Homework 4: The Hunt for the USS *Scorpion*

HMMs, forwards-backwards, variable elimination

STATS348, UChicago, Spring 2025

Your name here:

Hours spent:

(Please let us know how many hours in total you spent on this assignment so we can calibrate for future assignments. Your feedback is always welcome!)

Open in Colab

Instructions

The purpose of this homework is to apply the ideas in lectures 7-8:

- probabilistic graphical models
- hidden Markov models (HMMs)
- variable elimination
- the sum-product (SP) algorithm
- forwards-backwards algorithm (i.e., SP for HMMs).

You will also learn how to **use numpy and numpy.ndarrays** as an alternative to PyTorch and torch.Tensors for vectorized computation. In addition to learning numpy, you will also see examples using the useful Python libraries pandas and scipy.stats.

Assignment is due **Sunday May 4, 11:59pm** on GradeScope.

Setting

You are back to searching for the missing submarine, the USS *Scorpion*.

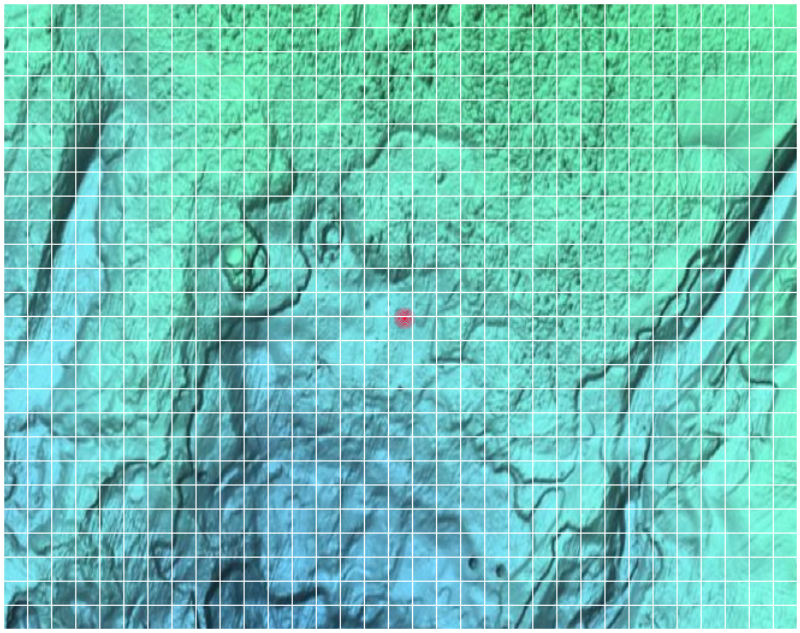
As a reminder, it is May 1968 and the USS *Scorpion* has just disappeared somewhere in the Atlantic Ocean, likely off the coast of Spain. You are the lone statistician on board the USS *Mizar*, which has been dispatched to find the missing submarine. Your job is to guide the search as best you can, given the data at your disposal.

The search grid

The search grid will play a central role in this episode. The Navy has divided the search into a (33×26) grid of square-mile cells. The center of the grid is the rough location of loud acoustic event, likely an explosion, that was recorded around the time when the USS *Scorpion* disappeared. The search grid, overlayed on a topographic map of the ocean floor is displayed below; the red dot is the rough location of the acoustic event.

In [ ]:
```
from hw_utils import plot_scorpion_search_grid, get_scorpion_search_grid
grid, img = get_scorpion_search_grid()
plot_scorpion_search_grid()
```

Out[ ]:



Cell coordinates versus cell indices

There are $K = 33 \times 26 = 858$ total cells in the grid. We can number each cell $k \in \{1, \ldots, K\}$. Each cell therefore has an index k as well as coordinates (x, y) on the grid.

The `ravel_multi_index` and `unravel_index` functions let us map between these two. See below for an explainer on these `numpy` functions.

In [ ]:
```
import numpy as np
import numpy.random as rn

xy = (16, 12)
k = np.ravel_multi_index(xy, dims=(33, 26))
print(f'Cell {k} is in location {xy}')
```

```
k = 4
xy = np.unravel_index(k, shape=(33, 26))
print(f'Cell {k} is in location {xy}')

# this indexing system is based on the "ravel" operation

# consider the following 4d numpy array of random numbers
shape = (4,5,6,7)
arr = rn.random(shape)

# we can ravel this array, so that it is 1d
arr_raveled = arr.ravel()
assert arr_raveled.shape == (np.prod(shape),)

# the functions above let us map between the raveled index and the original multi-index
multi_index = (1, 2, 0, 5)
ravel_index = np.ravel_multi_index(multi_index, shape)
assert arr[multi_index] == arr_raveled[ravel_index]

ravel_index = 50
multi_index = np.unravel_index(ravel_index, shape)
assert arr[multi_index] == arr_raveled[ravel_index]
```

```
Cell 428 is in location (16, 12)
Cell 4 is in location (0, 4)
```

Sonar blips

An unnamed intelligency agency has reported to us that its underwater listening stations detected an anomalous number of sonar blips occurring directly after the loud explosion in the same general area. The sonar blips may correspond to the USS *Scorpion*'s final movements after some catastrophic accident befell it.

Based on the sonar data, and on the captain's expert opinion, we believe the *Scorpion* was active for some period between 1-24 hours after the explosion, after which it likely sank. Define the following quantities:

$t = 0$	time of the explosion
$T = 24$	max num. of hours before the sub sank
$\tau \in \{1, \ldots, T\}$	(unknown) hour of sinking
$Z_t \in \{1, \ldots, K\}$	what cell the sub was in at hour t
$Z_T \in \{1, \ldots, K\}$	the sub's final resting place (where it is now)

Possible scenarios

We convened a panel of submarine experts to imagine what might have happened in the hours between the explosion and when the sub ultimately sank.

They imagined 6 possible scenarios that could have plausibly occurred.

Scenario s	**Description**	**Expected hours alive** $\mathbb{E}[\tau \mid s]$	**Prior probability of scenario** $P(s)$
flounder	The *Scorpion* suffered a catastrophic malfunction; it moved aimlessly before sinking quickly.	4	0.35
drift	The *Scorpion* suffered a catastrophic malfunction; it drifted East with the tide before sinking quickly.	6	0.2
reverse course	The *Scorpion* suffered a major malfunction; it reversed course back to base (Northeast), and sailed for a while before sinking.	12	0.15
seek shallow	The *Scorpion* suffered a major malfunction; it sought shallower waters, and sailed for a while before sinking.	6	0.1
evade	The *Scorpion* was seriously damaged by an enemy attack; it took evasive maneuvers, moving unpredictably and changing course often, but sank soon after.	4	0.1
seek ridge	The *Scorpion* was lightly damaged by an enemy attack; it manuevered quickly into an underwater ridge for cover, but collided with the narrow ridge and sank.	12	0.1

The experts have assigned each scenario a prior probability $P(S = s)$ for $s \in \mathcal{S} \equiv \{\text{flounder}, \text{evade}, \ldots\}$.

