Bayesian NBDA - A tutorial

Laland Lab

November 13, 2013

1 Description

This is a tutorial aimed at demonstrating Bayesian NBDA with the use of three examples. Each example comes with a dataset which is analysed using R code provided on the Laland lab website. The current R code provided is largely for demonstration purposes; a more comprehensive software package is being developed to handle more complicated datasets. Here we expand NBDA to allow it to quantify the evidence for social transmission across a number of diffusions. To do this it is important to account for individual differences in asocial learning by fitting models with appropriate random effects. As random effects can be difficult to implement using maximum likelihood methods, here we take a Bayesian approach. See Nightingale et al. (in Press) for technical details.

The examples build up to individual random effects and model discrimination. The first example involves a dataset which involves 4 diffusions in one population consisting of 10 individuals. This example demonstrates the analysis of a dataset to obtain posterior estimates of the social effect and baseline rate parameters of the NBDA model, but without random effects. The second example concerns the analysis of a dataset with 5 diffusions in a single population with 5 individuals. It illustrates the modelling of random effects at the individual level (i.e. the Bayesian NBDA model is extended to contain random effect parameters). Finally, Example 3, again involves 4 diffusions in a single population of 10 individuals. This example illustrates model discrimination in the Bayesian context with the use of the data analysed in Example 1.

2 Interpretation of the Bayesian NBDA model parameters

For an individual i at time t, and an association matrix A, the general form of the hazard function $\lambda_i(t)$ is expressed as a function of two parameters: the social effect parameter s', which quantifies the magnitude of social influence, and the baseline rate parameter (nuisance parameter), $\lambda_0(t)$, which quantifies asocial performance. For simplicity we treat the baseline rate parameter as a constant, λ_0.

In the case where there is no social component in the model, the hazard rate is written as:

$$\lambda_i(t) = \lambda_0. \tag{1}$$

The NBDA model with this specification of the hazard function is generally considered to be the 'reduced' or 'null' model. This model represents the research hypothesis that the rate at which the behaviour under consideration is transmitted is constant, and that the mechanism by which this is done is asocial. For example, all processes underlying the diffusion are entirely asocial. For ease of reference, we refer to this model as model 1.

When both social and asocial components are considered in the modelling process, the hazard rate is then expressed as:

$$\lambda_i(t) = s^{'} \left(\sum_{i \neq j} A[i,j] z_j(t) \right) + \lambda_0 \tag{2}$$

such that

$$z_j(t) = \begin{cases} 0, & \text{if j is naive} \\ 1, & \text{if j is informed} \end{cases}$$

We denote the model with this hazard function as model 2 which represents the research hypothesis that the transmission of the given behaviour occurs both socially and asocially. The social effect parameter $s^{'}$ may be interpreted as the median effect of the social interactions on the overall hazard rate. For instance, where social transmission underlies the diffusion, $s^{'}$ is a measure of its strength. The baseline rate parameter expresses the hazard of an individual in an asocial environment (for instance, the rate of asocial learning). This parameter is frequently termed a 'nuisance' parameter because the parameter of interest in the NBDA analyses is predominantly $s^{'}$.

When the model is extended to contain random effect parameters at the individual level, the hazard function is expressed as:

$$\lambda_i(t) = s^{'} \left(\sum_{i \neq j} A[i,j] z_j(t) \right) + \lambda_0 \exp(\epsilon_k) \tag{3}$$

where ϵ_k denotes the random effect contributed by individual k. We denote the model with this specification of the hazard function as model 3. This is the appropriate model where there are multiple diffusions for each population.

3 R packages required

Before trying out the analyses it is necessary to install the R packages 'calibrate' and 'mvtnorm'. R code for installing and loading an R package (uisng a Windows operating system) is shown below using the package 'calibrate' as an example.

