sparta - Species Presence/Absence R Trends Analyses

Introduction

Sparta provides a range of tools for analysing trends in species occurrence data. These data have three components, what where and when. This package is based on the work presented in Isaac et al (2014)

Installing the package is easy and can be done from CRAN. Alternatively the development version can be installed from GitHub.

NOTE: JAGS must be installed before the R package installation will work. JAGS can be found here - http://sourceforge.net/projects/mcmc-jags/files/JAGS/

```
# Install the package from CRAN
# THIS WILL WORK ONLY AFTER THE PACKAGE IS PUBLISHED
install.packages('sparta')

# Or install the development version from GitHub
library(devtools)
install_github('biologicalrecordscentre/sparta')
```

```
# Once installed, load the package
library(sparta)
```

```
## Loading required package: lme4
## Loading required package: Matrix
## Loading required package: Rcpp
```

The functions in sparta cover a range of tasks. Primarily they are focused on analysing trends in species occurrence data while accounting for biases (see Isaac et al, 2014). In this vignette we step through these functions and others so that you can understand how the package works. If you have any questions you can find the package maintainers email address using `maintainer('sparta')`, and if you have issues or bugs you can report them here

Create some example data

To show you how the functions work first we need some data, here we create a data set that is hopefully similar in format to your own data.

```
# Create data
n <- 15000 # size of dataset
nyr <- 20 # number of years in data
nSamples <- 100 # set number of dates
nSites <- 50 # set number of sites
set.seed(125) # set a random seed

# Create somes dates
first <- as.Date(strptime("1980/01/01", "%Y/%m/%d"))
last <- as.Date(strptime(paste(1980+(nyr-1),"/12/31", sep=''), "%Y/%m/%d"))
dt <- last-first
rDates <- first + (runif(nSamples)*dt)
```

```
# taxa are set as random letters
taxa <- sample(letters, size = n, TRUE)

# sites are visited randomly
site <- sample(paste('A', 1:nSites, sep=''), size = n, TRUE)

# the date of visit is selected at random from those created earlier
time_period <- sample(rDates, size = n, TRUE)

myData <- data.frame(taxa, site, time_period)

head(myData)
```

```
##   taxa site time_period
## 1    t  A22  1989-04-12
## 2    s  A27  1997-06-16
## 3    v  A40  1996-09-10
## 4    j  A28  1998-09-21
## 5    q  A11  1995-07-24
## 6    e  A22  1985-09-10
```

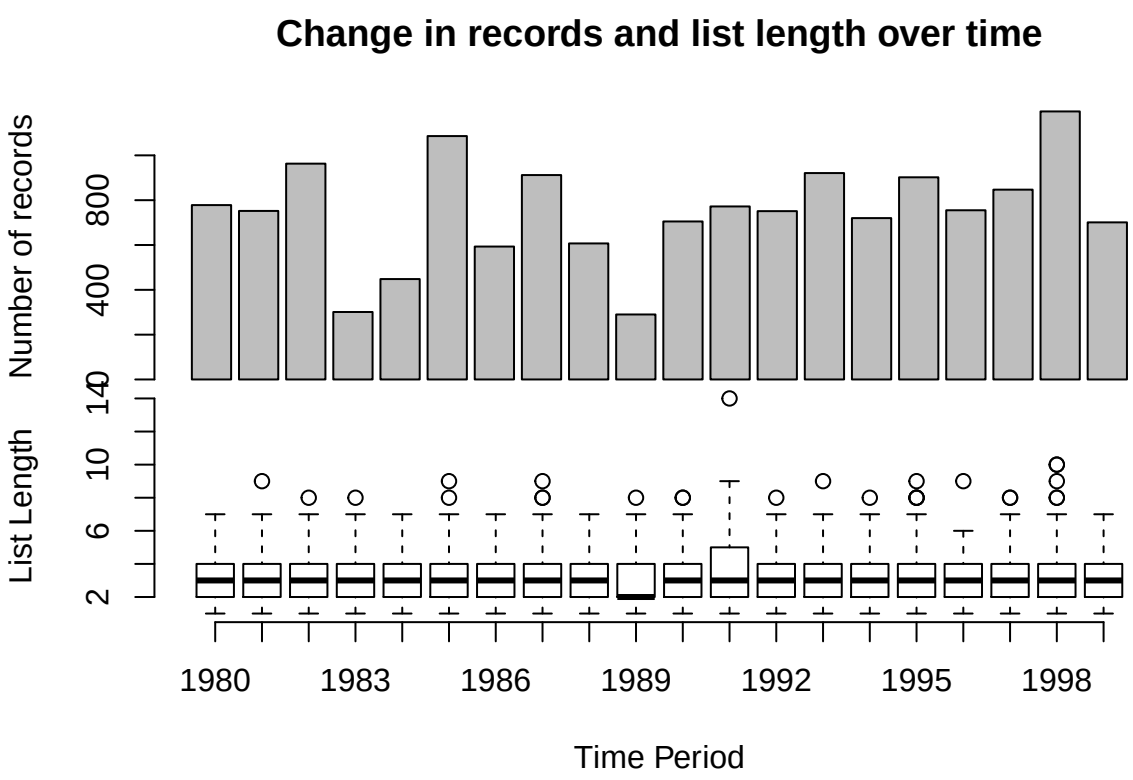
In general this is the format of data you will need for all of the function in sparta. The taxa and site columns should be characters and the time_period column should ideally be a date but can in some cases be a numeric.

Assessing the quality of data

It can be useful to have a look at your data before you do any analyses. For example it is important to understand the biases in your data. The function `dataDiagnostics` is designed to help with this.

```
# Run some data diagnostics on our data
results <- dataDiagnostics(taxa = myData$taxa,
                           site = myData$site,
                           time_period = myData$time_period,
                           progress_bar = FALSE)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period): 854 out of 15000 observations will be removed as duplicates
```



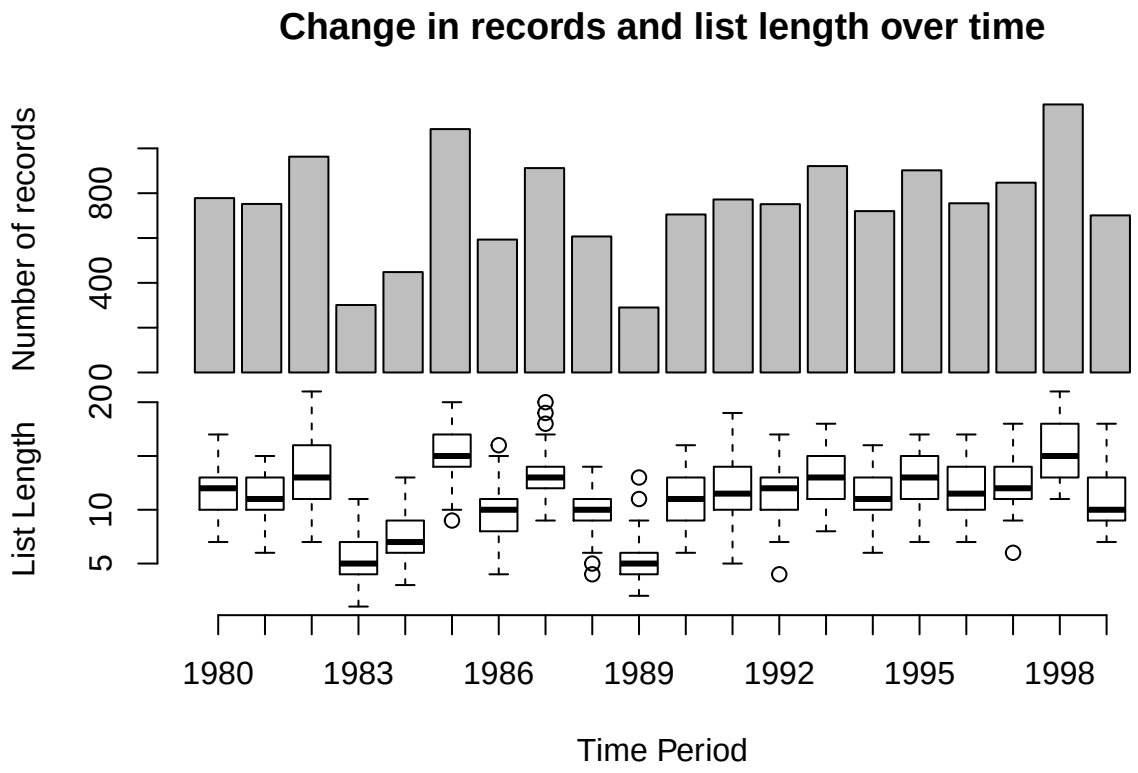

```
## ## Linear model outputs ##
##
## There is no detectable change in the number of records over time:
##
##                 Estimate   Std. Error   t value  Pr(>|t|)
## (Intercept) -20712.66617 17478.906362 -1.185009 0.2514240
## time_period     10.78797     8.785541  1.227923 0.2352915
##
##
## There is no detectable change in list lengths over time:
##
##                  Estimate   Std. Error   z value   Pr(>|z|)
## (Intercept)  1.152024e+00 2.990793e-02 38.519018 0.00000000
## time_period -6.714161e-06 3.850515e-06 -1.743705 0.08121062
```

