Getting started with the ALFAM2 package

Sasha D. Hafner (sasha@hafnerconsulting.com, sasha.hafner@eng.au.dk)

October 2, 2021

1 Introduction

The ALFAM2 project is on ammonia volatilization (emission) from field-applied manure, and includes two main products: a database with volatilization measurements and a model for estimating volatilization. The model, which is described in detail in [1], is the focus of the ALFAM2 R package and this document. The ALFAM2 package is an add-on package for R, which is an environment for statistical computing. With the model, it is possible to predict average volatilization rate and cumulative emission over time, as affected by application method, weather, and manure characteristics. This document provides an introduction to the use of the model, and is aimed for users who are new to R. Those with some knowledge of R can skip Section 2.

1.1 Excel or R?

The ALFAM2 model is available in an Excel spreadsheet in addition to the R package that is described in this document. If you would just like to know cumulative emission for a few scenarios with constant conditions, the Excel model is a good choice. But to work with many different scenarios, or when weather changes over time (e.g., wind or rain), or if you are interested in emission dynamics and not just final cumulative emission, you should use the R package. You can use the ALFAM2 package without much knowledge of R. If you are not currently an R user, but you plan on using the ALFAM2 model extensively, it is worthwhile to learn a little bit about R and use the ALFAM2 package, instead of the less efficient Excel spreadsheet model.

2 Some basics for new R users

The information given in this section should be enough for new R users to install the package and learn enough about R to start using the model (albeit with a lack of understanding about some of the code) as described in Section 3. For a better understanding, check out the sources mentioned below.

2.1 Getting started with R

To use R it must be installed on your computer. You can download R and find installation instructions from here: https://www.r-project.org/. And while not required, it is convenient to have a good script editor. The RStudio IDE (integrated development environment) is a good (and very popular) choice. It can be downloaded from here: https://rstudio.com/products/rstudio/download/.

To use the ALFAM2 package, you will need to install the package, and then call up the function. In R, you will need to be able to install and load packages, call functions, and, ideally, create data frames and export data. For information on these tasks and more, there are many free resources online. I recommend this book I use for a course on R: https://www.researchgate.net/publication/325170649_An_Introduction_to_R_for_Beginners. CRAN provides various manuals, including a good introduction: https://cran.r-project.org/ (select "Manuals" at the lower left). RStudio also provides various materials for learning R, although the focus is skewed toward packages developed by RStudio employees: https://education.rstudio.com/learn/. Alternatively, the instructions given below may be sufficient.

2.2 Installing the ALFAM2 package

The ALFAM2 package is available from a GitHub repository: https://github.com/sashahafner/ALFAM2. Installation of packages from GitHub requires a package called devtools. You can run the code below to install devtools and ALFAM2.

First, install devtools from CRAN.

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

Then load the package,

```
library(devtools)
```

and install the ALFAM2 package from GitHub.[1]

```
install_github("sashahafner/ALFAM2", build_vignettes = TRUE)
```

Alternatively, to avoid loading devtools, use this syntax.

```
devtools::install_github("sashahafner/ALFAM2", build_vignettes = TRUE)
```

[1] Some additional notes. You need the `build_vignettes = TRUE` bit to install this vignette that you are now reading (and any others that may be added in the future). To get the latest version of the package (possible bugs, incomplete testing, and all), add the argument `ref = "dev"`.

These steps only need to be carried out once.

Finally, every time you open R to use the ALFAM2 package, it must be loaded.

```
library(ALFAM2)
```

You can open this vignette with the following call.

```
vignette("ALFAM2-start")
```

3 The `ALFAM2mod()` function

The ALFAM2 package includes a single function that is an implementation of the ALFAM2 model: `ALFAM2mod()` After an explanation of the function, its use is shown in a few examples.

3.1 Overview of the function

The `ALFAM2()` function can be used for predicting average volatilization rate and cumulative emission over time. The function has several arguments, as shown below.

```
args(ALFAM2mod)

## function (dat, pars = ALFAM2::ALFAM2pars02, app.name = "TAN.app",
##     time.name = "ct", time.incorp = NULL, group = NULL, center = TRUE,
##     cmns = c(app.rate = 40, man.dm = 6, man.tan = 1.2, man.ph = 7.5,
##         air.temp = 13, wind.2m = 2.7, crop.z = 10), check.NA = TRUE,
##     pass.col = NULL, incorp.names = c("incorp", "deep", "shallow"),
##     add.incorp.rows = FALSE, warn = TRUE, parallel = FALSE, n.cpus = 1,
##     ...)
## NULL
```

You can find more details on the arguments (as well as examples) in the help file. As with any R function, you can open the file with `?`:

```
?ALFAM2mod
```

But the most important arguments are described here. Most arguments have default values, and the only one that is required to make predictions is the `dat` argument, which is a data frame containing some input data, i.e., values of predictor variables over time. The `dat` data frame can contain any number of rows (each representing a time interval), but must contain a column with cumulative time in hours, and the name of this column is indicated with `time.name`. Typically the data frame will have predictor variables as well, for example, manure

Table 1: Default predictor variables that can be used with `ALFAM2mod()`, as given in the `ALFAM2pars02` or `ALFAM2pars01` objects.

