Estimating Lotka-Volterra Predator-Prey Dynamics with Stan

Bob Carpenter

October 16, 2017

Abstract

The Lotka-Volterra equations define parametric differential equations for the fluctuation of predator and prey populations. To estimate the parameters of such a model, a model for measurement error and unexplained variation is layered on top of the deterministic dynamics. The model is coded in Stan and fit to data on Canadian lynxes and snowshoe hares based on numbers of pelts collected in the early 20th century by the Hudson Bay Company.

Predator: *Canadian lynx*

Lynxes and Hares, 1900-1920

The Hudson Bay Company recorded the number of captured pelts of two species between 1900 and 1920,

Prey: *snowshoe hare*

- snowshoe hares (https://en.wikipedia.org/wiki/Snowshoe_hare), an hervivorous cousin of rabbits, and
- Canadian lynxes (https://en.wikipedia.org/wiki/Canada_lynx), a feline predator whose diet consists almost exclusively of hares.

The date provided here was converted to comma-separated value (CSV) format from (Howard 2009).

```
lynx_hare_df <-
  read.csv("hudson-bay-lynx-hare.csv", comment.char="#")
head(lynx_hare_df, n = 3)
```

```
##   Year Lynx Hare
## 1 1900  4.0 30.0
## 2 1901  6.1 47.2
## 3 1902  9.8 70.2
```

The number of pelts taken by the Hudson Bay Company is shown over time as follows (first, the data is melted using the reshape package, then plotted by species using ggplot).

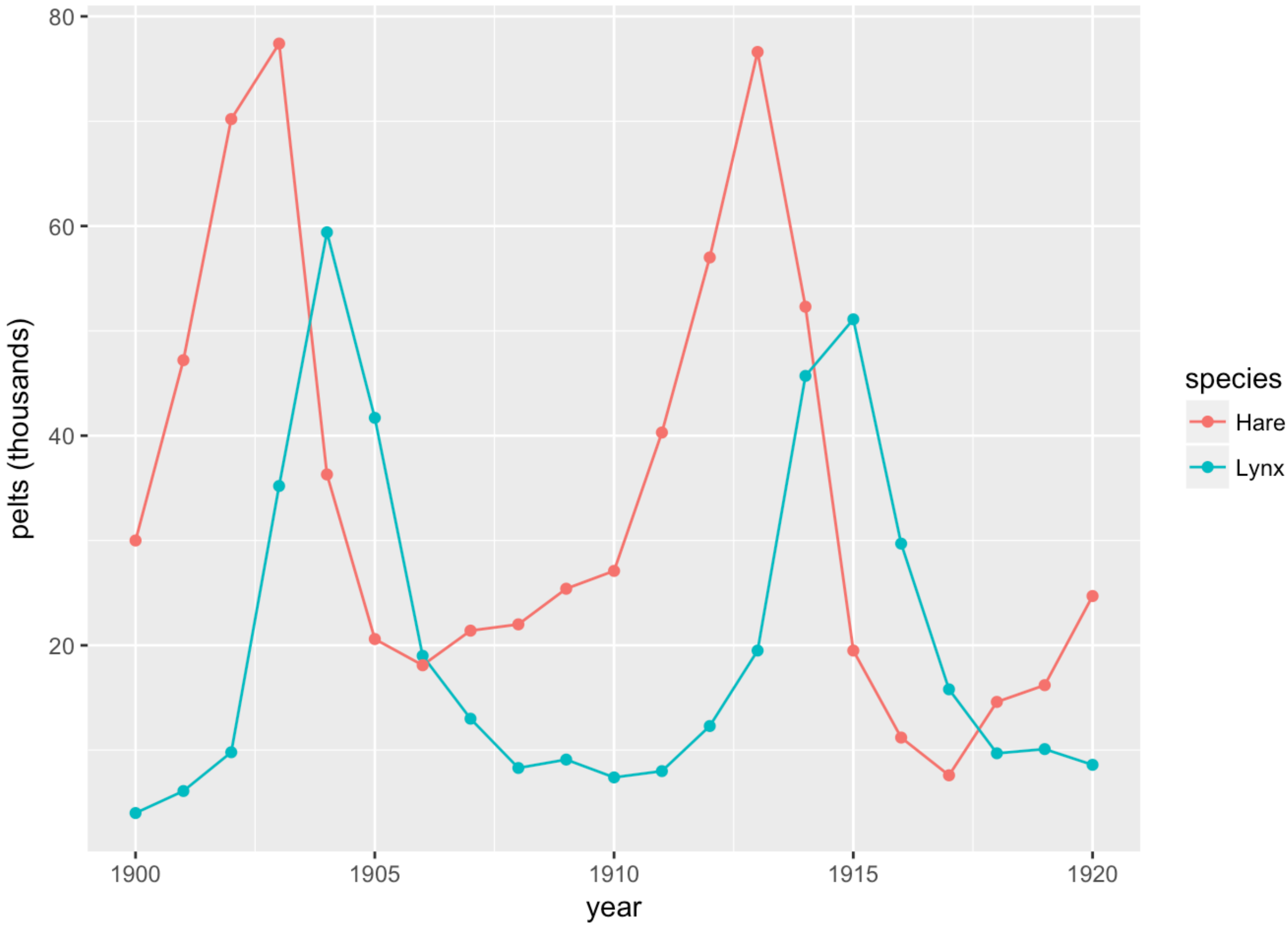
```
lynx_hare_melted_df <- melt(as.matrix(lynx_hare_df[, 2:3]))
colnames(lynx_hare_melted_df) <- c("year", "species", "pelts")
lynx_hare_melted_df$year <-
  lynx_hare_melted_df$year +
  rep(1899, length(lynx_hare_melted_df$year))
head(lynx_hare_melted_df, n=3)
```

```
##   year species pelts
## 1 1900    Lynx   4.0
## 2 1901    Lynx   6.1
## 3 1902    Lynx   9.8
```

```
tail(lynx_hare_melted_df, n=3)
```

```
##    year species pelts
## 40 1918    Hare  14.6
## 41 1919    Hare  16.2
## 42 1920    Hare  24.7
```

