Problem 1: Recommendation Model (20 points)

A certain app is deciding whether to show you a personalized **ad**, represented by a binary random variable A. It can possibly consider two factors: whether you browse the **virtual** marketplace and whether you make a trip to the **physical** store location (it can track your location!). These decisions are represented by binary random variables V and P. The joint distribution over all three variables is as follows:

V	P	A	$\Pr(V, P, A)$
$+v$	$+p$	$+a$	0.12
$+v$	$+p$	$-a$	0.08
$+v$	$-p$	$+a$	0.18
$+v$	$-p$	$-a$	0.12
$-v$	$+p$	$+a$	0.06
$-v$	$+p$	$-a$	0.14
$-v$	$-p$	$+a$	0.09
$-v$	$-p$	$-a$	0.21

1. Find the marginal distributions of each of the three random variables.

Pr(+v) = 0.12 + 0.08 + 0.18 + 0.12 = 0.5
Pr(-v) = 0.06 + 0.14 + 0.09 + 0.21 = 0.5

Pr(+p) = 0.12 + 0.08 + 0.06 + 0.14 = 0.4
Pr(-p) = 0.18 + 0.12 + 0.09 + 0.21 = 0.6

Pr(+a) = 0.12 + 0.18 + 0.06 + 0.09 = 0.45
Pr(-a) = 0.08 + 0.12 + 0.14 + 0.21 = 0.55

2. Find the joint distributions of each of the three different pairs of the random variables.

Pr(+v, +p) = 0.12 + 0.08 = 0.2
Pr(+v, -p) = 0.18 + 0.12 = 0.3
Pr(-v, +p) = 0.06 + 0.14 = 0.2
Pr(-v, -p) = 0.09 + 0.21 = 0.3

Pr(+v, +a) = 0.12 + 0.18 = 0.3
Pr(+v, -a) = 0.08 + 0.12 = 0.2
Pr(-v, +a) = 0.06 + 0.09 = 0.15
Pr(-v, -a) = 0.14 + 0.21 = 0.35

Pr(+p, +a) = 0.12 + 0.06 = 0.18
Pr(+p, -a) = 0.08 + 0.14 = 0.22
Pr(-p, +a) = 0.18 + 0.09 = 0.27
Pr(-p, -a) = 0.12 + 0.21 = 0.33

3. Using the distributions you found above, show whether each pair of random variables is independent.

Pr(+v, +p) = 0.12 + 0.08 = 0.2 = Pr(+v)Pr(+p) = 0.5 * 0.4 = 0.2
Pr(+v, -p) = 0.18 + 0.12 = 0.3 = Pr(+v)Pr(-p) = 0.5 * 0.6 = 0.3
Pr(-v, +p) = 0.06 + 0.14 = 0.2 = Pr(-v)Pr(+p) = 0.5 * 0.4 = 0.2
Pr(-v, -p) = 0.09 + 0.21 = 0.3 = Pr(-v)Pr(-p) = 0.5 * 0.6 = 0.3
The pair of random variables (V,P) is independent

Pr(+v, +a) = 0.12 + 0.18 = 0.3 != Pr(+v)Pr(+a) = 0.5 * 0.45 = 0.225
Pr(+v, -a) = 0.08 + 0.12 = 0.2 != Pr(+v)Pr(-a) = 0.5 * 0.55 = 0.275
Pr(-v, +a) = 0.06 + 0.09 = 0.15 != Pr(-v)Pr(+a) = 0.5 * 0.45 = 0.225
Pr(-v, -a) = 0.14 + 0.21 = 0.35 != Pr(-v)Pr(-a) = 0.5 * 0.55 = 0.275
The pair of random variables (V,A) is not independent