Variable name	Description	Units	Notes
`int`	Intercept terms	None	
`app.mthd.os`	Open slot application	None (logical)	Binary variable
`app.mthd.cs`	Closed slot application	None (logical)	Binary variable
`app.mthd.bc`	Broadcast application	None (logical)	Binary variable
`app.mthd.ts`	Trailing shoe application	None (logical)	Binary variable
`app.rate`	Manure application rate	t/ha	
`app.rate.ni`	Manure app. rate (excluding (no) injection)	t/ha	
`man.dm`	Manure dry matter	%	
`man.ph`	Manure pH	pH units	For acidification
`man.source.pig`	Pig manure	None (logical)	Binary variable
`incorp.deep`	Deep incorporation	None (logical)	Binary variable
`incorp.shallow`	Shallow incorporation	None (logical)	Binary variable
`air.temp`	Air tempreature	°C	
`wind.2m`	Wind speed (at 2 m)	m/s	
`rain.rate`	Rainfall rate	mm/h	
`rain.cum`	Cumulative rain	mm	
`cereal.hght`	Cereal height	cm	

dry matter, application method, air temperature, or wind speed. The name of the predictor columns are used to link predictor variables to model parameters, which are set by the `pars` argument. Usually the default values, based on the measurements in the ALFAM2 database, should be used. Predictor variables and their default names are given in Table 1 below.

Default model parameters and numeric values in the `ALFAM2pars02` object ("Set 2") should generally be used. For information on how these values were calculated, see the report on calculation of Danish emission factors [2]. (An earlier version ("Set 1") are available in `ALFAM2pars01`. Derivation of these is described in the 2019 paper [1].) Comparing the contents of `ALFAM2pars02` to the variable names given in Table 1, you can see an additional letter and number added to the end of the parameters.

```
ALFAM2pars02
```

```
##            int.f0    app.mthd.os.f0    app.rate.ni.f0
##       -0.60568338       -1.74351499       -0.01114900
##         man.dm.f0 man.source.pig.f0    app.mthd.cs.f0
##        0.39967070       -0.59202858       -7.63373787
##            int.r1    app.mthd.bc.r1         man.dm.r1
##       -0.93921516        0.79352480       -0.13988189
##       air.temp.r1        wind.2m.r1    app.mthd.ts.r1
##        0.07354268        0.15026720       -0.45907135
## ts.cereal.hght.r1         man.ph.r1            int.r2
```

```
##       -0.24471238        0.66500000       -1.79918546
##      rain.rate.r2            int.r3    app.mthd.bc.r3
##        0.39402156       -3.22841225        0.56153956
##    app.mthd.cs.r3         man.ph.r3 incorp.shallow.f4
##       -0.66647417        0.23800000       -0.96496655
## incorp.shallow.r3    incorp.deep.f4    incorp.deep.r3
##       -0.58052689       -3.69494954       -1.26569562
```

These numbers indicate a primary parameter. So, for example, the (secondary) parameter `wind.2m.r1`, which is 0.15 s/m by default, is used in the calculation of the primary parameter r_1. The most important message here is a simple one: names for predictor variables can be taken from the names given in the default `pars` argument value, but be sure to omit the last three characters (a ".", a number, and a letter).

By design, any time a predictor variable is omitted when `ALFAM2mod()` is called, the reference level or value is assumed for that variable.[2] The scenario with reference levels for all predictors is the default scenario, and is the one given in the first row of Table 4 in [1]. Predictor values for the default scenario can be found in the (cmns) argument (for centering means, see help file). The default application method is trailing hose. The `cmns` argument is used for centering predictor variables, and this approach facilities the behavior described above.

3.2 Cumulative emission for a single scenario

In this example, let's assume we are interested in manure application by broadcast when manure had 8% dry matter (DM), total TAN application is 50 kg/ha, wind is 3 m/s, and air temperature is 20°C.

First we need to create a data frame with the input data.

```
dat1 <- data.frame(ctime = 72, TAN.app = 50, man.dm = 8,
                   air.temp = 20, wind.2m = 3,
                   app.mthd.bc = TRUE)
dat1

##   ctime TAN.app man.dm air.temp wind.2m app.mthd.bc
## 1    72      50      8       20       3        TRUE
```

Our predictor variable values are in the columns `man.dm` and the following ones. The names for the predictor variables must match those names used in the model parameters, which can be seen by checking the parameter object contents (see just above).

