Using the Pipeline

Tom August

23 June 2015

Introduction

This document shows how to use the indicator pipeline to create biodiversiy indicators such as those for DEFRA's Biodiversity Indicators in Your Pocket. The pipeline is shared in the form of an R package called 'BRCindicators' making it easy to share and maintain.

The functions in BRCindicators work with yearly estimates of species abundance or occurrence and aggregate them into an scaled indicator value with bootstrapped confidence intervals

This package has the input to read in the output of occupancy models created in the R package sparta, a package for estimating species trends from occurrence data. This package can be installed from Github and details of how to use the package are given in the package vignette. There is no need to use sparta to create your yearly species estimates as BRCindicators can also work with other data.

To create an indicator we first need to have species trends, let's create some using hte sparta R package.

Creating yearly estimates of occurrence in sparta

If you already have yearly estimates of abundance or occurrence for your species you can skip this stage. Here we show how you can create these estimates from raw species observation data using sparta.

```
# 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')
```

Let's assume you have some raw data already, we can under take occupancy modelling like this

```
# Load the sparta package
library(sparta)
```

For demonstration purposes I have a faked dataset of 8000 species observations. In my dataset the species are named after the letters in the alphabet. Below I show how I can use the Bayesian occupancy models in sparta to create yearly estimates of occurrence. For more information please see the vignette for sparta

```
# Preview of my data
head(myData)
```

```
##   taxa site time_period
## 1    r  A51  1970-01-14
## 2    v  A87  1980-09-29
## 3    e  A56  1996-04-14
## 4    z  A28  1959-01-16
## 5    r  A77  1970-09-21
## 6    x  A48  1990-02-25
```

```r
# 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): 94 out of 8000 observations will be removed as duplicates
```

```r
# Here we are going to use the package snowfall to parallelise
library(snowfall)

# 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/BRCindicators/vignette/using_the_pipeline.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.
```

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

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

  library(sparta)

  # Note that this will write you results to your computer
  # the location is set to your user folder
  occ_out <- occDetFunc(taxa_name = as.character(taxa_name),
                        n_iterations = 200,
                        burnin = 15,
                        occDetdata = formattedOccData$occDetdata,
                        spp_vis = formattedOccData$spp_vis,
                        write_results = TRUE,
                        output_dir = '~/Testing_indicator_pipe',
                        seed = 123)
}

# I then run this in parallel
system.time({
para_out <- sfClusterApplyLB(unique(myData$taxa), occ_mod_function)
})
```

```
##    user  system elapsed
##    0.48    0.08  160.42
```

```
# Stop the cluster
sfStop()
```

```
##
## Stopping cluster
```

```
# We can see all the files this has created
list.files('~/Testing_indicator_pipe')
```

```
##  [1] "a.rdata"    "b.rdata"    "c.rdata"    "d.rdata"    "e.rdata"
##  [6] "f.rdata"    "g.rdata"    "h.rdata"    "i.rdata"    "j.rdata"
## [11] "k.rdata"    "l.rdata"    "m.rdata"    "myPlot.jpg" "n.rdata"
## [16] "o.rdata"    "p.rdata"    "q.rdata"    "r.rdata"    "s.rdata"
## [21] "t.rdata"    "u.rdata"    "v.rdata"    "w.rdata"    "x.rdata"
## [26] "y.rdata"    "z.rdata"
```

Summarising sparta output for indicator

Now that we have some species trends data to work with (no doubt you already have your own) we can use the first function in BRCindicators. This function reads in all the output files from sparta (which are quite large and complex) and returns a simple summary table that we can use for calculating the indicator. If you have done your analysis without using sparta you can skip to the next step.

```
library(BRCindicators)

# All we have to supply is the directory where out data is saved
# You will note this is the 'output_dir' passed to sparta above.
trends_summary <- summarise_occDet(input_dir = '~/Testing_indicator_pipe')
```

```
## Warning in summarise_occDet(input_dir = "~/Testing_indicator_pipe"): Not
## all files in ~/Testing_indicator_pipe are .rdata files, other file types
## have been ignored
```

