PBStools: User's Guide

Rowan Haigh and Jon T. Schnute

January 10, 2025

Contents

List of Tables

List of Figures

Page left blank for printing purposes.

What is PBStools?

PBStools provides an R interface for algorithms commonly used in fisheries. The scope of this package is by no means comprehensive, having grown from the need to satisfy tasks specific to British Columbia (BC) ocean fisheries. Many of the functions provide a quick way to visualize data, and in some cases perform preliminary analyses. Though oriented to users at the Pacific Biological Station (PBS), these functions may provide a broad utility to users at other locales. The User Guide is organised into sections that loosely classify the functions by theme – (1) Utility, (2) Biology, (3) Fishery, (4) Survey, (5) Spatial, (6) Temporal, and (7) Catch Reconstruction. Within each section, the functions are described alphabetically.

PBStools depends heavily on two other R package: **PBSmapping** (Schnute et al., 2004*a*; Boers et al., 2004) and **PBSmodelling** (Schnute et al., 2006). We use the latter to implement several Graphical User Interfaces (GUIs) that facilitate a few select functions. Most functions, however, are designed for use on the command line or in sourced code. Many of the functions invisibly return auxiliary output in a list object called `PBStool`, which is located in the temporary working environment `.PBStoolEnv`. Accessor functions called `ttget`, `ttcall`, and `ttprint` enable the user to get, call, and print objects from the `.PBStoolEnv` environment, while `ttput` puts (writes) obects to `.PBStoolEnv`.

Amongst the various functions in the package, we include those that implement some of the models published by the authors. For example, `calcSG` estimates parameters for the Schnute growth model (Schnute, 1981), which offers a versatile alternative to the von Bertalanffy growth model (`calcVB`). The functions presented in the Survey section make use of the binomial-gamma model (Schnute and Haigh, 2003) for characterizing survey strata populations when designing or simulating groundfish trawl surveys. The function `runCCA` implements the composition analysis of Schnute and Haigh (2007) to estimate total mortality Z of rockfish (*Sebastes* spp.) using three combined effects: survival at age (S_a), fishery selectivity at age (β_a), and relative recruitment anomalies (R_a). Finally, the function `buildCatch` provides an efficient method for reconstructing rockfish catch from 1918 to the present along the BC coast (Haigh and Yamanaka, 2011).

Also available in the package directory `./library/PBStools/sql` we provide a number of useful SQL queries for DFO[1] commercial fisheries databases – PacHarvest (trawl, 1996–2007), PacHarvHL (hook and line, 1994–2006), GFCatch (historical landings, 1954–1995), GFBioSQL (biological samples, 1946–2012), and GFFOS (integrated fisheries, 2002–present). To launch SQL queries, **PBStools** relies on the R package **RODBC**. If you have access to the DFO network and have privileges to query these databases, the function `getData` can send the queries to the remote servers and return a data frame called `PBSdat` in the global environment. In the document, we highlight queries for DFO personnel using text with a background shaded `moccasin`[2]. (Examples are shaded `aliceblue` and console output is shaded `honeydew`.) Note that many of these queries might act as useful templates for users outside DFO for similar purposes. Querying databases directly via SQL commands from R usually proves much more efficient than launching Microsoft Access Queries from a front-end database shell.

Originally, **PBStools** evolved over time (2007–2012) within the R package **PBSfishery**[3], along with a convenient Graphical User Interface (GUI) tool for interacting with **PBSmapping** and useful datasets (regional boundaries, key codes, example data). In April 2012, we decided to split

[1]Department of Fisheries and Oceans, a.k.a. Fisheries and Oceans Canada

[2]RGBs: moccasin=(255,228,181), aliceblue=(240,248,255), honeydew=(240,255,240)

[3]found on Google's Project Hosting site: http://code.google.com/p/pbs-fishery/

PBSfishery into three separate libraries – **PBStools**, **PBSmapx**, and **PBSdata** – for distribution on CRAN.

What is PBS?

The Pacific Biological Station is the oldest fisheries research centre on Canada's Pacific coast and forms part of a network of nine major scientific facilities operated by Fisheries and Oceans Canada. Located in Nanaimo, British Columbia, the Station is home to scientists, technicians, support staff and ships' crews whose common interests are the coastal waters of British Columbia, the Northeast Pacific Ocean, the Western Arctic and navigable waters east to the Manitoba, Saskatchewan border.

PBS was established in 1908 and is the principal centre for fisheries research on Canada's west coast. There are some 22 structures on the site including a four-story office/wet lab building, specialty storage structures for hazardous chemicals and salt water pumping facilities. PBS maintains a number of workshops for research support. There is a wharf used for loading, unloading, and berthage of research vessels, as well as a small boat dock for inshore research boats. PBS also maintains a library and meeting facilities. Aquatic facilities, primarily used by Aquaculture Science, include ambient temperature and heated salt water and fresh water.

Research at PBS responds to stock assessment, aquaculture, marine environment, habitat, ocean science, and fish productivity priorities. Some fisheries management activities are also conducted here.

Figure 1. Pacific Biological Station (PBS), Nanaimo BC

For more information, see:
http://www.pac.dfo-mpo.gc.ca/science/facilities-installations/pbs-sbp/index-eng.htm.

1 Utility functions

The utility functions described in this section cover a broad range of activities. The number of functions has grown over time and inevitably, some of the functions may replicate the behaviour of functions in other packages. Additionally, the utility of these functions can either decline or cease based on changes in dependent packages. Feedback is appreciated from any users of **PBStools**.

1.1 biteData

Subset a matrix or data frame using a vector object with the same name as one of the fields in the data matrix/frame and return the subset data matrix/frame. If there are no fields with the name of the vector object, the data matrix/frame is returned unaltered.

Due to the current nature of the algorithm, the user cannot supply an ad hoc vector directly as the argument. This means the vector object must exist before calling `biteData`.

Not allowed: `biteData(swiss,Education=1:5)`
Allowed: `Education=1:5; biteData(swiss,Education)`

Example (biteData):

```
pbsfun=function(){
  cat("Records in original swiss object by Education\n")
  print(sapply(split(swiss$Education,swiss$Education),length))
  Education=1:5; test=biteData(swiss,Education)
  cat("Records in swiss object subset where Education=1:5\n")
  print(sapply(split(test$Education,test$Education),length))
  invisible() }
pbsfun()
```

Output (biteData):

```
Records in original swiss object by Education
 1  2  3  5  6  7  8  9 10 11 12 13 15 19 20 28 29 32 53
 1  3  4  2  4  7  4  3  2  1  5  3  1  1  1  1  2  1  1
Records in swiss object subset where Education=1:5
1 2 3 5
1 3 4 2
```

1.2 chewData

Remove records from a data frame or matrix that contribute little information to unique categories of the factor specified. Records are removed if a specified factor's categories don't number more than the specified minimum. If there are no fields with the name of the specified factor, the data matrix/frame is returned unaltered.

Example (chewData):

```
pbsfun=function(){
  cat("Unique records in 'iris$Petal.Width'\n")
  print(sapply(split(iris$Petal.Width,iris$Petal.Width),length))
  test=chewData(iris,"Petal.Width",5)
  cat("'Petal.Width' categories fewer than 5 removed\n")
  print(sapply(split(test$Petal.Width,test$Petal.Width),length))
  invisible() }
pbsfun()
```

Output (chewData):

```
Unique records in 'iris$Petal.Width'
0.1 0.2 0.3 0.4 0.5 0.6   1 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9   2 2.1 2.2
  5  29   7   7   1   1   7   3   5  13   8  12   4   2  12   5   6   6   3
2.3 2.4 2.5
  8   3   3
'Petal.Width' categories fewer than 5 removed
0.1 0.2 0.3 0.4   1 1.2 1.3 1.4 1.5 1.8 1.9   2 2.1 2.3
  5  29   7   7   7   5  13   8  12  12   5   6   6   8
```

1.3 collectFigs

Collect figures (currently only encapsulated postscript `.eps` supported) into one document using a latex compiler (`latex.exe`, `dvips.exe`, `ps2pdf.exe`). The code constructs a `.tex` file that uses all available `.eps` files in the specified directory. The final result is a bookmarked PDF file called `<fout>.pdf`, where `fout` is user-supplied prefix for the output file.

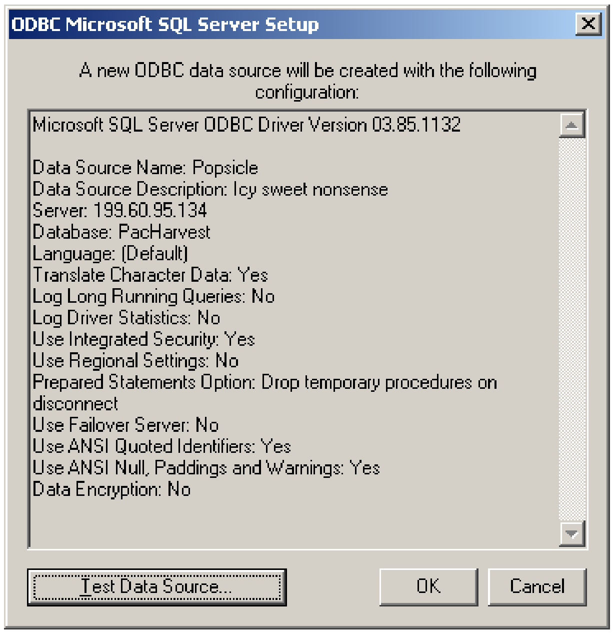
1.4 confODBC

Use a command line utility found in Windows OS called `odbcconf.exe` that configures an ODBC Data Source Name from the system's command line. The function runs the command line utility with the user-supplied information, and configures a User DSN on the his/her computer.

In Windows XP, the new User DSN can be seen by navigating to:
<Control Panel><Administrative Tools><Data Sources (ODBC)> (Figure 2).

Example (confODBC):

```
pbsfun=function(os=.Platform$OS.type,dfo=TRUE) {
  if (os=="windows" && dfo)
    confODBC(dsn="Popsicle", server="GFDB", db="PacHarvest",
      driver="SQL Server", descr="Icy sweet nonsense", trusted=TRUE)
  else showMessage("Only functional for DFO personnel using Microsoft Windows OS")
  invisible() }
pbsfun()
```



A new ODBC data source will be created with the following configuration:

Microsoft SQL Server ODBC Driver Version 03.85.1132

Data Source Name: Popsicle
Data Source Description: Icy sweet nonsense
Server: 199.60.95.134
Database: PacHarvest
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Integrated Security: Yes
Use Regional Settings: No
Prepared Statements Option: Drop temporary procedures on disconnect
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
Data Encryption: No

Figure 2. (*PBStools-confODBC*) ODBC data source configuration for DSN "Popsicle".

1.5 convFY

Convert a vector of dates into a vector of fishing or fiscal years based on a specified month to start the fishing/fiscal year. The function returns a numeric vector of fishing/fiscal years of equal length to the input vector of dates. The output vector has names representing year-month `YYYY-MM`. If the starting month is set to 1, the output dates also represent calendar years.

Example (convFY):

```
pbsfun=function() {
  cat("Fishing years starting in April\n")
  print(convFY(paste("2009-",pad0(1:12,2),"-15",sep="")))
  invisible() }
pbsfun()
```

Output (convFY):

```
Fishing years starting in April
2009-01 2009-02 2009-03 2009-04 2009-05 2009-06 2009-07 2009-08 2009-09
   2008    2008    2008    2009    2009    2009    2009    2009    2009
2009-10 2009-11 2009-12
   2009    2009    2009
```

1.6 convYM

Convert a 2-element vector of date limits into a vector of year-months where every month within the limits is represented. The function returns an expanded character vector of ordered year-months `YYYY-MM` between and including the specified date limits. This character vector can be used for labelling and other factor-like routines.

Example (convYM):

```
pbsfun=function() {
  cat("Year-month labels between the specified date limits\n")
  print(convYM(c("2009-07-01","2011-11-11")))
  invisible() }
pbsfun()
```

Output (convYM):

```
Year-month labels between the specified date limits
 [1] "2009-07" "2009-08" "2009-09" "2009-10" "2009-11" "2009-12" "2010-01"
 [8] "2010-02" "2010-03" "2010-04" "2010-05" "2010-06" "2010-07" "2010-08"
[15] "2010-09" "2010-10" "2010-11" "2010-12" "2011-01" "2011-02" "2011-03"
[22] "2011-04" "2011-05" "2011-06" "2011-07" "2011-08" "2011-09" "2011-10"
[29] "2011-11"
```

1.7 convYP

Convert a vector of dates into one of year-periods through binning the dates into intervals specified in days. The routine actually fudges the integer day interval to a real number that will create even breaks. The function returns a vector (with the same length as the input date vector) that specifies dates as real numbers where the integer part is the year and the fraction part is the proportion of the year corresponding to the bin's right-most value. Each element of the vector also has a name that specifies the year and bin number.

Example (convYP):

```
pbsfun=function() {
  x=paste("2009-",c("03-31","06-30","09-30","12-30"),sep="")
  cat("Year periods based on numeric value of input dates\n")
  print(convYP(x,30))
  invisible() };  pbsfun()
```

Output (convYP):

```
Year periods based on numeric value of input dates
2009-03 2009-06 2009-09 2009-12
2009.25 2009.50 2009.75 2010.00
```

1.8 createDSN

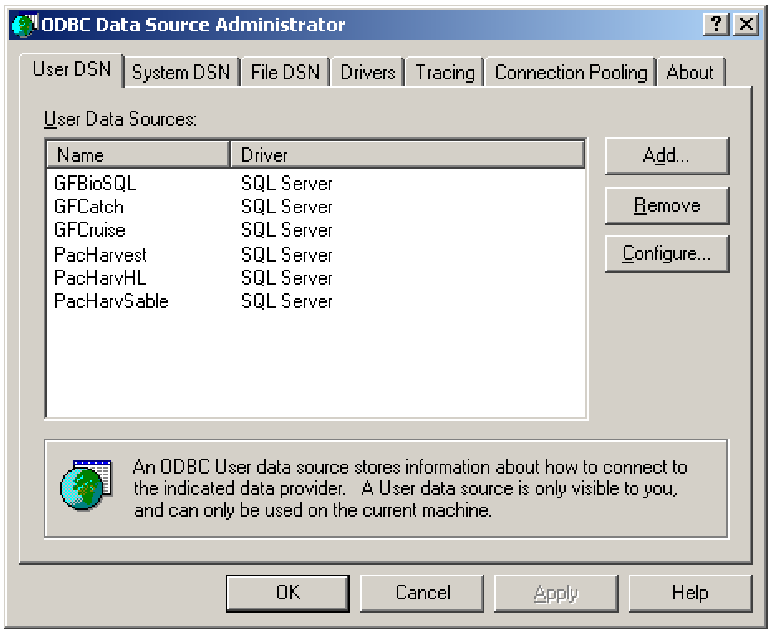
Create a suite of User DSNs for PBS groundfish databases, currently hosted on the server DFBCV9TWVASP001. This function uses `confODBC` and will overwrite existing DSNs if they exist. Currently, six DSNs for DFO databases will be created:

```
GFBioSQL   GFCatch   GFCruise   PacHarvest   PacHarvHL   PacHarvSable
```

In XP, the new User DSNs can be seen by navigating to:
<Control Panel><Administrative Tools><Data Sources (ODBC)> (Figure 3).

Example (createDSN):

```
pbsfun=function(os=.Platform$OS.type,dfo=FALSE) {
  if (os=="windows" && dfo) createDSN()
  else showMessage("Only functional for DFO personnel using Microsoft Windows OS")
  invisible() }
pbsfun()
```



User Data Sources:

Name	Driver
GFBioSQL	SQL Server
GFCatch	SQL Server
GFCruise	SQL Server
PacHarvest	SQL Server
PacHarvHL	SQL Server
PacHarvSable	SQL Server

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Figure 3. (*PBStools-createDSN*) ODBC data source configuration for DFO groundfish database DSNs.

1.9 crossTab

Run a cross tabulation on a data frame, summarizing z-values by specified 'factors'. This function requires the R package **reshape**, and uses the functions `melt.data.frame` and `cast` therein. The output reports a cross tabulation of `y` with values `z` summarized by `func`. The function `crossTab` is useful for summarizing data query results (e.g., total catch by year and PMFC major area) when exploring / debugging functions like `buildCatch`.

1.10 fitLogit

Fit binomial data using a `logit` link function in a `glm`. The function returns a GLM fit of the binomial data using the specified independent observations. See the 'Value' section of the `glm` help file in the **stats** package.

1.11 flagIt

Take a coordinate `(a,b)` and label it using a diagonal line of radius `r` and angle `A`. A diagonal dotted line in light grey radiates out from a central coordinate and the coordinate values are used to label the coordinate itself. The function adjusts for the aspect ratio created by different x- and y-limits, and for different x- and y-dimensions of the plot. This ensures that an angle looks correct in the plotting space. The code invisibly returns a list of the vectors (a,x) and (b,y), the angle in radians, the original x-value calculated in a square Cartesian system, and the final x- and y-coordinates on the periphery of a circle.

Example (flagIt):

```
pbsfun = function() {
  plot(0,0,type="n",xlim=c(0,20),ylim=c(0.2,0.8))
  points(10,0.5,pch=20,col="blue")
  for (i in seq(10,360,10))
    flagIt(a=10, b=0.5, r=0.25, A=i,col="blue",cex=0.7)
}; pbsfun()
```

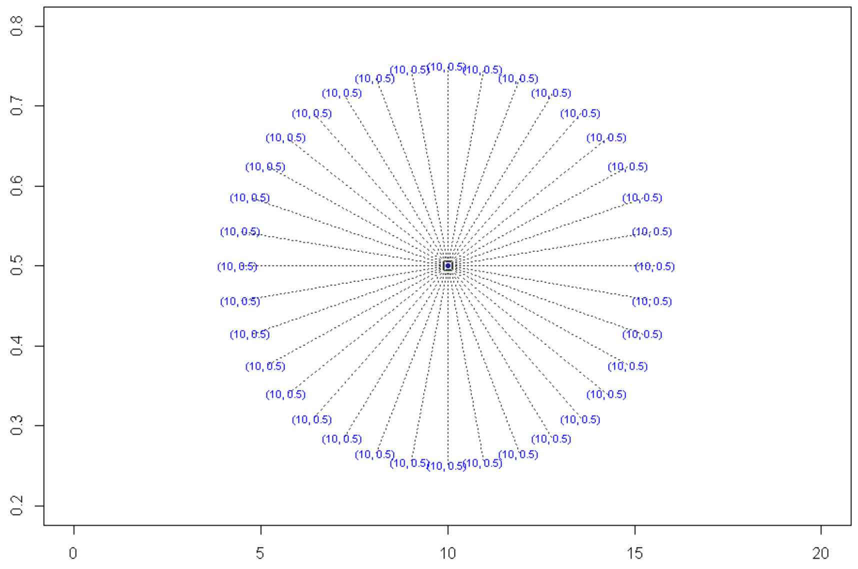


Figure 4. (*PBStools-flagIt*) The point (10, 0.5) flagged using vectors at 10° angles.

1.12 getData

Attempt to get data from a variety of types (e.g., data objects/files, queries/tables) and remote sources (`ORA`, `SQL`, `MDB`, `DBF`, `XLS`, `FILE`). The data table retrieved is placed in a global data frame object called `PBSdat`. If the type is `MDB`, the query/table name and the database name are attached as attributes. If the type is `SQL` or `ORA`, the DB name, query name, and SQL code are attached as attributes. This function, which remains one of the key utilities in **PBStools** for retrieving data, is essentially a glossy wrapper for the function `sqlQuery` in the package **RODBC**.

There are a few quirks that a user should keep in mind. This function was designed primarily to access remote SQL Server and Oracle databases using files containing structured query language (SQL). Unfortunately, SQL syntax varies between the two database systems but the differences are fewer than the similarities. Invariably, the Oracle system offers the most challenges to one's sanity.

A major disadvantage is the inabilty in Oracle to create temporary tables that exist only when the query is run. In SQL Server, temporary tables adopt the hatch/pound prefix '`#`' and are incredibly useful for running pieces of code that generate results for use by subsequent pieces of code. In Oracle, result subsets must take the form of embedded `SELECT` statements, which can get get very messy very quickly.

Another feature of Oracle is the need to prefix table names with the name of the database. The function `getData` does this automatically by replacing the variable `@table` in the SQL code file with the value from the argument `dbName`. The user could also spell out the table name when building a query file (e.g., `GFFOS.GEAR` instead of `@table.GEAR`). If the user chooses to send an SQL expression directly (not using an SQL code file) to an Oracle database (i.e., `type="ORAX"`), then the user must include the table name prefix explicitly.

Although, `getData` constructs an ODBC connection string, Oracle still needs information in a file call `tnsnames.ora` from a local installation of an Oracle client program (see http://www.orafaq.com/wiki/Tnsnames.ora). As we are biologists rather than computer programmers, it remains unclear to us whether the Oracle software installation is strictly necessary. It could be that RODBC simply needs the information in a file called `tnsnames.ora` that exists on the user's path. Feedback would be welcome on this issue.

1.12.1 Argument details

`fqtName`	string specifying name of file, query, or table; can also be an explicit SQL statement when `TYPE=SQLX` or `TYPE=ORAX`.
`dbName`	string specifying the name of a remote database; types supported: `XLS` (Excel), `MDB` (ACCESS), `SQL` (Sequel Server), `ORA` (Oracle).
`strSpp`	string specifying species code for the SQL variable `@sppcode`; IMPORTANT: to retrieve a remote table or query rather than send SQL code, set `strSpp=NULL`.
`server`	string specifying the name of a remote server (e.g., DFBCV9TWVASP001 for an SQL server, GFSH or ORAPROD for an Oracle server.
`type`	type of file: `FILE` = local data object, a saved binary file `*.rda`, a dumped ASCII file `.r`, a comma-delimited file `.csv` or `.txt`; specify the file name without the extension. `XLS` = Microsoft Excel spreadsheet (specific worksheet is specified by `fqtName`). `DBF` = Dbase IV table. As each `.dbf` file acts like a separate table, use `fqtName` to specify the `.dbf` file without its extension. `MDB` = Microsoft ACCESS database query or table. `SQL` = SQL Server SQL query (code file) or table. `ORA` = Oracle SQL query (code file) or table. `SQLX` = SQL Server SQL query code (direct expression). `ORAX` = Oracle SQL query code (direct expression).
`path`	string specifying path to local file, MDB database, or SQL query code file.
`trusted`	logical: if `TRUE`, allow SQL Server to use a trusted DFO login ID.

`uid, pwd`	user ID and password for authentication (if required).
`noFactors`	logical: if `TRUE`, convert all factor fields to character fields.
`noLogicals`	logical: if `TRUE`, convert all logical fields to characters "T" or "F".
`rownum`	numeric indicating how many rows of a table to return. The default 0 means all rows are returned (entire table); this argument only affects downloads of remote database tables.
`mindep`	numeric specifying minimum depth for the SQL variable `@mindep`.
`maxdep`	numeric specifying maximum depth for the SQL variable `@maxdep`.
`surveyid`	numeric specifying survey ID in GFBio for the SQL variable `@surveyid`.
`survserid`	numeric specifying survey series ID in GFBio for the SQL variable `@survserid`.
`fisheryid`	numeric specifying fishery ID number for the SQL variable `@fisheryid`.
`logtype`	string specifying log type code for the SQL variable `@logtypeval`.
`doors`	numeric specifying door spread width for the SQL variable `@doorsval`.
`speed`	numeric specifying vessel trawling speed for the SQL variable `@speedval`.
`mnwt`	numeric specifying mean weight (g) of a species for the SQL variable `@mnwt`.
`tarSpp`	string specifying species code(s) for the SQL variable `@tarcode`.
`major`	numeric specifying PMFC major area codes(s) for the SQL variable `@major`.
`top`	numeric specifying top *N* records for the SQL variable `@top`.
`dummy`	numeric or character to use *ad hoc* wherever the SQL variable `@dummy` appears.
`...`	additional arguments for `RODBC`'s function `sqlQuery` (specifically if the user wishes to specify `rows_at_time=1`).

1.13 getFile

Attempt to get data from the specified file name (without specifying the extension). If a data object with this name exists in the specified target environment (local or global), the function will not attempt to reload it unless the users specifies `reload=TRUE`. Failing to find an R data object, the function tries the binary libraries for package data objects, unless the user specifies `ignore.data=TRUE`. Thereafter, the code searches for remote data files in the following order: (i) local binary files with the extension `.rda`, (ii) ASCII data files created using `dump` and with the extension `.r`, (iii) comma-data delimited files `.csv`, and (iv) ordinary text files `.txt`.

Depending on the choice of scope (`L` = local, `G` = global), the function places the newly acquired data into an R-object with the same name as that used for the search into either the parent frame or the global environment, respectively.

1.13.1 Argument details

`...`	a sequence of names or literal character strings.
`list`	a character vector of potential objects.
`path`	string specifying path to data files `*.rda` and `*.r`.
`scope`	string specifying either local `"L"` or global `"G"` placement (destination) of data file.
`use.pkg`	logical: if `TRUE` ignore binaries called by the `data` command.
`reload`	logical: if `TRUE`, force a reloading of the data object into R's memory.

`try.all.frames`	logical: if `TRUE`, look for named object(s) in all frames, starting from the parent frame and working backwards.
`use.all.packages`	logical: if `TRUE`, look for data binaries in all the packages installed in the user's R library folder.

1.14 getName

Get string names from user supplied input. If the name supplied exists as an object in the parent frame, the object will be assessed for its potential as a source of names using the following criteria:

If `fnam` exists as a list, the function returns the names of the list.
If `fnam` exists as a string vector, the function returns the strings in the vector.
If `fnam` does not exist as an object, it simply returns itself as a string.

Example (getName):

```
pbsfun=function() {
  cat("Data object 'swiss' doesn't appear in the parent frame\n")
  print(getName(swiss))
  swiss=swiss
  cat("And now it does, so it acts like a source of names\n")
  print(getName(swiss))
  invisible() }
pbsfun()
```

Output (getName):

```
Data object 'swiss' doesn't appear in the parent frame
[1] "swiss"
attr(,"type")
[1] "literal"
attr(,"len")
[1] 1
And now it does, so it acts like a source of names
[1] "Fertility"        "Agriculture"      "Examination"
[4] "Education"        "Catholic"         "Infant.Mortality"
attr(,"type")
[1] "list"
attr(,"len")
[1] 6
```

1.15 isThere

Check to see whether objects physically exist in the specified environment. This differs from the function `exists` in that the latter sees objects across environments. This function looks in the specified environment and the object must be physically present to elicit a `TRUE` response.

Example (isThere):

```
pbsfun=function() {
  cat("Data object 'swiss' appears to exist in pos=0\n")
  cat(paste("   exists(\"swiss\",envir=sys.frame(0))",
    exists("swiss",envir=sys.frame(0))),"\n")
  cat("But it isn't really there...\n")
  cat(paste("   isThere(\"swiss\",envir=sys.frame(0))",
    isThere("swiss",envir=sys.frame(0))),"\n")
  invisible() }
pbsfun()
```

Output (isThere):

```
Data object 'swiss' appears to exist in pos=0
   exists("swiss",envir=sys.frame(0)) TRUE
But it isn't really there...
   isThere("swiss",envir=sys.frame(0)) FALSE
```

1.16 lenv, penv, genv

These shortcut functions get the local (`lenv`), parent (`penv`), and global (`genv`) environments, respectively.

