Hidden Markov Models and State Estimation

36-467/36-667

19 November 2020 (to accompany Lecture 23)

Contents

Opening: Noisy Observations of Markov Models Are Not Markovian **1**

Definition of HMMs **4**

Some Basic Calculations for HMMs **5**

The Recursive Trick for Likelihood, Prediction, and State Estimation **6**
- Back to the Likelihood; Time Complexity of the Forward Algorithm 8
- The Backward Algorithm 8
- The Kalman Filter 9
- The Particle Filter 9

Parameter Estimation **12**
- Some Important Special Cases 15

Exercises **15**

References **16**

The last few lectures have focused on Markov processes, where conditioning on the whole past is equivalent to conditioning on the most recent value:

$$\mathbb{P}\left(X(t+1)|X(1:t)\right) = \mathbb{P}\left(X(t+1)|X(t)\right)$$

When this is true, we say that $X(t)$ is the **state** of the process at time t, the variable which determines the whole distribution of future observations. When this assumption holds, we can easily do likelihood-based inference and prediction. But the Markov property commits us to $X(t+1)$ being independent of all earlier X's given $X(t)$.

Opening: Noisy Observations of Markov Models Are Not Markovian

The most natural route from Markov models to hidden Markov models is to ask what happens if we don't observe the state perfectly. I have been using $X(t)$ to always stand for the time series we observe, so I am going to introduce a new stochastic process, $S(t)$, which will be a Markov process, and say that what we observe is $X(t)$ plus independent noise. Will this $X(t)$ be Markov?

To be concrete, consider the following set-up. $S(t)$ is a Markov chain with two states, -1 and $+1$, and $\mathbb{P}\left(S(t+1) = +1|S(t) = +1\right) = \mathbb{P}\left(S(t+1) = -1|S(t) = -1\right) = q > 0.5$. This chain thus alternates between two states, but tends to stick in the same state for a while, to be "persistent". What we observe is

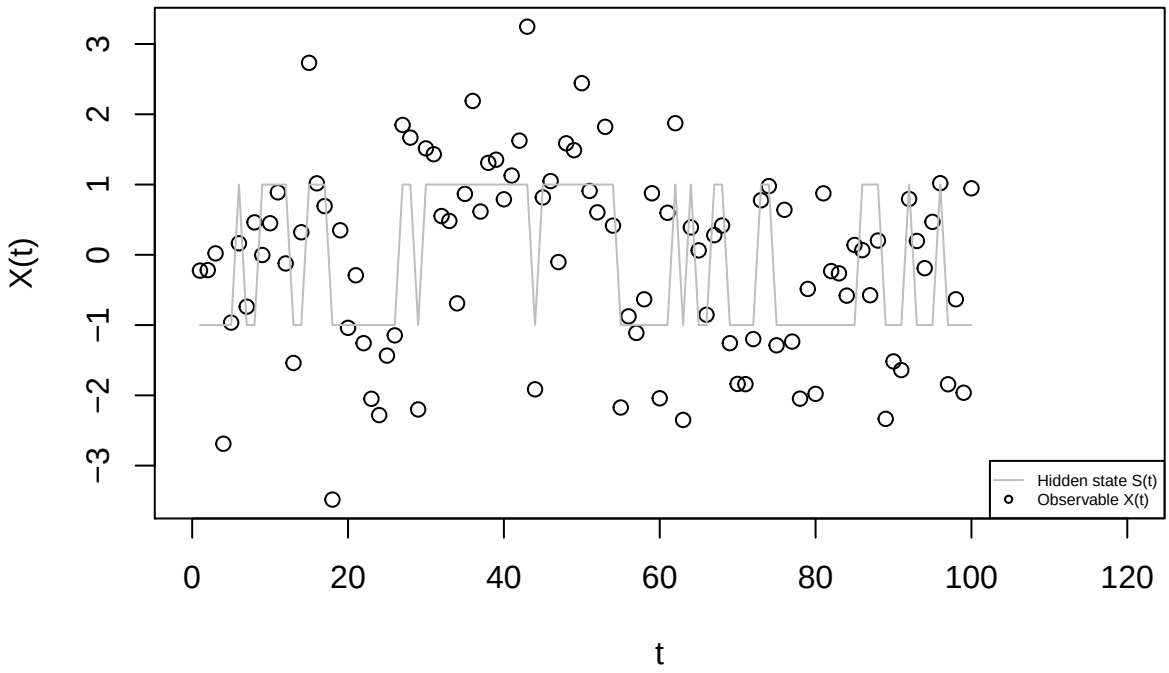
$X(t) = S(t) + \epsilon(t)$, where $\epsilon(t)$ are IID Gaussian noise variables with mean 0 and variance 1. (Nothing particularly turns on the choice of Gaussian noise or variance 1, etc.) The figure below illustrates both a typical realization of the observables $X(t)$, and of the chain $S(t)$ that generated it.

```
# From lecture 18: generic, abstract Markov simulator
rmarkov <- function(n, rinitial, rtransition) {
  x <- vector(length=n)
  x[1] <- rinitial()
  for (t in 2:n) {
    x[t] <- rtransition(x[t-1])
  }
  return(x)
}

# Simulate a chain that's +1/-1, with persistence in state
  # Pr(X(t+1)=+1|X(t)=+1) = Pr(X(t+1)=-1|X(t)=-1) = q, assumed > 0.5
  # Exercise: Prove that the invariant distribution is (1/2, 1/2) for all q
rpmchain <- function(n, q) {
    # Take one step of the chain
    transition.with.persistence <- function (x) {
        # Stay in the same state with probability q
        stay <- sample(c(TRUE, FALSE), size=1, prob=c(q, 1-q))
        if (stay) { return(x) }
        # Or flip to the opposite state
        if (!stay) { return(-x) }
    }
    pm.seq <- rmarkov(n, rinitial=function() { sample(c(-1, +1),
                                                      size=1) },
                      rtransition=transition.with.persistence)
    return(pm.seq)
}

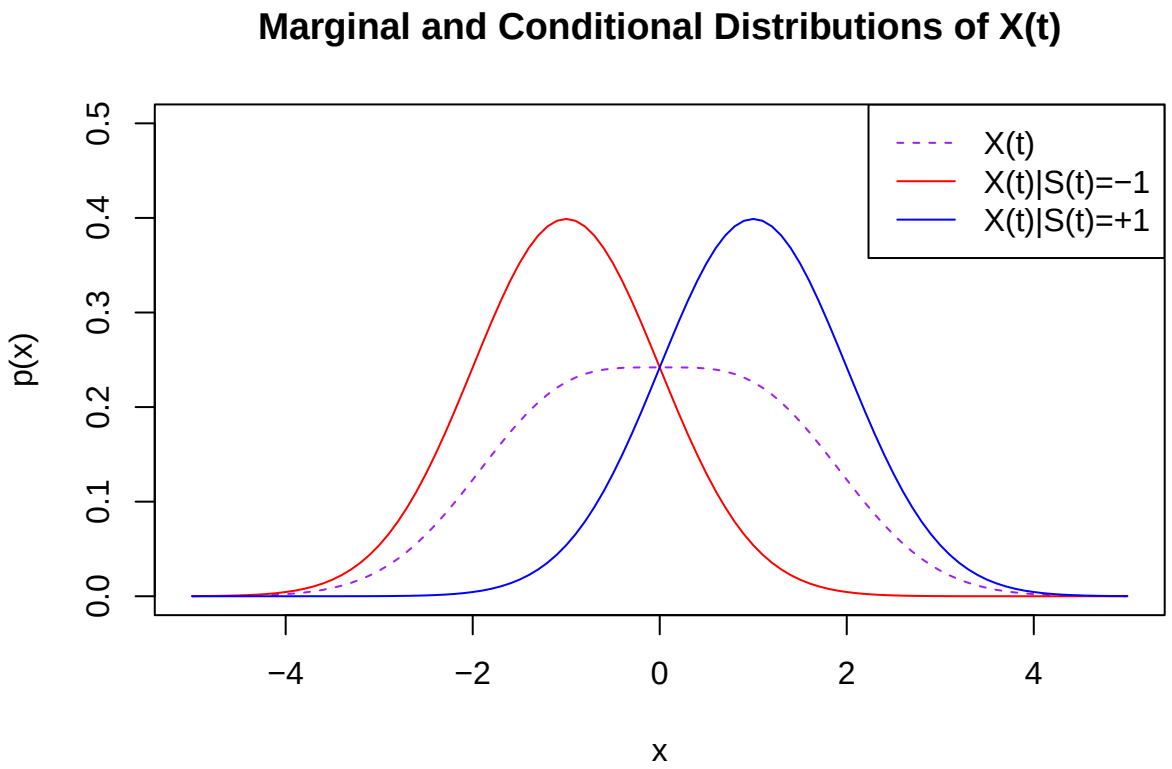
# Simulate a +1/-1 chain, but observe it through standard-Gaussian noise
noisy.rpmchain <- function(n, q, sd=1) {
    pm.seq <- rpmchain(n, q)
    x <- rnorm(n=n, mean=pm.seq, sd=sd)
    return(list(x=x, s=pm.seq))
}
```

```
sim <- noisy.rpmchain(n=100, q=0.75)
plot(sim$x, xlab="t", ylab="X(t)", xlim=c(0, 120))
lines(1:100, sim$s, col="grey")
legend("bottomright", legend=c("Hidden state S(t)", "Observable X(t)"),
       pch=c(NA, 1), lty=c("solid", "blank"), col=c("grey", "black"),
       cex=0.5)
```



