Markov chain logistic map

Christopher Burch

November 12, 2016

Logistic Map

The Logistic Map is a thing that exists. It's behavior primarily depends on the value of the variable r.

Markov Chain for r-values

The r-value in the traditional logistic map remains constant over all iterations. Using a Markov Chain to determine the value of r at each iteration introduces more randomness into the process and may better represent a biological system where the reproductive rate varies over time.

This was accomplished by constructing a matrix of transition values for each possible state. For this model, it was assumed that r-values can change at the beginning of each iteration, and that r-values close to the current r-value are more likely than r-values further away. In other words, the probability of transitioning to a given state is inversely proportional to that state's distance from the current state.

Generating the r-values

Once the transition matrix was computed, the actual moves from state to state were calculated. For each iteration of the model a random number is drawn and compared to the cumulative sum of tranistion probabilites, which then yields the appropriate r-value for the iteration.

```
getMoves <- function(states, transition.matrix, nsteps, init = 0){
    current.state <- ifelse(init == 0, sample(states, 1), init) # initial random state
    init.state <<- current.state
    moves <- numeric(length = nsteps)
    for(i in 1:nsteps){
        # check the current row of the transition matrix
        row.c <- which(states == current.state)
        # draw a random number from 0 to 1
        rn <- runif(1)
        # compare number with cumulative sum of pr on selected row
        # get new state from transition matrix row
        current.state <- states[which(transition.matrix[row.c, ] >= rn)[1]]
        # save to output vector
        moves[i] <- current.state
    }
    return(moves)
}
```

Setting initial values

There are several variables which control the output of the process:
nstates = the number of possible r-value states
start, **end** = the start and ending values of the range of possible r-values
pr.remain = the probability that the current r-value will not change

init = the initial r-value; setting to 0 causes a random value to be selected
nsteps = the number of iterations for the model to run

```
nstates = 40
start = 1
end = 4
pr.remain = .995
init = 1 # setting to 0 uses a random number
nsteps = 1000
```

Generating the outcome population values

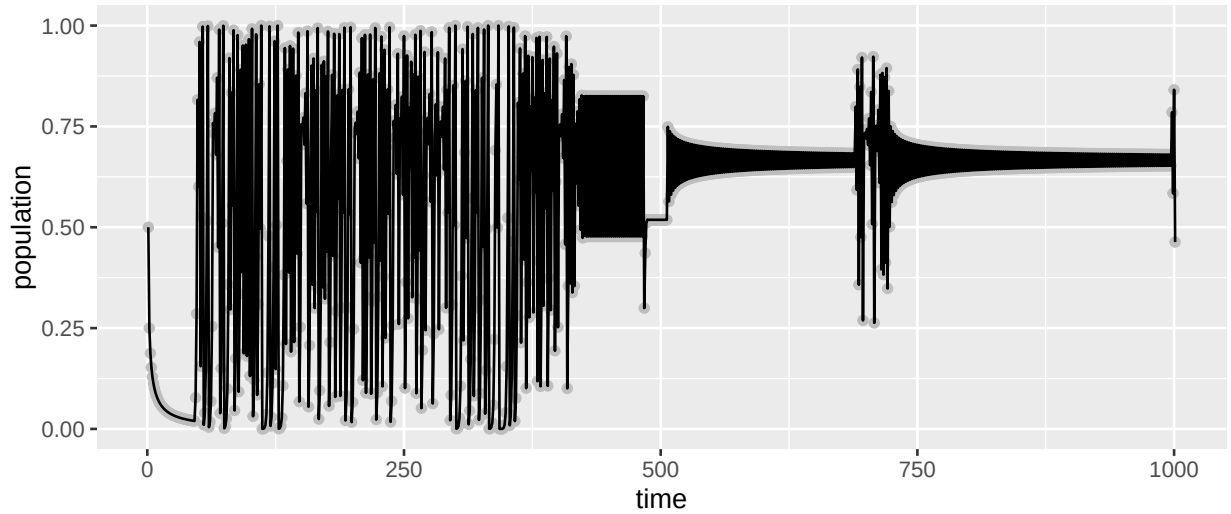
```
set.seed(1778)
prtrans <- buildMarkov(nstates, pr.remain, start, end) %>% apply(1, cumsum) %>% t
states <- seq(start, end, length.out = nstates)
init <- ifelse(init == 0, init, states[which(states >= init)[1]])
m <- getMoves(states, prtrans, nsteps, init)

vals <- logisticMapMarkov(m, x0 = 0.5)
```

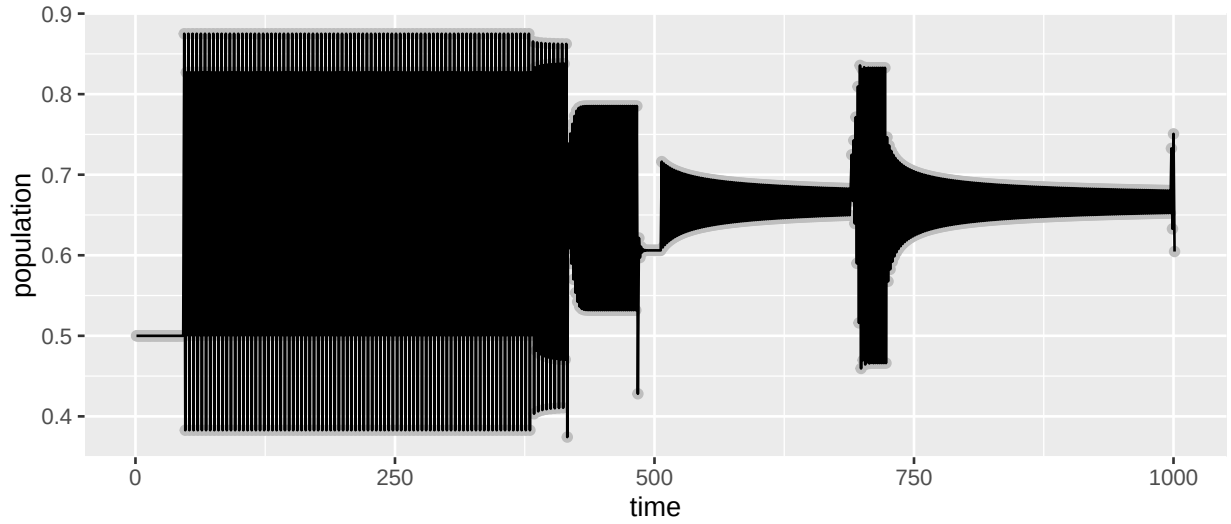
Plotting the output

The first plot shows a system with an initially low r-value (1) which then shifts to a very high r-value (4), resulting in wildly oscillating behavior. The population drops to almost zero several times; if the model were more accurate and didn't allow for close-to-zero values the population likely would have gone extinct. This pattern then changes again, and shows tight oscillation when the r-value shifts to 3.0.

```
vals.dt <- data.frame(time = 1:length(vals), population = vals, r_val = c(init.state, m))
g1 <- ggplot(vals.dt, aes(x = time, y = population))
g1 + geom_point(col = "grey") + geom_line(col = "black")
```



This plot shows a system in which the possible r-values have been restricted between 2 and 3.5.



And this plot shows how the same system varies when then probability of a r-value change increases from .005 to .25.