Introduction to Probabilistic Graphical Models
Lecture 6

Hidden Markov Models

Télécom ParisTech,

Université Paris-Saclay, Paris, France

Instructor: Umut Şimşekli

Lecture Outline

- Sequential data, Terminology
- Hidden Markov Models
- Implementation of the Forward-Backward algorithm
- Finding the MAP trajectory: the Viterbi algorithm

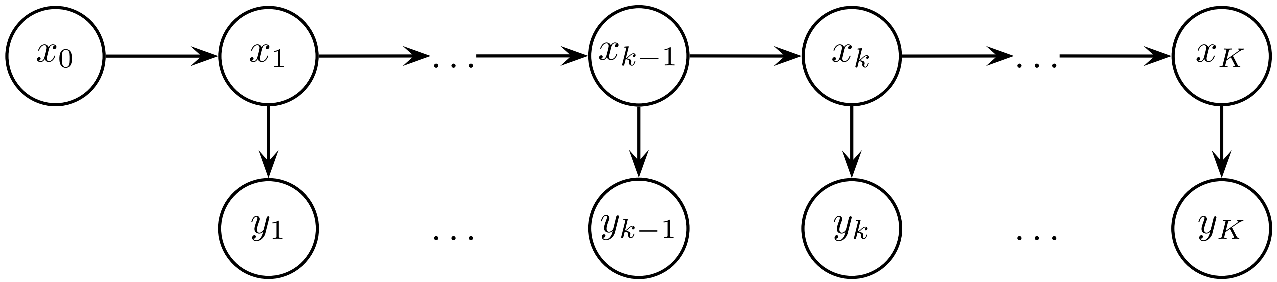
Disclaimer

- All the material that will be used within this course is adapted from the "Bayesian Statistics and Machine Learning" course that has been given by A. Taylan Cemgil at Boğaziçi University, Istanbul

- For more info, please see `http://www.cmpe.boun.edu.tr/~cemgil/`

Sequential Data: Models, Inference, Terminology

In signal processing, machine learning, robotics, statistics many phenomena are modelled by dynamical models



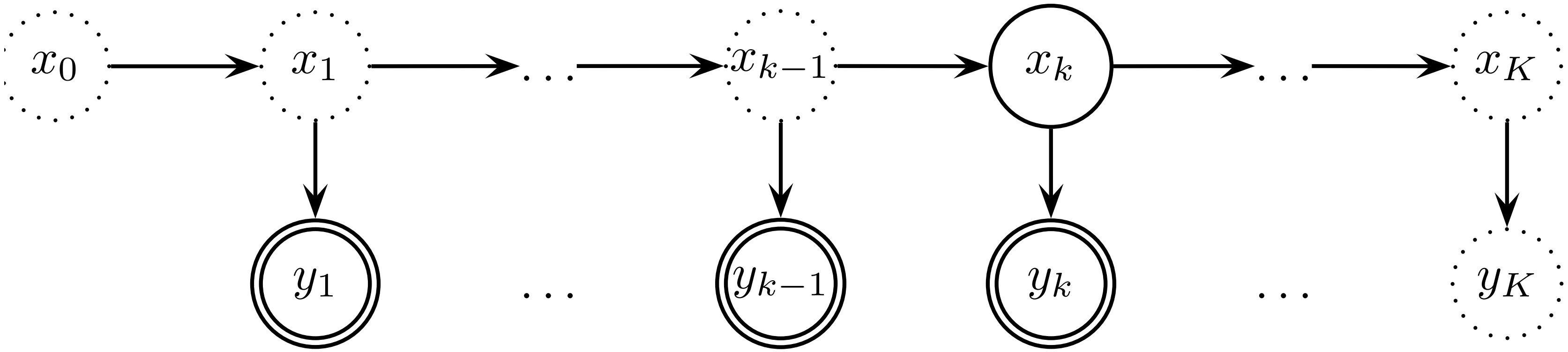
$$x_k \sim p(x_k|x_{k-1}) \quad \text{Transition Model}$$

$$y_k \sim p(y_k|x_k) \quad \text{Observation Model}$$

- x is the latent state (tempo, pitch, velocity, attitude, class label, ...)
- y are observations (samples, onsets, sensor reading, pixels, features, ...)
- In a full Bayesian setting, x includes unknown model parameters

Online Inference, Terminology

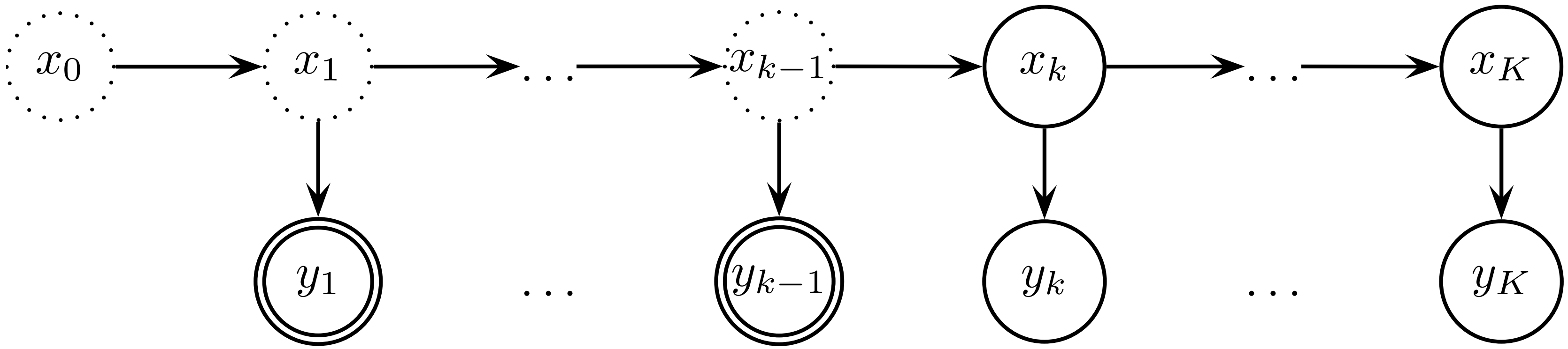
- **Filtering:** $p(x_k|y_{1:k})$
  - Distribution of current state given all past information
  - Realtime/Online/Sequential Processing



- Potentially confusing misnomer:
  - More general than “digital filtering” (convolution) in DSP – but algoritmically related for some models (KFM)

Online Inference, Terminology

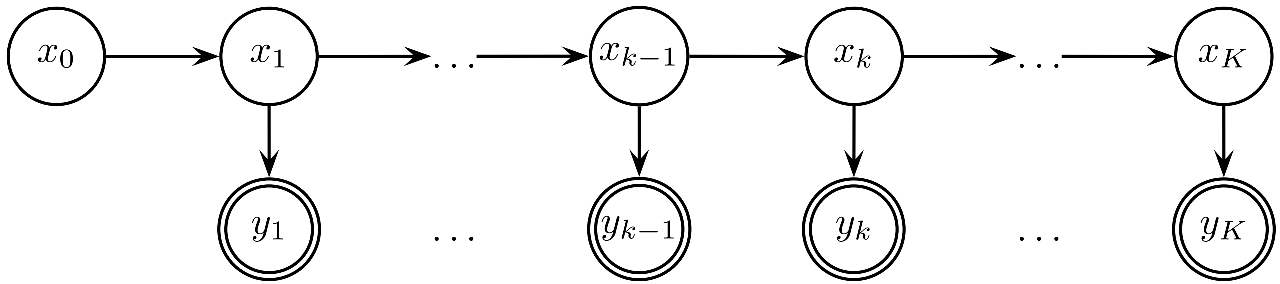
- **Prediction** $p(y_{k:K}, x_{k:K}|y_{1:k-1})$
  - evaluation of possible future outcomes; like filtering without observations



