Supporting Information

This is the supporting information of the manuscript "*A multinomial network method for the analysis of mate choice and assortative mating in spatially structured populations*". Here you will find additional results from the analyses of simulated data, and tutorials on how to calculate copulation probabilities using the multinomial network model, and how to optimize the model parameters in *R* in two different ways, by: (1) maximum likelihood optimization using the package *bbmle* (Bolker & R Core Team 2014), and (2) Bayesian Markov chain Monte Carlo using the package *rjags* (Plummer 2015). This document is divided in seven sections: in section 1, we present additional results, in sections 2, 5, 6 and 7, we present tutorials using simulated data, which is available as a series of *.txt* files. Finally, in sections 3 and 4, we present tutorials using the blue tit extra-pair copulation data available in the package *expp* (Valcu & Schlicht 2014).

1. Additional results: males with limited copulations

One of the assumptions of the multinomial network model is that males can mate an unlimited number of times, so that a male is always available to a female that would choose to mate with him. To investigate the effect of violating this assumption on the performance of the multinomial network model, we ran simulations in which males had a limit in their number of copulations. These simulations worked in the same way as the simulations of directional mate choice presented in the main text, with the difference that each male could mate only up to five times; after which, males become unavailable to females. We ran 50 simulations, and analysed the results as described in the main text, assuming that males could mate unlimitedly. The assumption that males

can mate unlimitedly had little influence on statistical power (Fig. S1A), but made the multinomial model underestimate female choosiness when sample size was small (Fig. S1B).

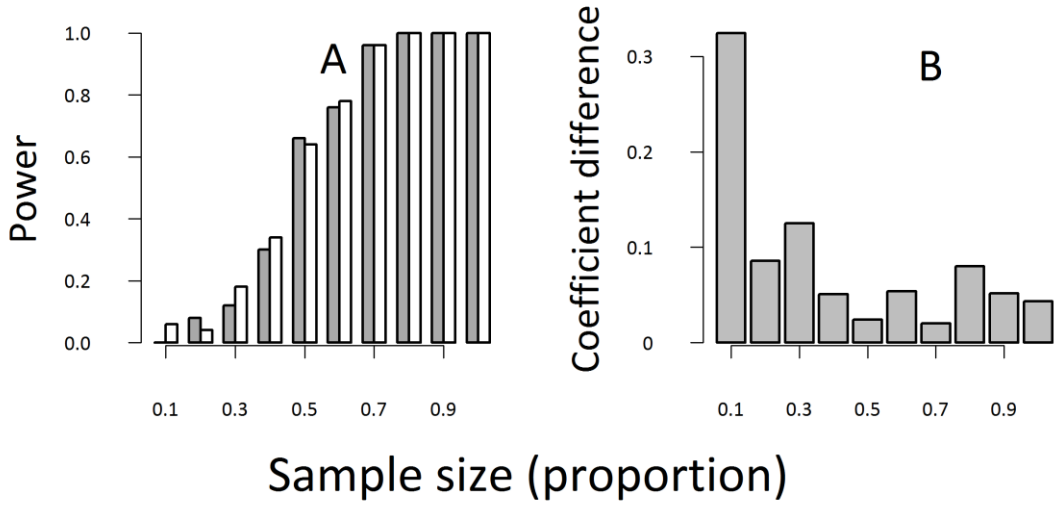


Fig. S1. Result of the analysis of simulations in which males had a limited number of copulations in relation to the proportion of data analysed (total female population = 200). (A) Statistical power of the model. Grey bars: simulations in which males had a limited number of copulations (up to five). White bars: simulations in which males could mate unlimitedly. (B) Difference in model coefficients: $B_{lim} - B_{unlim}$; B_{lim} is the mean coefficient in the analysis of simulations in which males had a limited number of copulations, and B_{unlim} is the mean coefficient from the simulations in which males could mate unlimitedly.

2. Tutorial of the multinomial network model with simulated data

Here we will use a simulated dataset to test the hypotheses that females (1) are more likely to mate with neighbouring males, and (2) prefer males with large trait values. The dataset we will use here comes from the simulation of directional mate choice described in the main text, thus we know that in this scenario both hypotheses are correct. But let us pretend we do not know that, and use the multinomial network model to estimate female preference for large male traits. We will estimate the model parameters using maximum likelihood optimization. If the reader is not familiar with model fitting by maximum likelihood, we suggest the textbook by Bolker (2008).

The first step is to load the data, which are saved in three files:

1. tmales.txt - a file with the information of the males.

```
males = read.table("tmales.txt", sep="\t", dec=".", header=TRUE)
head(males)
##   id          x          y    trait
## 1  1 0.61330978 0.10805031 4.321517
## 2  2 0.03135099 0.25517120 2.893953
## 3  3 0.54606595 0.49578497 5.420178
## 4  4 0.11690116 0.17813385 6.542484
## 5  5 0.12727202 0.09846337 4.691839
## 6  6 0.03660171 0.12933401 4.843570
```

In this file, *x* and *y* are spatial coordinates for each male, and the male's *trait* value.

2. tfemales.txt - a file with the information of the females.

```
females = read.table("tfemales.txt", sep="\t", dec=".", header=TRUE)
head(females)
##   id         x          y
```

```
## 1  1 0.6916669 0.41298186
## 2  2 0.3388266 0.12437187
## 3  3 0.4387549 0.19121923
## 4  4 0.5953950 0.22947917
## 5  5 0.1155520 0.15945149
## 6  6 0.3456921 0.03174804
```

Note that females also have *x* and *y* spatial coordinates, but they do not have trait values.

3. tcouples.txt - a file denoting male-female pairs that copulated.

```
couples = read.table("tcouples.txt", sep="\t", dec=".", header=TRUE)
head(couples)
##   female male
## 1      1    7
## 2      2   59
## 3      3   29
## 4      4   48
## 5      5   66
## 6      6   21
```

This is the edge list of the sexual network, *i.e.*, a list of who mated with whom using male and female IDs (which are also present in the male and female files).

The model we will fit includes as predictor variables the spatial distance, and male trait. Following what we did in the main text, we will log-transform the male trait *z*. Thus, the probability P_{ij} of female *i* mating with male *j* can be written as:

$$P_{ij} = \frac{exp\left(-A \cdot s_{ij} + B \cdot log(z_j)\right)}{\sum_{k=1}^{M}[exp(-A \cdot s_{ik} + B \cdot log(z_k))]}.$$

In the equation above, s_{ij} is the spatial distance between female i and male j, z_j is the trait value of male j, s_{ik} are the distances between the female i and all M males in the mating pool, and z_k are the values of the traits of all males in the mating pool. A and B are the model parameters. The parameter A quantifies the importance of spatial limitation on mate choice, and the parameter B quantifies female choosiness for values of z.

The easiest way to calculate the probabilities for all copulations is by using vectorised operations, and matrix manipulations. However, it is easier to understand the whole process by first calculating the probability of a single copulation. So, let us use the first copulation in the **couples** data.frame and assume parameter values of A = 30 and B = 2.

Let us look at the data:

```
couples[1,] #so we have female 1 and male 7
##   female male
## 1      1    7
```

As you can see, we are dealing with female 1 and male 7.

```
females[1,]
##    wid id         x         y
## 1 2647  1 0.6916669 0.4129819         4
males[7,]
##    wid id         x         y     trait
## 7 2647  7 0.6208285 0.3538615  4.417459
```

To continue this example, we need to calculate the spatial distances between female 1 and all males in the mating pool. So, it is time to introduce a function that calculates Euclidean distances between matrices of Cartesian (x and y) coordinates:

```
euclid = function(m1, m2=m1)
{
  return(sqrt((outer(m1[,1],m2[,1],"-")^2)+(outer(m1[,2],m2[,2],"-")^2)))
}
```

With this function, we can calculate all the s_{ik} distances:

```
sik = euclid(females[1, c("x","y")], males[, c("x","y")])
round(sik,3)[1:10]
##  [1] 0.315 0.679 0.167 0.621 0.646 0.714 0.092 0.369 0.310 0.527
```

Now that we have the distances, the z_k values can be easily recovered:

```
zk = males$trait
round(zk,3)[1:10]
##  [1] 4.322 2.894 5.420 6.542 4.692 4.844 4.417 3.143 3.077 2.131
```

With this we can calculate $exp(-A \cdot s_{ij} + B \cdot log(z_j))$:

```
A = 30
B = 2
exp(-A*sik[7]+B*log(zk[7]))
## [1] 1.225199
```

Since all mathematical operators in *R* are vectorised, we can calculate $\sum_{k=1}^{M}[exp(-A \cdot s_{ik} + B \cdot log(z_k))]$ with a similar command:

```
sum(exp(-A*sik+B*log(zk)))
## [1] 6.360233
```

Now it is easy to calculate the value $P_{1,7}$:

```
(Pij = exp(-A*sik[7]+B*log(zk[7]))/sum(exp(-A*sik+B*log(zk))))
## [1] 0.1926343
```

Another way to calculate P_{ij}, would be to build a vector with all values of $exp(-A \cdot s_{ik} + B \cdot log(z_k))$and then calculate P_{ij} using it:

```
vectorik = exp(-A*sik+B*log(zk))
(Pij = vectorik[7]/sum(vectorik))
## [1] 0.1926343
```

Note that this procedure leads to the same value obtained before.

