Package 'MQMF'

August 31, 2020

Title Modelling and Quantitative Methods in Fisheries

Version 0.1.1

Date 2020-08-27

Description Complements the book ``Using R for Modelling and Quantitative Methods in Fisheries" ISBN: 9780367469894, being published in September 2020 by Chapman & Hall in their ``Using R series". There are numerous functions and data-sets that are used in the book's many practical examples.

URL https://github.com/haddonm/MQMF

Depends R (>= 3.5.0)

License GPL-3

Encoding UTF-8

LazyData true

Imports MASS, mvtnorm

Suggests knitr, rmarkdown

RoxygenNote 7.1.1

NeedsCompilation no

Language en-US

Author Malcolm Haddon [aut, cre]

Maintainer Malcolm Haddon <malcolm.haddon@gmail.com>

Repository CRAN

Date/Publication 2020-08-31 05:50:02 UTC

R topics documented:

---

abdat *abdat A list of fishery data for blacklip abalone*

Description

A dataset of fishery data for blacklip abalone (*Haliotis rubra*) from part of the Tasmanian west coast for the years 1985 - 2008. It contains a data.frame containing the year, the catch, and the standardized CPUE from four statistical blocks of Tasmania's west coast combined. In particular, it can be used when fitting a surplus production model. Workable initial parameter values, before log-transformation might be: r= 0.4, K=9400, Binit=3400, sigma=0.05 for the Schaefer version, while these also work for the Fox model one could more efficiently use r=0.3, K=12000, Binit=4000, sigma=0.05.

Format

A data.frame of three columns

year the annual year in which the catches occurred

catch the reported landed catch in tonnes, to the nearest kilogram

cpue the standardized catch-per-unit-effort for this dive fishery

Subjects

- Surplus Production Modelling, Schaefer and Fox models
- Model fitting using maximum likelihood
- Uncertainty examples

Source

Catch data from Mundy, C. and J. McAllister (2019) Tasmanian abalone fishery assessment 2018, Institute for Marine and Antarctic Studies, University of Tasmania, 190p. ISBN: 978-1-925646-46-7. The cpue data is an unpublished early attempt at standardizing the cpue data with respect to month, block, and diver. Many more details are now included in such analyses.

Examples

```
data(abdat)
print(abdat)
oldpar <- par(no.readonly=TRUE)
plot(abdat$year,abdat$cpue, type="l",xlab="year",ylab="CPUE",
     panel.first=grid())
points(abdat$year,abdat$cpue,pch=16,cex=1.2)
par(oldpar)
```

---

addcontours *addcontours simplifies adding contours to an xy plot of points*

Description

addcontours is used to add contours to a dense plot of xy points such as might be generated when conducting an analysis of the the uncertainty associated with a stock assessment, or other analysis using a bootstrap, a Bayesian MCMC, or even using asymptotic errors and sampling from a multivariate normal distribution. addcontours first uses the kde2d function from the MASS package to translate the density of points into 2-D kernal densities, and then searches through the resulting densities for those points that would identify approximate contours. Finally it calls the contour function to add the identified approximate contours to the xy plot.

Usage

```
addcontours(
  xval,
  yval,
  xrange,
  yrange,
  ngrid = 100,
  contval = c(0.5, 0.9),
  lwd = 1,
  col = 1
)
```

Arguments

xval	the vector of x-axis values, one half of the data pairs
yval	the vector of y-axis values, the other half of the data
xrange	the range of x-axis data included in the graph
yrange	the range of y-axis data included in the graph
ngrid	the number of subdivisions by which to split the data along each axis; defaults to 100
contval	the contour values, defaults to those containing 50 and 90 percent i.e. c(0.5, 0.9)
lwd	the width of the contour lines, defaults=1
col	the col of the contour lines, default=1

Value

nothing but it does add contours to a plot of points

Examples

```
library(mvtnorm)
library(MASS)
data(abdat)
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,
               hessian=TRUE,logobs=log(abdat$cpue),
               typsize=magnitude(param),iterlim=1000)
optpar <- bestmod$estimate
vcov <- solve(bestmod$hessian)      # solve inverts matrices
columns <- c("r","K","Binit","sigma")
N <- 1000  # the contours improve as N increases; try 5000
mvnpar <- matrix(exp(rmvnorm(N,mean=optpar,sigma=vcov)),
                 nrow=N,ncol=4,dimnames=list(1:N,columns))
xv <- mvnpar[,"K"]
yv <- mvnpar[,"r"]
# plotprep(width=6,height=5,newdev = FALSE)
plot(xv,yv,type="p") # use default 0.5 and 0.9 contours.
addcontours(xv,yv,range(xv),range(yv),lwd=2,col=2)
points(mean(xv),mean(yv),pch=16,cex=1.5,col=2)
```

---

addlnorm *addlnorm estimates a log-normal distribution from output of hist*

Description

addlnorm estimates a log-normal distribution from the output of a histogram of a data set.

Usage

```
addlnorm(inhist, xdata, inc = 0.01)
```

Arguments

inhist	is the output from a call to 'hist' (see examples)
xdata	is the data that is being plotted in the histogram.
inc	defaults to a value of 0.01; is the fine grain increment used to define the normal curve. The histogram breaks should be coarse grained relative to this.

Value

a 4 x N matrix of x and y values to be used to plot the fitted normal probability density function.Combined with estimates of mean(log(indata)) and log(sd(indata))

Examples

```
oldpar <- par(no.readonly=TRUE)
egdata <- rlnorm(200,meanlog=0.075,sdlog=0.5)
outh <- hist(egdata,main="",col=2,breaks=seq(0,8,0.2))
ans <- addlnorm(outh,egdata)
lines(ans[,"x"],ans[,"y"],lwd=2,col=4)
par(oldpar)
```

---

addnorm *addnorm adds a normal distribution to a histogram of a data set.*

Description

addnorm adds a normal distribution to a histogram of a data set. This is generally to be used to illustrate whether log-transformation normalizes a set of catch or cpue data.

Usage

```
addnorm(inhist, xdata, inc = 0.01)
```

Arguments

inhist	is the output from a call to 'hist' (see examples)
xdata	is the data that is being plotted in the histogram.
inc	defaults to a value of 0.01; is the fine grain increment used to define the normal curve. The histogram breaks should be coarse grained relative to inc.

Value

a list with a vector of 'x' values and a vector of 'y' values (to be used to plot the fitted normal probability density function), and a vector called 'stats' containing the mean and standard deviation of the input data

Examples

```
oldpar <- par(no.readonly=TRUE)
x <- rnorm(1000,mean=5,sd=1)
#plotprep(height=6,width=4,newdev=FALSE)
par(mfrow= c(1,1),mai=c(0.5,0.5,0.3,0.05))
par(cex=0.75, mgp=c(1.5,0.35,0), font.axis=7)
outH <- hist(x,breaks=25,col=3,main="")
nline <- addnorm(outH,x)
lines(nline$x,nline$y,lwd=3,col=2)
print(nline$stats)
par(oldpar)
```

aicbic *aicbic returns the AIC and BIC for a given model*

Description

aicbic calculates and returns the AIC and BIC using the standard definitions. It defaults to assuming that negative log- likelihoods have been used in the model fitting, but provides the option of having used SSQ (set nLL to FALSE). If using SSQ it uses Burnham and Anderson's (2002) definition but sets BIC to NA. aicbic can recognize the outputs from optim, nlm, and nlminb.

Usage

```
aicbic(model, dat, nLL = TRUE)
```

Arguments

model	the optimum model fitted by either optim, nlm, or nlminb
dat	the data set used in the modelling, or just n the number of observations; it can distinguish between them
nLL	uses negative log-likelihood? default=TRUE

Value

a vector of four numbers, AIC first, then BIC, then negLL or SSQ, depending on nLL, then number of parameters p

References

Burnham, K.P. and D.R. Anderson (2002) *Model Selection and Inference. A Practical Information-Theoretic Approach.* Second Edition Springer-Verlag, New York. 488 p.

Examples

```
data(blackisland); bi <- blackisland
param <- c(Linf=170.0,K=0.3,sigma=4.0)
modelvb <- nlm(f=negNLL,p=param,funk=fabens,observed=bi$dl,indat=bi,
               initL="l1",delT="dt") # could have used the defaults
aicbic(modelvb,blackisland)  # 588.3382 596.3846 291.1691   3
```

altnegLL *altnegLL calculate the Normal negative log-likelihood*

Description

altnegLL calculates negLLM using the simplification from Haddon (2011) using the ssq calculated within the function spm

Usage

```
altnegLL(inp, indat)
```

Arguments

inp	a vector of model parameters (r,K,Binit)
indat	a matrix with at least columns 'year', 'catch', and 'cpue'

Value

a single value, the negative log-likelihood

Examples

```
data(dataspm)
pars <- log(c(r=0.2,K=6000,Binit=2800,sigma=0.2))
ans <- fitSPM(pars,fish=dataspm,schaefer=TRUE,maxiter=1000)
outfit(ans)
altnegLL(ans$estimate,dataspm) # should be -12.12879
```

bce *bce Baranov catch equation*

Description

bce the Baranov Catch Equation. The total mortality of fish in an exploited population is made up of fish being killed by fishing and others dying naturally. We use the bce to estimate the catch (those killed by fishing). The bce has value because some fish that would be expected to die naturally can be expected to be caught and killed by fishing so estimating the catch is slightly more complex than numbers of fish available times the harvest rate, Nt x Ht. It is invariably better to use the Baranov Catch Equation when calculating the expected catches.

Usage

```
bce(M, Fat, Nt, ages)
```

Arguments

M	instantaneous rate of natural mortality, assumed constant
Fat	a vector of age-specific instantaneous rates of fishing mortality over the time period t, this is usually estimated by multiplying the fully selected F by the selectivity/availability at age.
Nt	The population numbers-at-age at the start of time t
ages	the ages 0:maxage used in the calculations

Value

a matrix of surviving numbers-at age, total mortality-at-age, and catch-at-age

Examples

```
age <- 0:25
Ft <- -log(1 - 0.2) # harvest rate of 0.2
Faa <- rep(Ft,length(age))
M <- 0.12
Nt <- 1000
bce(M,Fat=Faa,Nt,ages=age)   # should give 188.8862
```

---

bh *bh represents one version of Beverton-Holt recruitment*

Description

bh implements the Beverton-Holt stock recruitment equation R = aB/(b + B), where R is the recruitment, a and b are the parameters and B is the spawning biomass. a is the maximum recruitment level and b is the biomass required to generate 0.5 x maximum recruitment

Usage

```
bh(p, B)
```

Arguments

p	a vector of the a and b parameters
B	a vector, possibly of length 1, of spawning biomass levels

Value

a vector, the same length as B, of the predicted recruitment(s)

Examples

```
B <- 1:3000
rec <- bh(c(1000,200),B)
plot1(B,rec,xlab="SpB",ylab="Recruitment",lwd=2)
```

blackisland *blackisland tagging data from a blacklip abalone population*

Description

A 108 x 4 data.frame containing dt, the time in years between tagging and recapture, l1 the shell length at tagging, and l2, the length at recapture, with the growth increment, dl as the last column. This data can be used to estimate the growth characteristics of abalone from the Black Island site, which is on the south west coast of Tasmania, Australia. The mean time interval between tagging and recapture is 1 year and 1 week, 1.02 years, which reflects the practical problems of taking a vessel around the bottom of Tasmania, where it is essential to wait on suitable weather for such sub-tidal field work.

Format

A data.frame of four columns

dt the time between tagging and recapture, in years

l1 the shell length when tagged in mm

l2 the shell length at recapture in mm

dl the growth increment between tagging and recapture in mm; there are zero values.

Subjects

- Tagging data
- Estimation of individual growth
- Binomial likelihoods
- Faben's version of the von Bertalanffy curve

Source

Thanks to Dr Craig Mundy and the abalone team at the Institute of Marine and Antarctic Studies, of the University of Tasmania for the use of this data.

Examples

```
data(blackisland)
print(head(blackisland,20))
oldpar <- par(no.readonly=TRUE)
plot(blackisland$l1,blackisland$dl,type="p",pch=16,
xlab="Initial Length mm",ylab="Growth Increment mm",
panel.first=grid())
abline(h=0)
par(oldpar)
```

bracket *bracket bounds a value on the x-axis and y-axis*

Description

bracket scans through a series of predicted values for the location of a target value of the y-axis and returns the two y values that bracket the target, perhaps finding the values closest to 0.5 in a vector between 0 and 1. It also returns the x-axis values that gave rise to the two values bracketing the target, and finally returns the target. For example, imagine generating the proportion mature for a given length of fish using an equation for which there was no analytical solution to what the value of the L50 or the inter-quartile distance was. Bracket can find the two lengths that generate proportions just below 0.5 and just above. It does not matter if, by chance, the target is one of those y-axis values.

Usage

```
bracket(x, yaxis, xaxis)
```

Arguments

x	the target predicted value of interest
yaxis	the predicted values reflecting the xaxis values
xaxis	the series of values used to generate the predicted values

Value

a vector of 5 values, left, right, bottom, top and target

See Also

linter

Examples

```
L = seq(60,160,1)
p=c(a=0.075,b=0.075,c=1.0,alpha=100)
asym <- srug(p=p,sizeage=L) # Schnute and Richards unified growth curve
L25 <- linter(bracket(0.25,asym,L))
L50 <- linter(bracket(0.5,asym,L))
L75 <- linter(bracket(0.75,asym,L))
ans <- c(L25,L50,L75,L50-L25,L75-L50)
{cat("   L25    L50      L75   L50-L25 L75-L50 \n")
cat(round(ans,4),"\n")}
```

calcprior *calcprior return the sum of a vector of constant values as priors*

Description

calcprior is used to include a prior probability into Bayesian calculations. calcprior is a template for generating such priors. The default given here is to return a constant small number for the prior probability, it needs to sum to 1.0 across the replicates returned by do_MCMC. If non-uniform priors are required write a different function and in do_MCMC point priorcalc at it. Whatever function you define needs to have the same input parameters as this calcprior, i.e. the parameters and N. If something else if required then do_MCMC will need modification in the two places where priorcalc is used. Alternatively, the ellipsis, ..., might be used.

Usage

```
calcprior(pars, N)
```

Arguments

pars	the parameters of the model being examined by the MCMC
N	the number of replicate parameter vectors to be returned from do_MCMC, remember to include the burn-in replicates

Value

the sum of a vector of small constant values to act as priors.

Examples

```
param <- log(c(0.4,9400,3400,0.05))
calcprior(pars=param,N=20000)  # should give -39.61395
```

chapter2 *chapter2 The 15 R-code chunks from A Non-Introduction to R*

Description

chapter2 contains no active function but rather acts as a repository for the various example code chunks found in chapter2. There are 15 r-code chunks in chapter2.

Examples

```
## Not run:
# All the example code from  # A Non-Introduction to R
### Using Functions

# R-chunk 1  Page 18
#make a function called countones2, don't overwrite original

countones2 <- function(x) return(length(which(x == 1)))  # or
countones3 <- function(x) return(length(x[x == 1]))
vect <- c(1,2,3,1,2,3,1,2,3)  # there are three ones
countones2(vect)  # should both give the answer: 3
countones3(vect)
set.seed(7100809) # if repeatability is desirable.
matdat <- matrix(trunc(runif(40)*10),nrow=5,ncol=8)
matdat #a five by eight matrix of random numbers between 0 - 9
apply(matdat,2,countones3)  # apply countones3 to 8 columns
apply(matdat,1,countones3)  # apply countones3 to 5 rows


# R-chunk 2 Page 19
 #A more complex function prepares to plot a single base graphic
 #It has the syntax for opening a window outside of Rstudio and
 #defining a base graphic. It includes oldpar<-par(no.readonly=TRUE)
 #which is returned invisibly so that the original 'par' settings
 #can be recovered using par(oldpar) after completion of your plot.

plotprep2 <- function(plots=c(1,1),width=6, height=3.75,usefont=7,
                      newdev=TRUE) {
  if ((names(dev.cur()) %in% c("null device","RStudioGD")) &
      (newdev)) {
    dev.new(width=width,height=height,noRStudioGD = TRUE)
  }
  oldpar <- par(no.readonly=TRUE)  # not in the book's example
  par(mfrow=plots,mai=c(0.45,0.45,0.1,0.05),oma=c(0,0,0,0))
  par(cex=0.75,mgp=c(1.35,0.35,0),font.axis=usefont,font=usefont,
      font.lab=usefont)
  return(invisible(oldpar))
}  #  see ?plotprep; see also parsyn() and parset()


### Random Number Generation
# R-chunk 3 pages 20 - 21
#Examine the use of random seeds.

seed <- getseed()  # you will very likely get different naswers
set.seed(seed)
round(rnorm(5),5)
set.seed(123456)
round(rnorm(5),5)
set.seed(seed)
round(rnorm(5),5)
```

```
### Plotting in R
# R-chunk 4  page 22
#library(MQMF)   # The development of a simple graph  see Fig. 2.1
#The statements below open the RStudio graphics window, but opening
#a separate graphics window using plotprep is sometimes clearer.

data("LatA")  #LatA = length at age data; try properties(LatA)
#plotprep(width=6.0,height=5.0,newdev=FALSE) #unhash for external plot
oldpar <- par(no.readonly=TRUE)  # not in the book's example
setpalette("R4") #a more balanced, default palette see its help
par(mfrow=c(2,2),mai=c(0.45,0.45,0.1,0.05))  # see ?parsyn
par(cex=0.75, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
hist(LatA$age) #examine effect of different input parameters
hist(LatA$age,breaks=20,col=3,main="") # 3=green #try ?hist
hist(LatA$age,breaks=30,main="",col=4) # 4=blue
hist(LatA$age, breaks=30,col=2, main="", xlim=c(0,43), #2=red
     xlab="Age (years)",ylab="Count")
par(oldpar)  # not in the book's example

### Dealing with Factors
# R-chunk 5  pages 23 - 24
#Dealing with factors/categories can be tricky

DepCat <- as.factor(rep(seq(300,600,50),2)); DepCat
try(5 * DepCat[3], silent=FALSE) #only returns NA and a warning!
as.numeric(DepCat) # returns the levels not the original values
as.numeric(levels(DepCat)) #converts 7 levels not the replicates
DepCat <- as.numeric(levels(DepCat))[DepCat] # try ?facttonum
#converts replicates in DepCat to numbers, not just the levels
5 * DepCat[3]   # now treat DepCat as numeric
DepCat <- as.factor(rep(seq(300,600,50),2)); DepCat
facttonum(DepCat)

## Writing Functions
# R-chunk 6  page 25
#Outline of a function's structure

functionname <- function(argument1, fun,...) {
  # body of the function
  #
  # the input arguments and body of a function can include other
  # functions, which may have their own arguments, which is what
  # the ... is for. One can include other inputs that are used but
  # not defined early on and may depend on what function is brought
  # into the main function. See for example negLL(), and others
  answer <- fun(argument1) + 2
  return(answer)
} # end of functionname
functionname(c(1,2,3,4,5),mean)  # = mean(1,2,3,4,5)= 3 + 2 = 5

### Simple Functions
# R-chunk 7  page 26
```

```
# Implement the von Bertalanffy curve in multiple ways

ages <- 1:20
nages <- length(ages)
Linf <- 50;  K <- 0.2;  t0 <- -0.75
# first try a for loop to calculate length for each age
loopLt <- numeric(nages)
for (ag in ages) loopLt[ag] <- Linf * (1 - exp(-K * (ag - t0)))
# the equations are automatically vectorized so more efficient
vecLt <- Linf * (1 - exp(-K * (ages - t0))) # or we can convert
# the equation into a function and use it again and again
vB <- function(pars,inages) { # requires pars=c(Linf,K,t0)
  Lt <- pars[1] * (1 - exp(-pars[2] * (inages - pars[3])))
  return(Lt)
}
funLt <- vB(c(Linf,K,t0),ages)
ans <- cbind(ages,funLt,vecLt,loopLt)


# R-chunk 8  page 26 - code not shown in book.
# Tabulate the ans from chunk 7

library(knitr)  # needed for the function knitr - pretty tables
kable(halftable(ans,yearcol="ages",subdiv=2),digits=c(0,3,3,3,0,3,3,3))


# R-chunk 9  page 27
#A vB function with some input error checking

vB <- function(pars,inages) { # requires pars=c(Linf,K,t0)
  if (is.numeric(pars) & is.numeric(inages)) {
    Lt <- pars[1] * (1 - exp(-pars[2] * (inages - pars[3])))
  } else { stop(cat("Not all input values are numeric! \n")) }
  return(Lt)
}
param <- c(50, 0.2,"-0.75")
funLt <- vB(as.numeric(param),ages) #try without the as.numeric
halftable(cbind(ages,funLt))


### Scoping of Objects
# R-chunk 10   page 29
# demonstration that the globel environment is 'visible' inside a
# a function it calls, but the function's environment remains
# invisible to the global or calling environment

vBscope <- function(pars) { # requires pars=c(Linf,K,t0)
  rhside <- (1 - exp(-pars[2] * (ages - pars[3])))
  Lt <- pars[1] * rhside
  return(Lt)
}
ages <- 1:10; param <- c(50,0.2,-0.75)
vBscope(param)
```

```
try(rhside)    # note the use of try() which can trap errors ?try


### Function Inputs and Outputs
# R-chunk 11  page 30
#Bring the data-set schaef into the working of global environment

data(schaef)


# R-chunk 12   page 30  Table 2.2 code not shown
#Tabulate the data held in schaef. Needs knitr

kable(halftable(schaef,yearcol="year",subdiv=2),digits=c(0,0,0,4))

# R-chunk 13  page 30
#examine the properties of the data-set schaef

class(schaef)
a <- schaef[1:5,2]
b <- schaef[1:5,"catch"]
c <- schaef$catch[1:5]
cbind(a,b,c)
mschaef <- as.matrix(schaef)
mschaef[1:5,"catch"]  # ok
d <- try(mschaef$catch[1:5]) #invalid for matrices
d  # had we not used try()eveerything would have stopped.


# R-chunk 14  page 31
#Convert column names of a data.frame or matrix to lowercase

dolittle <- function(indat) {
  indat1 <- as.data.frame(indat)
  colnames(indat) <- tolower(colnames(indat))
  return(list(dfdata=indat1,indat=as.matrix(indat)))
} # return the original and the new version
colnames(schaef) <- toupper(colnames(schaef))
out <- dolittle(schaef)
str(out, width=63, strict.width="cut")


# R-chunk 15  page 32
#Could have used an S3 plot method had we defined a class   Fig.2.2

plotspmdat(schaef) # examine the code as an eg of a custom plot

## End(Not run)
```

Description

chapter3 is not an active function but rather acts as a repository for the various example code chunks found in chapter3. There are 27 r-code chunks in chapter3.