The plot produced shows the number of records for each year in the top plot and the average list length in a box plot at the bottom. List length is the number of taxa observed on a visit to a site, where a visit is taken to be a unique combination of site and time_period. It is important to know if you have trends in either of these over time. To help in this regard this function also fits linear models to both these graphs and the results have been returned to the console. Unsurprisingly, since this is a random dataset, we have no trend in either the number of records or list length over time. This fucntion also works if we have a numeric for time period such as the year

```
# Run some data diagnostics on our data
results <- dataDiagnostics(taxa = myData$taxa,
```

```
                          site = myData$site,
                          time_period = as.numeric(format(myData$time_period, '%Y')),
                          progress_bar = FALSE)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period): 3841 out of 15000 observations will be removed as duplicates
```




```
## ## Linear model outputs ##
##
## There is no detectable change in the number of records over time:
##
##                 Estimate   Std. Error   t value  Pr(>|t|)
## (Intercept) -20712.66617 17478.906362 -1.185009 0.2514240
## time_period     10.78797     8.785541  1.227923 0.2352915
##
##
## There is a significant change in list lengths over time:
##
##                 Estimate  Std. Error   z value     Pr(>|z|)
## (Intercept) -20.97654491 3.271321795 -6.412254 1.433841e-10
## time_period   0.01175496 0.001643964  7.150377 8.654008e-13
```

If we want to view these results in more detail we can interrogate the object `results`

```r
# See what is in results..
names(results)
```

```
## [1] "RecordsPerYear"  "VisitListLength" "modelRecs"       "modelList"
```

```r
# Let's have a look at the details
head(results$RecordsPerYear)
```

```
## RecordsPerYear
## 1980 1981 1982 1983 1984 1985
##  778  752  963  301  448 1086
```

```r
head(results$VisitListLength)
```

```
##   time_period site listLength
## 1        1980   A1          8
## 2        1980  A10         16
## 3        1980  A11         10
## 4        1980  A12         10
## 5        1980  A13         10
## 6        1980  A14         13
```

```r
summary(results$modelRecs)
```

```
##
## Call:
## glm(formula = count ~ time_period, data = mData)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -454.61  -121.14   -10.08   131.17   384.55
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -20712.666  17478.906  -1.185    0.251
## time_period     10.788      8.786   1.228    0.235
##
## (Dispersion parameter for gaussian family taken to be 51328.51)
##
##     Null deviance: 1001306  on 19  degrees of freedom
## Residual deviance:  923913  on 18  degrees of freedom
## AIC: 277.57
##
## Number of Fisher Scoring iterations: 2
```

```r
summary(results$modelList)
```

```
##
## Call:
## glm(formula = listLength ~ time_period, family = "poisson", data = space_time)
```

```
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.7365  -0.7187   0.0185   0.6542   2.9568
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -20.976545   3.271322  -6.412 1.43e-10 ***
## time_period   0.011755   0.001644   7.150 8.65e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 1260.8  on 999  degrees of freedom
## Residual deviance: 1209.6  on 998  degrees of freedom
## AIC: 5414.8
##
## Number of Fisher Scoring iterations: 4
```

Telfer

Telfer's change index is designed to assess the relative change in range size of species between two time periods (Telfer et al, 2002). This function can take multiple time periods and will complete all pairwise comparisons.

Our data is not quite in the correct format for Telfer since it is used for comparing two time periods but our `time_period` column is a date. We can fix this by using the `date2timeperiod` function.

```
## Create a new column for the time period
# First define my time periods
time_periods <- data.frame(start=c(1980,1985,1990,1995),end=c(1984,1989,1994,1999))

time_periods
```

```
##   start  end
## 1  1980 1984
## 2  1985 1989
## 3  1990 1994
## 4  1995 1999
```

```
# Now use these to assign my dates to time periods
myData$tp <- date2timeperiod(myData$time_period, time_periods)

head(myData)
```

```
##   taxa site time_period tp
## 1    t  A22  1989-04-12  2
## 2    s  A27  1997-06-16  4
## 3    v  A40  1996-09-10  4
## 4    j  A28  1998-09-21  4
## 5    q  A11  1995-07-24  4
## 6    e  A22  1985-09-10  2
```

As you can see our new column indicates which time period each date falls into with 1 being the earliest time period, 2 being the second and so on. This function will also work if instead of a single date for each record you have a date range

```
## Create a dataset where we have date ranges
Date_range <- data.frame(startdate = myData$time_period,
                         enddate = (myData$time_period + 300))

head(Date_range)
```

```
##    startdate    enddate
## 1 1989-04-12 1990-02-06
## 2 1997-06-16 1998-04-12
## 3 1996-09-10 1997-07-07
## 4 1998-09-21 1999-07-18
## 5 1995-07-24 1996-05-19
## 6 1985-09-10 1986-07-07
```

```
# Now assign my date ranges to time periods
Date_range$time_period <- date2timeperiod(Date_range, time_periods)

head(Date_range)
```

```
##    startdate    enddate time_period
## 1 1989-04-12 1990-02-06          NA
## 2 1997-06-16 1998-04-12           4
## 3 1996-09-10 1997-07-07           4
## 4 1998-09-21 1999-07-18           4
## 5 1995-07-24 1996-05-19           4
## 6 1985-09-10 1986-07-07           2
```

As you can see in this example when a date range spans the boundaries of your time periods (as in the first row) NA is returned.

Now we have our data in the right format we can use the `telfer` function to analyse the data. Telfer is used for comparing two time periods and if you have more than this the `telfer` function will all pair-wise comparisons

```
# Here is our data
head(myData)
```

```
##   taxa site time_period tp
## 1    t  A22  1989-04-12  2
## 2    s  A27  1997-06-16  4
## 3    v  A40  1996-09-10  4
## 4    j  A28  1998-09-21  4
## 5    q  A11  1995-07-24  4
## 6    e  A22  1985-09-10  2
```

```
telfer_results <- telfer(taxa = myData$taxa,
                         site = myData$site,
                         time_period = myData$tp,
                         minSite = 2)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period, : 10105 out of 15000 observations will be removed as
## duplicates
```

We get a warning message indicating that a large number of rows are being removed as duplicates. This occurs since we are now aggregating records into time periods and therefore creating a large number of duplicates.

The results give is the change index for each species (rows) in each of the pairwise comparisons of time periods (columns).

```
head(telfer_results)
```

```
##   taxa Telfer_1_2 Telfer_1_3 Telfer_1_4 Telfer_2_3 Telfer_2_4 Telfer_3_4
## 1    a  1.1981584 -0.5277650 -0.9579121 -1.0469769 -1.2236419 -1.3157213
## 2    b -0.5604101 -0.9800187 -0.7893487 -1.0046869 -0.8275179 -1.0137634
## 3    c -1.0287486  0.4765040  0.3709731  0.6416992  0.4546273  0.5679879
## 4    d  2.5105096  0.6440287 -0.8691085  1.3099044 -1.0311405 -0.7817529
## 5    e  0.3592361  1.0006306 -0.3335825  0.7187069 -0.4214406 -0.2389855
## 6    f  1.0280531 -1.6496005  1.7450992 -1.4124309  1.2429233  1.0737873
```

Reporting Rate Models

The Reporting rate models in sparta are modular giving the user a number of options for their analyses.

