18 State space models

18.1 Introduction

A **state space model** or **SSM** is just like an HMM, except the hidden states are continuous. The model can be written in the following generic form:

$$\mathbf{z}_t = g(\mathbf{u}_t, \mathbf{z}_{t-1}, \boldsymbol{\epsilon}_t) \tag{18.1}$$

$$\mathbf{y}_t = h(\mathbf{z}_t, \mathbf{u}_t, \boldsymbol{\delta}_t) \tag{18.2}$$

where $\mathbf{z}_t$ is the hidden state, $\mathbf{u}_t$ is an optional input or control signal, $\mathbf{y}_t$ is the observation, g is the **transition model**, h is the **observation model**, $\boldsymbol{\epsilon}_t$ is the system noise at time t, and $\boldsymbol{\delta}_t$ is the observation noise at time t. We assume that all parameters of the model, $\boldsymbol{\theta}$, are known; if not, they can be included into the hidden state, as we discuss below.

One of the primary goals in using SSMs is to recursively estimate the belief state, $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}, \boldsymbol{\theta})$. (Note: we will often drop the conditioning on $\mathbf{u}$ and $\boldsymbol{\theta}$ for brevity.) We will discuss algorithms for this later in this chapter. We will also discuss how to convert our beliefs about the hidden state into predictions about future observables by computing the posterior predictive $p(\mathbf{y}_{t+1}|\mathbf{y}_{1:t})$.

An important special case of an SSM is where all the CPDs are linear-Gaussian. In other words, we assume

- The transition model is a linear function

$$\mathbf{z}_t = \mathbf{A}_t \mathbf{z}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \boldsymbol{\epsilon}_t \tag{18.3}$$

- The observation model is a linear function

$$\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \mathbf{D}_t \mathbf{u}_t + \boldsymbol{\delta}_t \tag{18.4}$$

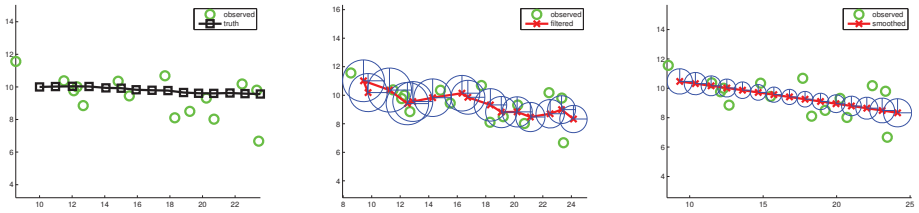
- The system noise is Gaussian

$$\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \tag{18.5}$$

- The observation noise is Gaussian

$$\boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \tag{18.6}$$

This model is called a **linear-Gaussian SSM** (**LG-SSM**) or a **linear dynamical system** (**LDS**). If the parameters $\boldsymbol{\theta}_t = (\mathbf{A}_t, \mathbf{B}_t, \mathbf{C}_t, \mathbf{D}_t, \mathbf{Q}_t, \mathbf{R}_t)$ are independent of time, the model is called **stationary**.



(a) (b) (c)

Figure 18.1 Illustration of Kalman filtering and smoothing. (a) Observations (green cirles) are generated by an object moving to the right (true location denoted by black squares). (b) Filtered estimated is shown by dotted red line. Red cross is the posterior mean, blue circles are 95% confidence ellipses derived from the posterior covariance. For clarity, we only plot the ellipses every other time step. (c) Same as (b), but using offline Kalman smoothing. Figure generated by `kalmanTrackingDemo`.

The LG-SSM is important because it supports exact inference, as we will see. In particular, if the initial belief state is Gaussian, $p(\mathbf{z}_1) = \mathcal{N}(\boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0})$, then all subsequent belief states will also be Gaussian; we will denote them by $p(\mathbf{z}_t|\mathbf{y}_{1:t}) = \mathcal{N}(\boldsymbol{\mu}_{t|t}, \boldsymbol{\Sigma}_{t|t})$. (The notation $\boldsymbol{\mu}_{t|\tau}$ denotes $\mathbb{E}[\mathbf{z}_t|\mathbf{y}_{1:\tau}]$, and similarly for $\boldsymbol{\Sigma}_{t|t}$; thus $\boldsymbol{\mu}_{t|0}$ denotes the prior for $\mathbf{z}_1$ before we have seen any data. For brevity we will denote the posterior belief states using $\boldsymbol{\mu}_{t|t} = \boldsymbol{\mu}_t$ and $\boldsymbol{\Sigma}_{t|t} = \boldsymbol{\Sigma}_t$.) We can compute these quantities efficiently using the celebrated Kalman filter, as we show in Section 18.3.1. But before discussing algorithms, we discuss some important applications.

18.2 Applications of SSMs

SSMs have many applications, some of which we discuss in the sections below. We mostly focus on LG-SSMs, for simplicity, although non-linear and/or non-Gaussian SSMs are even more widely used.

18.2.1 SSMs for object tracking

One of the earliest applications of Kalman filtering was for tracking objects, such as airplanes and missiles, from noisy measurements, such as radar. Here we give a simplified example to illustrate the key ideas. Consider an object moving in a 2D plane. Let z_{1t} and z_{2t} be the horizontal and vertical locations of the object, and $\dot{z}_{1t}$ and $\dot{z}_{2t}$ be the corresponding velocity. We can represent this as a state vector $\mathbf{z}_t \in \mathbb{R}^4$ as follows:

$$\mathbf{z}_t^T = \begin{pmatrix} z_{1t} & z_{2t} & \dot{z}_{1t} & \dot{z}_{2t} \end{pmatrix}. \tag{18.7}$$

Let us assume that the object is moving at constant velocity, but is "perturbed" by random Gaussian noise (e.g., due to the wind). Thus we can model the system dynamics as follows:

$$\mathbf{z}_t = \mathbf{A}_t \mathbf{z}_{t-1} + \boldsymbol{\epsilon}_t \tag{18.8}$$

$$\begin{pmatrix} z_{1t} \\ z_{2t} \\ \dot{z}_{1t} \\ \dot{z}_{2t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} z_{1,t-1} \\ z_{2,t-1} \\ \dot{z}_{1,t-1} \\ \dot{z}_{2,t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \\ \epsilon_{3t} \\ \epsilon_{4t} \end{pmatrix} \tag{18.9}$$

where $\boldsymbol{\epsilon}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{Q})$ is the system noise, and Δ is the **sampling period**. This says that the new location $z_{j,t}$ is the old location $z_{j,t-1}$ plus Δ times the old velocity $\dot{z}_{j,t-1}$, plus random noise, ϵ_{jt}, for $j = 1 : 2$. Also, the new velocity $\dot{z}_{j,t}$ is the old velocity $\dot{z}_{j,t-1}$ plus random noise, ϵ_{jt}, for $j = 3 : 4$. This is called a **random accelerations model**, since the object moves according to Newton's laws, but is subject to random changes in velocity.

Now suppose that we can observe the location of the object but not its velocity. Let $\mathbf{y}_t \in \mathbb{R}^2$ represent our observation, which we assume is subject to Gaussian noise. We can model this as follows:

$$\mathbf{y}_t = \mathbf{C}_t \mathbf{z}_t + \boldsymbol{\delta}_t \tag{18.10}$$

$$\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} z_{1t} \\ z_{2t} \\ \dot{z}_{1t} \\ \dot{z}_{2t} \end{pmatrix} + \begin{pmatrix} \delta_{1t} \\ \delta_{2t} \\ \delta_{3t} \\ \delta_{4t} \end{pmatrix} \tag{18.11}$$

where $\boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0}, \mathbf{R})$ is the measurement noise.

Finally, we need to specify our initial (prior) beliefs about the state of the object, $p(\mathbf{z}_1)$. We will assume this is a Gaussian, $p(\mathbf{z}_1) = \mathcal{N}(\mathbf{z}_1 | \boldsymbol{\mu}_{1|0}, \boldsymbol{\Sigma}_{1|0})$. We can represent prior ignorance by making $\boldsymbol{\Sigma}_{1|0}$ suitably "broad", e.g., $\boldsymbol{\Sigma}_{1|0} = \infty \mathbf{I}$. We have now fully specified the model and can perform sequential Bayesian updating to compute $p(\mathbf{z}_t | \mathbf{y}_{1:t})$ using an algorithm known as the Kalman filter, to be described in Section 18.3.1.

Figure 18.1(a) gives an example. The object moves to the right and generates an observation at each time step (think of "blips" on a radar screen). We observe these blips and filter out the noise by using the Kalman filter. At every step, we have $p(\mathbf{z}_t | \mathbf{y}_{1:t})$, from which we can compute $p(z_{1t}, z_{2t} | \mathbf{y}_{1:t})$ by marginalizing out the dimensions corresponding to the velocities. (This is easy to do since the posterior is Gaussian.) Our "best guess" about the location of the object is the posterior mean, $E[\mathbf{z}_t | \mathbf{y}_{1:t}]$, denoted as a red cross in Figure 18.1(b). Our uncertainty associated with this is represented as an ellipse, which contains 95% of the probability mass. We see that our uncertainty goes down over time, as the effects of the initial uncertainty get "washed out". We also see that the estimated trajectory has "filtered out" some of the noise. To obtain the much smoother plot in Figure 18.1(c), we need to use the Kalman smoother, which computes $p(\mathbf{z}_t | \mathbf{y}_{1:T})$; this depends on "future" as well as "past" data, as discussed in Section 18.3.2.

18.2.2 Robotic SLAM

Consider a robot moving around an unknown 2d world. It needs to learn a map and keep track of its location within that map. This problem is known as **simultaneous localization and**

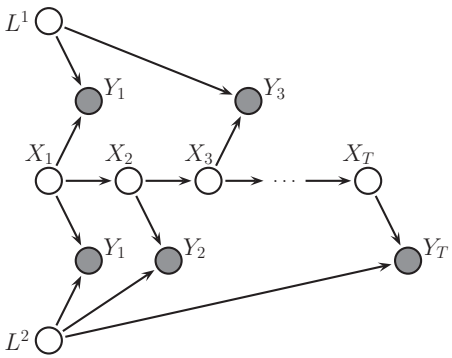


Figure 18.2 Illustration of graphical model underlying SLAM. L^i is the fixed location of landmark i, $\mathbf{x}_t$ is the location of the robot, and $\mathbf{y}_t$ is the observation. In this trace, the robot sees landmarks 1 and 2 at time step 1, then just landmark 2, then just landmark 1, etc. Based on Figure 15.A.3 of (Koller and Friedman 2009).

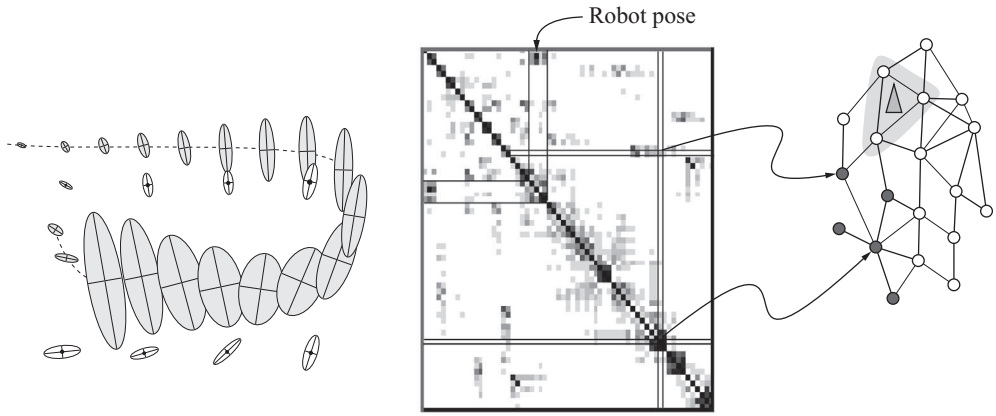


Figure 18.3 Illustration of the SLAM problem. (a) A robot starts at the top left and moves clockwise in a circle back to where it started. We see how the posterior uncertainty about the robot's location increases and then decreases as it returns to a familar location, closing the loop. If we performed smoothing, this new information would propagate backwards in time to disambiguate the entire trajectory. (b) We show the precision matrix, representing sparse correlations between the landmarks, and between the landmarks and the robot's position (pose). This sparse precision matrix can be visualized as a Gaussian graphical model, as shown. Source: Figure 15.A.3 of (Koller and Friedman 2009) . Used with kind permission of Daphne Koller.

mapping, or **SLAM** for short, and is widely used in mobile robotics, as well as other applications such as indoor navigation using cellphones (since GPS does not work inside buildings).

