Stock-Recruitment Database Administrator's Guide[*]

Nicholas Barrowman

Version 1.11, 7/18/97

Contents

[*] This manual is located in `/data/bioram/sr/guide/guide.tex`. Several example programs are in the same directory. RAM has added some more updates, 2003.

1 NEW

to remove commas, and to make columns out of stream data use

```
perl -p -i.bak -e "s/( 19[0-9][0-9] )/\n \$1/g" 7e.dat
perl -p -i.bak -e "s/,//g" 7e.dat
```

also use sepcol

2 Introduction

The stock-recruitment database is a directory of ASCII files containing spawner and recruitment data for several hundred different fish stocks (Myers et al. 1995). For more information on the data format, etc., see

```
/data/bioram/sr/data/README
```

In addition to the database itself, numerous S-PLUS functions and some programs in other languages have been written to read, manipulate, analyze, and display the data. Important directories relating to the stock-recruitment database include:

Directory	Purpose
`/data/bioram/sr/data`	The database.
`/data/bioram/sr/functions`	S-PLUS functions for manipulating the stock-recruitment object.
`/data/bioram/sr/s`	S-PLUS functions for maximum likelihood estimation of stock-recruitment functions.
`/data/bioram/general/s`	Local general-purpose S-PLUS functions.
`/data/bioram/general/scripts`	Local command scripts.
`/var/ftp/pub/sr`	Anonymous ftp directory.

3 S-PLUS

3.1 The Stock-Recruitment Data Object

For ease of manipulation, S-PLUS functions have been written to read the ASCII files in the stock-recruitment database into a single S-PLUS object, which we will call the **(S-PLUS) stock-recruitment data object**. As of July, 1997, the stock-recruitment data object was about 1.6 MB in size.

To use S-PLUS to analyze the stock-recruitment data, the following steps should be followed:

1. Perform some preliminary steps (section 3.2).

2. Create the stock-recruitment data object (section 3.3).

3. (Optionally) Fit stock-recruitment models (section 3.6).

4. Use the S-PLUS functions to analyze the data, plot fits, etc.

3.2 Preliminaries

Before using the S-PLUS stock-recruitment functions, several preliminary steps are required. To use the standard configuration, start from the UNIX shell and type:

`mkdir stockrec`	*Create a directory called* `stockrec` *(or whatever you want to call it).*
`cd stockrec`	*Go into the directory.*
`mkdir .Data`	*Create a* `.Data` *subdirectory.*
`cp /data/bioram/sr/data/basic.First.s First.s`	*Copy a file containing the definition of a* `.First` *function. See Appendix D for more details.*
`Splus`	*Start S-PLUS.*
`source("First.s")`	*Read in the* `.First` *function.*
`q()`	*Quit S-PLUS.*
`Splus`	*Restart S-PLUS.*
`?sr`	*Check that everything worked by looking at some online help. Then press* `q` *to quit from the online help. Note that the S-PLUS command to print out a help page is* `help("sr",offline=T)`*.*

From here on, unless otherwise noted, commands shown in `typewriter font` are S-PLUS commands.

3.3 Creating the Stock-Recruitment Data Object

To create the stock-recruitment data object, use the following commands:

`newobject()`	*Create an empty stock-recruitment object.*
`addall()`	*Add all stocks to the object. This takes about a half hour.*

3.4 Creating a Special Stock-Recruitment Data Object

To create the special stock-recruitment data object which leaves out certain stocks, stocks without both spawner and recruit data, and stocks with short series use the following commands:

`newobject()` *Create an empty stock-recruitment object.*

`addsome()` *Add some stocks to the object. This takes about a half hour.*

`buildsr()` *This functionas calls* `addsome()`*, and excludes species based upon a list within the function.*

The function `buildsr()` was used in the paper: Myers, R.A. and N. J. Barrowman. 1996. Is fish recruitment related to spawner abundance? Fish. Bull. 94: 707-724.

3.5 Generating a Summary Table

Once the stock-recruitment data object has been created, a LaTeX summary table can be generated. But first, several objects must be created. Type:

`makedatalen()` *Create an object called* `datalen` *giving the number of years of paired spawner-recruitment data.*

`makecvsdf()` *Create an object called* `cvsdf` *giving the coefficient of variability (cv), skewness (s), etc., in a data frame (df).*

`makestab()` *Create an object called* `stab` *giving summary information.*

`makeftab()` *Create an object called* `ftab` *which is a "formatted" table.*

Then to produce the table, type:

`smtable()` *Generate a LaTeX file called* `smtable.tex` *in the current directory or in a* `latex` *subdirectory if one exists.*

`latex()` *Run LaTeX. If there are any problems, LaTeX may stop and display the* ? *prompt (see Appendix C). Usually, pressing RETURN will cause LaTeX to continue until the end of the file or the occurrence of another problem. If this does not work, type* `x` *and press RETURN.*

