NBDA USER GUIDE V1.2

Written and maintained by Will Hoppitt

This document contains a guide to using the functions contained in the file "NBDA code 1.2.R". Further details of these functions can be found in Hoppitt, Boogert & Laland (2008) The guide will be updated as the accompanying functions are updated.

Last update: 7/1/10
Minor edits on 24/2/11

Contents

Introduction

Here we provide a tutorial on how to run an NBDA, using both OADA and TADA methods, using the NBDA functions provided in the file "NBDA code". In Part F we provide a glossary of all the functions available, with details of the different options for each. R is available for free from http://www.r-project.org/ , to run the NBDA functions, one will also require the packages "survival" and "combinat". On a Mac, this is done by clicking Packages & Data> Package Installer, then click "Get List", select the required packages and click "Install Selected". On a PC, this is done by clicking Packages>Install Package(s)… selecting a local "mirror" and then double clicking on the required packages.

Once R is installed, the NBDA functions can be loaded by cutting and pasting all the text contained in "NBDA code" into the command window, and pressing return. The functions defined in the file are then available for use.

In the following tutorial, we demonstrate how to run the analyses reported in the main text of the diffusion experiments of Boogert et al. (2008) on starlings. For clarity, the input to the R console is shown in blue, with the command prompt represented as >, and the output from R is shown in red.

OADA

Inputting OADA data

The first stage is to input the data for each diffusion. Here, we do this for task 1 for group 1. First input the association matrix (or social network) for the group:

```
> amg1<-matrix(data=c(0.0, 8.0, 6.5, 5.0, 1.0,
8.0, 0.0, 7.5, 2.0, 5.0,
6.5, 7.5, 0.0, 2.0, 3.5,
5.0, 2.0, 2.0, 0.0, 0.5,
1.0, 5.0, 3.5, 0.5, 0.0), nrow=5)
>amg1
     [,1] [,2] [,3] [,4] [,5]
[1,]  0.0  8.0  6.5  5.0  1.0
[2,]  8.0  0.0  7.5  2.0  5.0
[3,]  6.5  7.5  0.0  2.0  3.5
[4,]  5.0  2.0  2.0  0.0  0.5
[5,]  1.0  5.0  3.5  0.5  0.0
```

Next input the order of acquisition data as a vector, where the numbers indicate the row/column representing that individual in the association matrix:

```
> oag1t1<-c(4,1,2)
```

Next we create a matrix containing the individual-level variables for group 1:

```
> agonrank<-c(2.24, 6.55, 0.75, 3.20, 0.35)
> comprank<-c(3,4,2,1,5)
> novEnv<-c(-0.44487, -3.13171, 0.72296, 1.12335, 1.73027)
> objNeo<-c( 2.09151, -1.99598, -0.40689, -0.23706, 0.54842)
> AsocLearn<-c(-0.56429, -0.14960, -0.33799, -0.69712, 1.74900)
> asoc<-cbind(agonrank,comprank,novEnv,objNeo,AsocLearn)
> asoc
     agonrank comprank   novEnv   objNeo AsocLearn
```

```
[1,]     2.24         3 -0.44487  2.09151  -0.56429
[2,]     6.55         4 -3.13171 -1.99598  -0.14960
[3,]     0.75         2  0.72296 -0.40689  -0.33799
[4,]     3.20         1  1.12335 -0.23706  -0.69712
[5,]     0.35         5  1.73027  0.54842   1.74900
```

Now we create an oaData object containing the data necessary to run the OADA:

```
tempData<-oaData(assMatrix=amg1, asoc=asoc, orderAcq=oag1t1, groupid="1",
taskid="1")
```

It is important to specify a unique identifier for each task and group.

To save entering the data for all diffusions, we have provided the oaData objects for each diffusion in the file "starlingSolve". To load these objects, move the file into your working directory for R and enter:

```
> load("starlingSolve1.2")
```

The oaData objects are named as "nG1T1", "nG1T2", etc. To change your working directory on a Mac, click Misc> Change Working Directory… and on a PC click File>Change dir…

Adding seeded demonstrators

If the diffusion was "seeded" by introducing one or more informed individuals into the group, this should be accounted for using the argument demons, which takes a vector of length equal to the group size giving 1 for a demonstrator and a 0 for other individuals, in the same order as individuals appear in the association matrix, for example, if individual 3 was a demonstrator:

```
tempData<-oaData(assMatrix=amg1, asoc=asoc, orderAcq=oag1t1, groupid="1",
taskid="1", demons=c(0,0,1,0,0))
```

Adding transmission weights

In its initial form NBDA assumes that all individuals transmit the trait at the same rate per given unit of social network strength. However, it might be that different individuals perform the trait at a different rate once they have acquired it. We could fit a model that weights the rate of transmission from each individual using the weights argument, which takes a vector of length equal to the group size. For example, if we think individual 1 performed the trait ten times as often as other individuals:

```
tempData<-oaData(assMatrix=amg1, asoc=asoc, orderAcq=oag1t1, groupid="1",
taskid="1", weights=c(10,1,1,1,1))
```

This model can be compared against an un-weighted model using AIC criteria. If the weighted model is better than the un-weighted model, I suggest that this can be taken as evidence that performance of the trait results in social transmission.

Fitting an additive OADA model

We can fit an OADA model to a single diffusion as follows:

```
> tempModel1<-addFit(oadata=nG1T1)
```

However, here we want to fit the model to multiple diffusions, which we do by providing the oadata argument with a character vector containing the names of all the oaData objects:

```
> addModel1<-
addFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2","nG
2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"))
```

We can then examine a summary of the model:

```
> summary(addModel1)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                      Estimate    Bounded
Social Transmission 0.06946105 0.06494958


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC     AICc      LR       p
With Social Transmission     1  70.025 142.050 142.118   0.438   0.508
Without Social Transmission  0  70.244 140.488 140.488
```

The third line tells us that the unbounded parameterisation (s') was used to fit the model. The coefficients section tells us what the maximum likelihood estimator is for s' and when transformed to the bounded parameterisation (s). The section "Likelihood Ratio Test for Social Transmission" compares a model containing social transmission, with a model without (s=s'=0). It also gives the degrees of freedom, –log-likelihood, AIC and AIC_c for each.

We can now add in individual-level variables, by providing a numerical vector to the argument "asocialVar", specifying the columns in the matrix containing these variables. For example, to get the model containing object neophobia and asocial learning:

```
> addModel2<-
addFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2","nG
2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),
asocialVar=c(4,5))

> summary(addModel2)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                        Estimate Bounded
Social transmission 1  0.0000000       0
objNeo                -0.2714104      NA
AsocLearn             -0.3287048      NA
```

```
Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc        LR       p
With Social Transmission     3  67.092 142.185 140.606 1.850e-09   1.000
Without Social Transmission  2  67.092 138.185 138.392
```

The coefficients table now also contains the maximum likelihood estimators of the coefficients for each of the individual-level variables. The bottom table now shows an LRT comparing a model with both social transmission and the individual-level variables with the null model containing just the individual-level variables.

It is useful to check that the optimisation routines have converged, this can be done by examining:

```
> addModel2@optimisation
```

For details of how to interpret this output, type: `> ?nlminb`

To fit separate parameters for s for different diffusions, we use the sParam argument. For example, to fit separate s parameters for each group, we use:

```
>  addModel3<-
addFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2","nG
2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),bound
ed=F,sParam=c(1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3),asocialVar=c(4,5))
```

The vector provided to the sParam argument determines which parameter is fit for each diffusion. For the first 5 diffusions (group 1), parameter 1 is fit, for the next 6 diffusions (group 2), parameter 2 is fit, and for the last 5 (group 3) parameter 3 is fit. The parameter estimates are given in the summary:

```
> summary(addModel3)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                          Estimate    Bounded
Social transmission 1  2.159507e+07 0.99999995
Social transmission 2  6.919808e-02 0.06471961
Social transmission 3  0.000000e+00 0.00000000
objNeo                -4.115029e-01         NA
AsocLearn             -3.561302e-01         NA


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc      LR       p
With Social Transmission     5  64.271 140.542 139.633   5.642   0.130
```

```
Without Social Transmission  2  67.092 138.185 138.392
```

We can now fit a model in which s for groups 1 and 2 is constrained to be the same, so we can test for a significant difference between them:

```
> addModel4<-
addFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2","nG
2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),bound
ed=F,sParam=c(1,1,1,1,1,1,1,1,1,1,2,2,2,2,2),asocialVar=c(4,5))
> summary(addModel4)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                          Estimate Bounded
Social transmission 1  1.846320e+07       1
Social transmission 2  0.000000e+00       0
objNeo                -4.194009e-01      NA
AsocLearn             -4.545026e-01      NA


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                           Df  LogLik     AIC     AICc      LR       p
With Social Transmission    4  64.596 139.192 137.907   4.992   0.082
Without Social Transmission 2  67.092 138.185 138.392
```

