11 Dynamic Models

This chapter covers dynamic models, an important kind of multilevel model. It shows how to simulate dynamic models, discusses process and observation error, and illustrates methods for fitting models that assume only one or the other. For problems where we want to estimate process error when the magnitude of observation error is known, it introduces the SIMEX approach. Finally, it presents a brief introduction to fitting state-space models, which can estimate both process and observation error, via the Kalman filter or Markov chain Monte Carlo.

11.1 Introduction

This chapter covers concepts and techniques for fitting *dynamic* models—models that describe how ecological processes drive populations to change over time. Dynamic models are a special case of the multilevel models we introduced in Chapter 10. Dynamic models contain both *process error*, which feeds back on future states of the population, and *observation error*, which affects only the current observation.

We introduce dynamic models by describing how to simulate them. Knowing how to simulate dynamic models is important because fitting dynamic models to data is so tricky that it's essential to fit models to simulated data to confirm that the methods work. (Most of the examples in this chapter use simulated "data.")

The easiest way by far of dealing with observation and process error is to ignore one or the other (Section 11.4). If your data have little noise, you may be able to get away with this approach. When you can independently estimate the variance of the observation error, the more recently developed SIMEX (simulation-extrapolation) algorithm provides a way to get unbiased parameter estimates (Section 11.5).

State-space models (Section 11.6) can in principle estimate both process and observation error from a single data set, subject to the very strong constraint that the data actually provide enough information to separate them reliably. The *Kalman filter* (Section 11.6.1) is a relatively simple algorithm for estimating the parameters of state-space models with normally distributed error. More generally, computationally intensive Bayesian (Millar and Meyer, 2000) and frequentist (de Valpine and Hastings, 2002; Thomas et al., 2005; Lele et al., 2007) methods can simultaneously estimate deterministic parameters, observation error, and process error in nonlinear,

nonnormal ecological models (Section 11.6.2). The use of such methods has recently begun to explode in ecology (Solow, 1998; de Valpine and Hastings, 2002; Ellner et al., 2002; de Valpine, 2003; Jonsen et al., 2003, Buckland et al., 2004; Clark and Bjornstad, 2004; Thompson et al., 2005). This chapter attempts to provide a basic and relatively painless introduction. If you want to explore this area further, you will have to dig into the literature (e.g., Calder et al., 2003).

11.2 Simulating Dynamic Models

Dynamic models describe the changes in the size and characteristics of a population over time. At each time step except the first, the size and characteristics of the population depend on the size and characteristics at the previous time step (or one or more times further in the past). Writing down the mathematical formula that describes the population size at time t is often much harder than describing how $N(t)$ depends on $N(t-1)$. The difference between observation and process error becomes vitally important in dynamic models, because they act differently. Process error affects future population dynamics, while observation error does not.

To simulate a dynamic model:

- Set aside space (a vector or matrix) to record the state of the population (numbers of organisms, possibly categorized by species/size/age).
- Set the starting conditions for all state variables.
- For each time step, apply R commands to simulate population dynamics over the course of one time step. Then apply R commands to simulate the observation process and record the current *observed* state of the population.
- Plot and analyze the results.

11.2.1 Examples

We can construct dynamic models corresponding to the two simple static models (linear/normal and hyperbolic/Poisson) introduced in Chapter 5.

Figure 11.1a shows a dynamic model analogous to the static model shown in Figure 11.1a (p. 149). The closest analogue of the static linear model, $Y \sim \text{Normal}(a+bx)$, is a dynamic model with observation error only:

$$\begin{aligned} N(1) &= a \\ N(t+1) &= N(t) + b \\ N_{\text{obs}}(t) &\sim \text{Normal}\left(N(t), \sigma^2_{\text{obs}}\right). \end{aligned} \tag{11.2.1}$$

The first line in (11.2.1) specifies the initial or starting condition (the value of N at time $t = 1$). The second line is the updating rule that determines the population size one time step in the future, which in this case is purely deterministic. The third line specifies the observation process, in this case that the observed value of the population size at time t, $N_{\text{obs}}(t)$, is normally distributed around the true value $N(t)$ with variance σ^2_{obs}.

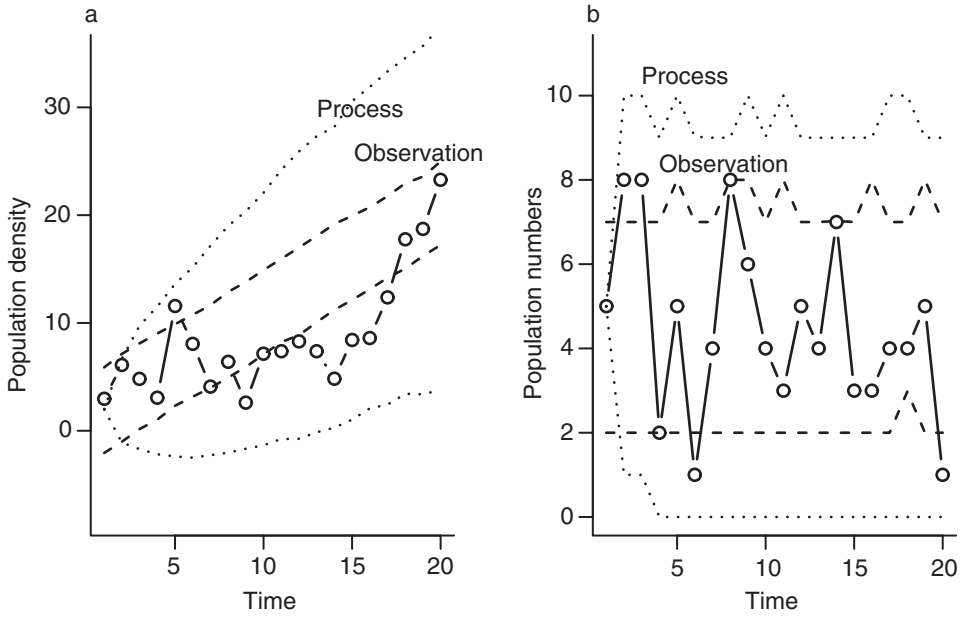

Figure 11.1 Dynamic models with process and observation error. (a) Linear, continuous (normal) model. (b) Nonlinear, discrete (hyperbolic/Poisson) model. In each case the envelopes (dotted and dashed lines) show the 95% confidence limits for equivalent models with pure process or pure observation error; the realizations shown are generated with a mixture of process and observation error.

The R code for this model would first specify `nt`, the number of time steps, and assign values for the parameters `a`, `b`, and `sd.obs`. Then:

```
> N = numeric(nt)
> Nobs = numeric(nt)
> N[1] = a
> for (t in 1:(nt - 1)) {
+     Nobs[t] = rnorm(1, mean = N[t], sd = sd.obs)
+     N[t + 1] = b + N[t]
+ }
> Nobs[nt] = rnorm(1, mean = N[nt])
```

Since the `for` loop runs only from 1 to `nt-1`, we have to set the observed value for $t =$ `nt` at the end. If we ran the loop to `nt`, we would be predicting the state of the population at time `nt+1`, beyond the end of the vector we have set aside for the results. R would cooperate by extending the length of the vector, but the too-long vector might lead to confusion or errors in subsequent steps.

By contrast, a model with pure process error is defined as

$$\begin{aligned} N(1) &= a \\ N(t+1) &\sim \text{Normal}\left(N(t)+b, \sigma^2_{\text{proc}}\right) \\ N_{\text{obs}}(t) &= N(t). \end{aligned} \tag{11.2.2}$$

The R code:

```
> N = numeric(nt)
> Nobs = numeric(nt)
> N[1] = a
> for (t in 1:(nt - 1)) {
+     N[t + 1] = rnorm(1, mean = b + N[t], sd = sd.proc)
+     Nobs[t] = N[t]
+ }
> Nobs[nt] = N[nt]
```

In this case, we assume that our observations are perfect ($N_{\text{obs}}(t) = N(t)$) but that the change in the population is noisy rather than deterministic.

The behavior of the mean in this dynamic model is exactly the same whether the variability in the model is caused by observation error or process error, and in fact it is identical to the deterministic part of a standard linear model $N = a + b(t-1)$. Furthermore, there is no way to separate process from observation error by simply looking at a single time series; the variation in the observed data will appear the same. (Figure 11.1 actually shows a single realization of a model with equal amounts of process and observation error; it falls outside the theoretical bounds of an observation-error-only model with slope $a = 1$, but only because we know the true slope. We couldn't tell the difference in a real data set.) The difference becomes apparent only when we simulate many realizations of the same process and look at how the variation *among realizations* changes over time (Figure 11.1a). With observation error only, the variance among realizations is constant over time; with process error only, there is initially no variance (we always start at the same density), but the variance among realizations increases over time.