First there are functions that allow the user to subset data according to certain rules

Data selection

The fist function allows you to subset your data by list length. This works out, for each combination of time_period and taxa (a visit), the number of species observed (list length). Any records that to not come from a list that meets your list length criteria are then dropped.

```
# Select only records which occur on lists of length 5 or more
myDataL <- siteSelectionMinL(taxa = myData$taxa,
                             site = myData$site,
                             time_period = myData$time_period,
                             minL = 5)
```

```
## Warning in errorChecks(taxa, site, time_period): 854 out of 15000
## observations will be removed as duplicates
```

```
head(myDataL)
```

```
##   taxa site time_period
## 1    e   A1  1981-07-11
## 2    j   A1  1981-07-11
## 3    a   A1  1981-07-11
## 4    n   A1  1981-07-11
## 5    t   A1  1981-07-11
## 6    x   A1  1982-02-23
```

```
# We now have a much smaller dataset after subsetting
nrow(myData)
```

```
## [1] 15000
```

```
nrow(myDataL)
```

```
## [1] 4217
```

We are also able to subset by the number of times a site is sampled. The function `siteSelectionMinTP` does this. When time_period is a date, as in this case, minTP is minimum number of years a site must be sampled in for it be included in the subset.

```
# Select only data from sites sampled in at least 3 years
myDataTP <- siteSelectionMinTP(taxa = myData$taxa,
                               site = myData$site,
                               time_period = myData$time_period,
                               minTP = 5)
```

```
## Warning in errorChecks(taxa, site, time_period): 854 out of 15000
## observations will be removed as duplicates
```

```
head(myDataTP)
```

```
##   taxa site time_period
## 1    t  A22  1989-04-12
## 2    s  A27  1997-06-16
## 3    v  A40  1996-09-10
## 4    j  A28  1998-09-21
## 5    q  A11  1995-07-24
## 6    e  A22  1985-09-10
```

```
# Here we have only lost a small number rows, this is because
# all sites have had 20 visits. Those rows that have been removed
# are duplicates
nrow(myData)
```

```
## [1] 15000
```

```
nrow(myDataTP)
```

```
## [1] 14146
```

Following the method in Roy et al (2012) we can combine these two functions to subset both by the length of lists and by the number of years that sites are sampled. This has been wrapped up in to the function `siteSelection` which takes all the arguments of the previous two functions plus the argument `LFirst` which indicates whether the data should be subset by list length first (`TRUE`) or second (`FALSE`).

```
# Subset our data as above but in one go
myDataSubset  <- siteSelection(taxa = myData$taxa,
                               site = myData$site,
                               time_period = myData$time_period,
                               minL = 5,
                               minTP = 5,
                               LFirst = TRUE)
```

```
## Warning in errorChecks(taxa, site, time_period): 854 out of 15000
## observations will be removed as duplicates
```

```
head(myDataSubset)
```

```
##   taxa site time_period
## 1    e   A1  1981-07-11
## 2    j   A1  1981-07-11
## 3    a   A1  1981-07-11
## 4    n   A1  1981-07-11
## 5    t   A1  1981-07-11
## 6    x   A1  1982-02-23
```

```
nrow(myDataSubset)
```

```
## [1] 4217
```

Running Reporting Rate Models

Once you have subset your data using the above functions (or perhaps not at all) the reporting rate models can be applied using the function `reportingtRateModel`. This function offers flexibility in the model you wish to fit, allowing the user to specify whether list length and site should be used as covariates, whether over-dispersion should be used, and whether the family should be binomial or Bernoulli.

```
# Run the reporting rate model
RR_out <- reportingRateModel(taxa = myData$taxa,
                             site = myData$site,
                             time_period = myData$time_period,
                             list_length = TRUE,
                             site_effect = TRUE,
                             overdispersion = FALSE,
                             family = 'Bernoulli',
                             print_progress = TRUE)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period, : 854 out of 15000 observations will be removed as duplicates
```

```
## Modelling 1 - Species 1 of 26
## Modelling 2 - Species 2 of 26
## Modelling 3 - Species 3 of 26
## Modelling 4 - Species 4 of 26
## Modelling 5 - Species 5 of 26
```

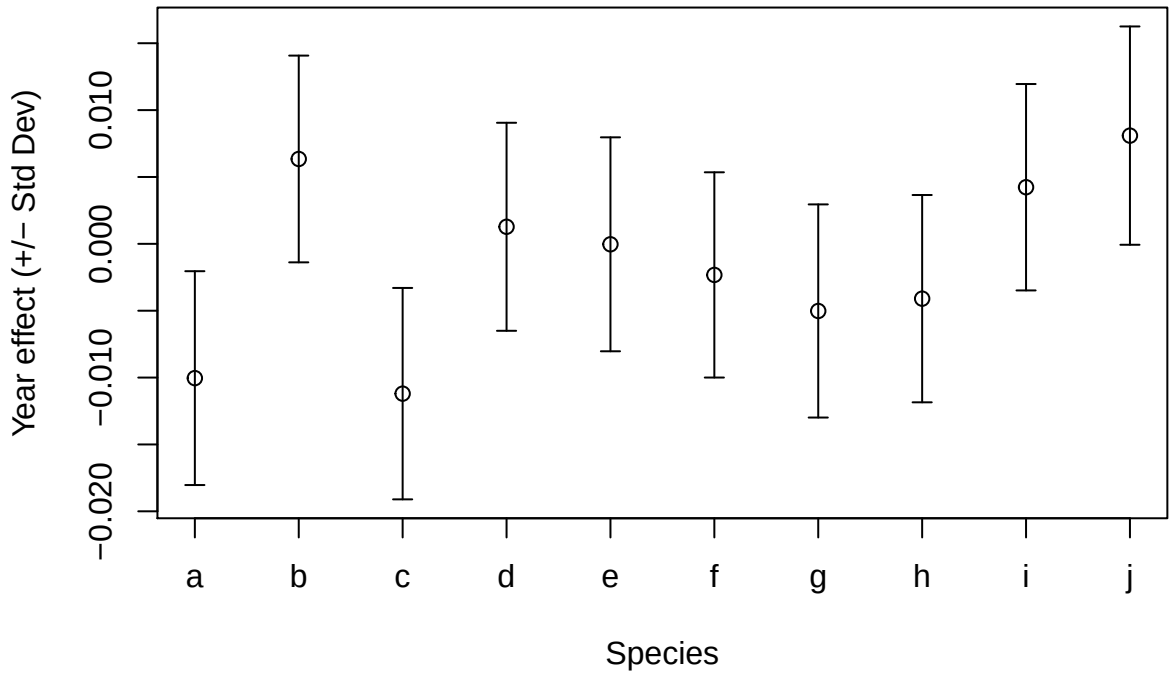
```
## Modelling 6 - Species 6 of 26
## Modelling 7 - Species 7 of 26
## Modelling 8 - Species 8 of 26
## Modelling 9 - Species 9 of 26
## Modelling 10 - Species 10 of 26
## Modelling 11 - Species 11 of 26
## Modelling 12 - Species 12 of 26
## Modelling 13 - Species 13 of 26
## Modelling 14 - Species 14 of 26
## Modelling 15 - Species 15 of 26
## Modelling 16 - Species 16 of 26
## Modelling 17 - Species 17 of 26
## Modelling 18 - Species 18 of 26
## Modelling 19 - Species 19 of 26
## Modelling 20 - Species 20 of 26
## Modelling 21 - Species 21 of 26
## Modelling 22 - Species 22 of 26
## Modelling 23 - Species 23 of 26
## Modelling 24 - Species 24 of 26
## Modelling 25 - Species 25 of 26
## Modelling 26 - Species 26 of 26
```

```
# Let's have a look at the data that is returned
str(RR_out)
```

```
## 'data.frame':    26 obs. of  14 variables:
##  $ species_name       : Factor w/ 26 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ intercept.estimate : num  -3.1 -2.97 -3.31 -3.16 -3.28 ...
##  $ year.estimate      : num  -1.00e-02 6.35e-03 -1.12e-02 1.28e-03 -3.38e-05 ...
##  $ listlength.estimate: num  0.302 0.289 0.375 0.345 0.358 ...
##  $ intercept.stderror : num  0.113 0.112 0.113 0.11 0.114 ...
##  $ year.stderror      : num  0.00799 0.00773 0.0079 0.00778 0.008 ...
##  $ listlength.stderror: num  0.0281 0.0273 0.0279 0.0274 0.0281 ...
##  $ intercept.zvalue   : num  -27.4 -26.5 -29.2 -28.8 -28.9 ...
##  $ year.zvalue        : num  -1.2559 0.82092 -1.41715 0.16402 -0.00423 ...
##  $ listlength.zvalue  : num  10.7 10.6 13.4 12.6 12.7 ...
##  $ intercept.pvalue   : num  4.22e-165 3.02e-155 2.03e-187 7.80e-182 1.76e-183 ...
##  $ year.pvalue        : num  0.209 0.412 0.156 0.87 0.997 ...
##  $ listlength.pvalue  : num  8.42e-27 4.03e-26 3.68e-41 3.17e-36 3.39e-37 ...
##  $ observations       : num  511 558 535 554 520 568 523 558 561 508 ...
##  - attr(*, "intercept_year")= num 1990
##  - attr(*, "min_year")= num -9.5
##  - attr(*, "max_year")= num 9.5
##  - attr(*, "nVisits")= int 4700
##  - attr(*, "model_formula")= chr "taxa ~ year + listLength + (1|site)"
```

```
# We could plot these to see the species trends
with(RR_out[1:10,],
     # Plot graph
     {plot(x = 1:10, y = year.estimate,
           ylim = range(c(year.estimate - year.stderror,
                          year.estimate + year.stderror)),
           ylab = 'Year effect (+/- Std Dev)',
```

```
      xlab = 'Species',
      xaxt="n")
  # Add x-axis with species names
  axis(1, at=1:10, labels = species_name[1:10])
  # Add the error bars
  arrows(1:10, year.estimate - year.stderror,
         1:10, year.estimate + year.stderror,
         length=0.05, angle=90, code=3)}
  )
```



