Frame Based Modeling

Therese Donovan

2020-04-28

In this exercise, we will build the frame-based models that Tony introduced in lecture.

Markov Matrix Models

```
tm <- matrix(data = c(0.7, 0.1, 0.5, 0.1, 0.8, 0, 0.2, 0.1, 0.5),
             nrow = 3,
             ncol = 3,
             byrow = TRUE)
rownames(tm) <- c("J", "W", "S")
colnames(tm) <- c("J", "W", "S")

tm
```

```
##     J   W   S
## J 0.7 0.1 0.5
## W 0.1 0.8 0.0
## S 0.2 0.1 0.5
```

The top of the matrix represents the state of the ecosystem at time step t. Each row gives the probability of a state at time t transitioning to a given state in time step t+1. For example, if the state is W (white pine) at time t, the probability that it will transition to J (jack pine) is 0.1, the probability that it will remain W is 0.8, and the probability that it will transition to S (spruce) is 0.1.

The column sums must equal 1, which we can confirm with the `colSums()` function:

```
colSums(tm)
```

```
## J W S
## 1 1 1
```

You can use `sample()` function to help model the state in time t+1. Here, we will will assume the ecosystem has a jack pine state at time t. Its state in t+1 is probabalistic. We will sample from the rownames (J, W, S) of our matrix one time, using the probabilities listed in column 1 of our matrix, tm.

```
sample(x = rownames(tm), size = 1, prob = tm[,1], replace = T)
```

```
## [1] "J"
```

Do this 100 times, and you should see that ~70% of trials result in a "J", 20% result in "S", and 10% result in "W".

```
trials <- sample(x = rownames(tm), size = 100, prob = tm[,1], replace = T)
trials
```

```
##   [1] "J" "J" "J" "S" "J" "J" "S" "S" "J" "W" "W" "J" "J" "J" "J" "J" "J" "S"
##  [19] "J" "J" "J" "J" "S" "J" "J" "S" "J" "J" "S" "J" "W" "J" "J" "J" "J" "J"
```

```
##  [37] "J" "J" "J" "J" "J" "S" "J" "J" "S" "J" "J" "J" "J" "J" "J" "S" "J" "S"
##  [55] "J" "W" "S" "J" "J" "J" "J" "J" "J" "S" "J" "J" "J" "J" "J" "J" "J" "J"
##  [73] "J" "J" "J" "S" "W" "S" "J" "J" "J" "J" "W" "J" "J" "J" "J" "J" "J" "J"
##  [91] "W" "J" "J" "J" "J" "S" "J" "J" "J" "W"
```

```
# compute the frequencies of the trials
table(trials)
```

```
## trials
##  J  S  W
## 76 16  8
```

If you repeat this line of code, you'll certainly get different results. To ensure repeatability in R, use the `set.seed()` function:

```
# run the first set
set.seed(104)
set1 <- sample(x = rownames(tm), size = 100, prob = tm[,1], replace = T)
set1
```

```
##   [1] "J" "S" "S" "W" "S" "J" "J" "J" "S" "J" "J" "J" "J" "J" "W" "J" "S" "J"
##  [19] "J" "J" "S" "S" "J" "J" "S" "J" "J" "J" "J" "J" "J" "S" "S" "J" "J" "J"
##  [37] "J" "W" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "S" "J" "J" "J" "J" "S"
##  [55] "S" "J" "J" "S" "J" "J" "W" "J" "J" "S" "J" "J" "J" "S" "S" "J" "J" "J"
##  [73] "J" "J" "J" "S" "S" "J" "J" "J" "J" "J" "S" "J" "J" "J" "S" "W" "J" "S"
##  [91] "J" "W" "J" "J" "S" "J" "J" "S" "J" "W"
```

```
# print a message
cat("New run...\n")
```

```
## New run...
```

```
# run the second set
set.seed(104)
set2 <- sample(x = rownames(tm), size = 100, prob = tm[,1], replace = T)
set2
```

```
##   [1] "J" "S" "S" "W" "S" "J" "J" "J" "S" "J" "J" "J" "J" "J" "W" "J" "S" "J"
##  [19] "J" "J" "S" "S" "J" "J" "S" "J" "J" "J" "J" "J" "J" "S" "S" "J" "J" "J"
##  [37] "J" "W" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "S" "J" "J" "J" "J" "S"
##  [55] "S" "J" "J" "S" "J" "J" "W" "J" "J" "S" "J" "J" "J" "S" "S" "J" "J" "J"
##  [73] "J" "J" "J" "S" "S" "J" "J" "J" "J" "J" "S" "J" "J" "J" "S" "W" "J" "S"
##  [91] "J" "W" "J" "J" "S" "J" "J" "S" "J" "W"
```

```
# test for equality
identical(set1, set2)
```

```
## [1] TRUE
```

