Australian National University

COMP4680/8650 Advanced Topics in Statistical Machine Learning

Week 6: Dynamic Bayesian Networks

Stephen Gould

Temporal Models

In many situations we are interested in reasoning about the state of the world as it evolves over time. Examples:

- robot localization, patient health, speech recognition

Instead of a single set of random variables X we model the variables' trajectory over time $X_{0:T} = X_0, \ldots, X_T$.

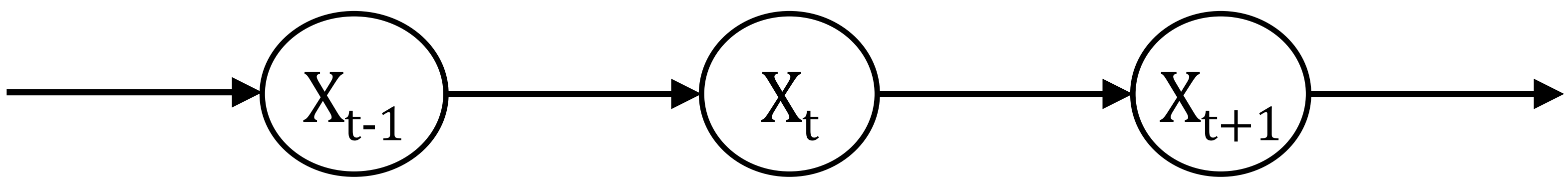
We will make two simplifications:

- time discretization
- Markov assumption

Markov Models

Markov Assumption: "the future is independent of the past given that we know the present"

$$p(X_{0:T}) = p_0(X_0) \prod_{t=1}^{T} p_t(X_t \mid X_{t-1})$$



Stationary Dynamical Systems

We say that a dynamical system is **stationary** (or time invariant) if $p_t(X_t \mid X_{t-1})$ is the same for all t.

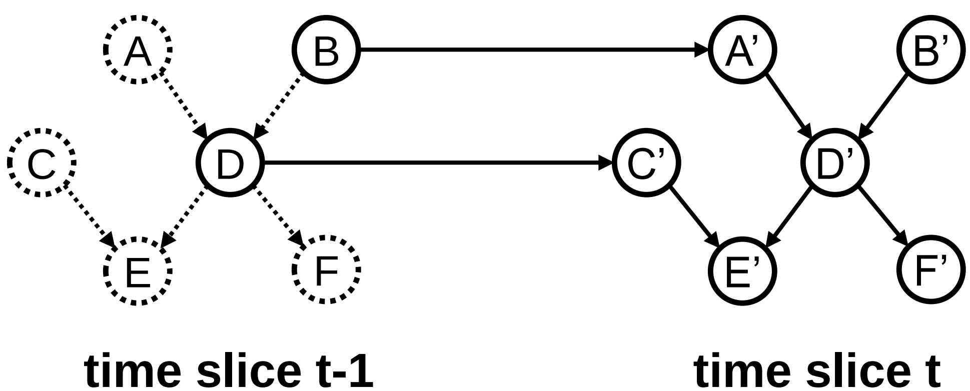
If this is the case then we can represent infinite trajectories very compactly. We only need to represent

- the initial state distribution $p_0(X_0)$
- the transition model $p(X' \mid X)$