Examples

```
## Not run:
### The Discrete Logistic Model
# R-chunk 1 page 36
# Code to produce Figure 3.1. Note the two one-line functions

surprod <- function(Nt,r,K) return((r*Nt)*(1-(Nt/K)))
densdep <- function(Nt,K) return((1-(Nt/K)))
r <- 1.2; K <- 1000.0; Nt <- seq(10,1000,10)
oldpar <- par(no.readonly=TRUE) # this line not in book
# plotprep(width=7, height=5, newdev=FALSE)
par(mfrow=c(2,1),mai=c(0.4,0.4,0.05,0.05),oma=c(0.0,0,0.0,0.0))
par(cex=0.75, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
plot1(Nt,surprod(Nt,r,K),xlab="Population Nt",defpar=FALSE,
      ylab="Production")
plot1(Nt,densdep(Nt,K),xlab="Population Nt",defpar=FALSE,
      ylab="Density-Dependence")
par(oldpar)  # this line not in book


### Dynamic Behaviour
# R-chunk 2 page 38
#Code for Figure 3.2. Try varying the value of rv from 0.5-2.8

yrs <- 100; rv=2.8;  Kv <- 1000.0; Nz=100; catch=0.0; p=1.0
ans <- discretelogistic(r=rv,K=Kv,N0=Nz,Ct=catch,Yrs=yrs,p=p)
avcatch <- mean(ans[(yrs-50):yrs,"nt"],na.rm=TRUE) #used in text
label <- paste0("r=",rv," K=",Kv," Ct=",catch, " N0=",Nz," p=",p=p)
oldpar <- par(no.readonly=TRUE) # this line not in book
plot(ans, main=label, cex=0.9, font=7) #Schaefer dynamics
par(oldpar)   # this line not in book


# R-chunk 3 page 39
#run discrete logistic dynamics for 600 years

yrs=600
ans <- discretelogistic(r=2.55,K=1000.0,N0=100,Ct=0.0,Yrs=yrs)

# R-chunk 4  page 40, code not in the book
#tabulate the last 30 years of the dynamics   needs knitr
library(knitr)
kable(halftable(ans[(yrs-29):yrs,],yearcol="year",subdiv=3),digits=c(0,1,1,0,1,1,0,1,1))


### Finding Boundaries between Behaviours.
# R-chunk 5 page 40
```

```
#run discretelogistic and search for repeated values of Nt

yrs <- 600
ans <- discretelogistic(r=2.55,K=1000.0,N0=100,Ct=0.0,Yrs=yrs)
avt <- round(apply(ans[(yrs-100):(yrs-1),2:3],1,mean),2)
count <- table(avt)
count[count > 1] # with r=2.55 you should find an 8-cycle limit

# R-chunk 6  page 41
#searches for unique solutions given an r value  see Table 3.2

testseq <- seq(1.9,2.59,0.01)
nseq <- length(testseq)
result <- matrix(0,nrow=nseq,ncol=2,
                 dimnames=list(testseq,c("r","Unique")))
yrs <- 600
for (i in 1:nseq) {  # i = 31
  rval <- testseq[i]
  ans <- discretelogistic(r=rval,K=1000.0,N0=100,Ct=0.0,Yrs=yrs)
  ans <- ans[-yrs,] # remove last year, see str(ans) for why
  ans[,"nt1"] <- round(ans[,"nt1"],3) #try hashing this out
  result[i,] <- c(rval,length(unique(tail(ans[,"nt1"],100))))
}


# R-chunk 7  page 41 - 42, Table 3.2. Code not in the book.
#unique repeated Nt values 100 = non-equilibrium or chaos

kable(halftable(result,yearcol = "r"),)


### Classical Bifurcation Diagram of Chaos
# R-chunk 8 pages 42 - 43
#the R code for the bifurcation function

bifurcation <- function(testseq,taill=100,yrs=1000,limy=0,incx=0.001){
  nseq <- length(testseq)
  result <- matrix(0,nrow=nseq,ncol=2,
                   dimnames=list(testseq,c("r","Unique Values")))
  result2 <- matrix(NA,nrow=nseq,ncol=taill)
  for (i in 1:nseq) {
    rval <- testseq[i]
    ans <- discretelogistic(r=rval,K=1000.0,N0=100,Ct=0.0,Yrs=yrs)
    ans[,"nt1"] <- round(ans[,"nt1"],4)
    result[i,] <- c(rval,length(unique(tail(ans[,"nt1"],taill))))
    result2[i,] <- tail(ans[,"nt1"],taill)
  }
  if (limy[1] == 0) limy <- c(0,getmax(result2,mult=1.02))

  oldpar <- parset() #plot taill values against taill of each r value
  on.exit(par(oldpar))    #  this line not in book
  plot(rep(testseq[1],taill),result2[1,],type="p",pch=16,cex=0.1,
       ylim=limy,xlim=c(min(testseq)*(1-incx),max(testseq)*(1+incx)),
```

```
      xlab="r value",yaxs="i",xaxs="i",ylab="Equilibrium Numbers",
      panel.first=grid())
  for (i in 2:nseq)
    points(rep(testseq[i],taill),result2[i,],pch=16,cex=0.1)
  return(invisible(list(result=result,result2=result2)))
} # end of bifurcation


# R-chunk 9 page 43
#Alternative r value arrangements for you to try; Fig 3.3
#testseq <- seq(2.847,2.855,0.00001) #hash/unhash as needed
#bifurcation(testseq,limy=c(600,740),incx=0.0001) # t
#testseq <- seq(2.6225,2.6375,0.00001) # then explore
#bifurcation(testseq,limy=c(660,730),incx=0.0001)

testseq <- seq(1.9,2.975,0.0005) # modify to explore
bifurcation(testseq,limy=0)


### The Effect of Fishing on Dynamics
# R-chunk 10  page 43 - 44.
#Effect of catches on stability properties of discretelogistic

yrs=50; Kval=1000.0
nocatch <- discretelogistic(r=2.56,K=Kval,N0=500,Ct=0,Yrs=yrs)
catch50 <- discretelogistic(r=2.56,K=Kval,N0=500,Ct=50,Yrs=yrs)
catch200 <- discretelogistic(r=2.56,K=Kval,N0=500,Ct=200,Yrs=yrs)
catch300 <- discretelogistic(r=2.56,K=Kval,N0=500,Ct=300,Yrs=yrs)


# R-chunk 11 page 45
#Effect of different catches on n-cyclic behaviour Fig3.4

plottime <- function(x,ylab) {
  yrs <- nrow(x)
  plot1(x[,"year"],x[,"nt"],ylab=ylab,defpar=FALSE)
  avB <- round(mean(x[(yrs-40):yrs,"nt"],na.rm=TRUE),3)
  mtext(avB,side=1,outer=F,line=-1.1,font=7,cex=1.0)
} # end of plottime
#the oma argument is used to adjust the space around the graph
oldpar <- par(no.readonly=TRUE) # this line not in book
par(mfrow=c(2,2),mai=c(0.25,0.4,0.05,0.05),oma=c(1.0,0,0.25,0))
par(cex=0.75, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
plottime(nocatch,"Catch = 0")
plottime(catch50,"Catch = 50")
plottime(catch200,"Catch = 200")
plottime(catch300,"Catch = 300")
mtext("years",side=1,outer=TRUE,line=-0.2,font=7,cex=1.0)
par(oldpar)

# R-chunk 12 page 46
#Phase plot for Schaefer model Fig 3.5
```

```
plotphase <- function(x,label,ymax=0) { #x from discretelogistic
  yrs <- nrow(x)
  colnames(x) <- tolower(colnames(x))
  if (ymax[1] == 0) ymax <- getmax(x[,c(2:3)])
  plot(x[,"nt"],x[,"nt1"],type="p",pch=16,cex=1.0,ylim=c(0,ymax),
       yaxs="i",xlim=c(0,ymax),xaxs="i",ylab="nt1",xlab="",
       panel.first=grid(),col="darkgrey")
  begin <- trunc(yrs * 0.6) #last 40% of yrs = 20, when yrs=50
  points(x[begin:yrs,"nt"],x[begin:yrs,"nt1"],pch=18,col=1,cex=1.2)
  mtext(label,side=1,outer=F,line=-1.1,font=7,cex=1.2)
} # end of plotphase
oldpar <- par(no.readonly=TRUE) # this line not in book
par(mfrow=c(2,2),mai=c(0.25,0.25,0.05,0.05),oma=c(1.0,1.0,0,0))
par(cex=0.75, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
plotphase(nocatch,"Catch = 0",ymax=1300)
plotphase(catch50,"Catch = 50",ymax=1300)
plotphase(catch200,"Catch = 200",ymax=1300)
plotphase(catch300,"Catch = 300",ymax=1300)
mtext("nt",side=1,outer=T,line=0.0,font=7,cex=1.0)
mtext("nt+1",side=2,outer=T,line=0.0,font=7,cex=1.0)
par(oldpar)    # this line not in book

### Determinism
## Age-Structured Modelling Concepts
### Survivorship in a Cohort
# R-chunk 13 pages 48 - 49
#Exponential population declines under different Z. Fig 3.6

yrs <- 50;  yrs1 <- yrs + 1 # to leave room for B[0]
years <- seq(0,yrs,1)
B0 <- 1000        # now alternative total mortality rates
Z <- c(0.05,0.1,0.2,0.4,0.55)
nZ <- length(Z)
Bt <- matrix(0,nrow=yrs1,ncol=nZ,dimnames=list(years,Z))
Bt[1,] <- B0
for (j in 1:nZ) for (i in 2:yrs1) Bt[i,j] <- Bt[(i-1),j]*exp(-Z[j])
oldp <- plot1(years,Bt[,1],xlab="Years",ylab="Population Size",lwd=2)
if (nZ > 1) for (j in 2:nZ) lines(years,Bt[,j],lwd=2,col=j,lty=j)
legend("topright",legend=paste0("Z = ",Z),col=1:nZ,lwd=3,
       bty="n",cex=1,lty=1:5)
par(oldp)  # this line not in book

### Instantaneous vs Annual Mortality Rates
# R-chunk 14 page 51
#Prepare matrix of harvest rate vs time to appoximate F

Z <- -log(0.5)
timediv <- c(2,4,12,52,365,730,2920,8760,525600)
yrfrac <- 1/timediv
names(yrfrac) <- c("6mth","3mth","1mth","1wk","1d","12h","3h","1h","1m")
nfrac <- length(yrfrac)
columns <- c("yrfrac","divisor","yrfracH","Remain")
result <- matrix(0,nrow=nfrac,ncol=length(columns),
```

```
                dimnames=list(names(yrfrac),columns))
for (i in 1:nfrac) {
  timestepmort <- Z/timediv[i]
  N <- 1000
  for (j in 1:timediv[i]) N <- N * (1-timestepmort)
  result[i,] <- c(yrfrac[i],timediv[i],timestepmort,N)
}


# R-chunk 15 page 51 Table 3.3, code not shown in book
#output of constant Z for shorter and shorter periods

kable(result,digits=c(10,0,8,4))


# R-chunk 16 page 51
#Annual harvest rate against instantaneous F, Fig 3.7

Fi <- seq(0.001,2,0.001)
H <- 1 - exp(-Fi)
oldpar <- parset()  # a wrapper for simplifying defining the par values
plot(Fi,H,type="l",lwd=2,panel.first=grid(),xlab="Instantaneous Fishing Mortality F",
     ylab="Annual Proportion Mortality H")
lines(c(0,1),c(0,1),lwd=2,lty=2,col=2)
par(oldpar)   # this line not in book


## Simple Yield per Recruit
# R-chunk 17  page 53
# Simple Yield-per-Recruit see Russell (1942)

age <- 1:11;  nage <- length(age); N0 <- 1000  # some definitions
# weight-at-age values
WaA <- c(NA,0.082,0.175,0.283,0.4,0.523,0.7,0.85,0.925,0.99,1.0)
# now the harvest rates
H <- c(0.01,0.06,0.11,0.16,0.21,0.26,0.31,0.36,0.55,0.8)
nH <- length(H)
NaA <- matrix(0,nrow=nage,ncol=nH,dimnames=list(age,H)) # storage
CatchN <- NaA;  CatchW <- NaA      # define some storage matrices
for (i in 1:nH) {                # loop through the harvest rates
  NaA[1,i] <- N0  # start each harvest rate with initial numbers
  for (age in 2:nage) {  # loop through over-simplified dynamics
    NaA[age,i] <- NaA[(age-1),i] * (1 - H[i])
    CatchN[age,i] <- NaA[(age-1),i] - NaA[age,i]
  }
  CatchW[,i] <- CatchN[,i] * WaA
}                      # transpose the vector of total catches to
totC <- t(colSums(CatchW,na.rm=TRUE))   # simplify later printing


# R-chunk 18 page 54 Table 3.4 code not shown in book
#Tabulate numbers-at-age for different harvest rates  needs knitr
```

```
kable(NaA,digits=c(0,0,0,0,0,0,0,0,1,1),row.names=TRUE)


# R-chunk 19 page 54, Table 3.5, code not shown in book.
#Tabulate Weight-at-age for different harvest rates

kable(CatchW[2:11,],digits=c(2,2,2,2,2,2,2,2,2,2),row.names=TRUE)


# R-chunk 20 page 54, Table 3.6, code not shown in book.
#Total weights vs Harvest rate

kable(totC,digits=c(1,1,1,1,1,1,1,1,1,1))


# R-chunk 21 page 55
#Use MQMF::plot1 for a quick plot of the total catches. Figure 3.8

oldpar <- plot1(H,totC,xlab="Harvest Rate",ylab="Total Yield",lwd=2)
par(oldpar) # to reset the par values if desired

### Selectivity in Yield-per-Recruit
# R-chunk 22 Page 56
#Logistic S shaped cureve for maturity

ages <- seq(0,50,1)
sel1 <- mature(-3.650425,0.146017,sizeage=ages) #-3.65/0.146=25
sel2 <- mature(-6,0.2,ages)
sel3 <- mature(-6,0.24,ages)
oldp <- plot1(ages,sel1,xlab="Age Yrs",ylab="Selectivity",cex=0.75,lwd=2)
lines(ages,sel2,col=2,lwd=2,lty=2)
lines(ages,sel3,col=3,lwd=2,lty=3)
abline(v=25,col="grey",lty=2)
abline(h=c(0.25,0.5,0.75),col="grey",lty=2)
legend("topleft",c("25_15.04","30_10.986","25_9.155"),col=c(1,2,3),
       lwd=3,cex=1.1,bty="n",lty=1:3)
par(oldp)

### The Baranov Catch Equation
# R-chunk 23 Page 58
# Baranov catch equation

age <- 0:12;  nage <- length(age)
sa <-mature(-4,2,age) #selectivity-at-age
H <- 0.2;  M <- 0.35
FF <- -log(1 - H)#Fully selected instantaneous fishing mortality
Ft <- sa * FF     # instantaneous Fishing mortality-at-age
N0 <- 1000
out <- cbind(bce(M,Ft,N0,age),"Select"=sa)  # out becomes Table 3.7


# R-chunk 24 page 59, Table 3.7, code not shown in book.
#tabulate output from Baranov Catch Equations
```

```
kable(out,digits=c(3,3,3,3))


### Growth and Weight-at-Age
## Full Yield-per-Recruit
# R-chunk 25 Page 60 - 61
# A more complete YPR analysis

age <- 0:20;  nage <- length(age) #storage vectors and matrices
laa <- vB(c(50.0,0.25,-1.5),age) # length-at-age
WaA <- (0.015 * laa ^ 3.0)/1000  # weight-at-age as kg
H <- seq(0.01,0.65,0.05);  nH <- length(H)
FF <- round(-log(1 - H),5)  # Fully selected fishing mortality
N0 <- 1000
M <- 0.1
numt <- matrix(0,nrow=nage,ncol=nH,dimnames=list(age,FF))
catchN <- matrix(0,nrow=nage,ncol=nH,dimnames=list(age,FF))
as50 <- c(1,2,3)
yield <- matrix(0,nrow=nH,ncol=length(as50),dimnames=list(H,as50))
for (sel in 1:length(as50)) {
  sa <- logist(as50[sel],1.0,age)  # selectivity-at-age
  for (harv in 1:nH) {
    Ft <- sa * FF[harv]      # Fishing mortality-at-age
    out <- bce(M,Ft,N0,age)
    numt[,harv] <- out[,"Nt"]
    catchN[,harv] <- out[,"Catch"]
    yield[harv,sel] <- sum(out[,"Catch"] * WaA,na.rm=TRUE)
  } # end of harv loop
} # end of sel loop


# R-chunk 26  Page 61
#A full YPR analysis  Figure 3.10

oldp <- plot1(H,yield[,3],xlab="Harvest Rate",ylab="Yield",cex=0.75,lwd=2)
lines(H,yield[,2],lwd=2,col=2,lty=2)
lines(H,yield[,1],lwd=2,col=3,lty=3)
legend("bottomright",legend=as50,col=c(3,2,1),lwd=3,bty="n",
       cex=1.0,lty=c(3,2,1))
par(oldp)

# R-chunk 27 page 62, Table 3.8, code not shown in book.
#Tabulate yield-per-recruit using Baranoc catch equation

kable(yield,digits=c(2,3,3,3))


## End(Not run)
```

---

chapter4 *chapter4 The 47 R-code chunks from Model Parameter Estimation*

Description

chapter4 is not an active function but rather acts as a repository for the various example code chunks found in chapter4. There are 47 r-code chunks in chapter3.

Examples

```
## Not run:
# All the example code from  # Model Parameter Estimation
# Model Parameter Estimation
## Introduction
### Optimization
## Criteria of Best Fit
## Model Fitting in R
### Model Requirements
### A Length-at-Age Example
### Alternative Models of Growth
## Sum of Squared Residual Deviations
### Assumptions of Least-Squares
### Numerical Solutions

# R-chunk 1  Page 75
#setup optimization using growth and ssq
#convert equations 4.4 to 4.6 into vectorized R functions
#These will over-write the same functions in the MQMF package

data(LatA)      # try ?LatA   assumes library(MQMF) already run
vB <- function(p, ages) return(p[1]*(1-exp(-p[2]*(ages-p[3]))))
Gz <- function(p, ages) return(p[1]*exp(-p[2]*exp(p[3]*ages)))
mm <- function(p, ages) return((p[1]*ages)/(p[2] + ages^p[3]))
#specific function to calc ssq. The ssq within MQMF is more
ssq <- function(p,funk,agedata,observed) {        #general and is
  predval <- funk(p,agedata)        #not limited to p and agedata
  return(sum((observed - predval)^2,na.rm=TRUE))
} #end of ssq
# guess starting values for Linf, K, and t0, names not needed
pars <- c("Linf"=27.0,"K"=0.15,"t0"=-2.0) #ssq should=1478.449
ssq(p=pars, funk=vB, agedata=LatA$age, observed=LatA$length)
# try misspelling LatA$Length with a capital. What happens?

### Passing Functions as Arguments to other Functions
# R-chunk 2  Page 76
# Illustrates use of names within function arguments

vB <- function(p,ages) return(p[1]*(1-exp(-p[2] *(ages-p[3]))))
ssq <- function(funk,observed,...) { # only define ssq arguments
  predval <- funk(...) # funks arguments are implicit
  return(sum((observed - predval)^2,na.rm=TRUE))
} # end of ssq
pars <- c("Linf"=27.0,"K"=0.15,"t0"=-2.0) # ssq should = 1478.449
ssq(p=pars, funk=vB, ages=LatA$age, observed=LatA$length) #if no
ssq(vB,LatA$length,pars,LatA$age) # name order is now vital!
```

```
# R-chunk 3  Page 77
# Illustrate a problem with calling a function in a function
# LatA$age is typed as LatA$Age but no error, and result = 0

ssq(funk=vB, observed=LatA$length, p=pars, ages=LatA$Age) # !!!

### Fitting the Models
# R-chunk 4  Page 77
#plot the LatA data set   Figure 4.2

oldpar <- parset()   # parset and getmax are two MQMF functions
ymax <- getmax(LatA$length) # simplifies use of base graphics. For
# full colour, with the rgb as set-up below, there must be >= 5 obs
plot(LatA$age,LatA$length,type="p",pch=16,cex=1.2,xlab="Age Years",
     ylab="Length cm",col=rgb(1,0,0,1/5),ylim=c(0,ymax),yaxs="i",
     xlim=c(0,44),panel.first=grid())
par(oldpar) # this line not in book

# R-chunk 5  Pages 78 - 79
# use nlm to fit 3 growth curves to LatA, only p and funk change

ages <- 1:max(LatA$age) # used in comparisons
pars <- c(27.0,0.15,-2.0) # von Bertalanffy
bestvB <- nlm(f=ssq,funk=vB,observed=LatA$length,p=pars,
              ages=LatA$age,typsize=magnitude(pars))
outfit(bestvB,backtran=FALSE,title="vB"); cat("\n")
pars <- c(26.0,0.7,-0.5) # Gompertz
bestGz <- nlm(f=ssq,funk=Gz,observed=LatA$length,p=pars,
              ages=LatA$age,typsize=magnitude(pars))
outfit(bestGz,backtran=FALSE,title="Gz"); cat("\n")
pars <- c(26.2,1.0,1.0) # Michaelis-Menton - first start point
bestMM1 <- nlm(f=ssq,funk=mm,observed=LatA$length,p=pars,
               ages=LatA$age,typsize=magnitude(pars))
outfit(bestMM1,backtran=FALSE,title="MM"); cat("\n")
pars <- c(23.0,1.0,1.0) # Michaelis-Menton - second start point
bestMM2 <- nlm(f=ssq,funk=mm,observed=LatA$length,p=pars,
               ages=LatA$age,typsize=magnitude(pars))
outfit(bestMM2,backtran=FALSE,title="MM2"); cat("\n")

# R-chunk 6  Page 81
#The use of args() and formals()

args(nlm) # formals(nlm) uses more screen space. Try yourself.


# R-chunk 7  Page 81, code not in the book
#replacement for args(nlm) to keep within page borders without truncation

{cat("function (f, p, ..., hessian = FALSE, typsize = rep(1,\n")
  cat("  length(p)),fscale = 1, print.level = 0, ndigit = 12, \n")
  cat("  gradtol = 1e-06, stepmax = max(1000 * \n")
  cat("  sqrt(sum((p/typsize)^2)), 1000), steptol = 1e-06, \n")
```

```
  cat("  iterlim = 100, check.analyticals = TRUE)\n")}


# R-chunk 8  Pages 81 - 82
#Female length-at-age + 3 growth fitted curves Figure 4.3

predvB <- vB(bestvB$estimate,ages) #get optimumpredicted lengths
predGz <- Gz(bestGz$estimate,ages) # using the outputs
predmm <- mm(bestMM2$estimate,ages) #from the nlm analysis above
ymax <- getmax(LatA$length) #try ?getmax or getmax [no brackets]
xmax <- getmax(LatA$age)  #there is also a getmin, not used here
oldpar <- parset(font=7) #or use parsyn() to prompt for par syntax
plot(LatA$age,LatA$length,type="p",pch=16, col=rgb(1,0,0,1/5),
     cex=1.2,xlim=c(0,xmax),ylim=c(0,ymax),yaxs="i",xlab="Age",
     ylab="Length (cm)",panel.first=grid())
lines(ages,predvB,lwd=2,col=4)        # vB    col=4=blue
lines(ages,predGz,lwd=2,col=1,lty=2)  # Gompertz  1=black
lines(ages,predmm,lwd=2,col=3,lty=3)  # MM        3=green
#notice the legend function and its syntax.
legend("bottomright",cex=1.2,c("von Bertalanffy","Gompertz",
                  "Michaelis-Menton"),col=c(4,1,3),lty=c(1,2,3),lwd=3,bty="n")
par(oldpar)  # this line not in book

### Objective Model Selection
### The Influence of Residual Error Choice on Model Fit
# R-chunk 9  Page 84 - 85
# von Bertalanffy

pars <- c(27.25,0.15,-3.0)
bestvBN <- nlm(f=ssq,funk=vB,observed=LatA$length,p=pars,
               ages=LatA$age,typsize=magnitude(pars),iterlim=1000)
outfit(bestvBN,backtran=FALSE,title="Normal errors"); cat("\n")
# modify ssq to account for log-normal errors in ssqL
ssqL <- function(funk,observed,...) {
  predval <- funk(...)
  return(sum((log(observed) - log(predval))^2,na.rm=TRUE))
} # end of ssqL
bestvBLN <- nlm(f=ssqL,funk=vB,observed=LatA$length,p=pars,
                ages=LatA$age,typsize=magnitude(pars),iterlim=1000)
outfit(bestvBLN,backtran=FALSE,title="Log-Normal errors")


# R-chunk 10  Pages 85 - 86
# Now plot the resultibng two curves and the data Fig 4.4

predvBN <- vB(bestvBN$estimate,ages)
predvBLN <- vB(bestvBLN$estimate,ages)
ymax <- getmax(LatA$length)
xmax <- getmax(LatA$age)
oldpar <- parset()
plot(LatA$age,LatA$length,type="p",pch=16, col=rgb(1,0,0,1/5),
     cex=1.2,xlim=c(0,xmax),ylim=c(0,ymax),yaxs="i",xlab="Age",
     ylab="Length (cm)",panel.first=grid())
```

```
lines(ages,predvBN,lwd=2,col=4,lty=2)   # add Normal dashed
lines(ages,predvBLN,lwd=2,col=1)        # add Log-Normal solid
legend("bottomright",c("Normal Errors","Log-Normal Errors"),
       col=c(4,1),lty=c(2,1),lwd=3,bty="n",cex=1.2)
par(oldpar)

### Remarks on Initial Model Fitting
## Maximum Likelihood
### Introductory Examples
# R-chunk 11  Page 88
# Illustrate Normal random likelihoods. see Table 4.1

set.seed(12345)       # make the use of random numbers repeatable
x <- rnorm(10,mean=5.0,sd=1.0)      # pseudo-randomly generate 10
avx <- mean(x)                      # normally distributed values
sdx <- sd(x)          # estimate the mean and stdev of the sample
L1 <- dnorm(x,mean=5.0,sd=1.0)   # obtain likelihoods, L1, L2 for
L2 <- dnorm(x,mean=avx,sd=sdx)    # each data point for both sets
result <- cbind(x,L1,L2,"L2gtL1"=(L2>L1))      # which is larger?
result <- rbind(result,c(NA,prod(L1),prod(L2),1)) # result+totals
rownames(result) <- c(1:10,"product")
colnames(result) <- c("x","original","estimated","est > orig")


# R-chunk 12  page 88, Table 4.1, code not shown in book.
#tabulate results of Normal Likelihoods

kable(result,digits=c(4,8,8,0),row.names = TRUE, caption='(ref:tab401)')

# R-chunk 13  Page 89
# some examples of pnorm, dnorm, and qnorm, all mean = 0

cat("x = 0.0        Likelihood =",dnorm(0.0,mean=0,sd=1),"\n")
cat("x = 1.95996395 Likelihood =",dnorm(1.95996395,mean=0,sd=1),"\n")
cat("x =-1.95996395 Likelihood =",dnorm(-1.95996395,mean=0,sd=1),"\n")
# 0.5 = half cumulative distribution
cat("x = 0.0        cdf = ",pnorm(0,mean=0,sd=1),"\n")