The returned object is a data frame with one row per species. Each column gives information on an element of the model output including covariate estimates, standard errors and p-values. This object also has some attributes giving the year that was chosen as the intercept, the number of visits in the dataset and the model formula used.

Using these functions it is possible to recreate the 'Well-sampled sites' method that is presented in Roy et al (2012). This is made available in the function `WSS` which is a simple wrapper around `siteSelection` and `reportingratemodel`.

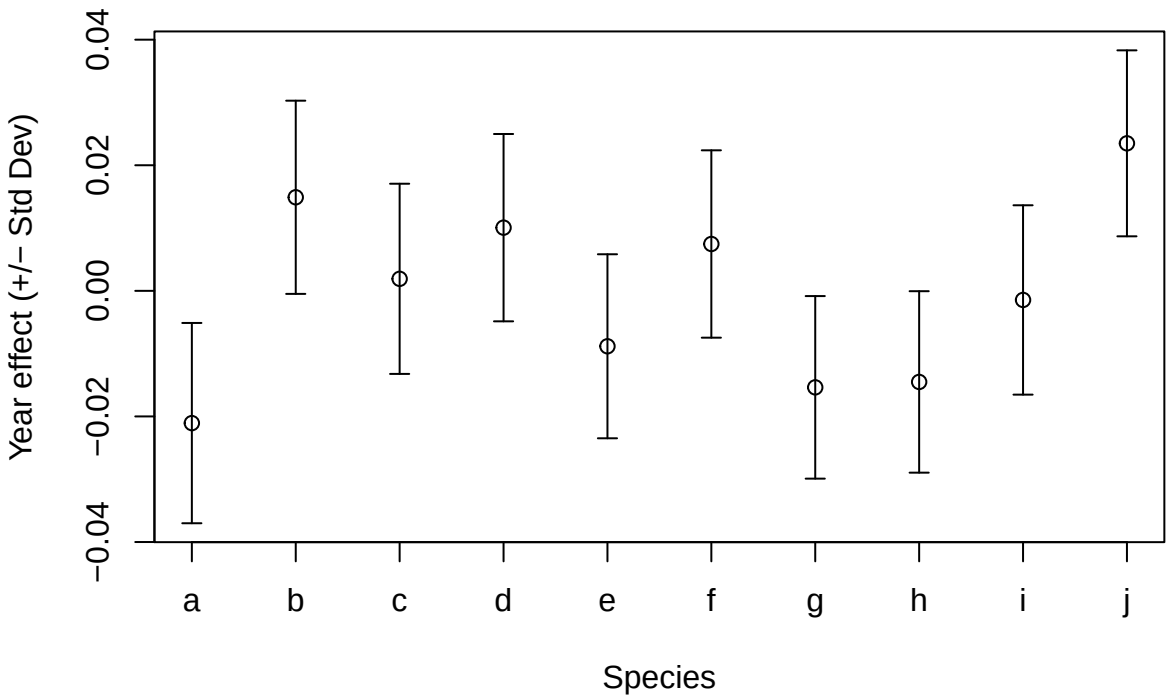
```
# Run our data through the well-sampled sites function
WSS_out <- WSS(taxa = myData$taxa,
               site = myData$site,
               time_period = myData$time_period,
               minL = 5,
               minTP = 5,
               print_progress = FALSE)
```

```
## Warning in errorChecks(taxa, site, time_period): 854 out of 15000
## observations will be removed as duplicates
```

```
# The data is returned in the same format as from reportingRateModel
str(WSS_out)
```

```
## 'data.frame':    26 obs. of  10 variables:
##  $ species_name      : Factor w/ 26 levels "a","b","c","d",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ intercept.estimate: num  -1.51 -1.38 -1.31 -1.24 -1.23 ...
##  $ year.estimate     : num  -0.02106 0.0149 0.00191 0.01006 -0.00883 ...
##  $ intercept.stderror: num  0.1093 0.099 0.0992 0.1016 0.0875 ...
##  $ year.stderror     : num  0.0159 0.0154 0.0151 0.0149 0.0147 ...
##  $ intercept.zvalue  : num  -13.8 -14 -13.2 -12.2 -14 ...
##  $ year.zvalue       : num  -1.321 0.968 0.126 0.675 -0.602 ...
##  $ intercept.pvalue  : num  3.09e-43 2.80e-44 4.51e-40 2.80e-34 1.28e-44 ...
##  $ year.pvalue       : num  0.187 0.333 0.899 0.5 0.547 ...
##  $ observations      : num  138 152 160 171 169 164 173 176 163 172 ...
##  - attr(*, "intercept_year")= num 1990
##  - attr(*, "min_year")= num -9.5
##  - attr(*, "max_year")= num 9.5
##  - attr(*, "nVisits")= int 747
##  - attr(*, "model_formula")= chr "cbind(successes, failures) ~ year + (1|site)"
##  - attr(*, "minL")= num 5
##  - attr(*, "minTP")= num 5
```

```
# We can plot these and see that we get different results to our
# previous analysis since this time the method includes subsetting
with(WSS_out[1:10,],
     # Plot graph
     {plot(x = 1:10, y = year.estimate,
           ylim = range(c(year.estimate - year.stderror,
                          year.estimate + year.stderror)),
           ylab = 'Year effect (+/- Std Dev)',
           xlab = 'Species',
           xaxt="n")
     # Add x-axis with species names
     axis(1, at=1:10, labels = species_name[1:10])
     # Add the error bars
     arrows(1:10, year.estimate - year.stderror,
            1:10, year.estimate + year.stderror,
            length=0.05, angle=90, code=3)}
     )
```



Occupancy models

Occupancy models were found by Isaac et al (2014) to be one of the better tools for analysing species occurrence data typical of citizen science projects. This model models the occupancy process separately from the detection process, but we will not go in to the details of the model here since there is a growing literature about occupancy models, how and when they should be used. Here we focus on how the occupancy model discussed in Isaac et al 2014 is implemented in `sparta`.