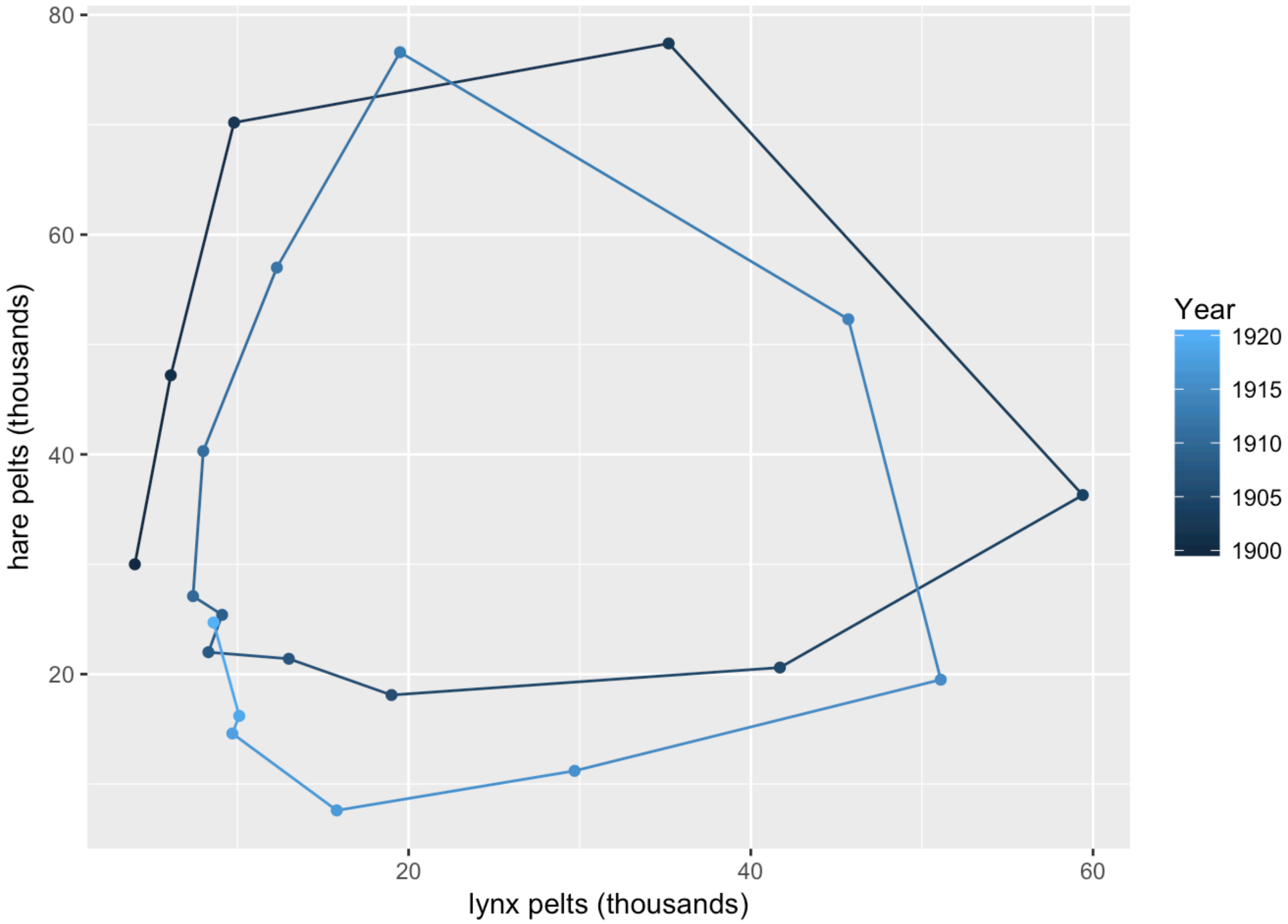
```
population_plot2 <-
  ggplot(data = lynx_hare_melted_df,
         aes(x = year, y = pelts, color = species)) +
  geom_line() +
  geom_point() +
  ylab("pelts (thousands)")
population_plot2
```



This plot makes it clear that the spikes in the lynx population lag those in the hare population. In both populations, the periodicity appears to be somewhere in the neighborhood of ten to twelve years.

Volterra (1926) plotted the temporal dynamics of predator and prey populations using an axis for each species and then plotting the temporal course as a line. The result for the lynx and hare population is easily plotted from the original data frame.

```
population_plot1 <-
  ggplot(data = lynx_hare_df,
         aes(x = Lynx, y = Hare, color = Year)) +
  geom_path() +
  geom_point() +
  xlab("lynx pelts (thousands)") +
  ylab("hare pelts (thousands)")
population_plot1
```



As can be seen from the diagram, the population dynamics orbit in an apparently stable pattern.

The Lotka-Volterra Equations

The Lotka-Volterra equations (Volterra 1926, 1927; Lotka 1925) are based on the assumptions that

- the predator population intrinsically shrinks,
- the prey population intrinsically grows,
- larger prey population leads to larger predator population, and
- larger predator population leads to smaller prey populations.