Now we need to do the same for all 100 copulations in the **couples** data.frame. If for one female we had to build a vector of mating probabilities, now we will need a matrix, in which each row will work just like the vector we built above. Our first step is to build the spatial matrix using the function *euclid()*:

```
sikMatrix = euclid(females[, c("x","y")], males[, c("x","y")])
round(sikMatrix[1:6,1:6],2)
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.31 0.68 0.17 0.62 0.65 0.71
## [2,] 0.27 0.33 0.43 0.23 0.21 0.30
## [3,] 0.19 0.41 0.32 0.32 0.33 0.41
## [4,] 0.12 0.56 0.27 0.48 0.49 0.57
## [5,] 0.50 0.13 0.55 0.02 0.06 0.08
## [6,] 0.28 0.39 0.51 0.27 0.23 0.32
```

The traits of the males are still only a vector of values. However, we must arrange the values in the form of a matrix. This matrix will contain the trait values of all males in each row:

```
zkMatrix = traitMatrix = matrix(males$trait, nrow=nrow(couples),
ncol=nrow(males),byrow=TRUE)
round(zkMatrix[1:6,1:6],2)
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 4.32 2.89 5.42 6.54 4.69 4.84
## [2,] 4.32 2.89 5.42 6.54 4.69 4.84
## [3,] 4.32 2.89 5.42 6.54 4.69 4.84
## [4,] 4.32 2.89 5.42 6.54 4.69 4.84
## [5,] 4.32 2.89 5.42 6.54 4.69 4.84
## [6,] 4.32 2.89 5.42 6.54 4.69 4.84
```

Thus, calculating all $exp(-A \cdot s_{ik} + B \cdot log(z_k))$ values is straightforward:

```
pikMatrix = exp(-A*sikMatrix+B*log(zkMatrix))
round(pikMatrix[1:6,1:6],2)
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] 0.00 0.00 0.19 0.00 0.00 0.00
## [2,] 0.00 0.00 0.00 0.05 0.04 0.00
## [3,] 0.06 0.00 0.00 0.00 0.00 0.00
## [4,] 0.47 0.00 0.01 0.00 0.00 0.00
## [5,] 0.00 0.18 0.00 24.40 3.42 1.86
## [6,] 0.00 0.00 0.00 0.01 0.02 0.00
```

Using indexation, we can retrieve from this matrix the values that represent the copulations that actually occurred. We can use the **couples** data.frame as our index:

```
head(couples)
##    wid female male        z
## 1 2647      1    7 4.417459
## 2 2647      2   59 4.575273
```

```
## 3 2647      3   29 6.281996
## 4 2647      4   48 4.821807
## 5 2647      5   66 9.865502
## 6 2647      6   21 4.214258
positions = cbind(couples[,"female"], couples[,"male"])
head(positions)
##      [,1] [,2]
## [1,]    1    7
## [2,]    2   59
## [3,]    3   29
## [4,]    4   48
## [5,]    5   66
## [6,]    6   21
(almostPij = pikMatrix[positions])
##   [1]  1.2251991  2.7473038  3.5153376  1.3741151  4.9822296  1.7692152
##   [7] 12.9957135  4.0311123  9.9076060  3.1192758  4.7139666  8.6563941
##  [13]  2.8595811 13.0394459  2.1275493  4.2196773  4.5595011  3.2182406
##  [19]  3.4303265  4.7636543  0.9990141  2.5403508  2.4553433  3.0574719
##  [25]  0.9871570  1.6231821  9.6647502  1.1266421  4.7130262  1.4294676
##  [31]  4.2470868  2.4100659  1.3195450  6.6378768  8.0886447  2.8758104
##  [37]  1.3154812 14.1936510 10.0043944  3.4841749  2.1752931  2.5943382
##  [43]  0.7981254  1.7188815  3.5973456  1.0447054  2.6992733  5.4327450
##  [49]  1.1508328  2.4562650  2.5734774  1.4608254  3.2424094  3.4500471
##  [55]  1.2546881  2.6332872  4.8646127  1.3316130  0.7541931  3.8438030
##  [61]  3.1661562  3.6355956 14.9904872  4.8193897  6.3963573  4.6548870
##  [67]  7.6882043  1.0033143  3.1973841 12.6569914  1.0306966  1.3936103
##  [73]  0.9185597  0.5238729  2.5522522  0.7735261  2.3756105  6.2837321
##  [79]  2.6885401  0.9643446  2.3976458 13.0604876  5.4656307  1.9523773
##  [85]  2.6973535  1.3037167  0.5191099  4.3714207 11.6794788 32.2975023
##  [91]  2.5521133  4.9389950  1.7311579  0.7967567  3.4220446  3.3482816
##  [97]  0.8302683  0.8584654  5.0731570  9.6682533
```

As you can see, the object **almostPij** is filled with values greater than one. To convert these values into probabilities, we must divide each value by the sum of the row from which it came from. These sums can be calculated using the *apply()* function:

```
pikSums = apply(pikMatrix,1,sum)
Pij = almostPij/pikSums
```

Alternatively, we can calculate the P_{ij} values with fewer commands:

```
pikMatrix = exp(-A*sikMatrix)*(zkMatrix^B)
positions = cbind(couples[,"female"], couples[,"male"])
(Pij = pikMatrix[positions]/apply(pikMatrix,1,sum))
##   [1] 0.19263429 0.24415094 0.31504209 0.26077126 0.11231963 0.25901715
##   [7] 0.33971830 0.20539876 0.30728727 0.51504517 0.18489917 0.32700436
##  [13] 0.15497012 0.30656655 0.10604164 0.12120410 0.10733901 0.36769808
##  [19] 0.17223845 0.18461556 0.07687760 0.08911502 0.09283111 0.05658513
##  [25] 0.05784575 0.51008600 0.72463613 0.05005829 0.48076771 0.23619879
##  [31] 0.19544018 0.23058113 0.16095793 0.44748977 0.21385911 0.11575955
##  [37] 0.09603008 0.31442101 0.65545577 0.38639096 0.08909005 0.12527440
##  [43] 0.02947112 0.04043135 0.11825758 0.17498228 0.33813887 0.18457009
##  [49] 0.19475782 0.24880582 0.12447465 0.17835525 0.07288797 0.21026959
##  [55] 0.16825303 0.32293294 0.22450453 0.02214967 0.06043860 0.10384960
##  [61] 0.28253367 0.34516751 0.34244775 0.27616088 0.15186929 0.24950219
##  [67] 0.38980614 0.04053801 0.05169385 0.36310293 0.12451539 0.03377290
##  [73] 0.04315512 0.01467642 0.49028567 0.13464362 0.28852361 0.27125362
##  [79] 0.14683500 0.14780411 0.15617482 0.49165454 0.35480588 0.04450449
##  [85] 0.37473064 0.17611629 0.02114285 0.18071569 0.51739602 0.53822757
##  [91] 0.07674842 0.08045837 0.30621004 0.01889029 0.25381708 0.21985125
##  [97] 0.08197074 0.03327845 0.13651250 0.14541491
```

The negative log-likelihood of the model (considering the parameter values of A = 30 and B = 2) is:

```
-sum(log(Pij))
## [1] 183.4593
```

Now, we can calculate the negative log-likelihood (which we will call just log-likelihood from now on) of our model, or at least, we can calculate the likelihood given certain parameter values. We can use the optimization function in the package *bbmle* to find the most likely values of our parameters. We will use the *mle2()* optimization function. The main parameters of this function are:

- *minuslogl* - which is a function that returns a negative likelihood value;
- *start* - which is a named list of starting parameter values for the optimizer;
- *data* - a named list with the objects containing the data.

What are the data needed by our function? Previously, we stored all the data in the objects **sikMatrix**, **zkMatrix**, and **positions**. With all this, we can write our log-likelihood function, which we will call *LL.copula()*. The function will have five parameters: *A* and *B*, which are the model parameters, *sik* and *zk*, which are the matrices of spatial distances and male traits, respectively, and *pos*, which is the indexing matrix.

```
LL.copula = function(A, B, sik, zk, pos)
{
  pik = exp(-A * sik) * (zk^B)
  pij = pik[pos]/apply(pik, 1, sum)
  return(-sum(log(pij)))
}
```

Finally, we can fit the model. To do so, we need to load the package *bbmle* and call the *mle2()* function. As we said before, we need starting values. Because we suppose both *A* and *B* to be positive, we will use starting values equal to 1.

```
library(bbmle)
fitted = mle2(minuslogl = LL.copula, start = list(A=1, B=1),
```

```
              data = list(sik=sikMatrix, zk=zkMatrix, pos=positions))
fitted
## Call:
## mle2(minuslogl = LL.copula, start = list(A = 1, B = 1), data = list(sik = sikMatrix,
##     zk = zkMatrix, pos = positions))
## Coefficients:
##          A          B
## 34.256275   2.053015
## Log-likelihood: -182.26
```

Model fitted!

As expected, the value of *B* is positive (2.05), indicating that females prefer males with large traits. Now we need to calculate 95% confidence intervals, and likelihood intervals to be able to say something about these values. The confidence intervals can be easily obtained using the *confint()* function:

```
 (confint95 = confint(fitted))
##        2.5 %     97.5 %
## A  40.354007  28.889100
## B   1.349098   2.805799
```

The likelihood intervals can be obtained using the *profile()* function. The two main parameters of this function are the fitted object and *zmax*. As you can see in the help file of the function *profile.mle2()*, the parameter *zmax* is the square root of the deviance difference of the likelihood interval. The deviance, in turn, is twice the log-likelihood. If we want a log-likelihood interval with a maximum difference of two units, we must set *zmax* to the square root of two times two, which is two!

```
(profile2 = profile(fitted, zmax=2))
## Likelihood profile:
```

```
## $A
##             z par.vals.A par.vals.B
## 1  -2.2315287 41.260358   2.260653
## 2  -1.8810289 40.093011   2.226024
## 3  -1.5225441 38.925664   2.191415
## 4  -1.1556622 37.758317   2.156830
## 5  -0.7799387 36.590970   2.122245
## 6  -0.3948921 35.423623   2.087646
## 7   0.0000000 34.256275   2.053015
## 8   0.4053057 33.088928   2.018337
## 9   0.8216461 31.921581   1.983587
## 10  1.2497013 30.754234   1.948741
## 11  1.6902192 29.586887   1.913757
## 12  2.1440257 28.419540   1.878608
## $B
##             z par.vals.A par.vals.B
## 1  -2.0697321 33.118251   1.310916
## 2  -1.6444932 33.310258   1.459336
## 3  -1.2249422 33.519912   1.607756
## 4  -0.8110445 33.747375   1.756176
## 5  -0.4027509 33.992795   1.904595
## 6   0.0000000 34.256275   2.053015
## 7   0.3972732 34.537881   2.201435
## 8   0.7891474 34.837573   2.349855
## 9   1.1757007 35.155293   2.498275
## 10  1.5570158 35.490870   2.646695
## 11  1.9331769 35.844065   2.795114
## 12  2.3042696 36.214565   2.943534
```

If this dataset had come from real data, the analysis would provide strong evidence that females in the population prefer males with large trait values and usually mate with neighbouring males.