1.17 listTables

List the tables in a specified SQL, Oracle or Microsoft ACCESS database. User can choose table type and/or search table names through pattern matching. The results of the pkgRODBC function `sqlTables` are dumped to a data frame object called `PBSdat`. If no pattern is specified, all table names in the target database are returned. If a pattern is provided, only table names containing the pattern are returned. Pattern matching is case-sensitive. Additionally, the user can choose table types, depending on the server type.

Example (listTables):

```
pbsfun=function(os=.Platform$OS.type,dfo=FALSE) {
  if (os=="windows") {
    if(dfo) {
      cat("Tables in 'PacHarvest' matching pattern 'Species'\n")
      listTables("PacHarvest",pattern="Species") }
    else {
      cat("Tables in 'Examples.mdb'\n")
      listTables("Examples", type="MDB",ttype=c("TABLE","VIEW"),
        path=.getSpath()) } }
  else showMessage("If logged onto DFO network, set argument 'dfo=T'")
  invisible() }
pbsfun()
```

Output (listTables):

```
Tables in 'Examples.mdb'
"Ex01_Sample_Info"   "Ex02_Species_Abundance"   "Ex03_Portfolio"
```

1.18 makeLTH

Make a longtable header for printing an `xtable` in Sweave, source:
http://tex.stackexchange.com/questions/41067/caption-for-longtable-in-sweave?rq=1

This code was provided by a clever Swedish fellow (pseudonym = `chepec`) to elegantly break a longtable across pages by specifying *Continued on next page* as a footer where the table breaks, and changes the caption on the following page to “Table xx – continued from previous page”.

1.19 prime

Report the prime numbers given an integer vector. If none are found, the function returns `NULL`. The function will reject non-vector objects, non-numeric vectors, and numeric vectors that do not comprise integers.

1.20 revStr

Reverse a string character by character. Code taken from examples in the **base** package `strsplit`.

Example (revStr):

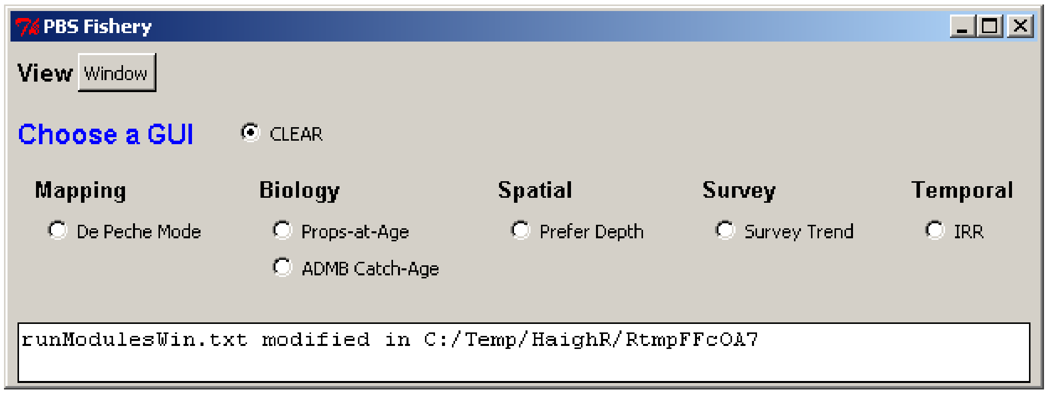
```
pbsfun=function() {
  cat(revStr("Nardwuar the Human Serviette"),"\n")
  invisible() }
pbsfun()
```

Output (revStr):

```
etteivreS namuH eht rauwdraN
```

1.21 runModules

Display an interactive GUI to run the various modules in **PBStools** (Figure 5).



```
runModulesWin.txt modified in C:/Temp/HaighR/RtmpFFcOA7
```

Figure 5. (*GUI-runModules*) GUI menu control to run available **PBStools** GUIs.

1.22 scaleVec

Scale a vector to span a target minimum and maximum (see **PBSmodelling** functions `scalePar` and `restorePar`.)

This function combines the utility of **PBSmodelling**'s `scalePar` and `restorePar` so that a user can rescale a vector to lie between any minimum and maximum value (including `c(0,1)`). The target minimum must be less than the target maximum, otherwise the function stops execution. The output is a numeric vector `V` re-scaled to range from `Tmin` and `Tmax`. Figure 6 shows the transformation of a random uniform vector between 100 and 1000 (x-axis) to lie between -3 and 5 (y-axis).

Example (scaleVec):

```
pbsfun=function() {
  V = runif(50,100,1000)
  T = scaleVec(V, Tmin=-3, Tmax=5)
  lmf = lm(T~V)
  par0 = par(no.readonly=TRUE); on.exit(par(par0))
  expandGraph(cex=1.2,mgp=c(1.8,0.5,0))
  plot(V,T,type="n",cex.lab=1.5)
  abline(lmf,col="dodgerblue",lwd=2)
  points(V,T,pch=21,cex=1.2,col="navyblue",bg="yellow")
  expr=paste(c("addLabel(0.1,0.85,expression(T == ",round(lmf$coeff[1],5),
    " + ",round(lmf$coeff[2],5),"* V~~~~R^2 == ",
    round(summary(lmf)$adj.r.squared,2),"),adj=0)"),collapse="")
  eval(parse(text=expr))
  invisible(expr) }
pbsfun()
```

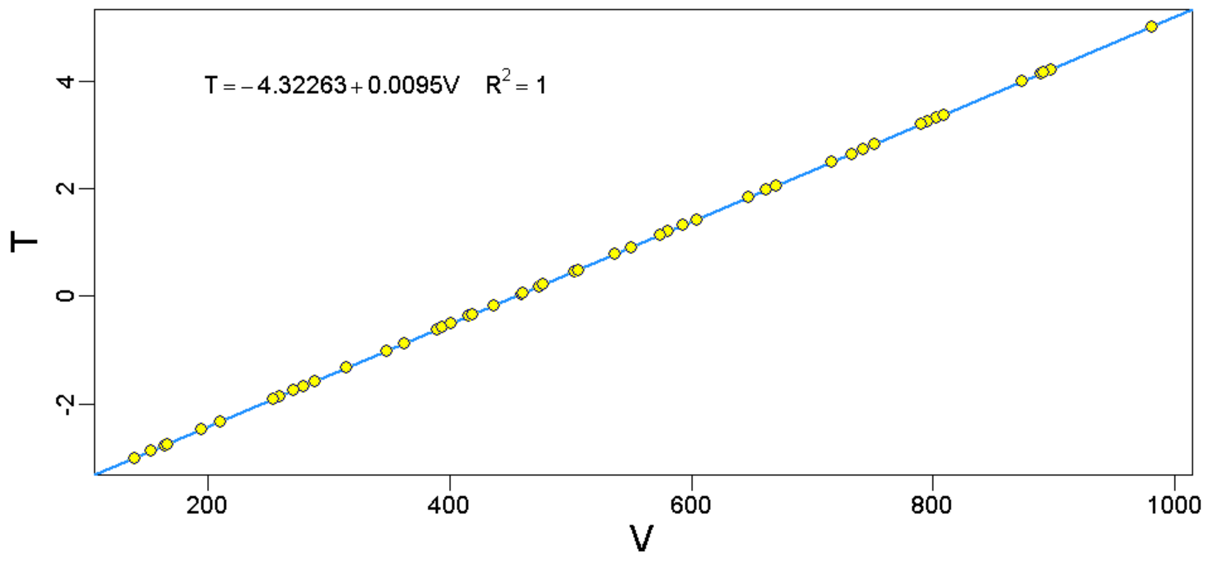


Figure 6. (*PBStools-scaleVec*) A sample of 50 points from the random uniform distribution betweeen 100 and 1000 rescaled to lie between -3 and 5.

1.23 showError, showMessage

Display a message on the current device. If `err==TRUE`, code execution stops. A wrapper function called `showError` is included for backwards compatibility.

1.24 spooler

Spool a named list or list of lists into a new field of an existing data frame where the names of the lists match existing fields in the target data frame. The elements of the list are values to match in the existing fields. The values placed in `newfld` are either matched values (if a simple list is supplied) or list names concatenated with matched values (if a list of lists is supplied).

This function is most useful when multiple fields exist that describe a similar concept. For example area might be any combination of the fields `major`, `minor`, `locality`, `srfa`, `srfs`, and `popa`. Obviously some of these overlap and the user has to be mindful of which combinations to collapse into one field. If multiple matches are available, the code gives preference to the first search field in the list. For example, `area` in lines 4 and 5 of the output below can either be `major-7+8` or `srfs-MR`, but preference is given to the former because the call supplied a list with `major` before `srfs`.

Example (spooler):

```
pbsfun=function(){
  data(pop.age)
  temp=pop.age[sample(1:nrow(pop.age),10),]
  spooler(list(major=list(3:4,7:8),srfs=list("GS","MI","MR")),"area",temp)
  print(temp)
  invisible() }
pbsfun()
```

Output (spooler):

```
      EID        X       Y    SID ttype stype       date year sex mat
 1  316045        NA      NA 184123     4     2 2001-06-28 2001   2   6
 2  354633 -129.3400 51.3192 227794     4     6 2003-05-24 2003   2  NA
 3   19177        NA      NA  50125     1     0 1990-06-25 1990   2   2
 4  106475        NA      NA  49478     1     2 1979-05-28 1979   1   7
 5    5662        NA      NA  49550     1     2 1980-06-25 1980   2   7
 6  157884        NA      NA  50102     1     2 1990-03-01 1990   2   4
 7  648040 -129.3825 51.2942 329663     2     5 1982-11-15 1982   1   1
 8  215189 -128.7558 51.3392 154697     2     0 1984-08-15 1984   2   2
 9  174570 -130.7317 51.7942 115700     4     2 1999-08-22 1999   2  NA
10  195574 -130.7400 51.7717 137896     4     2 2000-06-01 2000   1  NA
   age ameth len  wt gear major    catch srfa srfs      area
 1  50     3 460  NA    1     6 47174.09  5CD   MR   srfs-MR
 2   8     3 335  NA    1     6  6173.39  5AB   GS   srfs-GS
 3  12     3 410  NA    1     6  4436.00  5AB   GS   srfs-GS
 4  26     3 440  NA    1     7 13658.00  5CD   MR major-7+8
 5  31     3 470  NA    1     7 25141.00  5CD   MR major-7+8
 6  14     3 420  NA    1     6  8661.00  5CD   MR   srfs-MR
 7   6     3 280 305    1     6   284.00  5AB   GS   srfs-GS
 8   7     3 330  NA    1     5   319.00  5AB   GS   srfs-GS
 9  25     3 440  NA    1     6  4422.59  5CD   MR   srfs-MR
```

1.25 stdConc

Standardise a chemical concentration from various aavailable unit combinations to a common concentration and adjust for moisture. Note that the input object `dat` mixes data types (`numeric` and `character`). These types are maintained in a single-row data frame but a vector will change all numerics to character. The function `stdConc` automatically forces all inputs back to their intended mode. The output is one-row data frame with fields `c(stdAmt, unit)`, which represents the standardised concentration as a numeric value and a string unit fraction.

The available units for combinations in the numerator and denominator are:

`mcg`	microgram	`mg`	milligram	`g`	gram
`kg`	kilogram	`L`	litre	`pct`	percent (%)
`pdw`	% dry weight	`ppm`	parts per million	`ppt`	parts per thousand

1.26 toUpper

Capitalise the first letter of each word in a sentence or phrase. The function uses `strsplit` and `toupper` along with a whole mess of `sapply` calls. A vector of sentences or phrases can be passed to `x`.

1.27 ttget, ttcall, ttprint, ttput, tlisp

These functions are wrappers to the **PBSmodelling** accessor functions (`tget`, `tcall`, `tprint`, `tput`, and `lisp`) that get/print/list objects from or put objects into a temporary work environment, in this case `.PBStoolEnv`. Working objects include `PBStool`, which acts as a storage object for many of the functions. These accessor functions were developed as a response to the CRAN repository policy statement: “Packages should not modify the global environment (user’s workspace).”

1.28 wrapText

Wrap a long text string to a desired width, mark it with prefixes, and indent the lines following the first. Using the package **base** function `strwrap`, this function splits a long text string into a target width, indents lines following the first, and adds a prefix string `prefix[1]` to the first line and `prefix[2]` to all indented lines.

Example (wrapText):

```
pbsfun=function(){
  txt=wrapText(paste("USA state names:  ",
    paste(state.name,collapse=", "),sep=""),width=72,exdent=5)
  showMessage(txt,as.is=TRUE,adj=0,col="blue",cex=.9,x=.05)
  cat(txt,"\n"); invisible() }
pbsfun()
```

Output (wrapText):

```
> USA state names: Alabama, Alaska, Arizona, Arkansas, California,
+      Colorado, Connecticut, Delaware, Florida, Georgia, Hawaii,
+      Idaho, Illinois, Indiana, Iowa, Kansas, Kentucky, Louisiana,
+      Maine, Maryland, Massachusetts, Michigan, Minnesota,
+      Mississippi, Missouri, Montana, Nebraska, Nevada, New Hampshire,
+      New Jersey, New Mexico, New York, North Carolina, North Dakota,
+      Ohio, Oklahoma, Oregon, Pennsylvania, Rhode Island, South
+      Carolina, South Dakota, Tennessee, Texas, Utah, Vermont,
+      Virginia, Washington, West Virginia, Wisconsin, Wyoming
```

1.29 zapDupes

Zap (remove) records from a data frame using an codeindex to determine whether duplicated indexed records exist. The function orders the data frame based on the `index`, then removes records based on duplicated indices. There is no control over which of the duplicated records will be zapped. Pre-existing attributes other than `names` and `row.names` are transferred to the reduced object.

Example (zapDupes):

```
pbsfun=function(){
  data(ltmose07)
  cat(paste("'ltmose07' has",nrow(ltmose07),"rows\n"))
  zap=zapDupes(ltmose07,"PID")
  print(zap,quote=FALSE)
  cat(paste("After zapping using index 'PID', there are",nrow(zap),"rows\n"))
  invisible() }
pbsfun()
```

Output (zapDupes):

```
'ltmose07' has 55 rows
  PID POS         X        Y index
1   1   1 -125.1118 49.09033     1
2   2   1 -125.9806 49.47160     2
3   3   1 -125.9806 49.47160     3
4   4   1 -126.4951 49.86156     4
5   5   1 -127.7800 49.94667     5
6   6   1 -128.2633 50.28333     6
7   7   1 -127.9032 50.36946     7
After zapping using index 'PID', there are 7 rows
```

2 Biological functions

DFO:
A suitable dataset for the biological functions can be obtained by running the SQL query `gfb_bio.sql`:

```
getData("gfb_bio.sql", "GFBioSQL", strSpp="396", path=.getSpath())
```

Substitute your species code for `strSpp`. The package contains an example dataset for Pacific ocean perch (POP, code = “396”) *Sebastes alutus* in PMFC areas 5A, 5B, 5C, and 5D. To make it available in the global environment, type `data(pop.age)`.

2.1 calcLW

Calculate the length-weight relationship for a fish species. The functions spits results out to a comma-delimited text file (`.csv`). If the user specifies `plotit=TRUE` then image files are also generated. The code is currently set to spew `.png` files but with some code tweaking, other file types could be generated.

The parameterisation of the length-weight model is:

$$W_{s,i} = a_s (L_{s,i})^{b_s} \tag{1}$$

where

$W_{s,i}$ = the observation of weight (kg) of individual i of sex s,
$L_{s,i}$ = the observation of length (cm) of individual i of sex s,
a_s = the growth rate scalar for sex s , and
b_s = the growth rate exponent for sex s.

Example (calcLW):

```
pbsfun=function(){
  data(pop.age)
  calcLW(dat=pop.age,strSpp="396",ttype=list(c(2:3)),plotit=TRUE)
  invisible()}
pbsfun()
```

2.2 calcSG

Calculate the length or weight *vs.* age relationship by fitting the data using a versatile Schnute growth model (Schnute, 1981). Each column of the plot shows the fits for Males, Females, and M+F (Figure 8). If the data are plotted to the screen device, each row shows the fits by trip type, and each page shows the fits by area. If plots are sent to image files (`.png` or `.wmf`), each trip type and area combination goes to separate image files (handy for stacking in a `.doc` file).

The versatile Schnute fit requires initial parameter estimates and bounds. These are taken from the data object `parVec`. If the species code you are using is not contained in this file, the function uses the initial estimates for Pacific Ocean Perch (*Sebastes alutus*).

Example (calcSG):

```
pbsfun=function(){
  data(pop.age)
  calcSG(areas=list(srfa=list(c("5AB","5CD"))),ttype=list(2:3),tau=c(10,50),pix=TRUE)
  invisible() }
pbsfun()
```

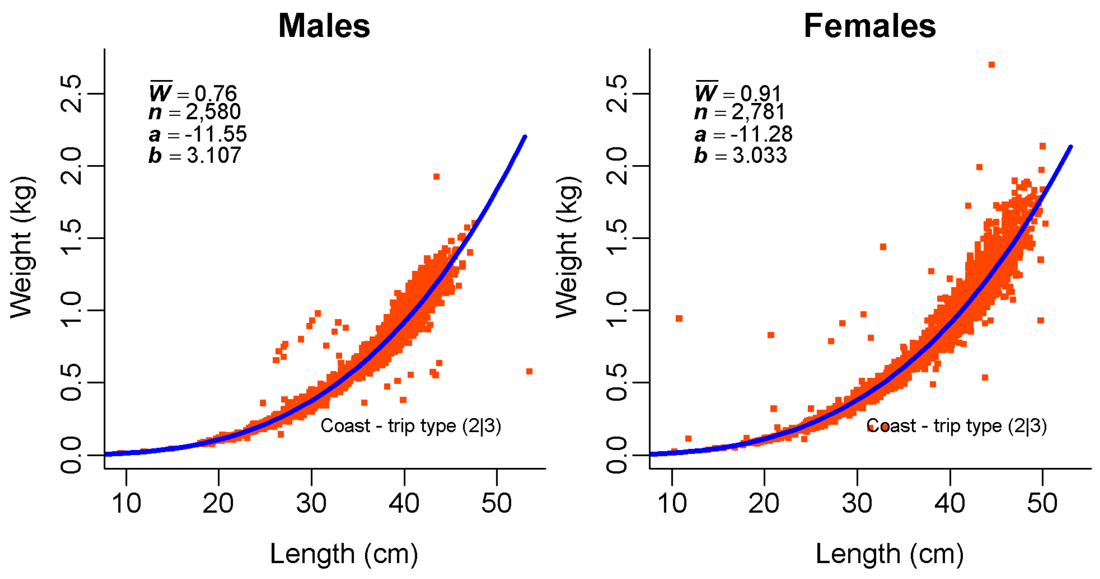


Figure 7. (*PBStools-calcLW*) Regression analyses showing the fitted model and survey length-weight pairs used to estimate a_s and b_s.

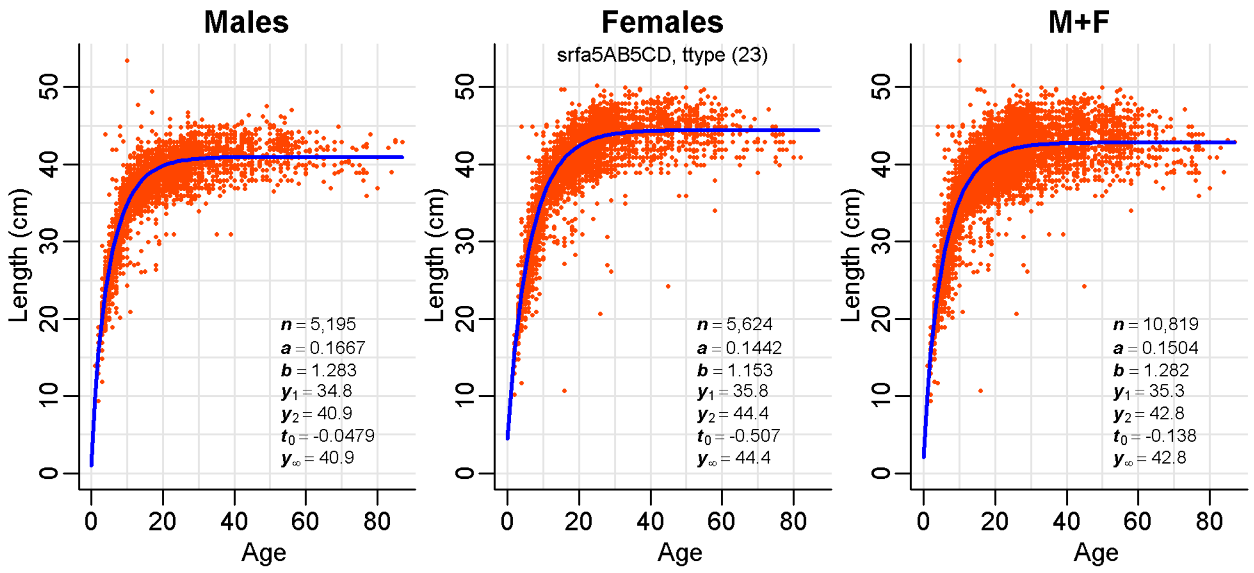


Figure 8. (*PBStools-calcSG*) Schnute versatile growth model fits to POP length and age survey data in slope rockfish assessment areas (`srfa`) 5AB and 5CD combined.

2.3 calcVB

Calculate the length (or weight) vs. age relationship by fitting these data to a von Bertalanffy growth model (Figure 9). This function provides a quick screen for combinations of management area and trip type. Figures generally comprise three columns where each column shows the model fits for Males, Females, and M+F. If the data are plotted to the screen device, each row shows the fits by trip type, and each page shows the fits by area. If plots are sent to image files `.png` or `.wmf`, each trip type and area combination goes to separate image files (handy for stacking in a `.doc` file). Note that areas can be combined if they are specified as lists within lists.

The von Bertalanffy fit requires initial parameter estimates and bounds. These are taken from the data object `parVec`. If a parameter vector is not available for a particular species, the user must either supply his own or use one available in the list
(e.g., `parVec[["vonB"]][["length"]][["396"]]`).

Example (calcVB):

```
pbsfun=function(){
  data(pop.age)
  calcVB(areas=list(srfa=list(c("5AB","5CD"))),ttype=list(2:3),pix=TRUE)
  invisible()}
pbsfun()
```

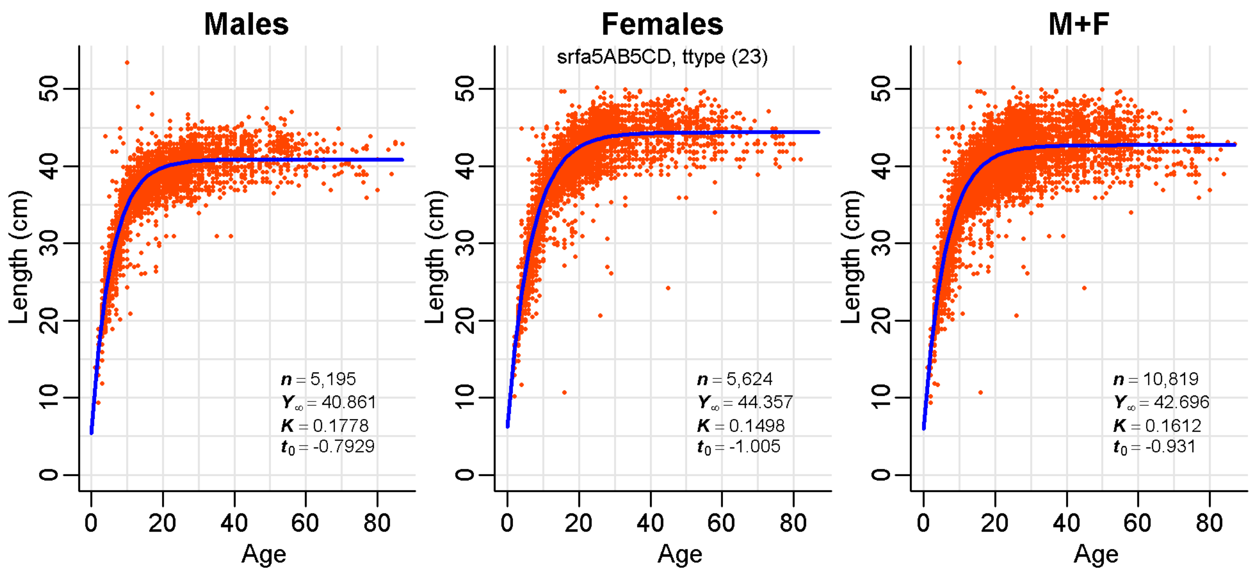


Figure 9. (*PBStools-calcVB*) von Bertalanffy growth model fits to POP length and age data in slope rockfish assessment areas (`srfa`) 5AB and 5CD combined.

2.4 compCsum

Compare cumulative relative frequency curves for specified factors such as age. Typically, factors are fields in the data object and comparisons are performed for each specified area and trip type. If the data are plotted to the screen device, each panel contains curves by combinations of trip type (rows) and factor (columns), and each page shows these combinations by area. If plots are sent to image files `.png` or `.wmf`, either each row is sent to separate files when `singles=TRUE` (handy for stacking in a `.doc` file) or each page is sent to a separate file when `pages=TRUE`.

In special cases `yfac="area"` or `yfac="trip"`, each panel depicts curves by year, and the factors (curves) are created as new fields from inputs `areas` or `ttype`. (The option `yfac="trip"` is not implemented at present.)

Example (compCsum):

```
pbsfun=function(){
  data(pop.age)
  compCsum(pop.age, areas=list(srfa=list(c("5AB","5CD"))),
    ttype=list(c(2,3)), years=1995:2008)
  invisible()}
pbsfun()
```

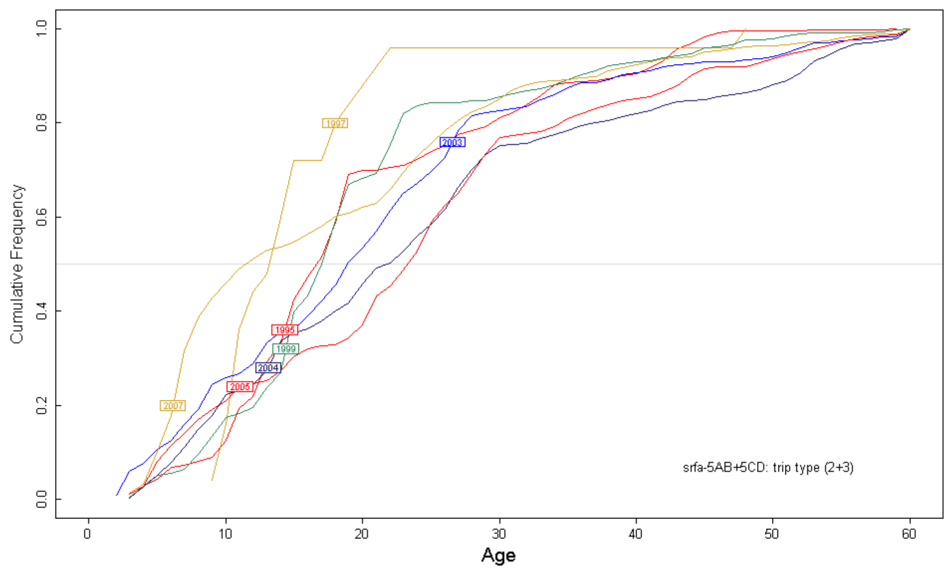


Figure 10. (*PBStools-compCsum*) Annual cumulative frequency curves for POP age proportions in slope rockfish assessment areas 5AB + 5CD using research and charter trip data.