Now any value of $X(t)$ *could* have come from either value of $S(t)$. (See next figure.) Large positive values of $X(t)$ are *really unlikely* if $S(t) = -1$, but not impossible, and values of $X(t)$ near zero are about equally likely under either state.

```
curve(0.5*dnorm(x,-1,1)+0.5*dnorm(x,1,1), xlab="x", ylab="p(x)",
      main="Marginal and Conditional Distributions of X(t)", xlim=c(-5,5), ylim=c(0, 0.5), lty="dashed"
curve(dnorm(x,-1,1), add=TRUE, col="red")
curve(dnorm(x,+1,1), add=TRUE, col="blue")
legend("topright", legend=c("X(t)", "X(t)|S(t)=-1", "X(t)|S(t)=+1"),
       lty=c("dashed", "solid", "solid"), col=c("purple","red","blue"))
```



This has consequences when we ask about the distribution of $X(t+1)$. If $X(t) = 3.2$, then it's very likely that $S(t) = +1$ — so likely that earlier history won't change much of anything. But if $X(t) = 0$, well, it will make a difference whether $X(t-1)$ was positive or negative.

This simple observation is enough, in itself, to tell us that $X(t)$ can't be a Markov process. In any Markov process, $X(t-1)$ is irrelevant to $X(t+1)$ given $X(t)$. In particular, if X is numerical and we regress $X(t+1)$ on both $X(t)$ and $X(t-1)$, the true coefficient on $X(t-1)$ is *exactly* zero. This is *not* the case here (Exercise 1).

Quite generally, if $S(t)$ is a Markov process, and $X(t)$ is a function of $S(t)$, plus, possibly, some extra noise, then $X(t)$ is *not* Markov[1].

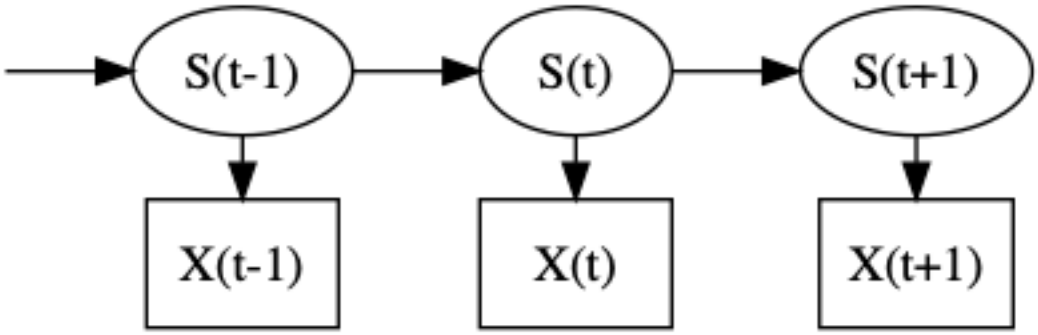
Definition of HMMs

Examples like these lead to a general notion of a **hidden Markov model**, or **state-space model**. In these models, there is a **latent** or **hidden** state $S(t)$, which follows a Markov process. We'll write $\mathbb{P}(S(t+1) = r|S(t) = s) = q(s,r)$. As in Markov models, the transitions need to be complemented with a distribution for the initial state. Because these states are hidden, we don't actually get to observe them. The **observable** or **manifest** variable $X(t)$ is a random or noisy function of $S(t)$. That is, there's some sequence of IID noise variables, say $\epsilon(t)$, and $X(t) = f(S(t), \epsilon(t))$ for some function f. As a consequence, given $S(t)$, $X(t)$ is independent of $X(t'), S(t')$ for all $t' \neq t$. An *example* of this would be when $X(t) = aS(t) + \epsilon(t)$, but the function can be nonlinear and the noise can be non-additive. In fact, the exact function f usually doesn't matter very much; what matters are the conditional observation probabilities $\mathbb{P}(X(t) = x|S(t) = s) = g(s,x)$.

[1] This observation suggests the interesting mathematical question of whether *any* functions of a Markov process are themselves Markovian, and, if so, which. This has been studied by a variety of authors, often with a view to "lumping" the states of large simulation models. Rosenblatt (1971) and Rogers and Pitman (1981) are particularly notable contributions to this problem; Nilsson Jacobi and Görnerup (2007) gives a necessary-and-sufficient algebraic condition. On the related problem of characterizing the processes which are functions of Markov chains (whether or not the result is Markovian), see Heller (1965) and Erickson (1970).

Some comments are in order on this definition.