3. Blue tit extra-pair copulation data

In this example, we are going to use the blue tit data from Schlicht *et al.* (2015) available in package *expp* (Valcu & Schlicht 2014). This is a dataset of extra-pair copulations (EPC) in which it is hypothesized that females prefer to mate with older, larger males. Thus, the model we are going to fit will include the spatial distance between male and female nest, male age and male tarsus length (a proxy of male size).

3.1. Loading the blue tit data

The dataset includes data from many years, but we will use the year with the largest sample size, 2003. As in any other year, data from 2003 include repeated EPCs by the same female. Firstly, we will ignore this, but in section 3 we will improve the model and include a female identity random effect . The first step is to load the data, which are available in the *R* package *expp*. The data are in the objects **bluetit_breeding** and **bluetit_epp**:

```
library(expp)
data(bluetit_breeding)
str(bluetit_breeding)
## 'data.frame':    1025 obs. of  10 variables:
##  $ year_      : int  2007 2007 2007 2007 2007 2007 2007 2007 2007 2007 ...
##  $ id         : int  13 170 129 173 111 178 182 276 167 23 ...
##  $ x          : num  4417656 4417608 4417414 4417477 4417785 ...
##  $ y          : num  5334188 5334576 5334479 5334591 5334369 ...
##  $ female     : chr  NA NA NA "f161" ...
##  $ male       : chr  NA NA "m245" "m173" ...
##  $ layingDate : int  100 105 106 103 104 101 100 106 104 103 ...
##  $ male_age   : chr  "adult" "adult" "adult" "adult" ...
##  $ male_tarsus: num  NA NA NA NA 16.5 ...
##  $ study_area : chr  "Westerholz" "Westerholz" "Westerholz" "Westerholz" ...
data(bluetit_epp)
str(bluetit_epp)
## 'data.frame':    425 obs. of  3 variables:
```

```
##  $ year_: int  2001 2001 2003 2001 2001 2002 2002 2002 2001 2001 ...
##  $ male: chr  "m525" "m515" "m406" "m7" ...
##  $ female: chr  "f2" "f4" "f4" "f5" ...
```

To make things easier, we will split the data of males and females into different data.frames. Moreover, we will save the EPC data in an object called **epp** (for extra-pair paternity), and the information about the social couples in the data.frame **social**. We will select only the columns that we need.

```
epp = bluetit_epp[bluetit_epp$year == 2003,]
males = bluetit_breeding[bluetit_breeding$year == 2003,
                         c("x", "y", "male", "male_age", "male_tarsus")]
males$binary_age = ifelse(males$male_age=="adult", yes=1, no=0)
females = bluetit_breeding[bluetit_breeding$year == 2003,
                         c("x", "y", "female", "male")]
social = unique(data.frame(female=bluetit_breeding$female[bluetit_breeding$year ==
2003],male=bluetit_breeding$male[bluetit_breeding$year == 2003], stringsAsFactors =
FALSE))
social = social[complete.cases(social),]
```

Some males and females are duplicated in the male and female data.frames because they had more than one nest in the same breeding season. For simplicity, we will calculate mean coordinates for these individuals. We will use the function *aggregate()* to calculate these means, and aggregate all the data in the same object. This is one of the reasons why we converted male ages to a binary value (below we will need this binary variable for the likelihood calculations).

```
females = aggregate(x = females[,c("x","y")], list(females$female), mean)
colnames(females)[1] = "id"
males = aggregate(x = males[,c("x","y","male_tarsus","binary_age")], list(males$male),
mean)
```

```
colnames(males)[1] = "id"
```

Now, we need to remove the males with incomplete information, and EPCs with these males from the dataset. Also, we only need data from females that performed EPCs and their social associations:

```
#keeping only the males with complete data
males = males[complete.cases(males),]
#keeping only the EPCs with these males
epp = epp[epp$male %in% males$id,]
females = females[females$id %in% epp$female,]
social = social[social$female %in% females$id & social$male %in% males$id,]
```

As before, we will arrange all our data in matrices. Let us start with the spatial matrix. In this dataset, each individual is assigned to a nest, and each nest has an x and y coordinates describing its spatial position. Both the male and the female data.frames contain x and y coordinates. As we mentioned before, some females have more than one EPC, whereas some females did not mate extra-pair. Thus, after we calculate the matrix of spatial distances, we need to adjust it so that each row represents one EPC and each cell represents the distance between the female that performed that EPC and all other males in the population. We can name the rows and columns of the distance matrix according to the female and male IDs with some indexing, as we show below:

```
#Spatial matrix
sik03 = euclid(females[,c("x","y")], males[,c("x","y")])
rownames(sik03) = females$id
colnames(sik03) = males$id
sik03[as.matrix(social)] = NA
sik03 = sik03[epp$female,]
rownames(sik03) = 1:nrow(epp)#adjusting rownames to avoid duplicates
```

```
sik03 = sik03/100 #just putting distance values in a scale easier to interpret
```

In addition to the adjustments we explained above, we also replaced all values that refer to the social partners of the females with NAs (after all, we are working with EPCs). With the distance matrix in the object **sik03**, we will create the trait matrices of males as we did in the previous example (see section 1). In this example, we have two male traits, one continuous (tarsus length), and one categorical (male age: juvenile or adult). As explained in the main text, categorical variables need to be transformed to binary values, thus we consider that adults are ones, and juveniles are zeroes.

```
#Male tarsus length matrix
tarsus03 = matrix(males$male_tarsus, nrow=nrow(epp), ncol=nrow(males),byrow=TRUE)
rownames(tarsus03) = 1:nrow(epp)
colnames(tarsus03) = males$id
#Male age matrix
age03 = matrix(males$binary_age, nrow=nrow(epp), ncol=nrow(males), byrow=TRUE)
rownames(age03) = 1:nrow(epp)
colnames(age03) = males$id
```

Before we continue, we need to assemble a matrix with the positions in the matrix that refer to the data of extra-pair partners. The female positions are simply a sequence of numbers, and the male positions are the male IDs in the *epp* data.frame:

```
positions03 = cbind(1:nrow(epp), epp$male)
```

3.2. Writing and fitting the model

Now that the data are ready, we can write our log-likelihood function as we did before. This model includes distance, tarsus length, and age as predictors. The hypotheses are

that females (1) will be more likely to mate with neighbouring males and (2) will prefer large, adult males. The equation of the model is:

$$P_{ij} = \frac{\exp(-A \cdot s_{ij} + B \cdot t_j + C \cdot g_j)}{\sum_{k=1}^{M}[\exp(-A \cdot s_{ik} + B \cdot t_k + C \cdot g_k)]},$$

where s_{ik} represents the distances between the female and each male in the mating pool, t_k is the tarsus length of males, and g_k is the males' binary ages.

```
LL.EPC = function(A, B, C, sik, tarsus, age, pos)
{
  pik = exp(-A*sik + B*tarsus + C*age)
  pij = pik[pos]/apply(pik, 1, sum, na.rm=TRUE)
  return(-sum(log(pij)))
}
```

As before, we use the *mle2()* function to do the model fitting:

```
fittedEPC = mle2(minuslogl = LL.EPC, start = list(A=1, B=0, C=0),
              data = list(sik=sik03, tarsus=tarsus03, age=age03, pos=positions03))
fittedEPC
##
## Call:
## mle2(minuslogl = LL.EPC, start = list(A = 1, B = 0, C = 0),
##     data = list(sik = sik03, tarsus = tarsus03, age = age03,
##         pos = positions03))
##
## Coefficients:
##           A           B           C
##  1.35813727 -0.05966734  1.36607848
##
## Log-likelihood: -121.43
```

As expected, we observe positive values of *A* and *C*. Curiously, we found a slightly negative value of *B*. The interpretation is that females are more likely to mate extra-pair with neighbouring males (*A* = 1.358) and with older males (*C* = 1.366). Moreover, they seem to prefer smaller males (*B* = -0.059). As before, you can check 95% confidence intervals or likelihood intervals using the functions *confint()* and *profile()*.

4. Blue tit extra-copulation data with a random effect

Now we will reanalyse the blue tit data including a female identity random effect in the model. To do this, we will use a Bayesian approach and Markov chain Monte Carlo (MCMC) optimization instead of the maximum likelihood approach we employed thus far. The main reason to do so is that it is simpler to fit this sort of complex model using MCMC than by any alternative method currently available. MCMC calculations are computer intensive, and are not performed by *R*, but by external programs that communicate with *R* via specific packages. In this tutorial, we will employ the program *jags* (freely available from http://mcmc-jags.sourceforge.net/) and the package *rjags* (Plummer 2015). If the reader is not familiar with Bayesian models and MCMC, we suggest the textbook by Gelman *et al.* (2014).

4.1. The model equation

As explained in the main text, random effects can be added as varying intercepts or varying slopes. When the identity of the choosing individuals (*i.e.*, females) is fitted as a random effect, this effect will represent varying slopes. In the model we fitted in section 2, there are three slopes: *A*, representing the spatial effect; *B*, representing mate-choice based on body size, and *C*, representing choice based on age. In this example, we will include a random effect in the form of varying *A* values, so that the model equation will be:

$$P_{ij} = \frac{\exp(-A_i \cdot s_{ij} + B \cdot t_j + C \cdot g_j)}{\sum_{k=1}^{M}[\exp(-A_i \cdot s_{ik} + B \cdot t_k + C \cdot g_k)]}.$$

In this equation, A_i represents the A value for female i, which means we will need to estimate one A value for each female. To do so using MCMC, we must assume that A follows some known distribution, and estimate the parameters describing this distribution, the so-called hyper-parameters. We will assume that A follows a normal distribution, so the hyper-parameters we need to estimate are the mean and standard deviation of A (A_{mean} and A_{sd}, respectively).

4.2. The data to fit the model

Now that we defined our model (at least mathematically), we need to run it. The data we need are essentially the same we used in section 2, but this time we do not need to convert all data to matrix format. This is because in the *BUGS* language used by *jags*, we need to calculate the likelihood of each observation separately (within a *for* loop), thus it is easier to work with the data in a more straightforward way. The data we will need are:

- A matrix of distances between males and females;
- A vector of male sizes (tarsus length);
- A vector of male ages (in binary form);
- A vector of female identities in each copulation;
- A vector of male identities in each copulation;
- The total number of observations (EPCs);
- The total number of females in the sample.