Time, in hours after application, is given in the column named `ctime` here, for cumulative time (although any name can be used).

[2] One exception is `app.rate.ni`.

And now we can call the model function, using default values for most other arguments. We can predict cumulative emission after 3 days (72 hours) with the following call.

```
pred1 <- ALFAM2mod(dat1, app.name = 'TAN.app', time.name = 'ctime')

## Default parameters (Set 2) are being used.
## Warning in ALFAM2mod(dat1, app.name = "TAN.app", time.name = "ctime"):
Running with 10 parameters.  Dropped 14 with no match.
## These secondary parameters have been dropped:
##   app.mthd.os.f0
##   app.rate.ni.f0
##   man.source.pig.f0
##   app.mthd.cs.f0
##   app.mthd.ts.r1
##   ts.cereal.hght.r1
##   man.ph.r1
##   rain.rate.r2
##   app.mthd.cs.r3
##   man.ph.r3
##   incorp.shallow.f4
##   incorp.shallow.r3
##   incorp.deep.f4
##   incorp.deep.r3
##
## These secondary parameters are being used:
##   int.f0
##   man.dm.f0
##   int.r1
##   app.mthd.bc.r1
##   man.dm.r1
##   air.temp.r1
##   wind.2m.r1
##   int.r2
##   int.r3
##   app.mthd.bc.r3
```

The warning message just tells us that the call included some parameters with no associated predictor variables in our data frame given in the `dat` argument. This is discussed more below. We will turn off the warning in the examples below.

Let's look at the predictions.

```
pred1

##   ct dt        f0       r1         r2          r3 f4
```

```
## 1 72 72 0.5482638 1.362777 0.01587869 0.002153413  1
##              f        s         j        e    e.int        er
## 1 2.130583e-42 19.61361 0.4220332 30.38639 30.38639 0.6077278
```

The most interesting columns here are called `e`, which has cumulative emission in the same units as TAN application, and `er`, which has relative cumulative emission, as a fraction of applied TAN. So in this example, 48% of applied TAN is predicted to be lost by volatilization.

The warning message above is related to an important point: Any excluded predictors are effectively assumed to be at their reference levels.

3.3 Adding incorporation

To include incorporation, we need to add a couple columns to our data frame. First let's make a new data frame for the example.

```
dat2 <- dat1
```

And add the two new columns. Here we are specifying that deep incorporation happens after 0.5 hours.

```
dat2$incorp.deep <- TRUE
dat2$t.incorp <- 0.5
dat2

##   ctime TAN.app man.dm air.temp wind.2m app.mthd.bc incorp.deep
## 1    72      50      8       20       3        TRUE        TRUE
##   t.incorp
## 1      0.5
```

```
pred2 <- ALFAM2mod(dat2, app.name = "TAN.app", time.name = "ctime",
                   time.incorp = "t.incorp", warn = FALSE)
pred2

##   ct dt        f0       r1         r2          r3         f4
## 1 72 72 0.5482638 1.362777 0.01587869 0.000116797 0.02424622
##              f        s         j        e    e.int        er
## 1 5.165858e-44 35.84833 0.1965509 14.15167 14.15167 0.2830333
```

Here we see that with incorporation, emission drops to 28% of applied TAN. Shallow incorporation has less of an effect.

```
dat3 <- dat1
dat3$incorp.shallow <- TRUE
dat3$t.incorp <- 0.5
dat3
```

```
##   ctime TAN.app man.dm air.temp wind.2m app.mthd.bc
## 1    72      50      8       20       3        TRUE
##   incorp.shallow t.incorp
## 1           TRUE      0.5
```

```
pred3 <- ALFAM2mod(dat3, app.name = "TAN.app", time.name = "ctime",
                   time.incorp = "t.incorp", warn = FALSE)
pred3
```

```
##   ct dt        f0       r1         r2           r3        f4
## 1 72 72 0.5482638 1.362777 0.01587869 0.0005657185 0.2758849
##              f        s         j        e    e.int        er
## 1 5.877957e-43 31.42921 0.2579277 18.57079 18.57079 0.3714159
```