- Both $S(t)$ and $X(t)$ can be numerical or qualitative variables, or indeed more complicated.
  - Some people make a distinction between "hidden Markov models", where $S(t)$ is discrete, and "state-space models", where $S(t)$ is continuous. Some of these people would further limit "state-space models" to situations where $X(t)$ is also continuous. Since the theory and methods are almost the same, however, I will use the terms interchangeably.
  - Obviously, when variables are continuous, probability mass functions need to get replaced by probability density functions, and sums by integrals.
- Both $S(t)$ and $X(t)$ can have multiple dimensions. I won't indicate this in the notation.
- Any type of state variable can, potentially, be combined with any type of observable. For instance, you can have discrete $S(t)$ and continuous-function-valued $X(t)$. (These are sometimes used in speech recognition, where the discrete states are the qualitative linguistic sounds, the "phonemes", which people are saying, and the function-valued observations are the actual continuous noises they make over brief intervals of time.)
- HMMs can be extra *useful* when the state is somehow simpler than the observable — say a low-dimensional or even binary state explaining a high-dimensional vector of observables — but that's not required by the definition.
  - In particular, if $S(t)$ has r dimensions, but we observe $k < r$ of them, the variables we *observe* will not be Markov, even without noise.



If you are familiar with graphical models, this may help.

Some Basic Calculations for HMMs

Because the states $S(t)$ form a Markov process, we can easily calculate the probability of a state sequence:

$$\begin{aligned}
\mathbb{P}\left(S(1:n) = s(1:n)\right) &= \mathbb{P}\left(S(1) = s(1)\right) \prod_{t=2}^{n} S(t) = s(t) | S(1:t-1) = s(1:t-1) && (1)\\
&= \mathbb{P}\left(S(1) = s(1)\right) \prod_{t=2}^{n} S(t) = s(t) | S(t-1) = s(t-1) && (2)\\
&= \mathbb{P}\left(S(1) = s(1)\right) \prod_{t=2}^{n} q(s(t-1), s(t)) && (3)
\end{aligned}$$

The first line is basic probability; the second uses the Markov property.

It's equally easy to calculate the probability of a sequence of observations, given a sequence of states:

$$\mathbb{P}\left(X(1:n) = x(1:n) | S(1:n) = s(1:n)\right) = \prod_{t=1}^{n} \mathbb{P}\left(X(t) = x(t) | S(t) = s(t)\right) \quad (4)$$

$$= \prod_{t=1}^{n} g(s(t), x(t)) \quad (5)$$

This is because $X(t)$ is independent of everything else, given $S(t)$.

Putting these together gives us the joint probability of a sequence of states and a sequence of observations:

$$\mathbb{P}\left(X(1:n) = x(1:n), S(1:n) = s(1:n)\right) = \mathbb{P}\left(S(1:n) = s(1:n)\right) \mathbb{P}\left(X(1:n) = x(1:n) | S(1:n) = s(1:n)\right) \quad (6)$$

$$= \mathbb{P}\left(S(1) = s(1)\right) \prod_{t=2}^{n} q(s(t-1), s(t)) \prod_{t=1}^{n} g(s(t), x(t)) \quad (7)$$

This, in turn, in principle gives us the probability of a sequence of observations:

$$\mathbb{P}\left(X(1:n) = x(1:n)\right) = \sum_{s(1:n)} \mathbb{P}\left(X(1:n) = x(1:n), S(1:n) = s(1:n)\right) \quad (8)$$

$$(9)$$

This probability is a very valuable thing to know. When we make predictions, we want

$$\mathbb{P}\left(X(t) = x | X(1:t-1) = x(1:t-1)\right) = \frac{\mathbb{P}\left(X(1:t) = x(1:t)\right)}{\mathbb{P}\left(X(1:t-1) = x(1:t-1)\right)}$$

so if we can calculate the probability of a sequence, and we can divide, then we can predict. If q and g contain some unknown parameters θ, and we want to do inference on them, then we really should be writing things like

$$\mathbb{P}\left(X(1:n) = x(1:n); \theta\right)$$

and so calculating sequence probabilities will let us do likelihood-based inference.

Unfortunately, the sum-over-state-histories expression, $\sum_{s(1:n)} \mathbb{P}\left(X(1:n) = x(1:n), S(1:n) = s(1:n)\right)$, is just not suitable for direct evaluation. If there are even two states, there are 2^n possible values of $s(1:n)$, and that's clearly not a sum you want to do if n is at all large. The over-all time complexity of evaluating Eq. 7 directly, from the definition would be $O(nk^n)$, since we need $O(n)$ multiplications for each of 2^n state histories.

The Recursive Trick for Likelihood, Prediction, and State Estimation

a.k.a. "The Forward Algorithm"

There is a neat trick which lets us avoid the exponential explosion in evaluating the likelihood, or in making predictions. The same trick will also let us estimate the hidden state at a given time. The trick is to do all of these things **recursively**, based on having solved the same problem with less of the data.

Let's start with the base case. If we knew $S(1)$, it'd be easy to find the distribution of $X(1)$:

$$\mathbb{P}\left(X(1) = x(1) | S(1) = s\right) = g(s, x(1))$$

It's therefore easy to find the marginal distribution of $X(1)$, i.e., our predictive distribution for $X(1)$:

$$\mathbb{P}\left(X(1) = x(1)\right) = \sum_{s} \mathbb{P}\left(X(1) = x(1) | S(1) = s\right) \mathbb{P}\left(S(1) = s\right) = \sum_{s} g(s, x(1)) \mathbb{P}\left(S(1) = s\right)$$

Now we can use the definition of conditional probability to estimate the state at time 1:

$$\begin{aligned}
\mathbb{P}(S(1)=s|X(1)=x) &= \frac{\mathbb{P}(S(1)=s, X(1)=x(1))}{\mathbb{P}(X(1)=x(1))} && (10)\\
&= \frac{\mathbb{P}(S(1)=s)\,\mathbb{P}(X(1)=x(1)|S(1)=s)}{\mathbb{P}(X(1)=x(1))} && (11)\\
&= \frac{\mathbb{P}(S(1)=s)\,g(s,x(1))}{\sum_{s'}\mathbb{P}(S(1)=s')\,g(s',x(1))} && (12)
\end{aligned}$$

This is, of course, just Bayes's rule. Because this will be important later, let's abbreviate it by $F_1(s)$.

Now that we know (the distribution of) the state at time 1, we can extrapolate forward to time 2:

$$\begin{aligned}
\mathbb{P}(S(2)=r|X(1)=x(1)) &= \sum_s \mathbb{P}(S(2)=r, S(1)=s|X(1)=x(1)) && (13)\\
&= \sum_s \mathbb{P}(S(2)=r|S(1)=s, X(1)=x(1))\,\mathbb{P}(S(1)=s|X(1)=x(1)) && (14)\\
&= \sum_s \mathbb{P}(S(2)=r|S(1)=s)\,\mathbb{P}(S(1)=s|X(1)=x(1)) && (15)\\
&= \sum_s q(s,r)F_1(s) && (16)
\end{aligned}$$

using the Markov property in the next-to-last line, and the definitions of g and F_1 in the last line.

We want to extrapolate the state forward because it lets us make a prediction:

$$\begin{aligned}
\mathbb{P}(X(2)=x|X(1)=x(1)) &= \sum_s \mathbb{P}(X(2)=x, S(2)=s|X(1)=x(1)) && (17)\\
&= \sum_s \mathbb{P}(X(2)=x|S(2)=s, X(1)=x(1))\,\mathbb{P}(S(2)=s|X(1)=x(1)) && (18)\\
&= \sum_s \mathbb{P}(X(2)=x|S(2)=s)\,\mathbb{P}(S(2)=s|X(1)=x(1)) && (19)\\
&= \sum_s g(s,x)\mathbb{P}(S(2)=s|X(1)=x(1)) && (20)
\end{aligned}$$

using the assumption that $S(t)$ screens off everything else from $X(t)$ in the next-to-last line.

