POMDP:s and HMM-filters

Overview of theory and algorithms

Axel Nathanson

December 2020

Relations to Markov Chains

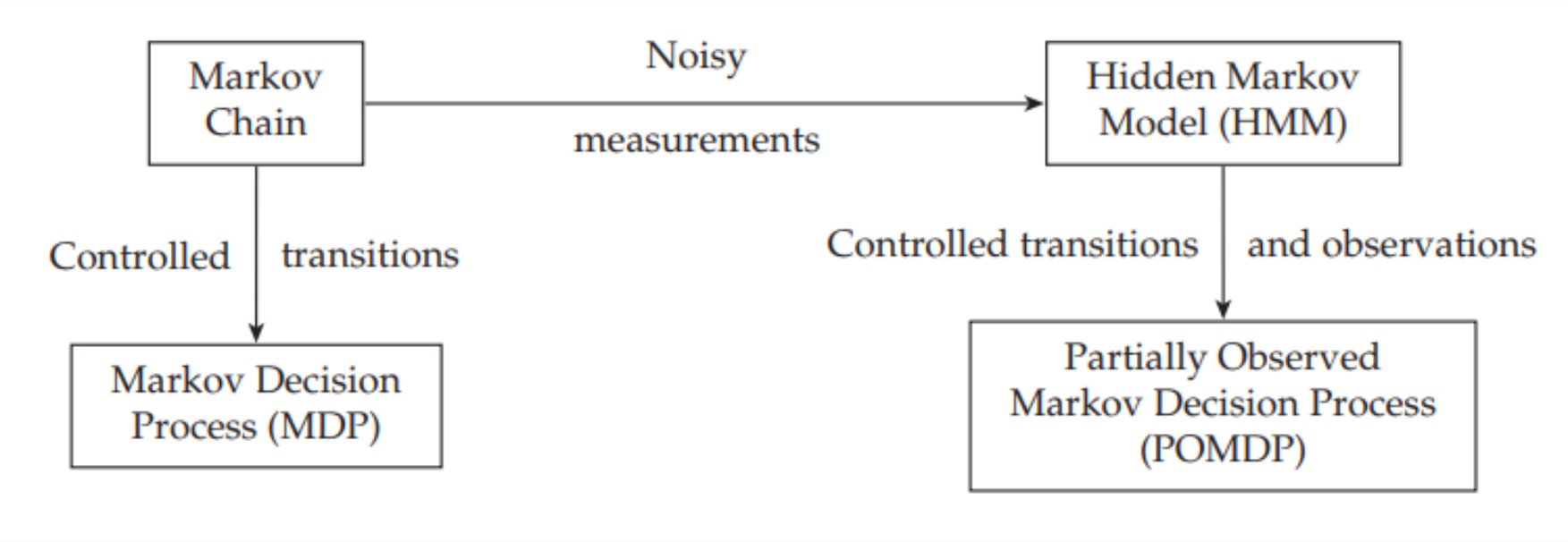
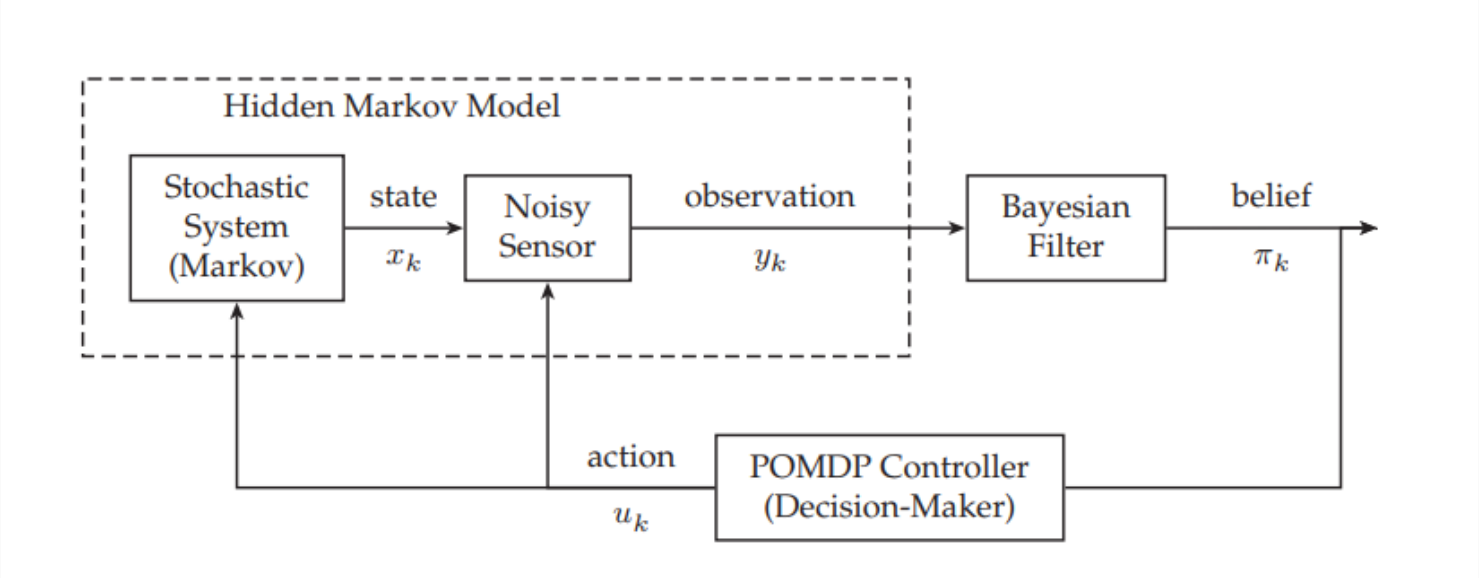


Figure 1: Relationship between different kinds of Markov Processes.

