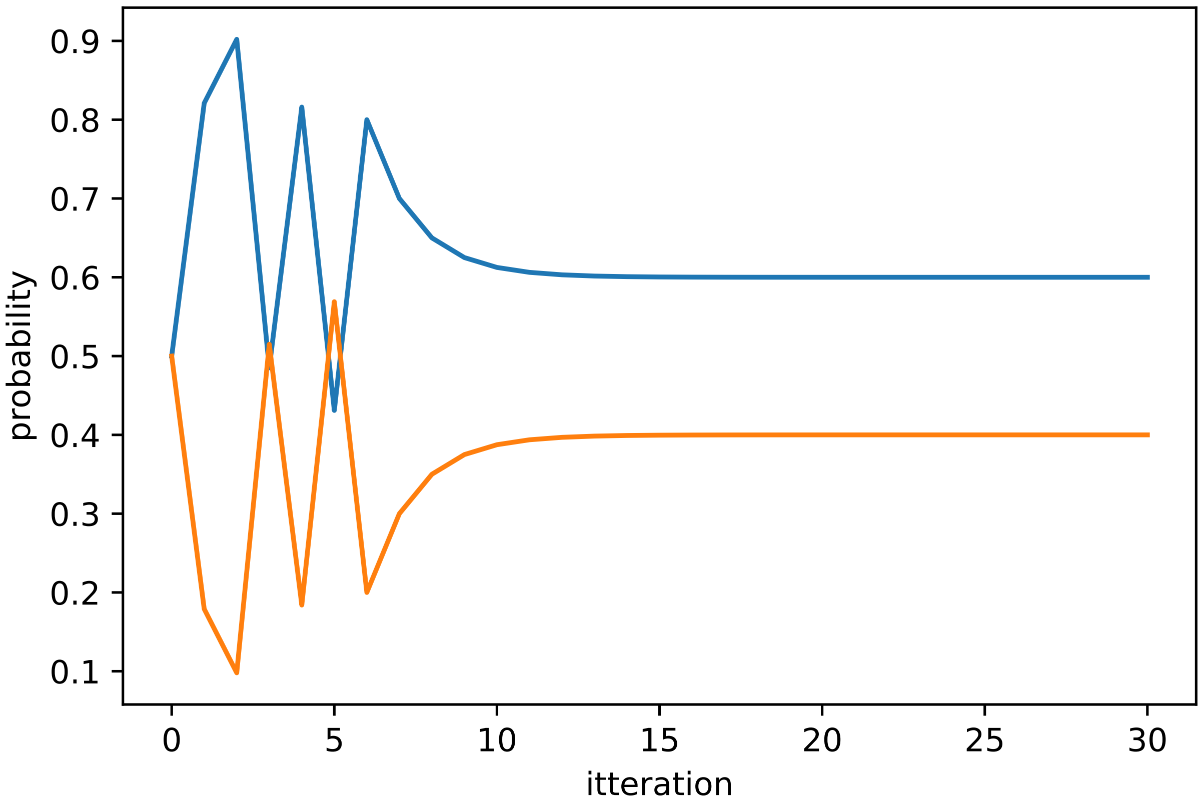
```
import numpy as np
import matplotlib.pyplot as plt

#P(X_{t+k+1}|e_1:t) = _{x_{t+k}} P(X_{t+k+1}|x_{t+k} )P(x_{t+k} |e_{1:t})
def filter_markov(initial, e, transition, emission, table):
    #transition
    temp = np.zeros(len(transition[0]))
    for i, value in enumerate(initial):
        P_X = np.add(temp, transition[i] * value)
        temp = P_X

    #emission
    P =  [emission[0][e[0]], emission[1][e[0]]]*P_X
    result = [round(x * 1/(sum(P)), 3) for x in P] #Normalize the probability values so
    table.append(result)

    if len(e) == 1: # Day 6
        return table

    return filter_markov(result.copy(), e[1:], transition, emission, table)

def prediction_markov(initial, transition, n, table):
    #transition
    temp = np.zeros(len(transition[0]))
    for i, value in enumerate(initial):
        P_X = np.add(temp, transition[i] * value)
        temp = P_X
    table.append(P_X.tolist())

    if n == 1:
        return table

    return prediction_markov(P_X, transition, n-1, table)

def smoothing_markov(b, e, filtered_table, transition, emission, table):
        if len(filtered_table) == 0:
            return table

        em = np.array([[emission[0][e[-1]], 0], [0, emission[1][e[-1]]]]) #Diagonal mat
        b_x = transition @ em @ b

        P_X = filtered_table[-1] * b_x
        result = [round(x * 1/(sum(P_X)), 3) for x in P_X] #Normalize
        table.append(result)
        return smoothing_markov(b_x, e[:-1], filtered_table[:-1], transition ,emission,

def viterbi(initial, e, transition, emission, table):
    if table == []: #First itteration
        temp = np.zeros(len(transition[0]))
        for i, value in enumerate(initial):
            P_X = np.add(temp, transition[i] * value)
            temp = P_X
        #emission
        P =  [emission[0][e[0]], emission[1][e[0]]]*P_X
        result = [round(x * 1/(sum(P)), 4) for x in P] #Normalize the probability values

    else:
        result = []
```

```python
        print(e[0])
        for i in range(len(transition)): #Calculates the 4 probabilties, and finds the
            temp = []
            for j, prob in enumerate(table[-1]):
                temp.append(transition[j][i] * emission[i][e[0]] * prob)
            result.append(round(max(temp), 4))

    table.append(result)
    if len(e) == 1:
        return table

    return viterbi(result.copy(), e[1:], transition, emission, table)

def problem1b():
    print("\nProblem 1b) \n")
    F_0 = np.array([0.5, 0.5]) #Initial state
    transition = np.array([[0.8, 0.2], [0.3, 0.7]]) #Given F_{t-1} is False => P(F_t is
    emission = np.array([[0.75, 0.25], [0.2, 0.8]]) #Given F_{t} is True => P(B_t is fa
    #e = [True, True, False, True, False, True] Birds nearby => True, e[0] = e_1
    e = [0, 0, 1, 0, 1, 0]
    table = filter_markov(F_0, e, transition, emission, [])
    table.insert(0, [0.5, 0.5])
    for i, P in enumerate(table):
        print(f"P(X_{i}|e_1:{i}) = {P}")
    return table

def problem1c(filtered_table):
    print("\nProblem 1c) \n")
    transition = np.array([[0.8, 0.2], [0.3, 0.7]])
    emission = np.array([[0.75, 0.25], [0.2, 0.8]])
    table = prediction_markov(filtered_table[-1], transition, 24, [])