cat("x = 0.6744899  cdf = ",pnorm(0.6744899,mean=0,sd=1),"\n")
cat("x = 0.75       Quantile =",qnorm(0.75),"\n") # reverse pnorm
cat("x = 1.95996395 cdf = ",pnorm(1.95996395,mean=0,sd=1),"\n")
cat("x =-1.95996395 cdf = ",pnorm(-1.95996395,mean=0,sd=1),"\n")
cat("x = 0.975      Quantile =",qnorm(0.975),"\n") # expect ~1.96
# try x <- seq(-5,5,0.2); round(dnorm(x,mean=0.0,sd=1.0),5)

## Likelihoods from the Normal Distribution
# R-chunk 14   Page 90
# Density plot and cumulative distribution for Normal   Fig 4.5

x <- seq(-5,5,0.1)  # a sequence of values around a mean of 0.0
NL <- dnorm(x,mean=0,sd=1.0)   # normal likelihoods for each X
CD <- pnorm(x,mean=0,sd=1.0)   # cumulative density vs X
oldp <- plot1(x,CD,xlab="x = StDev from Mean",ylab="Likelihood and CDF")
lines(x,NL,lwd=3,col=2,lty=3) # dashed line as these are points
```

```
abline(h=0.5,col=4,lwd=1)
par(oldp)

# R-chunk 15  Pages 91 - 92
#function facilitates exploring different polygons Fig 4.6

plotpoly <- function(mid,delta,av=5.0,stdev=1.0) {
  neg <- mid-delta;  pos <- mid+delta
  pdval <- dnorm(c(mid,neg,pos),mean=av,sd=stdev)
  polygon(c(neg,neg,mid,neg),c(pdval[2],pdval[1],pdval[1],
                               pdval[2]),col=rgb(0.25,0.25,0.25,0.5))
  polygon(c(pos,pos,mid,pos),c(pdval[1],pdval[3],pdval[1],
                               pdval[1]),col=rgb(0,1,0,0.5))
  polygon(c(mid,neg,neg,mid,mid),
          c(0,0,pdval[1],pdval[1],0),lwd=2,lty=1,border=2)
  polygon(c(mid,pos,pos,mid,mid),
          c(0,0,pdval[1],pdval[1],0),lwd=2,lty=1,border=2)
  text(3.395,0.025,paste0("~",round((2*(delta*pdval[1])),7)),
       cex=1.1,pos=4)
  return(2*(delta*pdval[1])) # approx probability, see below
} # end of plotpoly, a temporary function to enable flexibility
#This code can be re-run with different values for delta
x <- seq(3.4,3.6,0.05) # where under the normal curve to examine
pd <- dnorm(x,mean=5.0,sd=1.0) #prob density for each X value
mid <- mean(x)
delta <- 0.05  # how wide either side of the sample mean to go?
oldpar <- parset() #pre-defined MQMF base graphics set-up for par
ymax <- getmax(pd) # find maximum y value for the plot
plot(x,pd,type="l",xlab="Variable x",ylab="Probability Density",
     ylim=c(0,ymax),yaxs="i",lwd=2,panel.first=grid())
approxprob <- plotpoly(mid,delta)  #use function defined above
par(oldpar)  # this line not in the book

### Equivalence with Sum-of-Squares
### Fitting a Model to Data using Normal Likelihoods
# R-chunk 16  Page 94
#plot of length-at-age data  Fig 4.7

data(LatA) # load the redfish data set into memory and plot it
ages <- LatA$age;  lengths <- LatA$length
oldpar <- plot1(ages,lengths,xlab="Age",ylab="Length",type="p",cex=0.8,
      pch=16,col=rgb(1,0,0,1/5))
par(oldpar)


# R-chunk 17  Page 95
# Fit the vB growth curve using maximum likelihood

pars <- c(Linf=27.0,K=0.15,t0=-3.0,sigma=2.5) # starting values
# note, estimate for sigma is required for maximum likelihood
ansvB <- nlm(f=negNLL,p=pars,funk=vB,observed=lengths,ages=ages,
             typsize=magnitude(pars))
outfit(ansvB,backtran=FALSE,title="vB by minimum -veLL")
```

```
# R-chunk 18 Page 96
#Now fit the Michaelis-Menton curve

pars <- c(a=23.0,b=1.0,c=1.0,sigma=3.0) # Michaelis-Menton
ansMM <- nlm(f=negNLL,p=pars,funk=mm,observed=lengths,ages=ages,
             typsize=magnitude(pars))
outfit(ansMM,backtran=FALSE,title="MM by minimum -veLL")

# R-chunk 19  Page 96
#plot optimum solutions for vB and mm. Fig 4.8

Age <- 1:max(ages) # used in comparisons
predvB <- vB(ansvB$estimate,Age) #optimum solution
predMM <- mm(ansMM$estimate,Age) #optimum solution
oldpar <- parset()               # plot the deata points first
plot(ages,lengths,xlab="Age",ylab="Length",type="p",pch=16,
     ylim=c(10,33),panel.first=grid(),col=rgb(1,0,0,1/3))
lines(Age,predvB,lwd=2,col=4)     # then add the growth curves
lines(Age,predMM,lwd=2,col=1,lty=2)
legend("bottomright",c("von Bertalanffy","Michaelis-Menton"),
       col=c(4,1),lwd=3,bty="n",cex=1.2,lty=c(1,2))
par(oldpar)

# R-chunk 20   Pages 96 - 97
# residual plot for vB curve   Fig 4.9

predvB <- vB(ansvB$estimate,ages) # predicted values for age data
resids <- lengths - predvB               # calculate vB residuals
oldpar <- plot1(ages,resids,type="p",col=rgb(1,0,0,1/3),
           xlim=c(0,43),pch=16,xlab="Ages Years",ylab="Residuals")
abline(h=0.0,col=1,lty=2)    # emphasize the zero line
par(oldpar)

## Log-Normal Likelihoods
### Simplification of Log-Normal Likelihoods
### Log-Normal Properties
# R-chunk 21  Page 100
# meanlog and sdlog affects on mode and spread of lognormal Fig 4.10

x <- seq(0.05,5.0,0.01)  # values must be greater than 0.0
y <- dlnorm(x,meanlog=0,sdlog=1.2,log=FALSE) #dlnorm=likelihoods
y2 <- dlnorm(x,meanlog=0,sdlog=1.0,log=FALSE)#from log-normal
y3 <- dlnorm(x,meanlog=0,sdlog=0.6,log=FALSE)#distribution
y4 <- dlnorm(x,0.75,0.6)         #log=TRUE = log-likelihoods
oldpar <- parset(plots=c(1,2)) #MQMF base plot formatting function
plot(x,y3,type="l",lwd=2,panel.first=grid(),
     ylab="Log-Normal Likelihood")
lines(x,y,lwd=2,col=2,lty=2)
lines(x,y2,lwd=2,col=3,lty=3)
lines(x,y4,lwd=2,col=4,lty=4)
legend("topright",c("meanlog sdlog","    0.0      0.6    0.0",
                    "      1.0","    0.0      1.2","    0.75    0.6"),
```

```
       col=c(0,1,3,2,4),lwd=3,bty="n",cex=1.0,lty=c(0,1,3,2,4))
plot(log(x),y3,type="l",lwd=2,panel.first=grid(),ylab="")
lines(log(x),y,lwd=2,col=2,lty=2)
lines(log(x),y2,lwd=2,col=3,lty=3)
lines(log(x),y4,lwd=2,col=4,lty=4)
par(oldpar)  # return par to old settings; this line not in book


# R-chunk 22  Pages 100 - 101

set.seed(12354) # plot random log-normal numbers as Fig 4.11
meanL <- 0.7;   sdL <- 0.5  # generate 5000 random log-normal
x <- rlnorm(5000,meanlog = meanL,sdlog = sdL) # values
oldpar <- parset(plots=c(1,2)) # simplifies plots par() definition
hist(x[x < 8.0],breaks=seq(0,8,0.25),col=0,main="")
meanx <- mean(log(x)); sdx <- sd(log(x))
outstat <- c(exp(meanx-(sdx^2)),exp(meanx),exp(meanx+(sdx^2)/2))
abline(v=outstat,col=c(4,1,2),lwd=3,lty=c(1,2,3))
legend("topright",c("mode","median","bias-correct"),
       col=c(4,1,2),lwd=3,bty="n",cex=1.2,lty=c(1,2,3))
outh <- hist(log(x),breaks=30,col=0,main="")   # approxnormal
hans <- addnorm(outh,log(x)) #MQMF function; try  ?addnorm
lines(hans$x,hans$y,lwd=3,col=1) # type addnorm into the console
par(oldpar)  # return par to old settings; this line not in book


# R-chunk 23  Page 101
#examine log-normal propoerties. It is a bad idea to reuse

set.seed(12345) #'random' seeds, use getseed() for suggestions
meanL <- 0.7;   sdL <- 0.5  #5000 random log-normal values then
x <- rlnorm(5000,meanlog = meanL,sdlog = sdL) #try with only 500
meanx <- mean(log(x)); sdx <- sd(log(x))
cat("               Original  Sample \n")
cat("Mode(x)     = ",exp(meanL - sdL^2),outstat[1],"\n")
cat("Median(x)   = ",exp(meanL),outstat[2],"\n")
cat("Mean(x)     = ",exp(meanL + (sdL^2)/2),outstat[3],"\n")
cat("Mean(log(x) =  0.7     ",meanx,"\n")
cat("sd(log(x)   =  0.5     ",sdx,"\n")

### Fitting a Curve using Log-Normal Likelihoods
# R-chunk 24  Page 103
# fit a Beverton-Holt recruitment curve to tigers data Table 4.2

data(tigers)   # use the tiger prawn data set
lbh <- function(p,biom) return(log((p[1]*biom)/(p[2] + biom)))
#note we are returning the log of Beverton-Holt recruitment
pars <- c("a"=25,"b"=4.5,"sigma"=0.4)   # includes a sigma
best <- nlm(negNLL,pars,funk=lbh,observed=log(tigers$Recruit),
            biom=tigers$Spawn,typsize=magnitude(pars))
outfit(best,backtran=FALSE,title="Beverton-Holt Recruitment")
predR <- exp(lbh(best$estimate,tigers$Spawn))
#note exp(lbh(...)) is the median because no bias adjustment
```

```
result <- cbind(tigers,predR,tigers$Recruit/predR)

# R-chunk 25  Page 103
# Fig 4.12 visual examination of the fit to the tigers data

oldp <- plot1(tigers$Spawn,predR,xlab="Spawning Biomass","Recruitment",
      maxy=getmax(c(predR,tigers$Recruit)),lwd=2)
points(tigers$Spawn,tigers$Recruit,pch=16,cex=1.1,col=2)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 26  page 104, Table 4.12, code not shown in book.
#tabulating observed, predicted and residual recruitment

colnames(result) <- c("SpawnB","Recruit","PredR","Residual")
kable(result,digits=c(1,1,3,4), caption='(ref:tab402)')

### Fitting a Dynamic Model using Log-Normal Errors
# R-chunk 27  Page 106

data(abdat)  # plot abdat fishery data using a MQMF helper  Fig 4.13
plotspmdat(abdat) # function to quickly plot catch and cpue


# R-chunk 28  Pages 106 - 107
# Use log-transformed parameters for increased stability when
# fitting the surplus production model to the abdat data-set

param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
obslog <- log(abdat$cpue) #input log-transformed observed data
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=as.matrix(abdat),
               logobs=obslog)  # no typsize, or iterlim needed
#backtransform estimates, outfit's default, as log-transformed
outfit(bestmod,backtran = TRUE,title="abdat")        # in param


# R-chunk 29  Pages 107 - 108
# Fig 4.14 Examine fit of predicted to data

predce <- simpspm(bestmod$estimate,abdat) #compare obs vs pred
ymax <- getmax(c(predce,obslog))
oldp <- plot1(abdat$year,obslog,type="p",maxy=ymax,ylab="Log(CPUE)",
      xlab="Year",cex=0.9)
lines(abdat$year,predce,lwd=2,col=2)
par(oldp)  # return par to old settings; this line not in book


## Likelihoods from the Binomial Distribution
### An Example using Binomial Likelihoods
# R-chunk 30  Page 109
#Use Binomial distribution to test biased sex-ratio Fig 4.15

n <- 60    # a sample of 60 animals
p <- 0.5   # assume a sex-ration of 1:1
```

```
m <- 1:60  # how likely is each of the 60 possibilites?
binom <- dbinom(m,n,p)   # get individual likelihoods
cumbin <- pbinom(m,n,p)  # get cumulative distribution
oldp <- plot1(m,binom,type="h",xlab="Number of Males",ylab="Probability")
abline(v=which.closest(0.025,cumbin),col=2,lwd=2) # lower 95% CI
par(oldp)  # return par to old settings; this line not in book


# R-chunk 31  Page 111
# plot relative likelihood of different p values Fig 4.16

n <- 60  # sample size; should really plot points as each independent
m <- 20  # number of successes = finding a male
p <- seq(0.1,0.6,0.001) #range of probability we find a male
lik <- dbinom(m,n,p)    # R function for binomial likelihoods
oldp <- plot1(p,lik,type="l",xlab="Prob. of 20 Males",ylab="Prob.")
abline(v=p[which.max(lik)],col=2,lwd=2) # try "p" instead of "l"
par(oldp)  # return par to old settings; this line not in book

# R-chunk 32  Page 111
# find best estimate using optimize to finely search an interval

n <- 60; m <- 20  # trials and successes
p <- c(0.1,0.6) #range of probability we find a male
optimize(function(p) {dbinom(m,n,p)},interval=p,maximum=TRUE)

### Open Bay Juvenile Fur Seal Population Size
# R-chunk 33  Page 112
# Juvenile furseal data-set Greaves, 1992.  Table 4.3

furseal <- c(32,222,1020,704,1337,161.53,31,181,859,593,1125,
             135.72,29,185,936,634,1238,153.99)
columns <- c("tagged(m)","Sample(n)","Population(X)",
             "95%Lower","95%Upper","StErr")
furs <- matrix(furseal,nrow=3,ncol=6,dimnames=list(NULL,columns),
               byrow=TRUE)
#tabulate fur seal data Table 4.3
kable(furs, caption='(ref:tab403)')

# R-chunk 34  Pages 113 - 114
# analyse two pup counts 32 from 222, and 31 from 181, rows 1-2 in
# Table 4.3.   Now set-up storage for solutions

optsol <- matrix(0,nrow=2,ncol=2,
                 dimnames=list(furs[1:2,2],c("p","Likelihood")))
X <- seq(525,1850,1) # range of potential population sizes
p <- 151/X  #range of proportion tagged; 151 originally tagged
m <- furs[1,1] + 1 #tags observed, with Bailey's adjustment
n <- furs[1,2] + 1 # sample size with Bailey's adjustment
lik1 <- dbinom(m,n,p) # individaul likelihoods
#find best estimate with optimize to finely search an interval
#use unlist to convert the output list into a vector
#Note use of Bailey's adjustment (m+1), (n+1) Caughley, (1977)
```

```
optsol[1,] <- unlist(optimize(function(p) {dbinom(m,n,p)},p,
                              maximum=TRUE))
m <- furs[2,1]+1;  n <- furs[2,2]+1 #repeat for sample2
lik2 <- dbinom(m,n,p)
totlik <- lik1 * lik2 #Joint likelihood of 2 vectors
optsol[2,] <- unlist(optimize(function(p) {dbinom(m,n,p)},p,
                              maximum=TRUE))

# R-chunk 35  Page 114
# Compare outcome for 2 independent seal estimates Fig 4.17
# Should plot points not a line as each are independent

oldp <- plot1(X,lik1,type="l",xlab="Total Pup Numbers",
      ylab="Probability",maxy=0.085,lwd=2)
abline(v=X[which.max(lik1)],col=1,lwd=1)
lines(X,lik2,lwd=2,col=2,lty=3)  # add line to plot
abline(v=X[which.max(lik2)],col=2,lwd=1) # add optimum
#given p = 151/X, then X = 151/p and p = optimum proportion
legend("topright",legend=round((151/optsol[,"p"])),col=c(1,2),lwd=3,
       bty="n",cex=1.1,lty=c(1,3))
par(oldp)  # return par to old settings; this line not in book

### Using Multiple Independent Samples

# R-chunk 36  Pages 114 - 115
#Combined likelihood from 2 independent samples Fig 4.18

totlik <- totlik/sum(totlik) # rescale so the total sums to one
cumlik <- cumsum(totlik) #approx cumulative likelihood for CI
oldp <- plot1(X,totlik,type="l",lwd=2,xlab="Total Pup Numbers",
      ylab="Posterior Joint Probability")
percs <- c(X[which.closest(0.025,cumlik)],X[which.max(totlik)],
           X[which.closest(0.975,cumlik)])
abline(v=percs,lwd=c(1,2,1),col=c(2,1,2))
legend("topright",legend=percs,lwd=c(2,4,2),bty="n",col=c(2,1,2),
       cex=1.2)  # now compare with averaged count
m <- furs[3,1];  n <- furs[3,2] # likelihoods for the
lik3 <- dbinom(m,n,p)            # average of six samples
lik4 <- lik3/sum(lik3)  # rescale for comparison with totlik
lines(X,lik4,lwd=2,col=3,lty=2) #add 6 sample average to plot
par(oldp)  # return par to old settings; this line not in book

### Analytical Approaches
## Other Distributions
## Likelihoods from the Multinomial Distribution
### Using the Multinomial Distribution
# R-chunk 37  Page 119
#plot counts x shell-length of 2 cohorts   Figure 4.19

cw <- 2  # 2 mm size classes, of which mids are the centers
mids <- seq(8,54,cw) #each size class = 2 mm as in 7-9, 9-11, ...
obs <- c(0,0,6,12,35,40,29,23,13,7,10,14,11,16,11,11,9,8,5,2,0,0,0,0)
# data from (Helidoniotis and Haddon, 2012)
```

```
dat <- as.matrix(cbind(mids,obs)) #xy matrix needed by inthist
oldp <- parset()  #set up par declaration then use an MQMF function
inthist(dat,col=2,border=3,width=1.8, #histogram of integers
        xlabel="Shell Length mm",ylabel="Frequency",xmin=7,xmax=55)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 38  Page 121
#cohort data with 2 guess-timated normal curves Fig 4.20

oldp <- parset()  # set up the required par declaration
inthist(dat,col=0,border=8,width=1.8,xlabel="Shell Length mm",
        ylabel="Frequency",xmin=7,xmax=55,lwd=2)  # MQMF function
#Guess normal parameters and plot those curves on histogram
av <- c(18.0,34.5)    # the initial trial and error means and
stdev <- c(2.75,5.75)  # their standard deviations
prop1 <- 0.55       #  proportion of observations in cohort 1
n <- sum(obs) #262 observations, now calculate expected counts
cohort1 <- (n*prop1*cw)*dnorm(mids,av[1],stdev[1]) # for each
cohort2 <- (n*(1-prop1)*cw)*dnorm(mids,av[2],stdev[2])# cohort
#(n*prop1*cw) scales likelihoods to suit the 2mm class width
lines(mids,cohort1,lwd=2,col=1)
lines(mids,cohort2,lwd=2,col=4)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 39 Page 122
#wrapper function for calculating the multinomial log-likelihoods
#using predfreq and mnnegLL, Use ? and examine their code

wrapper <- function(pars,obs,sizecl,midval=TRUE) {
  freqf <- predfreq(pars,sum(obs),sizecl=sizecl,midval=midval)
  return(mnnegLL(obs,freqf))
} # end of wrapper which uses MQMF::predfreq and MQMF::mnnegLL
mids <- seq(8,54,2) # each size class = 2 mm as in 7-9, 9-11, ...
av <- c(18.0,34.5)   # the trial and error means and
stdev <- c(2.95,5.75)  # standard deviations
phi1 <- 0.55      # proportion of observations in cohort 1
pars <-c(av,stdev,phi1)  # combine parameters into a vector
wrapper(pars,obs=obs,sizecl=mids) # calculate total -veLL

# R-chunk 40  Page 122
# First use the midpoints

bestmod <- nlm(f=wrapper,p=pars,obs=obs,sizecl=mids,midval=TRUE,
               typsize=magnitude(pars))
outfit(bestmod,backtran=FALSE,title="Using Midpts"); cat("\n")
#Now use the size class bounds and cumulative distribution
#more sensitive to starting values, so use best pars from midpoints
X <- seq((mids[1]-cw/2),(tail(mids,1)+cw/2),cw)
bestmodb <- nlm(f=wrapper,p=bestmod$estimate,obs=obs,sizecl=X,
                midval=FALSE,typsize=magnitude(pars))
outfit(bestmodb,backtran=FALSE,title="Using size-class bounds")

# R-chunk 41 Page 123
```

```
#prepare the predicted Normal distribution curves

pars <- bestmod$estimate # best estimate using mid-points
cohort1 <- (n*pars[5]*cw)*dnorm(mids,pars[1],pars[3])
cohort2 <- (n*(1-pars[5])*cw)*dnorm(mids,pars[2],pars[4])
parsb <- bestmodb$estimate # best estimate with bounds
nedge <- length(mids) + 1  # one extra estimate
cump1 <- (n*pars[5])*pnorm(X,pars[1],pars[3])#no need to rescale
cohort1b <- (cump1[2:nedge] - cump1[1:(nedge-1)])
cump2 <- (n*(1-pars[5]))*pnorm(X,pars[2],pars[4])  # cohort 2
cohort2b <- (cump2[2:nedge] - cump2[1:(nedge-1)])

# R-chunk 42  Page 123
#plot the alternate model fits to cohorts  Fig 4.21

oldp <- parset() #set up required par declaration; then plot curves
pick <- which(mids < 28)
inthist(dat[pick,],col=0,border=8,width=1.8,xmin=5,xmax=28,
        xlabel="Shell Length mm",ylabel="Frequency",lwd=3)
lines(mids,cohort1,lwd=3,col=1,lty=2) # have used setpalette("R4")
lines(mids,cohort1b,lwd=2,col=4)      # add the bounded results
label <- c("midpoints","bounds")      # very minor differences
legend("topleft",legend=label,lwd=3,col=c(1,4),bty="n",
       cex=1.2,lty=c(2,1))
par(oldp)  # return par to old settings; this line not in book

# R-chunk 43  Page 124
# setup table of results for comparison of fitting strategies

predmid <- rowSums(cbind(cohort1,cohort2))
predbnd <- rowSums(cbind(cohort1b,cohort2b))
result <- as.matrix(cbind(mids,obs,predmid,predbnd,predbnd-predmid))
colnames(result) <- c("mids","Obs","Predmid","Predbnd","Difference")
result <- rbind(result,c(NA,colSums(result,na.rm=TRUE)[2:5]))


# R-chunk 44  page 125, Table 4.4, code not shown in book.
#tabulate the results of fitting cohort data  in two ways

kable(result,digits=c(0,0,4,4,4),align=c("r","r","r","r","r"))

## Likelihoods from the Gamma Distribution

# R-chunk 45  Pages 126 - 127
#Illustrate different Gamma function curves  Figure 4.22

X <- seq(0.0,10,0.1) #now try different shapes and scale values
dg <- dgamma(X,shape=1,scale=1)
oldp <- plot1(X,dg,xlab = "Quantile","Probability Density")
lines(X,dgamma(X,shape=1.5,scale=1),lwd=2,col=2,lty=2)
lines(X,dgamma(X,shape=2,scale=1),lwd=2,col=3,lty=3)
lines(X,dgamma(X,shape=4,scale=1),lwd=2,col=4,lty=4)
legend("topright",legend=c("Shape 1","Shape 1.5","Shape 2",
```

```
                            "Shape 4"),lwd=3,col=c(1,2,3,4),bty="n",cex=1.25,lty=1:4)
mtext("Scale c = 1",side=3,outer=FALSE,line=-1.1,cex=1.0,font=7)
par(oldp)  # return par to old settings; this line not in book

## Likelihoods from the Beta Distribution
# R-chunk 46  Pages 127 - 128
#Illustrate different Beta function curves. Figure 4.23

x <- seq(0, 1, length = 1000)
oldp <- parset()
plot(x,dbeta(x,shape1=3,shape2=1),type="l",lwd=2,ylim=c(0,4),
     yaxs="i",panel.first=grid(), xlab="Variable 0 - 1",
     ylab="Beta Probability Density - Scale1 = 3")
bval <- c(1.25,2,4,10)
for (i in 1:length(bval))
  lines(x,dbeta(x,shape1=3,shape2=bval[i]),lwd=2,col=(i+1),lty=c(i+1))
legend(0.5,3.95,c(1.0,bval),col=c(1:7),lwd=2,bty="n",lty=1:5)
par(oldp)  # return par to old settings; this line not in book


## Bayes' Theorem
### Introduction
### Bayesian Methods
### Prior Probabilities
# R-chunk 47  Pages 132 - 133
# can prior probabilities ever be uniniformative?  Figure 4.24

x <- 1:1000
y <- rep(1/1000,1000)
cumy <- cumsum(y)
group <- sort(rep(c(1:50),20))
xlab <- seq(10,990,20)
oldp <- par(no.readonly=TRUE)  # this line not in book
par(mfrow=c(2,1),mai=c(0.45,0.3,0.05,0.05),oma=c(0.0,1.0,0.0,0.0))
par(cex=0.75, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
yval <- tapply(y,group,sum)
plot(x,cumy,type="p",pch=16,cex=0.5,panel.first=grid(),
     xlim=c(0,1000),ylim=c(0,1),ylab="",xlab="Linear Scale")
plot(log(x),cumy,type="p",pch=16,cex=0.5,panel.first=grid(),
     xlim=c(0,7),xlab="Logarithmic Scale",ylab="")
mtext("Cumulative Probability",side=2,outer=TRUE,cex=0.9,font=7)
par(oldp)  # return par to old settings; this line not in book

## End(Not run)
```