`xdvi()` *Preview the table. (Press* `q` *to quit.)*

`dvips()` *Print the table.*

3.6 Fitting Stock-Recruitment Models

Before fitting stock-recruitment models, the **fits object** must be created. We will assume here that the stock recruitment object is called `sr` (which is the default name) and that the fits object is called `fits` (again, the default name). Type:

`fits <- vector("list",length(sr$ID))` *Create the fits object. It is an empty list with an element for each stock.*

`names(fits) <- sr$ID` *Name the elements of the fits object.*

The easiest way to fit models is to use the `fitallmodels` function. For help, type: `?fitallmodels`. To fit models to a particular stock, say `HERRNS`, type:

`fitallmodels("HERRNS")` *This takes a minute or two.*

The output from this command looks like this:

```
HERRNS : lslog colnl pwlnl rklnl srklnl bhlnl shlnl cogam pwgam rkgam srkgam bhgam shgam
```

These words have the following meanings

`lslog`	Least squares for log-transformed data.
`colnl`	Constant model (lognormal error).
`pwlnl`	Power model (lognormal error).
`rklnl`	Ricker model (lognormal error).
`srklnl`	Sigmoid Ricker model (lognormal error).
`bhlnl`	Beverton-Holt model (lognormal error).
`shlnl`	Shepherd model (lognormal error).
`cogam`	Constant model (gamma error).
`pwgam`	Power model (gamma error).
`rkgam`	Ricker model (gamma error).
`srkgam`	Sigmoid Ricker model (gamma error).
`bhgam`	Beverton-Holt model (gamma error).
`shgam`	Shepherd model (gamma error).

3.6.1 Fitting Models to All Stocks

Unfortunately, S-PLUS handles memory allocation inefficiently, so a special technique is required to automatically fit models to *all* of the stocks. The technique is described in Barrowman (1996). Before using the following procedure, ensure that your account is configured as described in Barrowman (1996). Then, starting in S-PLUS, type the following:

`initfits()`	*Initialize the fitting procedure.*
`q()`	*Quit S-PLUS.*
`cp /data/bioram/sr/data/fitall .`	*Copy a file that contains a program to fit models to all of the stocks.* **(Don't forget to type the space and the dot at the end.)**
`perl srun fitall &`	*Run the fitting procedure as a background process. This can take many hours. A file called* `fitall.out` *will be created. By examining this file you can see which stocks have been processed. When all of the stocks have been processed, a message will automatically be e-mailed to you.*

3.7 Example: Plotting Data and Fits

The following example illustrates some of the plotting functions available. Type:

`X11()`	*Open a graphics window.*
`srplot("HERRNS",heading=T)`	*Plot some stock-recruitment data.*
`lnlcurverk("HERRNS")`	*Draw the fitted lognormal (lnl) Ricker (rk) curve.*
`lnlcurvebh("HERRNS")`	*Draw the fitted lognormal (lnl) Beverton-Holt (bh) curve.*

To produce a plot a stock-recruit plot that includes the Ricker curve and the replacement line that is good for a talk try:

```
plotrk("HERRNS")
```

To produce a graph of the log recruitment time series for all stocks in a species, type:

```
graphlogspr(species="Herring")
```

3.8 Reproducing the Technical Report

The figures and tables in the “Summary of Worldwide Spawner and Recruitment Data” (Myers et al. 1995) can be easily reproduced provided you have already created the stock-recruitment object and fitted the stock-recruitment models using the `fitallmodels` function.

3.8.1 Producing a Stock-Recruitment Page

A useful display is a “stock-recruitment page” as in Myers et al. (1995). To produce the stock-recruitment page for, say `HERRNS`, type:

`graphstock("HERRNS")`	*Create a file called* `HERRNS.ps` *in the present directory or in a* `postscript` *subdirectory if one exists.*

To produce a “stock-recruitment page” for each stock in the database, type:

```
makeallsrpage()
```

3.8.2 Producing the Tables

Before producing the tables, several objects must be created. Type:

`makedatalen()` *Create an object called* `datalen` *giving the number of years of paired spawner-recruitment data.*

`makecvskewdf()` *Create an object called* `cvskewdf` *giving the coeficient of variability (cv), skewness (skew), etc., in a data frame (df).*

`makestocktable()` *Create an object called* `stocktable` *giving summary information.*

`makeftable()` *Create an object called* `ftable` *which is a "formatted" table.*

Then to produce the references table (p.A-1 in Myers et al. 1995), type:

`reftable()` *Generate a LaTeX file called* `reftable.tex` *in the current directory or in a* `latex` *subdirectory if one exists.*

`latex()` *Run LaTeX. If there are any problems, LaTeX may stop and display the* `?` *prompt (see Appendix C). Usually, pressing RETURN will cause LaTeX to continue until the end of the file or the occurrence of another problem. If this does not work, type* `x` *and press RETURN.*