In section 2, we used individual IDs in character form ("m48" for example), but within the *jags* model we need to be able to identify individuals by numbers, so that we can index their information in the matrix and vectors. To do so, we will add a numeric ID to the **males** and **females** data.frames, and then we will use indexation to add two new columns to the **epp** data.frame, which will contain the numeric IDs of the male and female involved in each copulation. We will keep these numeric IDs in columns called **nic** (one for males and other for females).

```
#adding the numeric IDs
males$nic = 1:nrow(males)
females$nic = 1:nrow(females)
#creating named vectors of male and female nics
malenics = males$nic
names(malenics) = males$id
femalenics = females$nic
names(femalenics) = females$id
#Now, using indexation to add the nics to the epp data.frame
epp$malenic = malenics[epp$male]
epp$femalenic = femalenics[epp$female]
```

We can check if the **nics** are correct by looking at some examples:

```
head(epp)
##    year_ male female malenic femalenic
## 3   2003 m406     f4      38        25
## 12  2003  m48    f14      43        19
## 15  2003  m21    f23      22        21
## 16  2003  m44    f23      41        21
## 24  2003 m417    f32      39        22
## 27  2003  m85    f32      69        22
```

Let us look at the first and sixth lines and see if the information is correct:

```
#The first copulation is between male m406 (a.k.a. male 38) and female f4 (a.k.a. female
25)
males[38,]
##      id       x       y male_tarsus binary_age nic
## 38 m406 4814559 5350853       16.85          1  38
females[25,]
##    id       x       y nic
## 52 f4 4814715 5350901  25
#The sixth copulation is between male m85 (a.k.a. male 69) and female f32 (a.k.a. female
22)
males[69,]
##     id       x       y male_tarsus binary_age nic
## 71 m85 4814609 5351224      17.425          0  69
females[22,]
##     id       x       y nic
## 49 f32 4814625 5351174  22
```

Everything is OK with the data and we can move on. The last piece of information we need is the social data. In section 2, we ensured that the social partners of females would not be included in the pool of males available for EPC by adding a NA to their spatial locations. We cannot use this trick with *jags*, so we will make a mating pool matrix, filled with zeroes and ones to store this information. In this matrix, each row represents a female, and each column represents a male. A value of one in this matrix means that the male in column *j* is available as an extra-pair partner to the female in line *i*, whereas a zero means the male is unavailable. In this example, we use this procedure to exclude the social partners, but in other situations, this could be used to deal with other situations in which specific males are unavailable for specific females. This mating pool information will be stored in the object **pool**.

```
#Adding numeric IDs to the social data.frame
```

```
social$malenic = malenics[social$male]
social$femalenic = femalenics[social$female]
#generating the pool matrix
pool = matrix(1, nrow=nrow(females),ncol=nrow(males))
pool[cbind(social$femalenic,social$malenic)] = 0
```

We can now generate the objects containing all the data we will need. The way to send these objects to the MCMC sampler is to keep everything in a list, so let us put everything in a list called **ldata**:

```
ldata = list(
  sik = euclid(females[,c("x","y")], males[,c("x","y")])/100, #distance matrix
  size = males$male_tarsus, #male size
  age = males$binary_age, #male age
  n = nrow(epp), #total n
  nfemales = nrow(females), #number of females
  male = epp$malenic, #male in each EPC
  female = epp$femalenic, #female in each EPC
  pool = pool,
  x = rep(0, nrow(epp)))
```

4.3. The zeroes trick

You may have noticed that we added a vector full of zeroes in our list, called **x** in the code above. This is because of the last thing we need to know before writing our *jags* model: the "zeroes trick". The software *jags* is programmed to deal with likelihood calculations for a fixed set of statistical distributions, and it is not ready to work with a model in which the user wants to customize the likelihood calculation. Thus, we are restricted to using one of the probability functions already present in *jags* in our code. To do this, we must make a probability function already present in *jags* return some value proportional to the probabilities P_{ij} that we will calculate. One way to make a

native *jags* function return exactly the value we want, is to pretend all our observations are zeroes and use the Poisson density function to calculate the likelihoods. The trick works because the probability of observing a zero in a Poisson distribution is equal to *exp(-λ)*, and the log-likelihood *L* of a probability *P* is calculated as *L = -log (P)*. Thus, given an *L* value we can calculate *P* as *P = exp(-L)*. Thus, the probability of observing a value of zero in a Poisson distribution with $\lambda = L$ will be equal to *P*. Using this mathematical trick, we can "fool" *jags*. In our code, we will pretend that all our observations are zeroes, and establish that the probability function associated with these zeroes is a Poisson distribution with $\lambda = L$, in which *L* will be the log-likelihood of our customized probabilities. It is a little convoluted, but you can see that it works in the following example code:

```
#Generating a series or increasing probability values
(probs = (1:10)/11)
##  [1] 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
##  [7] 0.63636364 0.72727273 0.81818182 0.90909091
#Then, we calculate the negative log-likelihood for these probabilities
(L = -log(probs))
##  [1] 2.39789527 1.70474809 1.29928298 1.01160091 0.78845736 0.60613580
##  [7] 0.45198512 0.31845373 0.20067070 0.09531018
#Then we calculate the probability of observing a zero considering that lambda=L
dpois(0, lambda=L)
##  [1] 0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
##  [7] 0.63636364 0.72727273 0.81818182 0.90909091
#We get the same values!
```

4.4. Writing and fitting the model

Now, we can write the model in *BUGS*. There is detailed information about how to do this in the book by Gelman *et al.* (2014), and in the *jags* manual

(http://sourceforge.net/projects/mcmc-jags/files/Manuals/). Essentially, the code needs to do three things:

1. Establish uninformative prior distributions for the parameters and hyper-parameters of the model;
2. Declare that each *A* value follows a certain distribution;
3. Calculate likelihood values for each observation in the data.

The order in which these components are declared does not matter, but if we do it in this order, the code is easier to read. The parameters (and hyper-parameters) of our model are A_{mean}, A_{sd}, *B* and *C*. The hyper-parameter A_{sd} defines the standard deviation of *A* values, however, in *jags*, normal distributions are not defined based on standard deviation, but based on precision, which is the inverse of variance. Thus, the model will work with precision values, and in the last lines of the code we will convert these values into standard deviations.

The model needs to be saved in a separate file to be used during model fitting. The code below specifies the model and writes a *.txt* file containing it. The calculation of the likelihood values is similar to what we did in section 2, but, here, we split the calculation of the numerator and denominator into separate commands to make the code easier to understand.

```
btmodel = "
model
{
  #uninformative priors for parameters B and C
  B ~ dnorm(0.0, 1E-06)
  C ~ dnorm(0.0, 1E-06)
  #uninformative priors for the hyper-parameters
  Amean ~ dnorm(0,1E-06)
  Aprec ~ dgamma(0.01,0.01)
  #Declaring individual A values
```

```
  for(l in 1:nfemales)
  {
    A[l] ~ dnorm(Amean, Aprec)
  }
  #calculating the likelihoods
  for(i in 1:n)
  {
    numerator[i]   <- exp(-A[female[i]]*sik[female[i],male[i]] + B*size[male[i]] +
C*age[male[i]])
    denominator[i] <- sum( exp(1)^(-A[female[i]]*sik[female[i],] + B*size + C*age) *
pool[female[i],] )
    logLike[i]  <- -log(numerator[i]/denominator[i])
    x[i] ~ dpois(logLike[i])
  }
  #Converting precision values to standard deviation values
  Asd <- Aprec^(-0.5)
}"
write(btmodel, "btmodel.txt")
```

Finally, before we use MCMC to find the parameter values, we need to create a *jags* model object. To do this, we will need a list with all the data (which we already created), a text file with our model (also done), and two other things:

1. A character vector specifying the variables we want to record at each MCMC iteration. We will keep only the parameters *B* and *C* and the hyper-parameters of *A*, but if you want to retrieve individual *A* values, add "A" to the **params** vector:

```
params = c("Amean", "Asd", "B", "C") #which parameters to keep
```

2. A list with initial values for the parameters. We are going to run three MCMC chains, so our list needs three sets of initial values:

```
initials = list(list(Amean = 0, Aprec = 0.1, B = 0, C = 0),
                list(Amean = 1, Aprec= 0.01, B = 3, C = 3),
                list(Amean = 10, Aprec = 0.001, B= -3, C = -3))
#The idea is that initial values should vary
```

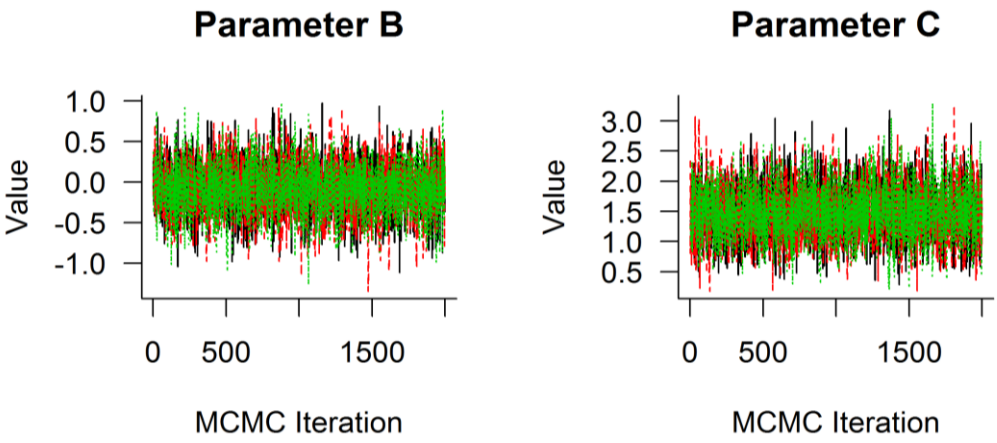
Now, we proceed to create our model and use MCMC to find parameter values:

```
library(rjags)
#These commands may take a few minutes to run
#creating the model
btmodel = jags.model("btmodel.txt",data=ldata, inits=initials, n.chains=3,n.adapt=300)
#running the MCMC
btmcmc = coda.samples(btmodel,params,n.iter=2000,thin=1)
```