This function works in a very similar fashion to that of the previous functions we have discussed. The data it takes is 'What, where, when' as in other functions, however here we have the option to specify which species we wish to model. This feature has been added as occupancy models are computationally intensive. The parameters of the function allow you control over the number of iterations, burnin, thinning, the number of chains, the seed and for advanced users there is also the possibility to pass in your own BUGS script.

```
# Here is our data
str(myData)
```

```
## 'data.frame':    15000 obs. of  4 variables:
##  $ taxa       : Factor w/ 26 levels "a","b","c","d",..: 20 19 22 10 17 5 23 7 21 10 ...
##  $ site       : Factor w/ 50 levels "A1","A10","A11",..: 15 20 35 21 3 15 8 23 15 48 ...
##  $ time_period: Date, format: "1989-04-12" "1997-06-16" ...
##  $ tp         : int  2 4 4 4 4 2 3 1 2 2 ...
```

```r
# Run an occupancy model for three species
# Here we use very small number of iterations
# to avoid a long run time
system.time({
occ_out <- occDetModel(taxa = myData$taxa,
                       site = myData$site,
                       time_period = myData$time_period,
                       species_list = c('a','b','c','d'),
                       write_results = FALSE,
                       n_iterations = 50,
                       burnin = 15,
                       n_chains = 3,
                       thinning = 3,
                       seed = 123)
})
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period): 854 out of 15000 observations will be removed as duplicates

##
## ###
## Modeling a - 1 of 4 taxa

## module glm loaded

## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
##    Graph Size: 40353
##
## Initializing model
##
##
## ###
## Modeling b - 2 of 4 taxa
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
##    Graph Size: 40349
##
## Initializing model
##
##
## ###
## Modeling c - 3 of 4 taxa
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
##    Graph Size: 40359
##
## Initializing model
##
##
```

```
## ###
## Modeling d - 4 of 4 taxa
## Compiling model graph
##    Resolving undeclared variables
##    Allocating nodes
##    Graph Size: 40353
##
## Initializing model
```

```
##    user  system elapsed
##   14.07    0.02   14.29
```

```
# This took about 13 seconds
# Lets look at the results
## The object returned is a list with one element for each species
names(occ_out)
```

```
## [1] "a" "b" "c" "d"
```

```
# Each of these is an object of class 'occDet'
class(occ_out$a)
```

```
## [1] "occDet"
```

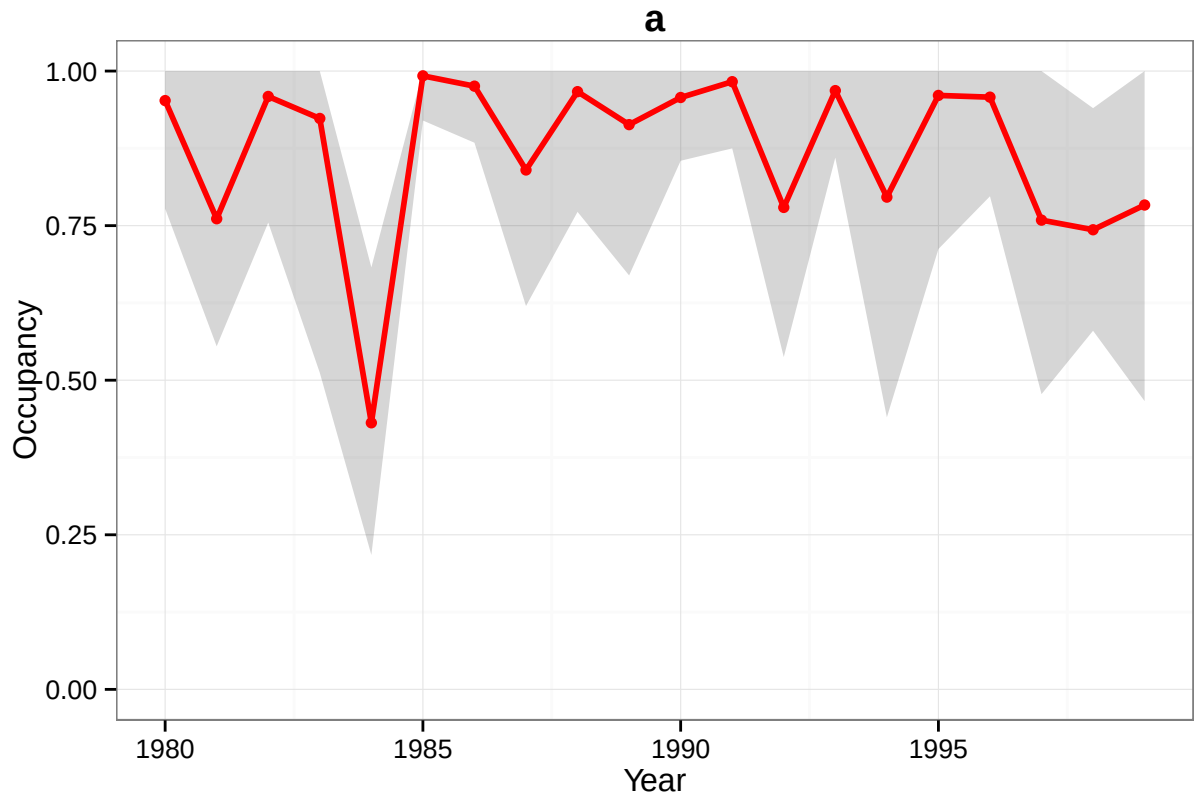
```
# Inside these elements is the information of interest
names(occ_out$a)
```

```
## [1] "model"              "BUGSoutput"         "parameters.to.save"
## [4] "model.file"         "n.iter"             "DIC"
## [7] "SPP_NAME"           "min_year"           "max_year"
```

```
# Of particular interest to many users will be the summary
# data in the BUGSoutput
head(occ_out$a$BUGSoutput$summary)
```

```
##                    mean          sd         2.5%          25%          50%
## LL.p          0.7780612  0.05488541    0.6957462    0.7395210    0.7682999
## deviance   3000.8725203 57.82779914 2875.0750613 2958.2829908 3008.2590013
## fit         958.1581660 34.37842659  906.1633694  929.4906754  949.5410807
## fit.new     960.0216557 47.61608746  888.6014168  919.0263640  956.9056626
## mean_early    0.9066667  0.07748425    0.7290814    0.8716619    0.9233273
## mean_late     0.7618519  0.09923659    0.5724957    0.6800000    0.7666667
##                    75%        97.5%      Rhat n.eff
## LL.p          0.8251476    0.8747336 0.9842896    36
## deviance   3045.1440744 3086.2091388 0.9750173    36
## fit         990.4513754 1014.8671315 0.9636719    36
## fit.new     994.6143954 1055.1697059 0.9757828    36
## mean_early    0.9666667    1.0000000 1.1865618    14
## mean_late     0.8299800    0.9200000 1.2333444    12
```

```
# We have included a plotting feature for objects of class
# occDet which provides a useful visualisation of the trend
# in occupancy over time
plot(occ_out$a)
```



He we have run a small example but in reality these models are usually run for many thousands of iterations, making the analysis of more than a handful of species impractical. For those with access to the necessary facilities it is possible to parallelise across species. To do this we use a pair of functions that are used internally by `occDetModel`. These are `formatOccData` which is used to format our occurrence data into the format needed by JAGS, and `occDetFunc`, the function which undertakes the modelling.

```
# First format our data
formattedOccData <- formatOccData(taxa = myData$taxa,
                                  site = myData$site,
                                  time_period = myData$time_period)
```

```
## Warning in errorChecks(taxa = taxa, site = site, time_period =
## time_period): 854 out of 15000 observations will be removed as duplicates
```

```
# This is a list of two elements
names(formattedOccData)
```

```
## [1] "spp_vis"    "occDetdata"
```

formatOccData returns a list of length 2; the first element 'spp_vis' is a data.frame with visit (unique combination of site and time period) in the first column and taxa for all the following columns. Values in taxa columns are either TRUE or FALSE depending on whether they were observed on that visit.

```
# Lets have a look at spp_vis
head(formattedOccData$spp_vis[,1:5])
```

```
##           visit     a     b     c     d
## 1 A101980-02-17  TRUE FALSE FALSE FALSE
## 2 A101980-04-13 FALSE FALSE FALSE FALSE
## 3 A101980-04-21 FALSE FALSE FALSE FALSE
## 4 A101980-08-28 FALSE FALSE FALSE FALSE
## 5 A101980-11-03 FALSE FALSE  TRUE  TRUE
## 6 A101981-02-08 FALSE FALSE FALSE FALSE
```

