cesr - Calculating trends from CES data

Rob Robinson

02 January 2020

Background

The use of marked individuals to study animal populations has a long history in ecology and bird ringing has played a central role in this history. Although many early bird ringing studies were largely concerned with studying site fidelity and migration, attention soon shifted to estimating population size and changes in this over time (Baillie 1990, 2001; DeSante *et al.* 1995). Three vital rates drive population change: birth rate (productivity), survival (or mortality) rate and, when the study population is a subset of a larger one, dispersal (immigration/emigration) rates. Monitoring of these vital rates (demographic monitoring) shifts focus from the population pattern to the underlying process and should provide greater accuracy and sensitivity in detecting impacts of environmental change. Demographic monitoring of many bird species can be effectively accomplished with bird ringing. Indeed, application of capture-recapture models to bird ringing data is the only reliable method of obtaining estimates of survival and dispersal in wild bird populations. By using standard field methods and capture effort of bird-ringing studies among sites and between years, calculating vital rates, and changes in them, becomes much easier as variation due to differences in catchability of individuals is much reduced. Such standardisation provides the basis of national and international constant-effort (CE) ringing programmes (Robinson *et al.* 2009).

In Britain, CE ringing as a means of monitoring bird populations began in the late 1960's when ringers, mostly unpaid volunteers, began to consider how to increase the value of their ringing; standardising mist-netting effort was one obvious way in which to do so. In the late 1970's a national CE scheme was proposed, and the CE Sites (CES) scheme was formally adopted as part of the BTO's Integrated Population Monitoring Programme in 1986 (Peach *et al.* 1996). Subsequently, CE ringing schemes started in at least 15 European countries (co-ordinated through EURING) and in North America. At around the same time as CE ringing was being developed in Britain, a scheme was initiated in Germany/Austria at three sites (Mettnau, Reit and Illmitz, MRI) using year-round CE ringing and provided some of the first evidence for declines in migratory bird populations (Berthold *et al.* 1986).

CE programmes can operate at large spatial scales because they rely heavily on volunteer input and so can gather detailed demographic data in a cost-effective way. Such programmes aim to fulfill three complementary and inter-linked goals:

1. Monitoring - to provide long-term estimates or indices of abundance, productivity and survival in a range of common species, usually in concert with other monitoring schemes.
2. Research - to investigate the contribution of different demographic rates in determining population dynamics and their relationship with various ecological and environmental drivers.
3. Management - to understand how habitat may best be managed in conserving local populations.

CE capture-recapture data fulfill these goals by providing information on productivity; recruitment, i.e., number of new adult birds entering the breeding population, and adult survival. Because recruitment combines aspects of both productivity and over-winter survival of first-year birds, by examining patterns of recruitment, survival, and productivity measured at the same set of sites, we gain unique insights into the relative importance of drivers acting on each of the different life-cycle stages in determining population change (Julliard 2004; Saracco *et al.* 2008). This can be essential for designing conservation and management plans that can reverse population declines and maintain healthy populations as it identifies the key stages affecting population change and narrows the range of environmental factors to be considered; data from CE sites have been, and will continue to be, critical in this regard.

What is cesr?

cesr provides tools to perform basic analyses of annual variation in abundance, survival and productivity which are suitable for simple monitoring purposes, it cannot provide a full suite of methods to do a complete analysis of all CES data. It is a collection of tools designed to be used with the statistics package R. R has a number of advantages for statistical analysis: it is easy to introduce new analyses to it; it has good options for summarising and plotting data; and best of all, it is free.

Before you start ...

To use cesr you first need to download and install R (if it is not already installed on your computer). To do this you need to go to R homepage and click on the 'download R' link for your system. There are many guides to using R, some are easy to understand, others are highly technical and less easy to understand. Good places to start is the introductory guide available on the R website, which gives a few basic commands, and the quick-R website. For those who want to undertake some statistical analyses, Mick Crawley has written a good introduction[1] , while his *R Book*[2] is quite comprehensive. For those who need more technical details on doing high-level analysis, these books[3] offer good explanations. If you intend to use R a lot, do have a look at the RStudio software which provides a nice environment for writing and developing R code.

The basic R program allows you to do many things, but not everything. The beauty of the R system is that people can contribute 'packages' (cesr is one) that add specific extra functionality to the base system. The cesr package is now available for download and you can install it with the following R command:

```
devtools::install_github("btorobrob/cesr")
```

This slightly cryptic command simply goes to my account (btorobrob) on Github and gets the cesr package. Don't worry, you only need to do this done once, unless you change computer, update R, or you want to update the cesr package itself. This should also download all the extra packages cesr needs (notably data.table to speed up the data extraction and RMark for the survival analysis) so you may see some text whizzing by on the screen.

Each time you start R, you will need to load, or 'attach', the cesr package (there are many hundreds of packages to do all sorts of different analyses, so for efficiency R only loads those you actually want to use). Do this using

```
library(cesr)
#> Loading required package: data.table
#> Welcome to cesr 0.50, use help(cesr) to get started
```

You should get the brief welcome message telling you which version of the package you are using.

One important thing to remember. You can do very little harm by getting things wrong, if you get in a mess just close R down and start again. R will usually tell you what it thinks has gone wrong, but it can be quite cryptic, especially if you are unfamiliar with how it works. I've tried to translate the commonest ones, if you come across others copy and paste the error message from the R window *and* the command you used which generated it into an email to meand I'll see if I can help.

A sample session

There are essentially four steps necessary to analyse CES data (or indeed any other data) in R: first, one needs to get all the data into a form that is ready for analysis; one then needs to select and extract the data

[1] Crawley, M. (2005) Statistics: an introduction using R. Wiley-Blackwell, London.

[2] Crawley, M. (2007). The R book. Wiley-Blackwell, London.

[3] Try Bolker (2008) or Zuur *et al.* (2007), both of which use ecology examples, or Venables & Ripley (2002), which is more general; Wood (2006) is more advanced but explains the backround statistics well

for the species of interest; then the analysis can be performed; finally, you will wish to output the results of the analysis, perhaps for publication in an annual report or on a webpage. Additionally, it can also be useful to plot and summarise different sets of data, both to reveal patterns and to check that mistakes have not been made.

The cesr package is arranged around these four tasks and provides functions to perform each of them. A 'function' in R is simply a command to tell R to perform certain tasks. In order to complete these tasks they normally need to be given some information saying, for example, where it may find the data to use - these are included in parentheses '( )' and each of these is called an 'argument'. If a function requires more than one argument, those given after the first are usually named (e.g. species=10990), which means they can be given in any order. So the very simplest cesr analysis might look like:

```
library(cesr)
setwd('d:/CESdata/')
cesdata = readces('filename')
ces.sites = extract.coverage(cesdata)
robin.data = extract.data(cesdata, species=10990, plots=ces.sites)
result = index(robin.data)
plot.trend(result, type='adult', file='robin_ad.png')
```

The first line simply loads the cesr package into R. The second line (shorthand for 'set working directory') points R to the folder where your CES data are stored and where you can write the graph images to. The next line reads a datafile of CES captures and assigns it to an object called 'cesdata' . An object in R can be thought of as a mini-spreadsheet; normally it will contain some rows of data (an individual capture of a bird in this case) with some columns, each of which contains a particular variable (for example the date of capture, ring number, species type, sex, etc). The next two lines extract the specific data necessary for analyses, the years in which each site was covered (put into 'ces.sites'), and the data on captures for a particular species (to be found in 'robin.data'). Then we are ready to do the analysis with the function index(). Note that we need to tell index() where to find the data containing the species information (the object robin.data) and the site coverage information (the object ces.sites). This stores indices of the annual abundance and productivity in the object 'result' and provides some summary output about the model fit to the screen. The final line creates a graph of the annual trend in a form that can be used in publications.