Figure 11.1b shows a discrete-population model with process and observation error. In this case, the model is a rational function with the same form as the Beverton-Holt or Michaelis-Menten function. Suppose that per capita plant fecundity declines with population density according to the hyperbolic function $F(N) = a/(b+N)$. Then let the next year's expected population size $N(t+1)$ equal (population size) $\times$ (per capita fecundity) = $N(t)(a/(b+N(t)))$. The population grows asymptotically to a stable population size of $a-b$. (Convince yourself that when $N(t) = (a-b)$, $N(t+1) = N(t)$, and the simulated dynamics in Figure 11.1b are indeed nearly constant.)

For the observation error model, we assume that we have a probability of only p of counting each individual that is present in the population, which leads to a binomial distribution of observations:

$$\begin{aligned} N(1) &= N_0 \\ N(t+1) &= aN(t)/(b+N(t)) \\ N_{\text{obs}}(t) &\sim \text{Binomial}(N(t), p). \end{aligned} \tag{11.2.3}$$

The R code:

```
> N = numeric(nt)
> Nobs = numeric(nt)
> N[1] = N0
```

```
> for (t in 1:(nt - 1)) {
+     N[t + 1] = a * N[t]/(b + N[t])
+     Nobs[t] = rbinom(1, size = round(N[t + 1]), prop = p)
+ }
> Nobs[nt] = rbinom(1, size = round(N[nt]), prop = p)
```

The only problem in this model is that $N(t+1)$ is usually not an integer, in which case the binomial doesn't make sense. I rounded the value in this case, although normally it would be more sensible to incorporate a more realistic process model with (discrete) process error.* Like the linear observation error model, the distribution of error stays constant over time—with a few random bumps on the upper confidence limit caused by sampling error (Figure 11.1b).

The process error model for the discrete population case is simpler:

$$\begin{aligned} N(1) &= N_0 \\ N(t+1) &\sim \text{Poisson}(aN(t)/(b+N(t))) \\ N_{\text{obs}}(t) &= N(t). \end{aligned} \tag{11.2.4}$$

The R code:

```
> N = numeric(nt)
> Nobs = numeric(nt)
> N[1] = N0
> for (t in 1:(nt - 1)) {
+     N[t + 1] = rpois(1, lambda = a * N[t]/(b + N[t]))
+     Nobs[t] = N[t]
+ }
> Nobs[nt] = N[nt]
```

The population size still converges to $a - b$ over time, but the distribution spreads out over the first few time steps. In fact, many of the simulated populations quickly go extinct. However, since this model has a stable equilibrium, the distribution of process error reaches its own equilibrium, rather than spreading out continuously like the linear model in Figure 11.1a.

11.2.1.1 CONTINUOUS-TIME MODELS

Many dynamic models in ecology are defined in continuous rather than discrete time. Typically these models are framed as ordinary differential equation (ODE) models. The rule $N(t+1) = f(N(t))$ is replaced by $dN/dt = f(N(t))$, which specifies the instantaneous population growth rate. Probably the best-known ODE model is the logistic, $dN/dt = rN(1 - N/K)$. Researchers use continuous-time models for a variety of reasons including realism (for populations with overlapping generations that can reproduce in any season), mathematical convenience (the dynamics of continuous-time models are often more stable than those of their discrete analogues), and consistency with theoretical models. Most dynamic models have no

*But Henson et al. (2001) describe some possible dynamic consequences of this kind of rounding, which they call "lattice effects," in ecological systems.

closed-form solution (we can't write down a simple equation for $N(t)$), so we often end up simulating them.

The simplest algorithm for simulating continuous-time models is *Euler's method*, which uses small time steps to approximate the continuous passage of time. Specifically, if we know the instantaneous growth rate $dN/dt = f(N(t))$, we can approximate the change in the population over a short time interval Δt by assuming that the population grows linearly at rate dN/dt, and thus that $\Delta N \approx dN/dt \cdot \Delta t$:

$$\begin{aligned} N(t+\Delta t) &= N(t) + \Delta N \\ &\approx N(t) + \frac{dN}{dt}\Delta t \\ &= N(t) + f(N(t))\Delta t. \end{aligned} \tag{11.2.5}$$

In order to find the population size at some arbitrary time t we make Δt "small enough" and work our way from the starting time to t, adding ΔN to the population at each time step Δt.

Euler's method is fine for small problems, but it tends to be both slow and unstable relative to more sophisticated approaches. If you are going to do serious work with continuous-time problems, you will need to solve them for thousands of different parameter values (which may in turn require experimenting with different values of Δt). The `lsoda` function in R's `odesolve` library, which implements an adaptive step size algorithm, will be much more efficient.

The central problem with comparing ODE models to data is that incorporating stochasticity in any other way than simply imposing normally distributed observation error is difficult. The mathematical framework that underlies stochastic differential equations is subtle (Roughgarden, 1997) and hard to apply to practical problems. For this reason, studies that attempt to estimate parameters of continuous-time models from data tend either to use simple least-squares criteria that correspond to normal observation error (Gani and Leach, 2001) or to revert to discrete-time models (Finkenstadt and Grenfell, 2000).

One can build dynamical models that are stochastic, are discrete-valued (and hence more sensible for populations), and run in continuous time, picking random numbers for the *waiting times* until the next event (birth, death, immigration, infection, etc.). The basic algorithm for simulating these models, called the *Gillespie algorithm* (Gillespie, 1977), is simple, but it and the advanced methods required to estimate parameters based on such models are beyond the scope of this chapter (Gibson and Renshaw, 1998; Gibson and Renshaw, 2001).

11.3 Observation and Process Error

In general, we describe dynamic data by setting up

- A deterministic function for the *expected* population dynamics—the relationship between the current density and the expected density at time $t+1$, $\bar{N}(t+1) = f(N(t))$—for example, the discrete logistic equation, $\bar{N}(t+1) = N(t) + rN(t)(1 - N(t)/K)$, with parameters r and K.

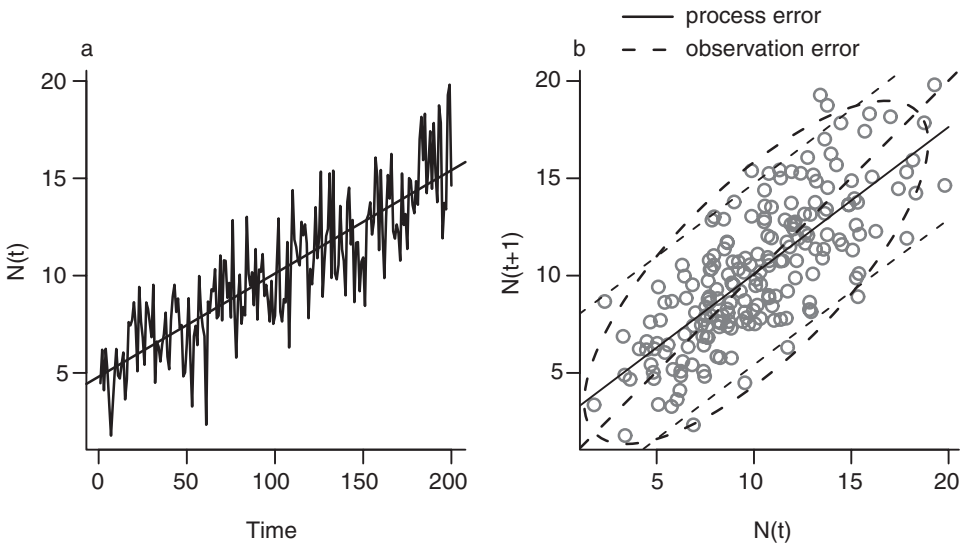

Figure 11.2 Time-series data: process or observation error?

- A model of process error—for example, $N(t)$ is negative binomially distributed with overdispersion parameter k, or $N(t) \sim \text{NegBin}(\mu = \bar{N}(t), k)$.
- A model of observation error—for example, a binomial sample with capture probability p from $N(t)$, or$N_{\text{obs}}(t) \sim \text{Binom}(p, N(t))$.

To understand some of the basic issues of dynamic data, let's look at the simplest deterministic model for population growth—a constant increase in the population density per time step, $f(N(t)) = N(t) + b$, with normally distributed process and observation error. Formally:

$$N(t+1) \sim \text{Normal}\left(N(t) + b, \sigma^2_{\text{proc}}\right) \tag{11.3.1}$$

$$N_{\text{obs}}(t) \sim \text{Normal}\left(N(t), \sigma^2_{\text{obs}}\right) \tag{11.3.2}$$

where σ^2_{proc} and σ^2_{obs} are the process and observation variances.

Suppose we recorded the data in Figure 11.2 and wanted to try to understand what was going on in the population. Depending on the combination of observation and process error that we assumed, we could draw very different conclusions about these data.