Presentation Overview



I will separate this presentation into three main parts

1. HMM filter
2. MDP
3. POMDP

Notation

- $\mathcal{X}$ will denote an state space and x_k the state of a Markov chain or process at time k.
- $\mathcal{Y}$ will denote an observation space and y_k the observation of a Markov chain or process at time k.
- $\mathcal{U}$ will denote an action space and u_k the chosen action at time k.
- N will denote the number of observations.

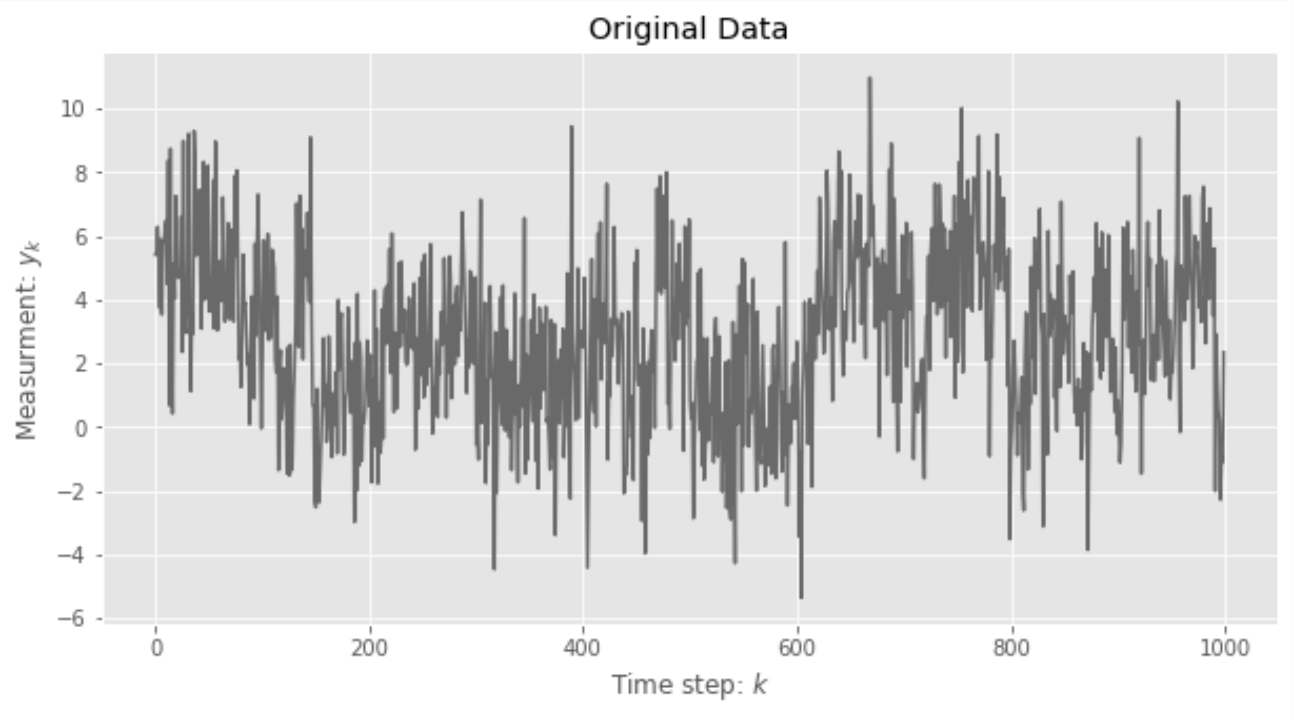
HMM Filters

HMM Definition

A HMM is a Markov model with a finite state space where we don't observe the states directly. We define

- States $x \in \mathcal{X}$
- Observations $y \in \mathcal{Y}$
- Transition probabilities $P_{ij} = p(x_{k+1} = j \mid x_k = i)$, with $i, j \in \mathcal{X}$.
- Observation probabilities $B_{xy} = p(y_k = y \mid x_k = x)$
- State level $C(x)$.