To install an R package, type (from your workspace in R):

```
install.packages("calibrate")
```

A message should appear on your screen as shown below:

```
--- Please select a CRAN mirror for use in this session ---
```

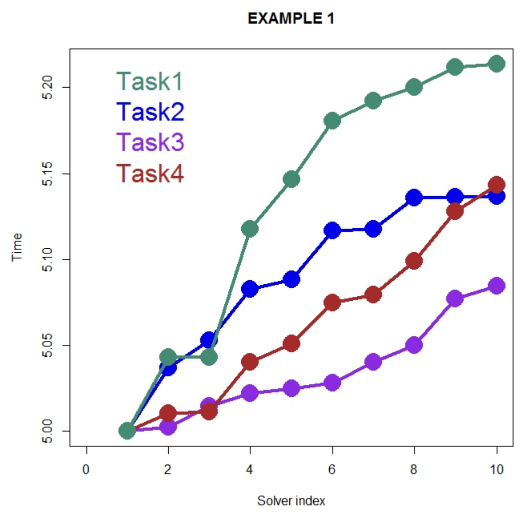
Simply select the nearest location to you, then type:

```
library(calibrate)
```

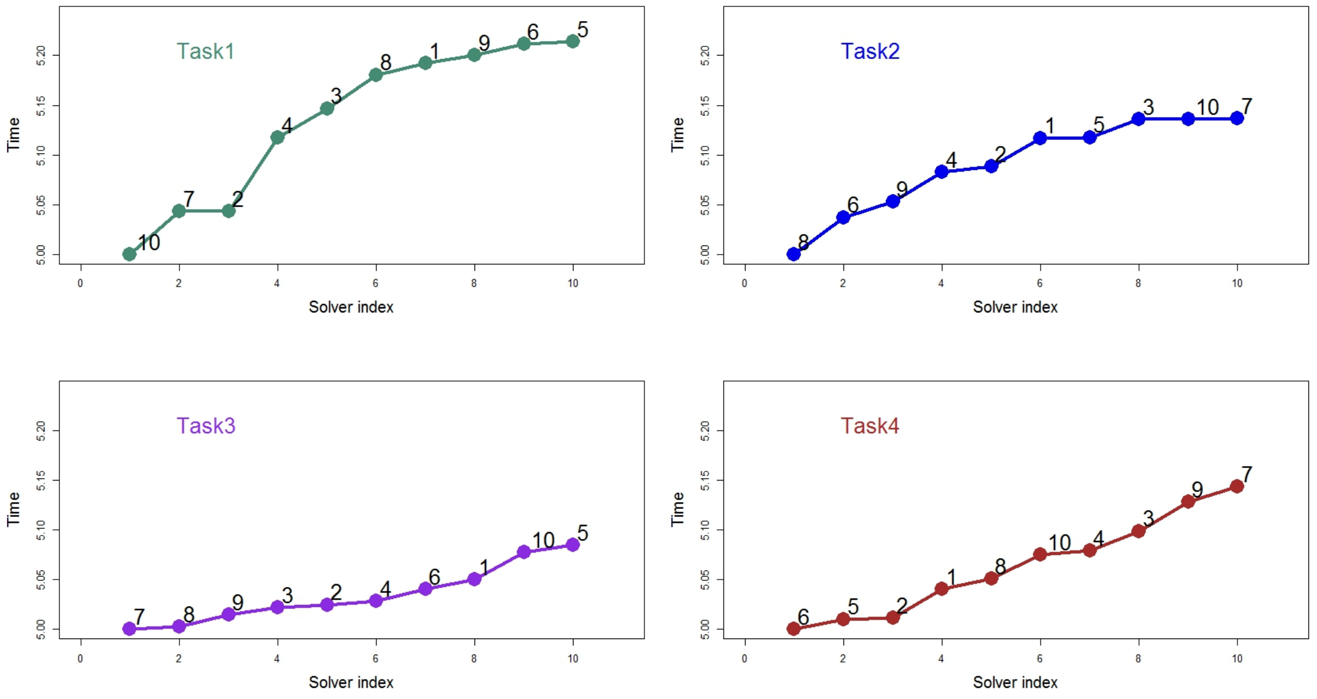
At this point you need to run the R code in the file *bnbdacode.r* so that the functions *CREATEdiffusion* and *MCMCanalysis* will be made available. These functions will be discussed in more detail later on.

4 Example 1

This example involves 10 individuals and 4 tasks. This dataset is analysed using the hazard function specified in 2 (i.e. without random effects). The data are shown in Figure 1 below.



