

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

## 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:

$$z_t = g(u_t, z_{t-1}, t) \quad (18.1)$$

$$y_t = h(z_t, u_t, \delta_t) \quad (18.2)$$

where  $z_t$  is the hidden state,  $u_t$  is an optional input or control signal,  $y_t$  is the observation,  $g$  is the transition model,  $h$  is the observation model,  $t$  is the system noise at time  $t$ , and  $\delta_t$  is the observation noise at time  $t$ . We assume that all parameters of the model,  $\theta$ , 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,  $p(z_t|y_1:t, u_1:t, \theta)$ . (Note: we will often drop the conditioning on  $u$  and  $\theta$  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  $p(y_{t+1}|y_1:t)$ .

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

$$z_t = A z_{t-1} + B u_t + t \quad (18.3)$$

- The observation model is a linear function

$$y_t = C z_t + D u_t + \delta_t \quad (18.4)$$

- The system noise is Gaussian

$$t \sim N(0, Q_t) \quad (18.5)$$

- The observation noise is Gaussian

$$\delta_t \sim N(0, R_t) \quad (18.6)$$

This model is called a linear-Gaussian SSM (LG-SSM) or a linear dynamical system (LDS). If the parameters  $\theta_t = (A_t, B_t, C_t, D_t, Q_t, R_t)$  are independent of time, the model is called stationary.

The LG-SSM is important because it supports exact inference, as we will see. In particular, if the initial belief state is Gaussian,  $p(z_1) = N(\mu_1 | 0, \Sigma_1 | 0)$ , then all subsequent belief states will also be Gaussian; we will denote them by

$p(z_t | y_{1:t}) = N(\mu_t | t, \Sigma_t | t)$ . (The notation  $\mu_t | t$  denotes  $E[z_t | y_{1:t}]$ , and similarly for  $\Sigma_t | t$ ; thus  $\mu_0 | 0$  denotes the prior for  $z_1$  before we have seen any data. For brevity we will denote the posterior belief states using  $\mu_t | t = \mu_t$  and  $\Sigma_t | t = \Sigma_t$ .) 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.

已思考 13s

下面先把原文逐句完整翻译（不遗漏），然后做通俗、详尽的解释，最后给出要点总结。

## 翻译

### 18 状态空间模型

#### 18.1 引言

状态空间模型 (state space model, SSM) 和隐马尔可夫模型 (HMM) 很像，只是它的隐藏状态是**连续的**。该模型可以写成如下通用形式：

$$z_t = g(u_t, z_{t-1}, \varepsilon_t) \quad (18.1)$$

$$y_t = h(z_t, u_t, \delta_t) \quad (18.2)$$

其中， $z_t$  是隐藏状态， $u_t$  是可选的输入或控制信号， $y_t$  是观测， $g$  是状态转移模型， $h$  是观测模型， $\varepsilon_t$  是时刻  $t$  的系统噪声， $\delta_t$  是时刻  $t$  的观测噪声。我们假设模型的所有参数  $\theta$  已知；如果未知，它们可以被并入隐藏状态中，下面我们会讨论这一点。

使用 SSM 的主要目标之一，是**递归地估计信念状态** (belief state)，即  $p(z_t | y_{1:t}, u_{1:t}, \theta)$ 。（注：为简洁起见，我们经常会省略对  $u$  和  $\theta$  的条件。）我们将在本章稍后讨论实现这一点的算法。我们还将讨论如何把关于隐藏状态的信念转化为对未来可观测量的预测，即计算**后验预测分布**  $p(y_{t+1} | y_{1:t})$ 。

SSM 的一个重要特例是：所有的条件概率分布 (CPD) 都是**线性-高斯的**。也就是说，我们假设

- **状态转移模型是线性的**

$$z_t = A_t z_{t-1} + B_t u_t + \varepsilon_t \quad (18.3)$$

- **观测模型是线性的**

$$y_t = C_t z_t + D_t u_t + \delta_t \quad (18.4)$$

- **系统噪声是高斯的**

$$\varepsilon_t \sim N(0, Q_t) \quad (18.5)$$

- **观测噪声是高斯的**

$$\delta_t \sim N(0, R_t) \quad (18.6)$$

这个模型被称为**线性-高斯状态空间模型 (LG-SSM)** 或**线性动态系统 (LDS)**。如果参数  $\theta_t = (A_t, B_t, C_t, D_t, Q_t, R_t)$  与时间无关，则称该模型是**平稳的 (stationary)**。

LG-SSM 之所以重要，是因为它支持**精确推断**，如下所示。特别地，如果初始信念状态是高斯分布， $p(z_1) = N(\mu_{1|0}, \Sigma_{1|0})$ ，那么随后所有的信念状态也都将是高斯分布；我们记为

$$p(z_t | y_{1:t}) = N(\mu_{t|t}, \Sigma_{t|t}).$$

(记号  $\mu_{t|\tau}$  表示  $E[z_t | y_{1:\tau}]$ ,  $\Sigma_{t|\tau}$  类似；因此  $\mu_{1|0}$  表示在我们尚未看到任何数据之前对  $z_1$  的先验。为简便起见，我们将后验信念状态记作  $\mu_{t|t} = \mu_t$  和  $\Sigma_{t|t} = \Sigma_t$ 。) 我们可以使用著名的**卡尔曼滤波器**高效地计算这些量，详见第 18.3.1 节。在讨论算法之前，我们先讨论一些重要的应用。

---

## 解释 (通俗+详尽)

### 1) 这类模型在干什么？

- 我们有一个“看不见”的真实状态  $z_t$ （比如：汽车的真实位置和速度、体温的真实曲线、经济里“看不见”的景气指数）。
- 我们能看到的是带噪声的观测  $y_t$ （比如：GPS 读数、体温计读数、统计局发布的指标），以及可能的外部输入  $u_t$ （比如：油门/刹车、空调的加热功率、政策刺激强度）。
- 状态如何随时间变化，由**转移模型**  $g(\cdot)$  描述；观测如何由状态产生，由**观测模型**  $h(\cdot)$  描述；二者都受噪声影响（系统噪声  $\varepsilon_t$  和观测噪声  $\delta_t$ ）。

直观比喻：

把世界想成“真实的你”（隐藏状态），你照镜子看到的自己（观测）会因为镜子变形、光照不好（观测噪声）而有误差；同时你本人也在走动、跳跃（状态在演化），这个演化过程本身也有不确定性（系统噪声）。

### 2) “信念状态”是什么？

- 信念状态  $p(z_t | y_{1:t}, u_{1:t})$  表示：在看到截至时刻  $t$  的所有观测后，我们对“真实状态  $z_t$ ”的不确定性描述。
- 它不是一个单一数值，而是一个**分布**（比如一个钟形的高斯分布），代表了“我们有多确定/不确定”。

为什么“递归”很重要？

- 现实中我们是一条条地接收新数据（如每秒来一条新传感器读数）。递归算法允许我们用上一步的结果 + 新数据，快速更新对当前状态的判断，而不用每次都把全部历史数据重新算一遍。这既高效又稳定。

### 3) 后验预测 $p(y_{t+1} | y_{1:t})$ 是啥?

- 有了对当前隐藏状态的信念，我们可以“往前看一步”，预测下一次会观测到什么值。这就是**后验预测分布**。
- 实际上：先用转移模型把  $z_t$  的不确定性“传到”  $z_{t+1}$  (得到  $p(z_{t+1} | y_{1:t})$ )，再用观测模型把它映射到  $y_{t+1}$ 。

### 4) 线性-高斯这个特例为什么特别香?

当

- 转移关系是线性的： $z_t = A_t z_{t-1} + B_t u_t + \varepsilon_t$
- 观测关系是线性的： $y_t = C_t z_t + D_t u_t + \delta_t$
- 两种噪声都是高斯分布

就会出现一个“闭包”性质：

如果起初  $z_1$  服从高斯分布，那么之后**所有时刻的后验**也都会是高斯分布。

这意味着我们只需要跟踪**均值**  $\mu_{t|t}$  和**协方差**  $\Sigma_{t|t}$  两个量，就能完整表示信念状态，计算量小、实现简单、数值稳定。

- $A_t$ : 状态如何从上一步线性地变到这一步 (例如“位置=位置+速度×时间步长；速度=速度+加速度×时间步长”)。
- $B_t$ : 外部输入  $u_t$  (如油门/刹车) 对状态的线性影响。
- $C_t$ : 隐藏状态如何线性地产生可观测量 (例如“测得的距离≈真实位置×某个比例”)。
- $D_t$ : 外部输入对观测的直接线性影响 (有些系统里会有)。
- $Q_t$ : 系统噪声的协方差矩阵，刻画“演化有多不确定”。
- $R_t$ : 观测噪声的协方差矩阵，刻画“测量有多嘈杂”。

**平稳 (stationary)**：如果这些矩阵不随时间变化 ( $A_t = A, \dots, R_t = R$ )，系统就更容易分析和实现。

### 5) 卡尔曼滤波器在做什么?

卡尔曼滤波器是处理线性-高斯 SSM 的**最优递归算法**。每个时间步做两件事：

1. **预测 (时间更新)**：

用  $A_t, B_t, Q_t$  把上一步的  $(\mu_{t-1|t-1}, \Sigma_{t-1|t-1})$  传播到当前，得到对  $z_t$  的先验  $(\mu_{t|t-1}, \Sigma_{t|t-1})$ 。

2. **校正 (测量更新)**：

拿到新观测  $y_t$  后，用  $C_t, D_t, R_t$  对先验进行修正，得到后验  $(\mu_{t|t}, \Sigma_{t|t})$ 。修正的力度由著名的**卡尔曼增益**决定：观测越精确 ( $R_t$  小)，就越相信测量；系统预测越不确定 ( $\Sigma_{t|t-1}$  大)，也越依赖观测来纠正。

结果：整个后验始终是高斯，只需更新均值和协方差，既快又准。

## 6) 参数未知怎么办？

原文提到：如果参数  $\theta$  (如  $A, C, Q, R$  等) 未知，可以把它们也并入隐藏状态一并估计（这会让“状态维度”变大，也通常需要更复杂的方法，如扩展/粒子滤波或 EM 学习等，后文会讲）。

---

## 要点总结

- SSM vs HMM：HMM 的隐藏状态是离散的；SSM 的隐藏状态是连续的。
- 基本结构：
  - 转移： $z_t = g(u_t, z_{t-1}, \varepsilon_t)$
  - 观测： $y_t = h(z_t, u_t, \delta_t)$
- 目标：递归估计信念状态  $p(z_t | y_{1:t}, u_{1:t})$ ，并得到下一步观测的后验预测  $p(y_{t+1} | y_{1:t})$ 。
- 线性-高斯特例 (LG-SSM/LDS)：
  - 线性转移、线性观测，两个噪声都高斯；若初始是高斯，整个过程的后验一直是高斯。
  - 只需跟踪均值  $\mu_{t|t}$  与协方差  $\Sigma_{t|t}$ 。
  - 若参数不随时间变化，称为平稳。
- 卡尔曼滤波器：在 LG-SSM 中提供精确且高效的递归推断（预测 + 校正）。
- 参数未知也可处理：可将参数并入状态共同估计，但算法会更复杂。

如果你愿意，我可以把“卡尔曼滤波”的两个更新步骤写成具体公式，并配一个“跟踪小车位置-速度”的数值小例子，帮助你更直观地走一遍。

这一节主要在讲什么？我不太懂

已思考 24s

没关系，这一节其实就讲了三件事（配个直觉版）：

### 1. 什么是状态空间模型 (SSM)

现实里有个“看不见的真相”在随时间变化（隐藏状态  $z_t$ ，比如车的真实位置和速度），我们能看到的是带噪声的读数（观测  $y_t$ ，比如抖动的 GPS）。还有可能有控制输入  $u_t$ （踩油门/刹车）。

- **转移模型**: 隐藏状态从上一刻到下一刻怎么变（再加一点随机性/系统噪声）。
  - **观测模型**: 隐藏状态如何通过“有噪声的仪器”变成我们看到的读数（观测噪声）。
- 目标**: 一边往前走一边更新我们对“此刻真实状态”的信念  $p(z_t \mid \text{到目前为止的观测})$ ，并据此预测下一次会看到什么  $p(y_{t+1} \mid \text{历史})$ 。

## 2. 线性-高斯这个特别重要的版本 (LG-SSM / LDS)

如果上面的两个关系都是**线性的**，而且两种噪声都是**高斯 (正态)**，奇妙的事发生了：

- 只要起初对状态的认识是高斯分布，那么之后每一步的“信念”仍然是高斯。
- 于是我们只用跟踪两个量就够了：**均值**（我认为现在大概在这儿）和**协方差**（我有多不确定）。
- 参数不随时间变就叫**平稳**，实现更省心。

## 3. 卡尔曼滤波器能把这事做得又快又准

对于线性-高斯系统，有个现成的最优“流水线”：

- **预测**: 根据物理/系统规律把上一刻的估计推到这一刻（带上系统噪声的不确定性）。
- **校正**: 拿到新观测后，按“观测有多可信”（观测噪声大小）把预测拉一拉，得到更新后的估计。

就这样每步“预测→校正”，一路滚动前进。

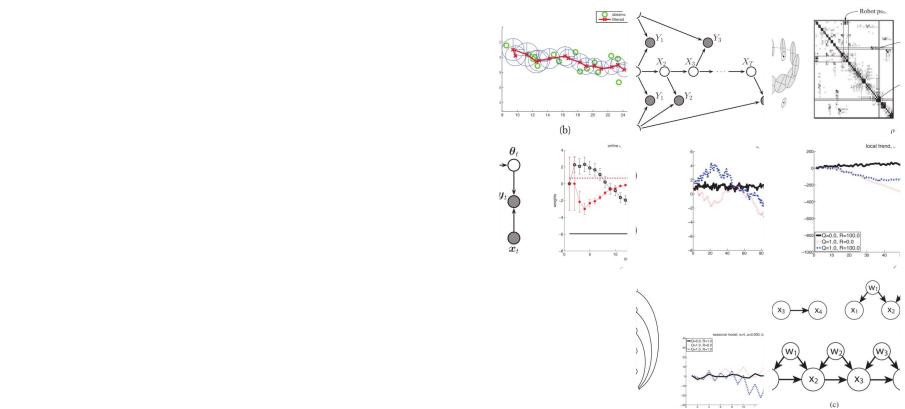
## 一个一眼能懂的例子

- 跟踪小车：状态 = (位置, 速度)，观测 = 带噪的GPS位置。
- 预测：用“位置=位置+速度×时间步”之类的公式推到下一刻。
- 校正：用新GPS把预测位置纠一纠；若GPS很准（噪声小），就更听GPS；若系统预测很不确定，也更依赖观测。

## 只记三句话

- **SSM**: 用“动态规律+噪声”连接“看不见的真实状态”和“看得见的嘈杂观测”。
- **LG-SSM**: 线性+高斯  $\Rightarrow$  所有时刻的后验都是高斯，只需跟踪均值和协方差。
- **卡尔曼滤波**: 每步“预测→校正”，在LG-SSM里做**精确且高效**的状态估计与下一步预测。

如果你愿意，我可以用这个小车例子写出卡尔曼滤波的两步公式，并给一个数字演示，让你看着每一步是怎么把“抖动的GPS”变成更平滑、更可信的轨迹的。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

Figure 18.1 Illustration of Kalman filtering and smoothing.

(a) Observations (green circles) 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 18.2 Illustration of graphical model underlying SLAM.  $l_i$  is the fixed location of landmark  $i$ ,  $x_t$  is the location of the robot, and  $y_t$  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.

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 familiar 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.

Figure 18.4 (a) A dynamic generalization of linear regression. (b) Illustration of the recursive least squares

algorithm applied to the model  $p(y|x, \theta) = N(y|w_0 + w_1x, \sigma^2)$ . We plot the marginal posterior of  $w_0$  and  $w_1$  vs number of data points. (Error bars represent  $E[w_j|y_{1:t}] \pm \text{var}[w_j|y_{1:t}]$ .) After seeing all the data, we converge to the offline ML (least squares) solution, represented by the horizontal lines.

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

Figure 18.7 (a) Seasonal model. (b) Sample output, for  $a_0 = b_0 = 0, c_0 = (1, 1, 1)$ , with a period of 4. Color code as in Figure 18.5.

18.8 (a) An AR(1) model. (b) An MA(1) model represented as a bi-directed graph. (c) An ARMA(1,1)

## 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  $z_t \in \mathbb{R}^4$  as follows:

$$z_t = [z_{1t} \ z_{2t} \ \dot{z}_{1t} \ \dot{z}_{2t}]^T. \quad (18.7)$$

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:

$$z_t = A z_{t-1} + t \quad (18.8)$$

$$\begin{bmatrix} z_{1t} \\ z_{2t} \\ \dot{z}_{1t} \\ \dot{z}_{2t} \end{bmatrix} = \begin{bmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} z_{1,t-1} \\ z_{2,t-1} \\ \dot{z}_{1,t-1} \\ \dot{z}_{2,t-1} \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} t \quad (18.9)$$

where  $t \sim N(0, Q)$  is the system noise, and  $\Delta$  is the sampling

period. This says that the new location  $z_{j,t}$  is the old location  $z_{j,t-1}$  plus  $\Delta$  times the old velocity  $\dot{z}_{j,t-1}$ , plus random noise,  $\delta_t$ , for  $j = 1:2$ . Also, the new velocity  $\dot{z}_{j,t}$  is the old velocity  $\dot{z}_{j,t-1}$  plus random noise,  $\delta_t$ , 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  $y_t \in \mathbb{R}^2$  represent our observation, which we assume is subject to Gaussian noise. We can model this as follows:

$$y_t = C z_t + \delta_t \quad (18.10)$$

$$y_1, y_2 = 1000 \ 0100 \ | \ | \ | \ z_1, z_2, \dot{z}_1, \dot{z}_2 \ | \ | \ | \ \delta_1, \delta_2, \delta_3, \delta_4 \ | \ | \ | \quad (18.11)$$

where  $\delta_t \sim N(0, R)$  is the measurement noise.

Finally, we need to specify our initial (prior) beliefs about the state of the object,  $p(z_1)$ . We will assume this is a Gaussian,  $p(z_1) = N(z_1 | \mu_1 | 0, \Sigma_1 | 0)$ . We can represent prior ignorance by making  $\Sigma_1 | 0$  suitably "broad", e.g.,  $\Sigma_1 | 0 = \infty$ . We have now fully specified the model and can perform sequential Bayesian updating to compute  $p(z_t | y_1:t)$  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  $p(z_t | y_1:t)$ , from which we can compute  $p(z_1, z_2 | y_1:t)$  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,  $E[z_t | y_1:t]$ , 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  $p(z_t | y_1:T)$ ; 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 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  $L_1, \dots, L_K$  (each is a vector in  $\mathbb{R}^2$ ). For simplicity, we will assume these are uniquely identifiable. Let  $x_t$  represent the unknown location of the robot at time  $t$ . We define the state space to be  $z_t = (x_t, L_1:K)$ ; we assume the landmarks are static, so their motion model is a constant, and they have no system noise. If  $y_t$  measures the distance from  $x_t$  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  $p(y_t|z_t, L)$  is linear-Gaussian, and we use a Gaussian motion model for  $p(x_t|x_{t-1}, u_t)$ , 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  $\text{cov}[x_t|y_1:t, u_1:t]$ , grow and then shrink. (Note that in this section, we assume that a human is joysticking the robot through the environment, so  $u_1:t$  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  $\Sigma_t$ . Actually, it is more interesting to visualize the posterior precision matrix,  $\Lambda_t = \Sigma_t^{-1}$ , 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  $L$ ), so the GGM is a disconnected graph, and  $\Lambda_t$  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  $y_{1:t}$ , because there is a path between them via the unknown sequence of  $x_{1:t}$  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,  $x_{1:t}$ , the landmark locations are independent. That is,  $p(L|x_{1:t}, y_{1:t}) = \prod_{k=1}^K p(L_k|x_{1:t}, y_{1:t})$ . 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.

### 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  $p(\theta) = N(\theta|\theta_0, \Sigma_0)$ . (If we want to do online ML estimation, we can just set  $\Sigma_0 = \infty I$ .) Let the hidden state be  $z_t = \theta$ ; if we assume the regression parameters do not change, we can set  $A_t = I$  and  $Q_t = 0I$ , so  $p(\theta_t|\theta_{t-1}) = N(\theta_t|\theta_{t-1}, 0I) = \delta\theta_{t-1}(\theta_t)$  (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  $C_t = x_t \theta_t$ , and  $R_t = \sigma^2$ , so the (non-stationary) observation model has the form

$$N(y_t | C_t z_t, R_t) = N(y_t | x_t \theta_t, \sigma^2) \quad (18.13)$$

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

$$\mu_t = A_t \mu_{t-1} + K_t(y_t - C_t A_t \mu_{t-1}) \quad (18.14)$$

where  $K_t$  is known as the Kalman gain matrix. Based on Equation 18.39, one can show that  $K_t = \Sigma_t C_t^T R_t^{-1}$ . In this context, we have  $K_t = \Sigma_t x_t / \sigma^2$ . Hence the update for the parameters becomes

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \sigma^2 \Sigma_t (y_t - x_t \hat{\theta}_{t-1}) x_t \quad (18.15)$$

If we approximate  $\sigma^2 \Sigma_t$  with  $\eta_t I$ , 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  $\eta_t$  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  $\hat{y}_{t+1} = f(y_{1:t}, \theta)$ , 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

$$y_t = a_t + \epsilon_t, \quad \epsilon_t \sim N(0, R) \quad (18.16)$$

$$a_t = a_{t-1} + \eta_t, \quad \eta_t \sim N(0, Q) \quad (18.17)$$

where the hidden state is just  $z_t = a_t$ . This model asserts that the observed data  $y_t \in R$  is equal to some unknown level term  $a_t \in R$ , 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.

#### 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:

$$y_t = a_t + \epsilon_t, \quad \epsilon_t \sim N(0, R) \quad (18.18)$$

$$a_t = a_{t-1} + b_t + \eta_t, \quad \eta_t \sim N(0, Q_a) \quad (18.19)$$

$$b_t = b_{t-1} + \beta_t, \quad \beta_t \sim N(0, Q_b) \quad (18.20)$$

See Figure 18.6(a). We can write this in standard form by defining  $z_t = (a_t, b_t)$  and

$$A = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix}, C = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}, Q = \begin{pmatrix} Q_a & 0 \\ 0 & Q_b \end{pmatrix} \quad (18.21)$$

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 hence  $E[y_t | y_{1:t-1}] = a_0 + t b_0$ . 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:

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

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

$$xt = p \sum_{i=1}^p \alpha_i xt - i + q \sum_{j=1}^q \beta_j wt - j + vt \quad (18.23)$$

where  $vt, wt \sim N(0, 1)$  are independent Gaussian noise terms. If  $q = 0$ , we have a pure AR model, where  $xt \perp xi | xt-1:t-p$ , for  $i < t - p$ . For example, if  $p = 1$ , we have the AR(1) model shown in Figure 18.8(a). (The  $vt$  nodes are implicit in the Gaussian CPD for  $xt$ .) This is just a first-order Markov chain. If  $p = 0$ , we have a pure MA model, where  $xt \perp xi$ , for  $i < t - q$ . For example, if  $q = 1$ , we have the MA(1) model shown in Figure 18.8(b). Here the  $wt$  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.