```
## Loading data...done
```

```
# Lets see the summary
head(trends_summary[,1:5])
```

```
##      year         a         b         c         d
## [1,] 1950 0.7054839 0.6651075 0.5957527 0.5584946
## [2,] 1951 0.7965054 0.3516667 0.4296774 0.3935484
## [3,] 1952 0.5819355 0.5019892 0.2509677 0.3822043
## [4,] 1953 0.2910753 0.5570430 0.5482258 0.3884946
## [5,] 1954 0.4201613 0.3294624 0.6808602 0.2032258
## [6,] 1955 0.0727957 0.3516129 0.6152688 0.3051613
```

Returned from this function is a summary of the data as a matrix. In each row we have the year, specified in the first column, and each subsequent column is a species. The values in the table are the mean of the posterior for the predicted proportion of sites occupied, a measure of occurrence.

Calculating indicator values

The next step is to rescale the data so that the value for all species in the first year is the same. Once this is done we calculate the geometric mean across species for each year creating the indicator value. This function also accounts for species that have no data at the beginning of the dataset by entering them at the geometric mean for that year, this stops them dramatically changing the indicator value in the year they join the dataset. It also accounts for species that leave the dataset before the end by holding them at their last value. Finally limits to species values can be given, preventing extremely high or low values biasing the indicator.

The data I have generated in 'trends_summary' is very easy to work with but to show off what this function can do I'm going to mess it up a bit.

```
trends_summary[1:3, 'a'] <- NA
trends_summary[1:5, 'b'] <- NA
trends_summary[2:4, 'c'] <- 1000
trends_summary[45:50, 'd'] <- NA

# Let's have a look at these changes
head(trends_summary[,1:5])
```

```
##      year         a         b            c         d
## [1,] 1950        NA        NA    0.5957527 0.5584946
## [2,] 1951        NA        NA 1000.0000000 0.3935484
## [3,] 1952        NA        NA 1000.0000000 0.3822043
## [4,] 1953 0.2910753        NA 1000.0000000 0.3884946
## [5,] 1954 0.4201613        NA    0.6808602 0.2032258
## [6,] 1955 0.0727957 0.3516129    0.6152688 0.3051613
```

```
tail(trends_summary[,1:5])
```

```
##       year         a          b          c  d
## [45,] 1994 0.1611828 0.66306452 0.05370968 NA
## [46,] 1995 0.4450538 0.27967742 0.77758065 NA
## [47,] 1996 0.5575806 0.58489247 0.55989247 NA
## [48,] 1997 0.2262366 0.77596774 0.82107527 NA
## [49,] 1998 0.7444624 0.07543011 0.48854839 NA
## [50,] 1999 0.2617742 0.51596774 0.70860215 NA
```

Now that I have 'messed up' the data a bit we have two species with data missing at the beginning and one species with data missing at the end. We also have one species with some very high values.

Now lets run this through the rescaling function.

```
# Let's run this data through our scaling function (all defaults used)
rescaled_trends <- rescale_species(Data = trends_summary)

# Here is the result
head(rescaled_trends[,c('year', 'indicator', 'a', 'b', 'c', 'd')])
```

```
##      year indicator         a        b          c         d
## [1,] 1950 100.00000        NA       NA   100.0000 100.00000
## [2,] 1951 110.79406        NA       NA 10000.0000  70.46592
## [3,] 1952 125.02114        NA       NA 10000.0000  68.43473
```

```
## [4,] 1953 112.80089 112.80089       NA 10000.0000  69.56103
## [5,] 1954  94.57442 162.82582       NA   114.2857  36.38814
## [6,] 1955  99.41400  28.21064 99.59997   103.2759  54.63997
```

```
tail(rescaled_trends[,c('year', 'indicator', 'a', 'b', 'c', 'd')])
```

```
##       year indicator         a         b          c        d
## [45,] 1994  90.76508  62.46344 187.82361   9.015432 120.8895
## [46,] 1995 108.95097 172.47244  79.22309 130.520711 120.8895
## [47,] 1996 115.49406 216.08017 165.68013  93.980688 120.8895
## [48,] 1997 114.75943  87.67384 219.80525 137.821496 120.8895
## [49,] 1998  99.59398 288.50276  21.36678  82.005234 120.8895
## [50,] 1999 102.69732 101.44580 146.15610 118.942334 120.8895
```

You can see that species 'a' and 'b' enter the dataset at the geometric mean (the indicator value), all species are indexed at 100 in the first year and the very high values in 'c' are capped at 10000 at the end 'd' has been held at it's end value.

The 'indicator' column that is returned here is our indicator, calculated as the geometric mean of all the species in the data set.

Getting confidence intervals for the indicator

We can get confidence intervals for this indicator by bootstrapping across species. We have a function for that too!