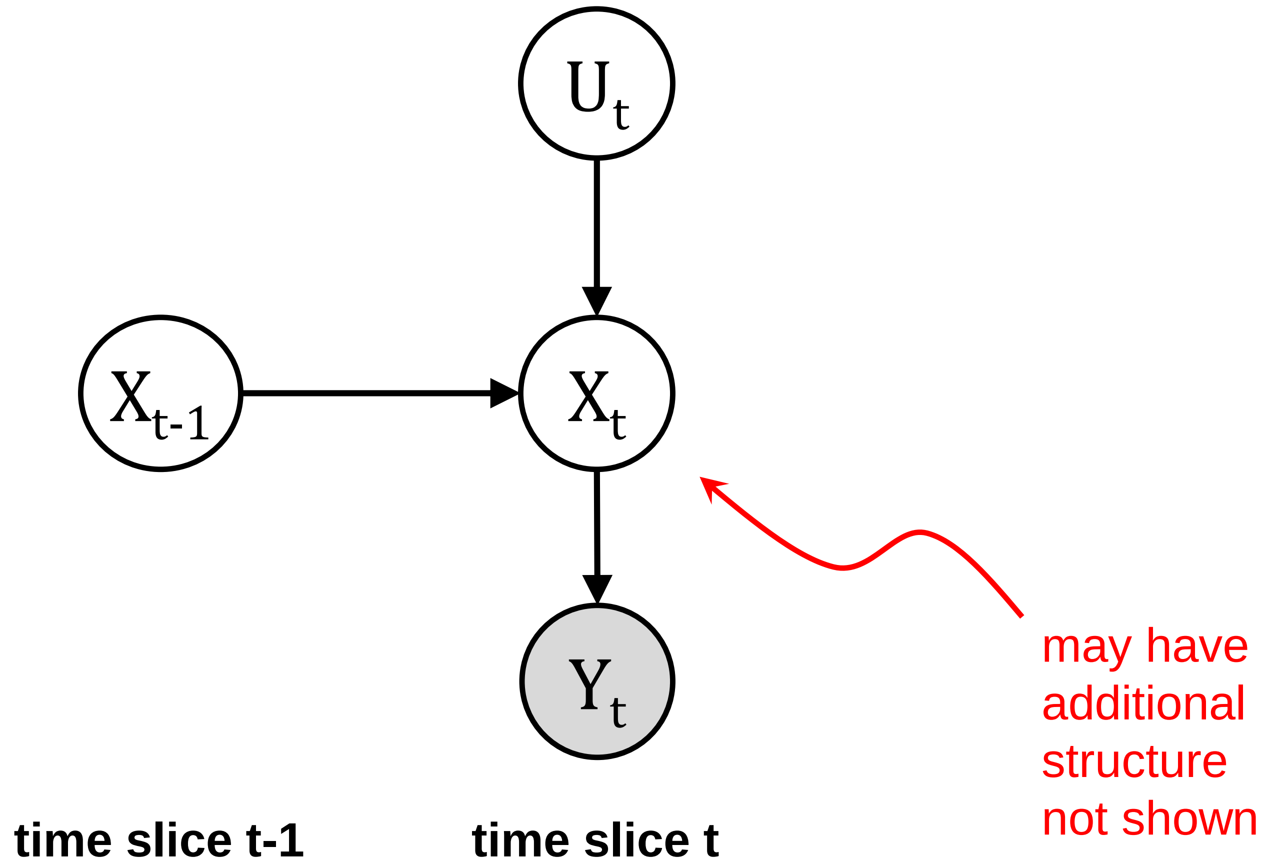
Dynamic Bayesian Networks

A **dynamic Bayesian network (DBN)** is a pair $(B_0, B_\rightarrow)$ where

- B_0 is a BN over the variables at time zero
- $B_\rightarrow$ is a 2-time-slice "template" defining
  - the dependence of variables from time t-1 to time t
  - the dependence of variables within time t



Typical DBN (Hidden Markov Model)



Inference in DBNs

- Consider a system evolving over time where:
  - x_t is the (unknown) system state at time t
  - u_t is the (known) control input at time t
  - y_t is some (observed) measurement at time t

- **filtering** estimates $p(x_{0:t} \mid y_{1:t}, u_{1:t})$

- **smoothing** estimates $p(x_{0:t} \mid y_{1:T}, u_{1:T})$
  - smoothing takes into account future observations

Efficient Recursive Filtering

Define the belief over the state at time t as

$$bel(x_t) = p(x_t \mid y_{1:t}, u_{1:t})$$

Bayesian updating proceeds in two steps:

- **prediction:** $bel'(x_t) = \int p(x_t \mid x_{t-1}, u_t) bel(x_{t-1}) dx_{t-1}$
- **correction:** $bel(x_t) \propto p(y_t \mid x_t) bel'(x_t)$

In order to implement the Bayes filter we often need to use tractable approximations. Considerations:

- computational efficiency, accuracy of approximation, ease of implementation

Linear Gaussian Systems

The linear Gaussian assumption for the **transition** (motion) and **measurement** models is a popular choice and leads to the famous Kalman filter.