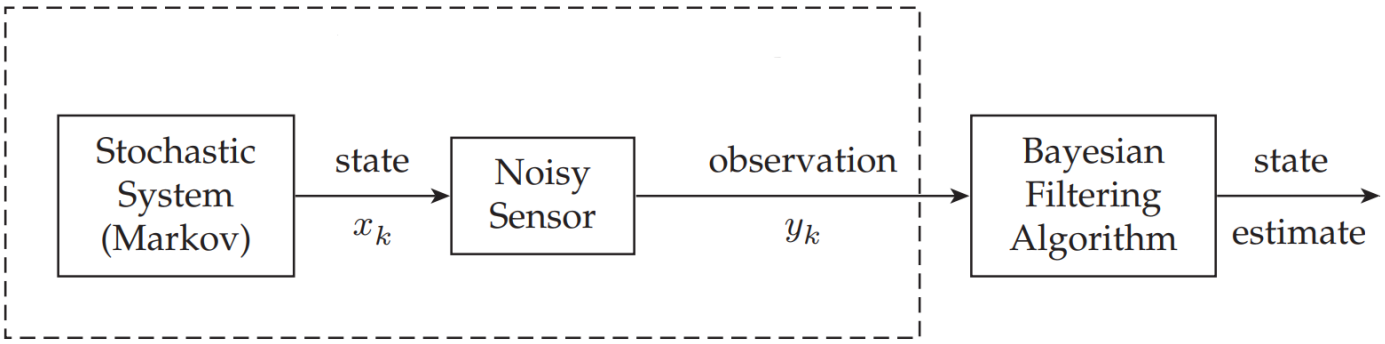
Example Data



This is a 3 state HMM, with low transition probability.

$$P = \begin{bmatrix} 0.95 & 0.025 & 0.025 \\ 0.025 & 0.95 & 0.025 \\ 0.025 & 0.025 & 0.95 \end{bmatrix}, \quad B_{xy} \sim \mathcal{N}(C(x), 2), \quad C = [1, 3, 5].$$

Optimal State Estimation



Our aim is to estimate x_k at time k given the observations up until time N, $y_{1:N}$.

Let $\boldsymbol{\kappa}$ denote a the set of all estimators that map a sequence of observations to a state. We then define the optimal estimator of a state x_k as the estimator that minimises the *mean square error*

$$\kappa^* = \underset{\kappa \in \boldsymbol{\kappa}}{\operatorname{argmin}} \, \mathbb{E}\left[(x_k - \kappa(y_{1:N}))^T (x_k - \kappa(y_{1:N}))\right]$$

Bayesian Filtering

The optimal state estimate $\hat{x}_k$ at time k given N observations is given by a conditional expectation

$$\hat{x}_k \equiv \kappa^*(y_{1:N}) = \mathbb{E}\left[x_k | y_{1:N}\right] = \int_{\mathcal{X}} x_k p(x_k | y_{1:N}) \; dx_k$$

and since evaluating this expectation involves computing the posterior with Bayes formula, optimal state estimation is also called *Bayesian state estimation*.

Prediction, filtering and Smoothing

- *Optimal Filtering:* $k = N$,
  estimating the state at current time k given measurements up until that time.
- *Optimal Prediction:* $k > N$,
  estimating the state at future time k, given N observations.
- *Optimal Smoothing:* $k < N$,
  estimating the state given, the past, present and future measurements.

Optimal Filtering

For easier notation we define the posterior of state at time k as

$$p(x_k = x \,|y_{1:k}) = \pi_k(x), \quad x \in \mathcal{X}.$$

The optimal filtering problem can then be stated as: Device an algorithm that recursivly can compute the posterior $\pi_k(x)$ given y_k and $\pi_{k-1}(x)$.

The resulting algorithm can be divided into two steps: prediction (1) and measurement update (2).

$$\pi_{k+1|k}(x_{k+1}) \equiv p(x_{k+1}|y_{1:k}) = \int_{\mathcal{X}} p(x_{k+1}|x_k)\pi_k(x_k)\,dx_k \tag{1}$$

$$\pi_{k+1}(x+1) = \frac{p(y_{k+1}|x_{k+1})\pi_{k+1|k}(x_{k+1})}{\int_{\mathcal{X}} p(y_{k+1}|x_{k+1})\pi_{k+1|k}(x_{k+1})\,dx_k} \tag{2}$$

(1) is know as the Chapman-Kolmogorov equation.

HMM filter

With a finite state space HMM, $\mathcal{X} = \{1, .., X\}$, we can reduce the integral to a sum over the finite state space

$$\pi_{k+1}(j) = \frac{p\left(y_{k+1} \mid x_{k+1} = j\right) \sum_{i=1}^{X} P_{ij} \pi_k(i)}{\sum_{l=1}^{X} p\left(y_{k+1} \mid x_{k+1} = l\right) \sum_{i=1}^{X} P_{il} \pi_k(i)} \quad j = 1, \ldots, X.$$

We can then reformulate this in matrix notation

$$B_{y_k} = diag(p(y_k \mid x = 1), ..., p(y_k \mid x = X)), \quad \pi_k = [\pi_k(1), ..., \pi_k(X)]$$

HMM filter

With a finite state space HMM, $\mathcal{X} = \{1, .., X\}$, we can reduce the integral to a sum over the finite state space

$$\pi_{k+1}(j) = \frac{p\left(y_{k+1} \mid x_{k+1} = j\right) \sum_{i=1}^{X} P_{ij} \pi_k(i)}{\sum_{l=1}^{X} p\left(y_{k+1} \mid x_{k+1} = l\right) \sum_{i=1}^{X} P_{il} \pi_k(i)} \quad j = 1, \ldots, X.$$

We can then reformulate this in matrix notation

$$B_{y_k} = diag(p(y_k \mid x = 1), ..., p(y_k \mid x = X)), \quad \pi_k = [\pi_k(1), ..., \pi_k(X)]$$

Filter Algorithm:

Given known parameters P, B, C and prior π_0.