Finally, once we've seen that $X(2)=x(2)$, we can use that to get the distribution of $S(2)$:

$$\begin{aligned}
F_2(s) &\equiv \mathbb{P}(S(2)=s|X(2)=x(2), X(1)=x(1)) && (21)\\
&= \frac{\mathbb{P}(S(2)=s, X(2)=x(2)|X(1)=x(1))}{\mathbb{P}(X(2)=x(2)|X(1)=x(1))} && (22)\\
&= \frac{\mathbb{P}(X(2)=x(2)|S(2)=s, X(1)=x(1))\,\mathbb{P}(S(2)=s|X(1)=x(1))}{\mathbb{P}(X(2)=x(2)|X(1)=x(1))} && (23)\\
&= \frac{\mathbb{P}(X(2)=x(2)|S(2)=s)\,\mathbb{P}(S(2)=s|X(1)=x(1))}{\mathbb{P}(X(2)=x(2)|X(1)=x(1))} && (24)\\
&= \frac{g(s,x(2))\mathbb{P}(S(2)=s|X(1)=x(1))}{\mathbb{P}(X(2)=x(2)|X(1)=x(1))} && (25)
\end{aligned}$$

Again, this is Bayes's rule. Notice that the role of "prior distribution" here isn't played by $\mathbb{P}(S(1))$ or even $\mathbb{P}(S(1)|X(1))$, but by $\mathbb{P}(S(2)|X(1))$.

Now we're ready to go on to $t=3$, and to state the general, recursive pattern. This is sometimes called the **forward algorithm**.

- Begin with $\mathbb{P}(S(t)=s|X(1:t)=x(1:t)) \equiv F_t(s)$

- Extrapolate the state forward:

$$\mathbb{P}\left(S(t+1) = s | X(1:t) = x(1:t)\right) = \sum_{s'} q(s', s) \mathbb{P}\left(S(t) = s' | X(1:t) = x(1:t)\right) = \sum_{s'} q(s', s) F_t(s')$$

- Predict the next observation:

$$\mathbb{P}\left(X(t+1) = x | X(1:t) = x(1:t)\right) = \sum_{s} g(s, x) \mathbb{P}\left(S(t+1) = s | X(1:t) = x(1:t)\right)$$

- Condition $S(t+1)$ on $X(t+1)$:

$$\mathbb{P}\left(S(t+1) = s | X(1:t+1) = x(1:t+1)\right) = \frac{g(s, x(t+1)) \mathbb{P}\left(S(t+1) = s | X(1:t) = x(1:t)\right)}{\mathbb{P}\left(X(t+1) = x(t+1) | X(1:t) = x(1:t)\right)}$$

- End with $F_{t+1}(s) \equiv \mathbb{P}\left(S(t+1) = s | X(1:t+1) = x(1:t+1)\right)$, or begin the cycle over again with $t+2$.

Back to the Likelihood; Time Complexity of the Forward Algorithm

We began this by wanting an alternative to direct sum-over-state-histories for the likelihood $\mathbb{P}\left(X(1:n) = x(1:n)\right)$. We can get that from this approach, by multiplying together predictive distributions:

$$\mathbb{P}\left(X(1:n) = x(1:n)\right) = \mathbb{P}\left(X(1) = x(1)\right) \prod_{t=2}^{n} \mathbb{P}\left(X(t) = x(t) | X(1:t-1) = x(1:t-1)\right)$$

We haven't, however, established that this approach is any faster than summing over histories!

To see that it is, think about how time-consuming it will be to go through the cycle above once. To make a prediction for $X(t+1)$, we need to sum over states $S(t+1)$, and to get the probability that $S(t+1) = s$, we need to sum over states $S(t)$. So if there are k states, we need to so $O(k^2)$ operations in that double sum. Each of the n time-steps requires one pass through the cycle, so we need up needing to do $O(k^2 n)$ operations, not $O(k^n n)$ as with direct summation.

In this analysis, it's important that the procedure is recursive, that we calculate F_{t+1} using just F_t and $x(t+1)$ (and the model!), so that we don't need to keep track of earlier states or their distributions, or even, in fact, earlier values of X. To the extent they matter, they've been summarized in F_t, which is a sufficient statistic[2] for predicting $X(t+1)$.

To sum up, in an HMM it's straightforward to calculate $\mathbb{P}\left(X(1) = x(1)\right)$ and $F_1(s) = \mathbb{P}\left(S(1) = s | X(1) = x(1)\right)$. Thereafter, we can predict by cycling through (i) extrapolating the state distribution F_t forward in time (using the transition probabilitiess q), (ii) predicting the next observation (using (i) and the observation probabilities g), and (iii) updating the distribution of the state to F_{t+1} (using (ii) and Bayes's rule). This lets us make predictions, and so calculate a likelihood, or log-likelihood.

The Backward Algorithm

You may have noticed that we're just calculating $\mathbb{P}\left(S(t) | X(1:t)\right)$ and not $\mathbb{P}\left(S(t) | X(1:n)\right)$, let alone $\mathbb{P}\left(S(1:n) | X(1:n)\right)$. Partly, this is because all we really *need* for prediction or the likelihood is $\mathbb{P}\left(S(t) | X(1:t)\right)$. Partly, however, it is because it's just a lot *easier* to calculate this than it is to calculate $\mathbb{P}\left(S(t) | X(1:n)\right)$. Nonetheless, there is *also* a recursive algorithm for the latter. To use it, one must first have run the forward algorithm, getting a sequence $\mathbb{P}\left(S(t) | X(1:t)\right)$ for each $t \in 1:n$. The last of these, $\mathbb{P}\left(S(n) | X(1:n)\right)$, is the last probability we want, so the recursion starts at the *end* of the time series and works backwards in time (hence the name). I will omit all the details here, because they're not as illuminating as in the forward algorithm, and because you can find them in readily-available references, such as Fraser (2008).

[2] Some people therefore call F_t the "information state", an idea elaborated on in, e.g., Kulhavý (1996).

The Kalman Filter

An important special case is when $S(t)$ and $X(t)$ are both vectors, and everything is linear-and-Gaussian:

$$S(t+1) = \mathbf{a}S(t) + \eta(t) \quad (26)$$
$$X(t) = \mathbf{b}S(t) + \epsilon(t) \quad (27)$$
$$\eta(t) \sim_{IID} \mathcal{N}(0, \mathbf{\Sigma}_S) \quad (28)$$
$$\epsilon(t) \sim_{IID} \mathcal{N}(0, \mathbf{\Sigma}_X) \quad (29)$$

In this case, it's possible to write out all the conditional probabilities in the recursion *exactly* — everything is Gaussian, with mean vectors and variance matrices that are straightforward functions of the data. Finding the filtering distribution then becomes known as the **Kalman filter**, after Kalman (1960) (and Kalman and Bucy (1961)). One of the clearest presentations I've seen is in Fraser (2008), which you should read if this is at all interesting. There is a base R implementation in the function `KalmanRun` (with various companion functions `KalmanLike`, `KalmanForecast`, etc.).