`xdvi()` *Preview the table. (Press* `q` *to quit.)*

`dvips()` *Print the table.*

To produce the summary statistics table (p.A-21 in Myers et al. 1995), type:

`gentable()` *Generate a LaTeX file called* `gentable.tex` *in the current directory or in a* `latex` *subdirectory if one exists.*

`latex()` *Run LaTeX. If there are any problems, LaTeX may stop and display the* `?` *prompt (see Appendix C). Usually, pressing RETURN will cause LaTeX to continue until the end of the file or the occurrence of another problem. If this does not work, type* `x` *and press RETURN.*

`xdvi()` *Preview the table. (Press* `q` *to quit.)*

`dvips()` *Print the table.*

Finally, to produce the id table (p.A-29 in Myers et al. 1995), type:

`makeidtable()` *Generate a LaTeX file called* `makeidtable.tex` *in the current directory or in a* `latex` *subdirectory if one exists.*

`latex()` *Run LaTeX. If there are any problems, LaTeX may stop and display the* ? *prompt (see Appendix C). Usually, pressing RETURN will cause LaTeX to continue until the end of the file or the occurrence of another problem. If this does not work, type* `x` *and press RETURN.*

`xdvi()` *Preview the table. (Press* `q` *to quit.)*

`dvips()` *Print the table.*

To produce all of the above tables, type:

```
makeallsrtab()
```

This function will generate the tables and then move them to the correct location.

3.9 Extracting Standardized Data

The function `getallstandard` produces an ASCII file of “standardized” spawner-recruitment data. The file has 6 columns with the following contents:

1. order
2. family
3. species (with dots where spaces should be, so that SAS, for example, treats it as one character string)
4. id
5. year or yearclass
6. adult mortality (missing values are given for Pacific salmon)
7. spawners in thousand tonnes or millions
8. recruitment standardized to the same units as spawners.

The standardization is achieved using

$$R_{\text{standardized}} = R \times \text{SPR}_{F=0}(1-e^{-M})$$

To run this function, type:

```
getallstandard()
```

This takes about a half hour. The ASCII file that it creates is called `allstandard.dat`.

3.10 Updating the Object to Reflect Changes to the Database

After the stock-recruitment data object has been built, changes to the ASCII database will *not* be automatically reflected in the object. Instead the object must either be recreated from scratch or updated.

3.10.1 Changes to an Existing Stock

Suppose that the `.doc` or `.dat` file for a stock that *already exists in the data object*, say `HERRNS`, has been changed. To update the data object, type:

```
updaterecord("HERRNS")
```

3.10.2 Adding a New Stock

Suppose a new stock (call it `NEWSTOCK`) has been added to the ASCII database (i.e., files called `NEWSTOCK.dat` and `NEWSTOCK.doc` now exist in the database). To append this stock to the data object, type:

```
appendrecord("NEWSTOCK")
```

Note that the stocks in the data object are usually sorted by alphabetical order of id, while `appendrecord` simply puts the new stock at the end.

You may also wish to fit stock-recruitment models for the new stock. Do this by typing:

```
fitallmodels("NEWSTOCK")
```

This takes a minute or two.

```
names(fits) <- sr$ID
```

Update the `names` *attribute of the* `fits` *object (this assumes that the fits object is called* `fits` *and that the stock-recruitment object is called* `sr`*).*

3.11 Example: Plots for each Stock

Suppose we wish to produce a plot for each stock in the database. First we design a function called `plotpredy` ("plot predicted *y*") to produce the plot for a single stock.

The main argument of the function will be `set`, which is a somewhat cryptic name that deserves some explanation.[1] What it really means is "dataset" or "stock". But "stock" is a little ambiguous because it could mean "spawning stock biomass". In fact, the stock-recruitment database functions are designed so that the `set` argument can be given as either

1. the id of a stock, e.g., `"COD2J3KL"`, or
2. the index of the stock within the stock-recruitment data object, e.g., 156. (of course, the index changes when stocks are added or removed from the database and the stock-recruitment object is updated or rebuilt).

The other argument of the function will be `srname`.[2] It won't normally need to be specified because we will give it the default value `sro.name`, which is the name of the stock-recruitment data object. Now that the `set` and `srname` arguments have been clarified, here's the definition of our plotting function[3]:

[1] This is cryptic in part for historical reasons.

[2] This is also cryptic, again for historical reasons. But once you are familiar with `set` and `srname`, you will be able to use most of the standard stock-recruitment database functions.