4.5. Exploring the results

First, let us verify if the optimization converged adequately. When there is convergence, the plots should look like they have no pattern (*i.e.*, horizontal noise):

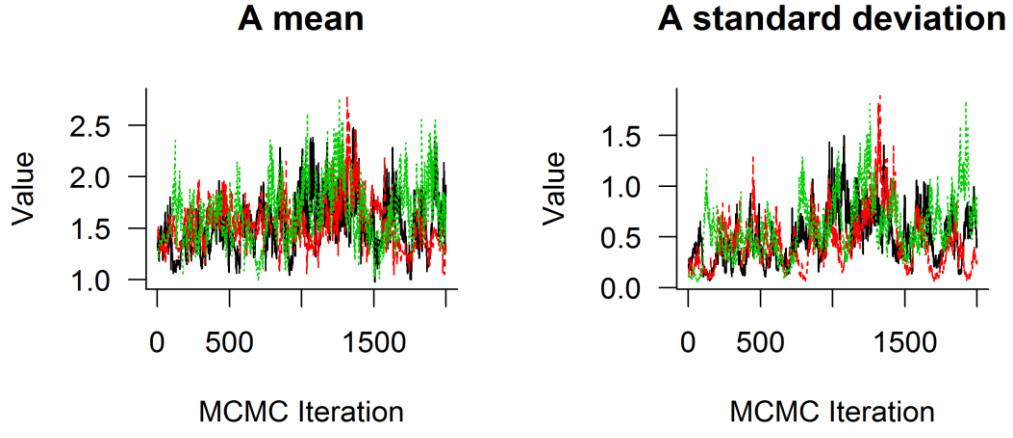
```
par(las=1, bty="l",mfrow=c(1,2))
matplot(cbind(btmcmc[[1]][,"B"],
              btmcmc[[2]][,"B"],
              btmcmc[[3]][,"B"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="Parameter B")
matplot(cbind(btmcmc[[1]][,"C"],
              btmcmc[[2]][,"C"],
              btmcmc[[3]][,"C"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="Parameter C")
```



We can see in the plots above that both parameters *B* and *C* converged adequately.

Now, let us look at the *A* hyper-parameters:

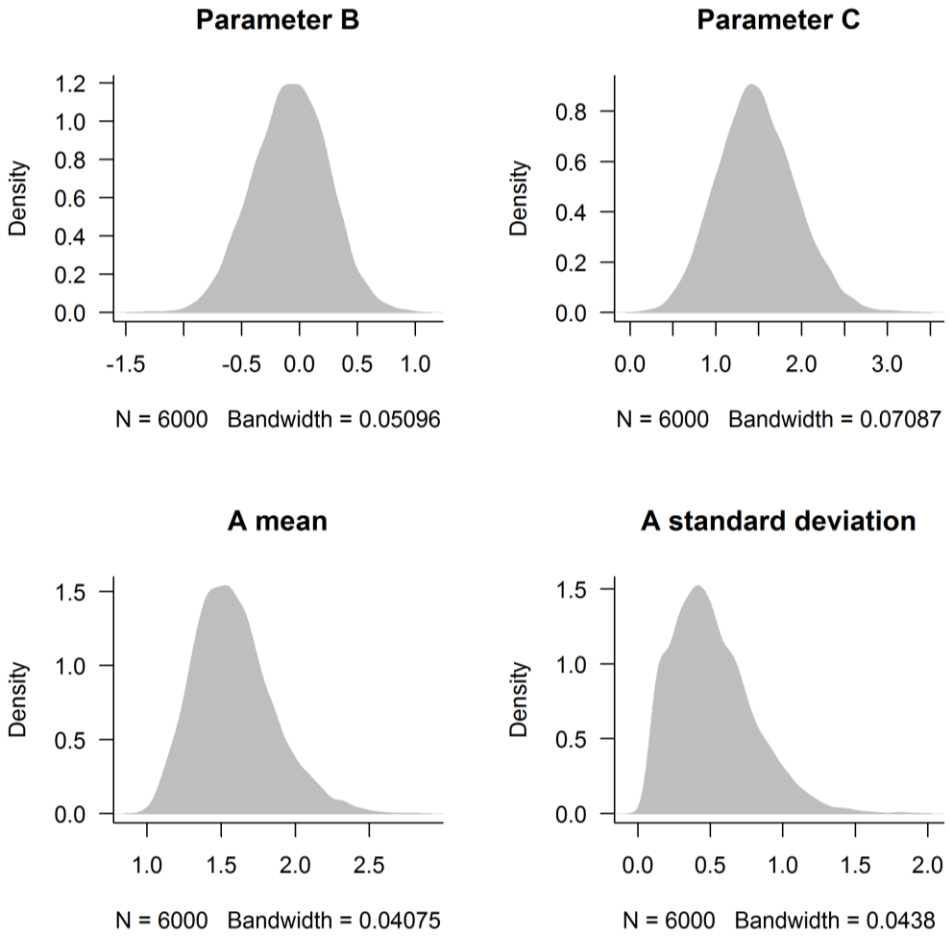
```
par(las=1, bty="l",mfrow=c(1,2))
matplot(cbind(btmcmc[[1]][,"Amean"],
              btmcmc[[2]][,"Amean"],
              btmcmc[[3]][,"Amean"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="A mean")
matplot(cbind(btmcmc[[1]][,"Asd"],
              btmcmc[[2]][,"Asd"],
              btmcmc[[3]][,"Asd"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="A standard deviation")
```



Convergence in these parameters could be better. To improve this, we should increase the number of MCMC iterations and the thinning value. Because this is just a tutorial, let us analyse the results we obtained.

The next step is to look at the posterior distributions and calculate medians and 95% credible intervals for each parameter. To do this, let us first combine the results from the three chains in one vector for each parameter and plot the posterior distributions:

```
mcmcAmean = c(btmcmc[[1]][,"Amean"], btmcmc[[2]][,"Amean"], btmcmc[[3]][,"Amean"])
mcmcAsd = c(btmcmc[[1]][,"Asd"], btmcmc[[2]][,"Asd"], btmcmc[[3]][,"Asd"])
mcmcB = c(btmcmc[[1]][,"B"], btmcmc[[2]][,"B"], btmcmc[[3]][,"B"])
mcmcC = c(btmcmc[[1]][,"C"], btmcmc[[2]][,"C"], btmcmc[[3]][,"C"])
par(las=1, bty="l",mfrow=c(2,2))
plot(density(mcmcB), type="h", col="grey", main="Parameter B")
plot(density(mcmcC), type="h", col="grey", main="Parameter C")
plot(density(mcmcAmean), type="h", col="grey", main="A mean")
plot(density(mcmcAsd), type="h", col="grey", main="A standard deviation")
```



The medians and 95% credible intervals can be obtained as follows:

```
btSummary = rbind(quantile(x = mcmcB, probs = c(0.025, 0.5, 0.975)),
                  quantile(x = mcmcC, probs = c(0.025, 0.5, 0.975)),
                  quantile(x = mcmcAmean, probs = c(0.025, 0.5, 0.975)),
                  quantile(x = mcmcAsd, probs = c(0.025, 0.5, 0.975)))
rownames(btSummary) = c("B", "C", "Amean", "Asd")
btSummary
##              2.5%         50%      97.5%
## B     -0.7253227 -0.07344121 0.5374547
## C      0.6455647  1.45634242 2.3897468
## Amean  1.1441210  1.55566893 2.1795896
## Asd    0.1064759  0.47150517 1.1486713
```

As we can see both by the plots and the table, *A* is, on average, statistically different from zero, *i.e.*, blue tit females are more likely to mate with neighbouring

males. Moreover, B is essentially zero, *i.e.*, there is little evidence for mate choice based on size (at least analysing only the 2003 data). On the other hand, C is positive, which is evidence that older males are more likely to perform EPCs. The results are similar to what we found in our simpler model, and similar to what is reported by Schlicht *et al.* (2015).

5. Analysis of simulated data with two random effects

In this section, we will continue to use a Bayesian approach to work with the multinomial network model. This time we will analyse data from a simulation, so that we can compare the results we obtain with the "real" values from the simulation. This simulation is similar to that used in the main text, but with the following differences:

- Females mate multiply; each female copulates at least once, plus an additional number of copulations sampled from a Poisson distribution with mean = 2 (*i.e.*, on average, females copulate three times);
- Females have access to all males in the mating pool, but prefer to mate with spatially closer males. Females have a trait A that determines how strong the preference for neighbouring males is. The A values were sampled from a truncated normal distribution with mean A_{mean}, standard deviation A_{sd} and minimum value 0.05;
- In addition to the trait z, each male is characterized by a value G that influences attractiveness to females. The values of G were sampled from a normal distribution with mean zero and standard deviation G_{sd}. Higher G values increase mating probabilities.

These changes influence the equation of mating probabilities, so that the probability of a female i mating with a male j in this simulation is proportional to $exp(G_j - A_i \cdot s_{ij} + B \cdot z_j)$, where G_j and z_j are the male descriptors, s_{ij} is the spatial distance between female i and male j, A_i is the A trait of female i and B is a fixed parameter describing female preference for trait z. We ran a single simulation of a population composed of 100 males and 100 females with the following parameter values (all other parameters were kept as in the main text):

Parameter	Value
G_{mean}	0
G_{sd}	2
A_{mean}	2
A_{sd}	0.5
B	1

5.1. The model equation

Following the simulation, we will fit a model including a random effect of male identity in the form of varying intercepts (G), and a random effect of female identity in the form of varying female slopes for distance (A). The equation of the model is:

$$P_{ij} = \frac{exp\left(G_j - A_i \cdot s_{ij} + B \cdot z_j\right)}{\sum_{k=1}^{M}[G_k - A_i \cdot s_{ik} + B \cdot z_k]}.$$

The parameters we need to estimate are 100 G values (one for each male), 100 A values (one for each female), and a single B value. As in the previous example, we must assume A and G to follow some known distribution, and estimate the hyper-parameters of these random distributions. We will assume that both A and G follow

normal distributions, thus the hyper-parameters we will estimate will be the mean and standard deviation of A (A_{mean} and A_{sd}), and the standard deviation of G (G_{sd}). There is no need to estimate the mean of G because, due to the exponential nature of the model, the mean of the intercept does not influence the final probabilities. This mathematical property can be viewed in this simple calculation:

```
G = 3
exp(G+1)/(exp(G+2)+exp(G+3))
## [1] 0.09893802
G = 234
exp(G+1)/(exp(G+2)+exp(G+3))
## [1] 0.09893802
```

The fixed intercept in the numerator and denominator cancels out, so that $\frac{\exp(G+1)}{\exp(G+2)+\exp(G+3)}$ will be the same for any value of G. Therefore, the mean of G is zero in the simulation, and this is why we can fix the mean of G at zero in the Bayesian model. We could fix the mean of G at any value, but zero is a good choice because it makes the individual G values of males easily interpretable: positive values indicate males more likely to mate than average, negative values indicate males less likely to mate than average.