2.5 estOgive

Estimate the cumulative distribution of the proportion of mature fish smoothed by binning ages. The cumulative distribution of mature fish is plotted and the point at 50% maturity is interpolated. The x-axis (age or length) can be binned to smooth out rough or sparse data.

Example (estOgive):

```
pbsfun=function(){
  data(pop.age)
  estOgive(obin=2, xlim=c(0,30), sex=1:2, method=c("empir","dblnorm"), cex=0.9)
  invisible()}
pbsfun()
```

2.6 genPa

Generate proportions-at-age using the catch curve composition of Schnute and Haigh (2007) (see Table 2, p.219 therein). Their model uses a design vector ϕ and a parameter vector θ; the latter contains the quantities to be estimated in a negative log likelihood using the complete parameter vector (θ, σ) for a logistic-normal model or (θ, n) for a Dirichlet model. Here we use θ primarily as a parameter vector, but include the design elements $b_1, ...b_m$ for expediency.

Additionally, for the selectivity component, we use the parameterisation of Edwards et al. (2012) where μ represents the age of full selectivity and $\sigma = \sqrt{\nu}$ represents the variance of the left limb of a double normal (see equation F.7 in the Pacific Ocean Perch assessment). This parameretisation replaces (T2.2) in Schnute and Haigh (2007), which uses α and β_k.

The output is a proportions-at-age vector of length `np`.
If `sim=TRUE`, the components S_a, β_a, R_a, and p_a are writtten to the global list object `PBStool`.

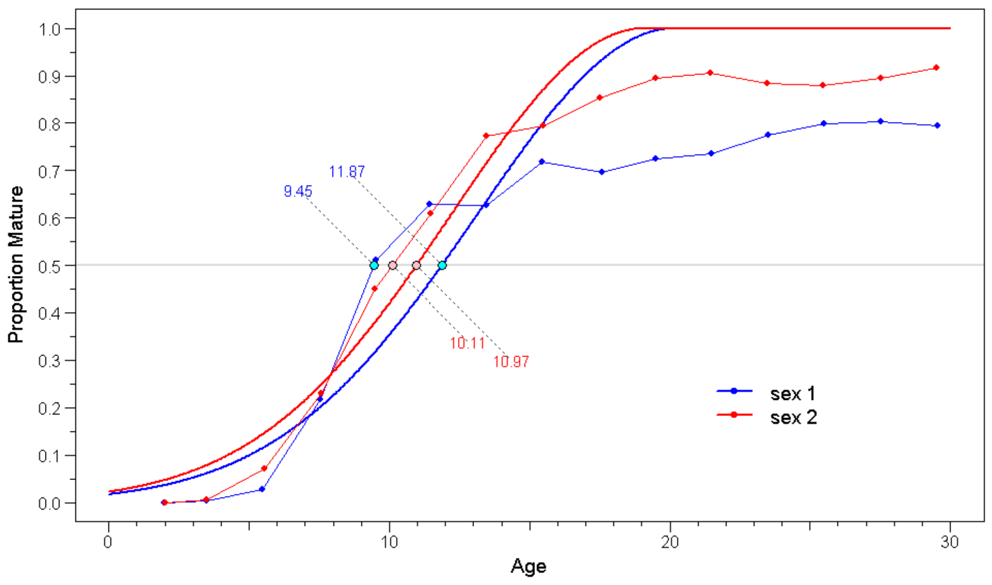


Figure 11. (*PBStools-estOgive*) Maturity ogives for POP estimated from plotting the cumulative sum of the proportion mature:total fish using binned ages. Diagonal dashed lines indicate ages at 50% maturity. Superimposed are model fits (smooth curves) to a double normal distribution.

Example (genPa):

```
pbsfun=function(){
  genPa(); unpackList(PBStool); unpackList(theta)
  par0 = par(no.readonly=TRUE); on.exit(par(par0))
  expandGraph(mfrow=c(2,2),mar=c(1,4,1,0.5),oma=c(3,0,0,0),mgp=c(2,.5,0))
  for (i in c("Sa","Ba","Ra","pa")) {
    expr = paste(
      c("plot(1:np,",i,",type=\"l\",lwd=2,col=\"blue\",xlab=\"\",ylab=\"\");",
      "mtext(expression(italic(",substring(i,1,1),
      "[a])),side=2,line=2,cex=1.5,las=1);",
      "points(1:np,",i,",pch=21,bg=\"aliceblue\",cex=1.2)"),collapse="")
    eval(parse(text=expr))
    if (i=="Ba") lines(rep(mu,2),c(0.99,par()$usr[3]),lty=2,col="grey30")
    if (i=="Ra") for (j in 1:length(rho))
      lines(rep(bh[j],2),c(rho[j]+0.98,par()$usr[3]),lty=2,col="grey30")
    if (i=="pa") lines(1:np,scaleVec(Sa,pa[np],max(pa)),lty=2,col="grey30") }
  mtext("     Age",outer=TRUE,side=1,line=1.5,cex=1.5)
  invisible() }
pbsfun()
```

2.7 histMetric

Plot a matrix of histograms or cumulative frequencies for a specified biology metric by year and trip type. The function displays the relative frequency of a specified metric like length by year (row) and trip type (column). Cells where the number of specimens <= `minN` are left blank. If only one trip type is specified or available, the cells display the annual histograms by row with an automatically calculated number of columns.

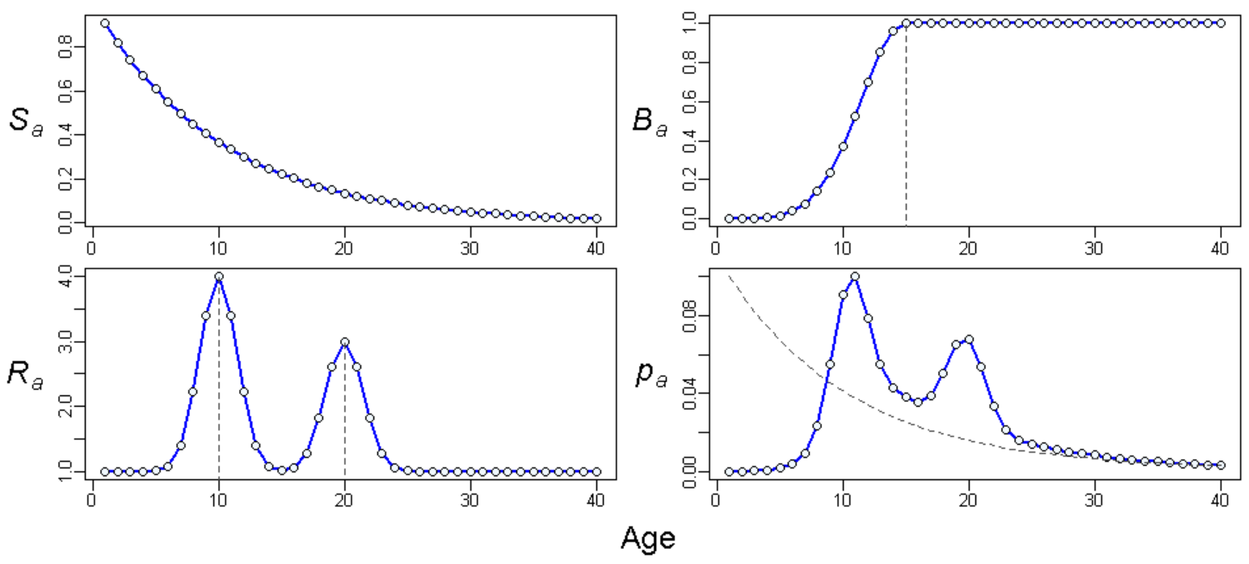


Figure 12. (*PBStools-genPa*) Plots of survival S_a, selectivity β_a, recruitment R_a, and predicted proportions p_a as functions of age a, based on the catch curve model of Schnute and Haigh (2007).

Example (histMetric):

```
pbsfun=function(){
  data(pop.age)
  histMetric(year=2003:2007, xlim=c(0,60), fill=TRUE, bg="moccasin", xint=2)
  invisible()}
pbsfun()
```

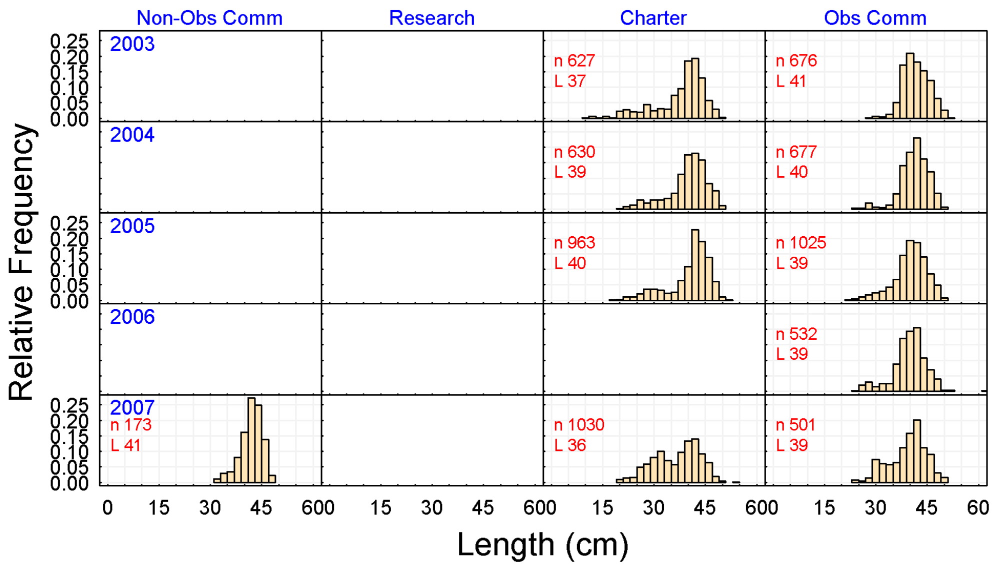


Figure 13. (*PBStools-histMetric*) Histograms of POP length samples from 2003 to 2007 for the four trip types. Values of n (number of specimens) and L (mean length in cm) appear in red.

2.8 histTail

Plot a histogram for a specified biology metric with an option to zoom in on the tail detail. The function displays the relative frequency of a specified metric like age. Its functionality is minimal, providing a quick histogram for a specified field in a data frame. The tail histogram appears as an inset on the right showing a zoom-in of the frequency of the upper tail data. The user can specify whether to plot the relative probability distribution (`prob=TRUE`) or a frequency distribution (`prob=FALSE`). The default provides the former for the main histogram and the latter for the tail zoom-in.

Example (histTail):

```
pbsfun=function(){
  data(pop.age)
  histTail(xfld="age", tailmin=60, prob=FALSE)
  invisible()}
pbsfun()
```

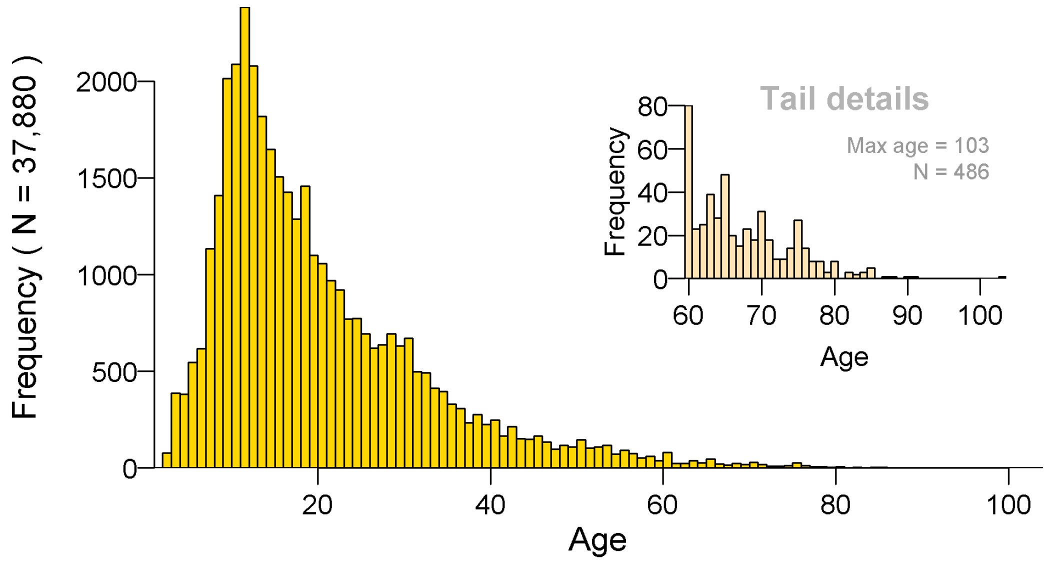


Figure 14. (*PBStools-histTail*) Histogram of POP ages with an inset that zooms in on the tail region of the distribution.

2.9 mapMaturity

Map each maturity code to show its relative occurrence by month. For each maturity code the function plots a row of monthly boxes colour-coded by the relative occurrence of the code in that month compared to the year.

Example (mapMaturity):

```
pbsfun=function(){
  data(pop.age)
  mapMaturity(brks=c(0,.01,.05,.25,.5,1), sex=2, hpage=3.5)
  invisible()}
pbsfun()
```

2.10 plotProp

Display proportions-at-age (or some other metric such as length or weight) vs. year using the **PBSmodelling** function `plotBubbles` controlled by an interactive GUI invoked by `plotProp()` (Figure 16). Data inputs appear on the left. The file name can be selected from the user's global environment `ls(1)` or from a list of binary files (`rda`) in the user's current working directory. If no field is specified for stratification the value is set to `year`, effectively meaning that the annual data are not stratified. If the check box is activated, the stratifying field can also serve to apply relative weightings to the observed proportions. Note that for most stratifying variables, weighting makes

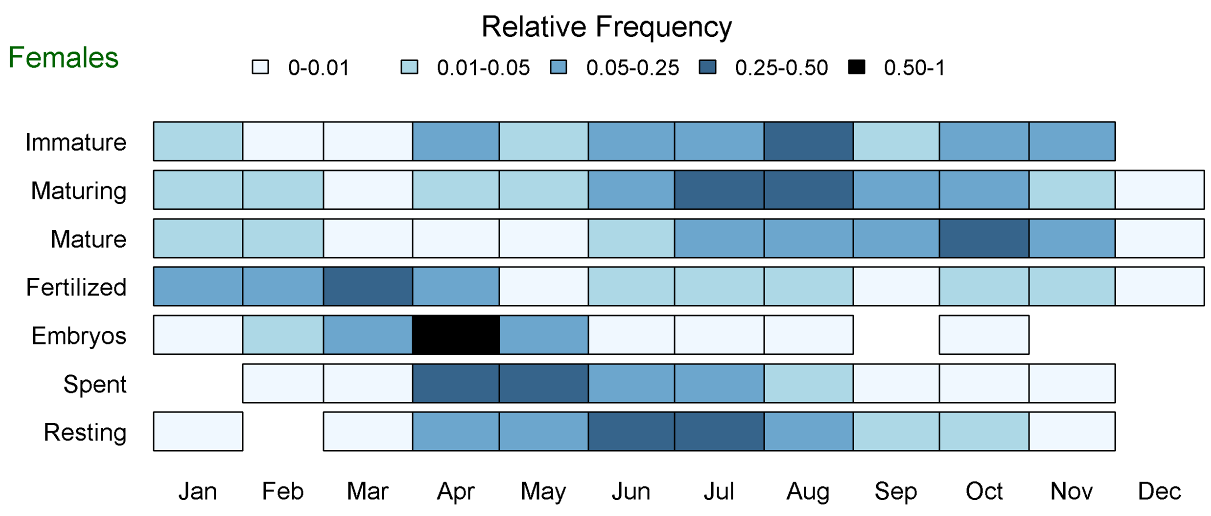
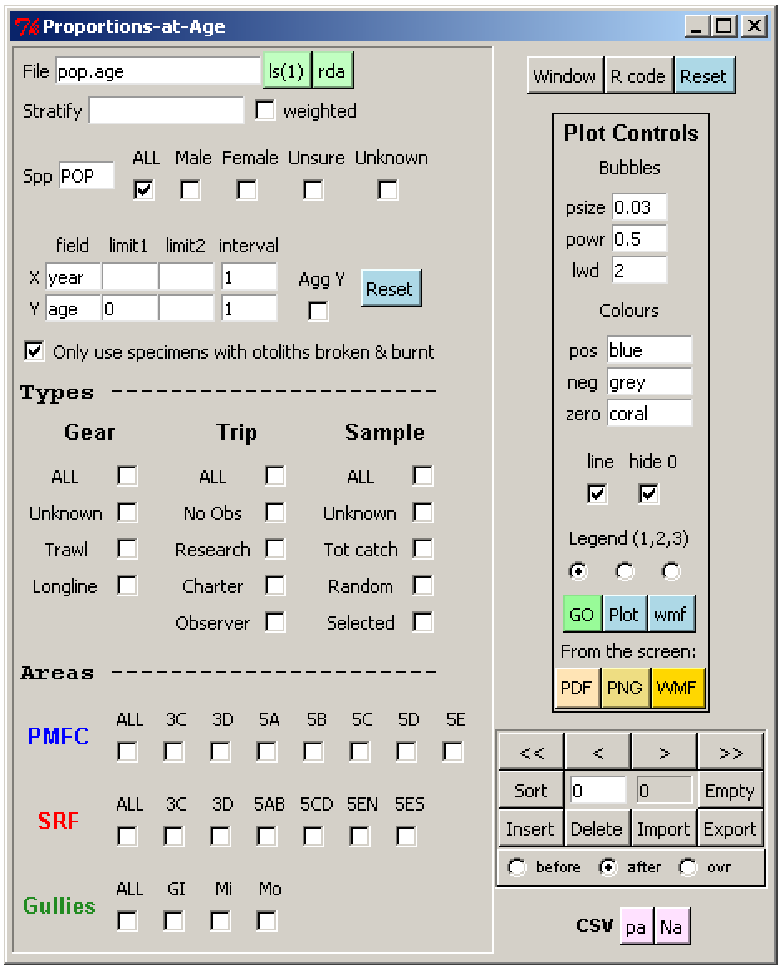


Figure 15. (*PBStools-mapMaturity*) Relative monthly frequency of observed maturity codes for female POP.

no sense. A field that would be meaningful is the `catch` of the species.



File pop.age

Stratify [ ] weighted

Spp POP — ALL [✓] Male [ ] Female [ ] Unsure [ ] Unknown [ ]

	field	limit1	limit2	interval
X	year			1
Y	age	0		1

Agg Y [ ]

[✓] Only use specimens with otoliths broken & burnt

Types ----------------------

Gear: ALL [ ] Unknown [ ] Trawl [ ] Longline [ ]

Trip: ALL [ ] No Obs [ ] Research [ ] Charter [ ] Observer [ ]

Sample: ALL [ ] Unknown [ ] Tot catch [ ] Random [ ] Selected [ ]

Areas ----------------------

PMFC: ALL [ ] 3C [ ] 3D [ ] 5A [ ] 5B [ ] 5C [ ] 5D [ ] 5E [ ]

SRF: ALL [ ] 3C [ ] 3D [ ] 5AB [ ] 5CD [ ] 5EN [ ] 5ES [ ]

Gullies: ALL [ ] GI [ ] Mi [ ] Mo [ ]

Plot Controls

Bubbles

psize 0.03

powr 0.5

lwd 2

Colours

pos blue

neg grey

zero coral

line [✓] hide 0 [✓]

Legend (1,2,3) [✓ 1] 2 3

From the screen:

0 0

before [✓ after] ovr

Figure 16. (*GUI-plotProp*) Proportions-at-age GUI to control the plotting of annual proportions as bubbles.

After choosing a species code, the user can select any combination of the four values of sex that might be available: male (`sex=1`), female (`sex=2`), observed but not sure (`sex=3`), and not observed (`sex=0`). The limits table of the two variables (usually year on the x-axis and age on the y-axis) can be left blank initially as the plotting routine will determine these automatically. Thereafter, the user will likely manipulate these to find a suitable plot. The Agg Y check box, if checked, will aggregate y-values at both ends of the specified y-limits. If the user wishes to aggregate only at the upper end, choose 0 or 1 for the lower limit. Most of the time, users will want to use only ages that have been determined from broken and burnt otoliths (`ameth=3`).

The GUI provides a menu of choices for certain fields in the data object: gear type (`gear`), trip type (`ttype`), sample type (`stype`), Pacific Marine Fisheries Commission areas (`major`), slope rockfish assessment areas (srfa), and slope rockfish gullies (`srfs`).

The right-hand side of the GUI provides some tweaks for output appearance and file format. The proportion bubbles can be manipulated through psize (maximum circle radius), powr (power transformation of proportions), and lwd (line thickness of bubbles). The bubble colours can be chosen separately for positive, negative, and zero values (even though proportions cannot be negative). Horizontal guide lines and zero-value proportions can be turned on or off.

The user has a choice of legend that accompanies the plot. The first choice is the most comprehensive in that it details how many records occur in various categories for seven of the important fields (along the bottom of the plot) as well the number of specimens used in each year (along the top). The second and third choices of legend simply display the number of specimens and samples per, respectively, along the bottom.

The action buttons GO and Plot both plot the results from the GUI choices (Figure 17); however, the former recalculates (qualifies, stratifies, weights, etc.) the input data object whereas the latter simply plots the massaged data object. The differences are supplied for users who may wish to change graphical parameters (e.g., bubble size) after the data have been qualified. This distinction must be noted as a user may change field selections like gear type or area and press the Plot button expecting the data object to reflect these changes. It is usually safest to use the GO button most of the time.

The blue wmf button redraws the plot on a metafile device directly instead of the graphical window. For dumping images from the screen to various file formats, push one of the yellow buttons marked PDF, PNG, or WMF. Below the history widget, there are two pink buttons pa and Na for dumping the proportions-at-age (or length or weight) and numbers-at-age, respectively, to `.csv` text files.

2.11 predictRER

A discriminant function for the accurate classification of Rougheye Rockfish (RER, *Sebastes aleutianus*) and Blackspotted Rockfish (BSR, *Sebastes melanostictus*). Exploring 35 morphometric and 9 meristic characters, Orr and Hawkins (2008) provide a discriminant function (using only 6 morphometrics L and 2 meristics N) that claims to correctly classify the two species 97.8% of the time.

The discriminant function:

$$\begin{aligned} D = {} & 101.557\lambda_1 + 52.453\lambda_2 + 51.920\lambda_3 - 38.604\lambda_4 - 22.601\lambda_5 - 10.203\lambda_6 \\ & + 0.294N_1 + 0.564N_2 - 10.445 \end{aligned} \tag{2}$$

where, $\lambda_n = 100\dfrac{L_n}{S}$, i.e., percent Standard Length

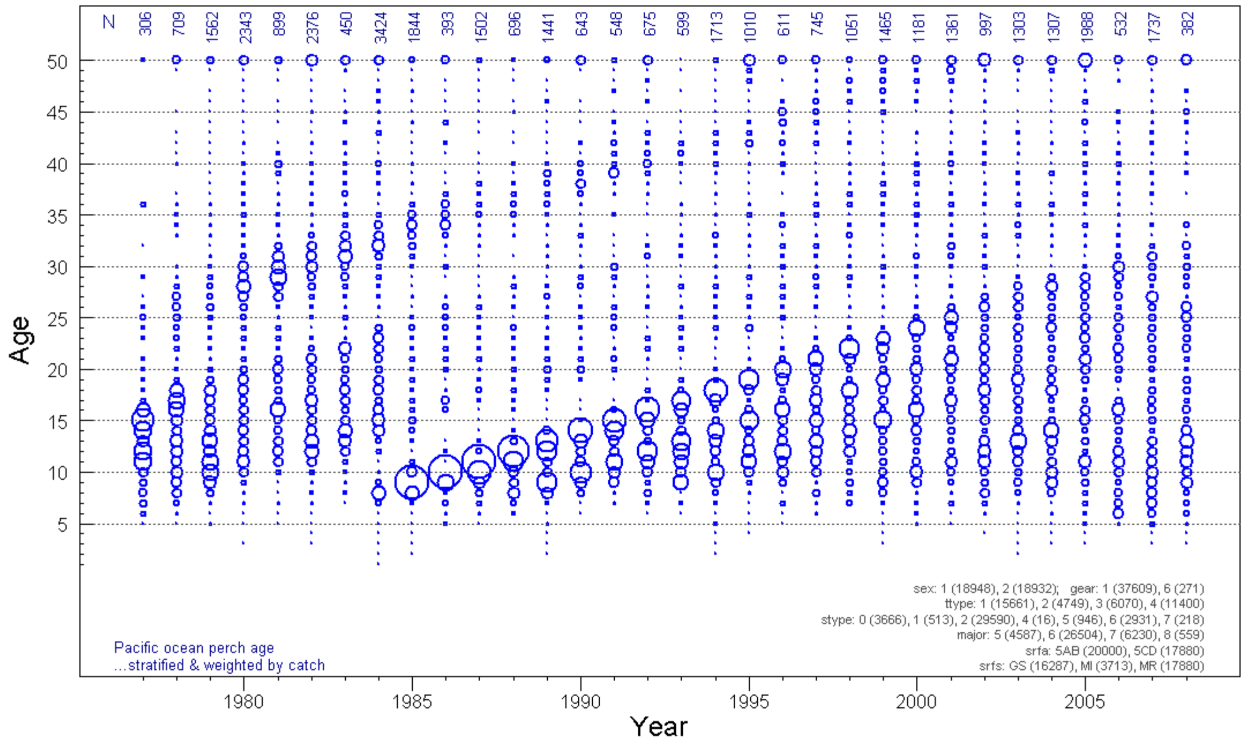


Figure 17. (*PBStools-plotProp*) Proportions-at-age for POP rendered as proportional bubbles.

S	=	standard fish length measured from tip of snout,
L_1	=	length of dorsal-fin spine 1,
L_2	=	snout length,
L_3	=	length of gill rakers,
L_4	=	length of pelvic-fin rays,
L_5	=	length of soft-dorsal-fin base,
L_6	=	preanal length,
N_1	=	number of gill rakers,
N_2	=	number of dorsal-fin rays.

Output is a numeric scalar D representing the calculated discriminant value.
When $D < 0$, the prediction is that the rockfish is *S. aleutianus*.
When $D > 0$, the prediction is that the rockfish is *S. melanostictus*.

Example (predictRER):

```
pbsfun = function(){
  test = simRER(1000)
  Dval = apply(test,1,function(x){predictRER(x[1],x[2:7],x[8:9])})
  cat(paste("simRER(1000)\nRER correctly predicted ",
    round((length(Dval[Dval<0])/length(Dval)*100),1),"% of the time\n\n",sep=""))
  test = simBSR(1000)
  Dval = apply(test,1,function(x){predictRER(x[1],x[2:7],x[8:9])})
  cat(paste("simBSR(1000)\nBSR correctly predicted ",
    round((length(Dval[Dval>0])/length(Dval)*100),1),"% of the time\n",sep=""))
  invisible() }
pbsfun()
```

Output (predictRER):

```
simRER(1000)
RER correctly predicted 98% of the time

simBSR(1000)
BSR correctly predicted 94.1% of the time
```

2.12 processBio

Process further the global data object `PBSdat` that results from the SQL query `gfb_bio.sql`. Currently, this function can add fields `srfa` (slope rockfish assessment areas), `srfs` (slope rockfish assessment subareas or gulllies), and `popa` (Pacific ocean perch areas identified by Schnute et al. (2001). These three areas do not exist as fields in GFBioSQL.