(a)



(b)

Figure 1: Plots of data where each line represents a unique task. (a) shows the solve times for all the diffusions and (b) consists of 4 plots, each illustrating the solve times for the 10 individuals for a given per task (diffusion). The id of each individual is indicated on the plots by a numerical number.

For simplicity, the social network for this dataset contains interactions which are all set at 1, although in reality more structured networks are expected. The R code to do so is shown below:

```
Social_Network = matrix(1,nrow=10,ncol=10).
```

The solve orders for each diffusion (which represent a unique task) should be stored in a matrix where each column represents a diffusion and each row represents the solve index as shown below. In the matrix below, the first individual that solved the first task (column 1) is individual 10, the last individual to solve the fourth task (column 4) is individual 7.

$$\begin{pmatrix} 10 & 8 & 7 & 6 \\ 7 & 6 & 8 & 5 \\ 2 & 9 & 9 & 2 \\ 4 & 4 & 3 & 1 \\ 3 & 2 & 2 & 8 \\ 8 & 1 & 4 & 10 \\ 1 & 5 & 6 & 4 \\ 9 & 3 & 1 & 3 \\ 6 & 10 & 10 & 9 \\ 5 & 7 & 5 & 7 \end{pmatrix}$$

The R code to construct the matrix is given below.

```
SolveOrders = matrix(c(10,8,7,6,7,6,8,5,2,9,9,2,4,4,3,1,3,2,2,8,8,1,4,10,
1,5,6,4,9,3,1,3,6,10,10,9,5,7,5,7),nrow=10,ncol=4,byrow=T)
```

Note that the R code constructs a matrix with 10 rows and 4 columns. The expression

```
byrow=T
```

specifies that the matrix should be constructed by filling in row by row from the values:

```
 (10,8,7,6,7,6,8,5,2,9,9,2,4,4,3,1,3,2,2,8,8,1,4,10,1,5,6,4,9,3,1,3,6,10,10,9,5,7,5,7).
```

Similarly, the solve times should be stored in a matrix where each column represents a diffusion and each row represents the solve index. This is expressed as:

$$\begin{pmatrix} 5.000000 & 5.000000 & 5.000000 & 5.000000 \\ 5.043175 & 5.037176 & 5.002482 & 5.010308 \\ 5.043215 & 5.052941 & 5.014653 & 5.011304 \\ 5.117502 & 5.082576 & 5.022125 & 5.040178 \\ 5.146615 & 5.088089 & 5.024597 & 5.051070 \\ 5.180519 & 5.116860 & 5.028307 & 5.074835 \\ 5.192106 & 5.117648 & 5.040428 & 5.079254 \\ 5.200314 & 5.135739 & 5.049815 & 5.098847 \\ 5.211655 & 5.136084 & 5.077242 & 5.128133 \\ 5.213930 & 5.136748 & 5.084685 & 5.143329. \end{pmatrix}$$

The R code to do this is:

```
SolveTimes = matrix(c(5.000000, 5.000000, 5.000000, 5.000000,5.043175,
 5.037176, 5.002482, 5.010308,5.043215, 5.052941, 5.014653, 5.011304,
5.117502, 5.082576, 5.022125, 5.040178,5.146615, 5.088089, 5.024597,
5.051070, 5.180519, 5.116860, 5.028307, 5.074835,5.192106, 5.117648,
5.040428, 5.079254,5.200314, 5.135739, 5.049815, 5.098847, 5.211655,
5.136084, 5.077242, 5.128133,5.213930, 5.136748, 5.084685, 5.143329),
nrow=10,ncol=4,byrow=T).
```

The code creates a matrix that has 10 rows representing a unique individual and 4 columns representing a unique diffusion (task). A diffusion object can be created using the function *CREATEdiffusion*. This function formats the data and the output is a data object which can be analysed using the function *MCMCanalysis*. The *CREATEdiffusion* function also outputs a plot of the data. To create an diffusion object using the data described above, the following R code is used:

```
diff = CREATEdiffusion(SolveTimes,Social_Network,SolveOrders).
```