Description

chapter5 is not an active function but rather acts as a repository for the various example code chunks found in chapter5. There are 23 r-code chunks in chapter5. You should, of course, feel free to use and modify any of these example chunks in your own work.

Examples

```
## Not run:
# All the example code from  # Static Models
# Static Models
## Introduction
## Productivity Parameters
## Growth
### Seasonal Growth Curves

# R-chunk 1  Page 138
 #vB growth curve fit to Pitcher and Macdonald derived seasonal data

data(minnow); week <- minnow$week; length <- minnow$length
pars <- c(75,0.1,-10.0,3.5); label=c("Linf","K","t0","sigma")
bestvB <- nlm(f=negNLL,p=pars,funk=vB,ages=week,observed=length,
              typsize=magnitude(pars))
predL <- vB(bestvB$estimate,0:160)
outfit(bestvB,backtran = FALSE,title="Non-Seasonal vB",parnames=label)

# R-chunk 2  Page 139
 #plot the non-seasonal fit and its residuals.  Figure 5.1

oldp <- parset(plots=c(2,1),margin=c(0.35,0.45,0.02,0.05))
plot1(week,length,type="p",cex=1.0,col=2,xlab="Weeks",pch=16,
      ylab="Length (mm)",defpar=FALSE)
lines(0:160,predL,lwd=2,col=1)
 # calculate and plot the residuals
resids <- length - vB(bestvB$estimate,week)
plot1(week,resids,type="l",col="darkgrey",cex=0.9,lwd=2,
    xlab="Weeks",lty=3,ylab="Normal Residuals",defpar=FALSE)
points(week,resids,pch=16,cex=1.1,col="red")
abline(h=0,col=1,lwd=1)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 3  Pages 139 - 140
 # Fit seasonal vB curve, parameters = Linf, K, t0, C, s, sigma

svb <- function(p,ages,inc=52) {
  return(p[1]*(1 - exp(-(p[4] * sin(2*pi*(ages - p[5])/inc) +
                           p[2] * (ages - p[3])))))
} # end of svB
spars <- c(bestvB$estimate[1:3],0.1,5,2.0)  # keep sigma at end
bestsvb <- nlm(f=negNLL,p=spars,funk=svb,ages=week,observed=length,
              typsize=magnitude(spars))
predLs <- svb(bestsvb$estimate,0:160)
outfit(bestsvb,backtran = FALSE,title="Seasonal Growth",
```

```
       parnames=c("Linf","K","t0","C","s","sigma"))

# R-chunk 4  Page 140
 #Plot seasonal growth curve and residuals   Figure 5.2

oldp <- parset(plots=c(2,1))  # MQMF utility wrapper function
plot1(week,length,type="p",cex=0.9,col=2,xlab="Weeks",pch=16,
      ylab="Length (mm)",defpar=FALSE)
lines(0:160,predLs,lwd=2,col=1)
 # calculate and plot the residuals
resids <- length - svb(bestsvb$estimate,week)
plot1(week,resids,type="l",col="darkgrey",cex=0.9,xlab="Weeks",
      lty=3,ylab="Normal Residuals",defpar=FALSE)
points(week,resids,pch=16,cex=1.1,col="red")
abline(h=0,col=1,lwd=1)
par(oldp)  # return par to old settings; this line not in book

### Fabens Method with Tagging Data
# R-chunk 5  Pages 142 - 143
 # tagging growth increment data from Black Island, Tasmania

data(blackisland);  bi <- blackisland # just to keep things brief
oldp <- parset()
plot(bi$l1,bi$dl,type="p",pch=16,cex=1.0,col=2,ylim=c(-1,33),
     ylab="Growth Increment mm",xlab="Initial Length mm",
     panel.first = grid())
abline(h=0,col=1)
par(oldp)  # return par to old settings; this line not in book

### Fitting Models to Tagging Data
# R-chunk 6  Page 144
 # Fit the vB and Inverse Logistic to the tagging data

linm <- lm(bi$dl ~ bi$l1) # simple linear regression
param <- c(170.0,0.3,4.0); label <- c("Linf","K","sigma")
modelvb <- nlm(f=negNLL,p=param,funk=fabens,observed=bi$dl,indat=bi,
               initL="l1",delT="dt") # could have used the defaults
outfit(modelvb,backtran = FALSE,title="vB",parnames=label)
predvB <- fabens(modelvb$estimate,bi)
cat("\n")
param2 <- c(25.0,130.0,35.0,3.0)
label2=c("MaxDL","L50","delta","sigma")
modelil <- nlm(f=negNLL,p=param2,funk=invl,observed=bi$dl,indat=bi,
               initL="l1",delT="dt")
outfit(modelil,backtran = FALSE,title="IL",parnames=label2)
predil <- invl(modelil$estimate,bi)

# R-chunk 7  Page 145
 #growth curves and regression fitted to tagging data Fig 5.4

oldp <- parset(margin=c(0.4,0.4,0.05,0.05))
plot(bi$l1,bi$dl,type="p",pch=16,cex=1.0,col=3,ylim=c(-2,31),
     ylab="Growth Increment mm",xlab="Length mm",panel.first=grid())
```

```
abline(h=0,col=1)
lines(bi$l1,predvB,pch=16,col=1,lwd=3,lty=1)  # vB
lines(bi$l1,predil,pch=16,col=2,lwd=3,lty=2)  # IL
abline(linm,lwd=3,col=7,lty=2) # add dashed linear regression
legend("topright",c("vB","LinReg","IL"),lwd=3,bty="n",cex=1.2,
                   col=c(1,7,2),lty=c(1,2,2))
par(oldp)  # return par to old settings; this line not in book

# R-chunk 8  Pages 145 - 146
 #residuals for vB and inverse logistic for tagging data Fig 5.5

oldp <- parset(plots=c(1,2),outmargin=c(1,1,0,0),margin=c(.25,.25,.05,.05))
plot(bi$l1,(bi$dl - predvB),type="p",pch=16,col=1,ylab="",
     xlab="",panel.first=grid(),ylim=c(-8,11))
abline(h=0,col=1)
mtext("vB",side=1,outer=FALSE,line=-1.1,cex=1.2,font=7)
plot(bi$l1,(bi$dl - predil),type="p",pch=16,col=1,ylab="",
     xlab="",panel.first=grid(),ylim=c(-8,11))
abline(h=0,col=1)
mtext("IL",side=3,outer=FALSE,line=-1.2,cex=1.2,font=7)
mtext("Length mm",side=1,line=-0.1,cex=1.0,font=7,outer=TRUE)
mtext("Residual",side=2,line=-0.1,cex=1.0,font=7,outer=TRUE)
par(oldp)  # return par to old settings; this line not in book

### A Closer Look at the Fabens Methods
### Implementation of Non-Constant Variances
# R-chunk 9  Page 149
 # fit the Fabens tag growth curve with and without the option to
 # modify variation with predicted length. See the MQMF function
 # negnormL. So first no variation and then linear variation.

sigfunk <- function(pars,predobs) return(tail(pars,1)) #no effect
data(blackisland)
bi <- blackisland # just to keep things brief
param <- c(170.0,0.3,4.0); label=c("Linf","K","sigma")
modelvb <- nlm(f=negnormL,p=param,funk=fabens,funksig=sigfunk,
               indat=bi,initL="l1",delT="dt")
outfit(modelvb,backtran = FALSE,title="vB constant sigma",parnames = label)

sigfunk2 <- function(pars,predo) { # linear with predicted length
  sig <- tail(pars,1) * predo      # sigma x predDL, see negnormL
  pick <- which(sig <= 0)          # ensure no negative sigmas from
  sig[pick] <- 0.01           # possible negative predicted lengths
  return(sig)
} # end of sigfunk2
param <- c(170.0,0.3,1.0); label=c("Linf","K","sigma")
modelvb2 <- nlm(f=negnormL,p=param,funk=fabens,funksig=sigfunk2,
                indat=bi,initL="l1",delT="dt",
                typsize=magnitude(param),iterlim=200)
outfit(modelvb2,backtran = FALSE,parnames = label,title="vB inverse DeltaL, sigma < 1")

# R-chunk 10  Page 150
 #plot to two Faben's lines with constant and varying sigma Fig 5.6
```

```
predvB <- fabens(modelvb$estimate,bi)
predvB2 <- fabens(modelvb2$estimate,bi)
oldp <- parset(margin=c(0.4,0.4,0.05,0.05))
plot(bi$l1,bi$dl,type="p",pch=1,cex=1.0,col=1,ylim=c(-2,31),
     ylab="Growth Increment mm",xlab="Length mm",panel.first=grid())
abline(h=0,col=1)
lines(bi$l1,predvB,col=1,lwd=2)         # vB
lines(bi$l1,predvB2,col=2,lwd=2,lty=2)  # IL
legend("topright",c("Constant sigma","Changing sigma"),lwd=3,
       col=c(1,2),bty="n",cex=1.1,lty=c(1,2))
par(oldp)  # return par to old settings; this line not in book

## Objective Model Selection
### Akiake's Information Criterion
# R-chunk 11  Page 152
 #compare the relative model fits of Vb and IL

cat("von Bertalanffy \n")
aicbic(modelvb,bi)
cat("inverse-logistic \n")
aicbic(modelil,bi)

### Likelihood Ratio Test
# R-chunk 12  Page 154
 # Likelihood ratio comparison of two growth models see Fig 5.4

vb <- modelvb$minimum # their respective -ve log-likelihoods
il <- modelil$minimum
dof <- 1
round(likeratio(vb,il,dof),8)

### Caveats on Likelihood Ratio Tests
## Remarks on Growth
## Maturity
### Introduction
### Alternative Maturity Ogives
# R-chunk 13  Page 158
 # The Maturity data from tasab data-set

data(tasab)       # see ?tasab for a list of the codes used
properties(tasab) # summarize properties of columns in tasab
table(tasab$site,tasab$sex) # sites 1 & 2 vs F, I, and M

# R-chunk 14  Page 158
 #plot the proportion mature vs shell length  Fig 5.7

propm <- tapply(tasab$mature,tasab$length,mean) #mean maturity at L
lens <- as.numeric(names(propm))            # lengths in the data
oldp <- plot1(lens,propm,type="p",cex=0.9,xlab="Length mm",
      ylab="Proportion Mature")
par(oldp)  # return par to old settings; this line not in book
```

```
# R-chunk 15  Pages 159 - 160
 #Use glm to estimate mature logistic

binglm <- function(x,digits=6) { #function to simplify printing
  out <- summary(x)
  print(out$call)
  print(round(out$coefficients,digits))
  cat("\nNull Deviance  ",out$null.deviance,"df",out$df.null,"\n")
  cat("Resid.Deviance ",out$deviance,"df",out$df.residual,"\n")
  cat("AIC  = ",out$aic,"\n\n")
  return(invisible(out)) # retain the full summary
} #end of binglm
tasab$site <- as.factor(tasab$site) # site as a factor
smodel <- glm(mature ~ site + length,family=binomial,data=tasab)
outs <- binglm(smodel)  #outs contains the whole summary object

model <- glm(mature ~ length, family=binomial, data=tasab)
outm <- binglm(model)
cof <- outm$coefficients
cat("Lm50 = ",-cof[1,1]/cof[2,1],"\n")
cat("IQ   = ",2*log(3)/cof[2,1],"\n")

# R-chunk 16  Page 161
 #Add maturity logistics to the maturity data plot Fig 5.8

propm <- tapply(tasab$mature,tasab$length,mean) #prop mature
lens <- as.numeric(names(propm))       # lengths in the data
pick <- which((lens > 79) & (lens < 146))
oldp <- parset()
plot(lens[pick],propm[pick],type="p",cex=0.9, #the data points
      xlab="Length mm",ylab="Proportion Mature",pch=1)
L <- seq(80,145,1) # for increased curve separation
pars <- coef(smodel)
lines(L,mature(pars[1],pars[3],L),lwd=3,col=3,lty=2)
lines(L,mature(pars[1]+pars[2],pars[3],L),lwd=3,col=2,lty=4)
lines(L,mature(coef(model)[1],coef(model)[2],L),lwd=2,col=1,lty=1)
abline(h=c(0.25,0.5,0.75),lty=3,col="grey")
legend("topleft",c("site1","both","site2"),col=c(3,1,2),lty=c(2,1,4),
       lwd=3,bty="n")
par(oldp)  # return par to old settings; this line not in book


### The Assumption of Symmetry
# R-chunk 17  Page 163
 #Asymmetrical maturity curve from Schnute and Richard's curve Fig5.9

L = seq(50,160,1)
p=c(a=0.07,b=0.2,c=1.0,alpha=100)
asym <- srug(p=p,sizeage=L)
L25 <- linter(bracket(0.25,asym,L))
L50 <- linter(bracket(0.5,asym,L))
L75 <- linter(bracket(0.75,asym,L))
oldp <- parset()
```

```
plot(L,asym,type="l",lwd=2,xlab="Length mm",ylab="Proportion Mature")
abline(h=c(0.25,0.5,0.75),lty=3,col="grey")
abline(v=c(L25,L50,L75),lwd=c(1,2,1),col=c(1,2,1))
par(oldp)  # return par to old settings; this line not in book

# R-chunk 18  Page 164 code not printed in the book
 #Variation possible using the Schnute and Richard's Curve fig 5.10
 # This code not printed in the book

tmplot <- function(vals,label) {
  text(170,0.6,paste0("  ",label),font=7,cex=1.5)
  legend("bottomright",legend=vals,col=1:nvals,lwd=3,bty="n",
         cex=1.25,lty=c(1:nvals))
}
L = seq(50,180,1)
vals <- seq(0.05,0.09,0.01) # a value
nvals <- length(vals)
asym <-  srug(p=c(a=vals[1],b=0.2,c=1.0,alpha=100),sizeage=L)
oldp <- parset(plots=c(2,2))
plot(L,asym,type="l",lwd=2,xlab="Length mm",ylab="Proportion Mature",
      ylim=c(0,1.05))
abline(h=c(0.25,0.5,0.75),lty=3,col="darkgrey")
for (i in 2:nvals) {
  asym <- srug(p=c(a=vals[i],b=0.2,c=1.0,alpha=100),sizeage=L)
  lines(L,asym,lwd=2,col=i,lty=i)
}
tmplot(vals,"a")
vals <- seq(0.02,0.34,0.08) # b value
nvals <- length(vals)
asym <-  srug(p=c(a=0.07,b=vals[1],c=1.0,alpha=100),sizeage=L)
plot(L,asym,type="l",lwd=2,xlab="Length mm",ylab="Proportion Mature",
      ylim=c(0,1.05))
abline(h=c(0.25,0.5,0.75),lty=3,col="darkgrey")
for (i in 2:nvals) {
  asym <- srug(p=c(a=0.07,b=vals[i],c=1.0,alpha=100),sizeage=L)
  lines(L,asym,lwd=2,col=i,lty=i)
}
tmplot(vals,"b")
vals <- seq(0.95,1.05,0.025) # c value
nvals <- length(vals)
asym <-  srug(p=c(a=0.07,b=0.2,c=vals[1],alpha=100),sizeage=L)
plot(L,asym,type="l",lwd=2,xlab="Length mm",ylab="Proportion Mature",
      ylim=c(0,1.05))
abline(h=c(0.25,0.5,0.75),lty=3,col="darkgrey")
for (i in 2:nvals) {
  asym <- srug(p=c(a=0.07,b=0.2,c=vals[i],alpha=100),sizeage=L)
  lines(L,asym,lwd=2,col=i,lty=i)
}
tmplot(vals,"c")
vals <- seq(25,225,50) # alpha value
nvals <- length(vals)
asym <-  srug(p=c(a=0.07,b=0.2,c=1.0,alpha=vals[1]),sizeage=L)
plot(L,asym,type="l",lwd=2,xlab="Length mm",ylab="Proportion Mature",
```

```
      ylim=c(0,1.05))
abline(h=c(0.25,0.5,0.75),lty=3,col="darkgrey")
for (i in 2:nvals) {
  asym <- srug(p=c(a=0.07,b=0.2,c=1.0,alpha=vals[i]),sizeage=L)
  lines(L,asym,lwd=2,col=i,lty=i)
}
tmplot(vals,"alpha")
par(oldp)  # return par to old settings; this line not in book

## Recruitment
### Introduction
### Properties of Good Stock Recruitment Relationships
### Recruitment Overfishing
### Beverton and Holt Recruitment
# R-chunk 19  Page 169
 #plot the MQMF bh function for Beverton-Holt recruitment  Fig 5.11

B <- 1:3000
bhb <- c(1000,500,250,150,50)
oldp <- parset()
plot(B,bh(c(1000,bhb[1]),B),type="l",ylim=c(0,1050),
      xlab="Spawning Biomass",ylab="Recruitment")
for (i in 2:5) lines(B,bh(c(1000,bhb[i]),B),lwd=2,col=i,lty=i)
legend("bottomright",legend=bhb,col=c(1:5),lwd=3,bty="n",lty=c(1:5))
abline(h=c(500,1000),col=1,lty=2)
par(oldp)  # return par to old settings; this line not in book

### Ricker Recruitment
# R-chunk 20  Page 170
 #plot the MQMF ricker function for Ricker recruitment  Fig 5.12

B <- 1:20000
rickb <- c(0.0002,0.0003,0.0004)
oldp <- parset()
plot(B,ricker(c(10,rickb[1]),B),type="l",xlab="Spawning Biomass",ylab="Recruitment")
for (i in 2:3)
   lines(B,ricker(c(10,rickb[i]),B),lwd=2,col=i,lty=i)
legend("topright",legend=rickb,col=1:3,lty=1:3,bty="n",lwd=2)
par(oldp)  # return par to old settings; this line not in book


### Deriso's Generalized Model

# R-chunk 21  Page 172
 # plot of three special cases from Deriso-Schnute curve  Fig. 5.13
deriso <- function(p,B) return(p[1] * B *(1 - p[2]*p[3]*B)^(1/p[3]))
B <- 1:10000
oldp <- plot1(B,deriso(c(10,0.001,-1),B),lwd=2,xlab="Spawning Biomass",
              ylab="Recruitment")
lines(B,deriso(c(10,0.0004,0.25),B),lwd=2,col=2,lty=2)  # DS
lines(B,deriso(c(10,0.0004,1e-06),B),lwd=2,col=3,lty=3) # Ricker
lines(B,deriso(c(10,0.0004,0.5),B),lwd=2,col=1,lty=3)   # odd line
legend(x=7000,y=8500,legend=c("BH","DS","Ricker","odd line"),
```

```
       col=c(1,2,3,1),lty=c(1,2,3,3),bty="n",lwd=3)
par(oldp)  # return par to old settings; this line not in book


### Re-Parameterized Beverton-Holt Equation
### Re-Parameterized Ricker Equation
## Selectivity
### Introduction
### Logistic Selection
# R-chunk 22  Page 177
 #Selectivity curves from logist and mature functions  See Fig 5.14

ages <- seq(0,50,1);   in50 <- 25.0
sel1 <- logist(in50,12,ages)         #-3.65/0.146=L50=25.0
sel2 <- mature(-3.650425,0.146017,sizeage=ages)
sel3 <- mature(-6,0.2,ages)
sel4 <- logist(22.0,14,ages,knifeedge = TRUE)
oldp <- plot1(ages,sel1,xlab="Age Years",ylab="Selectivity",cex=0.75,lwd=2)
lines(ages,sel2,col=2,lwd=2,lty=2)
lines(ages,sel3,col=3,lwd=2,lty=3)
lines(ages,sel4,col=4,lwd=2,lty=4)
abline(v=in50,col=1,lty=2); abline(h=0.5,col=1,lty=2)
legend("topleft",c("25_eq5.30","25_eq5.31","30_eq5.31","22_eq5.30N"),
       col=c(1,2,3,4),lwd=3,cex=1.1,bty="n",lty=c(1:4))
par(oldp)  # return par to old settings; this line not in book


### Dome Shaped Selection
# R-chunk 23  Page 179
 #Examples of domed-shaped selectivity curves from domed. Fig.5.15

L <- seq(1,30,1)
p <- c(10,11,16,33,-5,-2)
oldp <- plot1(L,domed(p,L),type="l",lwd=2,ylab="Selectivity",xlab="Age Years")
p1 <- c(8,12,16,33,-5,-1)
lines(L,domed(p1,L),lwd=2,col=2,lty=2)
p2 <- c(9,10,16,33,-5,-4)
lines(L,domed(p2,L),lwd=2,col=4,lty=4)
par(oldp)  # return par to old settings; this line not in book

## End(Not run)
```