While the Kalman-filter solution is is nicely straightforward, it is heavily tied to the linear-and-Gaussian assumptions. There are many important situations where that's good enough[3] (McGee and Schmidt 1985), but it can fail in important ways. This has led to a number of attempts to use local linear approximations to the Kalman filter in non-linear problems, and to alternative closed-form approximate filters (Koyama et al. 2010).

A different direction, however, is to go to back to the core ideas of the forward algorithm, and to simulate them. This is the "particle filter".

The Particle Filter

While the forward algorithm is straightforward in principle, it still involves a double sum over the state space. This can be a time-consuming when the number of states k is large. When the state-space is continuous, the sum becomes an integral, and doing even a double integral numerically can be quite time-consuming and/or inaccurate. In this situation, a useful trick is to replace complicated or time-consuming calculations with simulations, i.e., to use Monte Carlo.

Applied here, the **Monte Carlo filter**, or **particle filter**, approximates the distribution F_t with the empirical distribution of a large number of simulations of the HMM, conditioned on the data. It works as follows[4].

1. Make a large number of independent draws $S_1^*, S_2^*, \ldots S_m^*$ from the distribution $\mathbb{P}(S(1))$. Each random state value is called a **particle**, and the number of particles m is, ideally, as large as practical. (See more on this below.) Set $t = 1$.
2. For each particle S_i^*, calculate $g(S_i^*, x(t)) = w_i$. Adding these up approximates the predictive probability, $\mathbb{P}(X(t) = x(t)|X(1:t-1) = x(1:t-1)) \approx \sum_{i=1}^m w_i$.
3. Now resample the particles, i.e., draw m of them, with replacement, from S^*, with probabilities proportional to the w_i. Call the new values $\tilde{S}_1, \tilde{S}_2, \ldots \tilde{S}_m$. Set $\hat{F}_t$ to be the empirical distribution of the $\tilde{S}$. That is, their empirical distribution approximates $\mathbb{P}(S(t)|X(1:t) = x(1:t))$.
4. Increment t by 1, and evolve the particles forward in time. That is, for each $\tilde{S}_i$, pick a S_i^* according to $q(\tilde{S}_i, S_i^*)$. The distribution of these new S^* approximates $\mathbb{P}(S(t+1)|X(1:t) = x(1:t))$.

[3] In Lecture 10, we looked at the "Wiener filter", which is the optimal linear estimate of $S(t)$ from $X(1), X(2), \ldots X(t)$, without any Gaussian assumption (or assumption that any X *is* a linear-plus-noise function of $S(t)$). The Kalman filter can actually be derived from the Wiener filter, if we ask for the optimal linear prediction of $S(t)$ from $X(t)$ and $\hat{S}(t-1)$, the predicted state at the previous time-step[ˆkalmanforgottenlore]. (See, for instance, Liebelt (1967), pp. 161–165.) This approach to the Kalman filter doesn't require any assumption of a true linear model, or assuming that anything is Gaussian, but all it delivers is the best (least-squares) state estimate among recursive, linear state estimates. Without those assumptions, there will generally be better state estimates which are non-recursive and/or nonlinear.

[4] See Künsch (2013) for a good, brief review of particle filters, and Doucet, Freitas, and Gordon (2001) for a more exhaustive, though now a little old, treatment. The actual history of who introduced which idea when is a bit more involved than I feel like going into, but covered in both these references. Douc, Moulines, and Stoffer (2014) discusses statistical applications.

5. Go to (2).

This all lends itself wonderfully to code.

```
# Particle filter for an HMM
# Inputs: time series of length n (x)
  # function generating draws for initial state (rinitial)
  # function generating draws for next state (rtransition)
  # function giving observational probabilities (dobs)
  # number of particles (m)
# Output: n*m array of particles, approximating Pr(S(t)|X(1:t))
# Presumes: states can be elements in a vector
  # observations can be elements in a vector
  # rinitial is a no-argument function (for compatibility with rmarkov())
  # rtransition is a one-argument function (ditto)
  # dobs is a two-argument function
particle.filter <- function(x, rinitial, rtransition, dobs, m) {
    # How long is the time series?
    n <- length(x)
    # Our initial guesses at the state, before seeing anything
    particles <- replicate(m, rinitial())
    # A storage area for our guesses
    particle.history <- matrix(0, nrow=m, ncol=n)
    for (t in 1:n) {
        # Calculate observationa likelihoods/weights for current observation
        # given state=particles
        weights <- dobs(state=particles, observation=x[t])
        # Resample the particles, with probabilities (proportional to)
        # the weights
        particles <- sample(particles, size=m, replace=TRUE, prob=weights)
        # Store the current set of guesses
        particle.history[,t] <- particles
        # Evolve the particles forward using the Markov transitions for the
        # next time step
        particles <-sapply(particles, rtransition)
    }
    return(particle.history)
}
```

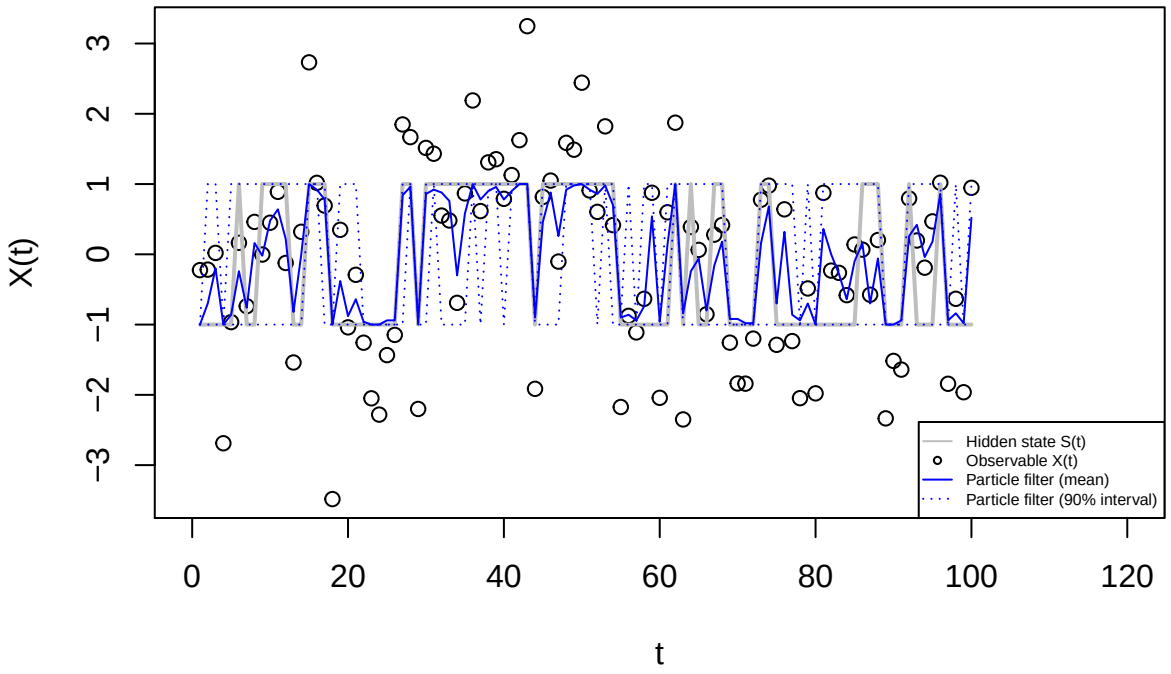
I will leave it as an exercise to extend this function so that it also returns (an approximation to) the sequence $\mathbb{P}(X(t)|X(1:t-1))$.