If we assumed there was only observation error, with no process error, then the simplest approach would be to solve the deterministic equation ($\bar{N}(t+1) = \bar{N}(t) + b$) as a function of time to get $\bar{N}(t) = \bar{N}(0) + bt$ and estimate b as the slope of an ordinary linear regression (`lm(N~time)`). We would interpret the population dynamics as a linear trend with time.

What if we instead wanted to use the plot of $N(t+1)$ against $N(t)$ (Figure 11.2b) to fit $f(N)$ ($f(N) = N + b$) directly? We would have to recognize that both $N_{\text{obs}}(t)$ and $N_{\text{obs}}(t+1)$ contain observation error, which doesn't fit the assumptions of ordinary linear regression. Instead we would minimize the *diagonal* deviations of points from

a line $y = a + bx$, a procedure sometimes called *model II* regression.* Our model of the points $\{N(t), N(t+1)\}$ would be that they were bivariate normal, with the mean of $N(t+1)$ equal to $N(t) + b$; the ellipse in Figure 11.2b represents the confidence limits for the points in this model.

On the other hand, if we assumed process error only (with no observation error), then we should fit an ordinary linear regression to the plot of $N(t)$ vs. $N(t+1)$, because we assume that we know the x variable ($N(t)$) perfectly and the only uncertainty comes in the population growth from t to $t+1$. If we allow the full linear model $N(t+1) = aN(t) + b$, then we are fitting an *autoregressive model*: while the overall trend would be the same as the ordinary linear model (provided $a < 1$), the variance structure is different.† Figure 11.2b also shows that this assumption gives different answers from the model II regression, with a larger intercept (which corresponds to a larger population growth rate—remember this is the graph of $N(t)$ vs. $N(t+1)$, not the graph of $N(t)$ vs. t) and a flatter slope.

What if we can't reasonably assume either pure process error or pure observation error? Intermediate assumptions can lead to any answer between the two slopes shown in the figure, which might lead to a wide range of different biological conclusions! Unfortunately the data don't easily show us what assumption to make. The noisier our data, the more the results of the linear-trend and autoregressive models will diverge. In the extreme where we have almost no information, the linear-trend model will say that $N(t) = N(t+1)$ (a 45° regression line), while the autoregressive model will say that $N(t+1)$ is independent of $N(t)$ (a flat line). Since we have no information, our conclusions are entirely driven by the structure of our assumptions. This example is the first indication that in analyzing dynamic models we may sometimes be attempting to separate processes (process and observation variability) for which we have very little distinguishing information. We will return to this sobering theme at various points during the chapter.

11.4 Process and Observation Error

Now we will see how the extreme assumptions of only process error or only observation error play out if we want to fit a model with more interesting dynamics than simple linear increase or decrease with time. For problems with small amounts of error, or if you want to keep things simple, use one of these approaches, as suggested by Hilborn and Mangel (1997). For example, pure process error would be a reasonable model for small discrete populations that could be counted exactly or for experimental populations observed in the lab (Drury and Dwyer, 2005). Pure observation error seems less plausible, but you would still be in good company picking one or the other: many sensible analyses of dynamic data have used these crude but simple methods (Ives et al., 1999; Gani and Leach, 2001; van Veen et al., 2005).

* Model II regression is a big topic (Warton et al., 2006); in special cases like this one (dynamic data with only observation error) where we can assume that the variances in x and y are the same, we can use *reduced major axis* regression, which gives the slope as σ_y/σ_x, or equivalently as $\sqrt{b_{yx}/b_{xy}}$, where b_{yx} is the slope of the ordinary regression of y on x and b_{xy} is the slope of the ordinary regression of x on y.

† We can fit the restricted model $f(N) = b + N$, assuming the slope of $N(t+1)$ vs. $N(t)$ is exactly 1, with `lm(y~offset(x))`.

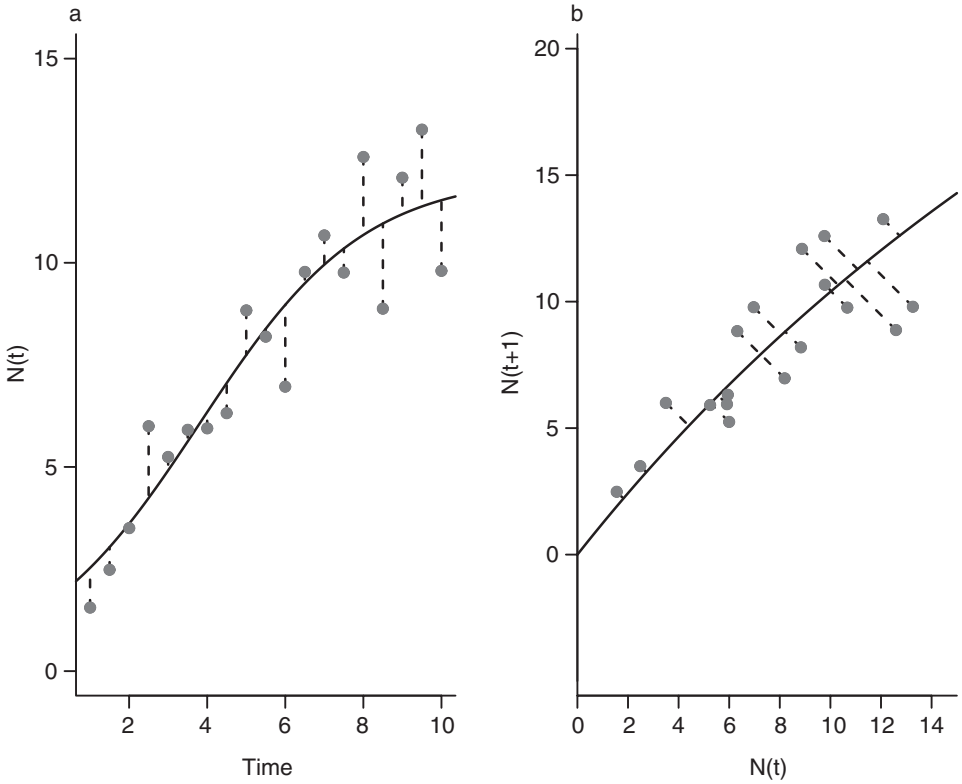

Figure 11.3 Logistic fit: shooting/trajectory matching (observation error only). True parameters $r = 1$, $K = 10$, $N(0) = 1$, $\sigma^2_{\text{obs}} = \sigma^2_{\text{proc}} = 1$. Estimated parameters $r = 0.48$, $K = 12.14$, $N(0) = 2.53$, $\sigma^2_{\text{obs}} = 1.41$. (a) Time dynamics, showing vertical residuals of observations from the fitted line. (b) Next vs. current observation, showing diagonal residuals from the fitted line.

11.4.1 Observation Error Only: Shooting or Trajectory Matching

If we assume observation error only we can start with the initial conditions of the system (e.g., the starting population sizes: we either assume we know these or take the starting values as additional parameters of the model) and "shoot" through the whole period, without correcting the model as we go along; this procedure is also called *trajectory matching* (Figure 11.3). If the deterministic dynamics are particularly simple (e.g., linear, exponential, or logistic), we may be able to derive a formula for $N(t)$ as a function of the starting conditions and calculate the predicted values in a single step (`N=a+b*time` or `N=a*exp(b*time)`), but much more often we will be able to compute the expected values only by using a loop to go from the value at each time step to the value at the next time step. (With a continuous-time model, you can use the `odesolve` package to solve numerically for each set of parameter values.) One way or the other, we compute the predicted values at all observation times, ignoring the variability in the actual data, and then compare the overall fit of the predicted curve to the data.

Since we assume there is no uncertainty in the predicted values for each time step given the starting conditions and the parameters, the only error is between the predicted values and the observed values. We can then do what we've been doing all along: assume independent observations and add up the log-likelihoods of observation error for every data point based on our model of observation error.

Trajectory matching is widely used because it is simple and requires no consideration of process variability. If one assumes normally distributed observation error with constant variance, it simplifies still further to least-squares fitting of the deterministic trajectory (e.g., Gani and Leach, 2001; van Veen et al., 2005). Trajectory matching also works with missing data or unobserved variables (Wood, 2001), although Ellner et al. (2002) warn that trajectory matching can be seriously misleading in cases where process variability qualitatively changes the dynamics of the population (e.g., Ellner et al., 1998).

11.4.2 Process Error Only: One-Step-Ahead Fitting

Alternatively, we can assume there is no observation error. Then the only uncertainty is in the relationship between N_t and N_{t+1}. If we plot the expected value of each N_{t+1} as a function of the (perfectly known) N_t, we have errors only in the Y variable. Instead of starting with the initial conditions and "shooting" (forecasting) through the whole observation time period, we take the observation from each time step and predict just the next time step (Figure 11.4). This way we need not worry about how process errors compound from step to step. (This procedure is more difficult with missing time points, because we then have to somehow figure out the expected relationship, including the process error, between (e.g.) $N(t)$ and $N(t+2)$ (Clark and Bjornstad, 2004).) This procedure is called *one-step-ahead prediction*. For population dynamics modeled in continuous rather than discrete time, a slightly more sophisticated analogue is called *gradient matching* (Ellner et al., 2002).