The analysis is run by using the following R code:

```
analysis = MCMCanalysis(diff).
```

This *MCMCanalysis* function runs the MCMC simulation and allows the user to input the number of iterations required and also to tune the analysis. There are default values for the number of iterations and tuning values. Tuning involves specifying the proposal distribution function used to propose candidate posterior parameter values. The output contains the log of the mean posterior estimates for both the social effect and baseline rate parameters (and standard deviations) and the simulated values for the each iteration in the simulation. A trace plot, density plot and ACF plot for the social effect parameter are provided. This is shown in Figure 2. In summary, a trace plot is a plot of the posterior parameter values and gives a general indication of how well mixed the Markov chain is. The density plot provides an indication of the spread and center of the posterior parameter values. Finally, the ACF plot provides an indication of whether or not there is autocorrelation involved.

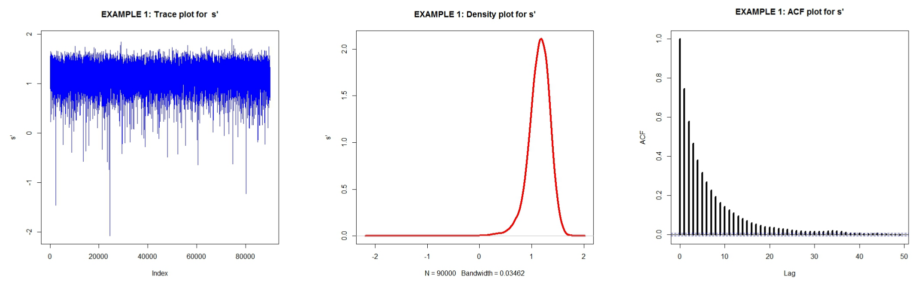


Figure 2: Plots for social effect parameter: Trace plot, Density plot and ACF plot

The output can be displayed on the screen by then typing

```
analysis
```

The mean posterior estimates (in natural logarithms) for the social effect and baseline rate parameters can be obtained by:

```
analysis@parameters # Rcode
```

The resulting output is:

```
[1]  1.147023 -4.171460 # R output for both parameters
```

For just the social effect parameter, type:

```
analysis@parameters[[1]]# R code for the social effect parameter
```

The resulting output is:

```
[1] 1.147023.          # R output for only the social effect parameter
```

The simulated values at each iteration can be found by typing:

```
analysis@coda.
```

To save the simulated values to a file named *output.csv* the following R code is used:

```
write.csv(analysis@coda,"output.csv").
```

Alternatively, the analysis can be run by replacing the social network with other ones such as:

```
Social_Network = matrix(runif(100,0.2,0.5),nrow=10,ncol=10)
```

or

```
Social_Network = matrix(0,nrow=10,ncol=10)
```

to observe the effect of a randomly derived social network with values from 0.2 to 0.5 or a social network containing all zeros, respectively. The expression

```
runif(100,0.2,0.5)
```

obtains 100 randomly generated values which are Uniformly distributed between 0.2 and 0.5. See the R script provided for more details on the output files provided for the analysis of the data using different social networks.

5 Example 2

This example involves 5 individuals and 5 diffusions (tasks) and unlike the first example, here we incorporate random effects using the hazard function described in Equation 3. The social network used contains all zeros. The data was simulated (Gillespie algorithm) with the social effect parameter set at 0 and that of the baseline rate parameter set at 1. In this example, we analyse the data and estimate the random effect of each individual. The social effect and baseline rate parameters are treated as constants (using the values used to simulate the data).

The R code used to construct the social network is:

```
Aij1 =  matrix(0,ncol=10,nrow=10).
```

The R code used to construct the matrix containing the solve times is:

```
diffusions = matrix(c(5.000000,  5.000000, 5.000000, 5.000000,
5.000000,5.002970,  5.066729, 5.000249, 5.002493, 5.000063,
6.791319,  5.187937, 5.026370, 8.663187, 5.189498,
9.214229,  6.911657, 7.443854, 9.749797, 7.437023,
23.008068, 14.075366, 8.870171, 9.855659, 8.833192
), nrow=5,ncol=5,byrow=TRUE).
```