To do this in a model, we simply provide our starting condition, and then loop through multiple decades.

```
# set the random number seed for reproducibility
set.seed(10)

# set up an empty vector to hold our results
results <- vector(length = 100)

# populate the first entry with the current state of the ecosystem
results[1] <- "J"
```

```r
# loop through 100 more time steps
for (i in 2:100) {
  col.index <- match(results[i - 1], colnames(tm))
  results[i] <- sample(x = rownames(tm),
                       size = 1,
                       prob = tm[,col.index],
                       replace = T)
}

results
```

```
##   [1] "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J" "J"
##  [19] "J" "J" "S" "S" "S" "S" "J" "J" "S" "S" "J" "S" "J" "J" "J" "J" "S" "J"
##  [37] "S" "S" "S" "S" "S" "J" "J" "J" "S" "J" "J" "J" "J" "J" "S" "J" "W" "W"
##  [55] "W" "W" "W" "W" "W" "W" "W" "W" "W" "W" "W" "S" "S" "S" "J" "J" "J" "J"
##  [73] "J" "J" "J" "J" "J" "J" "W" "W" "W" "S" "J" "J" "J" "J" "J" "S" "J" "J"
##  [91] "J" "J" "S" "S" "S" "J" "S" "J" "J" "S"
```

Markov models are fairly easily to implement (in R or in a spreadsheet). However, the mechanisms that drive the transition matrix probabilities get somewhat lost. The state and transition modeling approach provides a description of each transition can spell the mechanism, but these are not included in the model per se. The answer: frame-based models to the rescue!

Frame Based Models

Frame based modeling is an approach that describes each state as its own modeling frame. Each frame model asks two simple questions, assuming the model has been "entered" (i.e., the ecosystem is in a given state).

1. Am I still in this frame?
2. If not, which frame am I in?

At every time step, you are in a given frame, and you only consider those factors that answer the questions above. For instance, if it is a spruce ecosystem, there is a 50% chance of fire. If a fire occurs, the state changes to jack pine. If no fire, the system stays in spruce. Importantly, frame-based models can include management options. If we put the fire out, then the system remains in spruce.

Building a frame based model requires that you build tiny sub-models that spell out the transition rules, given the state you are currently in. For example, Tony outlines this transition:

- J to W: This switch can occur in one of two ways. If there is an interval of 40+ years between successive fires, which would allow white pine trees to grow to a height at which they will survive fire, then a fire will destroy almost all jack pine and spruce, but leave a cohort of white pine trees. If fires are strictly controlled for between 80-100 years, then jack pine will die, leaving behind an understory that is dominated by white pine, provided the density of deer is low.
- J to S: This switch occurs after 80-100 years if the fires are strictly controlled and the density of deer is high.

So each frame has its own set of rules that drive its transitions.

