A novel approach to detecting changes in dynamic rules governing population fluctuations

C.A. Bahlai And E.F. Zipkin

**Abstract**

**Introduction**

In this paper, we develop a generalizable tool for detecting shifts in dynamic regime in population time series data. The regime shift model is based on a script initially developed in (Bahlai et al. 2015) to understand changes governing the internal dynamics of population regulation in the invasion process of *Harmonia axyridis* in the two decades following their arrival in southwestern Michigan. This model was able to detect the precise timing external factors affecting the internal drivers of population cycling (namely, the invasion and then subsequent pesticidal control of a preferred prey item), suggesting that the model itself may have broader applicability in understanding drivers of population dynamics for other species important in conservation and ecosystem management.

This paper describes how this tool was developed as a suite of functions for examining population time series data for the presence, location, and magnitude of shifts in dynamic regime.

**Methods**

A model script, simulations, and two case studies using population data from insect population monitoring were scripted and run in R Version 3.3.3 “Another Canoe” (R Development Core Team 2017) run within the RStudio Integrated development environment 1.0.136 (RStudio Team 2015). All data manipulations, analyses and figure scripts, including the complete development history, are publicly available in a Github repository at <https://github.com/cbahlai/monarch_regime>.

*The model*

For our purposes, we use the Ricker model to describe the dynamics within a system.

The Ricker model is a single-variable discrete time model in which N(t+1), the population at time t+1, is a function of N(t), where N(t) is the measure of population size in year t, and the parameters K and r (the carrying capacity and per capita yearly rate of increase respectively) are estimated for the population during the time period modeled. The Ricker model was selected for our purposes because 1) it does not rely on any external information, other than population data over time, to be fit; 2) only two parameters need to be estimated, and those parameters have ecologically meaningful interpretations, and 3) in our experience, the model simply fits insect population data well.

We used an iterative, model-selection based process to determine if, and when, shifts in dynamic regime had occurred within a given time series. To achieve this, first, the Ricker model was fit to the entire population time series, then the population time series was subdivided into all possible combinations of 2, 3, …, n subsets of sequential data points (hereafter, ‘break point combination’, for example, a twelve year series with a break point combination of 4, 8 would be broken into subset 1 = 1, , 2, 3, 4; subset 2 = 5, 6, 7, 8; subset 3 = 9, 10, 11, 12) and the Ricker model was fitted to each of the subsets produced for each break point combination. Break point combinations were constrained to only include subsets with more than three sequential data points to avoid over-fitting.

After fitting each subset for a given break point combination, the Akaike Information Criteria for each subset were summed together, providing an overall AIC for the fit. To further account for the effect of small series sizes, we calculated AICc (AIC correction for small samples) by using the total number of parameters estimated for the fit (where nparameters = 3 (nbreaks + 1), because for each fit, r, K and error were estimated), and this factor was added to the total AIC for each break point combination. Then, AICc values were used to rank fits for each break point combination, and fits for break point combinations with lower AICc values were considered to have better performance. When AICc values differed by two units or less, model performance was considered equivalent (Burnham and Anderson 2002), so when models were found with equivalent performance, the simplest model (ie: the one with fewest parameters and break points) was selected for further analysis. If the top-ranked models had the same number of parameters, the one with the numerically lowest AICc was considered best-ranked for the purpose of further analysis.

*Technical implementation*

The ‘regime shift detector’ was implemented as a series of R functions to enable a user starting with a standard data frame of population observations at a standard time intervals to quickly generate a report on the fit of the model described above to their own data. Herein we summarize the role of each function, but we encourage the reader to download the script file directly from https://github.com/cbahlai/monarch\_regime/blob/master/regime\_shift\_detector.R for the details of implementation, included as line-by-line comments in the script.

addNt1- takes a raw data frame with columns for year, population abundance measure, and converts it to a three column data frame with year, population in year, population in the next year, and gives the data consistent column names for use in downstream functions. Most of the functions described below require this function to be used to transform data prior to use, unless otherwise specified.

AICcorrection- takes a data series and the number of breaks used in a given fit to calculate the AICc correction factor to be added to the total AIC for the fit

rickerfit- fits the Ricker model using the Levenberg-Marquart nonlinear least squares method for nonlinear model fitting to a given data frame. This function calls the nlsLM function from minpack.lm (Elzhov et al. 2016). To aide in model convergence, the function also computes a realistic starting value for K by calculating the mean of the population (based on the assumption that a population being fit to the Ricker model is likely fluctuating around its carrying capacity). The starting value for r was set at 1.5. The function outputs a vector containing the AIC, the estimate for r, its standard error, and the estimate for K and its standard error.

splitnfit- takes a given data fame, fits the complete data series with the rickerfit function, then subsets it in two by creating a break point three years after the start of the series, calls the rickerfit function to fit the data from each subset produced there. Then the function walks through the data, increasing the break point by one time step each iteration, and compiles the AICs and break pointss used for each fit, resulting in a data frame of break point combinations and respective AICs.

findbreakable- examines the output from the splitnfit function to determine if any of the break point combinations produced might be further subdivided (ie: has enough points to not violate the rule we set to only fit series with greater than 4 points).

subsequentsplit- used output from findbreakable function to identify cases where data can further be subsetted using the splitnfit function, feeds those cases in, and compiles results together with that produced by simpler break point combinations produced by splitnfit.

nbreaker- uses splitnfit, findbreakable, and subsequentsplit, combined with input data, to create a data frame consisting of a column of all possible break point combinations, and the respective AICs of the resultant fits. This function uses an iterative approach to allow simpler functions that break a data into two parts to be used to find an unlimited number of break points (within constraints of series length).