已思考 22s

下面先把你给出的内容逐条完整翻译（不遗漏），然后给出通俗且详尽的解释，最后总结要点。中间的数学我会配上直觉理解，不需要高数背景也能看懂。

## 翻译

**图 18.1 卡尔曼滤波与平滑的示意图。**

- (a) 观测（绿色圆点）由一个向右运动的物体生成（真实位置用黑色方块表示）。
- (b) 点状红线给出了滤波后的估计。红色叉号是后验均值，蓝色圆圈是由后验协方差导出的95% 置信椭圆。为清楚起见，我们只在每隔一个时间步绘制一次椭圆。
- (c) 与 (b) 相同，但使用离线的卡尔曼平滑。

### 图 18.2 SLAM (同时定位与建图) 底层图模型示意。

$L^i$  是地标  $i$  的固定位置,  $x_t$  是机器人的位置,  $y_t$  是观测。在这条轨迹中, 机器人在第 1 个时间步看到地标 1 和 2, 然后只看到地标 2, 然后只看到地标 1, 如此反复。

### 图 18.3 SLAM 问题示意。

- (a) 一个机器人从左上角出发, 顺时针绕一圈回到起点。我们可以看到: 关于机器人位置的后验不确定性 (用椭圆表示) 先增大, 随后当它回到熟悉位置并闭环 (closing the loop) 时又减小。如果我们执行平滑, 这些新的信息会向时间的过去方向传播, 从而消除整条轨迹中的歧义。
- (b) 我们显示了精度矩阵 (precision matrix), 它表示地标之间以及地标与机器人位姿 (pose) 之间的稀疏相关性。这个稀疏的精度矩阵可以如图所示, 被可视化为一个高斯图模型 (Gaussian graphical model)。

### 图 18.4

- (a) 线性回归的动态推广。
- (b) 对模型  $p(y | x, \theta) = N(y | w_0 + w_1 x, \sigma^2)$  应用递归最小二乘 (RLS) 算法的示意。我们把  $w_0$  与  $w_1$  的边际后验随数据点数量的变化画出来 (误差线表示  $E[w_j | y_{1:t}] \pm \text{var}[w_j | y_{1:t}]$ )。当看完所有数据后, 曲线收敛到离线的极大似然 (最小二乘) 解, 用横线表示。

### 图 18.5

- (a) 局部水平 (local level) 模型。
- (b) 样例输出,  $a_0 = 10$ 。黑色实线:  $Q = 0, R = 1$  (确定性的系统, 噪声观测); 红色虚线:  $Q = 0.1, R = 0$  (有噪声的系统, 确定性的观测); 蓝色点划线:  $Q = 0.1, R = 1$  (系统与观测均有噪声)。

### 图 18.6

- (a) 局部趋势 (local trend)。
- (b) 样例输出,  $a_0 = 10, b_0 = 1$ 。颜色标注与图 18.5 相同。

### 图 18.7

- (a) 季节性模型。
- (b) 样例输出,  $a_0 = b_0 = 0, c_0 = (1, 1, 1)$ , 周期为 4。颜色标注与图 18.5 相同。

### 图 18.8

- (a) 一个 AR(1) 模型。
- (b) 用双向图表示的 MA(1) 模型。
- (c) 一个 ARMA(1,1)。

## 18.2 状态空间模型 (SSM) 的应用

SSM 有很多应用，下面若干小节将予以讨论。为简化起见，我们主要聚焦于**线性-高斯 SSM (LG-SSM)**，尽管**非线性和/或非高斯**的 SSM 实际上应用得更广泛。

### 18.2.1 用 SSM 做目标跟踪

卡尔曼滤波最早的应用之一，就是用带噪声的测量（如雷达）跟踪目标（如飞机、导弹）。这里我们给出一个简化示例来说明关键思想。考虑一个在二维平面上运动的物体。令  $z_t^1$  与  $z_t^2$  分别为该物体的水平与垂直位置， $\dot{z}_t^1$  与  $\dot{z}_t^2$  为对应的速度。我们可以把它表示成 4 维状态向量  $z_t \in \mathbb{R}^4$ ：

$$z_t^\top = [z_t^1, z_t^2, \dot{z}_t^1, \dot{z}_t^2]. \quad (18.7)$$

假设物体以**恒定速度**运动，但会受到随机高斯噪声的“扰动”（例如风）。于是系统动力学可以写成

$$z_t = A_t z_{t-1} + \varepsilon_t, \quad (18.8)$$

$$\begin{pmatrix} z_t^1 \\ z_t^2 \\ \dot{z}_t^1 \\ \dot{z}_t^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & \Delta & 0 \\ 0 & 1 & 0 & \Delta \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} z_{t-1}^1 \\ z_{t-1}^2 \\ \dot{z}_{t-1}^1 \\ \dot{z}_{t-1}^2 \end{pmatrix} + \begin{pmatrix} \varepsilon_t^1 \\ \varepsilon_t^2 \\ \varepsilon_t^3 \\ \varepsilon_t^4 \end{pmatrix}, \quad (18.9)$$

其中  $\varepsilon_t \sim N(0, Q)$  是系统噪声， $\Delta$  是采样周期。这就是说：新位置  $z_{j,t} = \text{旧位置 } z_{j,t-1} + \Delta \times \text{旧速度 } \dot{z}_{j,t-1} + \text{噪声 } \varepsilon_{j,t}$  ( $j = 1, 2$ )；新速度  $\dot{z}_{j,t} = \text{旧速度 } \dot{z}_{j,t-1} + \text{噪声 } \varepsilon_{j,t}$  ( $j = 3, 4$ )。由于速度在随机变化，这称为**随机加速度模型**：物体遵循牛顿定律运动，但速度会有随机改变。

现在假设我们只能观测到位置，看不到速度。令  $y_t \in \mathbb{R}^2$  为观测，且其受高斯噪声影响。观测模型为

$$y_t = C_t z_t + \delta_t, \quad (18.10)$$

$$\begin{pmatrix} y_t^1 \\ y_t^2 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} z_t^1 \\ z_t^2 \\ \dot{z}_t^1 \\ \dot{z}_t^2 \end{pmatrix} + \begin{pmatrix} \delta_t^1 \\ \delta_t^2 \end{pmatrix}, \quad (18.11)$$

其中  $\delta_t \sim N(0, R)$  是测量噪声。

最后，我们需要给出物体状态的**先验**  $p(z_1)$ 。假设它是高斯分布：

$p(z_1) = N(z_1 | \mu_{1|0}, \Sigma_{1|0})$ 。我们可以通过把  $\Sigma_{1|0}$  设得足够“宽”（例如  $\Sigma_{1|0} = \infty I$ ）来表达先验无知。至此模型完全指定，可以用一种称为**卡尔曼滤波**的顺序贝叶斯更新算法（18.3.1 节会介绍）来计算  $p(z_t | y_{1:t})$ 。

图 18.1(a) 给出了一个例子。物体向右移动，并在每个时间步产生一个观测（可以想象为雷达屏上的“闪点”）。我们看到这些闪点后，使用卡尔曼滤波把噪声“滤掉”。在每一步我们都得到  $p(z_t | y_{1:t})$ ，由此可以通过把速度维度积分掉，得到  $p(z_t^1, z_t^2 | y_{1:t})$ （因为后验是高斯，这一步很容易）。我们对物体位置的“最佳猜测”是后验均值  $E[z_t | y_{1:t}]$ ，在图 18.1(b) 中用红色叉号表示；与其相伴的不确定性用一个“包含 95% 概率质量”的椭圆表示。可以看到，随着时间推移，我们的不确定性逐渐减小，因为初始不确定性的影响被“冲淡”了；同时估计的轨迹也“滤掉”了一些噪声。若要得到图 18.1(c) 那样更光滑的曲线，需要使用**卡尔曼平滑器**来计算  $p(z_t | y_{1:T})$ ；它既依赖“将来”也依赖“过去”的数据（18.3.2 节讨论）。

## 18.2.2 机器人 SLAM

考虑一个在未知二维环境中移动的机器人。它需要**同时构建地图并定位自己**。这个问题称为SLAM (Simultaneous Localization And Mapping)，广泛用于移动机器人，也用于室内导航（因为建筑内 GPS 不好用）。设我们用  $K$  个**固定地标的**二维位置来表示地图，记为  $L_1, \dots, L_K$ （每个  $L_i \in \mathbb{R}^2$ ）。为简化，假设这些地标**可唯一识别**。令  $x_t$  为机器人在时刻  $t$  的未知位置。定义状态为  $z_t = (x_t, L_{1:K})$ ；地标静止，因此它们的运动模型是常数，且**没有系统噪声**。如果  $y_t$  衡量的是从  $x_t$  到最近一组地标的距离，机器人就能基于所看到的东西来更新对地标位置的估计。图 18.2 展示了  $K = 2$  时的图模型，并展示了“先看到地标 1 和 2，再只看到 2，再只看到 1 ……”的情况。若假设观测模型  $p(y_t | z_t, L)$  是线性-高斯的，且采用高斯运动模型  $p(x_t | x_{t-1}, u_t)$ ，我们就可以用卡尔曼滤波维护关于“机器人位置”和“地标位置”的信念状态 (Smith and Cheeseman 1986; Choset and Nagatani 2001)。随着时间推移，由于车轮打滑等原因，机器人位置的不确定性会**增加**；但当机器人回到熟悉位置时，不确定性又会**减小**。这称为**闭环**，如图 18.3(a) 所示：表示  $\text{cov}[x_t | y_{1:t}, u_{1:t}]$  的不确定性椭圆先增后减。（本节里假设有人用摇杆控制机器人在环境中移动，所以  $u_{1:t}$  作为输入是已知的；我们不讨论“走到哪里去”的决策问题。）

由于信念状态是高斯的，我们可以把后验协方差矩阵  $\Sigma_t$  可视化。事实上，更有趣的是把**后验精度矩阵**  $\Lambda_t = \Sigma_t^{-1}$  可视化，因为它通常**相当稀疏**，如图 18.3(b)。原因在于：精度矩阵中的零对应于相应**无向高斯图模型**（见 19.4.4 节）里**缺失的边**。起初（若我们对  $L$  使用对角先验），所有地标两两不相关，所以图模型是不连通图， $\Lambda_t$  是对角的。然而，机器人移动的同时会在**相邻地标之间引入相关性**。直观地说：机器人用地标的距离来估计自己位置，而地标的位置又根据机器人位置来估计，于是大家彼此相关。从图 18.2 的图模型更能看清这一点：由于存在经过未知的  $x_{1:t}$  序列把  $L_1$  与  $L_2$  连接起来的路径，所以在  $y_{1:t}$  条件下， $L_1$  与  $L_2$ **并没有被 d-分离**。随着精度矩阵变得更稠密，**精确推断的时间复杂度达到  $O(K^3)$** （这是 DBN 推断中“纠缠问题”的一个例子），导致该方法难以用于大规模地图。

有两个主要解决办法。第一：注意到“相关性图案”会**随机器人位置一起移动**（见图 18.3(b))，其余相关性会随时间**逐渐变弱**。因此可以用一种叫**thin junction tree filter**

(Paskin 2003, 关于联结树见 20.4 节) 的技巧, 动态剪除图模型里的弱边。第二: 注意到在已知机器人路径  $x_{1:t}$  的条件下, 地标位置相互独立, 即

$$p(L \mid x_{1:t}, y_{1:t}) = \prod_{k=1}^K p(L_k \mid x_{1:t}, y_{1:t}).$$

这构成了 FastSLAM 的基础——它把卡尔曼滤波与粒子滤波结合 (见 23.6.3 节)。(Thrun 等, 2006) 对 SLAM 与移动机器人有更详细的论述。

### 18.2.3 用递归最小二乘进行在线参数学习

我们可以把许多统计模型的参数放进 SSM, 通过在线贝叶斯推断来学习它们。本节关注线性回归; 18.5.3.2 节讨论逻辑回归。基本思路是: 把隐藏状态设为回归参数, 把 (随时间变化的) 观测模型设为当前的数据向量。更具体地, 令先验

$p(\theta) = N(\theta \mid \theta_0, \Sigma_0)$ 。(如果想做在线极大似然估计, 只需取  $\Sigma_0 = \infty I$ 。) 令隐藏状态  $z_t = \theta$ ; 若我们假设参数不随时间变化, 可取  $A_t = I$ ,  $Q_t = 0I$ , 则

$$p(\theta_t \mid \theta_{t-1}) = N(\theta_t \mid \theta_{t-1}, 0I) = \delta_{\theta_{t-1}}(\theta_t). \quad (18.12)$$

(如果允许参数随时间变化, 就得到所谓的动态线性模型, Harvey 1990; West & Harrison 1997; Petris 等 2009。) 取  $C_t = x_t^\top$ ,  $R_t = \sigma^2$ , 则 (非平稳的) 观测模型为

$$N(y_t \mid C_t z_t, R_t) = N(y_t \mid x_t^\top \theta_t, \sigma^2). \quad (18.13)$$

对该模型应用卡尔曼滤波, 就能随着数据流入递归更新参数的后验, 这就是递归最小二乘 (RLS)。可以推出显式更新式如下。18.3.1 节给出了卡尔曼后验均值的更新:

$$\mu_t = A_t \mu_{t-1} + K_t (y_t - C_t A_t \mu_{t-1}), \quad (18.14)$$

其中  $K_t$  称为卡尔曼增益。由式 (18.39) 可得  $K_t = \Sigma_t C_t^\top R_t^{-1}$ 。在当前情形下  $K_t = \Sigma_t x_t / \sigma^2$ 。因此参数的更新为

$$\hat{\theta}_t = \hat{\theta}_{t-1} + \frac{1}{\sigma^2} \Sigma_{t|t} (y_t - x_t^\top \hat{\theta}_{t-1}) x_t. \quad (18.15)$$

若用  $\eta_t I$  来近似  $\frac{1}{\sigma^2} \Sigma_{t|t-1}$ , 就得到最小均方 (LMS) 算法 (见 8.5.3 节)。LMS 需要你自己设计算法的步长  $\eta_t$  以确保收敛, 并且可能需要多次遍历数据; 相比之下, RLS 会自动调步长, 且在一次遍历中就能收敛到最优的后验。参见图 18.4 的例子。

### 18.2.4 用 SSM 做时间序列预测

SSM 非常适合时间序列预测, 下面解释原因。为简洁起见, 我们只讨论标量 (一维) 时间序列。我们的叙述基于 Varian (2011), 还可以参考 Aoki (1987)、Harvey (1990)、West & Harrison (1997)、Durbin & Koopman (2001)、Petris 等 (2009)、Prado & West (2010)。

乍一看, SSM 似乎与预测无关, 因为预测的目标是未来的可见变量, 而不是某个系统的隐

**藏状态。**事实上，许多经典时间序列方法只写成

$\hat{y}_{t+1} = f(y_{1:t}, \theta)$ ，并不显式使用隐藏变量（见 18.2.4.4）。

**状态空间方法**的思想是：用**潜在过程**来生成数据，把信号的不同成分拆开建模；然后把隐藏变量**积分掉**，得到对可见量的**后验预测**。在线性-高斯情况下，这些潜在过程可以**线性相加**来解释观测数据。这类方法称为**结构化时间序列模型**。下面介绍几个基础“积木”。

#### 18.2.4.1 局部水平模型 (local level)

最简单的潜在过程是**局部水平模型**：

$$y_t = a_t + v_t, \quad v_t \sim N(0, R), \quad (18.16)$$

$$a_t = a_{t-1} + \omega_t, \quad \omega_t \sim N(0, Q), \quad (18.17)$$

其中隐藏状态就是  $z_t = a_t$ 。该模型断言：观测  $y_t \in \mathbb{R}$  等于某个未知的**水平项**  $a_t \in \mathbb{R}$  加上观测噪声（方差  $R$ ）；同时，水平项随时间演化，受系统噪声（方差  $Q$ ）影响。例子见图 18.5。

#### 18.2.4.2 局部线性趋势 (local linear trend)

许多时间序列至少在局部呈现**线性上升或下降**。我们可以令水平项每步由一个**斜率项**  $b_t$  来改变：

$$y_t = a_t + v_t, \quad v_t \sim N(0, R), \quad (18.18)$$

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

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

见图 18.6(a)。把  $z_t = (a_t, b_t)$ ，可写成标准形式

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad C = \begin{pmatrix} 1 & 0 \end{pmatrix}, \quad Q = \begin{pmatrix} Q_a & 0 \\ 0 & Q_b \end{pmatrix}. \quad (18.21)$$

当  $Q_b = 0$  时， $b_t = b_0$  为常数斜率；若再有  $Q_a = 0$ ，则  $a_t = a_{t-1} + b_0 t$ 。展开得到  $a_t = a_0 + b_0 t$ ，从而

$E[y_t | y_{1:t-1}] = a_0 + tb_0$ 。这就推广了**恒定线性趋势**模型，黑线示例如图 18.6(b)。

#### 18.2.4.3 季节性 (seasonality)

许多时间序列**周期性波动**（见图 18.7(b)）。可通过加入一个潜在过程  $c_t$  来建模：这一串**季节偏移**在一个完整周期  $S$  步内（平均）**和为零**：

$$c_t = - \sum_{s=1}^{S-1} c_{t-s} + \omega_t^c, \quad \omega_t^c \sim N(0, Q_c). \quad (18.22)$$

图 18.7(a) 给出了  $S = 4$  时的图模型（因为“和为零”的约束，只需要 3 个季节性变量）。把它写成标准 LG-SSM 形式作为习题 18.2。

#### 18.2.4.4 ARMA 模型

传统的时间序列预测以 ARMA 模型为基础 (Auto-Regressive Moving-Average) :

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

其中  $v_t, w_t \sim N(0, 1)$  彼此独立。若  $q = 0$ , 得到**纯 AR 模型**, 此时  $x_t \perp x_i \mid x_{t-1:t-p}$  ( $i < t - p$ )。例如  $p = 1$  时是图 18.8(a) 的 AR(1) (高斯 CPD 中隐含了  $v_t$ )。它就是一个一阶马尔可夫链。若  $p = 0$ , 得到**纯 MA 模型**, 此时  $x_t \perp x_i$  ( $i < t - q$ )。例如  $q = 1$  时, 是图 18.8(b) 的 MA(1): 这里的  $w_t$  是**隐藏的共同原因**, 会在相邻时间步之间引入依赖, 建模短程相关。若  $p = q = 1$ , 得到图 18.8(c) 的 ARMA(1,1), 它能同时刻画短期与长期的相关。事实证明, ARMA 模型可以表示成 SSM (见 Aoki 1987; Harvey 1990; West & Harrison 1997; Durbin & Koopman 2001; Petris 等 2009; Prado & West 2010)。不过, **结构化**时间序列方法往往比 ARMA 更易理解, 而且它允许参数随时间演化, 使模型对**非平稳性**更具适应性。

## 通俗解释 (逐点拆开)

### 1) 图 18.1: 滤波 vs 平滑

- **滤波 (filtering)** : 一边走一边估计“此刻”的位置, 只用**过去到现在**的数据。红叉 = “此刻的最佳估计”, 蓝椭圆 = 这个估计的 95% 不确定范围。随着数据积累, 椭圆变小, 表示更确定。
- **平滑 (smoothing)** : 回头看全程, 用**全体数据 (过去+未来)** 来估计每一刻的位置, 会更光滑 (噪声更少)。这通常是离线做的。

### 2) 图 18.2–18.3: SLAM 的图模型和“闭环”

- **状态**包含“机器人位置 + 所有地标位置”。
- 机器人移动时位置误差会**越积越大**; 当回到熟悉地点并再次看到老地标, “误差一下子缩回去”——**闭环**。
- 由于位置和地标互相影响, 地标之间会变得相关; 把这种相关性的稀疏结构画成**精度矩阵或高斯图模型**, 有助于更快推断。
- 大图会让推断变慢 (相关变多), 于是有人发明了**剪弱边** (thin junction tree) 和 **FastSLAM** (把“路径”用粒子采样出来, 地标条件独立后再用卡尔曼) 两条路来加速。

### 3) 图 18.4: 把“参数”当作隐藏状态 → 递归最小二乘

- 在线学习线性回归：把参数  $\theta = (w_0, w_1)$  当成不会动的“状态”，每来一个样本  $x_t$  就是一条“观测方程”。
- 卡尔曼滤波给出**最优递归更新**，这就是 RLS。它会**自动调步长**，通常一遍数据就收敛到“离线最小二乘”的解。
- 如果把协方差近似成  $\eta_t I$ ，就变成常见的 LMS（需要手调步长，可能多轮遍历）。

### 4) 图 18.5–18.7: 结构化时间序列的 3 块“积木”

- **局部水平**：数据 = “缓慢漂移的水平” + 噪声。 $Q$  控制水平的“漂得多不多”， $R$  控制观测噪声大不大。
- **局部趋势**：再加一个“斜率”  $b_t$ 。当  $Q_b = 0$  时，斜率恒定，退化成“直线趋势”； $Q_b > 0$  让斜率也慢慢变。
- **季节性**：加一个会在周期内“和为 0”的循环偏移  $c_t$ （如季度、每周）。三块积木线性相加，就得到非常实用的预测模型。

### 5) 图 18.8: AR、MA、ARMA 与 SSM

- AR(p)：当前由若干**过去值**线性决定（+噪声）。
- MA(q)：当前由若干**过去的噪声脉冲**线性决定（+噪声），等价于“看不见的共同原因”导致**短程相关**。
- ARMA 结合二者，能描述不同时间尺度的相关。
- 它们都能重写为**状态空间模型**；而结构化 SSM 更直观、可“随时间自适应”。

### 6) 目标跟踪那段公式该怎么看？

- 状态  $z_t = [\text{位置}_x, \text{位置}_y, \text{速度}_x, \text{速度}_y]$ 。
- **状态转移**：新位置 = 旧位置 + 速度  $\times$  时间步（+系统噪声）；新速度 = 旧速度（+系统噪声）→ 这就是“随机加速度”。
- **观测**：只量到位置（速度不可见），读数还有**测量噪声**。
- **先验**：一开始很不确定就把协方差设大。
- **输出**：卡尔曼滤波每步给出后验均值（红叉）和不确定性（蓝椭圆）。平滑则用全数据把整条轨迹“抹平”。

小注：你贴的式 (18.11) 原文里常见有排版小错（把  $\delta_t$  写成 4 维）。与“只观测位置”一致的应是 2 维噪声： $\delta_t \in \mathbf{R}^2$ 。