They have also constructed a prior over the hour-of-sinking τ under each scenario. Since they strongly believe the sub could not have survived longer than 24 hours, the prior is binomial with $T = 24$:

$$P(\tau \mid s) = \text{Binom}(\tau;\, T,\, \rho_s), \quad \tau \in \{0, 1, 2, \ldots, T\} \tag{1}$$

where $\rho_s \in (0, 1)$ and $\mathbb{E}[\tau \mid s] = T\rho_s$

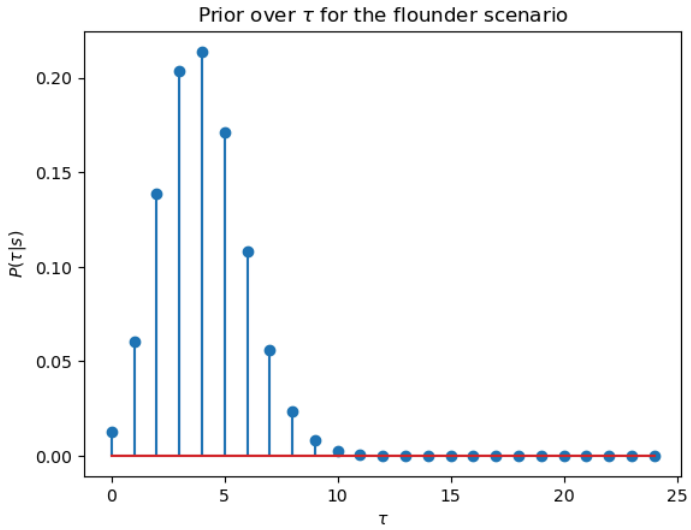
We have implemented the expert's prior in the code below. The function `get_tau_prior` takes T and $(\rho_s)_{s \in \mathcal{S}}$ and uses closure to define and return a conditional function `tau_prior` which will return $P(\tau \mid s)$ for an input scenario s.

In [ ]:
```
import pandas as pd
import scipy.stats as st

df_scenarios = pd.DataFrame()
df_scenarios['scenario'] = ['flounder', 'drift', 'reverse_course', 'seek_shallow', 'evade', 'seek_ridge']
df_scenarios['rho_s'] = [4./24, 6./24, 12./24, 6./24, 4./24, 12./24]
df_scenarios['p(s)'] = [0.35, 0.2, 0.15, 0.1, 0.1, 0.1]

def get_tau_prior(max_timesteps, rho_dict):
    def tau_prior(scenario):
        rho = rho_dict[scenario]
        return st.binom(max_timesteps, rho).pmf(np.arange(max_timesteps+1))
    return tau_prior

rho_dict = df_scenarios.set_index('scenario').to_dict()['rho_s']
tau_prior = get_tau_prior(max_timesteps=24, rho_dict=rho_dict)
```

You can interact with `tau_prior` in the following way.

In [ ]:
```
import matplotlib.pyplot as plt
P_tau_given_flounder = tau_prior('flounder')
assert P_tau_given_flounder.shape == (25,) and np.allclose(P_tau_given_flounder.sum(), 1)

_ = plt.stem(P_tau_given_flounder)
_ = plt.xlabel('$\\tau$')
```

```
_ = plt.ylabel('$P(\\tau | s)$')
_ = plt.title('Prior over $\\tau$ for the flounder scenario')
```



A scenario-based Markov model

For each of the 6 scenarios, the experts have also put together a simple model of the *Scorpion*'s likely movements while it was still alive. Each model is a *Markov model*, which posits the following transition behavior that conditions on the scenario s and time-of-death τ:

For $t = 1, \ldots, T$:

$$P_{s,\tau}(Z_t = k \mid Z_{t-1} = j) = \begin{cases} \Lambda_s(j,k) & \text{if } t \leq \tau \\ 0 & \text{if } t > \tau \text{ and } j \neq k \\ 1 & \text{if } t > \tau \text{ and } j = k \end{cases} \tag{2}$$

For $t = 0$:

$$P(Z_0 = k) = \pi_0(k) \tag{3}$$

$\pi_0 \in (0,1)^K$ is the *initial distribution* over cells where the *Scorpion* may have been at $t = 0$.

$\Lambda_s \in (0,1)^{K \times K}$ is a row-stochastic *transition matrix*, where the rows sum to 1:

$$\sum_{k=1}^{K} \Lambda_s(j,k) = 1, \ \forall j \tag{4}$$

There is a different transition matrix Λ_s for each scenario s; each one describes a different pattern of movement. You can load them using `pickle` below.

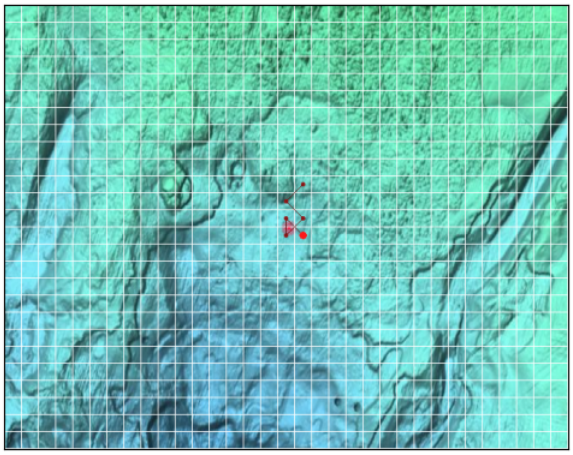
In [ ]:
```
import pickle
with open('dat/transition_matrices.pkl', 'rb') as f:
    transition_matrices = pickle.load(f)

print(transition_matrices.keys())
assert all(transition_matrices[s].shape == (858, 858) for s in transition_matrices.keys())
```

```
dict_keys(['flounder', 'evade', 'drift', 'reverse_course', 'seek_shallow', 'seek_ridge'])
```

We also provide functions to simulate paths with a given transition matrix and to plot the path on the search grid.

You can play with them to see what kinds of paths are typical under each scenario.

In [ ]:
```
from hw_utils import plot_scorpion_path

def sample_markov(transition_matrix,  n_timesteps=10, start_cell=428):
    """ Sample a path from a Markov chain with the given transition matrix.

    Parameters
    ----------
    transition_matrix : np.ndarray
        A square matrix of shape (n_cells, n_cells) where n_cells is the number of cells in the grid.
        The (i,j) entry of this matrix is the probability of transitioning from cell i to cell j.
    n_timesteps : int
        The number of timesteps to sample.
    start_cell : int
        The index of the cell to start the path from.
        Default value is 428, corresponding to (16,12) on the grid, where the explosion occurred.
    """
    n_cells = transition_matrix.shape[0]

    path = [start_cell]
    for _ in range(n_timesteps):
        next_cell = rn.choice(n_cells, p=transition_matrix[path[-1]])
        path.append(next_cell)
    return path

path = sample_markov(transition_matrices['evade'], n_timesteps=12)
path = [np.unravel_index(k, shape=(33, 26)) for k in path]  # convert to (x,y) coordinates
plot_scorpion_path(path, show=True)
```



You can load the initial distribution $\pi_0 \in (0,1)^K$ below using `numpy.load`.

