):
[[1.76232783e-01 5.44311780e+33]
 [2.72234931e+00 3.16799028e-34]]

Estimated Emission Probabilities (B):
[[1.00000000e+00 2.04131373e-35]
 [7.36630343e-35 1.00000000e+00]]
```