Let us assume we can represent the map as the 2d locations of a fixed set of K landmarks, denote them by $L^1, \ldots, L^K$ (each is a vector in $\mathbb{R}^2$). For simplicity, we will assume these are uniquely identifiable. Let $\mathbf{x}_t$ represent the unknown location of the robot at time t. We define the state space to be $\mathbf{z}_t = (\mathbf{x}_t, \mathbf{L}^{1:K})$; we assume the landmarks are static, so their motion model is a constant, and they have no system noise. If $\mathbf{y}_t$ measures the distance from $\mathbf{x}_t$ to the set of closest landmarks, then the robot can update its estimate of the landmark locations based on what it sees. Figure 18.2 shows the corresponding graphical model for the case where $K = 2$, and where on the first step it sees landmarks 1 and 2, then just landmark 2, then just landmark 1, etc.

If we assume the observation model $p(\mathbf{y}_t|\mathbf{z}_t, \mathbf{L})$ is linear-Gaussian, and we use a Gaussian motion model for $p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{u}_t)$, we can use a Kalman filter to maintain our belief state about the location of the robot and the location of the landmarks (Smith and Cheeseman 1986; Choset and Nagatani 2001).

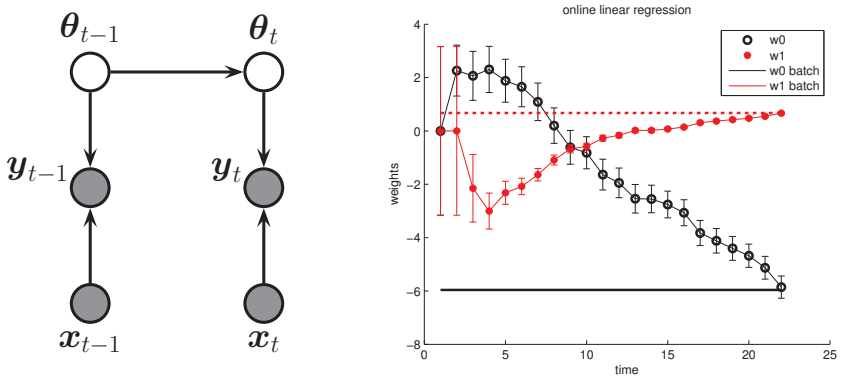
Over time, the uncertainty in the robot's location will increase, due to wheel slippage etc., but when the robot returns to a familiar location, its uncertainty will decrease again. This is called **closing the loop**, and is illustrated in Figure 18.3(a), where we see the uncertainty ellipses, representing $\mathrm{cov}\left[\mathbf{x}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}\right]$, grow and then shrink. (Note that in this section, we assume that a human is joysticking the robot through the environment, so $\mathbf{u}_{1:t}$ is given as input, i.e., we do not address the decision-theoretic issue of choosing where to explore.)

Since the belief state is Gaussian, we can visualize the posterior covariance matrix $\mathbf{\Sigma}_t$. Actually, it is more interesting to visualize the posterior precision matrix, $\mathbf{\Lambda}_t = \mathbf{\Sigma}_t^{-1}$, since that is fairly sparse, as shown in Figure 18.3(b). The reason for this is that zeros in the precision matrix correspond to absent edges in the corresponding undirected Gaussian graphical model (see Section 19.4.4). Initially all the landmarks are uncorrelated (assuming we have a diagonal prior on $\mathbf{L}$), so the GGM is a disconnected graph, and $\mathbf{\Lambda}_t$ is diagonal. However, as the robot moves about, it will induce correlation between nearby landmarks. Intuitively this is because the robot is estimating its position based on distance to the landmarks, but the landmarks' locations are being estimated based on the robot's position, so they all become inter-dependent. This can be seen more clearly from the graphical model in Figure 18.2: it is clear that L^1 and L^2 are not d-separated by $\mathbf{y}_{1:t}$, because there is a path between them via the unknown sequence of $\mathbf{x}_{1:t}$ nodes. As a consequence of the precision matrix becoming denser, exact inference takes $O(K^3)$ time. (This is an example of the entanglement problem for inference in DBNs.) This prevents the method from being applied to large maps.

There are two main solutions to this problem. The first is to notice that the correlation pattern moves along with the location of the robot (see Figure 18.3(b)). The remaining correlations become weaker over time. Consequently we can dynamically "prune out" weak edges from the GGM using a technique called the thin junction tree filter (Paskin 2003) (junction trees are explained in Section 20.4).

A second approach is to notice that, conditional on knowing the robot's path, $\mathbf{x}_{1:t}$, the landmark locations are independent. That is, $p(\mathbf{L}|\mathbf{x}_{1:t}, \mathbf{y}_{1:t}) = \prod_{k=1}^{K} p(\mathbf{L}^k|\mathbf{x}_{1:t}, \mathbf{y}_{1:t})$. This forms the basis of a method known as FastSLAM, which combines Kalman filtering and particle filtering, as discussed in Section 23.6.3.

(Thrun et al. 2006) provides a more detailed account of SLAM and mobile robotics.



(a) (b)

Figure 18.4 (a) A dynamic generalization of linear regression. (b) Illustration of the recursive least squares algorithm applied to the model $p(y|\mathbf{x}, \boldsymbol{\theta}) = \mathcal{N}(y|w_0 + w_1 x, \sigma^2)$. We plot the marginal posterior of w_0 and w_1 vs number of data points. (Error bars represent $\mathbb{E}[w_j|y_{1:t}] \pm \sqrt{\mathrm{var}[w_j|y_{1:t}]}$.) After seeing all the data, we converge to the offline ML (least squares) solution, represented by the horizontal lines. Figure generated by `linregOnlineDemoKalman`.

18.2.3 Online parameter learning using recursive least squares

We can perform online Bayesian inference for the parameters of various statistical models using SSMs. In this section, we focus on linear regression; in Section 18.5.3.2, we discuss logistic regression.

The basic idea is to let the hidden state represent the regression parameters, and to let the (time-varying) observation model represent the current data vector. In more detail, define the prior to be $p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\theta}_0, \boldsymbol{\Sigma}_0)$. (If we want to do online ML estimation, we can just set $\boldsymbol{\Sigma}_0 = \infty\mathbf{I}$.) Let the hidden state be $\mathbf{z}_t = \boldsymbol{\theta}$; if we assume the regression parameters do not change, we can set $\mathbf{A}_t = \mathbf{I}$ and $\mathbf{Q}_t = 0\mathbf{I}$, so

$$p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, 0\mathbf{I}) = \delta_{\boldsymbol{\theta}_{t-1}}(\boldsymbol{\theta}_t) \tag{18.12}$$

(If we do let the parameters change over time, we get a so-called **dynamic linear model** (Harvey 1990; West and Harrison 1997; Petris et al. 2009).) Let $\mathbf{C}_t = \mathbf{x}_t^T$, and $\mathbf{R}_t = \sigma^2$, so the (non-stationary) observation model has the form

$$\mathcal{N}(\mathbf{y}_t|\mathbf{C}_t\mathbf{z}_t, \mathbf{R}_t) = \mathcal{N}(\mathbf{y}_t|\mathbf{x}_t^T\boldsymbol{\theta}_t, \sigma^2) \tag{18.13}$$

Applying the Kalman filter to this model provides a way to update our posterior beliefs about the parameters as the data streams in. This is known as the **recursive least squares** or **RLS** algorithm.

We can derive an explicit form for the updates as follows. In Section 18.3.1, we show that the Kalman update for the posterior mean has the form

$$\boldsymbol{\mu}_t = \mathbf{A}_t\boldsymbol{\mu}_{t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{C}_t\mathbf{A}_t\boldsymbol{\mu}_{t-1}) \tag{18.14}$$

where $\mathbf{K}_t$ is known as the Kalman gain matrix. Based on Equation 18.39, one can show that $\mathbf{K}_t = \boldsymbol{\Sigma}_t \mathbf{C}_t^T \mathbf{R}_t^{-1}$. In this context, we have $\mathbf{K}_t = \boldsymbol{\Sigma}_t \mathbf{x}_t / \sigma^2$. Hence the update for the parameters becomes

$$\hat{\boldsymbol{\theta}}_t = \hat{\boldsymbol{\theta}}_{t-1} + \frac{1}{\sigma^2} \boldsymbol{\Sigma}_{t|t} (y_t - \mathbf{x}_t^T \hat{\boldsymbol{\theta}}_{t-1}) \mathbf{x}_t \tag{18.15}$$

If we approximate $\frac{1}{\sigma^2}\boldsymbol{\Sigma}_{t|t-1}$ with $\eta_t \mathbf{I}$, we recover the **least mean squares** or **LMS** algorithm, discussed in Section 8.5.3. In LMS, we need to specify how to adapt the update parameter η_t to ensure convergence to the MLE. Furthermore, the algorithm may take multiple passes through the data. By contrast, the RLS algorithm automatically performs step-size adaptation, and converges to the optimal posterior in one pass over the data. See Figure 18.4 for an example.

18.2.4 SSM for time series forecasting *

SSMs are very well suited for time-series forecasting, as we explain below. We focus on the case of scalar (one dimensional) time series, for simplicity. Our presentation is based on (Varian 2011). See also (Aoki 1987; Harvey 1990; West and Harrison 1997; Durbin and Koopman 2001; Petris et al. 2009; Prado and West 2010) for good books on this topic.

At first sight, it might not be apparent why SSMs are useful, since the goal in forecasting is to predict future visible variables, not to estimate hidden states of some system. Indeed, most classical methods for time series forecasting are just functions of the form $\hat{y}_{t+1} = f(\mathbf{y}_{1:t}, \boldsymbol{\theta})$, where hidden variables play no role (see Section 18.2.4.4). The idea in the state-space approach to time series is to create a generative model of the data in terms of latent processes, which capture different aspects of the signal. We can then integrate out the hidden variables to compute the posterior predictive of the visibles.

Since the model is linear-Gaussian, we can just add these processes together to explain the observed data. This is called a **structural time series** model. Below we explain some of the basic building blocks.

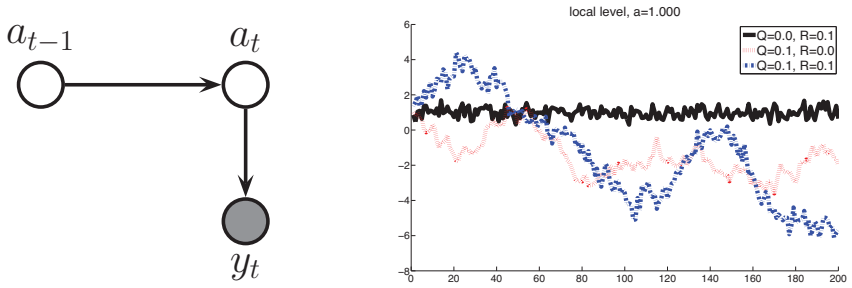
18.2.4.1 Local level model

The simplest latent process is known as the **local level model**, which has the form

$$y_t = a_t + \epsilon_t^y, \quad \epsilon_t^y \sim \mathcal{N}(0, R) \tag{18.16}$$

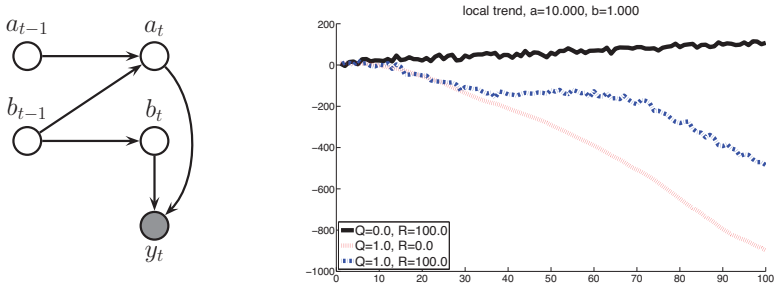
$$a_t = a_{t-1} + \epsilon_t^a, \quad \epsilon_t^a \sim \mathcal{N}(0, Q) \tag{18.17}$$

where the hidden state is just $\mathbf{z}_t = a_t$. This model asserts that the observed data $y_t \in \mathbb{R}$ is equal to some unknown level term $a_t \in \mathbb{R}$, plus observation noise with variance R. In addition, the level a_t evolves over time subject to system noise with variance Q. See Figure 18.5 for some examples.



(a) (b)

Figure 18.5 (a) Local level model. (b) Sample output, for $a_0 = 10$. Black solid line: $Q = 0$, $R = 1$ (deterministic system, noisy observations). Red dotted line: $Q = 0.1$, $R = 0$ (noisy system, deterministic observation). Blue dot-dash line: $Q = 0.1$, $R = 1$ (noisy system and observations). Figure generated by `ssmTimeSeriesSimple`.



(a) (b)

Figure 18.6 (a) Local Trend. (b) Sample output, for $a_0 = 10$, $b_0 = 1$. Color code as in Figure 18.5. Figure generated by `ssmTimeSeriesSimple`.