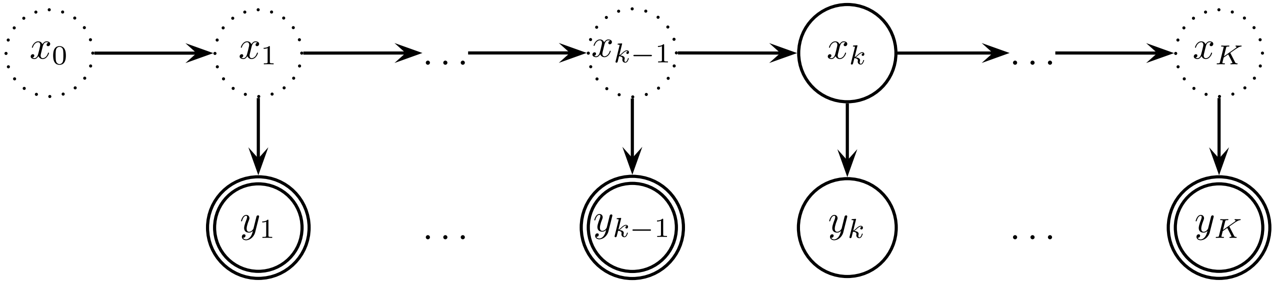
- Accompaniment, Tracking, Restoration

Offline Inference, Terminology

- **Smoothing** $p(x_{0:K}|y_{1:K})$,
  Most likely trajectory – Viterbi path $\arg\max_{x_{0:K}} p(x_{0:K}|y_{1:K})$
  better estimate of past states, essential for learning

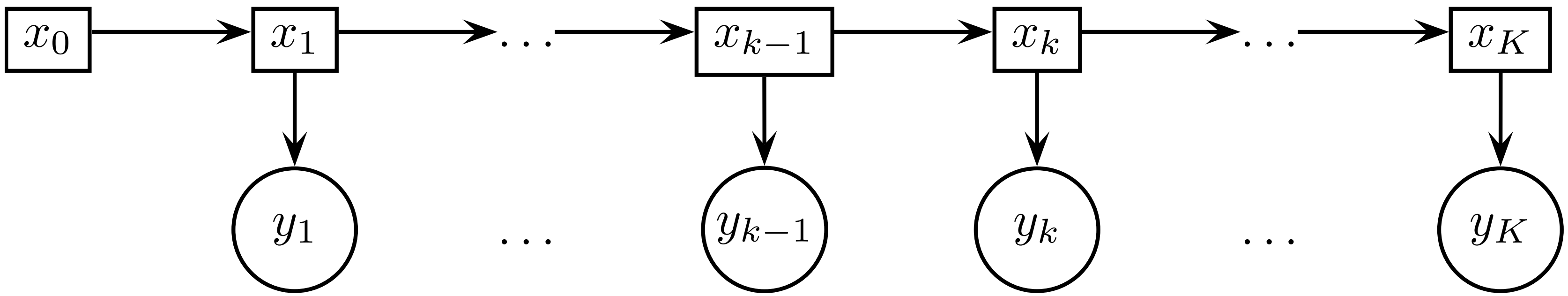


- **Interpolation** $p(y_k, x_k|y_{1:k-1}, y_{k+1:K})$
  fill in lost observations given past and future


x_0
x_1
$\dots$
x_{k-1}
x_k
$\dots$
x_K
y_1
$\dots$
y_{k-1}
y_k
$\dots$
y_K

Hidden Markov Model [?]

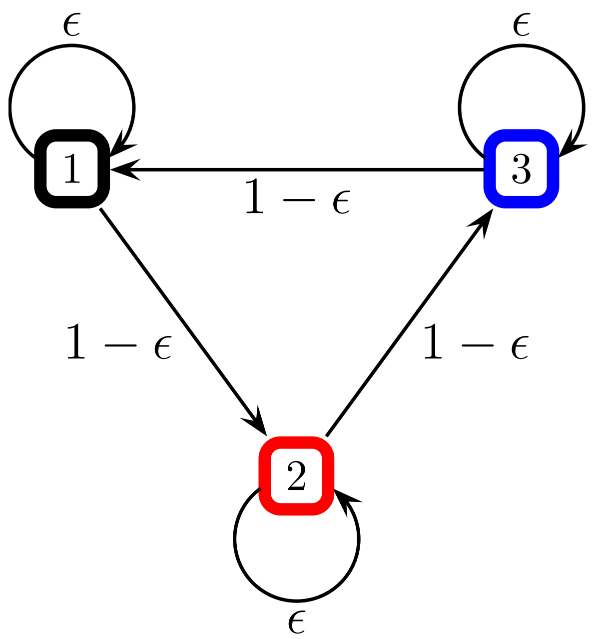
- Mixture model evolving in time



- Observations y_k are continuous or discrete
- Latent variables x_k are discrete
  - Represents the fading memory of the process
- Exact inference possible if x_k has a "small" number of states

Example: Hidden Markov Model

- State transition model (a N by N matrix)

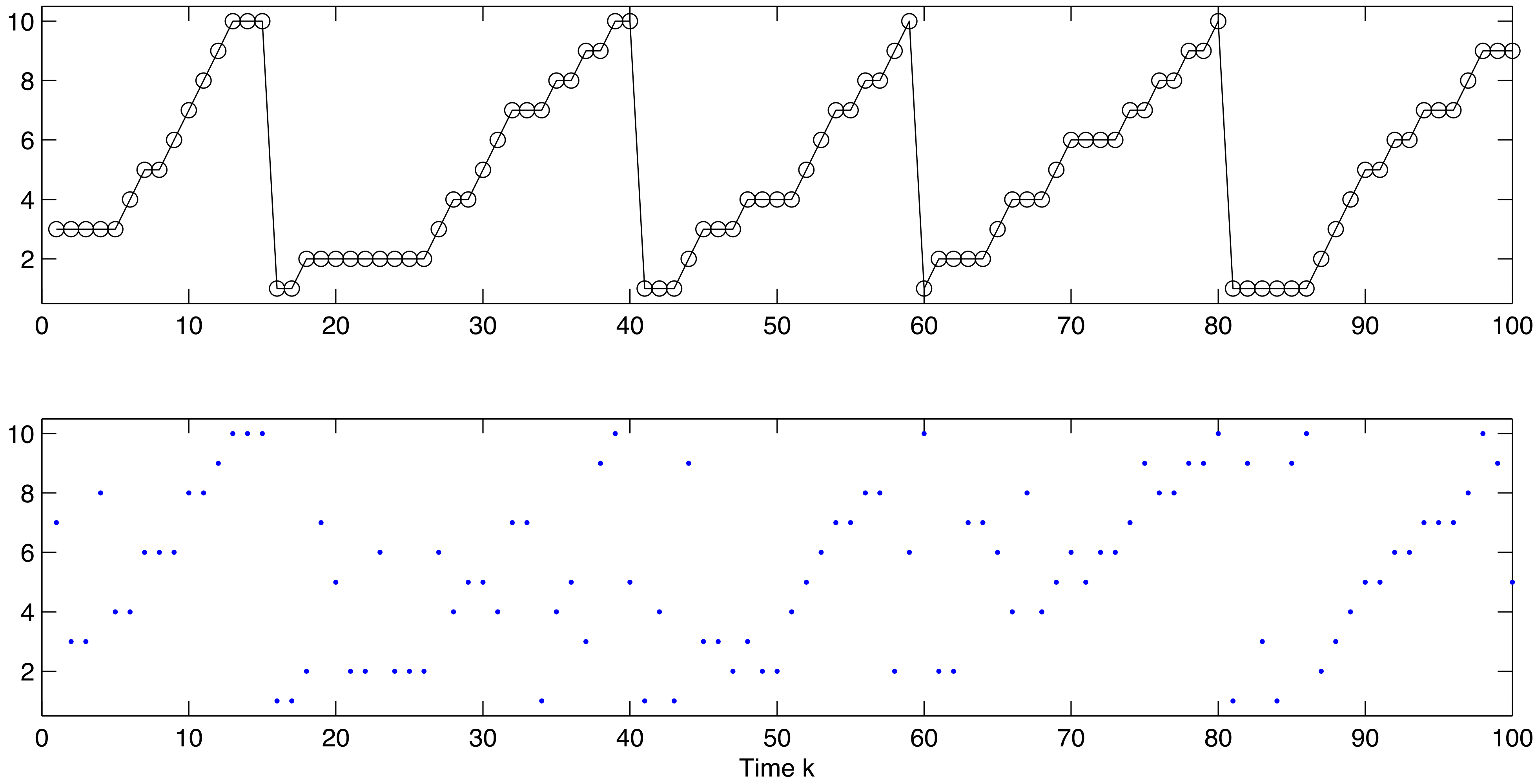


$$(1-\epsilon)\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} + \epsilon \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