3.4 Multiple scenarios in a single call

A single function call can be used for multiple scenarios. For example, perhaps we would like to compare 5 different incorporation times. First let's create a new data frame that contains this information. We will need to add a new column with a grouping variable also, to let `ALFAM2mod()` know that each row represents a different scenario.

```
dat4 <- data.frame(scenario = 1:5, ctime = 72, TAN.app = 50,
                   man.dm = 8, air.temp = 20, wind.2m = 4,
                   app.mthd.bc = TRUE,
                   incorp.deep = TRUE,
                   t.incorp = c(0.1, 1, 6, 24, Inf))
dat4
```

```
##   scenario ctime TAN.app man.dm air.temp wind.2m app.mthd.bc
## 1        1    72      50      8       20       4        TRUE
## 2        2    72      50      8       20       4        TRUE
## 3        3    72      50      8       20       4        TRUE
## 4        4    72      50      8       20       4        TRUE
## 5        5    72      50      8       20       4        TRUE
##   incorp.deep t.incorp
## 1        TRUE      0.1
## 2        TRUE      1.0
## 3        TRUE      6.0
## 4        TRUE     24.0
## 5        TRUE      Inf
```

It may seem strange to have a `scenario` column–isn't it clear that each row is a different scenario? The answer is no, not when there are multiple time intervals per scenario, for example when one is interested in volatilization rates

over time and how they change. Note that there is no incorporation for scenario 5. We could also specify this behavior with `t.incorp = NA`.[3]

Let's run the model for these 5 scenarios.

```
pred4 <- ALFAM2mod(dat4, app.name = "TAN.app", time.name = "ctime",
                   time.incorp = "t.incorp", group = "scenario", warn = FALSE)
pred4
```

```
##   scenario ct dt        f0       r1         r2          r3
## 1        1 72 72 0.5482638 1.926158 0.01587869 0.000116797
## 2        2 72 72 0.5482638 1.926158 0.01587869 0.000116797
## 3        3 72 72 0.5482638 1.926158 0.01587869 0.000116797
## 4        4 72 72 0.5482638 1.926158 0.01587869 0.000116797
## 5        5 72 72 0.5482638 1.926158 0.01587869 0.002153413
##           f4            f        s          j         e
## 1 0.02424622 1.249271e-61 44.27971 0.07944843  5.720287
## 2 0.02424622 1.249271e-61 26.34738 0.32850861 23.652620
## 3 0.02424622 1.249271e-61 22.34567 0.38408785 27.654325
## 4 0.02424622 1.249271e-61 21.54112 0.39526229 28.458885
## 5 1.00000000 5.152434e-60 19.53496 0.42312550 30.465036
##       e.int        er
## 1  5.720287 0.1144057
## 2 23.652620 0.4730524
## 3 27.654325 0.5530865
## 4 28.458885 0.5691777
## 5 30.465036 0.6093007
```

We can see that predicted emission increases substantially as incorporation time goes up. And incorporation after 24, or really even 6, hours is not much better than no incorporation!

Scenarios could differ in any way. Below, both temperature and application method vary. For scenario 2, application method is not explicitly specified, which means it is the default–trailing hose.

```
dat5 <- data.frame(scenario = 1:3, ctime = 72, TAN.app = 50,
                   man.dm = 8, air.temp = 10 + (0:2 * 10),
                   wind.2m = 3,
                   app.mthd.bc = c(TRUE, FALSE, FALSE),
                   app.mthd.os = c(FALSE, FALSE, TRUE)
                   )
dat5
```

```
##   scenario ctime TAN.app man.dm air.temp wind.2m app.mthd.bc
## 1        1    72      50      8       10       3        TRUE
```

[3] Rhis would provide identical results, but for package v0.3.2 and prior, this approach did not work correctly unless `incorp.deep` was set to FALSE.

```
## 2        2    72      50      8       20       3       FALSE
## 3        3    72      50      8       30       3       FALSE
##   app.mthd.os
## 1       FALSE
## 2       FALSE
## 3        TRUE
```

```
pred5 <- ALFAM2mod(dat5, app.name = "TAN.app", time.name = "ctime",
                   group = "scenario", warn = FALSE)
pred5
```