Pr(+p, +a) = 0.12 + 0.06 = 0.18 = Pr(+p)Pr(+a) = 0.4 * 0.45 = 0.18
Pr(+p, -a) = 0.08 + 0.14 = 0.22 = Pr(+p)Pr(-a) = 0.4 * 0.55 = 0.22
Pr(-p, +a) = 0.18 + 0.09 = 0.27 = Pr(-p)Pr(+a) = 0.6 * 0.45 = 0.27
Pr(-p, -a) = 0.12 + 0.21 = 0.33 = Pr(-p)Pr(-a) = 0.6 * 0.55 = 0.33
The pair of random variables (P,A) is independent

4. Find the following conditional distributions: Pr(V | + a), Pr(P| + a), Pr(V, P| + a).

Pr(V | + a)
Pr(+v| +a) = Pr(+v, +a)/Pr(+a) = 0.3 / 0.45 = ⅔
Pr(-v| +a) = Pr(-v, +a)/Pr(+a) = 0.15 / 0.45 = ⅓

Pr(P| + a)
Pr(+p| +a) = p(+p, +a)/p(+a) = 0.18 / 0.45 = 0.4
Pr(-p| +a) = p(-p, +a)/p(+a) = 0.27 / 0.45 = 0.6

Pr(V, P| + a)
Pr(+v, +p| +a) = Pr(+v, +p, +a)/Pr(+a) = 0.12 / 0.45 = 4/15
Pr(+v, -p| +a) = Pr(+v, -p, +a)/Pr(+a) = 0.18 / 0.45 = 0.4
Pr(-v, +p| +a) = Pr(-v, +p, +a)/Pr(+a) = 0.06 / 0.45 = 2/15
Pr(+v, +p| +a) = Pr(-v, -p, +a)/Pr(+a) = 0.09 / 0.45 = 0.2

5. Can we conclude that V and P are conditionally independent given A without further calculations? If not, what additional calculations do we need to verify?

Cannot conclude that V and P are conditionally independent without further calculations.

Need to verify:
Pr(+v| +a)Pr(+p| +a) = ⅔ * 0.4 = 4/15 = Pr(+v, +p| +a)
Pr(+v| +a)Pr(-p| +a) = ⅔ * 0.6 = 0.4 = Pr(+v, -p| +a)
Pr(-v| +a)Pr(+p| +a) = ⅓ * 0.4 = 2/15 = Pr(-v, +p| +a)
Pr(-v| +a)Pr(-p| +a) = ⅓ * 0.6 = 0.2 = Pr(+v, +p| +a)

We would also need to check for A=−a to conclusively determine conditional independence.

6. Explain whether it is possible to recover Pr(V, P, A) using only the three marginal distributions you computed in part 1.

V, P are independent
P, A are independent
V, A are not independent

Pr(V, P, A) = Pr(V|P, A)Pr(P, A) = Pr(V|A)Pr(P)Pr(A) = Pr(V|A)Pr(A)Pr(P) = Pr(V, A)Pr(P)

It is not possible to recover Pr(V, P, A) using only the three marginal distributions Pr(V), Pr(P), and Pr(A) because V and A are not independent so Pr(V, A) cannot be simplified to Pr(V)Pr(A).

Consider the same question if we were to only use three joint distributions you computed in part 2.

Pr(V, P, A) = Pr(V|P, A)Pr(P, A) = Pr(V|A)Pr(P)Pr(A) = Pr(V|A)Pr(A)Pr(P) = Pr(V, A)Pr(P)
Pr(V,A) is known.
Pr(P) can be computed from the joint distribution Pr(P,A) by summing over A.
Pr(+p) = Pr(+p, +a) + Pr(+p, -a)
Pr(-p) = Pr(-p, +a) + Pr(-p, -a)

Of the six distributions, what is the minimal set needed (i.e., minimum number of table entries) to recover Pr(V, P, A)?