The variables of the model are all assumed to be continuous-valued vectors:

- state (hidden), $x_t \in \mathbf{R}^n$
- control (input), $u_t \in \mathbf{R}^q$
- measurement (output), $y_t \in \mathbf{R}^p$

Linear Gaussian Assumptions

- The **state** is assumed to evolve as a linear function of the previous state and the current control input,
$$x_t = A_t x_{t-1} + B_t u_t + \eta_t$$
where ϵ_t is additive zero-mean Gaussian noise.

- The **measurements** are also a linear function of state,
$$y_t = C_t x_t + \delta_t$$
where δ_t is additive zero-mean Gaussian noise.

- Our belief over the **initial state**, x_0, must also be Gaussian distributed.

Linear Gaussian Probabilities

- Initial belief: $p(x_0) = \mathcal{N}(x_0; \mu_0, \Sigma_0)$
- State transition probability:
$$p(x_t \mid u_t, x_{t-1}) = \mathcal{N}(x_t; A_t x_{t-1} + B_t u_t, Q_t)$$
- Measurement probability:
$$p(y_t \mid x_t) = \mathcal{N}(y_t; C_t x_t, R_t)$$

Under these assumptions the posterior distribution over x_t will be Gaussian.

Kalman Filter Algorithm

Prediction Step

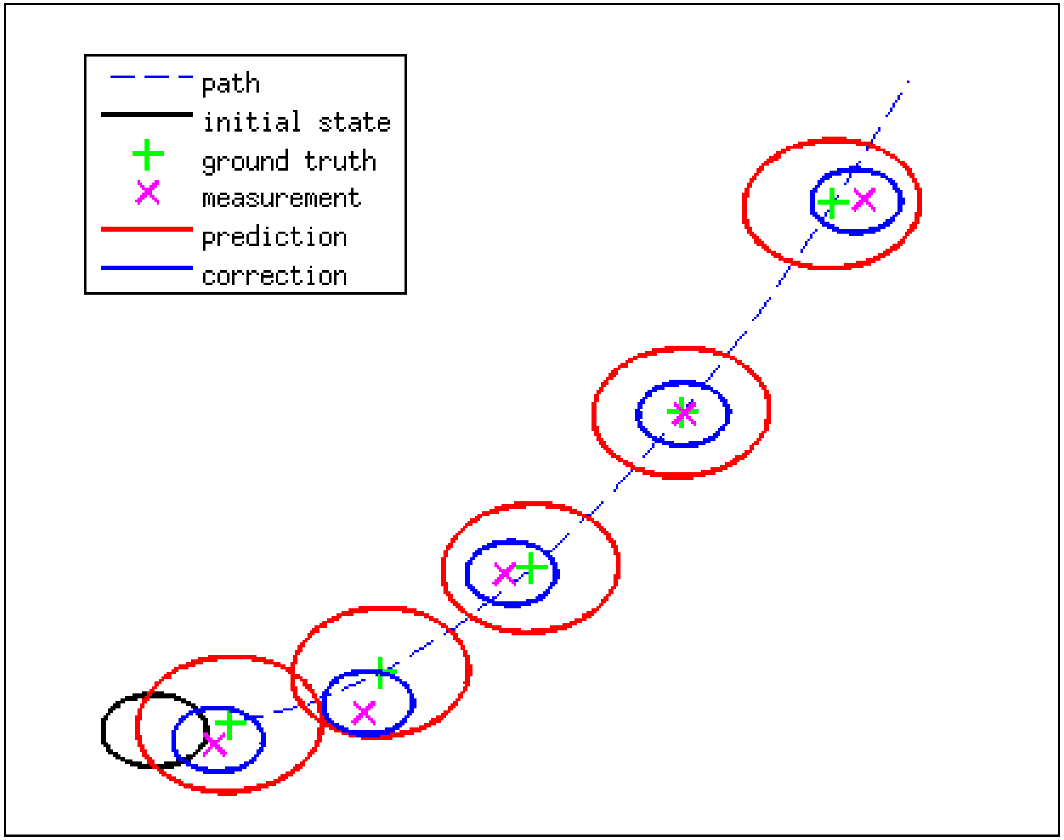
$$\mu_{t|t-1} = A_t \mu_{t-1|t-1} + B_t u_t$$
$$\Sigma_{t|t-1} = A_t \Sigma_{t-1|t-1} A_t^T + Q_t$$

Correction Step

$$K_t = \Sigma_{t|t-1} C_t^T \left(C_t \Sigma_{t|t-1} C_t^T + R_t \right)^{-1}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t \left(y_t - C_t \mu_{t|t-1} \right)$$
$$\Sigma_{t|t} = (I - K_t C_t) \Sigma_{t|t-1}$$

Computational efficiency: $O(p^3)$ for matrix inversion and $O(n^2)$ for covariance update. Measurement does not appear in Σ update.