    for i, P in enumerate(table):
        print(f"P(X_{i+7}|e_1:{i+7}) = {P}")
    x_arr = [x for x in range(31)]
    y_arr = filtered_table + table
    plt.plot(x_arr, y_arr)
    plt.ylabel(’probability’)
    plt.xlabel(’itteration’)
    plt.show()

def problem1d(filtered_table):
    print("\nProblem 1d) \n")
    transition = np.array([[0.8, 0.2], [0.3, 0.7]])
    emission = np.array([[0.75, 0.25], [0.2, 0.8]])
    e = [0, 0, 1, 0, 1, 0]
    table = smoothing_markov([1,1], e, filtered_table[:-1], transition, emission, [])
    table.reverse() #Calculating from F_5 -> F_0
    for i, P in enumerate(table):
        print(f"P(X_{i}|e_1:6) = {P}")

def problem1e():
    print("\nProblem 1e) \n")

    F_0 = np.array([0.5, 0.5])
    transition = np.array([[0.8, 0.2], [0.3, 0.7]])
    emission = np.array([[0.75, 0.25], [0.2, 0.8]])
    e = [0, 0, 1, 0, 1, 0]
    table = viterbi(F_0, e, transition, emission, []) #For 2 possible paths
    boolean_table = []
    for list in table:
        if list[0] > list[1]:
            boolean_table.append(True)
        else:
            boolean_table.append(False)

    print("The most likely sequence is:", boolean_table)

if __name__ == '__main__':
    filtered_table = problem1b()
    problem1c(filtered_table)
    problem1d(filtered_table)
    problem1e()
```

```

Problem 1b)

P(X_0|e_1:0) = [0.5, 0.5]
P(X_1|e_1:1) = [0.821, 0.179]
P(X_2|e_1:2) = [0.902, 0.098]
P(X_3|e_1:3) = [0.485, 0.515]
P(X_4|e_1:4) = [0.816, 0.184]
P(X_5|e_1:5) = [0.431, 0.569]
P(X_6|e_1:6) = [0.8, 0.2]

Problem 1c)

P(X_7|e_1:7) = [0.7000000000000002, 0.30000000000000004]
P(X_8|e_1:8) = [0.6500000000000001, 0.3500000000000001]
P(X_9|e_1:9) = [0.6250000000000001, 0.3750000000000001]
P(X_10|e_1:10) = [0.6125000000000002, 0.38750000000000007]
P(X_11|e_1:11) = [0.6062500000000002, 0.3937500000000001]
P(X_12|e_1:12) = [0.6031250000000001, 0.3968750000000001]
P(X_13|e_1:13) = [0.6015625000000002, 0.39843750000000006]
P(X_14|e_1:14) = [0.6007812500000002, 0.39921875000000007]
P(X_15|e_1:15) = [0.6003906250000002, 0.39960937500000004]
P(X_16|e_1:16) = [0.6001953125000001, 0.39980468750000003]
P(X_17|e_1:17) = [0.6000976562500001, 0.39990234375]
P(X_18|e_1:18) = [0.6000488281250002, 0.399951171875]
P(X_19|e_1:19) = [0.6000244140625002, 0.39997558593750004]
P(X_20|e_1:20) = [0.6000122070312501, 0.39998779296875003]
P(X_21|e_1:21) = [0.6000061035156251, 0.39999389648437506]
P(X_22|e_1:22) = [0.6000030517578125, 0.39999694824218757]
P(X_23|e_1:23) = [0.6000015258789063, 0.3999984741210938]
P(X_24|e_1:24) = [0.6000007629394533, 0.3999992370605469]
P(X_25|e_1:25) = [0.6000003814697267, 0.3999996185302735]
P(X_26|e_1:26) = [0.6000001907348634, 0.39999980926513673]
P(X_27|e_1:27) = [0.6000000953674318, 0.3999999046325684]
P(X_28|e_1:28) = [0.600000047683716, 0.39999995231628427]
P(X_29|e_1:29) = [0.600000023841858, 0.3999999761581422]
P(X_30|e_1:30) = [0.6000000119209291, 0.3999999880790711]

Problem 1d)

P(X_0|e_1:6) = [0.665, 0.335]
P(X_1|e_1:6) = [0.876, 0.124]
P(X_2|e_1:6) = [0.866, 0.134]
P(X_3|e_1:6) = [0.598, 0.402]
```

```
P(X_4|e_1:6) = [0.766, 0.234]
P(X_5|e_1:6) = [0.57, 0.43]

Problem 1e)

0
1
0
1
0
The most likely sequence is: [True, True, True, True, True, True]
```


probability
itteration