---

chapter6 *chapter6 The 53 R-code chunks from On Uncertainty*

Description

chapter6 is not an active function but rather acts as a repository for the various example code chunks found in chapter6. There are 53 r-code chunks in chapter6. You should, of course, feel free to use and modify any of these example chunks in your own work.

Examples

```
## Not run:
# All the example code from  # On Uncertainty
# On Uncertainty
## Introduction
### Types of Uncertainty
### The Example Model

# R-chunk 1  Page 189
 #Fit a surplus production model to abdat fisheries data

data(abdat); logce <- log(abdat$cpue)
param <- log(c(0.42,9400,3400,0.05))
label=c("r","K","Binit","sigma") # simpspm returns
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,logobs=logce)
outfit(bestmod,title="SP-Model",parnames=label) #backtransforms

# R-chunk 2  Page 190
 #plot the abdat data and the optimum sp-model fit  Fig 6.1

predce <- exp(simpspm(bestmod$estimate,abdat))
optresid <- abdat[,"cpue"]/predce #multiply by predce for obsce
ymax <- getmax(c(predce,abdat$cpue))
oldp <- plot1(abdat$year,(predce*optresid),type="l",maxy=ymax,cex=0.9,
      ylab="CPUE",xlab="Year",lwd=3,col="grey",lty=1)
points(abdat$year,abdat$cpue,pch=1,col=1,cex=1.1)
lines(abdat$year,predce,lwd=2,col=1)  # best fit line
par(oldp)  # return par to old settings; this line not in book

## Bootstrapping
### Empirical Probability Density Distributions
## A Simple Bootstrap Example
# R-chunk 3  Page 193
 #regression between catches of NPF prawn species Fig 6.2

data(npf)
model <- lm(endeavour ~ tiger,data=npf)
oldp <- plot1(npf$tiger,npf$endeavour,type="p",xlab="Tiger Prawn (t)",
      ylab="Endeavour Prawn (t)",cex=0.9)
abline(model,col=1,lwd=2)
correl <- sqrt(summary(model)$r.squared)
pval <- summary(model)$coefficients[2,4]
label <- paste0("Correlation ",round(correl,5)," P = ",round(pval,8))
text(2700,180,label,cex=1.0,font=7,pos=4)
par(oldp)  # return par to old settings; this line not in book


# R-chunk 4  Page 194
# 5000 bootstrap estimates of correlation coefficient Fig 6.3

set.seed(12321)     # better to use a less obvious seed, if at all
N <- 5000                           # number of bootstrap samples
```

```
result <- numeric(N)          #a vector to store 5000 correlations
for (i in 1:N) {          #sample index from 1:23 with replacement
   pick <- sample(1:23,23,replace=TRUE)   #sample is an R function
   result[i] <- cor(npf$tiger[pick],npf$endeavour[pick])
}
rge <- range(result)                  # store the range of results
CI <- quants(result)     # calculate quantiles; 90%CI = 5% and 95%
restrim <- result[result > 0] #remove possible -ve values for plot
oldp <- parset(cex=1.0)  #set up a plot window and draw a histogram
bins <- seq(trunc(range(restrim)[1]*10)/10,1.0,0.01)
outh <- hist(restrim,breaks=bins,main="",col=0,xlab="Correlation")
abline(v=c(correl,mean(result)),col=c(4,3),lwd=c(3,2),lty=c(1,2))
abline(v=CI[c(2,4)],col=4,lwd=2) # and 90% confidence intervals
text(0.48,400,makelabel("Range ",rge,sep="  ",sigdig=4),font=7,pos=4)
label <- makelabel("90%CI ",CI[c(2,4)],sep="  ",sigdig=4)
text(0.48,300,label,cex=1.0,font=7,pos=4)
par(oldp)  # return par to old settings; this line not in book

## Bootstrapping Time-Series Data
# R-chunk 5  Page 196
 # fitting Schaefer model with log-normal residuals with 24 years

data(abdat); logce <- log(abdat$cpue) # of abalone fisheries data
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05)) #log values
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,logobs=logce)
optpar <- bestmod$estimate      # these are still log-transformed
predce <- exp(simpspm(optpar,abdat))      #linear-scale pred cpue
optres <- abdat[,"cpue"]/predce     # optimum log-normal residual
optmsy <- exp(optpar[1])*exp(optpar[2])/4
sampn <- length(optres)        # number of residuals and of years

# R-chunk 6  Page 196 Table 6.1 code not included in the book

outtab <- halftable(cbind(abdat,predce,optres),subdiv=2)
kable(outtab, digits=c(0,0,3,3,3,0,0,3,3,3), caption='(ref:tab601)')

# R-chunk 7  Pages 196 - 197
 # 1000 bootstrap Schaefer model fits; takes a few seconds

start <- Sys.time() # use of as.matrix faster than using data.frame
bootfish <- as.matrix(abdat)  # and avoid altering original data
N <- 1000;   years <- abdat[,"year"] # need N x years matrices
columns <- c("r","K","Binit","sigma")
results <- matrix(0,nrow=N,ncol=sampn,dimnames=list(1:N,years))
bootcpue <- matrix(0,nrow=N,ncol=sampn,dimnames=list(1:N,years))
parboot <- matrix(0,nrow=N,ncol=4,dimnames=list(1:N,columns))
for (i in 1:N) {  # fit the models and save solutions
  bootcpue[i,] <- predce * sample(optres, sampn, replace=TRUE)
  bootfish[,"cpue"] <- bootcpue[i,] #calc and save bootcpue
  bootmod <- nlm(f=negLL,p=optpar,funk=simpspm,indat=bootfish,
        logobs=log(bootfish[,"cpue"]))
  parboot[i,] <- exp(bootmod$estimate)  #now save parameters
  results[i,] <- exp(simpspm(bootmod$estimate,abdat))  #and predce
```

```
}
cat("total time = ",Sys.time()-start, "seconds   \n")

# R-chunk 8  Page 197
 # bootstrap replicates in grey behind main plot Fig 6.4

oldp <- plot1(abdat[,"year"],abdat[,"cpue"],type="n",xlab="Year",
              ylab="CPUE") # type="n" just lays out an empty plot
for (i in 1:N)      # ready to add the separate components
  lines(abdat[,"year"],results[i,],lwd=1,col="grey")
points(abdat[,"year"],abdat[,"cpue"],pch=16,cex=1.0,col=1)
lines(abdat[,"year"],predce,lwd=2,col=1)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 9  Pages 198 - 199
 #histograms of bootstrap parameters and model outputs Fig 6.5

dohist <- function(invect,nmvar,bins=30,bootres,avpar) { #adhoc
  hist(invect[,nmvar],breaks=bins,main="",xlab=nmvar,col=0)
  abline(v=c(exp(avpar),bootres[pick,nmvar]),lwd=c(3,2,3,2),
         col=c(3,4,4,4))
}
msy <- parboot[,"r"]*parboot[,"K"]/4 #calculate bootstrap MSY
msyB <- quants(msy)        #from optimum bootstrap parameters
oldp <- parset(plots=c(2,2),cex=0.9)
bootres <- apply(parboot,2,quants); pick <- c(2,3,4) #quantiles
dohist(parboot,nmvar="r",bootres=bootres,avpar=optpar[1])
dohist(parboot,nmvar="K",bootres=bootres,avpar=optpar[2])
dohist(parboot,nmvar="Binit",bootres=bootres,avpar=optpar[3])
hist(msy,breaks=30,main="",xlab="MSY",col=0)
abline(v=c(optmsy,msyB[pick]),lwd=c(3,2,3,2),col=c(3,4,4,4))
par(oldp)  # return par to old settings; this line not in book

### Parameter Correlation
# R-chunk 10  Page 200
 #relationships between parameters and MSY  Fig 6.6

parboot1 <- cbind(parboot,msy)
 # note rgb use, alpha allows for shading, try 1/15 or 1/10
pairs(parboot1,pch=16,col=rgb(red=1,green=0,blue=0,alpha = 1/20))

## Asymptotic Errors
# R-chunk 11  Page 203
 #Fit Schaefer model and generate the Hessian

data(abdat)
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
 # Note inclusion of the option hessian=TRUE in nlm function
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,
               logobs=log(abdat[,"cpue"]),hessian=TRUE)
outfit(bestmod,backtran = TRUE) #try typing bestmod in console
 # Now generate the confidence intervals
vcov <- solve(bestmod$hessian)      # solve inverts matrices
```

```
sterr <- sqrt(diag(vcov)) #diag extracts diagonal from a matrix
optpar <- bestmod$estimate      #use qt for t-distrib quantiles
U95 <- optpar + qt(0.975,20)*sterr # 4 parameters hence
L95 <- optpar - qt(0.975,20)*sterr # (24 - 4) df
cat("\n               r      K     Binit    sigma \n")
cat("Upper 95% ",round(exp(U95),5),"\n") # backtransform
cat("Optimum   ",round(exp(optpar),5),"\n")#\n =linefeed in cat
cat("Lower 95% ",round(exp(L95),5),"\n")

### Uncertainty about the Model Outputs
### Sampling from a Multi-Variate Normal Distribution
# R-chunk 12  Page 204
 # Use multi-variate normal to generate percentile CI    Fig 6.7

library(mvtnorm) # use RStudio, or install.packages("mvtnorm")
N <- 1000 # number of multi-variate normal parameter vectors
years <- abdat[,"year"];  sampn <- length(years)  # 24 years
mvncpue <- matrix(0,nrow=N,ncol=sampn,dimnames=list(1:N,years))
columns <- c("r","K","Binit","sigma")
 # Fill parameter vectors with N vectors from rmvnorm
mvnpar <- matrix(exp(rmvnorm(N,mean=optpar,sigma=vcov)),
                 nrow=N,ncol=4,dimnames=list(1:N,columns))
 # Calculate N cpue trajectories using simpspm
for (i in 1:N) mvncpue[i,] <- exp(simpspm(log(mvnpar[i,]),abdat))
msy <- mvnpar[,"r"]*mvnpar[,"K"]/4 #N MSY estimates
 # plot data and trajectories from the N parameter vectors
oldp <- plot1(abdat[,"year"],abdat[,"cpue"],type="p",xlab="Year",
              ylab="CPUE",cex=0.9)
for (i in 1:N) lines(abdat[,"year"],mvncpue[i,],col="grey",lwd=1)
points(abdat[,"year"],abdat[,"cpue"],pch=16,cex=1.0)#orig data
lines(abdat[,"year"],exp(simpspm(optpar,abdat)),lwd=2,col=1)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 13  Page 205
 #correlations between parameters when using mvtnorm Fig 6.8

pairs(cbind(mvnpar,msy),pch=16,col=rgb(red=1,0,0,alpha = 1/10))

# R-chunk 14  Pages 206 - 207
 #N parameter vectors from the multivariate normal Fig 6.9

mvnres <- apply(mvnpar,2,quants)  # table of quantiles
pick <- c(2,3,4)   # select rows for 5%, 50%, and 95%
meanmsy <- mean(msy)     # optimum bootstrap parameters
msymvn <- quants(msy)   # msy from mult-variate normal estimates
plothist <- function(x,optp,label,resmvn) {
  hist(x,breaks=30,main="",xlab=label,col=0)
  abline(v=c(exp(optp),resmvn),lwd=c(3,2,3,2),col=c(3,4,4,4))
} # repeated 4 times, so worthwhile writing a short function
oldp <- par(no.readonly=TRUE)
par(mfrow=c(2,2),mai=c(0.45,0.45,0.05,0.05),oma=c(0.0,0,0.0,0.0))
par(cex=0.85, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
plothist(mvnpar[,"r"],optpar[1],"r",mvnres[pick,"r"])
```

```
plothist(mvnpar[,"K"],optpar[2],"K",mvnres[pick,"K"])
plothist(mvnpar[,"Binit"],optpar[3],"Binit",mvnres[pick,"Binit"])
plothist(msy,meanmsy,"MSY",msymvn[pick])
par(oldp)  # return par to old settings; this line not in book

# R-chunk 15   Page 208 Table 6.2 code not included in the book
 #Tabulate percentile CI from bootstrap (B) and multi-variate (mvn)

kable(cbind(bootres,msyB),digits=c(4,3,3,4,3), caption='(ref:tab602)')
kable(cbind(mvnres,msymvn),digits=c(4,3,3,4,3))

## Likelihood Profiles
# R-chunk 16  Page 209
 #Fit the Schaefer surplus production model to abdat

data(abdat); logce <- log(abdat$cpue)    # using negLL
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
optmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,logobs=logce)
outfit(optmod,parnames=c("r","K","Binit","sigma"))

# R-chunk 17  Page 210
 #the code for MQMF's negLLP function

negLLP <- function(pars, funk, indat, logobs, initpar=pars,
                   notfixed=c(1:length(pars)),...) {
  usepar <- initpar  #copy the original parameters into usepar
  usepar[notfixed] <- pars[notfixed] #change 'notfixed' values
  npar <- length(usepar)
  logpred <- funk(usepar,indat,...) #funk uses the usepar values
  pick <- which(is.na(logobs))  # proceed as in negLL
  if (length(pick) > 0) {
    LL <- -sum(dnorm(logobs[-pick],logpred[-pick],exp(pars[npar]),
                     log=T))
  } else {
    LL <- -sum(dnorm(logobs,logpred,exp(pars[npar]),log=T))
  }
  return(LL)
} # end of negLLP

# R-chunk 18  Page 211
 #does negLLP give same answers as negLL when no parameters fixed?

param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
bestmod <- nlm(f=negLLP,p=param,funk=simpspm,indat=abdat,logobs=logce)
outfit(bestmod,parnames=c("r","K","Binit","sigma"))

# R-chunk 19  Page 211
 #Likelihood profile for r values 0.325 to 0.45

rval <- seq(0.325,0.45,0.001)  # set up the test sequence
ntrial <- length(rval)        # create storage for the results
columns <- c("r","K","Binit","sigma","-veLL")
result <- matrix(0,nrow=ntrial,ncol=length(columns),
```

```
                dimnames=list(rval,columns))# close to optimum
bestest <- c(r= 0.32,K=11000,Binit=4000,sigma=0.05)
for (i in 1:ntrial) {  #i <- 1
  param <- log(c(rval[i],bestest[2:4])) #recycle bestest values
  parinit <- param  #to improve the stability of nlm as r changes
  bestmodP <- nlm(f=negLLP,p=param,funk=simpspm,initpar=parinit,
                  indat=abdat,logobs=log(abdat$cpue),notfixed=c(2:4),
                  typsize=magnitude(param),iterlim=1000)
  bestest <- exp(bestmodP$estimate)
  result[i,] <- c(bestest,bestmodP$minimum)  # store each result
}
minLL <- min(result[,"-veLL"]) #minimum across r values used.

# R-chunk 20   Page 212 Table 6.3 code not included in the book
 #tabulate first 12 records from likelihood profile

kable(head(result,12),digits=c(3,3,3,4,5), caption='(ref:tab603)')

### Likelihood Ratio Based Confidence Intervals
# R-chunk 21  Page 213
 #likelihood profile on r from the Schaefer model Fig 6.10

plotprofile(result,var="r",lwd=2)  # review the code

# R-chunk 22  Page 214
 #Likelihood profile for K values 7200 to 12000

Kval <- seq(7200,12000,10)
ntrial <- length(Kval)
columns <- c("r","K","Binit","sigma","-veLL")
resultK <- matrix(0,nrow=ntrial,ncol=length(columns),
                  dimnames=list(Kval,columns))
bestest <- c(r= 0.45,K=7500,Binit=2800,sigma=0.05)
for (i in 1:ntrial) {
  param <- log(c(bestest[1],Kval[i],bestest[c(3,4)]))
  parinit <- param
  bestmodP <- nlm(f=negLLP,p=param,funk=simpspm,initpar=parinit,
                  indat=abdat,logobs=log(abdat$cpue),
                  notfixed=c(1,3,4),iterlim=1000)
  bestest <- exp(bestmodP$estimate)
  resultK[i,] <- c(bestest,bestmodP$minimum)
}
minLLK <- min(resultK[,"-veLL"])
 #kable(head(result,12),digits=c(4,3,3,4,5))  # if wanted.


# R-chunk 23  Page 214
 #likelihood profile on K from the Schaefer model Fig 6.11

plotprofile(resultK,var="K",lwd=2)

### -ve Log-Likelihoods or Likelihoods
# R-chunk 24  Page 215
```

```
 #translate -velog-likelihoods into likelihoods

likes <- exp(-resultK[,"-veLL"])/sum(exp(-resultK[,"-veLL"]),na.rm=TRUE)
resK <- cbind(resultK,likes,cumlike=cumsum(likes))

# R-chunk 25   Page 216 Table 6.4 code not included in the book
 #tabulate head of likelihood profile matrix for K

kable(head(resK,8),digits=c(4,0,3,4,5,9,7),caption='(ref:tab604)')

# R-chunk 26  Page 216 Figure 6.12 code not in the book
 #K parameter likelihood profile  Fig 6.12

oldp <- plot1(resK[,"K"],resK[,"likes"],xlab="K value",
              ylab="Likelihood",lwd=2)
lower <- which.closest(0.025,resK[,"cumlike"])
mid <- which(resK[,"likes"] == max(resK[,"likes"]))
upper <- which.closest(0.975,resK[,"cumlike"])
abline(v=c(resK[c(lower,mid,upper),"K"]),col=1,lwd=c(1,2,1))
label <- makelabel("",resK[c(lower,mid,upper),"K"],sep="  ")
text(9500,0.005,label,cex=1.2,pos=4)
par(oldp)  # return par to old settings; this line not in book

### Percentile Likelihood Profiles for Model Outputs
# R-chunk 27  Page 217 - 218
 #examine effect on -veLL of MSY values from 740 - 1050t
 #need a different negLLP() function, negLLO(): O for output.
 #now optvar=888.831 (rK/4), the optimum MSY, varval ranges 740-1050
 #and wght is the weighting to give to the penalty

negLLO <- function(pars,funk,indat,logobs,wght,optvar,varval) {
  logpred <- funk(pars,indat)
  LL <- -sum(dnorm(logobs,logpred,exp(tail(pars,1)),log=T)) +
             wght*((varval - optvar)/optvar)^2  #compare with negLL
  return(LL)
} # end of negLLO
msyP <- seq(740,1020,2.5);
optmsy <- exp(optmod$estimate[1])*exp(optmod$estimate[2])/4
ntrial <- length(msyP)
wait <- 400
columns <- c("r","K","Binit","sigma","-veLL","MSY","pen","TrialMSY")
resultO <- matrix(0,nrow=ntrial,ncol=length(columns),dimnames=list(msyP,columns))
bestest <- c(r= 0.47,K=7300,Binit=2700,sigma=0.05)
for (i in 1:ntrial) {  # i <- 1
  param <- log(bestest)
  bestmodO <- nlm(f=negLLO,p=param,funk=simpspm,indat=abdat,
                  logobs=log(abdat$cpue),wght=wait,
                  optvar=optmsy,varval=msyP[i],iterlim=1000)
  bestest <- exp(bestmodO$estimate)
  ans <- c(bestest,bestmodO$minimum,bestest[1]*bestest[2]/4,
           wait *((msyP[i] - optmsy)/optmsy)^2,msyP[i])
  resultO[i,] <- ans
}
```

```
minLLO <- min(resultO[,"-veLL"])

# R-chunk 28   Page 218 Table 6.5 code not included in the book
 #tabulate first and last few records of profile on MSY

kable(head(resultO[,1:7],4),digits=c(3,3,3,4,2,3,2),caption='(ref:tab605)')
kable(tail(resultO[,1:7],4),digits=c(3,3,3,4,2,3,2))

# R-chunk 29   Page 219 Figure 6.13 code not included in the book
 #likelihood profile on MSY from the Schaefer model Fig 6.13

plotprofile(resultO,var="TrialMSY",lwd=2)

## Bayesian Posterior Distributions
### Generating the Markov Chain
### The Starting Point
### The Burn-in Period
### Convergence to the Stationary Distribution
### The Jumping Distribution
### Application of MCMC to the Example

# R-chunk 30  Page 225
 #activate and plot the fisheries data in abdat  Fig 6.14

data(abdat)   # type abdat in the console to see contents
plotspmdat(abdat) #use helper function to plot fishery stats vs year


### Markov Chain Monte Carlo
### A First Example of an MCMC
# R-chunk 31  Pages 228 - 229
 # Conduct MCMC analysis to illustrate burn-in. Fig 6.15

data(abdat);  logce <- log(abdat$cpue)
fish <- as.matrix(abdat) # faster to use a matrix than a data.frame!
begin <- Sys.time()       # enable time taken to be calculated
chains <- 1                # 1 chain per run; normally do more
burnin <- 0                # no burn-in for first three chains
N <- 100                        # Number of MCMC steps to keep
step <- 4       # equals one step per parameter so no thinning
priorcalc <- calcprior # define the prior probability function
scales <- c(0.065,0.055,0.065,0.425) #found by trial and error
set.seed(128900) #gives repeatable results in book; usually omitted
inpar <- log(c(r= 0.4,K=11000,Binit=3600,sigma=0.05))
result1 <- do_MCMC(chains,burnin,N,step,inpar,negLL,calcpred=simpspm,
                   calcdat=fish,obsdat=logce,priorcalc,scales)
inpar <- log(c(r= 0.35,K=8500,Binit=3400,sigma=0.05))
result2 <- do_MCMC(chains,burnin,N,step,inpar,negLL,calcpred=simpspm,
                   calcdat=fish,obsdat=logce,priorcalc,scales)
inpar <- log(c(r= 0.45,K=9500,Binit=3200,sigma=0.05))
result3 <- do_MCMC(chains,burnin,N,step,inpar,negLL,calcpred=simpspm,
                   calcdat=fish,obsdat=logce,priorcalc,scales)
burnin <- 50 # strictly a low thinning rate of 4; not enough
```

```
step <- 16   # 16 thinstep rate = 4 parameters x 4 = 16
N <- 10000   # 16 x 10000 = 160,000 steps + 50 burnin
inpar <- log(c(r= 0.4,K=9400,Binit=3400,sigma=0.05))
result4 <- do_MCMC(chains,burnin,N,step,inpar,negLL,calcpred=simpspm,
                   calcdat=fish,obsdat=logce,priorcalc,scales)
post1 <- result1[[1]][[1]]
post2 <- result2[[1]][[1]]
post3 <- result3[[1]][[1]]
postY <- result4[[1]][[1]]
cat("time   = ",Sys.time() - begin,"\n")
cat("Accept = ",result4[[2]],"\n")


# R-chunk 32  Pages 229 - 230
#first example and start of 3 initial chains for MCMC Fig6.15

oldp <- parset(cex=0.85)
P <- 75  # the first 75 steps only start to explore parameter space
plot(postY[,"K"],postY[,"r"],type="p",cex=0.2,xlim=c(7000,13000),
   ylim=c(0.28,0.47),col=8,xlab="K",ylab="r",panel.first=grid())
lines(post2[1:P,"K"],post2[1:P,"r"],lwd=1,col=1)
points(post2[1:P,"K"],post2[1:P,"r"],pch=15,cex=1.0)
lines(post1[1:P,"K"],post1[1:P,"r"],lwd=1,col=1)
points(post1[1:P,"K"],post1[1:P,"r"],pch=1,cex=1.2,col=1)
lines(post3[1:P,"K"],post3[1:P,"r"],lwd=1,col=1)
points(post3[1:P,"K"],post3[1:P,"r"],pch=2,cex=1.2,col=1)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 33  Pages 230 - 231
 #pairs plot of parameters from the first MCMC Fig 6.16

posterior <- result4[[1]][[1]]
msy <-posterior[,1]*posterior[,2]/4
pairs(cbind(posterior[,1:4],msy),pch=16,col=rgb(1,0,0,1/50),font=7)

# R-chunk 34  Pages 231 - 232
 #plot the traces from the first MCMC example Fig 6.17

posterior <- result4[[1]][[1]]
oldp <- par(no.readonly=TRUE)   # this line not in book
par(mfrow=c(4,2),mai=c(0.4,0.4,0.05,0.05),oma=c(0.0,0,0.0,0.0))
par(cex=0.8, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
label <- colnames(posterior)
N <- dim(posterior)[1]
for (i in 1:4) {
  ymax <- getmax(posterior[,i]); ymin <- getmin(posterior[,i])
  plot(1:N,posterior[,i],type="l",lwd=1,ylim=c(ymin,ymax),
       panel.first=grid(),ylab=label[i],xlab="Step")
  plot(density(posterior[,i]),lwd=2,col=2,panel.first=grid(),main="")
}
par(oldp)  # return par to old settings; this line not in book

# R-chunk 35  Page 233
```

```
 #Use acf to examine auto-correlation with thinstep = 16   Fig 6.18

posterior <- result4[[1]][[1]]
label <- colnames(posterior)[1:4]
oldp <- parset(plots=c(2,2),cex=0.85)
for (i in 1:4) auto <- acf(posterior[,i],type="correlation",lwd=2,
                           plot=TRUE,ylab=label[i],lag.max=20)
par(oldp)  # return par to old settings; this line not in book


# R-chunk 36  Pages 233 - 234
 #setup MCMC with thinstep of 128 per parameter  Fig 6.19

begin=gettime()
scales <- c(0.06,0.05,0.06,0.4)
inpar <- log(c(r= 0.4,K=9400,Binit=3400,sigma=0.05))
result <- do_MCMC(chains=1,burnin=100,N=1000,thinstep=512,inpar,
                  negLL,calcpred=simpspm,calcdat=fish,
                  obsdat=logce,calcprior,scales,schaefer=TRUE)
posterior <- result[[1]][[1]]
label <- colnames(posterior)[1:4]
oldp <- parset(plots=c(2,2),cex=0.85)
for (i in 1:4) auto <- acf(posterior[,i],type="correlation",lwd=2,
                           plot=TRUE,ylab=label[i],lag.max=20)
par(oldp)  # return par to old settings; this line not in book
cat(gettime() - begin)



### Marginal Distributions
# R-chunk 37  Pages 235 - 236
 # plot marginal distributions from the MCMC  Fig 6.20

dohist <- function(x,xlab) { # to save a little space
  return(hist(x,main="",breaks=50,col=0,xlab=xlab,ylab="",
              panel.first=grid()))
}
 # ensure we have the optimum solution available
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
bestmod <- nlm(f=negLL,p=param,funk=simpspm,indat=abdat,
               logobs=log(abdat$cpue))
optval <- exp(bestmod$estimate)
posterior <- result[[1]][[1]] #example above N=1000, thin=512
oldp <- par(no.readonly=TRUE)
par(mfrow=c(5,1),mai=c(0.4,0.3,0.025,0.05),oma=c(0,1,0,0))
par(cex=0.85, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
np <- length(param)
for (i in 1:np) { #store invisible output from hist for later use
  outH <- dohist(posterior[,i],xlab=colnames(posterior)[i])
  abline(v=optval[i],lwd=3,col=4)
  tmp <- density(posterior[,i])
  scaler <- sum(outH$counts)*(outH$mids[2]-outH$mids[1])
  tmp$y <- tmp$y * scaler
```

```
  lines(tmp,lwd=2,col=2)
}
msy <- posterior[,"r"]*posterior[,"K"]/4
mout <- dohist(msy,xlab="MSY")
tmp <- density(msy)
tmp$y <- tmp$y * (sum(mout$counts)*(mout$mids[2]-mout$mids[1]))
lines(tmp,lwd=2,col=2)
abline(v=(optval[1]*optval[2]/4),lwd=3,col=4)
mtext("Frequency",side=2,outer=T,line=0.0,font=7,cex=1.0)
par(oldp)  # return par to old settings; this line not in book


## The Use of Rcpp
# R-chunk 38  Pages 236 - 237
 #profile the running of do_MCMC  using the now well known abdat

data(abdat); logce <- log(abdat$cpue); fish <- as.matrix(abdat)
param <- log(c(r=0.39,K=9200,Binit=3400,sigma=0.05))
Rprof(append=TRUE)  # note the use of negLL1()
result <- do_MCMC(chains=1,burnin=100,N=20000,thinstep=16,inpar=param,
                infunk=negLL1,calcpred=simpspm,calcdat=fish,
                obsdat=logce,priorcalc=calcprior,
                scales=c(0.07,0.06,0.07,0.45))
Rprof(NULL)
outprof <- summaryRprof()


# R-chunk 39   Page 238 Table 6.6 code not included in the book
 #tabulate output of Rprof on do_MCMC function

kable(head(outprof$by.self,12),caption='(ref:tab606)')

### Addressing Vectors and Matrices
### Replacement for simpspm()
# R-chunk 40  Page 240

library(Rcpp)
 #Send a text string containing the C++ code to cppFunction this will
 #take a few seconds to compile, then the function simpspmC will
 #continue to be available during the rest of your R session. The
 #code in this chunk could be included into its own R file, and then
 #the R source() function can be used to include the C++ into a
 #session. indat must have catch in col2 (col1 in C++), and cpue in
 #col3 (col2 in C++). Note the use of ; at the end of each line.
 #Like simpspm(), this returns only the log(predicted cpue).
cppFunction('NumericVector simpspmC(NumericVector pars,
             NumericMatrix indat, LogicalVector schaefer) {
    int nyrs = indat.nrow();
    NumericVector predce(nyrs);
    NumericVector biom(nyrs+1);
    double Bt, qval;
    double sumq = 0.0;
    double p = 0.00000001;
```

```
    if (schaefer(0) == TRUE) {
      p = 1.0;
    }
    NumericVector ep = exp(pars);
    biom[0] = ep[2];
    for (int i = 0; i < nyrs; i++) {
      Bt = biom[i];
      biom[(i+1)]=Bt+(ep[0]/p)*Bt*(1-pow((Bt/ep[1]),p))-
                     indat(i,1);
      if (biom[(i+1)] < 40.0) biom[(i+1)] = 40.0;
      sumq += log(indat(i,2)/biom[i]);
    }
    qval = exp(sumq/nyrs);
    for (int i = 0; i < nyrs; i++) {
      predce[i] = log(biom[i] * qval);
    }
    return predce;
}')


# R-chunk 41  Page 241
 #Ensure results obtained from simpspm and simpspmC are same

library(microbenchmark)
data(abdat)
fishC <- as.matrix(abdat) # Use a matrix rather than a data.frame
inpar <- log(c(r= 0.389,K=9200,Binit=3300,sigma=0.05))
spmR <- exp(simpspm(inpar,fishC)) # demonstrate equivalence
 #need to declare all arguments in simpspmC, no default values
spmC <- exp(simpspmC(inpar,fishC,schaefer=TRUE))
out <- microbenchmark( # verything identical calling function
  simpspm(inpar,fishC,schaefer=TRUE),
  simpspmC(inpar,fishC,schaefer=TRUE),
  times=1000
)
out2 <- summary(out)[,2:8]
out2 <- rbind(out2,out2[2,]/out2[1,])
rownames(out2) <- c("simpspm","simpspmC","TimeRatio")


# R-chunk 42   Page 241 Table 6.7 code not included in the book
 #compare results from simpspm and simpspmC

kable(halftable(cbind(spmR,spmC)),row.names=TRUE,digits=c(4,4,4,4,4,4),caption='(ref:tab607)')


# R-chunk 43   Page 242 Table 6.8 code not included in the book
 #output from microbenchmark comparison of simpspm and simpspmC

kable(out2,row.names=TRUE,digits=c(3,3,3,3,3,3,3,0),caption='(ref:tab608)')

# R-chunk 44  Pages 242 - 243
 #How much does using simpspmC in do_MCMC speed the run time?
```

```
 #Assumes Rcpp code has run, eg source("Rcpp_functions.R")

set.seed(167423) #Can use getseed() to generate a suitable seed
beginR <- gettime()  #to enable estimate of time taken
setscale <- c(0.07,0.06,0.07,0.45)
reps <- 2000  #Not enough but sufficient for demonstration
param <- log(c(r=0.39,K=9200,Binit=3400,sigma=0.05))
resultR <- do_MCMC(chains=1,burnin=100,N=reps,thinstep=128,
                   inpar=param,infunk=negLL1,calcpred=simpspm,
                   calcdat=fishC,obsdat=log(abdat$cpue),schaefer=TRUE,
                   priorcalc=calcprior,scales=setscale)
timeR <- gettime() - beginR
cat("time = ",timeR,"\n")
cat("acceptance rate = ",resultR$arate," \n")
postR <- resultR[[1]][[1]]
set.seed(167423)     # Use the same pseudo-random numbers and the
beginC <- gettime()  # same starting point to make the comparsion
param <- log(c(r=0.39,K=9200,Binit=3400,sigma=0.05))
resultC <- do_MCMC(chains=1,burnin=100,N=reps,thinstep=128,
                  inpar=param,infunk=negLL1,calcpred=simpspmC,
                  calcdat=fishC,obsdat=log(abdat$cpue),schaefer=TRUE,
                  priorcalc=calcprior,scales=setscale)
timeC <- gettime() - beginC
cat("time = ",timeC,"\n")  # note the same acceptance rates
cat("acceptance rate = ",resultC$arate," \n")
postC <- resultC[[1]][[1]]
cat("Time Ratio = ",timeC/timeR)


# R-chunk 45  Page 243
#compare marginal distributions of the 2 chains  Fig 6.21

oldp <- par(no.readonly=TRUE)  # this line not in the book
par(mfrow=c(1,1),mai=c(0.45,0.45,0.05,0.05),oma=c(0.0,0,0.0,0.0))
par(cex=0.85, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
maxy <- getmax(c(density(postR[,"K"])$y,density(postC[,"K"])$y))
plot(density(postR[,"K"]),lwd=2,col=1,xlab="K",ylab="Density",
     main="",ylim=c(0,maxy),panel.first=grid())
lines(density(postC[,"K"]),lwd=3,col=5,lty=2)
par(oldp)  # return par to old settings; this line not in book


### Multiple Independent Chains
# R-chunk 46  Page 244
 #run multiple = 3 chains

setscale <- c(0.07,0.06,0.07,0.45)  # I only use a seed for
set.seed(9393074) # reproducibility within this book
reps <- 10000   # reset the timer
beginC <- gettime()  # remember a thinstep=256 is insufficient
resultC <- do_MCMC(chains=3,burnin=100,N=reps,thinstep=256,
                   inpar=param,infunk=negLL1,calcpred=simpspmC,
                   calcdat=fishC,obsdat=log(fishC[,"cpue"]),
```

```
                    priorcalc=calcprior,scales=setscale,schaefer=TRUE)
cat("time = ",gettime() - beginC," secs  \n")


# R-chunk 47  Pages 244 - 245
 #3 chain run using simpspmC, 10000 reps, thinstep=256 Fig 6.22

oldp <- par(no.readonly=TRUE)
par(mfrow=c(2,2),mai=c(0.4,0.45,0.05,0.05),oma=c(0.0,0,0.0,0.0))
par(cex=0.85, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
label <- c("r","K","Binit","sigma")
for (i in 1:4) {
   plot(density(resultC$result[[2]][,i]),lwd=2,col=1,
        xlab=label[i],ylab="Density",main="",panel.first=grid())
   lines(density(resultC$result[[1]][,i]),lwd=2,col=2)
   lines(density(resultC$result[[3]][,i]),lwd=2,col=3)
}
par(oldp)  # return par to old settings; this line not in book

# R-chunk 48  Pages 245 - 246
 #generate summary stats from the 3 MCMC chains

av <- matrix(0,nrow=3,ncol=4,dimnames=list(1:3,label))
sig2 <- av  # do the variance
relsig <- av # relative to mean of all chains
for (i in 1:3) {
  tmp <- resultC$result[[i]]
  av[i,] <- apply(tmp[,1:4],2,mean)
  sig2[i,] <- apply(tmp[,1:4],2,var)
}
cat("Average \n")
av
cat("\nVariance per chain \n")
sig2
cat("\n")
for (i in 1:4) relsig[,i] <- sig2[,i]/mean(sig2[,i])
cat("Variance Relative to Mean Variance of Chains \n")
relsig


# R-chunk 49  Pages 246 - 247
 #compare quantile from the 2 most widely separate MCMC chains

tmp <- resultC$result[[2]] # the 10000 values of each parameter
cat("Chain 2 \n")
msy1 <- tmp[,"r"]*tmp[,"K"]/4
ch1 <- apply(cbind(tmp[,1:4],msy1),2,quants)
round(ch1,4)
tmp <- resultC$result[[3]]
cat("Chain 3 \n")
msy2 <- tmp[,"r"]*tmp[,"K"]/4
ch2 <-  apply(cbind(tmp[,1:4],msy2),2,quants)
round(ch2,4)
```

```
cat("Percent difference ")
cat("\n2.5%  ",round(100*(ch1[1,] - ch2[1,])/ch1[1,],4),"\n")
cat("50%   ",round(100*(ch1[3,] - ch2[3,])/ch1[3,],4),"\n")
cat("97.5% ",round(100*(ch1[5,] - ch2[5,])/ch1[5,],4),"\n")


### Replicates Required to Avoid Serial Correlation
# R-chunk 50  Page 248
 #compare two higher thinning rates per parameter in MCMC

param <- log(c(r=0.39,K=9200,Binit=3400,sigma=0.05))
setscale <- c(0.07,0.06,0.07,0.45)
result1 <- do_MCMC(chains=1,burnin=100,N=2000,thinstep=1024,
                   inpar=param,infunk=negLL1,calcpred=simpspmC,
                   calcdat=fishC,obsdat=log(abdat$cpue),
                   priorcalc=calcprior,scales=setscale,schaefer=TRUE)
result2 <- do_MCMC(chains=1,burnin=50,N=1000,thinstep=2048,
                   inpar=param,infunk=negLL1,calcpred=simpspmC,
                   calcdat=fishC,obsdat=log(abdat$cpue),
                   priorcalc=calcprior,scales=setscale,schaefer=TRUE)


# R-chunk 51  Page 248
 #autocorrelation of 2 different thinning rate chains Fig6.23

posterior1 <- result1$result[[1]]
posterior2 <- result2$result[[1]]
label <- colnames(posterior1)[1:4]
oldp <- par(no.readonly=TRUE)
par(mfrow=c(4,2),mai=c(0.25,0.45,0.05,0.05),oma=c(1.0,0,1.0,0.0))
par(cex=0.85, mgp=c(1.35,0.35,0), font.axis=7,font=7,font.lab=7)
for (i in 1:4) {
  auto <- acf(posterior1[,i],type="correlation",plot=TRUE,
              ylab=label[i],lag.max=20,xlab="",ylim=c(0,0.3),lwd=2)
  if (i == 1) mtext(1024,side=3,line=-0.1,outer=FALSE,cex=1.2)
  auto <- acf(posterior2[,i],type="correlation",plot=TRUE,
              ylab=label[i],lag.max=20,xlab="",ylim=c(0,0.3),lwd=2)
  if (i == 1) mtext(2048,side=3,line=-0.1,outer=FALSE,cex=1.2)
}
mtext("Lag",side=1,line=-0.1,outer=TRUE,cex=1.2)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 52  Page 249
#visual comparison of 2 chains marginal densities  Fig 6.24

oldp <- parset(plots=c(2,2),cex=0.85)
label <- c("r","K","Binit","sigma")
for (i in 1:4) {
   plot(density(result1$result[[1]][,i]),lwd=4,col=1,xlab=label[i],
        ylab="Density",main="",panel.first=grid())
   lines(density(result2$result[[1]][,i]),lwd=2,col=5,lty=2)
}
par(oldp)  # return par to old settings; this line not in book
```

```
# R-chunk 53  Pages 250 - 251
 #tablulate a summary of the two different thinning rates.

cat("1024 thinning rate \n")
posterior <- result1$result[[1]]
msy <-posterior[,1]*posterior[,2]/4
tmp1 <- apply(cbind(posterior[,1:4],msy),2,quants)
rge <- apply(cbind(posterior[,1:4],msy),2,range)
tmp1 <- rbind(tmp1,rge[2,] - rge[1,])
rownames(tmp1)[6] <- "Range"
print(round(tmp1,4))
posterior2 <- result2$result[[1]]
msy2 <-posterior2[,1]*posterior2[,2]/4
cat("2048 thinning rate \n")
tmp2 <- apply(cbind(posterior2[,1:4],msy2),2,quants)
rge2 <- apply(cbind(posterior2[,1:4],msy2),2,range)
tmp2 <- rbind(tmp2,rge2[2,] - rge2[1,])
rownames(tmp2)[6] <- "Range"
print(round(tmp2,4))
cat("Inner 95% ranges and Differences between total ranges \n")
cat("95% 1 ",round((tmp1[5,] - tmp1[1,]),4),"\n")
cat("95% 2 ",round((tmp2[5,] - tmp2[1,]),4),"\n")
cat("Diff  ",round((tmp2[6,] - tmp1[6,]),4),"\n")

## End(Not run)
```