Together, these dynamics lead to a cycle of rising and falling populations. With a low lynx population, the hare population grows. As the hare population grows, it allows the lynx population to grow. Eventually, the lynx population is large enough to start cutting down on the hare population. That in turn puts downward pressure on the lynx population. The cycle then resumes from where it started.

The Lotka-Volterra equations (Volterra 1926, 1927; Lotka 1925) are a pair of first-order differential equations describing the population dynamics of a pair of species, one predator and one prey Suppose that

- $u(t) \geq 0$ is the population size of the prey species at time t, and
- $v(t) \geq 0$ is the population size of the predator species.

Volterra modeled the temporal dynamics of the two species (i.e., population sizes over times) in terms of four parameters, $\alpha, \beta, \gamma, \delta > 0$, as

$$\frac{\mathrm{d}}{\mathrm{d}t}u = (\alpha - \beta v)u \qquad = \alpha u - \beta uv$$

$$\frac{\mathrm{d}}{\mathrm{d}t}v = (-\gamma + \delta\, u)\, v \;\; = -\gamma v + \delta uv$$

As usual in writing differential equations, $u(t)$ and $v(t)$ are rendered as u and v to simplify notation.

Error model: measurement and unexplained variation

The Lotka-Volterra model is deterministic. Given the system parameters and the initial conditions, the population dynamics are fully determined. We will specify a statistical model that allows us to infer the parameters of the model and predict future population dynamics based on noisy measurements and a model that does not explain all of the observed variation in the data. We will consider two sources of error.

First, the theory is not expected to be that good in this case, so there will be resulting unexplained variation. For example, the weather in a particular year is going to have an impact on the populations, but it is not taken into account, leading to variation that is not explained by the model.

The second source of error is noisy measurements of the population. We cannot measure the population directly, so instead make do with noisy measurements, such as the number of pelts collected. In more elaborate models (beyond what we consider here), measurements of pelts collected could be supplemented with output of other measurements, such as mark-recapture studies.

A linear regression analogy

Like a simple linear regression, or non-linear GLM, the trick is to treat the underlying determinstic model as providing a value which is expected to have error from both measurement and unexplained variance due to the simplifications in the scientific model. Consider the typical formulation of a linear regression, where y_n is the scalar outcome, x_n is a row vector of predictors, β is a coefficient vector parameter, and ϵ_n is a latent scalar parameter and $\sigma > 0$ is another parameter,

$$y_n = x_n\beta + \epsilon_n$$

$$\epsilon_n \sim \text{Normal}(0, \sigma)$$

The deterministic part of the equation is the linear predictor $x\beta$. The stochastic error term, ϵ_n, gets a normal distribution located at zero with scale parameter $\sigma > 0$ (this error model ensures that the maximum likelihood value for β is at the least squares solution). We can alternatively write this model without the latent value ϵ_n as

$$y_n \sim \text{Normal}(x_n\beta, \sigma).$$

Here, $\epsilon_n = y_n - x_n\beta$ is implicit.

Noise model for Lotka-Volterra dynamics

The data y_i consists of measurements of the prey $y_{i,1}$ and predator $y_{i,2}$ populations at times t_i. The Lotka-Volterra equations will replace the determinsitic parts of the linear regression equations.

The true population sizes at time $t = 0$ are unknown—we only have measurements $y0_1$ and $y0_2$ for it. The true population initial population sizes at time $t = 0$ will be represented by a parameter $z0$, so that

$$z0_1 = u(t = 0) \quad \text{and} \quad z0_2 = v(t = 0).$$

Next, let $z_1, \ldots, z_N$ be the solutions to the Lotka-Volterra differential equations at times $t_1, \ldots, t_N$ given initial conditions $z(t = 0) = z0$. Each z_n is a pair of prey and predator population sizes at the specified times,

$$z_{n,1} = u(t_n) \quad \text{and} \quad z_{n,2} = v(t_n)$$

The z_n are deterministic functions of $z0$ and the system parameters $\alpha, \beta, \gamma, \delta$; thus z is not a parameter but a derived quantity.

The observed data is the form of measurements $y0$ of the initial population of prey and predators, and subsequent measurements y_n at times t_n, where $y0$ and the y_n consist of a pair of measured population sizes, for the prey and predator species.

Putting this together, the y_n (and $y0$) are measurements of the underlying predicted population z_n ($z0$). Because they are positive, the noise will be modeled on the log scale. This has the convenient side effect of making the error multiplicative (i.e., proportional) rather than additive.

$$\log y_{n,k} = \log z_{n,k} + \epsilon_{n,k}$$

$$\epsilon_{n,k} \sim \text{Normal}(0, \sigma_k)$$

where the z_n are the solutions to the Lotka-Volterra equations at times $t_1, \ldots, t_N$ given initial population $z0$. The prey and predator populations have error scales (on the log scale) of σ_1 and σ_2.

Weakly informative priors

The only remaining question is what to use for priors on the parameters. In general, the Stan Development Team has been recommending at least weakly informative priors. In practice, the parameter ranges for the Lotka-Volterra model leading to stable populations are well known.