Here is an implementation using the HMM introduced at the beginning, where $S(t)$ is either $+1$ or -1, and $X(t)|S(t) \sim \mathcal{N}(S(t), 1)$.

```
sim.pf <- particle.filter(x=sim$x,
                          rinitial=function() { sample(-1,+1, size=1) },
                          rtransition=function (x) {
                              stay <- sample(c(TRUE, FALSE), size=1, prob=c(0.75, 0.25))
                              if (stay) { return(x) }
                              if (!stay) { return(-x) }
                          },
                          dobs=function(state, observation) {
                              dnorm(x=observation, mean=state, sd=1)
                          },
                          m=100)
```

We have 100 particles, i.e., 100 guesses for $S(t)$ at each time point. Rather than try to visualize this directly —- it's rather "busy" — I'll summarize with lines at the 5% and 95% percentiles (so a 90% probability interval).

```
plot(sim$x, xlab="t", ylab="X(t)", xlim=c(0, 120))
lines(1:100, sim$s, col="grey", lwd=2)
pf.limits <- apply(sim.pf, 2, quantile, prob=c(0.05, 0.95))
pf.means <- apply(sim.pf, 2, mean)
lines(1:100, pf.means, col="blue")
lines(1:100, pf.limits[1,], col="blue", lty="dotted")
lines(1:100, pf.limits[2,], col="blue", lty="dotted")
legend("bottomright", legend=c("Hidden state S(t)", "Observable X(t)", "Particle filter (mean)", "Parti
       pch=c(NA, 1, NA, NA), lty=c("solid", "blank", "solid", "dotted"), col=c("grey", "black", "blue",
       cex=0.5)
```



Notice that there are places where the limits become very narrow — these are mostly situations where $X(t)$ has a large absolute value, and do it's very easy to tell what $S(t)$ must be.

An Analogy for the Particle Filter

A common analogy for understanding how the particle filter works is to think of the particles as an evolving population, with the state of each particle being its genotype. The probability w_i that particle i gives to the current observation is then the "fitness" of particle i. In the resampling step, each particle has a chance to reproduce, and the expected number of offspring of particle i is $\propto w_i$. Finally, moving the new particles according to the Markov chain is like introducing mutations. So the whole particle filter process is analogous to "breeding" a population with the goal of giving high likelihood (=fitness) to the data.[5]

[5]In one direction, you can extend this analogy to techniques of "evolutionary optimization", such as "genetic algorithms", which try to breed good solutions to complex optimization problems (Holland 1992; Mitchell 1996). On the other, you can extend it to all forms of Bayesian inference, which are mathematically equivalent to natural selection without any sort of variation (Shalizi 2009).

Issues with the Particle Filter

1. Using a finite number of particles (= Monte Carlo samples) introduces approximation error — the distribution $\hat{F}_t$ isn't quite the same as F_t. This error shrinks as the number of particles $m \to \infty$. All else being equal, m should always be as large as possible.
   - What might not be equal is that increasing m costs more time to run. Each step of the particle filter — moving each particle according to the Markov process, calculating a probability for $X(t) = x(t)$ given the particle state, and re-sampling the particles — takes $O(m)$ operations. (You can parallelize the transitions and the likelihood calculations, but not the resampling.)
   - There are diminishing returns to increasing m — the approximation error is generally $O(1/\sqrt{m})$.
   - Exactly how large you need to make m thus depends on how you value computing time vs. numerical accuracy, which I can't tell you.
2. The particle filter will perform badly if at some time t, one particle i gives much higher probability to $X(t)$ than all the others, since then $w_i \gg w_j$, and all (or almost all) of the resampled particles will just be copies of w_i. This "particle collapse" is most apt to be a problem when the data does something which is very improbable according to the model, either because of an actual outlier, o because of model mis-specification. Bengtsson, Bickel, and Li (2008) studies this problem in some detail. A slightly hacky solution is to add a little bit of noise on to the resampled particles. (This is equivalent to smoothing the empirical distribution.)
3. I have presented the particle filter assuming the state-transition probabilities q and the state-observation probabilities g are *known*. Obviously, this is often not the case for real data problems, which brings us to the topic of parameter estimation.

Parameter Estimation

Up to now, I have been taking the state-transition probabilities $q(s, s')$ and the state-observation probabilities $g(s, x)$ as known functions. This is obviously unrealistic. They will generally depend on one or more unknown parameters θ, so I should really be writing $q(s, s'; \theta)$ and $g(s, x; \theta)$. How might we estimate θ?

- *Use the likelihood* We can use the forward algorithm to calculate, or at least approximate, $\mathbb{P}(X(1:n) = x(1:n); \theta)$ for each θ, as $\prod_t \mathbb{P}(X(t) = x(t)|X(1:t-1) = x(1:t-1); \theta)$. This is a function of θ, which we can maximize. In the usual way, it will be easier, and more numerically stable, to work with the log-likelihood,

  $$\sum_t \log \mathbb{P}(X(t) = x(t)|X(1:t-1) = x(1:t-1); \theta)$$

  A fully rigorous proof that maximum likelihood estimation is consistent for general HMMs is surprisingly tricky, but doable (Cappé, Moulines, and Rydén 2005). As a practical matter, however, the log-likelihood is a function, and can be optimized like anything else.
  - One issue with the likelihood-based approach is that if either $X(t)$ or $S(t)$ is a continuous variable, we *must* impose some parametric structure on g and/or q, since otherwise maximizing the likelihood leads to bad answers. Checking these assumptions is important, and potentially tricky.
- *Alternate between guessing at states and estimation* If we *knew* the sequence of states $S(t)$, it would be straightforward to estimate both q and g. We could estimate q using any technique we'd use to estimate transition probabilities for a Markov process, and we'd estimate g using any technique for conditional distributions. (We might even use regression in both problems, if we could believe in additive noise.) In the other direction, if we knew q and g, we can estimate $S(t)$ for each t. This suggests a whole slew of iterative-approximation methods, which start *either* with a guess for the S's or for q and g, and then alternate between using the current estimate for one to improve the estimate of the other.
  - The most famous iterative-approximation approach is the **Baum-Welch**, **forward-backward**, **expectation-maximization**, or **EM** algorithm. It uses the backward algorithm (see above) to find $\mathbb{P}(S(t)|X(1:n))$, using the current estimates for q and g. (This is the "E" step, since a probability is an expectation.) It then does weighted maximum likelihood to re-estimate q

and g, with every possible sequence of states getting a weight given by its probability under the backward algorithm. (This is the "M" step, since it's maximizing the likelihood.) It then re-runs the backward algorithm to re-estimate the states, and so forth to convergence. One can show that both the E-step and the M-step increase the log-likelihood, so this has to converge on at least a local maximum, though perhaps not a global one. (See Shalizi (n.d.) for a tutorial introduction to EM.)
    - Variants include replacing finding the whole distribution $\mathbb{P}(S(t)|X(1:n))$ with drawing samples from it (perhaps with the particle filter), or even just finding the most probable sequence $\hat{s}(1:n)$.
    - Non-parametric variants of the iterative-approximation idea are straightforward to code up, but hard to prove things about.
- *Simulation-based inference* Even with all the tricks for the forward algorithm, filtering, etc., calculating the likelihood can be difficult for complex HMMs. These can, however, be very straightforward to simulate. All of the techniques for simulation-based inference from earlier lectures can be brought to bear here, especially indirect inference.