We can test for a significant difference using an LRT:

```
> 1- pchisq(2*(64.596-64.271),1)
[1] 0.4201127
```

Giving p=0.420. We can also constrain parameters to zero, allowing us to test whether they are significantly different to zero using an LRT. We do this by putting zeroes in the vector provided to the sParam argument, for those diffusions for which we wish to constrain the parameter. For example, to fit the final model given in the main text, with no social transmission in groups 2 and 3:

```
> addModel5<-
addFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2","nG
2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),bound
ed=F,sParam=c(1,1,1,1,1,0,0,0,0,0,0,0,0,0,0),asocialVar=c(4,5))
> summary(addModel5)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                          Estimate Bounded
Social transmission 1  1.951408e+07       1
objNeo                -3.481498e-01      NA
AsocLearn             -3.504357e-01      NA
```

```
Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc      LR       p
With Social Transmission     3  64.329 134.658 135.079   5.527   0.019
Without Social Transmission  2  67.092 138.185 138.392
```

Fitting a multiplicative OADA model

The procedure for fitting a multiplicative OADA is slightly different, due to the use of a Cox Proportional Hazards model which enables one to incorporate random effects (see Part A). To fit a model without individual-level variables:

```
> multiModel1<-
multiCoxFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2
","nG2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"))
```

```
Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                     Estimate    Bounded
Social Transmission 0.06946105 0.06494958


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine
multiplicatively

                            Df  LogLik     AIC    AICc      LR       p
With Social Transmission     1  70.025 142.050 142.118   0.438   0.508
Without Social Transmission  0  70.244 140.488 140.488
```

Note that this is the same output as for the addFit above, as it should always be when there are no individual-level variables.

To add individual-level variables, we need to specify a formula of the form:
formula= ~.+ VARIABLE1 + VARIABLE2 +…
with the variables named as they are in the oaData objects. For example, to fit the model with object neophobia and asocial learning:

```
> multiModel2<-
multiCoxFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2
","nG2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),
formula=~.+objNeo+AsocLearn)
> summary(multiModel2)


Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                      Estimate    Bounded        se         z          p
Social Transmission  0.05889902 0.05562289        NA        NA         NA
objNeo              -0.29095829         NA 0.1593579 -1.825817 0.06787791
AsocLearn           -0.30203537         NA 0.1792897 -1.684622 0.09206155


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified varaibles
Social transmission and asocial learning assumed to combine
multiplicatively

                            Df  LogLik     AIC    AICc      LR       p
With Social Transmission     3  66.932 139.864 140.285   0.321   0.571
Without Social Transmission  2  67.092 138.185 138.392
```

In this case, the table of coefficients provides standard errors and hypothesis tests based on the Wald statistic for each individual-level variable: this is taken from the output from the coxph function used to fit the model (see Part A).

We can fit different social transmission parameters for each group using the sParam argument to multiCoxFit in an equivalent manner to addFit. For example, to fit a separate parameter for each group:

```
> multiModel3<-
multiCoxFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2
","nG2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),
bounded=F,sParam=c(1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3),formula=~.+objNeo+AsocL
earn)
> summary(multiModel3)

Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                          Estimate    Bounded        se         z          p
Social transmission 1  1.582514e+06 0.99999937        NA        NA         NA
Social transmission 2  1.894226e-02 0.01859012        NA        NA         NA
Social transmission 3  0.000000e+00 0.00000000        NA        NA         NA
objNeo                -2.549473e-01         NA 0.1561986 -1.632199 0.10263750
AsocLearn             -3.162671e-01         NA 0.1841179 -1.717742 0.08584365


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine multiplicatively

                           Df  LogLik     AIC    AICc      LR       p
With Social Transmission    5  64.351 138.701 139.792   5.484   0.140
Without Social Transmission 2  67.092 138.185 138.392
```

And to fit the final model presented in the main text:

```
> multiModel4<-
multiCoxFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2
","nG2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),
bounded=F,sParam=c(1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0),formula=~.+objNeo+AsocL
earn)
> summary(multiModel4)

Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                        Estimate Bounded        se         z          p
Social transmission 1 998.9998976   0.999        NA        NA         NA
objNeo                 -0.2468009      NA 0.1557507 -1.584590 0.11305962
AsocLearn              -0.3222195      NA 0.1842196 -1.749105 0.08027285


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine
multiplicatively

                           Df  LogLik     AIC    AICc      LR       p
With Social Transmission    3  64.359 134.718 135.139   5.467   0.019
Without Social Transmission 2  67.092 138.185 138.392
```

To include a random effect to account for repeated observations on individuals, we can include a frailty term (see Therneau & Grambsch 2000 for details). To do this we need to ensure that each individual has a unique identifier which is the same across diffusions. This is done when inputting the data using the oaData function, with the idname argument. For example, to input the data for all three groups for task 1:

```
> nG1T1<-oaData(idname=1:6,assMatrix=amg1, asoc=asoc, orderAcq=oag1t1,
groupid="1", taskid="1")
> nG2T1<-oaData(idname=7:10,assMatrix=amg2, asoc=asoc1, orderAcq=oag2t1,
groupid="2", taskid="1")
> nG3T1<-oaData(idname=11:15,assMatrix=amg3, asoc=asoc2, orderAcq=oag3t1,
groupid="3", taskid="1")
```

[This code is for illustration of how to specify a unique id for each individual, it will not work unless asoc1, asoc2, oag2t1 and oag3t1 are specified. The id's are included in the oadata objects provided in "startlingSolve", and so do not need to be specified here] We include the random effect by including it as a term in the model:

```
> frailtyModel<-
multiCoxFit(oadata=c("nG1T1","nG1T2","nG1T4","nG1T5","nG1T6","nG2T1","nG2T2
","nG2T3","nG2T4","nG2T5","nG2T6","nG3T1","nG3T2","nG3T3","nG3T4","nG3T5"),
bounded=F,sParam=c(1,1,1,1,1,0,0,0,0,0,0,0,0,0,0),formula=~.+frailty(id)+
objNeo+AsocLearn)
```

The coxph function automatically prints the model output to the screen when a frailty term is included. The first part shows there is not strong evidence for a random effect:

```
Call:
coxph(formula = Surv(time1, time2, status) ~ strata(group, task) +
    frailty(id) + objNeo + AsocLearn + offset(log(sVect * stMetric +
    1)), data = oadata@coxdata)

  n= 211
            coef   se(coef) se2   Chisq DF   p
frailty(id)                       2.83  1.33 0.140
objNeo      -0.241 0.171    0.162 1.99  1.00 0.160
AsocLearn   -0.346 0.189    0.185 3.36  1.00 0.067
```

Because of the way the coxph function works the coefficients for individual-level variables are not included in the summary for the model when a frailty term is included. However, we can see how the test for social transmission is affected by the inclusion of the frailty term:

```
> summary(frailtyModel)

Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                     Estimate Bounded
Social Transmission 998.9999   0.999


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine
multiplicatively

                            Df  LogLik     AIC    AICc      LR       p
With Social Transmission     4  62.043 132.087 132.801   3.994   0.046
Without Social Transmission  3  64.040 134.081 134.502
```

Extracting coefficients from a null model

In cases where there is no evidence for social transmission, you may wish to obtain estimates of the other coefficients in the model after removing the social learning component. This can be done by using the nullSummary function:

```
>nullSummary(addModel2)

Summary of Null Model
Order of Acquisition Data


Coefficients
            Estimate
objNeo    -0.2714150
AsocLearn -0.3287107
```

This can currently only be done for additive OADA models, so if a multiplicative model has been used, you must first fit the equivalent additive model (for which the null model will be the same), and then use the nullSummary function.

Continuous versus discrete TADA

Here I describe the difference between the two approaches, taken from the supplementary material from Hoppitt, Boogert & Laland (2010).

For model fitting, Franz and Nunn (2009) convert the TADA model into a discrete form by splitting time into time steps of equal length. This allows one to model the probability an individual, *i*, learns in a given time step, *d*, as, in our notation:

$$p_{i,d} = 1 - \exp\left(-\tau \sum_{j=1}^{N} a_{i,j} z_j(t_d)\right), \quad \text{(E1)}$$

where $\tau = sL$, where L is the length of each time step, and t_d is the time at which time step d starts.