For time $k = 0, 1, ...$, given observation y_{k+1} we can update our posterior according to

$$\pi_{k+1} = \frac{B_{y_{k+1}} P' \pi_k}{\sigma\left(\pi_k, y_{k+1}\right)}, \quad \sigma\left(\pi_k, y_{k+1}\right) = \mathbf{1}' B_{y_{k+1}} P' \pi_k$$

Optimal Smoothing for fixed interval

Smoothing aims to compute a state estimation at time k, given N observations, where $N > k$. There are different methods of choosing N but I will focus on *Fixed-interval smoothing*, but the principals are similar.

We can then formulate our posterior distribution for the state at time k as a product of two functions, which we then normalise,

$$\pi_{k|N}(x) = p\left(x_k = x \mid y_{1:N}\right) = \frac{\pi_k(x)\beta_{k|N}(x)}{\int_{\mathcal{X}} \pi_k(x)\beta_{k|N}(x)\,dx}.$$

Here we recognise $\pi_k(x)$ as the filter density presented earlier, dependent on the history. The new function $\beta_{k|N}(x)$ on the other hand is dependent on the future

$$\beta_{k|N}(x) = p\left(y_{k+1:N} \mid x_k = x\right)$$

$$\beta_{k|N}(x) = \int_{\mathcal{X}} p(x_{k+1} = z \mid x_k = x) p(y_{k+1} \mid x_{k+1} = z)\beta_{k+1|N}(z)\,dz, \quad k \in \{N-1, 0\}.$$

HMM Smoothing with Forward Backward

We can now again reformulate the problem in a matrix notation

$$\beta_{k|N}(x) = p\left(y_{k+1:N} \mid x_k = x\right), \quad \beta_{k|N} = \left[\beta_{k|N}(1), ..., \beta_{k|N}(X)\right]$$

Smoothing Algorithm

Compute $\beta_{k|N}$ via backward recursion:

$$\beta_{k|N} = PB_{y_{k+1}}\beta_{k+1|N}, \quad \text{for } k = N-1, ..., 1$$

initialised at $\beta_{N|N} = \mathbf{1}$.

Combining this backward recursion, with our filter computation of π_k we can compute the HMM smoother

$$\pi_{k|N}(x) = p\left(x_k = x \mid y_{1:N}\right) = \frac{\pi_k(x)\beta_{k|N}(x)}{\sum_{\mathcal{X}} \pi_k(x)\beta_{k|N}(x)}.$$

Filter Applied to Data

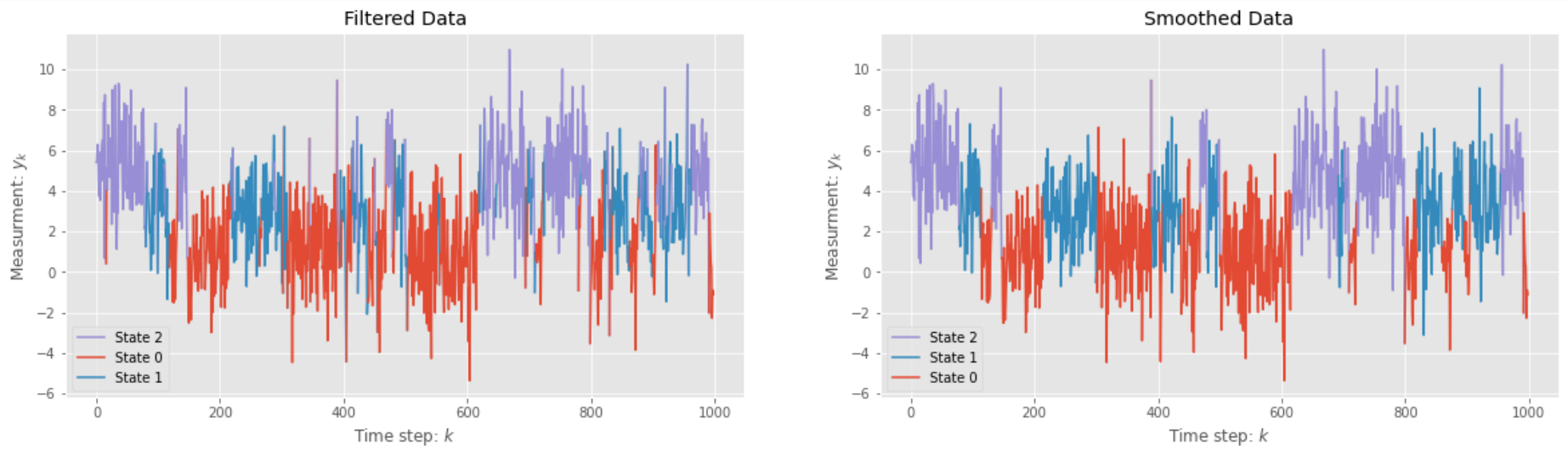


Figure 2: Filtering and Smoothing algorithm applied to the example data

The smoothing algorithm here shows its advantage, with the right state predicted 89% of the time, while the filter only predicts the right state 83% of the time.

However, that is to be expected since one of the methods is ”online” using information up until time k, and the other ”offline” using the entire data sequence.