For the parameters,

$$\alpha, \gamma \sim \text{Normal}(1, 0.5)$$

$$\beta, \delta \sim \text{Normal}(0.05, 0.05)$$

The noise scale is proportional, so the following prior should be weakly informative,

$$\sigma \sim \text{Lognormal}(0, 0.5)$$

Then, for the initial population of predator and prey, the following priors are weakly informative

$$z_{0,1} \sim \text{Normal}(\log(30), 1)$$

$$z_{0,2} \sim \text{Normal}(\log(5), 1)$$

Coding the model in Stan

Coding the system dynamics

Whenever a system of differential equations is involved, the system equations must be coded as a Stan function. In this case, the model is relatively simple as the state is only two dimensional and there are only four parameters. Stan requires the system to be defined with exactly the signature defined here for the function `dz_dt()`. The first argument is for time, which is not used here because the Lotka-Voltarra equations are not time-dependent. The second argument is for the system state, and here it is coded as an array $z = (u, v)$. The third argument is for the parameters of the equation, of which the Lotka-Voltarra equations have four, which are coded as $\theta = (\alpha, \beta, \gamma, \delta)$. The fourth and fifth argument are for data constants, but none areneeded here, so these arguments are unused.

```
real[] dz_dt(real t,       // time (unused)
             real[] z,     // system state
             real[] theta, // parameters
             real[] x_r,   // data (unused)
             int[] x_i) {
  real u = z[1];
  real v = z[2];

  real alpha = theta[1];
  real beta = theta[2];
  real gamma = theta[3];
  real delta = theta[4];

  real du_dt = (alpha - beta * v) * u;
  real dv_dt = (-gamma + delta * u) * v;

  return { du_dt, dv_dt };
}
```

After unpacking the variables from their containers, the derivatives of population with respect to time are defined just as in the mathematical specification. The return value uses braces to construct the two-element array to return, which consists of the derivatives of the system components with respect to time,

$$\frac{\mathrm{d}}{\mathrm{d}t} z \;=\; \frac{\mathrm{d}}{\mathrm{d}t}(u, v) \;=\; \left(\frac{\mathrm{d}}{\mathrm{d}t} u, \, \frac{\mathrm{d}}{\mathrm{d}t} v \right).$$

The data and parameters are coded following their specifications.

```
data {
  int<lower = 0> N;          // num measurements
  real ts[N];                // measurement times > 0
  real y0[2];                // initial measured population
  real<lower = 0> y[N, 2];   // measured population at measurement times
}
parameters {
  real<lower = 0> theta[4];   // theta = { alpha, beta, gamma, delta }
  real<lower = 0> z0[2];      // initial population
  real<lower = 0> sigma[2];   // measurement errors
}
```

The solutions to the Lotka-Volterra equations for a given initial state $z0$ are coded up as transformed parameters. This will allow them to be used in the model and inspected in the output. It also makes it clear that they are all functions of the initial population and parameters (as well as the solution times).

```
transformed parameters {
  // population for remaining years
  real z[N, 2]
    = integrate_ode_rk45(dz_dt, z0, 0, ts, theta,
                         rep_array(0.0, 0), rep_array(0, 0),
                         1e-6, 1e-5, 1e3);
}
```

The Runge-Kutta 4th/5th-order solver is specified here for efficiency (with suffix `_rk45`) because the equations are not stiff in the parameter ranges encountered for this data. The required real and integer data arguments in the second line are both given as size-zero arrays. The last line provides relative and absolute tolerances, along with the maximum number of steps allowed in the solver. For further efficiency, the tolerances for the differential equation solver are relatively loose for this example; usually tighter tolerances are required (smaller numbers).

If the solver runs into stiffness (the symptom of which is very slow iterations that may appear to be hanging), it is best to switch to the backward-differentiation formula solver, called with `integrate_ode_bdf`. The Runge-Kutta solver is twice as fast as the BDF solver for this problem on this data.

With the solutions in hand, the only thing left are the prior and likelihood. As with the other parts of the model, these directly follow the notation in the mathematical specification of the model.

```
model {
  // priors
  sigma ~ normal(0, 0.5);
  theta[1:2] ~ normal(0, 1);
  theta[3:4] ~ normal(0, 0.2);
  z0[1] ~ normal(10, 10);
  z0[2] ~ normal(50, 50);

  // likelihood
  y0 ~ lognormal(log(z0), sigma);
  for (k in 1:2)
    y[ , k] ~ lognormal(log(z[, k]), sigma[k]);
}
```

Fitting the Hudson Bay Company lynx-hare data

First, the data is setup in a form suitable for Stan.

```r
N <- length(lynx_hare_df$Year) - 1                      # num observations
after first
ts <- 1:N                                               # observation time
s just years
y0 <- c(lynx_hare_df$Hare[1], lynx_hare_df$Lynx[1])  # first observatio
n
y <- as.matrix(lynx_hare_df[2:(N + 1), 2:3])          # remaining observ
ations
y <- cbind(y[ , 2], y[ , 1]);                         # reverse order
lynx_hare_data <- list(N, ts, y0, y)
```

Next, the model is translated to C++ and compiled.

```r
model <- stan_model("lotka-volterra.stan")
```

Finally, the compiled model and data are used for sampling. Stan's default settings are sufficient for this data set and model.

```r
fit <- sampling(model, data = lynx_hare_data,
                seed=123)
```