Shooting and one-step-ahead prediction are approximations, but they are simple and usually worth trying before you do anything more sophisticated. If the answers are not (biologically) significantly different, the fancier techniques may not be worth the effort. Furthermore, if you find in the end that the distinction between process and observation variability is unidentifiable, stating the results of process-error-only and observation-error-only analyses and saying that the true value is likely to be somewhere between those answers may be the best you can do.

11.5 SIMEX

In our one-step-ahead example, ignoring observation error led to a high estimate of r (1.22 vs. true value 1) and a low estimate of K (9.88 vs. true value 10). It's impossible to infer from a single example, but in fact ignoring observation error will generally give upward biased answers for r because observation error suggests that the population is changing faster than it really is. In this example K is biased downward as well. It's hard to figure out in general what direction of bias to expect—it depends in detail on the nonlinearities in the model—but estimates of nonlinear

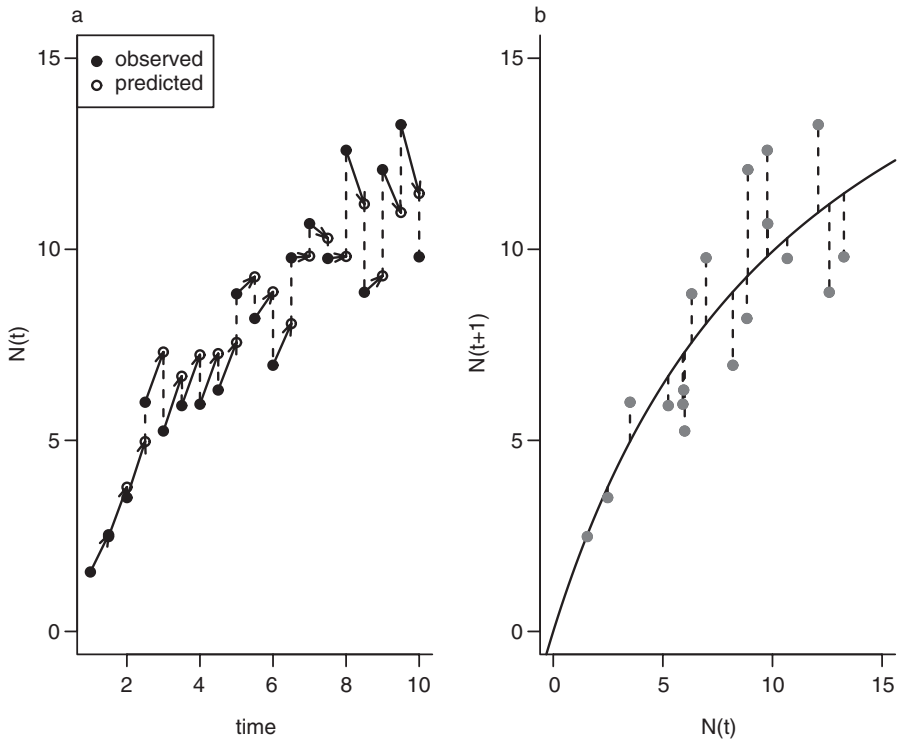

Figure 11.4 Logistic fit: one-step-ahead (process error only). True parameters $r = 1$, $K = 10$, $N(0) = 1$, $\sigma^2_{\text{obs}} = \sigma^2_{\text{proc}} = 1$. Estimated parameters $r = 1.22$, $K = 9.88$, $\sigma^2_{\text{obs}} = 2.66$. (a) Time dynamics and predictions. (b) Current vs. next observations, showing vertical residuals from the fitted line.

model parameters that ignore observation error are very likely to be biased one way or the other.

However, if you do have an estimate of the magnitude of the observation error, you can use the SIMEX (simulation-extrapolation) algorithm to correct for the bias caused by neglecting observation error. SIMEX works by inflating the observation error—adding additional noise to the data set—and reestimating the parameters (Cook and Stefanski, 1994; Carroll et al., 1995; Carroll et al., 1999; Stefanski and Cook, 1995). After estimating how increasing levels of observation error change the parameter estimates, you can then extrapolate to estimate the parameter values you *would* get with zero observation error. (Yes, this seems like black magic, but it works.)

More specifically, the procedure for SIMEX is as follows:

- Based on your estimate of observation error, pick a range of increased error values; tripling the existing observation variance in four to eight steps is a reasonable rule of thumb. (For example, if the estimate of observation error is σ^2_{obs}, pick observation variances of $\{1.5\sigma^2_{\text{obs}}, 2\sigma^2_{\text{obs}}, 2.5\sigma^2_{\text{obs}}, 3\sigma^2_{\text{obs}}\}$.)
- For each error magnitude in your range, generate a data set with that increased error. The procedure is more stable if you pick a single set of normally distributed random values and then multiply them by increasing factors for each

simulation. (If y_i are your values and ϵ_i is a set of normal deviates with variance σ^2_{obs}, the first simulated data set with the inflation factors above would be $y_i + \sqrt{0.5}\epsilon_i$; the variance of this data set is $\sigma^2_{\text{obs}} + 0.5\sigma^2_{\text{obs}} = 1.5\sigma^2_{\text{obs}}$. The second data set with $y_i + \epsilon_i$ would have variance $2\sigma^2_{\text{obs}}$.)

- For each simulated data set, estimate the values of the parameters using one-step-ahead prediction and save them.
- Estimate a relationship between the total variance and the values of the parameters (a separate regression for each parameter, typically a linear or quadratic regression: `lm1 = lm(param~measerr+I(measerr^2))`).
- Find the SIMEX bias-corrected estimates of the parameters by extrapolating the regressions to zero variance (for a linear or quadratic regression, the first coefficient is the intercept: `coef(lm1)[1]`).

11.6 State-Space Models

The final, most sophisticated and most general but most challenging category of statistical estimation procedures for dynamic data are so-called *state-space models*. In principle state-space models can allow you to estimate parameters of the deterministic process, observation error, *and* process error from a single observed time series—always subject to the constraints of identifiability. Trying to fit state-space models to time series that are too short, vary too little, or otherwise contain insufficient information to identify the parameters will lead to numerical problems and wide confidence intervals if you're lucky (and skilled), and misleading answers if you're unlucky. Schnute's warning about identifiability (p. 334) refers specifically to state-space models. With that warning in mind, here we go.

In general, we know that the amount of observation error will be the same for each observation (or at least will depend only on the true value, and not on when it is measured), while the amount of process error will tend to increase over time. The longer we wait between observations, the more random variation will decrease our certainty about the state of the system.

The key insight of state-space models is that every observation we make does several things:

- It provides information that shrinks the cloud of uncertainty around the true but unknown current state of the system.
- It provides *indirect* information about the likelihood of the next state. For example, a higher-than-expected population count in 2000 increases the expectation for the 2001 count.
- It also provides indirect information about the *previous* state of the system. For example, a higher-than-expected population count in 2000 also makes us think that the true population size in 1999 might have been higher than we previously thought.

If this discussion sounds Bayesian to you (updating our expectations of the probability of the state of the system based on prior observations), you're right. Lots of state-space modeling has a Bayesian flavor, although it can also be done in a frequentist framework (de Valpine and Hastings, 2002; Ionides et al., 2006; Lele et al., 2007). At

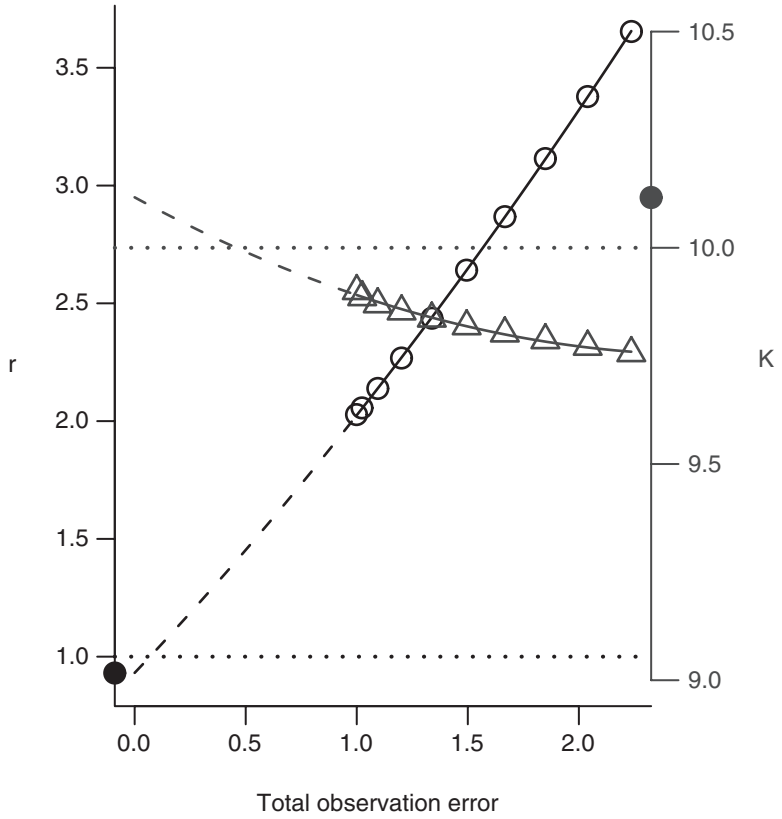

Figure 11.5 SIMEX extrapolation for the logistic model. Horizontal lines show true values: black circles show estimates for r and gray triangles show estimates for K, both extrapolated quadratically back to $\sigma^2 = 0$.

this cutting-edge level, there's much more interplay between Bayesian and frequentist approaches than at more basic levels.