Kalman Filter Example



Conditional Gaussians

Consider jointly Gaussian random variables x and y,

$$\begin{bmatrix} x \\ y \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix} \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{xy}^T & \Sigma_{yy} \end{bmatrix} \right)$$

Then $P(x \mid y)$ is Gaussian with

$$\mu_{x|y} = \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}\left(y - \mu_y\right)$$
$$\Sigma_{x|y} = \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{xy}^T$$

(See Murphy §4.3.4.3)

Kalman Filter Derivation

Prediction Step

$$
\begin{aligned}
\mu_{t|t-1} &= E[x_t] \\
&= E[Ax_{t-1} + Bu_t + \eta_t] \\
&= AE[x_{t-1}] + Bu_t + E[\eta_t] \\
&= A\mu_{t-1|t-1} + Bu_t
\end{aligned}
$$

$$
\begin{aligned}
\Sigma_{t|t-1} &= Var[x_t] \\
&= Var[Ax_{t-1} + Bu_t + \eta_t] \\
&= AVar[x_{t-1}]A^T + Var[\eta_t] \\
&= A\Sigma_{t-1|t-1}A^T + Q_t
\end{aligned}
$$

Kalman Filter Derivation

Correction Step. First write as joint Gaussian over state and measurements

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_{t|t-1} \\ C_t \mu_{t|t-1} \end{bmatrix} \begin{bmatrix} \Sigma_{t|t-1} & \Sigma_{t|t-1} C_t^T \\ \mathrm{C_t} \Sigma_{t|t-1}^T & \mathrm{C_t} \Sigma_{t|t-1} C_t^T + R_t \end{bmatrix} \right)$$

Now apply the conditional Gaussian rule,

$$\mu_{t|t} = \mu_{t|t-1} + \Sigma_{t|t-1} C_t^T \left(\mathrm{C_t} \Sigma_{t|t-1} C_t^T + R_t \right)^{-1} \left(y_t - C_t \mu_{t|t-1} \right)$$

$$\Sigma_{t|t} = \Sigma_{t|t-1} - \Sigma_{t|t-1} C_t^T \left(\mathrm{C_t} \Sigma_{t|t-1} C_t^T + R_t \right)^{-1} \mathrm{C_t} \Sigma_{t|t-1}^T$$

Extended Kalman Filter

The **extended Kalman filter** (EFK) allows the use of non-linear motion and measurement models,

$$x_t = g(x_{t-1}, u_t) + \eta_t$$
$$y_t = h(x_t) + \delta_t$$

These models are linearized around the current mean of the state posterior, $\mu_{t-1|t-1}$, before each filter update

$$g(x_{t-1}, u_t) \approx g\left(\mu_{t-1|t-1}, u_t\right) + G_t\left(x_{t-1} - \mu_{t-1|t-1}\right)$$
$$h(x_t) \approx h\left(\mu_{t|t-1}\right) + H_t\left(x_t - \mu_{t|t-1}\right)$$

where G_t and H_t are Jacobians.

The accuracy of the EKF depends heavily on the goodness of the linear approximation.

Extended Kalman Filter

The **extended Kalman filter** (EFK) allows the use of non-linear motion and measurement models,