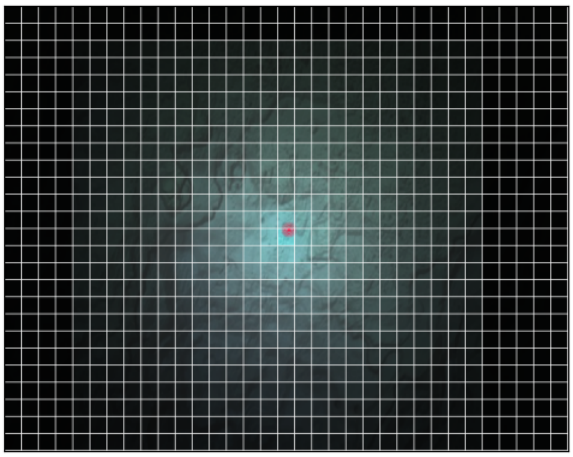
We have provided a function to plot probability distributions over the search grid.

This function will darken cells proportional to their probability under the distribution.

For instance, if $\pi_0(k) \approx 0$, then cell k will be black.

The function requires you to first `reshape` the distribution into a 2d array where the (row,col) indices match the (x, y)-coordinates on the grid.

In [ ]:
```
from hw_utils import plot_dist_on_scorpion_search_grid

initial_dist_K = np.load('dat/initial_dist.npy') # \pi_0
assert initial_dist_K.shape == (858,)

plot_dist_on_scorpion_search_grid(dist=initial_dist_K.reshape((33, 26)), show=True)
```



Sonar likelihood

As mentioned before, underwater listening stations picked up an anomalous pattern of sonar blips in the hours after the explosion.

There were D listening devices in the area. Let $u_d \in \{1, \ldots, K\}$ be the cell location of the d^{th} device.

Assume the following conditional likelihood for $t \in \{1, \ldots, T\}$ and sonar station $d \in \{1, \ldots, D\}$. This likelihood conditions on the location $Z_t = z_t$ of the sub at time t, as well as on the hour-of-sinking τ.

$$P(x_{t,d} \mid Z_t = k, \tau) = \begin{cases} \text{Pois}(x_{t,d};\, \mu_{d,0} + \mu_d(z_t)) & \text{if } t \le \tau \\ \text{Pois}(x_{t,d};\, \mu_{d,0}) & \text{if } t > \tau \end{cases} \tag{5}$$

where $\mu_d(z_t) = \gamma_d \exp\left(-\frac{1}{2}\,\text{dist}(z_t, u_d)\right)$. Here, $\text{dist}(z_t, u_d)$ is the distance between the sub at t and listening cell d, and γ_d is a parameter that is fixed and known.

$\mu_{0,d}$ is the background rate of sonar blips for device d; it is also fixed and known.

It is also useful to define the *likelihood vector* at time t, conditional on τ; this is:

$$\boldsymbol{\ell}_t^\tau = \begin{bmatrix} P_\tau(\boldsymbol{x}_t \mid Z_t = 1) \\ P_\tau(\boldsymbol{x}_t \mid Z_t = 2) \\ \vdots \\ P_\tau(\boldsymbol{x}_t \mid Z_t = K) \end{bmatrix} \tag{6}$$

An element of it is $\ell_t^\tau(k) = P_\tau(\boldsymbol{x}_t \mid Z_t = k) \equiv \prod_{d=1}^D P(x_{t,d} \mid Z_t = k, \tau)$.

For your convenience, we have implemented the following function `get_sonar_likelihood`. It takes in the grid shape, the listening device locations $(u_d)_{d=1}^D$, and the parameters $(\mu_{0,d}, \gamma_d)_{d=1}^D$, and then uses closure to define and return a conditional function `sonar_likelihood` that will return a 2-d table $(\boldsymbol{\ell}_t^\tau)_{t=1}^T$ for a given τ.

(You do not need to study the details of this function as long as you understand the input and output of `sonar_likelihood`.)

In [ ]:
```
import scipy.stats as st

def distance_between_cells(k1, k2, grid_shape=(33, 26)):
    x1, y1 = np.unravel_index(k1, grid_shape)
    x2, y2 = np.unravel_index(k2, grid_shape)
```

```
    return np.sqrt((x1 - x2)**2 + (y1 - y2)**2)

def get_sonar_likelihood(grid_shape, u_D, mu0_D, gam_D):
    n_stations = len(u_D)
    assert n_stations == mu0_D.size == gam_D.size

    # possible values of Z_t (0-indexed, so 0...K-1)
    n_cells = np.prod(grid_shape)

    # compute distance between each listening station and every other cell in the grid
    distances_DK = np.zeros((n_stations, n_cells))
    for d, k in np.ndindex((n_stations, n_cells)):
        distances_DK[d, k] = distance_between_cells(u_D[d], k)

    # define a likelihood function that will output the likelihood table, for a given tau
    def sonar_likelihood(sonar_data, tau):
        assert sonar_data.ndim == 2
        n_timesteps, n_stations = sonar_data.shape
        assert 0 <= tau <= n_timesteps

        Mu_TKD = np.ones((n_timesteps, n_cells, n_stations)) * mu0_D
        if tau > 0:
            Mu_TKD[:tau] += gam_D * np.exp(-0.5 * distances_DK.T)

        L_TK = st.poisson.pmf(sonar_data[:, None, :], Mu_TKD).prod(axis=-1)

        return L_TK

    return sonar_likelihood
```

We can now load in the parameters of the listening devices $(u_d, \mu_{0,d}, \gamma_d)_{d=1}^D$ and use them to define a likelihood.

In [ ]:
```
arrays = np.load('dat/listening_stations.npz')
# the cell indices of each listening device
u_D = arrays['u_D']
# the background rate of each listening device \mu_{0,d}
mu0_D = arrays['mu0_D']
# the parameter \gamma_{d} for each listening device
gam_D = arrays['gam_D']

# define a likelihood function for these parameters
sonar_likelihood = get_sonar_likelihood((33, 26), u_D, mu0_D, gam_D)
```

And finally, we can load in the sonar data $\boldsymbol{x}_{1:T}$, and evaluate its likelihood for different values of τ:

In [ ]:
```
# load in the sonar data
sonar_data = np.load('dat/sonar_data.npy')
assert sonar_data.shape == (24, len(u_D))

# compute the likelihood table for tau=0
L_TK = sonar_likelihood(sonar_data, tau=0)
assert L_TK.shape == (24, 858)

# compute the likelihood table for tau=24
L_TK = sonar_likelihood(sonar_data, tau=24)
assert L_TK.shape == (24, 858)
```

Question 1: Draw the graphical model [Visualize]

Create a directed graphical model for the joint distribution:

$$p(s, \tau, \boldsymbol{z}_{0:T}, \boldsymbol{x}_{1:T} \mid \theta)$$

You do not need to visualize the parameters $\theta = \{\pi_0, (\Lambda_s, \rho_s)_{s \in \mathcal{S}}, (u_d, \mu_{0,d}, \gamma_d)_{d=1}^D\}$.