Of course the package can do quite a bit more than this, and the rest of this document will show the range of tools available, and how to use them. If you need any more information or are just plain confused about what things do at any point, try help(function_name). Incidentally, in books about R, you will sometimes see an '=' used to assign an object and sometimes a '<-' (a left-bracket, followed by a hypen), for almost all purposes they are interchangeable, so I will continue to use the '='. Note also that R is **case sensitive**, so, for example, the variables 'netlength' and 'NetLength' are not the same; in most cases you should use only lower case for typing, since R may not recognise your commands otherwise.

The nice thing about R is that you can save these commands as a simple text file (usually with a '.R' extension) which means you can easily re-run them, or adapt them if you want to do a slightly different analysis (say for a different species, just copy the code and change the Euring number and species names). Getting into the habit of saving code files is a good one, especially if you might later want to re-run the analysis, or simply remember what you did! ON a similar note, if you start a line with a hash sign ('#') then R will ignore the rest of the line; this can be useful to add comments to your code explaining to future-you why you did things in a particular way. A good general rule is to use more comments than you think necessary - future-you will thank you as you try and figure out what you did and why in several months (or years!) time.

Data Preparation

The package comes with some sample data from Britain for Robin *Erithacus rubecula* and Song Thrush *Turdus philomelos* which we will use to demonstrate the use of this package. Often the hardest part in any analysis is often getting the data in a form that is suitable for analysis, consequently, cesr expects to get the data in a particular format. In an effort to reduce the options for things going wrong, most of the functions in the package require an object belonging the ces 'class'. These are generated automatically by the functions, so in most cases you don't need to worry about it, it just ensures all the necessary information is available and in the correct format. Because of this, to avoid problems later on, it is strongly recommended that you format your data correctly guidelines here and read it in using the readces() function, which sets up the correct variables. If you use another method (e.g. read.csv()), even if the data are in the right format, R should tell you don't have 'a CES object'; expect all kinds of pain if you persist!

Your file should be a comma-separated plain text file with all the data for every species caught by one scheme in one file. Schemes should screen the data for appropriate sites and years to use and all ringing additional to the standard CES effort (either extra nets or visits) should be excluded, i.e. the data should be ready to use as is. Each row should represent the capture of one bird on a particular day, thus an individual bird may appear on multiple rows, both on different dates within the same year and if it is recaptured in subsequent years. The first row should be column identifiers, and all birds should be included even if they include missing data (for example, birds that are unaged, that is Euring age-code 0 or 2).

Reading in the data

Before you start, remember to load the cesr package

```
library(cesr)
```

It is likely to be simpler if you keep a particular folder (or directory) for your data file and results. The first thing to do is tell R which folder this is using the command to set the working directory:

```
setwd('d:/data/ces')
```

where 'd:/data/ces' represents the path to your folder. (Note: If you are using Windows, R uses a forward slash '/' to separate directory levels rather than the usual backslash ".) With a datafile prepared in the correct format and copied into this folder it is easy to get the data into R, we will put it into an object called ukdata, but you can choose whatever name you like:

```
ukdata = readces('ukdata.csv')
```

If you have already set the working directory (above) and the file is in there, you only need to give the filename; if not, you will need to give the full path name (for example, 'd:/data/ces/ukdata.csv'). If this works correctly, R will read the data in silently and just give you the next prompt. It will do some basic checks of the data (e.g. for invalid age codes) and do it's best to warn you of any problems it finds.

If you want to try things on the example data set, just type

```
data(ukdata)
```

since these data have already been read in and saved in 'R format'.

Note, if you do not tell it otherwise, R will use scientific names for species. To make things easier it is possible to choose which language R will report species names, for example:

```
setceslang('English')
```

The allowable choices are 'Danish', 'Dutch', 'English', 'Finnish', 'French', 'German', 'Italian', 'Norwegian', 'Polish', 'Portuguese', 'Spanish' and 'Swedish'. Note the capitals and the use of quotes. In general it does not matter whether you use single '.' or double "." quotes, but avoid backquotes (`.`). For rarer species, names may not be available, in this case the EURING number will be given; it may be wise to check these EURING numbers to ensure that they have been entered into the data file correctly.

Checking the data

First, to check the data have loaded correctly we can ask for a summary of the data:

```r
summary(ukdata)
#> 4879 captures from 3737 individual birds of 2 species at 19 sites
#>
#>  Euring Code     Species Count
#>        10990       Robin  3006
#>        12000 Song Thrush   731
```

Thus, in our example dataset, we have capture data on 3,737 individuals of two species. You can use summary() on any CES object to get an idea of what it contains.

Although it may not be obvious from this, R will normally list species in taxonomic ("Voous") order; that is in order of the EURING code number. You can change this by using the sp.order argument:

```r
summary(ukdata, sp.order='count')
```

this will list species in descending order of the number of captures, you can list species alphabetically using sp.order='alpha', though both of these will give the same result in our case! You can also check how many young birds were caught:

```r
summary(ukdata, age=3, sp.order='count')
#> 3186 captures from 2737 juvenile birds of 2 species at 19 sites
#>
#>  Euring Code     Species Count
#>        10990       Robin  2365
#>        12000 Song Thrush   372
```

and similarly, how many adult birds by using age=4. Note, in general, the first argument is always the CES data object. It is best to include the name of the remaining arguments, in which case it does not matter in which order they are given.

If we wish to look a bit more closely at the data we have read in, then two commands may be helpful. The first:

```r
head(ukdata, n=4)
#>   countryID habitat visit day month year netlength scheme    ring species sex
#> 1       GBT      RD     6  23     6 2001       110    GBT K393817   10990   M
#> 2       GBT      RD     9  24     7 2001       110    GBT K393817   10990   M
#> 3       GBT      RD     2   6     5 2001       110    GBT K394567   10990   F
#> 4       GBT      RD     2   6     5 2001       110    GBT K394809   10990   M
#>   age sitename site  race julian      lat      long
#> 1   4      401    1 10990    173 51.51444 -3.743889
#> 2   4      401    1 10990    204 51.51444 -3.743889
#> 3   4      401    1 10990    125 51.51444 -3.743889
#> 4   4      401    1 10990    125 51.51444 -3.743889
```

shows the first n lines of data, while the second:

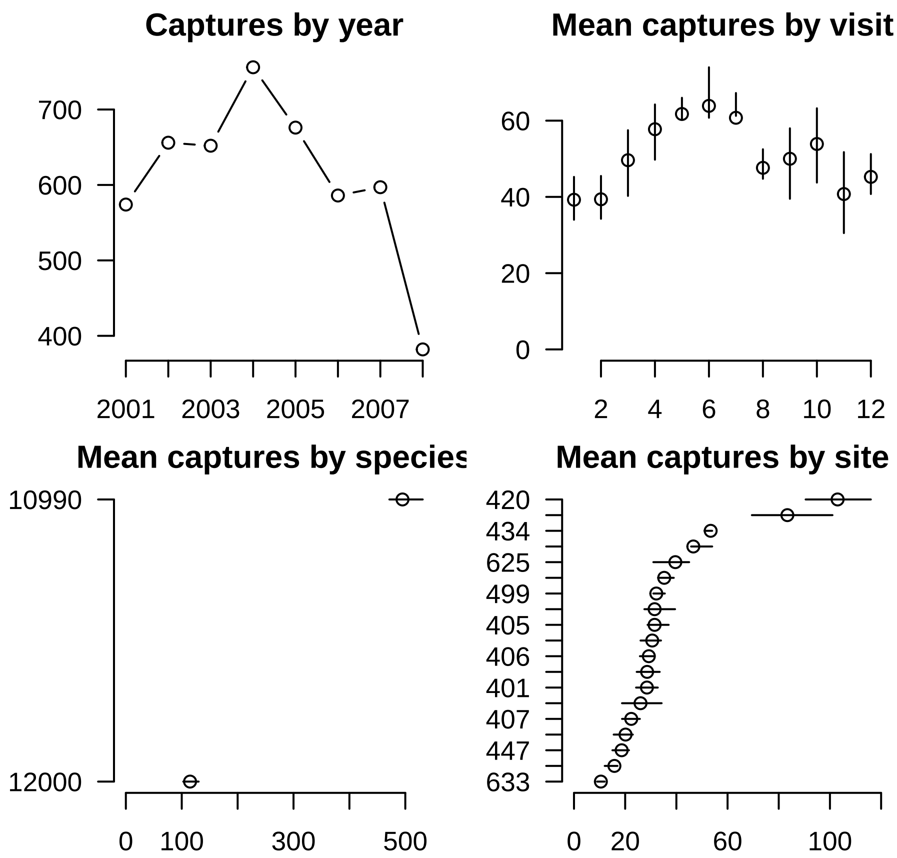
```r
summary(ukdata, df=TRUE)
#>   countryID          habitat       visit            day            month
#>  Length:4879        DS:2538   Min.   : 1.00   Min.   : 1.00   Min.   :4.00
#>  Class :character   RD: 722   1st Qu.: 4.00   1st Qu.: 8.00   1st Qu.:6.00
#>  Mode  :character   WD: 645   Median : 6.00   Median :16.00   Median :7.00
#>                     WS: 974   Mean   : 6.52   Mean   :15.94   Mean   :6.55
#>                               3rd Qu.: 9.00   3rd Qu.:24.00   3rd Qu.:8.00
#>                               Max.   :12.00   Max.   :31.00   Max.   :9.00
#>
#>       year        netlength     scheme          ring              species
```

```
#>  Min.   :2001   Min.   : 67.0   GBT:4879   Length:4879        10990:3960
#>  1st Qu.:2002   1st Qu.:102.0              Class :character   12000: 919
#>  Median :2004   Median :113.0              Mode  :character
#>  Mean   :2004   Mean   :144.8
#>  3rd Qu.:2006   3rd Qu.:158.0
#>  Max.   :2008   Max.   :279.0
#>
#>    sex            age            sitename         site            race
#>  F   : 589   Min.   :3.000   420    : 721   Min.   : 1.000   10990:3960
#>  M   : 886   1st Qu.:3.000   403    : 667   1st Qu.: 4.000   12000: 919
#>  NA's:3404   Median :3.000   417    : 373   Median : 9.000
#>              Mean   :3.347   408    : 282   Mean   : 8.568
#>              3rd Qu.:4.000   434    : 267   3rd Qu.:12.000
#>              Max.   :4.000   405    : 252   Max.   :19.000
#>                              (Other):2317
#>      julian           lat             long
#>  Min.   :103.0   Min.   :50.91   Min.   :-6.3950
#>  1st Qu.:155.0   1st Qu.:51.54   1st Qu.:-2.9161
#>  Median :183.0   Median :52.80   Median :-1.7536
#>  Mean   :182.7   Mean   :53.27   Mean   :-1.9051
#>  3rd Qu.:212.0   3rd Qu.:54.37   3rd Qu.:-0.4603
#>  Max.   :246.0   Max.   :57.07   Max.   : 1.8833
#>
```

gives a summary of each variable. Note there are two ways in which variables are summarised: for numeric variables it gives, the minimum, quartile, median and maximum values; for character variables it gives a frequency count of the commonest entries.

As with summary(), all CES objects can be 'plot'ted, which will give a graphical summary of the information therein, So:

```
plot(ukdata)
```



The top left graph shows the total number of records by species for each year. The top right graph, the mean number of captures (across the years) in each visit period, the bars give the interquartile range (i.e. encompassing 25% - 75% of the annual values). The bottom graph gives the mean number of captures by species (only 2 in our data set!) and site (note not all the site labels are printed here).

Normally, we will want to analyse the whole dataset to produce a national trend, but sometimes we may want to look at only a single site, or a group of sites representing a particular region or habitat. There is a function select.data() to do this. To use this function we first need to create a list of sites to be selected. The best way to do this is by using the in-built R function c(), which simply creates (or collects) a list of numbers. In our sample dataset we may want to include only those sites that were operated in each of the ten years, so we might type:

```
sitelist = c(401, 403, 405, 407, 408, 414, 417, 447, 463)
```

we can then select just those sites using:

```
ukcomplete = select.data(ukdata, sites=sitelist)
```

We have created a copy of the data (ukcomplete) so we can go back to the original dataset (ukdata) at any time. It is important to note that if your site codes include a combination of letters and numbers, you will have to enclose each site in the list in quotes, like this:

```
sitelist = c('C01', 'C03', 'C05')
```

Because habitat is a particularly important variable, we can select for this directly:

```r
ukdata_scrub = select.data(ukdata, habitat='DS')
```

if we wanted only dry/thorn scrub sites; other habitats can be selected in the same way using the two-letter codes listed in Table 1. Selection of birds by age will happen a bit later on, for the moment we just select the sites of interest. Of course, more experienced R users can subset the data directly if necessary, which provides more flexibility than this simple method.

Extracting the data

Having got just the data we want in the right format, it is now time to extract the information we need to do our analysis. First, we have to identify which sites were operated in which years , then we need to extract the data for the species we are interested in. We only need to extract the site data once, then we can look at as many species as we like. To extract the site coverage from all the UK data, which we will put in an object called uk_sites, type:

```r
uk_sites = extract.coverage(ukdata, early=c(6,4), late=c(6,4), all.visits=12)
#> 19 sites contributed 1321 visits, with 275 visits missing
```

we should specify (using all.visits) how many visits the protocol requires (in the UK case 12, one every ten days or so through the season). Most adults will tend to be caught earlier in the season (as they disperse after breeding), while most juveniles will tend to be caught later in the season. Thus an additional constraint can be imposed in that a certain number (y) of the first x visits, and similarly of later visits, should also be made. This can be specified using early=c(x,y), and similarly late=c(x,y). This has the added advantage that if a site is only covered in the early period of the year, say, it can still be used for calculating adult abundance. So, in the UK example, four of first six visits must be covered early=c(6,4), and four of the last six visits late=c(6,4). Note that the early and late period may overlap (particularly if your season has an odd number of visits), it doesn't affect the totals, only whether a site is included in the analysis or not. A simpler alternative is to just require a certain number of visits irrespective of timing (especially in those countries where the season has fewer visits in total), just replace the early= and late= arguments with min.visits=n, where n is the minimum of visits required, for example:

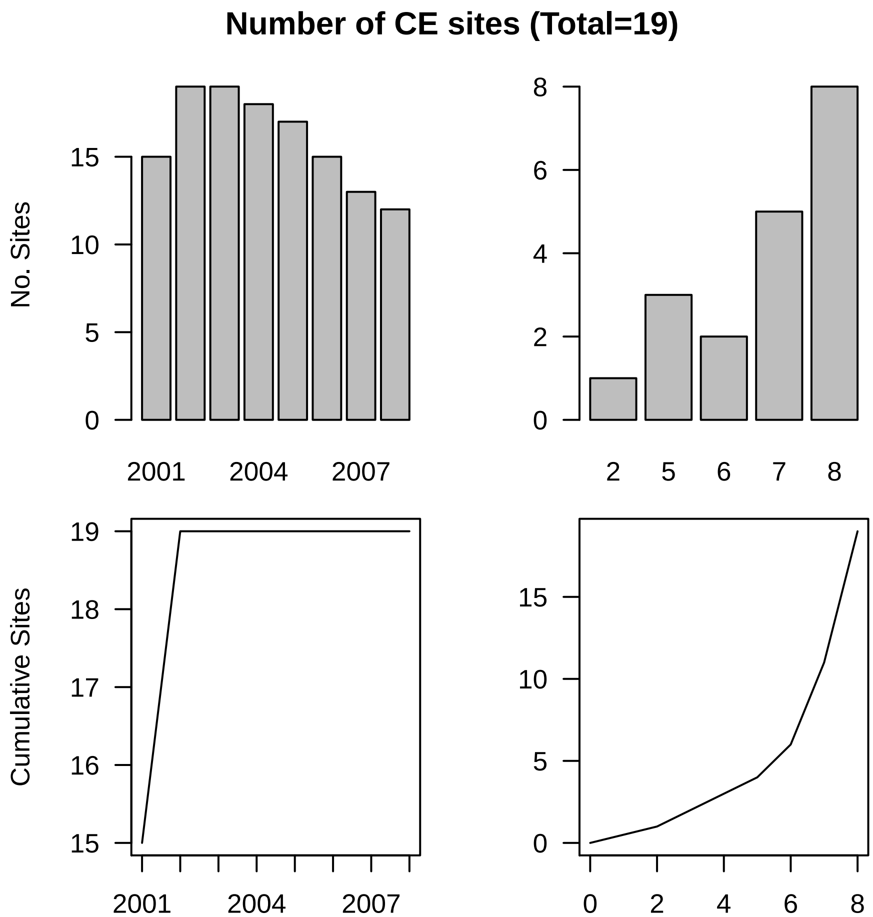
```r
sites = extract.coverage(ukdata, min.visits=8, all.visits=12)
```