To recover Pr(V, P, A), need a minimum of 7 out of the 8 entries. The 8th can be calculated as:

$$Pr(-v,-p,-a) = 1 - \sum_{\substack{V \neq -v \\ P \neq -p \\ A \neq -a}} Pr(V,P,A)$$

Problem 2: Wandering Robot (18 points)

A robot is wandering around a room with some obstacles, labeled as # in the grid below. It can occupy any of the free cells labeled with a letter, but we are uncertain about its true location and so we keep a belief distribution over its current location. At each timestep, it may either stay in its current cell, or move to an adjacent cardinal cell, all with uniform probability. For example, from A the robot can move to A, B, or D each with probability $\frac{1}{3}$, while from C it can move to B or C, each with probability $\frac{1}{2}$.

A	B	C
D	E	#
#	F	#

The robot also makes an observation after each transition, returning what it sees in a random cardinal direction. Possibilities include observing #, "wall", or "empty" (for a free cell). For example, in D the robot observes "wall", # (each with probability $\frac{1}{4}$), or "empty" (probability $\frac{1}{2}$).

We highly recommend that you use Python or an equivalent computing library to complete this problem. You can screenshot all of the relevant code that you wrote or attach a notebook file in place of showing work. Please still include the final answer for each part in your writeup.

1. Suppose the robot wanders around for a long time without making any observations. What is the stationary distribution π over the robot's predicted location?

```
π = [0.16666667 0.22222222 0.11111111 0.16666667 0.22222222 0.11111111]
```

2. The robot's sensors just started working. Take the stationary distribution that you found above to be $\Pr(X_0)$. Starting from this belief state, the robot makes one transition and observes e_1 = "wall". What is the updated belief distribution $\Pr(X_1 \mid e_1)$?

```
Pr(X1|e1) = [0.31578947 0.21052632 0.21052632 0.15789474 0.     0.10526316]
```

3. The robot makes a second transition and observes e2 = #. What is the updated belief distribution Pr(X2 | e1,e2)?

```
Pr(X2|e1,e2) = [0.      0.      0.27272727 0.27272727 0.27272727 0.18181818]
```

4. Compute the joint distribution Pr(X1, X2 | e1, e2). You can either write the result as a matrix A, where Aij = Pr(X1 = i, X2 = j | e1, e2), or just list the joint probability values that are nonzero.

```
A = [[0.         0.         0.         0.18181818 0.         0.        ]
     [0.         0.         0.09090909 0.         0.09090909 0.        ]
     [0.         0.         0.18181818 0.         0.         0.        ]
     [0.         0.         0.         0.09090909 0.09090909 0.        ]
     [0.         0.         0.         0.         0.         0.        ]
     [0.         0.         0.         0.         0.09090909 0.18181818]]
```

5. Using your result above, compute the distribution Pr(X1|e1, e2).

```
Pr(X1|e1,e2) = [0.18181818 0.18181818 0.18181818 0.18181818 0. 0.27272727]
```

```
import numpy as np
P = np.array([
    [1/3, 1/3, 0,   1/3, 0,   0  ],  # A -> A, B, D
    [1/4, 1/4, 1/4, 0,   1/4, 0  ],  # B -> A, B, C, E
    [0,   1/2, 1/2, 0,   0,   0  ],  # C -> B, C
    [1/3, 0,   0,   1/3, 1/3, 0  ],  # D -> A, D, E
    [0,   1/4, 0,   1/4, 1/4, 1/4],  # E -> B, D, E, F
    [0,   0,   0,   0,   1/2, 1/2]]) # F -> E, F
n = P.shape[0]
A = np.eye(n) - P.T
A[-1,:] = 1
b = np.zeros(n)
b[-1] = 1
pi = np.linalg.solve(A, b)
print("pi = ", pi)

O_e1 = np.array([1/2, 1/4, 1/2, 1/4, 0, 1/4])
Pr_X0 = pi
Pr_X1 = np.matmul(Pr_X0, P)
Pr_X1_and_e1 = Pr_X1 * O_e1
Pr_X1_given_e1 = Pr_X1_and_e1 / Pr_X1_and_e1.sum()
print("Pr(X1|e1) = ", Pr_X1_given_e1)

O_e2 = np.array([0, 0, 1/4, 1/4, 1/4, 1/2])
Pr_X2_given_e1 = np.matmul(Pr_X1_given_e1, P)
Pr_X2_and_e2_given_e1 = Pr_X2_given_e1 * O_e2
Pr_X2_given_e1_e2 = Pr_X2_and_e2_given_e1 / Pr_X2_and_e2_given_e1.sum()
print("Pr(X2|e1,e2) = ", Pr_X2_given_e1_e2)

A = np.zeros((6, 6))
for i in range(6):
    for j in range(6):
        A[i, j] = Pr_X1_given_e1[i] * P[i, j] * O_e2[j]
A = A / A.sum()
print("A = ")
print(A)

Pr_X1_given_e1_e2 = np.sum(A, axis=1)
print("Pr(X1|e1, e2) = ", Pr_X1_given_e1_e2)
```