## 要点总结

- **滤波 vs 平滑**: 滤波只用“到当前”的数据估计当前状态；平滑用“全程数据”回头估计每一刻，曲线更平滑。
- **SLAM**: 同时估计机器人位置和地标位置；闭环会显著降低不确定性。利用**稀疏精度矩阵、剪枝或 FastSLAM**可扩展到更大地图。
- **RLS (递归最小二乘)** : 把回归参数当成隐藏状态，卡尔曼递归更新；自动调步长，一遍数据收敛到最小二乘。
- **结构化时间序列三件套**：
  1. 局部水平（慢漂）； 2) 局部趋势（斜率可变）； 3) 季节性（周期项）； 线性相加，解释性强、预测稳。
- **AR/MA/ARMA 与 SSM**: ARMA 能等价写成 SSM；但结构化 SSM 更容易理解，也更适应**非平稳场景**。
- **跟踪示例**: 状态=位置+速度，观测=位置，系统/测量噪声都高斯；卡尔曼滤波给出红叉（均值）+蓝椭圆（不确定性）。

如果你愿意，我可以用一个 2D 小车的数值玩具例子，直接跑出“观测点、滤波估计、平滑估计”的图，让这些概念一眼可见。

这一节主要在讲什么？我不太懂

已思考 16s

没事~这节其实就是在说\*\*“状态空间模型 (SSM) 到底能拿来干啥，以及它怎么在这些应用里工作”\*\*。核心内容用大白话拆一下：

## 一句话总览

SSM把“看不见的真实状态”与“带噪的观测”连起来；在实际问题里，我们用**卡尔曼滤波/平滑**来边走边估计当前状态、回头重估整段轨迹，并把它用在**目标跟踪、机器人SLAM、在线学参数、时间序列预测**等任务上。

## 图里都在表达什么

- **图18.1:**
  - 绿色点=嘈杂观测；黑方块=真实轨迹。
  - **滤波** (红叉+蓝椭圆) =只用到当前为止的数据估计“此刻”；蓝椭圆越小，不确定性越低。
  - **平滑**=用全程数据回看每一刻，曲线更顺更准。
- **图18.2–18.3 (SLAM) :**

- 同时估计**机器人位置**和**地标位置**。走远了误差会涨；回到熟悉地方会**闭环**、误差降。
- 看精度矩阵**（协方差的逆）能发现相关结构是**稀疏的**；用剪枝或 FastSLAM（路径用粒子、地标用卡尔曼）可扩展到大地图。
- 图18.4 (RLS) :**
  - 把**回归参数**当“**隐藏状态**”，每来一条数据就**递归更新**，这就是**递归最小二乘**；它会自动调步长，一遍数据就能跑到离线最小二乘的解。
- 图18.5–18.7 (时间序列的结构化积木) :**
  - 局部水平**：数据=缓慢漂的基线 + 噪声（系统噪声  $Q$  决定“漂不漂”，观测噪声  $R$  决定“量得准不准”）。
  - 局部趋势**：再加“斜率”  $b_t$ ，可让趋势随时间慢变。
  - 季节性**：加周期项  $c_t$ （如季度/周），整体可线性相加成好用的预测模型。
- 图18.8 (AR/MA/ARMA) :**
  - 传统 ARMA 也能改写成 SSM；但“**结构化SSM**”更直观，还能让参数随时间变，适应**非平稳**数据。

## 这节讲到的四类代表性应用

### 1. 目标跟踪（飞机/导弹/人/车）：

状态=位置+速度；观测=位置（带噪）；用卡尔曼**滤波/平滑**得到平滑轨迹和不确定性（蓝椭圆）。

### 2. 机器人SLAM：

一边走一边**建图+定位**；闭环会大幅降不确定性；用稀疏结构与FastSLAM来加速。

### 3. 在线学参数 (RLS)：

数据流式到来时，递归更新回归参数，比LMS更省心（自动步长，单遍收敛）。

### 4. 时间序列预测（结构化模型）：

用“**水平+趋势+季节性**”三块积木拼出可解释、可拓展的预测模型；线性高斯时直接相加很方便。

## 关键词的直觉

- 滤波 vs 平滑**：只看过去到现在 vs 看全程再回估。
- $Q$ （系统噪声）：真状态自身“晃动”的程度； $R$ （观测噪声）：测量“抖”的程度。
- 精度矩阵** ( $\Lambda = \Sigma^{-1}$ )：零元素  $\approx$  图模型里“没边”，常是稀疏的  $\rightarrow$  算得快。

## 只记这几句就够

- SSM=用“动态规律+噪声”连接**隐藏状态**和**观测**。
- 卡尔曼**滤波/平滑**给出当前/全程的最佳估计和不确定性。

- 典型应用：跟踪、SLAM、在线学参数、时间序列预测。
- 结构化时间序列=水平+趋势+季节性三块积木线性相加，直观又好用。
- ARMA 也能写成 SSM，但结构化 SSM 更易理解且更适应非平稳。

如果你想，我可以用一个“1D 温度”或“2D 小车”的小数据，现场跑一遍滤波和平滑，让这几张图的含义立刻落地。

↳ SSM把“看不见的真实状态”与“带噪的观测”连起来

这是什么意思

已思考 18s

这句话的意思是：

**在现实里我们关心的“真实量”（状态）看不见或量不到；只能拿到带误差的读数（观测）。状态空间模型（SSM）把两者用一个概率模型连起来，好让我们从嘈杂读数中“还原”出真实量，并且知道自己有多不确定。**

换成大白话分三步看就明白了：

### 1. 看不见的真实状态是什么？

比如车子的“真实位置和速度”、体温的“真实曲线”、经济里“真实景气度”。这些往往不能直接量，或量不到全部（只能量位置，量不到速度）。

### 2. 带噪观测是什么？

我们手里是有误差的读数：抖动的 GPS、误差 $\pm 0.2^\circ\text{C}$  的体温计、含统计误差的经济指标。它们=真实值 + 测量误差（噪声）。

### 3. “连起来”到底怎么连？

用两条关系把“真实”和“读数”连接起来：

- **状态演化方程**（世界怎么自己变）：

现在的真实状态 = 上一刻的真实状态按物理/经验规律推过去 + 系统噪声

例：新位置  $\approx$  旧位置 + 旧速度 $\times$ 时间步（再加点随机抖动）。

- **观测方程**（读数从真实怎么来）：

读数 = 真实状态经过测量装置的映射 + 观测噪声

例：GPS读数  $\approx$  真实位置（再加测量误差）。

有了这两个“桥”，SSM就能做事了：

- **滤波**：每来一个新读数，就把“对当前真实状态的猜测”更新一下；读数越准（观测噪声小），就越相信它；系统本身越不确定（系统噪声大），就越依赖新读数。

- **平滑**: 事后把整段数据都看一遍，给每一时刻一个更平滑、更准的估计。

再用一个一眼懂的例子：

- **车跟踪**:

- 隐藏状态：位置与速度（看不见）；
- 观测：GPS位置（有误差）；
- 连接：物理运动方程 + “GPS=位置+测量误差”；
- 结果：卡尔曼滤波把抖动的GPS变成**红色均值点**（最佳估计）+**蓝色椭圆**（不确定性范围）。

小提示：文中常见的两个噪声参数

- $Q$ : **系统噪声**，表示“真实状态自己会不会乱晃”；
- $R$ : **观测噪声**，表示“仪器读数抖不抖”。

算法会自动根据  $Q$  和  $R$  来权衡“信物理预测”还是“信新读数”。

## 一句话总结

**SSM=两条方程（状态演化 + 观测生成）把“看不见的真相”和“有噪的读数”连接起来，用来从嘈杂数据中估计真正想要的量（还告诉你不确定性），典型实现就是卡尔曼滤波/平滑。**

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

### 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  $\pi(x_t | y_{1:t})$ . 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:

$$p(z_t|y_1:t-1, u_1:t) = N(z_t|A\mu_{t-1} + B\mu_{t-1}, Q_t)N(\mu_{t-1}, \Sigma_{t-1})$$

$$\mu_{t-1}, \Sigma_{t-1} d\mu_{t-1} (18.25)$$

$$= N(z_t|\mu_t, \Sigma_t) (18.26)$$

$$\mu_t = A\mu_{t-1} + B\mu_{t-1} (18.27)$$

$$\Sigma_t = A\Sigma_{t-1}A^T + Q_t (18.28)$$

### 18.3.1.2 Measurement step

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

$$p(z_t|y_t, y_1:t-1, u_1:t) \propto p(y_t|z_t, u_t)p(z_t|y_1:t-1, u_1:t) (18.29)$$

In Section 18.3.1.6, we show that this is given by

$$p(z_t|y_1:t, u_t) = N(z_t|\mu_t, \Sigma_t) (18.30)$$

$$\mu_t = \mu_{t-1} + K_t r_t (18.31)$$

$$\Sigma_t = (I - K_t C_t) \Sigma_{t-1} (18.32)$$

where  $r_t$  is the residual or innovation, given by the difference between our predicted observation and the actual observation:

$$r_t = y_t - \hat{y}_t (18.33)$$

$$\hat{y}_t = E[y_t|y_1:t-1, u_1:t] = C\mu_{t-1} + D\mu_{t-1} (18.34)$$

and  $K_t$  is the Kalman gain matrix, given by

$$K_t = \Sigma_{t-1} C^T S^{-1} t (18.35)$$

where

$$S_t = \text{cov}[r_t|y_1:t-1, u_1:t] (18.36)$$

$$= E[(C\mu_{t-1} + \delta_t - \hat{y}_t)(C\mu_{t-1} + \delta_t - \hat{y}_t)^T | y_1:t-1, u_1:t] (18.37)$$

$$= C\Sigma_{t-1} C^T + R_t (18.38)$$

where  $\delta_t \sim N(0, R_t)$  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

$$K_t = \Sigma_{t-1} C^T (C\Sigma_{t-1} C^T + R_t)^{-1} = (\Sigma_{t-1} C^T + C^T R_t^{-1} C)^{-1} (18.39)$$

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:  $\mu_t = \mu_{t-1} + K_t r_t$ . This says that the new mean is the old mean plus a correction factor, which is  $K_t$  times the error signal  $r_t$ . The amount of weight placed on the error signal depends on the Kalman gain matrix. If  $C_t = I$ , then  $K_t = \Sigma_{t-1} S^{-1}$ , 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,  $|K_t|$  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  $|K_t|$  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

$$\log p(y_1:T|u_1:T) = t \log p(y_t|y_1:t-1, u_1:t) \quad (18.40)$$

where

$$p(y_t|y_1:t-1, u_1:t) = N(y_t|C\mu_t|t-1, S_t) \quad (18.41)$$

#### 18.3.1.4 Posterior predictive

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

$$p(y_t|y_1:t-1, u_1:t) = N(y_t|Cz_t, R)N(z_t|\mu_t|t-1, \Sigma_t|t-1)dz_t \quad (18.42)$$

$$= N(y_t|C\mu_t|t-1, C\Sigma_t|t-1C^T + R) \quad (18.43)$$

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,  $K_t$ , which takes  $O(|y_t|^3)$  time; and the matrix-matrix multiply to compute  $\Sigma_t$ , which takes  $O(|z_t|^2)$  time. In some applications (e.g., robotic mapping), we have  $|z_t| \gg |y_t|$ , so the latter cost dominates. However, in such cases, we can sometimes use sparse approximations (see (Thrun et al. 2006)). In cases where  $|y_t| \gg |z_t|$ , we can precompute  $K_t$ , since, surprisingly, it does not depend on the actual observations  $y_1:t$  (an unusual property that is specific to linear Gaussian systems). The iterative equations for updating  $\Sigma_t$  are called the Riccati equations, and for time invariant systems (i.e., where  $\theta_t = \theta$ ), 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,  $\Lambda_t = \Sigma_t^{-1} t$  and  $\eta_t = \Lambda_t \mu_t$ , instead of the moment parameters. Another approach is the square root filter, which works with the Cholesky decomposition or the  $U_t D_t U_t^T$  decomposition of  $\Sigma_t$ . This is much more numerically stable than directly updating  $\Sigma_t$ . 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  $u_{1:t}$ . From Bayes rule for Gaussians (Equation 4.125), we have that the posterior precision is given by

$$\Sigma^{-1}|t = \Sigma^{-1}|t-1 + C_t R^{-1} t C_t \quad (18.44)$$

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

$$\begin{aligned} \Sigma_t &= \Sigma|t-1 - \Sigma|t-1 C_t (R_t + C_t \Sigma|t-1 C_t)^{-1} C_t \Sigma|t-1 \\ &\quad (18.45) \end{aligned}$$

$$= (I - K_t C_t) \Sigma|t-1 \quad (18.46)$$

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

$$\mu_t = \Sigma_t C_t R^{-1} t y_t + \Sigma|t-1 t \mu|t-1 \quad (18.47)$$

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

$$\begin{aligned} \Sigma_t C_t R^{-1} t y_t &= (\Sigma|t-1 + C_t R^{-1} t C_t)^{-1} C_t R^{-1} t y_t \\ &\quad (18.48) \end{aligned}$$

$$= \Sigma|t-1 C_t (R_t + C_t \Sigma|t-1 C_t)^{-1} y_t = K_t y_t \quad (18.49)$$

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

$$\Sigma|t-1 t \mu|t-1 \quad (18.50)$$

$$= (\Sigma|t-1 + C_t R^{-1} t C_t)^{-1} \Sigma|t-1 \mu|t-1 \quad (18.51)$$

$$= \Sigma|t-1 - \Sigma|t-1 C_t (R_t + C_t \Sigma|t-1 C_t)^{-1} C_t \Sigma|t-1 \Sigma|t-1 t \mu|t-1 \quad (18.52)$$

$$= (\Sigma|t-1 - K_t C_t \Sigma|t-1) \Sigma|t-1 t \mu|t-1 \quad (18.53)$$

$$= \mu|t-1 - K_t C_t \mu|t-1 \quad (18.54)$$

Putting the two together we get

$$\mu_t = \mu|t-1 + K_t (y_t - C_t \mu|t-1) \quad (18.55)$$

### 18.3.2 The Kalman smoothing algorithm

In Section 18.3.1, we described the Kalman filter, which sequentially computes  $p(z_t|y_{1:t})$  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  $p(z_t|y_{1:T})$ . 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 forwardsbackwards 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  $p(zT|y1:T)$ . Now we work backwards, from right to left, sending information from the future back to the past, and then 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

$$p(zt|y1:T) = N(\mu_t|T, \Sigma_t|T) \quad (18.56)$$

$$\mu_t|T = \mu_t|t + J_t(\mu_{t+1}|T - \mu_{t+1}|t) \quad (18.57)$$

$$\Sigma_t|T = \Sigma_t|t + J_t(\Sigma_{t+1}|T - \Sigma_{t+1}|t)J_t^T \quad (18.58)$$

$$J_t = \Sigma_t|t A T t+1 \Sigma_{t-1}^{-1} t+1 |t \quad (18.59)$$

where  $J_t$  is the backwards Kalman gain matrix. The algorithm can be initialized from  $\mu_T|T$  and  $\Sigma_T|T$  from the Kalman filter. Note that this backwards pass does not need access to the data, that is, it does not need  $y1:T$ . 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  $z_t$  is independent of future data,  $y_{t+1:T}$ , as long as  $z_{t+1}$  is known. Of course,  $z_{t+1}$  is not known, but we have a distribution over it. So we condition on  $z_{t+1}$  and then integrate it out, as follows.

$$p(zt|y1:T) = p(zt|y1:T, z_{t+1})p(z_{t+1}|y1:T)dz_{t+1} \quad (18.60)$$

$$= p(zt|y1:t, y_{t+1:T}, z_{t+1})p(z_{t+1}|y1:T)dz_{t+1} \quad (18.61)$$

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

$$p(z_{t+1}|y1:T) = N(z_{t+1}|\mu_{t+1}|T, \Sigma_{t+1}|T) \quad (18.62)$$

The question is: how do we perform the integration? First,

we compute the filtered two-slice distribution  $p(z_t, z_{t+1}|y_1:t)$  as follows:

$$p(z_t, z_{t+1}|y_1:t) = N(z_t|z_t + \mu_t|t, \Sigma_t|t) \Sigma_t|t A_t^T t + 1 \\ A_t + 1 \Sigma_t|t \Sigma_t + 1|t \quad (18.63)$$

Now we use Gaussian conditioning to compute  $p(z_t|z_{t+1}, y_1:t)$  as follows:

$$p(z_t|z_{t+1}, y_1:t) = N(z_t|\mu_t|t + J_t(z_{t+1} - \mu_{t+1}|t), \Sigma_t|t - J_t \Sigma_t + 1|t J_t^T t) \quad (18.64)$$

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

$$\mu_t|T = E[z_t|z_{t+1}, y_1:T] \quad (18.65)$$

$$= E[z_t|z_{t+1}, y_1:t] \quad (18.66)$$

$$= E[\mu_t|t + J_t(z_{t+1} - \mu_{t+1}|t)] \quad (18.67)$$

$$= \mu_t|t + J_t(\mu_{t+1}|T - \mu_{t+1}|t) \quad (18.68)$$

Now the covariance:

$$\Sigma_t|T = \text{cov}[z_t|z_{t+1}, y_1:T] \quad (18.69)$$

$$= \text{cov}[E[z_t|z_{t+1}, y_1:t] | y_1:T] + E[\text{cov}[z_t|z_{t+1}, y_1:t] | y_1:T] \quad (18.70)$$

$$= \text{cov}[\mu_t|t + J_t(z_{t+1} - \mu_{t+1}|t) | y_1:T] + E[\Sigma_t|t - J_t \Sigma_t + 1|t J_t^T t | y_1:T] \quad (18.71)$$

$$= J_t \text{cov}[z_{t+1} - \mu_{t+1}|t | y_1:T] J_t^T t + \Sigma_t|t - J_t \Sigma_t + 1|t J_t^T t \quad (18.72)$$

$$= J_t \Sigma_t + 1|T J_t^T t + \Sigma_t|t - J_t \Sigma_t + 1|t J_t^T t \quad (18.73)$$

$$= \Sigma_t|t + J_t(\Sigma_t + 1|T - \Sigma_t + 1|t) J_t^T t \quad (18.74)$$

The algorithm can be initialized from  $\mu_T|T$  and  $\Sigma_T|T$  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

$$p(z_t|y_1:T) = p(z_t|y_1:t, z_{t+1}) p(z_{t+1}|y_1:T) dz_{t+1} \quad (18.75)$$

$$= p(z_t, z_{t+1}|y_1:t) p(z_{t+1}|y_1:T) p(z_{t+1}|y_1:t) dz_{t+1} \quad (18.76)$$

Now

$$p(z_{t+1}|y_1:T) = p(y_{t+1:T}|z_{t+1}, y_{1:t})p(z_{t+1}|y_{1:t}) p(y_{t+1:T}|y_{1:t}) \quad (18.77)$$

so

$$p(z_{t+1}|y_1:T) p(z_{t+1}|y_{1:t}) = p(z_{t+1}|y_{1:t})p(y_{t+1:T}|z_{t+1})$$

$$p(z_{t+1}|y_{1:t})p(y_{t+1:T}|y_{1:t}) \propto p(y_{t+1:T}|z_{t+1}) \quad (18.78)$$

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 integrate to 1 over  $z_t$  – in fact, it may not always be possible to represent  $p(y_{t+1:T}|z_{t+1})$  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  $p(z_t|y_{1:t})$  in the forwards pass as usual, and the filtered posterior  $p(z_t|y_{t+1:T})$  in the backwards pass. These can then be easily combined to compute  $p(z_t|y_1:T)$ . See (Kitagawa 2004; Briers et al. 2010) for details.

已思考 39s

下面先把你提供的段落逐句完整翻译（不遗漏），然后给出通俗且详尽的解释，最后总结要点。其中个别地方原文有轻微残缺或排版小误差，我会在解释处备注不影响理解的修正/补充。

## 翻译

### 18.3 线性-高斯 SSM (LG-SSM) 的推断

在本节中，我们讨论在 LG-SSM 模型中的精确推断。我们先考虑在线情形，它与 HMM 的前向算法类似。然后我们考虑离线情形，它与 HMM 的前向-后向算法类似。

#### 18.3.1 卡尔曼滤波算法

卡尔曼滤波是一种对线性-高斯状态空间模型进行精确贝叶斯滤波的算法。我们将在时刻  $t$  以

(注：原文此处省略了具体记法；按照上下文，一般写作  $p(z_t | y_{1:t}) = N(z_t | \mu_{t|t}, \Sigma_{t|t})$ )

来表示边缘后验。由于一切都是高斯分布，我们可以如下面所述，对**预测与更新**两步给出**闭式计算**。所得算法就是第 17.4.2 节 HMM 滤波在高斯情形下的对应物。

### 18.3.1.1 预测步

预测步很容易推出：

$$p(z_t | y_{1:t-1}, u_{1:t}) = \int N(z_t | A_t z_{t-1} + B_t u_t, Q_t) N(z_{t-1} | \mu_{t-1}, \Sigma_{t-1}) dz_{t-1} \quad (18.25)$$

$$= N(z_t | \mu_{t|t-1}, \Sigma_{t|t-1}) \quad (18.26)$$

$$\mu_{t|t-1} = A_t \mu_{t-1} + B_t u_t \quad (18.27)$$

$$\Sigma_{t|t-1} = A_t \Sigma_{t-1} A_t^\top + Q_t \quad (18.28)$$

### 18.3.1.2 观测（测量）步

测量步可用贝叶斯公式计算如下：

$$p(z_t | y_t, y_{1:t-1}, u_{1:t}) \propto p(y_t | z_t, u_t) p(z_t | y_{1:t-1}, u_{1:t}) \quad (18.29)$$

在第 18.3.1.6 节我们将证明，它等价于

$$p(z_t | y_{1:t}, u_t) = N(z_t | \mu_t, \Sigma_t) \quad (18.30)$$

$$\mu_t = \mu_{t|t-1} + K_t r_t \quad (18.31)$$

$$\Sigma_t = (I - K_t C_t) \Sigma_{t|t-1} \quad (18.32)$$

其中  $r_t$  为**残差/创新 (innovation)**，即“预测观测”和“实际观测”的差：

$$r_t = y_t - \hat{y}_t \quad (18.33)$$

$$\hat{y}_t = E[y_t | y_{1:t-1}, u_{1:t}] = C_t \mu_{t|t-1} + D_t u_t \quad (18.34)$$

而  $K_t$  是**卡尔曼增益矩阵**，为

$$K_t = \Sigma_{t|t-1} C_t^\top S_t^{-1} \quad (18.35)$$

其中

$$S_t = \text{cov}[r_t | y_{1:t-1}, u_{1:t}] \quad (18.36)$$

$$= E[(C_t z_t + \delta_t - \hat{y}_t)(C_t z_t + \delta_t - \hat{y}_t)^\top | y_{1:t-1}, u_{1:t}] \quad (18.37)$$

$$= C_t \Sigma_{t|t-1} C_t^\top + R_t \quad (18.38)$$

其中  $\delta_t \sim N(0, R_t)$  是与所有其他噪声源独立的观测噪声项。注意，使用**矩阵求逆引理**，卡尔曼增益也可写成

$$K_t = \Sigma_{t|t-1} C^\top (C \Sigma_{t|t-1} C^\top + R)^{-1} = (\Sigma_{t|t-1}^{-1} + C^\top R^{-1} C)^{-1} C^\top R^{-1} \quad (18.39)$$