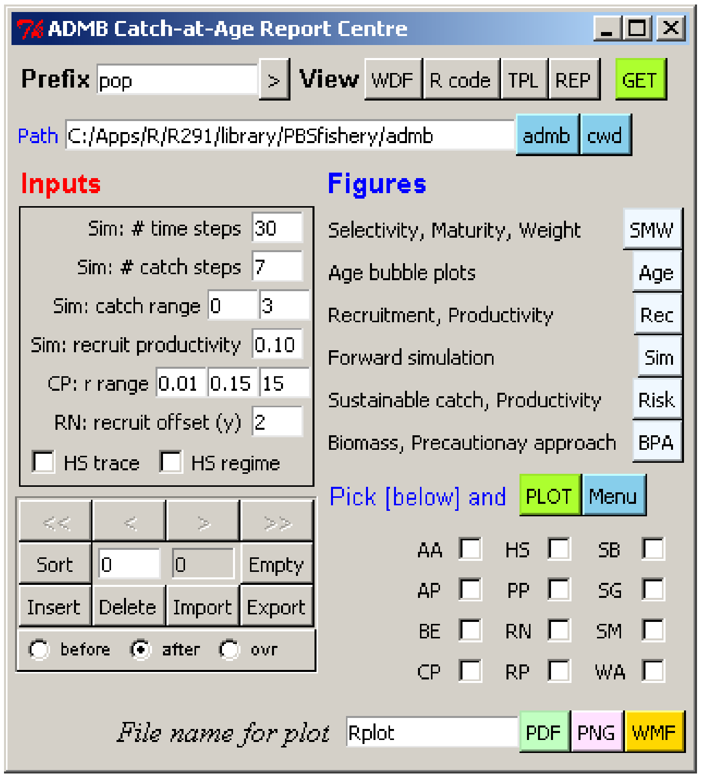
Example (processBio):

```
getData("gfb_bio.sql", "GFBioSQL", strSpp="396", path=.getSpath())
pop.bio = processBio(PBSdat)
```

2.13 reportCatchAge

This function is very specific in that it takes as its input the report file generated by Jon Schnute's ADMB template file (`.tpl`) for a Pacific ocean perch catch-at-age model. The Report Section of the template file (displayed below) saves a list object as a text file (`.rep`) using PBS formatting. Once the function reads this list object into R, a GUI (Figure 18) provides the user with various plotting options to view the results of the analysis.

```
REPORT_SECTION
Calc_SDpars();
report << "# POP Catch-at-Age Results" << endl;
report << "$nyr\n"     << "$$vector mode=\"numeric\" \n"    << NYear   << endl;
report << "$nag1\n"    << "$$vector mode=\"numeric\" \n"    << NAge1   << endl;
report << "$nage\n"    << "$$vector mode=\"numeric\" \n"    << NAge    << endl;
report << "$k\n"       << "$$vector mode=\"numeric\" \n"    << k       << endl;
report << "$pmin\n"    << "$$vector mode=\"numeric\" \n"    << PMin    << endl;
report << "$objfn\n"   << "$$vector mode=\"numeric\" \n"    << lf      << endl;
report << "$pnlty\n"   << "$$vector mode=\"numeric\" \n"    << penalty << endl;
report << "$fmor\n"    << "$$vector mode=\"numeric\" \n"    << F       << endl;
report << "$mmor\n"    << "$$vector mode=\"numeric\" \n"    << M       << endl;
report << "$rmean\n"   << "$$vector mode=\"numeric\" \n"    << R       << endl;
report << "$gma\n"     << "$$vector mode=\"numeric\" \n"    << gamma   << endl;
report << "$al\n"      << "$$vector mode=\"numeric\" \n"    << alpha   << endl;
report << "$bta\n"     << "$$vector mode=\"numeric\" \n"    << beta_a  << endl;
report << "$qq\n"      << "$$vector mode=\"numeric\" \n"    << q       << endl;
report << "$BPars\n"   << "$$vector mode=\"numeric\" \n"    << BPars   << endl;
report << "$rho\n"     << "$$vector mode=\"numeric\" \n"    << rho     << endl;
report << "$kap2\n"    << "$$vector mode=\"numeric\" \n"    << kap_2   << endl;
report << "$sig12\n"   << "$$vector mode=\"numeric\" \n"    << sig1_2  << endl;
report << "$tau12\n"   << "$$vector mode=\"numeric\" \n"    << tau1_2  << endl;
report << "$tau22\n"   << "$$vector mode=\"numeric\" \n"    << tau2_2  << endl;
report << "$Bt\n"      << "$$vector mode=\"numeric\" \n"    << B_t     << endl;
report << "$SSB\n"     << "$$vector mode=\"numeric\" \n"    << SSB     << endl;
report << "$RecProd\n" << "$$vector mode=\"numeric\" \n"    << RecProd << endl;
report << "$mcvec0\n"  << "$$vector mode=\"numeric\" \n"    << mcvec   << endl;
report << "$cdata\n"   << "$$data ncol=6 modes=\"integer numeric numeric numeric numeric numeric\"
         colnames=\"yr catch cf B1 B2 B3\" byrow=TRUE\n" << data_cpue  << endl;
report << "$uat\n"     << "$$matrix mode=\"numeric\" ncol=" << NYear << "\n" << u_at    << endl;
report << "$Nat\n"     << "$$matrix mode=\"numeric\" ncol=" << NYear << "\n" << Nat     << endl;
report << "$wgtat\n"   << "$$matrix mode=\"numeric\" ncol=" << NYear << "\n" << w_at    << endl;
report << "$pat\n"     << "$$matrix mode=\"numeric\" ncol=" << NYear << "\n" << p_at    << endl;
report << "$wgta\n"    << "$$vector mode=\"numeric\" \n"    << wgta    << endl;
report << "$mata\n"    << "$$vector mode=\"numeric\" \n"    << mata    << endl;
```



Prefix pop

Path C:/Apps/R/R291/library/PBSfishery/admb

Sim: # time steps 30

Sim: # catch steps 7

Sim: catch range 0 3

Sim: recruit productivity 0.10

CP: r range 0.01 0.15 15

RN: recruit offset (y) 2

☐ HS trace ☐ HS regime

0 0

before [✓ after] ovr

AA ☐ HS ☐ SB ☐

AP ☐ PP ☐ SG ☐

BE ☐ RN ☐ SM ☐

CP ☐ RP ☐ WA ☐

File name for plot Rplot

Figure 18. (*GUI-reportCatchAge*) ADMB Catch-at-Age Report Centre GUI to control the plotting of POP model results contained in the report file generated by code in the template file.

ADMB projects are controlled by a prefix (e.g., pop) that ADMB uses to accept input (`pop.tpl`, `pop.dat`, `pop.pin`) and spawn copious output (`pop.exe`, `pop.cpp`, `pop.o`, `pop.std`, `pop.rep`, `...`). The R package **PBSadmb** (available on CRAN) provides an easy to use GUI system for running ADMB directly from R.

Aside from reading in the report file (e.g., `pop.rep`), the GUI offers a few user-controlled tweaks (see `reportCatchAge` help file for details) that affect the simulation and recruitment plots. The light blue buttons on the right recreate six pairs of plots similar to those that appear in the 2001 POP assessment (Schnute et al., 2001). Alternatively, the user can pick and choose combinations of plots from the menu check boxes (lower right), then press the green PLOT button. The plot codes appear in the help file; however, we reproduce them below for convenience. Figure 19 results from choosing the BE check box and plotting.

2.14 requestAges

This function queries the DFO database GFBioSQL for available otoliths to age. It chooses random otoliths based on trips' commercial catch weights of the specified species (e.g., "396" for POP). The function invisibly returns the otolith data with supplementary quarterly commercial catch. Also, it writes this otolith data to binary `.rda` and ascii `.csv` files. The random samples formatted to simulate an otolith tray (for ageing personnel) is written to a file called `oto123-C(YYYY)-areas(0+0+0)-sex(1+2)-N999.csv` (where `123` = species code, `C` = Commercial or `S` = Survey, `YYYY` = year, `0` = PMFC area code, and `999` = number of otoliths).

Table 1. Menu of figure codes to select and PLOT.

Code	Plot type from report file
AA	Ages actual
AP	Ages predicted
BE	Biomass estimates
CP	Catch (sustainable) vs. productivity
HS	Harvest strategy (precautionary approach)
PP	Probability of achieving productivity
RN	Recruitment numbers
RP	Recruitment productivity
SB	Simulation: biomass at fixed catches
SG	Simulation: growth rate vs. catch
SM	Selectivity and maturity vs. age
WA	Weight vs. age

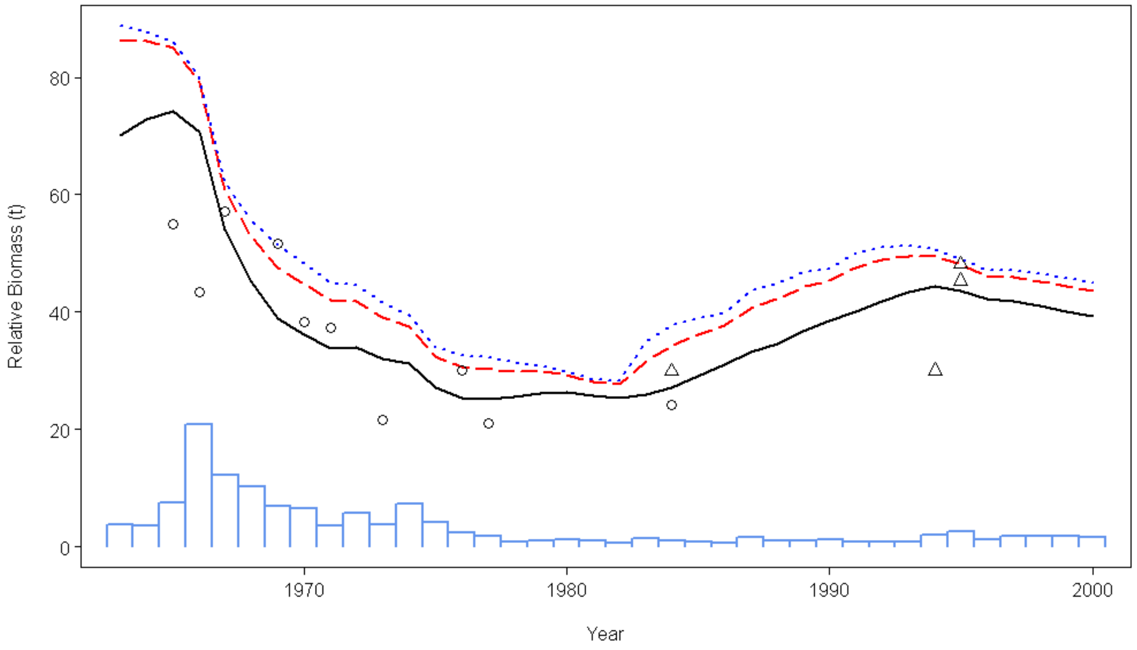


Figure 19. (*PBStools-reportCatchAge*) Annual estimates of POP biomass: available to the fishery (solid back line), mature (dashed red line), and total (dotted blue line) for the Goose Island Gully population in Queen Charlotte Sound. Circles indicate research survey estimates, scaled to available biomass by q_1. Plus symbols indicate charter survey estimates, scaled with q_2. The blue bar plot represents annual catches.

2.15 simBSR, simRER

Simulate biological data that best characterizes Blackspotted Rockfish (BSR, *Sebastes melanostictus*) and Rougheye Rockfish (RER, *S. aleutianus*), using parameter estimates from Table 2 in Orr and Hawkins (2008). This table provides a range of BSR standard lengths (mm) and distributions for morphometrics and meristics. The functions `simBSR` and `simRER` sample from a random uniform distribution for S and from random normal distributions for model inputs of λ and N – tables below give (μ, σ).

Balckspotted Rockfish:

S	=	standard fish length measured from tip of snout	95.5–539
λ_1	=	length of dorsal-fin spine 1	(7.8, 0.7)
λ_2	=	snout length	(8.0, 0.6)
λ_3	=	length of gill rakers	(5.6, 0.6)
λ_4	=	length of pelvic-fin rays	(21.4, 1.2)
λ_5	=	length of soft-dorsal-fin base	(21.4, 1.5)
λ_6	=	preanal length	(70.2, 2.6)
N_1	=	number of gill rakers	(33.0, 1.2)
N_2	=	number of dorsal-fin rays	(13.7, 0.5)

Rougheye Rockfish:

S	=	standard fish length measured from tip of snout	63.4–555.2
λ_1	=	length of dorsal-fin spine 1	(5.8, 0.6)
λ_2	=	snout length	(7.5, 0.7)
λ_3	=	length of gill rakers	(4.9, 0.6)
λ_4	=	length of pelvic-fin rays	(22.1, 1.1)
λ_5	=	length of soft-dorsal-fin base	(22.8, 1.2)
λ_6	=	preanal length	(71.8, 2.3)
N_1	=	number of gill rakers	(31.2, 1.0)
N_2	=	number of dorsal-fin rays	(13.7, 0.5)

where, $\lambda_n = 100\frac{L_n}{S}$, i.e., percent Standard Length

The function returns a numeric matrix with dimensions `c(Nfish,9)` where columns are labelled `c('S', 'L1', 'L2', 'L3', 'L4', 'L5', 'L6', 'N1', 'N2')`. The values are described above, but generally, `S` = standard length of the fish (mm), `L` = six diagnostic length measurements (mm), and `N` = numbers of gill rakers and dorsal fin rays.

2.16 sumBioTabs

This function provides a simple utility to summarise the number of samples and specimens by factor pairs in a biological data object. Aside from the factor strings chosen, the object must contain the field `SID` (sample ID) if a sample summary is indicated. The latter also require the package **reshape**. The summary is sent to a `CSV` file.

2.17 weightBio

This function executes a method for representing commercial age structure for a given species through weighting age frequencies n_a or proportions n'_a by catch. For simplicity, we assume that age frequencies n_a are weighted, unless otherwise specified. The weighting occurs at two levels: quarterly q and annually y. Notation is summarized in Table 2.

For each quarter q we weight trip frequencies n_{at} by trip catch of a given species. (We use trip as the sample unit, though sometimes one trip can contain multiple samples. This means we sometimes merge samples into the sample unit.) Within any quarter q and year y there is a set of trip catches S_{tay} that can be transformed into a set of catch proportions:

$$S'_{tay} = \frac{S_{tay}}{\sum_t S_{tay}} \ . \tag{3}$$

The age frequencies are weighted using S'_{tay} to derive weighted age frequencies by quarter:

$$m_{aqy} = \sum_t n_{atqy} S'_{tay} \ . \tag{4}$$

Table 2. Notation for weighted commercial age equations for a given species.

Symbol	Description
	Indices
a	age class (1 to 60, where 60 is an accumulator age-class)
t	trips as sample units (may contain more than one sample)
q	quarters (1 to 4, defined mechanically to occur every 91.5 days) within years
y	years (1977 to present)
	Data
n_{atqy}	frequency at age a for trip t in quarter q of year y
S_{tqy}	trip catch of a given species for trip t in quarter q of year y
S'_{tqy}	S_{tqy} as a proportion of total catch $S_{qy} = \sum_t S_{tqy}$
m_{aqy}	weighted age frequencies at age a in quarter q of year y
C_{qy}	commercial catch in quarter q of year y
C'_{qy}	C_{qy} as a proportion of total catch $C_y = \sum_q C_{qy}$
w_{aq}	weighted age frequencies at age a in year y
p_{aq}	weighted proportions at age a in year y

If this transformation is applied to frequencies (as opposed to proportions), it reduces them from the original, and so we rescale m_{aqy}:

$$m_{aqy} = m_{aqy} \frac{\sum_a n_{aqy}}{\sum_a m_{aqy}} \ . \tag{5}$$

to retain the original number of observations. This step is not strictly necessary because at the end of the two-step weighting, we standardize the weighted frequencies to represent proportions-at-age.

At the second level of stratification by year y, we calculate the annual proportion of quarterly commercial

$$C'_{qy} = \frac{C_{qy}}{\sum_q C_{qy}} \tag{6}$$

to weight m_{aqy} and derive weighted age frequencies by year:

$$w_{ay} = \sum_q m_{aqy} C'_{qy} \ . \tag{7}$$

Again, if this transformation is applied to frequencies (as opposed to proportions), it reduces them from the original, and so we rescale w_{ay}:

$$w_{ay} = w_{ay} \frac{\sum_a m_{ay}}{\sum_a w_{ay}} \ . \tag{8}$$

to retain the original number of observations.

Finally, we standardize the weighted frequencies to represent proportions-at-age:

$$p_{ay} = \frac{w_{ay}}{\sum_a w_{ay}} \ . \tag{9}$$

If initially we had used proportions n'_{atqy} instead of frequencies n_{atqy} , the final standardization would not be necessary; however, its application does not affect the outcome.

It sometimes matters if we choose to weight frequencies rather than proportions: the numeric outcome can be very different, especially if the input samples comprise few observations. Theoretically, weighting frequencies emphasizes our belief in individual observations at specific ages while weighting proportions emphasizes our belief in sampled age distributions. Neither method yields inherently better results; however, if your original sampling methodology favours sampling few fish from many tows rather than sampling many fish from few tows, then weighting frequencies probably makes more sense than weighting proportions.

3 Fishery functions

Most of the fishery functions use commercial catch and effort data. Some functions tap into the DFO databases (SQL server and Oracle) directly, and therefore cannot be used by those not logged onto the DFO network. Other functions can use local data files. An example dataset, `testdatC` included in the **PBSdata** package, provides a random subset of commercial observer trawl data in DFO's PacHarvest database.

3.1 calcRatio

Calculate the annual ratios of one catch to another (or one field to another) for various PMFC major areas and coastwide. A suitable dataset can be obtained by DFO personnel by running the SQL query `pht_catORF.sql`, where ORF refers to rockfish other than POP, for a specified species code. The query returns a data frame with fields of `catch` and `discard` for the target, as well as `POP` and `ORF` catch. Total rockfish catch (`TRF`) can be calculated by summing `POP` and `ORF`. The mean ratios are invisibly returned in a matrix where rows are fishing years and columns are PMFC area codes. Figure 13 provides visual results of yellowmouth rockfish `catch` to `ORF` (all rockfish catch other than POP). Discard rates could be represented by setting the ratio to `discard` over `catch`.

Example (calcRatio):

```
pbsfun=function(dfo=FALSE){
  if (dfo) {
    getData("pht_tcatORF.sql",strSpp="440",path=.getSpath())
    ymr.pop.orf = PBSdat
    pmean=calcRatio(dat=ymr.pop.orf, nfld="landed", dfld="ORF",
      major=3:9, plot=TRUE, ylim=c(0,.35))
    print(pmean) }
    else cat("If you are logged onto the DFO network,\nrun 'pbsfun(dfo=TRUE)'\n.")
  invisible() }
pbsfun(TRUE)
```

3.2 dumpMod, dumpRat

Dump catch from modern databases that has been collected and organised during a catch reconstruction (see `buildCatch`) into CSV tables. The function `dumpCat` flattens the input array `dat` from four or five dimensions to two dimensions and sends the result to a comma-delimited text file (`.csv`).

Dump catch ratios calculated during a catch reconstruction into CSV tables. The reconstruction procedure stores ratio arrays in a list object `PBStool123`, where `123` is the species Hart code. The function `dumpRat` extracts the arrays and sends them to a comma-delimited text file (`.csv`).

3.3 formatCatch

Format a table of numeric values so that each cell displays N significant figures in character format. The algorithm relies on string manipulation to achieve a result that cannot be obtained by the base package's `format` function alone. The formatted table can then be passed to **xtable**'s function `xtable`, which produces output that can be passed to `print.xtable` for use in LaTeX.

3.4 getCatch

Extract catch records for a species from various databases and combine them into one catch file. This function provides a quick method of extracting contiguous datasets and combining them into one data object. The primary use for this dataset is to create the catch input for the function `weightBio`. Currently, `getCatch` only functions for the trawl fishery as the hook and line databases are messy and not contiguous. For a comprehensive catch reconstruction, see

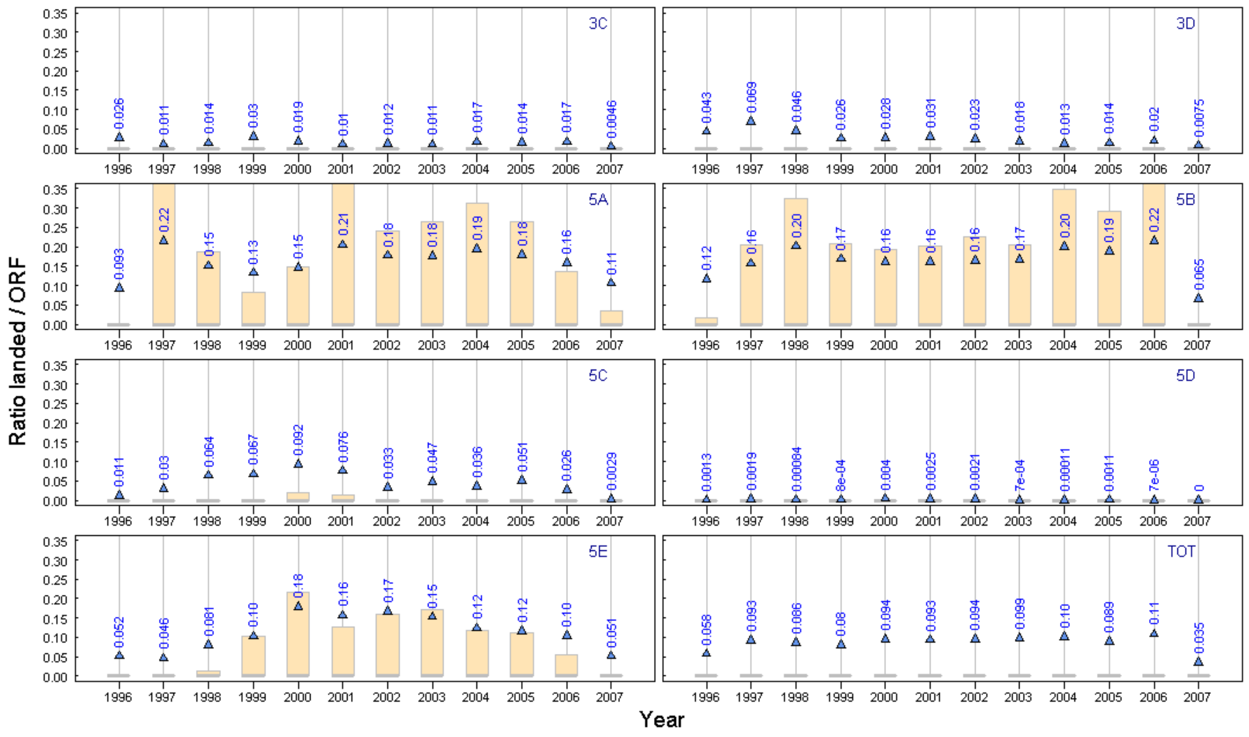


Figure 20. (*PBStools-calcRatio*) Distribution of proportions of yellowmouth rockfish landings to landings of all rockfish other than Pacific ocean perch in the trawl fishery by fishing year (Apr to Mar). Vertical grey bars indicate the extent of the data. Horizontal grey bars within the boxplots indicate median ratios (all zero here). Blue triangles indicate mean ratios (also presented as numbers). Panels summarize ratios by PMFC areas; the final panel summarizes ratios on a coastwide basis.

`buildCatch`.

The function returns a data object of survey catch records `Scat123.wB` and commercial catch records `Ccat123.wB` resulting from the merge of various database query results. The results are also written to system binary files `Scat123.wB.rda`, `Ccat123.wB.rda`, where `123` specifies the species code and `wB` specifies creation for use in the function `weightBio`.

3.5 glimmer

This function performs a standardised GLM analysis and plots the results. It calculates parameter coefficients using the **stats** package functions `lm` and `aov`. (The `aov` results are not currently used for anything.) See Section 7 of Schnute et al. (2004*b*) for a detailed discussion on ‘Standardized CPUE Analyses’ and the additive lognormal model:

$$\log y_{hijklmn} = \mu_h + \alpha_i + \beta_j + \gamma_k + \delta_l + \phi_m + \sigma\epsilon_{hijklmn} \qquad (10)$$

The example below uses the example dataset `testdatC` (from **PBSdata**), which provides a random subset of commercial trawl data. The GLM model (Figure 21) suggests an increasing trend in CPUE for these particular data, though index values appear to be imprecise. Factor effects are fairly typical for a shelf/slope rockfish species, especially the dome-shaped curve for depth. We don’t anticipate a large vessel effect in the subset because the vessel codes assigned to each record have not only been masked using simple integers, but have also been randomly re-assigned (without replacement) to the tow set. In real life, certain vessels are considerably better at capturing fish than others and can produce a strong factor effect. The latitudinal effect highlights the predominance of POP in the three gullies (Goose Island, Mitchell's, Moresby) of Queen Charlotte Sound.

Example (glimmer):

```
pbsfun=function(){
  data(testdatC);  x=testdatC
  glmdat=data.frame(
    year=as.numeric(substring(x$date,1,4)),
    month=as.numeric(substring(x$date,6,7)),
    depth=x$fdep,latitude=x$Y,vessel=x$cfv,
    effort=x$eff,catch=x$POP)
  attr(glmdat,"spp")="396"
  glimmer(glmdat,Vpro=.025,dlim=c(100,500,50))
  invisible() }
pbsfun()
```

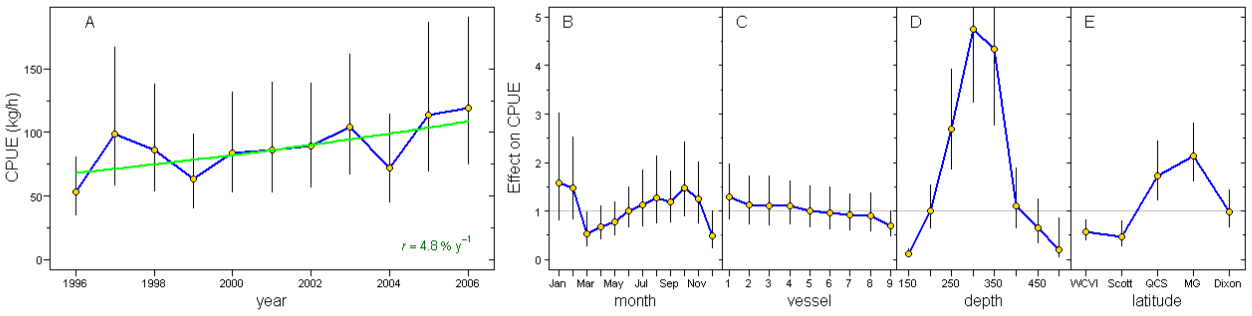


Figure 21. (*PBStools-glimmer*) Commercial POP CPUE U (kg/h) estimated from an additive lognormal model as $\log_2 U$: (A) year as $2^{\mu+\alpha_i}$, (B) month as 2^{β_j}, (C) vessel as 2^{γ_k}, (D) depth zone (50m intervals) as 2^{δ_l}, and (E) latitudinal groups as 2^{ϕ_m}. Vertical error bars indicate 95% confidence limits. Green line in (A) shows the back-transformed linear fit through α_i; implies an annual gain of 4.8%/y.