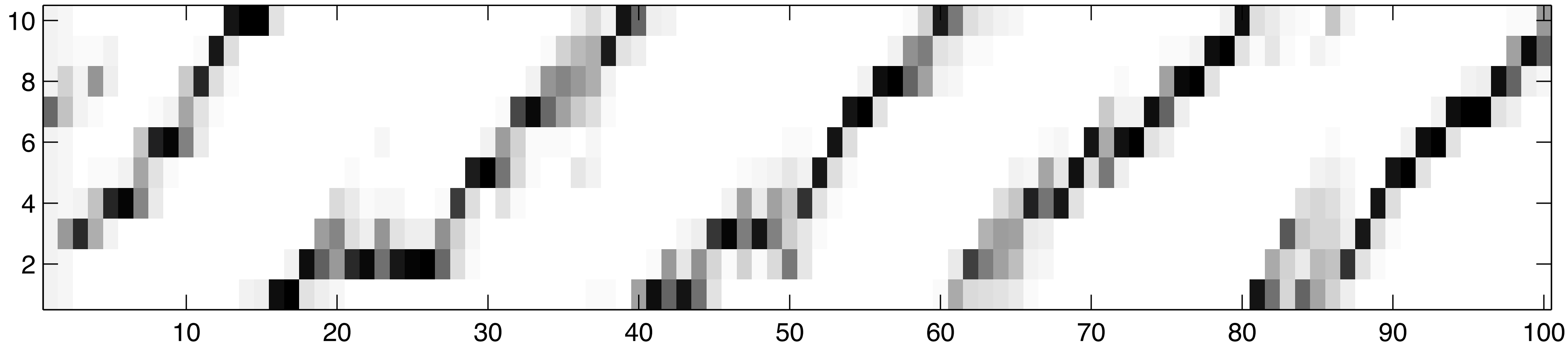
- Observation model $p(y_k|x_k)$

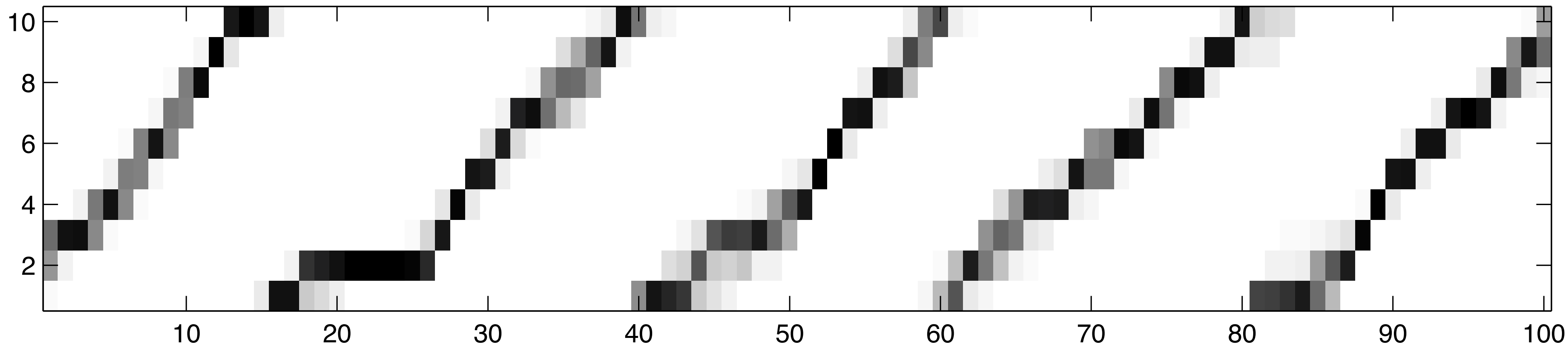
$$y_k \sim w\delta(y_k - x_k) + (1-w)u(1, N)$$

Example: Hidden Markov Model

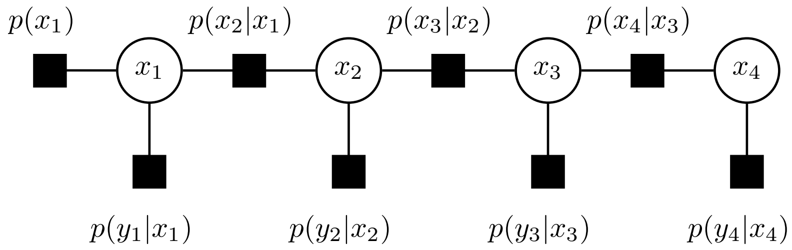


Example: Hidden Markov Model





Exact Inference in HMM, Forward/Backward Algorithm



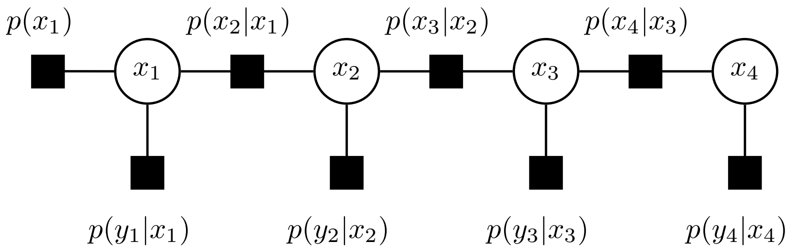
- Forward Pass

$$
\begin{aligned}
p(y_{1:K}) &= \sum_{x_{1:K}} p(y_{1:K}|x_{1:K})p(x_{1:K}) \\
&= \underbrace{\sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1})}_{\alpha_K} \cdots \sum_{x_2} p(x_3|x_2)\, \underbrace{p(y_2|x_2) \overbrace{\sum_{x_1} p(x_2|x_1)}^{\alpha_{2|1}}}_{\alpha_2} \underbrace{p(y_1|x_1)\, \overbrace{p(x_1)}^{\alpha_{1|0}}}_{\alpha_1}
\end{aligned}
$$

- Backward Pass

$$
p(y_{1:K}) = \sum_{x_1} p(x_1)p(y_1|x_1) \ldots \underbrace{\sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})}_{\beta_{K-2}} \underbrace{\sum_{x_K} p(x_K|x_{K-1})p(y_K|x_K)}_{\beta_{K-1}} \underbrace{\mathbf{1}}_{\beta_K}
$$

Exact Inference in HMM, Viterbi Algorithm



- Merely replace sum by max, equivalent to dynamic programming
- Forward Pass

$$
\begin{aligned}
p(y_{1:K}|x^*_{1:K}) &= \max_{x_{1:K}} p(y_{1:K}|x_{1:K})p(x_{1:K}) \\
&= \underbrace{\max_{x_K} p(y_T|x_K) \max_{x_{K-1}} p(x_K|x_{K-1})}_{\alpha_K} \dots \max_{x_2} p(x_3|x_2)\, \underbrace{p(y_2|x_2)\, \overbrace{\max_{x_1} p(x_2|x_1)}^{\alpha_{2|1}}}_{\alpha_2}\, \underbrace{p(y_1|x_1)\, \overbrace{p(x_1)}^{\alpha_{1|0}}}_{\alpha_1}
\end{aligned}
$$