You may use the TikZ package, an illustrator like Omnigraffle, or a slides-maker like Keynote/Powerpoint. You can also handwrite and scan it, so long as it is very neat and readable.

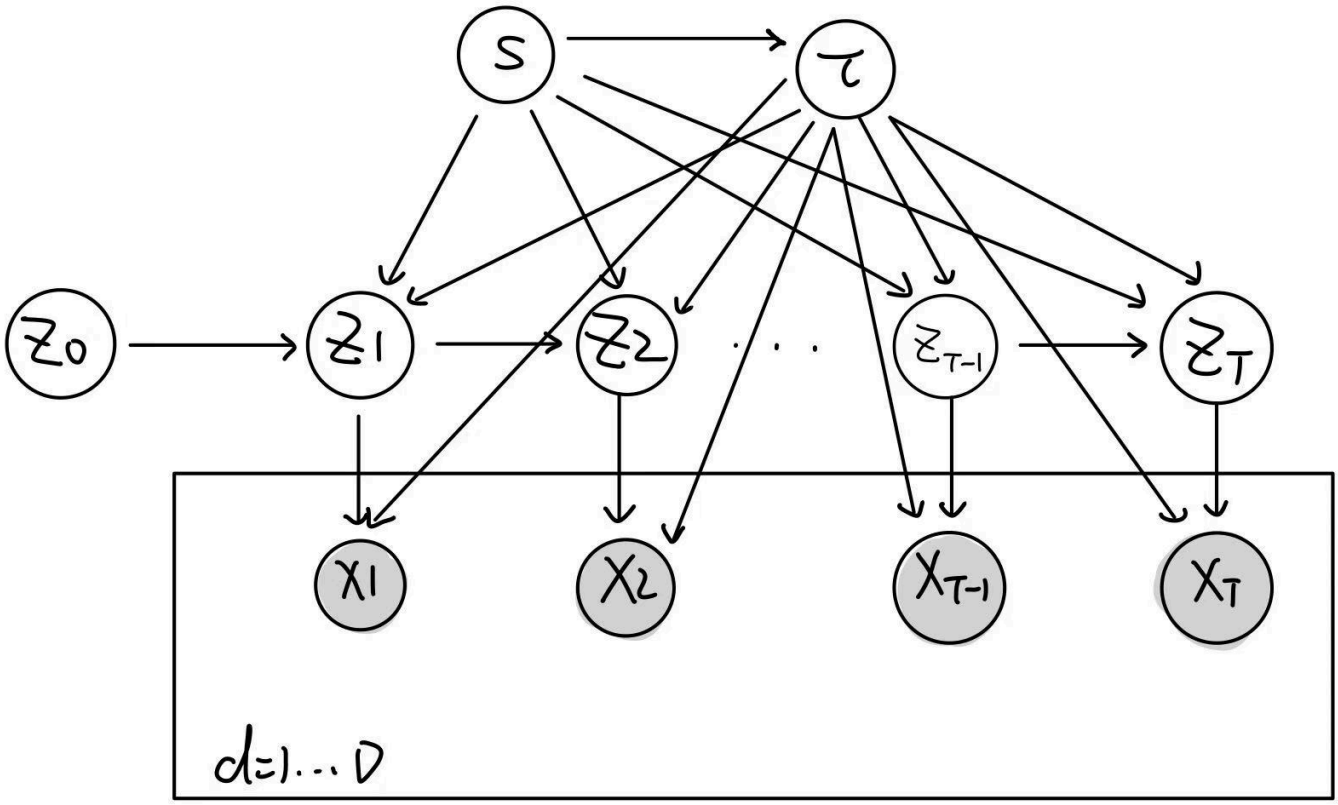
Please insert your figure below:

Using either sofware (e.g., TikZ, Keynote) or **very neat handwriting**, create a probabilistic graphical model that describes the generative process above.

Your PGM should:

- Explicitly show $t \in \{0, 1, 2\}$ and $t \in \{T-1, T\}$, using an ellipse $(\cdots)$ to denote the time points in between
- Use plate notation to denote repeated sampling over d.
- Use shaded circular nodes to denote observed variables.
- Use un-shaded circular nodes to denote latent variables.

Include your image in the space below.



Question 2: Forward pass [Code]

The **forward pass** in this case can be defined by the following recursion. For $t = 2, \ldots, T$:

$$\alpha_t^{\tau,s}(k) = \sum_{j=1}^{K} P_{s,\tau}(Z_t = k \mid Z_{t-1} = j)\, P_\tau(\boldsymbol{x}_{t-1} \mid Z_{t-1} = j)\, \alpha_{t-1}^{\tau,s}(j)$$

where the base case of the recursion is $t = 1$:

$$\alpha_1^{\tau,s}(k) = \sum_{j=1}^{K} P_{s,\tau}(Z_1 = k \mid Z_0 = j)\, P(Z_0 = j)$$

(There are other ways to define the forward pass, but this is one.)

Implement the **forward pass** for a given scenario s and hour-of-sinking τ, by filling in the code for the method below.

- When implementing the forward pass, do not use nested for-loops. Your code should contain **only a single for-loop over** t and otherwise involve only matrix-vector operations.
- **NOTE**: it is sometimes convenient to compute the forward pass an extra step to $t = T + 1$ (e.g., for **predictive** problems, involving Z_{T+1}); the method below takes as an argument a flag indicating whether to compute to T or $T + 1$.
- **NOTE:** Python **zero-indexes**, so the mathematical expression $t = 1$ will translate to the software expression `t=0`.

In [ ]:
```python
def forward_pass(tau, transition_matrix_KK, initial_dist_K, likelihood_TK, predictive=False):
    """ Compute the forward pass of the HMM for a given tau and scenario.

    Parameters
    ----------
    tau : int
        The time-of-sinking for the USS Scorpion: 0...24.
    transition_matrix_KK : np.ndarray
        A (KxK) row-stochastic matrix for a given scenario.
    initial_dist_K : np.ndarray
        A (K,) vector representing the initial distribution over cells.
    likelihood_TK : np.ndarray
        A (TxK) matrix of likelihood values for the sonar data, for a given tau.
    predictive : bool
        If True, compute the forward messages to T+1.
        If False, compute the forward messages to T.
    Returns
    -------
    alpha_TK : np.ndarray
        A (TxK) or (T+1xK) matrix of forward messages
    """
    T, K = likelihood_TK.shape
    assert 0 <= tau <= T
    T_ = T + 1 if predictive else T

    alpha_TK = np.zeros((T_, K))

    # Your code here
    alpha_TK[0] = initial_dist_K @ transition_matrix_KK

    for t in range(1, T_):
```

```
        if t <= tau:
            alpha_TK[t] = (alpha_TK[t - 1] * likelihood_TK[t - 1]) @ transition_matrix_KK
        else:
            alpha_TK[t] = alpha_TK[t - 1] * likelihood_TK[t - 1]

    return alpha_TK
```

Question 3: Filtering probabilities [Math, code]

Provide a way to compute the following **filtering probability** using the forward pass $\alpha_t^{\tau,s}(k)$ values.