Example of Simulation-Based Inference

Let's work through an example of simulation-based inference, with our toy problem. (The advantage of trying it out on a toy is that we can tell whether or not it's working.) Specifically, we'll use indirect inference, with an autoregressive auxiliary model. We'll need (i) a function to simulate the generative model, (ii) a function to estimate the auxiliary model, and (iii) a function which optimizes the generative model so that auxiliaries fit to its simulation match auxiliaries fit to the data.

We wrote (i) already, `noisy.rpmchain`. We need to write (ii) and (iii).

For (ii), the estimator of the auxiliary model, we'll use an autoregressive model. There is one[6] parameter we need to estimate, the probability $\mathbb{P}(S(t+1) = +1|S(t) = +1) = \mathbb{P}(S(t+1) = -1|S(t) = -1) = q$. This suggests an AR(1) should be enough. To give this a bit more information, though, we'll fit an AR(2).

```
# Estimate the auxiliary model (here, an AR(2)) for indirect inference
# Inputs: a time series (x)
# Output: Point estimate of the auxiiary model (here, a vector of length 2)
aux.estimator <- function(x) {
    ar2.fit <- ar.ols(x, aic=FALSE, order=2)
    return(ar2.fit$ar)
}

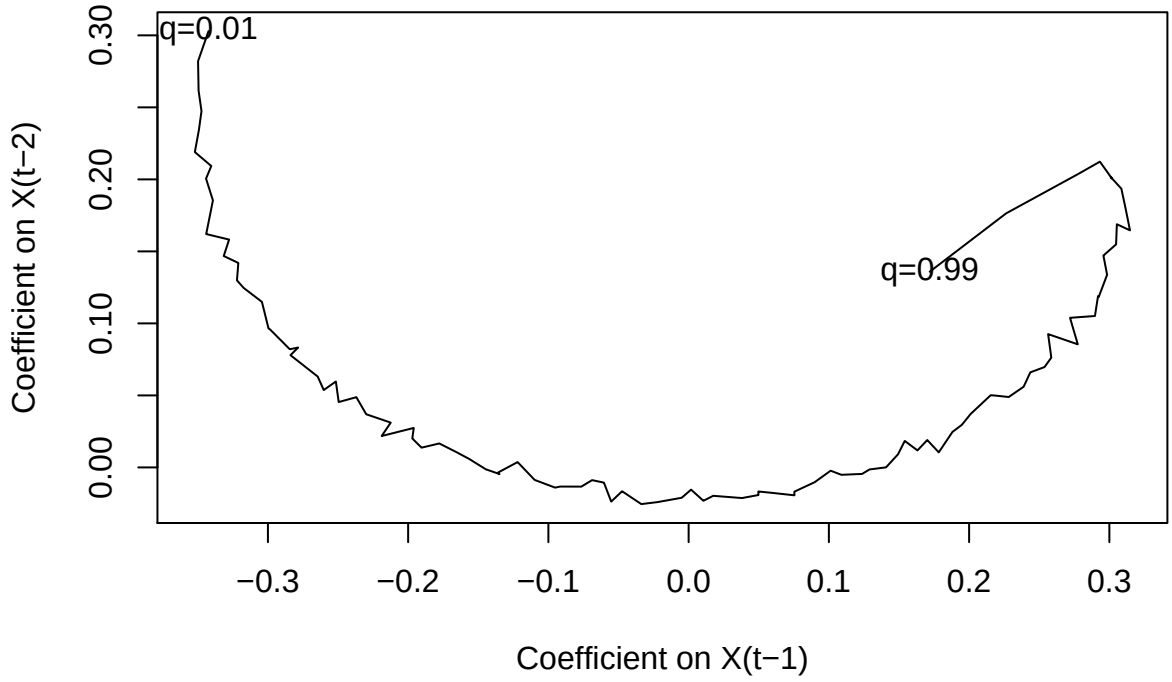
# Repeatedly simulate the generative model, fit the auxiliary, and average
# Inputs: parameter of generative model (q)
  # number of simulations to run (s)
  # length of each simulation (n)
# Output: vector of averaged auxiliary parameter estimates
aux.from.sim <- function(q, s, n=length(sim$x)) {
    estimates <- replicate(s, aux.estimator(noisy.rpmchain(n=n, q=q)$x))
    return(rowMeans(estimates))
}
```

We should check that there's a unique mapping from the parameter(s) in the generative model to the auxiliary parameters (so that inverting this mapping in principle lets us recover the generative parameter). We can do this here by plotting how the auxiliary parameters vary as we sweep over the generative parameter. (Obviously this sort of visual check gets harder with more parameters on each side.)

[6] If we wanted to, we could treat all the entries in the transition matrix for S as unknown, and estimate them, in the same way we're doing here, but this is just a demo. Similarly, we could treat the variance of the observational noise as unknown and estimate it, etc.

```
plottable.aux.from.sim <- Vectorize(aux.from.sim, vectorize.args="q")
ar2.vs.q <- sapply(seq(from=0.01, to=0.99, by=0.01),
                   plottable.aux.from.sim, s=500)
```

```
plot(ar2.vs.q[1,], ar2.vs.q[2,], type="l", xlab="Coefficient on X(t-1)",
     ylab="Coefficient on X(t-2)")
text(ar2.vs.q[1,1], ar2.vs.q[2,1], labels="q=0.01")
text(ar2.vs.q[1,99], ar2.vs.q[2,99], labels="q=0.99")
```



We can now follow the examples in earlier lectures to write out the indirect-inference function. One thing worth noting here is that we know the parameter q has to be between 0 and 1 (it's a probability), and, fortunately, `optim` has a good method for optimizing a single parameter between known limits. With multiple parameters, it might be worthwhile to use a package like `alabama`, which allows for very general constraints on the parameters.

```
toy.indirect.inference <- function(x, starting.q, s) {
    n <- length(x)
    # What's the auxiliary model on the actual data?
    aux.data <- aux.estimator(x)
    # Calculate the discrepancy between the auxiliary on the data and
    # the auxiliary we get from simulating at a particular parameter value
    discrepency <- function(param) {
        q <- param
        # Get the average of auxiliary estimates over s simulation runs
        aux.mean <- aux.from.sim(q=q, s=s, n=n)
        # Mean squared difference over auxiliary parameters
        return(sum((aux.data-aux.mean)^2))
    }
    # Now minimize the discrepancy
    fit <- optim(par=starting.q, fn=discrepency, method="Brent", lower=0,
                 upper=1)
    # Return just the best-fitting value
    return(fit$par)
```

```
}
```

Now let's actually run it:

```
(toy.ii <- toy.indirect.inference(x=sim$x, starting.q=0.5, s=500))
```

```
## [1] 0.7306343
```

Recall that the true value of q was 0.75, so this is doing pretty well. If we want to get a standard error, we could either follow the theory in Lecture 19, or just do a bootstrap. (Since we know this is a stationary process, that could either be a parametric or a non-stationary block bootstrap.)