AICtally- takes data in, subjects it to nbreaker, pulls out the AICs produced by nbreaker, adds them together and counts the number of fits performed, number of breaks in the data, computes the corrected AICc using AICcorrection. and returns these values as a data frame.

allfits- appends the results of nbreaker and AICtally together into a single data frame.

equivalentfit- takes in data, feeds it to allfits, and uses the output from allfits to pull out the subset of all equivalently-performing breakpoint combination fits (here, within 2 units of AICc), and outputs these fits as a data frame.

bestfit- feeds data to the equivalentfit function to get a data frame describing equivalent fits, and uses our decision rules (in this case, select the break point combination requiring the fewest parameter estimates from the list of equivalent models, and if there are multiple equivalent models still, select the one with the numerically lowest AICc) to output the specifics of that break point combination as a data frame.

bestmodel- feeds data to the bestfit function to identify the best break point combination, and then use that information to create a data frame describing the parameter estimates (r, K and standard error for each) for fitting the Ricker model to each of the subsets of timeseries, allowing a user to quantify how the dynamic rule found by fitting the model changes at each break point.

RSdetector- uses the raw time series data to produce a report, calling all the previous functions, either directly, or through other functions, with short explanatory text preceding each result. First, a simple plot of population over time is produced (N(t) by t), then data is fed to the addNt1 function, and the resultant N(t), N(t+1) data is plotted to visualize the potential for the data to conform to a Ricker curve. Then, the data is fed through the allfits function, producing a complete list of all break point combinations tested and their respective fit statistics. The data is subsequently fed through the equivalentfit and bestfit functions so that a user can assess how the decision rules specified impacted the selection of the best model. Finally, the data is fed through the bestmodel function to produce the set of regression parameters for each time series subset produced by the best break point combination found.

*Simulations*

A function was created to simulate time series data following Ricker dynamics under set break point combination conditions. The function takes values for of start year, number of years to simulate, % noise, a starting population N, starting values for K and r, a break point combination, a % change in K and a % change in r to be simulated for each break point. Percent noise was included as a means to simulate sampling error that would be observed in a real sampling plan, and was simulated by creating a continuous interval from 100%-noise to 100%+noise, randomly sampling from within that interval, and multiplying the predicted observation N(t+1) by the resultant value. Change of K and r at each break point were randomly selected by the script to either increase or decrease by the given % change.

The simulated data were fed into a function that tested the regime shift detector model was able to identify the breaks as set for the simulation by comparing the input conditions to those output by the bestmodel function in the regime shift detector script file. Results of comparing the input to the output were encoded as follows:

1. script was successful at detecting all break points and simulation conditions
2. script identified all simulated breaks, but also found one or more ‘extra’ breaks
3. script missed one of the simulated breaks, but all others found were correct
4. script identified the correct number of breaks, but one or more breaks were mismatched
5. no correct breaks were identified by the script, or breaks were identified in a no-break scenario

A base scenario was constructed, with start year =1, number of years = 25, a starting population of 3000, a sampling error of up to 5%, a starting value for K = 2000, a starting value for r =2, a % change at each break point of 40 and 20% for K and r respectively, and a set of 0, 1, 2, or 3 break points randomly selected from within the possible values defined by start year and number of years.  
  
This script was used to test the frequency of a given result under a variety of different scenarios by changing the value of one parameter at a time from the base scenario. Specifically, the impact of different sizes of regime shift was tested by creating scenarios where the % change in r and K were individually modified at 10% intervals from 0 to 90%, the impact of length of time series was examined by extending the length of the time series by 2 year intervals from 25 to 33, and the impact of sampling error was tested at 1, 5, 10%, and every 10% interval thereafter to 90%. Each of these scenarios was run on 0, 1, 2 and 3 break point combinations, and each scenario by break point combination was iterated 500 times with newly simulated data. Results from the simulations were compiled at intervals of 50 iterations and stored as csv files within the project repository for further analysis. The complete script which simulates the data, applies the regime shift detector, and compiles the results is available at: <https://github.com/cbahlai/monarch_regime/blob/master/simulations.R>

*Case studies*

We used two case studies to test the performance of the regime shift detector script on population time series data produced under natural conditions as parts of observational experiments. Both case studies involve nearly two decades of observations of economically or culturally important insect species, however, one case examines an invasion process, and another examines a population decline, both occurring over approximately the same time period in recent history.

*Harmonia axyridis* in southwestern Michigan

Monarch butterflies in Mexican overwintering grounds

Results

Discussion

The rickerfit function was set to have a starting value of r at 1.5. for populations expected to deviate from this value dramatically, setting this value to one closer to the expected value will aide in model convergence.

Conclusions

References

**Bahlai, C. A., W. vander Werf, M. O’Neal, L. Hemerik, and D. A. Landis**. **2015**. Shifts in dynamic regime of an invasive lady beetle are linked to the invasion and insecticidal management of its prey. Ecological Applications.

**Burnham, K. P., and D. R. Anderson**. **2002**. Model selection and multimodal inference: a practical information-theoretic approach, 2nd ed. Springer Science + Business Media, LLC, New York.

**Elzhov, T. V., K. M. Mullen, A.-N. Spiess, and B. Bolker**. **2016**. minpack. lm: R Interface to the Levenberg-Marquardt Nonlinear Least-Squares Algorithm Found in MINPACK, R package version 1.2-1.

**R Development Core Team**. **2017**. R: A Language and Environment for Statistical Computing 3.3.3. R Foundation for Statistical Computing.

**RStudio Team**. **2015**. RStudio: Integrated Development for R (RStudio, Inc., Boston, MA, 2015). URL: https://www. rstudio. com/products/rstudio.