Optimal Prediction

The optimal predicted state estimated at time $k+\Delta$ given observations up to time k obtained by Chapman-Kolmogorov's equation

$$\pi_{k+\Delta|k} = p(x_{k+\Delta} \mid y_{1:k}) = \int_{\mathcal{X}} \pi_k(x_k) \left(\prod_{t=1}^{\Delta} \int_{\mathcal{X}} p(x_{k+t} \mid x_{k+t-1}) \, dx_{k+t-1} \right)$$

Optimal Prediction

The optimal predicted state estimated at time $k+\Delta$ given observations up to time k obtained by Chapman-Kolmogorov's equation

$$\pi_{k+\Delta|k} = p(x_{k+\Delta} \mid y_{1:k}) = \int_{\mathcal{X}} \pi_k(x_k) \left(\prod_{t=1}^{\Delta} \int_{\mathcal{X}} p(x_{k+t} \mid x_{k+t-1})\, dx_{k+t-1} \right)$$

Prediction Algorithm

Which is very easily implemented with our filter as a propagation of the current belief state

$$\pi_{k+\Delta|k} = P^{\Delta} \pi_k$$

Viterbi Algorithm

Instead of the optimal filter with regard to the mean squared error, we can use other metrics. If we use the maximum a posterior (MAP) state estimator, with regard to the joint probability of the state and observation sequence

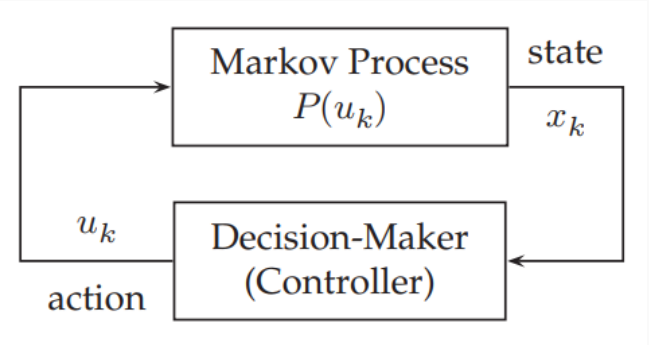
$$x^*_{0:N} = \underset{x_{0:N}}{\operatorname{argmax}}\, p(x_{0:N}, y_{1:N}).$$

This computation can be performed by forward dynamic programming and when performed for a HMM is called the *Viterbi* algorithm.

The big difference from the other presented algorithms is that this algorithm provides ONE estimated state at each time step, rather than a distribution.

MDP

Markov Decision Process



A MDP is a Markov Process with added control. A decision-maker uses the state at time k, x_k to choose action u_k.

To describe a MDP we need:

1. A states and state space $x \in \mathcal{X}$
2. Actions and action space $u \in \mathcal{U}$.
3. Transition probability from state i to j: $P_{ij}(u, k), \ \ i, j \in \mathcal{X}$.
   Where k denotes the time.
4. A cost function $c(x, u, k)$ and terminal cost $c(N)$ (if finite).

The decision makers objective is then to chose a sequence of actions that minimises the expected cumulative cost function.

Objective Function

To chose our actions we need a *policy* μ_k which maps out information available at time k, $\mathcal{I}_k$, to the action space: $u_k = \mu_k(\mathcal{I}_k)$. We can also denote the sequence of policies $\boldsymbol{\mu} = (\mu_0, ..., \ \mu_{N-1})$.

For a finite horizon, time up to N we get the objective function

$$J_{\boldsymbol{\mu}}(x) = \mathbb{E}_{\boldsymbol{\mu}} \left\{ \sum_{k=0}^{N-1} c\left(x_k, \mu_k\left(\mathcal{I}_k\right), k\right) + c_N\left(x_N\right) \mid x_0 = x \right\}$$

where we can define our optimal policy as

$$\boldsymbol{\mu}^* = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \ J_{\boldsymbol{\mu}}(x).$$

$\boldsymbol{\mu}^*$ is the policy that finds the lowest expected cumulative cost given every initial state x.

Polices

We can divide the different kind of policies possible into three categories (each a subcategory of the other):

1. *General policies*, $u_k \sim \mu_k(\mathcal{I}_k)$.
   The action is chosen from a probability distribution dependent on the current information.
2. *Randomized Markovian Policies*, $u_k \sim \mu_k(x_k)$.
   The action is chosen from a distribution dependent on the current state.
3. *Deterministic Markovian policies* (indexpolicies), $u_k = \mu_k(x_k)$.
   Deterministic policies mapping the state space to the action space.

It can be proven (left out here) that it's sufficient to find a deterministic Markovian policy to find $\boldsymbol{\mu}^*$.

Bellman's stochastic dynamic programming algorithm

The optimal policy $\boldsymbol{\mu}^*$ can be obtained by backward recursion:

Initialize $J_N(x) = c(x, N)$, then for $k = N-1, ..., 0$ do:

$$J_k(x) = \min_{u \in \mathcal{U}} \left\{ c(x, u, k) + \sum_j P_{xj}(u, k) J_{k+1}(j) \right\}$$

$$\mu_k^*(x) = \operatorname*{argmin}_{u \in \mathcal{U}} \left\{ c(x, u, k) + \sum_j P_{xj}(u, k) J_{k+1}(j) \right\}.$$

For any initial state $x \in \mathcal{X}$ the cumulative cost of the optimal policy is obtained as $J_0(x)$ from the equations.

