cowsgopoo: Instructions for practitioners

Natalie Cooper (nhcooper12@gmail.com)

cowsgopoo is a simple R package designed to allow easier calculation of Nitrogen depositions on fields by herds of cattle. This vignette will explain how to use the package and assumes minimal knowledge of R.

Throughout, R code will be in shaded boxes:

```
source("cowsgopoo")
```

R output will be preceded by ## and important comments will be in boxes.

Installing cowsgopoo from GitHub

> You will need to be connected to the internet to do this step. However, after you install the `devtools` and `cowsgopoo` packages you will be able to use `cowsgoopoo` offline in the future.

You can install the `cowsgopoo` package via a website called GitHub as follows. First you will need to install a clever package called `devtools` that allows us to install things from GitHub.

```
install.packages("devtools")
```

You can then install `cowsgopoo` as follows

```
devtools::install_github("cowsgopoo", username = "nhcooper123")
```

Loading the cowsgopoo package in R

You've installed `cowsgopoo` but packages don't automatically get loaded into your R session. Instead you need to tell R to load them **every time** you start a new R session and want to use functions from these packages. To load the package `cowsgopoo` into your current R session:

```
library(cowsgopoo)
```

Making a folder and finding its path

Before we talk about data we first need to know where we are going to put it. Create a new folder which we'll call "RAnalyses". You should put your data into this folder. You'll need to know what the **path** of the folder is. For example on Natalie's Windows machine, the path is:

```
C:/Users/Natalie/Desktop/RAnalyses
```

The path is really easy to find in a Windows machine, just click on the address bar of the folder and the whole path will appear.

> In Windows, paths usually include \ but R can't read these. It's easy to fix in your R code, just change any \ in the path to / or \\.

On Natalie's Mac the path is:

```
~/Desktop/RAnalyses
```

It's a bit trickier to find the path on a Mac, so Google for help if you need it. Note that the tilde ~ is a shorthand for /Users/Natalie.

Inputting your data

Your dataset needs to contain two columns: an column with some form of identifier for each animal, and a column with the dates of birth of the animals. There can be other columns in the dataset as these will be ignored by the function. However, there should only be **one row for each animal**.

Dates of birth should be in the format: dd/mm/yyyy. IDs can be in any format.

R can read files in lots of formats, including comma-delimited and tab-delimited files. Excel (and many other applications) can output files in this format (it's an option in the "Save As" dialog box under the "File" menu). Here let's assume you have a tab-delimited file called "HerdData.txt" that contains your data. You should put your data into the "RAnalyses" folder you created above.

Reading your data into R

Next we need to load the data we are going to use for the analysis. Here we have a file called "HerdData.txt" which we are going to use. Load these data as follows.

> You will need to replace MYPATH with the name of the path to the folder containing the data on your computer.

```
myherd <- read.table("MYPATH/HerdData.txt", sep = "\t", header = TRUE)
```

Note that `sep = "\t"` indicates that you have a tab-delimited file, `sep = ","` would indicate a comma-delimited csv file. You can also use `read.delim` for tab delimited files or `read.csv` for comma delimited files. `header = TRUE`, indicates that the first line of the data contains column headings.

This is a good point to note that unless you **tell** R you want to do something, it won't do it automatically. So here if you successfully entered the data, R won't give you any indication that it worked. Instead you need to specifically ask R to look at the data.

We can look at the data by typing:

```
str(myherd)
```

```
## data.frame': 4 obs. of  2 variables:
##   $ cowID: Factor w/ 4 levels "a","b","c","d": 1 2 3 4
##   $ dob  : Factor w/ 4 levels "01/11/2013","13/01/2013",..: 2 3 4 1
```

This shows the structure of the data frame (this can be a really useful command when you have a big data file). It also tells you what kind of variables R thinks you have (characters, integers, numeric, factors etc.). Some R functions need the data to be certain kinds of variables so it's useful to check this.