Problem 3: Robot Learning (16 points)

The robot suspects that the transition and observation models described above are no longer accurate. We will try to update them by running through an iteration of the Baum-Welch algorithm, starting with the same initial distribution and models that you used in the last problem.
As with the last problem, we recommend completing this in Python.

1. Compute (in order) the probability arrays β2, β1, and β0.

```
β2 = [1. 1. 1. 1. 1. 1.]
β1 = [0.08333333 0.125      0.125      0.16666667 0.25       0.375     ]
β0 = [0.03819444 0.03385417 0.046875   0.02777778 0.04166667 0.046875  ]
```

2. Compute the distribution of state occurrences in each timestep: γ0, γ1, γ2. What is the new initial state distribution Pr(ˆX0)?

```
γ0 = [0.16666667 0.1969697  0.13636364 0.12121212 0.24242424 0.13636364]
γ1 = [0.18181818 0.18181818 0.18181818 0.18181818 0.         0.27272727]
γ2 = [0.         0.         0.27272727 0.27272728 0.27272727 0.18181818]
Pr(ˆX0) = γ0 = [0.16666667 0.1969697  0.13636364 0.12121212 0.24242424
0.13636364]
```

3. Compute the new set of observation probabilities Pr(ˆe|X) for each of the three observation values. Would you say that the parameters for e = “empty” are accurate?

```
Pr(e_wall|X) =  [0.5  0.25 0.5  0.25 0.   0.25]
Pr(e_hash|X) =  [0.   0.   0.25 0.25 0.25 0.5 ]
Pr(e_empty|X) =  [0.5  0.75 0.25 0.5  0.75 0.25]

O_e_empty = 1 - e1 - e2 = [1/2, 3/4, 1/4, 1/2, 3/4, 1/4] = Pr(e_empty|X)
Accurate!!
```

4. Compute the two sets of “expected” transition matrices ξ0 and ξ1.

```
xi_0 =
[[0.00231481 0.00173611 0.         0.00231481 0.         0.        ]
 [0.00231481 0.00173611 0.00347222 0.         0.         0.        ]
 [0.         0.00173611 0.00347222 0.         0.         0.        ]
 [0.00231481 0.         0.         0.00231481 0.         0.        ]
 [0.         0.00173611 0.         0.00231481 0.         0.00520833]
 [0.         0.         0.         0.         0.         0.00520833]]

xi_1 =
[[0.         0.         0.         0.00694444 0.         0.        ]
 [0.         0.         0.00347222 0.         0.00347222 0.        ]
 [0.         0.         0.00694444 0.         0.         0.        ]
 [0.         0.         0.         0.00347222 0.00347222 0.        ]
 [0.         0.         0.         0.         0.         0.        ]
 [0.         0.         0.         0.         0.00347222 0.00694444]]
```