The output can be displayed in tabular form, here limited to the median (0.5 quantile) and 80% interval (0.1 and 0.9 quantiles).

```r
print(fit, probs=c(0.1, 0.5, 0.9), digits=3)
```

```
## Inference for Stan model: lotka-volterra.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##               mean se_mean    sd    10%    50%    90% n_eff  Rhat
## theta[1]     0.546   0.002 0.074  0.456  0.543  0.644   994 1.000
## theta[2]     0.028   0.000 0.005  0.022  0.027  0.035  1211 1.001
## theta[3]     0.805   0.004 0.109  0.675  0.797  0.942   940 1.001
## theta[4]     0.024   0.000 0.004  0.019  0.024  0.030   968 1.002
## z0[1]       34.231   0.065 3.452 30.122 34.012 38.565  2862 1.003
## z0[2]        5.906   0.012 0.612  5.167  5.885  6.698  2558 1.001
## sigma[1]     0.290   0.001 0.054  0.228  0.284  0.361  2685 1.000
## sigma[2]     0.295   0.001 0.055  0.231  0.287  0.370  2461 1.001
## z[1,1]      49.597   0.130 5.457 43.257 49.159 56.512  1752 1.003
## z[1,2]       7.156   0.012 0.755  6.246  7.109  8.098  4000 1.000
## z[2,1]      66.030   0.213 8.131 56.476 65.388 76.265  1454 1.002
## z[2,2]      12.857   0.027 1.699 10.806 12.781 15.030  4000 0.999
## z[3,1]      65.681   0.192 8.258 55.751 65.180 76.195  1841 1.000
```

```
## z[3,2]       29.338   0.069  4.100 24.370 29.093  34.587  3543 0.999
## z[4,1]       38.468   0.076  4.800 32.707 38.213  44.680  4000 0.999
## z[4,2]       46.566   0.126  6.082 39.175 46.227  54.242  2323 1.000
## z[5,1]       19.320   0.033  2.116 16.812 19.177  21.944  4000 1.001
## z[5,2]       40.318   0.104  4.988 34.309 40.021  46.666  2288 1.001
## z[6,1]       13.389   0.027  1.388 11.735 13.305  15.124  2611 1.001
## z[6,2]       26.323   0.050  2.744 23.054 26.147  29.861  3065 1.002
## z[7,1]       13.005   0.029  1.334 11.402 12.959  14.677  2166 1.001
## z[7,2]       16.099   0.024  1.472 14.281 16.049  17.986  3833 1.002
## z[8,1]       15.701   0.030  1.464 13.922 15.643  17.507  2323 1.001
## z[8,2]       10.158   0.020  0.944  8.973 10.159  11.351  2337 1.001
## z[9,1]       21.395   0.029  1.700 19.280 21.341  23.525  3443 1.000
## z[9,2]        7.086   0.016  0.708  6.187  7.070   8.001  1966 1.001
## z[10,1]      30.908   0.040  2.347 28.039 30.782  33.996  3415 1.000
## z[10,2]       5.914   0.013  0.615  5.159  5.904   6.697  2143 1.001
## z[11,1]      45.128   0.095  4.173 40.153 44.832  50.381  1941 1.000
## z[11,2]       6.534   0.013  0.706  5.700  6.496   7.440  3018 1.001
## z[12,1]      62.320   0.186  7.288 53.603 61.764  71.685  1528 1.000
## z[12,2]      10.567   0.021  1.291  8.994 10.490  12.200  3660 1.001
## z[13,1]      69.037   0.216  8.680 58.708 68.317  79.964  1617 1.001
## z[13,2]      23.704   0.052  3.305 19.701 23.447  28.095  4000 1.001
## z[14,1]      46.260   0.108  5.812 39.360 45.895  53.749  2905 1.002
## z[14,2]      44.030   0.113  5.872 36.996 43.621  51.529  2693 1.000
## z[15,1]      22.714   0.046  2.816 19.290 22.512  26.362  3810 1.001
## z[15,2]      43.623   0.119  5.531 36.982 43.316  50.678  2175 1.000
## z[16,1]      14.244   0.029  1.552 12.348 14.169  16.235  2815 1.000
## z[16,2]      29.839   0.064  3.370 25.767 29.626  34.177  2808 1.000
## z[17,1]      12.794   0.028  1.322 11.218 12.743  14.473  2220 1.001
## z[17,2]      18.362   0.029  1.846 16.122 18.260  20.749  4000 1.000
## z[18,1]      14.751   0.030  1.493 12.957 14.677  16.638  2472 1.002
## z[18,2]      11.419   0.018  1.114 10.039 11.380  12.844  4000 1.000
## z[19,1]      19.623   0.031  1.952 17.218 19.521  22.124  4000 1.001
## z[19,2]       7.708   0.016  0.758  6.775  7.680   8.691  2328 1.000
## z[20,1]      28.066   0.046  2.930 24.549 27.865  31.825  4000 1.000
## z[20,2]       6.091   0.013  0.606  5.344  6.062   6.891  2066 1.000
## y0_rep[1]    35.609   0.180 11.388 22.875 33.948  50.712  4000 1.000
## y0_rep[2]     6.185   0.032  1.980  4.014  5.885   8.793  3717 1.000
## y_rep[1,1]   51.722   0.273 16.758 33.010 49.157  72.395  3756 1.000
## y_rep[1,2]    7.445   0.040  2.528  4.766  7.059  10.508  4000 0.999
## y_rep[2,1]   68.432   0.394 22.962 43.011 64.949  96.851  3403 1.000
## y_rep[2,2]   13.385   0.071  4.505  8.488 12.717  19.067  4000 1.000
## y_rep[3,1]   68.274   0.373 22.498 43.602 64.890  95.402  3647 1.000
## y_rep[3,2]   30.689   0.180 10.344 19.252 29.185  43.696  3306 1.000
## y_rep[4,1]   40.219   0.211 13.144 25.724 38.013  57.584  3867 1.000
## y_rep[4,2]   48.686   0.300 17.636 30.579 45.926  69.385  3446 1.001
## y_rep[5,1]   20.072   0.104  6.559 12.637 19.181  28.391  4000 0.999
```

```