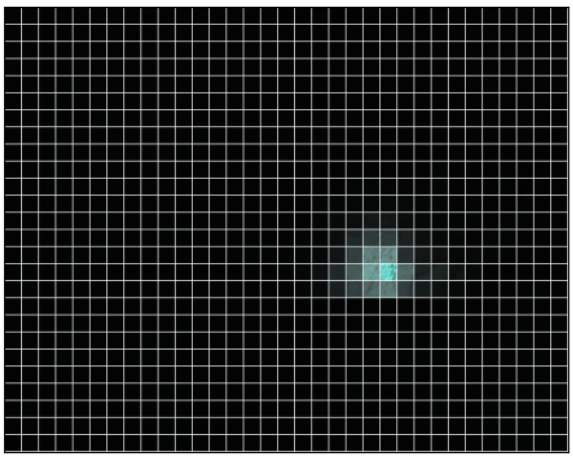
$$P(Z_T = k \mid s, \tau, \boldsymbol{x}_{1:T}) = ?$$

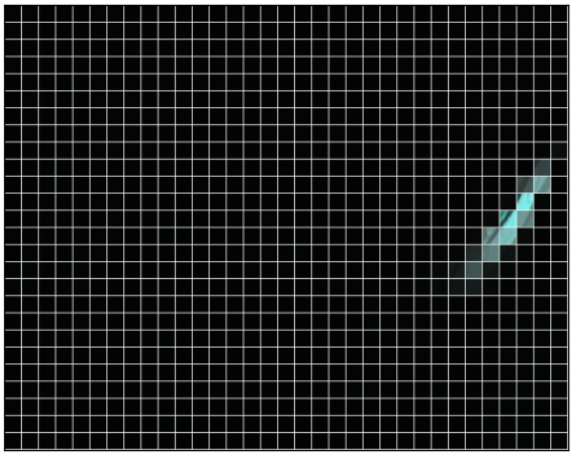
Your answer here:

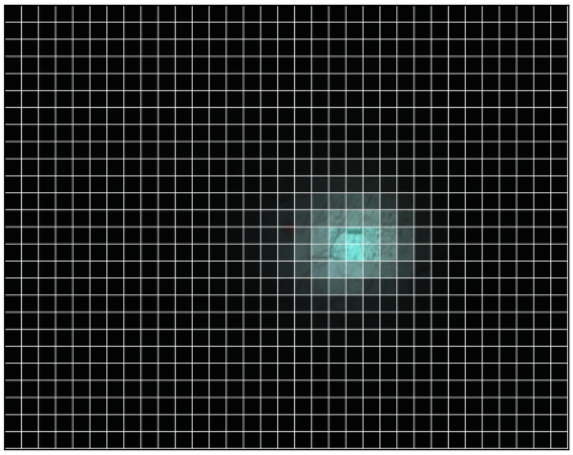
$$\begin{aligned} P(Z_T = k \mid s, \tau, \boldsymbol{x}_{1:T}) &= \frac{P(Z_T = k, x_{1:T} | s, \tau)}{\sum_j P(Z_T = j, x_{1:T} | s, \tau)} \\ &= \frac{P(x_{1:T|z_T=k,s,\tau}) \cdot P(z_T = k | s, \tau)}{\sum_j P(x_{1:T|z_T=j,s,\tau}) \cdot P(z_T = j | s, \tau)} \\ &= \frac{\alpha_T^{\tau,s}(k) \cdot l_T^{\tau}(k)}{\sum_j \alpha_T^{\tau,s}(j) \cdot l_T^{\tau}(j)} \end{aligned}$$

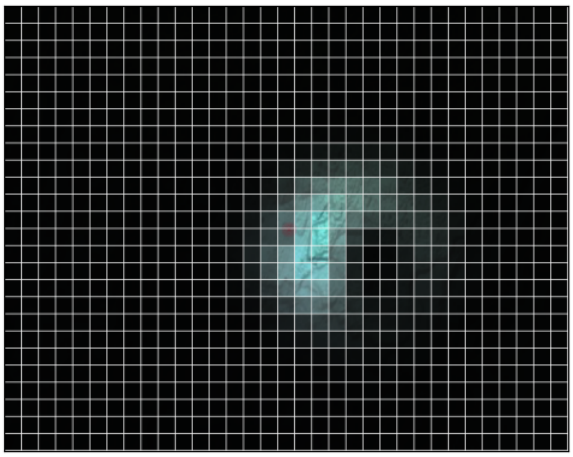
Then use your implementation of the forward pass to compute and visualize $P(Z_T = k \mid s, \tau, \boldsymbol{x}_{1:T})$ for the four combinations of (s, τ) given. (Your code should use `forward_pass` to compute `P_Z`.)

In [ ]:
```
for s in ['seek_ridge', 'flounder']:
    for tau in [5, 24]:

        # Your code here.
        transition_matrix_KK = transition_matrices[s]
        L_TK = sonar_likelihood(sonar_data, tau)
        alpha_TK = forward_pass(tau, transition_matrix_KK, initial_dist_K, L_TK)  # shape: (T, K)

        numerator = alpha_TK[-1] * L_TK[-1]
        # Normalize the final alpha to get P(Z_T = k | s, tau, x_{1:T})
        P_Z = numerator/numerator.sum()  # shape: (K,)

        assert P_Z.shape == (858,) and np.allclose(P_Z.sum(), 1)
        plot_dist_on_scorpion_search_grid(P_Z.reshape((33, 26)), show=True)
```









Question 4: Backward pass [Code]

The backward pass is defined by the following recursion for $t = T-1, \ldots, 0$:

$$\beta_t^{\tau,s}(k) = \sum_{j=1}^{K} P_{s,\tau}(Z_{t+1} = j \mid Z_t = k)\, P_\tau(\boldsymbol{x}_{t+1} \mid Z_{t+1} = j)\, \beta_{t+1}^{\tau,s}(j)$$

where the base case of the recursion is for $t = T$:

$$\beta_T^{\tau,s}(k) = 1$$

Implement the backward pass.

- Again, your code should contain **only a single for-loop over** t and otherwise involve only matrix-vector operations.
- Again, be mindful of **zero-indexing**.