As you can see, the data contains the following variables: cowID and dob

Age classes

`cowsgopoo` is designed to work with a set of user defined age classes. So you can input whatever age classes you need for your forms. These must be in months (e.g. 0-3 months) but can overlap if desired (e.g. 0-3 months, 2-4 months etc.). These must be made into a list in R like this:

```
myclasses <- list(c(0, 3), c(3, 12))
```

The first number in each bracket is the start month, the second number is the end month. If you want to use classes such as 24 months and older you need to replace the end month with `Inf` which stands for infinity:

```
myclasses <- list(c(0, 24), c(24, Inf))
```

Reporting period

You will also need to know the start and end date of the reporting period. These should also be in the format dd/mm/yyyy and will be entered directly into the function.

> Note that the function will break if the end date is before any of the dates of birth of the animals in the dataset. If you want to look back at historical data remember to remove animals born after the reporting period from your dataset first.

Running a cowsgopoo analysis

These are only needed once per computer (unless you reinstall R for some reason):

```
install.packages("devtools")

devtools::install_github("cowsgopoo", username = "nhcooper123")
```

This is needed every time you start a new R session:

```
library(cowsgopoo)
```

First let's make an example dataset with IDs (cowID) a-d and random dates of birth (dob)

```
myherd <- data.frame(cowID = c("a", "b", "c", "d"),
           dob = c("13/01/2013", "20/06/2013", "27/11/2010", "01/11/2013"))
```

To use your own data you just need to make `myherd` represent your data by reading it into R as described in the above:

```
myherd <- read.table("MYPATH/HerdData.txt", sep = "\t", header = TRUE)
```

Remember to replace MYPATH with the path to your folder that contains the data.

Next we make an example list of age classes of 0-3 months, 3-12 months, 12-24 months and over 24 months. Note the use of Inf to indicate over 24 months.

```
myclasses <- list(c(0, 3), c(3, 12), c(12, 24), c(24, Inf))
```

We can now run `cowsgopoo` as follows where `data` is the name of the dataset, `ID` is the name of the column with the ID/identifier for each animal, `DOB` is the column with the date of birth for each animal, `start` is the start date of the reporting period, `end` is the end date of the reporting period, and `age.classes` is the list of age classes we made above.

```
cowsgopoo(data = myherd, ID = "cowID", DOB = "dob", start = "01/01/2013",
          end = "31/12/2013", age.classes = myclasses)
```

The output looks like this:

```
## $results
##    ID age class 1 age class 2 age class 3 age class 4
## 1  a          90         262           0           0
## 2  b          92         102           0           0
## 3  c           0           0           0         364
## 4  d          60           0           0           0

## $totals
## age class 1 age class 2 age class 3 age class 4
##         242         364           0         364

## $age.classes
##   age class start month end month
## 1         1           0         3
```

```
## 2         2           3        12
## 3         3          12        24
## 4         4          24       Inf

## $start.date
## [1] "2013-01-01"

## $end.date
## [1] "2013-12-31"

## $reporting.period.days
## Time difference of 364 days

## $number.individuals
## [1] 4
```

The function outputs a table containing the number of days each animal was in each age class (`$results`), the total number of days all animals spent in each age class (`$totals`), a table of the age classes you input (`$age.classes`), the start and end dates of the reporting period (`$start.date` and `$end.date`), the number of days in the reporting period (`$reporting.period`), and the number of individual animals in the analysis (`$number.individuals`).

We can store the results using the <- function. For example, let's re-run the analysis and call the results `myresults`.

```
myresults <- cowsgopoo(data = myherd,  ID = "cowID", DOB = "dob",
                       start = "01/01/2013", end = "31/12/2013",
                       age.classes = myclasses)
```

To look at all the results type in `myresults` and press enter. This will show the same results as above. To get just one of the results items you can use the $ codes above. For example, if you only want to look at the total days all animals spend in each class you can use:

```
myresults$totals
```

The output will look like this:

```
## age class 1 age class 2 age class 3 age class 4
##         242         364           0         364
```

You can also save your output as a file, rather than seeing the results in your R window. This works the same as above but after we store the results using the <- function, we then tell R to write a file of this data. The data will appear in your folder defined by MYPATH

> Remember to replace MYPATH with the path to your folder that contains the data.

```
write.table(file = "MYPATH/MyResultsFile.csv", myresults$results,  sep = ",",
            quote = FALSE, col.names = TRUE, row.names = FALSE)
```

This will create a .csv file that can be opened in Excel. Just right click and choose "Open With" and select Excel. Otherwise the file will open in your text editor.