## y_rep[5,2]  42.138   0.229 14.141 26.496 39.932  59.978  3817 0.999
## y_rep[6,1]  13.998   0.072  4.560  8.991 13.277  19.568  3996 1.000
## y_rep[6,2]  27.348   0.143  9.024 17.570 25.882  38.971  4000 1.000
## y_rep[7,1]  13.515   0.072  4.567  8.680 12.850  18.830  3990 1.000
## y_rep[7,2]  16.769   0.087  5.497 10.914 15.925  23.477  4000 1.000
## y_rep[8,1]  16.375   0.086  5.218 10.616 15.650  23.140  3691 1.000
## y_rep[8,2]  10.637   0.057  3.443  6.820 10.145  15.011  3622 0.999
## y_rep[9,1]  22.215   0.111  6.957 14.580 21.320  30.907  3910 0.999
## y_rep[9,2]   7.401   0.042  2.472  4.763  7.009  10.528  3513 1.000
## y_rep[10,1] 32.583   0.164 10.380 21.367 31.104  45.750  4000 1.000
## y_rep[10,2]  6.161   0.032  1.981  3.948  5.880   8.749  3871 1.000
## y_rep[11,1] 47.371   0.244 15.080 30.840 45.172  66.066  3824 1.000
## y_rep[11,2]  6.760   0.038  2.267  4.310  6.416   9.647  3596 1.000
## y_rep[12,1] 65.170   0.365 21.333 41.759 61.560  93.365  3408 1.000
## y_rep[12,2] 11.041   0.061  3.755  7.021 10.446  15.530  3823 1.000
## y_rep[13,1] 71.578   0.400 23.225 45.589 67.832 100.801  3365 1.002
## y_rep[13,2] 24.836   0.136  8.604 15.696 23.432  35.376  4000 1.000
## y_rep[14,1] 48.445   0.277 16.715 30.875 45.560  69.762  3652 1.001
## y_rep[14,2] 46.205   0.246 15.565 29.285 44.282  65.262  4000 1.000
## y_rep[15,1] 23.638   0.135  8.003 14.949 22.369  33.551  3518 1.000
## y_rep[15,2] 45.713   0.258 15.716 28.659 43.312  65.617  3717 1.001
## y_rep[16,1] 14.834   0.079  4.763  9.499 14.218  20.729  3675 1.000
## y_rep[16,2] 31.065   0.177 10.493 19.811 29.442  44.113  3510 1.001
## y_rep[17,1] 13.225   0.069  4.212  8.405 12.623  18.545  3773 1.001
## y_rep[17,2] 19.181   0.098  6.168 12.484 18.261  26.998  4000 1.000
## y_rep[18,1] 15.362   0.079  4.980  9.827 14.714  21.514  4000 1.001
## y_rep[18,2] 11.860   0.062  3.850  7.608 11.330  16.555  3855 1.000
## y_rep[19,1] 20.654   0.108  6.806 13.256 19.569  29.732  4000 0.999
## y_rep[19,2]  8.031   0.044  2.657  5.088  7.663  11.335  3673 1.000
## y_rep[20,1] 29.251   0.146  9.265 19.171 27.954  40.696  4000 1.000
## y_rep[20,2]  6.342   0.032  2.015  4.100  6.053   8.899  4000 1.000
## lp__        21.011   0.063  2.219 18.101 21.382  23.503  1247 1.004
##
## Samples were drawn using NUTS(diag_e) at Tue Nov  7 16:13:15 2017.
## For each parameter, n_eff is a crude measure of effective sample siz
e,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The R-hat values are all near 1, which is consistent with convergence. The effective sample size estimates for each parameter are sufficient for inference. Thus we have reason to trust this fit.

The expected values `z` are unlike the replicated draws `y_rep` in two ways. First, their posterior has much lower variance and much narrower 80% intervals. This is to be expected, as the `y_rep` additional takes into account measurement and unexplained variance, whereas `z` only takes into account parameter estimation uncertainty. Second, the mean values of `z` are lower than the corresponding values of `y_rep`. This is because `y_rep` is adding a lognormal error term, which has a positive expectation as it is constrained to be positive; this positivity is also a factor in the fits that are derived for `z`.

Comparing the fitted model to data

Using a non-statistically motivated error term and optimization, Howard (2009, Figure 2.10) provides the following approximate point estimates for the model parameters based on the data.

$$\hat{\alpha} = 0.55, \;\; \hat{\beta} = 0.028, \;\; \hat{\gamma} = 0.84, \;\; \hat{\delta} = 0.026$$

Our model produced the following point estimates based on the posterior mean, which minimizes expected squared error,

$$\hat{\alpha} = 0.55, \;\; \hat{\beta} = 0.028, \;\; \hat{\gamma} = 0.80, \;\; \hat{\delta} = 0.024$$

and the posterior median, which minimizes expected absolute error,

$$\hat{\alpha} = 0.54, \;\; \hat{\beta} = 0.035, \;\; \hat{\gamma} = 0.80, \;\; \hat{\delta} = 0.030.$$

The estimates are very similar to each other and to Howard's.

Howard then plugs in these point estimates and derives the most likely populations z (including the initial population $z0$). Rather than plugging in point estimates to get point predictions, we will adjust for the two forms of uncertainty inherent in our model. First, there is estimation uncertainty, which we characterize with the posterior density $p(\alpha, \beta, \gamma, \delta, z_0, \sigma \mid y)$. The second form of uncertainty is the observation error and unexplained variation, which are both rolled into a single sampling distribution, $\log y_n \sim \mathsf{Normal}(\log z_n, \sigma)$. As in the Stan implementation, z_n is the solution to the differential equation conditioned on the parameters $\alpha, \beta, \gamma, \delta$ and initial state z_0. Altogether, we will be repulating new y values, which we write as y^{rep}, according to the posterior predictive distribution,

$$p(y^{\text{rep}}|y) \;\; = \;\; \int p(y^{\text{rep}}|\theta) \; p(\theta|y) \; \mathrm{d}\theta.$$

where $\theta = (\alpha, \beta, \gamma, \delta, z_0, \sigma)$ is the vector of parameters for the model. Then, we calculate the posterior mean, which is itself an expectation,

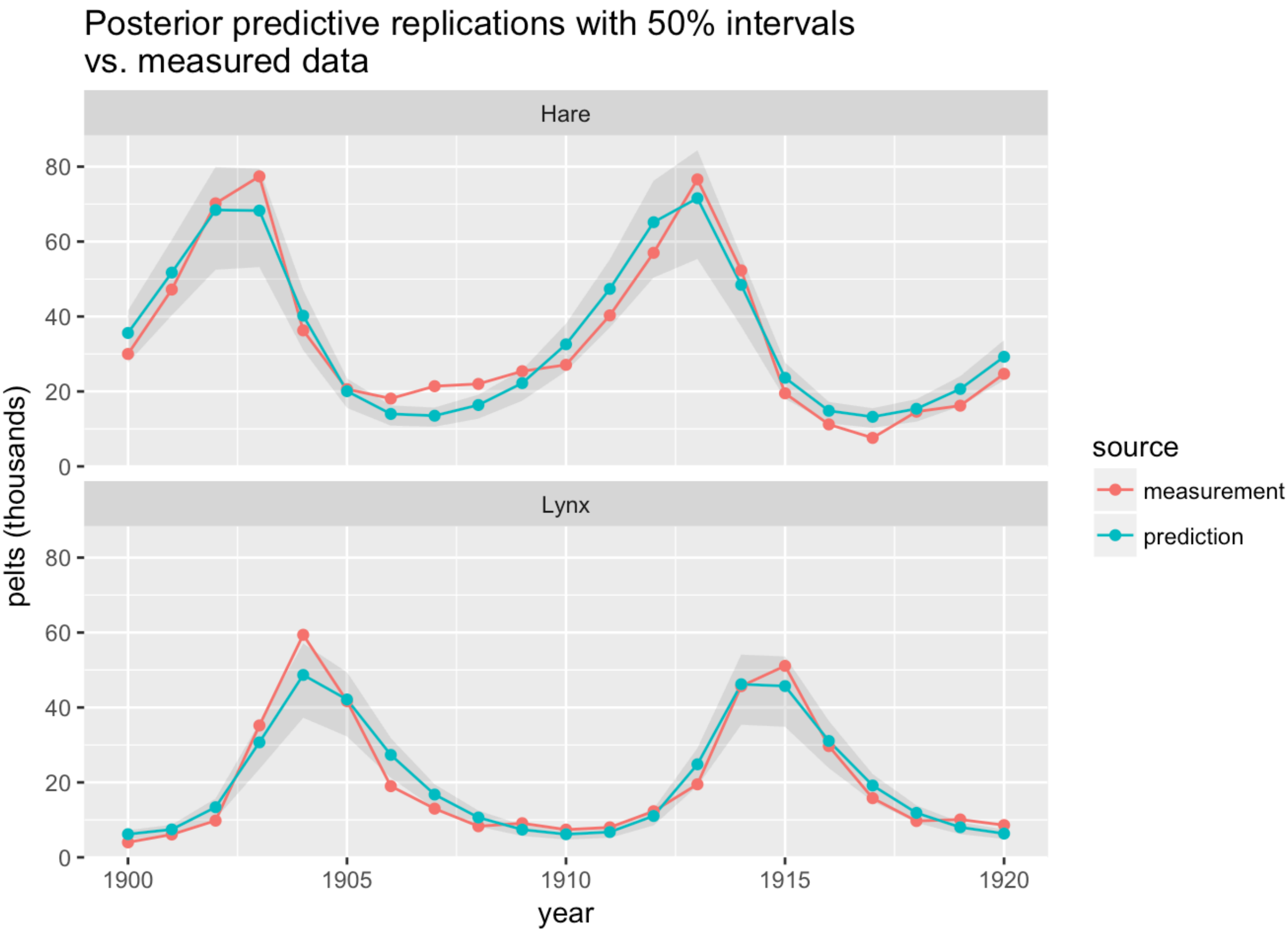
$$
\begin{aligned}
\hat{y}^{\text{rep}} &= \mathbb{E}[y^{\text{rep}}|y] \\
&= \int y^{\text{rep}}\, p(y^{\text{rep}}|y)\, \mathrm{d}y^{\text{rep}} \\
&= \int y^{\text{rep}}\, p(y^{\text{rep}}|\theta)\, p(\theta|y)\, \mathrm{d}y^{\text{rep}}\, \mathrm{d}\theta \\
&\approx \frac{1}{M} \sum_{m=1}^{M} y^{\text{rep}(m)}
\end{aligned}
$$

As with other posterior expectations, the Bayesian point estimate is given by a simple average over simulated values, where $y^{\text{rep}(m)}$ is just the result of simulating the value of y^{rep} according to the generative model based on parameter draw $\theta^{(m)}$.

The posterior predictive estimates of the dynamics are shown below, along with the raw data on number of pelts collected.