现在我们已经拥有实现该算法所需的一切量；Matlab 代码见 `kalmanFilter`。

我们来理解这些等式，特别是均值更新式  $\mu_t = \mu_{t|t-1} + K_t r_t$ 。它表示：**新均值=旧均值+修正量**，修正量等于卡尔曼增益乘以误差信号  $r_t$ 。对误差信号加多少权，取决于  $K_t$ 。若  $C_t = I$ ，则  $K_t = \Sigma_{t|t-1} S_t^{-1}$ ，它是“来自动态模型的先验协方差”与“测量误差协方差”的比值。如果我们有**很强的先验且/或很嘈杂的传感器**，那么  $|K_t|$  会小，我们对修正项赋予很小权重；反之，若先验很弱且/或传感器精度高， $|K_t|$  就大，我们对修正项赋予很大权重。

### 18.3.1.3 边际似然

作为该算法的副产品，我们还可以计算序列的对数似然：

$$\log p(y_{1:T} | u_{1:T}) = \sum_t \log p(y_t | y_{1:t-1}, u_{1:t}) \quad (18.40)$$

其中

$$p(y_t | y_{1:t-1}, u_{1:t}) = N(y_t | C_t \mu_{t|t-1}, S_t). \quad (18.41)$$

### 18.3.1.4 后验预测

对观测的一步后验预测密度可计算为

$$p(y_t | y_{1:t-1}, u_{1:t}) = \int N(y_t | Cz_t, R) N(z_t | \mu_{t|t-1}, \Sigma_{t|t-1}) dz_t \quad (18.42)$$

$$= N(y_t | C\mu_{t|t-1}, C\Sigma_{t|t-1}C^\top + R). \quad (18.43)$$

这对时间序列预测很有用。

### 18.3.1.5 计算问题

卡尔曼滤波有两类主要开销：

- 计算卡尔曼增益  $K_t$  需要矩阵求逆，时间复杂度  $O(|y_t|^3)$ ；
- 计算  $\Sigma_t$  需要矩阵-矩阵乘，时间复杂度  $O(|z_t|^2)$ 。

在某些应用（如机器人建图）中， $|z_t| \gg |y_t|$ ，因此后者占主导。不过在这种情形下，有时可以使用**稀疏近似**（见 Thrun 等，2006）。在  $|y_t| \gg |z_t|$  的情形，我们可以**预先计算**  $K_t$ ，因为令人惊讶的是，它**不依赖于实际观测  $y_{1:t}$** （这是线性高斯系统特有的性质）。

用于更新  $\Sigma_t$  的迭代方程称为**Riccati 方程**；对**时不变系统**（即  $\theta_t = \theta$ ），它们会收敛到一个**定常点**。随后可用该**稳态解**取代逐时刻的增益矩阵。

在实践中，为了**数值稳定性**，应采用更精细的实现。一个做法是**信息滤波器**（information filter），它递归更新高斯的**规范参数**  $\Lambda_t = \Sigma_t^{-1}$  与  $\eta_t = \Lambda_t \mu_t$ ，而不是“矩（mean/cov）参数”。另一个做法是**平方根滤波器**（square-root filter），它使

用  $\Sigma_t$  的 Cholesky 分解或  $U_t D_t U_t^\top$  分解，比直接更新  $\Sigma_t$  稳定得多。更多细节见 <http://www.cs.unc.edu/~welch/kalman/man/> 以及 (Simon, 2006) 等书。

### 18.3.1.6 推导

现在我们来推导卡尔曼滤波方程。为简化记号，忽略输入项  $u_{1:t}$ 。由**高斯的贝叶斯法则**（式 4.125），后验精度满足

$$\Sigma_t^{-1} = \Sigma_{t|t-1}^{-1} + C_t^\top R_t^{-1} C_t. \quad (18.44)$$

由**矩阵求逆引理**（式 4.106），得

$$\Sigma_t = \Sigma_{t|t-1} - \Sigma_{t|t-1} C_t^\top (R_t + C_t \Sigma_{t|t-1} C_t^\top)^{-1} C_t \Sigma_{t|t-1} \quad (18.45)$$

$$= (I - K_t C_t) \Sigma_{t|t-1}. \quad (18.46)$$

由**高斯的贝叶斯法则**（式 4.125），后验均值为

$$\mu_t = \Sigma_t C_t^\top R_t^{-1} y_t + \Sigma_t \Sigma_{t|t-1}^{-1} \mu_{t|t-1}. \quad (18.47)$$

我们把它化为先前给出的形式。对式 (18.47) 的第一项应用**第二个矩阵求逆引理**（式 4.107）：

$$\Sigma_t C_t^\top R_t^{-1} y_t = (\Sigma_{t|t-1}^{-1} + C_t^\top R_t^{-1} C_t)^{-1} C_t^\top R_t^{-1} y_t \quad (18.48)$$

$$= \Sigma_{t|t-1} C_t^\top (R_t + C_t \Sigma_{t|t-1} C_t^\top)^{-1} y_t = K_t y_t. \quad (18.49)$$

对式 (18.47) 的第二项应用**矩阵求逆引理**（式 4.106）：

$$\Sigma_t \Sigma_{t|t-1}^{-1} \mu_{t|t-1} \quad (18.50)$$

$$= (\Sigma_{t|t-1}^{-1} + C_t^\top R_t^{-1} C_t)^{-1} \Sigma_{t|t-1}^{-1} \mu_{t|t-1} \quad (18.51)$$

$$= (\Sigma_{t|t-1} - \Sigma_{t|t-1} C_t^\top (R_t + C_t \Sigma_{t|t-1} C_t^\top)^{-1} C_t \Sigma_{t|t-1}) \Sigma_{t|t-1}^{-1} \mu_{t|t-1} \quad (18.52)$$

$$= (\Sigma_{t|t-1} - K_t C_t \Sigma_{t|t-1}) \Sigma_{t|t-1}^{-1} \mu_{t|t-1} \quad (18.53)$$

$$= \mu_{t|t-1} - K_t C_t \mu_{t|t-1}. \quad (18.54)$$

合并两项得到

$$\mu_t = \mu_{t|t-1} + K_t (y_t - C_t \mu_{t|t-1}). \quad (18.55)$$

### 18.3.2 卡尔曼平滑算法

在第 18.3.1 节里，我们描述了卡尔曼滤波，它对每个  $t$  顺序地计算  $p(z_t | y_{1:t})$ ，适合**在线**推断（如跟踪）。然而在**离线**情形，我们可以等所有数据到齐，再计算  $p(z_t | y_{1:T})$ 。由于结合了过去与未来的数据，我们的不确定性将显著降低。这在图 18.1(c) 中有所体现：与滤

波轨迹相比，平滑轨迹的后验协方差椭球更小。（在轨迹的开始与结束处椭球更大，因为边界附近可借的信息邻居较少。）

下面解释如何计算平滑估计，使用一个叫 **RTS 平滑器** 的算法（以其发明者 Rauch、Tung、Striebel 命名；Rauch 等，1965），也称**卡尔曼平滑算法**。该算法与 HMM 的前向-后向算法类似，但也有一些小差别，见下。

### 18.3.2.1 算法

可以把卡尔曼滤波看作图上的**消息传递**，从左到右。当消息传到图的末端，我们已经计算出  $p(z_T \mid y_{1:T})$ 。现在我们**反向**（从右到左）工作，把来自**未来**的信息传回**过去**，再把两种信息结合。问题是：后向等式如何计算？我们先给出等式，然后给出推导。

$$p(z_t \mid y_{1:T}) = N(\mu_{t|T}, \Sigma_{t|T}) \quad (18.56)$$

$$\mu_{t|T} = \mu_{t|t} + J_t(\mu_{t+1|T} - \mu_{t+1|t}) \quad (18.57)$$

$$\Sigma_{t|T} = \Sigma_{t|t} + J_t(\Sigma_{t+1|T} - \Sigma_{t+1|t})J_t^\top \quad (18.58)$$

$$J_t = \Sigma_{t|t} A_{t+1}^\top \Sigma_{t+1|t}^{-1} \quad (18.59)$$

其中  $J_t$  是**后向卡尔曼增益**。该算法用滤波得到的  $\mu_{T|T}, \Sigma_{T|T}$  作为初始化。注意这一步**不需要访问原始数据**  $y_{1:T}$ ，因此可以丢掉（可能维度很高的）观测向量，只保留**滤波信念状态**，通常更省内存。

### 18.3.2.2 推导

我们按 (Jordan, 2007, §15.7) 来推导卡尔曼平滑器。关键是利用**马尔可夫性质**：给定  $z_{t+1}$ ， $z_t$  与未来数据  $y_{t+1:T}$  独立。当然  $z_{t+1}$  不已知，但我们有它的分布。于是对  $z_{t+1}$  条件化再积分掉：

$$p(z_t \mid y_{1:T}) = \int p(z_t \mid y_{1:T}, z_{t+1}) p(z_{t+1} \mid y_{1:T}) dz_{t+1} \quad (18.60)$$

$$= \int p(z_t \mid y_{1:t}, y_{t+1:T}, z_{t+1}) p(z_{t+1} \mid y_{1:T}) dz_{t+1} \quad (18.61)$$

归纳假设我们已算出  $t+1$  的平滑分布：

$$p(z_{t+1} \mid y_{1:T}) = N(z_{t+1} \mid \mu_{t+1|T}, \Sigma_{t+1|T}). \quad (18.62)$$

问题是**如何进行积分**。首先计算滤波得到的**两片分布**  $p(z_t, z_{t+1} \mid y_{1:t})$ ：

$$p(z_t, z_{t+1} \mid y_{1:t}) = N\left(\begin{bmatrix} z_t \\ z_{t+1} \end{bmatrix} \mid \begin{bmatrix} \mu_{t|t} \\ \mu_{t+1|t} \end{bmatrix}, \begin{bmatrix} \Sigma_{t|t} & \Sigma_{t|t} A_{t+1}^\top \Sigma_{t+1|t}^{-1} \\ A_{t+1} \Sigma_{t|t} & \Sigma_{t+1|t} \end{bmatrix}\right). \quad (18.63)$$

然后对高斯作条件化，得到

$$p(z_t \mid z_{t+1}, y_{1:t}) = N(z_t \mid \mu_{t|t} + J_t(z_{t+1} - \mu_{t+1|t}), \Sigma_{t|t} - J_t \Sigma_{t+1|t} J_t^\top). \quad (18.64)$$

利用**迭代期望/方差定理**计算平滑分布。先是均值：

$$\mu_{t|T} = E[E[z_t \mid z_{t+1}, y_{1:T}] \mid y_{1:T}] \quad (18.65)$$

$$= E[E[z_t \mid z_{t+1}, y_{1:t}] \mid y_{1:T}] \quad (18.66)$$

$$= E[\mu_{t|t} + J_t(z_{t+1} - \mu_{t+1|t}) \mid y_{1:T}] \quad (18.67)$$

$$= \mu_{t|t} + J_t(\mu_{t+1|T} - \mu_{t+1|t}). \quad (18.68)$$

再是协方差：

$$\Sigma_{t|T} = \text{cov}(E[z_t \mid z_{t+1}, y_{1:T}] \mid y_{1:T}) + E[\text{cov}(z_t \mid z_{t+1}, y_{1:T}) \mid y_{1:T}] \quad (18.69)$$

$$= \text{cov}(E[z_t \mid z_{t+1}, y_{1:t}] \mid y_{1:T}) + E[\text{cov}(z_t \mid z_{t+1}, y_{1:t}) \mid y_{1:T}] \quad (18.70)$$

$$= \text{cov}(\mu_{t|t} + J_t(z_{t+1} - \mu_{t+1|t}) \mid y_{1:T}) + E[\Sigma_{t|t} - J_t \Sigma_{t+1|t} J_t^\top \mid y_{1:T}] \quad (18.71)$$

$$= J_t \text{cov}(z_{t+1} - \mu_{t+1|t} \mid y_{1:T}) J_t^\top + \Sigma_{t|t} - J_t \Sigma_{t+1|t} J_t^\top \quad (18.72)$$

$$= J_t \Sigma_{t+1|T} J_t^\top + \Sigma_{t|t} - J_t \Sigma_{t+1|t} J_t^\top \quad (18.73)$$

$$= \Sigma_{t|t} + J_t(\Sigma_{t+1|T} - \Sigma_{t+1|t}) J_t^\top. \quad (18.74)$$

算法用滤波最后一步的  $\mu_{T|T}, \Sigma_{T|T}$  初始化。

### 18.3.2.3 与 HMM 前向–后向的比较

注意，对 LDS 的前向与后向两次传递，我们始终处理的是**归一化的分布**（分别条件化在过去数据或全部数据上），且**后向**依赖于**前向**的结果。这不同于 HMM 的常见表述：HMM 的后向传递可以**独立于前向**计算（见 17.4.3）。事实上，我们可以把卡尔曼平滑改写成更像 HMM 前向–后向的形式。具体地，

$$p(z_t \mid y_{1:T}) = \int p(z_t \mid y_{1:t}, z_{t+1}) p(z_{t+1} \mid y_{1:T}) dz_{t+1} \quad (18.75)$$

$$= \int \frac{p(z_t, z_{t+1} \mid y_{1:t}) p(z_{t+1} \mid y_{1:T})}{p(z_{t+1} \mid y_{1:t})} dz_{t+1}. \quad (18.76)$$

而

$$p(z_{t+1} \mid y_{1:T}) = \frac{p(y_{t+1:T} \mid z_{t+1}, y_{1:t}) p(z_{t+1} \mid y_{1:t})}{p(y_{t+1:T} \mid y_{1:t})} \quad (18.77)$$

因此

$$\frac{p(z_{t+1} \mid y_{1:T})}{p(z_{t+1} \mid y_{1:t})} = \frac{p(z_{t+1} \mid y_{1:t}) p(y_{t+1:T} \mid z_{t+1})}{p(z_{t+1} \mid y_{1:t}) p(y_{t+1:T} \mid y_{1:t})} \propto p(y_{t+1:T} \mid z_{t+1}), \quad (18.78)$$

这就是**未来数据的条件似然**。这个后向消息可**独立于前向消息计算**。然而这种方式有若干缺点：(1) 需要访问原始观测序列；(2) 后向消息是**似然而非后验**，因此对  $z_t$  不一定能积分为 1——事实上， $p(y_{t+1:T} \mid z_{t+1})$  并不总能表示成协方差正定的高斯（这个问题在 HMM 使用的**离散状态空间里不会出现**）；(3) 当无法精确推断时，更合理的做法是近似**平滑分布**而不是后向似然项（见 22.5 节）。还有一个变体叫**双滤波平滑**（two-filter smoothing）：前向照常算  $p(z_t \mid y_{1:t})$ ，后向算“反向滤波后验” $p(z_t \mid y_{t+1:T})$ ，二者再容易地结合得到  $p(z_t \mid y_{1:T})$ 。细节见 (Kitagawa, 2004; Briers 等, 2010)。

---

## 通俗解释（一步步讲清）

### 1) 卡尔曼滤波 = “预测一步 + 用新数据校正”

- **预测** (18.27–18.28)：用系统方程把上一步的估计  $(\mu_{t-1}, \Sigma_{t-1})$  推到这一刻，得到“先验” $(\mu_{t|t-1}, \Sigma_{t|t-1})$ 。  
直觉：根据物理规律/趋势，猜一下“此刻大概在哪”，并给出不确定性。
- **更新** (18.31–18.38)：拿到新观测  $y_t$ ，比较“预测的观测  $\hat{y}_t$ ”与“真实观测  $y_t$ ”的差  $r_t$ ；用**卡尔曼增益**  $K_t$  来决定修正多少。
  - $S_t$ : **创新协方差**，是“预测误差会有多大”的量；
  - $K_t$ : **相信观测 vs 相信预测的权衡器**。传感器噪声  $R_t$  大  $\Rightarrow K_t$  小  $\Rightarrow$  更信预测；模型不确定性  $\Sigma_{t|t-1}$  大  $\Rightarrow K_t$  大  $\Rightarrow$  更信观测。
  - 更新后的均值/协方差为  $\mu_t, \Sigma_t$ 。

小口诀：**新 = 旧 + 增益 × (观测 – 预测观测)**。

### 2) 顺带算两件实用的东西

- **边际似然** (18.40–18.41)：每步都能算  $p(y_t \mid y_{1:t-1})$ ，乘起来就是整段数据在模型下的似然  $\rightarrow$  可用于**参数估计/模型选择**。
- **一步后验预测** (18.42–18.43)：直接给出下一步观测的预测分布  $\rightarrow$  做**时间序列预测**。

### 3) 计算与工程要点 (18.3.1.5)

- 大头开销：求逆  $O(|y|^3)$  和**矩阵乘**  $O(|z|^2)$ 。
- $|y| \gg |z|$  时， $K_t$  与  $y$  无关可**预计算**； $|z| \gg |y|$  时可用**稀疏技巧**。
- **稳态解**：时不变系统里 Riccati 方程常收敛到固定点，可用**固定的增益**。
- **数值稳定**：信息滤波（更新  $\Lambda, \eta$ ）或平方根滤波（Cholesky/UDU）更稳。

## 4) 卡尔曼平滑 = “把未来的信息倒回去用”

- 滤波只用到  $t$  为止的数据，得到  $p(z_t | y_{1:t})$ 。
- 平滑在所有数据到齐后，用**后向增益**  $J_t$  把“未来对现在的修正”加回来：
  - $\mu_{t|T} = \mu_{t|t} + J_t(\mu_{t+1|T} - \mu_{t+1|t})$
  - $\Sigma_{t|T} = \Sigma_{t|t} + J_t(\Sigma_{t+1|T} - \Sigma_{t+1|t})J_t^\top$
- 直觉：若下一时刻**最终**（用全数据）看起来比“当时的预测”更靠某个方向，那么现在这刻也应朝那个方向**小幅回调**；协方差亦相应缩小。
- **边界效应**：开头/结尾邻居少，平滑后不确定性仍相对大。

## 5) 与 HMM 前向-后向的关系 (18.3.2.3)

- HMM 后向消息可**独立**计算；LG-SSM 的常规 RTS 写法里，后向依赖前向（我们用前向得到的  $\mu_{t|t}, \Sigma_{t|t}, \mu_{t+1|t}, \Sigma_{t+1|t}$ ）。
  - 也能把平滑写成“像 HMM 那样”的形式，但会遇到：需要原始观测、后向是**似然**（不一定归一）、在连续高斯里可能出现协方差非正定等问题。实践中仍以直接近似**平滑分布**为宜。另有**双滤波平滑**的变体。
- 

## 要点总结

- **卡尔曼滤波 (在线)**：
  - 预测： $\mu_{t|t-1} = A_t \mu_{t-1} + B_t u_t, \Sigma_{t|t-1} = A_t \Sigma_{t-1} A_t^\top + Q_t$ 。
  - 更新： $\hat{y}_t = C_t \mu_{t|t-1} + D_t u_t, r_t = y_t - \hat{y}_t, S_t = C_t \Sigma_{t|t-1} C_t^\top + R_t, K_t = \Sigma_{t|t-1} C_t^\top S_t^{-1}, \mu_t = \mu_{t|t-1} + K_t r_t, \Sigma_t = (I - K_t C_t) \Sigma_{t|t-1}$ 。
  - 直觉：**新估计=旧估计+“信观测的权重” $\times$  (观测-预测)**。
- **顺带**：每步有  $p(y_t | y_{1:t-1})$ （可做似然/模型选择）；还能给出一步**观测预测**。
- **工程**：关注矩阵求逆/乘的复杂度；利用稀疏、稳态解、信息/平方根滤波**稳又快**。
- **卡尔曼平滑 (离线)**：
  - 后向增益  $J_t = \Sigma_{t|t} A_{t+1}^\top \Sigma_{t+1|t}^{-1}$ 。
  - 平滑均值/协方差：
 
$$\mu_{t|T} = \mu_{t|t} + J_t(\mu_{t+1|T} - \mu_{t+1|t}), \Sigma_{t|T} = \Sigma_{t|t} + J_t(\Sigma_{t+1|T} - \Sigma_{t+1|t})J_t^\top.$$
  - 效果：不确定性更小，轨迹更平滑；边界仍较不确定。

- 与 HMM：思路类比“前向/前向-后向”，但在连续高斯里后向消息的处理方式与离散 HMM 有重要差异。

如果你想，我可以把这些公式改成一页“**抄写卡**”版的伪代码，或用一个小例子（如 1D 温度/2D 小车）跑出滤波和平滑的数值演示，帮你把每一个量 ( $\hat{y}_t, r_t, S_t, K_t$  等) 都看个明白。

这一节主要在讲什么？我不太懂

已思考 23s

不慌～这节（18.3）其实就讲一件大事、两个场景：

## 一句话总览

在线性-高斯状态空间模型（LG-SSM）里，怎么把“到目前为止的观测”或“全程观测”转成对隐藏状态的最优估计。

- 在线（边走边算）：**卡尔曼滤波**
- 离线（事后回看）：**卡尔曼平滑（RTS）**

## 卡尔曼滤波（在线）——每一步都做“预测 + 校正”

把“此刻状态”的不确定性用**高斯分布**表示，只需要跟踪两样：**均值  $\mu$ 、协方差  $\Sigma$** 。

### 1. 预测（基于系统规律）

- 先把上一步的估计推进到现在：
    - 位置/状态的预测： $\mu^t|t-1 = A \mu^{t-1} + B u^t$
    - 不确定性传播： $\Sigma^t|t-1 = A \Sigma^{t-1} A^T + Q$
- 直觉：根据物理/趋势，先“猜一下”现在在哪，同时误差也随之扩大或缩小。

### 2. 校正（用新观测修正）

- 先算出“预测会看到什么”： $\hat{y}^t = C \mu^t|t-1 + D u^t$
- 看真实观测  $y^t$  与预测的差：**残差/创新**  $r^t = y^t - \hat{y}^t$
- 评估这差值应有多大： $S^t = C \Sigma^t|t-1 C^T + R$
- 决定“该信观测多少”：**卡尔曼增益**  $K^t = \Sigma^t|t-1 C^T (S^t)^{-1}$
- 更新：
  - 均值： $\mu^t = \mu^t|t-1 + K^t r^t$

- 协方差:  $\Sigma^t = (I - K^t C) \Sigma^{t-1}$   
直觉口诀: 新 = 旧 + 增益 × (观测 - 预测观测)。
- 观测噪声 R 大 (传感器糙) → K 小 → 更信预测;  
模型不确定  $\Sigma^{t-1}$  大 → K 大 → 更信观测。

## 顺带产物

- 每一步都有  $p(y^t | \text{过去})$  (高斯, 均值  $C\mu^{t-1}$ 、协方差  $S^t$ ) , 把 log 概率加起来就是整段数据的对数似然 (做参数估计/模型比较)。
- 也直接给出一步后验预测分布 (做时间序列预测)。

## 工程要点

- 主要开销: 求逆  $O(|y|^3)$  (算 K) , 矩阵乘  $O(|z|^2)$  (算  $\Sigma$ )。
- 时不变系统时,  $\Sigma$  的递推 (Riccati) 会收敛到稳态, 可用固定增益加速。
- 为数值稳定, 常用信息滤波 (更新  $\Sigma^{-1}$ 、 $\Sigma^{-1}\mu$ ) 或平方根滤波 (Cholesky/UDU 分解)。

## 卡尔曼平滑 (离线) —— 把“未来的信息”倒回去

当所有数据到齐, 再把“未来对现在的纠偏”加回来, 得到更小不确定性的估计。

- 先做一遍前向滤波, 得到每一步的  $\mu^t|t$ 、 $\Sigma^t|t$  以及一步预测  $\mu^{t+1}|t$ 、 $\Sigma^{t+1}|t$ 。
- 然后从 T → 1 反向做:
  - 后向增益:  $J^t = \Sigma^t|t A^T (\Sigma^{t+1}|t)^{-1}$
  - 平滑均值:  $\mu^t|T = \mu^t|t + J^t (\mu^{t+1}|T - \mu^{t+1}|t)$
  - 平滑协方差:  $\Sigma^t|T = \Sigma^t|t + J^t (\Sigma^{t+1}|T - \Sigma^{t+1}|t) (J^t)^T$
- 直觉: 如果下一刻“最终更靠右”, 那现在这刻也应稍微往右修一修; 中间位置受益最大, 首尾因邻居少仍较不确定。

## 跟 HMM 的关系

- 思想类比 HMM 的前向与前向-后向。
- 不同点: LG-SSM 的常用平滑 (RTS) 后向依赖前向的结果; 若硬把后向写成“纯似然消息”, 会遇到归一化/正定性等麻烦, 实务中还是用 RTS 更稳。

## 只记这三句

1. 滤波 = 预测 + 校正, 核心更新:  $\mu \leftarrow \mu + K(\text{观测} - \text{预测观测})$ ,  $\Sigma \leftarrow (I - KC)\Sigma$ 。

2. 平滑 = 把未来的信息倒回去，用后向增益 J 把每一刻再修一遍，不确定性更小。

3. 这些都成立是因为线性 + 高斯，所以整个过程始终保持高斯、能写出闭式公式。

如果你愿意，我可以把滤波/平滑写成一页“伪代码抄写卡”，或用一个小玩具数据跑一下，把  $\mu$ 、 $\Sigma$ 、 $K$ 、 $r$ 、 $S$  每一步的数值都展示出来。

翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

## 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  $C$  and the transition matrix  $A$  are both known and fixed, by definition of the model. In such cases, all that needs to be learned are the noise covariances  $Q$  and  $R$ . (The initial state estimate  $\mu_0$  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  $Q$  and  $R$  offline, using the methods described below, it is also possible to derive a recursive procedure to exactly compute the posterior  $p(z_t, R, Q | y_{1:t})$ , 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  $A$  and  $C$ .