- Backward Pass

$$
p(y_{1:K}|x^*_{1:K}) = \max_{x_1} p(x_1)p(y_1|x_1) \dots \underbrace{\max_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})}_{\beta_{K-2}}\, \underbrace{\max_{x_K} p(x_K|x_{K-1})p(y_K|x_K)}_{\beta_{K-1}}\, \underbrace{\mathbf{1}}_{\beta_K}
$$

Implementation of Forward-Backward

1. Setup a parameter structure

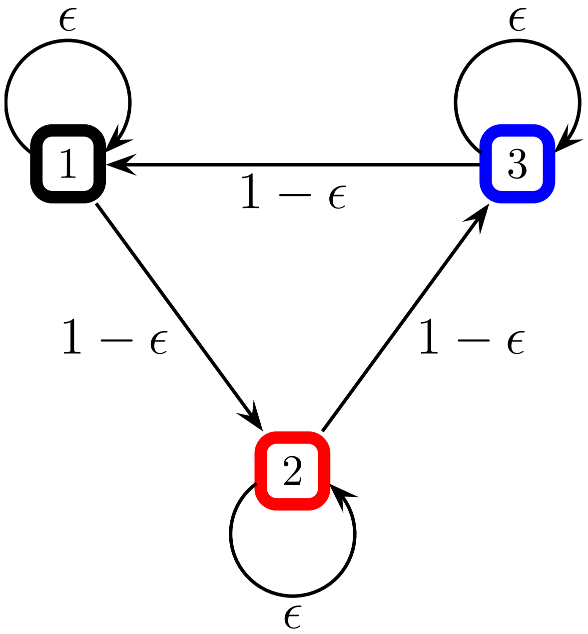
2. Generate data from the true model

3. Inference given true model parameters

4. Test and Visualisation

Example: Hidden Markov Model

- State transition model (a N by N matrix)



$$(1-\epsilon)\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} + \epsilon \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

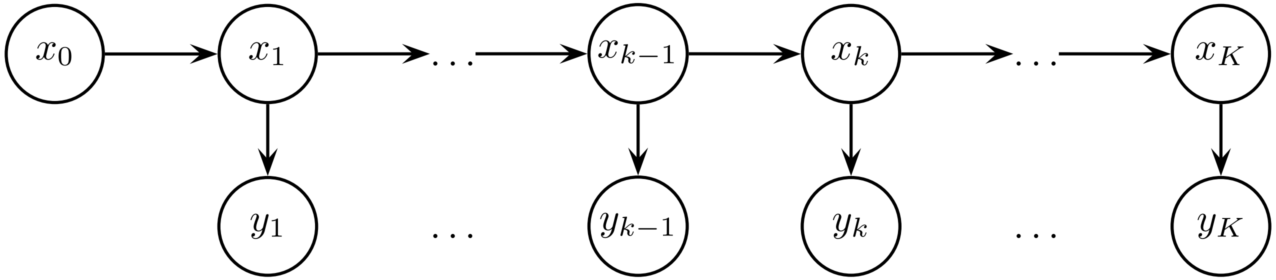
- Observation model $p(y_k|x_k)$

$$y_k \quad \sim \quad w\delta(y_k - x_k) + (1-w)u(1, N)$$

1. Setup a parameter structure

```
N = 50;     % Number of states

% Transition model;
ep = 0.5;  % Probability of not-moving
E = eye(N);
A = ep*E + (1-ep)*E(:, [2:N 1]); % Transition Matrix

% Observation model
w = 0.3; % Probability of observing true state
C = w*E + (1-w)*ones(N)/N; % Observation matrix

% Prior p(x_1)
pri = ones(N, 1)/N;

% Create a parameter structure
hm = struct('A', A, 'C', C, 'p_x1', pri);
```

2. Generate data from the true model

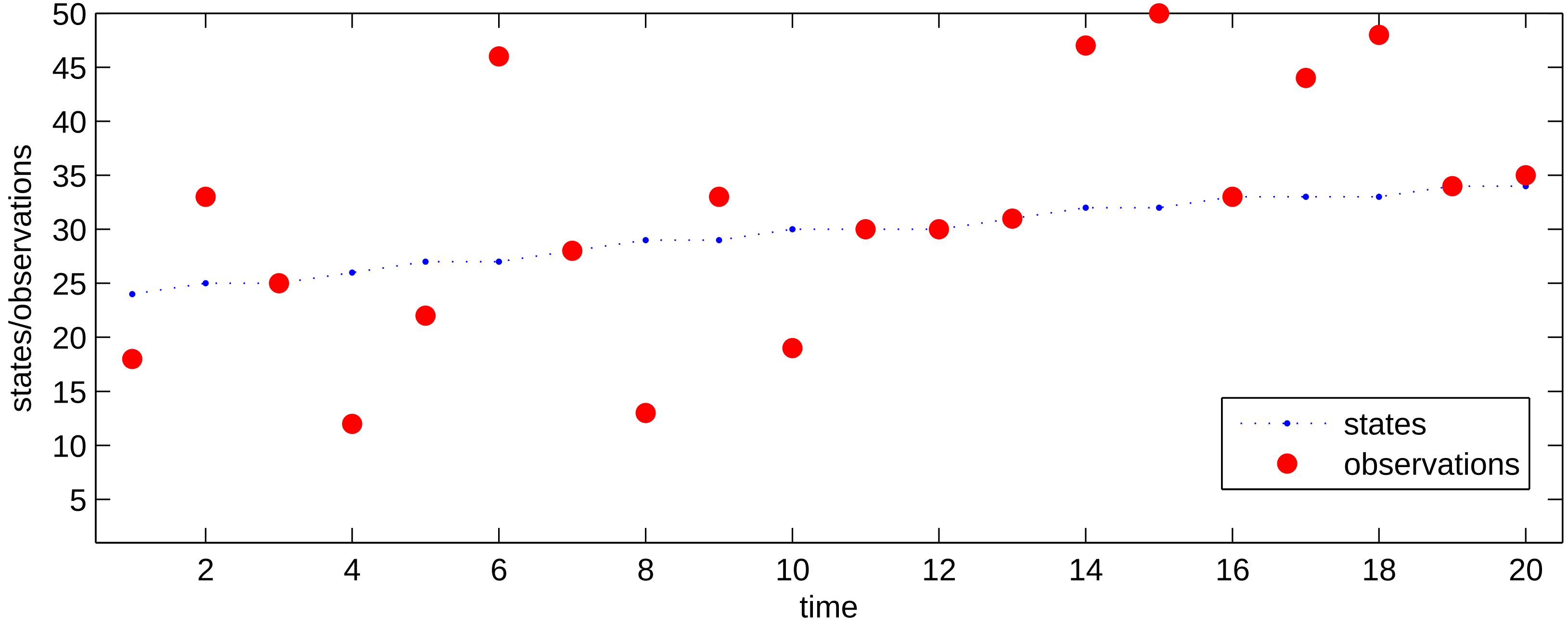


$$
\begin{aligned}
x_k | x_{k-1} &\sim p(x_k | x_{k-1}) \\
y_k | x_k &\sim p(y_k | x_k)
\end{aligned}
$$