Estimation algorithms for state-space models are essentially systems that carry out the complicated bookkeeping required to keep track of the current estimates of the true state of the system at a particular time. For each new choice of parameters, the algorithm works through the data set one observation at a time, updating estimates of the true value and variance at that time based on the parameters and the current estimate of the previous time step (and in some systems, of the next time step as well). Once this is done for the whole data set, you can use the estimates and variances to calculate the likelihood for the new set of parameters and decide how to pick the next one, using a standard algorithm such as Nelder-Mead or MCMC.

11.6.1 Kalman Filter

The *Kalman filter* is an algorithm for calculating the expected means and covariances of the observed values for a whole time series in the presence of observation and process error. In its original form it works only for linear population models (i.e.,

exponential increase or decrease or expected constant population size over time) with multivariate normal error; the *extended Kalman filter* uses an approximation that works for nonlinear population dynamics. The Kalman filter's great strengths are its relative simplicity and speed.

The Kalman filter works by stepping through the data set one observation at a time, updating what we know about the mean and variance of the true state variables at time t. It is an inductive procedure, giving the rules for figuring out the mean and variance at time t if we already know the mean and variance at time $t-1$. Clearly, then, if we can figure out starting values for the mean and variance at time 1, we can work through the whole data set this way.

I'll illustrate this with a very simple example (keeping in mind that we can add many realistic complications), with a single population growing linearly at rate a per year, with an autoregressive term b that means that N_{t-1} and N_t have a correlation coefficient of b (over and above the general linear trend with time). I assume there is both process (σ^2_{proc}) and observation (σ^2_{obs}) error, both normally distributed.

So our model is

$$N_t \sim \text{Normal}\left(a + bN_{t-1}, \sigma^2_{\text{proc}}\right) \tag{11.6.1}$$

$$N_{\text{obs},t} \sim \text{Normal}\left(N_t, \sigma^2_{\text{obs}}\right). \tag{11.6.2}$$

If $b < 1$, then the population is stable, because random deviations in N shrink by a factor b every year; if $b > 1$, then the population is unstable and random deviations grow over time.

Suppose, based on all the observations up through time $t-1$, we believe that the mean of the true population size at time $t-1$, N_{t-1}, is μ_0, and its variance is σ_0^2. We can calculate based only on the population parameters a and b what we expect the mean and variance to be at the next time step. The change in the mean is a direct reflection of the population model; the variance term is a combination of multiplying the previous variance by b^2 since we have multiplied the population size by b, and adding the new variability introduced by process error between $t-1$ and t. So

$$\text{mean}\left(N_t | N_{\text{obs},t-1}\right) = \mu_1 = a + b\mu_0 \tag{11.6.3}$$

$$\text{Var}\left(N_t | N_{\text{obs},t-1}\right) = \sigma_1^2 = b^2\sigma_0^2 + \sigma^2_{\text{proc}}. \tag{11.6.4}$$

More stable populations, indicated by low values of b, imply lower variance. As b gets very small, no variance carries over from one time step to the next and the standing variance of the population becomes just σ^2_{proc}.

The mean of the observation at time t equals the mean of the true value (we assume variance, but no bias, in the observation process). The variance equals the current variance of the true population size plus the observation variance:

$$\text{mean}\left(N_{\text{obs},t} | N_{\text{obs},t-1}\right) = \mu_2 = \mu_1 \tag{11.6.5}$$

$$\text{Var}\left(N_t | N_{\text{obs},t-1}\right) = \sigma_2^2 = \sigma_1^2 + \sigma^2_{\text{obs}}. \tag{11.6.6}$$

The last step of the Kalman filter, taking the information about the current observation into account, is the hardest. The current observation, changes our estimate of the mean of the true population state. How much it changes it depends on how far

the current observation is from where it was expected to be based on the previous information ($N_{\text{obs},t} - \mu_2$), as well as the ratio of the variances of the true value and of the observation. If there is no observation error, then the variance of the observation is the same as the variance of the true state of the population, and (as shown by the formula below) we simply set the mean of the population equal to the current observation. If there is lots of observation error, then the current observation doesn't tell us very much and we don't let an unexpected observation change our value of the mean.

$$\text{mean}(N_t|N_{\text{obs},t}) = \mu_3 = \mu_1 + \frac{\sigma_1^2}{\sigma_2^2}(N_{\text{obs},t} - \mu_2). \tag{11.6.7}$$

The Bayesian approach suggests another interpretation of this equation: our best estimate of the current population size is a weighted average of our prior—what we think the population size is based on previous time steps (μ_1)—and the current observational data ($N_{\text{obs},t}$).

Finally, we need to update the variance based on the current observation. Here we actually reduce the current variance of the true value, again based on the ratio of the variance of the true value to the variance of the observation.

$$\text{Var}(N_t|N_{\text{obs},t}) = \sigma_3^2 = \sigma_1^2\left(1 - \frac{\sigma_1^2}{\sigma_2^2}\right). \tag{11.6.8}$$

If there is no observation error, then $\sigma_1^2 = \sigma_2^2$ and the variance of the true value becomes zero. Unlike the mean, the variance is independent of the observed data.

Now that we've figured out the mean and variance of N and N_{obs} based on all the observations up to time t, we can repeat the procedure to calculate the values at time $t + 1$. Once we have worked through the whole data set, we know the expected mean and variance at each time step, and we can calculate the standard normal log-likelihood for the observed values.

The concepts are the same but the formulas are considerably more complicated in the general case described by Schnute (1994). The one extension I will describe here is how to estimate a nonlinear population growth function $f(N)$, called the *extended Kalman filter*.

All we have to do is replace (11.6.3) and (11.6.4) with appropriate generalizations. For example, let's replace the linear equation in (11.6.1) with the discrete logistic equation:

$$N_t \sim \text{Normal}(N_{t-1} + rN_{t-1}\left(1 - \frac{N_{t-1}}{K}\right), \sigma_{\text{proc}}^2). \tag{11.6.9}$$

Then substitute this equation for (11.6.3):

$$\text{mean}(N_t|N_{\text{obs},t-1}) = \mu_1 = \mu_0 + r\mu_0\left(1 - \frac{\mu_0}{K}\right). \tag{11.6.10}$$

For the variance, we need to find the equivalent of b, the per capita growth rate, to substitute into (11.6.4). A reasonable approximation for the current per

capita growth rate is the derivative of the population growth rate with respect to the population size:

$$\frac{\partial f}{\partial N} = \frac{\partial(N + rN(1 - N/K))}{\partial N} = 1 + r - 2N/K. \tag{11.6.11}$$

Since this equation is based on a first-order Taylor expansion, it is good only for relatively small noise or short time steps.

Evaluating the derivative at the current mean value of the population size ($N = \mu_1$) gives

$$\mathrm{Var}(N_t | N_{\mathrm{obs},t-1}) = \sigma_1^2 = (1 + r - 2\mu_1/K)^2 \sigma_0^2 + \sigma_{\mathrm{proc}}^2. \tag{11.6.12}$$

If the population is currently growing ($\frac{\partial f}{\partial N} > 1$), the variance is inflated. If it is shrinking, the variance is deflated.

So how do we implement this in R?

The R supplement defines a function `nlkfpred` that calculates the nonlinear Kalman filter predictions for a set of time-series data and a `nlkflik` that uses those predictions to compute the negative log-likelihood for a set of parameters. We fit all the parameters on the log scale to avoid the possibility of negative parameter values.