Let's start by creating a "transition matrix" for our frame-based model.

```r
tm <- matrix(data = NA, nrow = 3, ncol = 3)

colnames(tm) <- c("J", "W", "S")
rownames(tm) <- c("Prob Fire", "Deer Density", "Harvest Age")

tm
```

```
##                J  W  S
## Prob Fire     NA NA NA
## Deer Density NA NA NA
## Harvest Age  NA NA NA
```

Next, let's fill in our matrix for J. Here, instead of low or high for deer density, we will let 0 be low and 1 be high. We need to do this because a matrix in R can only consist of one kind of class, and in this case our matrix will store only numbers.

```
tm[,"J"] <- c(0.15, 0, NA)

tm[,"W"] <- c(NA, NA, 120)

tm[, "S"] <- c(0.5, NA, NA)

tm
```

```
##                  J   W   S
## Prob Fire     0.15  NA 0.5
## Deer Density 0.00  NA  NA
## Harvest Age    NA 120  NA
```

This is our "transition" matrix so to speak - but not really – it just stores information that we can use when determining how states transition from one to another. Let's set up some initial conditions for time step t. Some of these will use the entries in the **tm** object.

```
# set state.t
state.t <- "J"

# set time since fire
tsf <- 10

# set stand age
stand.age <- 10

# set fire result
fire <- rbinom(n = 1,
               size = 1,
               prob = tm["Prob Fire", state.t])

# fill in harvest (0 or 1)
harvest <- ifelse(test = (state.t == "W") * (stand.age > tm["Harvest Age", "W"]) == TRUE,
                  yes = 1,
                  no = 0)
```

Look in your global environment for each of these variables. There are two functions here that you should note: `rbinom()` and `ifelse()`. The `ifelse()` function is very similar to Excel's IF function. The `rbinom()` function will draw a random value from a given binomial distribution. In this case, we want to draw just 1 value, and the probability of a success is stored in the **tm** matrix in the row "Prob Fire" and column that matches **state.t**.

Next, we need to determine the state of the ecosystem in the next time step. We have rules to follow, and these rules depend on the current state of the system. Below, we use a series of *if* conditions to set these rules.

```
# rules if current state is S
if (state.t == "S") {
  if (fire == 1) {state.t1 <- "J"} else {
```

```
    state.t1 <- "S"
  }
} # end of "S"


# rules if current state is W
if (state.t == "W") {
  if (harvest == 1) {state.t1 <- "S"} else {
    state.t1 <- "W"
  }
} # end of "W"


# rules if current state is J
if (state.t == "J") {
  if (all(tsf > 40, fire == 1)) {
    state.t1 <- "W"
  }
  else if (all(tsf > 80, tm['Deer Density', "J"] == 0, fire == 0) == TRUE) {
    state.t1 <- "W"
  }
  else if (all(tsf > 80, tm['Deer Density', "J"] == 1, fire == 0) == TRUE) {
    state.t1 <- "S"
  }
  else {state.t1 <- "J"}

} # end of "J"
```

There you have it. We now know the state of the system in the next time step. However, we want our simulation to encompass many time steps, not just one. And for each time step, we need to invoke the same rule-based operations that we just explored.

There are many ways one could set up this frame-based simulation for 50 time steps (or 500). Here is just one approach. There certainly may be better ways of doing this!

First, let's create a dataframe that will store our results. This dataframe (for now) will have just 1 row and 5 columns. We'll populate the first 3 columns with values.

```
# set up a dataframe to store simulation results, and populate the first row with starting values
results <- data.frame("state" = "J",
                      "tsf" = 10,
                      "stand.age" = 10,
                      "fire" = NA,
                      "harvest" = NA,
                      stringsAsFactors = F)

# look at this so far
results
```

```
##   state tsf stand.age fire harvest
## 1     J  10        10   NA      NA
```