The R code used to create the matrix containing the solve order is:

```
diffusionid= matrix(c(2,3,5,5,4,3,2,3,3,3,5,5,2,2,5,
 1,1,1,1,1,4,4,4,4,2), nrow=5,ncol=5,byrow=TRUE).
```

To run the analysis, run the R code in the file *nbdaFRAILTY*. The posterior summaries and plots of the the social effect parameter will be displayed on the screen. To save this output to a file named 'frailty.csv' type

```
write.csv(Jumbo,"frailty.csv").
```

The term 'Jumbo' refers to the collection of simulated values generated from the analysis. This is an arbitrary name used in the code. An output file containing the results for this analysis (where the simulation involved contained 100000 iterations) is provided as *"Eimanyits.csv"*. Before reading in this file it is important that your working directory is changed to the folder where the output file is stored. an example of the R code that you could use do to this is:

```
setwd("C:/Users/user/Desktop/yourfoldername").
```

The posterior summaries (for the random effects contributed by each individual) from this output file can be obtained by running the following R code:

```
analysis = read.csv("Eimanyits.csv") # reading in the output file

#Posterior means
mean(analysis[10001:100000,15])  # individual 1
mean(analysis[10001:100000,16])  # individual 2
mean(analysis[10001:100000,17])  # individual 3
mean(analysis[10001:100000,18])  # individual 4
mean(analysis[10001:100000,19])  # individual 5

#95% symmetric credible intervals
```

```
quantile(analysis[10001:100000,15],probs=c(0.025,.975))  # individual 1
quantile(analysis[10001:100000,16],probs=c(0.025,.975))  # individual 2
quantile(analysis[10001:100000,17],probs=c(0.025,.975))  # individual 3
quantile(analysis[10001:100000,18],probs=c(0.025,.975))  # individual 4
quantile(analysis[10001:100000,19],probs=c(0.025,.975)).  # individual 5
```

From the results from this analysis it is apparent that individual 3 exerts a relatively large positive on the overall baseline rate of solving. Importantly, the credible interval for this effect does not contain zero. This suggests that we have 95% credibility that the true value for this effect lies within this interval and that the effect is not equal to 0.

After running the code the densities for each random effect are shown on a plot as shown in Figure 3 below:

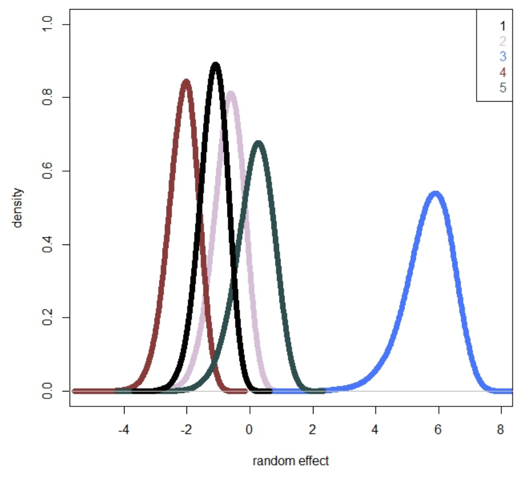


Figure 3: Plot of densities for each random effect

6 Example 3

This third example illustrates methods for model discrimination. Typically there are various competing models which may be plausible for a given set of observations (data). Model discrimination in a Bayesian framework offers a means by which competing models (and the inherent hypotheses) can be compared to each other. Firstly, when a reversible jump Markov chain Monte Carlo (RJMCMC) algorithm is employed, posterior model probabilities can be calculated, thus providing a means by which many models can be compared with each other simultaneously. In addition, Bayes Factors (Kass and Raftery, 1995; Lee, 1989) can be used to summarise the evidence provided in favour of one model (and its accompanying hypothesis) against another candidate model. Kass and Raftery (1995) provide a guide to interpreting Bayes factors as shown in Table 4.