We need to pick starting values for the estimation. I'm going to cheat here since I know the true values, but it would be easy enough to do a one-step-ahead or trajectory-matching fit to the data, or even eyeball, to estimate reasonable starting values for r, K, and the variances.

```
> startvec = list(logr = log(0.25), logK = log(10),
+     logprocvar = log(0.5), logobsvar = log(0.5),
+     logM.n.start = log(3), logVar.n.start = -2)
```

Maximum-likelihood estimation of the parameters:

```
> m4 = mle2(minuslogl = nlkflik, start = startvec,
+     data = list(obs.data = y.procobs2),
+     method = "Nelder-Mead", control = list(maxit = 2000))
```

The fitted parameters are reasonable (although K appears slightly biased upward) and the confidence intervals bracket the true values, as seen in Figure 11.6 and Table 11.1. It's not surprising that the confidence intervals are narrow for K and, slightly wider for r (the population spends more time around its carrying capacity than in the growth phase), or that the confidence intervals for the variances are larger than the confidence intervals for the deterministic parameters.

The Kalman filter has been widely used in fisheries modeling, where the need to squeeze information out of rare data is so strong that researchers are always looking for the next powerful technique. Early on, researchers applied the technique to abundant but noisy catch-per-unit-effort data. More recent applications have used the Kalman filter as a way to estimate the locations of animals from noisy telemetry data, allowing the observed position at a previous time to help constrain the expected location at the current time (Jonsen et al., 2003). The KF has more recently begun to make its way into mainstream terrestrial ecology, as a way of estimating parameters for the growth of species of conservation concern in the presence of both observation and observation error (Lindley, 2003). While the assumption of linear population

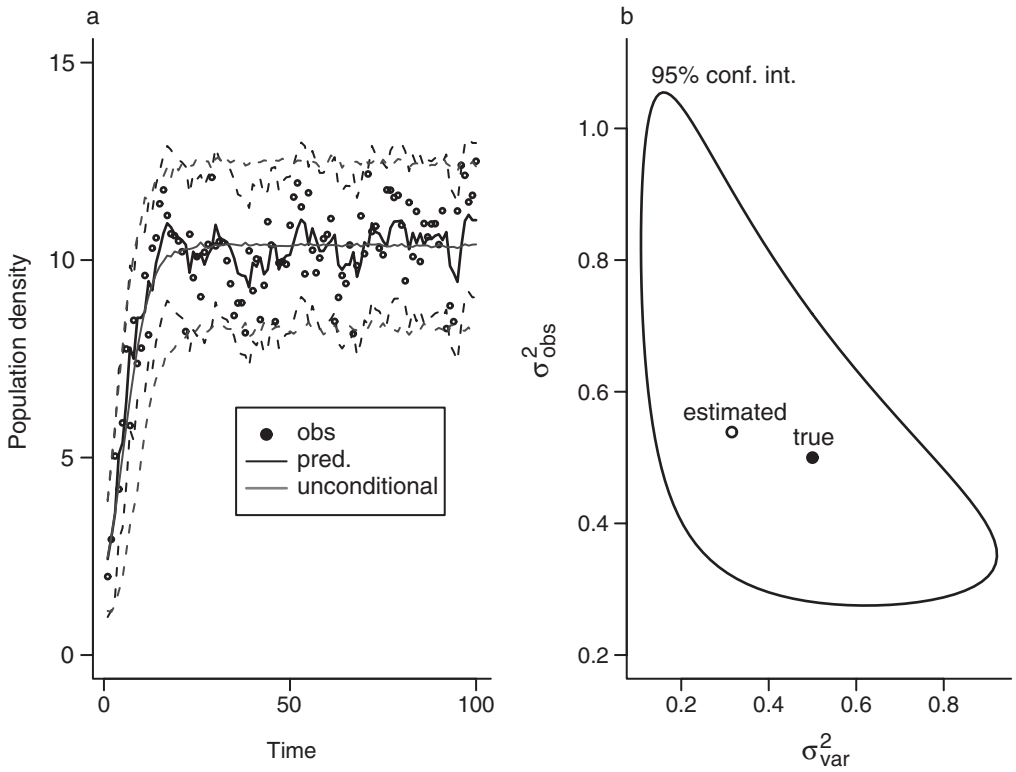

Figure 11.6 Results of Kalman filter: (a) observed, predicted, and results of unconditional simulations; (b) MLE, true value, and approximate 95% bivariate confidence interval.

dynamics in the standard Kalman filter might seem constraining, the autoregressive equation $N_t = a + bN_{t-1}$ does allow a range of population dynamics, from fluctuation around a stable equilibrium if $a > 0$ and $b < 1$, to exponential dynamics if $a = 0$ (declining if $b < 1$, increasing if $b > 1$), to a pure random walk if $a = 0$ and $b = 1$. Much of conservation biology is built on linear models, which will often apply when species are rare and thus intraspecific competition is low (Caswell, 2000). And if you do need nonlinearity, you can use the extended Kalman filter.

There are ways to incorporate many other biological complexities in the Kalman filter such as multiple species, time lags, bias and imperfect catchability in observations, correlated observations, time-varying control parameters, and covariates

TABLE 11.1

	True	Fitted	2.5%	97.5%
r	0.25	0.30	0.18	0.48
K	10.00	10.43	10.01	10.87
σ^2_{proc}	0.50	0.32	0.13	0.74
σ^2_{obs}	0.50	0.54	0.31	0.92

measured with error; see Schnute (1994) for details. Don't be scared by the notation. If you follow through it carefully, you can match up the special case here with all the details in that paper.

11.6.2 Markov Chain Monte Carlo Approaches (WinBUGS et al.)

The Kalman filter has limitations. In particular, it assumes normal, or lognormal, distributions. More subtly, the Kalman filter is a *prospective* algorithm (Schnute, 1994). It uses only the information up to time t to predict the mean and variance of the population size, even though the observation at time $t+1$ also gives us information about the population size at time t—*retrospective* information that we might be able to use to improve estimation.

Retrospective bookkeeping can be done several ways. Schnute discusses a frequentist approach called the *errors-in-variables* method, and de Valpine (de Valpine and Hastings, 2002; de Valpine, 2003) has also developed such a frequentist method. Here I'm going to present Bayesian methods (e.g., Millar and Meyer, 2000), which are rapidly growing in popularity because BUGS makes it simple to develop and estimate the parameters of relatively complex population dynamic models (Lele et al. (2007) suggest a way to use BUGS to calculate maximum likelihood estimates for complex models). The basic idea carries over from the Kalman filter; if you assume you know all the observations and the true values at every *other* time step, you can use them to estimate the population size *now*. The Markov chain Monte Carlo approach alternates between picking new random values for each true population size, one at a time (at each time step pretending you know the population sizes at all the other time steps), and picking new random values for the parameters that are consistent with the current assumed population size. Figure 11.7 shows the dependency graph for the first four steps of a logistic process. Each observed value depends on the true value at that time step and the observation error; each true value depends on the parameters and determines the observed value and the value at the next time step. In this kind of graph, though, you can also follow arrows *backward* to see that as well as depending on the value at time 1, the true value at time 2 is also influenced by the observed value at time 2 and by the true value at time 3—and hence indirectly by the observation at time 3, just as I suggested above.

To use BUGS to analyze dynamic data you must first decide on a model. Here we'll again use the discrete logistic equation with normally distributed observation and process error for comparison, although BUGS would allow us to be much more flexible. You also need to set priors for the parameters. Translating (11.6.3) and (11.6.4) into BUGS syntax produces the following model:

```
model {
  t[1] <- n0
  o[1] ~ dnorm(t[1],tau.obs)
  for (i in 2:N) {
     v[i] <- t[i-1]+r*t[i-1]*(1-t[i-1]/K)
     t[i] ~ dnorm(v[i],tau.proc)
     o[i] ~ dnorm(t[i],tau.obs)
  }
```

−1
0
1

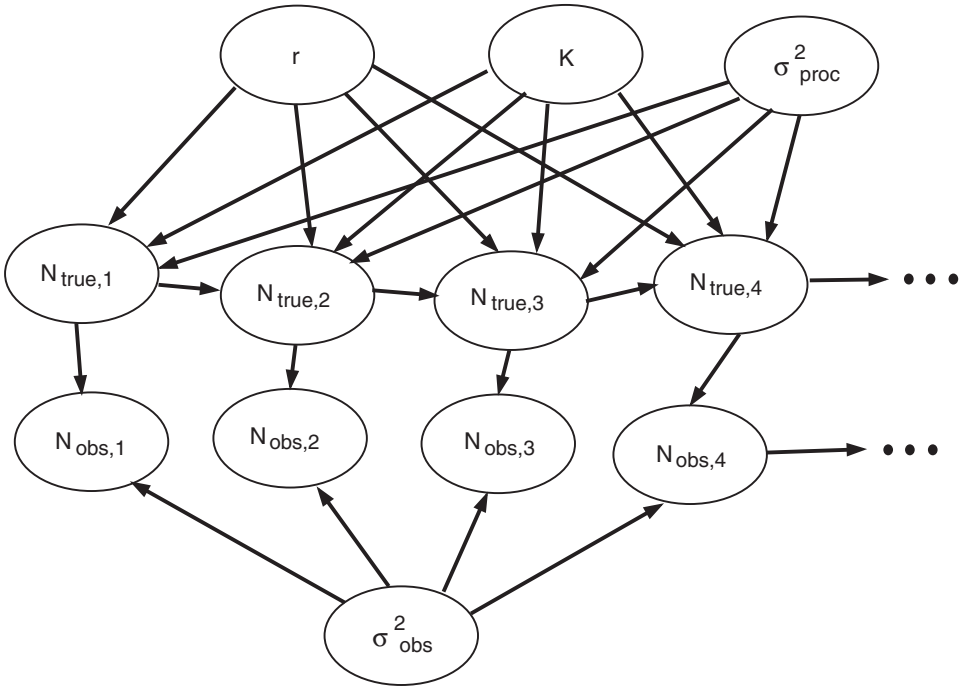

Figure 11.7 Dependency structure for the logistic model.