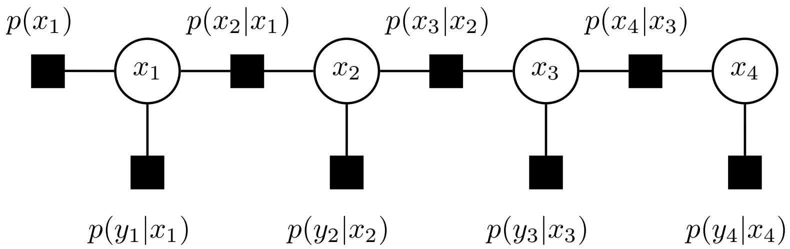
2. Generate data from the true model

```
function [obs, state] = hmm_generate_data(hm, K)
% Inputs :
%          hm : A HMM parameter structure
%           K : Number of time slices to simulate
% Outputs :
%           obs, state : Observations and the state trajectory

state = zeros(1, K);
obs = zeros(1, K);
for k=1:K,
    if k==1,
        state(k) = randgen(hm.p_x1);
    else
        state(k) = randgen(hm.A(:, state(k-1)));
    end;
    obs(k) = randgen(hm.C(:, state(k)));
end;
```

2. Generate data from the true model



3. Inference. Forward pass



- Predict

$$
\begin{aligned}
\alpha_{k|k-1}(x_k) &= p(y_{1:k-1}, x_k) = \sum_{x_{k-1}} p(x_k|x_{k-1}) p(y_{1:k-1}, x_{k-1}) \\
&= \sum_{x_{k-1}} p(x_k|x_{k-1}) \alpha_{k-1|k-1}(x_{k-1})
\end{aligned}
$$

- Update

$$
\begin{aligned}
\alpha_{k|k}(x_k) &= p(y_{1:k}, x_k) = p(y_k|x_k) p(y_{1:k-1}, x_k) \\
&= p(y_k|x_k) \alpha_{k|k-1}(x_k)
\end{aligned}
$$

$$
\begin{aligned}
p(y_{1:K}) &= \sum_{x_{1:K}} p(y_{1:K}|x_{1:K})p(x_{1:K}) \\
&= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2) \sum_{x_1} p(x_2|x_1) \underbrace{p(y_1|x_1) \overbrace{p(x_1)}^{\alpha_{1|0}}}_{\alpha_{1|1}} \\
&= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2) \sum_{x_1} p(x_2|x_1) \color{red}{\alpha_{1|1}(x_1)} \\
&= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2)p(y_2|x_2) \color{blue}{\alpha_{2|1}(x_2)} \\
&= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \cdots \sum_{x_2} p(x_3|x_2) \color{red}{\alpha_{2|2}(x_2)} \\
&= \sum_{x_K} p(y_K|x_K) \sum_{x_{K-1}} p(x_K|x_{K-1}) \ldots \color{blue}{\alpha_{3|2}(x_3)}
\end{aligned}
$$

3. Inference: Forward pass

```
log_alpha = zeros(N, K);
log_alpha_predict = zeros(N, K);
for k=1:K,
    if k==1,
        log_alpha_predict(:,k) = log(hm.p_x1);
    else
        log_alpha_predict(:,k) ...
            = state_predict(hm.A, log_alpha(:, k-1));
    end;
    log_alpha(:, k) ...
          = state_update(hm.C(y(k), :), log_alpha_predict(:,k));
end;
```

3. Inference. Predict

```
function [lpp] = state_predict(A, log_p)
% STATE_PREDICT Computes A*p in log domain
%
%  [lpp] = state_predict(A, log_p)
%
% Inputs :
%  A : State transition matrix
%   log_p : log p(x_{k-1}, y_{1:k-1}) Filtered potential
%
% Outputs :
%   lpp : log p(x_{k}, y_{1:k-1});  Predicted potential

mx = max(log_p(:));  % Stable computation
p = exp(log_p - mx);
lpp = log(A*p) + mx;
```

Numerically Stable computation of $\log(\sum_i \exp(l_i)))$

- Derivation

$$
\begin{aligned}
L &= \log(\sum_i \exp(l_i)) \\
&= \log(\sum_i \exp(l_i) \frac{\exp(l^*)}{\exp(l^*)}) \\
&= \log(\exp(l^*) \sum_i \exp(l_i - l^*)) \\
&= l^* + \log(\sum_i \exp(l_i - l^*))
\end{aligned}
$$

- We take l^* as the maximum $l^* = \max_i l_i$

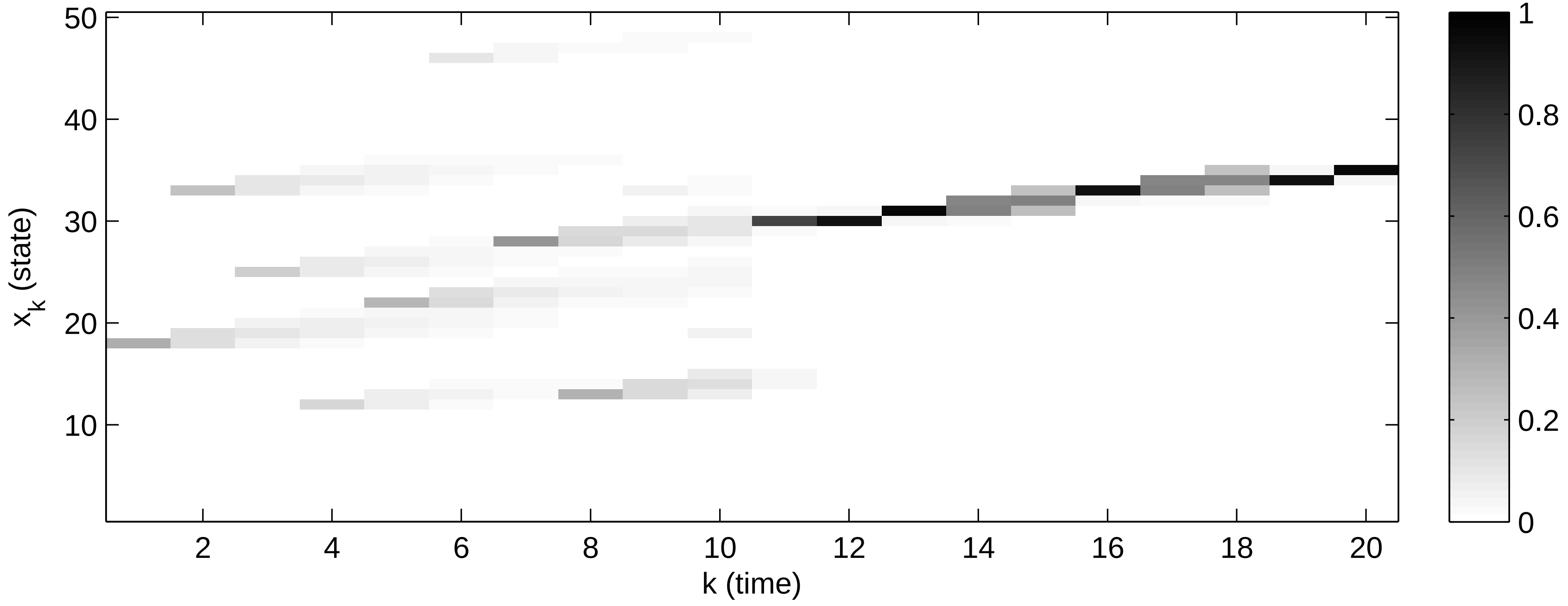
- Assignment: Implement above as a function `logsumexp(l)`

3. Inference. Update

```
function [lup] = state_update(obs, log_p)
% STATE_UPDATE State update in log domain
%
%  [lup] = state_update(obs, log_p)
%
% Inputs :
%         obs : p(y_k| x_k)
%         log_p : log p(x_k, y_{1, k-1})
%
% Outputs :
% lup : log p(x_k, y_{1, k-1})  p(y_k| x_k)

lup = log(obs(:)) + log_p;
```