$$x_t = g(x_{t-1}, u_t) + \eta_t$$
$$y_t = h(x_t) + \delta_t$$

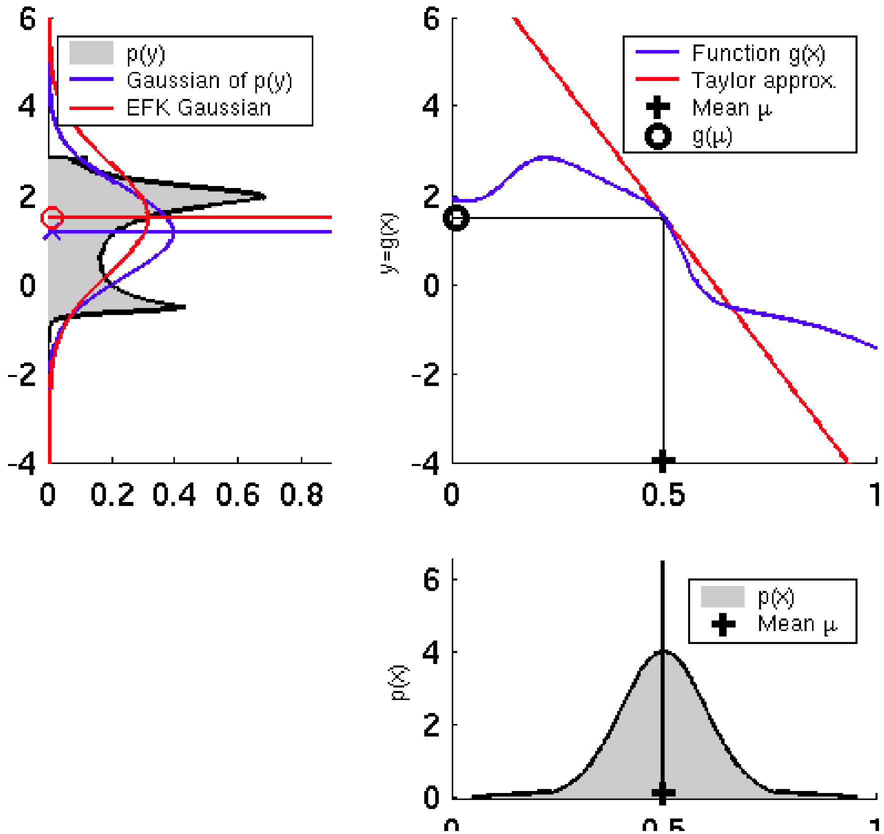
These models are linearized around the current mean of the state posterior, $\mu_{t-1|t-1}$, before each filter update

$$g(x_{t-1}, u_t) \approx g\left(\mu_{t-1|t-1}, u_t\right) + G_t\left(x_{t-1} - \mu_{t-1|t-1}\right)$$
$$h(x_t) \approx \underbrace{h\left(\mu_{t|t-1}\right)}_{\text{constant}} + \underbrace{H_t\left(x_t - \mu_{t|t-1}\right)}_{\text{linear fcn of } x_t}$$

where G_t and H_t are Jacobians.

The accuracy of the EKF depends heavily on the goodness of the linear approximation.

Extended Kalman Filter Illustration



Extended Kalman Filter Algorithm

The extended Kalman filter is then

$$\mu_{t|t-1} = g\left(\mu_{t-1|t-1}, u_t\right)$$
$$\Sigma_{t|t-1} = G_t \Sigma_{t-1|t-1} G_t^T + R_t$$

$$K_t = \Sigma_{t|t-1} H_t^T \left(H_t \Sigma_{t|t-1} H_t^T + Q_t\right)^{-1}$$
$$\mu_{t|t} = \mu_{t|t-1} + K_t\left(y_t - h(\mu_{t|t-1})\right)$$
$$\Sigma_{t|t} = (I - K_t H_t)\Sigma_{t|t-1}$$

Unscented Kalman Filter

Instead of using a taylor series expansion to linearize the motion and measurement models, the **unscented Kalman filter** (UKF) performs stochastic linearization, which preserves statistical properties of the belief state.

- Weighted samples are generated from $N(\mu_{t-1|t-1}, \Sigma_{t-1|t-1})$ and propagated though $g(x_{t-1}, u_t)$
- A Gaussian is then fitted to the transformed results

The UKF does not require calculation of analytical derivatives. It also works better in practice than the EKF.

Particle Filtering

- The particle filter is a nonparametric Bayes filter, also known as **Sequential Monte Carlo**.
- The posterior $p(x_t \mid x_{0:t-1}, y_{1:t})$ is represented by a set of random samples

$$\mathcal{X}_t = \left\{x_t^{(1)}, \ldots, x_t^{(m)}\right\}$$

- The representation is approximate (unlike the Kalman filter) but can model a much broader range of probability distributions (and non-linear dynamics).