An alternative approach is to keep time of acquisition data in continuous form. Then, for the n^{th} acquisition event the likelihood of individual *i* acquiring the behaviour next is the product of the probability density for individual *i* acquiring the trait and the probability density for all other naïve individuals not acquiring the trait, in the period since the $(n\text{-}1)^{th}$ acquisition event. This is given by

$$l_n = \lambda_{i,n-1} \exp(-\lambda_{i,n-1}(T_n - T_{n-1})) \prod_{j \neq i} \exp(-\lambda_{j,n-1}(T_n - T_{n-1})), \quad \text{(E2)}$$

where *i* is the individual acquiring the trait, $\lambda_{i,n-1}$ is $\lambda_a(t)$ given by Eq. (9) and evaluated immediately after acquisition event *n-1*, and T_n is the time of event *n* (*n*>1) or the start of the period of observation (*n*=0). For complete diffusions where all individuals acquired the trait the total -log-likelihood is the sum of -log-likelihoods across all acquisition events. For an incomplete diffusion the total -log-likelihood is given by

$$-\log L(data) = -\sum_{n=1}^{n=N} \log(l_n) - \log\left(\prod_{j} \exp(-\lambda_{j,n-1}(T_{end} - T_Q))\right), \quad \text{(E3)}$$

where *Q* is the last acquisition event observed, and T_{end} is the time at which observation ceased.

One would expect the discrete TADA to be inaccurate when step length is large, since it does not allow for the fact that individuals learning in the same time step can learn from each other. However, the discrete and continuous TADA models will have equivalent results as step size becomes small (see Part F). In practice, we find that optimisation is much faster for the continuous TADA, and suggest this is used when exact times of acquisition are known.

The discrete TADA has the advantage that it can be used when exact times of acquisition are not known, for example, when the population status is not monitored continuously, but instead in scans, giving a series of "snapshots" telling us which individuals have acquired the trait. Whilst Franz & Nunn's algorithm assumes that step lengths are equal, the model can easily be modified to unequal step lengths, by replacing Eqn. E1 with:

$$p_{i,d} = 1 - \exp\left(-sL_d \sum_{j=1}^{N} a_{i,j} z_j(t_d)\right), \quad \text{(E4)}$$

where L_d is the length of the d^{th} time step.

Continuous TADA

Inputting TADA data

We need to create a taData object: this is easy if an oaData object has already been created, we merely need to specify the oaData object, the times at which each acquisition event occurred, and the time observation ceased (if it is a complete diffusion any time after the last acquisition event can be specified as the end time). For example, for group 1 task 1:

```
> tempData<-taData(timeAcq=c(2024,2281,2507),oadata=nG1T1,endTime=3473)
```

The taData objects for the Boogert et al. data are contained in the "starlingSolve" file, which can be loaded as described above. These are named "ntG1T1","ntG1T2" etc.

Fitting a TADA model with a constant baseline rate

Additive and multiplicative TADA models with a constant baseline rate are fit in the same way, using the tadaFit function. This fits a model that assumes that the baseline rate of acquisition is constant (see below for a non-constant case). Single and multiple diffusions are included in the analysis in the same way as for OADA, for example:

```
> model1<-tadaFit(tadata=ntG1T1)

> tadaModel1<-
tadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1","ntG
2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","ntG3T4
","ntG3T5"))

> summary(tadaModel1)

Summary of Additive Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1    0.1802310 0.152708225905150
1/Rate scaling        9829.0485605


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc      LR         p
With Social Transmission     2  580.70 1165.40 1165.61   14.99 0.0001081
Without Social Transmission  1  588.20 1178.39 1178.46
```

There is now an extra line in the coefficients table that corresponds to the inverse of the rate scaling parameter (λ_0), denoted the main text as L_0.

We can add in individual-level variables in the same way as for addFit. For example, to fit the additive model with LTFNE:

```
> tadaModel2<-
tadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1","ntG
2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","ntG3T4
","ntG3T5"),asocialVar=3)

> summary(tadaModel2)

Summary of Additive Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1     0.1950484 0.163213835413724
1/Rate scaling        10393.2950779
novEnv                   -0.2290814


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                           Df  LogLik     AIC    AICc      LR         p
With Social Transmission    3  577.91 1161.82 1162.24   15.54 8.074e-05
Without Social Transmission 2  585.68 1175.36 1175.56
```

The additive model is fitted as default, however, if we specify additive=F:

```
> tadaModel3<-
tadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1","ntG
2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","ntG3T4
","ntG3T5"),asocialVar=3,additive=F)

> summary(tadaModel3)

Summary of Multiplicative Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1     0.2007080 0.167158042742246
1/Rate scaling        10191.1892771
novEnv                   -0.1643009


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine
multiplicatively

                           Df  LogLik     AIC    AICc      LR         p
With Social Transmission    3  577.30 1160.60 1161.02   16.75 4.254e-05
Without Social Transmission 2  585.68 1175.36 1175.56
```

We can also add in variables representing differences in rate of acquisition between groups and/or tasks, using the group and task arguments. For example, we can test for an effect of groups thus:

```
> tadaModel4<-
tadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1","ntG
2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","ntG3T4
","ntG3T5"),asocialVar=3,group=T)
> summary(tadaModel4)

Summary of Additive Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1    0.16146118 0.139015563359281
1/Rate scaling        8664.17804107
novEnv                  -0.23627786
Group 2                 -0.41193333
Group 3                 -0.06243383


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                           Df  LogLik     AIC    AICc      LR         p
With Social Transmission    5  577.48 1164.96 1166.05   15.87 6.801e-05
Without Social Transmission 4  585.41 1178.82 1179.54
```

The summary shows the fitted effects of each group as treatment contrasts, estimating the difference each group has with group 1. We can test for an overall effect of "group" using an LRT to compare tadaModel4 with tadaModel2:

```
> 1- pchisq(2*(577.91 - 577.48),1)
[1] 0.3537387
```

So there is no evidence for a difference between groups in the overall rate of acquisition.

As with addFit and multiCoxFit, we can fit different social transmission parameters for each diffusion, or constrain some to be zero. This is done using the sParam argument as before. For example, we can fit a model with separate s parameters for each group as follows:

```
> tadaModel5<-
tadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1","ntG
2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","ntG3T4
","ntG3T5"),asocialVar=3,sParam=c(1,1,1,1,1,2,2,2,2,2,2,3,3,3,3,3))
> summary(tadaModel5)

Summary of Additive Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1     0.1581271 0.136536934107653
Social Transmission 2     0.2069510 0.171465934096756
Social Transmission 3     0.2187780 0.179506038378442
1/Rate scaling        10399.9343379
novEnv                   -0.2332289
```

```
Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc      LR        p
With Social Transmission     5  577.77 1165.54 1166.63   15.82 0.001235
Without Social Transmission  2  585.68 1175.36 1175.56
```

We can compare this with a tadaModel2 with an LRT:

```
> 1- pchisq(2*(577.91 - 577.77),2)
[1] 0.8693582
```

So there is also no evidence for a difference between groups in the strength of social transmission.

Fitting a pure social learning model

In their original TADA Franz and Nunn (2009) suggest comparing model in which all acquisition is asocial learning with a model in which all acquisition is social transmission, and comparing the two using AIC. The then introduce the 'Extended NBDA' (since renamed 'Extended TADA' Franz & Nunn 2010), which allows both to occur. It is this Extended TADA that is implemented by tadaFit. However, our code also allows a continuous time version of Franz & Nunn's pure social transmission model, using the function pureSocialTada. This only makes sense for diffusions with a seeded demonstrator, or the pure social transmission model is impossible.

```
tempData<-oaData(assMatrix=amg1, asoc=asoc, orderAcq=oag1t1, groupid="1",
taskid="1", demons=c(0,0,1,0,0))

tempData.ta<-taData(timeAcq=c(2024,2281,2507),oadata=tempData,endTime=3473)

> pureModel1<-pureSocialTada(tempData.ta)
> summary(pureModel1)

Summary of Pure Social Transmission Model
Time of Acquisition Data
Social transmission 1 constrained to 1.

Coefficients
                     Estimate
Social Transmission 1     1.00
1/Rate scaling        18602.67


Pure social transmission model

All acquisition through social network

                         Df LogLik    AIC   AICc
Pure Social Transmission  1 26.495 54.991 58.991
```

For a single diffusion the social transmission parameter is constrained to 1, meaning only a single parameter is fitted, which is the inverse of the rate of social transmission per unit of

social network connection. If multiple diffusions are included with different social learning parameters, then the first is constrained to 1, and the others are fitted relative to the first social transmission parameter.

Discrete TADA

Note the models fitted here are to illustrate the use of the functions and, containing very little data, do not represent a sensible analysis. Discrete time TADA can be implemented using function discreteTadaFit. This takes a taData object in which timeAcq is a vector of integers giving the step number at which each individual acquired the behaviour and endTime, an integer giving the total number of time steps. discreteTadaFit requires an additional argument, stepLength, which can be:

1. A number giving step length if all step lengths are equal,
2. A numeric vector of length endTime, giving the length of each time step, if step lengths were unequal within a single diffusion,
3. An SxD matrix if there are D different diffusions with unequal time steps, with a maximum of S time steps.