5.2. The data to fit the model

The data are stored in the same format as the data in the example of section 2: a file of male information, a file of female information, and a file describing who mated with whom:

```
females = read.table("ufemales.txt", header=TRUE, sep="\t", dec=".")
males = read.table("umales.txt", header=TRUE, sep="\t", dec=".")
couples = read.table("ucouples.txt", header=TRUE, sep="\t", dec=".")
```

```
#checking if everything is ok
head(males)
##   id         x         y    trait
## 1  1 0.4038541 0.4555880 4.198301
## 2  2 0.1828069 0.0861425 4.950366
## 3  3 0.2598226 0.5549116 3.950946
## 4  4 0.2603827 0.6217047 2.151949
## 5  5 0.3394526 0.4803402 5.040044
## 6  6 0.6781395 0.4018767 3.848726
head(females)
##   id         x          y
## 1  1 0.1385829 0.04288353
## 2  2 0.3380741 0.59083060
## 3  3 0.5512357 0.48279461
## 4  4 0.4987520 0.19375876
## 5  5 0.6662745 0.30184172
## 6  6 0.6409023 0.31898770
head(couples)
##   female male
## 1      1   71
## 2      2   40
## 3      2    1
## 4      2   12
## 5      3    1
## 6      3   40
```

Just as in the previous example, we will need to put these data in a specific format to run the model, and then keep all objects into a list. Our list this time will contain the following:

- A matrix of distances between male and female;
- A vector of male traits;
- A vector of female identities in each copulation;
- A vector of male identities in each copulation;

- The number of observed copulations;
- The number of males and females in the sample;
- A vector full of zeroes to do the zeroes trick.

```
ldata = list(dist = euclid(females[,c("x","y")], males[,c("x","y")]),
             mtrait = males$trait,
             male = couples$male,
             female = couples$female,
             n = nrow(couples),
             nmales = nrow(males),
             nfemales = nrow(females),
             x = rep(0, nrow(couples)))
```

The other objects we are going to need to fit our model are the vector with the parameters we want to retrieve from the MCMC iterations and the initial values for the parallel MCMC chains:

```
params = c("B", "Gmean", "Gsd", "Amean", "Asd")
initials = list(list(B = 1, Gprec = (1/2)^2, Amean = 2, Aprec = 1/(0.5)^2),
                #"real" values from the simulation
                list(B = 5, Gprec = (1/10)^2, Amean = 10, Aprec = 1/(5)^2),
                #higher values
                list(B = 0.2, Gprec = (1/0.4)^2, Amean = 0.4, Aprec = 1/(0.1)^2))
                #lower values
```

5.3. Writing and fitting the model

This model is similar to the one we fitted to the blue tit data. The main difference is that now we have one additional random effect (G), and the G values of individual males influence their copulation probabilities. Overall, however, the code of the model is similar:

```
umodel ="
model
{
  #uninformative prior for parameter B
  B ~ dnorm(0.0, 1E-06)
  #uninformative priors for the hyper-parameters
  Gprec ~ dgamma(0.01,0.01)
  Amean ~ dnorm(0,1E-06)
  Aprec ~ dgamma(0.01,0.01)
  #Declaring individual G values
  for(k in 1:nmales)
  {
    G[k] ~ dnorm(0, Gprec)
  }
  #Declaring individual A values
  for(l in 1:nfemales)
  {
    A[l] ~ dnorm(Amean, Aprec)
  }
  #calculating the likelihoods
  for(i in 1:n)
  {
    numerator[i]   <- exp(G[male[i]]-A[female[i]]*dist[female[i],male[i]] +
B*mtrait[male[i]])
    denominator[i] <- sum( exp(1)^(G-A[female[i]]*dist[female[i],] + B*mtrait) )
    logLike[i]  <- -log(numerator[i]/denominator[i])
    x[i] ~ dpois(logLike[i])
  }
  #Converting precision values to standard deviation values
  Gsd <- Gprec^(-0.5)
  Asd <- Aprec^(-0.5)
}"
write(umodel, "umodel.txt")
```

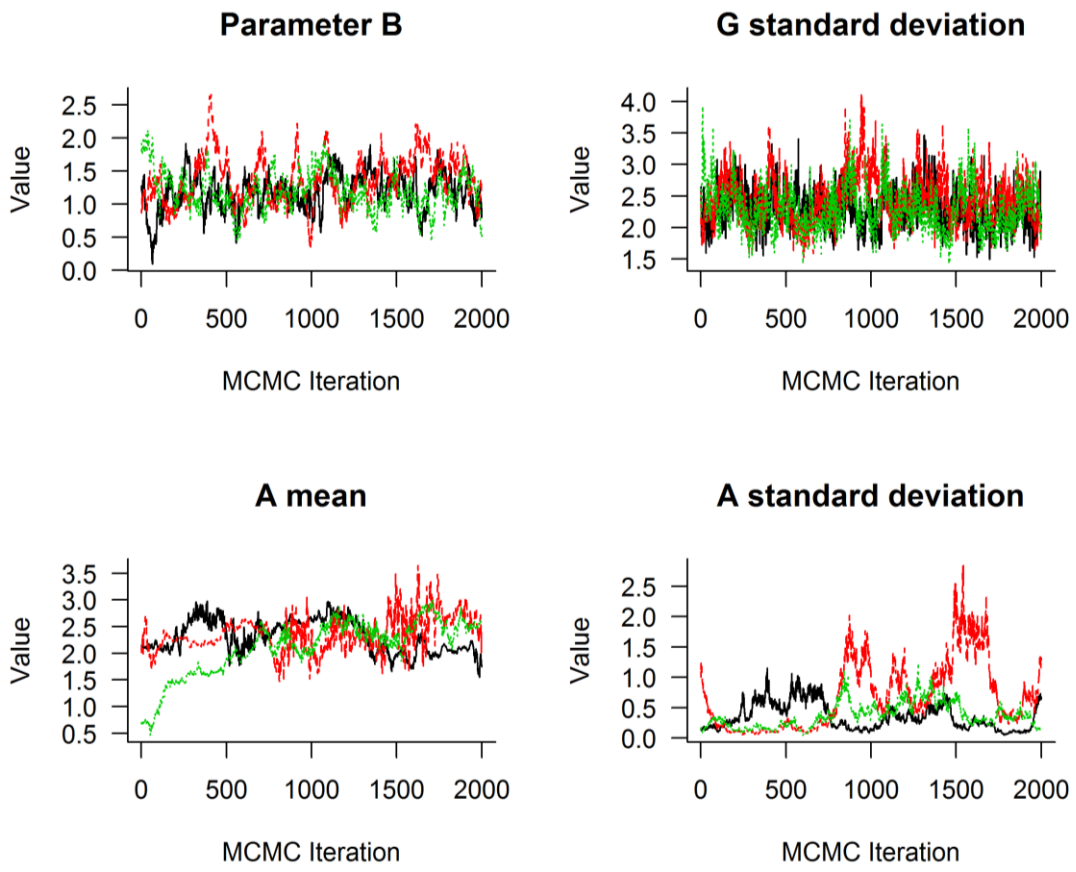
Now we can fit the model. These commands may take a long time to run because sample size is large, and there are two random effects. In a PC with 4 GB of memory, and a 2.16 GHz processor, the whole process took approximately 4 hours.

```
umodel = jags.model("MNbayes.txt",data=ldata, inits=initials, n.chains=3,n.adapt=300)
umcmc = coda.samples(model,params,n.iter=2000,thin=1)
```

5.4. Checking the results

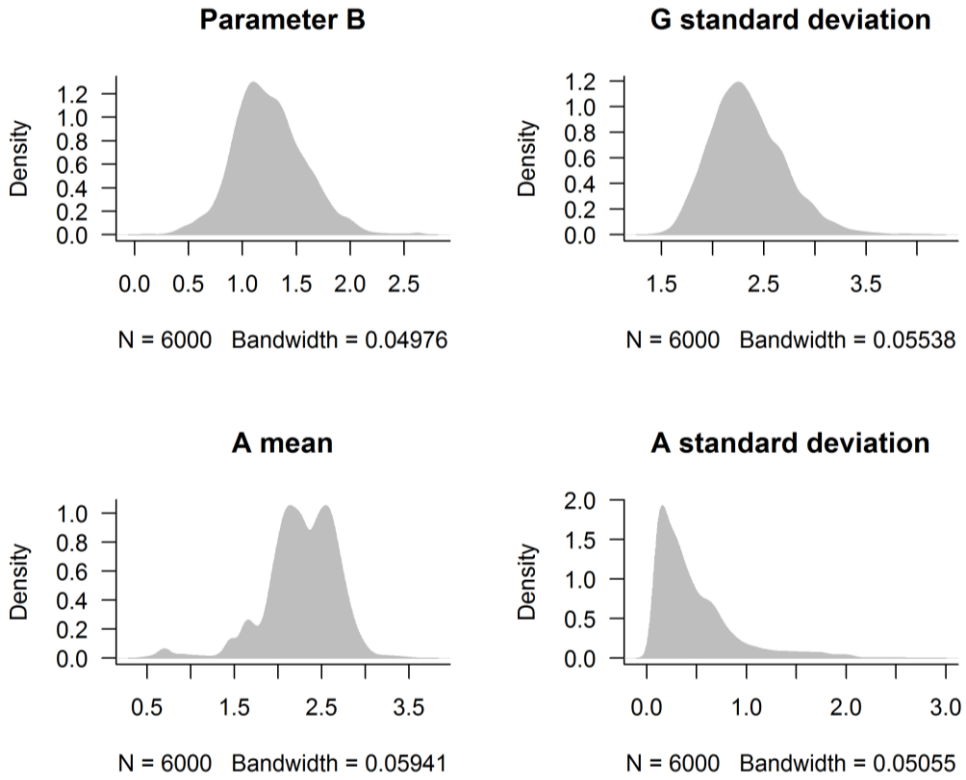
Let us begin by checking if the optimization converged adequately:

```
par(las=1, bty="l",mfrow=c(2,2))
matplot(cbind(umcmc[[1]][,"B"],
              umcmc[[2]][,"B"],
              umcmc[[3]][,"B"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="Parameter B")
matplot(cbind(umcmc[[1]][,"Gsd"],
              umcmc[[2]][,"Gsd"],
              umcmc[[3]][,"Gsd"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="G standard deviation")
matplot(cbind(umcmc[[1]][,"Amean"],
              umcmc[[2]][,"Amean"],
              umcmc[[3]][,"Amean"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="A mean")
matplot(cbind(umcmc[[1]][,"Asd"],
              umcmc[[2]][,"Asd"],
              umcmc[[3]][,"Asd"]), col=1:3, type="l",
              ylab="Value", xlab="MCMC Iteration", main="A standard deviation")
```