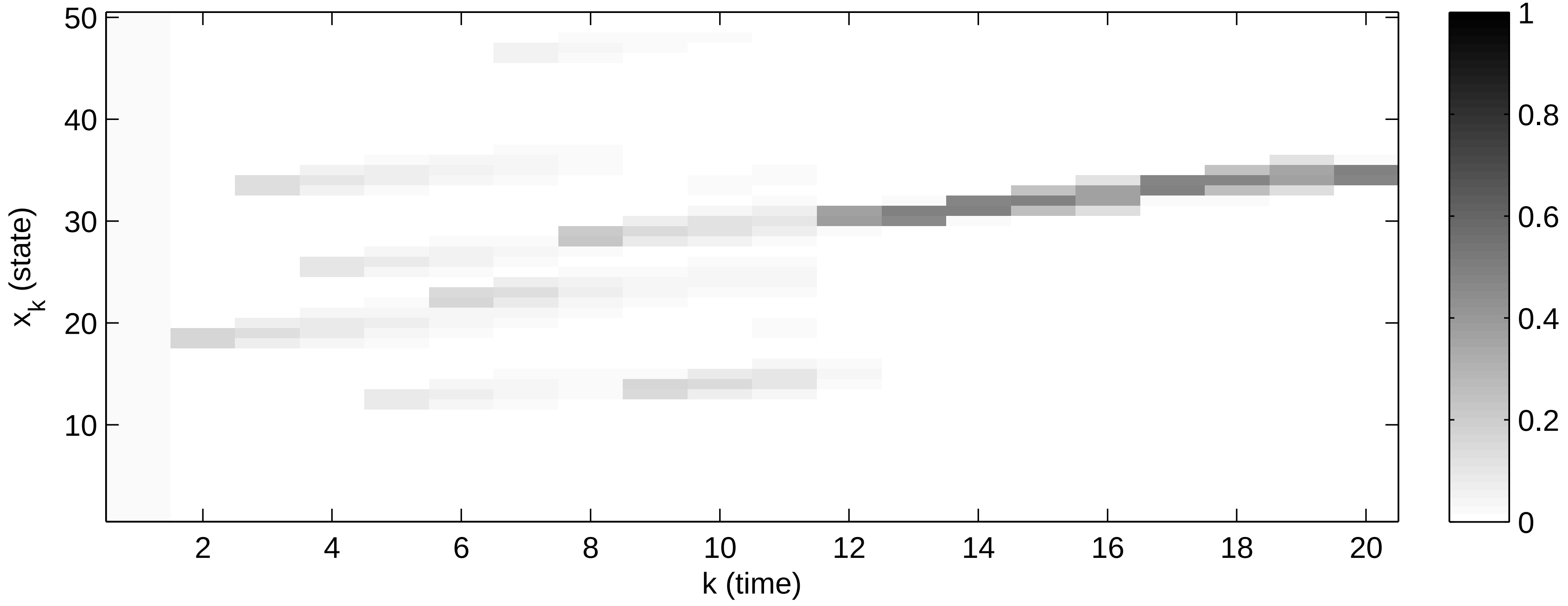
3. Inference. Forward pass.

$$\alpha_{k|k} \quad \equiv \quad p(y_{1:k}, x_k)$$

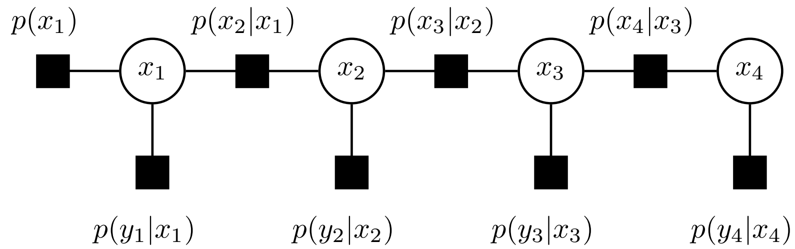


3. Inference. Forward pass

$$\color{red}{\alpha_{k|k-1}} \equiv p(y_{1:k-1}, x_k)$$



3. Inference. Backward pass



- "Postdict"

$$
\begin{aligned}
\beta_{k|k+1}(x_k) &= p(y_{k+1:K}|x_k) = \sum_{x_{k+1}} p(x_{k+1}|x_k) p(y_{k+1:K}|x_{k+1}) \\
&= \sum_{x_{k+1}} p(x_{k+1}|x_k) \beta_{k+1|k+1}(x_{k+1})
\end{aligned}
$$

- Update

$$
\begin{aligned}
\beta_{k|k}(x_k) &= p(y_{k:K}|x_k) = p(y_k|x_k) p(y_{k+1:K}|x_k) \\
&= p(y_k|x_k) \beta_{k|k+1}(x_k)
\end{aligned}
$$

$$
\begin{aligned}
p(y_{1:K}) &= \sum_{x_1} p(x_1)p(y_1|x_1)\dots\sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})\sum_{x_K} p(x_K|x_{K-1})p(y_K|x_K)\underbrace{\mathbf{1}}_{\beta_{K|K+1}} \\
&= \sum_{x_1} p(x_1)p(y_1|x_1)\cdots\sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})\sum_{x_K} p(x_K|x_{K-1})\beta_{K|K} \\
&= \sum_{x_1} p(x_1)p(y_1|x_1)\cdots\sum_{x_{K-1}} p(x_{K-1}|x_{K-2})p(y_{K-1}|x_{K-1})\beta_{K-1|K} \\
&= \sum_{x_1} p(x_1)p(y_1|x_1)\cdots\sum_{x_{K-1}} p(x_{K-1}|x_{K-2})\beta_{K-1|K-1} \\
&= \sum_{x_1} p(x_1)p(y_1|x_1)\dots\beta_{K-2|K-1}
\end{aligned}
$$

3. Inference. Backward pass

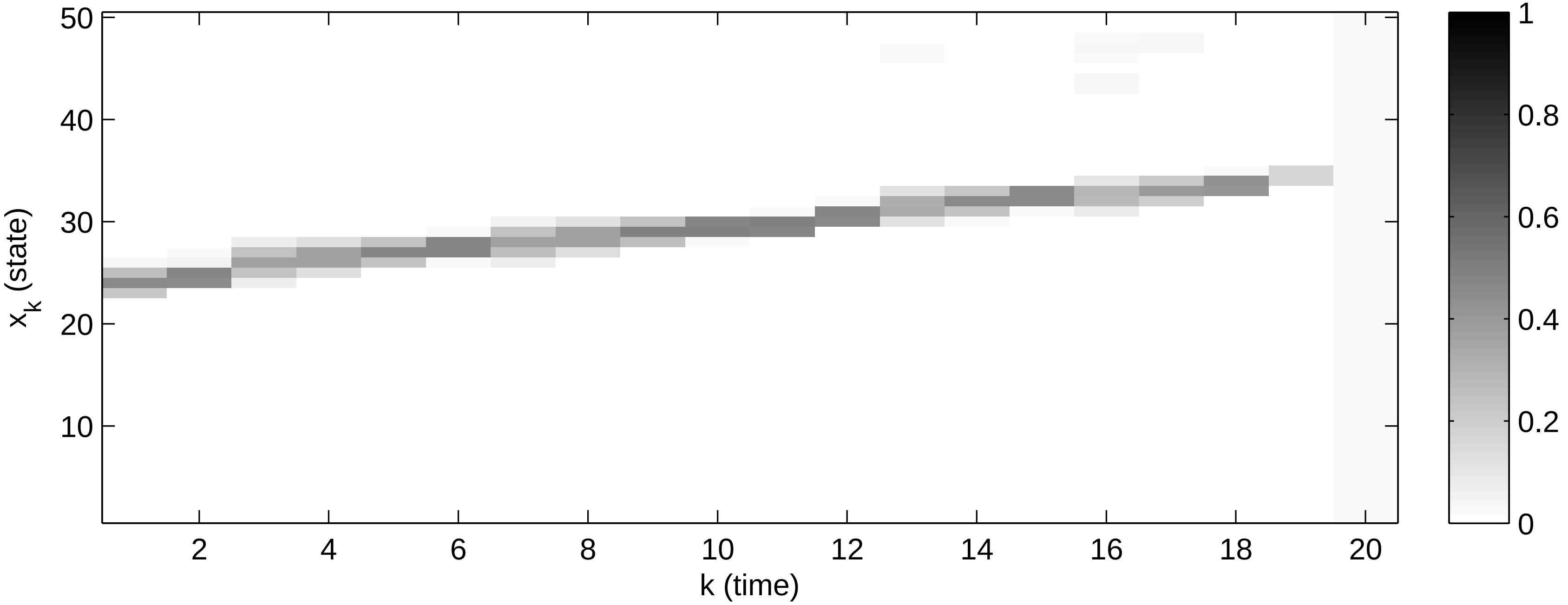
```
log_beta = zeros(N, T);
log_beta_postdict = zeros(N, T);
for t=T:-1:1,
    if t==T,
        log_beta_postdict(:,t) = zeros(N,1);
    else
        log_beta_postdict(:,t) ...
               = state_postdict(hm.A, log_beta(:, t+1));
    end;
    log_beta(:, t) ...
         = state_update(hm.C(y(t), :), log_beta_postdict(:,t));
end;
```