---

chapter7 *chapter7 The 67 R-code chunks from Surplus Production Models*

Description

chapter7 is not an active function but rather acts as a repository for the various example code chunks found in chapter7. There are 67 r-code chunks in chapter7 You should, of course, feel free to use and modify any of these example chunks in your own work.

Examples

```
## Not run:
# All the example code from  # Surplus Production Models
# Surplus Production Models
## Introduction
### Data Needs
### The Need for Contrast
### When are Catch-Rates Informative

# R-chunk 1  Page 256
 #Yellowfin-tuna data from Schaefer 12957

# R-chunk 2  Page 256 Table 7.1 code not in the book
```

```
data(schaef)
kable(halftable(schaef,subdiv=2),digits=c(0,0,0,4))

# R-chunk 3  Page 256
 #schaef fishery data and regress cpue and catch    Fig 7.1

oldp <- parset(plots=c(3,1),margin=c(0.35,0.4,0.05,0.05))
plot1(schaef[,"year"],schaef[,"catch"],ylab="Catch",xlab="Year",
      defpar=FALSE,lwd=2)
plot1(schaef[,"year"],schaef[,"cpue"],ylab="CPUE",xlab="Year",
      defpar=FALSE,lwd=2)
plot1(schaef[,"catch"],schaef[,"cpue"],type="p",ylab="CPUE",
      xlab="Catch",defpar=FALSE,pch=16,cex=1.0)
model <- lm(schaef[,"cpue"] ~ schaef[,"catch"])
abline(model,lwd=2,col=2)   # summary(model)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 4  Page 257
 #cross correlation between cpue and catch in schaef Fig 7.2

oldp <- parset(cex=0.85) #sets par values for a tidy base graphic
ccf(x=schaef[,"catch"],y=schaef[,"cpue"],type="correlation",
    ylab="Correlation",plot=TRUE)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 5  Page 257
 #now plot schaef data with timelag of 2 years on cpue   Fig 7.3

oldp <- parset(plots=c(3,1),margin=c(0.35,0.4,0.05,0.05))
plot1(schaef[1:20,"year"],schaef[1:20,"catch"],ylab="Catch",
      xlab="Year",defpar=FALSE,lwd=2)
plot1(schaef[3:22,"year"],schaef[3:22,"cpue"],ylab="CPUE",
      xlab="Year",defpar=FALSE,lwd=2)
plot1(schaef[1:20,"catch"],schaef[3:22,"cpue"],type="p",
      ylab="CPUE",xlab="Catch",defpar=FALSE,cex=1.0,pch=16)
model2 <- lm(schaef[3:22,"cpue"] ~ schaef[1:20,"catch"])
abline(model2,lwd=2,col=2)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 6  Page 259
 #write out a summary of he regression model2

summary(model2)

## Some Equations
### Production Functions
# R-chunk 7  Page 262
 #plot productivity and density-dependence functions Fig7.4

prodfun <- function(r,Bt,K,p) return((r*Bt/p)*(1-(Bt/K)^p))
densdep <- function(Bt,K,p) return((1/p)*(1-(Bt/K)^p))
r <- 0.75; K <- 1000.0; Bt <- 1:1000
sp <- prodfun(r,Bt,K,1.0)  # Schaefer equivalent
```

```
sp0 <- prodfun(r,Bt,K,p=1e-08)  # Fox equivalent
sp3 <- prodfun(r,Bt,K,3) #left skewed production, marine mammal?
oldp <- parset(plots=c(2,1),margin=c(0.35,0.4,0.1,0.05))
plot1(Bt,sp,type="l",lwd=2,xlab="Stock Size",
      ylab="Surplus Production",maxy=200,defpar=FALSE)
lines(Bt,sp0 * (max(sp)/max(sp0)),lwd=2,col=2,lty=2) # rescale
lines(Bt,sp3*(max(sp)/max(sp3)),lwd=3,col=3,lty=3)   # production
legend(275,100,cex=1.1,lty=1:3,c("p = 1.0 Schaefer","p = 1e-08 Fox",
                "p = 3 LeftSkewed"),col=c(1,2,3),lwd=3,bty="n")
plot1(Bt,densdep(Bt,K,p=1),xlab="Stock Size",defpar=FALSE,
      ylab="Density-Dependence",maxy=2.5,lwd=2)
lines(Bt,densdep(Bt,K,1e-08),lwd=2,col=2,lty=2)
lines(Bt,densdep(Bt,K,3),lwd=3,col=3,lty=3)
par(oldp)  # return par to old settings; this line not in book

### The Schaefer Model
### Sum of Squared Residuals
### Estimating Management Statistics

# R-chunk 8  Page 266
 #compare Schaefer and Fox MSY estimates for same parameters

param <- c(r=1.1,K=1000.0,Binit=800.0,sigma=0.075)
cat("MSY Schaefer = ",getMSY(param,p=1.0),"\n") # p=1 is default
cat("MSY Fox      = ",getMSY(param,p=1e-08),"\n")

### The Trouble with Equilibria
## Model Fitting
### A Possible Workflow for Stock Assessment
# R-chunk 9  Page 269
 #Initial model 'fit' to the initial parameter guess  Fig 7.5

data(schaef); schaef <- as.matrix(schaef)
param <- log(c(r=0.1,K=2250000,Binit=2250000,sigma=0.5))
negatL <- negLL(param,simpspm,schaef,logobs=log(schaef[,"cpue"]))
ans <- plotspmmod(inp=param,indat=schaef,schaefer=TRUE,
                addrmse=TRUE,plotprod=FALSE)

# R-chunk 10  Pages 270 - 271
 #Fit the model first using optim then nlm in sequence

param <- log(c(0.1,2250000,2250000,0.5))
pnams <- c("r","K","Binit","sigma")
best <- optim(par=param,fn=negLL,funk=simpspm,indat=schaef,
            logobs=log(schaef[,"cpue"]),method="BFGS")
outfit(best,digits=4,title="Optim",parnames = pnams)
cat("\n")
best2 <- nlm(negLL,best$par,funk=simpspm,indat=schaef,
           logobs=log(schaef[,"cpue"]))
outfit(best2,digits=4,title="nlm",parnames = pnams)

# R-chunk 11  Page 271
 #optimum fit. Defaults used in plotprod and schaefer Fig 7.6
```

```
ans <- plotspmmod(inp=best2$estimate,indat=schaef,addrmse=TRUE,
                  plotprod=TRUE)

# R-chunk 12  Page 272
 #the high-level structure of ans; try str(ans$Dynamics)

str(ans, width=65, strict.width="cut",max.level=1)

# R-chunk 13  Page 273
 #compare the parameteric MSY with the numerical MSY

round(ans$Dynamics$sumout,3)
cat("\n Productivity Statistics \n")
summspm(ans) # the q parameter needs more significantr digits

### Is the Analysis Robust?
# R-chunk 14  Page 274
 #conduct a robustness test on the Schaefer model fit

data(schaef); schaef <- as.matrix(schaef); reps <- 12
param <- log(c(r=0.15,K=2250000,Binit=2250000,sigma=0.5))
ansS <- fitSPM(pars=param,fish=schaef,schaefer=TRUE,    #use
               maxiter=1000,funk=simpspm,funkone=FALSE) #fitSPM
 #getseed() #generates random seed for repeatable results
set.seed(777852) #sets random number generator with a known seed
robout <- robustSPM(inpar=ansS$estimate,fish=schaef,N=reps,
                    scaler=40,verbose=FALSE,schaefer=TRUE,
                    funk=simpspm,funkone=FALSE)
 #use str(robout) to see the components included in the output


# R-chunk 15  Page 275 Table 7.2 code not in the book
 #outcome of robustness tests

kable(robout$results[,1:5],digits=c(3,4,3,4,3))
kable(robout$results[,6:11],digits=c(3,4,3,4,5,0))

# R-chunk 16 Pages 275 - 276
 #Repeat robustness test on fit to schaef data 100 times

set.seed(777854)
robout2 <- robustSPM(inpar=ansS$estimate,fish=schaef,N=100,
                     scaler=25,verbose=FALSE,schaefer=TRUE,
                     funk=simpspm,funkone=TRUE,steptol=1e-06)
lastbits <- tail(robout2$results[,6:11],10)

# R-chunk 17  Page 276 Table 7.3 code not in the book
 #last 10 rows of robustness test showing deviations

kable(lastbits,digits=c(5,1,1,4,5,0))

# R-chunk 18  Page 276
```

```
 # replicates from the robustness test        Fig 7.7

result <- robout2$results
oldp <- parset(plots=c(2,2),margin=c(0.35,0.45,0.05,0.05))
hist(result[,"r"],breaks=15,col=2,main="",xlab="r")
hist(result[,"K"],breaks=15,col=2,main="",xlab="K")
hist(result[,"Binit"],breaks=15,col=2,main="",xlab="Binit")
hist(result[,"MSY"],breaks=15,col=2,main="",xlab="MSY")
par(oldp)  # return par to old settings; this line not in book

# R-chunk 19  Page 277
 #robustSPM parameters against each other  Fig 7.8

pairs(result[,c("r","K","Binit","MSY")],upper.panel=NULL,pch=1)

### Using Different Data?
# R-chunk 20  Page 278
 #Now use the dataspm data-set, which is noisier

set.seed(777854) #other random seeds give different results
data(dataspm);   fish <- dataspm #to generalize the code
param <- log(c(r=0.24,K=5174,Binit=2846,sigma=0.164))
ans <- fitSPM(pars=param,fish=fish,schaefer=TRUE,maxiter=1000,
            funkone=TRUE)
out <- robustSPM(ans$estimate,fish,N=100,scaler=15, #making
               verbose=FALSE,funkone=TRUE) #scaler=10 gives
result <- tail(out$results[,6:11],10) #16 sub-optimal results


# R-chunk 21  Page 279 Table 7.4 code not in the book
 #last 10 trials of robustness on dataspm fit

kable(result,digits=c(4,2,2,4,4,3))

## Uncertainty
### Likelihood Profiles
# R-chunk 22  Page 280
 # Fig 7.9 Fit of optimum to the abdat data-set

data(abdat);     fish <- as.matrix(abdat)
colnames(fish) <- tolower(colnames(fish))  # just in case
pars <- log(c(r=0.4,K=9400,Binit=3400,sigma=0.05))
ans <- fitSPM(pars,fish,schaefer=TRUE) #Schaefer
answer <- plotspmmod(ans$estimate,abdat,schaefer=TRUE,addrmse=TRUE)

# R-chunk 23  Pages 280 - 282
 # likelihood profiles for r and K for fit to abdat  Fig 7.10
 #doprofile input terms are vector of values, fixed parameter
 #location, starting parameters, and free parameter locations.
 #all other input are assumed to be in the calling environment

doprofile <- function(val,loc,startest,indat,notfix=c(2:4)) {
  pname <- c("r","K","Binit","sigma","-veLL")
```

```
  numv <- length(val)
  outpar <- matrix(NA,nrow=numv,ncol=5,dimnames=list(val,pname))
  for (i in 1:numv) {  #
    param <- log(startest) # reset the parameters
    param[loc] <- log(val[i]) #insert new fixed value
    parinit <- param   # copy revised parameter vector
    bestmod <- nlm(f=negLLP,p=param,funk=simpspm,initpar=parinit,
                   indat=indat,logobs=log(indat[,"cpue"]),notfixed=notfix)
    outpar[i,] <- c(exp(bestmod$estimate),bestmod$minimum)
  }
  return(outpar)
}
rval <- seq(0.32,0.46,0.001)
outr <- doprofile(rval,loc=1,startest=c(rval[1],11500,5000,0.25),
                  indat=fish,notfix=c(2:4))
Kval <- seq(7200,11500,200)
outk <- doprofile(Kval,loc=2,c(0.4,7200,6500,0.3),indat=fish,notfix=c(1,3,4))
oldp <- parset(plots=c(2,1),cex=0.85,outmargin=c(0.5,0.5,0,0))
plotprofile(outr,var="r",defpar=FALSE,lwd=2) #MQMF function
plotprofile(outk,var="K",defpar=FALSE,lwd=2)
par(oldp)  # return par to old settings; this line not in book

### Bootstrap Confidence Intervals
# R-chunk 24  Page 283
 #find optimum Schaefer model fit to dataspm data-set Fig 7.11

data(dataspm)
fish <- as.matrix(dataspm)
colnames(fish) <- tolower(colnames(fish))
pars <- log(c(r=0.25,K=5500,Binit=3000,sigma=0.25))
ans <- fitSPM(pars,fish,schaefer=TRUE,maxiter=1000) #Schaefer
answer <- plotspmmod(ans$estimate,fish,schaefer=TRUE,addrmse=TRUE)

# R-chunk 25  Page 284
 #bootstrap the log-normal residuals from optimum model fit

set.seed(210368)
reps <- 1000 # can take 10 sec on a large Desktop. Be patient
 #startime <- Sys.time()  # schaefer=TRUE is the default
boots <- spmboot(ans$estimate,fishery=fish,iter=reps)
 #print(Sys.time() - startime) # how long did it take?
str(boots,max.level=1)

# R-chunk 26  Page 285
 #Summarize bootstrapped parameter estimates as quantiles  Table 7.6

bootpar <- boots$bootpar
rows <- colnames(bootpar)
columns <- c(c(0.025,0.05,0.5,0.95,0.975),"Mean")
bootCI <- matrix(NA,nrow=length(rows),ncol=length(columns),
                 dimnames=list(rows,columns))
for (i in 1:length(rows)) {
   tmp <- bootpar[,i]
```

```
   qtil <- quantile(tmp,probs=c(0.025,0.05,0.5,0.95,0.975),na.rm=TRUE)
   bootCI[i,] <- c(qtil,mean(tmp,na.rm=TRUE))
}
kable(bootCI,digits=c(4,4,4,4,4,4))

# R-chunk 28  Page 286
 #boostrap CI. Note use of uphist to expand scale  Fig 7.12

colf <- c(1,1,1,4); lwdf <- c(1,3,1,3); ltyf <- c(1,1,1,2)
colsf <- c(2,3,4,6)
oldp <- parset(plots=c(3,2))
hist(bootpar[,"r"],breaks=25,main="",xlab="r")
abline(v=c(bootCI["r",colsf]),col=colf,lwd=lwdf,lty=ltyf)
uphist(bootpar[,"K"],maxval=14000,breaks=25,main="",xlab="K")
abline(v=c(bootCI["K",colsf]),col=colf,lwd=lwdf,lty=ltyf)
hist(bootpar[,"Binit"],breaks=25,main="",xlab="Binit")
abline(v=c(bootCI["Binit",colsf]),col=colf,lwd=lwdf,lty=ltyf)
uphist(bootpar[,"MSY"],breaks=25,main="",xlab="MSY",maxval=450)
abline(v=c(bootCI["MSY",colsf]),col=colf,lwd=lwdf,lty=ltyf)
hist(bootpar[,"Depl"],breaks=25,main="",xlab="Final Depletion")
abline(v=c(bootCI["Depl",colsf]),col=colf,lwd=lwdf,lty=ltyf)
hist(bootpar[,"Harv"],breaks=25,main="",xlab="End Harvest Rate")
abline(v=c(bootCI["Harv",colsf]),col=colf,lwd=lwdf,lty=ltyf)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 29  Page 286
 #Fig7.13 1000 bootstrap trajectories for dataspm model fit

dynam <- boots$dynam
years <- fish[,"year"]
nyrs <- length(years)
oldp <- parset()
ymax <- getmax(c(dynam[,,"predCE"],fish[,"cpue"]))
plot(fish[,"year"],fish[,"cpue"],type="n",ylim=c(0,ymax),
     xlab="Year",ylab="CPUE",yaxs="i",panel.first = grid())
for (i in 1:reps) lines(years,dynam[i,,"predCE"],lwd=1,col=8)
lines(years,answer$Dynamics$outmat[1:nyrs,"predCE"],lwd=2,col=0)
points(years,fish[,"cpue"],cex=1.2,pch=16,col=1)
percs <- apply(dynam[,,"predCE"],2,quants)
arrows(x0=years,y0=percs["5%",],y1=percs["95%",],length=0.03,
       angle=90,code=3,col=0)
par(oldp)  # return par to old settings; this line not in book


# R-chunk 30  Page 288
 #Fit the Fox model to dataspm; note different parameters

pars <- log(c(r=0.15,K=6500,Binit=3000,sigma=0.20))
ansF <- fitSPM(pars,fish,schaefer=FALSE,maxiter=1000) #Fox version
bootsF <- spmboot(ansF$estimate,fishery=fish,iter=reps,schaefer=FALSE)
dynamF <- bootsF$dynam

# R-chunk 31 Pages 288 - 289
```

```
 # bootstrap trajectories from both model fits  Fig 7.14

oldp <- parset()
ymax <- getmax(c(dynam[,,"predCE"],fish[,"cpue"]))
plot(fish[,"year"],fish[,"cpue"],type="n",ylim=c(0,ymax),
     xlab="Year",ylab="CPUE",yaxs="i",panel.first = grid())
for (i in 1:reps) lines(years,dynamF[i,,"predCE"],lwd=1,col=1,lty=1)
for (i in 1:reps) lines(years,dynam[i,,"predCE"],lwd=1,col=8)
lines(years,answer$Dynamics$outmat[1:nyrs,"predCE"],lwd=2,col=0)
points(years,fish[,"cpue"],cex=1.1,pch=16,col=1)
percs <- apply(dynam[,,"predCE"],2,quants)
arrows(x0=years,y0=percs["5%",],y1=percs["95%",],length=0.03,
       angle=90,code=3,col=0)
legend(1985,0.35,c("Schaefer","Fox"),col=c(8,1),bty="n",lwd=3)
par(oldp)  # return par to old settings; this line not in book


### Parameter Correlations
# R-chunk 32  Page 290
 # plot variables against each other, use MQMF panel.cor  Fig 7.15

pairs(boots$bootpar[,c(1:4,6,7)],lower.panel=panel.smooth,
      upper.panel=panel.cor,gap=0,lwd=2,cex=0.5)

### Asymptotic Errors
# R-chunk 33  Page 290
 #Start the SPM analysis using asymptotic errors.

data(dataspm)    # Note the use of hess=TRUE in call to fitSPM
fish <- as.matrix(dataspm)     # using as.matrix for more speed
colnames(fish) <- tolower(colnames(fish))  # just in case
pars <- log(c(r=0.25,K=5200,Binit=2900,sigma=0.20))
ans <- fitSPM(pars,fish,schaefer=TRUE,maxiter=1000,hess=TRUE)

# R-chunk 34  page 291
 #The hessian matrix from the Schaefer fit to the dataspm data
outfit(ans)

# R-chunk 35  Page 292
 #calculate the var-covar matrix and the st errors

vcov <- solve(ans$hessian) # calculate variance-covariance matrix
label <- c("r","K", "Binit","sigma")
colnames(vcov) <- label; rownames(vcov) <- label
outvcov <- rbind(vcov,sqrt(diag(vcov)))
rownames(outvcov) <- c(label,"StErr")

# R-chunk 36  Page 290 Table 7.6 code not in the book
 # tabulate the variance covariance matrix and StErrs

kable(outvcov,digits=c(5,5,5,5))

# R-chunk 37  Pages 292 - 293
```

```
 #generate 1000 parameter vectors from multi-variate normal

library(mvtnorm)   # use RStudio, or install.packages("mvtnorm")
N <- 1000 # number of parameter vectors, use vcov from above
mvn <- length(fish[,"year"]) #matrix to store cpue trajectories
mvncpue <- matrix(0,nrow=N,ncol=mvn,dimnames=list(1:N,fish[,"year"]))
columns <- c("r","K","Binit","sigma")
optpar <- ans$estimate # Fill matrix with mvn parameter vectors
mvnpar <- matrix(exp(rmvnorm(N,mean=optpar,sigma=vcov)),nrow=N,
                 ncol=4,dimnames=list(1:N,columns))
msy <- mvnpar[,"r"]*mvnpar[,"K"]/4
nyr <- length(fish[,"year"])
depletion <- numeric(N) #now calculate N cpue series in linear space
for (i in 1:N) { # calculate dynamics for each parameter set
  dynamA <- spm(log(mvnpar[i,1:4]),fish)
  mvncpue[i,] <- dynamA$outmat[1:nyr,"predCE"]
  depletion[i] <- dynamA$outmat["2016","Depletion"]
}
mvnpar <- cbind(mvnpar,msy,depletion) # try head(mvnpar,10)

# R-chunk 38  Page 293
 #data and trajectories from 1000 MVN parameter vectors   Fig 7.16

oldp <-  plot1(fish[,"year"],fish[,"cpue"],type="p",xlab="Year",
              ylab="CPUE",maxy=2.0)
for (i in 1:N) lines(fish[,"year"],mvncpue[i,],col="grey",lwd=1)
points(fish[,"year"],fish[,"cpue"],pch=1,cex=1.3,col=1,lwd=2) # data
lines(fish[,"year"],exp(simpspm(optpar,fish)),lwd=2,col=1)# pred
percs <- apply(mvncpue,2,quants)  # obtain the quantiles
arrows(x0=fish[,"year"],y0=percs["5%",],y1=percs["95%",],length=0.03,
       angle=90,code=3,col=1) #add 90% quantiles
msy <- mvnpar[,"r"]*mvnpar[,"K"]/4  # 1000 MSY estimates
text(2010,1.75,paste0("MSY ",round(mean(msy),3)),cex=1.25,font=7)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 39  Pages 293 - 294
 #Isolate errant cpue trajectories Fig 7.17

pickd <- which(mvncpue[,"2016"] < 0.40)
oldp <- plot1(fish[,"year"],fish[,"cpue"],type="n",xlab="Year",
              ylab="CPUE",maxy=6.25)
for (i in 1:length(pickd))
  lines(fish[,"year"],mvncpue[pickd[i],],col=1,lwd=1)
points(fish[,"year"],fish[,"cpue"],pch=16,cex=1.25,col=4)
lines(fish[,"year"],exp(simpspm(optpar,fish)),lwd=3,col=2,lty=2)
par(oldp)  # return par to old settings; this line not in book


# R-chunk 40  Page 294
 #Use adhoc function to plot errant parameters Fig 7.18

oldp <- parset(plots=c(2,2),cex=0.85)
outplot <- function(var1,var2,pickdev) {
```

```
  plot1(mvnpar[,var1],mvnpar[,var2],type="p",pch=16,cex=1.0,
        defpar=FALSE,xlab=var1,ylab=var2,col=8)
  points(mvnpar[pickdev,var1],mvnpar[pickdev,var2],pch=16,cex=1.0)
}
outplot("r","K",pickd) # assumes mvnpar in working environment
outplot("sigma","Binit",pickd)
outplot("r","Binit",pickd)
outplot("K","Binit",pickd)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 41  Page 296
 #asymptotically sampled parameter vectors  Fig 7.19

pairs(mvnpar,lower.panel=panel.smooth, upper.panel=panel.cor,
      gap=0,cex=0.25,lwd=2)


# R-chunk 42  Page 297
 # Get the ranges of parameters from bootstrap and asymptotic

bt <- apply(bootpar,2,range)[,c(1:4,6,7)]
ay <- apply(mvnpar,2,range)
out <- rbind(bt,ay)
rownames(out) <- c("MinBoot","MaxBoot","MinAsym","MaxAsym")

# R-chunk 43  Page 297 Table 7.7 code not in the book
 #tabulate ranges from two approsches

kable(out,digits=c(4,3,3,4,3,4))

### Sometimes Asymptotic Errors Work
# R-chunk 44  Pages 297 - 298
 #repeat asymptotice errors using abdat data-set Figure 7.20

data(abdat)
fish <- as.matrix(abdat)
pars <- log(c(r=0.4,K=9400,Binit=3400,sigma=0.05))
ansA <- fitSPM(pars,fish,schaefer=TRUE,maxiter=1000,hess=TRUE)
vcovA <- solve(ansA$hessian) # calculate var-covar matrix
mvn <- length(fish[,"year"])
N <- 1000   # replicates
mvncpueA <- matrix(0,nrow=N,ncol=mvn,dimnames=list(1:N,fish[,"year"]))
columns <- c("r","K","Binit","sigma")
optparA <- ansA$estimate  # Fill matrix of parameter vectors
mvnparA <- matrix(exp(rmvnorm(N,mean=optparA,sigma=vcovA)),
                  nrow=N,ncol=4,dimnames=list(1:N,columns))
msy <- mvnparA[,"r"]*mvnparA[,"K"]/4
for (i in 1:N) mvncpueA[i,]<-exp(simpspm(log(mvnparA[i,]),fish))
mvnparA <- cbind(mvnparA,msy)
oldp <- plot1(fish[,"year"],fish[,"cpue"],type="p",xlab="Year",
              ylab="CPUE",maxy=2.5)
for (i in 1:N) lines(fish[,"year"],mvncpueA[i,],col=8,lwd=1)
points(fish[,"year"],fish[,"cpue"],pch=16,cex=1.0) #orig data
```

```
lines(fish[,"year"],exp(simpspm(optparA,fish)),lwd=2,col=0)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 45  Page 298
 #plot asymptotically sampled parameter vectors Figure 7.21

pairs(mvnparA,lower.panel=panel.smooth, upper.panel=panel.cor,
      gap=0,pch=16,col=rgb(red=0,green=0,blue=0,alpha = 1/10))

### Bayesian Posteriors
# R-chunk 46  Page 299
 #Fit the Fox Model to the abdat data Figure 7.22

data(abdat); fish <- as.matrix(abdat)
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
foxmod <- nlm(f=negLL1,p=param,funk=simpspm,indat=fish,
              logobs=log(fish[,"cpue"]),iterlim=1000,schaefer=FALSE)
optpar <- exp(foxmod$estimate)
ans <- plotspmmod(inp=foxmod$estimate,indat=fish,schaefer=FALSE,
                addrmse=TRUE, plotprod=TRUE)

# R-chunk 47  Page 301
 # Conduct an MCMC using simpspmC on the abdat Fox SPM
 # This means you will need to compile simpspmC from appendix
set.seed(698381) #for repeatability, possibly only on Windows10
begin <- gettime()  # to enable the time taken to be calculated
inscale <- c(0.07,0.05,0.09,0.45) #note large value for sigma
pars <- log(c(r=0.205,K=11300,Binit=3200,sigma=0.044))
result <- do_MCMC(chains=1,burnin=50,N=2000,thinstep=512,
                  inpar=pars,infunk=negLL,calcpred=simpspmC,
                  obsdat=log(fish[,"cpue"]),calcdat=fish,
                  priorcalc=calcprior,scales=inscale,schaefer=FALSE)
 # alternatively, use simpspm, but that will take longer.
cat("acceptance rate = ",result$arate," \n")
cat("time = ",gettime() - begin,"\n")
post1 <- result[[1]][[1]]
p <- 1e-08
msy <- post1[,"r"]*post1[,"K"]/((p + 1)^((p+1)/p))

# R-chunk 48 Page 302
 #pairwise comparison for MCMC of Fox model on abdat  Fig 7.23

pairs(cbind(post1[,1:4],msy),upper.panel = panel.cor,lwd=2,cex=0.2,
      lower.panel=panel.smooth,col=1,gap=0.1)

# R-chunk 49  Page 302
 # marginal distributions of 3 parameters and msy  Figure 7.24

oldp <- parset(plots=c(2,2), cex=0.85)
plot(density(post1[,"r"]),lwd=2,main="",xlab="r") #plot has a method
plot(density(post1[,"K"]),lwd=2,main="",xlab="K")   #for output from
plot(density(post1[,"Binit"]),lwd=2,main="",xlab="Binit")  # density
plot(density(msy),lwd=2,main="",xlab="MSY")   #try str(density(msy))
```

```
par(oldp)  # return par to old settings; this line not in book

# R-chunk 50  Page 304
 #MCMC r and K parameters, approx 50 + 90% contours. Fig7.25

puttxt <- function(xs,xvar,ys,yvar,lvar,lab="",sigd=0) {
  text(xs*xvar[2],ys*yvar[2],makelabel(lab,lvar,sep="  ",
       sigdig=sigd),cex=1.2,font=7,pos=4)
} # end of puttxt - a quick utility function
kran <- range(post1[,"K"]);  rran <- range(post1[,"r"])
mran <- range(msy)         #ranges used in the plots
oldp <- parset(plots=c(1,2),margin=c(0.35,0.35,0.05,0.1)) #plot r vs K
plot(post1[,"K"],post1[,"r"],type="p",cex=0.5,xlim=kran,
     ylim=rran,col="grey",xlab="K",ylab="r",panel.first=grid())
points(optpar[2],optpar[1],pch=16,col=1,cex=1.75) # center
addcontours(post1[,"K"],post1[,"r"],kran,rran,  #if fails make
            contval=c(0.5,0.9),lwd=2,col=1)   #contval smaller
puttxt(0.7,kran,0.97,rran,kran,"K= ",sigd=0)
puttxt(0.7,kran,0.94,rran,rran,"r= ",sigd=4)
plot(post1[,"K"],msy,type="p",cex=0.5,xlim=kran,  # K vs msy
     ylim=mran,col="grey",xlab="K",ylab="MSY",panel.first=grid())
points(optpar[2],getMSY(optpar,p),pch=16,col=1,cex=1.75)#center
addcontours(post1[,"K"],msy,kran,mran,contval=c(0.5,0.9),lwd=2,col=1)
puttxt(0.6,kran,0.99,mran,kran,"K= ",sigd=0)
puttxt(0.6,kran,0.97,mran,mran,"MSY= ",sigd=3)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 51  Page 305
 #Traces for the Fox model parameters from the MCMC  Fig7.26

oldp <- parset(plots=c(4,1),margin=c(0.3,0.45,0.05,0.05),
               outmargin = c(1,0,0,0),cex=0.85)
label <- colnames(post1)
N <- dim(post1)[1]
for (i in 1:3) {
  plot(1:N,post1[,i],type="l",lwd=1,ylab=label[i],xlab="")
  abline(h=median(post1[,i]),col=2)
}
msy <- post1[,1]*post1[,2]/4
plot(1:N,msy,type="l",lwd=1,ylab="MSY",xlab="")
abline(h=median(msy),col=2)
mtext("Step",side=1,outer=T,line=0.0,font=7,cex=1.1)
par(oldp)  # return par to old settings; this line not in book

# R-chunk 52  Page 306
 #Do five chains of the same length for the Fox model

set.seed(6396679)  # Note all chains start from same place, which is
inscale <- c(0.07,0.05,0.09,0.45)  # suboptimal, but still the chains
pars <- log(c(r=0.205,K=11300,Binit=3220,sigma=0.044))  # differ
result <- do_MCMC(chains=5,burnin=50,N=2000,thinstep=512,
                  inpar=pars,infunk=negLL1,calcpred=simpspmC,
                  obsdat=log(fish[,"cpue"]),calcdat=fish,
```

```
                priorcalc=calcprior,scales=inscale,
                schaefer=FALSE)
cat("acceptance rate = ",result$arate," \n") # always check this

# R-chunk 53  Page 306
 #Now plot marginal posteriors from 5 Fox model chains    Fig7.27

oldp <- parset(plots=c(2,1),cex=0.85,margin=c(0.4,0.4,0.05,0.05))
post <- result[[1]][[1]]
plot(density(post[,"K"]),lwd=2,col=1,main="",xlab="K",
     ylim=c(0,4.4e-04),panel.first=grid())
for (i in 2:5) lines(density(result$result[[i]][,"K"]),lwd=2,col=i)
p <- 1e-08
post <- result$result[[1]]
msy <-  post[,"r"]*post[,"K"]/((p + 1)^((p+1)/p))
plot(density(msy),lwd=2,col=1,main="",xlab="MSY",type="l",
     ylim=c(0,0.0175),panel.first=grid())
for (i in 2:5) {
  post <- result$result[[i]]
  msy <-  post[,"r"]*post[,"K"]/((p + 1)^((p+1)/p))
  lines(density(msy),lwd=2,col=i)
}
par(oldp)  # return par to old settings; this line not in book

# R-chunk 54  Page 307
 # get quantiles of each chain

probs <- c(0.025,0.05,0.5,0.95,0.975)
storeQ <- matrix(0,nrow=6,ncol=5,dimnames=list(1:6,probs))
for (i in 1:5) storeQ[i,] <- quants(result$result[[i]][,"K"])
x <- apply(storeQ[1:5,],2,range)
storeQ[6,] <- 100*(x[2,] - x[1,])/x[2,]

# R-chunk 55 Page 308 Table 7.8 code not in the book
 #tabulate qunatiles of the five chains

kable(storeQ,digits=c(3,3,3,3,3))

## Management Advice
### Two Views of Risk
### Harvest Strategies
## Risk Assessment Projections
### Deterministic Projections

# R-chunk 56  Pages 310 - 311
 #Prepare Fox model on abdat data for future projections Fig7.28

data(abdat); fish <- as.matrix(abdat)
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
bestmod <- nlm(f=negLL1,p=param,funk=simpspm,schaefer=FALSE,
               logobs=log(fish[,"cpue"]),indat=fish,hessian=TRUE)
optpar <- exp(bestmod$estimate)
ans <- plotspmmod(inp=bestmod$estimate,indat=fish,schaefer=FALSE,
```

```
                target=0.4,addrmse=TRUE, plotprod=FALSE)


# R-chunk 57 Page 312

out <- spm(bestmod$estimate,indat=fish,schaefer=FALSE)
str(out, width=65, strict.width="cut")

# R-chunk 58  Page 312 Table 7.9 code not in the book
 #

kable(out$outmat[1:10,],digits=c(0,4,4,4,4,4,4))

# R-chunk 59  Page 313
 #  Fig 7.29

catches <- seq(700,1000,50)   # projyr=10 is the default
projans <- spmprojDet(spmobj=out,projcatch=catches,plotout=TRUE)

### Accounting for Uncertainty
### Using Asymptotic Errors
# R-chunk 60  Page 315
 # generate parameter vectors from a multivariate normal
 # project dynamics under a constant catch of 900t

library(mvtnorm)
matpar <- parasympt(bestmod,N=1000) #generate parameter vectors
projs <- spmproj(matpar,fish,projyr=10,constC=900)#do dynamics

# R-chunk 61  Page 315
 # Fig 7.30  1000 replicate projections asymptotic errors

outp <- plotproj(projs,out,qprob=c(0.1,0.5),refpts=c(0.2,0.4))

### Using Bootstrap Parameter Vectors
# R-chunk 62  Page 316
 #bootstrap generation of plausible parameter vectors for Fox

reps <- 1000
boots <- spmboot(bestmod$estimate,fishery=fish,iter=reps,schaefer=FALSE)
matparb <- boots$bootpar[,1:4] #examine using head(matparb,20)

# R-chunk 63  Page 316
 #bootstrap projections. Lower case b for boostrap  Fig7.31

projb <- spmproj(matparb,fish,projyr=10,constC=900)
outb <- plotproj(projb,out,qprob=c(0.1,0.5),refpts=c(0.2,0.4))

### Using Samples from a Bayesian Posterior
# R-chunk 64  Pages 317 - 318
 #Generate 1000 parameter vectors from Bayesian posterior

param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
```

```
set.seed(444608)
N <- 1000
result <- do_MCMC(chains=1,burnin=100,N=N,thinstep=2048,
                  inpar=param,infunk=negLL,calcpred=simpspmC,
                  calcdat=fish,obsdat=log(fish[,"cpue"]),
                  priorcalc=calcprior,schaefer=FALSE,
                  scales=c(0.065,0.055,0.1,0.475))
parB <- result[[1]][[1]] #capital B for Bayesian
cat("Acceptance Rate = ",result[[2]],"\n")

# R-chunk 65  Page 318
 # auto-correlation, or lack of, and the K trace Fig 7.32

oldp <- parset(plots=c(2,1),cex=0.85)
acf(parB[,2],lwd=2)
plot(1:N,parB[,2],type="l",ylab="K",ylim=c(8000,19000),xlab="")
par(oldp)  # return par to old settings; this line not in book

# R-chunk 66  Page 318
 #  Fig 7.33

matparB <- as.matrix(parB[,1:4]) # B for Bayesian
projs <- spmproj(matparB,fish,constC=900,projyr=10) # project them
plotproj(projs,out,qprob=c(0.1,0.5),refpts=c(0.2,0.4)) #projections

## Concluding Remarks
## Appendix: The Use of Rcpp to Replace simpspm
# R-chunk 67  Page 321

library(Rcpp)
cppFunction('NumericVector simpspmC(NumericVector pars,
             NumericMatrix indat, LogicalVector schaefer) {
   int nyrs = indat.nrow();
   NumericVector predce(nyrs);
   NumericVector biom(nyrs+1);
   double Bt, qval;
   double sumq = 0.0;
   double p = 0.00000001;
   if (schaefer(0) == TRUE) {
     p = 1.0;
   }
   NumericVector ep = exp(pars);
   biom[0] = ep[2];
   for (int i = 0; i < nyrs; i++) {
      Bt = biom[i];
      biom[(i+1)] = Bt + (ep[0]/p)*Bt*(1 - pow((Bt/ep[1]),p)) -
                         indat(i,1);
      if (biom[(i+1)] < 40.0) biom[(i+1)] = 40.0;
      sumq += log(indat(i,2)/biom[i]);
    }
    qval = exp(sumq/nyrs);
    for (int i = 0; i < nyrs; i++) {
      predce[i] = log(biom[i] * qval);
```

```
    }
    return predce;
 }')

## End(Not run)
```

---

countgtone *countgtone used in apply to count the number > 1 in a vector*

Description

countgtone used in the base function 'apply', or 'tapply' to count the number > 1 in a vector

Usage

```
countgtone(invect)
```

Arguments

invect	vector of values

Value

the number of values in the vector > 1

Examples

```
x <- matrix(trunc(runif(20)*10),nrow=4,ncol=5)
print(x)
apply(x,1,countgtone)
```

---

countgtzero *countgtzero used in apply to count numbers > zero in a vector*

Description

countgtzero used in the base function 'apply', or 'tapply' to count how many numbers are greater than zero in a vector

Usage

```
countgtzero(invect)
```

Arguments

invect	vector of values

Value

An integer counting how many numbers are > 0

Examples

```
x <- matrix(trunc(runif(20)*10),nrow=4,ncol=5)
print(x)
apply(x,1,countgtzero) # count by rows
apply(x,2,countgtzero) # count by columns
```

`countNAs` *countNAs used in apply to count the number of NAs in a vector*

Description

countNAs used in the base function 'apply', or 'tapply' to count the number of NAs in a vector

Usage

```
countNAs(invect)
```

Arguments

`invect`	vector of values

Value

A single value of zero or the number of NAs

Examples

```
x <- matrix(trunc(runif(20)*10),nrow=4,ncol=5)
diag(x) <- NA
print(x)
apply(x,1,countNAs)
```

countones *countones used in apply to count the number of ones in a vector*

Description

countones used in the base function 'apply', or 'tapply' to count the number of ones in a vector

Usage

```
countones(invect)
```

Arguments

invect	vector of values

Value

A single value of zero or the number of ones

Examples

```
x <- matrix(trunc(runif(20)*10),nrow=4,ncol=5)
print(x)
apply(x,1,countones)  # by rows
```

countzeros *countzeros used in apply to count the number of zeros in a vector*

Description

countzeros used in the base function 'apply', or 'tapply' to count the number of zeros in a vector

Usage

```
countzeros(invect)
```

Arguments

invect	vector of values

Value

A single value of zero or the number of zeros

Examples

```
x <- matrix(trunc(runif(20)*10),nrow=4,ncol=5)
print(x)
apply(x,1,countzeros) # count by rows
apply(x,2,countzeros) # count by columns
```

`dataspm` *dataspm A data.frame of fisheries catch and cpue data*

Description

A data.frame containing 31 years of catch, standardized cpue, number of records, and the unstandardized geometric mean cpue for Pink Ling (*Genypterus blacodes*). The fisheries data can be used in the surplus production modelling in Chapter 7. Initial parameter estimates very close to the optimum values could be param <- log(c(r=0.25, K=5500, Binit=3000,sigma=0.2)) for the Schaefer model and log(c(r=0.15, K=6500, Binit=3000, sigma=0.2)) for the Fox model

Format

A 31 x 5 data.frame

year the year from 1986 t0 2016

catch the catch in tonnes to the nearest 100kg

cpue the standardized cpue scaled to the mean of the series

records the number of records making up the yearly totals

geom the naive geometric mean cpue of the raw data as kg/hr, also rescaled to the mean of the series

Subjects

- Fishery data-set
- Surplus Production Modelling
- Log-Normal likelihoods

Source

Haddon, M. and M. Sporcic (2017) Catch rate standardizations for selected SESSF Species (data to 2016) pp 43-406 in Tuck, G.N. (*ed*) *Stock Assessment for the Southern and Eastern scalefish and shark fishery 2016 and 2017.* 837 pp. ISBN 978-1-4863-1012-8 data extracted from Table 7.96 PinkLing4050 page 216.

Examples

```
data(dataspm)
oldpar <- par(no.readonly=TRUE)
plot(dataspm$year,dataspm$geom,type="l",lwd=2,xlab="Year",
ylab="CPUE",panel.first=grid())
lines(dataspm$year,dataspm$cpue*mean(dataspm$geom),lwd=2,col=2)
legend("topright",c("cpue","geom"), col=c(1,2), lwd=3, bty="n",
cex=1.2)
par(oldpar)
```

`discretelogistic` *discretelogistic example and figure 3.2 Discrete logistic model*

Description

discretelogistic is an implementation of equation 3.1 in the Simple Population Models chapter 3. It enables the exploration of the dynamics of the Discrete logistic model, based around the classical Schaefer model.

The time-series nature of population growth is clear from the fact that Nt+1 is a function of Nt. One can thus expect serial correlation. Setting the r parameter to <= 1.0, would generate monotonically damped equilibria. r values between 1 < r < 2.03 would generate damped oscillatory equilibria, r values from 2.03 < r < 2.43 should generate stable limit cycles on a cycle of 2, 2.43 < r < 2.54 gives stable limit cycles of cycle 4, then 2.54 < r < 2.57 gives cycles > 4, and ~2.575 < r gives chaos (though r = 2.63 appears to generate a repeat period of six!). discretelogistic should be used in conjunction with plot, for which an S3 method has been defined plot.dynpop. As the dynamics are obviously sequential (i.e. n at t+1 is dependent upon n at t) the last line of the dynamics is removed to avoid an empty nt1 row.

Usage

```
discretelogistic(r = 0.5, K = 1000, N0 = 50, Ct = 0, Yrs = 50, p = 1)
```

Arguments

`r`	intrinsic rate of population increase; default = 0.5
`K`	carrying capacity; default = 1000.0
`N0`	Initial population size; default=50.0 = 5 percent depletion. Note that the term 'depletion' can be confusing. Surely 50 remaining from 1000 should be a depletion of 95 percent? But no, it is deemed to be the complement of 5 percent.
`Ct`	annual catch default = 0.0
`Yrs`	years of population growth, default=50
`p`	the production curve asymmetry parameter. the default value of 1.0 gives the classical Schaefer model. A value of 1e-08 approximates the Fox model where the mode is skewed to the left of centre.

Value

invisibly returns a matrix of year, nt, and nt1

Examples

```
discretelogistic(0.5,1000.0,25,0.0,50) # asymptotic
discretelogistic(2.5,1000.0,25,0.0,50) # 4-phase stable limit
ans <- discretelogistic(r=2.55,K=1000.0,N0=100,Ct=95.0,Yrs=100)
plot(ans)  # uses an S3 plot method for dynpop objects
```

---

domed *domed calculates domed selectivity curves*

Description

domed uses 6 parameters and a set of mean size or age classes to calculate a domed selectivity curve with a maximum of 1.0 (rescaling can be done outside the function), but has parameters for the selectivity of the initial and final size/age classes. There is an ascending limb and a descending limb with the potential of a plateau in between. The six parameters are 1) the age/size where selectivity first becomes 1.0, 2) the size/age where selectivity first begins to decline, 3) the steepness of the ascending limb, 4) the steepness of the descending limb, 5) the selectivity of the first age/size class, and 6) the selectivity of the last age/size class.

Usage

```
domed(p, L)
```

Arguments

p	a vector of six parameters.
L	a vector of the mean of nL age/size classes

Value

a vector of selectivities

References

Methot, R.D. and C.R, Wetzel (2013) Stock synthesis: A biological and statistical framework for fish stock assessment and fishery management. Supplementary material, Appendix A. Equs A1.30 onwards. *Fisheries Research* 142:86-99.

Examples

```
L <- seq(1,30,1)
p <- c(10,11,16,33,-5,-2)
sel <- domed(p,L)
plot1(L,sel,xlab="Age",ylab="Selectivity",lwd=2)
```

do_MCMC *do_MCMC conducts an MCMC using Gibbs within Metropolis*

Description

do_MCMC conducts a Gibbs within Metropolis algorithm. One can define the number of chains, the burnin before candidate parameter vectors and associated probabilities are stored, the total number of candidate vectors to keep and the step or thinning rate between accepting a result into the Markov Chain. One needs to input three functions: 1) 'infunk' to calculate the likelihoods, 2) 'calcpred' used within 'infunk' to calculate the predicted values used to compare with the observed (found in 'obsdat'), and 3) 'priorcalc' used to calculate the prior probabilities of the various parameters in each trial. (N + burnin) * thinstep iterations are made in total although only N are stored. The jumping function uses random normal deviates (mean=0, sd=1) to combine with each parameter value (after multiplication by the specific scaling factor held in the scales argument). Determination of suitable scaling values is generally done empirically, perhaps by trialing a small number of iterations to begin with. Multiple chains would be usual and the thinstep would be larger eg. 128, 256, or 512, but it would take 8, 16, or 32 times longer, depending on the number of parameters, these numbers are for four parameters only. The scales are usually empirically set to obtain an acceptance rate between 20 - 40 It is also usual to run numerous diagnostic plots on the outputs to ensure convergence onto the final stationary distribution. There are three main loops: 1) total number of iterations (N + burnin)* thinstep, used by priorcalc, 2) thinstep/(number of parameters) so that at least all parameters are stepped through at least once (thinstep = np) before any combinations are considered for acceptance, this means that the true thinning rate is thinstep/np, and 3) the number of parameters loop, that steps through the np parameters varying each one, one at a time.

Usage

```
do_MCMC(
  chains,
  burnin,
  N,
  thinstep,
  inpar,
  infunk,
  calcpred,
  calcdat,
  obsdat,
  priorcalc,
  scales,
  ...
)
```

Arguments

chains	the number of independent MCMC chains produced
burnin	the number of steps made before candidate parameter vectors begin to be kept.