As before we can check this has worked by looking at a summary of the data and seeing that all is as expected:

```r
summary(uk_sites)
#> Total number of sites operated: 19
#>
#> Number of sites in each year:
#>
#> 2001 2002 2003 2004 2005 2006 2007 2008
#>   15   19   19   18   17   15   13   12
#>
#> Number of years sites run for:
#>
#> 2 5 6 7 8
#> 1 3 2 5 8
```

Thus we have 19 sites operated in the eight years for which we have data, of which 8 have operated for all eight years and one has only operated for two years. We can see similar summary information by using the plot() command.

```r
plot(uk_sites)
```

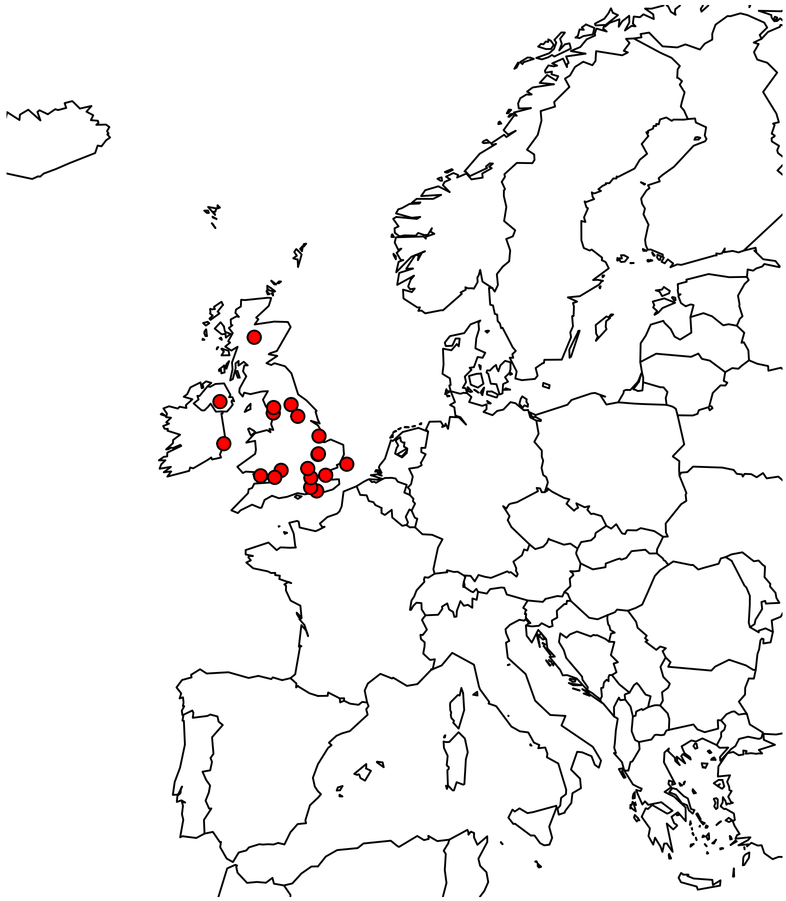


The figure should have four panels, the top two illustrate the frequency of sites in each year, and the number of years that sites have run. The bottom two illustrate the evolution of sites contributing to teh scheme , that is the cumulative number of sites that have started operating by each year[4] and the cumulative number of sites operated for a certain number of years. In our example, six sites have operated for six years or fewer and eight sites for all eight years.

Finally, we might want to plot a map of where our sites are, we can do this using the map.ces() command. To plot a map simply use:

```
map.ces(ukdata)
```

[4]In our abbreviated example, 16 sites started in 2001, and the remaining three in 2002, the second year, so the graph is not terribly informative!



If you give a filename, R will print the map to a file for later use[5], if not, it will simply print it to the screen. If you are using these maps in publications then you may wish to specify the size (in pixels) by also using the height= and width= arguments[6]; the size otherwise will be 640 by 480 pixels. Unless you say otherwise, the sites will be plotted onto a map of Europe, to zoom into a smaller area use the xlim= and ylim= arguments to specify longitude and latitude bounds respectively[7]. For example, this will draw a map of Britain and Ireland:

```
map.ces(uk_sites, xlim=c(-10,2), ylim=c(50,60), file='sitemap.png')
```

Note the use of c() to concatenate the minimum and maximum bounds into one R object. In this case, sites are identified as current (red) or no longer operating (white), you can also colour sites by the number of years they have operated (use type='n') or habitat (type='h'), in which case you can specify up to 4 colours using the col=c(. . .) argument. For more details, use help(map.ces) or its shorthand ?map.ces; these, of course, work for almost all R functions.

Having extracted the site coverage data, we can now extract the species data using extract.data(). Normally, we will want to extract data for both adults and juveniles together, but if for some reason just one of these is needed, then you can use the age= argument (where age may be either 3 for juveniles of 4 for adults). To extract the Robin data from our ukdata object we would type:

```
robin.dat = extract.data(ukdata, species=10990, plots=uk_sites)
```

Note that we use the Euring code number to tell R which species we want, and the plots argument inform it about which sites are covered and which we extracted earlier. The Robin data are now in the robin.dat

[5] the format will be determined by the file extension; '.jpg', '.tif' or '.pdf' will also work

[6] More detailed maps are possible using the maprec package, or other R map plotting packages

[7] if R tries to 'prettify' these too much try using lforce='e', which asks for those 'e'xact limits