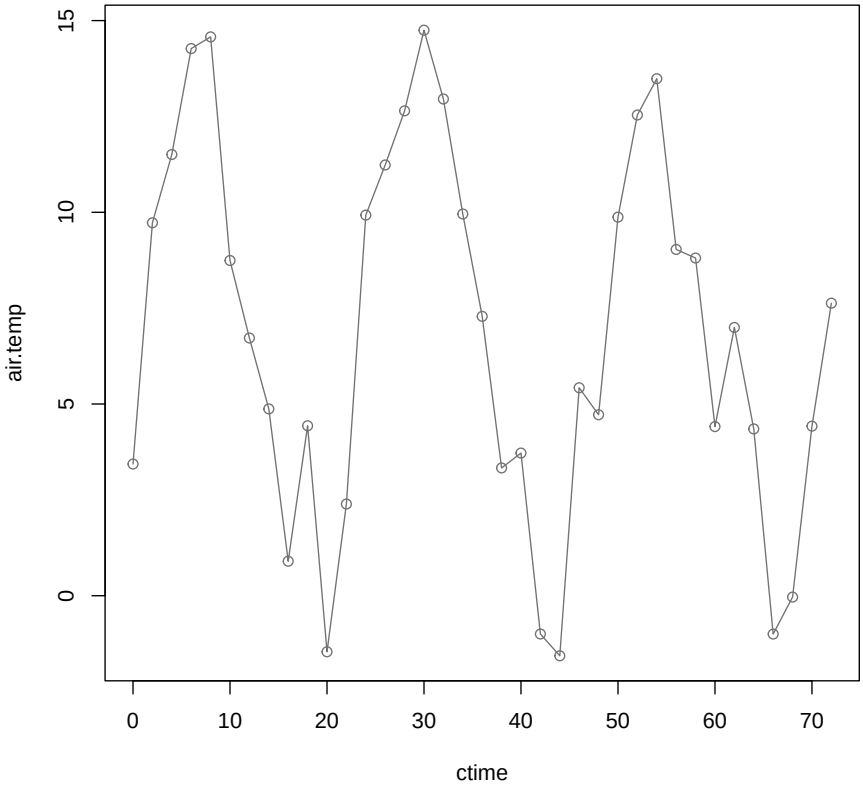
```
##   scenario ct dt        f0        r1         r2           r3 f4
## 1        1 72 72 0.5482638 0.2506098 0.01587869 0.0021534129  1
## 2        2 72 72 0.5482638 0.2192300 0.01587869 0.0005910004  1
## 3        3 72 72 0.1751069 1.1921385 0.01587869 0.0005910004  1
##              f        s         j        e    e.int        er
## 1 1.273732e-07 20.75301 0.4062081 29.24699 29.24699 0.5849397
## 2 1.219828e-06 23.42462 0.3691024 26.57538 26.57538 0.5315075
## 3 1.474136e-37 39.63677 0.1439338 10.36323 10.36323 0.2072646
```

3.5 Volatilization dynamics

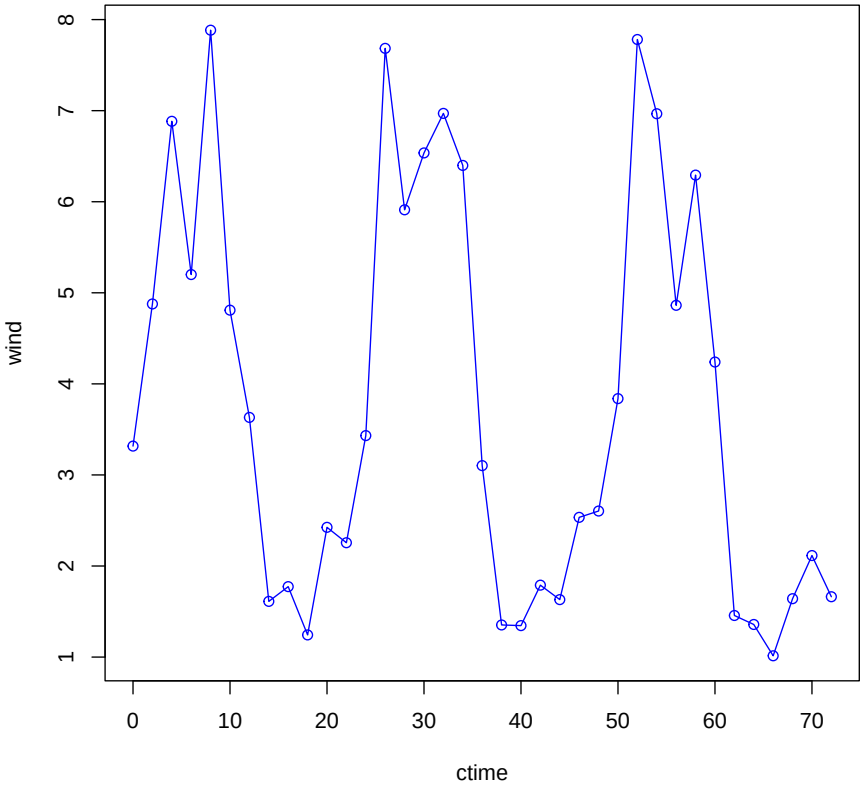
All the calls above returned results for a single time per scenario. The function also predicts dynamics. If your interest is final cumulative emission, it is not necessary to look at dynamics. The model uses an analytical expression in each interval, and so results are independent of time step size, as long as conditions (e.g., wind or air temperature) are constant. However, if detailed temporal weather data are available, running the model with with multiple intervals will generally improve the accuracy of prediction of final cumulative emission. Where needed for incorporation calculations, the function will add an interval (row), but these rows are excluded from the results by default (set the `add.incorp.rows` argument to `TRUE` to show them).