18.2.4.2 Local linear trend

Many time series exhibit linear trends upwards or downwards, at least locally. We can model this by letting the level a_t change by an amount b_t at each step as follows:

$$y_t = a_t + \epsilon_t^y, \quad \epsilon_t^y \sim \mathcal{N}(0, R) \tag{18.18}$$

$$a_t = a_{t-1} + b_{t-1} + \epsilon_t^a, \quad \epsilon_t^a \sim \mathcal{N}(0, Q_a) \tag{18.19}$$

$$b_t = b_{t-1} + \epsilon_t^b, \quad \epsilon_t^b \sim \mathcal{N}(0, Q_b) \tag{18.20}$$

See Figure 18.6(a). We can write this in standard form by defining $\mathbf{z}_t = (a_t, b_t)$ and

$$\mathbf{A} = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \ \mathbf{C} = \begin{pmatrix} 1 & 0 \end{pmatrix}, \ \mathbf{Q} = \begin{pmatrix} Q_a & 0 \\ 0 & Q_b \end{pmatrix} \tag{18.21}$$

When $Q_b = 0$, we have $b_t = b_0$, which is some constant defining the slope of the line. If in addition we have $Q_a = 0$, we have $a_t = a_{t-1} + b_0 t$. Unrolling this, we have $a_t = a_0 + b_0 t$, and

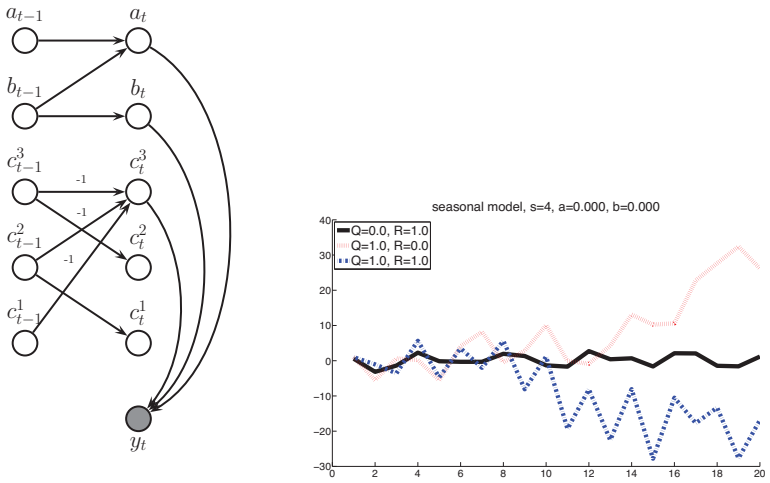


Figure 18.7 (a) Seasonal model. (b) Sample output, for $a_0 = b_0 = 0$, $\mathbf{c}_0 = (1, 1, 1)$, with a period of 4. Color code as in Figure 18.5. Figure generated by `ssmTimeSeriesSimple`.

hence $\mathbb{E}[y_t|\mathbf{y}_{1:t-1}] = a_0 + tb_0$. This is thus a generalization of the classic constant linear trend model, an example of which is shown in the black line of Figure 18.6(b).

18.2.4.3 Seasonality

Many time series fluctuate periodically, as illustrated in Figure 18.7(b). This can be modeled by adding a latent process consisting of a series offset terms, c_t, which sum to zero (on average) over a complete cycle of S steps:

$$c_t = -\sum_{s=1}^{S-1} c_{t-s} + \epsilon_t^c, \;\; \epsilon_t^c \sim \mathcal{N}(0, Q_c) \tag{18.22}$$

See Figure 18.7(a) for the graphical model for the case $S = 4$ (we only need 3 seasonal variable because of the sum-to-zero constraint). Writing this in standard LG-SSM form is left to Exercise 18.2.

18.2.4.4 ARMA models *

The classical approach to time-series forecasting is based on **ARMA** models. "ARMA" stands for auto-regressive moving-average, and refers to a model of the form

$$x_t = \sum_{i=1}^{p} \alpha_i x_{t-i} + \sum_{j=1}^{q} \beta_j w_{t-j} + v_t \tag{18.23}$$

where $v_t, w_t \sim \mathcal{N}(0, 1)$ are independent Gaussian noise terms. If $q = 0$, we have a pure AR model, where $x_t \perp x_i | x_{t-1:t-p}$, for $i < t - p$. For example, if $p = 1$, we have the AR(1) model

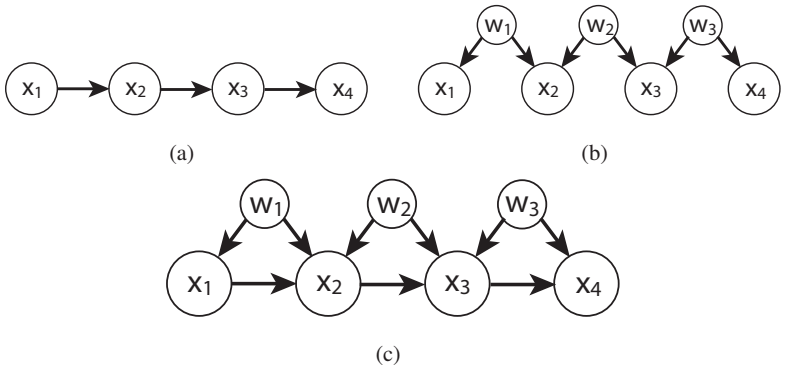


Figure 18.8 (a) An AR(1) model. (b) An MA(1) model represented as a bi-directed graph. (c) An ARMA(1,1) model. Source: Figure 5.14 of (Choi 2011). Used with kind permission of Myung Choi.

shown in Figure 18.8(a). (The v_t nodes are implicit in the Gaussian CPD for x_t.) This is just a first-order Markov chain. If $p = 0$, we have a pure MA model, where $x_t \perp x_i$, for $i < t - q$. For example, if $q = 1$, we have the MA(1) model shown in Figure 18.8(b). Here the w_t nodes are hidden common causes, which induces dependencies between adjacent time steps. This models short-range correlation. If $p = q = 1$, we get the ARMA(1,1) model shown in Figure 18.8(c), which captures correlation at short and long time scales.

It turns out that ARMA models can be represented as SSMs, as explained in (Aoki 1987; Harvey 1990; West and Harrison 1997; Durbin and Koopman 2001; Petris et al. 2009; Prado and West 2010). However, the structural approach to time series is often easier to understand than the ARMA approach. In addition, it allows the parameters to evolve over time, which makes the models more adaptive to non-stationarity.

18.3 Inference in LG-SSM

In this section, we discuss exact inference in LG-SSM models. We first consider the online case, which is analogous to the forwards algorithm for HMMs. We then consider the offline case, which is analogous to the forwards-backwards algorithm for HMMs.

18.3.1 The Kalman filtering algorithm

The **Kalman filter** is an algorithm for exact Bayesian filtering for linear-Gaussian state space models. We will represent the marginal posterior at time t by

$$p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}) = \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \tag{18.24}$$

Since everything is Gaussian, we can perform the prediction and update steps in closed form, as we explain below. The resulting algorithm is the Gaussian analog of the HMM filter in Section 17.4.2.

18.3.1.1 Prediction step

The prediction step is straightforward to derive:

$$p(\mathbf{z}_t|\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}) = \int \mathcal{N}(\mathbf{z}_t|\mathbf{A}_t\mathbf{z}_{t-1}+\mathbf{B}_t\mathbf{u}_t,\mathbf{Q}_t)\mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}_{t-1},\boldsymbol{\Sigma}_{t-1})d\mathbf{z}_{t-1} \quad (18.25)$$

$$= \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t-1},\boldsymbol{\Sigma}_{t|t-1}) \quad (18.26)$$

$$\boldsymbol{\mu}_{t|t-1} \triangleq \mathbf{A}_t\boldsymbol{\mu}_{t-1}+\mathbf{B}_t\mathbf{u}_t \quad (18.27)$$

$$\boldsymbol{\Sigma}_{t|t-1} \triangleq \mathbf{A}_t\boldsymbol{\Sigma}_{t-1}\mathbf{A}_t^T+\mathbf{Q}_t \quad (18.28)$$

18.3.1.2 Measurement step

The measurement step can be computed using Bayes rule, as follows

$$p(\mathbf{z}_t|\mathbf{y}_t,\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}) \propto p(\mathbf{y}_t|\mathbf{z}_t,\mathbf{u}_t)p(\mathbf{z}_t|\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}) \quad (18.29)$$

In Section 18.3.1.6, we show that this is given by

$$p(\mathbf{z}_t|\mathbf{y}_{1:t},\mathbf{u}_t) = \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t,\boldsymbol{\Sigma}_t) \quad (18.30)$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1}+\mathbf{K}_t\mathbf{r}_t \quad (18.31)$$

$$\boldsymbol{\Sigma}_t = (\mathbf{I}-\mathbf{K}_t\mathbf{C}_t)\boldsymbol{\Sigma}_{t|t-1} \quad (18.32)$$

where $\mathbf{r}_t$ is the **residual** or **innovation**, given by the difference between our predicted observation and the actual observation:

$$\mathbf{r}_t \triangleq \mathbf{y}_t-\hat{\mathbf{y}}_t \quad (18.33)$$

$$\hat{\mathbf{y}}_t \triangleq \mathbb{E}\left[\mathbf{y}_t|\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}\right] = \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}+\mathbf{D}_t\mathbf{u}_t \quad (18.34)$$

and $\mathbf{K}_t$ is the **Kalman gain matrix**, given by

$$\mathbf{K}_t \triangleq \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T\mathbf{S}_t^{-1} \quad (18.35)$$

where

$$\mathbf{S}_t \triangleq \mathrm{cov}\left[\mathbf{r}_t|\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}\right] \quad (18.36)$$

$$= \mathbb{E}\left[(\mathbf{C}_t\mathbf{z}_t+\boldsymbol{\delta}_t-\hat{\mathbf{y}}_t)(\mathbf{C}_t\mathbf{z}_t+\boldsymbol{\delta}_t-\hat{\mathbf{y}}_t)^T|\mathbf{y}_{1:t-1},\mathbf{u}_{1:t}\right] \quad (18.37)$$

$$= \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T+\mathbf{R}_t \quad (18.38)$$

where $\boldsymbol{\delta}_t \sim \mathcal{N}(\mathbf{0},\mathbf{R}_t)$ is an observation noise term which is independent of all other noise sources. Note that by using the matrix inversion lemma, the Kalman gain matrix can also be written as

$$\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^T(\mathbf{C}\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^T+\mathbf{R})^{-1} = (\boldsymbol{\Sigma}_{t|t-1}^{-1}+\mathbf{C}^T\mathbf{R}\mathbf{C})^{-1}\mathbf{C}^T\mathbf{R}^{-1} \quad (18.39)$$

We now have all the quantities we need to implement the algorithm; see `kalmanFilter` for some Matlab code.

Let us try to make sense of these equations. In particular, consider the equation for the mean update: $\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1}+\mathbf{K}_t\mathbf{r}_t$. This says that the new mean is the old mean plus a

correction factor, which is $\mathbf{K}_t$ times the error signal $\mathbf{r}_t$. The amount of weight placed on the error signal depends on the Kalman gain matrix. If $\mathbf{C}_t = \mathbf{I}$, then $\mathbf{K}_t = \boldsymbol{\Sigma}_{t|t-1}\mathbf{S}_t^{-1}$, which is the ratio between the covariance of the prior (from the dynamic model) and the covariance of the measurement error. If we have a strong prior and/or very noisy sensors, $|\mathbf{K}_t|$ will be small, and we will place little weight on the correction term. Conversely, if we have a weak prior and/or high precision sensors, then $|\mathbf{K}_t|$ will be large, and we will place a lot of weight on the correction term.

18.3.1.3 Marginal likelihood

As a byproduct of the algorithm, we can also compute the log-likelihood of the sequence using

$$\log p(\mathbf{y}_{1:T}|\mathbf{u}_{1:T}) = \sum_t \log p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) \tag{18.40}$$

where

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) = \mathcal{N}(\mathbf{y}_t|\mathbf{C}_t\boldsymbol{\mu}_{t|t-1}, \mathbf{S}_t) \tag{18.41}$$

18.3.1.4 Posterior predictive

The one-step-ahead posterior predictive density for the observations can be computed as follows

$$p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) = \int \mathcal{N}(\mathbf{y}_t|\mathbf{C}\mathbf{z}_t, \mathbf{R})\mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t-1}, \boldsymbol{\Sigma}_{t|t-1})d\mathbf{z}_t \tag{18.42}$$

$$= \mathcal{N}(\mathbf{y}_t|\mathbf{C}\boldsymbol{\mu}_{t|t-1}, \mathbf{C}\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^T + \mathbf{R}) \tag{18.43}$$

This is useful for time series forecasting.

18.3.1.5 Computational issues

There are two dominant costs in the Kalman filter: the matrix inversion to compute the Kalman gain matrix, $\mathbf{K}_t$, which takes $O(|\mathbf{y}_t|^3)$ time; and the matrix-matrix multiply to compute $\boldsymbol{\Sigma}_t$, which takes $O(|\mathbf{z}_t|^2)$ time. In some applications (e.g., robotic mapping), we have $|\mathbf{z}_t| \gg |\mathbf{y}_t|$, so the latter cost dominates. However, in such cases, we can sometimes use sparse approximations (see (Thrun et al. 2006)).

