Population growth and decline: Grizzly bears in Yellowstone

Andrew Park & John Drake, Odum School of Ecology

Background

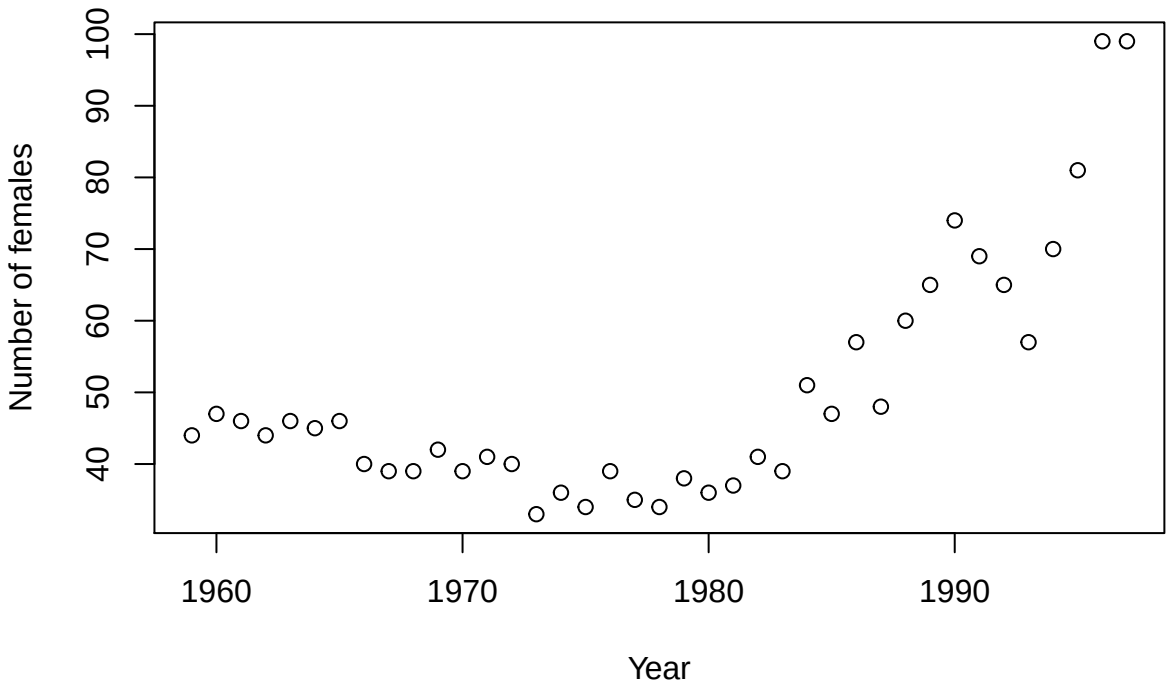
The grizzly bear *Ursus arctos horribilis* used to inhabit most of North America, but these days its continental range is restricted to the northwestern US (including Alaska) and western Canada. The Yellowstone National Park population is one of the most southern in the US and has experienced recent periods of decline and growth. Grizzly bears are a species vulnerable to local extinction. They have one of the lowest birth rates among all terrestrial mammals in the US.

Load and plot data

```
b <- read.csv("bears.csv",header=T)
```

Plot the number of female grizzly bears over time:

```
plot(b$year,b$females,xlab="Year",ylab="Number of females")
```



To help us understand the changes in the grizzly bear population we can make use of certain facts:

- The decline in the 1960s and 1970s coincided with closure of garbage dumps in the parks following a number of bear-inflicted human injuries
- At the beginning of the growth period the grizzly bear was listed as a threatened species in the lower 48 states under the Endangered Species Act (1975)
- In the 1980s a recovery plan was developed and implemented in Yellowstone

Growth phase, subsetting data

In which year do you consider that recovery began?

Subset your data to look at this phase

```
s <- subset(b,year>1979)
```

Referencing data

The recovery phase of bears is now stored in a 2-column data frame, with the same column names (years, females) as the original data frame. We can pick out various parts of a data frame, including single columns and rows, and parts of columns and rows (even just one data cell).