Particle Filter Algorithm

Update and Weight Particles

for i = 1 to m do

    sample $x_t^{(i)}$ from $p(x_t \mid u_t, x_{t-1}^{(i)})$

    set $w_t^{(i)} = p(y_t \mid x_t^{(i)})$

Resample Particles

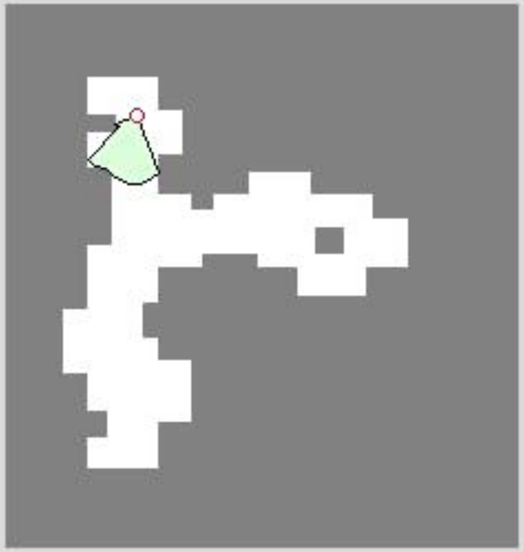
initialise $\mathcal{X}_t = \emptyset$

for j = 1 to m do

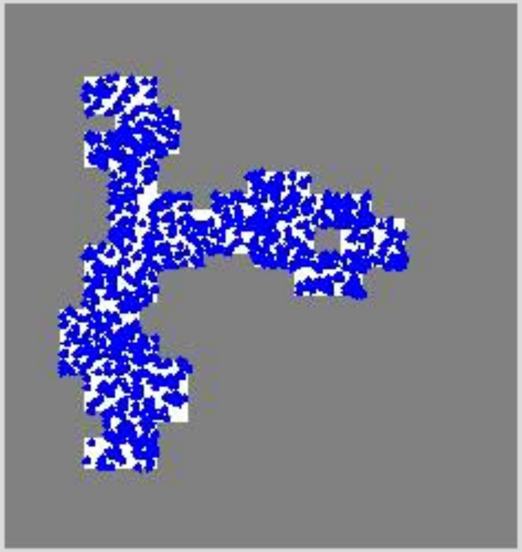
    draw i with probability proportional to $w_t^{(k)}$