In cases where $|\mathbf{y}_t| \gg |\mathbf{z}_t|$, we can precompute $\mathbf{K}_t$, since, suprisingly, it does not depend on the actual observations $\mathbf{y}_{1:t}$ (an unusual property that is specific to linear Gaussian systems). The iterative equations for updating $\boldsymbol{\Sigma}_t$ are called the **Ricatti equations**, and for time invariant systems (i.e., where $\boldsymbol{\theta}_t = \boldsymbol{\theta}$), they converge to a fixed point. This steady state solution can then be used instead of using a time-specific gain matrix.

In practice, more sophisticated implementations of the Kalman filter should be used, for reasons of numerical stability. One approach is the **information filter**, which recursively updates the canonical parameters of the Gaussian, $\boldsymbol{\Lambda}_t = \boldsymbol{\Sigma}_t^{-1}$ and $\boldsymbol{\eta}_t = \boldsymbol{\Lambda}_t\boldsymbol{\mu}_t$, instead of the moment parameters. Another approach is the **square root filter**, which works with the Cholesky decomposition or the $\mathbf{U}_t\mathbf{D}_t\mathbf{U}_t$ decomposition of $\boldsymbol{\Sigma}_t$. This is much more numerically stable than directly updating $\boldsymbol{\Sigma}_t$. Further details can be found at `http://www.cs.unc.edu/~welch/kalman/` and in various books, such as (Simon 2006).

18.3.1.6 Derivation *

We now derive the Kalman filter equations. For notational simplicity, we will ignore the input terms $\mathbf{u}_{1:t}$. From Bayes rule for Gaussians (Equation 4.125), we have that the posterior precision is given by

$$\boldsymbol{\Sigma}_t^{-1} = \boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{C}_t^T \mathbf{R}_t^{-1} \mathbf{C}_t \tag{18.44}$$

From the matrix inversion lemma (Equation 4.106) we can rewrite this as

$$\boldsymbol{\Sigma}_t = \boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{R}_t + \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T)^{-1}\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1} \tag{18.45}$$

$$= (\mathbf{I} - \mathbf{K}_t\mathbf{C}_t)\boldsymbol{\Sigma}_{t|t-1} \tag{18.46}$$

From Bayes rule for Gaussians (Equation 4.125), the posterior mean is given by

$$\boldsymbol{\mu}_t = \boldsymbol{\Sigma}_t\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t + \boldsymbol{\Sigma}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} \tag{18.47}$$

We will now massage this into the form stated earlier. Applying the second matrix inversion lemma (Equation 4.107) to the first term of Equation 18.47 we have

$$\boldsymbol{\Sigma}_t\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t = (\boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{C}_t^T\mathbf{R}_t^{-1}\mathbf{C}_t)^{-1}\mathbf{C}_t\mathbf{R}_t^{-1}\mathbf{y}_t \tag{18.48}$$

$$= \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T(\mathbf{R}_t + \mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T)^{-1}\mathbf{y}_t = \mathbf{K}_t\mathbf{y}_t \tag{18.49}$$

Now applying the matrix inversion lemma (Equation 4.106) to the second term of Equation 18.47 we have

$$\boldsymbol{\Sigma}_t\boldsymbol{\Sigma}_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} \tag{18.50}$$

$$= (\boldsymbol{\Sigma}_{t|t-1}^{-1} + \mathbf{C}_t^T\mathbf{R}_t^{-1}\mathbf{C}_t)^{-1}\boldsymbol{\Sigma}_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} \tag{18.51}$$

$$= \left[\boldsymbol{\Sigma}_{t|t-1} - \boldsymbol{\Sigma}_{t|t-1}\mathbf{C}^T(\mathbf{R}_t + \mathbf{C}_t^T\boldsymbol{\Sigma}_{t|t-1}\mathbf{C}_t^T)\mathbf{C}_t\boldsymbol{\Sigma}_{t|t-1}\right]\boldsymbol{\Sigma}_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} \tag{18.52}$$

$$= (\boldsymbol{\Sigma}_{t|t-1} - \mathbf{K}_t\mathbf{C}_t^T\boldsymbol{\Sigma}_{t|t-1})\boldsymbol{\Sigma}_{t|t-1}^{-1}\boldsymbol{\mu}_{t|t-1} \tag{18.53}$$

$$= \boldsymbol{\mu}_{t|t-1} - \mathbf{K}_t\mathbf{C}_t^T\boldsymbol{\mu}_{t|t-1} \tag{18.54}$$

Putting the two together we get

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{C}_t\boldsymbol{\mu}_{t|t-1}) \tag{18.55}$$

18.3.2 The Kalman smoothing algorithm

In Section 18.3.1, we described the Kalman filter, which sequentially computes $p(\mathbf{z}_t|\mathbf{y}_{1:t})$ for each t. This is useful for online inference problems, such as tracking. However, in an offline setting, we can wait until all the data has arrived, and then compute $p(\mathbf{z}_t|\mathbf{y}_{1:T})$. By conditioning on past and future data, our uncertainty will be significantly reduced. This is illustrated in Figure 18.1(c), where we see that the posterior covariance ellipsoids are smaller for the smoothed trajectory than for the filtered trajectory. (The ellipsoids are larger at the beginning and end of the trajectory, since states near the boundary do not have as many useful neighbors from which to borrow information.)

We now explain how to compute the smoothed estimates, using an algorithm called the **RTS smoother**, named after its inventors, Rauch, Tung and Striebel (Rauch et al. 1965). It is also known as the **Kalman smoothing** algorithm. The algorithm is analogous to the forwards-backwards algorithm for HMMs, although there are some small differences which we discuss below.

18.3.2.1 Algorithm

Kalman filtering can be regarded as message passing on a graph, from left to right. When the messages have reached the end of the graph, we have successfully computed $p(\mathbf{z}_T|\mathbf{y}_{1:T})$. Now we work backwards, from right to left, sending information from the future back to the past, and them combining the two information sources. The question is: how do we compute these backwards equations? We first give the equations, then the derivation.

We have

$$
\begin{aligned}
p(\mathbf{z}_t|\mathbf{y}_{1:T}) &= \mathcal{N}(\boldsymbol{\mu}_{t|T}, \boldsymbol{\Sigma}_{t|T}) && (18.56)\\
\boldsymbol{\mu}_{t|T} &= \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) && (18.57)\\
\boldsymbol{\Sigma}_{t|T} &= \boldsymbol{\Sigma}_{t|t} + \mathbf{J}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{J}_t^T && (18.58)\\
\mathbf{J}_t &\triangleq \boldsymbol{\Sigma}_{t|t}\mathbf{A}_{t+1}^T\boldsymbol{\Sigma}_{t+1|t}^{-1} && (18.59)
\end{aligned}
$$

where $\mathbf{J}_t$ is the backwards Kalman gain matrix. The algorithm can be initialized from $\boldsymbol{\mu}_{T|T}$ and $\boldsymbol{\Sigma}_{T|T}$ from the Kalman filter. Note that this backwards pass does not need access to the data, that is, it does not need $\mathbf{y}_{1:T}$. This allows us to "throw away" potentially high dimensional observation vectors, and just keep the filtered belief states, which usually requires less memory.

18.3.2.2 Derivation *

We now derive the Kalman smoother, following the presentation of (Jordan 2007, sec 15.7).

The key idea is to leverage the Markov property, which says that $\mathbf{z}_t$ is independent of future data, $\mathbf{y}_{t+1:T}$, as long as $\mathbf{z}_{t+1}$ is known. Of course, $\mathbf{z}_{t+1}$ is not known, but we have a distribution over it. So we condition on $\mathbf{z}_{t+1}$ and then integrate it out, as follows.

$$
\begin{aligned}
p(\mathbf{z}_t|\mathbf{y}_{1:T}) &= \int p(\mathbf{z}_t|\mathbf{y}_{1:T}, \mathbf{z}_{t+1})p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T})d\mathbf{z}_{t+1} && (18.60)\\
&= \int p(\mathbf{z}_t|\mathbf{y}_{1:t}, \cancel{\mathbf{y}_{t+1:T}}, \mathbf{z}_{t+1})p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T})d\mathbf{z}_{t+1} && (18.61)
\end{aligned}
$$

By induction, assume we have already computed the smoothed distribution for $t+1$:

$$p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T}) = \mathcal{N}(\mathbf{z}_{t+1}|\boldsymbol{\mu}_{t+1|T}, \boldsymbol{\Sigma}_{t+1|T}) \quad (18.62)$$

The question is: how do we perform the integration?

First, we compute the filtered two-slice distribution $p(\mathbf{z}_t, \mathbf{z}_{t+1}|\mathbf{y}_{1:t})$ as follows:

$$p(\mathbf{z}_t, \mathbf{z}_{t+1}|\mathbf{y}_{1:t}) = \mathcal{N}\left(\begin{pmatrix}\mathbf{z}_t\\ \mathbf{z}_{t+1}\end{pmatrix} | \begin{pmatrix}\boldsymbol{\mu}_{t|t}\\ \boldsymbol{\mu}_{t+1|t}\end{pmatrix} \begin{pmatrix}\boldsymbol{\Sigma}_{t|t} & \boldsymbol{\Sigma}_{t|t}\mathbf{A}_{t+1}^T\\ \mathbf{A}_{t+1}\boldsymbol{\Sigma}_{t|t} & \boldsymbol{\Sigma}_{t+1|t}\end{pmatrix}\right) \quad (18.63)$$

Now we use Gaussian conditioning to compute $p(\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:t})$ as follows:

$$p(\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:t}) = \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\mathbf{z}_{t+1} - \boldsymbol{\mu}_{t+1|t}), \boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^T) \tag{18.64}$$

We can compute the smoothed distribution for t using the rules of iterated expectation and iterated covariance. First, the mean:

$$\begin{aligned}
\boldsymbol{\mu}_{t|T} &= \mathbb{E}\left[\mathbb{E}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:T}\right]\middle|\mathbf{y}_{1:T}\right] && (18.65)\\
&= \mathbb{E}\left[\mathbb{E}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:t}\right]\middle|\mathbf{y}_{1:T}\right] && (18.66)\\
&= \mathbb{E}\left[\boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\mathbf{z}_{t+1} - \boldsymbol{\mu}_{t+1|t})\middle|\mathbf{y}_{1:T}\right] && (18.67)\\
&= \boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\boldsymbol{\mu}_{t+1|T} - \boldsymbol{\mu}_{t+1|t}) && (18.68)
\end{aligned}$$

Now the covariance:

$$\begin{aligned}
\boldsymbol{\Sigma}_{t|T} &= \mathrm{cov}\left[\mathbb{E}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:T}\right]\middle|\mathbf{y}_{1:T}\right] + \mathbb{E}\left[\mathrm{cov}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:T}\right]\middle|\mathbf{y}_{1:T}\right] && (18.69)\\
&= \mathrm{cov}\left[\mathbb{E}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:t}\right]\middle|\mathbf{y}_{1:T}\right] + \mathbb{E}\left[\mathrm{cov}\left[\mathbf{z}_t|\mathbf{z}_{t+1}, \mathbf{y}_{1:t}\right]\middle|\mathbf{y}_{1:T}\right] && (18.70)\\
&= \mathrm{cov}\left[\boldsymbol{\mu}_{t|t} + \mathbf{J}_t(\mathbf{z}_{t+1} - \boldsymbol{\mu}_{t+1|t})|\mathbf{y}_{1:T}\right] + \mathbb{E}\left[\boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^T|\mathbf{y}_{1:T}\right] && (18.71)\\
&= \mathbf{J}_t\mathrm{cov}\left[\mathbf{z}_{t+1} - \boldsymbol{\mu}_{t+1|t}|\mathbf{y}_{1:T}\right]\mathbf{J}_t^T + \boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^T && (18.72)\\
&= \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|T}\mathbf{J}_t^T + \boldsymbol{\Sigma}_{t|t} - \mathbf{J}_t\boldsymbol{\Sigma}_{t+1|t}\mathbf{J}_t^T && (18.73)\\
&= \boldsymbol{\Sigma}_{t|t} + \mathbf{J}_t(\boldsymbol{\Sigma}_{t+1|T} - \boldsymbol{\Sigma}_{t+1|t})\mathbf{J}_t^T && (18.74)
\end{aligned}$$

The algorithm can be initialized from $\boldsymbol{\mu}_{T|T}$ and $\boldsymbol{\Sigma}_{T|T}$ from the last step of the filtering algorithm.

18.3.2.3 Comparison to the forwards-backwards algorithm for HMMs *

Note that in both the forwards and backwards passes for LDS, we always worked with normalized distributions, either conditioned on the past data or conditioned on all the data. Furthermore, the backwards pass depends on the results of the forwards pass. This is different from the usual presentation of forwards-backwards for HMMs, where the backwards pass can be computed independently of the forwards pass (see Section 17.4.3).