5. Compute the updated transition matrix Pr(X ˆ t |Xt−1) using your results above.

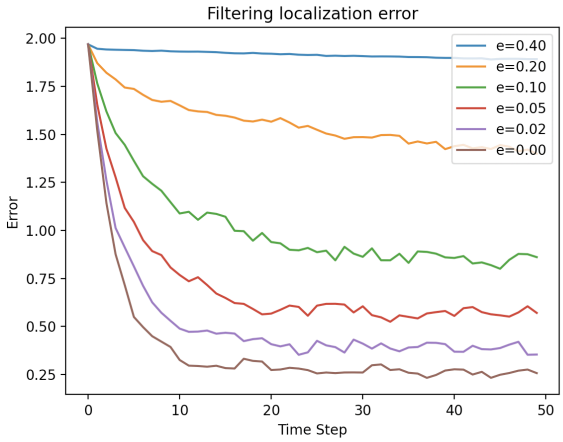
```
Pr(X ^ t |X_t-1) =
[[0.17391304 0.13043478 0.         0.69565217 0.         0.        ]
 [0.16       0.12       0.48       0.         0.24       0.        ]
 [0.         0.14285714 0.85714286 0.         0.         0.        ]
 [0.2        0.         0.         0.5        0.3        0.        ]
 [0.         0.1875     0.         0.25       0.         0.5625    ]
 [0.         0.         0.         0.         0.22222222 0.77777778]]
```

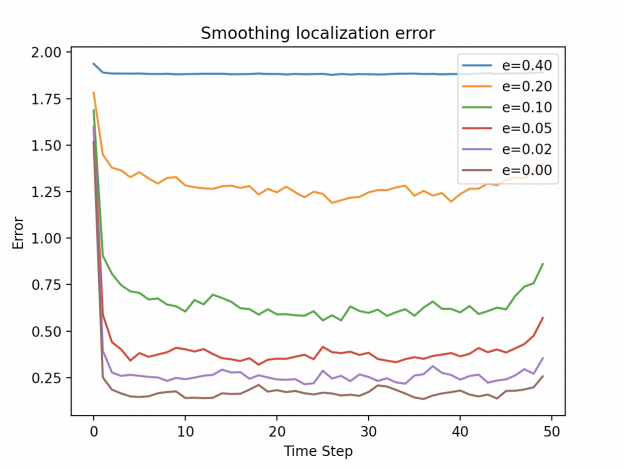
```
beta_2 = np.ones(6)
beta_1 = np.matmul((beta_2*O_e2), P.T)
beta_0 = np.matmul((beta_1*O_e1), P.T)
print("beta_2 = ", beta_2)
print("beta_1 = ", beta_1)
print("beta_0 = ", beta_0)

alpha_0 = Pr_X0
alpha_1 = np.matmul(alpha_0.T, P) * O_e1
alpha_2 = np.matmul(alpha_1.T, P) * O_e2

gamma_0 = alpha_0 * beta_0 / (alpha_0 * beta_0).sum()
gamma_1 = alpha_1 * beta_1 / (alpha_1 * beta_1).sum()
gamma_2 = alpha_2 * beta_2 / (alpha_2 * beta_2).sum()

print("gamma_0 = ", gamma_0)
print("gamma_1 = ", gamma_1)
print("gamma_2 = ", gamma_2)
print("Pr(^X0) = γ0 = ", gamma_0)

O_wall = np.array([1/2, 1/4, 1/2, 1/4, 0, 1/4])
O_empty = 1 - O_wall - O_e2
O_hash = O_e2

expected_wall = gamma_0 * O_wall + gamma_1 * O_wall + gamma_2 * O_wall
expected_empty = gamma_0 * O_empty + gamma_1 * O_empty + gamma_2 * O_empty
expected_hash = gamma_0 * O_hash + gamma_1 * O_hash + gamma_2 * O_hash

Pr_e_wall = expected_wall / (gamma_0 + gamma_1 + gamma_2)
Pr_e_empty = expected_empty / (gamma_0 + gamma_1 + gamma_2)
Pr_e_hash = expected_hash / (gamma_0 + gamma_1 + gamma_2)

print("Pr(e_wall|X) = ", Pr_e_wall)
print("Pr(e_hash|X) = ", Pr_e_hash)
print("Pr(e_empty|X) = ", Pr_e_empty)

xi_0 = np.dot(np.dot(np.dot(np.diag(alpha_0), P), np.diag(O_e1)),
np.diag(beta_1))
xi_1 = np.dot(np.dot(np.dot(np.diag(alpha_1), P), np.diag(O_e2)),
np.diag(beta_2))
print("xi_0 = ", xi_0)
print("xi_1 = ", xi_1)

updated_transition = xi_0 + xi_1
updated_transition = updated_transition / updated_transition.sum(axis=1,
keepdims=True)
print("Pr(X ^ t |X_t-1) = ", updated_transition)
```