However, in this case we can set  $Q = I$  without loss of generality, since an arbitrary noise covariance can be modeled by appropriately modifying  $A$ . Also, by analogy with factor analysis, we can require  $R$  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  $A$ . 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

$$z_t = A z_{t-1} = U \Lambda t U^{-1} z_{t-1} \quad (18.79)$$

where  $U$  is the matrix of eigenvectors for  $A$ , and  $\Lambda = \text{diag}(\lambda_i)$  contains the eigenvalues. If any  $\lambda_i > 1$ , then for large  $t$ ,  $z_t$  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  $E[z_t] = 0$  for large  $t$ , so the state will return to the origin. Fortunately, when we add noise, the state becomes 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  $z_{t-1} \rightarrow z_t$  and for  $z_t \rightarrow y_t$ . That is, we can estimate  $A$  by solving the least squares problem  $J(A) = \sum_{t=1}^T (z_t - A z_{t-1})^2$ , and similarly for  $C$ . We can estimate the system noise covariance  $Q$  from the residuals in predicting  $z_t$  from  $z_{t-1}$ , and estimate the observation noise covariance  $R$  from the residuals in predicting  $y_t$  from  $z_t$ .

#### 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  $z_t = A z_{t-1}$  and  $y_t = C z_t$ , and hence  $y_t = C A z_{t-1}$ .

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.

已思考 29s

下面先把你给出的段落逐句完整翻译（不遗漏），然后做通俗且详尽的解释，最后给出要点总结。我会在解释里顺带指出一些直觉、常见坑与实务做法，尽量不依赖高等数学。

## 翻译

### 18.4 线性-高斯 SSM (LG-SSM) 的学习 (参数估计)

在本节中，我们简要讨论如何估计 LG-SSM 的参数。在控制理论社群里，这件事被称为系统辨识 (systems identification) (Ljung 1987)。

当把 SSM 用于时间序列预测、以及某些物理状态估计问题时，观测矩阵  $C$  和 状态转移矩阵  $A$  依照模型的定义是已知且固定的。在这种情况下，唯一需要学习的是噪声协方差  $Q$  与  $R$ 。（初始状态估计  $\mu_0$  通常不那么重要，因为它会在若干时间步之后被数据“冲淡”。通过把初始状态协方差设得较大（表示先验较弱）可以强化这一点。）虽然我们可以用下面的方法离线估计  $Q$  与  $R$ ，也可以推导一种递归过程来精确计算后验  $p(\mathbf{z}_t, R, Q \mid y_{1:t})$ ，其形式是正态-逆 Wishart (Normal-inverse-Wishart)；细节见 (West and Harrison 1997; Prado and West 2010)。

#### 18.4.1 可识别性与数值稳定性

在更一般的设定中，如果隐藏状态没有预先指定的含义，我们需要学习  $A$  与  $C$ 。然而在这

种情形下，我们可以令  $Q = I$  而不失一般性，因为任意的噪声协方差都可以通过适当地修改  $A$  来建模。同样地，类比于因子分析，我们可以要求  $R$  为对角阵而不失一般性。这样做能减少自由参数数量，并提升数值稳定性。

另一个有用的约束是对动力学矩阵  $A$  的特征值施加限制。为理解其重要性，考虑没有系统噪声的情形。在这种情况下，时刻  $t$  的隐藏状态为

$$z_t = A^t z_1 = U \Lambda^t U^{-1} z_1 \quad (18.79)$$

其中  $U$  是  $A$  的特征向量矩阵， $\Lambda = \text{diag}(\lambda_i)$  包含特征值。如果存在某个  $\lambda_i > 1$ ，那么当  $t$  很大时， $z_t$  的幅度将会发散。因此，为了保证稳定性，要求所有特征值都小于 1 是有用的 (Siddiqi et al. 2007)。当然，如果所有特征值都小于 1，那么当  $t$  很大时  $E[z_t] = 0$ ，状态会回到原点。幸运的是，当我们加入噪声时，状态将变为非零，因此模型不会退化。

下面我们讨论如何估计参数。不过，为了叙述简洁，**下文不对上述任何约束做强制施加**。

#### 18.4.2 全观测数据下的训练

如果我们能够观察到隐藏状态序列，就可以通过多元线性回归来为  $z_{t-1} \rightarrow z_t$  与  $z_t \rightarrow y_t$  这两条关系求参数的 MLE (甚至完整后验)。也就是说，我们可以通过解最小二乘问题

$$J(A) = \sum_t \|z_t - Az_{t-1}\|^2$$

来估计  $A$ ，对  $C$  也类似。我们可以根据用  $z_{t-1}$  预测  $z_t$  的残差来估计系统噪声协方差  $Q$ ，并根据用  $z_t$  预测  $y_t$  的残差来估计观测噪声协方差  $R$ 。

#### 18.4.3 LG-SSM 的 EM

如果我们只观察到输出序列，可以用 EM 来计算参数的 ML 或 MAP 估计。这个方法在概念上与 HMM 的 Baum–Welch 算法 (17.5 节) 非常相似，只不过 E 步用卡尔曼平滑替代前向–后向，而 M 步的计算也不相同。细节留作习题 18.1。

#### 18.4.4 子空间方法

EM 并不总是给出令人满意的结果，因为它对初始参数很敏感。避免这一点的一种方法是采用称为子空间方法 (subspace method) 的不同途径 (Overschee and Moor 1996; Katayama 2005)。

为理解该方法，先假设没有观测噪声也没有系统噪声。在此情况下，有  $z_t = Az_{t-1}$  且  $y_t = Cz_t$ ，因此  $y_t = CA^{t-1}z_1$ 。于是，所有观测必然由一个  $\dim(z_t)$ -维的线性流形/子空间生成。我们可以用 PCA 来识别这个子空间 (细节见上述参考)。一旦拿到了  $z_t$  的估计，我们就可以把模型当作全观测来拟合。可以直接使用这些估计，也可以用它们来初始化 EM。

### 18.4.5 “拟合” LG-SSM 的贝叶斯方法

除了 EM，还有多种离线贝叶斯替代方案，包括变分 Bayes-EM (Beal 2003; Barber and Chiappa 2007) 和分块 Gibbs 采样 (Carter and Kohn 1994; Cappe et al. 2005; Frühwirth-Schnatter 2007)。贝叶斯方法也可以用于在线学习，如我们在 18.2.3 节讨论的那样。不幸的是，一旦把 SSM 的参数也并入状态空间，模型通常就不再是线性高斯。因此我们必须使用后文将讨论的一些近似的在线推断方法。

---

## 通俗解释（循序渐进）

### 这节的核心问题

怎么学出（估计）LG-SSM 里的参数：

- 系统矩阵  $A$ （状态怎么变）
- 观测矩阵  $C$ （状态如何变成观测）
- 噪声协方差  $Q$ （系统噪声）和  $R$ （观测噪声）
- 以及初始状态分布（通常影响不大，会被数据“冲淡”）

在很多工程/物理场景里，系统结构是已知的（比如牛顿运动、传感器几何）， $A$  和  $C$  直接给定；这时只要学  $Q$ 、 $R$  就够了。直观上：

- $Q$  控制“系统自身乱不乱”（模型外的扰动，如风、摩擦不确定）；
  - $R$  控制“测量抖不抖”（传感器有多吵）；
- 把它们设准了，滤波/预测就会“信该信的”。

此外，文中提到可以递推地得到  $p(z_t, Q, R \mid y_{1:t})$  的闭式后验（正态-逆 Wishart 结构），意思是：当只学协方差、且配合高斯先验/共轭结构时，在线也能“边看数据边更新不确定性”。

### 识别性 (identifiability) 与数值稳定 (18.4.1)

**识别性：**如果我们对“状态”本身不做固定含义（只是一个抽象的潜变量），就可能出现缩放/旋转的歧义：

- 例如把  $z$  换成  $\tilde{z} = Sz$ ，再把  $A$  换成  $\tilde{A} = SAS^{-1}$ ， $C$  换成  $\tilde{C} = CS^{-1}$ ，**模型生成同样的数据**。
- 为了“定标”，可**固定**  $Q = I$ （把随机抖动的单位定下来），并要求  $R$  为**对角**（各观测噪声互不相关）。这会**减少自由度**，不容易跑偏，也更稳。

**稳定性：**看  $A$  的特征值。

- 如果没有系统噪声， $z_t = A^t z_1$ 。特征值  $|\lambda_i| > 1$  会导致  $z_t$  随时间**爆炸**（数值上也会崩）。

- 要稳，常要求谱半径（最大特征值模） $< 1$ 。这样状态不会失控，模型也更好算。
- 加上噪声后，状态均值虽趋近 0，但由于噪声，状态本身不为 0（不退化），这是正常的“随机平衡”。

小结：为了好学、好用、好算，常做三件事：**定标** ( $Q = I$ ) 、**去相关** ( $R$  对角) 、**保稳定** (谱半径 $<1$ ) 。

## 全观测时最好学 (18.4.2)

如果连隐藏状态  $z_t$  都能看到（比如你有一个几乎无噪的高精度定位器），学习就变成两次线性回归：

1. 用  $z_{t-1}$  回归  $z_t \Rightarrow$  学  $A$ ；残差的协方差  $\Rightarrow$  学  $Q$ 。
2. 用  $z_t$  回归  $y_t \Rightarrow$  学  $C$ ；残差的协方差  $\Rightarrow$  学  $R$ 。

这和普通最小二乘一模一样，稳、快、直观。（原文里目标函数的求和指标写得很简略，实作中是对有效的  $t = 2, \dots, T$  求和。）

## 只有输出可见时：EM (18.4.3)

多数现实里我们看不到  $z_t$ ，只有  $y_t$ 。怎么办？用 EM：

- **E 步**：跑一遍卡尔曼滤波 + 平滑，拿到对每一时刻的“最佳  $z_t$  估计”及其二阶矩（形象地说：把看不见的东西“补齐成期望”）。
- **M 步**：把 E 步的“期望充当数据”，重新解“回归式”来更新  $A, C, Q, R$ 。
- 交替迭代，直到收敛。

直觉：EM 在“猜  $z$ ”和“用  $z$  学参数”之间来回，逐步把两者磨合到一致。**缺点**：对初值敏感，容易卡在局部最优。

常见的 M 步闭式（作为理解，不是原文强制给出）：

$$\begin{aligned} A &\leftarrow \left( \sum_t E[z_t z_{t-1}^\top] \right) \left( \sum_t E[z_{t-1} z_{t-1}^\top] \right)^{-1} \\ C &\leftarrow \left( \sum_t y_t E[z_t]^\top \right) \left( \sum_t E[z_t z_t^\top] \right)^{-1} \\ Q &\leftarrow \frac{1}{T-1} \sum_t (E[z_t z_t^\top] - A E[z_t z_{t-1}^\top]^\top) \\ R &\leftarrow \frac{1}{T} \sum_t (y_t y_t^\top - C E[z_t] y_t^\top) \end{aligned}$$

（真实实现里要用到平滑得到的  $E[z_t]$ ,  $E[z_t z_t^\top]$ ,  $E[z_t z_{t-1}^\top]$  等“充分统计量”。）

## 子空间方法 (18.4.4)

**为什么要它：**EM 可能对初始值敏感、慢。

**怎么做**（先讲无噪声直觉）：

- 若  $y_t = CA^{t-1}z_1$  且无噪声，那么所有  $y_t$  都落在一个**低维子空间里**（维度=状态维数）。
- 用 PCA 找到这个子空间，等于估计出了隐藏的“状态轨迹” $\{z_t\}$ 。
- 接着就像“全观测”一样去学  $A, C, Q, R$ 。
- 现实中有噪声，但这仍是**很好的初始化**（先用 PCA 找一个不错的起点，再丢给 EM 微调）。

## 贝叶斯方法 (18.4.5)

离线贝叶斯替代 EM：

- **变分 Bayes (VB-EM)**：用可因式化的近似后验来最大化 ELBO。
- **分块 Gibbs**：交替采样隐藏状态轨迹、参数等，得到全后验的不确定性。

**在线贝叶斯**：也可用（如 18.2.3 的递归最小二乘/共轭更新思路）。

**注意：**一旦把参数也当作状态并联动估计，模型通常**不再是线性高斯**（因为参数乘以状态引入了非线性）；这时就要用**近似推断**（如粒子法、扩展/无迹卡尔曼、变分近似等）。

## 要点总结

- **已知  $A, C$  的工程场景：**主要学  $Q$ （系统抖动）和  $R$ （传感器噪声）。初始均值影响小，可把初始协方差设大以“快忘记”。
- **识别性 & 稳定性：**
  - 为定标与简化，常取  $Q = I$ 、 $R$  对角。
  - 为避免发散，限制  $A$  的**谱半径**  $< 1$ （特征值模  $< 1$ ）。
- **全观测最好学：**看到  $z_t$  时，一切变成两次最小二乘 + 残差协方差估计。
- **只见输出就用 EM：** E 步=卡尔曼平滑估“看不见的  $z$ ”；M 步=像回归那样更新  $A, C, Q, R$ 。对初值敏感，可用子空间（PCA）**做好初始化**。
- **子空间方法：**把观测看作落在低维子空间，用 PCA 找到“状态子空间”，再拟合或初始化 EM。
- **贝叶斯路线：**VB、Gibbs 等给出**后验不确定性**；把参数并入状态后不再 LG，需要**近似在线/离线推断**。

如果你愿意，我可以用一个**1 维信号 + 简单 LG-SSM**的小例子，演示：

1. 仅学  $Q, R$  时的效果；

2. 用 EM 从随机初值学  $A, C, Q, R$ ;
3. 用 PCA 子空间做初始化再跑 EM 的收敛对比。

这一节主要在讲什么？我不太懂

已思考 9s

不慌～这节（18.4：LG-SSM 的学习/参数估计）说的就是：**怎么把线性-高斯状态空间模型里的“未知旋钮”学出来**。这些旋钮包括：系统矩阵  $A$ 、观测矩阵  $C$ 、系统噪声协方差  $Q$ 、观测噪声协方差  $R$ （以及初始状态分布）。

## 用大白话说它讲了什么

1. **有些场景  $A, C$  已知**（物理/工程里模型结构给定）：

只要学**噪声强度**  $Q$ （系统有多“晃”）和  $R$ （传感器有多“吵”）。初始均值影响小，初始协方差设大，数据会很快“冲淡”它。还可以用共轭先验做**递推更新**（ $Q, R$  的后验是 Normal-inverse-Wishart）。

2. **可识别性 & 稳定性（避免歧义和发散）**：

- 为了“定标”和简化，常**固定**  $Q = I$ ，并让  $R$  **对角**（观测噪声互不相关）——不损失一般性，还更稳。
- 限制  $A$  的特征值**小于** 1，否则状态会越算越大（数值发散）。加了噪声后不会退化到全零，放心。

3. **如果能“看见”隐藏状态  $z_t$ （全观测）**：

学习=两次最小二乘：

- $z_{t-1} \rightarrow z_t$  学  $A$ ；残差协方差给  $Q$ 。
- $z_t \rightarrow y_t$  学  $C$ ；残差协方差给  $R$ 。

4. **只有输出  $y_t$  可见时**：用 EM 学 ML/MAP 参数。

- **E 步**：卡尔曼平滑估“看不见的  $z_t$ ”及其二阶矩。
- **M 步**：像回归那样更新  $A, C, Q, R$ 。
- 缺点：对**初值敏感**。

5. **子空间方法（给 EM 打好底）**：

把无噪声直觉： $y_t = CA^{t-1}z_1 \Rightarrow$ 所有观测落在一个低维**子空间**。先用 PCA 找这个子空间，得到  $\{z_t\}$  的好初值，再当“全观测”拟合或用来**初始化** EM。初值好，EM 就更稳。

## 6. 贝叶斯做法（离线/在线）：

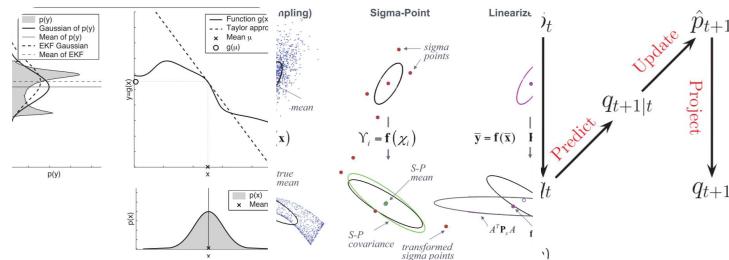
变分 Bayes、Gibbs 等能给出**参数不确定性**；但一旦把“参数也当作状态”一起估计，模型**不再线性-高斯**，需要用**近似推断**（粒子/扩展卡尔曼/无迹卡尔曼/变分等）。

## 什么时候用哪招（速配表）

- 已知  $A, C$ : 学  $Q, R$  (可离线估计或递推共轭更新)。
- 能拿到  $z_t$ : 直接最小二乘 + 残差协方差。
- 只有  $y_t$ : 先子空间/PCA 初始化 → 再EM。
- 要不确定性/贝叶斯口味: VB 或 Gibbs; 在线时用近似方法。

## 只记三句话

- 这节在教你**怎么估计**  $A, C, Q, R$ 。
- 为了稳和可识别，常用  $Q = I$ 、 $R$  对角、谱半径( $A$ )<1。
- 可见**  $z$  **用最小二乘**；**只见**  $y$  **用 EM (配子空间初始化)**；**要全后验用贝叶斯近似**。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

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 nonlinear function induces a complex distribution. The solid line is the best Gaussian approximation to this; the dotted line is the EKF approximation to this.

Figure 18.10 An example of the unscented transform in two dimensions.

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).

## 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

$$z_t = g(u_t, z_{t-1}) + N(0, Q_t) \quad (18.80)$$

$$y_t = h(z_t) + N(0, R_t) \quad (18.81)$$

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 ( $Q$  and  $R$ ) 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  $E[g(x)]$  and  $\text{var}[g(x)]$  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,  $\mu$ , 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

$$p(yt|zt) \approx N(yt|h(\mu t|t-1) + Ht(yt - \mu t|t-1), R_t) \quad (18.82)$$

where  $H_t$  is the Jacobian matrix of  $h$  evaluated at the prior mode:

$$H_{ij} = \partial h_i(z) / \partial z_j \quad (18.83)$$

$$H_t = H|z=\mu t|t-1 \quad (18.84)$$

Similarly, we approximate the system model using

$$p(zt|zt-1, ut) \approx N(zt|g(ut, \mu t-1) + Gt(zt-1 - \mu t-1), Q_t) \quad (18.85)$$

where

$$G_{ij}(u) = \partial g_i(u, z) / \partial z_j \quad (18.86)$$

$$G_t = G(ut)|z=\mu t-1 \quad (18.87)$$

so  $G$  is the Jacobian matrix of  $g$  evaluated at the prior mode. Given this, we can then apply the Kalman filter to compute the posterior as follows:

$$\mu t|t-1 = g(ut, \mu t-1) \quad (18.88)$$

$$Vt|t-1 = GtVt-1Gt^T + Q_t \quad (18.89)$$

$$Kt = Vt|t-1Ht^T (HtVt|t-1Ht^T + R_t)^{-1} \quad (18.90)$$

$$\mu t = \mu t|t-1 + Kt(yt - h(\mu t|t-1)) \quad (18.91)$$

$$Vt = (I - KtHt)Vt|t-1 \quad (18.92)$$

We see that the only difference from the regular Kalman filter is that, when we compute the state prediction, we use  $g(ut, \mu t-1)$  instead of  $A\mu t-1 + Bt$ , and when we compute the measurement update we use  $h(\mu t|t-1)$

instead of  $C\mu|t-1$ . It is possible to improve performance by repeatedly re-linearizing the equations around  $\mu_t$  instead of  $\mu|t-1$ ; 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  $g$ , and once to approximate passing through the measurement model  $h$ . 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  $p(x) = N(x|\mu, \Sigma)$ , and consider estimating

$p(y)$ , where  $y = f(x)$  for some nonlinear function  $f$ . The unscented transform does this as follows. First we create a set of  $2d + 1$  sigma points  $x_i$ , given by

$$x = \{\mu + (\lambda(d + \kappa)\Sigma)^{1/2}\}_i \quad i=1, \quad \{\mu - (\lambda(d + \kappa)\Sigma)^{1/2}\}_i \quad i=1 \# \quad (18.93)$$

where  $\lambda = \alpha^2(d + \kappa) - d$  is a scaling parameter to be specified below, and the notation  $M:i$  means the  $i$ 'th column of matrix  $M$ . These sigma points are propagated through the nonlinear function to yield  $y_i = f(x_i)$ , and the mean and covariance for  $y$  is computed as follows:

$$\mu_y = \frac{1}{2d} \sum_{i=0}^{2d} w_i y_i \quad (18.94)$$

$$\Sigma_y = \frac{1}{2d} \sum_{i=0}^{2d} w_i (y_i - \mu_y)(y_i - \mu_y)^T \quad (18.95)$$

where the  $w$ 's are weighting terms, given by

$$w_i = \lambda / (d + \lambda) \quad (18.96)$$

$$w_i c = \lambda / (d + \lambda) + (1 - \alpha^2 + \beta) \quad (18.97)$$

$$w_i m = w_i c = 1 / (2(d + \lambda)) \quad (18.98)$$

See Figure 18.10 for an illustration. In general, the optimal values of  $\alpha$ ,  $\beta$  and  $\kappa$  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  $\mu$ ,  $\mu + \sqrt{3}\sigma$  and  $\mu - \sqrt{3}\sigma$ .