It turns out that we can rewrite the Kalman smoother in a modified form which makes it more similar to forwards-backwards for HMMs. In particular, we have

$$\begin{aligned}
p(\mathbf{z}_t|\mathbf{y}_{1:T}) &= \int p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{z}_{t+1})p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T})d\mathbf{z}_{t+1} && (18.75)\\
&= \int p(\mathbf{z}_t, \mathbf{z}_{t+1}|\mathbf{y}_{1:t})\frac{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:t})}d\mathbf{z}_{t+1} && (18.76)
\end{aligned}$$

Now

$$p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T}) = \frac{p(\mathbf{y}_{t+1:T}|\mathbf{z}_{t+1}, \cancel{\mathbf{y}_{1:t}})p(\mathbf{z}_{t+1}|\mathbf{y}_{1:t})}{p(\mathbf{y}_{t+1:T}|\mathbf{y}_{1:t})} \tag{18.77}$$

so

$$\frac{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:T})}{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:t})} = \frac{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:t})p(\mathbf{y}_{t+1:T}|\mathbf{z}_{t+1})}{p(\mathbf{z}_{t+1}|\mathbf{y}_{1:t})p(\mathbf{y}_{t+1:T}|\mathbf{y}_{1:t})} \propto p(\mathbf{y}_{t+1:T}|\mathbf{z}_{t+1}) \tag{18.78}$$

which is the conditional likelihood of the future data. This backwards message can be computed independently of the forwards message. However, this approach has several disadvantages: (1) it needs access to the original observation sequence; (2) the backwards message is a likelihood, not a posterior, so it need not to integrate to 1 over $\mathbf{z}_t$ – in fact, it may not always be possible to represent $p(\mathbf{y}_{t+1:T}|\mathbf{z}_{t+1})$ as a Gaussian with positive definite covariance (this problem does not arise in discrete state-spaces, as used in HMMs); (3) when exact inference is not possible, it makes more sense to try to approximate the smoothed distribution rather than the backwards likelihood term (see Section 22.5).

There is yet another variant, known as **two-filter smoothing**, whereby we compute $p(\mathbf{z}_t|\mathbf{y}_{1:t})$ in the forwards pass as usual, and the filtered posterior $p(\mathbf{z}_t|\mathbf{y}_{t+1:T})$ in the backwards pass. These can then be easily combined to compute $p(\mathbf{z}_t|\mathbf{y}_{1:T})$. See (Kitagawa 2004; Briers et al. 2010) for details.

18.4 Learning for LG-SSM

In this section, we briefly discuss how to estimate the parameters of an LG-SSM. In the control theory community, this is known as **systems identification** (Ljung 1987).

When using SSMs for time series forecasting, and also in some physical state estimation problems, the observation matrix $\mathbf{C}$ and the transition matrix $\mathbf{A}$ are both known and fixed, by definition of the model. In such cases, all that needs to be learned are the noise covariances $\mathbf{Q}$ and $\mathbf{R}$. (The initial state estimate $\boldsymbol{\mu}_0$ is often less important, since it will get "washed away" by the data after a few time steps. This can be encouraged by setting the initial state covariance to be large, representing a weak prior.) Although we can estimate $\mathbf{Q}$ and $\mathbf{R}$ offline, using the methods described below, it is also possible to derive a recursive procedure to exactly compute the posterior $p(\mathbf{z}_t, \mathbf{R}, \mathbf{Q}|\mathbf{y}_{1:t})$, which has the form of a Normal-inverse-Wishart; see (West and Harrison 1997; Prado and West 2010) for details.

18.4.1 Identifiability and numerical stability

In the more general setting, where the hidden states have no pre-specified meaning, we need to learn $\mathbf{A}$ and $\mathbf{C}$. However, in this case we can set $\mathbf{Q} = \mathbf{I}$ without loss of generality, since an arbitrary noise covariance can be modeled by appropriately modifying $\mathbf{A}$. Also, by analogy with factor analysis, we can require $\mathbf{R}$ to be diagonal without loss of generality. Doing this reduces the number of free parameters and improves numerical stability.

Another constraint that is useful to impose is on the eigenvalues of the dynamics matrix $\mathbf{A}$. To see why this is important, consider the case of no system noise. In this case, the hidden state at time t is given by

$$\mathbf{z}_t = \mathbf{A}^t \mathbf{z}_1 = \mathbf{U}\boldsymbol{\Lambda}^t \mathbf{U}^{-1} \mathbf{z}_1 \tag{18.79}$$

where $\mathbf{U}$ is the matrix of eigenvectors for $\mathbf{A}$, and $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_i)$ contains the eigenvalues. If any $\lambda_i > 1$, then for large t, $\mathbf{z}_t$ will blow up in magnitude. Consequently, to ensure stability, it is useful to require that all the eigenvalues are less than 1 (Siddiqi et al. 2007). Of course, if all the eigenvalues are less than 1, then $\mathbb{E}[\mathbf{z}_t] = \mathbf{0}$ for large t, so the state will return to the origin. Fortunately, when we add noise, the state become non-zero, so the model does not degenerate.

Below we discuss how to estimate the parameters. However, for simplicity of presentation, we do not impose any of the constraints mentioned above.

18.4.2 Training with fully observed data

If we observe the hidden state sequences, we can fit the model by computing the MLEs (or even the full posteriors) for the parameters by solving a multivariate linear regression problem for $\mathbf{z}_{t-1} \to \mathbf{z}_t$ and for $\mathbf{z}_t \to \mathbf{y}_t$. That is, we can estimate $\mathbf{A}$ by solving the least squares problem $J(\mathbf{A}) = \sum_{t=1}^{2}(\mathbf{z}_t - \mathbf{A}\mathbf{z}_{t-1})^2$, and similarly for $\mathbf{C}$. We can estimate the system noise covariance $\mathbf{Q}$ from the residuals in predicting $\mathbf{z}_t$ from $\mathbf{z}_{t-1}$, and estimate the observation noise covariance $\mathbf{R}$ from the residuals in predicting $\mathbf{y}_t$ from $\mathbf{z}_t$.

18.4.3 EM for LG-SSM

If we only observe the output sequence, we can compute ML or MAP estimates of the parameters using EM. The method is conceptually quite similar to the Baum-Welch algorithm for HMMs (Section 17.5), except we use Kalman smoothing instead of forwards-backwards in the E step, and use different calculations in the M step. We leave the details to Exercise 18.1.

18.4.4 Subspace methods

EM does not always give satisfactory results, because it is sensitive to the initial parameter estimates. One way to avoid this is to use a different approach known as a **subspace method** (Overschee and Moor 1996; Katayama 2005).

To understand this approach, let us initially assume there is no observation noise and no system noise. In this case, we have $\mathbf{z}_t = \mathbf{A}\mathbf{z}_{t-1}$ and $\mathbf{y}_t = \mathbf{C}\mathbf{z}_t$, and hence $\mathbf{y}_t = \mathbf{C}\mathbf{A}^{t-1}\mathbf{z}_1$. Consequently all the observations must be generated from a $\dim(\mathbf{z}_t)$-dimensional linear manifold or subspace. We can identify this subspace using PCA (see the above references for details). Once we have an estimate of the $\mathbf{z}_t$'s, we can fit the model as if it were fully observed. We can either use these estimates in their own right, or use them to initialize EM.

18.4.5 Bayesian methods for "fitting" LG-SSMs

There are various offline Bayesian alternatives to the EM algorithm, including variational Bayes EM (Beal 2003; Barber and Chiappa 2007) and blocked Gibbs sampling (Carter and Kohn 1994; Cappe et al. 2005; Fruhwirth-Schnatter 2007). The Bayesian approach can also be used to perform online learning, as we discussed in Section 18.2.3. Unfortunately, once we add the SSM parameters to the state space, the model is generally no longer linear Gaussian. Consequently we must use some of the approximate online inference methods to be discussed below.

18.5 Approximate online inference for non-linear, non-Gaussian SSMs

In Section 18.3.1, we discussed how to perform exact online inference for LG-SSMs. However, many models are non linear. For example, most moving objects do not move in straight lines. And even if they did, if we assume the parameters of the model are unknown and add them

to the state space, the model becomes nonlinear. Furthermore, non-Gaussian noise is also very common, e.g., due to outliers, or when inferring parameters for GLMs instead of just linear regression. For these more general models, we need to use approximate inference.

The approximate inference algorithms we discuss below approximate the posterior by a Gaussian. In general, if $Y = f(X)$, where X has a Gaussian distribution and f is a non-linear function, there are two main ways to approximate $p(Y)$ by a Gaussian. The first is to use a first-order approximation of f. The second is to use the exact f, but to project $f(X)$ onto the space of Gaussians by moment matching. We discuss each of these methods in turn. (See also Section 23.5, where we discuss particle filtering, which is a stochastic algorithm for approximate online inference, which uses a non-parametric approximation to the posterior, which is often more accurate but slower to compute.)

18.5.1 Extended Kalman filter (EKF)

In this section, we focus on non-linear models, but we assume the noise is Gaussian. That is, we consider models of the form

$$\mathbf{z}_t = g(\mathbf{u}_t, \mathbf{z}_{t-1}) + \mathcal{N}(\mathbf{0}, \mathbf{Q}_t) \tag{18.80}$$

$$\mathbf{y}_t = h(\mathbf{z}_t) + \mathcal{N}(\mathbf{0}, \mathbf{R}_t) \tag{18.81}$$

where the transition model g and the observation model h are nonlinear but differentiable functions. Furthermore, we focus on the case where we approximate the posterior by a single Gaussian. (The simplest way to handle more general posteriors (e.g., multi-modal, discrete, etc). is to use particle filtering, which we discuss in Section 23.5.)

The **extended Kalman filter** or **EKF** can be applied to nonlinear Gaussian dynamical systems of this form. The basic idea is to linearize g and h about the previous state estimate using a first order Taylor series expansion, and then to apply the standard Kalman filter equations. (The noise variance in the equations ($\mathbf{Q}$ and $\mathbf{R}$) is not changed, i.e., the additional error due to linearization is not modeled.) Thus we approximate the stationary non-linear dynamical system with a non-stationary linear dynamical system.

The intuition behind the approach is shown in Figure 18.9, which shows what happens when we pass a Gaussian distribution $p(x)$, shown on the bottom right, through a nonlinear function $y = g(x)$, shown on the top right. The resulting distribution (approximated by Monte Carlo) is shown in the shaded gray area in the top left corner. The best Gaussian approximation to this, computed from $\mathbb{E}[g(x)]$ and $\mathrm{var}[g(x)]$ by Monte Carlo, is shown by the solid black line. The EKF approximates this Gaussian as follows: it linearizes the g function at the current mode, μ, and then passes the Gaussian distribution $p(x)$ through this linearized function. In this example, the result is quite a good approximation to the first and second moments of $p(y)$, for much less cost than an MC approximation.

In more detail, the method works as follows. We approximate the measurement model using

$$p(\mathbf{y}_t|\mathbf{z}_t) \approx \mathcal{N}(\mathbf{y}_t|\mathbf{h}(\boldsymbol{\mu}_{t|t-1}) + \mathbf{H}_t(\mathbf{y}_t - \boldsymbol{\mu}_{t|t-1}), \mathbf{R}_t) \tag{18.82}$$

where $\mathbf{H}_t$ is the Jacobian matrix of $\mathbf{h}$ evaluated at the prior mode:

$$H_{ij} \triangleq \frac{\partial h_i(\mathbf{z})}{\partial z_j} \tag{18.83}$$

$$\mathbf{H}_t \triangleq \mathbf{H}|_{\mathbf{z}=\boldsymbol{\mu}_{t|t-1}} \tag{18.84}$$

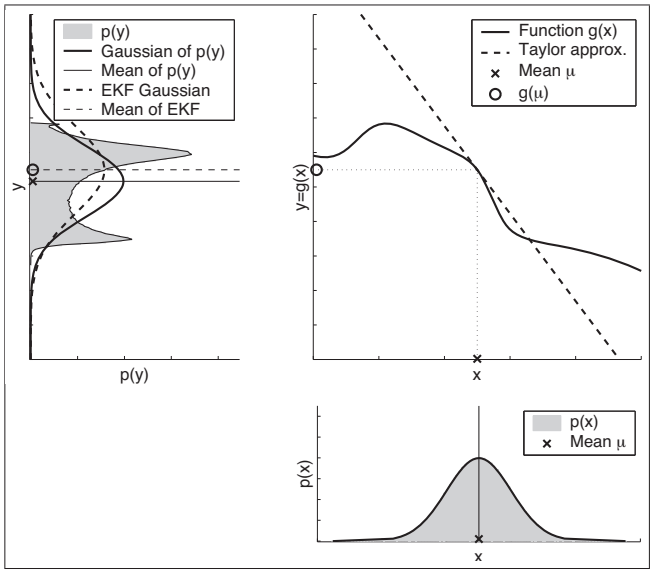


Figure 18.9 Nonlinear transformation of a Gaussian random variable. The prior $p(x)$ is shown on the bottom right. The function $y = g(x)$ is shown on the top right. The transformed distribution $p(y)$ is shown in the top left. A linear function induces a Gaussian distribution, but a non-linear function induces a complex distribution. The solid line is the best Gaussian approximation to this; the dotted line is the EKF approximation to this. Source: Figure 3.4 of (Thrun et al. 2006). Used with kind permission of Sebastian Thrun.