Infinite horizon discount cost

Up until now we have expressed the cost as a function of time, as well as the state and action. However, it is often useful to consider the infinite horizon case, $N = \infty$.

To put bigger weight on earlier decisions and stop the objective function from exploding we add a discount factor $\rho \in [0, 1)$. Which results in the objective function

$$J_{\boldsymbol{\mu}}(x) = \mathbb{E}_{\boldsymbol{\mu}} \left\{ \sum_{k=0}^{\infty} \rho^k c\left(x_k, u_k\right) \mid x_0 = x \right\}$$

Since our horizon now is infinite we can't use the same recursion, but need to reformulate Bellman's dynamic programming equation.

Bellman's equation

Our solution for the discounted MDP with infinite horizon will be independent of time.

1. For any initial state i, the optimal J_{μ^*} is obtained by the value function $V(i)$ which satisfies (3).
2. For any initial state i, the optimal J_{μ^*} is a result of the optimal policy $\boldsymbol{\mu}^*$ which satisfies (3).
3. The value function V is the unique solution to (3).

$$
\begin{aligned}
V(i) &= \min_{u \in \mathcal{U}} Q(i,u), \quad \mu^*(i) = \operatorname*{argmin}_{u \in \mathcal{U}} Q(i,u) \\
Q(i,u) &= c(i,u) + \rho \sum_j P_{ij}(u)\, V(j)
\end{aligned}
\tag{3}
$$

Here the resulting Q-function is considered greedy since it is only taking into account the current state rather than future time points as earlier.

Methods for solving Bellman's equation for infinite horizon

Value iteration algorithms

These are approximate algorithms to compute the value function V. The idea is to solve it similarly as with dynamic programming, and iterate until time N, but then use μ_N^* at each time during implementation.

Methods for solving Bellman's equation for infinite horizon

Value iteration algorithms

These are approximate algorithms to compute the value function V. The idea is to solve it similarly as with dynamic programming, and iterate until time N, but then use μ_N^* at each time during implementation.

Policy iteration algorithms

We iteratively compute and update a stationary policy function until it converges.

Continuous State MDP

A MDP can also be formulated with a continuous state and observation space. We will reformulate our state transitions to

$$x_{k+1} = \phi_k(x_k, u_k, w_k),$$

where $\{w_k\}$ is a i.i.d random sequence with probability density p_w independent of the initial state x_0.

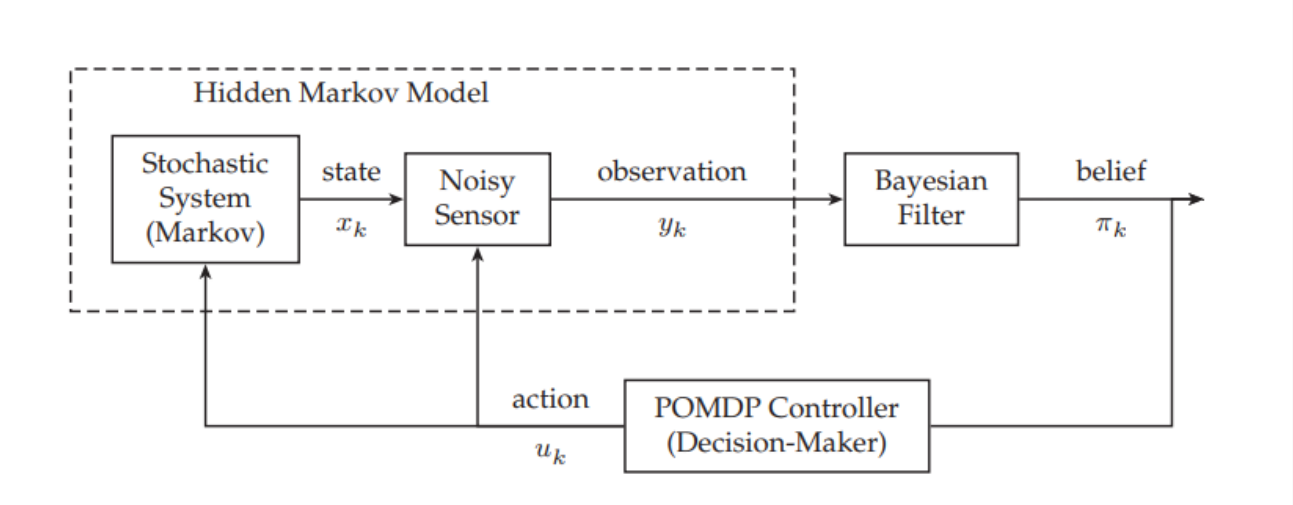
With this reformulation, while keeping the same for the action space and cost functions we obtain a similar Bellman equation

$$J_k(x) = \min_{u \in \mathcal{U}} \left\{ c(x, u, k) + \int J_{k+1} \left(\phi_k(x, u, w) \right) p_w(w) \, dw \right\}$$

$$\mu_k^*(x) = \operatorname*{argmin}_{u \in \mathcal{U}} \left\{ c(x, u, k) + \int J_{k+1} \left(\phi_k(x, u, w) \right) p_w(w) \, dw \right\}.$$

POMDP

POMDP Overview



We have now arrived at the goal, and we are going to be putting the pieces together.