Problem 4.

1. Inspect the results of mode 0 with no additional parameters, and do the same with a similar set of ϵ values for mode 1 (e.g., 0.0, 0.1, and 0.4). How does the value of ϵ affect the performance of each inference algorithm? Explain how the animations of mode 1 verify your hypotheses.

```
python main.py -m 0
```






```
python main.py -m 1 -e 0.0
```


```
python main.py -m 1 -e 0.1
```


```
python main.py -m 1 -e 0.4
```

Results of Mode 0:

Error decreases in general as time goes on.
Error is the smallest when e = 0.0 for both filtering and smoothing.
As e increases, the observations become noisier.Both filtering and smoothing are expected to have higher errors.

Animations of mode 1:

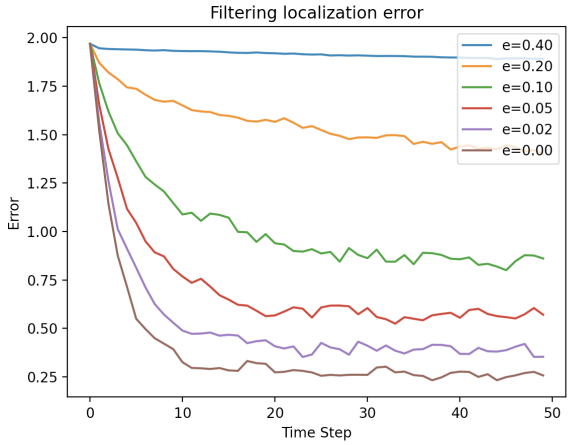
For both e = 0.0 and 0.1, the algorithm is highly confident about the agent's location. The belief is concentrated at the true location of the agent.

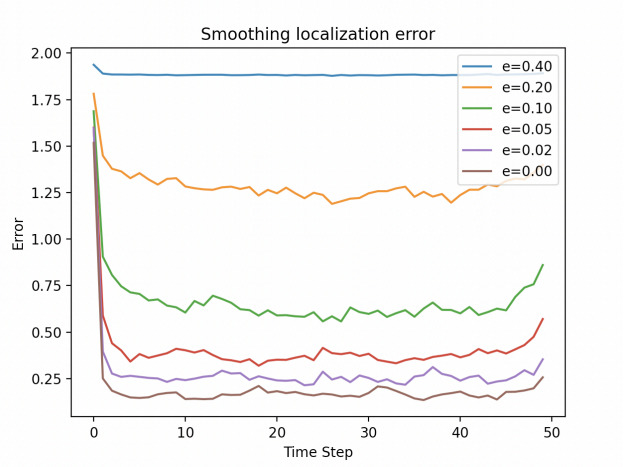
For e = 0.4, significant noise causes the belief to spread widely across the grid, indicating high uncertainty. The brightest areas may still indicate where the agent is likely to be, but the confidence is much lower.

This supports the hypothesis that low e values suggest the sensors are almost always correct, leading to strong and accurate belief states. Conversely, high values suggest the sensors are often incorrect, which leads to diffuse and uncertain belief states. The performance of each inference algorithm is therefore inversely proportional to the value of e.