The second element ('occDetData') is a data frame giving the site, list length (the number of species observed on a visit) and year for each visit.

```
# Lets have a look at occDetData
head(formattedOccData$occDetdata)
```

```
##            visit site L year
## 1  A101980-02-17  A10 5 1980
## 6  A101980-04-13  A10 2 1980
## 8  A101980-04-21  A10 1 1980
## 9  A101980-08-28  A10 5 1980
## 14 A101980-11-03  A10 5 1980
## 19 A101981-02-08  A10 1 1981
```

With our data in the correct format this can now go into the modelling function

```
# Use the occupancy modelling function to parrellise the process
# Here we are going to use the package snowfall
library(snowfall)
```

```
## Loading required package: snow
```

```
# I have 4 cpus on my PC so I set cpus to 4
# when I initialise the cluster
sfInit(parallel = TRUE, cpus=4 )
```

```
## Warning in searchCommandline(parallel, cpus = cpus, type = type,
## socketHosts = socketHosts, : Unknown option on commandline:
## rmarkdown::render('W:/PYWELL_SHARED/Pywell Projects/BRC/Tom August/R
## Packages/Trend analyses/sparta/vignette/sparta_vignette.Rmd', encoding
```

```
## R Version:  R version 3.2.0 (2015-04-16)
```

```
## snowfall 1.84-6 initialized (using snow 0.3-13): parallel execution on 4 CPUs.
```

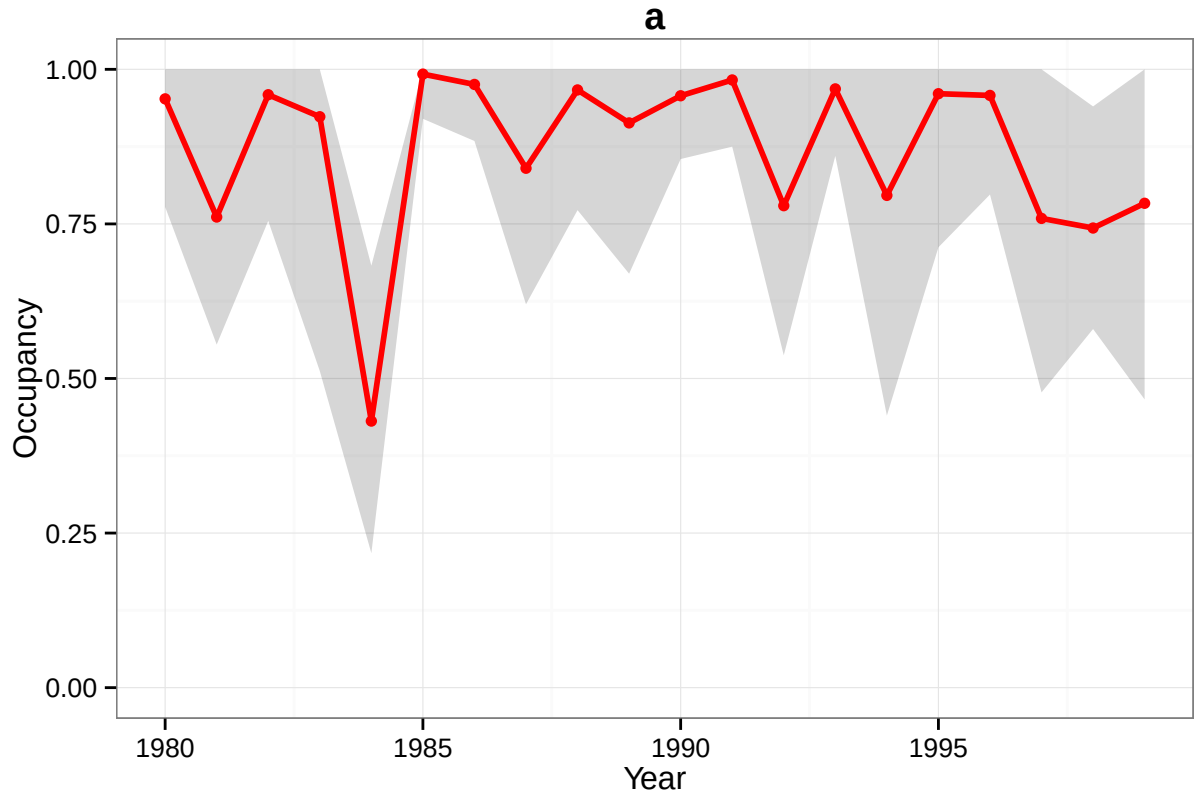
```
# Export my data to the cluster
sfExport('formattedOccData')

# I create a function that takes a species name and runs my model
myfunction <- function(taxa_name){

  library(sparta)

  occ_out <- occDetFunc(taxa_name = taxa_name,
                        n_iterations = 50,
                        burnin = 15,
                        occDetdata = formattedOccData$occDetdata,
                        spp_vis = formattedOccData$spp_vis,
                        write_results = FALSE,
                        seed = 123)
}

# I then run this in parallel
system.time({
para_out <- sfClusterApplyLB(c('a','b','c','d'), myfunction)
names(para_out) <- c('a','b','c','d')
})
```

```
##    user  system elapsed
##     0.0     0.0     7.3
```

```
# This takes about half the time of the
# serial version we ran earlier, and the resulting object
# is the same (since we set the random seed to be the same
# in each)
head(para_out$a$BUGSoutput$summary)
```

```
##                    mean          sd         2.5%          25%          50%
## LL.p          0.7780612  0.05488541    0.6957462    0.7395210    0.7682999
## deviance   3000.8725203 57.82779914 2875.0750613 2958.2829908 3008.2590013
## fit         958.1581660 34.37842659  906.1633694  929.4906754  949.5410807
## fit.new     960.0216557 47.61608746  888.6014168  919.0263640  956.9056626
## mean_early    0.9066667  0.07748425    0.7290814    0.8716619    0.9233273
## mean_late     0.7618519  0.09923659    0.5724957    0.6800000    0.7666667
##                     75%        97.5%      Rhat n.eff
## LL.p          0.8251476    0.8747336 0.9842896    36
## deviance   3045.1440744 3086.2091388 0.9750173    36
## fit         990.4513754 1014.8671315 0.9636719    36
## fit.new     994.6143954 1055.1697059 0.9757828    36
## mean_early    0.9666667    1.0000000 1.1865618    14
## mean_late     0.8299800    0.9200000 1.2333444    12
```

```
plot(para_out$a)
```



This same approach can be used on cluster computers, which can have hundreds of processors, to dramatically reduce run times.

Frescalo

The frescalo method is outlined in Hill (2012) and is a means to account for both spatial and temporal bias. This method was shown by Isaac et al (2014) to be a good method for data that is aggregated into time periods such as when comparing atlases. The frescalo method is run using a .exe, you will need to download this first (https://github.com/BiologicalRecordsCentre/frescalo) and make a note of the directory you place it in, we will need that in a moment.