3.6 makeCATtables

This function creates tables of commercial catch by management fishing year. Various fisheries can be specified. The code scrolls through the specified fisheries, issuing the appropriate SQL query for each catch summary. The results are sent to a text file called `cattab-XXX.csv`, where `XXX` = the specified `strSpp` (species code). Table 3 lists the available SQL queries for this function.

Table 3. Codes for the available fisheries.

code	description	SQL query
1	GF Catch trawl tows (sales receipts)	gfc_catch_fyear.sql
2	PacHarvest trawl tows (observer or fisher logs)	pht_catch_fyear.sql
3	PacHarvHL Zn hook & line validation records	phhl_vcatch_fyear.sql
4	PacHarvHL Zn hook & line fisherlog records	phhl_fcatch_fyear.sql
5	PacHarvHL Schedule II hook & line validation records	phhl_vcatch_fyear.sql
6	PacHarvHL Schedule II hook & line fisherlog records	phhl_fcatch_fyear.sql
7	PacHarvHL Halibut longline validation records	phhl_hcatch_fyear.sql

Example (makeCATtables):

```
makeCATtables(strSpp="396",comm=2)
```

Sends the output displayed in Table 4 to `cattab-396.csv`:

Table 4. POP catch (t) from running `makeCATtables`.

fyear	3C	3D	4B	5A	5B	5C	5D	5E	UNK
1996	118	497	0	407	4180	561	32	694	0
97	54	47	NA	36	521	62	0	372	NA
1997	404	214	0	876	3323	372	181	647	NA
1998	299	164	0	916	3098	477	226	766	NA
1999	309	246	NA	981	3275	610	136	666	NA
2000	293	233	NA	578	3082	369	160	1257	NA
2001	293	207	0	660	3084	329	173	1133	NA
2002	300	264	0	794	3136	320	171	1125	6
2003	304	218	0	762	3551	245	108	920	NA
2004	317	220	0	752	3722	130	72	904	NA
2005	285	219	0	856	2894	139	185	761	NA
2006	275	207	0	503	3101	81	44	853	NA
UNK	NA	NA	NA	NA	NA	NA	NA	NA	508

3.7 plotCatch

This function displays the annual catch history contained in the specified file as stacked bars (Figure 22). Each bar shows the cumulative catch from specified fields. Decadal mean catch is displayed in horizontal bands. If the user wishes to change the bar colours by passing in a vector `col` to the function, s/he must also pass in a similar vector `fill` for the legend.

Example (plotCatch):

```
plotCatch(dat=ymr.rem)
```

3.8 plotConcur

Create a barplot of the top 20 concurrent species from tows capturing the target species `strSpp` between a minimum and maximum depth. This function uses the SQL code `pht_concurrent.sql` to query the observer trawl database PacHarvest. A variant of this (`phhl_concurent.sql`) taps into the hook and line database PacHarvHL. This function can only be used by personnel logged onto the DFO network.

The default settings for this function create a one-panel plot (Figure 23). If a multiple figure plot is desired, say `par(mfrow=c(2,2))`, then set the argument `reset.mf=FALSE` in calls to `plotConcur` after setting the `mfrow` option in `par`.

Example (plotConcur):

```
plotConcur(strSpp="410", mindep=150, maxdep=435) # DFO only
```

3.9 plotFOScatch

This function queries the DFO FOS[4] database for catches of the specified species, and plots monthly catches as bars stacked by PMFC area. The SQL query `fos_catch.sql` (DFO personnel only) obtains catch from all fisheries, and the user chooses which fishery to display. Barplot categories are combinations of year and month, and catch is colour-coded by PMFC area (blues for 3C-3D, yellow-orange-red for 5A-5B-5C, and greens for 5D-5E) to provide a visual feel

[4] Fishery Operations System

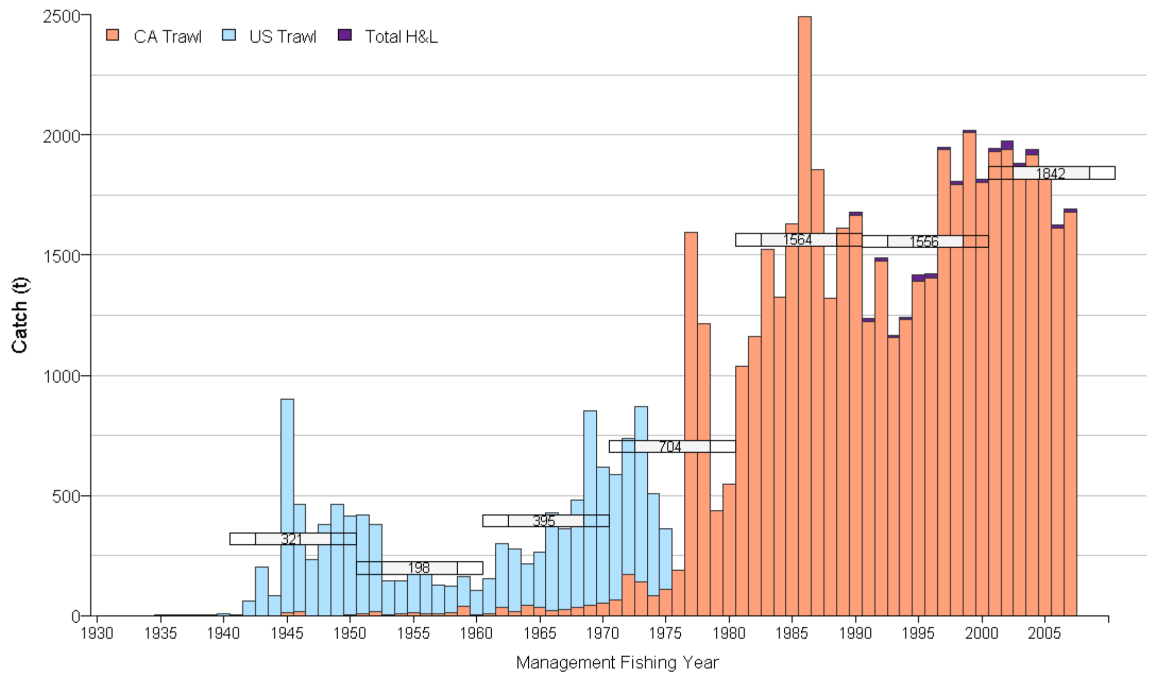


Figure 22. (*PBStools-plotCatch*) Catch history of Yellowmouth Rockfish (*Sebastes reedi*) by US and Canadian fleets along the BC coast. Mean annual catches by decade are displayed in horizontal boxes.

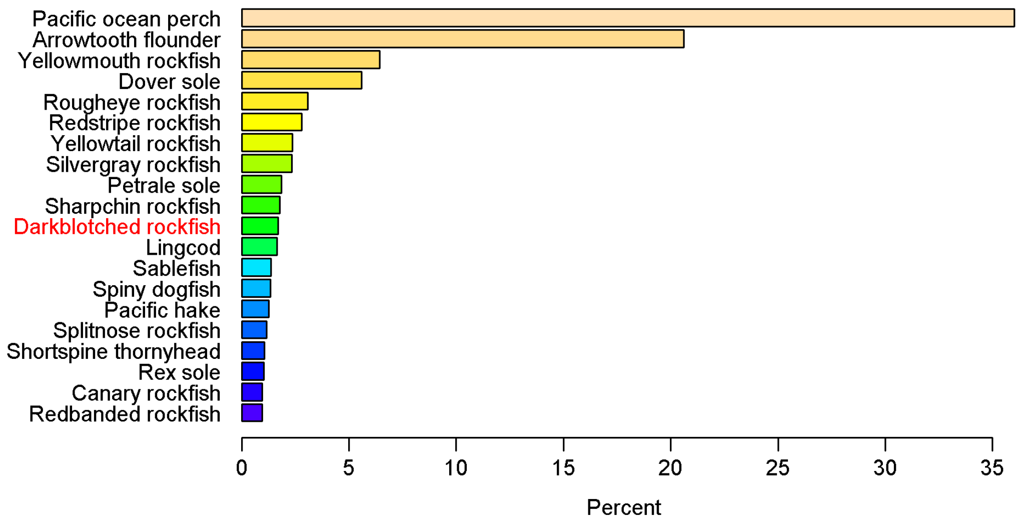


Figure 23. (*PBStools-plotConcur*) Concurrence of species in trawl tows (1996-2007) capturing Darkblotched Rockfish (*Sebastes crameri*) at depths 150–435 m. Abundance is expressed as a percent of total catch weight. The target species is highlighted in red.

for the part of the coast (WCVI[5], QSC[6], or WCHG[7]) that provides the predominant catch (Figure 24). The total catch is reported at the end of each fishing year.

[5]west coast of Vancouver Island

[6]Queen Charlotte Sound or central BC coast

[7]west coast Haida Gwaii (offshore west, Hecate Strait to the east, & Dixon Entrance to the north)

In addition to the plot, catch summaries by fishery, year, and PMFC area are dumped to a file called `catFOSfyr.csv`. Also, internally-derived objects of interest (such as catch-by-month and catch-by-year matrices) are saved in the list object `PBStool` located in the environment `.PBStoolEnv`.

Example (plotFOScatch):

```
plotFOScatch(strSpp="401")  # DFO only
```

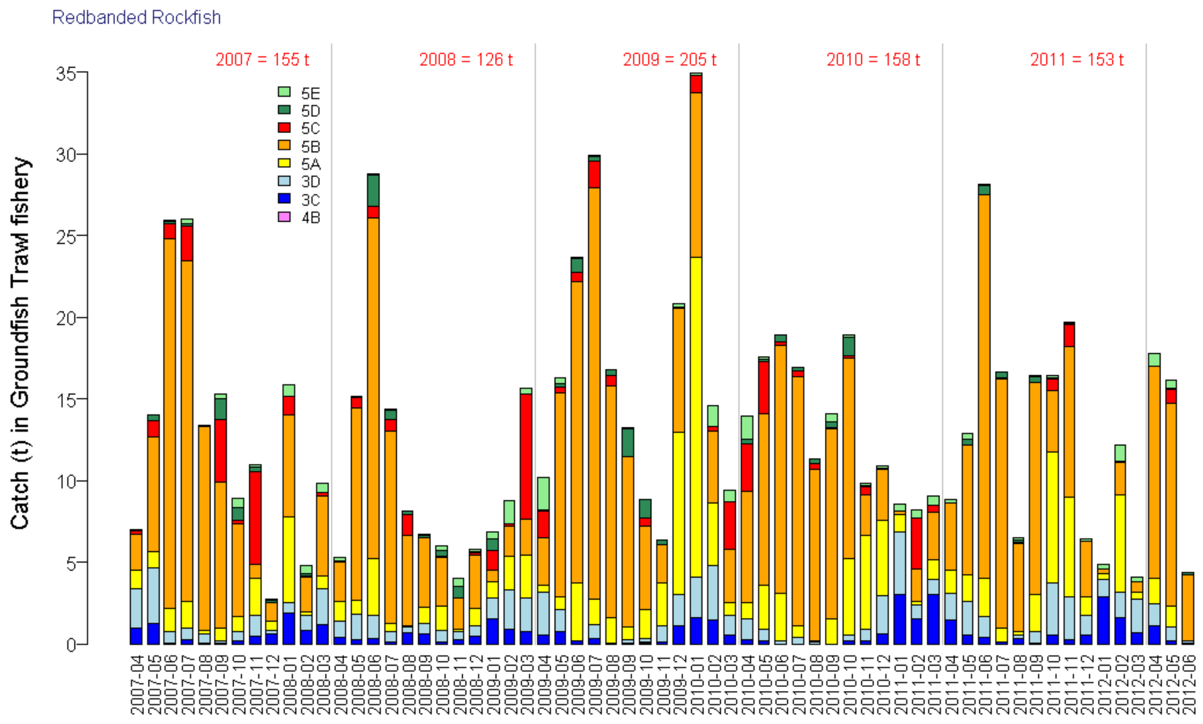


Figure 24. (*PBStools-plotFOScatch*) Monthly catch of Redbanded Rockfish (*Sebastes babcocki*) from BC's seven offshore PMFC areas. Fishing year is assumed to run from April to March.

3.10 runCCA

Perform a catch-curve analysis using a model formulated by Schnute and Haigh (2007). The code allows a user to perform both frequentist (NLM) and Bayesian (BRugs) analyses.

The function opens up a notebook GUI with two tabs named NLM and BRugs. The user needs to find a decent modal fit using `NLM` before attempting to run the Bayesian analysis, which employs Markov chain Monte Carlo (MCMC) techniques using the Gibbs sample algorithm.

Currently, the NLM fit can use one of three sampling distributions: multinomial, Dirichlet, and logistic-normal. The BRugs model is only coded for four cases using the Dirichlet distribution:

`M`	case 1: survival only, which depends only on mortality,
`MS`	case 2: survival and selectivity only,
`MA`	case 3: survival and recruitment anomalies only,
`MSA`	case 4: survival, selectivity, and recruitment anomalies.

The Bayesian model is coded in the OpenBUGS language and can be found in the examples subdirectory `library/PBStools/examples`.

3.11 sumCatTabs

The catch reconstruction algorithm of Haigh and Yamanaka (2011) builds input catch data files for a specified rockfish species from numerous databases (SQL Server and Oracle). This function takes those input files and summarizes catch by year and PMFC area.

3.12 trackBycat

Track fish group catches by year and PMFC area that occur between depths corresponding to a target species' depth-of-occurrence. The function currently uses three SQL query files called `gfb_bycatch.sql`, `pht_bycatch.sql`, and `fos_bycatch.sql`, which tap into the DFO databases GFBioSQL, PacHarvest, and GFFOS, respectively.

The function displays two barplots of annual catch, absolute and relative, where bars are sectioned by fish groups.

If `rda=NULL`, then three SQL queries extract annual fish group catches from the databases above. These catches are transferred to a 3-dimensional array `bycatch`:

`year`	years spanning all datasets;
`fish`	fish groups: POP, rockfish, turbot, flatfish, hake, sharks, other;
`db`	DFO databases: `gfb` = GFBioSQL, `pht` = PacHarvest, `fos` = GFFOS.

The catch array is saved to an `.rda` file with a name that indicates major areas and trawl type (bottom or midwater).

If the argument `rda` is given an `.rda` name, the `bycatch` array is loaded, by-passing the SQL calls.

The `bycatch` array is returned invisibly. Additionally, a list object called `PBStool`, located in the user's global environment, is populated with:

`module`	the name of the **PBStools** module to which this function belongs;
`call`	the call to the function `trackBycat`;
`plotname`	name of the plot, should the user which to use it, e.g. `.plotDev(act="png")`;
`bartab`	matrix of catch (tonnes) by year and fish group, summed over the databases;
`amat`	matrix of absolute catch (kt) used in the upper barplot;
`rmat`	matrix of relative catch (0:1) used in the lower barplot;
`clrs`	vector of colour names to distinguish the fish groups within any one bar.

4 Survey functions

The survey functions arose from a paper we wrote for designing trawl surveys (Schnute and Haigh, 2003). Essentially, every survey stratum hosts a unique population for the target species of interest. Each population can be summarized by three parameters that determine a binomial-gamma distribution: p = proportion of tows that do not catch the target species, μ = mean density of the target species in tows that catch the target species, and ρ = the coefficient of variation of these positive-catch densities.

4.1 bootBG

Bootstrap stratified biomass estimates using random draws from the binomial-gamma distribution using strata population parameters (Schnute and Haigh, 2003) supplied. This function depends on the R-package **boot**. These parameters are used to estimate the survey population using random draws from the binomial-gamma distribution.

No value is explicitly returned by the function. A list object `PBStool` (located in the environment `.PBStoolEnv`) provides results of the analysis:

`dat`	data frame: a subset from the user input that contains population parameters for each unique stratum.
`bgtab`	matrix: parameter values and moment estimates for each stratum.
`dStab`	matrix: sampled density estimates for each stratum.
`BStab`	matrix: sampled biomass estimates for each stratum.
`bgsamp`	list: final simulated set of densities sampled from the binomial-gamma distribution and listed by stratum.
`Bboot`	list: results of boot::boot() listed for each simulation.
`Bqnt`	array: bootstrap quantiles for each simulation and type of confidence interval calculation.
`Best`	vector: biomass estimate for each simulation.
`Bobs`	scalar: observed biomass estimate (moment calculation).
`group`	vector: stratum grouping showing allocation of K tows.

The results contained in `PBStool` can be displayed using the function `showAlpha` (Figure 25), which plots bootstrapped biomass values and their quantile confidence levels α (see Fig. 3 in Schnute and Haigh (2003)).

Example (bootBG):

```
pbsfun=function(SID=1,S=500){
  bootBG(dat=pop.pmr.qcss,SID=SID,S=S)
  showAlpha()
  invisible() }
pbsfun()
```

4.2 calcMoments

Calculate survey moments, including relative biomass, for a species from raw survey data. The function uses `getData` to calculate survey specs using the SQL queries:

```
gfb_survey_stratum.sql
gfb_survey_catch.sql
```

These queries are modified SQL procedures originally designed by Norm Olsen (PBS, Nanaimo) to build the GFBioSQL tables:

```
BOOT_HEADER
BOOT_DETAIL
```

The function invisibly returns a list of survey moments (`B` = biomass estimate, `N` = number of tows, `V` = variance of biomass estimate, `sig` = standard deviation of biomass estimate, `CV` = coefficient of variation) by strata. The following example gets moment estimates for Sablefish (*Anoplopoma fimbria*) from the QCS Synoptic survey in 2003 and 2007.

Example (calcMoments):

```
pbsfun=function(strSpp="455", survID=c(1,121),dfo=TRUE){
  if (dfo) {
    momList=calcMoments(strSpp,survID)
    print(momList) }
  else showMessage("If connected to DFO network, set argument 'dfo=TRUE'")
  invisible() }
pbsfun()
```

Output (calcMoments):

```
$`1`
                B   N            V         sig         CV
18       1.058758  29 1.253268e-06 0.001119494 0.10573660
19     240.446606  56 1.186942e-02 0.108946846 0.04531020
20     343.011654  29 4.480791e-02 0.211678786 0.06171183
21      97.324284   6 5.088662e-03 0.071334858 0.07329605
22       2.187556   5 5.840734e-06 0.002416761 0.11047767
23      26.737186  39 1.892251e-04 0.013755912 0.05144861
24     105.330602  51 1.364367e-03 0.036937341 0.03506801
25     354.018155  19 3.207522e-02 0.179095553 0.05058937
Total 1170.114799 234 9.540189e-02 0.308871962 0.02639672

$`121`
               B   N            V         sig         CV
18      1.429550  33 2.599952e-06 0.001612437 0.11279330
19    168.390766  62 8.226319e-03 0.090699057 0.05386225
20    142.210387  24 4.710287e-03 0.068631531 0.04826056
21    126.053435   7 6.162359e-02 0.248240990 0.19693314
22      5.755936  19 8.220823e-05 0.009066875 0.15752218
23     15.778769  57 1.383198e-04 0.011760944 0.07453651
24    147.342124  48 1.772628e-03 0.042102586 0.02857471
25    265.025548   7 1.680502e-02 0.129634159 0.04891383
Total 871.986514 257 9.336097e-02 0.305550267 0.03504071
```

4.3 calcPMR

This function calculates the binomial-gamma parameters (p, μ, ρ) of a sample population, as described by Schnute and Haigh (2003):

`p` = proportion of zero values,
`mu` = the mean of the non-zero values,
`rho` = the CV of the non-zero values.

The output is a vector `c(n, p, mu, rho)`.

Example (calcPMR):

```
pbsfun=function(P=list(n=1000,p=0.25,mu=1,rho=0.5)) {
  unpackList(P)
  x = sampBG(n,p,mu,rho) # create a sample population
  y = calcPMR(x)
  cat("True parameters for sampling the binomial-gamma distribution:\n")
  print(unlist(P))
  cat("Estimated parameters of the sampled binomial-gamma distribution:\n")
  print(y)
  invisible() }
pbsfun()
```

Output (calcPMR):

```
True parameters for sampling the binomial-gamma distribution:
      n       p      mu     rho
1000.00    0.25    1.00    0.50
Estimated parameters of the sampled binomial-gamma distribution:
           n            p           mu          rho
1000.0000000    0.2340000    1.0244274    0.4912166
```

4.4 getBootRuns

This function calls the SQL query `gfb_boot.sql` to download the bootstrap results for the selected species from GFBioSQL tables `BOOT_HEADER` and `BOOT_DETAIL`. The bootstrap tables are maintained by Norm Olsen (PBS, Nanaimo).

The function returns a data frame containing the bootstrapped biomass estimates and confidence limits for `strSpp` from all surveys in which the species was observed (see output below for Roughtail Skate *Bathyraja trachura*). The data frame is ordered by `date` and `bootID`.

Example (getBootRuns):

```
pbsfun=function(strSpp="057",dfo=TRUE) {
  if (dfo) {
    bootTab=getBootRuns(strSpp)
    print(bootTab) }
  else showMessage("If connected to DFO network, set argument 'dfo=TRUE'")
  invisible() }
pbsfun()
```

Output (getBootRuns):

```
  survID bootID       date year defDoor defSpeed
1      1     17 2003-07-03 2003    72.4      5.8
2      1     35 2003-07-03 2003    72.4      5.8
3      3     19 2005-07-05 2005    75.2      5.8
4      3     37 2005-07-05 2005    75.2      5.8
6    123     10 2006-08-28 2006    64.3      5.5
5    122     12 2007-09-11 2007    68.5      5.4
7    129     34 2008-08-25 2008    80.7      5.1

                                         runDesc    biomass   bootMean
1                              2003 QCS Synoptic 15351.1431 15235.4372
2 April 2009 Update for 2003 QCS Synoptic Survey 15395.4115 15158.6269
3                              2005 QCS Synoptic   607.6942   604.6557
4 April 2009 Update for 2005 QCS Synoptic Survey   607.6942   585.2095
6            2006 WQCI with 2007 stratification.  1187.0096  1187.4331
5                            2007 WCQCI Synoptic  7818.5815  7850.4116
7                            2008 WCQCI Synoptic 26656.8828 26645.2189

  bootMedian  bootLoCI  bootHiCI    bootRE catchWt numSets numPosSets
1 14366.3844  4515.370 44178.042 0.5411575   27.20     235          4
2 13944.6661  5033.791 46233.239 0.5540315   27.20     234          4
3   607.6942     0.000  1215.388 0.9110855    1.20     224          1
4   607.6942     0.000  1823.083 0.9677396    1.20     224          1
6  1187.0096     0.000  3409.161 0.6316274    4.91      96          3
5  7540.2430  2155.460 22393.798 0.5037474   25.58     112          4
7 25591.3002 10155.392 60289.184 0.4287963   26.56     117          8
```

4.5 getPMR

Get `p`, `mu`, and `rho` values for populations by strata from survey data housed in GFBioSQL using the query `gfb_pmr.sql` stored in `library/PBStools/sql`.
The function executes the SQL query for each survey ID `i`:

```
getData("gfb_pmr.sql","GFBioSQL",strSpp,path=.getSpath(),surveyid=i)
```

and collects the results in a data frame `pmrTab`. Each row of this data frame describes the population parameters (`p`, `mu`, `rho`) for each unique survey ID and survey stratum `h`. The output table also has named attributes: `fqt` (query name), `strSpp` (string species code), `h` (strata), `surveys.all` (data frame from a call to `getBootRuns`), and `surveys.selected` (records from `surveys.all` with the specified `survID`).

Example (getPMR):

```
pbsfun=function(dfo=TRUE){
  if (dfo) {
    pmrTab=getPMR(strSpp="396", survID=c(1,121))
    print(pmrTab) }
  else showMessage("If connected to DFO network, set argument 'dfo=TRUE'")
  invisible() }
pbsfun()
```

Output (getPMR):

```
pmr for species 396
-------------------
Survey ID: 1, 121
Done and done in 2.4 sec
   SID  h          p           mu       rho    A  n k
1    1 18 0.86206897 0.0086473494 1.0956975 5093 29 1
2    1 22 0.80000000 0.0020885547 0.4472136 2024  5 1
3    1 24 0.03921569 1.4766607698 1.5224993 3976 51 1
4    1 23 0.46153846 0.1064740216 1.6957557 4332 39 1
5    1 21 0.00000000 1.7507531455 0.5205263  496  6 1
6    1 25 0.00000000 0.3422540526 1.5298557 1220 19 1
7    1 20 0.00000000 3.1035614080 0.8172012 2931 29 1
8    1 19 0.21428571 1.5815733782 2.2459383 5582 56 1
9  121 18 0.96969697 0.0013227513 0.1740777 5093 33 1
10 121 22 0.94736842 0.0002308403 0.2294157 2024 19 1
11 121 24 0.00000000 0.9134459070 1.9488203 3976 48 1
12 121 23 0.45614035 0.2017947511 2.7973655 4332 57 1
13 121 21 0.14285714 0.9402312891 1.1255596  496  7 1
14 121 25 0.28571429 0.2134502924 0.5085555 1220  7 1
15 121 20 0.00000000 1.7714366587 1.1180694 2931 24 1
16 121 19 0.50000000 0.7139279217 1.9723779 5582 62 1
```

4.6 makePMRtables

Create `.csv` files containing (p, μ, ρ) (Schnute and Haigh, 2003) values for strata in the specified surveys. The code scrolls through the survey IDs, issuing sequential SQL queries to the DFO database GFBioSQL, and collects the debris in `.csv` files, one for each name in the list surveys.

Example (makePMRtables):

```
pbsfun=function(strSpp="451",dfo=FALSE) {
  if (dfo) makePMRtables(strSpp)
  else showMessage("If connected to DFO network, set argument 'dfo=TRUE'")
  invisible() }
pbsfun()
pbsfun()
```

Output:
See files `'pmrqcss.csv'`, `'pmrwcvi.csv'`, and `'pmrwqci.csv'` in your current working directory. These file contain binomial-gamma parameters for Shortspine Thornyhead (*Sebastolobus alascanus*) in each of the survey's strata.

4.7 makeSSID

Make a data object of survey series information called `ssid` for use in the package **PBSdata**. The function imports the SQL table `SURVEY` from the DFO database GFBioSQL using:

```
getData("SURVEY","GFBioSQL")
```

The survey information is then summarised by its series number and saved to a binary `.rda` file.

4.8 sampBG

Take random samples from the binomial-gamma distribution. This function generates random samples from both the binomial and the gamma distributions, and returns the product of the two.