    add $x_t^{(i)}$ to $\mathcal{X}_t$

Particle Filter Example



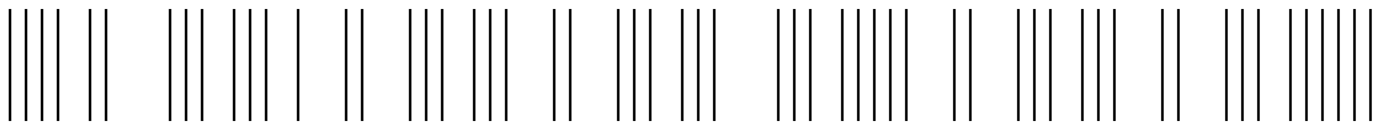
real robot location



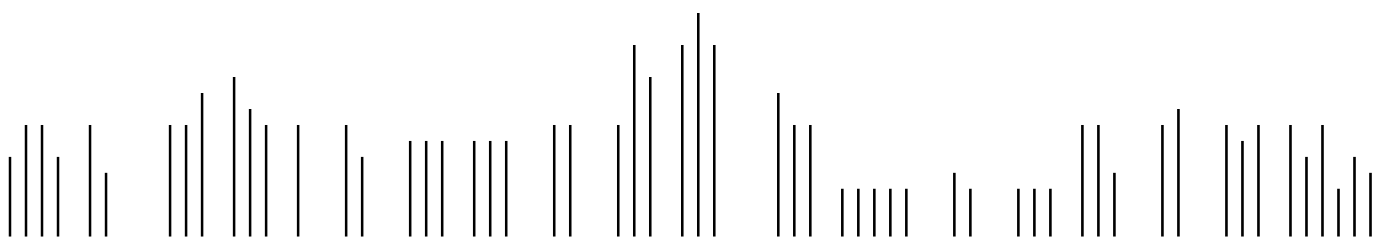
particle filter

Resampling Step

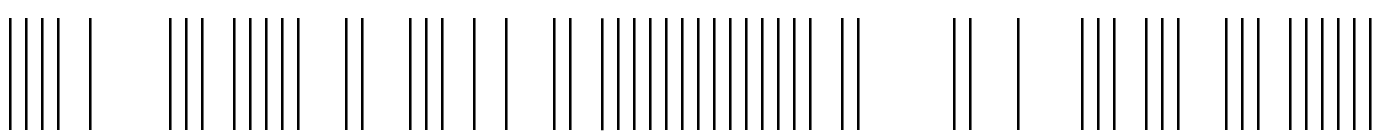
unweighted particles



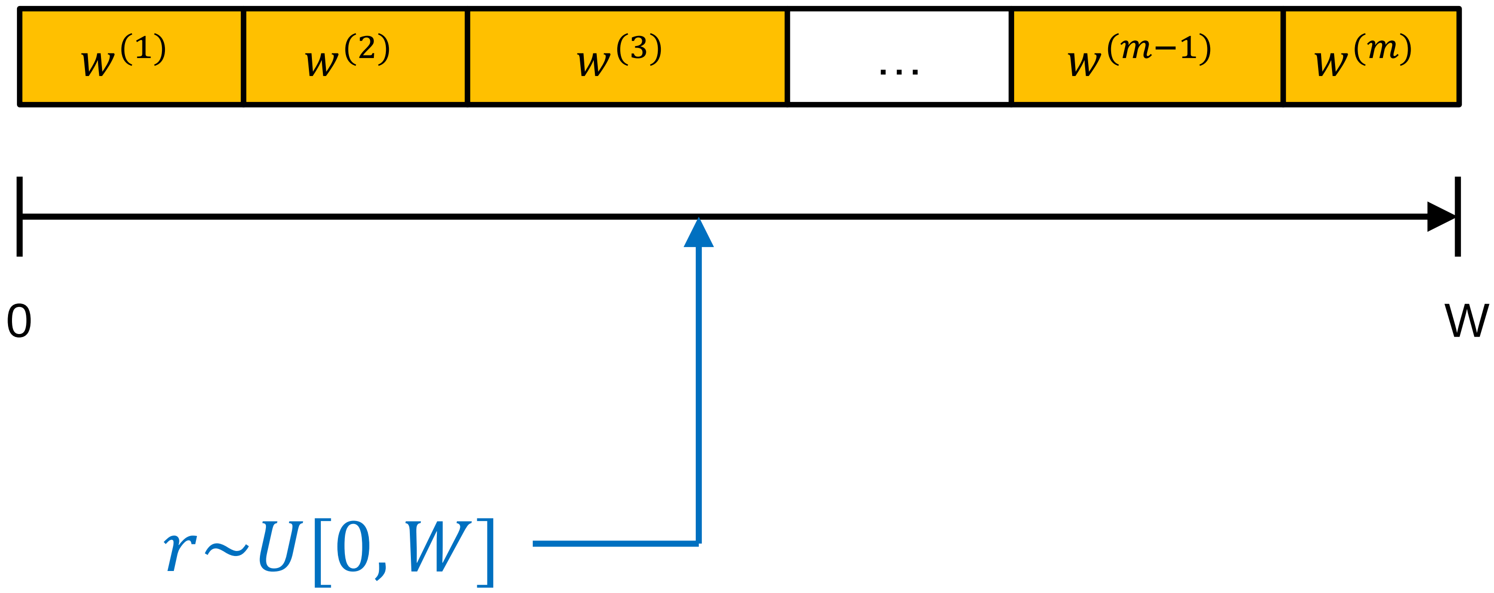
weighted particles



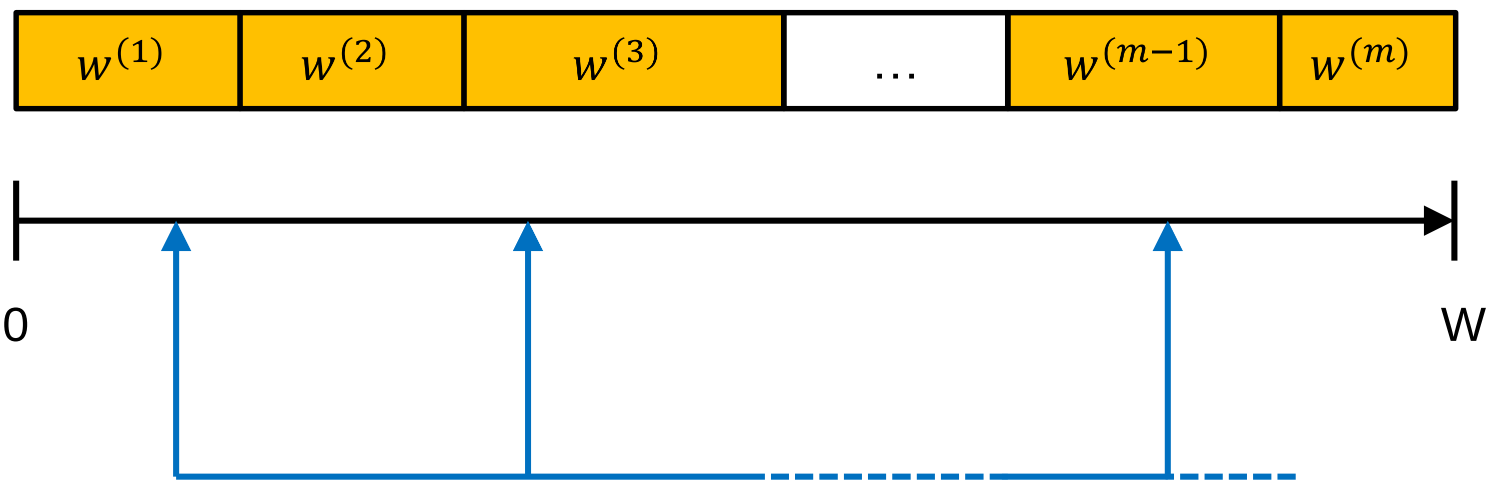
unweighted resampled particles



Sampling from [0, W]



Low Variance Sampler



$r \sim U[0, m^{-1}W]$ $\quad$ $r + m^{-1}W$ $\quad$ $r + km^{-1}W$

makes samples dependent

Low Variance Sampler Algorithm

initialise $\mathcal{X}_t = \emptyset$

set $W = \sum_{i=1}^{m} w_t^{(i)}$

sample r ~ U[0, m^{-1}W]

set i = 1 and c = $w_t^{(1)}$

for j = 1 to m do

    U = r + (j - 1)m^{-1}W

    while U > c do

        set i += 1 and c += $w_t^{(i)}$

    add x_t^i to $\mathcal{X}_t$

Particle Deprivation

- **Particle Deprivation**: no particles are near the true state

- This is a result of variance in random sampling
  - An unlucky series of random numbers can wipe out all particles near the true state
  - Since this has a non-zero probability of happening at each time slice it will happen eventually