Let's assume we have some high resolution measurements of weather conditions. We'll create some data to represent this below.

```
set.seed(1201)
dat6 <- data.frame(ctime = 0:36*2, TAN.app = 100, man.dm = 8,
                   air.temp = 7 + 7*sin(0:36*2 * 2*pi/24) + rnorm(37, 0, 2),
                   wind = 10^(0.5 + 0.4*sin(0:36*2 * 2*pi/24) +
                              rnorm(37, 0, 0.12)),
                   app.mthd.bc = TRUE)
plot(air.temp ~ ctime, data = dat6, type = 'o', col = 'gray45')
```




```
plot(wind ~ ctime, data = dat6, type = 'o', col = 'blue')
```



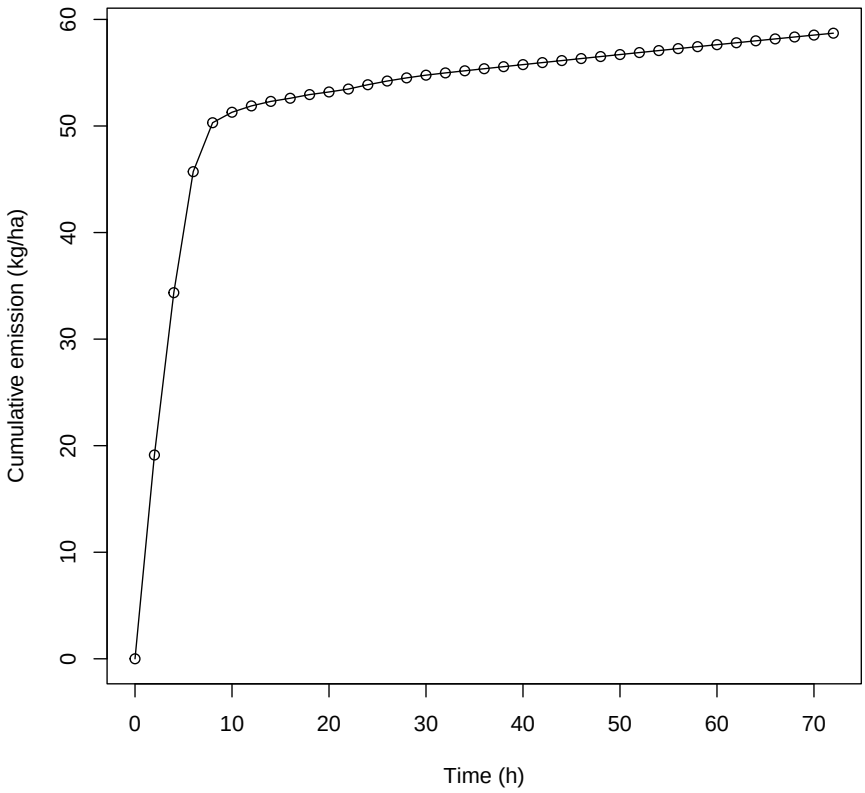
Predictions are made as above. By default, multiple rows in `dat` are assumed to all below to the same scenario (same plot, same emission trial).[4]

```
pred6 <- ALFAM2mod(dat6, app.name = 'TAN.app', time.name = 'ctime',
                   warn = FALSE)
```

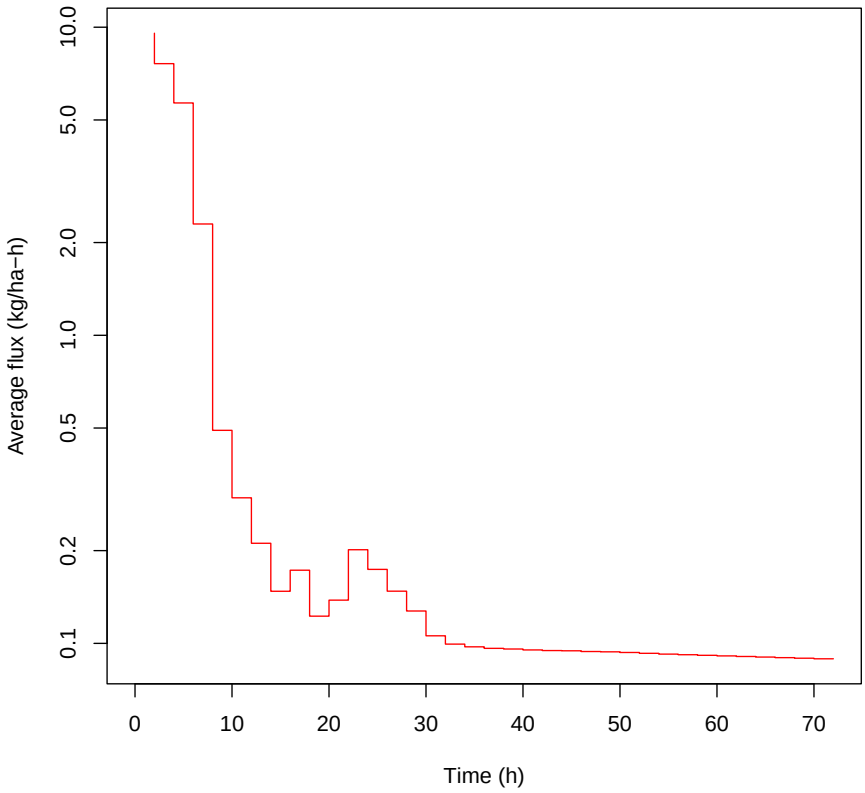
Cumulative emission and average interval flux are plotted below.

```
plot(e ~ ct, data = pred6, type = 'o', xlab = 'Time (h)',
     ylab = 'Cumulative emission (kg/ha)')
```