Similarly, we approximate the system model using

$$p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{u}_t) \approx \mathcal{N}(\mathbf{z}_t|\mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) + \mathbf{G}_t(\mathbf{z}_{t-1} - \boldsymbol{\mu}_{t-1}), \mathbf{Q}_t) \tag{18.85}$$

where

$$G_{ij}(\mathbf{u}) \triangleq \frac{\partial g_i(\mathbf{u}, \mathbf{z})}{\partial z_j} \tag{18.86}$$

$$\mathbf{G}_t \triangleq \mathbf{G}(\mathbf{u}_t)|_{\mathbf{z}=\boldsymbol{\mu}_{t-1}} \tag{18.87}$$

so $\mathbf{G}$ is the Jacobian matrix of $\mathbf{g}$ evaluated at the prior mode.

Given this, we can then apply the Kalman filter to compute the posterior as follows:

$$\boldsymbol{\mu}_{t|t-1} = \mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1}) \tag{18.88}$$

$$\mathbf{V}_{t|t-1} = \mathbf{G}_t\mathbf{V}_{t-1}\mathbf{G}_t^T + \mathbf{Q}_t \tag{18.89}$$

$$\mathbf{K}_t = \mathbf{V}_{t|t-1}\mathbf{H}_t^T(\mathbf{H}_t\mathbf{V}_{t|t-1}\mathbf{H}_t^T + \mathbf{R}_t)^{-1} \tag{18.90}$$

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_{t|t-1} + \mathbf{K}_t(\mathbf{y}_t - \mathbf{h}(\boldsymbol{\mu}_{t|t-1})) \tag{18.91}$$

$$\mathbf{V}_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H}_t)\mathbf{V}_{t|t-1} \tag{18.92}$$

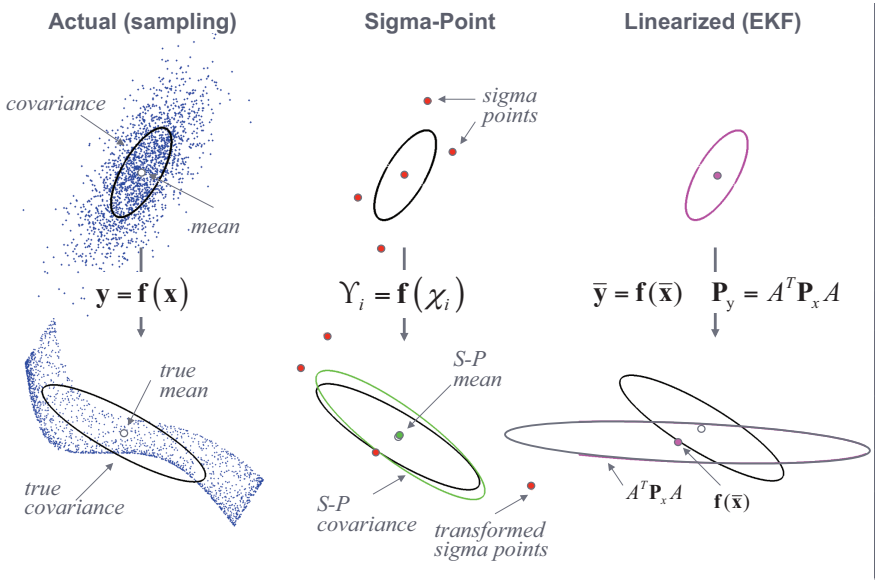


Figure 18.10 An example of the unscented transform in two dimensions. Source: (Wan and der Merwe 2001). Used with kind permission of Eric Wan.

We see that the only difference from the regular Kalman filter is that, when we compute the state prediction, we use $\mathbf{g}(\mathbf{u}_t, \boldsymbol{\mu}_{t-1})$ instead of $\mathbf{A}_t \boldsymbol{\mu}_{t-1} + \mathbf{B}_t \mathbf{u}_t$, and when we compute the measurement update we use $\mathbf{h}(\boldsymbol{\mu}_{t|t-1})$ instead of $\mathbf{C}_t \boldsymbol{\mu}_{t|t-1}$.

It is possible to improve performance by repeatedly re-linearizing the equations around $\boldsymbol{\mu}_t$ instead of $\boldsymbol{\mu}_{t|t-1}$; this is called the **iterated EKF**, and yields better results, although it is of course slower.

There are two cases when the EKF works poorly. The first is when the prior covariance is large. In this case, the prior distribution is broad, so we end up sending a lot of probability mass through different parts of the function that are far from the mean, where the function has been linearized. The other setting where the EKF works poorly is when the function is highly nonlinear near the current mean. In Section 18.5.2, we will discuss an algorithm called the UKF which works better than the EKF in both of these settings.

18.5.2 Unscented Kalman filter (UKF)

The **unscented Kalman filter** (**UKF**) is a better version of the EKF (Julier and Uhlmann 1997). (Apparently it is so-called because it "doesn't stink"!) The key intuition is this: it is easier to approximate a Gaussian than to approximate a function. So instead of performing a linear approximation to the function, and passing a Gaussian through it, instead pass a deterministically chosen set of points, known as **sigma points**, through the function, and fit a Gaussian to the resulting transformed points. This is known as the **unscented transform**, and is sketched in Figure 18.10. (We explain this figure in detail below.)

The UKF basically uses the unscented transform twice, once to approximate passing through the system model $\mathbf{g}$, and once to approximate passing through the measurement model $\mathbf{h}$. We give the details below. Note that the UKF and EKF both perform $O(d^3)$ operations per time step where d is the size of the latent state-space. However, the UKF is accurate to at least second order, whereas the EKF is only a first order approximation (although both the EKF and UKF can be extended to capture higher order terms). Furthermore, the unscented transform does not require the analytic evaluation of any derivatives or Jacobians (a so-called **derivative free filter**), making it simpler to implement and more widely applicable.

18.5.2.1 The unscented transform

Before explaining the UKF, we first explain the unscented transform. Assume $p(\mathbf{x}) = \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$, and consider estimating $p(\mathbf{y})$, where $\mathbf{y} = \mathbf{f}(\mathbf{x})$ for some nonlinear function $\mathbf{f}$. The unscented transform does this as follows. First we create a set of $2d + 1$ sigma points $\mathbf{x}_i$, given by

$$\mathbf{x} = \left(\boldsymbol{\mu}, \{\boldsymbol{\mu} + (\sqrt{(d+\lambda)\boldsymbol{\Sigma}})_{:i}\}_{i=1}^{d}, \{\boldsymbol{\mu} - (\sqrt{(d+\lambda)\boldsymbol{\Sigma}})_{:i}\}_{i=1}^{d}\right) \tag{18.93}$$

where $\lambda = \alpha^2(d + \kappa) - d$ is a scaling parameter to be specified below, and the notation $\mathbf{M}_{:i}$ means the i'th column of matrix $\mathbf{M}$.

These sigma points are propagated through the nonlinear function to yield $\mathbf{y}_i = f(\mathbf{x}_i)$, and the mean and covariance for $\mathbf{y}$ is computed as follows:

$$\boldsymbol{\mu}_y = \sum_{i=0}^{2d} w_m^i \mathbf{y}_i \tag{18.94}$$

$$\boldsymbol{\Sigma}_y = \sum_{i=0}^{2d} w_c^i (\mathbf{y}_i - \boldsymbol{\mu}_y)(\mathbf{y}_i - \boldsymbol{\mu}_y)^T \tag{18.95}$$

where the w's are weighting terms, given by

$$w_m^i = \frac{\lambda}{d+\lambda} \tag{18.96}$$

$$w_c^i = \frac{\lambda}{d+\lambda} + (1 - \alpha^2 + \beta) \tag{18.97}$$

$$w_m^i = w_c^i = \frac{1}{2(d+\lambda)} \tag{18.98}$$

See Figure 18.10 for an illustration.

In general, the optimal values of α, β and κ are problem dependent, but when $d = 1$, they are $\alpha = 1$, $\beta = 0$, $\kappa = 2$. Thus in the 1d case, $\lambda = 2$, so the 3 sigma points are μ, $\mu + \sqrt{3}\sigma$ and $\mu - \sqrt{3}\sigma$.

18.5.2.2 The UKF algorithm

The UKF algorithm is simply two applications of the unscented tranform, one to compute $p(\mathbf{z}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t})$ and the other to compute $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t})$. We give the details below.

The first step is to approximate the predictive density $p(\mathbf{z}_t|\mathbf{y}_{1:t-1}, \mathbf{u}_{1:t}) \approx \mathcal{N}(\mathbf{z}_t|\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t)$ by passing the old belief state $\mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}_{t-1}, \boldsymbol{\Sigma}_{t-1})$ through the system model $\mathbf{g}$ as follows:

$$\mathbf{z}_{t-1}^0 = \left(\boldsymbol{\mu}_{t-1}, \{\boldsymbol{\mu}_{t-1} + \gamma(\sqrt{\boldsymbol{\Sigma}_{t-1}})_{:i}\}_{i=1}^d, \{\boldsymbol{\mu}_{t-1} - \gamma(\sqrt{\boldsymbol{\Sigma}_{t-1}})_{:i}\}_{i=1}^d\right) \tag{18.99}$$

$$\overline{\mathbf{z}}_t^{*i} = \mathbf{g}(\mathbf{u}_t, \mathbf{z}_{t-1}^{0i}) \tag{18.100}$$

$$\overline{\boldsymbol{\mu}}_t = \sum_{i=0}^{2d} w_m^i \overline{\mathbf{z}}_t^{*i} \tag{18.101}$$

$$\overline{\boldsymbol{\Sigma}}_t = \sum_{i=0}^{2d} w_c^i (\overline{\mathbf{z}}_t^{*i} - \overline{\boldsymbol{\mu}}_t)(\overline{\mathbf{z}}_t^{*i} - \overline{\boldsymbol{\mu}}_t) + \mathbf{Q}_t \tag{18.102}$$

where $\gamma = \sqrt{d + \lambda}$.

The second step is to approximate the likelihood $p(\mathbf{y}_t|\mathbf{z}_t) \approx \mathcal{N}(\mathbf{y}_t|\hat{\mathbf{y}}_t, \mathbf{S}_t)$ by passing the prior $\mathcal{N}(\mathbf{z}_t|\overline{\boldsymbol{\mu}}_t, \overline{\boldsymbol{\Sigma}}_t)$ through the observation model $\mathbf{h}$:

$$\overline{\mathbf{z}}_t^0 = \left(\overline{\boldsymbol{\mu}}_t, \{\overline{\boldsymbol{\mu}}_t + \gamma(\sqrt{\overline{\boldsymbol{\Sigma}}_t})_{:i}\}_{i=1}^d, \{\overline{\boldsymbol{\mu}}_t - \gamma(\sqrt{\overline{\boldsymbol{\Sigma}}_t})_{:i}\}_{i=1}^d\right) \tag{18.103}$$

$$\overline{\mathbf{y}}_t^{*i} = \mathbf{h}(\overline{\mathbf{z}}_t^{0i}) \tag{18.104}$$

$$\hat{\mathbf{y}}_t = \sum_{i=0}^{2d} w_m^i \overline{\mathbf{y}}_t^{*i} \tag{18.105}$$

$$\mathbf{S}_t = \sum_{i=0}^{2d} w_c^i (\overline{\mathbf{y}}_t^{*i} - \hat{\mathbf{y}}_t)(\overline{\mathbf{y}}_t^{*i} - \hat{\mathbf{y}}_t)^T + \mathbf{R}_t \tag{18.106}$$

Finally, we use Bayes rule for Gaussians to get the posterior $p(\mathbf{z}_t|\mathbf{y}_{1:t}, \mathbf{u}_{1:t}) \approx \mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$:

$$\overline{\boldsymbol{\Sigma}}_t^{z,y} = \sum_{i=0}^{2d} w_c^i (\overline{\mathbf{z}}_t^{*i} - \overline{\boldsymbol{\mu}}_t)(\overline{\mathbf{y}}_t^{*i} - \hat{\mathbf{y}}_t)^T \tag{18.107}$$

$$\mathbf{K}_t = \overline{\boldsymbol{\Sigma}}_t^{z,y} \mathbf{S}_t^{-1} \tag{18.108}$$