For example, we can model the starling data from group 1, task 1 in time steps of 100 seconds thus:

```
> tempData1<-taData(timeAcq=c(20,22,25),oadata=nG1T1,endTime=35)
> model1<-discreteTadaFit(tadata=tempData1, stepLength=100)
```

If time steps had unequal length, we could specify endTime as a vector, giving the length of each time step:

```
> G1T1stepLength=c(10,5,15,10,10,5,15,
10,10,5,15,10,15,10,10,5,15,10,10,5,15,10,10,5,15,10,15,10,10,5,15,10,10,5,
15)
> model1<-discreteTadaFit(tadata=tempData1, stepLength=G1T1stepLength)
```

If we added another diffusion with 16 equal time steps of 400 seconds:

```
> tempData2<-taData(timeAcq=c(12,15),oadata=nG2T1,endTime=16)
> G2T1stepLength=rep(20,16)

> model2<-discreteTadaFit(tadata=c("tempData1","tempData2"),
stepLength=cbind(G1T1stepLength,G2T1stepLength))
```

If the number of time steps in each diffusion is unequal, you will get a warning message like this:

```
In cbind(G1T1stepLength, G2T1stepLength) :
  number of rows of result is not a multiple of vector length (arg 2)
```

But this can be safely ignored.

Whilst the model is fitted using the discrete TADA method of Franz & Nunn (2009), the parameterisation is still that introduced by Hoppitt et al (2010) to aid comparison between models of different step length, or discrete and continuous TADAs.

Seeded demonstrators can be defined using the demons argument to the oaData function (see above). In this case a pure social transmission model (see above) can also be fitted:

```
> tempData<-oaData(assMatrix=amg1, asoc=asoc, orderAcq=c(1,2,4,5),
groupid="1", taskid="1", demons=c(0,0,1,0,0))
```

```
> tempData1<-taData(timeAcq=c(12,15,22,25),oadata= tempData,endTime=35)
> PSLdiscreteModel1<-PSLdiscreteTadaFit(tadata=tempData1,
stepLength=G1T1stepLength)
> summary(PSLdiscreteModel1)

Summary of Pure Social Transmission Model
Dicrete Time of Acquisition Data
Social transmission 1 constrained to 1.

Coefficients
                      Estimate
Social Transmission 1   1.0000
1/Rate scaling        387.4999


Pure social transmission model

All acquisition through social network

                         Df LogLik    AIC   AICc
Pure Social Transmission  1 17.094 36.189 38.189
```

Again the social transmission parameter is constrained to 1, so only a single parameter is fitted.

Non-constant baseline rate TADA

We have found that the original TADA is susceptible to false positives if the baseline rate of acquisition increases over time. This can occur if neophobia decreases over time, or if there is a multi-step structure to the task, which we suggest can result in an approximately gamma distribution of latencies. We can allow for this by including an appropriate baseline rate function. In the supplementary material for Hoppitt, Kandler, Kendal & Laland (2010), we derive the likelihood functions for both continuous and discrete TADA for any specified form of baseline rate function. Here we provide a function, gammaTadaFit, that fits a continuous TADA with a baseline corresponding to a gamma distribution of latency to solve under asocial conditions. This allows for a baseline rate that increases or decreases over time, as determined by the shape parameter, which is fitted to the data. This is used as follows for a continuous TADA:

```
>gammaTadaModel1<-
gammaTadaFit(tadata=c("ntG1T1","ntG1T2","ntG1T4","ntG1T5","ntG1T6","ntG2T1"
,"ntG2T2","ntG2T3","ntG2T4","ntG2T5","ntG2T6","ntG3T1","ntG3T2","ntG3T3","n
tG3T4","ntG3T5"),asocialVar=3)

>summary(gammaTadaModel1)

Summary of Additive Social Transmission Model
Time of Acquisition Data
Unbounded parameterisation.

Coefficients
                          Estimate           Bounded
Social Transmission 1     0.2421479 0.194942885353028
1/Rate scaling        17143.3050829
Shape                     0.7759429
novEnv                   -0.2272496


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                           Df  LogLik     AIC    AICc      LR         p
With Social Transmission    4  576.59 1161.18 1161.89   17.89 2.345e-05
Without Social Transmission 3  585.53 1177.06 1177.49
```

If the shape parameter is >1, the baseline rate is increasing, if it is <1 it is decreasing (this can be tested using an LRT or comparing AIC, see below). Here the model has fitted a decreasing baseline rate, and social transmission remains significant.

We currently do not provide functions to fit a discrete TADA with a non-constant baseline, nor a continuous TADA any other form of baseline function. If such functions are required, please contact the author at the e mail given at the start of the document.

Stratified OADA

The gammaTadaFit function allows one to account for changes in baseline rate, but assumes a specific form for the baseline function. OADA, on the other hand, makes no assumptions about the form of the baseline function, assuming only that it is the same for all individuals within each diffusion. However, OADA has the disadvantage that it is not sensitive to time of acquisition.

A compromise is available: one can fit a modified version of OADA that assumes the baseline rate is the same across all diffusions, and consequently is sensitive to the order of acquisition times across diffusions (Webster, Atton, Hoppitt & Laland, In Prep). As such it is sensitive to cases where individuals within a diffusion tend to acquire the trait at a similar time, in a way that the standard OADA is not. This amounts to merely running an OADA including all individuals in the same diffusion, with one large association matrix/ social network which has a zero connection between all individuals in different diffusions.

Alternatively, one could specify that different sets of diffusions had the same baseline rate, such as all those on the same task. In the terms used in survival analysis, one is including these diffusions in the same stratum. If this is on the basis of task, then one is "stratifying by task". (The standard OADA actually stratifies by diffusion).

Our combineInStratum function allows one to combine any number of diffusions in a single stratum, but preserves information about which individual comes from which diffusion so as to allow different social transmission rates in each diffusion using the sParam argument to addFit or multiCoxFit. Currently this requires all diffusions to end at the same time, so we use a new stickleback dataset to illustrate the use of the function (this will be available when the data is published):

```
> stratByTask<-combineInStratum(taNames=c("G1struc","G2struc",
"G3struc","G4struc","G5struc","G1open","G2open","G3open","G4open",
"G5open"))
> stratModel1<-addFit(stratByTask)
> summary(stratModel1)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                       Estimate   Bounded
Social transmission 1 0.3745303 0.2724788


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df LogLik    AIC   AICc     LR         p
With Social Transmission     1 237.12 476.23 476.29  38.11 6.693e-10
Without Social Transmission  0 256.17 512.34 512.34
```

If we wanted to include two strata, we could do this as follows:

```
> strat1<-combineInStratum(taNames= c("G1struc","G2struc",
```

```
"G3struc","G4struc","G5struc"))
> strat2<-combineInStratum(taNames= c("G1open","G2open","G3open","G4open",
"G5open"))
> stratModel2<-addFit(c("strat1","strat2"))
> summary(stratModel2)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                      Estimate   Bounded
Social transmission 1 0.5380311 0.3498181


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df LogLik    AIC   AICc     LR         p
With Social Transmission     1 190.63 383.26 383.32  29.82 4.751e-08
Without Social Transmission  0 205.54 411.07 411.07
```

Note how the likelihood differs for the null model in each case: this is because the dependent data is different in each case, due to the different stratification structure. This means that OADA models with different stratification structures cannot be compared by AIC.

The combineInStratum function keeps information about the diffusion, group and task for each individual in the stratum, meaning group and task can be used as factors (unlike in the standard OADA), and the sParam function can be used to test for different social transmission rates in each diffusion, for example:

```
> stratModel3<- addFit("stratByTask",group=T,
sParam=c(1,1,1,1,1,2,2,2,2,2))
> summary(stratModel3)

Summary of Additive Social Transmission Model
Order of acquisition data
Unbounded parameterisation

Coefficients
                        Estimate    Bounded
Social transmission 1  0.03976654 0.03824564
Social transmission 2  0.02507733 0.02446384
GroupO2               -1.86985766         NA
GroupO3               -2.16176165         NA
GroupO4               -1.66279269         NA
GroupO5               -2.39932356         NA
GroupS1               -2.40042081         NA
GroupS2               -2.34127345         NA
GroupS3               -2.60695982         NA
GroupS4               -2.25341566         NA
GroupS5               -0.77322849         NA


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively
```

```
                           Df LogLik    AIC   AICc     LR        p
With Social Transmission   11 232.22 486.45 491.08  13.58 0.001125
Without Social Transmission 9 239.01 496.03 499.08
```

(Currently the group argument only works if the data is specified as a string- this will hopefully be solved out soon!).