### 18.5.2.2 The UKF algorithm

The UKF algorithm is simply two applications of the unscented transform, one to compute  $p(z_t|y_1:t-1, u_1:t)$  and the other to compute  $p(z_t|y_1:t, u_1:t)$ . We give the details below.

The first step is to approximate the predictive density  $p(z_t|y_1:t-1, u_1:t) \approx N(z_t|\mu_t, \Sigma_t)$  by passing the old belief state  $N(z_{t-1}|\mu_{t-1}, \Sigma_{t-1})$  through the system model  $g$  as follows:

$$z_{0:t-1} = \{\mu_{t-1} + \gamma(\Sigma_{t-1})^{1/2}\}_i \quad i=1, \quad \{\mu_{t-1} - \gamma(\Sigma_{t-1})^{1/2}\}_i \quad i=1 \# \quad (18.99)$$

$$z_{*i:t} = g(u_t, z_{0:t-1}) \quad (18.100)$$

$$\mu_t = \frac{1}{2d} \sum_{i=0}^{2d} w_i m z_{*i:t} \quad (18.101)$$

$$\Sigma_t = \frac{1}{2d} \sum_{i=0}^{2d} w_i c (z_{*i:t} - \mu_t)(z_{*i:t} - \mu_t)^T + Q_t \quad (18.102)$$

where  $\gamma = \sqrt{d + \lambda}$ . The second step is to approximate the likelihood  $p(y_t|z_t) \approx N(y_t|\hat{y}_t, S_t)$  by passing the prior  $N(z_t|\mu_t, \Sigma_t)$  through the observation model  $h$ :

$$z_{0:t} = \mu_t, \quad \{\mu_t + \gamma(\Sigma_t)^{1/2}\}_i \quad i=1, \quad \{\mu_t - \gamma(\Sigma_t)^{1/2}\}_i \quad i=1 \quad (18.103)$$

$$y_{*i:t} = h(z_{0:t}) \quad (18.104)$$

$$\hat{y}_{*i:t} = \frac{1}{2d} \sum_{i=0}^{2d} w_i m y_{*i:t} \quad (18.105)$$

$$S_t = \frac{1}{2d} \sum_{i=0}^{2d} w_i c (y_{*i:t} - \hat{y}_{*i:t})(y_{*i:t} - \hat{y}_{*i:t})^T + R_t \quad (18.106)$$

Finally, we use Bayes rule for Gaussians to get the posterior  $p(z_t|y_1:t, u_1:t) \approx N(z_t|\mu_t, \Sigma_t)$ :

$$\sum z_i y_t = 2d \sum i w_i c(z_i * i - \mu t) (y_i * i - \hat{y}_t) T \quad (18.107)$$

$$K_t = \sum z_i y_t S^{-1} t \quad (18.108)$$

$$\mu_t = \mu_{t-1} + K_t(y_t - \hat{y}_t) \quad (18.109)$$

$$\Sigma_t = \Sigma_{t-1} - K_t S_t K_t \quad (18.110)$$

### 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  $\theta_t$ . Suppose that  $Q$  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  $q_{t-1}(\theta_{t-1}) \approx p(\theta_{t-1}|y_{1:t-1})$ , where  $q_{t-1} \in Q$ . We can update this with the new measurement to get the approximate posterior

$$\hat{p}(\theta_t) = Z_t p(y_t|\theta_t) q_t|_{t-1}(\theta_t) \quad (18.111)$$

where

$$Z_t = p(y_t|\theta_t) q_t|_{t-1}(\theta_t) d\theta_t \quad (18.112)$$

is the normalization constant and

$$q_t|_{t-1}(\theta_t) = p(\theta_t|\theta_{t-1}) q_{t-1}(\theta_{t-1}) d\theta_{t-1} \quad (18.113)$$

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,  $\hat{p}(\theta_t) \notin Q$ . So after updating we seek the best tractable approximation by computing

$$q(\theta_t) = \operatorname{argmin}_{q \in Q} KL(\hat{p}(\theta_t) || q(\theta_t)) \quad (18.114)$$

This minimizes the Kullback-Leibler divergence (Section 2.8.2) from the approximation  $q(\theta_t)$  to the "exact" posterior  $\hat{p}(\theta_t)$ , 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  $\theta_{tj}$  is the  $j$ 'th hidden variable at time  $t$ , then the exact posterior  $p(\theta_t)$  becomes intractable to compute because of the entanglement

problem. Suppose we use a fully factored approximation of the form  $q(\theta_t) = \prod_{j=1}^D q(\theta_{t,j} | \pi_{t,j})$ , where  $\pi_{t,j} = q(\theta_{t,j} = k)$  is the probability variable  $j$  is in state  $k$ , and  $D$  is the number of variables. In this case, the moment matching operation becomes

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

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

$$\mu_{t,j} = E[\theta_{t,j}], \tau_{t,j} = \text{var}[\theta_{t,j}] \quad (18.116)$$

This method can be used to do online inference for the parameters of many statistical models. For example, the TrueSkill 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

$$p(y_t|x_t, \theta_t) = \text{Ber}(y_t|\text{sigm}(x_t^\top \theta_t)) \quad (18.117)$$

$$p(\theta_t|\theta_{t-1}) = N(\theta_t|\theta_{t-1}, \sigma^2 I) \quad (18.118)$$

where  $\sigma^2$  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  $q(\theta_{t-1}) = \prod_{j=1}^D N(\theta_{t-1,j} | \mu_{t-1,j}, \tau_{t-1,j})$  is the tractable prior. We can compute the one-step-ahead predictive density  $q_t|t-1(\theta_t)$  using the standard linear-Gaussian update. So now we concentrate on the measurement update step. Define the deterministic quantity  $s_t = \theta_t^\top x_t$ , as shown in Figure 18.11(b). If  $q_t|t-1(\theta_t) = \prod_{j=1}^D N(\theta_{t,j} | \mu_{t,j}, \tau_{t,j})$ , then we can compute the predictive distribution for  $s_t$  as follows:

$$q_t|t-1(s_t) = N(s_t | m_t, v_t) \quad (18.119)$$

$$m_{t|t-1} = j x_t, \mu_{t|t-1,j} (18.120)$$

$$v_{t|t-1} = j x_2 t, \tau_{t|t-1,j} (18.121)$$

The posterior for  $s_t$  is given by

$$q_t(s_t) = N(s_t | m_t, v_t) (18.122)$$

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

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

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

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  $q(\theta | s_t)$ . This can be done as follows. Define  $\delta_m$  as the change in the mean of  $s_t$  and  $\delta_v$  as the change in the variance:

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

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

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

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

$$\tau_{t,j} = \tau_{t|t-1,j} + a_{2,j} \delta_v (18.129)$$

$$a_j = x_{t,j} \tau_{t|t-1,j} j x_2 t, j \tau_2 t | t-1, j (18.130)$$

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).

已思考 21s

下面我先把你给的文字逐条完整翻译（不遗漏），然后给出通俗详细的解释，最后总结要点。文中出现的图（18.9–18.11）我也逐一说明它们想表达的直觉。

## 翻译

**图 18.9 非线性对高斯随机变量的变换。**右下角显示的是先验  $p(x)$ 。右上角显示的是函数  $y = g(x)$ 。左上角显示的是变换后的分布  $p(y)$ 。**线性**函数会产生**高斯**分布，但**非线性**函数会产生**复杂**分布。实线是对此的**最佳高斯近似**；虚线是**EKF (扩展卡尔曼滤波)** 对此的近似。

**图 18.10 二维情形下无迹变换 (unscented transform) 的例子。**

**图 18.11 (a)** 说明了**假设密度滤波 (ADF)** 的“预测–更新–投影”循环。**(b)** 一个动态逻辑回归模型。可与图 18.4(a) 对比。

## 18.5 非线性、非高斯 SSM 的近似在线推断

在 18.3.1 节里，我们讨论了如何对 **LG-SSM** (线性–高斯 SSM) 做**精确的在线推断**。然而，很多模型是**非线性的**。比如，大多数运动物体并不会沿直线运动。即使运动是直线的，如果我们假设模型的**参数未知**并把它们加入状态空间，模型也会变成**非线性**。此外，**非高斯噪声**也很常见，例如由异常值造成的，或者当我们为 **GLM** (广义线性模型) 而非仅仅线性回归来估计参数时。对于这些更一般的模型，我们需要用**近似推断**。我们下面讨论的近似推断算法，会用**高斯**来近似后验。一般地，若  $Y = f(X)$ ，其中  $X$  服从高斯分布而  $f$  是非线性函数，那么把  $p(Y)$  近似为高斯有两条主要途径：

**第一**，对  $f$  做**一阶近似**；**第二**，使用**原始的**  $f$ ，但把  $f(X)$  通过“**矩匹配**”投影到高斯族上。我们依次讨论这两种方法。（另见 23.5 节，我们会讨论**粒子滤波**：这是一种随机的近似在线推断算法，用**非参数**的方式近似后验，通常更准，但更慢。）

### 18.5.1 扩展卡尔曼滤波 (EKF)

在本节中，我们专注于**非线性**模型，但假设**噪声是高斯**。即考虑

$$z_t = g(u_t, z_{t-1}) + \mathcal{N}(0, Q_t) \quad (18.80)$$

$$y_t = h(z_t) + \mathcal{N}(0, R_t) \quad (18.81)$$

其中**状态转移**  $g$  与**观测**  $h$  都是**非线性但可微**的函数。此外，我们专注于用**单个高斯**来近似后验。（处理更一般的后验，如多峰、离散等，最简单的方法是用**粒子滤波**，见 23.5 节。）**扩展卡尔曼滤波 (EKF)** 可用于这种**非线性高斯的动态系统**。基本思想是：**围绕上一时刻的估计**，用**一阶泰勒展开**把  $g$  与  $h$  **线性化**，然后套用标准卡尔曼滤波方程。（式中的**噪声方差**  $Q$  与  $R$  不变，也就是说，**线性化额外造成的误差并未被建模**。）因此，我们把**平稳的非线性系统**近似为**非平稳的线性系统**。

该方法的直觉如图 18.9：把右下角的高斯  $p(x)$  通过右上角的非线性  $y = g(x)$  变换，得到左上角阴影区域中的分布（用蒙特卡洛近似）。基于  $E[g(x)]$  与  $\text{var}[g(x)]$  的**最佳高斯近似**。

似用黑色实线表示。EKF 的做法是：在当前模态（均值） $\mu$  处线性化  $g$ ，然后把高斯  $p(x)$  通过这个线性化函数传过去。在这个例子里，它对  $p(y)$  的一阶、二阶矩的近似相当不错，而且代价远小于 MC 近似。更具体地，方法如下。把测量模型近似为

$$p(y_t | z_t) \approx N(y_t | h(\mu_{t|t-1}) + H_t(y_t - \mu_{t|t-1}), R_t) \quad (18.82)$$

其中  $H_t$  是在先验模态处计算的  $h$  的雅可比矩阵：

$$H_{ij} = \frac{\partial h_i(z)}{\partial z_j} \quad (18.83)$$

$$H_t = H \Big|_{z=\mu_{t|t-1}} \quad (18.84)$$

同理，把系统模型近似为

$$p(z_t | z_{t-1}, u_t) \approx N(z_t | g(u_t, \mu_{t-1}) + G_t(z_{t-1} - \mu_{t-1}), Q_t) \quad (18.85)$$

其中

$$G_{ij}(u) = \frac{\partial g_i(u, z)}{\partial z_j} \quad (18.86)$$

$$G_t = G(u_t) \Big|_{z=\mu_{t-1}} \quad (18.87)$$

于是可以套用卡尔曼滤波得到后验：

$$\mu_{t|t-1} = g(u_t, \mu_{t-1}) \quad (18.88)$$

$$V_{t|t-1} = G_t V_{t-1} G_t^\top + Q_t \quad (18.89)$$

$$K_t = V_{t|t-1} H_t^\top (H_t V_{t|t-1} H_t^\top + R_t)^{-1} \quad (18.90)$$

$$\mu_t = \mu_{t|t-1} + K_t(y_t - h(\mu_{t|t-1})) \quad (18.91)$$

$$V_t = (I - K_t H_t) V_{t|t-1} \quad (18.92)$$

可见，与普通卡尔曼滤波的唯一区别是：状态预测用  $g(u_t, \mu_{t-1})$  替代  $A_t \mu_{t-1} + B_t u_t$ ；测量预测用  $h(\mu_{t|t-1})$  替代  $C_t \mu_{t|t-1}$ 。通过在  $\mu_t$  处反复线性化而非在  $\mu_{t|t-1}$  处一次线性化，可以改进效果——这叫迭代 EKF，更准但更慢。两种情形下 EKF 表现较差：其一，先验协方差很大（分布很宽，很多概率质量经过了远离均值的非线性区域）；其二，函数在当前均值附近高度非线性。18.5.2 节将介绍一种在这两种设置下都优于 EKF 的算法——UKF。

## 18.5.2 无迹卡尔曼滤波 (UKF)

无迹卡尔曼滤波 (UKF) 是 EKF 的改良版 (Julier 和 Uhlmann, 1997) —— (据说叫“无迹”是因为它“没有异味/不臭”！)。核心直觉是：近似一个高斯比近似一个函数要容易。

所以我们不去线性近似函数再传递高斯，而是选择一组**确定性的点**（称为Sigma 点）穿过函数，然后对**变换后的点\*\*\*\*拟合高斯**。这叫**无迹变换**，示意见图 18.10（细节如下）。UKF 基本上对无迹变换用**两次**：一次近似穿过系统模型  $g$ ，一次近似穿过观测模型  $h$ 。二者每步的计算量都是  $O(d^3)$ ，其中  $d$  为隐藏状态维度。但 UKF 至少**二阶准确**，而 EKF 只有**一阶准确**（两者都可扩展加入更高阶项）。另外，无迹变换**不需要解析求导/雅可比**（所谓**无导数滤波器**），实现更简单、更通用。

### 18.5.2.1 无迹变换

先讲无迹变换本身。设  $p(x) = N(x | \mu, \Sigma)$ ，要估计  $p(y)$ ，其中  $y = f(x)$  是某个非线性函数。做法：先构造  $2d + 1$  个 Sigma 点  $x_i$ ：

$$x = \{\mu, \{\mu + (\sqrt{(d + \lambda)\Sigma})_{:,i}\}_{i=1}^d, \{\mu - (\sqrt{(d + \lambda)\Sigma})_{:,i}\}_{i=1}^d\} \quad (18.93)$$

其中  $\lambda = \alpha^2(d + \kappa) - d$  是一个缩放参数，“ $M_{:,i}$ ”表示矩阵  $M$  的第  $i$  列。将这些点通过非线性函数： $y_i = f(x_i)$ 。然后用如下**加权**计算  $y$  的均值与协方差：

$$\mu_y = \sum_{i=0}^{2d} w_i^m y_i \quad (18.94)$$

$$\Sigma_y = \sum_{i=0}^{2d} w_i^c (y_i - \mu_y)(y_i - \mu_y)^\top \quad (18.95)$$

权重为

$$w_0^m = \frac{\lambda}{d + \lambda} \quad (18.96)$$

$$w_0^c = \frac{\lambda}{d + \lambda} + (1 - \alpha^2 + \beta) \quad (18.97)$$

$$w_i^m = w_i^c = \frac{1}{2(d + \lambda)}, \quad i = 1, \dots, 2d \quad (18.98)$$

见图 18.10 的示意。一般地， $\alpha, \beta, \kappa$  的最优取值依赖于问题；当  $d = 1$  时，可取  $\alpha = 1, \beta = 0, \kappa = 2$ 。此时  $\lambda = 2$ ，三个 Sigma 点是  $\mu, \mu + \sqrt{3}\sigma, \mu - \sqrt{3}\sigma$ 。

### 18.5.2.2 UKF 算法

UKF 就是无迹变换的两次应用：一次得到  $p(z_t | y_{1:t-1}, u_{1:t})$ ，一次得到  $p(z_t | y_{1:t}, u_{1:t})$ 。细节如下。

**第一步（系统模型）**：把旧的信念状态  $N(z_{t-1} | \mu_{t-1}, \Sigma_{t-1})$  通过  $g$ ，得到对预测分布  $N(z_t | \mu_t, \Sigma_t)$  的近似：

$$z_{t-1}^0 = \{\mu_{t-1}, \{\mu_{t-1} + \gamma(\sqrt{\Sigma_{t-1}})_{:i}\}_{i=1}^d, \{\mu_{t-1} - \gamma(\sqrt{\Sigma_{t-1}})_{:i}\}_{i=1}^d\} \quad (18.99)$$

$$z_t^{*i} = g(u_t, z_{t-1}^{0i}) \quad (18.100)$$

$$\mu_t = \sum_{i=0}^{2d} w_i^m z_t^{*i} \quad (18.101)$$

$$\Sigma_t = \sum_{i=0}^{2d} w_i^c (z_t^{*i} - \mu_t)(z_t^{*i} - \mu_t)^\top + Q_t \quad (18.102)$$

其中  $\gamma = \sqrt{d + \lambda}$ 。

**第二步 (观测模型)** : 把先验  $N(z_t | \mu_t, \Sigma_t)$  通过  $h$ , 得到似然的高斯近似  $N(y_t | \hat{y}_t, S_t)$  :

$$z_t^0 = \{\mu_t, \{\mu_t + \gamma(\sqrt{\Sigma_t})_{:i}\}_{i=1}^d, \{\mu_t - \gamma(\sqrt{\Sigma_t})_{:i}\}_{i=1}^d\} \quad (18.103)$$

$$y_t^{*i} = h(z_t^{0i}) \quad (18.104)$$

$$\hat{y}_t = \sum_{i=0}^{2d} w_i^m y_t^{*i} \quad (18.105)$$

$$S_t = \sum_{i=0}^{2d} w_i^c (y_t^{*i} - \hat{y}_t)(y_t^{*i} - \hat{y}_t)^\top + R_t \quad (18.106)$$

**第三步 (贝叶斯更新)** : 用高斯-高斯的规则得到后验近似  $N(z_t | \mu_t, \Sigma_t)$ :

$$\Sigma_t^{z,y} = \sum_{i=0}^{2d} w_i^c (z_t^{*i} - \mu_t)(y_t^{*i} - \hat{y}_t)^\top \quad (18.107)$$

$$K_t = \Sigma_t^{z,y} S_t^{-1} \quad (18.108)$$

$$\mu_t = \mu_t + K_t(y_t - \hat{y}_t) \quad (18.109)$$

$$\Sigma_t = \Sigma_t - K_t S_t K_t^\top \quad (18.110)$$

### 18.5.3 假设密度滤波 (ADF)

本节讨论这样一种推断: **更新这一步时是精确的**, 但随后把后验逼近成某个**方便的分布族** (例如高斯)。更准确地, 设我们要推断的未知量为  $\theta_t$ 。设  $Q$  是一族“可处理”的分布, 比如**对角协方差**的高斯, 或离散分布的乘积。假设我们有一个近似先验  $q_{t-1}(\theta_{t-1}) \approx p(\theta_{t-1} | y_{1:t-1})$ , 且  $q_{t-1} \in Q$ 。我们可以用新观测来更新, 得到近似后验

$$\hat{p}(\theta_t) = \frac{1}{Z_t} p(y_t | \theta_t) q_{t|t-1}(\theta_t) \quad (18.111)$$

其中

$$Z_t = \int p(y_t | \theta_t) q_{t|t-1}(\theta_t) d\theta_t \quad (18.112)$$

是归一化常数，而

$$q_{t|t-1}(\theta_t) = \int p(\theta_t | \theta_{t-1}) q_{t-1}(\theta_{t-1}) d\theta_{t-1} \quad (18.113)$$

是一部预测分布。若先验来自足够受限的族，这个一步更新通常是可解的。但我们经常会发现结果不再属于我们可处理的族，即  $\hat{p}(\theta_t) \notin Q$ 。因此，在更新之后，我们通过解

$$q(\theta_t) = \arg \min_{q \in Q} \text{KL}(\hat{p}(\theta_t) \| q(\theta_t)) \quad (18.114)$$

来寻找最佳的可处理近似。这最小化了从近似  $q(\theta_t)$  到“精确”后验  $\hat{p}(\theta_t)$  的KL 散度 (2.8.2 节)，可以理解为把  $\hat{p}$  投影到“可处理分布”的空间。整个算法就是预测-更新-投影的循环，称为**假设密度滤波 ADF** (Maybeck, 1979)。见图 18.11(a) 的示意。若  $q$  属于**指数族**，可以证明这个 KL 最小化可以通过**矩匹配**完成。我们在下面给一些例子。

### 18.5.3.1 DBN 在线推断的 Boyen–Koller 算法

如果我们在**离散状态**的动态贝叶斯网 (17.6.7 节) 中做推断， $\theta_t^j$  是时刻  $t$  的第  $j$  个隐变量，那么由于**纠缠问题**，精确后验  $p(\theta_t)$  的计算会变得**不可 tractable**。假设我们采用一个完全分解的近似：

$$q(\theta_t) = \prod_{j=1}^D \text{Cat}(\theta_{t,j} | \pi_{t,j}),$$

其中  $\pi_{tjk} = q(\theta_{t,j} = k)$  表示第  $j$  个变量取第  $k$  个状态的概率， $D$  是变量数。在这种情形，**矩匹配操作**就是

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

它可以通过用分解先验做一次“预测-更新”，然后计算**后验边缘**来得到。这称作**Boyen–Koller 算法**，以 (Boyen & Koller, 1998) 命名；他们证明，在某些系统随机性的假设下，这一系列重复近似所引入的误差是有界的。

### 18.5.3.2 GLM 在线推断的高斯近似

现在设

$$q(\theta_t) = \prod_{j=1}^D N(\theta_{t,j} | \mu_{t,j}, \tau_{t,j}),$$

其中  $\tau_{t,j}$  是方差。那么对后验的最佳可处理近似的参数为

$$\mu_{t,j} = \text{E}_{\hat{p}}[\theta_{t,j}], \quad \tau_{t,j} = \text{var}_{\hat{p}}[\theta_{t,j}] \quad (18.116)$$

这种方法可用于很多统计模型的**在线参数推断**。例如微软 Xbox 的 TrueSkill 系统（用于随时间给玩家排名）就使用了这种形式的近似（Herbrich 等, 2007）。我们也可以把它应用到更简单的模型，如 GLM；其优点是后验是**对数凹的**。下面按 (Zoeter, 2007) 的表述，解释**二元逻辑回归**的做法。模型为

$$p(y_t | x_t, \theta_t) = \text{Ber}(y_t | \text{sigm}(x_t^\top \theta_t)) \quad (18.117)$$

$$p(\theta_t | \theta_{t-1}) = \mathcal{N}(\theta_t | \theta_{t-1}, \sigma^2 I) \quad (18.118)$$