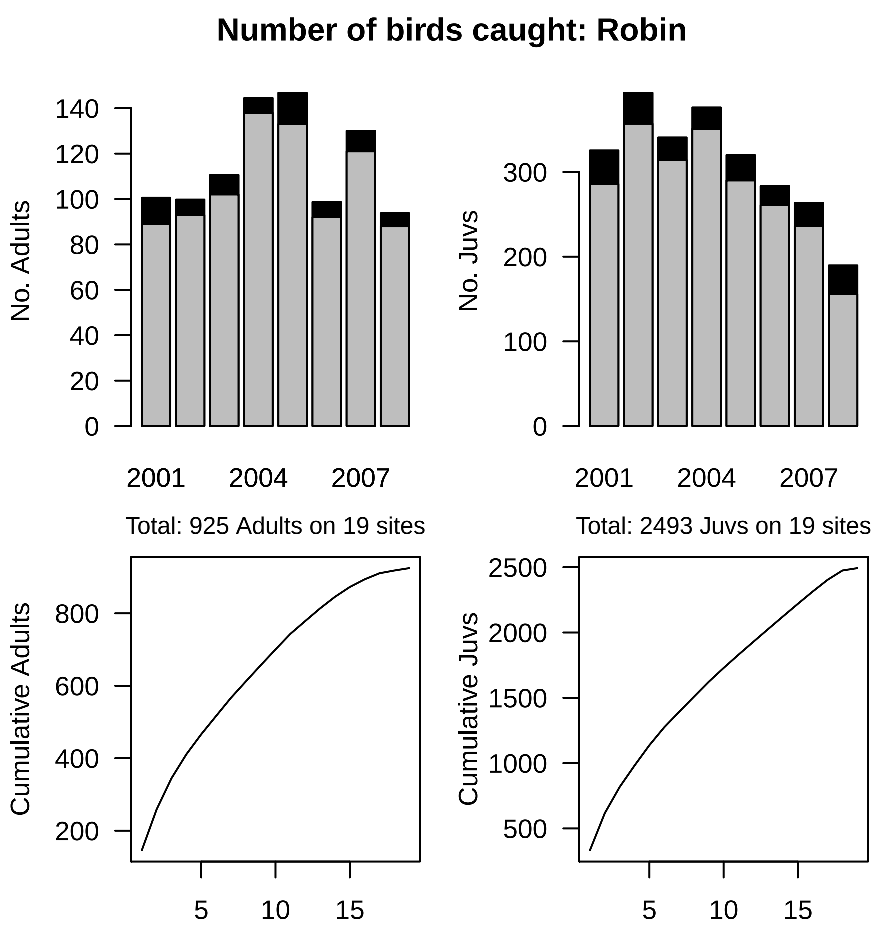
object. As before we can see a summary of these data using:

```
summary(robin.dat)
#> Total number of Robin caught: 856 adults and 2251 juveniles
#>
#> Number of adults caught each year:
#> 2001 2002 2003 2004 2005 2006 2007 2008
#>   89   93  102  138  133   92  121   88
#>
#> Number of juveniles caught each year:
#> 2001 2002 2003 2004 2005 2006 2007 2008
#>  286  357  314  351  290  261  236  156
```

As you can see, when we do this R gives some basic summary information about how many individuals have been caught. It is worth checking that these are what you expected.

We can also get some summary plots of the data

```
plot(robin.dat)
```



This will again give four graphs, the top two give the number of adult and juvenile birds caught in each year. The grey bars represent those actually caught, the black tops to the bars represent the additional number of birds that would have been caught on the missed visits. In the bottom two graphs, sites are ranked (from highest to lowest) by the number of birds caught and the total number of birds caught on sites of that rank or greater is plotted. This gives an indication of how birds are distributed across sites: if a few birds are caught at many sites, the graph will look more like a straight line (each site catches a similar number of birds so adding each site causes a roughly increase), if many birds are caught at a few sites (as might be the case for a habitat specialist like Reed Warbler *Acrocephalus scirpaceus*) the line will increase steeply before flattening out.

Indices of Abundance and Productivity

Now that we have all the data in place, we can calculate annual indices of abundance and productivity at the same time using:

```
robin.res = index(robin.dat)
#> Adult Abundance (Robin)
#> Model Fit: Dispersion parameter is 0.80 (resid. deviance  72.5 on    88 d.f.)
#> Change between 2001 and 2008: 28.8%
#>
#> Juvenile Abundance (Robin)
#> Model Fit: Dispersion parameter is 1.35 (resid. deviance 126.0 on    95 d.f.)
#> Change between 2001 and 2008: -19.2%
#>
#> Productivity (Robin)
#> Model Fit: Dispersion parameter is 0.19 (resid. deviance  15.0 on    81 d.f.)
#> Change between 2001 and 2008: -5.7%
```

Assuming data for both adults and juveniles were extracted with extract.data() then index will analyse both adult and juvenile abundances and productivity (the ratio of juvenile to adult birds caught, see below) and produce a simple summary of the results for each. The dispersion parameter (calculated by dividing the deviance by the degrees of freedom) is a quick way of assessing how well the model we have fitted fits our data, if the model fit is good then it should approximately equal 1, though values of up to 2 or 3 are acceptable; for more details, see the description about the models fitted below[8]. You will also see printed the change between the first and last years for which there are data. Note, unless you say otherwise, the function will attempt to correct for missing visits using the standard BTO method (see Peach *et al.* 1996; Robinson *et al.* 2007), if you do not want this you can use the argument visit.corr:

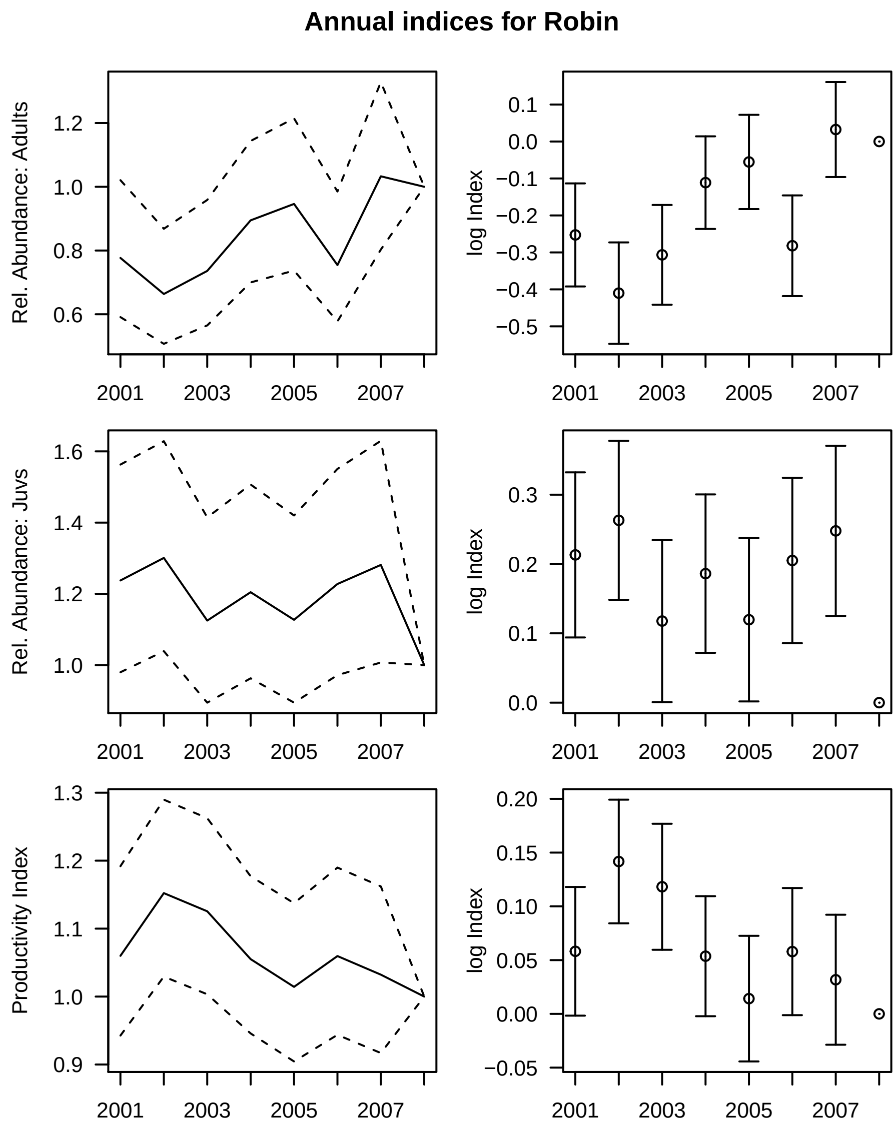
```
robin.res = index(robin.dat, visit.corr=FALSE)
```

From analyses of BTO data, it seems that the typical level of missing visits makes relatively little difference to the final indices in any case (Miles *et al.* 2007; Cave *et al.* 2009).