[3] The source code is in `/data/bioram/sr/guide/plotpredy.s`.

```
plotpredy <- function(set,srname=sro.name) {    Define a function called plotpredy.
# compare history of landing (and SSB) with     Comments are always nice.
# predicted from gordon's model f/(f+m)
  cat(set,"\n")                                 Show which stock is being plotted.
  S <- get(srname)                              Get the stock-recruitment data object.[4]
  m <- as.numeric(S$NATMORT[set])               Get the natural mortality directly from the object.
  SSB <- getSSBalone(set)                       Get annual spawner data.[5]
  f <- getFRPL(set)                             Get annual fishing mortality data.
  year <- getyearalone(set)-1900                Get year number (two digits).
  predy <- f/(f+m)                              Here's what we want to plot on the y-axis.
  if (all(is.na(predy))) return(NULL)           Don't plot if there's no data for the y-axis.
  if (all(is.na(SSB))) return(NULL)             Don't plot if there's no data for the x-axis.
  plot(predy,SSB,type="n",xlab="",ylab="")      Set up the plot, but don't plot points.
  text(predy,SSB,year)                          Use the year as the data point.
  mtext(getsrname(set),side=3,line=.5,cex=.7)   Put a title on the plot.
}
```

Now, to produce plots, we first have to open a graphics device. For example to open an X graphics window, type

```
X11()
```

Then we can produce the plot for a single stock:

```
plotpredy("COD2J3KL")
```

But for multiple plots, and to get better control over the output, instead of using `X11`, we'll open a PostScript file:

```
ps()
```

This is a local, enhanced version of the `postscript` function. Now let's set up some graphics parameters:

[4] For convenience some functions (e.g., `getSSBalone`) will extract data from the stock-recruitment data object for you. However, in many cases you will need to extract the data yourself, e.g., to get the natural mortality from the `NATMORT` field. In this case you need to first get a copy of the stock-recruitment data object (by convention, store the copy in an object called `S`), and then extract the data from this object. If this seems a bit long-winded, it is — for historical reasons.

[5] Note that here and elsewhere, since we did not specify the name of the stock-recruitment object, the function will assume that it is the name stored in `sro.name`. Yes, this is cryptic too.

`par(oma=c(3,3,3,3))`	*For the outer margins, use 3 character spaces for the bottom, left, top, and right.*
`par(mfrow=c(5,2))`	*Use 5 rows by 2 columns of plotting panels (going from left to right, top to bottom).*
`par(mar=c(3,2,2,2))`	*For the inner margins, use 3 character spaces for the bottom, and 2 the for left, top, and right.*

Now to generate a plot for each of the stocks in the database:

`for (id in sr$ID) plotpredy(id)`	*This might take a while.*

As the loop above proceeds, the name of each stock will be printed out. When it is finished, type

`dev.off()`	*Turn of the graphics device; i.e., in this case, close the PostScript file*
`view()`	*Now view the PostScript file using the OpenWindows program* `pageview`*.* *When you are finished viewing it, select Quit from the* `pageview` *menu.* *Or press CTRL-C in the S-PLUS session to do the same thing.*
`laser()`	*Print the PostScript file if you wish.*

Instead of generating plots for all of the stocks, you might want to just generate plots for, say, the cod stocks:[6]

`for (id in sr$ID[sr$SP=="Cod"]) plotpredy(id)`	*Just plot the stocks for which the* `SP` *field (species) is set to* `"Cod"`*.*

When this is finished, you'll again have to turn off the graphics device, and then view or print the results (as explained above).

3.12 Example: Making a Taxonomic Table

Suppose you want a table containing taxonomic information from the stock-recruitment database, e.g., the order, family, scientific name, and common name for all of the species in the database[7].

In the following example, it is assumed that the stock-recruitment data object is called `sr`.

[6] Note that if you just closed the graphics device using `dev.off()` you will now need to open it up again and re-set the graphics parameters

[7] See the following subsection for a much nicer taxonomic table.

```
x <- cbind(sr$ORDER,sr$FAMILY,          Bind together the taxonomic information as columns
    sr$LATIN,sr$SP)                     of a matrix.
y <- x[!duplicated(sr$SP),]             Remove any duplicated species.
dimnames(y) <- NULL                     Remove row and/or column names.
z <- y[order(y[,4]),]                   Sort by common name (i.e., the 4th column of the ma-
                                        trix).
putcolumns("table",z,sep="|")           Use the local function putcolumns to write the infor-
                                        mation to a file called table using a vertical bar as a
                                        column separator.
```

3.13 Example: Making a Really Nice Taxonomic Table

Suppose you want a pretty (LaTeX) table giving, say, coefficient of variation (CV) of recruitment for each stock in the database, and also the average CV for each species, family, and order. Here's what the table should look like:

species	cv
AULOPIFORMES	45
Synodontidae	45
Saurida tumbil (Greater lizardfish)	45
East China Sea	45
CLUPEIFORMES	87
Clupeidae	89
Alosa aestivalis (Blueback herring)	87
Saint John River	87
Alosa pseudoharengus (Anadromous alewife)	40
Damariscotta Lake, Maine	14
Saint John River	65
Alosa pseudoharengus (Freshwater alewife)	98
Lake Ontario	98
Alosa sapidissima (Anadromous american shad)	48
Connecticut River	48
Brevoortia patronus (Gulf Menhaden)	44
Gulf of Mexico	44
Brevoortia tyrannus (Atlantic Menhaden)	66
U.S. Atlantic	66

and so forth. Before creating the table, you need to compute the CV's. To do this, type:

`makecvsdf()` *Create an object called* `cvsdf` *giving the coefficient of variability (cv), skewness (s), etc., in a data frame (df).*

The key to producing the table is a local function called `hierarchy`. For more information on hierarchy, see Appendix B. Now, here's a function called `cvtable` to produce the pretty table[8]:

[8] The source code is in `/data/bioram/sr/guide/cvtable.s`.

```
cvtable <-                                                 Define a function called cvtable.
function(cv=cvsdf$cv,srname=sro.name) {                    You can call it with no arguments.
# Produce a LaTeX table giving coefficient                 As usual, there is a comment here.
# of variation (CV) for each stock and also
# the average CV for each species, family,
# and order.

  S <- get(srname)                                         Get the stock-recruitment data object.
  cv <- round(cv)                                          Round the CV's for display in the table.
  species <- paste("{\\it ",S$LATIN," }",                  Paste together the scientific name
    " (",S$SP,")",sep="")                                    and the common name in parentheses.
  x <- cbind(S$ORDER,S$FAMILY,species,S$STOCK,cv)          Make a matrix.
  dimnames(x) <- NULL                                      Get rid of any row or column names.
  w <- msort(x,1,2,3,4)                                    Sort by order, family, species, and stock.
  dimnames(w) <- list(NULL,
    c("order","family","species","stock","CV"))            Name the columns of the matrix.
  hw <- hierarchy(w,3)                                     Format first 3 columns hierarchically.

  for (i in 1:(dim(hw)[1])) {                              For each row of the matrix ...
    order <- hw[i,"order"]                                 Get the order entry.
    family <- hw[i,"family"]                               Get the family entry.
    species <- hw[i,"species"]                             Get the species entry.
    if (order!="")                                         If the order entry is present,
      hw[i,"CV"] <-                                          set the CV to be
        round(mean(w[w[,"order"]==order,"CV"]))              the mean CV for that order.
    if (family!="")                                        If the family entry is present,
      hw[i,"CV"] <-                                          set the CV to be
        round(mean(w[w[,"family"]==family,"CV"]))            the mean CV for that family.
    if (species!="")                                       If the species entry is present,
      hw[i,"CV"] <-                                          set the CV to be
        round(mean(w[w[,"species"]==species,"CV"]))          the mean CV for that species.
  }

  formatted <- paste(                                      Now make a formatted taxonomic vector.
    uppercase(hw[,"order"]),                               Put the order in uppercase.
    "~~~~",                                                Put in some indentation.
    "{\\bf",hw[,"family"],"}",                             Put the family in boldface.
    "~~~~",                                                Put in some indentation.
    hw[,"species"],                                        Put in the species.
    "~~~~",                                                Put in some indentation.
    hw[,"stock"])                                          Put in the stock.

  m <- cbind(species=formatted,cv=hw[,"CV"])               Make a 2-column matrix.
  matrixlatex(m)   # write a LaTeX file.                   Write it as a LaTeX table.
}
```

To make the table, type:

`cvtable()` *Generate a LaTeX file called* `cvtable.tex` *in the current directory or in a* `latex` *subdirectory if one exists.*

`latex()` *Run LaTeX.*

`xdvi()` *Preview the table. (Press* `q` *to quit.)*

`dvips()` *Print the table.*

3.14 Index of S-PLUS Help Pages

Help pages in S-PLUS are a useful form of online documentation. To see help on, say the `newobject` function, type `?newobject` (press `q` to quit from the help). To print out the help page, type

```
help("newobject",offline=T)
```

3.14.1 Stock-Recruitment Database Functions

The following help pages are available[9]. The entry in boldface is the primary one.

Name	Purpose
`addall`	Read data into stock-recruitment data object.
`appendrecord`	Add new stock record to end of stock-recruitment object.
`calcrepslope`	Calculate slope of replacement line.
`extractunits`	Convert unit character string into numeric.
`fitallmodels`	Fit all stock-recruitment models for a stock.
`fits.object`	Stock and recruitment fits object.
`getRec`	Return vector of recruitment.
`getSSB`	Return vector of spawning stock biomass.
`newobject`	Set up an object to hold stock-recruitment data.
`repline`	Draw a replacement line.
`sr.object`	Stock and recruitment data object.
`sr`	**S-PLUS stock-recruitment database help file.**
`srplot`	Plot stock-recruitment data.
`unitsRec`	Compute units of recruitment in numbers of fish.
`unitsSSB`	Compute units of SSB in kg or numbers.
`updaterecord`	Update stock-recruitment data object.