2. Looking at mode 0, which inference algorithm generally has better localization error over time and why? Also explain the differences in the shapes of the error curves over time. Please include the two localization plots in your writeup.

python main.py -m 0



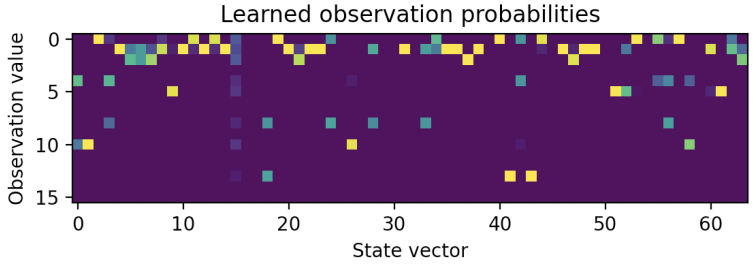


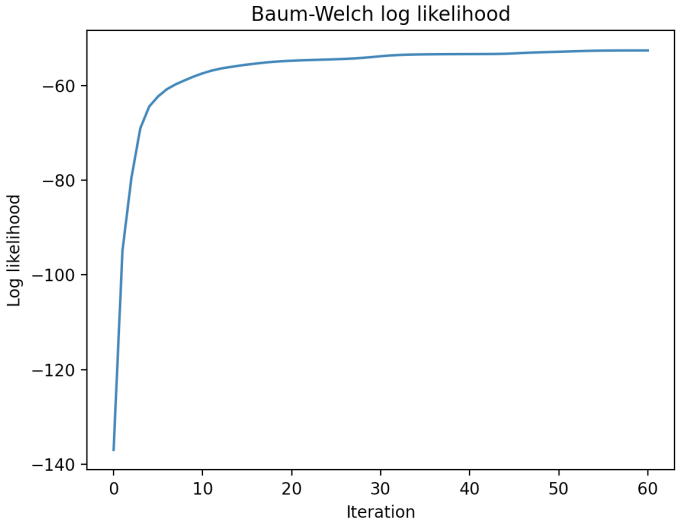
In general, the smoothing algorithm tends to have a lower localization error compared to the filtering algorithm. And larger e value results in larger localization error for both algorithms.

For filtering algorithm, the error starts at 2 for all e(s), while for smoothing algorithm, the initial error depends on the e we choose. Higher e results in higher initial error. While filtering localization error shows a gradual decline of localization error as more observations are incorporated, error of the smoothing algorithm drops more quickly because it immediately uses future observations to improve its estimates.

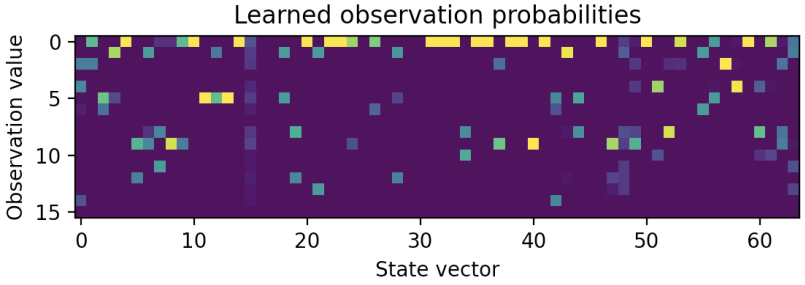
3. Run mode 2 for a few different values of ϵ (e.g., 0.0, 0.1, and 0.4) and take a look at the log likelihood curves of the Baum-Welch learning experiment. Give an intuitive explanation for what this measures (do not just repeat the definition log Pr(e1:T)), and explain whether the shape of the curve indicates whether the learning algorithm is successful. Please include one of the log likelihood plots in your writeup.