In [ ]:
```python
def backward_pass(tau, transition_matrix_KK, likelihood_TK):
    """ Compute the forward pass of the HMM for a given tau and scenario.

    Parameters
    ----------
    tau : int
        The time-of-sinking for the USS Scorpion: 0...24.
    transition_matrix_KK : np.ndarray
        A (KxK) row-stochastic matrix for a given scenario.
    initial_dist_K : np.ndarray
        A (K,) vector representing the initial distribution over cells.
    likelihood_TK : np.ndarray
        A (TxK) matrix of likelihood values for the sonar data, for a given tau.

    Returns
    -------
    beta_TK : np.ndarray
        A (TxK) matrix of backward messages
    """

    # Your code here
    # beta: (T+1, K) t=0...T
    T, K = likelihood_TK.shape
    beta_TK = np.zeros((T+1, K))
    # Base case: beta_T = 1 for all k
    beta_TK[-1,:] = np.ones(K)
    for t in reversed(range(T)):
        # Vectorized update: beta_t(k) = sum_j P(k->j) * likelihood(t+1, j) * beta_{t+1}(j)
        beta_TK[t] = (transition_matrix_KK if t+1 <= tau else np.eye(K)) @ (likelihood_TK[t] * beta_TK[t+1])

    return beta_TK
```

Question 5: Smoothing probabilities [Math, code]

Provide a way to compute the following **smoothing probability** using the forward (α) and backward (β) values.

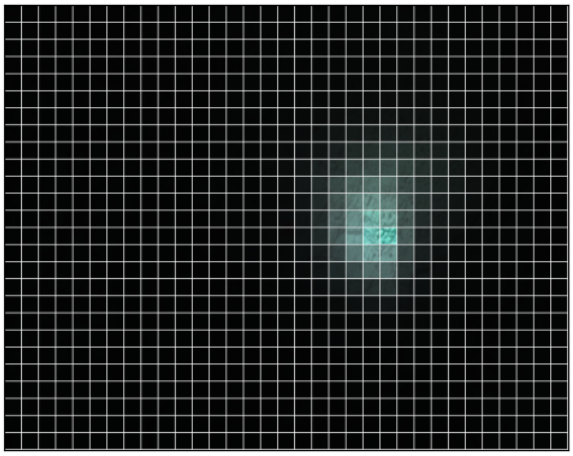
$$P(Z_t = k \mid s, \tau, \boldsymbol{x}_{1:T}) = ?$$

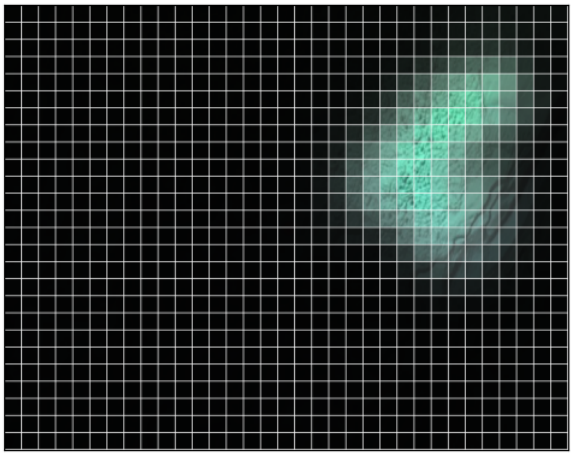
Your answer here:

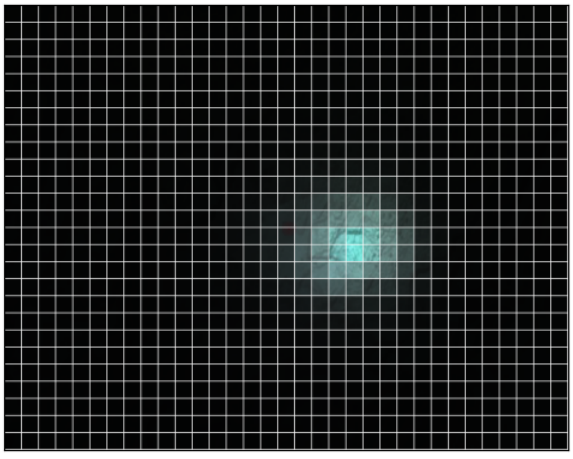
$$\begin{aligned} P(Z_t = k \mid s, \tau, \boldsymbol{x}_{1:T}) &= \frac{P(Z_t = k, x_{1:T} | s, \tau)}{\sum_j p(z_t = j, x_{1:T} | s, \tau)} \\ &= \frac{p(z_t = k, x_{1:t-1} | s, \tau) p(x_t | z_t = k, s, \tau) p(x_{t+1:T} | z_t = k, s, \tau)}{\sum_j p(z_t = j, x_{1:t-1} | s, \tau) p(x_t | z_t = j, s, \tau) p(x_{t+1:T} | z_t = j, s, \tau)} \\ &= \frac{\alpha_t^{\tau,s}(k) l_t^{\tau}(k) \beta_t^{\tau,s}(k)}{\sum_j \alpha_t^{\tau,s}(j) l_t^{\tau}(j) \beta_t^{\tau,s}(j)} \end{aligned}$$

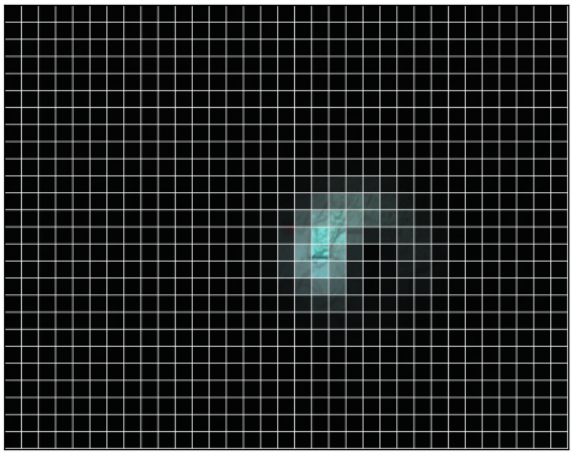
Then use your implementation of the forward and backward pass to compute and visualize the distribution $P(Z_t = k \mid s, \tau, \boldsymbol{x}_{1:T})$ for hour $t = 15$ and the four combinations of (s, τ) given. (Your code should use `forward_pass` to compute `P_Z`.)

In [ ]:
```
tau_vals = [5, 24]
s_vals = ['reverse_course', 'flounder']
for s in s_vals:
    transition_matrix_KK = transition_matrices[s]
    for tau in tau_vals:

        # Your code here.
        # P_Z = ...
        t = 15
        L_TK = sonar_likelihood(sonar_data, tau)
        alpha_TK = forward_pass(tau, transition_matrix_KK, initial_dist_K, L_TK)
        beta_TK = backward_pass(tau, transition_matrix_KK, L_TK)
        numerator = alpha_TK[t-1]*L_TK[t-1]*beta_TK[t]
        P_Z = numerator / sum(numerator)
        plot_dist_on_scorpion_search_grid(P_Z.reshape((33, 26)).copy(), show=True)
```









Question 6: Posterior marginal of final resting place [Math, code]

All of the previous calculations conditioned on a specific scenario s and hour-of-sinking τ.