Example (sampBG):

```
pbsfun=function(P=list(n=1000,p=0.25,mu=1,rho=0.5)) {
  unpackList(P)
  x = sampBG(n,p,mu,rho) # create a sample population
  y = calcPMR(x)
  cat("True parameters for sampling the binomial-gamma distribution:\n")
  print(unlist(P))
  cat("Estimated parameters of the sampled binomial-gamma distribution:\n")
  print(y)
  invisible() }
pbsfun()
```

Output (sampBG):

```
True parameters for sampling the binomial-gamma distribution:
      n       p      mu     rho
1000.00    0.25    1.00    0.50
Estimated parameters of the sampled binomial-gamma distribution:
           n            p           mu          rho
1000.0000000    0.2600000    0.9897658    0.5169940
```

4.9 showAlpha

Show quantile confidence levels α for bootstrapped biomass. This function assumes that `bootBG()` has been run and that the data list object `PBStool` exists in the global position. The figure generated (Figure 25) follows the format of Figure 3 in Schnute and Haigh (2003). The panels show:

(a) histogram of bootstrapped biomass;
(b) cumulative probability distribution of bootstrapped biomass;
(c) normal quantiles of the estimates $\hat{B}$;
(d,e) observed probabilities that observed value B lies in the j^{th} confidence interval;
(f,g) $\hat{\alpha}_j$ vs. α_j for chosen confidence limit types.

Example (showAlpha):

```
ppbsfun=function(SID=1,S=1000){
  bootBG(dat=pop.pmr.qcss,SID=SID,S=S)
  showAlpha()
  invisible() }
pbsfun()
```

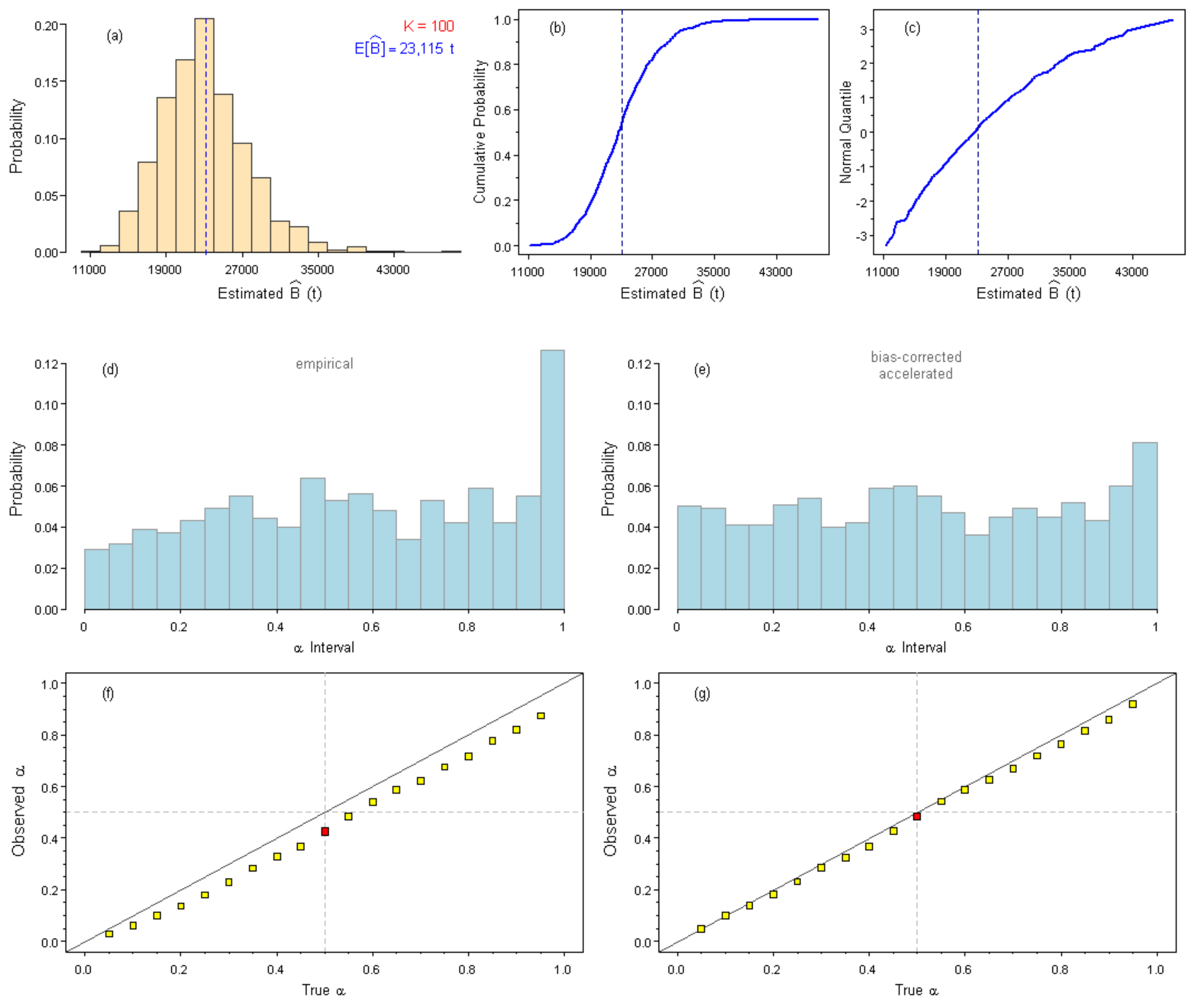


Figure 25. (*PBStools-showAlpha*) Results from S=1000 simulations, R=500 bootstraps, and tow budget K=100 for POP caught in eight strata of the Queen Charlotte Sound survey in 2003 (235 tows). (**Panels a-c**) (a) histogram, (b) cumulative probability distribution, and (c) normal quantiles for the S estimates $\hat{B}$. Vertical dashed lines in (b) and (c) denote the observed value B = 23t. (**Panels d-e**) Observed probabilities that the observed value B lies in the j^{th} confidence interval, based on (d) empirical confidence limits and (e) limits adjusted for bias correction and acceleration. (**Panels f-g**) plot of $\hat{\alpha}_j$ vs. α_j based on (f) empirical confidence limits and (g) limits adjusted for bias correction and acceleration. Vertical and horizontal dashed lines in (f g) show median levels at 50%, and solid lines denote the equality $\hat{\alpha} = \alpha$.

4.10 showIndices

Show survey indices for a species from bootstrapped biomass estimates stored in the SQL database GFBioSQL. The function relies on a short query function called `getBootRuns` which in turn queries tables of survey information (`BOOT_HEADER`) and relative biomass estimates by species for each survey (`BOOT_DETAIL`). The results are plotted with their confidence intervals (Figure 26). The user should be aware that any survey ID `survID` can have multiple boot runs `bootID` (e.g., re-stratification trials).

Example (showIndices):

```
pbsfun=function(strSpp="602",region="QCS",dfo=TRUE){
  if (dfo) {
    choices=c("WCVI","QCS","WQCI","HS")
    i=match(region,choices,nomatch=0)
    if (i==0) showError(paste("Choose region from\n'",
      paste(choices,collapse="', '"),"'",sep=""),as.is=TRUE)
    survID=switch(i,
      c(16,58,70,124,125,126,128),
      c(1,2,3,59,121,127),
      c(57,122,123,129),
      c(15,111,162,163) )
    showIndices(strSpp,survID) }
  else showMessage("If connected to DFO network, set argument 'dfo=TRUE'")
  invisible() }
pbsfun()
```

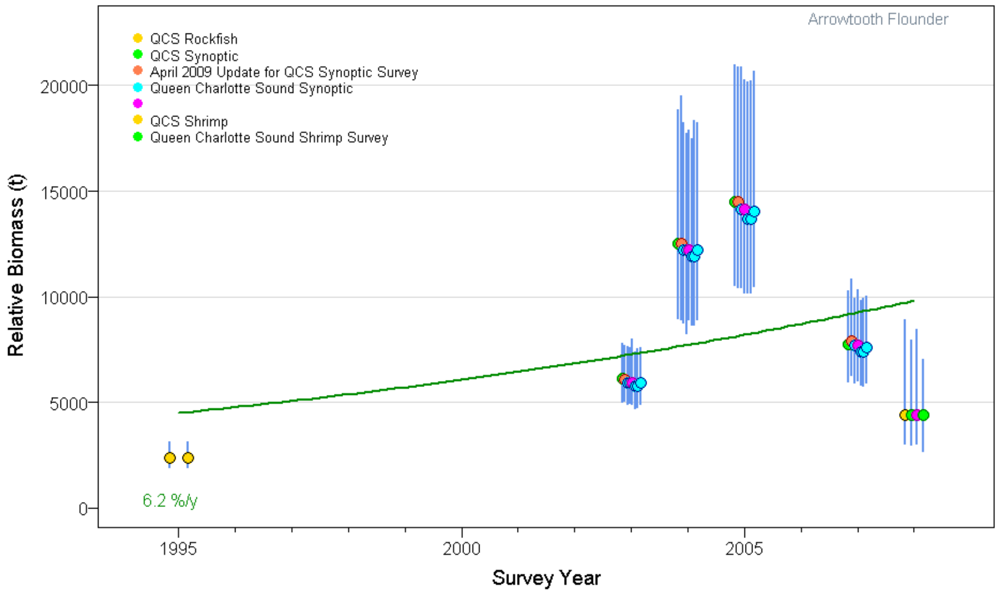


Figure 26. (*PBStools-showIndices*) Relative abundance indices from surveys in Queen Charlotte Sound for Arrowtooth Flounder (*Atheresthes stomias*). The bootstrapped results are stored on DFO's SQL Server database GFBioSQL. Duplicated points usually indicate a restratification exercise but can occasionally represent two surveys in the same year.

4.11 simBGtrend

Simulate a population projection based on prior binomial-gamma parameters and display the results. This function attempts to simulate a population past the observed year by randomly sampling the binomial-gamma distribution using weighted prior values of the population parameters `p`, `mu`, and `rho`.

The forward simulation is essentially based on momentum without reference to age classes, environment, or mortality. Users should be aware that this simulation remains naïve.

Example (simBGtrend):

```
pbsfun=function(Nsim=1,alpha=1) {
  simBGtrend(Nsim=Nsim,alpha=alpha)
  invisible() }
pbsfun(Nsim=7)
```

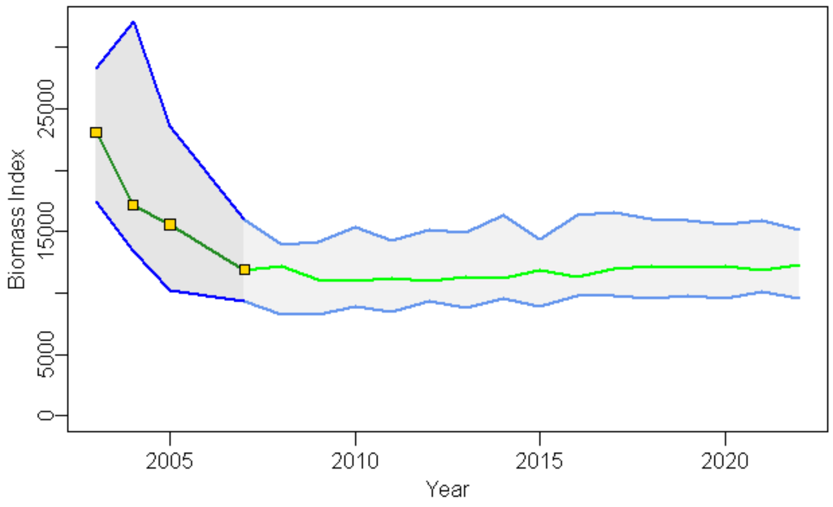


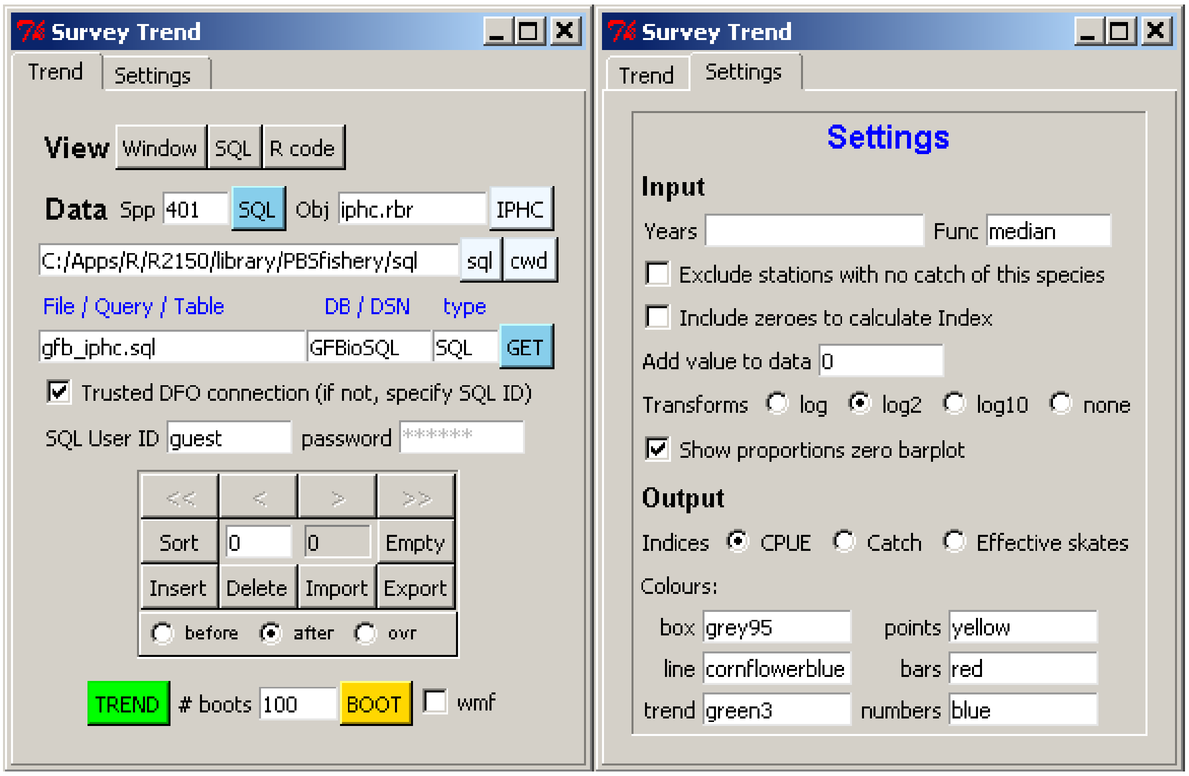
Figure 27. (*PBStools-simBGtrend*) Forward simulation (based on 7 trajectories) of POP abundance in Queen Charlotte Sound using past values of stratum population parameters (p, μ, ρ).

4.12 trend

An interactive GUI menu (Figure 28) created by a call to `trend` facilitates the exploration of trends in survey data. Once the user chooses a dataset, pressing the TREND button displays boxplots of catch or CPUE grouped by year. A trend line (Figure 29) fit through annual points, summarized by a user-specified summary function (e.g., `mean`), yields an annual rate of change r and the accumulated change R over the period of the fit (Schnute et al., 2004*b*). Pressing the BOOT button generates and displays bootstrapped estimates of slope b and annual rate r.

The package includes three sample datasets from the International Pacific Halibut Commission (IPHC)[8]: `iphc.rbr`, `iphc.rer`, and `iphc.yyr` for Redbanded Rockfish (*Sebastes babcocki*), Rougheye Rockfish (*S. aleutianus*), and Yelloweye Rockfish (*S. ruberrimus*), respectively. These data have been summarized extensively, the latest by Obradovich et al. (2008). The IPHC director, Dr. Bruce Leaman, has kindly granted PBS permission to include them here.

[8]International Pacific Halibut Commission (IPHC), P.O. Box 95009, Seattle, WA 98145-2009, U.S.A


Survey Trend

Trend | Settings

View: Window | SQL | R code

Data Spp 401 SQL Obj iphc.rbr IPHC

C:/Apps/R/R2150/library/PBSfishery/sql sql cwd

File / Query / Table: gfb_iphc.sql DB / DSN: GFBioSQL type: SQL GET

[✓] Trusted DFO connection (if not, specify SQL ID)

SQL User ID guest password *******

<< < > >>

Sort 0 0 Empty

Insert Delete Import Export

before [✓ after] ovr

TREND # boots 100 BOOT [ ] wmf

Survey Trend

Trend | Settings

Settings

Input

Years Func median

[ ] Exclude stations with no catch of this species

[ ] Include zeroes to calculate Index

Add value to data 0

Transforms log [✓ log2] log10 none

[✓] Show proportions zero barplot

Output

Indices [✓ CPUE] Catch Effective skates

Colours:

box grey95 points yellow

line cornflowerblue bars red

trend green3 numbers blue

Figure 28. (*GUI-trend*) GUI menu system created by a call to `trend`. The user can either specify a data object or launch an SQL query to GFBioSQL (DFO only).

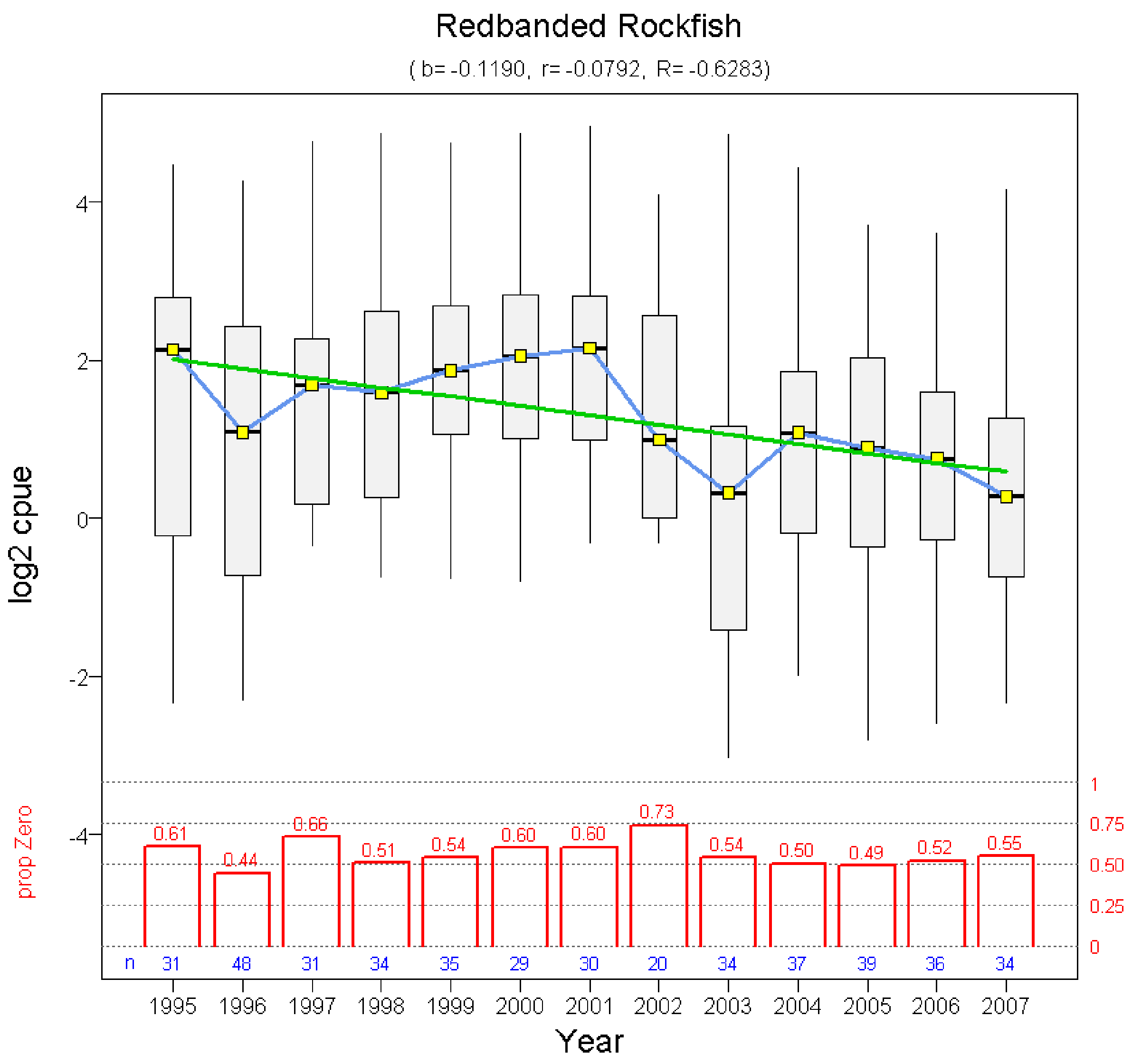


Figure 29. (*PBStools-trend*) IPHC survey CPUE index ($\log_2$(number of fish/effective skate)) for Redbanded Rockfish (*Sebastes babcocki*). The boxplots summarize annual non-zero data, where box whiskers show the extent of the data and box horizontal division indicates the median. Square symbols show the annual values of the specified summary function (in this case `median`). Superimposed on the boxplots is a regression line of the summary function. Barplots below show the proportion of non-zero values in the original dataset, and values at the base of the bars indicate the number of points used in the index calculation. Values in the subtitle summarize the trend line, where b = slope of the regression, r = annual rate of change, and R = accumulated change over the period of the trend line.

5 Spatial functions

Spatial functions arose primarily from work that DFO contributed to COSEWIC[9] in collaboration with the federal Ministry of Environment. Species-at-risk legislation requires that we know the spatial extent of marine species flagged for concern.

5.1 calcHabitat

This function calculates potential habitat using specified bathymetry limits. The default topography dataset `bctopo` was downloaded from the site `http://topex.ucsd.edu/cgi-bin/get_data.cgi` after specifying these boundaries: west (-134.5), east (-124.5), south (48), north (55). A user from another region could download a similar data file with different boundaries.

`calcHabitat` uses the **PBSmapping** function `makeTopography`, which slows down as the precision chosen for the longitude-latitude data increases. Setting `digits=1` provides a fast but roughly-estimated habitat region. Increasing the precision to `digits=2` slows the code markedly but provides a much more detailed bathymetry outline (Figure 30).

Example (calcHabitat):

```
calcHabitat(isob=c(150,435), digits=2)
```

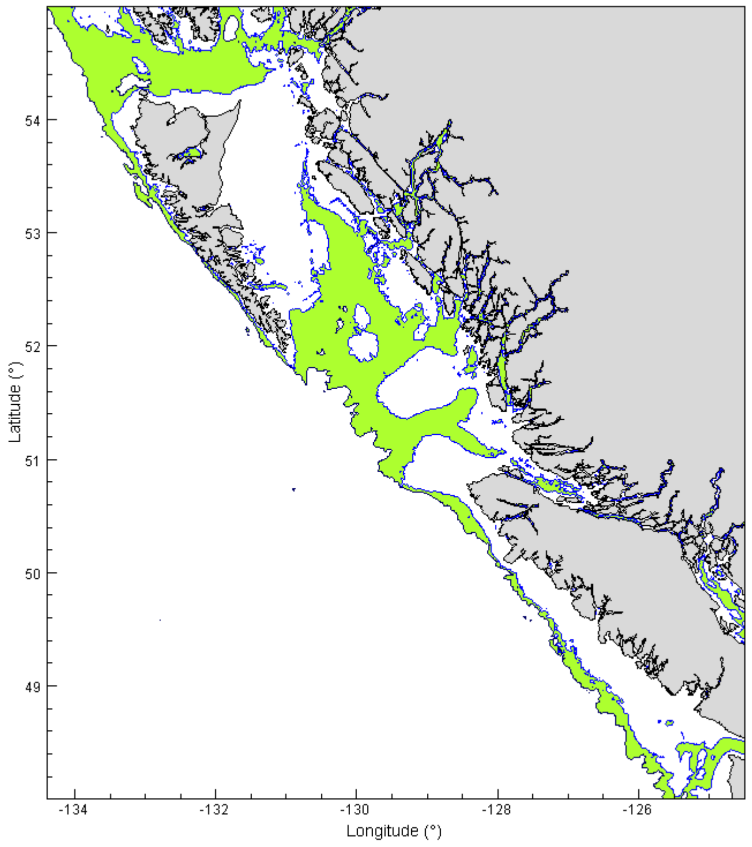


Figure 30. (*PBStools-calcHabitat*) Highlighted bathymetry (green) between 150 and 435 m serves as a proxy for Darkblotched Rockfish (*Sebastes crameri*) habitat along the BC coast.

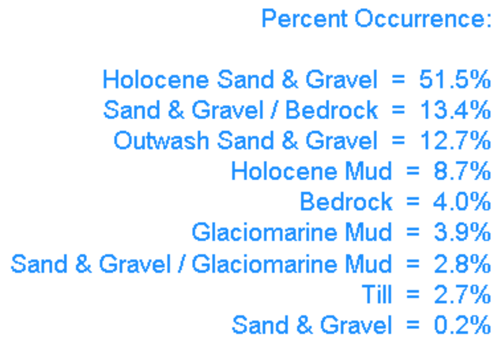
[9]Committee on the Status of Endangered Wildlife in Canada (`http://www.cosewic.gc.ca/`)

5.2 calcOccur

This function calculates the percent occurrence of events (`EventData`) in a set of labelled polygons (`PolySet`). Consequently, the `PolySet` must have a `PolyData` attribute with a category field called `label` that provides categories or bins in which to count events and calculate percent occurrence. The results for the tow subset `testdatC` in various categories of surficial geology for the Queen Charlotte Basin (Barrie et al., 1991; Sinclair et al., 2005) are displayed in Figure 31.

Example (calcOccur):

```
pbsfun=function(){
  data(testdatC)
  dat=testdatC[!is.na(testdatC$X) & !is.na(testdatC$Y),]
  edat=as.EventData(dat,projection="LL",zone=9)
  calcOccur(polyset="qcb", events="edat", mess=TRUE)
  invisible() }
pbsfun()
```



```
                                    Percent Occurrence:

                    Holocene Sand & Gravel  =  51.5%
                  Sand & Gravel / Bedrock  =  13.4%
                   Outwash Sand & Gravel  =  12.7%
                            Holocene Mud  =  8.7%
                                 Bedrock  =  4.0%
                         Glaciomarine Mud  =  3.9%
       Sand & Gravel / Glaciomarine Mud  =  2.8%
                                    Till  =  2.7%
                           Sand & Gravel  =  0.2%
```

Figure 31. (*PBStools-calcOccur*) Percent occurrence of tows from `testdatC` in the surficial geology categories of the Queen Charlotte Basin `PolySet`.

5.3 calcSRFA

Determine the slope rockfish assessment areas from combinations of PMFC major and minor areas as well as locality codes for fishing grounds within PMFC minor areas. The function either accepts three vectors (`major`, `minor`, `loc`) of equal length or one matrix/data frame `major` with three columns corresponding to the first three arguments. Slope rockfish assessment subareas or gullies can be determined by setting the argument `subarea=TRUE`.

Example (calcSRFA):

```
pbsfun=function(){
  dat=data.frame(major=c(3:7,9,9),
    minor=c(23,26,11,8,8,35,31),loc=c(1,1,1,3,6,1,1))
  sdat=cbind(dat,srfa=calcSRFA(dat),srfs=calcSRFA(dat,subarea=TRUE))
  print(sdat); invisible() }
pbsfun()
```

5.4 calcSurficial

Calculate the intersection of two `PolySets`, specifically surficial geology and potential habitat using a bathymetry interval. The results of the intersection are displayed on a map, including the area (km^2) of surficial geology categories that occurs in the bathymetry interval or potential habitat (Figure 32).