In order to fit a multiplicative model using multiCoxFit, one has to supply an additional "stratify" argument specifying the variable by which one is stratifying, either "none", meaning all diffusions are modelled in a single stratum, "task" stratifying by task, "group" stratifying by group, or "both" which gives a standard OADA. It is currently not possible to specify other stratifications for the multiCoxFit function, though this could be accomplished by appropriate choice of "taskid" or "groupid" when inputting the data.

```
> multiStratModel<-multiCoxFit("stratByTask", formula=~.+group,
sParam=c(1,1,1,1,1,2,2,2,2,2), stratify="none")
> summary(multiStratModel)

Summary of Multiplicative Social Transmission Model
Order of acquisition data
Unbounded parameterisation.

Coefficients:
                        Estimate   Bounded        se          z         p
Social transmission 1  0.47975560 0.3242127        NA         NA        NA
Social transmission 2  0.28804997 0.2236326        NA         NA        NA
groupO2               -0.19655937        NA 0.5116385 -0.3841762 0.7008478
groupO3               -0.61493135        NA 0.5310266 -1.1580048 0.2468621
groupO4                0.36353519        NA 0.5175721  0.7023856 0.4824387
groupO5               -0.18775216        NA 0.5195169 -0.3613976 0.7178023
groupS1               -0.71328429        NA 0.6262025 -1.1390633 0.2546768
groupS2               -0.40208583        NA 0.5480367 -0.7336842 0.4631412
groupS3               -0.43345499        NA 0.6319708 -0.6858782 0.4927899
groupS4               -0.13707312        NA 0.5165826 -0.2653460 0.7907430
groupS5               -0.05361406        NA 0.5235275 -0.1024093 0.9184318


Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine
multiplicatively

                           Df LogLik    AIC   AICc     LR      p
With Social Transmission   11 233.82 489.64 494.27  10.39   0.01
Without Social Transmission 9 239.01 496.03 499.08
```

Note here that stratify= "none" and stratify= "task" fit the same model here, since there is only one task.

Model selection

In choosing the best model, a researcher must take into account each individual level variable, group and task differences in rate, additive or multiplicative model, constant or non-constant baseline function, and of course, social transmission or not. We recommend that a researcher fits models including every possible combination of variables and compares AIC or AIC_c values (Hoppitt & Laland In Prep). This can be done automatically using our function aicTable. Here the user enters the data, and specifies which individual level variables are to be considered (asocialVar argument) whether group is to be considered (group argument), which differences in social transmission parameters are to be considered (sParamMatrix argument), whether a pure social transmission model is to be considered (pure argument), and whether AIC (aic="aic") or AIC_c (aic="aicc") values are to be returned. Burnham & Anderson (2002) suggest that AIC_c be used when the ratio of data points to parameters is <40. For NBDA we (Hoppitt, Boogert, et al., 2010) suggest that the number of data points be taken as the number of acquisition events, so we anticipate that AIC_c will be used for most datasets, and consequently set "aicc" as default.

For example to use this with continuous TADA, we first specify what different sParam values we are to consider by specifying them as the rows of a matrix:

```
> sParamMatrix<-rbind(
 rep(1,10),
 rep(c(1,2),each=5),
 rep(c(1,0),each=5),
 rep(c(0,1),each=5))
> sParamMatrix

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    1    1    1     1
[2,]    1    1    1    1    1    2    2    2    2     2
[3,]    1    1    1    1    1    0    0    0    0     0
[4,]    0    0    0    0    0    1    1    1    1     1
```

Here the first line corresponds to equal social transmission rates in all diffusions, the second to different social transmission rates in the first and last five diffusions, the third to no social transmission in diffusions 6-10 and the last to no social transmission in diffusions 1-5. We then specify which variables are to be considered:

```
> load("stratDiffs")
> aicTable(data=c("G1struc","G2struc","G3struc","G4struc",
"G5struc","G1open","G2open","G3open","G4open","G5open"),
asocialVar=1,group=T,task=F,sParamMatrix=sParamMatrix, aic="aicc",pure=F)
```

The output, shown overleaf, is a table giving the models in order of AIC or (in this case AIC_c), with "Baseline" giving whether the baseline was "constant" or "non-constant", "Additive" being "TRUE" for an additive model or "FALSE" for a multiplicative model, and "NA" if the model is asocial or contains no individual level variables. "Group?" is "TRUE" if group is a factor affect rate of acquisition in the model, and likeswise for "Task?". "ILVs" list the other individual level variables in the model, as they are encoded in the input files, with "0" denoting no other ILVs. "sParamIndex" gives the row of sParamMatrix used to parameterise social transmission in each diffusion, so here the top model is equal social transmission (per unit of social network connection) in all diffusions. "Social?" is "social" if social transmission is included in the model, and "asocial" otherwise. "AICc" (or "AIC")

gives the requested information criterion, and “deltaAICc” (or “deltaAICc”) gives the difference in information criterion from the best model.

	Baseline	Additive?	Group?	Task?	ILVs	sParamIndex	Social?	AICc	deltaAICc
12	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"1"	"social"	"1003.87185537143"	"0"
14	"non-constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"1005.90310893142"	"2.03"
31	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"2"	"social"	"1006.07280108474"	"2.2"
16	"non-constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"1006.12718465730"	"2.26"
1	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"1"	"social"	"1006.51735539717"	"2.65"
3	"constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"1007.70479595568"	"3.83"
5	"constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"1007.94893685677"	"4.08"
33	"non-constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"1008.10813643168"	"4.24"
34	"non-constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"1008.10813643617"	"4.24"
23	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"2"	"social"	"1008.39701269479"	"4.53"
25	"constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"1009.92488774745"	"6.05"
26	"constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"1009.92488774878"	"6.05"
8	"constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"1013.94517926437"	"10.07"
19	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"1015.84495309710"	"11.97"
28	"constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"1016.86758091667"	"13"
11	"constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"1016.88502889259"	"13.01"
6	"constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"1018.00733660787"	"14.14"
36	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"1018.89165123464"	"15.02"
22	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"1018.89170634416"	"15.02"
30	"constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"1019.91433416354"	"16.04"
17	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"1020.75414926212"	"16.88"
27	"constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"1020.81985091583"	"16.95"
9	"constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"1020.94718623239"	"17.08"
38	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"1022.05124719146"	"18.18"
35	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"1023.76358692333"	"19.89"
20	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"1023.80090250929"	"19.93"
29	"constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"1023.86660416518"	"19.99"
55	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"4"	"social"	"1024.56262760441"	"20.69"
57	"constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"1025.75289708169"	"21.88"
58	"constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"1025.75289708176"	"21.88"
63	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"4"	"social"	"1026.14004061494"	"22.27"
60	"constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"1026.41165913337"	"22.54"
67	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"1026.69479519860"	"22.82"
37	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"1026.923182881"	"23.05"
68	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"1027.19272568309"	"23.32"
59	"constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"1027.20818918484"	"23.34"
66	"non-constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"1027.83958757400"	"23.97"
65	"non-constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"1027.83958757442"	"23.97"
62	"constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"1029.35150875792"	"25.48"
69	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"1029.74154846670"	"25.87"
61	"constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"1030.14803883788"	"26.28"
70	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"1030.23947892973"	"26.37"
49	"non-constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"1035.14386638796"	"31.27"
50	"non-constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"1035.14386638796"	"31.27"
41	"constant"	"TRUE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"1035.99742485693"	"32.13"
42	"constant"	"FALSE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"1035.99742485693"	"32.13"
52	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"1036.04750791669"	"32.18"
18	"non-constant"	"NA"	"TRUE"	"FALSE"	"0"	"1"	"asocial"	"1036.71391829560"	"32.84"
51	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"1038.45103805037"	"34.58"
54	"non-constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"1039.09426116199"	"35.22"
21	"non-constant"	"NA"	"TRUE"	"FALSE"	"1"	"1"	"asocial"	"1039.65376791999"	"35.78"
53	"non-constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"1041.49779129791"	"37.63"
44	"constant"	"FALSE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"1042.09916686877"	"38.23"
15	"non-constant"	"NA"	"FALSE"	"FALSE"	"1"	"1"	"asocial"	"1044.40332328187"	"40.53"
46	"constant"	"FALSE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"1045.03901649323"	"41.17"
43	"constant"	"TRUE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"1045.25490681728"	"41.38"
45	"constant"	"TRUE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"1048.19475645728"	"44.32"
7	"constant"	"NA"	"TRUE"	"FALSE"	"0"	"1"	"asocial"	"1051.74461921057"	"47.87"
4	"constant"	"NA"	"FALSE"	"FALSE"	"1"	"1"	"asocial"	"1051.89379083425"	"48.02"
10	"constant"	"NA"	"TRUE"	"FALSE"	"1"	"1"	"asocial"	"1054.58309471183"	"50.71"
13	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"1"	"asocial"	"1057.25953462932"	"53.39"
32	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"2"	"asocial"	"1057.25953462932"	"53.39"
48	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"3"	"asocial"	"1057.25953462932"	"53.39"
64	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"4"	"asocial"	"1057.25953462932"	"53.39"
47	"non-constant"	"NA"	"FALSE"	"FALSE"	"0"	"3"	"social"	"1057.86973704694"	"54"
39	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"3"	"social"	"1058.36173668654"	"54.49"
2	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"1"	"asocial"	"1060.35395747393"	"56.48"
24	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"2"	"asocial"	"1060.35395747393"	"56.48"
40	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"3"	"asocial"	"1060.35395747393"	"56.48"
56	"constant"	"NA"	"FALSE"	"FALSE"	"0"	"4"	"asocial"	"1060.35395747393"	"56.48"