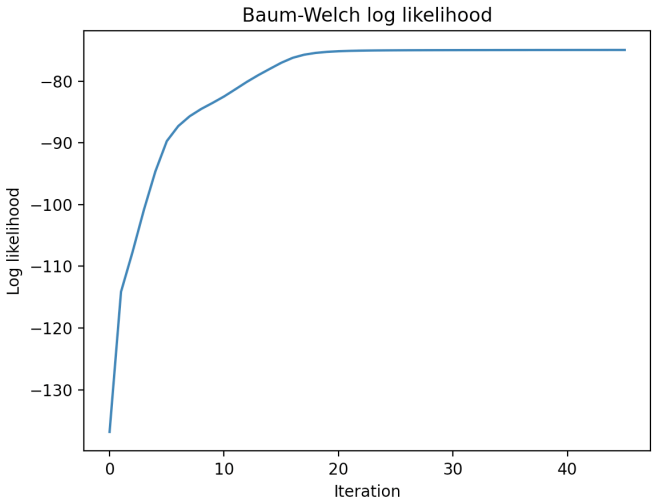
```
python main.py -m 2 -e 0.0
```



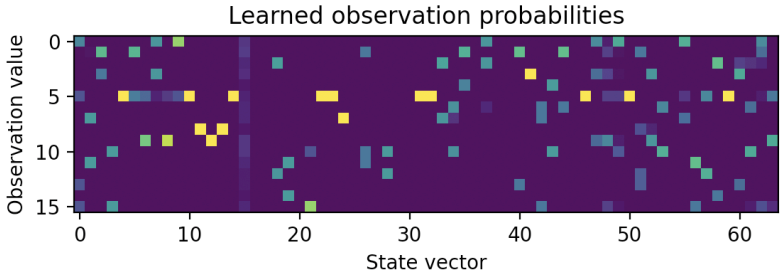


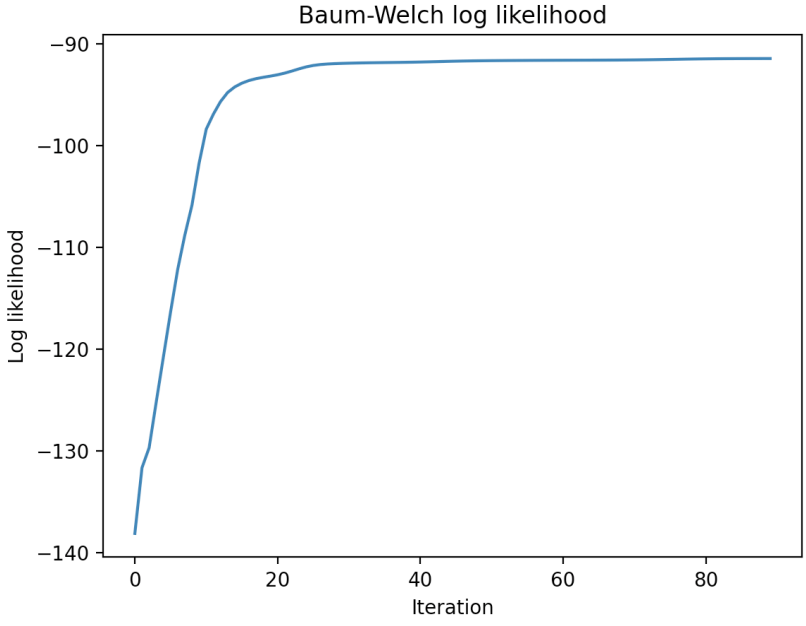

```
python main.py -m 2 -e 0.1
```






```
python main.py -m 2 -e 0.4
```





The log likelihood is a measure of how well the model explains the data. It represents the probability of the observed sequence given the current model parameters.

Initially the model fits poorly to the data. However, it learns fast and results in rapid increase and get closer to the optimal values that best explain the observed data. The curve then becomes flatter, it suggests that the algorithm has converged and the parameters has stabalized. The model has learned to the best of its ability.

The shape of the curves indicate that the learning process is successful as it is both effective and stable. There is a monotonic increase in log likelihood, indicating that the model is continuously improving in explaining the observation sequence. It also shows a trend of converging, without having large fluctuations.