N	the total number of posterior parameter vectors retained
thinstep	the number of interations before a vector is kept; must be divisible by the number of parameters
inpar	the initial parameter values (usually log-transformed)
infunk	the function used to calculate the negative log-likelihood
calcpred	the function used by infunk to calculate the predicted values for comparison with obsdat; defaults to simpspm.
calcdat	the data used by calcpred to calculate the predicted values
obsdat	the observed data (on the same scale as the predicted, ie usually log-transformed), against with the predicted values are compared.
priorcalc	a function used to calculate the prior probability for each of the parameters in each trial parameter vector.
scales	The re-scaling factors for each parameter to convert the normal random deviates into +/- values on a scale that leads to acceptance rates of between 20 and 40percent.
...	needed by the simpspm function = calcpred

Value

a list of the result chains, the acceptance and rejection rates

Examples

```
data(abdat); fish <- as.matrix(abdat) # to increase speed
param <- log(c(0.4,9400,3400,0.05))
N <- 500  # usually very, very  many more  10s of 1000s
result <- do_MCMC(chains=1,burnin=20,N=N,thinstep=8,inpar=param,
                  infunk=negLL,calcpred=simpspm,calcdat=fish,
                  obsdat=log(fish[,"cpue"]),priorcalc=calcprior,
                  scales=c(0.06,0.05,0.06,0.42))
# a thinstep of 8 is whofully inadequate, see the runs in the plots
cat("Acceptance Rate = ",result[[2]],"\n")
cat("Failure Rate    = ",result[[3]],"\n")
oldpar <- par(no.readonly=TRUE)
#plotprep(width=6,height=5,newdev=FALSE)
out <- result[[1]][[1]] # get the list containing the matrix
pairs(out[,1:4],col=rgb(1,0,0,1/5)) # adjust the 1/5 to suit N

parset(plots=c(1,2)) # Note the serial correlation in each trace
plot1(1:N,out[,1],ylab="r",xlab="Replicate",defpar=FALSE)
plot1(1:N,out[,2],ylab="K",xlab="Replicate",defpar=FALSE)
par(oldpar)
```

fabens *fabens calculates predicted growth increment for tagging data*

Description

fabens requires at least two parameters, Linf and K from the von Bertalanffy growth curve in a vector, as well as the initial length and the change in time between tag release and recapture. It then calculates the expected growth increment.

Usage

```
fabens(par, indat, initL = "l1", delT = "dt")
```

Arguments

par	a vector of at least Linf, and K from the von Bertalanffy growth curve
indat	the matrix or data.frame of data columns containing at least the initial lengths and the deltaT, time intervals between tag release and recapture.
initL	column name of the initial lengths within indat, default="l1"
delT	column name of the time interval, deltaT, within indat, default="dt"

Value

a vector of predicted growth increments

Examples

```
data(blackisland)
oldpar <- par(no.readonly=TRUE)
plot(blackisland$l1,blackisland$dl,type="p",pch=16,
xlab="Initial Length mm",ylab="Growth Increment mm",
panel.first=grid())
abline(h=0)
param <- c(170, 0.3, 4.0) # Linf, K, sigma
predDL <- fabens(param,blackisland,initL="l1",delT="dt")
lines(blackisland$l1,predDL,col=2,lwd=2)
par(oldpar)
```

facttonum *facttonum converts a vector of numeric factors into numbers*

Description

facttonum converts a vector of numeric factors into numbers. If the factors are not numeric then the outcome will be a series of NA. It is up to you to apply this function only to numeric factors. A warning will be thrown if the resulting output vector contains NAs

Usage

```
facttonum(invect)
```

Arguments

invect	the vector of numeric factors to be converted back to numbers

Value

an output vector of numbers instead of the input factors

Examples

```
DepCat <- as.factor(rep(seq(100,600,100),2))
print(DepCat)
5 * DepCat[3]    # throws an error, cannot multiply a factor
as.numeric(DepCat) # ordinal values of the factors
as.numeric(levels(DepCat)) #converts the levels not the replicates
DepCat <- facttonum(DepCat)
DepCat / 2.0     # now all values back to values
x <- factor(letters)
facttonum(x)     # this would be silly, characters are not numbers
```

fitSPM *fitSPM fits a surplus production model*

Description

fitSPM fits a surplus production model (either Schaefer or Fox) by first applying optim (using Nelder-Mead) and then nlm. Being automated it is recommended that this only be used once plausible initial parameters have been identified (through rules of thumb or trial and error). It uses negLL1 to apply a negative log-likelihood, assuming log-normal residual errors and uses a penalty to prevent the first parameter (r) from becoming < 0.0. If that is not wanted then set funkone to FALSE, which would then use negLL by itself. The output object is the usual object output from nlm, which can be neatly printed using the MQMF function outfit. The $estimate values can be

used in plotspmmod to plot the outcome, or in spmboot to conduct bootstrap sampling of the residuals from the CPUE model fit, to gain an appreciation of any uncertainty in the analysis. Because it requires log(parameters) it does not use the magnitude function to set the values of the parscale parameters.

Usage

```
fitSPM(
  pars,
  fish,
  schaefer = TRUE,
  maxiter = 1000,
  funk = simpspm,
  funkone = TRUE,
  hess = FALSE,
  steptol = 1e-06
)
```

Arguments

pars	the initial parameter values to start the search for the optimum. These need to be on the log-scale (log-transformed)
fish	the matrix containing the fishery data 'year', 'catch', and 'cpue' as a minimum. These exact column names are required.
schaefer	if TRUE, the default, then simpspm is used to fit the Schaefer model. If FALSE then the approximate Fox model is fitted by setting the p parameter to 1e-08 inside simpspm.
maxiter	the maximum number of iterations to be used by nlm
funk	the function used to generate the predicted cpue
funkone	default = TRUE. Means use negLL1, which constrains the first parameter (r) to be greater than 0. If FALSE then use negLL which is identical to negLL1 but lacks the constraint.
hess	default is FALSE; should one calculate the hessian matrix?
steptol	the internal step tolerance, required in case nlm reports the steptol as being too small. defaults to 1e-06

Value

an nlm output object as a list

Examples

```
data(dataspm)
dataspm <- as.matrix(dataspm) # faster than a data.frame
pars <- log(c(r=0.2,K=6000,Binit=2800,sigma=0.2))
ans <- fitSPM(pars,fish=dataspm,schaefer=TRUE,maxiter=1000)
outfit(ans)   # Schaefer model  -12.12879
ansF <- fitSPM(pars,dataspm,schaefer=FALSE,maxiter=1000)
outfit(ansF)  # Fox model       -12.35283
```

freqMean *freqMean calculates the mean and stdev of count data*

Description

freqMean calculates the mean and stdev of count data it requires both the values and their associated counts and return a vector of two numbers.

Usage

```
freqMean(values, infreqs)
```

Arguments

values	the values for which there are counts
infreqs	the counts for each of the values empty cells can be either 0 or NA

Value

a vector containing the mean and st.dev.

Examples

```
vals <- c(1,2,3,4,5)
counts <- c(3,NA,7,4,2)
freqMean(vals,counts)  # should give 3.125 and 1.258306
```

getlag *getlag is used to look for the response of cpue to previous catches*

Description

getlag is a wrapper for the ccf function (cross correlation) that is used within the spm analyses to determine at what negative lag, if any, cpue data is informative about the stock dynamics beyond any information already available in the catch data. If the cpue is directly correlated with catches (lag=0 has a strong correlation) then cpue will not add much more information to an analysis. Only if there is a significant negative correlation is it likely that the cpue will increase the information available and make it more likely that an assessment model may be able to be fitted meaningfully to the available data. If there is no significant negative correlations then it becomes much more unlikely that a useful model fit to the cpue will be possible. The getlag function first finds those rows for which both catch and cpue have values and then it runs the cross-correlation analysis. Thus, you cannot have gaps in your cpue data although there can be catches at the start or end of the time-series, or both, for which there are no cpue data.

Usage

```
getlag(fish, maxlag = 10, plotout = TRUE, indexI = 1)
```

Arguments

fish	the matrix or data.frame containing the fishery data (year, catch, and cpue)
maxlag	the lag.max parameter for the ccf function; defaults to 10
plotout	should a plot be made; default=TRUE. If FALSE then, assuming the result of the analysis is put into an object called 'ans' a call to plot(ans) will generate the required plot.
indexI	if there are more than one time-series of cpue/indices then this parameter selects which to use

Value

an object of class acf, which can be plotted

Examples

```
year <- 1985:2008
catch <- c(1018,742,868,715,585,532,566,611,548,499,479,428,657,481,
           645,961,940,912,955,935,940,952,1030,985)
cpue <- c(0.6008,0.6583,0.6791,0.6889,0.7134,0.7221,0.7602,0.7931,0.8582,
          0.8876,1.0126,1.1533,1.2326,1.2764,1.3307,1.3538,1.2648,1.2510,
          1.2069,1.1552,1.1238,1.1281,1.1113,1.0377)
dat <- as.data.frame(cbind(year,catch,cpue))
out <- getlag(dat,plotout=FALSE)
plot(out,lwd=3,col=2)
str(out)
```

---

getmax *getmax can be used to define the upper bound for a plot*

Description

getmax generates an upper bound for a plot where it is unknown whether the maximum is greater than zero of not. If > 0 then multiplying by the default mult of 1.05 works well but if the outcome if < 0 then the multiplier needs to be adjusted appropriately so the maximum is slightly higher than the maximum of the data

Usage

```
getmax(x, mult = 1.05)
```

Arguments

x	the vector of data to be tested for its maximum
mult	the multiplier for both ends, defaults to 1.05 (=0.95 if < 0)

Value

a suitable upper bound for a plot if required

Examples

```
vect <- rnorm(10,mean=0,sd=2)
sort(vect,decreasing=TRUE)
getmax(vect,mult=1.0)
vect <- rnorm(10,mean = -5,sd = 1.5)
sort(vect,decreasing=TRUE)
getmax(vect,mult=1.0)
```

---

getmin *getmin can be used to define the lower bound for a plot*

Description

getmin generates a lower bound for a plot where it is unknown whether the minumum is less than zero of not. If less than 0 then multiplying by the default mult of 1.05 works well but if the outcome if > 0 then the multiplier is adjusted appropriately so that the minimum is slightly lower than the minimum of the data

Usage

```
getmin(x, mult = 1.05)
```

Arguments

x	the vector of data to be tested for its minimum
mult	the multiplier for both ends, defaults to 1.05 (=0.95 if >0)

Value

a suitable lower bound for a plot if required

Examples

```
vect <- rnorm(10,mean=0,sd=2)
sort(vect)
getmin(vect)
getmin(vect,mult=1.0)  # finds actual minimum
```

getMSY *getMSY calculates the MSY for the Polacheck et al 1993 equation*

Description

getMSY calculates the MSY for the Polacheck et al 1993 generalized surplus production equation. This simplifies to rK/4 when p = 1.0. But this is a general equation that covers off for all positive values of p.

Usage

```
getMSY(pars, p = 1)
```

Arguments

pars	the model parameters r, K, Binit, sigma; p is kept separate
p	asymmetry parameter for the Polacheck et al 1993 equation, default=1.0 = Schaefer

Value

the MSY

References

Polacheck, T., Hilborn, R., and A.E. Punt (1993) Fitting surplus production models: Comparing methods and measuring uncertainty. *Canadian Journal of Fisheries and Aquatic Sciences*, 50: 2597-2607.

Examples

```
param <- c(r=1.1,K=1000.0,Binit=800.0,sigma=0.075)
getMSY(param,p=1.0)     #  275       Schaefer equivalent
getMSY(param,p=1e-08)   #  404.6674  Fox equivalent
```

getname *getname returns the name of a variable as character*

Description

getname runs 'deparse(substitute(x))' to get the name of the input variable. Saves remembering the syntax

Usage

```
getname(x)
```

Arguments

x	any variable whose name is wanted as a character string

Value

a character string with the name of input variable

Examples

```
a_variable <- c(1,2,3,4,5,6,7,8)
getname(a_variable)
```

---

getrmse *getrmse calculates the rmse of the input 'invar' series*

Description

getrmse calculates the root mean square error (rmse) of the input invar series (defaults to 'cpue') against an input 'year' time series. This is primarily designed to generate an alternative estimate of the intrinsic variability of a cpue time-series to that which may be obtained from a cpue standardization

Usage

```
getrmse(indat, invar = "cpue", inyr = "year")
```

Arguments

indat	the matrix, spmdat, or data.frame containing both a 'year' column and an invar column (default to 'cpue')
invar	the column name of the variable whose rmse is wanted; defaults to 'cpue'
inyr	the column name that points to the 'year' name

Value

a list of the rmse and the loess predicted values of the invar for each year in the time-series

Examples

```
year <- 1986:1994
cpue <- c(1.2006,1.3547,1.0585,1.0846,0.9738,1.0437,0.7759,1.0532,1.284)
dat <- as.matrix(cbind(year,cpue))
getrmse(dat,invar="cpue")  # should be 0.08265127
getrmse(dat,invar="cpue")$rmse
```

getseed *getseed generates a random number seed*

Description

getseed generates a seed for use within set.seed. It produces up to a 7 digit integer from the Sys.time. Initially, at the start of a session there is no seed; a new one is created from the current time and the process ID when one is first required. Here, in getseed, we do not use the process ID so the process is not identical but this at least allows the set.seed value to be stored should the need to repeat a set of simulations arise. The process generates up to a seven digit number it then randomly reorders those digits and that becomes the seed. That way, if you were to call getseed in quick succession the seeds generated should differ even when they are generated close together in time.

Usage

```
getseed()
```

Value

an integer up to 7 digits long

Examples

```
useseed <- getseed()
set.seed(useseed)
rnorm(5)
set.seed(12345)
rnorm(5)
set.seed(useseed)
rnorm(5)
```

getsingle *getsingle extracts one number from an input line of characters*

Description

getsingle splits up a text line and translates the first non-empty character string into a number. This function can be useful when parsing an input data file.

Usage

```
getsingle(inline, sep = ",")
```

Arguments

inline	the line of text, usually taken after using readLines to read in a text file
sep	the separator used to divide the numbers from descriptive text, defaults to a comma.

Value

a single number

Examples

```
x <- "12.3 , this is a number"
y <- "21.3 # 22.3 # here are two numbers"
getsingle(x)
getsingle(y,sep="#")
getsingle(y) # be sure to get the separator correct
```

---

gettime *gettime calculates time in seconds passed each day*

Description

gettime is a function designed to facilitate the measurement of time between intervals within R software that are expected to take a maximum of hours. It calculates the time as seconds elapsed from the start of each day. As long as the timing of events does not pass from one day to the next accurate results will be generated. To measure the time taken one would store an initial value that would be subtracted from a final value.

Usage

```
gettime()
```

Value

the time in seconds from the start of a day

Examples

```
begin <- gettime()
for (i in 1:1e6) sqrt(i)
finish <- gettime()
print(finish - begin)
```

getvector *getvector extracts a vector of numbers from a line of characters*

Description

getvector, when reading in a csv file using readLines, getvector extracts a line of numbers from a specified line within the readLines object.This function works out how many numbers there are. If you wish to add a comment at the end of a vector of numbers it must be separated from them by the separator. e.g. a comma

Usage

```
getvector(indat, locate, sep = ",")
```

Arguments

indat	the readLines object
locate	the line number from which to extract the numbers
sep	the separator between numbers, defaults to ","

Value

a vector of numbers

Examples

```
x <- "12.3, 15.1, 8.7,10.3,  # this is a vector of four numbers"
y <- "21.3 # 22.3 # 8.7 # 10.3 # here are another four numbers"
getvector(x)    # uses default separator
getvector(y,sep="#")
```

Gz *Gz calculates predicted Gompertz length-at-age growth curve*

Description

Gz calculates length-at-age for the Gompertz curve. This curve can have an inflection near the origin as well as at an age where growth slows due to maturity occurring.

Usage

```
Gz(p, ages)
```

Arguments

p	is a vector the first three values of which are a, b, c, for the Gz curve.
ages	is a vector of ages; could be a single number

Value

a vector of predicted lengths for a vector of ages in 'ages'

Examples

```
ages <- seq(0,20,1)   # sigma is ignored here
oldpar <- par(no.readonly=TRUE)
pars <- c(a=26.0,b=2.8,c=-0.65,sigma=1.0) # a, b, c, sigma;
plot1(ages,Gz(pars,ages),xlab="Age",ylab="Length",lwd=2)
par(oldpar)
```

---

halftable *halftable halves the height of a tall narrow data.frame for printing*

Description

halftable would be used when printing a table using kable from knitr where one of the columns was Year. The objective would be to split the table in half taking the bottom half and attaching it on the right hand side of the top half. The year column would act as the index.

Usage

```
halftable(inmat, yearcol = "year", subdiv = 2)
```

Arguments

inmat	the data.frame to be subdivided
yearcol	the column name of the year field default="year"
subdiv	the number of times the data.frame should be subdivided; the default is 2 but the numbers can only be 2 or 3.

Value

a data.frame half the height and double the width of the original

Examples

```
x <- as.data.frame(matrix(runif(80),nrow=20,ncol=4))
x[,1] <- 1986:2005
x[,4] <- paste0("text",1:20)
halftable(x,yearcol="V1",subdiv=2)
halftable(x[,c(1,2,4)],yearcol="V1")
x1 <- rbind(x,x[1,])
x1[21,"V1"] <- 2006
halftable(x1,yearcol="V1",subdiv=3)
```

`inthist` *inthist a replacement for the hist function for use with integers*

Description

inthist is a replacement for the 'hist' function for use with integers because the ordinary function fails to count them correctly. It treats integers (counts) as if they were real numbers. The function is designed for integers and if it is given real numbers it will issue a warning and then round all values before plotting. It can accept a vector of integers to be counts, or a matrix of values and associated counts.

Usage

```
inthist(
  x,
  col = 1,
  border = 1,
  width = 1,
  xlabel = "",
  ylabel = "",
  main = "",
  lwd = 1,
  xmin = NA,
  xmax = NA,
  ymax = NA,
  plotout = TRUE,
  prop = FALSE,
  inc = 1,
  xaxis = TRUE
)
```

Arguments

`x`	the vector of integers to be counted and plotted OR a matrix of values in column 1 and counts in column 2
`col`	the colour of the fill; defaults to black = 1, set this to 0 for an empty bar, but then give a value for border
`border`	the colour of the outline of each bar defaults to 1
`width`	denotes the width of each bar; defaults to 1, should be >0 and usually <= 1. A warning will be issued outside this range. If < 0 then it will be reset to 1.0
`xlabel`	the label for the x axis; defaults to ""
`ylabel`	the label for the y axis; defaults to ""
`main`	the title for the individual plot; defaults to ""
`lwd`	the line width of the border; defaults to 1
`xmin`	sets the lower bound for x-axis; used to match plots

xmax	sets the upper bound for x axis; used with multiple plots
ymax	enables external control of the maximum y value; mainly of use when plotting multiple plots together.
plotout	plot the histogram or not? Defaults to TRUE
prop	plot the proportions rather than the counts
inc	sets the xaxis increment; used to customize the axis; defaults to 1.
xaxis	set to FALSE to define the xaxis outside of inthist; defaults to TRUE

Value

a matrix of values and counts is returned invisibly

Examples

```
oldpar <- par(no.readonly=TRUE)
x <- trunc(runif(1000)*10) + 1
#plotprep(width=6,height=4)
inthist(x,col="grey",border=3,width=0.75,xlabel="Random Uniform",
        ylabel="Frequency")
abline(h=100)
par(oldpar)
```

---

invl *invl calculates growth increments for the inverse logistic*

Description

invl requires at least three parameters, MaxDL, L50, and delta, in a vector, as well as the initial length and the change in time between tag release and recapture. Given those it calculates the expected growth increment according to the inverse logistic curve. The parameter delta is equivalent to L95 - L50, the length difference between the length at half MaxDL, and the length at 5

Usage

```
invl(par, indat, initL = "l1", delT = "dt")
```

Arguments

par	a vector of at least MaxDL, L50, and delta from the inverse logistic growth curve
indat	the matrix or data.frame of data columns containing at least the initial lengths and the deltaT, time intervals between tag release and recapture.
initL	column name of the initial lengths within indat, default="l1"
delT	column name of the time interval, deltaT, within indat, default="dt"

Value

a vector of predicted growth increments

References

Haddon, M., Mundy, C., and D. Tarbath (2008) Using an inverse-logistic model to describe growth increments of blacklip abalone (Haliotis rubra) in Tasmania. *Fishery Bulletin* 106:58-71

Examples

```
data(blackisland)
oldpar <- par(no.readonly=TRUE)
plot(blackisland$l1,blackisland$dl,type="p",pch=16,
xlab="Initial Length mm",ylab="Growth Increment mm",panel.first=grid())
abline(h=0)
param <- c(25, 130, 35, 3) # MaxDL, L50, delta, sigma
predDL <- invl(param,blackisland,initL="l1",delT="dt")
lines(blackisland$l1,predDL,col=2,lwd=2)
par(oldpar)
```

---

iscol *incol is a utility to determine if a column is present in a matrix*

Description

incol is a utility to determine whether a named column is present in a given matrix or data.frame. Is case sensitive

Usage

```
iscol(incol = "year", inmat)
```

Arguments

incol	the name of the column; defaults to "year" as an example
inmat	the matrix or data.frame within which to search for incol

Value

TRUE or FALSE

Examples

```
x <- 1:10
test <- matrix(x,nrow=5,ncol=2,dimnames=list(1:5,c("year","Catch")))
print(test)
iscol("year",test)
iscol("Catch",test)
iscol("catch",test)
iscol("ages",test)
```

LatA *LatA Simulated length-at-age for 358 female fish*

Description

A data.frame containing the simulated age for an array of different lengths based upon the properties of an extensive collection of redfish (*Centroberyx affinis*) length-at-age data from eastern Australia sampled in the 1990's.

Format

A data.frame with 358 rows and 2 variables:

age simulated ages in years

length consequent simulated fork length of the fish, in cms

Subjects

- Estimating individual growth from length-at-age data
- von Bertalanffy growth curve
- Gompertz growth curve
- Michaelis-Menton curve used as a growth curve

Source

The data this simulation is based upon is from length-at-age data for one species collected over many years by the many excellent people running the Integrated Stock Monitoring Program in the Australian South East Fishery over the years of its existence. The simulation is based on a characterization of redfish properties and includes random error in the hypothetical measurements as well as the processes of growth (i.e. both measurement and process error). The other inputs were a selected set of growth parameters and the relative frequency of different ages vs lengths.

Examples

```
data(LatA)
pars <- c(27.0,0.15,-2.0) # von Bertalanffy
bestvB <- nlm(f=ssq,funk=vB,observed=LatA$length,p=pars,
              ages=LatA$age,typsize=magnitude(pars))
outfit(bestvB,backtran=FALSE,title="vB")
```

likeratio *likeratio conducts a likelihood ratio test*

Description

likeratio conducts a likelihood ratio test on two negative log-likelihoods. It produces the LR plus related statistics detailing if a significant difference has been found. The order in which the log-likelihoods are entered does not matter as that is checked for so a positive likelihood ratio is always generated.

Usage

```
likeratio(nLL1, nLL2, df = 1)
```

Arguments

nLL1	the first -ve log-likelihood
nLL2	the second -ve log-likelihood
df	the number of degrees of freedom difference between the two model, a minimum of 1, which is the default.

Value

a vector of the likelihood ratio, the significance of any difference, the minimum difference required for significance, and the degrees of freedom.

Examples

```
one <- 291.1691
two <- 277.0122
dof <- 1
round(likeratio(one,two,dof),8)  # LR = 28.3138
```

linter *linter finds a value in a series using its location in another*

Description

linter is a tool for linearly interpolating in a 2-D cartesian space to search out an unknown value between two known points on the x-axis, based on a known value between two known points on the y-axis. This might be answering the question of what would be the length at 50 percent maturity for a curve with no analytical solution. We could find two points in a series of proportion values on the y-axis that bracketed the 50 percent value using the function bracket. They would be associated with the two length values on the x-axis used to generate the predicted proportion values. If we assume the various points in the 2-D space to be approximated by linear relations then the location

between the two known x-axis length values corresponding to the L50 would have the same ratio as the 50 percent value has to the two points on the y-axis. See the example for details. The input arguments include five values, left, right, bottom, top, and target. So, left and right are sequential values on the x-axis, bottom and top are the corresponding sequential values on the y-axis, and target is the value we are looking for on the y-axis.

Usage

```
linter(pars)
```

Arguments

pars	a vector of 5 values, left, right, bottom, top and target

Value

a single value being the x-axis value between left and right corresponding to the target on the x-axis

See Also

bracket

Examples

```
L = seq(60,160,1)
p=c(a=0.075,b=0.075,c=1.0,alpha=100)
asym <- srug(p=p,sizeage=L)
L25 <- linter(bracket(0.25,asym,L))
L50 <- linter(bracket(0.5,asym,L))
L75 <- linter(bracket(0.75,asym,L))
ans <- c(L25,L50,L75,L50-L25,L75-L50)
{cat("   L25    L50      L75   L50-L25 L75-L50 \n")
cat(round(ans,4),"\n")}
```

---

logist *logist Logistic selectivity function*

Description

logist calculates a Logistic curve that can be used as a selectivity function, or maturity curve, of wherever a logistic is required. This version uses the logistic function 1/(1+exp(-log(19.0)*(lens-inL50)/delta)), which explicitly defines the L50 and uses delta = (inL95-inL50) as the second parameter.

Usage

```
logist(inL50, delta, depend, knifeedge = FALSE)
```

Arguments

inL50	is the length at 50 percent selection/maturity/whatever
delta	is the difference in selection/maturity/whatever between inL50 and inL95
depend	a vector of lengths/ages for which the logistic value will be calculated.
knifeedge	defaults to FALSE. If knifeedge is TRUE then the logistic values < the depend value of inL50 is set to zero, and all those >= inL50 are set to 1.0, approximating knife-edge selectivity

Value

A vector of length(depend) with the predicted logistic values

Examples

```
in50 <- 100.0
deltaS <- 8.0
lens <- seq(2,210,2)
select <- logist(inL50=in50,delta=deltaS,depend=lens)
selectk <- logist(in50,deltaS,lens,knifeedge=TRUE)
round(cbind(lens[35:70],select[35:70],selectk[35:70]),5)
```

---

magnitude *magnitude returns the magnitude of numbers in base 10*

Description

magnitude is useful when using an optimizer such as optim, which uses a parscale parameter. magnitude can determine the respective parscale value for each parameter value.

Usage

```
magnitude(x)
```

Arguments

x	the vector of numbers (parameters) whose magnitudes are needed

Value

a vector of magnitudes

Examples

```
x <- c(0,0.03,0.3,3,30,300,3000)
magnitude(x)
```

makelabel *makelabel generates a label from text and values*

Description

It is common to want a label with text and a series of values. But paste and paste0 cycles the text and the values. To avoid this makelabel first combines the values as text and then adds the input text to the front of the values

Usage

```
makelabel(txt, vect, sep = "_", sigdig = 3)
```

Arguments

txt	the input text for the label
vect	the series of values to be included in the label
sep	the separator for the components; defaults to _
sigdig	how many significant digits for the values; default = 3

Value

a character string made up of text and values

Examples

```
pars <- c(18.3319532,33.7935124,3.0378107,6.0194465,0.5815360,0.4270468)
makelabel("Cohort1",pars[c(1,3,5)],sep="__")
makelabel("Cohort1",pars[c(1,3,5)],sep="__",sigdig=4)
```

mature *mature alternative logistic function commonly used for maturity*

Description

mature a function 1/(1+(1/exp(a + b x sizeage))) which can be expressed as exp(a + b x sizeage)/(1 + exp(a + b x sizeage)). This describes a symmetric logistic curve that has the property Lm50 = -a/b and the interquartile distance is 2.log(3)/b.

Usage

```
mature(a, b, sizeage)
```

Arguments

a	is the intercept of the exponential function usually -ve
b	is the gradient of the exponential function
sizeage	is a vector of lengths/ages for which the logistic maturity value will be calculated

Value

A vector of predicted proportion mature for the sizeage

Examples

```
a <- -14.383
b <- 0.146017
lens <- seq(2,210,2)
round(mature(a,b,sizeage=lens),5) # length based
round(mature(-2.5,0.95,0:25),5)   # age based
```

---

minnow — *minnow contains weekly growth data for use with seasonal growth curves*

Description

minnow is a dataset of mean length against time in weeks for minnows (*Phoxinus phoxinus*), derived from Pitcher & Macdonald (1973) for use when fitting growth curves, especially seasonal growth curves. The data exhibit increases and decreases in length as each year progresses because these are mean lengths rather than individual measurements (which would, more typically, be used these days). The data have been read off a graph within the original paper as it is not reported explicitly, and are therefore only approximate, but will do for our purposes (but expect different parameters to those reported in the original paper). This is length at time not age. Though time is being used as a proxy for age there is no age 0.

Format

A data.frame of mean length-at-time data

week the week of sampling minnows for lengths

length the estimated mean length in the corresponding week in mm

Subjects

- seasonal growth curves
- von Bertalanffy
- Model residuals

Source

data measured from Figure 2, page 602 in Pitcher, T.J., and P.D M. MacDonald. (1973) Two models of seasonal growth. *Journal of Applied Ecology* 10:599–606.

Examples

```
data(minnow)
oldpar <- par(no.readonly=TRUE)
plot1(minnow$week,minnow$length,type="p",pch=16,cex=1.2,
      xlab="Week",ylab="Length mm")
par(oldpar)
```

---

mm *mm calculates the predicted Michaelis-Menton length-at-age*

Description

mm calculates length-at-age for the generalized Michaelis-Menton curve. The equation being (a x ages)/(b + ages^c).