3. Inference. Postdict.

```
function [lpp] = state_postdict(A, log_p)
% STATE_POSTDICT Computes A'*p in log domain
%
%  [lpp] = state_postdict(A, log_p)
%
% Inputs :
% A : State transition matrix
%           log_p : log p(y_{k+1:K}|x_{k+1})    Updated potential
%
% Outputs :
% lpp : log p(y_{k+1:K}| x_k)    Postdicted potential

mx = max(log_p(:));  % Stable computation
p = exp(log_p - mx);
lpp = log(A'*p) + mx;
```

3. Inference. Backward pass

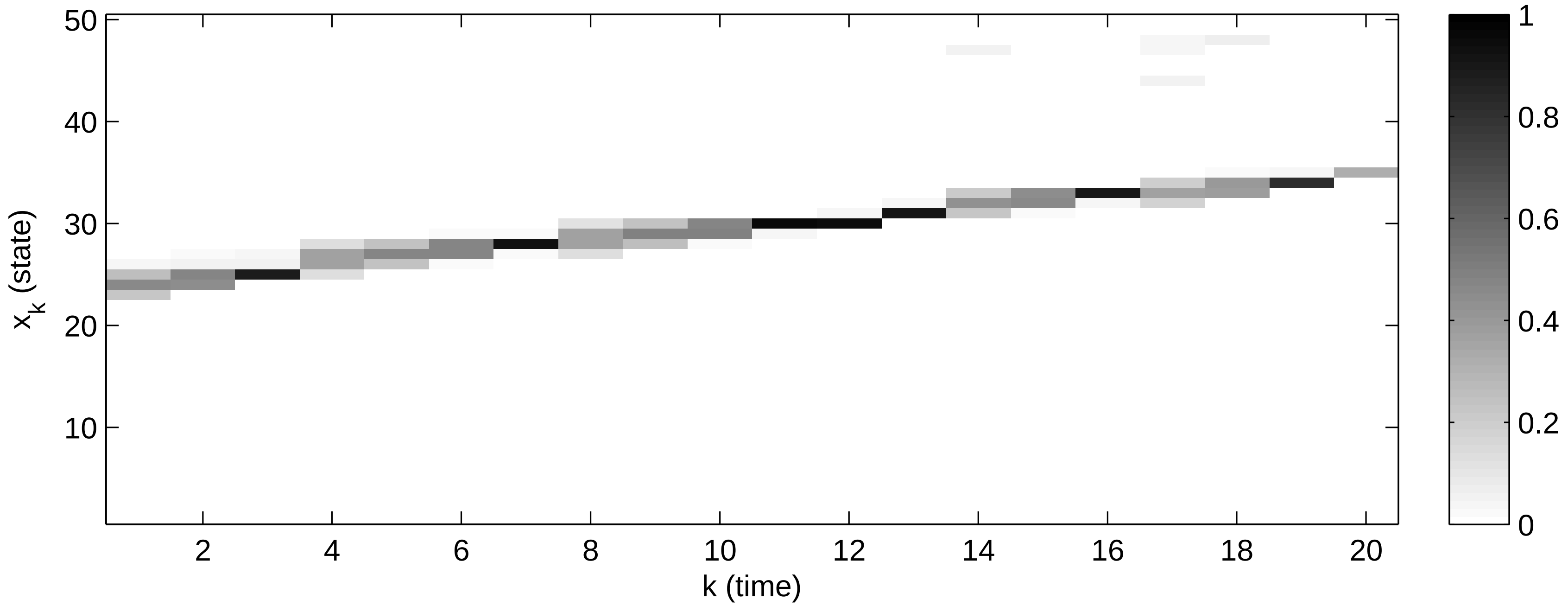
$$\beta_{k|k+1}(x_k) \;=\; p(y_{k+1:K}|x_k)$$



We visualise $\hat{\beta} \propto \beta_{k|k+1}(x_k)u(x_k)$

3. Inference. Backward pass

$$\color{red}{\beta_{k|k}(x_k)} = p(y_{k:K}|x_k)$$



3. Inference. Smoothing.

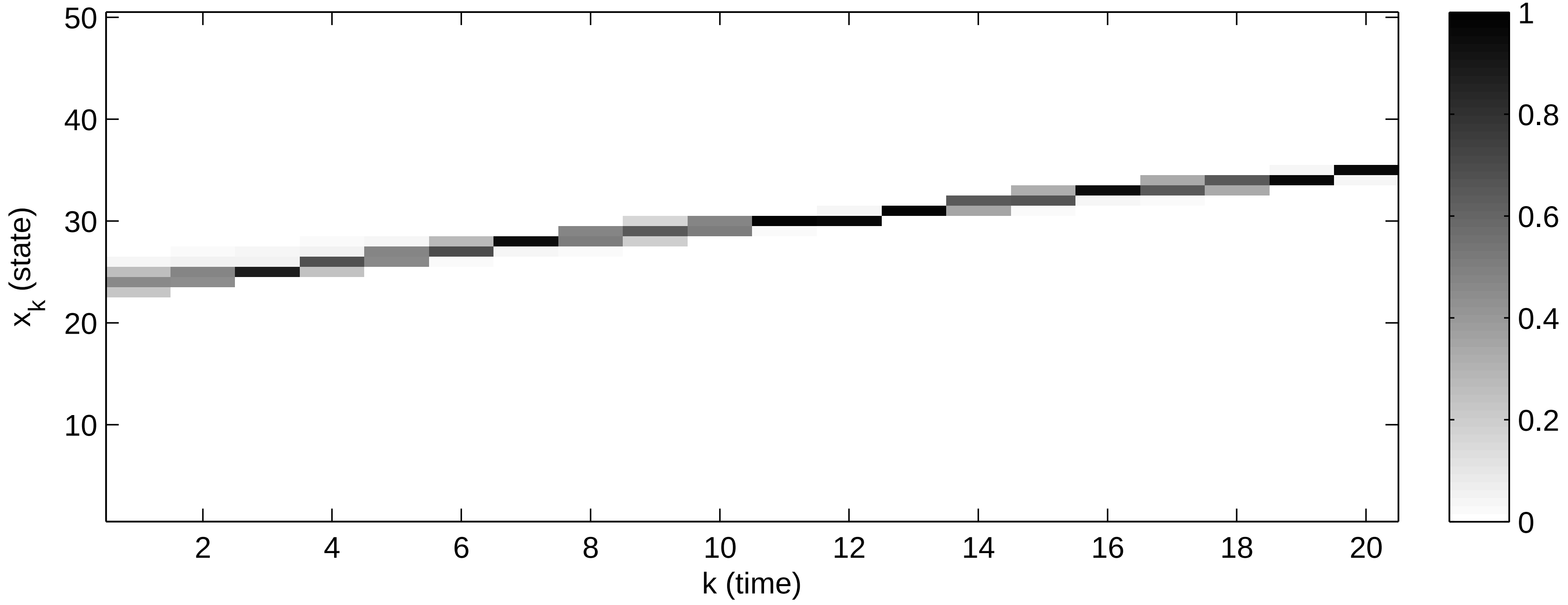
$$
\begin{aligned}
p(y_{1:K}, x_k) &= p(y_{1:k}, x_k) p(y_{k+1:K} | x_k) \\
&= \alpha_{k|k}(x_k) \beta_{k|k+1}(x_k) \\
&\equiv \gamma_k(x_k)
\end{aligned}
$$