[9] They actually reside in `/data/bioram/sr/functions/.Data/.Help`

3.14.2 Stock-Recruitment Modelling

The following help pages are available[10]. The entry in boldface is the primary one.

Name	Purpose
`BevertonHolt`	Beverton-Holt stock-recruitment function.
`drawsrcruve`	Draw a stock-recruitment curve.
`MLBevertonHolt`	Maximum likelihood fitting for Beverton-Holt.
`MLPower`	Maximum likelihood fitting for Power.
`MLRicker`	Maximum likelihood fitting for Ricker.
`MLShepherd`	Maximum likelihood fitting for Shepherd.
`MLSigmoidRicker`	Maximum likelihood fitting for Sigmoid Ricker.
`Power`	Power stock-recruitment function.
`Ricker`	Ricker stock-recruitment function.
`Shepherd`	Sigmoid Ricker stock-recruitment function.
`Ssr`	**S-PLUS Stock-Recruitment help file.**

3.15 Other Useful Functions

There are a wide variety of supporting functions written for use with the stock-recruitment database. Most of them can be found in the directory `/data/bioram/sr/functions`.

`renamestock` *translates the stock name (often given as a NAFO or ICES code) into a common name.*

4 World Map

This section is an add-on (updated July 28, 1997). It tells where the program for producing the world map is. It was originally created by Jessica Bridson. Stacey Fowlow and Keith Bowen have both modified it. The program plotallworld.s and plotworld.s are in `/data/bioram/sr/maps`. Just run the programs.

[10] They actually reside in `/data/bioram/sr/s/.Data/.Help`

5 The srep program

srep is a perl program designed to automatically modify files that are under SCCS control.[11] For help on using srep, type:

```
srep
```

Here is an actual example. We wanted to change the NATMORT entry in all of the yellowtail flounder .doc files in the stock-recruitment database. First we used grep to see the current entries:

```
grep NATMORT YELL*
```

This gave the following output:

```
YELL3LNO.doc:NATMORT   @ .
YELL5Z.doc:NATMORT   @ .
YELLSNE.doc:NATMORT   @ .
```

We wanted to set all of these NATMORT entries to 0.2. First we used the test mode (-t) of srep to do a dry run[12]:

```
srep -t ’NATMORT   @ .’ ’NATMORT   @ 0.2’ YELL*.doc
```

The following output appeared:

```
    YELL3LNO.doc
    < NATMORT   @ .      The < means this is the current line.
    > NATMORT   @ 0.2    The > means this is what the line would become.

    YELL5Z.doc
    < NATMORT   @ .
    > NATMORT   @ 0.2

    YELLSNE.doc
    < NATMORT   @ .
    > NATMORT   @ 0.2
```

This output indicates that the desired replacement will take place, so it is safe to run the command without the -t (i.e., not using the test mode):

[11] The srep program is located in /data/bioram/general/scripts/srep.

[12] It is a good idea to always use the test mode of srep first!

```
srep  ’NATMORT   @ .’ ’NATMORT   @ 0.2’ YELL*.doc
```

This produces a long output that starts with

```
sccs edit YELL3LNO.doc
1.3
new delta 1.4
59 lines
---changes=1---
```

and ends with a table summarizing the changes:

```
    Files Modified  Number of changes
     YELL3LNO.doc  1
       YELL5Z.doc  1
      YELLSNE.doc  1
```

The `srep` program can be used to perform much more sophisticated replacements than in the above example (using Perl regular expressions). For example,

```
  srep ’(ˆCF.*)@ \.’ ’$1@ Codfishes’ COD*.doc
```

will search all files whose names start with `COD` for lines that start with `CF` and have a dot for a value, and replace the dot with `Codfishes`.

6 The `srlist` program

`srlist` is a perl program designed to read information from the `.doc` and `.dat` files located in `/data/bioram/sr/data` and produce a LaTeX file summarizing the information.[13]

To use `srlist`, you must first `cd` to a directory you own, and then type:

`srlist`	*Create a file called* `srlist.tex` *in the current directory. Usually takes a couple of minutes.*
`latex srlist`	*Run LaTeX once.*
`latex srlist`	*Run LaTeX again.*
`dvips srlist`	*Print it out.*

[13] The `srlist` program is located in `/data/bioram/sr/scripts/srlist`.

7 The `srview` program

`srview` is a c-shell program designed to display the stock-recruitment page for a particular stock. [14]

To use `srview`, type:

```
srview id
```

where *id* is the identification code for the desired stock. If an identification code is not provided, then `srview` will display help information. If an identification code is provided, then the stock-recruitment page for the stock will be displayed using ghostview.

[14] The `srview` program is located in `/data/bioram/general/scripts`.