```
# This function takes just the species columns
scaled_species <- rescaled_trends[,!colnames(rescaled_trends) %in% c('year', 'indicator')]
indicator_CIs <- bootstrap_indicator(Data = scaled_species)
```

```
## Running bootstrapping for 10000 iterations...done
```

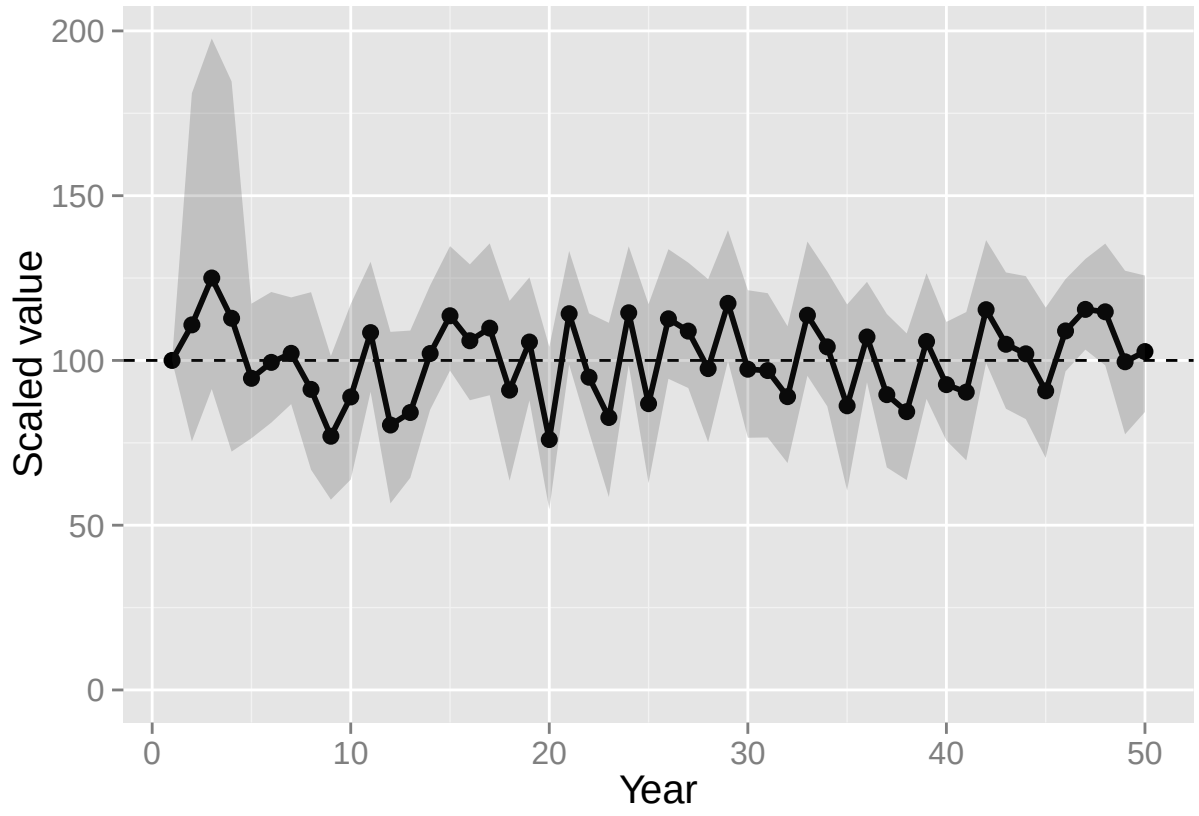
```
# Returned are the CIs for our indicator
head(indicator_CIs)
```

```
##      quant_025 quant_975
## [1,] 100.00000  100.0000
## [2,]  75.43295  181.0968
## [3,]  91.32019  197.7013
## [4,]  72.34008  184.6565
## [5,]  76.39099  117.2289
## [6,]  81.12554  120.7703
```

Plotting the indicator

We now have our indicator and the confidence intervals around it. The next step is to plot it. We have included a function that creates a simple plot using ggplot2, however you could easily create your own plots in R using the data.