Alternatives

$$
\begin{aligned}
\gamma_k(x_k) &= \alpha_{k|k-1}(x_k) \beta_{k|k}(x_k) \\
&= \alpha_{k|k-1}(x_k) p(y_k | x_k) \beta_{k|k+1}(x_k)
\end{aligned}
$$

3. Inference. Smoothing.

$$p(x_k|y_{1:K}) \quad \propto \quad p(y_{1:K}, x_k) = \color{red}{\alpha_{k|k}(x_k)}\color{blue}{\beta_{k|k+1}(x_k)} \equiv \gamma_k(x_k)$$



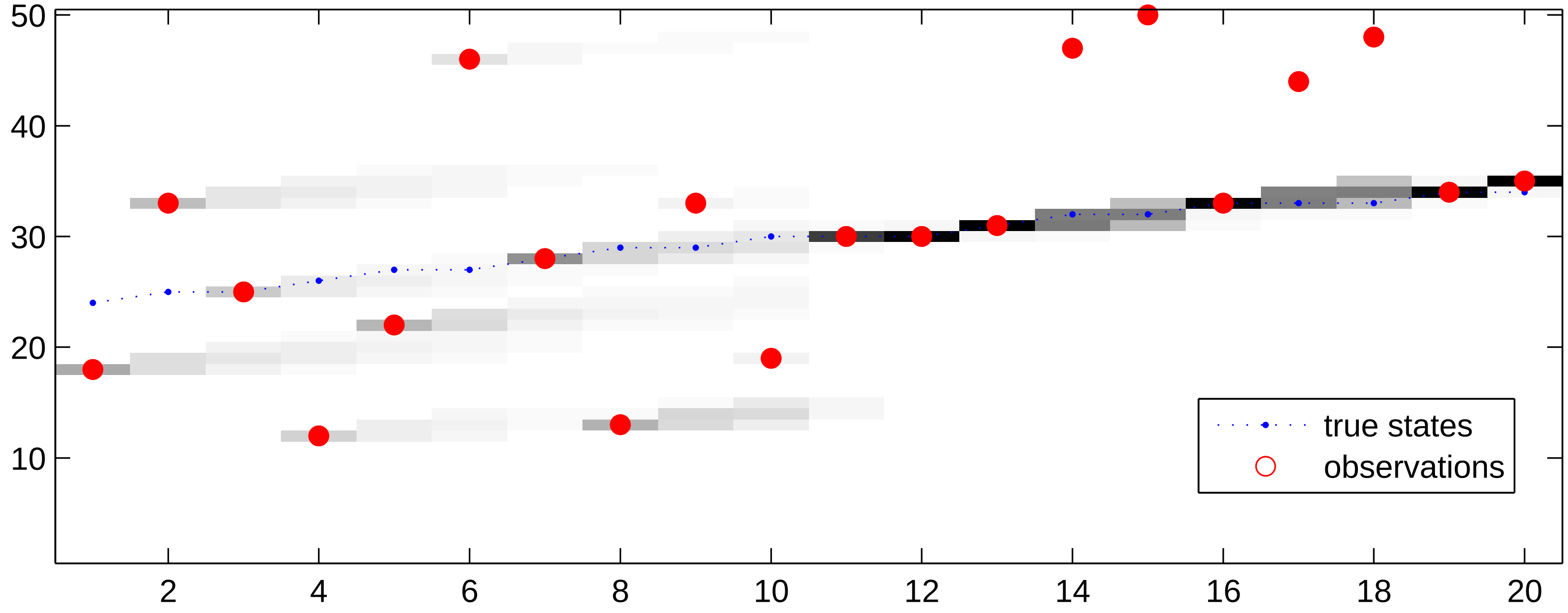
3. Inference. Smoothing.

```
log_gamma = log_alpha + log_beta_postdict
```

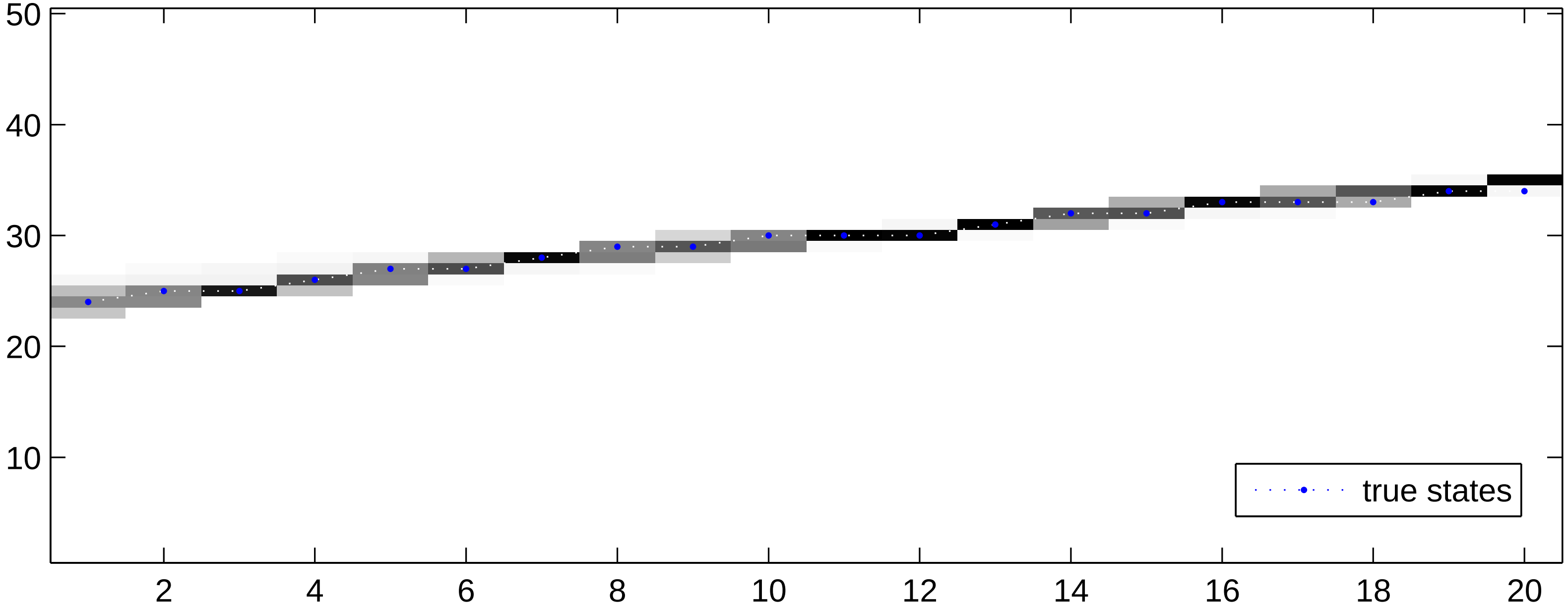
4. Test and Visualisation

```
imagesc(normalize_exp(log_gamma, 1));
set(gca, 'ydir', 'n');
colormap(flipud(gray));
xlabel('k (time)'); ylabel('x_k (state)');
caxis([0 1]);
colorbar

% This has to be constant !! (why)
plot(log_sum_exp(log_gamma, 1));
```

4. Test and Visualise. Filter.



4. Test and Visualise. Smoother.



Keywords Summary

Forward-Backward