8 The `upview` program

`upview` is a c-shell/S-PLUS program designed to update information about a stock in the sr data object using the the `.doc` and `.dat` files located in /data/bioram/sr/data+. The updated information is then used to create a stock-recruitment page. [15]

To use `upview`, you must first `cd` to `/data/bioram/sr/srobject`, and then type:

```
upview id
```

where *id* is the identification code for the stock to be updated. If an identification code is not provided, then `upview` will display help information. If an identification code is provided, then the S-PLUS function `updateview` will be called to update the sr object and produce a postscript file of the sr information for the updated stock. The postscript file is then displayed using ghostview.

[15] The `upview` program is located in `/data/bioram/sr/srobject`.

9 The nview program

nview is a c-shell/S-PLUS program designed to add information about a new stock to the sr data object using the the .doc and .dat files located in /data/bioram/sr/data+. The information about the new stock is then used to create a stock-recruitment page. [16]

To use nview, you must first cd to /data/bioram/sr/srobject, and then type:

```
nview id
```

where *id* is the identification code for the new stock. If an id code is not provided, then upview will display help information. If an identification code is provided, then the S-PLUS function newview will be called to add the new information to the sr object and produce a postscript file of the sr information for the new stock. The postscript file is then displayed using ghostview.

[16] The nview program is located in /data/bioram/sr/srobject.

10 Making the Database Available by Anonymous FTP

The anonymous ftp directory is

```
/var/ftp/pub/sr
```

When copies of the files from `/data/bioram/sr/data` are placed in this directory, they are available for anonymous downloading.

`srftp` is a perl program designed to extract the most recent versions of the files in the stock-recruitment database (including the `README` file) and put them in the anonymous ftp directory[17]. To use it, simply type (from RAM):

```
srftp
```

This takes a few minutes. **When it is finished, be sure to remove any files that you do not want to be available by anonymous ftp**. Stock-recruitment files mentioned in the file `OMIT.tex` [18] will automatically be deleted from the ftp directory. Here are the instructions to give to people[19]:

```
The data are available by anonymous ftp from RAM.biology.dal.ca
(ip address 129.173.16.164), in the directory pub/sr.
The README file contains information on the data format, etc.
```

[17] The `srftp` program is located in `/data/bioram/general/scripts/srftp`.

[18] The file `OMIT.tex+ is located in /data/bioram/sr/data.`

[19] These instructions are located in `/data/bioram/sr/guide/ftp`.

11 Putting the data on the Web

See README in myers/public$_{h}tml/functions/$

12 The species database

We often need data that is common to species not populations. For this we have a separate database located in `/data/bioram/species/data`. The `README` file in that directory gives the data format.

12.1 S-PLUS programs to read the species database

S-PLUS programs to read the species database are in `/data/bioram/species/s`. To run these programs, change to this directory, start S-PLUS, and type:

```
readspecies()
```

Then build a summary table by typing:

```
makespeciestable()
```

12.2 The `speciesfile` program

A perl program, located in `/data/bioram/general/scripts/speciesfile`, creates a template species data file by using information from a `.doc` file in the stock-recruitment database. For example, to build a species data file based on information in the `NZSNAPPLENTY.doc` stock-recruitment file, type:

```
speciesfile NZSNAPPLENTY
```

12.3 Combining information from the species database and the stock-recruitment database

An S-PLUS function called `makeallinfo()` creates an ASCII file called `allinfo.dat`. There is a file located in `/data/bioram/sr/srobject/allinfo.doc` that describes the format of the file as follows:

```
Fields are whitespace separated, as follows:

1. ID
2. SP
3. LATIN
4. FAMILY
```

```
5. ORDER
6. STOCK
7. AGERECDAT
8. AGEMAT
9. NATMORT
10.LATITUDE
11.Fecundity
12.EGGDIAM
13.HABITAT
14.EGGSORYOUNG
15.LHATCH

Missing values are represented by dots.
```

The `makeallinfo()` function must be run in the same directory as the `getallstandard()` function that creates `allstandard.dat`. In fact, before running `makeallinfo()`,
the following must be true:

1. the S-PLUS stock-recruitment object must exist,

2. the `allstandard.dat` file must exist (in the current directory),

3. the S-PLUS species objects must exist

A References

Barrowman, N. J. 1996. A scheme for running big S-PLUS jobs. Unpublished manuscript.

Barrowman, N. J. and R. A. Myers. 1995. Programming conventions and style. Unpublished manuscript.

Myers, R.A., J. Bridson, and N. J. Barrowman 1995. Summary of Worldwide Spawner and Recruitment Data. Can. Tech. Rep. Fish. Aquat Sci. 2024: iv + 327p.