Figure 4: Interpretation of Bayes Factors

Bayes factor	Interpretation
1 – 3.2	Not worth mentioning
3.2 – 10	Substantial
10 – 100	Strong
> 100	Decisive

This section provides an illustration of the analysis of two competing models for a given dataset. Consider the data described in Example 1. For this dataset, the two competing model hypotheses are represented by models 1 and 2 described in Section 2. Recall that model 1 is the null model (containing **only** the baseline rate paramter, λ_0) with the hazard function described in Equation 1. In contrast, model 2 contains two parameters: $s^{'}$ and λ_0 (See Equation 2). To conduct this analysis run the code in the file *demoRJ.r*. Save the analysis to a file entitled *comparemodels.csv* by typing:

```
write.csv(Jumbo,"comparemodels.csv")
```

The Bayes factor in favour of model 2 against model 1 will be printed on the screen. To do this calculation manually, type:

```
BF = summary(as.factor(Jumbo[201:2000,6]))
BF[[2]]/BF[[1]]
```

The simulation is run for 2000 iterations (for demonstration purposes), as a result the code above is written such that 10% of the iterations are treated as 'burn in'. Typically these simulations should be run for a longer period (for example, 10,0000 iterations, 100,000 iterations or even 1000,000 iterations) so that the chain explores both model and parameter space adequately. Guidance to the interpretation of the Bayes Factor obtained from the analysis can be found in Table 4. Figure 5 illustrates the transitions made between model states during a typical simulation for similar models. The grey line represents the transition between model states and the coloured lines give a rough indication of the model state at a given iteration.

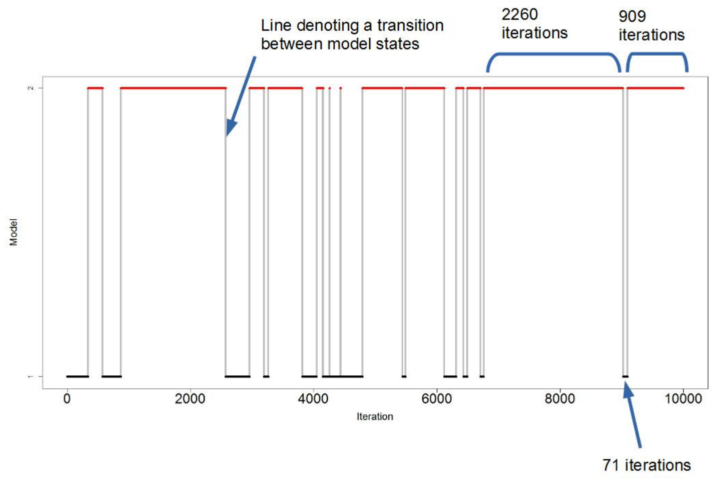


Figure 5: Plot showing the transitions between two model states for 10000 iterations. The grey line denotes a model transition and the coloured lines denote the model state at a given iteration. Model 2 is represented by a red coloured line.

7 Ongoing software development

Ongoing software development is aimed at providing software to handle more complicated datasets. The software would be able to handle data which contains censored individuals, other covariates (individual level variables) and model discrimination between more than 2 models. In addition, the software would allow the user to run models that involve not only individual level random effects, but shared frailties. Finally, flexiblity will be improved by providing the users the option to perform prior sensitiviy tests. The sample code is written such that Uniform priors are specified for the social effect and baseline rate parameters. We aim to allow the user to alter the priors specified by altering the hyper parameters of the Uniform distribution.

A comprehensive description of network based diffusion analysis is provided by Hoppitt et al. (2010) and Hoppitt and Laland (2013).

8 Contact

Contact Glenna Nightingale at glenna.evans@gmail.com for any queries related to using this tutorial

References

Hoppitt, W., N. J. Boogert, and K. N. Laland (2010). Detecting social transmission in networks. *Journal of Theoretical Biology*.

Hoppitt, W. and K. N. Laland (2013). *Social Learning:An Introduction to Mechanisms, Methods, and Models*. Princeton University Press.

Kass, R. E. and A. E. Raftery (1995). Bayes factors. *Journal of the American Statistical Association 90*, 773–795.

Lee, P. (1989). *Bayesian Statistics: An Introduction*. UK: Hodder Arnold.

Nightingale, Boogert, Hoppitt, and Laland (In Press). *Bayesian Network Based Diffusion Analysis - A case study using simulated data*.