[4]This is the reason the `group` argument was needed above.




```
plot(j ~ ct, data = pred6, type = 'S', log = 'y', col = 'red',
     xlab = 'Time (h)', ylab = 'Average flux (kg/ha-h)')
```

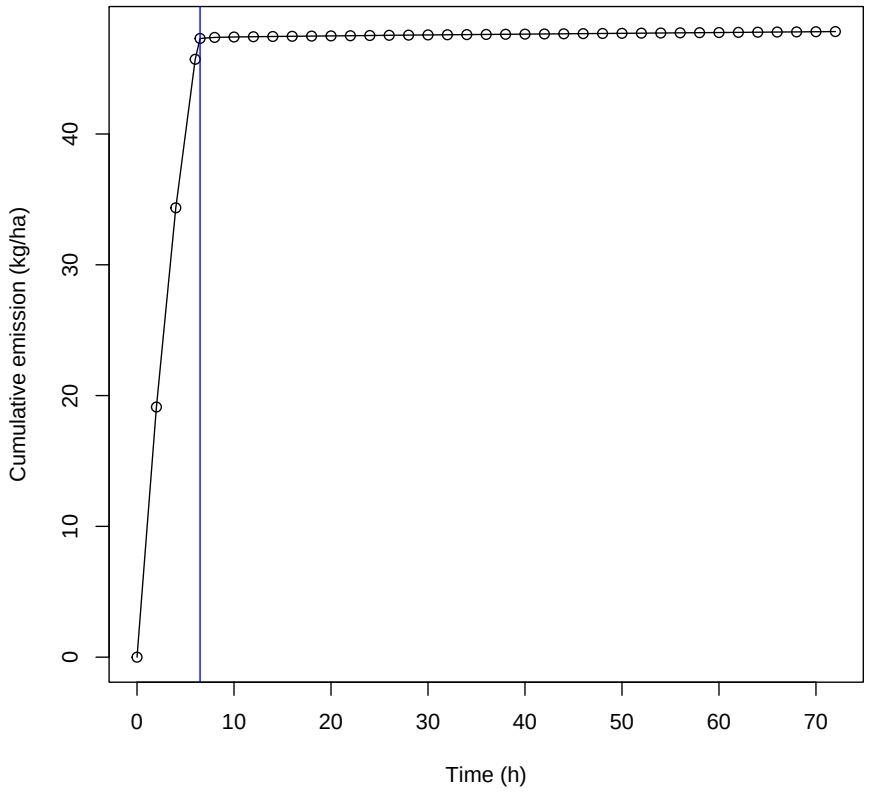


Dynamics in the case of incorporation may be interesting. The additional interval required internally because incorporation does not line up exactly with an interval in the input data frame can be returned in the output by using the `add.incorp.rows` argument. But recall that this has no effect on cumulative emission.