其中  $\sigma^2$  是**过程噪声**，允许参数随时间慢慢变化（如愿也可设为 0，就得到 18.2.3 节的递归最小二乘思想）。设近似先验为

$$q_{t-1}(\theta_{t-1}) = \prod_j \mathcal{N}(\theta_{t-1,j} | \mu_{t-1,j}, \tau_{t-1,j}).$$

我们可用线性-高斯的标准更新算出一步预测  $q_{t|t-1}(\theta_t)$ 。因此接下来只需关注**测量更新**。

定义确定量

$$s_t = \theta_t^\top x_t,$$

如图 18.11(b) 所示。若

$$q_{t|t-1}(\theta_t) = \prod_j \mathcal{N}(\theta_{t,j} | \mu_{t|t-1,j}, \tau_{t|t-1,j}),$$

则  $s_t$  的预测分布为

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

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

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

$s_t$  的后验为

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

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

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

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

其中  $p(y_t | s_t) = \text{Ber}(y_t | s_t)$ 。这些积分是一维的，可用**高斯求积**计算（细节见 Zoeter 2007）。这和 UKF 的一步是同一类操作。获得  $q(s_t)$  后，需要计算  $q(\theta | s_t)$ 。做法如下。令  $\delta m$  为  $s_t$  的均值变化， $\delta v$  为方差变化：

$$m_t = m_{t|t-1} + \delta m, \quad v_t = v_{t|t-1} + \delta v \quad (18.126)$$

可推出参数后验的新的分解形式为

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

$$\mu_{t,j} = \mu_{t|t-1,j} + a_j \delta m \quad (18.128)$$

$$\tau_{t,j} = \tau_{t|t-1,j} + a_j^2 \delta v \quad (18.129)$$

$$a_j = \frac{x_{t,j} \tau_{t|t-1,j}}{\sum_j x_{t,j}^2 \tau_{t|t-1,j}^2} \quad (18.130)$$

因此可见，与**绝对值更大的输入分量**（更大  $|x_{t,j}|$ ）或**不确定性更大的参数**（更大  $\tau_{t|t-1,j}$ ）相关的参数会被更新得更多，这很直观。Opper (1998) 用 probit 似然（见 9.4 节）推导了该算法的一个版本；此时测量更新可**闭式**完成，无需数值积分。不论哪种情况，该算法每步只需  $O(D)$  的计算量，适合**高维参数**；且由于是**在线**算法，可处理**超大数据集**。例如 Zhang 等 (2010) 用其一个变体在线拟合**多类分类器**，数据量很大，效果超过其他（非贝叶斯）在线学习算法，有时甚至**优于**最先进的批量（离线）方法如 SVM（14.5 节）。

## 通俗解释（不依赖太多数学）

### 1) 为什么要近似？

卡尔曼滤波的“精确好用”靠的是**线性 + 高斯**。一旦**函数是弯的**（非线性）或**噪声不高斯**，就没法精确算了——只能**近似**。图 18.9 告诉你：把一个**高斯云**通过一个**弯弯的函数**，出来的**形状很怪**；我们用一种“尽量像高斯”的方法去**逼近它**。

### 2) 两条主路：线性化 vs 无迹变换

- **EKF (线性化)**：在当前估计点附近把函数**拉直**（泰勒一阶）。好处：简单、快。坏处：
  - 如果**先验很宽**（不确定性大），大量概率跑到“离均值很远的地方”，而那里函数已经不是真的了 → 误差大；
  - 若函数在均值附近就很**弯**，一阶也抓不住形状。**迭代 EKF**能好些，但更慢。

- **UKF (无迹)**：不去近似函数，而是用少量“代表点”（Sigma 点）代表整个高斯，把点穿过真正的函数，再把变换后的点匹配成高斯（对齐均值与协方差）。好处：
  - **二阶准确**，通常比 EKF 稳准；
  - **不需要求导**（不用写雅可比），好实现、泛用性强。
  - 复杂度跟 EKF 同阶（每步  $O(d^3)$ ）。
  - 直觉见图 18.10：左边“采样真值”是海量点；中间“Sigma 点”只取  $2d + 1$  个代表点，穿过函数后，能把新的均值和协方差抓住；右边“线性化”只得到一个拉直的近似，常常偏。

### 3) ADF：每步“先精确、再压回可处理的形状”

- **预测**：把上一步的近似分布（比如对角高斯/离散乘积）推到下一步。
- **更新**：用新数据做一个**精确的乘法**（得到更接近真实的后验  $\hat{p}$ ）。
- **投影**：把  $\hat{p}$  投影回“我们能处理的族”里（比如再变回对角高斯），方法是**最小化 KL**，常常就是**匹配均值与方差**。
- 如图 18.11(a)：“Predict–Update–Project”三角循环来回走。
- 两个实例：
  - **Boyen–Koller (离散 DBN)**：把复杂后验投影成“各变量独立的分类分布”的乘积，保证误差有界。
  - **GLM/逻辑回归参数在线学习**：把参数当作随机变量，保持“各维独立的高斯”。每来一条样本，用一维的积分（同 UKF 一步）**更新线性响应**  $s = \theta^\top x$  的均值/方差，再把变化量分配回各个参数维（谁权重大、谁不确定性大，谁更新多）。超快 ( $O(D)$ )、很适合大数据；这也是 TrueSkill 等系统背后的思想。

### 4) 图的直觉

- **图 18.9**：高斯过非线性会“变形”；EKF 用切线近似；“最佳高斯”用真实函数的均值/方差匹配。
- **图 18.10**：比较三种思路：
  - “实际采样”：很多点穿过函数，最准确但最慢；
  - “Sigma 点”：少量代表点穿过函数，再匹配高斯，**准确又省点**；
  - “线性化(EKF)”：一条切线，可能偏得多。
- **图 18.11**：
  - (a) ADF 的三步循环：Predict→Update→Project。
  - (b) 动态逻辑回归的图模型：参数  $\theta_t$  随时间演化，输入  $x_t$  生成输出  $y_t$ 。

## 要点总结 (背下来就能用)

- 非线性/非高斯时不能用“精确卡尔曼”，要用近似。
- 两大思路：
  1. EKF：在当前估计点线性化函数（快、简单；在分布很宽或高度非线性时易失真）。
  2. UKF：用  $2d + 1$  个 Sigma 点穿过真实函数，再矩匹配成高斯（二阶准确、无需求导、通常更准）。
- ADF：每步精确更新后再把分布投影回“可处理的族”（常用矩匹配）。
  - Boyen–Koller：离散 DBN 的在线近似，误差有界。
  - GLM/逻辑回归的在线参数学习：把参数当随机变量、保持独立高斯，用一维积分更新  $s = \theta^\top x$  的均值/方差，再把变化分摊到各参数（权重大、方差大→更新多），整体每步  $O(D)$ 。
- 何时选谁：
  - 要快且函数不太弯 → EKF；
  - 函数弯、先验宽、或不想算雅可比 → UKF；
  - 结构更复杂（离散/大规模参数）→ ADF（或粒子滤波 23.5 节）。

如果你愿意，我可以用一个小例子（例如 1D 非线性观测）把 EKF 和 UKF 的曲线、均值、方差一步步画出来，让差异更直观。

这一节主要在讲什么？我不太懂

已思考 16s

这段 (18.5) 讲的是：当状态空间模型不再“线性+高斯”时，怎样在“在线”场景里用近似方法做推断。卡尔曼滤波只能精确解决线性高斯；一旦函数是非线性的或噪声/似然非高斯，就得用近似。主要给了三条路，并配了三张图说明直觉。

## 讲了哪三招？

1. EKF (扩展卡尔曼滤波) —— 把函数拉直再用卡尔曼
  - 思想：在当前估计点对状态方程  $g$ 、观测方程  $h$  做一阶泰勒展开（线性化），再按普通卡尔曼滤波更新均值/协方差。
  - 优点：实现简单，复杂度与 KF 同阶。
  - 缺点：先验很宽或函数在当前很弯都会失真；要算雅可比。

- 图 18.9：高斯穿过非线性会变形；EKF用切线逼近，和“最佳高斯”相比可能有偏。

## 2. UKF (无迹卡尔曼滤波) —不拉直函数，改“代表点”穿过去

- 思想：用  $2d + 1$  个Sigma 点代表高斯，把这些点穿过真实非线性，再用矩匹配（均值、协方差）拟合出一个高斯。
- 优点：**二阶精度、不需要求导**（实现省心），通常比 EKF 稳准；复杂度同阶。
- 图 18.10：对比“真采样”（很多点、最准最慢）、“Sigma 点”（少点也能抓住均值/协方差），“线性化”（可能偏）。

## 3. ADF (假设密度滤波) —每步精确更新后再“投影回”可处理族

- 思想：**预测→更新→投影**：先用新观测做真更新得到  $\hat{p}$ ，再解  $\min_q \text{KL}(\hat{p} \| q)$  把它投影回“方便的分布族”（如对角高斯/分解的离散分布）。指数族里这一步等价于**矩匹配**。
- 图 18.11(a)：就是这个三步循环示意。
- 两个应用：
  - **Boyen–Koller**（离散 DBN）：把复杂后验投影成**各变量独立**的分类分布，误差可证有界。
  - **GLM/逻辑回归的在线参数学习**：把参数当随机变量、保持**各维独立高斯**；先对  $s_t = \theta^\top x_t$  做一维积分更新其均值/方差，再把变化量按权重分回每个参数（ $|x_{t,j}|$  大或不确定性大的参数更新更多）。图 18.11(b) 是这个动态逻辑回归的图模型。

## 何时用谁（速配）

- **函数不太弯、可算雅可比** → EKF（快、简单）。
- **函数较弯、先验较宽、或不想/不能算导** → UKF（更准、免雅可比）。
- **结构复杂/高维/离散变量或要非常快的在线参数更新** → ADF（或粒子滤波见 23.5）。

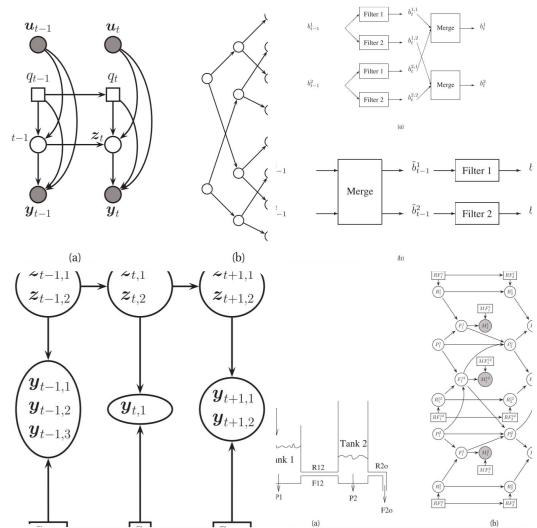
## 三张图想表达什么

- 18.9：非线性会把高斯“捏歪”；EKF用切线近似，和“最佳高斯”有差距。
- 18.10：无迹变换用少量 Sigma 点就能很好重建变换后的均值、协方差。
- 18.11：ADF 的“预测–更新–投影”循环；以及动态逻辑回归结构。

## 要点小结

- 非线性/非高斯 ⇒ **精确卡尔曼不适用，需近似在线推断**。
- **两大近似思路：**
  - **线性化函数**（EKF，快但可能偏）；
  - **保留函数、近似分布**（UKF 用 Sigma 点做矩匹配，通常更稳）。

- ADF 提供了通用“更新后再投影”的框架，支持离散 DBN 与 GLM 等大规模在线学习（每步可做到  $O(D)$ ）。



翻译并解释以下内容，翻译不能有遗漏，解释请通俗易懂且详尽，因为我的数学基础不太好，最后总结要点。

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.

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

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.

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.

## 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,  $q_t \in \{1, \dots, K\}$ , a continuous latent variable,  $z_t \in \mathbb{R}^L$ , an observed response  $y_t \in \mathbb{R}^D$  and an optional continuous observed input or control  $u_t \in \mathbb{R}^U$ . We then assume that the continuous variables have linear Gaussian CPDs, conditional on the discrete states:

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

$$p(z_t | z_{t-1}, q_t = k, u_t, \theta) = N(z_t | A_k z_{t-1} + B_k u_t, Q_k) \quad (18.132)$$

$$p(y_t | z_t, q_t = k, u_t, \theta) = N(y_t | C_k z_t + D_k u_t, R_k) \quad (18.133)$$

See Figure 18.12(a) for the DGM representation.

### 18.6.1 Inference

Unfortunately inference (i.e., state estimation) in hybrid models, including the switching LGSSM model, is intractable. To see why, suppose  $q_t$  is binary, but that only the dynamics  $A$  depend on  $q_t$ , not the observation matrix. Our initial belief state will be a mixture of 2 Gaussians, corresponding to  $p(z_1 | y_1, q_1 = 1)$  and  $p(z_1 | y_1, q_1 = 2)$ . The one-step-ahead predictive density will be a mixture of 4 Gaussians  $p(z_2 | y_1, q_1 = 1, q_2 = 1)$ ,  $p(z_2 | y_1, q_1 = 1, q_2 = 2)$ ,  $p(z_2 | y_1, q_1 = 2, q_2 = 1)$ , and  $p(z_2 | y_1, q_1 = 2, q_2 = 2)$ , 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  $y_2$ . 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 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:

$$b_{t-1} = p(z_{t-1}, q_{t-1} = i | y_1:t-1) = \pi_{t-1,i} N(z_{t-1} | \mu_{t-1,i}, \Sigma_{t-1,i}) \quad (18.134)$$

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

$$b_{ij,t} = p(z_t, q_t = i, q_{t-1} = j | y_1:t) = \pi_{t-1,i} \pi_{t,j} N(z_t | \mu_{t,j}, \Sigma_{t,j}) \quad (18.135)$$

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

$$b_{j,t} = p(z_t, q_t = j | y_1:t) = \pi_{t,j} N(z_t | \mu_{t,j}, \Sigma_{t,j}) \quad (18.136)$$

See Figure 18.13(a) for a sketch. The optimal way to approximate a mixture of Gaussians with a single Gaussian is given by  $q = \text{argmin}_q KL(q || p)$ , where  $p(z) = k \pi_k N(z | \mu_k, \Sigma_k)$  and  $q(z) = N(z | \mu, \Sigma)$ . This can be solved by moment matching, that is,

$$\mu = E[z] = k \pi_k \mu_k \quad (18.137)$$

$$\Sigma = \text{cov}[z] = k \pi_k \Sigma_k + (\mu_k - \mu)(\mu_k - \mu)^T \quad (18.138)$$

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  $b_{ij,t}$  to  $b_{j,t}$  as follows (where we drop the  $t$  subscript for brevity):

$$\pi_j = i \pi_{ij} \quad (18.139)$$

$$\pi_j \mu_i = \pi_{ij} j \pi_{ij} \quad (18.140)$$

$$\mu_j = i \pi_j \mu_i \quad (18.141)$$

$$\Sigma_j = i \pi_j \Sigma_{ij} + (\mu_{ij} - \mu_j)(\mu_{ij} - \mu_j)^T \quad (18.142)$$

This algorithm requires running K2 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  $y_{tk}$  and the K objects  $z_{tj}$ . 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 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).

#### 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.

已思考 16s

下面我先把你给出的文字**完整翻译**（不遗漏），然后给出**通俗详尽的解释**，最后**总结要点**。

## 翻译

**图 18.12 切换线性动态系统。**

(a) 方块表示离散结点，圆圈表示连续结点。

(b) 说明信念状态（后验）的模态数如何随时间指数式增长。我们假设有两个二值状态。

**图 18.13 用于切换线性动态系统的 ADF。**

- (a) GPB2 方法。
- (b) IMM 方法。详见正文。

**图 18.14 一个在数据关联存在歧义的情形下跟踪两个目标的模型。我们在前三个时间步分别观测到 3、1 和 2 个检测。**

**图 18.15**

- (a) **双水箱系统。** 目标是根据（有噪声的）观测：来自水箱 1 的流量  $F_{1o}$ 、来自水箱 2 的流量  $F_{2o}$ 、以及水箱 1 与 2 之间的流量  $F_{12}$ ，去推断管道是否堵塞或破裂，或传感器是否损坏。 $R_{1o}$  是表示水箱 1 出口管道阻力的隐藏变量， $P_1$  是表示水箱 1 压力的隐藏变量，等等。来源：(Koller and Lerner 2001) 的图 11，经 Daphne Koller 授权使用。
  - (b) 双水箱系统的动态贝叶斯网表示。离散结点是方块，连续结点是圆圈。缩写： $R$ =阻力， $P$ =压力， $F$ =流量， $M$ =测量， $RF$ =阻力故障， $MF$ =测量故障。
- 

## 18.6 离散/连续混合的 SSM

许多系统同时包含离散与连续的隐藏变量；这些称为**混合 (hybrid)** 系统。例如，离散变量可以表示某个测量传感器是否失效，或者系统处于哪个“**机制/工况**”。下面我们会看到其他例子。一个混合系统的特殊情形是把 HMM 与 LG-SSM 结合起来，这叫做**切换线性动态系统 (SLDS)**、**跳变马尔可夫线性系统 (JMLS)**，或**切换状态空间模型 (SSSM)**。更精确地，我们有一个**离散潜变量**  $q_t \in \{1, \dots, K\}$ 、一个**连续潜变量**  $z_t \in \mathbb{R}^L$ 、一个**连续可观测响应**  $y_t \in \mathbb{R}^D$  以及一个可选的**连续输入/控制**  $u_t \in \mathbb{R}^U$ 。我们假设在给定离散状态的条件下，连续变量具有**线性-高斯**条件概率分布：

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

$$p(z_t \mid z_{t-1}, q_t = k, u_t, \theta) = \mathcal{N}(z_t \mid A_k z_{t-1} + B_k u_t, Q_k) \quad (18.132)$$

$$p(y_t \mid z_t, q_t = k, u_t, \theta) = \mathcal{N}(y_t \mid C_k z_t + D_k u_t, R_k) \quad (18.133)$$

见图 18.12(a) 的有向图表示。

### 18.6.1 推断

不幸的是，在混合模型（包括切换 LG-SSM）中进行推断（即状态估计）是**不可 tractable** 的。原因如下：设  $q_t$  为二值，且只有**动态矩阵**  $A$  依赖于  $q_t$ ，观测矩阵不依赖。初始信念将是**两个高斯的混合**，对应  $p(z_1 \mid y_1, q_1 = 1)$  与  $p(z_1 \mid y_1, q_1 = 2)$ 。一步预测会变成**四个高斯的混合**： $p(z_2 \mid y_1, q_1 = 1, q_2 = 1)$ 、 $p(z_2 \mid y_1, q_1 = 1, q_2 = 2)$ 、 $p(z_2 \mid$

$y_1, q_1 = 2, q_2 = 1$ 、 $p(z_2 | y_1, q_1 = 2, q_2 = 2)$ ，这是把每个先验模态都通过两种可能的转移模型得到的。第 2 步的信念状态也是 4 个高斯的混合（再用  $y_2$  更新得到）。到第 3 步，信念状态会变成 8 个高斯的混合，以此类推。可见**模态数指数爆炸**（见图 18.12(b)）。针对该模型已有多种近似推断方法，例如：

- 剪枝离散树中概率较小的轨迹；这就是**多假设跟踪**（MHT）（Bar-Shalom and Fortmann 1988；Bar-Shalom and Li 1993）的基础。
- 使用**蒙特卡洛**：抽样离散轨迹，并在给定轨迹的条件下对连续变量应用**解析滤波**。细节见 23.6 节。
- 使用 ADF：把指数多的高斯混合**近似为少量高斯混合**。详见 18.6.1.1。

### 18.6.1.1 切换 SSM 的高斯和滤波器

**高斯和滤波器**（Sorenson and Alspach 1971）在每一步把信念状态近似为 $K$  个高斯的混合\*\*。可以通过**并行运行  $K$  个卡尔曼滤波器**来实现，这特别适合切换 SSM。下面描述一种版本，叫“**二阶广义伪贝叶斯滤波器**（GPB2）”（Bar-Shalom and Fortmann 1988）。假设先验信念  $b_{t-1}$  是每个离散状态一个高斯的混合：

$$b_{t-1}^i = p(z_{t-1}, q_{t-1} = i | y_{1:t-1}) = \pi_{t-1,i} N(z_{t-1} | \mu_{t-1,i}, \Sigma_{t-1,i}) \quad (18.134)$$

将其通过  $K$  个不同的线性模型：

$$b_t^{ij} = p(z_t, q_{t-1} = i, q_t = j | y_{1:t}) = \pi_{ij}^t N(z_t | \mu_{t,ij}, \Sigma_{t,ij}) \quad (18.135)$$

其中  $\pi_{ij}^t = \pi_{t-1,i} p(q_t = j | q_{t-1} = i)$ 。最后，对每个  $j$ ，把这  $K$  个高斯混合**合并**成一个混合得到

$$b_t^j = p(z_t, q_t = j | y_{1:t}) = \pi_{t,j} N(z_t | \mu_{t,j}, \Sigma_{t,j}) \quad (18.136)$$

图 18.13(a) 给出了示意。把一个高斯混合用**单个高斯**来近似的最优方式是解

$$q = \arg \min_q \text{KL}(q \| p), \quad p(z) = \sum_k \pi_k N(z | \mu_k, \Sigma_k), \quad q(z) = N(z | \mu, \Sigma)$$

它可以通过**矩匹配**求出：

$$\mu = E[z] = \sum_k \pi_k \mu_k \quad (18.137)$$

$$\Sigma = \text{cov}[z] = \sum_k \pi_k \Sigma_k + (\mu_k - \mu)(\mu_k - \mu)^\top \quad (18.138)$$

在图模型文献中，这叫**弱边缘化**（Lauritzen 1992），因为它保留了一二阶矩。把这些式子用于我们的模型，可从  $b_t^{ij}$  得到  $b_t^j$ （为简洁起见省略下标  $t$ ）：

$$\pi_j = \sum_i \pi_{ij} \quad (18.139)$$

$$\pi_{j|i} = \frac{\pi_{ij}}{\sum_j \pi_{ij}} \quad (18.140)$$

$$\mu_j = \sum_i \pi_{j|i} \mu_{ij} \quad (18.141)$$

$$\Sigma_j = \sum_i \pi_{j|i} (\Sigma_{ij} + (\mu_{ij} - \mu_j)(\mu_{ij} - \mu_j)^\top) \quad (18.142)$$

该算法每步需要运行  $K^2$  个滤波器。更便宜的替代是：每步把信念状态表示为**单个高斯**，即在每步**边际化离散开关**——这是 ADF 的直接应用。它的一个**离线扩展**叫 expectation correction (Barber 2006; Mesot and Barber 2009)。另一个启发式方法叫**交互式多模型** (IMM) (Bar-Shalom and Fortmann 1988)：先把先验**合并成单个高斯** (矩匹配)，然后用对应于每个  $q_t$  值的  $K$  个卡尔曼滤波器分别进行更新。示意见图 18.13(b)。