For a quick look at the annual indices from these models, simply type:

```
plot(robin.res)
```

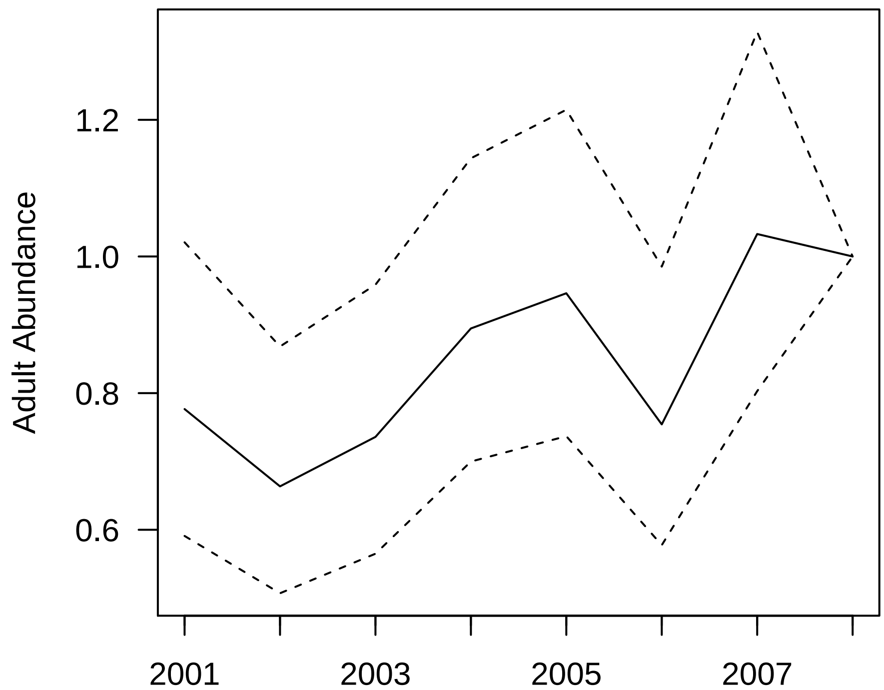
[8] not yet written!



which should yield two columns of, probably, three plots (assuming you extracted data for both adult and juvenile birds, otherwise fewer will be presented). In this view the graphs down the left-hand side (and probably most useful) are plots of the annual indices, those down the right-hand side, the actual (transformed) parameter indices from the fitted model, in almost all cases though the pattern through the years will look similar.

In most cases, however, we will be interested in producing a graph of a particular trend for inclusion in a paper or presentation. To do this, use the plot.trend() command and specify which trend you wish to plot and, optionally, a filename, thus:

```
plot.trend(robin.res, type='adult')
```



will print a simple graph showing the annual variation in adult abundance index to the file “adults.png”. To get a similar graph for the number of juveniles use graph=‘juvenile’ and for the productivity index, use graph=‘productivity’; you can shorten these to just the first letter (‘a’, ‘j’, or ‘p’) if you wish. Use “file=” to send the graph to a file with an appropriate extension[9].

The graph produced is quite a simple one, but you can tweak various aspects by using standard R arguments. For example, to change the colour of the lines, use add col=‘. . . ’ (where . . . is the name of a colour, e.g. col=‘red’); to change the thickness use lwd=x. Change the y-axis title (perhaps to something in a non-English language) by using ylab=‘. . . ’).

For those who are used to drawing graphs in R, plot.trend() returns a dataframe which can be captured into an object:

```
robin.trend = plot.trend(robin.res, type='adult')
```

This has six columns, the models parameter estimates (and their standard errors) and the annual index values (with lower and upper confidence limits). So, for example

```
head(robin.trend)
#>   years       parm        se     index       lcl       ucl
#> 1  2001 -0.2527254 0.1394144 0.7766812 0.5909803 1.0207339
#> 2  2002 -0.4101548 0.1371861 0.6635475 0.5071063 0.8682505
#> 3  2003 -0.3066323 0.1349024 0.7359212 0.5649398 0.9586508
#> 4  2004 -0.1112956 0.1253052 0.8946743 0.6998501 1.1437337
```

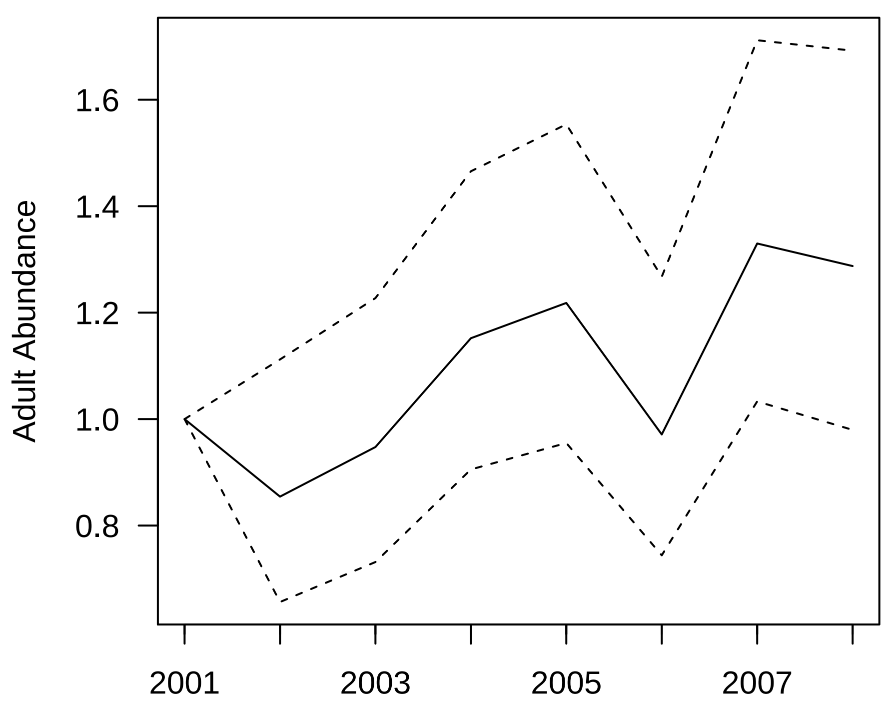
[9]‘.jpg’, ‘.png’, ‘.pdf’ should all work

```
#> 5  2005 -0.0553331 0.1274868 0.9461699 0.7369740 1.2147478
#> 6  2006 -0.2820056 0.1362389 0.7542695 0.5775103 0.9851294
```

These can then be plotted using the any of the R plotting tools, or "copy and pasted" into a spreadsheet like Microsoft's Excel or OpenOffice's Calc and a graph drawn there (hint: you may find the 'Text to Columns' tool useful).

You will note in the graph that the final year is set 1. This is because we can only estimate an index of abundance, not actual abundance, and all the annual indices are relative to a given year. You may prefer to have the trend starting at 1, rather than finishing at 1, this easy to do:

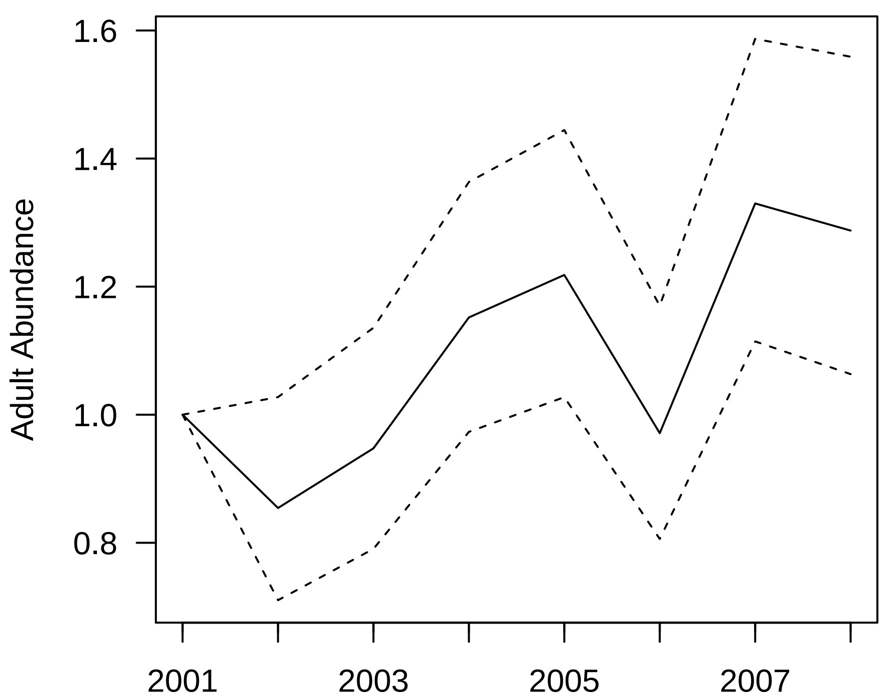
```
robin.res = index(robin.dat, year=2001)
#> Adult Abundance (Robin)
#> Model Fit: Dispersion parameter is 0.80 (resid. deviance  72.5 on    88 d.f.)
#> Change between 2008 and 2001: 28.8%
#>
#> Juvenile Abundance (Robin)
#> Model Fit: Dispersion parameter is 1.35 (resid. deviance 126.0 on    95 d.f.)
#> Change between 2008 and 2001: -19.2%
#>
#> Productivity (Robin)
#> Model Fit: Dispersion parameter is 0.19 (resid. deviance  15.0 on    81 d.f.)
#> Change between 2008 and 2001: -5.7%
plot.trend(robin.res, type='a')
```