We will now combine a HMM and a MDP to obtain a Partially Observable Markov Decision Process (POMDP).

POMDP

To describe a POMDP we use building blocks already mentioned in the HMM and MDP-models.

- States $x \in \mathcal{X}$.
- Observations $y \in \mathcal{Y}$.
- Actions $u \in \mathcal{U}$.
- Transition probability from state i to j: $P_{ij}(u),\ i, j \in \mathcal{X}$, dependent on the chosen action u.
- Observation probabilities $B_{xy}(u) = p(y_k = y \mid x_k = x, u_k = u)$ dependent on the chosen action.
- A cost function $c(x_k, u_k)$ and terminal cost $c_N(x_N)$ (if finite).

Objective functions for POMDP:s

Similarly as with the MDP, we can formulate our objective function both for a finite and infinite time horizon POMDP.

$$J_\mu(\pi_0) = \mathbb{E}_{\boldsymbol{\mu}}\left\{\sum_{k=0}^{N-1} c(x_k, u_k) + c_N(x_N) \mid \pi_0\right\},$$

$$J_\mu(\pi_0) = \mathbb{E}_{\mu}\left\{\sum_{k=0}^{\infty} \rho^k c(x_k, u_k)\right\}, \qquad \text{where } u_k = \mu(\pi_k)$$

The problem that arises now however, as stated before, is that we don't actually observe x_k but instead a noisy observation.

Belief State

As we stated for a MDP, the optimal action could be obtained with a deterministic policy $u_k = \mu_k^*(x_k)$. For a POMDP this is not possible, instead the optimal state action chosen by the decision maker needs to take all information into account,

$$u_k = \mu_k(\mathcal{I}_k), \quad \mathcal{I}_k = (\pi_0, u_0, y_1, ..., u_{k-1}, y_{k-1}).$$

Because of this the dimension grows with each increment of k. To combat this we need to compute a statistic which do not grow with time, a *belief state*.

The posterior distribution, $\pi_k = T(\pi_{k-1}, y_k, u_{k-1})$ computed with a filter T (in our case a HMM-filter), is such a statistic for $\mathcal{I}_k$. Instead of using all information $\mathcal{I}_k$ when computing the policy, we then use the belief state

$$u_k = \mu_k^*(\pi_k), \quad \pi_k \in \Pi$$

Bellman's equation for POMDP:s

For a finite horizon we can formulate Bellman's equation similarly as before and we achieve the optimal policy $\boldsymbol{\mu}^*$ by solving it.

Algorithm:
Initialise $J_N(\pi) = c_N\pi$ and then for $k = N{-}1, ..., 0$:

$$J_k(\pi) = \min_{u\in\mathcal{U}} \left\{ c_u\pi + \sum_{y\in\mathcal{Y}} J_{k+1}(\,T(\pi, y, u))\sigma(\pi, y, u) \right\}$$

$$\mu_k^*(\pi) = \operatorname*{argmin}_{u\in\mathcal{U}} \left\{ c_u\pi + \sum_{y\in\mathcal{Y}} J_{k+1}(\,T(\pi, y, u))\sigma(\pi, y, u) \right\}$$

Where T is the filter, $c_u = [c(1, u), ..., c(X, u)]$ and σ a normalising term from the filter equation.

But, since our belief space is uncountable, we don't have any methods of solving it efficiently.

Finite dimensional controller for finite horizon

Given a POMDP with finite observation and action space, at every time point the value function and optimal policy have the following characteristic:

1. The value function $J_k(\pi)$ is piecewise linear and concave with regard to $\pi \in \Pi$.
2. The belief space Π can be separated into a finite amount of convex polytopes and within each polytope the optimal policy $\mu_k^*(\pi)$ is a constant corresponding to a single action.

Monahan's and Witness algorithms use these characteristics to achieve exact algorithms. The idea is to eliminate redundant parts of the value function, and only keep the necessary parts.