We can look at just the years by either refering to `s$year` or `s[,1]`

```
s$year
```

```
##  [1] 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993
## [15] 1994 1995 1996 1997
```

```
s[,1]
```

```
##  [1] 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991 1992 1993
## [15] 1994 1995 1996 1997
```

In general we can access parts of a data frame (here, called s) by using square brackets, `s[x,y]` where `x` and `y` refer to the rows (x) and columns (y), respectively. If you want to refer to all rows or columns you leave the corresponding `x` or `y` empty. Hence, `s[6,]` is all columns but just the 6th row, and `s[,]` is the full data frame. Note, in R we can refer to a range of numbers using `:`. For example

```
2:5
```

```
## [1] 2 3 4 5
```

gives all integers between 2 and 5, inclusive. If you don't want the range of numbers to be continuous, use `c()` with the desired numbers placed in the round brackets, separated by commas. For example, if instead of 2 through 5, you wanted to miss out number 4, then you'd write:

```
c(2,3,5)
```

```
## [1] 2 3 5
```

How would you reference the first three population sizes (but not years) of the recovery phase?

```
s[1:3,2]
```

```
## [1] 36 37 41
```

While these examples seem trivial, referencing data is an important part of scientific programming.

Generating a simple rule for population change, writing functions and using loops

Let's try a simple rule for how the population recovered. We'll assume that each year there were 3 more bears than last year. While this rule is simple enough to proceed by hand or with a calculator, we'll take the opportunity to write a function. R has many built in functions (you've used a lot of them already). They are useful for tasks that we might want to carry out often. We can write custom made functions. But first, what is a function? For our purposes, it's a small piece of code that when suppled with some inputs, will return

the desired output. We want to provide a current population size and have it return the next (predicted) population size (one with 3 more bears).

```
add3 <- function(x){
  return(x+3)
}
```

We've called the function `add3`. We have to be careful not to try and name a function with the same name as one of R's built-in functions. If you're not sure, first type your desired function name into the console window (bottomleft) and hit return. If you get an error message `object not found` then you're good to go, since R doesn't recognize that name yet. If instead you see some code starting with the word `function`, then this is likely a reserved name and you should try something else.

We should first convince ourselves that the function is doing what we think it should:

```
add3(5)
```

```
## [1] 8
```

Looks good. We supply the number 5, the function adds 3 to it and spits out the number 8. We can use this function to propagate forward in time; each time we call the function with the latest population size. This kind of repetition can be done using loops. We'll create a place to store our prediction, and just like the data, we'll store it in two columns: the first will contain the year and the second will contain the predicted population size (starting with the actual population size in the first year of the recovery phase). Note that we can write the code without commiting to which year marks the beginning of the recovery phase. We just take the relevant information from the `s` data frame, which you can create by subsetting to your own satisfaction.

```
p <- s[1,]
for (y in 2:dim(s)[1]){
  p[y,1] <- p[(y-1),1]+1
  p[y,2] <- add3(p[(y-1),2])
}
```

OK, we have a prediction data frame `p` which is the same size and structure as the data frame `s`. To evaluate if this prediction is any good, we're going to need a rule for measuring how accurate it is. We'll keep this rule simple: for each year, we'll look at the difference between the real data and the prediction and we'll sum up those differences over all the years. Our measure won't pay attention to the sign (+/-) of this difference. In other words, over estimating will be just as bad as underestimating. A good prediction, therefore, should have a low value of this measure - which describes discrepencies between the data and the prediction. We can proceed by writing another function. We'll supply the two data frames (`s` and `p`) and the function will return the measure we've just described.

```
my.error <- function(a,b){
  value <- 0
  for (y in 1:dim(a)[1]){
    value <- value + abs(a[y,2]-b[y,2])
  }
  return(value)
}
```

Let's see how we did

```
my.error(s,p)
```

```
## [1] 98
```

Over to you...

This model at least has the bear population growing, which it did during the recovery period. Your goal is to come up with an improved prediction, which will involve writing a function that better captures what's going on during population growth than the `add3` function was able to do. To be pursuasive, you should demonstrate that this new rule performs better than `add3`. Once you've written your function and calculated the discrepency between your prediction and the data you should also try plotting the data and the prediction on one graph. Have a look at the plots we did for the Common Terns for some starting points, and ask your instructors and class mates for their advice and opinions.