B The `hierarchy` function

The `hierarchy` function is a locally-written function that is useful when producing hierarchical LaTeX tables in S-PLUS. Here's an example to illustrate how `hierarchy` works. First, we define a matrix:

```
x <- c("cat","cat","dog","dog","dog","mouse","mouse")
y <- c("small","big","small","small","big","small","small")
z <- c("Felix","Morris","King","Spot","Butch","Mickey","Minney")
a <- c("1","2","3","4","5","6","7")
w <- cbind(x,y,z,a)
```

The `w` matrix looks like this:

```
           x       y        z   a
[1,] "cat"   "small" "Felix"  "1"
[2,] "cat"   "big"   "Morris" "2"
[3,] "dog"   "small" "King"   "3"
[4,] "dog"   "small" "Spot"   "4"
[5,] "dog"   "big"   "Butch"  "5"
[6,] "mouse" "small" "Mickey" "6"
[7,] "mouse" "small" "Minney" "7"
```

Now, to format it hierarchically, type:

`hierarchy(w,2)` *Treat the first 2 columns of the matrix as classification variables. Note that the matrix should already be sorted by the classification variables. (The* `msort` *function can be used to sort by several columns of a matrix.)*

The result is

```
            x       y        z   a
 [1,] "cat"   ""      ""       ""
 [2,] ""      "small" ""       ""
 [3,] ""      ""      "Felix"  "1"
 [4,] ""      "big"   ""       ""
 [5,] ""      ""      "Morris" "2"
 [6,] "dog"   ""      ""       ""
 [7,] ""      "small" ""       ""
 [8,] ""      ""      "King"   "3"
 [9,] ""      ""      "Spot"   "4"
[10,] ""      "big"   ""       ""
[11,] ""      ""      "Butch"  "5"
[12,] "mouse" ""      ""       ""
[13,] ""      "small" ""       ""
[14,] ""      ""      "Mickey" "6"
[15,] ""      ""      "Minney" "7"
```

If, instead, we had typed `hierarchy(w,1)`, the result would be

```
            x       y        z   a
 [1,] "cat"   ""      ""       ""
 [2,] ""      "small" "Felix"  "1"
 [3,] ""      "big"   "Morris" "2"
 [4,] "dog"   ""      ""       ""
 [5,] ""      "small" "King"   "3"
 [6,] ""      "small" "Spot"   "4"
 [7,] ""      "big"   "Butch"  "5"
 [8,] "mouse" ""      ""       ""
 [9,] ""      "small" "Mickey" "6"
[10,] ""      "small" "Minney" "7"
```

C LaTeX Errors in Automatically-Produced Tables

Several stock-recruitment summary tables can be produced automatically. The tables are written in LaTeX and use information from the stock-recruitment database. For example, the `SOURCE` field in the database is used to produce a list of references. For this reason, the information in the `SOURCE` field should be in LaTeX format. For example, consider the following `SOURCE` field:

```
SOURCE    @ Gavaris, S., D. Clark, and P. Perley. 1994. Assessment of Cod
in Division 4X. DFO Atlantic Fisheries Res. Doc. 94/(# not given): 27 p.
```

There is a LaTeX error here, namely the # symbol, which has a special meaning in LaTeX. To get a # symbol in LaTeX, you have to use `\#`. This means that in the automatically generated reference list, which is a LaTeX file, an error will occur. In other words, LaTeX will stop and display the `?` prompt. Usually, pressing RETURN will cause LaTeX to continue until the end of the file or the occurrence of another problem. If this does not work, type `x` and press RETURN.

To fix the problem, the relevant `.doc` file in the database should be corrected. Alternatively, the LaTeX file that was automatically generated can be edited on a once-off basis[20].

D The `basic.First.s` File

If you define an S-PLUS function called `.First` in a particular directory, it will be executed whenever you start up S-PLUS in that directory. To use the S-PLUS stock-recruitment functions, several things need to be put in a `.First` function. A good default `.First` function is given in

```
/data/bioram/sr/data/basic.First.s
```

Here is what it contains:

```
.First <- function() {
  attach("/data/bioram/sr/functions/.Data")             The stock-recruitment database functions.
  attach("/data/bioram/sr/s/.Data")                     The stock-recruitment modelling functions.
  dyn.load("/data/bioram/myers/myers/math/lngam.o")     A replacement for the lgamma function.[21]
  sr.database.path <<- "/data/bioram/sr/data"           Location of the ASCII files.
  sro.name <<- "sr"                                     Name for the stock-recruitment data object.
  debug <<- F                                           Do not use the debug mode by default.
  fito.name <<- "fits"                                  Name for the fits object.
  new.fits <<- T                                        Use the “new fitting scheme”.
}
```

[20] This is a poor solution, however, since the problem is bound to crop up again. It’s better to fix the `.doc` file once and for all.

[21] For the purposes of stock-recruitment model fitting assuming a gamma distribution, the $\log\Gamma(\cdot)$ function is required. There is a built-in S-PLUS function called `lgamma`, but it was found to be somewhat unreliable. Instead, we use a custom function called `lngam`. Before using it the `dyn.load` function call above must be used.