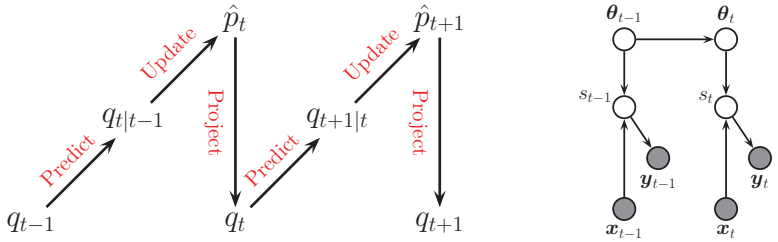
$$\boldsymbol{\mu}_t = \overline{\boldsymbol{\mu}}_t + \mathbf{K}_t(\mathbf{y}_t - \hat{\mathbf{y}}_t) \tag{18.109}$$

$$\boldsymbol{\Sigma}_t = \overline{\boldsymbol{\Sigma}}_t - \mathbf{K}_t \mathbf{S}_t \mathbf{K}_t^T \tag{18.110}$$

18.5.3 Assumed density filtering (ADF)

In this section, we discuss inference where we perform an exact update step, but then approximate the posterior by a distribution of a certain convenient form, such as a Gaussian. More precisely, let the unknowns that we want to infer be denoted by $\boldsymbol{\theta}_t$. Suppose that $\mathcal{Q}$ is a set of tractable distributions, e.g., Gaussians with a diagonal covariance matrix, or a product of discrete distributions. Suppose that we have an approximate prior $q_{t-1}(\boldsymbol{\theta}_{t-1}) \approx p(\boldsymbol{\theta}_{t-1}|\mathbf{y}_{1:t-1})$, where $q_{t-1} \in \mathcal{Q}$. We can update this with the new measurement to get the approximate posterior

$$\hat{p}(\boldsymbol{\theta}_t) = \frac{1}{Z_t} p(\mathbf{y}_t|\boldsymbol{\theta}_t) q_{t|t-1}(\boldsymbol{\theta}_t) \tag{18.111}$$



(a) (b)

Figure 18.11 (a) Illustration of the predict-update-project cycle of assumed density filtering. (b) A dynamical logistic regression model. Compare to Figure 18.4(a).

where

$$Z_t = \int p(\mathbf{y}_t|\boldsymbol{\theta}_t) q_{t|t-1}(\boldsymbol{\theta}_t) d\boldsymbol{\theta}_t \tag{18.112}$$

is the normalization constant and

$$q_{t|t-1}(\boldsymbol{\theta}_t) = \int p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) q_{t-1}(\boldsymbol{\theta}_{t-1}) d\boldsymbol{\theta}_{t-1} \tag{18.113}$$

is the one step ahead predictive distribution. If the prior is from a suitably restricted family, this one-step update process is usually tractable. However, we often find that the resulting posterior is no longer in our tractable family, $\hat{p}(\boldsymbol{\theta}_t) \notin \mathcal{Q}$. So after updating we seek the best tractable approximation by computing

$$q(\boldsymbol{\theta}_t) = \underset{q \in \mathcal{Q}}{\operatorname{argmin}} \, \mathbb{KL}\left(\hat{p}(\boldsymbol{\theta}_t) || q(\boldsymbol{\theta}_t)\right) \tag{18.114}$$

This minimizes the the Kullback-Leibler divergence (Section 2.8.2) from the approximation $q(\boldsymbol{\theta}_t)$ to the "exact" posterior $\hat{p}(\boldsymbol{\theta}_t)$, and can be thought of as projecting $\hat{p}$ onto the space of tractable distributions. The whole algorithm consists of **predict-update-project** cycles. This is known as **assumed density filtering** or **ADF** (Maybeck 1979). See Figure 18.11(a) for a sketch.

If q is in the exponential family, one can show that this KL minimization can be done by **moment matching**. We give some examples of this below.

18.5.3.1 Boyen-Koller algorithm for online inference in DBNs

If we are performing inference in a discrete-state dynamic Bayes net (Section 17.6.7), where θ_{tj} is the j'th hidden variable at time t, then the exact posterior $p(\boldsymbol{\theta}_t)$ becomes intractable to compute because of the entanglement problem. Suppose we use a fully factored approximation of the form $q(\boldsymbol{\theta}_t) = \prod_{j=1}^{D} \mathrm{Cat}(\theta_{t,j}|\boldsymbol{\pi}_{t,j})$, where $\pi_{tjk} = q(\theta_{t,j} = k)$ is the probability variable j is in state k, and D is the number of variables. In this case, the moment matching operation becomes

$$\pi_{tjk} = \hat{p}(\theta_{t,j} = k) \tag{18.115}$$

This can be computed by performing a predict-update step using the factored prior, and then computing the posterior marginals. This is known as the **Boyen-Koller** algorithm, named after the authors of (Boyen and Koller 1998), who demonstrated that the error incurred by this series of repeated approximations remains bounded (under certain assumptions about the stochasticity of the system).

18.5.3.2 Gaussian approximation for online inference in GLMs

Now suppose $q(\boldsymbol{\theta}_t) = \prod_{j=1}^{D} \mathcal{N}(\theta_{t,j}|\mu_{t,j}, \tau_{t,j})$, where $\tau_{t,j}$ is the variance. Then the optimal parameters of the tractable approximation to the posterior are

$$\mu_{t,j} = \mathbb{E}_{\hat{p}}\left[\theta_{t,j}\right], \ \tau_{t,j} = \mathrm{var}_{\hat{p}}\left[\theta_{t,j}\right] \tag{18.116}$$

This method can be used to do online inference for the parameters of many statistical models. For example, theTrueSkill system, used in Microsoft's Xbox to rank players over time, uses this form of approximation (Herbrich et al. 2007). We can also apply this method to simpler models, such as GLM, which have the advantage that the posterior is log-concave. Below we explain how to do this for binary logistic regression, following the presentation of (Zoeter 2007).

The model has the form

$$p(y_t|\mathbf{x}_t, \boldsymbol{\theta}_t) = \mathrm{Ber}(y_t|\mathrm{sigm}(\mathbf{x}_t^T\boldsymbol{\theta}_t)) \tag{18.117}$$

$$p(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}) = \mathcal{N}(\boldsymbol{\theta}_t|\boldsymbol{\theta}_{t-1}, \sigma^2\mathbf{I}) \tag{18.118}$$

where σ^2 is some process noise which allows the parameters to change slowly over time. (This can be set to 0, as in the recursive least squares method (Section 18.2.3), if desired.) We will assume $q_{t-1}(\boldsymbol{\theta}_{t-1}) = \prod_j \mathcal{N}(\theta_{t-1,j}|\mu_{t-1,j}, \tau_{t-1,j})$ is the tractable prior. We can compute the one-step-ahead predictive density $q_{t|t-1}(\boldsymbol{\theta}_t)$ using the standard linear-Gaussian update. So now we concentrate on the measurement update step.

Define the deterministic quantity $s_t = \boldsymbol{\theta}_t^T\mathbf{x}_t$, as shown in Figure 18.11(b). If $q_{t|t-1}(\boldsymbol{\theta}_t) = \prod_j \mathcal{N}(\theta_{t,j}|\mu_{t|t-1,j}, \tau_{t|t-1,j})$, then we can compute the predictive distribution for s_t as follows:

$$q_{t|t-1}(s_t) = \mathcal{N}(s_t|m_{t|t-1}, v_{t|t-1}) \tag{18.119}$$

$$m_{t|t-1} = \sum_j x_{t,j}\mu_{t|t-1,j} \tag{18.120}$$

$$v_{t|t-1} = \sum_j x_{t,j}^2\tau_{t|t-1,j} \tag{18.121}$$

The posterior for s_t is given by

$$q_t(s_t) = \mathcal{N}(s_t|m_t, v_t) \tag{18.122}$$

$$m_t = \int s_t \frac{1}{Z_t} p(y_t|s_t) q_{t|t-1}(s_t) ds_t \tag{18.123}$$

$$v_t = \int s_t^2 \frac{1}{Z_t} p(y_t|s_t) q_{t|t-1}(s_t) ds_t - m_t^2 \tag{18.124}$$

$$Z_t = \int p(y_t|s_t) q_{t|t-1}(s_t) ds_t \tag{18.125}$$

where $p(y_t|s_t) = \text{Ber}(y_t|s_t)$. These integrals are one dimensional, and so can be computed using Gaussian quadrature (see (Zoeter 2007) for details). This is the same as one step of the UKF algorithm.

Having inferred $q(s_t)$, we need to compute $q(\boldsymbol{\theta}|s_t)$. This can be done as follows. Define δ_m as the change in the mean of s_t and δ_v as the change in the variance:

$$m_t = m_{t|t-1} + \delta_m, \; v_t = v_{t|t-1} + \delta_v \tag{18.126}$$

Then one can show that the new factored posterior over the model parameters is given by

$$q(\theta_{t,j}) = \mathcal{N}(\theta_{t,j}|\mu_{t,j}, \tau_{t,j}) \tag{18.127}$$

$$\mu_{t,j} = \mu_{t|t-1,j} + a_j \delta_m \tag{18.128}$$

$$\tau_{t,j} = \tau_{t|t-1,j} + a_j^2 \delta_v \tag{18.129}$$

$$a_j \triangleq \frac{x_{t,j}\tau_{t|t-1,j}}{\sum_{j'} x_{t,j'}^2 \tau_{t|t-1,j}^2} \tag{18.130}$$

Thus we see that the parameters which correspond to inputs with larger magnitude (big $|x_{t,j}|$) or larger uncertainty (big $\tau_{t|t-1,j}$) get updated most, which makes intuitive sense.

In (Opper 1998) a version of this algorithm is derived using a probit likelihood (see Section 9.4). In this case, the measurement update can be done in closed form, without the need for numerical integration. In either case, the algorithm only takes $O(D)$ operations per time step, so it can be applied to models with large numbers of parameters. And since it is an online algorithm, it can also handle massive datasets. For example (Zhang et al. 2010) use a version of this algorithm to fit a multi-class classifier online to very large datasets. They beat alternative (non Bayesian) online learning algorithms, and sometimes even outperform state of the art batch (offline) learning methods such as SVMs (described in Section 14.5).

18.6 Hybrid discrete/continuous SSMs

Many systems contain both discrete and continuous hidden variables; these are known as **hybrid systems**. For example, the discrete variables may indicate whether a measurement sensor is faulty or not, or which "regime" the system is in. We will see some other examples below.

A special case of a hybrid system is when we combine an HMM and an LG-SSM. This is called a **switching linear dynamical system** (SLDS), a **jump Markov linear system** (JMLS), or a **switching state space model** (SSSM). More precisely, we have a discrete latent variable, $q_t \in \{1, \ldots, K\}$, a continuous latent variable, $\mathbf{z}_t \in \mathbb{R}^L$, an continuous observed response $\mathbf{y}_t \in \mathbb{R}^D$ and an optional continuous observed input or control $\mathbf{u}_t \in \mathbb{R}^U$. We then assume that the continuous variables have linear Gaussian CPDs, conditional on the discrete states:

$$p(q_t = k|q_{t-1} = j, \boldsymbol{\theta}) = A_{ij} \tag{18.131}$$

$$p(\mathbf{z}_t|\mathbf{z}_{t-1}, q_t = k, \mathbf{u}_t, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{z}_t|\mathbf{A}_k\mathbf{z}_{t-1} + \mathbf{B}_k\mathbf{u}_t, \mathbf{Q}_k) \tag{18.132}$$

$$p(\mathbf{y}_t|\mathbf{z}_t, q_t = k, \mathbf{u}_t, \boldsymbol{\theta}) = \mathcal{N}(\mathbf{y}_t|\mathbf{C}_k\mathbf{z}_t + \mathbf{D}_k\mathbf{u}_t, \mathbf{R}_k) \tag{18.133}$$

See Figure 18.12(a) for the DGM representation.

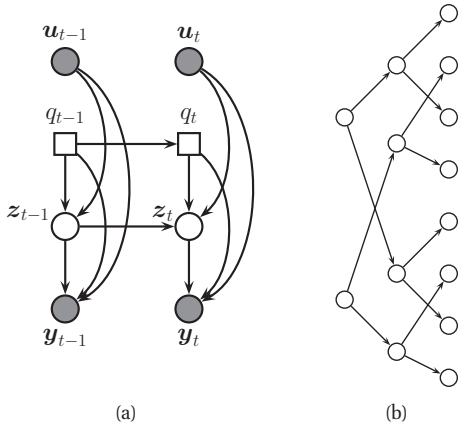


Figure 18.12 A switching linear dynamical system. (a) Squares represent discrete nodes, circles represent continuous nodes. (b) Illustration of how the number of modes in the belief state grows exponentially over time. We assume there are two binary states.