## 18.6.2 应用：数据关联与多目标跟踪

假设我们在跟踪  $K$  个目标 (如飞机)，在时刻  $t$  观测到  $K'$  个检测 (例如雷达屏上的“光点”)。由于遮挡或漏检可有  $K' < K$ ；由于杂波或虚警可有  $K' > K$ ；也可能  $K' = K$ 。无论如何，我们需要确定这  $K'$  个检测  $y_{t,k}$  与  $K$  个目标  $z_{t,j}$  之间的**对应关系**，这叫**数据关联**问题，出现在许多应用领域。图 18.14 给出了  $K = 2$  的例子：每个时间步， $q_t$  是未知的**映射**，它说明哪个目标产生了哪个观测；它决定了该时间切片的**连线图**。标准做法是为目标  $j$  与测量  $k$  计算一个**权重**，度量它们的“**相容性**” (通常基于测量  $k$  与模型预测的目标  $j$  位置有多接近——所谓最近邻启发式)。这得到一个  $K' \times K$  的权重矩阵。通过加入**虚拟背景目标** (解释所有虚警) 与**虚拟观测** (解释所有漏检)，我们把它扩展为  $N \times N$  的方阵，其中  $N = \max(K, K')$ 。然后用**匈牙利算法**求最大权匹配，时间复杂度  $O(N^3)$  (参见 Burkard 等, 2009)。在此条件下，我们做一次卡尔曼滤波更新；被分配到“虚拟观测”的目标**不做测量更新**。该方法的扩展，用来处理**数量可变/未知**的目标，称为**多目标跟踪**。这需要处理**可变维度**的状态空间；很多方法可行，其中较简单稳健的是基于**序贯蒙特卡洛** (如 Ristic 等, 2004) 或 MCMC (如 Khan 等, 2006; Oh 等, 2009)。

## 18.6.3 应用：故障诊断

考虑图 18.15(a) 的模型：它代表一个由若干**液体水箱**与**管道**构成的工业装置。本例为简化只有两个水箱。我们希望基于对各水箱进/出流量的带噪观测来估计每个水箱内部的**压力**。然而**测量装置可能失效**，而且**管道可能破裂或堵塞** (称为**阻力故障**)。该模型在故障诊断领域被广泛用作基准 (Mosterman and Biswas 1999)。图 18.15(b) 给出了该系统的概率模型：**方块是离散变量** (如测量故障与阻力故障)，其余为**连续变量**。可以对该模型应用多种近似推断算法；见 (Koller and Lerner 2001) 的一种方法，基于Rao-Blackwellized **粒子滤波** (23.6 节将解释)。

## 18.6.4 应用：计量经济学预测

切换 LG-SSM 在计量经济预测中使用广泛，被称为**机制/状态切换模型**。例如，我们可以把两个**线性趋势模型**（见 18.2.4.2）结合起来：一个满足  $b_t > 0$  的“经济增长”机制，另一个是  $b_t < 0$  的“经济收缩”机制。更多细节见 West and Harrison (1997)。

---

## 通俗解释（直觉版）

### 1) 什么是“混合 SSM/切换 LDS”？

把离散的“模式/开关”  $q_t$ （比如“传感器坏/好”，“经济扩张/萎缩”）和**连续的物理状态**  $z_t$ （位置、速度、压力……）放在一个模型里：

- 一旦  $q_t = k$  被确定，连续部分就按**某个线性-高斯模型**运行（矩阵  $A_k, B_k, C_k, D_k, Q_k, R_k$ ）。
- 但  $q_t$  自己按**马尔可夫链**在模式之间切换（用  $A_{ij}$  给出从  $i$  到  $j$  的概率）。

### 2) 为什么推断很难？

因为**每个模式对应一个高斯**，时间滚动时要把所有可能的模式序列都考虑进去：

- 2 个模式  $\Rightarrow$  第 1 步最多 2 个高斯；第 2 步变 4 个；第 3 步 8 个……**指数爆炸**。
- 所以不能像纯 LG-SSM 那样一次卡住。

### 3) 怎么近似着做？

三条经典路子：

- 剪枝 (MHT)**：只保留高概率的模式路径，低的丢掉。
- 蒙特卡洛：采样**离散路径，每条路径上对连续部分用**精确卡尔曼**（这就是 Rao-Blackwellization 思想）。
- 高斯和/ADF 系列**：保留**少量高斯混合**，每步\*\*合并\*\*（矩匹配）\*\*控制数量\*\*。

GPB2 与 IMM 的差别（图 18.13）：

- GPB2：把每个“上一步模式  $i$ ”分别通过“本步模式  $j$ ”的模型（共  $K^2$  条支路） $\rightarrow$  得到  $b_t^{ij} \rightarrow$  对每个  $j$  合并（矩匹配）成一个高斯  $b_t^j$ 。**更准，但每步要跑  $K^2$  个滤波器**。
- IMM：先把“上一步所有模式的先验”**合并成一个高斯**，再并行跑  $K$  个滤波器（每个对应不同  $j$ ）。**更省算力**，效果通常也不错，是工程常用方案。

### 4) 两个代表性应用

- 数据关联 / 多目标跟踪**（图 18.14）：每步先解“哪个观测属于哪个目标”的匹配问题（构造权重矩阵 + **匈牙利算法**  $O(N^3)$ ，用“虚拟目标/观测”处理虚警与漏检），再对每个匹配的目标做卡尔曼更新。目标数量可变时常用**粒子滤波/MCMC**扩展。

- 故障诊断（图 18.15）：把“是否故障”建成离散结点，把“压力/流量/阻力”建成连续结点 → 混合图模型。常用Rao-Blackwellized 粒子滤波：对离散部分采样、对连续部分解析更新。

## 5) 经济学里的“机制切换”

用离散变量表示“增长/衰退”等宏观机制，每个机制下是不同的线性趋势/噪声；模型能学到何时切换，提升预测。

---

## 要点总结

- 混合 SSM = 离散模式  $q_t$  + 连续状态  $z_t$ 。切换 LDS/SSSM 是其代表。
- 难点：信念是“高斯混合”，模态随时间指数增长 → 精确推断不可行。
- 近似法：
  1. MHT：剪枝低概率模式路径；
  2. 粒子法：采样离散路径 + 连续部分用卡尔曼 (Rao-Blackwellized)；
  3. 高斯和/ADF：把大量高斯合并成少量（矩匹配保留均值/协方差）。
- GPB2：每步跑  $K^2$  个滤波器后合并到  $K$  个（更准更贵）。
- IMM：先合并再跑  $K$  个滤波器（更省算力，工程常用）。
- 数据关联：先做最大权二分匹配（匈牙利算法  $O(N^3)$ ），再做卡尔曼更新；扩展到多目标跟踪用 SMC/MCMC。
- 故障诊断：把“是否故障”建成离散变量、物理量建成连续变量，常用 RB 粒子滤波。
- 经济预测：机制切换模型（增长/收缩）是切换 LDS 的常见应用。

这一节主要在讲什么？我不太懂

已思考 13s

这节（18.6：离散/连续混合的状态空间模型）主要讲：

## 一句话版

把“**离散开关/模式**”（如：增长/衰退、传感器好/坏、目标A或B）和“**连续状态**”（位置、速度、压力等）放到同一个模型里——这就叫**混合 SSM**。其中最典型的是**切换线性动态系统 (SLDS/SSSM)**：离散变量决定当前用哪套线性高斯模型来演化连续状态。

## 主要内容（通俗版）

## • 模型长什么样

每个时刻有一个离散状态  $q_t$  (模式/工况) 和一个连续状态  $z_t$ 。给定  $q_t = k$  时, 连续部分按“第  $k$  套”线性一高斯方程运动与观测 (矩阵  $A_k, B_k, C_k, \dots$ )。

## • 为什么难

后验不再是一个高斯, 而是**高斯混合**, 而且**模态数随时间指数增长** (两种模式: 第1步最多2个、第2步4个、第3步8个...) , 精确推断不可行。

## • 怎么办 (近似推断三路)

1. **剪枝(MHT)**: 只保留高概率的离散模式路径。
2. **蒙特卡洛/粒子法**: **采样离散路径**, 在给定路径下对连续部分用卡尔曼 (Rao-Blackwellized PF)。
3. **高斯和/ADF 系列**: 用**少量高斯混合去近似大量混合**; 典型做法:
  - **GPB2**: 每步跑  $K^2$  个卡尔曼 (上一模式  $i \times$  这一模式  $j$ ) , 再对每个  $j$ **矩匹配合并** (更准更贵)。
  - **IMM**: 先把先验合成一个高斯, 再并行跑  $K$  个卡尔曼 (更省算力, 工程常用)。

## • 应用举例

- **数据关联/多目标跟踪**: 观测点数与目标数不一定相等, 先用**匈牙利算法**做最大权匹配 (处理虚警/漏检可加“虚拟”点), 再对匹配的目标做卡尔曼更新; 目标数量可变时常用 SMC/MCMC。
- **故障诊断 (双水箱)** : 离散结点表示“测量/管道是否故障”, 连续结点表示压力、流量等; 常用 RB 粒子滤波。
- **经济预测**: **机制切换** (增长/衰退) 模型, 本质就是切换 LDS。

## 记住这三点

1. **混合 = 离散模式  $q_t$  + 连续状态  $z_t$** ; 模式决定用哪套线性高斯。
2. **难点是高斯混合数目爆炸**, 精确滤波不现实。
3. **常用近似**: MHT 剪枝、粒子法 (采样离散+解析连续) 、高斯和/IMM (矩匹配合并)。

基于本对话里我贴过的整章原文, 输出: ①知识点总结  
(分点+关键词+用途) ②子章节关联 (依赖/对比/递进+关系句, 附Mermaid可选) ③本章总结。

已思考 26s

太棒了，你已经把第 18 章的核心材料都贴给我了。下面按你要的格式给出整章的“高能浓缩版”。

---

## ① 知识点总结 (分点 | 关键词 | 用途)

- **状态空间模型 (SSM)**
  - 关键词：隐藏状态  $z_t$ 、输入  $u_t$ 、观测  $y_t$ 、转移  $g$ 、观测  $h$
  - 用途：把“随时间演化的系统”统一建模，用于跟踪、预测、控制、时序分析。
- **线性-高斯 SSM (LG-SSM / LDS)**
  - 关键词：线性转移  $A, B$ 、线性观测  $C, D$ 、噪声  $Q, R$ 、高斯性
  - 用途：支持精确推断（卡尔曼滤波/平滑），是许多工程系统的默认模型。
- **卡尔曼滤波 (KF)**
  - 关键词：预测 ( $\mu_{t|t-1}, \Sigma_{t|t-1}$ )、更新 (残差  $r_t$ 、增益  $K_t$ )、Riccati
  - 用途：在线估计当前状态  $p(z_t|y_{1:t})$ ，降噪、实时跟踪、一步预测。
- **卡尔曼平滑 (RTS / Smoother)**
  - 关键词：反向增益  $J_t, \mu_{t|T}, \Sigma_{t|T}$
  - 用途：离线重估整条轨迹  $p(z_t|y_{1:T})$ ，不确定性更小，轨迹更平滑。
- **边际似然与后验预测**
  - 关键词： $\log p(y_{1:T})$ 、 $p(y_{t+1}|y_{1:t})$
  - 用途：模型比较、超参数选择、短期预测与异常检测。
- **计算与数值稳定**
  - 关键词：矩阵求逆  $O(|y|^3)$ 、信息滤波、平方根滤波
  - 用途：在高维/长序列下保证稳定、加速更新。
- **应用：目标跟踪**
  - 关键词：常速度/随机加速度、椭圆不确定度、滤波 vs 平滑
  - 用途：雷达/视觉跟踪，实时估计位置与速度。
- **应用：SLAM (定位与建图)**
  - 关键词：机器人位姿  $x_t$ 、地标  $L$ 、回环检测、稀疏精度矩阵
  - 用途：移动机器人/室内导航；引出稀疏近似、FastSLAM 等。
- **应用：在线参数学习 (RLS/LMS)**
  - 关键词：把参数当状态、Kalman 增益、递推最小二乘 (RLS)、LMS
  - 用途：流式回归/自适应滤波，单遍收敛、自动步长。

- **应用：结构化时间序列 (STS)**
  - 关键词：局部水平、局部趋势、季节项（和约束）、可加性
  - 用途：可解释预测，组件级别诊断；与 ARMA 对照（SSM 表达更直观、可非平稳）。
- **近似在线推断（非线性/非高斯）总览**
  - 关键词：用高斯近似后验；两路思路：线性化 vs 矩匹配
  - 用途：当  $g, h$  非线性或噪声非高斯时仍能实时推断。
- **EKF（扩展卡尔曼）**
  - 关键词：一阶泰勒、雅可比、在  $\mu$  处线性化、迭代 EKF
  - 用途：轻量、实现简单；先验很宽或强非线性时可能失真。
- **UKF（无迹卡尔曼）**
  - 关键词：Sigma 点、无导数、二阶精度、无迹变换
  - 用途：比 EKF 稳准，免求导；适合强非线性或难求雅可比的场景。
- **ADF（假设密度滤波）**
  - 关键词：预测→更新→投影、KL 最小化、矩匹配、指数族
  - 用途：把“难后验”投影回**可处理族**（如对角高斯/离散乘积），适配 DBN、GLM 等。
- **Boyen–Koller（离散 DBN 在线）**
  - 关键词：全因子化近似、误差有界
  - 用途：大规模离散动态系统的快速边缘估计。
- **GLM 在线参数（逻辑回归示例）**
  - 关键词：把  $s = \theta^\top x$  变成 1D、数值积分/高斯求积、 $O(D)$  更新
  - 用途：TrueSkill、在线分类/排序；大规模、流式友好。
- **混合 SSM / 切换 LDS (SLDS/JMLS)**
  - 关键词：离散模式  $q_t$  + 连续状态  $z_t$ 、模式专属  $(A_k, C_k, Q_k, R_k)$
  - 用途：“工况/机制切换”系统、传感器好坏、多模型目标运动。
- **混合模型的推断难点**
  - 关键词：**高斯混合模态数指数爆炸**
  - 用途：说明精确解不可行，需近似。
- **近似：MHT / 粒子法 / 高斯和 (GPB2、IMM)**
  - 关键词：剪枝、Rao–Blackwellized PF、矩匹配合并、 $K^2$  vs  $K$  滤波器
  - 用途：多目标跟踪、机制切换识别、工程可落地的折中方案。
- **数据关联与多目标跟踪**

- 关键词：兼容性权重、匈牙利算法  $O(N^3)$ 、虚拟观测/目标
  - 用途：把“哪个观测属于哪个目标”与状态更新解耦，鲁棒跟踪。
  - **故障诊断（两水箱）**
    - 关键词：离散故障结点（阻力/测量故障）+ 连续物理量、RBPF
    - 用途：工业监测、异常定位、根因分析。
  - **经济计量中的机制切换**
    - 关键词：增长/衰退两趋势、切换时间、预测
    - 用途：场景化预测、不同时期用不同线性模型。
- 

## ② 子章节关联（依赖 | 对比 | 递进）

### 关系句（文字版）

- 18.1 定义 → 奠基所有后续；18.3 推断建立在 18.1 的 LG-SSM 之上，给出 KF/RTS 的精确算法。
- 18.2 应用 ← 直接使用 18.3 的推断器（目标跟踪、SLAM、RLS、STS 等）展示威力与直觉。
- 18.4 学习 ↔ 依赖 18.3 的推断（E 步需要平滑期望）来估计参数（EM/子空间/贝叶斯）。
- 18.5 近似推断 ↑ 当 18.3 的线性高斯假设不再成立时，给出 EKF/UKF/ADF 的替代路线；与粒子滤波（第 23 章）形成对比。
- 18.6 混合模型 → 在 18.5 的近似思想上再前进一步，把离散模式并入，使用 MHT/SMC/高斯和（GPB2、IMM）应对指数混合难题；应用扩展到数据关联、故障诊断、经济切换。

### 对比链接

- KF vs RTS：在线 vs 离线、均值同形但后者更小方差。
- 结构化时间序列 vs ARMA：可解释组件（水平/趋势/季节）vs 纯自回归-移动平均；二者均可写成 SSM。
- EKF vs UKF：线性化（需雅可比、一阶精度）vs Sigma 点（免导、二阶精度）。
- GPB2 vs IMM：精度 ( $K^2$  滤波) vs 复杂度 ( $K$  滤波)，均用矩匹配合并。

### （可选）Mermaid 总览图

```
graph LR A[18.1 定义: SSM/LG-SSM] --> B[18.3 推断: KF/RTS] B --> C[18.2 应用: 跟踪/SLAM/RLS/STS] B --> D[18.4 学习: EM/子空间/贝叶斯] B --> E[18.5 近似:]
```

EKF/UKF/ADF] E --> F[18.6 混合: SLDS/JMLS] F --> G[应用: 数据关联/多目标/故障/经济切换] C ..对比.-> H[ARMA] E -.对比.-> I[粒子滤波(第23章)]

### ③ 本章总结（一段话掌握全局）

第18章把“时间动态+不确定性”统一装进状态空间模型。当系统线性+高斯时，用卡尔曼滤波/平滑可做精确的在线估计与离线重构，并服务于跟踪、SLAM、在线回归、结构化时序预测等实战。参数未知时，可用EM/子空间/贝叶斯学习。当系统非线性或非高斯，改用EKF/UKF/ADF等高斯近似；而当还存在离散模式切换时，后验变成高斯混合并指数增长，需要MHT/粒子法或\*\*高斯和(GPB2/IMM)\*\*等近似策略。整章给出了一条清晰路线：建模→推断（精确/近似）→学习→应用，并指出何时该选哪把“滤波器”，在工程与数据科学中形成一套可落地的动态建模手册。

请用通俗易懂的话总结本章，使我能够理解

已思考 18s

下面用“看不清的汽车”来打个比方，把整章讲清楚：

### 这章想解决的根本问题

真实世界里很多东西看不见或看不全：车子在雾里行驶（真实位置、速度是隐藏状态），我们只能看到带噪声的“雷达点/相机点”（观测）。

\*\*状态空间模型(SSM)\*\*就是一本说明书：告诉你

- 隐藏状态怎么随时间变化（动力学）；
- 你能看到什么（观测模型）；
- 怎么用这些观察估计现在/过去/未来。

### 最重要的一类：线性 + 高斯 → 卡尔曼一家

当“状态变化是线性的、噪声是高斯的”，一切特别好办：

- **卡尔曼滤波（在线）**：每一步两件事
  1. **预测**：根据上一步的状态，推到这一步（像开盲盒前先猜）。
  2. **更新**：拿到这一步的观测，和预测折中（权重由“传感器噪声 vs 先验不确定”自动决定）。

结果：得到“这一刻最靠谱的估计 + 不确定度（方差/椭圆）”。

- **卡尔曼平滑（离线）**：等整段数据都到了，再往回修正（未来信息也帮到过去），轨迹更平滑、误差更小。
  - 你还能顺手得到：
    - **一步预测**（拿来做短期预报、异常检测），
    - **序列的似然**（做模型比较）。
  - **应用**：目标跟踪、SLAM（建图与定位）、在线回归（递推最小二乘 RLS）、结构化时间序列（水平/趋势/季节叠加，直观可解释）。
- 

## 如果不是线性或不是高斯：三条近似路

现实常常**非线性/有异常噪声**，这时精确卡尔曼不行了，但还有好用的近似：

1. **EKF（扩展卡尔曼）**：把非线性函数在当前估计点做**一阶线性化**再套卡尔曼。
    - 优点：简单快速。
    - 易翻车：先验很宽、函数很弯时会失真，还得算雅可比。
  2. **UKF（无迹卡尔曼）**：不拉直函数，改用少量Sigma 点代表高斯，把这些点穿过**真正的非线性**，再用**矩匹配**恢复均值/协方差。
    - 优点：**更准、免求导**，对强非线性更稳。
  3. **ADF（假设密度滤波）**：每步先**真更新**得到复杂后验，再把它**投影成好处理的形状**（比如高斯/因子化分布），通过**最小化 KL或矩匹配**实现。
    - 能搞：大规模离散 DBN 的在线推断（Boyen–Koller），GLM/逻辑回归的在线参数学习（每步  $O(D)$ ）。
- 

## 离散 + 连续一起上：切换模型（混合 SSM）

有时系统有\*\*“模式/工况切换”\*\*：增长/衰退、传感器好/坏、物体走直线/转弯……

这时引入离散变量  $q_t$  表示模式；给定模式，用对应的一套线性高斯去更新连续状态  $z_t$ 。

- 难点：后验变成**高斯混合**，模式数随时间**指数爆炸**，精确推断不现实。
- 常用招：
  - **MHT**：剪枝掉低概率的模式路径；
  - **粒子法（Rao–Blackwellized）**：采样离散路径，连续部分走解析卡尔曼；
  - **高斯和/工程方案**：
    - **GPB2**：每步跑  $K^2$  个滤波器后**矩匹配合并**（更准更贵）；

- IMM：先把先验合成一个高斯，再并行跑  $K$  个滤波器（更省算力，工程常用）。
  - 典型应用：
    - **数据关联/多目标跟踪**：先用匈牙利算法把“观测点 $\rightarrow$ 目标”匹配，再做卡尔曼；目标数可变时用 SMC/MCMC。
    - **故障诊断（双水箱）**：离散结点=是否故障，连续结点=压力/流量；常用 RB 粒子滤波。
    - **经济计量：机制切换**（扩张/衰退）组合不同趋势模型。
- 

## 参数怎么学

- 已知结构时，常只学噪声方差  $Q, R$ （有时在线也能贝叶斯更新）。
  - 若要同时学  $A, C$ ：注意**可辨识性/稳定性**（特征值别让系统爆掉）；
  - 算法：**EM**（E 步用平滑器）、**子空间方法**（先用 PCA 找状态子空间，再拟合）、**贝叶斯**（变分/Gibbs，或在线版本）。
- 

## 选型小抄（实用）

- **线性 + 高斯**  $\rightarrow$  直接 KF / RTS。
  - **非线性，能算雅可比且不太弯**  $\rightarrow$  EKF；更弯/不想算导  $\rightarrow$  UKF。
  - **模型很大或必须极快在线**  $\rightarrow$  ADF 类（GLM/DBN）。
  - **有模式切换**  $\rightarrow$  IMM / GPB2 / **粒子法**（数据关联配匈牙利算法）。
  - **要可解释的时序预测**  $\rightarrow$  **结构化时间序列（水平+趋势+季节）**；要传统基线可用 ARMA（也能写成 SSM）。
- 

## 一口气的总括

这章给了你一套统一的时序建模思路：

**用 SSM 描述世界**  $\rightarrow$  **用（卡尔曼/近似）滤波与平滑估计状态**  $\rightarrow$  **学参数**  $\rightarrow$  **上应用**。

只要记住：**线性高斯用卡尔曼，非线性用 EKF/UKF，离散切换用 IMM/粒子，复杂后验用 ADF 投影**。这样你就能在“看不清的世界”里，持续、稳定地定位现在、回溯过去、预测未来。