In interpreting the output, we suggest researchers follow Burnham and Anderson's (2002) rules of thumb, taking any model with deltaAICc/ deltaAIC less than 2 as having substantial support, with there being little to chose between them, and model with deltaAICc/ deltaAIC greater than 4 as having considerably less support, and any model with deltaAICc/ deltaAIC greater than 10 as having no support. We suggest that a truncated table is presented as an appendix to any paper using NBDA giving at least all models with AIC /AIC_c <4, and if either are not included in this subset, an extra line showing the best model with or without social transmission.

In my example, the best model without social transmission has deltaAICc= 32.84, indicating strong evidence for social transmission. There is very little evidence for a difference between groups in rate of acquisition (that is not attributed to social transmission), and there is no strong evidence for an effect of the only ILV, nor for a difference between groups in the rate of social transmission (per unit of social network connection). There is more support for a non-constant than constant baseline rate. I would now go on to investigate the parameter values for the best model(s) and obtain confidence intervals for these (see below).

One could make a more stringent (and frequencist) test for social transmission by identifying the best model without social transmission and testing it against a model with social transmission added in:

```
> bestAsocialModel<-
gammaTadaFit(tadata=c("G1struc","G2struc","G3struc","G4struc",
"G5struc","G1open","G2open","G3open","G4open","G5open"), group=T,
additive=T)
> anova(bestAsocialModel)

Likelihood Ratio Test for Social Transmission:

Null model includes all other specified variables
Social transmission and asocial learning assumed to combine additively

                            Df  LogLik     AIC    AICc      LR         p
With Social Transmission    12  495.46 1014.92 1020.49   19.02 1.291e-05
Without Social Transmission 11  504.97 1031.95 1036.58
```

Here one still has to choose which social transmission model to compare against, i.e. multiplicative versus additive model, and whether different diffusions have different rates, so I prefer the AIC approach.

One can use aicTable function for OADA as follows:

```
>aicTable(data=c("G1struc.oa","G2struc.oa","G3struc.oa","G4struc.oa",
"G5struc.oa","G1open.oa","G2open.oa","G3open.oa","G4open.oa","G5open.oa"),
asocialVar=0,group=F,task=F,sParamMatrix=sParamMatrix, aic="aicc",pure=F)

[output omitted]
```

Note that we cannot include differences between groups and tasks here, because such differences will be accounted for in the separate baseline functions for each diffusion. The same applies for any ILV that does not vary with diffusions, such as the "open" ILV in this dataset. However, we can include these if we include multiple diffusions in each stratum. In this case we include all diffusions in a single stratum. Here one currently has to specify the

stratified data as a string, and specify the stratification using the argument “stratify” to allow the multiplicative model to fit. Here stratify=”none” or stratify=”task” achieve the same as there is only one task.

```
>aicTable(data="stratByTask", asocialVar=1, group=T, task=F,
sParamMatrix=sParamMatrix, aic="aicc", pure=F, stratify="none")
```

	Additive?	Group?	Task?	ILVs	sParamIndex	Social?	AICc	deltaAICc
1	"NA"	"FALSE"	"FALSE"	"0"	"1"	"social"	"476.293203418507"	"0"
12	"NA"	"FALSE"	"FALSE"	"0"	"2"	"social"	"476.611309368467"	"0.32"
5	"FALSE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"477.529601097527"	"1.24"
3	"TRUE"	"FALSE"	"FALSE"	"1"	"1"	"social"	"478.392736265737"	"2.1"
13	"TRUE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"478.566889369028"	"2.27"
14	"FALSE"	"FALSE"	"FALSE"	"1"	"2"	"social"	"478.566889369029"	"2.27"
6	"TRUE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"489.280660170298"	"12.99"
15	"TRUE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"491.079545967498"	"14.79"
8	"FALSE"	"TRUE"	"FALSE"	"0"	"1"	"social"	"491.775798738241"	"15.48"
9	"TRUE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"492.119135669647"	"15.83"
17	"TRUE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"494.019395591444"	"17.73"
16	"FALSE"	"TRUE"	"FALSE"	"0"	"2"	"social"	"494.273064048653"	"17.98"
11	"FALSE"	"TRUE"	"FALSE"	"1"	"1"	"social"	"494.614274237338"	"18.32"
20	"TRUE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"495.945441537121"	"19.65"
21	"FALSE"	"FALSE"	"FALSE"	"1"	"3"	"social"	"495.945441538666"	"19.65"
26	"NA"	"FALSE"	"FALSE"	"0"	"4"	"social"	"496.569989501547"	"20.28"
29	"TRUE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"496.870646172907"	"20.58"
18	"FALSE"	"TRUE"	"FALSE"	"1"	"2"	"social"	"497.212913672715"	"20.92"
27	"TRUE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"498.398462771571"	"22.11"
28	"FALSE"	"FALSE"	"FALSE"	"1"	"4"	"social"	"498.398462789375"	"22.11"
7	"NA"	"TRUE"	"FALSE"	"0"	"1"	"asocial"	"499.078360406132"	"22.79"
30	"FALSE"	"TRUE"	"FALSE"	"0"	"4"	"social"	"499.29787896608"	"23"
22	"TRUE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"499.423095979957"	"23.13"
31	"TRUE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"499.709121672594"	"23.42"
23	"FALSE"	"TRUE"	"FALSE"	"0"	"3"	"social"	"500.248165993605"	"23.95"
10	"NA"	"TRUE"	"FALSE"	"1"	"1"	"asocial"	"501.820616389311"	"25.53"
32	"FALSE"	"TRUE"	"FALSE"	"1"	"4"	"social"	"502.136354465173"	"25.84"
24	"TRUE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"502.261571479235"	"25.97"
25	"FALSE"	"TRUE"	"FALSE"	"1"	"3"	"social"	"503.086641492697"	"26.79"
4	"NA"	"FALSE"	"FALSE"	"1"	"1"	"asocial"	"505.230734359594"	"28.94"
19	"NA"	"FALSE"	"FALSE"	"0"	"3"	"social"	"512.232861422403"	"35.94"
2	"NA"	"FALSE"	"FALSE"	"0"	"1"	"asocial"	"512.34163287964"	"36.05"

The output (overleaf) shows very similar results to the TADA model selection above, although there is now more support for models including the “open” ILV and with different social transmission parameters.

Note that aicTable might take a long time to fit all models if there are a lot of ILVs and different social transmission parameterisations, but given the time it takes to collect the data, I suggest it is worth waiting for an exhaustive search of all the models of interest, rather than using a stepwise procedure.

Tied data

OADA and TADA allow us to model, and thus correct for confounding variables. However, there is another way in which we might detect a spurious social transmission effect: individuals who associate together might simply be more likely to encounter a 'task' at similar times, and so acquire a behavioural trait at a similar time. Though this might be described as social learning in the broad sense, it would not be social transmission as defined in the main text, since the acquisition of the trait by one individual does not exert a positive causal influence on others' rate of acquisition. One way to allow for this is to only update the status of demonstrators at an appropriate time. For example, in an experimental diffusion (e.g. Boogert et al. 2008) when the experimental apparatus is removed at the end of a session, or for a diffusion in a natural population of birds, at the end of the day when they have returned to their roosting site. In this way, one considers any individuals learning on the same day/session to be 'tied', since we assume social transmission did not occur between them, though we do acknowledge the order in which they acquired the trait. Though this might rule out cases where social transmission is rapid, it could be considered a conservative approach. This type of tie can be included when inputting the data using function oaData.

