```
import numpy as np

class HMM:
    def __init__(self, N, M):
        self.N = N  # number of states
        self.M = M  # number of observation symbols

        # Initialize probabilities randomly
        self.A = np.random.rand(N, N)
        self.A /= self.A.sum(axis=1, keepdims=True)

        self.B = np.random.rand(N, M)
        self.B /= self.B.sum(axis=1, keepdims=True)

        self.pi = np.random.rand(N)
        self.pi /= self.pi.sum()

    def forward(self, O):
        T = len(O)
        alpha = np.zeros((T, self.N))

        alpha[0] = self.pi * self.B[:, O[0]]

        for t in range(1, T):
            for j in range(self.N):
                alpha[t, j] = np.sum(alpha[t-1] * self.A[:, j]) * self.B[j, O[t]]

        return alpha

    def backward(self, O):
        T = len(O)
        beta = np.zeros((T, self.N))

        beta[T-1] = 1

        for t in range(T-2, -1, -1):
            for i in range(self.N):
                beta[t, i] = np.sum(self.A[i] * self.B[:, O[t+1]] * beta[t+1])

        return beta

    def baum_welch(self, O, max_iter=100):
        T = len(O)

        for iteration in range(max_iter):
            alpha = self.forward(O)
            beta = self.backward(O)
```

```
            xi = np.zeros((T-1, self.N, self.N))
            gamma = np.zeros((T, self.N))

            for t in range(T-1):
                denom = np.sum(alpha[t] * beta[t])
                for i in range(self.N):
                    numer = alpha[t, i] * self.A[i] * self.B[:, O[t+1]] * beta[t+1]
                    xi[t, i] = numer / denom

            gamma = np.sum(xi, axis=2)
            gamma = np.vstack((gamma, np.sum(xi[T-2], axis=0)))

            # Update pi
            self.pi = gamma[0]

            # Update A
            for i in range(self.N):
                denom = np.sum(gamma[:-1, i])
                for j in range(self.N):
                    numer = np.sum(xi[:, i, j])
                    self.A[i, j] = numer / denom

            # Update B
            for i in range(self.N):
                denom = np.sum(gamma[:, i])
                for k in range(self.M):
                    mask = (O == k)
                    numer = np.sum(gamma[mask, i])
                    self.B[i, k] = numer / denom

        return self.A, self.B, self.pi


# Example usage
if __name__ == "__main__":
    # Observations (encoded as integers)
    O = np.array([0, 1, 0, 2, 1])

    N = 2  # states
    M = 3  # observation symbols

    hmm = HMM(N, M)

    A, B, pi = hmm.baum_welch(O, max_iter=50)

    print("Transition Matrix A:\n", A)
    print("Emission Matrix B:\n", B)
    print("Initial Probabilities pi:\n", pi)
```