Convergence could be better, especially for the *A* hyper-parameters, but running 5,000 or 10,000 iterations of this model could take many hours. Thus, let us verify how close we got to the real values, even though convergence is far from optimal. First, we can plot the posterior distributions:

```
uAmean = c(umcmc[[1]][,"Amean"], umcmc[[2]][,"Amean"], umcmc[[3]][,"Amean"])
uAsd = c(umcmc[[1]][,"Asd"], umcmc[[2]][,"Asd"], umcmc[[3]][,"Asd"])
uB = c(umcmc[[1]][,"B"], umcmc[[2]][,"B"], umcmc[[3]][,"B"])
uGsd = c(umcmc[[1]][,"Gsd"], umcmc[[2]][,"Gsd"], umcmc[[3]][,"Gsd"])
par(las=1, bty="l",mfrow=c(2,2))
plot(density(uB), type="h", col="grey", main="Parameter B")
plot(density(uGsd), type="h", col="grey", main="G standard deviation")
plot(density(uAmean), type="h", col="grey", main="A mean")
plot(density(uAsd), type="h", col="grey", main="A standard deviation")
```



Finally, we can calculate the medians and 95% credible intervals and compare them with the real values from the simulation:

```
uSummary = rbind(quantile(x = uB, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = uGsd, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = uAmean, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = uAsd, probs = c(0.025, 0.5, 0.975)))
rownames(uSummary) = c("B", "Gsd", "Amean", "Asd")
uSummary = cbind(uSummary, realValues = c(B = 1, Gsd =2, Amean=2, Asd=0.5))
uSummary
##             2.5%       50%    97.5% realValues
## B     0.63582707 1.2217036 1.946752        1.0
## Gsd   1.74359383 2.3033738 3.138849        2.0
## Amean 1.40574189 2.2942567 2.899904        2.0
## Asd   0.08648745 0.3470208 1.673547        0.5
```

Most parameters are a little overestimated, whereas A_{sd} is underestimated, but in all cases, real values are within the 95% credible intervals. Thus, even with poor MCMC convergence, we achieved a decent description of our simulated population.

6. Analysis of simulated data with moving individuals

Here, again, we will use a Bayesian approach to fit the multinomial network model to simulated data. However, in addition to females mating multiple times, individuals move between observations, so that spatial distances must be recalculated at each time step. The simulation used to generate this data works as the simulation presented in section 5, with the difference that individuals move between time steps. The simulation has three time steps: in the first time step, all females copulate, whereas in the second, and third time steps each female has mating probabilities of 0.5 and 0.25, respectively. Additionally, between time steps, individuals move following Brownian motion. Each individual moves towards a random direction, and a distance between 0 and 0.1 units sampled from a uniform distribution. If an individual tries to move to a position outside the 1 x 1 landscape, we change the movement direction by 180^{o} to prevent individuals from leaving the landscape. The equation of mating probabilities, and the parameter values are the same as in section 5.

6.1. The data to fit the model

The data for this model is in a different format from the previous examples because each individual has three different spatial coordinates. The data needed to fit the model is stored in four files, two of them are similar to the ones we used so far, the **wmales.txt** file with the male information, and the **wcouples.txt** file, with the copulation data. Additionally, there are two separate files of spatial coordinates,

namely **wmalesxy.txt** and **wfemalesxy.txt**. The files of spatial coordinates contain male and female Cartesian coordinates at each time step.

The first thing to do, as before, is to read the data:

```
couples = read.table("wcouples.txt", header=TRUE, sep="\t", dec=".")
males = read.table("wmales.txt", header=TRUE, sep="\t", dec=".")
malesxy = read.table("wmalesxy.txt", header=TRUE, sep="\t", dec=".")
femalesxy = read.table("wfemalesxy.txt", header=TRUE, sep="\t", dec=".")
#as usual, let's check if everything is ok
head(couples)
##   time female male
## 1    1      1   38
## 2    1      2   53
## 3    1      3   20
## 4    1      4    9
## 5    1      5   35
## 6    1      6   66
head(males)
##   id    trait
## 1  1 5.370958
## 2  2 3.435302
## 3  3 4.363128
## 4  4 4.632863
## 5  5 4.404268
## 6  6 3.893875
head(malesxy)
##   male time         x          y
## 1    1    1 0.6427937 0.06798640
## 2    2    1 0.6363429 0.02609452
## 3    3    1 0.1360115 0.28350748
## 4    4    1 0.3763861 0.00815805
## 5    5    1 0.3691979 0.40155959
## 6    6    1 0.1133745 0.66818050
head(femalesxy)
##   female time         x         y
## 1      1    1 0.3522499 0.5413047
## 2      2    1 0.1999445 0.3571204
## 3      3    1 0.5490296 0.1432234
## 4      4    1 0.2148563 0.5070937
## 5      5    1 0.3645497 0.5431124
```

```
## 6      6    1 0.3379622 0.4994840
```

As in the previous examples with MCMC optimization, the data needs to be arranged in a list to be used by the *rjags* functions. The main difference between the current and the previous examples is that, because individuals move, we need to build one separate spatial distance matrix for each time step. Then, to make things easier during model fitting, we will rearrange this data in a single distance matrix, in which each line will represent one copulation and each column will represent a male in the mating pool. In this matrix, each cell will contain the distance between the female that performed the copulation and an available male at the time of the copulation. First, let us build the three spatial matrices and store them into a list, each matrix representing one time step:

```
slist = list()

for(i in 1:3)

  slist[[i]] = euclid(femalesxy[femalesxy$time == i,c("x","y")], malesxy[malesxy$time ==
i,c("x","y")])
```

Now, using another *for()*, we can easily rearrange the data as described above. Every line of the **couples** *data.frame* has information about the time of each copulation and the female involved. The time column determines which list we should access, and the female number is the line of the distance matrix inside the list.

```
#creates a matrix with the adequate dimensions

smatrix = matrix(NA, nrow=nrow(couples), ncol = nrow(males))


#retrieves information from the appropriate matrix

for(i in 1:nrow(couples))

  smatrix[i,] = slist[[couples$time[i]]][couples$female[i],]
```

This was the only thing we needed to do differently to cope with moving individuals. To proceed to model fitting, we need a list of data similar to the one we used before:

```
#The process to discover the number of females is a little different this time.
nfemales = length(unique(femalesxy$female))
#Other than that, the list is essentially the same as before
ldata = list(dist = smatrix,
             mtrait = males$trait,
             male = couples$male,
             female = couples$female,
             n = nrow(couples),
             nmales = nrow(males),
             nfemales = nfemales,
             x = rep(0, nrow(couples)))
```

Now we need to determine which parameters we want to retrieve from the MCMC fitting, the number of chains to run, and the initial values. The model has the same parameters as before, B, G_{sd}, A_{mean} and A_{sd}.

```
params = c("B", "Gsd", "Amean", "Asd")
n.cha = 3
initials = list(list(B = 1, Gprec = (1/2)^2, Amean = 2, Aprec = 1/(0.5)^2),
                list(B = 5, Gprec = (1/10)^2, Amean = 10, Aprec = 1/(5)^2),
                list(B = 0.2, Gprec = (1/0.4)^2, Amean = 0.4, Aprec = 1/(0.1)^2))
```

6.2. Writing and fitting the model

The code of the model is almost the same as in section 5. The difference is in the spatial data matrix. Now, each row of the matrix represents one copulation, instead of a female. The result is that the indexation is simpler than before.

```
wmodel = "
model
{
  #uninformative prior for parameter B
  B ~ dnorm(0.0, 1E-06)
```

```
  #uninformative priorr for the meta-parameters
  Gprec ~ dgamma(0.01,0.01)
  Amean ~ dnorm(0,1E-06)
  Aprec ~ dgamma(0.01,0.01)
  #-------------------
  #Declaring individual G values
  for(k in 1:nmales)
  {
    G[k] ~ dnorm(0, Gprec)
  }
  #Declaring individual A values
  for(l in 1:nfemales)
  {
    A[l] ~ dnorm(Amean, Aprec)
  }
  #-------------------
  #calculating the likelihoods
  for(i in 1:n)
  {
    #the only change is in the indexation of the dist matrix
    numerator[i]   <- exp(G[male[i]] - A[female[i]]*dist[i,male[i]] + B*mtrait[male[i]])
    denominator[i] <- sum( exp(1)^(G - A[female[i]]*dist[i,] + B*mtrait) )
    logLike[i]  <- -log(numerator[i]/denominator[i])
    x[i] ~ dpois(logLike[i])
  }
#Converting precision values to standard deviation values
Gsd <- Gprec^(-0.5)
Asd <- Aprec^(-0.5)
}"
write(wmodel, "wmodel.txt")
```