Again we will assume that your data is in a 'what, where, when' format similar to that we used in the previous method:

```
head(myData)
```

```
##   taxa site time_period tp
## 1    t  A22  1989-04-12  2
## 2    s  A27  1997-06-16  4
## 3    v  A40  1996-09-10  4
## 4    j  A28  1998-09-21  4
## 5    q  A11  1995-07-24  4
## 6    e  A22  1985-09-10  2
```

Frescalos requirements in terms of data structure and types is a little different to that we have seen in other functions. Firstly the entire dataframe is passed in as an argument called `Data`, and the column names of

your various elements (taxa, site, etc) are given as other arguments. Secondly frescalo require that the 'when' component is either a column of year or two columns, one of 'start date' and one of 'end date'. Our data as presented above does not fit into this format so first we must reformat it. In our situation the simplest thing to do is to add a column giving the year. Since frescalo aggregates across time periods (often decades or greater) this loss of temporal resolution is not an issue.

```
# Add a year column
myData$year <- as.numeric(format(myData$time_period, '%Y'))
head(myData)
```

```
##   taxa site time_period tp year
## 1    t  A22  1989-04-12  2 1989
## 2    s  A27  1997-06-16  4 1997
## 3    v  A40  1996-09-10  4 1996
## 4    j  A28  1998-09-21  4 1998
## 5    q  A11  1995-07-24  4 1995
## 6    e  A22  1985-09-10  2 1985
```

Now we have our data in the correct format for frescalo there is one other major component we need, a weights file. You can find out more about the weights file and what it is used for in the original paper (Hill, 2012). In short this file outlines the similarity between sites in your dataset. This information is used to weight the analysis of each site accordingly. If you are undertaking this analysis in the UK at 10km square resolution there are some built in weights files you can use. Some of these weights files use the UK landcover map instead of floristic similarity (as used in Hill (2012)). You can find out more about these in the frescalo help file.

For the sake of demonstration let us assume that you do not have a weights file for your analysis, or that you want to create your own. To create a weights file you need two things, a measure of physical distance between your sites and a measure of similarity. In the original paper this similarity measure was floristic similarity, but it could also be habitat similarity or whatever is relevant for the taxa you are studying.

```
# I'm going to create some made up data
mySites <- unique(myData$site)

# Build a table of distances
myDistances <- merge(mySites, mySites)
# add random distances
myDistances$dist <- runif(n = nrow(myDistances), min = 10, max = 10000)
# to be realistic the distance from a site to itself should be 0
myDistances$dist[myDistances$x == myDistances$y] <- 0
head(myDistances)
```

```
##     x   y      dist
## 1 A22 A22    0.0000
## 2 A27 A22 4664.9649
## 3 A40 A22 2667.0667
## 4 A28 A22 8579.6989
## 5 A11 A22  467.8534
## 6 A16 A22 4427.5787
```

```
# Build a table of attributes
# This can be done in numerous ways, here is one example
# Lets say I have habitat data for all my sites
```

```
myHabitatData <- data.frame(site = mySites,
                            grassland = runif(length(mySites), 0, 1),
                            woodland = runif(length(mySites), 0, 1),
                            heathland = runif(length(mySites), 0, 1),
                            urban = runif(length(mySites), 0, 1),
                            freshwater = runif(length(mySites), 0, 1))

# This pretend data is supposed to be proportional cover so lets
# make sure each row sums to 1
multiples <- apply(myHabitatData[,2:6], 1, sum)

for(i in 1:length(mySites)){

  myHabitatData[i,2:6] <- myHabitatData[i,2:6]/multiples[i]

}

# Here is our pretend habitat data, and the rows sum to 1
head(myHabitatData)
```

```
##   site  grassland   woodland  heathland     urban freshwater
## 1  A22 0.18208661 0.22393964 0.46698459 0.0906582 0.03633096
## 2  A27 0.32849480 0.26680443 0.08950346 0.2802727 0.03492459
## 3  A40 0.17227774 0.30871279 0.21966734 0.2337297 0.06561242
## 4  A28 0.09382081 0.14285823 0.03478153 0.4741866 0.25435288
## 5  A11 0.21121112 0.08105993 0.28685580 0.1579623 0.26291085
## 6  A16 0.06219117 0.24826969 0.35240575 0.2898218 0.04731158
```

```
# With our distance and habitat tables in hand we can
# use the createWeights function to build our weights file
# I have changed the defualts of dist_sub and sim_sub since
# we have a very small example dataset of only 50 sites
myWeights <- createWeights(distances = myDistances,
                           attributes = myHabitatData,
                           dist_sub = 20,
                           sim_sub = 10)
```

```
## Creating similarity distance table...Complete
## Creating weights file...
## 0%
## 10%
## 20%
## 30%
## 40%
## 50%
## 60%
## 70%
## 80%
## 90%
## 100%
## Complete
```

```
head(myWeights)
```

```
##   target neighbour weight
## 1    A22       A10 0.0001
## 2    A22       A11 0.0000
## 3    A22       A12 0.0053
## 4    A22       A13 0.0835
## 5    A22       A14 0.0650
## 6    A22       A22 1.0000
```

The `createWeights` function follows the procedure outlined in Hill (2012) for creating weights and more information can be found in the help file of the function. With our data and weights file we are now ready to proceed with frescalo. As with other functions frescalo can take a range of additional arguments which you can see by entering `?frescalo` at the console, here we will do a minimal example.

```
# First we need to enter the location where we placed the .exe
# In my case I saved it to my documents folder
myFrescaloPath <- 'C:/Users/tomaug/Documents/Frescalo_3a_windows.exe'

# I then want to set up the time periods I want to analyse
# Here I say I want to compare 1980-89 to 1990-99
myTimePeriods <- data.frame(start=c(1980,1990),end=c(1989,1999))
head(myTimePeriods)
```

```
##   start  end
## 1  1980 1989
## 2  1990 1999
```

```
# I also need to specify where I want my results to be saved
# I'm going to save it in a folder in my working directory
myFolder <- '~/myFolder'

# Simple run of frescalo
frescalo_results <- frescalo(Data = myData,
                             frespath = myFrescaloPath,
                             time_periods = myTimePeriods,
                             site_col = 'site',
                             sp_col = 'taxa',
                             year = 'year',
                             Fres_weights = myWeights,
                             sinkdir = myFolder)
```

```
## Warning in frescalo(Data = myData, frespath = myFrescaloPath, time_periods
## = myTimePeriods, : sinkdir already contains frescalo output. New data saved
## in ~/myFolder/frescalo_150604(1)

##
## SAVING DATA TO FRESCALO WORKING DIRECTORY
## ********************
##
##
## RUNNING FRESCALO
## ********************
```

```
## Warning in run_fresc_file(fres_data = Data, output_dir = fresoutput,
## frescalo_path = frespath, : Your value of phi (0.74) is smaller than the
## 98.5 percentile of input phi (1). It is reccommended your phi be similar to
## this value. For more information see Hill (2011) reference in frescalo help
## file

## Building Species List - Complete
## Outputting Species Results
##  Species 1 of 26 - a - 04/06/2015 13:46:14
##  Species 2 of 26 - b - 04/06/2015 13:46:14
##  Species 3 of 26 - c - 04/06/2015 13:46:14
##  Species 4 of 26 - d - 04/06/2015 13:46:14
##  Species 5 of 26 - e - 04/06/2015 13:46:14
##  Species 6 of 26 - f - 04/06/2015 13:46:14
##  Species 7 of 26 - g - 04/06/2015 13:46:14
##  Species 8 of 26 - h - 04/06/2015 13:46:14
##  Species 9 of 26 - i - 04/06/2015 13:46:14
##  Species 10 of 26 - j - 04/06/2015 13:46:14
##  Species 11 of 26 - k - 04/06/2015 13:46:14
##  Species 12 of 26 - l - 04/06/2015 13:46:14
##  Species 13 of 26 - m - 04/06/2015 13:46:14
##  Species 14 of 26 - n - 04/06/2015 13:46:14
##  Species 15 of 26 - o - 04/06/2015 13:46:14
##  Species 16 of 26 - p - 04/06/2015 13:46:14
##  Species 17 of 26 - q - 04/06/2015 13:46:14
##  Species 18 of 26 - r - 04/06/2015 13:46:14
##  Species 19 of 26 - s - 04/06/2015 13:46:14
##  Species 20 of 26 - t - 04/06/2015 13:46:14
##  Species 21 of 26 - u - 04/06/2015 13:46:14
##  Species 22 of 26 - v - 04/06/2015 13:46:14
##  Species 23 of 26 - w - 04/06/2015 13:46:14
##  Species 24 of 26 - x - 04/06/2015 13:46:14
##  Species 25 of 26 - y - 04/06/2015 13:46:14
##  Species 26 of 26 - z - 04/06/2015 13:46:14
## [1] "frescalo complete"
```

We get a warning from this analysis that our value of phi is too low. In this case this is because our simulated data suggests every species is found on every site in our time periods. This is a little unreaslistic but should you get a similar warning with your data you might want to consult Hill (2012) and change your input value of phi.