```
z0_draws <- extract(fit)$z0
z_draws <- extract(fit)$z
y0_rep_draws <- extract(fit)$y0_rep
y_rep_draws <- extract(fit)$y_rep
predicted_pelts <- matrix(NA, 21, 2)
min_pelts <- matrix(NA, 21, 2)
max_pelts <- matrix(NA, 21, 2)
for (k in 1:2) {
  predicted_pelts[1, k] <- mean(y0_rep_draws[ , k])
  min_pelts[1, k] <- quantile(y0_rep_draws[ , k], 0.25)
  max_pelts[1, k] <- quantile(y0_rep_draws[ , k], 0.75)
  for (n in 2:21) {
    predicted_pelts[n, k] <- mean(y_rep_draws[ , n - 1, k])
    min_pelts[n, k] <- quantile(y_rep_draws[ , n - 1, k], 0.25)
    max_pelts[n, k] <- quantile(y_rep_draws[ , n - 1, k], 0.75)
  }
}

lynx_hare_melted_df <- melt(as.matrix(lynx_hare_df[, 2:3]))
colnames(lynx_hare_melted_df) <- c("year", "species", "pelts")
lynx_hare_melted_df$year <-
  lynx_hare_melted_df$year +
  rep(1899, length(lynx_hare_melted_df$year))