Finally, we can optimize model parameters. As before, these commands may take several hours to run:

```
wmodel = jags.model("MNbayes.movement.txt",data = ldata, inits = initials, n.chains=n.ch
a,n.adapt=300)
wmcmc = coda.samples(wmodel,params,n.iter=2000,thin=1)
```

6.3. Checking the results

Since this is the third time we do this in this tutorial, let us just assemble a summary table. If you wish, you can plot graphs using the same commands we used before.

```
wAmean = c(wmcmc[[1]][,"Amean"], wmcmc[[2]][,"Amean"], wmcmc[[3]][,"Amean"])
wAsd = c(wmcmc[[1]][,"Asd"], wmcmc[[2]][,"Asd"], wmcmc[[3]][,"Asd"])
wB = c(wmcmc[[1]][,"B"], wmcmc[[2]][,"B"], wmcmc[[3]][,"B"])
wGsd = c(wmcmc[[1]][,"Gsd"], wmcmc[[2]][,"Gsd"], wmcmc[[3]][,"Gsd"])
wSummary = rbind(quantile(x = wB, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = wGsd, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = wAmean, probs = c(0.025, 0.5, 0.975)),
                 quantile(x = wAsd, probs = c(0.025, 0.5, 0.975)))
rownames(wSummary) = c("B", "Gsd", "Amean", "Asd")
wSummary = cbind(wSummary, realValues = c(B = 1, Gsd =2, Amean=2, Asd=0.5))
wSummary
##              2.5%       50%    97.5% realValues
## B     0.36662486 0.8010901 1.294380        1.0
## Gsd   1.24348947 1.6614367 2.262461        2.0
## Amean 0.57272209 1.6770582 2.781761        2.0
## Asd   0.08517368 0.3846312 1.735045        0.5
```

The fitted values are all relatively close to the real values, and in all cases the real value is within the 95% credibility interval.

7. Pollination data example

In the main text, we mention that the multinomial network model can be used to investigate not only mate choice, but also choice in other ecological contexts. Here, we show an example of flower choice by pollinators. One of the subjects of research on pollination is flower constancy, a behaviour exhibited by pollinators that restrict visits to a single floral type, bypassing flowers of other species (Waser 1986). This behaviour is exhibited by both insect and vertebrate pollinators, and is mostly well understood in honeybees (Amaya-Márquez 2009). The main explanations involve memory

restrictions and the use of search images to locate flowers in the environment (Amaya-Márquez 2009). When flowers of the preferred type are unavailable or too far away, pollinators may switch flower types, so that both flower species and spatial distance are likely to influence floral constancy by pollinators. Thus, the multinomial network model can be used to investigate flower choice by pollinators, including the quantification of pollinator tendency to remain visiting the same flower species.

Here, we use simulated data to illustrate how to use the multinomial model to investigate floral constancy. The simulation contains 20 pollinators from a single species and 100 flowers from two equally abundant species. Flowers were randomly distributed in a squared landscape of 1 X 1 arbitrary units. The simulation comprises five time steps. In the first time step, each pollinator is already on a flower, and in each subsequent time step, each pollinator visits a new flower. Pollinators chose their flowers based on a visitation rule in which a pollinator i will go to plant j with a probability proportional to $exp(-A \cdot s_{ij} + B \cdot t_{ij})$. In this equation, t_{ij} is a binary variable of "flower equality", which has value of one if the flower where pollinator i is located and flower j belong to the same species, and has a value of zero otherwise. The variable s_{ij} is the spatial distance between the location of pollinator i and flower j, and A and B are the model parameters. High B values make pollinators more flower constant, whereas high A values increase pollinator preference for nearby flowers. In addition, a pollinator never returns to a flower it has already visited. We ran one simulation with $A = 2$ and $B = 2$.

The quantification of flower constancy is commonly made using an index devised by Bateman (1951). This index was designed simply to answer if there is flower constancy in a given population. By using the multinomial network model, it is possible to investigate flower constancy (or flower choice in general) in greater detail,

estimating the additive effects of spatial distance and flower characteristics and allowing uncertainty estimates of the parameters. The statistical model includes as predictor variables the spatial distances s_{ij} and flower equality t_{ij}. In addition, because in this example pollinators never visit the same flower twice, we need to ensure that the probability of visiting an already visited flower is zero. To do this, we will add a third variable to the equation, v_{ij}, which will be one if the flower j was never visited by pollinator i and zero if it was. Another way to think about v_{ij} is that it represents that flower j is available to pollinator i, and the availability rule is that a flower becomes unavailable to pollinator i after the visit. Thus, the model equation is:

$$P_{ij} = \frac{v_{ij} \cdot exp\left(-A \cdot s_{ij} + B \cdot t_{ij}\right)}{\sum_{k=1}^{M}[v_{ik} \cdot exp\left(-A \cdot s_{ik} + B \cdot t_{ik}\right)]}$$

The pollination data is available as two separate files. The **switches.txt** file describes in which flower each pollinator was per time step, and the columns are:

- time: the time step in which each flower switch was observed;
- polli: the ID of the pollinator;
- oFlower: the ID of the flower of origin, from which the pollinator first left;
- dFlower: the ID of the destination flower, to which the pollinator went.

The second file, **flowers.txt**, describes the flowers, and contains x and y coordinates for each flower and its species. We can read the data with the following commands:

```
switches = read.table("switches.txt", sep="\t", dec=".", header=TRUE)
flowers = read.table("flowers.txt", sep="\t", dec=".", header=TRUE)
```

Then, we can check if the data is correct:

```
head(switches)
##   time polli oFlower dflower
## 1    1     1      18      88
## 2    1     2      41      61
## 3    1     3      22      82
## 4    1     4      56      11
## 5    1     5      38      58
## 6    1     6      26      93
head(flowers)
##   id         x         y species
## 1  1 1.5766002 0.3192473       1
## 2  2 0.9862844 1.5446147       1
## 3  3 2.9169653 0.4063237       1
## 4  4 1.1708911 2.7920999       2
## 5  5 2.1789880 0.1788175       2
## 6  6 1.0640823 0.5753927       2
```

As in previous examples (sections 2 and 3), we need to store all data in matrix format, so that we can take advantage of *R*'s vectorised operations during calculations. In each matrix, each line represents one floral visit (a pollinator going from one flower to the next), and each column represents one flower. To calculate the probabilities P_{ij} that a pollinator *i* will visit flower *j* at a given time, we will need three matrices: (1) spatial distances, (2) flower equality, and (3) flower availability.

To build the spatial matrix, we first calculate the distances between all flowers using the *euclid()* function again. Next, we can use indexation to create a new matrix in which each line contains the spatial distances from flowers of origin (column **oFlower** in the **switches** data.frame) to all other flowers in the population (similar to what we did with the extra-pair copulation data).

```
sMatrix = euclid(flowers[,c("x","y")])
sik = sMatrix[switches$oFlower,]
```

The flower equality matrix can be created easily too:

```
#compares the species identity of all individual flowers
tik = outer(flowers$species[switches$oFlower], flowers$species, "==")
#substitutes TRUE and FALSE values by ones and zeroes
tik = ifelse(tik, 1, 0)
```

Finally, we can create the availability matrix. To create this matrix, we will start with a matrix full of ones. Then, for each floral visit (including the flower in which pollinators were first observed), we change the one for a zero in the cells representing the visited flower in the lines representing visits by pollinator *i* in all subsequent time steps. After a visitation, the visited flower becomes unavailable to the pollinator:

```
vmatrix = matrix(1, nrow=nrow(switches), ncol=nrow(flowers))
for(i in 1:nrow(switches))
    vmatrix[switches$polli == switches$polli[i] & switches$time >=
switches$time[i],switches$oFlower[i]] = 0
```

The last piece of data we need is a matrix specifying which was the flower of destination in each flower switch by the pollinators. The first column of the matrix is simply a sequence from one to the number of observations, whereas the other column specifies the IDs of the destination flowers:

```
swMatrix = cbind(1:nrow(switches), switches$dflower)
```

The log-likelihood function for this model is similar to the ones we presented in sections 2 and 3. Again, we can fit the model using the *mle2()* function:

```
LL.polli = function(A, B, tik, sik, vmatrix, pos)
{
  pik = exp(-A*sik + B*tik) * vmatrix
  pij = pik[pos]/apply(pik, 1, sum)
```

```
  return(-sum(log(pij)))
}
#model fitting
fitted = mle2(minuslogl = LL.polli, start = list(A=1, B=1),
              data = list(sik=sik, tik=tik, vmatrix=vmatrix, pos=swMatrix))
fitted
## Call:
## mle2(minuslogl = LL.polli, start = list(A = 1, B = 1), data = list(sik = sik,
##     tik = tik, vmatrix = vmatrix, pos = swMatrix))
##
## Coefficients:
##        A        B
## 2.021294 2.152839
##
## Log-likelihood: -276.7
confint(fitted)
##      2.5 %   97.5 %
## A 1.626822 2.440872
## B 1.506378 2.922845
```

As we can see, both fitted values were close to the real simulation values (A = 2, B = 2) and both 95% confidence intervals include the real values and do not include zero.

8. References

Amaya-Márquez, M. (2009). Floral constancy in bees: a revision of theories and a comparison with other pollinators. *Revista Colombiana de Entomología*, 35, 206–216.

Bateman, A.J. (1951). The taxonomic discrimination of bees. *Heredity*, 5, 271e278.

Bolker, B.M. (2008). *Ecological models and data in R*. Princeton University Press, Princeton.

Bolker. B.M. & R Core Team (2014). *bbmle*: tools for general maximum likelihood estimation. R package version 1.0.17.

Gelman, A., Carlin, J.B., Stern, H.S. & Rubin, D.B. (2014). *Bayesian data analysis*, 3rd Edition. Chapman & Hall, London.

Plummer, M. (2015). *rjags*: bayesian graphical models using MCMC. R package version 4-4.

Schlicht, L., Valcu, M. & Kempenaers, B. (2015). Spatial patterns of extra-pair paternity: beyond paternity gains and losses. *Journal of Animal Ecology*, **84**, 518–531.

Valcu, M. & Schlicht, L. (2014). *expp*: spatial analysis of extra-pair paternity. R package version 1.1.

Waser, N.M. (1986). Flower constancy: definition, cause, and measurement. *American Naturalist*, 127, 593–603.