```
# Plot our indicator.
plot_indicator(indicator = rescaled_trends[,'indicator'],
               CIs = indicator_CIs)
```



Creating a custom pipeline function

We have demonstrated how you might run the indicator functions one at a time now however in a 'pipeline we want data to flow through seamlessly. Addtionally there are a number of parameters in the functions that we ahev not shown you that you might find useful. Here is an example of how you can create your own pipeling function. Our function will wrap around the functions described above, setting the parameters to meet our needs. Once we have done this it will allow use to execute our pipeline in one line.

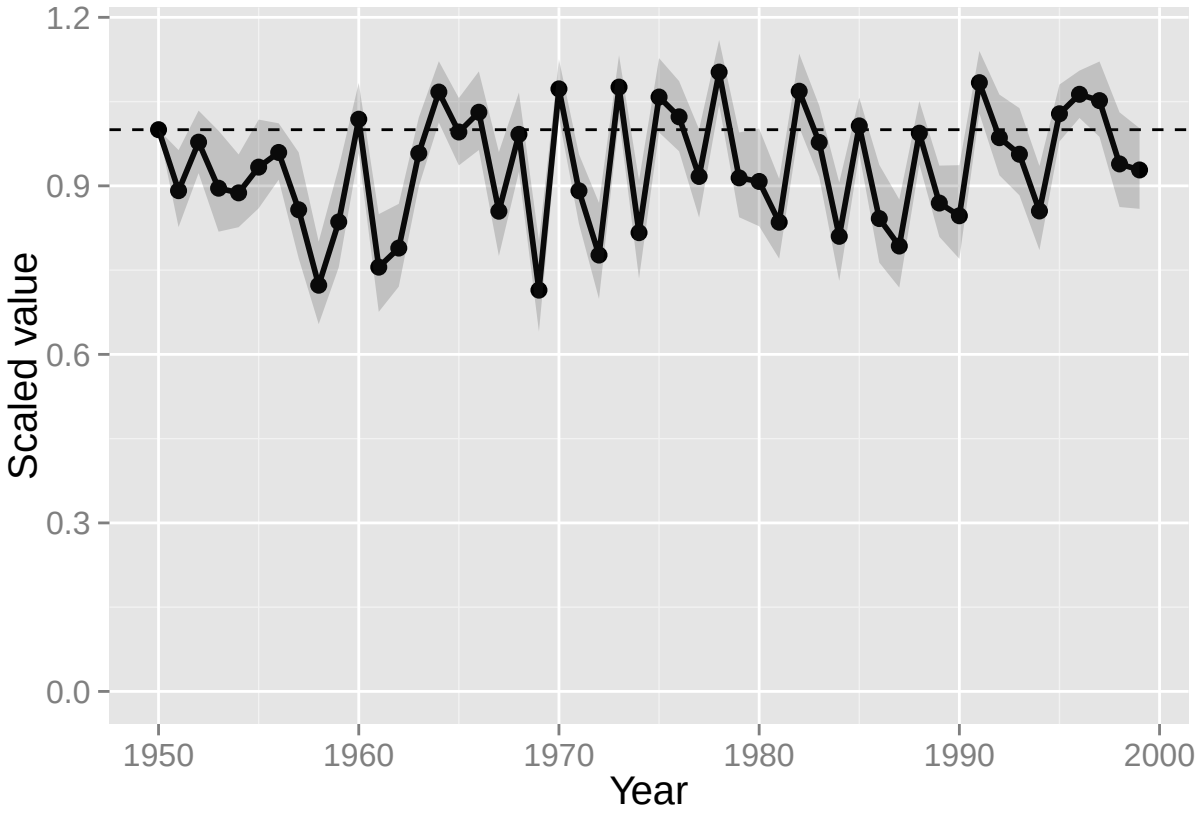
```
# I call my function 'run_pipeline' and the only arguement it
# takes is the directory of sparta's output
run_pipeline <- function(input_dir){

  ## Create the trends summary
  trends_summary <- summarise_occDet(input_dir = input_dir)

  ## Rescale the values and get the indicator values
  # Here I set the index to 1 and change the value limits
  rescaled_trends <- rescale_species(Data = trends_summary,
                                     index = 1,
                                     max = 100,
```

```
                                    min = 0.001)

  ## Bootstrap the indicator to get CIs
  scaled_species <- rescaled_trends[,!colnames(rescaled_trends) %in% c('year', 'indicator')]
  # This time I set the iterations to twice the default and
  # use custom confidence intervals
  indicator_CIs <- bootstrap_indicator(Data = scaled_species,
                                       CI_limits = c(0.25, 0.75),
                                       iterations = 20000)

  # Save a plot
  # This time I specify the years and index value
  jpeg(filename = file.path(input_dir, 'myPlot.jpg'),
       width = 600, height = 400)
    plot_indicator(indicator = rescaled_trends[,'indicator'],
                   year = rescaled_trends[,'year'],
                   index = 1,
                   CIs = indicator_CIs)
  dev.off()

  # Show the plot aswell
  plot_indicator(indicator = rescaled_trends[,'indicator'],
                 year = rescaled_trends[,'year'],
                 index = 1,
                 CIs = indicator_CIs)
}
```

Once we have created this function we can run this pipeline on a directory in one line, or put it in a loop to run across many directories.

```
# Now we can run the pipeline in one line, like a boss
run_pipeline(input_dir = '~/Testing_indicator_pipe')
```

```
## Warning in summarise_occDet(input_dir = input_dir): Not all files in ~/
## Testing_indicator_pipe are .rdata files, other file types have been ignored

## Loading data...done
## Running bootstrapping for 20000 iterations...done
```


1.2
0.9
0.6
0.3
0.0
Scaled value
1950
1960
1970
1980
1990
2000
Year