The first two lines define the initial conditions. The rest of the model steps through the data set, calculating the deterministic expectation (`v[i]`) and then defining the distribution of the true values (`t[i]`) and observed values (`o[i]`).

The rest of the model file defines the priors:

```
  r ~ dunif(0.1,maxr)
  K ~ dgamma(0.005,0.005)
  tau.obs ~ dgamma(0.005,0.005)
  tau.proc ~ dgamma(0.005,0.005)
  n0 ~ dgamma(1,n0scale)
}
```

The prior growth rate r is uniformly distributed between 0.1 and a maximum value, which I made a parameter so I could vary it within R without changing my BUGS input file. The priors for the carrying capacity K and the precisions (inverse variances) τ_{obs} and τ_{proc} are Gamma distributions with rate and shape parameters of 0.005, giving them a mean of 1 and a large variance ($0.005/0.005^2 = 200$: remember that BUGS uses a shape + rate parameterization rather than R's shape + scale parameterization). The initial density n_0 has a prior distribution that is Gamma with shape parameter 1 and a rate parameter equal to the reciprocal of the first observed value—again defined as a parameter to be calculated in R—which gives it an exponential distribution with mean equal to the first observed value. Though weak, this prior is stronger than the prior distributions for the carrying capacity and precisions.

A bit of R code to define the upper limit of the r prior and the parameter of the initial-state prior:

```
> maxr <- 2
> n0rate <- 1/y.procobs2[1]
```

We will set up the model, using the same data series `y.procobs2` as before and defining five different chains, using the `perturb.params` function from the `emdbook` package to change the values of r and the precisions (τ_{obs}, τ_{proc}). We should probably vary the starting values of the precisions a bit more systematically, although BUGS tends to crash if the starting values are too extreme.

```
> o <- y.procobs2
> N <- length(y.procobs2)
> statespace.data <- list("N", "o", "maxr", "n0rate")
> inits = perturb.params(list(n0 = y.procobs2[1], r = 0.2,
+     K = 10, tau.obs = 1, tau.proc = 1), alt = list(r = c
+     (0.1, 0.4), tau.obs = 3, tau.proc = 3))
```

We next define the parameters we want to keep track of; we could also track the estimated true values at each time step.

```
> parameters <- c("r", "K", "tau.obs", "tau.proc",
+     "n0")
```

After running WinBUGS from within R, we convert the output to a CODA object—the CODA format has slightly different uses from the format returned by R2WinBUGS.

```
> statespace.sim <- bugs(data = statespace.data, inits,
+     param = parameters, model = "statespace.bug",
+     n.chains = length(inits), n.iter = 15000)
> s1 = as.mcmc.bugs(statespace.sim)
```

R2WinBUGS's defaults for running an MCMC analysis are to take the total number of iterations (the default is 2,000); set aside half of them as "burn-in"; divide the other half equally among all the chains specified by the user (the default is 3); and "thin" the results to save a total of 1000 iterations across all chains. In this case I chose to run 15,000 iterations with five chains, so each chain ran for 3000 steps; the first 1500 were discarded; and then 13% of the remaining iterates were kept for a total of 1000.

Checking convergence:

```
> gelman.diag(s1)

Potential scale reduction factors:

           Point est.   97.5% quantile
r             1.00           1.02
K             1.01           1.02
tau.obs       1.02           1.05
tau.proc      1.05           1.09
n0            1.01           1.02
deviance      1.02           1.05
```

TABLE 11.2

	2.5%	Median	97.5%
r	0.17	0.30	0.48
K	9.99	10.45	10.98
σ^2_{proc}	0.14	0.36	0.82
σ^2_{obs}	0.22	0.52	0.90

```
Multivariate psrf

1.02
```

Based on the G-R rule of thumb that a scale reduction factor < 1.2 for all variables means adequate convergence, the G-R diagnostic suggests that the chains did in fact run long enough to mix with each other.

The summary of a CODA object provides the quantiles of the chains; these results are practically identical to those from the Kalman filter (Table 11.2). I have inverted the precisions ($\tau_{\text{proc}} = 1/\sigma^2_{\text{proc}}$, $\tau_{\text{obs}} = 1/\sigma^2_{\text{obs}}$ to make it easier to compare directly with the KF results; the median is not identical to the mode (which in turn is close to the maximum likelihood estimate if the priors are weak), but it's close.

Figure 11.8 shows the results of the R2WinBUGS run for σ^2_{obs} and σ^2_{proc}. The values from the Kalman filter (Figure 11.6) are shown in gray. The 95% credible interval matches the approximate 95% confidence interval reasonably well, especially considering that the confidence interval is an approximation based on the local curvature. The mode of the posterior density, as expected, is very close to the MLE—with a weak prior probability distribution, the likelihood surface and the posterior probability distribution are close to the same shape. The mean is slightly larger than the mode—there is some skew toward large values of the process variance—while the median, not shown, falls between the mean and the mode. All four summary values (posterior mean, mode, and median, and the MLE) and the true value all fall within the 50% credible interval; as is often the case, all of these estimates give us *approximately* the same answer.

Finally, Figure 11.9 shows density plots for the R2WinBUGS analysis. The densities are all reasonably symmetric and bracket the known true values (the density of `tau.obs` extends to very high values; this is the result of a single freakish excursion in one of the chains to a very high value). Each chain's density is drawn with a different line type; they all fall on top of each other, reassuring us that the chains have converged and are all telling the same story.

11.7 Conclusions

This chapter covered a variety of methods for estimating the parameters of dynamic models, ranging from crude (assuming either process error or observation error, but not both) to sophisticated (state-space models).

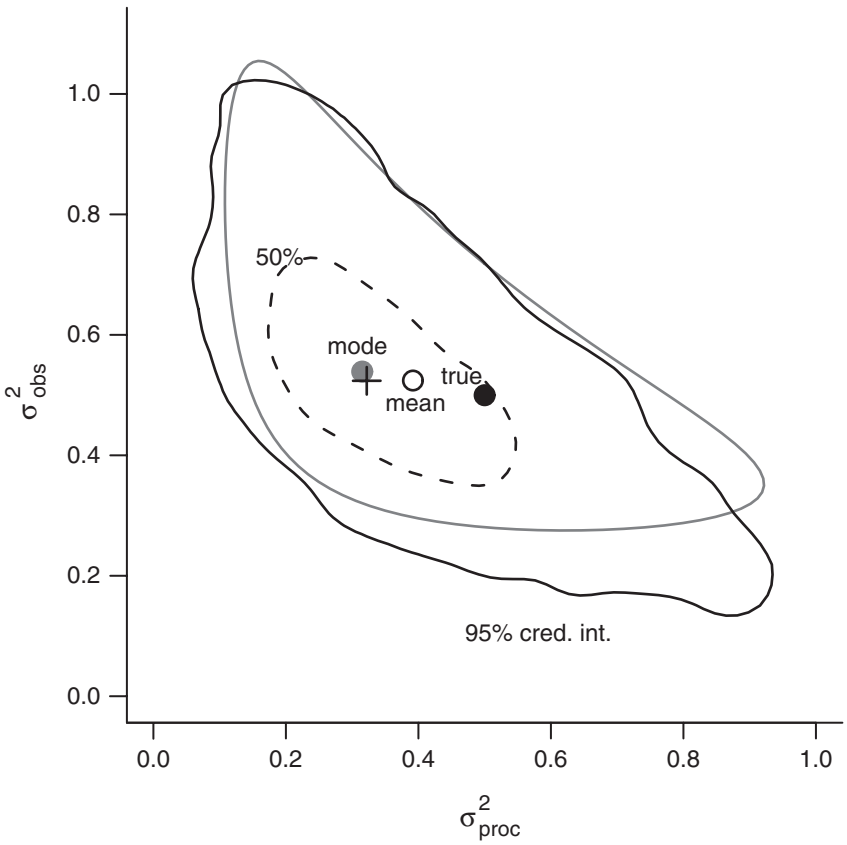

Figure 11.8 Results of R2WinBUGS analysis for logistic equation with process and observation error. Gray ellipse and point represent MLE and approximate (information-based) 95% confidence interval for Kalman filter fit (Section 11.6.1). Solid line is 95% credible interval based on BUGS chain; dashed lines is 50% credible interval. Points show the true (known) value, mean, and mode of the posterior density.