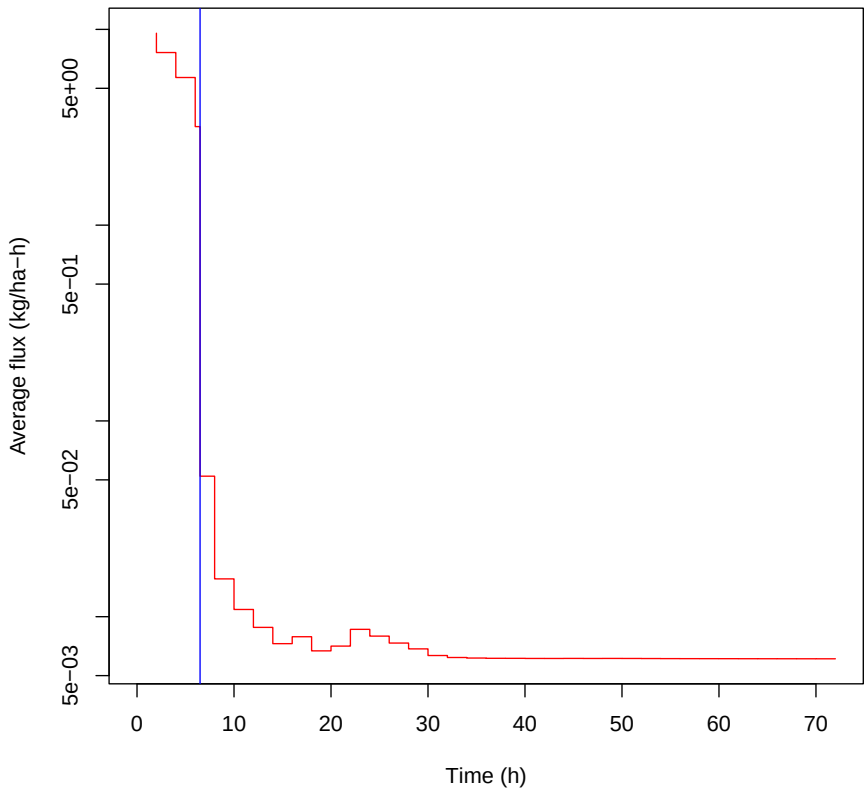
```
dat7 <- dat6
dat7$incorp.deep <- TRUE
dat7$t.incorp <- 6.5
```

```
pred7 <- ALFAM2mod(dat7, app.name = 'TAN.app', time.name = 'ctime',
                   time.incorp = 't.incorp', warn = FALSE, add.incorp.rows = TRUE)
```

```
plot(e ~ ct, data = pred7, type = 'o', xlab = 'Time (h)',
     ylab = 'Cumulative emission (kg/ha)')
abline(v = 6.5, col = 'blue')
```




```
plot(j ~ ct, data = pred7, type = 'S', log = 'y', col = 'red',
     xlab = 'Time (h)', ylab = 'Average flux (kg/ha-h)')
abline(v = 6.5, col = 'blue')
```



The drop in flux immediately after incorportaion is particularly clear in the flux (second) plot.

3.6 Data import and export

Any of the results shown above can be exported as with any data frame in R. The simplest function for this is `write.csv()`. The following call will create a comma delimited text file that can be opened with spreadsheet or text editor programs.

```
write.csv(pred6, 'pred6.csv', row.names = FALSE)
```

Alternatives include `write.csv2`, `write.table`, and any of the various functions in add-on packages for writing to Excel files.

Except for simple scenarios, it is not very efficient to create a data frame for entering predictor variable values. A more typical approach will be to read data into R from a file, especially when using the model in association with emission

measurements. The simplest approach here is to use the `read.csv()` function or some of the related functions. Alternatively, data can be easily read from Excel files with the `read_xls` and related functions in the readxl package. See the book mentioned above for details.

3.7 More with the ALFAM2 model

Dynamic predictions can be combined with multiple scenarios, although this is not shown here. In fact, the only difference between these dynamic calls and the simple examples given above is the number of measurement intervals.

All the calls in this document used the default parameter values. However, it is possible to use completely different parameter values for the same parameters, or even different secondary parameters. These are set with the `pars` argument.

For large datasets, or parameter estimation (where the `ALFAM2mod()` function is called many times), parallel processing will be helpful. See the `parallel` argument.

Acknowledgements

Christoph Haeni, Johanna Maria Pedersen, and Anders Peter Adamsen provided helpful suggestions and identified errors in earlier drafts of this vignette. Thank you!

References

[1] Hafner, S.D., Pacholski, A., Bittman, S., Carozzi, M., Chantigny, M., Génermont, S., Häni, C., Hansen, M.N., Huijsmans, J., Kupper, T., Misselbrook, T., Neftel, A., Nyord, T., Sommer, S.G. A flexible semi-empirical model for estimating ammonia volatilization from field-applied slurry. Atmospheric Environment. *Atmospheric Environment*, 199:474-484, 2018. https://doi.org/10.1016/j.atmosenv.2018.11.034

[2] Hafner, S.D., Nyord, T., Sommer, S.G., Adamsen, A.P.S. 2021. Estimation of Danish emission factors for ammonia from field-applied liquid manure for 1980 to 2019. Danish Centre for Food and Agriculture, Aarhus University, Aarhus, Denmark. Report no. 2021-0251862. https://pure.au.dk/portal/files/223538048/EFreport23092021.pdf