The object that is returned (`frescalo_results` in my case) is an object of class `frescalo`. this means there are a couple of special methds we can use with it.

```
# Using 'summary' give a quick overview of our data
# This can be useful to double check that your data was read in correctly
summary(frescalo_results)
```

```
##  Actual numbers in data
##      Number of samples            50
##      Number of species            26
##      Number of time periods        2
##      Number of observations     2593
##      Neighbourhood weights       500
```

```
##      Benchmark exclusions          0
##      Filter locations included     0
```

```
# Using 'print' we get a preview of the results
print(frescalo_results)
```

```
##
## Preview of $paths - file paths to frescalo log, stats, freq, trend .csv files:
##
## [1] "~/myFolder/frescalo_150604(1)/Output/Log.txt"
## [2] "~/myFolder/frescalo_150604(1)/Output/Stats.csv"
## [3] "~/myFolder/frescalo_150604(1)/Output/Freq.csv"
## [4] "~/myFolder/frescalo_150604(1)/Output/Trend.csv"
## [5] "~/myFolder/frescalo_150604(1)/Output/Freq_quickload.txt"
##
##
## Preview of $trend - trends file, giving the tfactor value for each species at each time period:
##
##   Species   Time TFactor StDev  X Xspt Xest N.0.00 N.0.98
## 1       a 1984.5   2.984 1.903 49   49 49.0     50      0
## 2       a 1994.5   2.944 1.860 49   49 49.0     50      0
## 3       j 1984.5   2.984 1.903 49   49 49.0     50      0
## 4       j 1994.5   4.741 4.463 50   50 49.9     50      0
## 5       k 1984.5   4.824 4.611 50   50 49.9     50      0
## 6       k 1994.5   4.741 4.463 50   50 49.9     50      0
##
##
## Preview of $stat - statistics for each hectad in the analysis:
##
##   Location Loc_no No_spp Phi_in Alpha Wgt_n2 Phi_out Spnum_in Spnum_out
## 1       A1      1     26      1  0.12   3.19    0.74       26      19.2
## 2      A10      2     26      1  0.12   3.54    0.74       26      19.2
## 3      A11      3     26      1  0.12   3.27    0.74       26      19.2
## 4      A12      4     26      1  0.12   2.16    0.74       26      19.2
## 5      A13      5     26      1  0.12   2.36    0.74       26      19.2
## 6      A14      6     26      1  0.12   3.00    0.74       26      19.2
##   Iter
## 1   31
## 2   31
## 3   31
## 4   31
## 5   31
## 6   31
##
##
## Preview of $freq - rescaled frequencies for each location and species:
##
##   Location Species Pres Freq  Freq1 SDFrq1 Rank Rank1
## 1       A1       a    1    1 0.7399 0.2384    1 0.052
## 2       A1       j    1    1 0.7399 0.2384    2 0.104
## 3       A1       k    1    1 0.7399 0.2384    3 0.156
## 4       A1       l    1    1 0.7399 0.2384    4 0.208
## 5       A1       m    1    1 0.7399 0.2384    5 0.260
## 6       A1       n    1    1 0.7399 0.2384    6 0.312
```

```
##
##
## Preview of $log - log file:
##
##     Number of species            26
##     Number of time periods        2
##     Number of observations     2593
##     Neighbourhood weights       500
##     Benchmark exclusions          0
##     Filter locations included     0
##
##
##  98.5 percentile of input phi  1.00
##  Target value of phi           0.74
##
##
##
##
## Preview of $lm_stats - trends in tfactor over time:
##
##    SPECIES NAME       b          a b_std_err b_tval b_pval a_std_err
## 1       S1    a -0.0040   10.92200        NA     NA     NA        NA
## 12      S2    b -0.0083   21.29535        NA     NA     NA        NA
## 20      S3    c -0.0083   21.29535        NA     NA     NA        NA
## 21      S4    d -0.0083   21.29535        NA     NA     NA        NA
## 22      S5    e  0.1757 -345.69265        NA     NA     NA        NA
## 23      S6    f -0.0083   21.29535        NA     NA     NA        NA
##    a_tval a_pval adj_r2 r2 F_val F_num_df F_den_df   Ymin   Ymax
## 1      NA     NA     NA  1    NA        1        0 1984.5 1994.5
## 12     NA     NA     NA  1    NA        1        0 1984.5 1994.5
## 20     NA     NA     NA  1    NA        1        0 1984.5 1994.5
## 21     NA     NA     NA  1    NA        1        0 1984.5 1994.5
## 22     NA     NA     NA  1    NA        1        0 1984.5 1994.5
## 23     NA     NA     NA  1    NA        1        0 1984.5 1994.5
##          Z_VAL SIG_95
## 1  -0.01503185  FALSE
## 12 -0.01293411  FALSE
## 20 -0.01293411  FALSE
## 21 -0.01293411  FALSE
## 22  0.36213507  FALSE
## 23 -0.01293411  FALSE
```

```
# There is a lot of information here and you can read more about
# what these data mean by looking at the frescalo help file
# The files detailed in paths are also in the object returned
frescalo_results$paths
```

```
## [1] "~/myFolder/frescalo_150604(1)/Output/Log.txt"
## [2] "~/myFolder/frescalo_150604(1)/Output/Stats.csv"
## [3] "~/myFolder/frescalo_150604(1)/Output/Freq.csv"
## [4] "~/myFolder/frescalo_150604(1)/Output/Trend.csv"
## [5] "~/myFolder/frescalo_150604(1)/Output/Freq_quickload.txt"
```

```r
names(frescalo_results)
```

```
## [1] "paths"    "trend"    "stat"     "freq"     "log"      "lm_stats"
```

```r
# However we additionally get some model results in our returned object
# under '$lm_stats'
```

The results from frescalo may seem complex at first and I suggest reading the Value section of the frescalo help file for details. In brief: `frescalo_results$paths` lists the file paths of the raw data files for `$log`, `$stat`, `$freq` and `$trend`, in that order. `frescalo_results$trend` is a data.frame providing the list of time factors for each species-timeperiod. `frescalo_results$stat` is a data.frame giving details about sites such as estimated species richness. `frescalo_results$freq` is a data.frame of the species frequencies, that is the probabilities that a species was present at a certain location. `frescalo_results$log`, a simple report of the console output from the .exe. `frescalo_results$lm_stats` is a dataframe giving the results of a linear regression of Tfactors for each species when more than two time periods are used, if only 2 time periods are used (as in our example) the linear modeling section of this data.frame is filled with NAs and a z-test is performed instead (results are given in the last columns).

```r
# Lets look at some results fo the first three species
frescalo_results$lm_stats[1:3, c('NAME','Z_VAL','SIG_95')]
```

```
##    NAME       Z_VAL SIG_95
## 1     a -0.01503185  FALSE
## 12    b -0.01293411  FALSE
## 20    c -0.01293411  FALSE
```

```r
# None of these have a significant change using a z-test
# Lets look at the raw data
frescalo_results$trend[frescalo_results$trend$Species %in% c('a', 'b', 'c'),
                       c('Species', 'Time', 'TFactor', 'StDev')]
```

```
##    Species   Time TFactor StDev
## 1        a 1984.5   2.984 1.903
## 2        a 1994.5   2.944 1.860
## 23       b 1984.5   4.824 4.611
## 24       b 1994.5   4.741 4.463
## 39       c 1984.5   4.824 4.611
## 40       c 1994.5   4.741 4.463
```

```r
# We can see from these results that the big standard deviations on
# the tfactor values means there is no real difference between the
# two time periods
```

References

1. Hill, M.O. (2012) Local frequency as a key to interpreting species occurrence data when recording effort is not known. Methods Ecol. Evol. 3, 195–205
2. Isaac, N.J.B. et al. (2014) Statistics for citizen science: extracting signals of change from noisy ecological data. Methods Ecol. Evol. 5, 1052–1060
3. Telfer, M.G. et al. (2002) A general method for measuring relative change in range size from biological atlas data. Biol. Conserv. 107, 99–109