We largely skipped over the question of how to decide which dynamic models to fit. The logistic and a few simple discrete stochastic models were mentioned here, and the theta-logistic was noted in Chapter 3, but most of the book has focused on static models. Understanding dynamic models is a huge topic, mostly focused on deterministic models—Ellner and Guckenheimer (2006) or the other references listed in the ecological modeling section in Chapter 1 (section 1.3.1) make a good starting point on the dynamics side, Clark (2007) includes a review of dynamic modeling in his presentation of Bayesian methods, and Bjørnstad and Grenfell (2001) give an overview of more recent advances in the field.

On the other hand, it's easy to incorporate some additional ecology in the single-species logistic model. For example, with either the Kalman filter or MCMC you can incorporate the effects of covariates on the growth rate. To incorporate a linear effect of rainfall on the growth rate, you could just change the appropriate line of the BUGS model file to

```
v[i] <- t[i-1]+(r0+r1*rain[i-1])*t[i-1]*(1-t[i-1]/K)
```

and change the `parameters` and `data` values accordingly in the R code.

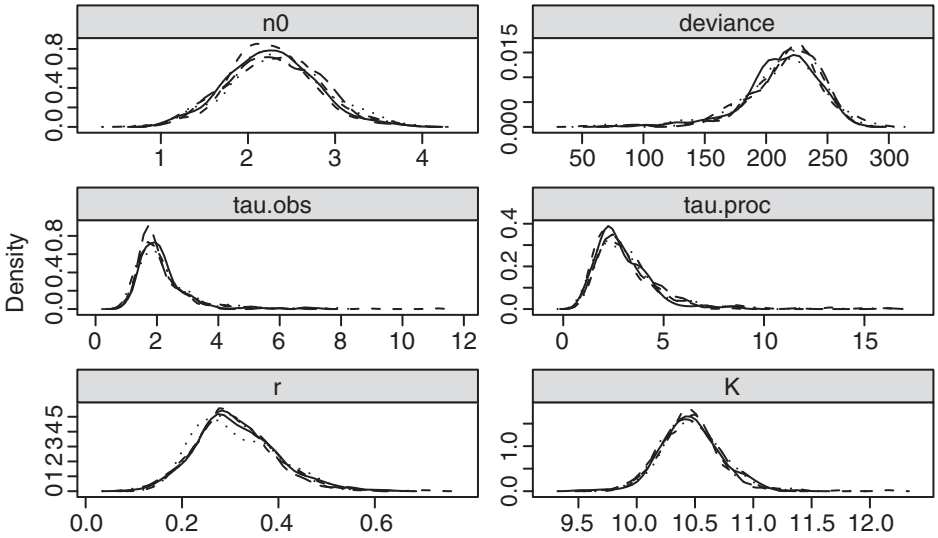

Figure 11.9 Density plots of R2WinBUGS results for the logistic equation. Different line types show results from different chains.

All too often, you can observe only one facet of a complicated ecological interaction. For example, we might be able to sample just hare populations in a complex Canadian ecosystem consisting of lynx, hare, vegetation, and birds of prey. While trying to reconstruct an entire ecosystem from observations of a single species is hopeless, we could in principle include additional unobserved variables in a state-space model—remember that the "true" population sizes are also unobserved. Be very careful not to incorporate more complexity in your model than the data can support: try your model out with some optimistic, but plausible, simulation data. Such reconstruction has been shown to work, for example, in simple epidemic models, where the number of new cases is observed but the number of possibly susceptible individuals left in the population is not. A formal process of "susceptible reconstruction" provides a time series of susceptibles to go along with the time series of infected individuals, which then allows estimation of a transmission parameter (Finkenstadt and Grenfell, 2000; Lekone and Finkenstadt, 2006).

This chapter has presented only analyses of discrete-time models, where the methods are much better developed. It is unfortunate that continuous-time methods for dynamical data are so sparse, since most theoretical models of ecological systems are defined in continuous time. Analysis is feasible if you assume only observation error (Gani and Leach, 2001; van Veen et al., 2005) or know the amount of observation error and use SIMEX to correct bias (Ellner et al., 2002; Melbourne and Chesson, 2006), but the Kalman filter and MCMC approaches have been used almost exclusively in discrete time (although Fujiwara et al. (2005) provide a recent counterexample). Gibson developed such methods (Gibson and Renshaw, 1998; Gibson and Renshaw, 2001; Gibson, 1997; Streftaris and Gibson, 2004), but they have yet to be widely used or made practical.

Right now Bayesian analyses of dynamic models are easier than frequentist analyses, but the frequentists are catching up fast. *Particle filtering* and *sequential importance sampling* are powerful frequentist alternatives to the Bayesian MCMC methods presented here (Doucet et al., 2001; Buckland et al., 2004; Thomas et al.,

2005; Harrison et al., 2006; Ionides et al., 2006). Particle filtering starts with a large number of random samples ("particles," e.g., 250,000 in Thomas et al. (2005)) from a prior (or pseudo-prior) distribution, including the distribution of the initial values of the state variables. Each sample is projected forward (simulated) one step, and a likelihood based on the first observation is calculated for each sample. The same number of particles is then resampled, but with weights proportional to their likelihoods. After simulating one more step, the likelihoods based on the next observation are calculated and the particles are resampled again (thus taking the observations at both t and $t+1$ into account). This process is iterated for the whole time series of observations, with various algorithms used to prevent all of the resamples from coming from a very small number of particles.

Estimating parameters of dynamic ecological models is still clearly an exercise on the cutting edge of science. Most of the papers that have appeared to date are technical and methods-oriented rather than applications to particular ecological questions. As time goes on the tools will improve and more examples will appear, giving potential users a better idea how much data (at least within an order of magnitude) is needed to apply these methods successfully. In the meantime, always check your answers against the results of simulations and against one-step-ahead (process error only) and trajectory-matching (observation error only) fits.

11.8 R supplement

11.8.1 Kalman Filter

Here's a function that computes the Kalman filter predictions. `Nobs` is the data set; `r` and `K` are the population dynamic parameters; `procvar` and `obsvar` are the process and observation variances; and `M.n.start` and `Var.n.start` are the starting values of the mean and variance. This code sets aside numeric vectors for the results on the mean and variance of the observed population size at each time step. When estimating parameters we don't have to save the mean and variance of the true population size since we don't have anything to compare them with. The function then sets the starting values and works through the data set one time step at a time, applying the Kalman filter equations:

```
> nlkfpred = function(r, K, procvar, obsvar, M.n.start,
+     Var.n.start, Nobs) {
+     nt = length(Nobs)
+     M.nobs = numeric(nt)
+     Var.nobs = numeric(nt)
+     M.n = M.n.start
+     Var.n = Var.n.start
+     M.nobs[1] = M.n.start
+     Var.nobs[1] = Var.n.start + obsvar
+     for (t in 2:nt) {
+         M.ni = M.n + r * M.n * (1 - M.n/K)
+         b = 1 + r - 2 * r * M.n/K
+         Var.ni = b^2 * Var.n + procvar
```

```
+         M.nobs[t] = M.ni
+         Var.nobs[t] = Var.ni + obsvar
+         M.n = M.ni + Var.ni/Var.nobs[t] * (Nobs[t] -
+             M.nobs[t])
+         Var.n = Var.ni * (1 - Var.ni/Var.nobs[t])
+     }
+     list(mean = M.nobs, var = Var.nobs)
+ }
```

Our likelihood function takes a set of parameters (all fitted on the log scale so we don't run into trouble with negative values of the parameters), runs the Kalman filter to predict the values of the means and variances, and then plugs these values into a normal likelihood comparison with a set of observed values (taking the square root of the estimated variance since `dnorm` uses the standard deviation, not the variance, as a parameter):

```
> nlkflik = function(logr, logK, logprocvar, logobsvar,
+     logM.n.start, logVar.n.start, obs.data) {
+     pred = nlkfpred(r = exp(logr), K = exp(logK),
+         procvar = exp(logprocvar), obsvar =
+         exp(logobsvar), M.n.start = exp(logM.n.start),
+         Var.n.start = exp(logVar.n.start),
+         Nobs = y.procobs2)
+     -sum(dnorm(obs.data, mean = pred$mean, sd = sqrt
+         (pred$var), log = TRUE))
+ }
```

−1
0
1