Note that now that we could estimate the parameter of the generative model without also *having* to estimate the unobserved state. Having estimated q, however, we can now estimate the states $S(t)$, say with the particle filter.

Some Important Special Cases

Mixture Models

A sequence of IID variables is, technically, a Markov process. So if the sequence $S(1), \ldots S(n)$ is IID, we *could* apply the techniques above to estimate the latent variables $S(t)$. But most of the work would be unnecessary. Because $S(t)$ is independent of both $S(t-1)$ and $S(t+1)$, it breaks up into a collection of independent problems, of working out the distribution of $S(t)$ given $X(t)$ (and *only* given $X(t)$). When the latent variable $S(t)$ is discrete, we call this problem a "mixture model" or a "latent class model". When $S(t)$ is continuous, the problem has multiple names; "factor model" is common when $X(t)$ is linearly related to $S(t)$.

Shalizi (n.d.) goes over these situations in some detail.

Deterministic Latent Dynamics

If there is actually a *deterministic* relationship between $S(t)$ and $S(t+1)$, say $S(t+1) = \phi(S(t))$ for some function ϕ, that, too, is technically a Markov process. What makes this case special is that giving the initial value $S(1)$ fixes the whole rest of the $S(t)$ sequence. Thus there isn't the exponential growth in the number of possible trajectories we see with other Markov models. In fact, it makes sense to treat the initial state $S(1)$ as an unknown but *fixed* parameter, say $s(1)$. The likelihood is then

$$\prod_{t=1}^{n} g(s(t), x(t); \theta)$$

with the understanding that $s(t) = \phi^t(s(1))$. Of course, if the dynamics in the function ϕ are very sensitive to small changes in the initial state, optimizing this likelihood can be very difficult, and so it might make more sense to still fall back on techniques like simulation-based inference.

Exercises

To think through / practice on, not to hand in.

1. Think back to the first example: $S(t)$ is a Markov chain with states $+1$ and -1, $\mathbb{P}\left(S(t+1) = +1 | S(t) = +1\right) = \mathbb{P}\left(S(t+1) = -1 | S(t) = -1\right) = q$, and $X(t) \sim \mathcal{N}(S(t), 1)$. Assume that $S(1)$ is drawn from the invariant distribution, which is $(1/2, 1/2)$, so that the process is stationary. Find
    a. The variance of $X(t)$;
    b. All the covariances between $X(t-1)$, $X(t)$ and $X(t+1)$; and

c. The optimal coefficients for a linear prediction of $X(t+1)$ from $X(t)$ and $X(t-1)$. (You will need to invert a 2×2 matrix.)

2. a. Modify the code for the `particle.filter()` function so that it returns two matrices, one giving the estimated distribution of states $S(t)|X(1:t)$ and the other giving the propagation of states forward in time, $S(t+1)|X(1:t)$.
   b. Run the modified code on one run of the simulation. At each time point, plot both the predicted state distribution, $S(t)|X(1:t-1)$, and the filtered state distribution, $S(t)|X(1:t)$. (You may want to shift one of these slightly, either vertically or horizontally, so the two sets of points don't overlap.)

References

Bengtsson, Thomas, Peter Bickel, and Bo Li. 2008. "Curse-of-Dimensionality Revisited: Collapse of the Particle Filter in Very Large Scale Systems." In *Probability and Statistics; Essays in Honor of David a. Freedman*, edited by Deborah Nolan and Terry Speed, 316–34. Brentwood, Ohio: Institute of Mathematical Statistics. http://projecteuclid.org/euclid.imsc/1207580091.

Cappé, Olivier, Eric Moulines, and Tobias Rydén. 2005. *Inference in Hidden Markov Models*. New York: Springer.

Douc, Randal, Eric Moulines, and David S. Stoffer. 2014. *Nonlinear Time Series: Theory, Methods, and Applications with R Examples*. Boca Raton, Florida: Chapman Hall/CRC.

Doucet, Arnaud, Nando de Freitas, and Neil Gordon, eds. 2001. *Sequential Monte Carlo Methods in Practice*. Berlin: Springer-Verlag.

Erickson, R. V. 1970. "Functions of Markov Chains." *Annals of Mathematical Statistics* 41:843–50. https://doi.org/10.1214/aoms/1177696962.

Fraser, Andrew M. 2008. *Hidden Markov Models and Dynamical Systems*. Philadelphia: SIAM Press. http://www.siam.org/books/ot107/.

Heller, Alex. 1965. "On Stochastic Processes Derived from Markov Chains." *Annals of Mathematical Statistics* 36:1286–91. https://doi.org/10.1214/aoms/1177700000.

Holland, John H. 1992. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. 2nd ed. Cambridge, Massachusetts: MIT Press.

Kalman, R. E., and R. S. Bucy. 1961. "New Results in Linear Filtering and Prediction." *ASME Transactions, Journal of Basic Engineering* 83D:95–108.

Kalman, Rudolf E. 1960. "A New Approach to Linear Filtering and Prediction Problems." *ASME Transactions, Journal of Basic Engineering* 82D:35–50.

Koyama, Shinsuke, Lucia Castellanos Pérez-Bolde, Cosma Rohilla Shalizi, and Robert E. Kass. 2010. "Approximate Methods for State-Space Models." *Journal of the American Statistical Association* 105:170–80. http://arxiv.org/abs/1004.3476.

Kulhavý, Rudolf. 1996. *Recursive Nonlinear Estimation: A Geometric Approach*. Vol. 216. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag.

Künsch, Hans R. 2013. "Particle Filters." *Bernoulli* 19:1391–13403. https://doi.org/10.3150/12-BEJSP07.

Liebelt, Paul B. 1967. *An Introduction to Optimal Estimation*. Reading, Massachusetts: Addison-Wesley.

McGee, Leonard A., and Stanley F. Schmidt. 1985. "Discovery of the Kalman Filter as a Practical Tool for Aerospace and Industry." 86847. NASA Technical Memorandum. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19860003843_1986003843.pdf.

Mitchell, Melanie. 1996. *An Introduction to Genetic Algorithms*. Cambridge, Massachusetts: MIT Press.

Nilsson Jacobi, Martin, and Olof Görnerup. 2007. "A Dual Eigenvector Condition for Strong Lumpability of Markov Chains." arxiv:0710.1986. https://arxiv.org/abs/0710.1986.

Rogers, L. C. G., and J. W. Pitman. 1981. "Markov Functions." *Annals of Probability* 9:573–82. https://doi.org/10.1214/aop/1176994363.

Rosenblatt, Murray. 1971. *Markov Processes: Structure and Asymptotic Behavior*. Berlin: Springer-Verlag.

Shalizi, Cosma Rohilla. 2009. "Dynamics of Bayesian Updating with Dependent Data and Misspecified Models." *Electronic Journal of Statistics* 3:1039–74. https://doi.org/10.1214/09-EJS485.

———. n.d. *Advanced Data Analysis from an Elementary Point of View*. Cambridge, England: Cambridge University Press. http://www.stat.cmu.edu/~cshalizi/ADAfaEPoV.