Nmelt <- dim(lynx_hare_melted_df)[1]
lynx_hare_observe_df <- lynx_hare_melted_df
lynx_hare_observe_df$source <- rep("measurement", Nmelt)
```

```
lynx_hare_predict_df <-
  data.frame(year = rep(1900:1920, 2),
             species = c(rep("Lynx", 21), rep("Hare", 21)),
             pelts = c(predicted_pelts[, 2],
                       predicted_pelts[, 1]),
             min_pelts = c(min_pelts[, 2], min_pelts[, 1]),
             max_pelts = c(max_pelts[, 2], max_pelts[, 1]),
             source = rep("prediction", 42))

lynx_hare_observe_df$min_pelts = lynx_hare_predict_df$min_pelts
lynx_hare_observe_df$max_pelts = lynx_hare_predict_df$max_pelts

lynx_hare_observe_predict_df <-
  rbind(lynx_hare_observe_df, lynx_hare_predict_df)

population_plot2 <-
  ggplot(data = lynx_hare_observe_predict_df,
         aes(x = year, y = pelts, color = source)) +
  facet_wrap( ~ species, ncol = 1) +
  geom_ribbon(aes(ymin = min_pelts, ymax = max_pelts),
          colour = NA, fill = "black", alpha = 0.1) +
  geom_line() +
  geom_point() +
  ylab("pelts (thousands)") +
  ggtitle("Posterior predictive replications with 50% intervals\nvs. me
asured data")
population_plot2
```



This posterior predictive check shows that the model fit is consistent with the data, with around 50% of the data points falling within the 50% intervals.

How large are the populations?

Going on the assumption that the number of pelts collected is proportional to the population, we only know how the relative sizes of the populations change, not their actual sizes.

This model could be combined with a mark-recapture model to get a better handle on the actual population size. Mark-recapture gives you an estimate of actual numbers and the Lotka-Volterra model would provide information on relative change in the predator and prey populations.

Extensions to the model

The Lotka-Volterra model is easily extended for realistic applications in several ways.

1. Predictors can be rolled into the system state to take into the dynamnics to account for things like the correlation of populations with the abundance of food.
2. The model may be extended beyond two species. The dynamics for each species will reflect that it may stand in predator-prey relations to multiple other species.

3. Additional data for population observations may be included, such as adding a mark-recapture model for tag-release-recapture data of populations.

Exercises

1. Extend predictions another 50 years into the future and plot as in the last plot. This can be done by extending the solution points in the transformed parameters, but is more efficiently done in the generated quantities block.
2. Write a Stan model to simulate data from this model. First simulate parameters from the prior (or pick ones consistent with the priors). Then simulate data from the parameters. Finally, fit the model in Stan and compare the coverage as in the last plot in the case study.
3. Suppose that several of the measurements are missing. Write a Stan program that uses only the observed measurements. How robust is the fit to missing a few data points?
4. Write a Stan model that predicts the population at finer-grained intervals than a year (such as every three months). Can the model be formulated to only use the yearly data? Do the smoother plots for predicted populations make sense? Does this fit better or worse than the original model?
5. Replace the lognormal error with a simple normal error model. What does this do to the `z` estimates and to the basic parameter estimates? Which error model fits better?

References

- Howard, P. (2009). Modeling Basics. Lecture Notes for Math 442, Texas A&M University.
- Lotka, A. J. (1925). *Principles of physical biology*. Baltimore: Waverly.
- Volterra, V. (1926). Fluctuations in the abundance of a species considered mathematically. *Nature*, 118(2972), 558-560.
- Volterra, V. (1927). *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari.

Appendix: Session information

```
sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X Yosemite 10.10.5
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] rstan_2.16.2         StanHeaders_2.16.0-1 ggplot2_2.2.1
## [4] reshape_0.8.7        rmarkdown_1.5
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.8      knitr_1.17       magrittr_1.5     munsell_0.4.
3
##  [5] colorspace_1.3-2 stringr_1.1.0    plyr_1.8.4       tools_3.3.2
##  [9] parallel_3.3.2   grid_3.3.2       gtable_0.2.0     htmltools_0.
3.6
## [13] yaml_2.1.14      lazyeval_0.2.0   rprojroot_1.2    digest_0.6.1
0
## [17] assertthat_0.1   tibble_1.2       gridExtra_2.2.1  codetools_0.
2-15
## [21] inline_0.3.14    evaluate_0.10    labeling_0.3     stringi_1.1.
2
## [25] scales_0.4.1     backports_1.0.5  stats4_3.3.2
```

Appendix: Licenses