Provide a way to exactly compute the following probability distribution for any t:

$$P(Z_t = k \mid \boldsymbol{x}_{1:T})$$

Your answer here: when t>0:

$$\begin{aligned} P(Z_t = k \mid \boldsymbol{x}_{1:T}) &= \frac{P(Z_t = k, x_{1:T})}{P(x_{1:T})} \\ &= \frac{\sum_{s,\tau} P(s)P(\tau|s)P_{s,\tau}(Z_t = k, x_{1:T})}{\sum_{s,\tau,j} P(s)P(\tau|s)P_{s,\tau}(Z_t = j, x_{1:T})} \\ &= \frac{\sum_{s,\tau} P(s)P(\tau|s)(\alpha_t^{\tau,s} \cdot l_t^{\tau} \cdot \beta_t^{\tau,s})(k)}{\sum_{s,\tau,j} P(s)P(\tau|s)(\alpha_t^{\tau,s} \cdot l_t^{\tau} \cdot \beta_t^{\tau,s})(j)} \end{aligned}$$

when t=0:

$$\begin{aligned} P(Z_0 = k \mid \boldsymbol{x}_{1:T}) &= \frac{P(Z_0 = k, x_{1:T})}{P(x_{1:T})} \\ &= \frac{\sum_{s,\tau} P(s)P(\tau|s)P_{s,\tau}(Z_0 = k, x_{1:T})}{\sum_{s,\tau,j} P(s)P(\tau|s)P_{s,\tau}(Z_0 = j, x_{1:T})} \\ &= \frac{\sum_{s,\tau} P(s)P(\tau|s)(\Pi_0 \cdot \beta_t^{\tau,s})(k)}{\sum_{s,\tau,j} P(s)P(\tau|s)(\Pi_0 \cdot \beta_t^{\tau,s})(j)} \end{aligned}$$

Now implement your answer, and then run it for $t = 0$ and $t = T$ so that it computes the following two marginals:

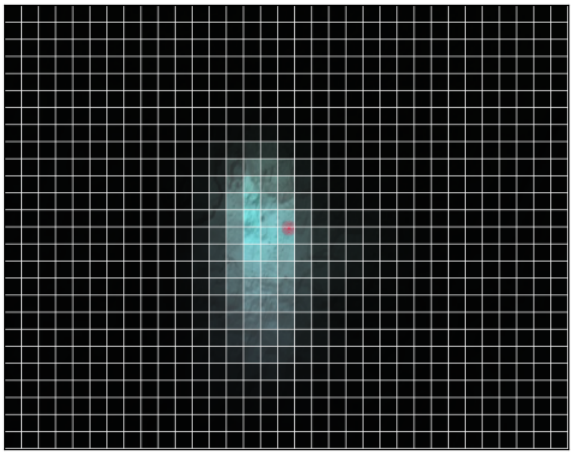
$$P(Z_0 = k \mid \boldsymbol{x}_{1:T})$$

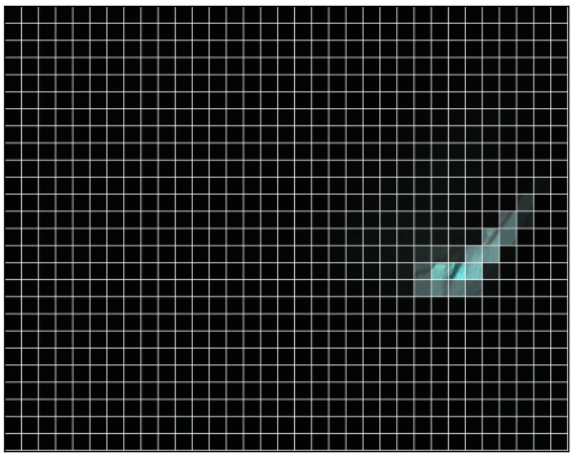
$$P(Z_T = k \mid \boldsymbol{x}_{1:T})$$

which are the posteriors over where the *Scorpion* was initially (Z_0), and as well as its final resting place (Z_T).

- Your code below should involve calls to both `forward_pass` and `backward_pass`.
- It should ultimately produce `posterior_Z_T` and `posterior_Z_0` (which are the two marginals) and then plot them over the grid.
- Make sure both plots are visible.

In [ ]:
```
# Your code here.

ps = df_scenarios['p(s)'].values
scenarios = df_scenarios['scenario'].values
P_0 = np.zeros(858)
P_T = np.zeros(858)

for p_s, s in zip(ps, scenarios):
    transition_matrix_KK = transition_matrices[s]
    ptau = tau_prior(s)
    for tau, p_tau in enumerate(ptau):
        L_TK = sonar_likelihood(sonar_data, tau)
        alpha_TK = forward_pass(tau, transition_matrix_KK, initial_dist_K, L_TK)
        beta_TK = backward_pass(tau, transition_matrix_KK, L_TK)
        P_0 += p_s * p_tau * beta_TK[0,:] * initial_dist_K
        P_T += p_s * p_tau * alpha_TK[-1,:] * beta_TK[-1,:] * L_TK[-1,:]

posterior_Z_0 = P_0 / sum(P_0)
posterior_Z_T = P_T / sum(P_T)

plot_dist_on_scorpion_search_grid(posterior_Z_0.reshape((33, 26)), show=True)
plot_dist_on_scorpion_search_grid(posterior_Z_T.reshape((33, 26)), show=True)
```