Now, we can begin our loop. The next block of code is the full loop, so read through it carefully. It begins by populating the 'fire' and 'harvest' columns in the first time step. Then, it uses the same rules we specified before to determine the state in the *next* time step, and updates the time since fire (tsf) and stand.age columns based on whether there was a fire or harvest event.

```r
# each step is 10 years in length

# set the seed for reproducibility
set.seed(443)

# set the time steps
ts <- 50

# begin the loop
for (i in 1:ts) {

# set fire result
results[i, "fire"] <- rbinom(n = 1,
                size = 1,
                prob = tm["Prob Fire", state.t])


# fill in harvest (0 or 1)
if (results[i, "state"] == "W" && results[i, "stand.age"] > tm["Harvest Age", "W"]) {
  results[i, "harvest"] <- 1} else {
  results[i, "harvest"] <- 0}


# get the next state; these are the same rules as above but just reference values stored in the results

# ------------------------------------------------------------------------

# reset the state.t1 to be NULL
state.t1 <- NULL


# rules if current state is S
if (results[i, "state"] == "S") {
  if (results[i, "fire"] == 1) {state.t1 <- "J"} else {
    state.t1 <- "S"
  }
} # end of "S"


# rules if current state is W
if (results[i, "state"] == "W") {
  if (results[i, "harvest"] == 1) {state.t1 <- "S"} else {
    state.t1 <- "W"
  }
} # end of "W"


# rules if current state is J
if (results[i, "state"] == "J") {
  if (all(results[i, "tsf"] > 40, results[i, "fire"] == 1)) {
    state.t1 <- "W"
  }
  else if (all(results[i, "tsf"] > 80, tm['Deer Density', "J"] == 0, results[i, "fire"] == 0)) {
```

```
    state.t1 <- "W"
  }
  else if (all(results[i, "tsf"] > 80, tm['Deer Density', "J"] == 1, results[i, "fire"] == 0)) {
    state.t1 <- "S"
  }
  else {state.t1 <- "J"}

} # end of "J"

#-------------------------------------------------------------------------

# add state for t+1
results[i + 1, "state"] <- state.t1

# add tsf for t+1
if (results[i, "fire"] == 1 ) {
  results[i + 1, "tsf"] <- 0
} else {
  results[i + 1, "tsf"] <- results[i, "tsf"] + 10
}

# add stand age; resets to 0 in event of harvest or fire
if (any(results[i, "harvest"] == 1, results[i, "fire"] == 1)) {
  results[i + 1 , "stand.age"] <- 0
} else {
  results[i + 1 , "stand.age"] <- results[i , "stand.age"] + 10
}

} # end of loop i


# show the simulation results
results
```

```
##    state tsf stand.age fire harvest
## 1      J  10        10    0       0
## 2      J  20        20    0       0
## 3      J  30        30    0       0
## 4      J  40        40    0       0
## 5      J  50        50    0       0
## 6      J  60        60    0       0
## 7      J  70        70    0       0
## 8      J  80        80    0       0
## 9      J  90        90    1       0
## 10     W   0         0    1       0
## 11     W   0         0    0       0
## 12     W  10        10    0       0
## 13     W  20        20    0       0
## 14     W  30        30    0       0
## 15     W  40        40    0       0
## 16     W  50        50    0       0
## 17     W  60        60    0       0
## 18     W  70        70    0       0
## 19     W  80        80    0       0
```