The results are the same as before, only the graph is 'shifted-up' the y-axis. Note that you can abbreviate the graph type to just the first letter if you wish.

Clearly, Robin abundance varies between years, but are these differences significant? One quick visual check can be done by looking for overlapping confidence intervals. You can check whether any year is different from the reference year just by looking to see whether the confidence limits overlap 1, either graphically, or by inspecting robin.res (as it is called in this case). But what if you wish to look between years other than the reference year? Looking for overlap in the 95% confidence limits will result in too stringent a test, Payton *et al.* (2003) showed that actually the 83% confidence intervals will overlap with a probability of (approximately) 0.05. We can get these using the cl= argument

```
robin.res83 = index(robin.dat, year=2001, cl=0.83)
#> Adult Abundance (Robin)
#> Model Fit: Dispersion parameter is 0.80 (resid. deviance  72.5 on    88 d.f.)
#> Change between 2008 and 2001: 28.8%
#>
#> Juvenile Abundance (Robin)
#> Model Fit: Dispersion parameter is 1.35 (resid. deviance 126.0 on    95 d.f.)
#> Change between 2008 and 2001: -19.2%
#>
#> Productivity (Robin)
#> Model Fit: Dispersion parameter is 0.19 (resid. deviance  15.0 on    81 d.f.)
#> Change between 2008 and 2001: -5.7%
robin.trend = plot.trend(robin.res83, type='a')
```



and again looking at the graph, or by typing robin.trend to look at a table of the annual estimates. This suggests, for example, that numbers in 2004/05 on these sites were significantly higher than in 2001 (the lower confidence limit, lcl, for 2004/05 is greater than the upper confidence limit, ucl, for 2001), but not other years (note this is slightly sensitive to which year is set as the reference year). If you are doing multiple analyses, one can check which limits have been used by typing:

```
robin.res83$limits
#> [1] 0.83
```

though in this case it should be obvious from the name.

More commonly, you might want to check whether this year's results are higher (or lower) than over some previous time period (say the last five years) or whether there is a trend over time. You can use index() to do this too. To look at a (linear) time trend, use the trend= argument, for example:

```
robin.trend = index(robin.dat, trend=8, year=2005)
#> Adult Abundance (Robin)
#> Model Fit: Dispersion parameter is 0.84 (resid. deviance  82.5 on    94 d.f.)
#> Slope for past 8 years is 0.06 +/- 0.02 (t=3.650, P=0.002)
#>
#> Juvenile Abundance (Robin)
#> Model Fit: Dispersion parameter is 1.65 (resid. deviance 167.4 on   101 d.f.)
#> Slope for past 8 years is -0.01 +/- 0.01 (t=-0.976, P=0.341)
#>
#> Productivity (Robin)
#> Model Fit: Dispersion parameter is 0.19 (resid. deviance  16.2 on    87 d.f.)
#> Slope for past 8 years is -0.01 +/- 0.01 (t=-2.397, P=0.026)
```

In this case we have asked for the trend to be calculated over our entire time-series (8 years from 2001-08), you can estimate the trend over a shorter period (working back from the current year) by entering a smaller number. Note you can still set the reference year to be any year (here we've set it to 2005, roughly halfway through the period, try plotting the adult and juvenile trends in robin.trend and see what they look like). To compare this year's results to the previous n years, use compare=n instead of trend=n.

```
robin.compare = index(robin.dat, compare=3, year=2005, cl=0.83)
#> Adult Abundance (Robin)
#> Model Fit: Dispersion parameter is 0.83 (resid. deviance  77.9 on    90 d.f.)
#> Last year is 109.8% that in previous  3 years: Estimate=0.09 +/- 0.11 (t=0.836, P=0.412)
#>
#> Juvenile Abundance (Robin)
#> Model Fit: Dispersion parameter is 1.35 (resid. deviance 128.2 on    97 d.f.)
#> Last year is 83.1% that in previous  3 years: Estimate=-0.18 +/- 0.10 (t=-1.781, P=0.088)
#>
#> Productivity (Robin)
#> Model Fit: Dispersion parameter is 0.18 (resid. deviance  15.1 on    83 d.f.)
#> Last year is 96.6% that in previous  3 years: Estimate=-0.03 +/- 0.05 (t=-0.679, P=0.504)
```

index() also calculates a measure of productivity from the ratio of juveniles to adults that are caught each year. To see these results, use type='p' when using plot.trend(). Note this is probably not a good measure, even relatively, of the number of young produced on a CES site since young birds can disperse some distance even at a relatively young age. For example, wetland sites typically have a higher productivity than drier sites since birds may congregate there to forage. Thus this measure of productivity is better thought of as a measure the number of young fledging from the wider area surrounding the site; how far this will be will depend on the site. Note also that, following Peach *et al.* (1996) and Robinson *et al.* (2007), the index is expressed in terms of the number of juveniles per adult, rather than simply the proportion of juveniles[10]

[10] This is achieved by using a log rather than a logistic back-transform to create the annual indices, see Peach *et al.* (1996) for

Calculating Adult Survival

To estimate survival probabilities, we first need to extract capture histories for individual birds, we can do this for adult birds using:

```
robin.ch = extract.ch(ukdata, species=10990, plots=uk_sites)
#> Loading required package: reshape
#>
#> Attaching package: 'reshape'
#> The following object is masked from 'package:data.table':
#>
#>     melt
```

(The data are required in a different format, hence the need for a different extract command). Use summary() to see how many individuals we histories for

```
summary(robin.ch)
#> 3478 adults of Robin captured on 8 occasions, at 19 sites starting in 2001
#>
#> Number of birds first ringed in each year
#> 2001 2002 2003 2004 2005 2006 2007 2008
#>  435  494  453  562  478  406  400  250
#>
#> Number of times individuals have been recaptured
#>    0    1    2    3    4    5    6
#> 2807  573   72   18    6    1    1
#>
#> Number of individuals captured at each site
#>   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
#> 182 532 191 151 143 201 116 144 255 534 228  93  70 160 162 122 137  22  35
```

Survival probabilities for adults are estimated using function mark.ces(). This runs Program MARK (White & Burnham 1999) through the RMark package (Laake & Rexstad 2008) to fit a form of the standard CJS mark-recapture model modified to account for transients (after the manner of Pradel *et al.* 1997), see below for details[11]. Briefly, this allows us to account for individuals that are only caught once, either because they are migrating through the area, or because they are breeding a fair distance from the nets and so are much less likely to be caught than those with nests close to the net locations.

```
robin.surv = mark.ces(robin.ch)
```

You can safely ignore the warning message about the number of parameters being adjusted upwards[12] This will estimate a number of quantities, which again we can see by using summary()

```
summary(robin.surv)
#> Probability of residency: 0.461 +/- 0.033
#> Annual survival varies between 0.272 and 0.642 with an average of 0.462
#> Recapture probabilities vary between 0.051 and 0.609
```

The probability of residency is, loosely, the proportion of birds caught on the site that is likely to be resident and breeding there. Adult survival rates are estimated annually and assumed to be the same across all sites (this could be relaxed by using the group= option when extracting the data and re-running mark.ces). The recapture probabilities are site-specific, but constant through time (reflecting constant effort!). In this case the summary of recapture probabilities isn't very informative, looking at the full output robin.surv$recapture shows us that sites 656 and 447 have recapture probabilities of (close to) 0 and 1 respectively. This is likely

details.