For example, consider a group of three individuals, A, B and C, with the following association matrix:

```
> am<-matrix(data=c(0,1,1,1,0,2,1,2,0), nrow=3)
> am

     [,1] [,2] [,3]
[1,]    0    1    1
[2,]    1    0    2
[3,]    1    2    0
```

Imagine now that A learns first, on day 1, followed by B and C in rapid succession on day 2. If we enter the data without ties, we assume that that C has a total association of 3, since it could have learned from A or B. If we enter the data with B and C tied, we assume that C could not have learned from D, and so its total association with demonstrators at time of acquisition is 1. We specify a tie by giving a vector with length equal to the diffusion chain, giving a 1 for events that were tied with the event before, and a 0 for those that were not. For the example above:

```
> tempData<-oaData(assMatrix=am, asoc=matrix(0,nrow=3), orderAcq=c(1,2,3),
groupid="1", taskid="1",ties=c(0,0,1))
```

This procedure works for both OADA and the continuous TADA fitted by tadaFit.

An alternative to this for TADA (new in v1.2) is to specify the times at which you want each individuals status to be updated (in the sense of when they start transmitting to others). This is done by providing a vector of times for all individuals to acquire the trait to the updateTimes argument of the taData function.

Another type of OADA tie (which we term a "true tie") occurs when we are not sure of the order in which two or more individuals learned, for example, because the data was collected as a sequence of discrete scans. The appropriate way to allow for "true ties" is to calculate the likelihood of every possible order of acquisition that could have led to the observed data, and add these to get the true likelihood of the data (though this may be unfeasible if there are a

large number of ties). In the example above, imagine that we are not sure whether individual 2 learned first or individual 3. The –log-likelihood for s=0.5 for the order 1,2,3 is 1.791759, and it is the same for 1,3,2. The likelihood for each order is therefore exp(-1.791759), and the likelihood for either 1,2,3 or 1,3,2 is 2*exp(-1.791759). Therefore the appropriate –log-likelihood is -log(2*exp(-1.791759))= 1.098612.

The addFit function automatically corrects for true ties if they are specified by the trueTies argument to the oaData function. Here one specifies a list of length corresponding to the number of ties, and each component of the list containing a vector specifying which events are involved in that true tie. For example, imagine we have a population of eight individuals (labelled 1-8), and we collect the data in four discrete scans, which indicates that individual 8 learned first, followed by 6 and 5, then 1,2,3 with 7 and 4 last. We can specify the order of acquisition as:

```
orderAcq=c(8,6,5,1,2,3,7,4)
```

and the true ties as:

```
trueTies=list(c(2,3),c(4,5,6),c(9,10))
```

In the continuous TADA we assume that times of acquisition are accurate, so we should not have any problems with ambiguity in the order of acquisition. If two individuals have tied times of acquisition it is assumed they acquired the trait at the exact same time, and the likelihood is calculated according to the usual algorithm. Our taData function codes the data appropriately so that this happens.

In the discrete TADA, true ties are modelled when two individuals are recorded as having acquired the trait in the same time step. Unlike in OADA, the discrete TADA does not allow for the fact that these individuals might have learned from each other.

We ran simulations to compare the performance of OADA and the discrete TADA in analysing data with true ties. 1000 replicates of data were generated for a group size N=20 , with $\lambda_0(t) = 0.002$ and (s=0, 0.4, 0.8) using the procedure described in the main text. We then converted the dataset so it was in the discrete format that would have arisen if the group were surveyed at a regular time interval T for T=(1, 5, 10, 20, 50, 100, 200, 500, 750, 1000, 1250, 1500, 1750, 2000, 2250, 2500), resulting in an increasing number of true ties. We then used OADA and TADA to analyse each continuous and discrete dataset. The results are shown in Fig F.1 below.

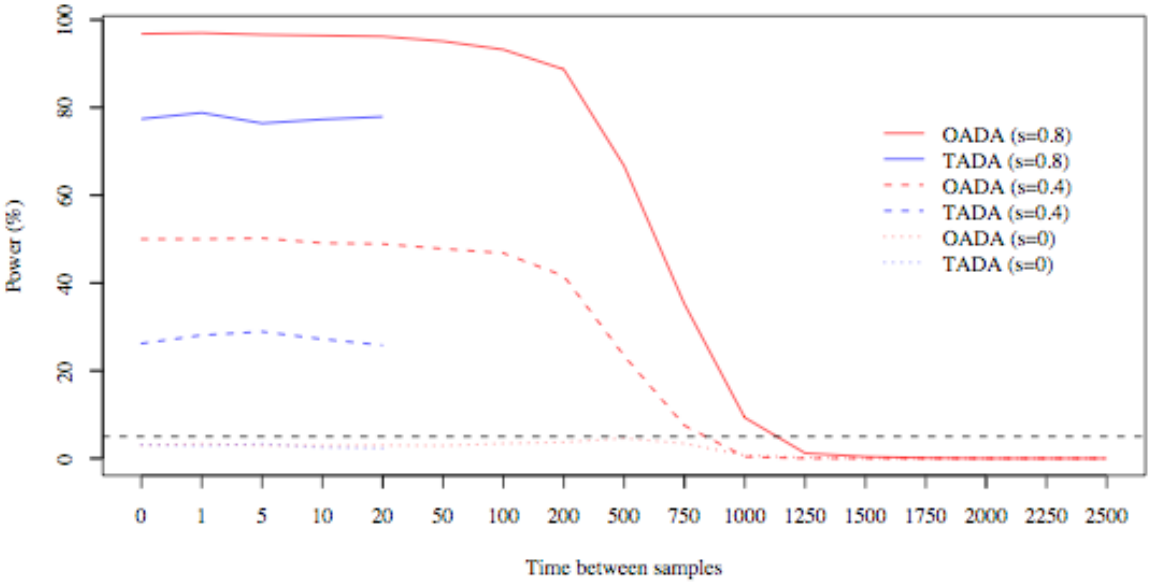

Fig F.1 Power of OADA and TADA when data are recorded at discrete times, as a function of the time between samples (T). The baseline rate of acquisition was set to 0.002. OADA models took too long to fit when T>20 due to the large number of ties.*The key is currently wrong on this chart, OADA is blue and TADA is red

We found that for OADA the power and type 1 error rate remained constant for $T \leq 50$, but that after this the model fitting took too long. This is a result of the computational intensiveness of calculating the likelihood for every possible order of acquisition within each time step. For TADA, the power and type 1 error remained comparable to the continuous form up to $T \approx 200$, after which power dropped dramatically. At this point we also found that the MLE for s became positively biased, though the type 1 error rate was reduced, indicating that such over-inflated estimates for s are unlikely to be accepted as statistically significant.

We conclude that the discrete TADA is likely to perform well so long as the time steps are small relative to the average time of acquisition, but that researchers should expect low statistical power if the data is poorly resolved. A procedure that allows for the fact that tied individuals could have learned from each other might improve performance further.

Glossary of NBDA functions

Currently not updated for V1.2

Here we give a concise description of each function in the file "NBDA functions", including details of the arguments for functions we intend to be called by the user.

```
addCorrectTrueTie(l, data, tiePosition, asocialVar=NULL, bounded=FALSE)
```

Corrects the likelihood for an additive OADA for "true" ties (see Part F). This function is called by addFit and is not intended for the user.

```
addFit(oadata, sParam=NULL, startValue=NULL, bounded=FALSE,
interval=c(0,999), method="optimise", asocialVar=NULL)
```

Fits an additive OADA, returning an object of class addFit. The model output is best interpreted by using the addFit object as an argument to "summary" or "anova" functions (see Part C)

oadata: an object of class oaData, or a character vector giving the names of multiple oaData objects.
sParam: A vector of the same length as oadata, specifying which diffusions have the same parameter for s, and which are constrained to zero (see Part C). If sParam=NULL, a single parameter is fit.
startValue: start values for the nlminb optimisation routine. These are selected automatically be default.
bounded: logical, specified whether the parameterisation in Eqn. 7 (bounded=T) or Eqn. 8 (bounded=F) is to be used.
interval: the search interval for s if the optimise function is used.
method: specifies the function to be used for optimisation. By default, "optimise" is used if there is a single parameter to be fitted, whereas "nlminb" is always used when there is more than one.
asocialVar: numeric vector giving the individual level variables to be included in the model. The numbers refer to the variable's column in oadata@asoc (see Part C).

```
addLikelihood(l,data,asocialVar=NULL,bounded=FALSE, sParam=NULL)
```

Returns the likelihood for an additive OADA. This function is called by addFit and nbdaPlot and is not intended for the user.

```
addLikelihoodBounded(l,data,asocialVar=NULL, sParam=NULL)
```

Returns the likelihood for a bounded additive OADA. This function is called by addLikelihood and is not intended for the user.

```
addLikelihoodNotBounded(l,data,asocialVar=NULL,sParam=NULL)
```