Illustration of piecewise characteristic

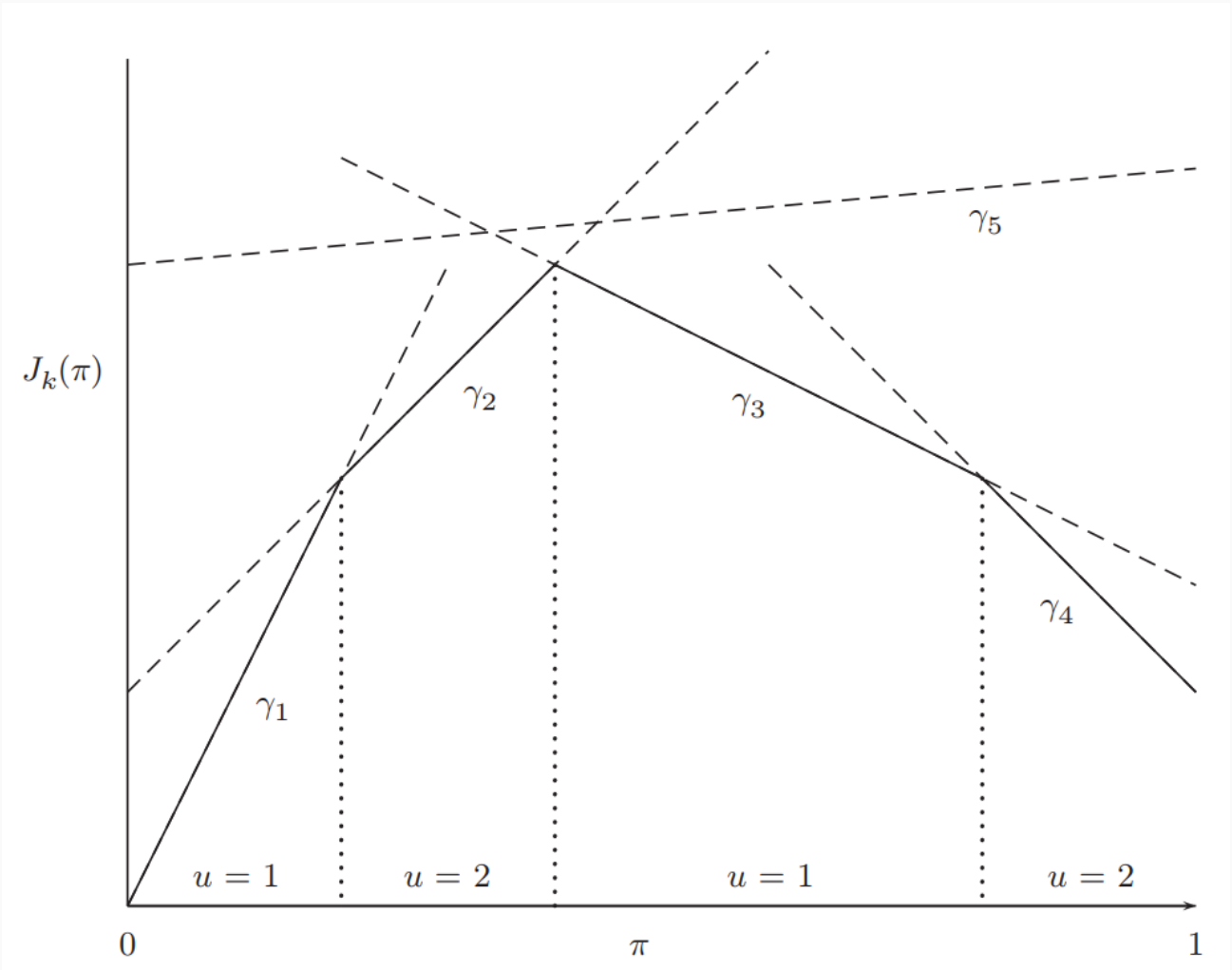


Figure 3: Illustration of a piecewise linear value function for a POMDP, for one if the states.

Approximate algorithms - Lovejoy

Lovejoy's suboptimal algorithm

Computing value functions via exact methods are intractable, but we can instead compute upper and lower bounds to it.

We can construct a upper bound by only considering a subset of all piecewise segments, and prune those with similar methods as the exact methods. The lower limit is then created through interpolation of the value function, using that the interpolation of a concave function lies bellow the concave function.

Approximate algorithms - Point-based

Point-based value iteration methods

Instead of trying to prune down the value function, another approach which has been rather successful is to instead approximate the value functions in chosen parts of the belief space.

The main idea behind these methods is to sample from the belief space and only solve the POMDP for these states. An example of this kind of algorithm is the SARSOP.

Bellman's equation for infinite horizon

Similarly as with the MDP, for a infinite horizon we use a similar equations but the current state is substituted by our filter T and its corresponding normalising term σ. The same statements as earlier are true:

1. The optimal expected cumulative cost is achieved by a stationary deterministic Markovian policy μ^*.
2. The optimal policy $\mu^*(\pi)$ and value function $V(\pi)$ satisfy Bellman's dynamic programming equation.
3. The value function $V(\pi)$ is continuous and concave in $\pi \in \Pi(X)$.

$$\mu^*(\pi) = \operatorname*{argmin}_{u \in \mathcal{U}} Q(\pi, u), \quad J_{\mu^*}(\pi_0) = V(\pi_0)$$

$$V(\pi) = \min_{u \in \mathcal{U}} Q(\pi, u), \quad Q(\pi, u) = c_u \pi + \rho \sum_{y \in Y} V(T(\pi, y, u))\sigma(\pi, y, u).$$

Reinforcement Learning

In reinforcement learning the different methods used can be categorised as

Q-learning: approximate the Q_θ-function, mapping belief state action pairs to values $\Pi \times \mathcal{U} \to \mathbb{R}$. Given this function you can create your policy, which results in a greedy approach.

Policy gradient methods: estimates the optimal policy function with a parametric function, μ_θ.

Summary

References

All the flow charts, equations and the majority of the information in this presentation can be found in:

Krishnamurthy, V. (2016). *Partially Observed Markov Decision Processes: From Filtering to Controlled Sensing*. Cambridge: Cambridge University Press.
DOI: 10.1017/CBO9781316471104.012.