[11]not yet written!

[12]If you see the error message: Error in setwd("markfiles"): cannot change working directory, you probably need to use setwd() to change the working folder to one you have permission to write to.

to mean that MARK does not have enough data to estimate recapture probabilities for these two sites, so we can exclude them, using the exclude= argument to mark.ces:

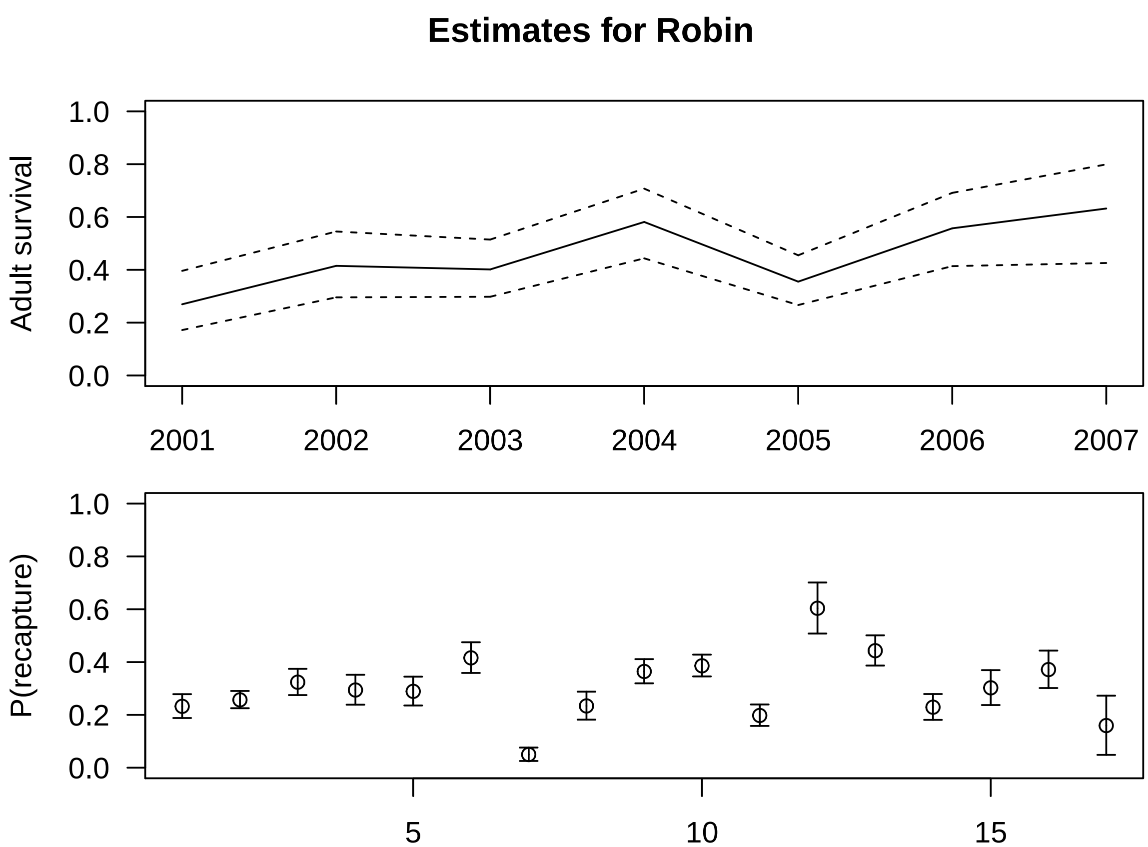
```
robin.surv = mark.ces(robin.ch, exclude=c(447,656))
```

which gives slightly better results, note that the other estimates also change slightly because we are excluding some data:

```
summary(robin.surv)
#> Probability of residency: 0.467 +/- 0.033
#> Annual survival varies between 0.270 and 0.632 with an average of 0.459
#> Recapture probabilities vary between 0.050 and 0.604
```

It will probably be more informative to look at a graph of the annual survival estimates and site-specific recaptures, by simply :

```
plot(robin.surv)
```



You can also use plot.trend as before, but with type='survival' (or 's' for short), to get the survival trend for presentation in a paper or report.

References

Baillie, SR (1990) Integrated population monitoring of breeding birds in Britain and Ireland. *Ibis* 132:151-161.

Baillie, SR (2001) The contribution of ringing to the conservation and management of bird populations: a review. *Ardea* 89:S167-184.

Berthold, P, Fliege, G, Querner, U & Winkler, H (1986) The development of songbird populations in central Europe: analysis of trapping data. *Journal fur Ornithologie* 127:397-437.

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

Cave, VM, Freeman, SN, Brooks, SP, King, R & Balmer, DE (2009) On adjusting for missed visits in the indexing of abundance from 'constant effort' ringing. *Environmental and Ecological Statistics* 3:949-963.

DeSante, DF, Burton, KM, Saracco, JF & Walker, BL (1995) Productivity indices and survival rate estimates from MAPS, a continent-wide programme of constant-effort mist netting in North America. *Journal Applied Statistics* 22:935-947.

Julliard, R (2004) Estimating the contribution of survival and recruitment to large scale population dynamics. *Animal Biodiversity & Conservation* 27:417-426.

Laake, J. & Rexstad, E. (2008) RMark - an alternative approach to building linear models in MARK. In Cooch, E. & White, G.C. (eds) Mark - a gentle introduction..

Miles, W, Freeman, SN, Harrison, NM & Balmer, DE (2007) Measuring passerine productivity using constant effort sites: the effect of missed visits. *Ringing & Migration* 23:231-237.

Payton, ME, Greenstone, MH & Schenker, N (2003) Overlapping confidence intervals or standard error intervals: What do they mean in terms of statistical significance? *Journal of Insect Science* 3:34.

Peach, WJ, Buckland, ST & Baillie, SR (1996) The use of constant effort mist-netting to measure between-year changes in abundance and productivity of common passerines. *Bird Study* 43:142-156.

Pradel, R., Hines, J. E., Lebreton, J. D., & Nichols, J. D. (1997) Capture-recapture survival models taking account of transients. *Biometrics* 53:60-72.

Robinson, RA, Freeman, SN, Balmer, DE & Grantham, MJ (2007) Cetti's warbler Cettia cetti: analysis of an expanding population. *Bird Study* 54:230-235.

Robinson, RA, Julliard, R & Saracco, JF (2009) Constant effort: studying avian population processes using standardized ringing. *Ringing & Migration* 24:199-204.

Saracco, JF, DeSante, DF & Kaschube, DR (2008) Assessing landbird monitoring programs and demographic causes of population change. *Journal of Wildlife Management* 72:1665-1673.

Venables, WN & Ripley, BD (2002) *Modern applied statistics with S, 4th ed.* Springer, Berlin.

White, G.C. & Burnham, K.P. (1999) Program MARK: survival estimation from populations of marked animals. *Bird Study* 46:S120-S139.

Wood, SN (2006) *Generalized additive models in R.* Chapman & Hall, London.

Zuur, AF, Ieno, EN & Smith, GM (2007) *Analyzing ecological data.* Springer, Berlin.