Returns the likelihood for an unbounded additive OADA. This function is called by addLikelihood and is not intended for the user.

```
combineOaCoxData(oaNames, sParam=NULL)
```

Combines the data from multiple oaData objects and returns a single oaData object that can be used in multiCoxLikelihood. Called by multiCoxFit and nbdaPlot and not intended for the user.

```
combineTadaData(taNames, sParam=NULL)
```

Combines the data from multiple taData objects and returns a single taData object that can be used in tadaLikelihood. Called by tadaFit and nbdaPlot and not intended for the user.

```
discreteTadaFit(tadata, sParam=NULL, asocialVar=NULL, startValue=NULL,
bounded=FALSE, task=FALSE, group=FALSE, additive=TRUE, stepLength=1)
```

Fits a discrete TADA, returning an object of class discreteTadaFit. The model output is best interpreted by using the discreteTadaFit object as an argument to "summary" or "anova" functions (see Part C). For a single diffusion with equal time steps this is the same procedure presented by Franz and Nunn.
Parameters are as for tadaFit, except:

stepLength: a number, numerical vector, or matrix giving the length of each time step (see Part E)

```
discreteTadaLikelihood(l, tadata, sParam=NULL, bounded=FALSE,
asocialVar=NULL, task=FALSE, group=FALSE, additive=TRUE, stepLength=1)
```

Returns the likelihood for a discrete TADA. This function is called by discreteTadaFit and nbdaPlot and is not intended for user.

```
multiCoxFit(oadata, sParam=NULL, formula=NULL, startValue=NULL,
bounded=FALSE, interval=c(0,999), method="optimise")
```

Fits a multiplicative OADA, returning an object of class multiCoxFit. The model output is best interpreted by using the multiCoxFit object as an argument to "summary" or "anova" functions (see Part C)

oadata: an object of class oaData, or a character vector giving the names of multiple oaData objects.
sParam: A vector of the same length as oadata, specifying which diffusions have the same parameter for s, and which are constrained to zero (see Part C). If sParam=NULL, a single parameter is fit.

formula: a formula object giving the individual level variables, of the form ~.+{variable 1} + {variable 1}+…
startValue: start values for the nlminb optimisation routine. These are selected automatically be default.
bounded: logical, specified whether the parameterisation in Eqn. 7 (bounded=T) or Eqn. 8 (bounded=F) is to be used.
interval: the search interval for s if the optimise function is used.
method: specifies the function to be used for optimisation. By default, "optimise" is used if there is a single s parameter to be fitted, whereas "nlminb" is always used when there is more than one.

```
multiCoxLikelihood(s,oadata,formula=NULL,bounded=FALSE)
```

Returns the likelihood for a multiplicative OADA. This function is called by multiCoxFit and nbdaPlot and is not intended for the user.

```
nbdaProfile(data=NULL, model=NULL, range=NULL, range2=NULL, additive=TRUE,
bounded=F, resolution=1000, confInt=c(0.95,0.99), ylim=NULL, param=1,
otherParam=NULL, progress=F)
```

Plots a 2D or 3D profile likelihood plot (the latter using the filled.contour function), and returns confidence intervals for the selected parameters (see Part D).

data: an object of class oaData, an object of class taData, or a character vector giving the names of multiple oaData, or multiple taData objects.
model: a object of class multiCoxFit, addFit, tadaFit or discreteTadaFit.
range: the range of the first parameter for which the likelihood is to be plotted.
range2: the range of the second parameter for which the likelihood is to be plotted.
additive: logical, indicating whether the additive (T) or multiplicative (F) is to be used.
bounded: logical, specified whether the parameterisation in Eqn. 7 (bounded=T) or Eqn. 8 (bounded=F) is to be used.
resolution: numeric, the number of values of each parameter for which the likelihood is to be plotted
confInt: numeric vector, the confidence intervals that are to be plotted and returned. By default, 95% and 99%.
ylim: specifies the range over which the likelihood is to be plotted for 2D plots.
param: a numeric vector giving the parameters for which the likelihood is to be plotted (see Part C).
otherParam: a numeric vector giving the values at which the other parameters are to be fixed. If NULL then these are set to their fitted values in the object specified by "model".
progress: logical, if T then the progress of the likelihood calculations is printed to the screen.

```
nulladdLikelihood(l, data, asocialVar=NULL, bounded=FALSE)
```

Returns the likelihood for an additive OADA with s=0. This function is called by tadaFit and nbdaPlot and is not intended for user.

```
discreteNullTadaLikelihood(l, tadata, sParam=NULL, asocialVar=NULL,
task=FALSE, group=FALSE, additive=TRUE, stepLength=1)
```

Returns the likelihood for a discrete TADA with s=0. This function is called by discreteTadaFit and nbdaPlot and is not intended for user.

```
nullTadaLikelihood(l, tadata, sParam=NULL, asocialVar=NULL, task=FALSE,
group=FALSE, additive=TRUE)
```

Returns the likelihood for a TADA with s=0. This function is called by tadaFit and nbdaPlot and is not intended for user.

```
oaData(idname=NULL, assMatrix, asoc, orderAcq, ties=NULL,
trueTies=list(NULL), groupid="1", taskid="1")
```

Creates an object of class oaData, containing all the data necessary for an OADA. Below we assume there are N individuals in the group.

idname: A vector of length N containing a unique identity for each individual, necessary if a frailty model is being fitted (see Part C).
assMatrix: An NxN matrix containing the associations between individuals.
asoc: An NxI matrix containing the I individual-level variables, with column names giving the name of each variable. Individuals must be given in the same order as in assMatrix.
orderAcq: a numeric vector of length <= N giving the order in which individuals acquired the trait. The numbers must refer to the position of each individual in assMatrix.
ties: A vector giving the position of ties (see Part F).
trueTies: A list giving the "true" ties, i.e. where we do not know the order of two or more acquisitions (see Part F).
groupid: A string giving the name of the group. Required for multiCoxFit and tadaFit.
taskid: A string giving the name of the task. Required for multiCoxFit and tadaFit.

```
sampSizeExtract(oadata)
```

Extracts the sample size for one or more oaData objects or taData objects to allow calculation of AIC_c. As Burnham and Anderson (2002) point out, sample size is not always a straightforward issue. Here we take it to be the number of acquisition events. This function is called by addFit and nbdaFit and is not intended for the user.

```
taData(timeAcq, endTime, idname=NULL, oadata=NULL, assMatrix=NULL,
asoc=NULL, orderAcq=NULL, groupid=NULL, taskid=NULL)
```

Creates an object of class taData, containing all the data necessary for a TADA. Arguments are as for oaData except:

timeAcq: a numeric vector of the same length as orderAcq, giving the times of acquistion.

endTime: numeric giving the end of the session. If the session ended at the time of last acquisition, and number greater than this time can be entered.
oaData: an object of class oaData. This can be specified instead of the remaining parameters.

```
tadaFit(tadata, sParam=NULL, asocialVar=NULL, startValue=NULL,
bounded=FALSE, task=FALSE, group=FALSE, additive=TRUE)
```

Fits a TADA, returning an object of class tadaFit. The model output is best interpreted by using the tadaFit object as an argument to "summary" or "anova" functions (see Part C)
Parameters are as for addFit, except:

tadata: an object of class taData, or a character vector giving the names of multiple taData objects.
task: logical, indicating whether task is to be included as a factor in the analysis (see Part C).
group: logical, indicating whether group is to be fitted as a factor in the analysis (see Part C).
additive: logical, T fits the additive TADA, F fits the multiplicative TADA.

```
tadaLikelihood(l, tadata, sParam=NULL, bounded=FALSE, asocialVar=NULL,
task=FALSE, group=FALSE, additive=TRUE)
```

Returns the likelihood for a TADA. This function is called by tadaFit and nbdaPlot and is not intended for user.

References

Boogert, N. J., Reader, S. M., Hoppitt, W., & Laland, K. N. (2008). The origin and spread of innovations in starlings. *Animal Behaviour, 75*, 1509-1518.

Burnham, K. P., & Anderson, D. R. (2002). *Model Selection and Multimodel Inference: A Practical Information-Theoretic Approach* (2 ed.). New York: Springer.

Franz, M., & Nunn, C. L. (2009). Network-based diffusion analysis: a new method for detecting social learning. *Proceedings of the Royal Society B-Biological Sciences, 276*(1663), 1829-1836.

Hoppitt, W., Boogert, N. J., & Laland, K. N. (2010). Detecting social transmission in networks. *J Theor Biol, 263*(4), 544-555.

Hoppitt, W., Kandler, A., Kendal, J. R., & Laland, K. N. (2010). The effect of task structure on diffusion dynamics: Implications for diffusion curve and network-based analyses. *Learning & Behavior, 38*(3), 243-251.