- A popular solution is to add a small number of randomly generated particles when resampling
  - Also addresses the “kidnapped robot” problem

Distributional Particles

- Computational cost/number of particles explodes as dimensionality increases
- Distributional particles (aka **Rao-Blackwellization**) by keeping part of the distribution in analytical form
- Example: conditionally linear Gaussian models
  - Let the state be represented by $(x_t, z_t) \in \mathbf{R}^n \times [1, \ldots, K]$
  - Assume that conditioned on z_t the system can be modelled with linear Gaussian dynamics
$$x_t = A(z_{t-1})x_{t-1} + B(z_{t-1})u_t + \eta_t$$
$$y_t = C(z_t)x_t + \eta_t$$
  - Then we can approximate the distribution over (x_t, z_t) by a set of particles with $z_t \sim [0, \ldots, K]$ and $x_t \sim N(\mu_t, \Sigma_t)$.

RBPF Algorithm

Update and Weight Particles

for i = 1 to m do

sample $z_t^{(i)}$ from $p(z_t \mid u_t, z_{t-1}^{(i)})$

Kalman filter update for $\mu_t^{(i)}$ and $\Sigma_t^{(i)}$ given $z_t^{(i)}$

set $w_t^{(i)} = p(y_t \mid x_t^{(i)}, z_t^{(i)})$

Resample Particles

initialise $\mathcal{X}_t = \emptyset$

for j = 1 to m do

draw i with probability proportional to $w_t^{(k)}$

add $\left(x_t^{(i)}, z_t^{(i)}\right)$ to $\mathcal{X}_t$