Table 5. Output from `calcSRF` function.

	major	minor	loc	srfa	srfs
1	3	23	1	3C	
2	4	26	1	3D	
3	5	11	1	5AB	GS
4	6	8	3	5AB	MI
5	7	8	6	5CD	MR
6	9	35	1	5EN	
7	9	31	1	5ES	

Example (calcSurficial):

```
pbsfun=function() {
  calcHabitat(isob=c(150,435), digits=1, plot=FALSE) # DBR
  habitat = PBStool$habitat
  calcSurficial(surf="qcb", hab=habitat)
  invisible() }
pbsfun()
```

5.5 clarify

This function summarizes large datasets of catch represented by many species into n CLARA[10] clusters using the function `clara` in the package **cluster** (Kaufman and Rousseeuw, 1990). Essentially, `clara` sub-samples the large data set, identifying the best k medoids (centre is defined as the item which has the smallest sum of distances to the other items in the cluster) using a dissimilarity metric. In the end, all records are assigned to one of the k clusters. Further routines in the R-package **PBSmapping** locate fishing events in grid cells and display the results (Figure 33).

The demo `EventData` set used below is a subset of a potentially much larger file that might comprise hundreds of thousands of tows and 80+ fish species. This dataset has been pared down to the region bounded by 130.5°W, 128°W, 50.8°N, and 52°N. Additionally, it excludes all depths less than 100 m and only reports catch for 13 fish species.

Example (clarify):

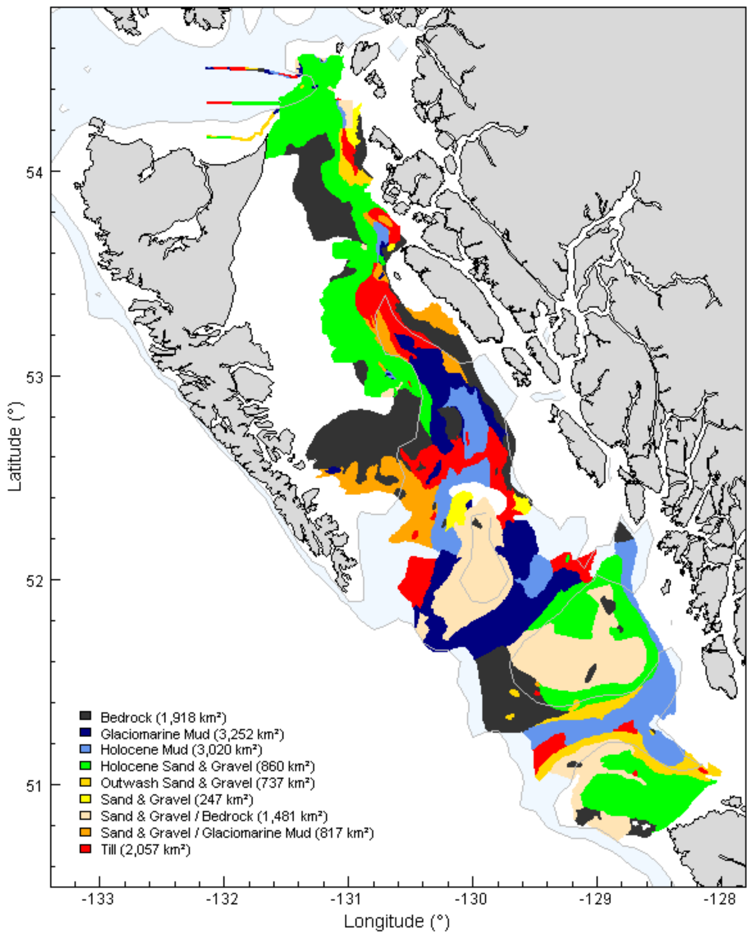
```
pbsfun=function(){
  clarify(claradat, xlim=c(-132,-127.8), ylim=c(50.6,52.1),
    cell=c(.05,.03),nG=7)
  invisible() }
pbsfun()
```

5.6 findHoles

Identifies polygons within polygons of the same PID, and transforms them into holes, if they are not already set that way. Also weeds out small polygons before the hole identification routine runs. This function can become very slow when there are hundreds of polygons. Hence the argument `minVerts` can reduce the number of candidate polygons. Ideally, `findHoles` should be programmed in C.

The output is a PolySet that identifies parents and holes based on the position of polygons within polygons.

[10] **C**lustering **Lar**ge **A**pplications



54
53
52
51
Latitude (°)
-133
-132
-131
-130
-129
-128
Longitude (°)
■ Bedrock (1,918 km²)
■ Glaciomarine Mud (3,252 km²)
■ Holocene Mud (3,020 km²)
■ Holocene Sand & Gravel (860 km²)
■ Outwash Sand & Gravel (737 km²)
■ Sand & Gravel (247 km²)
■ Sand & Gravel / Bedrock (1,481 km²)
■ Sand & Gravel / Glaciomarine Mud (817 km²)
■ Till (2,057 km²)

Figure 32. (*PBStools-calcSurficial*) Surficial geology of the Queen Charlotte basin and Hecate Strait. A rough bathymetry contour (150–145 m, derived from `calcHabitat` with `digits=1`) lies beneath the highlighted geology. The legend shows geology categories and the area (km^2) intersecting the bathymetry interval.

5.7 plotGMA

Plot the groundfish management areas (GMA) for Pacific Ocean Perch (*Sebastes alutus*) and Yellowmouth Rockfish (*S. reedi*) used by Pacific groundfish managers at RHQ in Vancouver. The PolySet `gma.popymr` can be found in the R package **PBSvault**, which is only available to DFO personnel, and contains an attribute called `PolyData` that defines colours and labels for each of the GMAs.

Example (plotGMA):

```
pbsfun=function(){
  if(is.element("PBSvault",.packages(all.available=TRUE))) {
    require(PBSvault); data(gma.popymr)
    pdata=attributes(gma.popymr)$PolyData
    plotMap(gma.popymr,polyProps=pdata)
    addLabels(pdata,cex=2,font=2) }
  invisible() }
pbsfun()
```

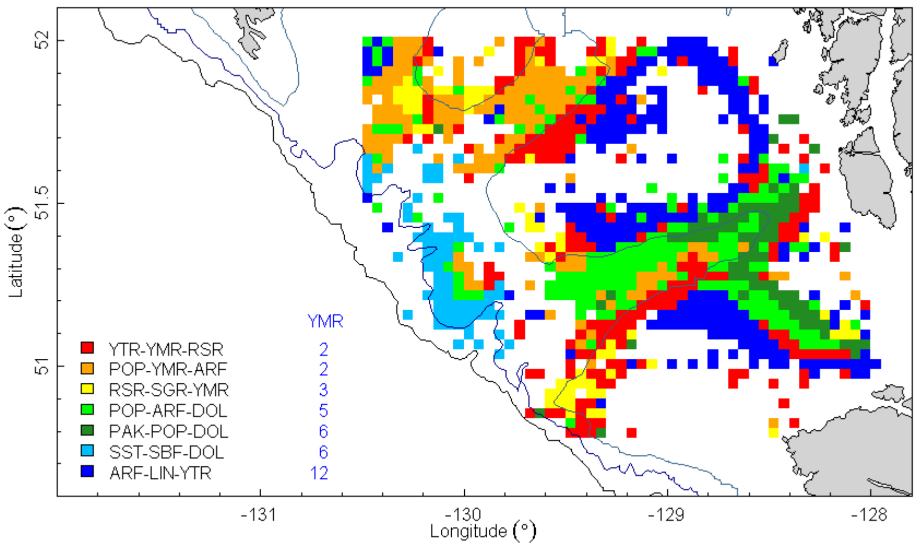


Figure 33. (*PBStools-clarify*) Groups identified by `clara` in R's package **cluster** and summarized by **PBSmapping** spatial functions called by `clarify`. Isobaths trace the 200, 1000, and 1800 m depth contours. The legend identifies seven clusters by the top three species comprising the medoids; the clusters are ordered by the contribution of Yellowmouth Rockfish (YMR) to each medoid.

Table 6. Species codes displayed in Figure 33.

ARF	Arrowtooth Flounder	*Atheresthes stomias*
BIS	Big Skate	*Raja binoculata*
DOL	Dover Sole	*Microstomus pacificus*
LIN	Lingcod	*Ophiodon elongatus*
PAK	Pacific Hake	*Merluccius productus*
POP	Pacific Ocean Perch	*Sebastes alutus*
ROL	Rock Sole	*Lepidopsetta bilineatus*
RSR	Redstripe Rockfish	*Sebastes proriger*
SBF	Sablefish	*Anoplopoma fimbria*
SGR	Silvergrey Rockfish	*Sebastes brevispinis*
SST	Shortspine Thornyhead	*Sebastolobus alascanus*
YMR	Yellowmouth Rockfish	*Sebastes reedi*
YTR	Yellowtail Rockfish	*Sebastes flavidus*

5.8 plotTernary

Plot a ternary diagram using compositional data classified into three categories. When population components are amalgamated into $g = 3$ groups, vectors of proportions can be portrayed in a graph called a ternary diagram (Aitchison, 1986, p. 5). The diagram begins with an equilateral triangle that has vertices labelled "1", "2", and "3". A vector **p** (p_1, p_2, p_3) of proportions can then be represented as a point within this triangle, where the perpendicular distance to the side opposite vertex "i" is proportional to p_i.

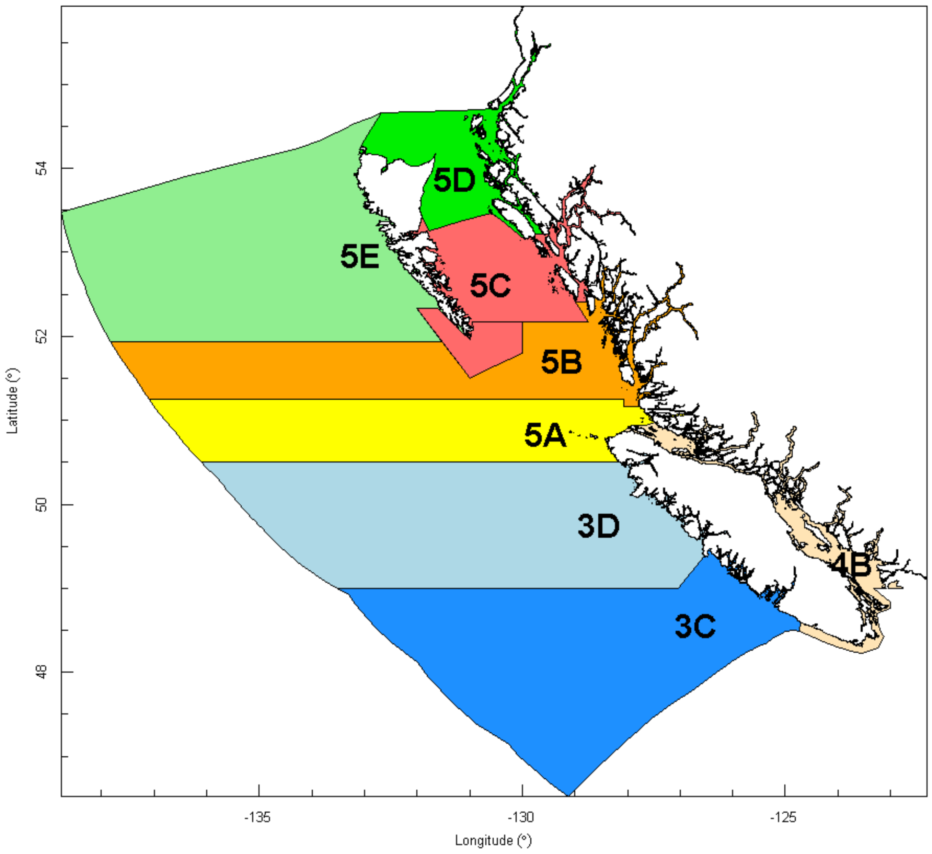


Figure 34. (*PBStools-plotGMA*) Groundfish Management Areas for Pacific Ocean Perch and Yellowmouth Rockfish.

Example (plotTernary):

```
pbsfun=function(){
  plotTernary(c(1/2,1/3,1/6))
  invisible() }
pbsfun()
```

5.9 plotTertiary

Plot a compositional diagram using data classified into n categories. This function seeks to extend the ternary diagram from 3 to n proportions. However, the geometry of mutiple vertices (squares, pentagons, hexagons, etc.) does not translate into deterministic solutions as it does for triangles (Aitchison, 1986; Schnute and Haigh, 2007). The algorithm currently yields a compromise solution because the constraint $\Sigma_{i=1}^{n} p_i = 1$ rarely yields a common point within polygons that are not equilateral triangles. The problem still needs more thought.

When only one proportion set (vector or single-row matrix) is supplied, the function shows lines that connect the mode to the vertices and to each side of the polygon. When multiple proportion sets are supplied, the modes are connected together to form a trace diagram.

Example (plotTertiary):

```
pbsfun=function(){
  plotTertiary(c(1,2,2,5,5))
  invisible() }
pbsfun()
```

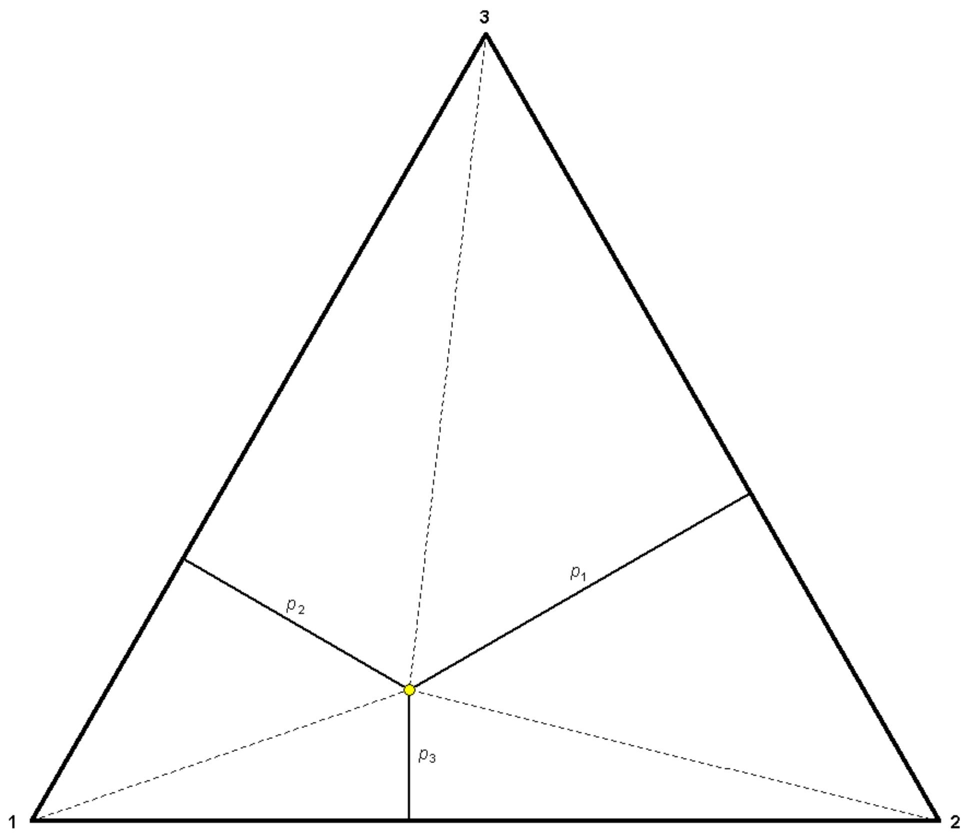


Figure 35. (*PBStools-plotTernary*) Ternary diagram for compositions amalgamated into $g = 3$ groups. The indicated point represents a vector of proportions. Solid lines perpendicular to the sides of an equilateral triangle have lengths proportional to $p_i(i = 1, 2, 3)$. Dotted lines facilitate a proof that this method actually works, given the constraint $\Sigma_{i=1}^{3} p_i = 1$ (Schnute and Haigh, 2007, Fig. 2).

5.10 preferDepth

This function displays the depth distribution for a selected species using an interactive GUI (Figure 37). The SQL code `pht_fdep.sql` grabs the depth table from the remote SQL Server database PacHarvest and the Oracle database GFFOS. The user may also specify an R object or a data file. The menu control offers some ability to specify years and/or areas (if available in the file). The depth-of-capture frequency for the selected species appears as a histogram (Figure 38). The cumulative catch is superimposed to show the depths at which the species is caught in relation to the tow frequency histogram. The total fleet effort is displayed as a shaded histogram in the background.

5.11 prepClara

Prepare a data object for use by `clarify`, which uses the **cluster** function `clara`. The grouping variable `gfld` will determine how event data are collapsed into spatial units. The Lon-Lat default is likely equivalent to a sinlge field identifying each fishing event unless events happen to occupy the same Lon-Lat coordinate. A user may wish to create a field that delimits grid cells, but these should be the same as or smaller than those specified in the function `clarify`.

The function assumes that the input data object is always called ‘dat’, but a source file can be called anything as long as its import to R creates `dat`. Similarly, the output data object is always called ‘claradat’ but can be saved to a system file with any name. The function automatically

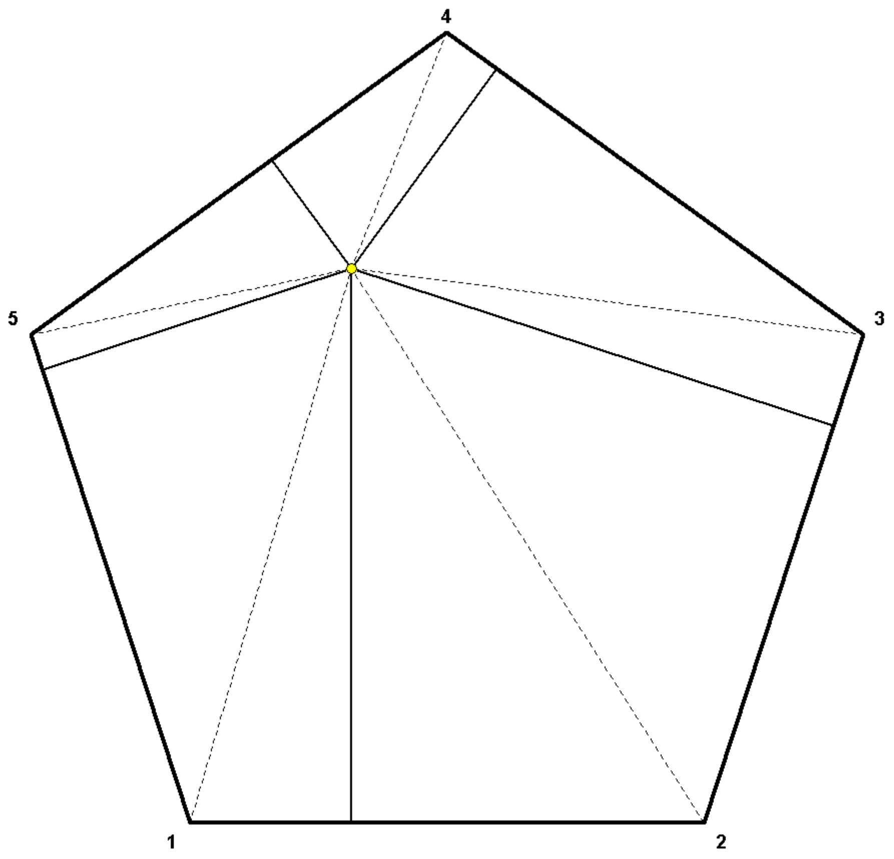


Figure 36. (*PBStools-plotTertiary*) Tertiary diagram for compositions amalgamated into $g = 5$ groups. The indicated point represents a vector of proportions. Solid lines perpendicular to the opposite sides of a polygon have lengths roughly proportional to $p_i (i = 1, ..., n)$.

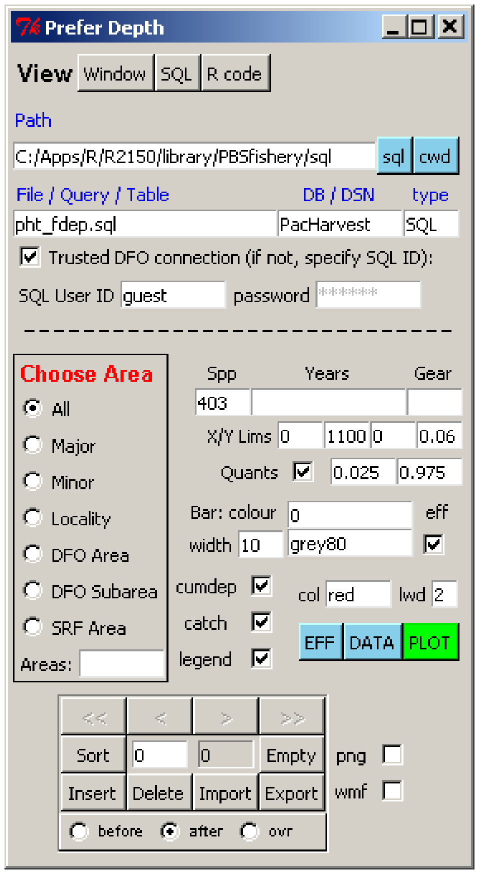
names the output file containing `claradat` depending on whether an input name `fnam` was specified or SQl code was executed to get the input data.

5.12 zapHoles

The function attempts to remove (zap) holes that will ultimately be filled anyway. It uses centroid matching between holes and solids to identify candidate holes for removal. The value returned is a `PolySet` potentially modified to remove holes that will be filled. Non-essential attributes will be retained and supplemented with two additional ones created by the **PBSmapping** functions `calcCentroid` and `calcArea`: (i) `keep` – data frame summarizing the kept polygons, and (ii) `zap` – data frame summarizing the holes removed.

The original rationale for this function is to groom a `PolySet` before using in `addPolys(pset,...,colHoles="white")`. The `colHoles` option was designed to get rid of retrace lines that appear in `.pdf` files made from metafiles.

That said, the `PolySet` created by this function essentially comprises layers. To see all the layers, the user must add them sequentially from largest to smallest, which is not terribly efficient.


Prefer Depth

View Window SQL R code

Path: C:/Apps/R/R2150/library/PBSfishery/sql sql cwd

File / Query / Table: pht_fdep.sql   DB / DSN: PacHarvest   type: SQL

[✓] Trusted DFO connection (if not, specify SQL ID):

SQL User ID: guest   password: ******

Choose Area: [● All] ○ Major ○ Minor ○ Locality ○ DFO Area ○ DFO Subarea ○ SRF Area

Areas:

Spp: 403   Years: Gear:

X/Y Lims: 0 1100 0 0.06

Quants [✓] 0.025 0.975

Bar: colour 0   eff

width 10   grey80   [✓]

cumdep [✓]   col red   lwd 2

catch [✓]

legend [✓]

EFF DATA PLOT

<< < > >>

Sort 0 0 Empty   png [ ]

Insert Delete Import Export   wmf [ ]

○ before [● after] ○ ovr

Figure 37. (*GUI-preferDepth*) GUI menu system created by a call to `preferDepth`. The SQL queries will only work for users on the DFO network. Alternatively, a data object or file can be specified in the File box and the type must be set to `FILE`.

Example (zapHoles):

```
pbsfun=function(){
  data(qcb); qcbz=zapHoles(qcb)
  pdata=attributes(qcb)$PolyData
  pdata=pdata[rev(order(pdata$area)),]
  expandGraph(mfrow=c(1,1),plt=c(.05,1,.05,1))
  plotMap(qcbz)
  for (i in 1:nrow(pdata)) {
    pid=pdata$PID[i]
    qcbi=qcbz[is.element(qcbz$PID,pid),]
    if (nrow(qcbi)==0) next
    addPolys(qcbi,col=pdata$col[i]) }
  invisible() }
pbsfun()
```

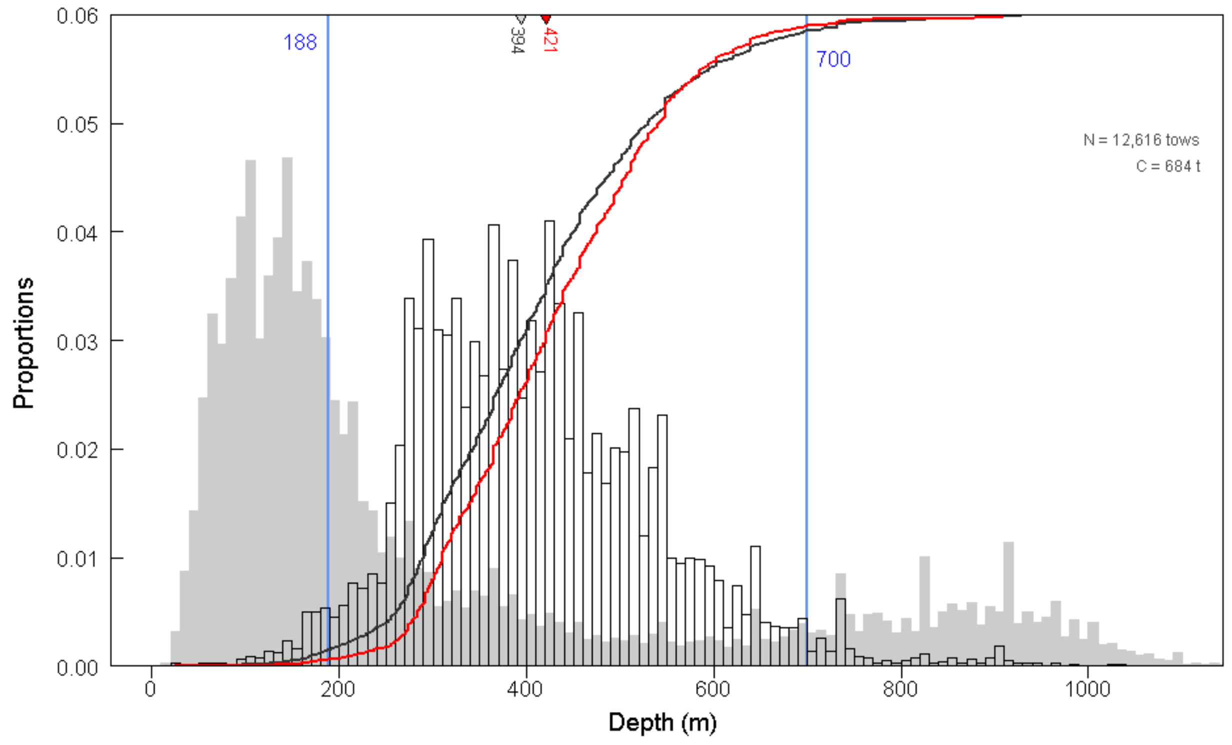


Figure 38. (*PBStools-preferDepth*) Relative frequency of tows in 10-m depth bins that capture Shortraker Rockfish (SKR, *Sebastes borealis*) from commercial trawl records (1996–2007). The vertical solid lines denote the 2.5% and 97.5% quantiles. The shaded histogram in the background reports the relative trawl effort on all species. The cumulative catch of SKR, superimposed on the histogram in relative space (0 to 1), provides confirmation that the bulk of SKR comes from these depths. The depth of median cumulative catch is indicated by an inverted red triangle at top. ‘N’ reports the total number of tows; ‘C’ reports the total catch (t).

6 Temporal functions

The temporal functions here are few, consisting of a moving average and an ad hoc method for calculating the rate of return on a portfolio of anything measurable (e.g. stocks). Other functions that help scientists visualize how populations change over time – changes in species composition or the diversity of phytoplankton samples – have been moved to a package called **PBSplankton**.