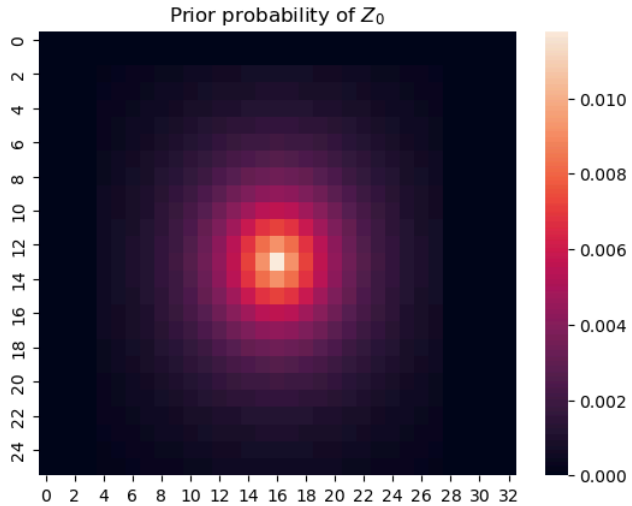


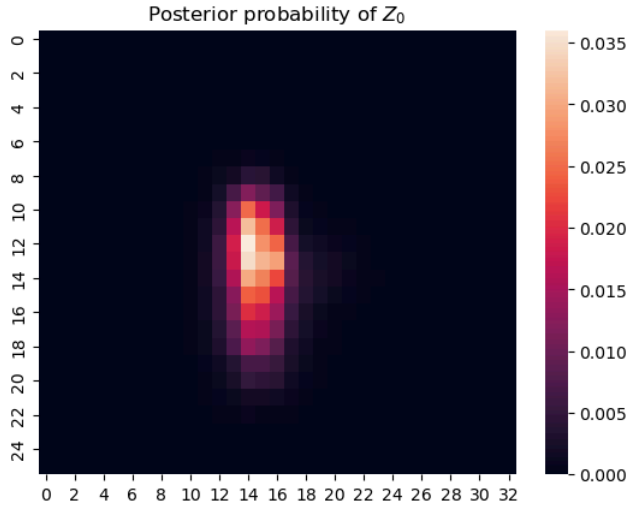


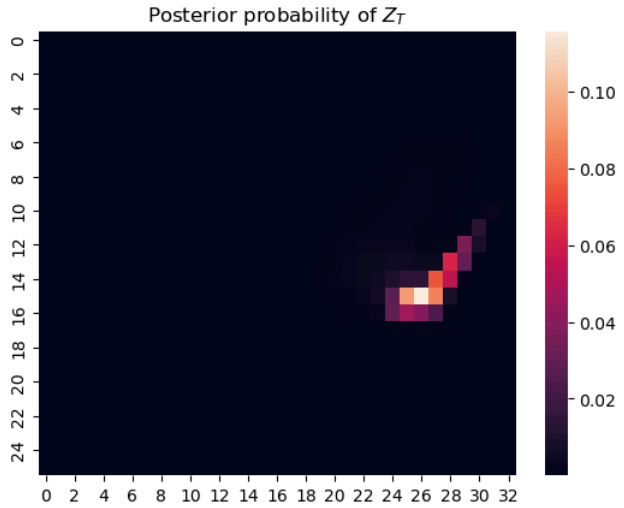
Run the following visualization code so that we see the exact probabilities of the posteriors versus the initial distribution.

This will do the same thing as above, but using `seaborn.heatmap` with a colorbar.

In [ ]:
```
import seaborn as sns
_ = sns.heatmap(np.rot90(initial_dist_K.reshape((33, 26))))
_ = plt.title('Prior probability of $Z_0$')
plt.show()

_ = sns.heatmap(np.rot90(posterior_Z_0.reshape((33, 26))))
_ = plt.title('Posterior probability of $Z_0$')
plt.show()

_ = sns.heatmap(np.rot90(posterior_Z_T.reshape((33, 26))))
_ = plt.title('Posterior probability of $Z_T$')
```







Question 7: Posterior marginal of scenario [Code]

Finally, write code to compute the following probability:

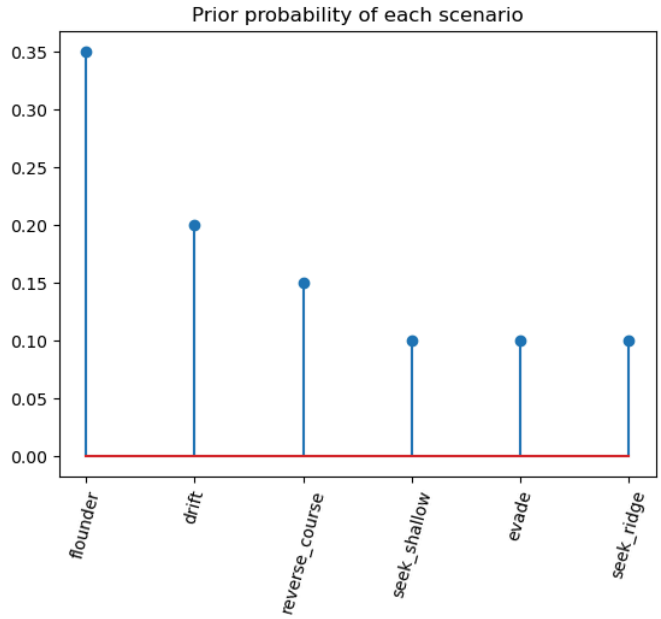
$$P(S = s \mid \boldsymbol{x}_{1:T})$$

which is our posterior over what scenario actually befell the *Scorpion*.

- Your existing code should already be able to compute this. **Re-use your existing code** with as few modifications as possible.
- In the space below, compute `posterior_S`, and then run the visualization code.
- Make sure the indices of `posterior_S` are properly aligned with `prior_S`.

In [ ]:
```
scenarios = df_scenarios['scenario'].values
prior_S = df_scenarios['p(s)'].values

# Your code here.
posterior_S = np.zeros(len(scenarios))
likelihood_norm = 0.0

for s_idx, (p_s, s) in enumerate(zip(prior_S, scenarios)):
    transition_matrix_KK = transition_matrices[s]
    total_likelihood_s = 0.0
    ptau = tau_prior(s)
    for tau, p_tau in enumerate(ptau):
        likelihood_TK = sonar_likelihood(sonar_data, tau)
        alpha = forward_pass(tau, transition_matrix_KK, initial_dist_K, likelihood_TK)
        L_TK = np.sum(alpha[-1])  # scalar
        weight = L_TK * p_s *  p_tau # prior over (s, tau)
        total_likelihood_s += weight
        likelihood_norm += weight
    posterior_S[s_idx] = total_likelihood_s

posterior_S /= likelihood_norm


assert posterior_S.shape == prior_S.shape and np.allclose(posterior_S.sum(), 1)

_ = plt.stem(scenarios, prior_S, use_line_collection=True)
_ = plt.xticks(rotation=75)
plt.title('Prior probability of each scenario')
plt.show()

_ = plt.stem(scenarios, posterior_S, use_line_collection=True)
_ = plt.xticks(rotation=75)
plt.title('Posterior probability of each scenario')
plt.show()
```

```
C:\Users\23677\AppData\Local\Temp\ipykernel_48900\1739825450.py:26: MatplotlibDeprecationWarning: The 'use_line_collection' parameter of stem() was deprecated in Matplotlib 3.6 and will be removed two minor releases later. If any parameter follows 'use_line_collection', they should be passed as keyword, not positionally.
  _ = plt.stem(scenarios, prior_S, use_line_collection=True)
```




```
C:\Users\23677\AppData\Local\Temp\ipykernel_48900\1739825450.py:31: MatplotlibDeprecationWarning: The 'use_line_collection' parameter of stem() was deprecated in Matplotlib 3.6 and will be removed two minor releases later. If any parameter follows 'use_line_collection', they should be passed as keyword, not positionally.
  _ = plt.stem(scenarios, posterior_S, use_line_collection=True)
```