18.6.1 Inference

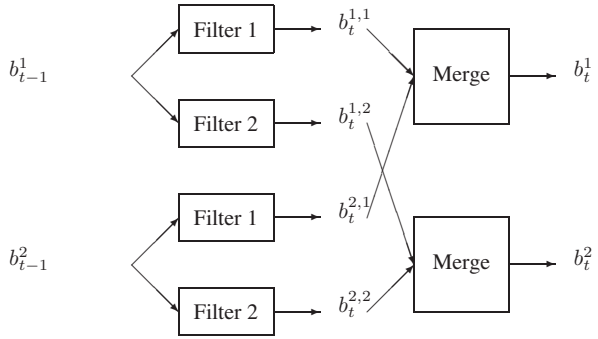
Unfortunately inference (i.e., state estimation) in hybrid models, including the switching LG-SSM model, is intractable. To see why, suppose q_t is binary, but that only the dynamics $\mathbf{A}$ depend on q_t, not the observation matrix. Our initial belief state will be a mixture of 2 Gaussians, corresponding to $p(\mathbf{z}_1|\mathbf{y}_1, q_1 = 1)$ and $p(\mathbf{z}_1|\mathbf{y}_1, q_1 = 2)$. The one-step-ahead predictive density will be a mixture of 4 Gaussians $p(\mathbf{z}_2|\mathbf{y}_1, q_1 = 1, q_2 = 1)$, $p(\mathbf{z}_2|\mathbf{y}_1, q_1 = 1, q_2 = 2)$, $p(\mathbf{z}_2|\mathbf{y}_1, q_1 = 2, q_2 = 1)$, and $p(\mathbf{z}_2|\mathbf{y}_1, q_1 = 2, q_2 = 2)$, obtained by passing each of the prior modes through the 2 possible transition models. The belief state at step 2 will also be a mixture of 4 Gaussians, obtained by updating each of the above distributions with $\mathbf{y}_2$. At step 3, the belief state will be a mixture of 8 Gaussians. And so on. So we see there is an exponential explosion in the number of modes (see Figure 18.12(b)).

Various approximate inference methods have been proposed for this model, such as the following:

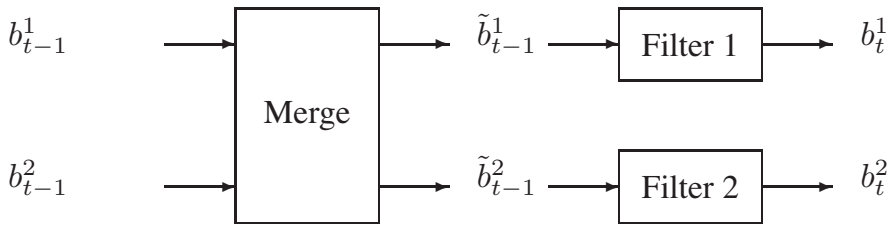
- Prune off low probability trajectories in the discrete tree; this is the basis of **multiple hypothesis tracking** (Bar-Shalom and Fortmann 1988; Bar-Shalom and Li 1993).
- Use Monte Carlo. Essentially we just sample discrete trajectories, and apply an analytical filter to the continuous variables conditional on a trajectory. See Section 23.6 for details.
- Use ADF, where we approximate the exponentially large mixture of Gaussians with a smaller mixture of Gaussians. See Section 18.6.1.1 for details.

18.6.1.1 A Gaussian sum filter for switching SSMs

A **Gaussian sum filter** (Sorenson and Alspach 1971) approximates the belief state at each step by a mixture of K Gaussians. This can be implemented by running K Kalman filters in



(a)



(b)

Figure 18.13 ADF for a switching linear dynamical system. (a) GPB2 method. (b) IMM method. See text for details.

parallel. This is particularly well suited to switching SSMs. We now describe one version of this algorithm, known as the "second order **generalized pseudo Bayes filter**" (GPB2) (Bar-Shalom and Fortmann 1988). We assume that the prior belief state b_{t-1} is a mixture of K Gaussians, one per discrete state:

$$b^i_{t-1} \triangleq p(\mathbf{z}_{t-1}, q_{t-1} = i|\mathbf{y}_{1:t-1}) = \pi_{t-1,i}\mathcal{N}(\mathbf{z}_{t-1}|\boldsymbol{\mu}_{t-1,i}, \boldsymbol{\Sigma}_{t-1,i}) \tag{18.134}$$

We then pass this through the K different linear models to get

$$b^{ij}_t \triangleq p(\mathbf{z}_t, q_{t-1} = i, q_t = j|\mathbf{y}_{1:t}) = \pi_{tij}\mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t,ij}, \boldsymbol{\Sigma}_{t,ij}) \tag{18.135}$$

where $\pi_{tij} = \pi_{t-1,i}p(q_t = j|q_{t-1} = i)$. Finally, for each value of j, we collapse the K Gaussian mixtures down to a single mixture to give

$$b^j_t \triangleq p(\mathbf{z}_t, q_t = j|\mathbf{y}_{1:t}) = \pi_{tj}\mathcal{N}(\mathbf{z}_t|\boldsymbol{\mu}_{t,j}, \boldsymbol{\Sigma}_{t,j}) \tag{18.136}$$

See Figure 18.13(a) for a sketch.

The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by $q = \arg\min_q \mathbb{KL}(q||p)$, where $p(\mathbf{z}) = \sum_k \pi_k \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ and $q(\mathbf{z}) = \mathcal{N}(\mathbf{z}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$. This can be solved by moment matching, that is,

$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{z}] = \sum_k \pi_k \boldsymbol{\mu}_k \tag{18.137}$$

$$\boldsymbol{\Sigma} = \mathrm{cov}[\mathbf{z}] = \sum_k \pi_k \left(\boldsymbol{\Sigma}_k + (\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T\right) \tag{18.138}$$

In the graphical model literature, this is called **weak marginalization** (Lauritzen 1992), since it preserves the first two moments. Applying these equations to our model, we can go from b_t^{ij} to b_t^j as follows (where we drop the t subscript for brevity):

$$\pi_j = \sum_i \pi_{ij} \tag{18.139}$$

$$\pi_{j|i} = \frac{\pi_{ij}}{\sum_{j'} \pi_{ij'}} \tag{18.140}$$

$$\boldsymbol{\mu}_j = \sum_i \pi_{j|i} \boldsymbol{\mu}_{ij} \tag{18.141}$$

$$\boldsymbol{\Sigma}_j = \sum_i \pi_{j|i} \left(\boldsymbol{\Sigma}_{ij} + (\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)(\boldsymbol{\mu}_{ij} - \boldsymbol{\mu}_j)^T\right) \tag{18.142}$$

This algorithm requires running K^2 filters at each step. A cheaper alternative is to represent the belief state by a single Gaussian, marginalizing over the discrete switch at each step. This is a straightforward application of ADF. An offline extension to this method, called **expectation correction**, is described in (Barber 2006; Mesot and Barber 2009).

Another heuristic approach, known as **interactive multiple models** or **IMM** (Bar-Shalom and Fortmann 1988), can be obtained by first collapsing the prior to a single Gaussian (by moment matching), and then updating it using K different Kalman filters, one per value of q_t. See Figure 18.13(b) for a sketch.

18.6.2 Application: data association and multi-target tracking

Suppose we are tracking K objects, such as airplanes, and at time t, we observe K' detection events, e.g., "blips" on a radar screen. We can have $K' < K$ due to occlusion or missed detections. We can have $K' > K$ due to clutter or false alarms. Or we can have $K' = K$. In any case, we need to figure out the **correspondence** between the K' detections $\mathbf{y}_{tk}$ and the K objects $\mathbf{z}_{tj}$. This is called the problem of **data association**, and it arises in many application domains.

Figure 18.14 gives an example in which we are tracking $K = 2$ objects. At each time step, q_t is the unknown mapping which specifies which objects caused which observations. It specifies the "wiring diagram" for time slice t. The standard way to solve this problem is to compute a weight which measures the "compatibility" between object j and measurement k, typically based on how close k is to where the model thinks j should be (the so-called **nearest neighbor data association** heuristic). This gives us a $K \times K'$ weight matrix. We can make this into a

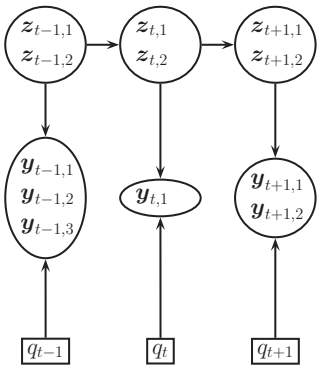


Figure 18.14 A model for tracking two objects in the presence of data-assocation ambiguity. We observe 3, 1 and 2 detections in the first three time steps.

square matrix of size $N \times N$, where $N = \max(K, K')$, by adding dummy background objects, which can explain all the false alarms, and adding dummy observations, which can explain all the missed detections. We can then compute the maximal weight bipartite matching using the **Hungarian algorithm**, which takes $O(N^3)$ time (see e.g., (Burkard et al. 2009)). Conditional on this, we can perform a Kalman filter update, where objects that are assigned to dummy observations do not perform a measurement update.

An extension of this method, to handle a variable and/or unknown number of objects, is known as **multi-target tracking**. This requires dealing with a variable-sized state space. There are many ways to do this, but perhaps the simplest and most robust methods are based on sequential Monte Carlo (e.g., (Ristic et al. 2004)) or MCMC (e.g., (Khan et al. 2006; Oh et al. 2009)).

18.6.3 Application: fault diagnosis

Consider the model in Figure 18.15(a). This represents an industrial plant consisting of various tanks of liquid, interconnected by pipes. In this example, we just have two tanks, for simplicity. We want to estimate the pressure inside each tank, based on a noisy measurement of the flow into and out of each tank. However, the measurement devices can sometimes fail. Furthermore, pipes can burst or get blocked; we call this a "resistance failure". This model is widely used as a benchmark in the **fault diagnosis** community (Mosterman and Biswas 1999).

We can create a probabilistic model of the system as shown in Figure 18.15(b). The square nodes represent discrete variables, such as measurement failures and resistance failures. The remaining variables are continuous. A variety of approximate inference algorithms can be applied to this model. See (Koller and Lerner 2001) for one approach, based on Rao-Blackwellized particle filtering (which is explained in Section 23.6).

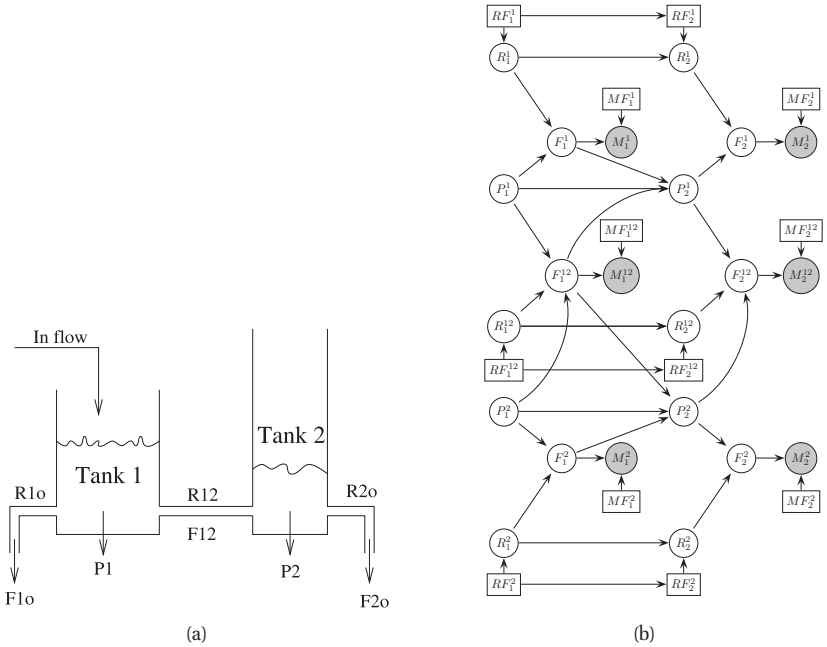


Figure 18.15 (a) The two-tank system. The goal is to infer when pipes are blocked or have burst, or sensors have broken, from (noisy) observations of the flow out of tank 1, $F1o$, out of tank 2, $F2o$, or between tanks 1 and 2, $F12$. $R1o$ is a hidden variable representing the resistance of the pipe out of tank 1, $P1$ is a hidden variable representing the pressure in tank 1, etc. Source: Figure 11 of (Koller and Lerner 2001) . Used with kind permission of Daphne Koller. (b) Dynamic Bayes net representation of the two-tank system. Discrete nodes are squares, continuous nodes are circles. Abbreviations: R = resistance, P = pressure, F = flow, M = measurement, RF = resistance failure, MF = measurement failure. Based on Figure 12 of (Koller and Lerner 2001).

18.6.4 Application: econometric forecasting

The switching LG-SSM model is widely used in **econometric forecasting**, where it is called a **regime switching** model. For example, we can combine two linear trend models (see Section 18.2.4.2), one in which $b_t > 0$ reflects a growing economy, and one in which $b_t < 0$ reflects a shrinking economy. See (West and Harrison 1997) for further details.

Exercises

Exercise 18.1 Derivation of EM for LG-SSM

Derive the E and M steps for computing a (locally optimal) MLE for an LG-SSM model. Hint: the results are in (Ghahramani and Hinton 1996b); your task is to derive these results.

Exercise 18.2 Seasonal LG-SSM model in standard form

Write the seasonal model in Figure 18.7(a) as an LG-SSM. Define the matrices $\mathbf{A}$, $\mathbf{C}$, $\mathbf{Q}$ and $\mathbf{R}$.