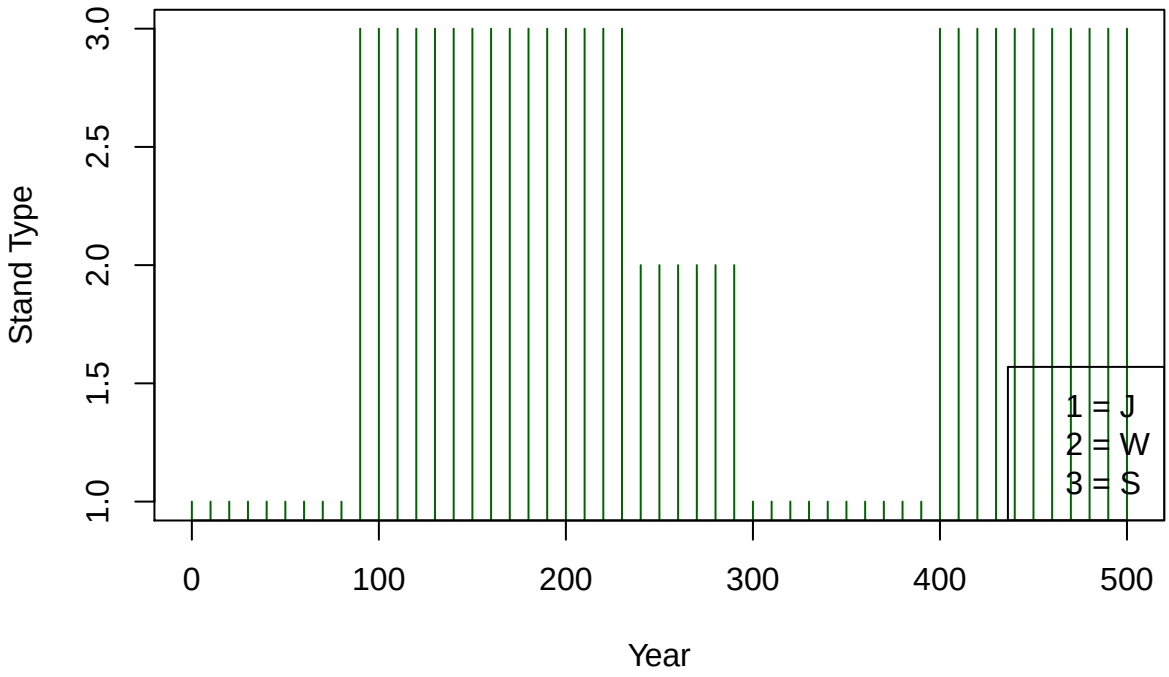
```
## 20     W  90        90    0       0
## 21     W 100       100    0       0
## 22     W 110       110    0       0
## 23     W 120       120    0       0
## 24     W 130       130    0       1
## 25     S 140         0    0       0
## 26     S 150        10    0       0
## 27     S 160        20    0       0
## 28     S 170        30    0       0
## 29     S 180        40    0       0
## 30     S 190        50    1       0
## 31     J   0         0    0       0
## 32     J  10        10    0       0
## 33     J  20        20    0       0
## 34     J  30        30    0       0
## 35     J  40        40    0       0
## 36     J  50        50    0       0
## 37     J  60        60    0       0
## 38     J  70        70    0       0
## 39     J  80        80    0       0
## 40     J  90        90    1       0
## 41     W   0         0    0       0
## 42     W  10        10    0       0
## 43     W  20        20    0       0
## 44     W  30        30    0       0
## 45     W  40        40    0       0
## 46     W  50        50    0       0
## 47     W  60        60    0       0
## 48     W  70        70    0       0
## 49     W  80        80    0       0
## 50     W  90        90    1       0
## 51     W   0         0   NA      NA
```

Tony mentions that it would be really useful to give a user feedback for each and every line of output here that tells a user why the result is what it is. We won't do that here (after all, his model was a full-fledged research program!). But what we can do is inspect every row where the state changes, and see if we can identify the reason for the switch as outlined by our rules.

```
# add our timestep column
results$year <- seq(from = 0, by = 10, length.out = ts + 1)

# change the state to a factor
results$state <- as.factor(results$state)

# plot
plot(x = results$year, y = results$state,
     xlab = "Year",
     ylab = "Stand Type",
     type = 'h',
     col = 'darkgreen')
legend("bottomright", legend = c("1 = J", "2 = W", "3 = S"))
```


Stand Type
1.0
1.5
2.0
2.5
3.0
0
100
200
300
400
500
Year
1 = J
2 = W
3 = S