6.1 calcMA

Calculate a moving average of a series using a fixed period occurring every number of specified base units (usually days). The moving average is accumulated backwards from the last observation.

Example (calcMA):

```
pbsfun=function(os=.Platform$OS.type) {
  if (os=="windows") {
    getData("Ex03_Portfolio","Examples",type="MDB",path=.getSpath())
    # Annual average every 2 months:
    ma=calcMA(PBSdat[[2]],PBSdat[[3]],period=180,every=30)
    dat=attributes(ma)$dat
    plot(dat$x,dat$y,type="l")
    lines(ma$x,ma$y,col="blue",lwd=2) }
  invisible() }
pbsfun()
```

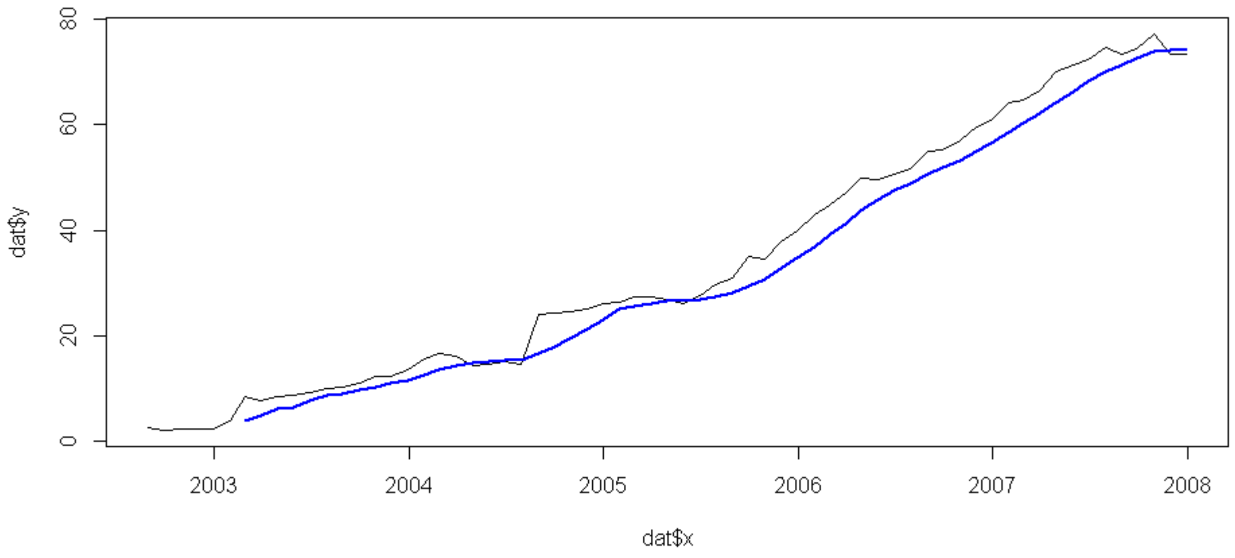


Figure 39. (*PBStools-calcMA*) Semi-annual moving average calculated every 30 days.

6.2 imputeRate

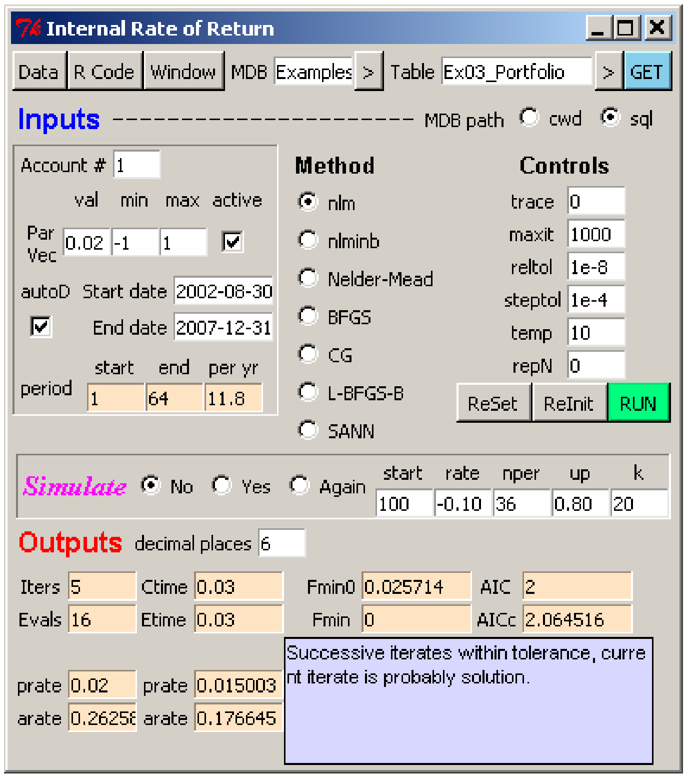
Impute the rate of return for an investment (existing account or simulated) that experiences regular or irregular contributions and/or withdrawals. This function creates an interactive GUI (Figure 40) to facilitate the process. The code adopts a variant of a formula in Microsoft Excel's XIRR function where the "internal rate of return" r is derived iteratively when the net present value (`NPV`) is set to 0.

$$\text{NPV} = \sum_{i=1}^{N} \frac{C_i}{(1-r)^{\left(\frac{d_i - d_1}{365}\right)}} \qquad (11)$$

where:

d_i = the i^{th} contribution date;
d_1 = the initial date;
C_i = the i^{th} contribution; and
C_1 = initial value of the portfolio (includes 1^{st} contribution).

Here we use the **PBSmodelling** function `calcMin` to minimize the squared difference between the calculated NPV of the contribution stream and the final value of the portfolio (Figure 41). User's can either specify an MDB table in the current working directory or `Examples.mdb` in the package's `sql` directory. There is also an option to simulate a dataset using the random pareto distribution.



MDB Examples > Table Ex03_Portfolio >

Inputs ---------------------- MDB path [ ] cwd [✓ sql]

Account # 1

	val	min	max	active
Par Vec	0.02	-1	1	✓

autoD ✓ Start date 2002-08-30 End date 2007-12-31

	start	end	per yr
period	1	64	11.8

Method: [✓ nlm] [ ] nlminb [ ] Nelder-Mead [ ] BFGS [ ] CG [ ] L-BFGS-B [ ] SANN

Controls

trace	0
maxit	1000
reltol	1e-8
steptol	1e-4
temp	10
repN	0

Simulate [✓ No] [ ] Yes [ ] Again

start	rate	nper	up	k
100	-0.10	36	0.80	20

Outputs decimal places 6

Iters	5	Ctime	0.03	Fmin0	0.025714	AIC	2
Evals	16	Etime	0.03	Fmin	0	AICc	2.064516
prate	0.02	prate	0.015003				
arate	0.26258	arate	0.176645				

Successive iterates within tolerance, current iterate is probably solution.

Figure 40. (*GUI-imputeRate*) GUI menu Internal Rate of Return to estimate the annualized return on an investment that experiences episodic contributions.

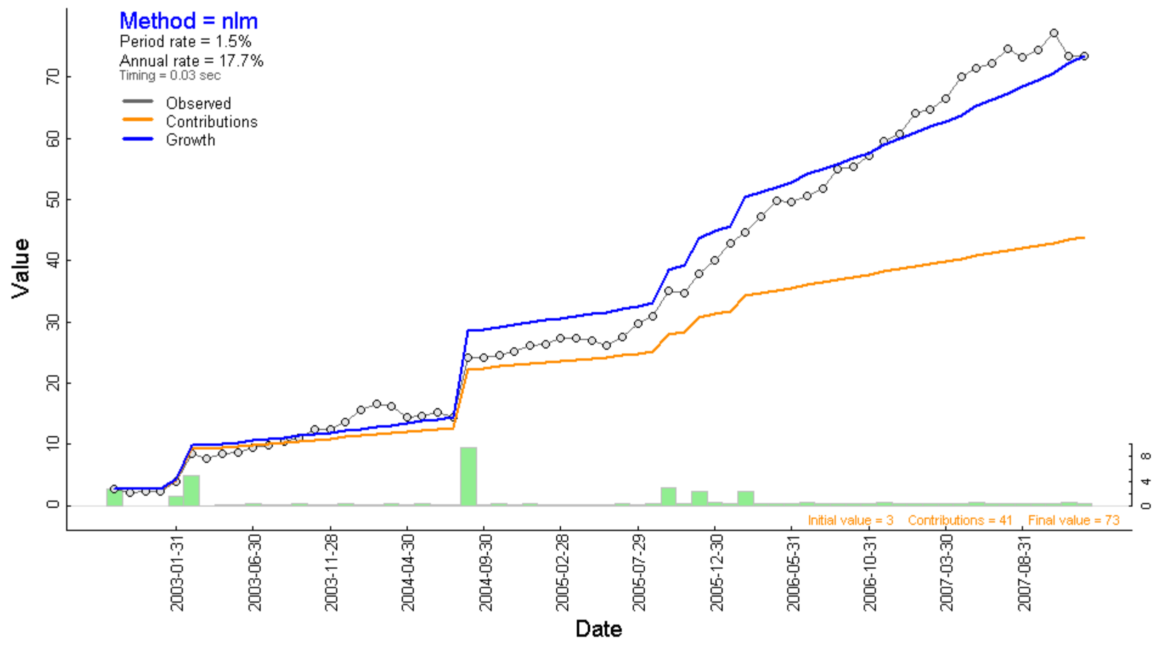


Figure 41. (*PBStools-imputeRate*) Portfolio progress where green bars represent episodic contributions, the orange line shows the cumulative contributions, the grey dots report the monthly portfolio values, and the blue line shows a calculated fit using the estimated internal rate of return.

7 Catch Reconstruction

7.1 buildCatch

The function `buildCatch` offers an automated algorithm for reconstructing rockfish (RRF) catches in British Columbia (BC); full details are given in Haigh and Yamanaka (2011). The catch reconstruction uses: (i) reliable landings from modern databases, (ii) estimated landings using ratios of RRF catch to other rockfish (ORF) or total rockfish (TRF) catch, (iii) estimated discards using discard ratios from observer logs.

The reconstruction algorithm estimates annual removals (landed catch + discards) in metric tonnes (t) of specified rockfish, by:
(i) calendar year,
(ii) Pacific Marine Fisheries Commission (PMFC[11]) area:
4B = (major 1) inside waters including Strait of Georgia,
3C = (major 3) SW coast of Vancouver Island,
3D = (major 4) NW coast of Vancouver Island,
5A = (major 5) southern Queen Charlotte Sound,
5B = (major 6) northern Queen Charlotte Sound,
5C = (major 7) southern Hecate Strait,
5D = (major 8) northern Hecate Strait and Dixon Entrance,
5E = (major 9) west coast Haida Gwaii (formerly Queen Charlotte Islands); and
(iii) fishery type (FID):
1 = groundfish trawl,
2 = Halibut longline,
3 = Sablefish trap/longline,
4 = schedule II comprising primarily Dogfish and Lingcod, and
5 = hook and line (H&L) rockfish, eventually regulated under ZN licensing.

Because historical catch data do not provide landings of individual rockfish species, we use broad rockfish categories (POP = Pacific Ocean Perch, ORF = rockfish other than POP, and TRF = total rockfish) and apply ratios that we calculate from modern catches (e.g., POP/TRF for Pacific Ocean Perch (*Sebastes alutus*) or YYR/ORF for Yelloweye Rockfish (*S. ruberrimus*)) to disaggregate historical catches. This assumes that modern ratios reflect historical ones. Many factors may invalidate this assumption – selective fishing, environmental shifts, habitat destruction; however, we have little choice if we want historical catch estimates for specific rockfish species.

The reconstruction ratios calculated by `buildCatch`:
α proportion of RRF caught in a PMFC area for each fishery,
β proportion of RRF caught in a PMFC area for Hook and Line (H&L) fisheries,
γ ratio of RRF to a larger group like ORF or TRF,
δ discard rate of RRF per other catch category from observer logs, and
λ proportion of early ORF or TRF catch by general gear type
are applied to the historical data to estimate historical landings of the RRF species. For POP the historical trawl landings are known back to 1956 and so these are used instead of deriving estimates.

Most sources of historical landings provide data that overlap in some years. Two of the sources provide unique and consequently additive information, regardless of the year. The first

[11]see footnote in Forrester (1969)

additive series is the very early landed catch from 1918 to 1950. It reflects a basal low exploitation rate that occurred prior to World War II. The second additive series is that of the foreign (Russian and Japanese) fleets that fished heavily along the BC coast from 1965 to 1976 until the inception of the 200-mile exclusive economic zone (EEZ). All other historical series may contain redundant information where they overlap, and so we take the maximum annual value from these.

For the early time period 1918–1949, landings of ORF and TRF are identical because catch agencies did not identify Pacific Ocean Perch (POP). POP started showing up in catch records in 1950, but appear artificially low from 1950–1952. Therefore, for the period 1918–1952 we predict ORF and POP using a linear regression of ORF *vs.* TRF landings from the 1953–1995 data (excluding an anomalous 1966 point):

$$O = 2^a T^b \tag{12}$$

where a = -0.8063372, b = 1.006260, O = ORF, T = TRF, and POP = TRF - ORF.

Landings (metric tonnes) from eight modern catch databases (GFCatch, PacHarvest, PacHarvHL, PacHarvSable, PacHarvHL, PacHarvHL, and GFFOS) are summarized in a 5-dimensional array where i = calendar year, j = PMFC major area code, k = fishery ID, l = species or catch group, and m = the database. Then for each element (i, j, k, l), the maximum landing across database m is taken. This methodology is utilized due to the non-contiguous nature of the data sources. This also means that compiling individual catch records across the database sources, as performed for the trawl fishery by `getCatch`, is too complicated for the hook and line fisheries. It is possible if definite cut points can be identified for the various time series (which we no longer do for the reconstruction).

The reconstruction executes a complex series of rules using SQL (Structured Query Language) queries (shown in the DFO boxes below) combined with R code (R Core Team, 2024) to derive an annual catch series for five fisheries (Trawl, Halibut, Sablefish, Dogfish-Lingcod, H&L (hook and line) Rockfish) in BC's eight PMFC areas (4B, 3C, 3D, 5A, 5B, 5C, 5D, 5E). This yields 40 annual series (48 when fisheries are combined).

DFO:

PacHarv3 catch summary for fishery IDs 1:5 and 0 (unknown):

```
getData("ph3_fcatORF.sql",dbName="HARVEST_V2_0",strSpp=strSpp,path=.getSpath(),
   server="ORAPROD",type="ORA",trusted=FALSE,uid=uid[1],pwd=pwd[1])
   assign("ph3dat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   dimnames(ph3dat)[[1]]=1:nrow(ph3dat)
   save("ph3dat",file="ph3dat.rda")
```

GFCatch records for fishery IDs (1,3,5):

```
getData("gfc_fcatORF.sql","GFCatch",strSpp=strSpp,path=.getSpath())
   assign("gfcdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   gfcdat$year=as.numeric(substring(gfcdat$date,1,4))
   dimnames(gfcdat)[[1]]=1:nrow(gfcdat)
   save("gfcdat",file="gfcdat.rda")
```

PacHarvest records for fishery ID (1):

```
getData("pht_tcatORF.sql","PacHarvest",strSpp=strSpp,path=.getSpath())
   assign("phtdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   save("phtdat",file="phtdat.rda")
```

PacHarvHL halibut validation records for fishery IDs (2,7):

```
getData("phhl_hcatORF.sql","PacHarvHL",strSpp=strSpp,path=.getSpath()) ### validated DMP
   assign("phhdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   save("phhdat",file="phhdat.rda")
```

PacHarvSable fisherlogs for fishery ID (3):

```
getData("phs_scatORF.sql","PacHarvSable",strSpp=strSpp,path=.getSpath(),
   fisheryid=3,logtype="FISHERLOG")
   assign("phsdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   save("phsdat",file="phsdat.rda")
```

PacHarvHL validation records for fishery IDs (2,4,5):

```
getData("phhl_vcatORF.sql","PacHarvHL",strSpp=strSpp,path=.getSpath()) ### validated DMP
   assign("phvdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   save("phvdat",file="phvdat.rda")
```

PacHarvHL fisherlog records for fishery IDs (4,5):

```
getData("phhl_fcatORF.sql","PacHarvHL",strSpp=strSpp,path=.getSpath(),
   logtype="FISHERLOG") ### fisherlog catch
   assign("phfdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   save("phfdat",file="phfdat.rda")
```

GFFOS catch from all fishery IDs (1:5):

```
getData("fos_vcatORF.sql",dbName="GFFOS",strSpp=strSpp,path=.getSpath(),
   server="GFSH",type="ORA",trusted=FALSE,uid=uid[2],pwd=pwd[2])
   assign("fosdat",PBSdat); rm(PBSdat,envir=.GlobalEnv) ### just to be safe
   dimnames(fosdat)[[1]]=1:nrow(fosdat)
   save("fosdat",file="fosdat.rda")
```

The final reconstructed catch array is invisibly returned. Additionally, if `saveinfo = TRUE`, various data objects internal to the function are saved to a list object called `PBStool` (located in the environment `.PBStoolEnv`). Also, various data objects, including the eight database catch files consolidated and massaged, are saved as system binary files `.rda`. The final reconstructed catch array is also flattened to a comma-delimited text file `Catch-History-123.csv`, where `123` represents the species code. Summary catch barplots, specified by the argument `fidout`, are plotted on the R graphics device or sent to `wmf` file(s) (e.g., Figure 42).

The function only has one argument for an Oracle database user ID and one for its password. If the user's authentication credentials are different for PacHarv3 and GFFOS, specify `uid` and `pwd` as two-element vectors. The SQL servers assume a trusted relationship with the user's DFO logon identity.

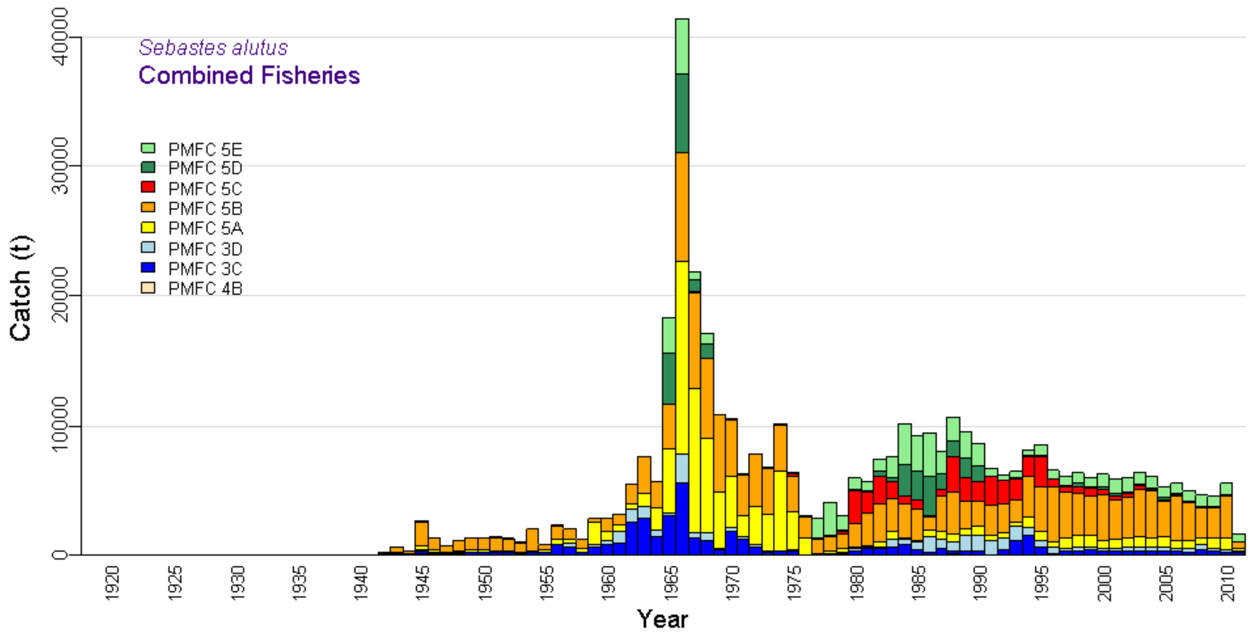


Figure 42. (*PBStools-buildCatch*) Reconstructed total (landed + discarded) catch (t) for Pacific Ocean Perch from all fisheries combined in PMFC major areas.

7.1.1 Argument details

`dbdat`	a list object of landing records from eight DFO databases on the DFBCV9TWVASP001 SQL server: (1) `ph3dat` = PacHarv3 (all fisheries) actually called HARVEST_V2_0 on the PROD Oracle server (host= `vsbciosxd76.ent.dfo-mpo.ca`, (2) `gfcdat` = GFCatch (trawl, trap, h&l), (3) `phtdat` = PacHarvest (groundfish trawl), (4) `phhdat` = PacHarvHL (Halibut bycatch from DMP validated landings), (5) `phs dat` = PacHarvSable (Sablefish trap), (6) `phhdat` = PacHarvHL (validated landings Sched II & ZN), (7) `phfdat` = PacHarvHL (fisherlog records Sched II & ZN), (8) `fosdat` = GFFOS (all fisheries).
`sql`	logical: if `TRUE` query the databases, otherwise load catch records from binary files saved from a previous query run (saves time).
`strSpp`	character string specifying the Hart species code for the rockfish to be reconstructed (`RRF`).
`dfld`	field name of the denominator in the ratio of RRF to other rockfish (usually `ORF` but can be `TRF` if total rockfish is more appropriate).
`major`	major PMFC area codes in which catch is reconstructed (usually `c(1,3:9)`).
`refyrs`	reference years to use for ratio calculations.
`fidout`	fishery IDs for which an annual series barplot stacked by PMFC area is produced: `1` = groundfish trawl, `2` = Halibut longline, `3` = Sablefish trap, `4` = Dogfish-Lingcod,

	`5` = hook and line (H&L) rockfish, `10` = all fisheries combined.
`saveinfo`	logical: if `TRUE`, save various data objects created within the function to a list object called `PBStool`; setting to `FALSE` may speed reconstruction.
`wmf`	logical: if `TRUE` send the figures to `.wmf` files.
`uid, pwd`	user ID and password for Oracle DB account authentication.
`reconstruct`	logical: if `TRUE` (default) complete the reconstruction as previously; if `FALSE`, terminate the code once the modern catch array has been compiled and saved (to a binary file `cat123mod.rda`, where `123` = species code).

7.2 plotData

Plot data components of a catch reconstruction for user inspection and diagnostic checking. The function plots `z`-values of `y` for each `x`. The default plot shows lines because a typical `x` comprises years. An alternative is to show the `z`-values as bars for each `y` grouped by `x`.

This plot is used by the catch reconstruction algorithm `buildCatch` which automatically increments the plot number `pD` that is logged in the temporary list object `PBStool`. Additionally, the catch reconstruction creates a subdirectory `CRdiag` to which `plotData` sends image files.

7.3 plotRecon

Plot reconstructed catch using barplots stacked by catch in PMFC areas. This function allows greater control over the plotting details (see Figure 42) than is currently offered by the `buildCatch` routine. Ideally, the latter should call `plotRecon`, but the implementation has not occurred yet.

Contact Information

Rowan Haigh
Research Biologist, Groundfish Section
Marine Ecosystems and Aquaculture Division
Fisheries and Oceans Canada
Pacific Biological Station, Nanaimo, BC V9T 6N7
Tel. +1 250-756-7123 fac simile +1 250-756-7053
Email rowan.haigh@dfo-mpo.gc.ca

References

Aitchison, J. 1986. The Statistical Analysis of Compositional Data. The Blackburn Press, Caldwell, NJ.

Barrie, V., Bornhold, B.D., Conway, K.W. and Luternauer, J.L. 1991. Surficial geology of the northwestern Canadian continental shelf. Cont. Shelf Res. 19S(8-10). 701–715.

Boers, N.M., Haigh, R. and Schnute, J.T. 2004. PBS Mapping 2: developer's guide. Can. Tech. Rep. Fish. Aquat. Sci. 2550. iv + 38 p.

Edwards, A.M., Starr, P.J. and Haigh, R. 2012. Stock assessment for Pacific ocean perch (*Sebastes alutus*) in Queen Charlotte Sound, British Columbia. DFO Can. Sci. Advis. Sec. Res. Doc. 2011/111. viii + 172 p.

Forrester, C.R. 1969. Results of English Sole tagging in British Columbia waters. Bull. Pac. Mar. Fish. Comm. 7. 1–10.

Haigh, R. and Yamanaka, K.L. 2011. Catch history reconstruction for rockfish (*Sebastes* spp.) caught in British Columbia coastal waters. Can. Tech. Rep. Fish. Aquat. Sci. 2943. viii + 124 p.

Kaufman, L. and Rousseeuw, P.J. 1990. Finding Groups in Data: An Introduction to Cluster Analysis. Wiley, New York, NY.

Obradovich, S.G., Yamanaka, K.L., Cooke, K., Lacko, L.C. and Dykstra, C. 2008. Summary of non-halibut catch from the standardized stock assessment survey conducted by the international pacific halibut commission in british columbia from june 4 to july 7, 2007. Can. Tech. Rep. Fish. Aquat. Sci. 2807. x + 83 p.

Orr, J.W. and Hawkins, S. 2008. Species of the rougheye rockfish complex: resurrection of *Sebastes melanostictus* (Matsubara, 1934) and a redescription of *Sebastes aleutianus* (Jordan and Evermann, 1898) (Teleostei: Scorpaeniformes). Fish. Bull. 106. 111–134.

R Core Team. 2024. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.

Schnute, J.T. 1981. A versatile growth model with statistically stable parameters. Can. J. Fish. Aquat. Sci. 38(9). 1128–1140.

Schnute, J.T., Boers, N.M. and Haigh, R. 2004*a*. PBS Mapping 2: user's guide. Can. Tech. Rep. Fish. Aquat. Sci. 2549. viii + 126 p.

Schnute, J.T., Couture-Beil, A. and Haigh, R. 2006. PBS Modelling 1: user's guide. Can. Tech. Rep. Fish. Aquat. Sci. 2674. viii + 114 p.

Schnute, J.T. and Haigh, R. 2003. A simulation model for designing groundfish trawl surveys. Can. J. Fish. Aquat. Sci. 60(6). 640–656.

Schnute, J.T. and Haigh, R. 2007. Compositional analysis of catch curve data, with an application to *Sebastes maliger*. ICES J. Mar. Sci. 64(2). 218–233.

Schnute, J.T., Haigh, R., Krishka, B.A., Sinclair, A. and Starr, P.J. 2004*b*. The British Columbia Longspine Thornyhead fishery: analysis of survey and commercial data (1996-2003). DFO Can. Sci. Advis. Sec. Res. Doc. 2004/059. iii + 75 p.

Schnute, J.T., Haigh, R., Krishka, B.A. and Starr, P.J. 2001. Pacific Ocean Perch assessment for the west coast of Canada in 2001. DFO Can. Sci. Advis. Sec. Res. Doc. 2001/138. iv + 90 p.

Sinclair, A.F., Conway, K.W. and Crawford, W.R. 2005. Associations between bathymetric, geologic and oceanographic features and the distribution of the british columbia bottom trawl fishery. The Spatial Dimension of Ecosystem Structure and Dynamics ICES CM 2005/L:25.