Usage

```
mm(p, ages)
```

Arguments

p	is a vector the first three cells of which are a, b, c for the mm curve.
ages	is a vector of ages; could be a single number

Value

a vector of predicted lengths for a vector of ages in 'ages'

Examples

```
ages <- seq(0,20,1)    # sigma is ignored here
pars <- c(a=23.0,b=1.0,c=1.0,sigma=1.0) # a, b, c, sigma
plot1(ages,mm(pars,ages),xlab="Age",ylab="Length",lwd=2)
```

mnnegLL *mnnegLL generic multinomial negative log-likelihoods*

Description

mnnegLL a generic multinomial negative log-likelihood that requires observed frequencies and predicted frequencies, although the predicted frequencies could also be the final proportions, as long as they sum to one. It checks that the number of predicted values matches the number of observed values

Usage

```
mnnegLL(obs, predf)
```

Arguments

obs	the original observed frequencies
predf	the predicted frequencies or proportions

Value

a single scalar value

Examples

```
  obs <- c(0,0,6,12,35,40,29,23,13,7,10,14,11,16,11,11,9,8,5,2,0)
  predf <- c(0.1,0.9,4.5,14.4,29.7,39.9,35.2,21.3,10.9,8.0,9.5,12.1,
             14.1,14.7,13.7,11.5,8.6,5.8,3.5,1.9,0.9)
mnnegLL(obs,predf)   # should be  705.5333
```

MQMF *MQMF R functions for the New Book*

Description

The MQMF package Provides R functions for use with the new book: (*Using R for Modelling and Quantitative Methods in Fisheries*), being published by CRC Press / Chapman & Hall in their (*Using R*) series. Currently no vignettes are included but all examples from the chapters are included as help pages for functions. Try ?chapter2, ?chapter3, ..., ?chapter7, which will provide a listing of all the code chunks included in the book (and some that are not). The list of functions below is not complete so scroll to the bottom of any help file and click the link to the index of functions instead. A development version is available on GitHub at github.com/haddonm/MQMF.

Some of the utility functions

countones used in apply to count the ones in a vector

countzeros used in apply to count the zeros in a vector

countgtzero halves the height of a tall narrow data.frame

countNAs used in apply to count the NAs in a vector

countgtone used in apply to count the numbers > 1 in a vector

freqMean calculates the mean and st dev count data

getmin find the minimum of a series to help with ylim or xlim

getmax find the maximum of a series to help with ylim or xlim

getname extracts the name of a variable as character

halftable subdivides a table to make it shorter and wider

magnitude defines the relative size of parameters for use when using nlm or optim without log-transforming the parameters

makelabel simplifies combining a name with a vector of numbers for use as a label or a legend entry

outfit prints a pretty version of the results from optim, nlm, or nlminb

printV prints a vector of numbers vertically rather than horizontally

quants used in 'apply' to estimate quantiles across a vector

which.closest finds the value in a vector closest to a given value.

Some of the plotting and printing functions

addnorm fits a normal distribution to the output from hist

addlnorm fits a log-normal distribution to output from hist

inthist plots a histogram of integer values more precisely than hist.

parset defines the par statement for a single plot

parsyn types the standard syntax for the par command to the console

plot1 simplifies the plotting of two variables in a single plot

plotprep Sets up a window and the par values for a plot. it checks to see if a graphics device is open and opens a new one if not. This is simply a utility function to save typing the standard syntax. Some of the defaults can be changed. Typing the name without () will provide a template for modification. If 'windows' is called repeatedly this will generate a new active graphics device each time leaving the older ones inactive but present. For quick exploratory plots this behaviour is not wanted, hence the check if an active device exists already or not.

printV returns a vector cbinded to 1:length(invect), which effectively prints the numbers vertically

properties prints a listing of the properties of the column variables within a data.frame

References

Haddon, M. (2020) Using R for Modelling and Quantitative Methods in Fisheries, CRC Press / Chapman & Hall/ Boca Raton 337p. ISBN: 9780367469894

negLL *negLL calculate log-normal log-likelihoods*

Description

negLL calculates log-normal negative log-likelihoods. It expects the input parameters to be log-transformed, so the funk used to calculate the log or the predicted values also needs to expect log-transformed parameters. In addition, it checks that there are no missing data (NA) within the input observed log-transformed data. If there are it uses only those records for which there are values.

Usage

```
negLL(pars, funk, logobs, ...)
```

Arguments

pars	the log-transformed parameters to be used in the funk for calculating the log of the predicted values against which the log observed values will be compared
funk	the function used to calculate the log-predicted values of whatever variable is being used (eg. cpue, catches, etc.)
logobs	the observed values, log-transformed, ready for comparison with the log-predicted values from funk and pars.
...	required to allow funk to access its other arguments without having to explicitly declare them in negLL. In the example below, indat is passed via the ...

Value

the negative log-likelihood using log-normal errors.

Examples

```
data(abdat)  # expect an answer of -31.65035
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
negLL(pars=param,funk=simpspm,logobs=log(abdat[,"cpue"]),indat=abdat)
```

negLL1 *negLL1 calculate log-normal log-likelihoods with a penalty*

Description

negLL1 calculates log-normal negative log-likelihoods. It expects the input parameters to be log-transformed, so the funk used to calculate the log or the predicted values also needs to expect log-transformed parameters. In addition to estimating the negative log-likelihoods for log-normally distributed data it also places a penalty on the first parameter if that parameter approaches very close to zero; see the help page for penalty0. With SPM the first parameter is the population growth rate r, which obviously should never be negative. The use of penalty0 prevents this happening.

Usage

```
negLL1(pars, funk, logobs, ...)
```

Arguments

pars	the log-transformed parameters to be used in the funk for calculating the log of the predicted values against which the log observed values will be compared
funk	the function used to calculate the log-predicted values of whatever variable is being used (eg. cpue, catches, etc.)
logobs	the observed values log-transformed ready for comparison with the log-predicted values from funk and pars.
...	required to allow funk to access its other parameters without having to explicitly declare them in negLL

Value

the negative log-likelihood using log-normal errors.

Examples

```
data(abdat)  #expect an answer of -31.65035
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
negLL1(pars=param,funk=simpspm,logobs=log(abdat[,"cpue"]),indat=abdat)
```

---

negLLM *negLLM -ve log-normal likelihoods for multiple index time-series*

Description

negLLM we have negLL and negLL1 for use when using -ve log-likelihoods to fit surplus production models that only have a single index of relative abundance, but there are many fisheries that have more than one index of relative abundance. negLLM is for those cases that have multiple (M) time-series of such indices. It is used in conjunction with simpspmM and spmCE.

Usage

```
negLLM(pars, funk, logobs, indat, index = "cpue", harvpen = TRUE, ...)
```

Arguments

pars	the log-transformed parameter starting points. For a surplus production model these are r, K, Binit (if initial depletion is likely, otherwise omit this and it will be set =K inside the function), then as many sigma values as there are time-series of abundance indices; these are the associated standard deviations of the log-normal residuals.
funk	the function that generates the predicted cpue values. for multiple time-series in a SPM use simpspmM

logobs	the log-transformed observed cpue columns in indat, the data needed by funk, tranferred inside the ...
indat	the fisheries data used in the analysis
index	the prefix of the columns of each of the indices, defaults to cpue
harvpen	default = TRUE, which sets a penalty1 on each of the implied harvest rates to ensure we do not get harvest rates > 1.0
...	the continuation ellipsis to allow the transfer of other arguments required by funk. The argument 'schaefer' in the example below illustrates such usage.

Value

a single scalar as the -ve log-likelihood of the input data

Examples

```
data(twoindex)
fish <- as.matrix(twoindex)
pars <- log(c(0.04,155000,0.4,0.3))
bestSP <- nlm(f=negLLM,p=pars,funk=simpspmM,indat=fish,
            schaefer=TRUE,logobs=log(fish[,c("cpue1","cpue2")]),
            steptol=1e-06,harvpen=TRUE)
outfit(bestSP,digits=5,title="negLLM example") #optimum solution
answer <- plotspmmod(bestSP$estimate,indat=fish,
                     plotprod=TRUE,maxy=3.4)
```

---

negLLP *negLLP -ve log-likelihood for normally distributed variables*

Description

negLLP calculates the negative log-likelihood for normally distributed variables allowing for some parameters to remain fixed. It assumes the presence of a function 'funk' that will calculate predicted values of a dependent variable from a vector of independent values (logobs). By having a separate vector of parameters in 'initpar' and a vector of the indices of those parameters that will be fitted (notfixed) it is possible to fit only a subset of parameters. This is useful, for example, if generating a likelihood profile, or setting up a likelihood ratio test. With more complex models it is often a useful strategy to estimate the full number of parameters in a sequence of phases, increasing the number being estimated each time while keeping the rest fixed. 'negLLP' makes such phasing of the fitting of a model to data possible. This function can be applied directly to log-transformed data for log-normally distributed data, in which case funk would need to generate log-transformed values. But can also be applied to normally distributed data, in which case one would not log-transform the data being input to the logobs argument and funk would generated the linear-space predicted values.

The selection of which parameters to vary is simply implemented through repeatedly copying the original input values from initpar and then changing those notfixed from the varying par values

Usage

```
negLLP(
  pars,
  funk,
  indat,
  logobs,
  initpar = pars,
  notfixed = c(1:length(pars)),
  ...
)
```

Arguments

pars	a vector containing the log-transformed parameters being used in funk, plus an extra sigma which is the standard deviation of the log-normal random likelihoods in dnorm
funk	the function name that calculates the predicted values from the independent values
indat	the data.frame that contains the data used by the input funk
logobs	the log-transformed observed values for comparison with the values that the model will predict for each of the independent values
initpar	this defaults to the same as pars - using all parameters
notfixed	a vector identifying the indices of the parameters to be fitted, which also defines those that will be fixed; defaults to all parameters set to vary. If some need to be kept constant so as to generate a likelihood profile then omit their index from 'notfixed'.
...	required to allow funk to access its other parameters without having to explicitly declare them in negLL

Value

the sum of the negative log-likelihoods using a normal PDF

Examples

```
data(abdat)
param <- log(c(r= 0.42,K=9400,Binit=3400,sigma=0.05))
optmod <- nlm(f=negLLP,p=param,funk=simpspm,initpar=param,
             notfixed=c(1,2,3,4),indat=abdat,logobs=log(abdat$cpue))
outfit(optmod,backtran= TRUE) #backtran=TRUE is default anyway

rval <- seq(0.325,0.45,0.0125)  # set up the test sequence
columns <- c("r","K","Binit","sigma","-veLL")
result <- matrix(0,nrow=11,ncol=5,dimnames=list(rval,columns))
profest <- c(r= 0.32,K=11500,Binit=4250,sigma=0.05) # end of sequence
for (i in 1:11) {
  param <- log(c(rval[i],profest[2:4])) #recycle the profest values to
  parinit <- param    # improve the stability of nlm as the r value
```

```
  bestmodP <- nlm(f=negLLP,p=param,funk=simpspm,initpar=parinit, #changes
                  indat=abdat,logobs=log(abdat$cpue),notfixed=c(2:4))
  bestest <- exp(bestmodP$estimate)
  result[i,] <- c(bestest,bestmodP$minimum)  # store each result
}
result   #Now you can plot -veLL againt r values for the profile
# parset()
# plot(result[,"r"],result[,"-veLL"],type="l",lwd=2,panel.first=grid())
# points(result[,"r"],result[,"-veLL"],pch=16,cex=1.1)
```

---

negNLL *negNLL -ve log-likelihood for normally distributed variables*

Description

negNLL - Calculates the negative log-likelihood for normally distributed variables. It requires a function, 'funk' as an argument, that will calculate predicted values of a variable from a vector of input values. In the example below the predicted values are lengths-at-age and the input data are ages. Only the arguments used within negNLL are listed explicitly, which leaves the data required to drive the funk to generate the predicted values for comparison with the observed to be passed using the ..., so take care with spelling of the variable name required by whatever funk is being used.

Usage

```
negNLL(pars, funk, observed, ...)
```

Arguments

pars	a vector containing the parameters being used in funk, plus the sigma, which is the standard deviation of the normal random likelihoods in dnorm, an extra estimated parameter. Because negNLL refers explicitly to sigma, which must be the last parameter in the vector, it must be listed in the arguments.
funk	the function name that calculates the predicted values from the independent values passed using the ellipsis.
observed	the observed values of the variable that the model will predict to compare with each of the input observed values.
...	required to allow funk to access its other input data without having to explicitly define them in negNLL

Value

the sum of the negative log-likelihoods using a normal PDF

Examples

```
data(minnow)
pars <- c(89, 0.01,-15,3.75)  # ssq    = 785.6183
age <- minnow$week            # negNLL = 151.1713
ssq(funk=vB,observed=minnow$length,p=pars,ages=age)
negNLL(pars,funk=vB,observed=minnow$length,ages=age)
oldpar <- par(no.readonly=TRUE)
plot1(age,vB(pars,age),xlab="Age",ylab="Length",lwd=2)
points(age,minnow$length,pch=16,cex=1.2)
par(oldpar)
```

---

negnormL *negnormL an alternative -log-likelihood for normal errors*

Description

negnormL is an alternative to negNLL to produce -ve log-likelihoods for normal random errors, allowing for the sigma parameter to vary as a function of the predicted value. In negnormL only one needs both a funk and a funksig, the former to calculate the predicted values using funk, and funksig to calculate the changing sigma values relative to the predicted values. The example code illustrates an example funksig that does nothing to the sigma value. See the chapter on Static Models to see further examples.

Usage

```
negnormL(pars, funk, funksig, indat, obs = "dl", ...)
```

Arguments

pars	the vector of parameters, with sigma, the standard deviation of the normal random deviates at the end.
funk	the funk needed to generate the predicted values
funksig	the function used to calculate the sigma value based on the last parameter (=constant sigma) and the predicted values
indat	the data.frame or matrix containing the obs and the independent variable used by funk
obs	identifies the column name or column number that contains the observed data for comparison with the predicted from funk, the default="dl"
...	the standard R for including extra parameters needed by funk but without having to be explicitly defined.

Value

the negative log-likelihood, for use in an optimizer, eg nlm

Examples

```
data(blackisland)
param <- c(Linf=173.65,K=0.2666,sigma=3.6)
sigfunk <- function(pars,predobs) return(tail(pars,1))
negnormL(par=param,funk=fabens,funksig=sigfunk,indat=blackisland,
         obs="dl",initL="l1",delT="dt")   # should be 291.1757
param2 <- c(21.03,130.94,40.65,3.162)
negnormL(par=param2,funk=invl,funksig=sigfunk,indat=blackisland,
          obs="dl",initL="l1",delT="dt")  # should be 277.0186
```

`npf` *npf fishery catch data from Northern Prawn Fishery 1970-1992*

Description

npf is fishery catch data from Australia's Northern Prawn Fishery from 1970 to 1992 summarized from Robins and Somers, 1994. It contains the catches, in tonnes, of banana prawns (*Penaeus merguiensis* and *P. indicus*), tiger prawns (brown - *P. esculentus*) and (grooved - *P. semisulcatus*), endeavour prawns, (*Metapenaeus endeavouri* and *M. ensis*), king prawns (*P. latisulcatus* and *P. longistylus*), the number of vessels fishing, and the annual effort as boat-days.

Format

A data.frame of fisheries data

year the fishing year from 1970 - 1992.

banana banana prawn catches, tonnes.

tiger tiger prawn catches, tonnes.

endevaour endeavour prawn catches, tonnes.

king king prawn catches, tonnes.

boats the number of vessels fishing in that year.

boatday the total annual effort as boatdays.

Subjects

- correlation analysis and regression
- Bootstrap percentile confidence intervals
- Model residuals

Source

Robins, C. and I. Somers (1994) Appendix A. Fishery Statistics pp 141 - 164 in Pownall, P.C. (ed.) Australia's Northern Prawn Fishery: The first 25 years. NPF25. Cleveland, Australia. 179p.

Examples

```
data(npf)
npf
oldpar <- par(no.readonly=TRUE)
plot1(npf$year,npf$tiger,xlab="Year",ylab="Tonnes",lwd=2)
lines(npf$year,npf$endeavour,col=2,lwd=2)
legend("topleft",c("Tiger","Endeavour"),col=c(1,2),lwd=3,
       bty="n",cex=1.5)
par(oldpar)
```

---

outfit *outfit tidy print of output from optim, nlminb, or nlm*

Description

outfit takes in the output list from either optim, nlminb, or nlm and prints it more tidily to the console, In the case of nlm it also prints the conclusion regarding the solution. It might be more effective to implement an S3 method.

Usage

```
outfit(inopt, backtran = TRUE, digits = 5, title = "", parnames = "")
```

Arguments

inopt	the list object output by nlm, nlminb, or optim
backtran	a logical default = TRUE If TRUE it assumes that the parameters have been log-transformed for stability and need back-transforming
digits	the number of digits to round the backtransformed parameters. defaults to 5.
title	character string used to label the output if desired, default = empty character string
parnames	default="" which means the estimated parameters will merely be numbered. If a vector of names is given then this will be used instead, at least, for nlm and optim.

Value

nothing but it does print the list to the console tidily

Examples

```
x <- 1:10  # generate power function data from c(2,2) + random
y <- c(2.07,8.2,19.28,40.4,37.8,64.68,100.2,129.11,151.77,218.94)
alldat <- cbind(x=x,y=y)
pow <- function(pars,x) return(pars[1] * x ^ pars[2])
ssq <- function(pars,indat) {
   return(sum((indat[,"y"] - pow(pars,indat[,"x"]))^2))
```

```
}  # fit a power curve using normal random errors
pars <- c(2,2)
best <- nlm(f=ssq,p=pars,typsize=magnitude(pars),indat=alldat)
outfit(best,backtran=FALSE) #a=1.3134, b=2.2029 ssq=571.5804
```

---

panel.cor *panel.cor is a version of a function given in the help for pairs*

Description

panel.cor is a panel function modified from that described in the help file for the pairs function from the graphics package. This has been customized both to show that one can make such customizations, and to enable this one to be used to calculate the correlations between the variables included in a pairs plot.

Usage

```
panel.cor(x, y, digits = 3, ...)
```

Arguments

x	the first variable - provided by pairs
y	the second variable, provided by pairs, see examples
digits	how many digits to use on the pairs plot for the correlations
...	any other graphics parameters to be passed to pairs.

Value

this prints the correlations in a square of the pairs plot

Examples

```
dat <- matrix(rnorm(900,mean=5,sd=0.5),nrow=300,ncol=3)
pairs(dat[,1:3],lower.panel=panel.smooth,  # all should be
      upper.panel=panel.cor,gap=0.25,lwd=2) #low correlations
```

parasympt *parasympt generates N vectors from a multi-variate normal*

Description

parasympt generates N vectors from a multi-variate normal distribution for a surplus production model. This can be used when estimating the uncertainty around an spm fit, or when conducting projections from a model fit while attempting to account for uncertainty. Use of this function requires the mvnnorm package. It could be generalized to suit any model. It is designed for use only with models fitted using maximum likelihood.

Usage

```
parasympt(bestmod, N)
```

Arguments

bestmod	the output from nlm containing the optimal parameters in log-space, and the hessian
N	the number of parameter vectors to be sampled from the multi- variate normal defined by the optimal parameters and the inverse of the hessian (the variance-covariance matrix).

Value

an N x numpar matrix of parameter vectors

Examples

```
data(abdat)
schf <- FALSE
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
bestmod <- nlm(f=negLL1,p=param,funk=simpspm,logobs=log(abdat$cpue),
               indat=abdat,typsize=magnitude(param),iterlim=1000,
               schaefer=schf,hessian = TRUE)
out <- spm(bestmod$estimate,indat=abdat,schaefer=schf)
matpar <- parasympt(bestmod,1000)
head(matpar,15)
pairs(matpar)
```

parset *parset alters the current base graphics par settings*

Description

parset alters the current base graphics par settings to suit a single standard plot. It is merely here to simplify and speed the coding for exploratory base graphics. The font and its font size defaults to 0.75 and font 7 (Times bold). The default values can be seen by typing parset with no brackets in the console or use args(parset). If a different set of par values are wanted then the parset arguments can be modified, or the function parsyn() can be used to act as a prompt to the console for the correct syntax. The console output can be copied to your script and modified.

Usage

```
parset(
  plots = c(1, 1),
  cex = 0.75,
  font = 7,
  outmargin = c(0, 0, 0, 0),
  margin = c(0.45, 0.45, 0.05, 0.05)
)
```

Arguments

plots	vector of number of rows and columns, defaults to c(1,1)
cex	the size of the font used, defaults to 0.75
font	the font used, defaults to 7 which is Times Bold, 6 is Times, 1 is Sans and 2 is Sans Bold.
outmargin	defines whether to leave extra space on the bottom, left, top, or right hand sides of the plot. Used when plots != c(1,1). Allows room for mtext statements.
margin	defines the space allowed for labels on axes. Again, likely needs to change is having more than one plot

Value

nothing but it changes the base graphics par settings. The original par values are returned invisibly if user wishes to reset after plotting their graphic.

Examples

```
x <- rnorm(100,mean=5,sd=0.5)
y <- rnorm(100,mean=5,sd=0.5)
oldpar <- parset(plots=c(1,2))
plot1(x,y,defpar=FALSE)
plot1(y,x,defpar=FALSE)
par(oldpar)
```

parsyn *parsyn types the standard par command syntax to the console*

Description

parsyn prints the standard par command syntax to the console so it can be copied and pasted into your own code and modified as suits your needs. It is simply a memory aid.

Usage

```
parsyn()
```

Value

nothing but it writes two lines of R code to the console

Examples

```
## Not run:
  parsyn()

## End(Not run)
```

penalty0 *penalty0 enables the adding of a large penalty as one approaches 0.0*

Description

penalty0 allows for the option of adding a large penalty as a parameter approaches 0.0 . See negLL1 for example code that contains such a parameter. For example, when fitting an spm sometimes the optimal mathematical model fit can occur by depressing the r value to 0 or even go negative. Input values < 0.006 begin to generate large values as one goes smaller. The examples below illustrate this.

Usage

```
penalty0(x)
```

Arguments

x	the parameter value that potentially incurs a penalty

Value

a single value as a penalty to be added to a Log-Likelihood or SSQ

Examples

```
penalty0(0.5)
penalty0(0.1)
penalty0(0.01)
penalty0(0.005)
```

`penalty1` *penalty1 adds an increasingly large penalty as a value approaches 1.0*

Description

penalty1 allows for the option of adding a large penalty as a parameter approaches 1.0 and moves to become larger than 1. For example, when fitting a surplus production model sometimes the optimal mathematical model fit can occur by implying catches greater than available biomass, implying harvest rates > 1.0. By adding a large penalty to such values and adding those to the likelihood such strange outcomes can be avoided. This will accept a single value or a vector.

Usage

```
penalty1(x)
```

Arguments

x	the parameter value that potentially incurs a penalty

Value

a single value as a penalty to be added to a Log-Likelihood or SSQ

Examples

```
x <- c(0.5,0.8,0.88,0.9,0.98,0.99,1.01)
round(cbind(x,penalty1(x)),4)
```

`plot.dynpop` *plot.dynpop an S3 method for plotting dynpop objects*

Description

plot.dynpop an S3 method for plotting dynpop objects which are generated by the function discretelogistic. It generates a plot of the numbers through time, that is year x Nt, and a second plot that is a phase plot of Nt+1 vs Nt, to better illustrate the dynamics and simplify the search for equilibria. The input is designed to be the invisibly produced output from the MQMF function discretelogistic, but as long as there is at least a matrix of year, Nt, and Nt+1 with a class of dynpop, then a call to plot should generate the two graphs.

Usage

```
## S3 method for class 'dynpop'
plot(x, y = NULL, main = "", cex = 0.9, font = 7, ...)
```

Arguments

x	a dynpop matrix containing at least year, nt, and nt1 as columns, as returned invisibly from discretelogistic.
y	a second y value added to the first plot if present
main	defines text to print across top of plot with the intention of including the parameter values, default=""
cex	the size of the fonts used, default=0.9
font	the font used default=7 (bold serif) 1 = non-serif, 2 = bold non-serif, 6 = serif
...	ready for extra graphical parameters

Value

it returns the dynpop matrix invisibly

Examples

```
## Not run:
  r <- 2.5; K <- 1000.0; N0 <- 50; Ct <- 0.0; yrs <- 50; p <- 1
  pop <- discretelogistic(r=r,K=K,N0=N0,Ct=Ct,Yrs=yrs,p=p)
  plot(pop,main=paste0("r=",r,"  K=",K,"  Ct=",Ct, "  N0=",N0,
                       "  p=",p),cex=0.85,font=7)

## End(Not run)
```

---

plot1 *plot1 a quick way to plot an xy line or point plot*

Description

plot1 provides a quick way to plot out a single xy line or point plot. It can be used with plotprep to generate a plot outside of Rstudio or by itself to generate one within Rstudio. It uses a standard par setup and permits custom labels, font, and font size (cex). It checks the spread of y and if a ymax is not given in the parameters finds the ymax and checks to see if y goes negative in which case it uses getmin, so the y-axis is set to 0 - ymax or ymin - ymax

Usage

```
plot1(
  x,
  y,
  xlab = "",
  ylab = "",
  type = "l",
  usefont = 7,
  cex = 0.75,
  maxy = 0,
  defpar = TRUE,
  ...
)
```

Arguments

x	The single vector of x data
y	the single vector of y data. If more are required they can be added separately after calling plot1.
xlab	the label for the x-axis, defaults to empty
ylab	the label for the y-axis, defaults to empty
type	the type of plot "l" is for line, the default, "p" is points. If you want both, then plot a line and add points afterwards.
usefont	which font to use, defaults to 7 which is Times bold
cex	the size of the fonts used. defaults to 0.75
maxy	defaults to 0, which does nothing. If a value is given then this value is used rather than estimating from the input y
defpar	if TRUE then plot1 will declare a par statement. If false it will expect one outside the function. In this way plot1 can be used when plotting multiple graphs, perhaps as mfrow=c(2,2)
...	required to allow the plot to access other parameters without having to explicitly declare them in plot1, these include col, default = black, pch, if the type = "p", lwd, etc.

Value

plots a graph and sets the default plotting par values. This changes the current plotting options! The original par values are returned invisibly if the user wishes to reset.

Examples

```
x <- rnorm(20,mean=5,sd=1)
oldpar <- plot1(x,x,xlab="x-values",ylab="yvalues")
points(x,x,pch=16,cex=1.5)
y <- rnorm(20,mean=5,sd=1)
plot1(x,y,type="p",cex=1.2,panel.first=grid())
par(oldpar)
```

plotfishM *plotfishM plots the catch and optionally the cpue*

Description

plotfishM uses a matrix of fishery data. It requires the matrix or data.frame to contain the columns 'year', 'catch', and optionally 'cpue'.

Usage

```
plotfishM(
  fish,
  spsname = "",
  ce = TRUE,
  title = TRUE,
  fnt = 7,
  both = TRUE,
  maxy = c(0, 0),
  year = "year",
  catch = "catch",
  cpue = "cpue"
)
```

Arguments

fish	the matrix or data.frame containing year, catch, and cpue.
spsname	the name of the species concerned
ce	a logical parameter determining whether to plot the cpue or not. the default = TRUE
title	determines whether or not the spsname is printed at the top of the plot. Default = TRUE but for a more formal publication it might need to be set to FALSE, which also reallocates the room given to the title to the plot.
fnt	the font used in the plot and axes.
both	plot both the catches and the CPUE
maxy	a vector of two zeros. These define the maximum y-axis value for each plot. If it remains zero then getmax will be used to find the maximum.
year	the name of the column containing the years, default="year"
catch	the name of the column containing catches, default="catch"
cpue	the name of the column containing cpue, default="cpue"

Value

plots a graph but returns nothing to the console

Examples

```
## Not run:
  data(dataspm)
  plotfishM(fish=dataspm,spsname="Pink Ling",ce=TRUE)

## End(Not run)
```

---

plotlag *plotlag plots the effect of a lag between two variables*

Description

the use of the base function ccf can suggest a lagged relationship between a driver variable and a react(ing) variable. For example, cpue may respond to catches in a negative manner after a lag of a few years. One looks for a negative lag, which would imply that the driver variable influences the react(ing) variable after the given lag has passed. The lag is always assumed to be based on yearly intervals, though this can be changed.

Usage

```
plotlag(
  x,
  driver = "catch",
  react = "cpue",
  lag = 0,
  interval = "year",
  filename = "",
  resol = 200,
  fnt = 7
)
```

Arguments

x	the matrix containing columns of the named variables. It must contain columns with the same names as the driver and react(ing) variables
driver	the variable doing the influencing
react	the variable being influenced
lag	the time lag before the influence is felt
interval	the name of the time-interval variable, default='year'
filename	default is empty. If a filename is put here a .png file with that name will be put into the working directory.
resol	the resolution of the png file, defaults to 200 dpi
fnt	the font used in the plot and axes. Default=7, bold Times. Using 6 gives Times, 1 will give SansSerif, 2 = bold Sans

Value

a list containing some summary results, the anova of the linear model fitted in aov, and a summary of the linear model in summ

Examples

```
year <- 1985:2008
catch <- c(1018,742,868,715,585,532,566,611,548,499,479,428,657,481,645,
           961,940,912,955,935,940,952,1030,985)
cpue <- c(0.6008,0.6583,0.6791,0.6889,0.7134,0.7221,0.7602,0.7931,0.8582,
          0.8876,1.0126,1.1533,1.2326,1.2764,1.3307,1.3538,1.2648,1.2510,
          1.2069,1.1552,1.1238,1.1281,1.1113,1.0377)
dat <- cbind(year,catch,cpue)
out <- plotlag(dat,driver="catch",react="cpue",lag=7)
round(out$results,5)
out$summ
```

---

plotprep *plotprep sets up a window and the par values for plotting*

Description

plotprep sets up a window and changes the par values for plots. This is simply a utility function to save typing the standard syntax. Some of the defaults can be changed. Typing the name without () will provide a template for modification. If different par values are wanted then just include a par statement after plotprep(). Calling plotprep saves the current par settings and returns them invisibly. So to recover the original par settings use oldpar <- plotprep(), and, once you have completed your plot, you can include a par(oldpar) to recover your original settings. The default ratio of width to height approximates the golden ratio = (width x height)/width

Usage

```
plotprep(
  width = 6,
  height = 3.7,
  usefont = 7,
  cex = 0.75,
  newdev = FALSE,
  filename = "",
  resol = 300,
  verbose = FALSE
)
```

Arguments

width	defaults to 6 inches = 15.24cm - width of plot
height	defaults to 3.7 inches = 9.38cm - height of plot

usefont	default=7 (bold Times) 1=sans serif, 2=sans serif bold
cex	default=0.75, size of font used for text within the plots
newdev	reuse a previously defined graphics device or make a new one; default=FALSE
filename	default="" ie do not save to a filename. If filename is defined it makes that file as a png file with resolution resol
resol	resolution of the png file, if defined, default=300
verbose	set this to TRUE to turn on the reminder to include a graphics.off() command after tsaving a png file. Default=FALSE

Value

sets up a graphics device, if needed, and resets the default plotting par values. This changes the current plotting options! The original par values are returned invisibly.

Examples

```
x <- rnorm(1000,mean=0,sd=1.0)
plotprep(newdev=TRUE)
hist(x,breaks=30,main="",col=2)
oldpar <- plotprep(width=6,height=5,newdev=FALSE)
par(mfrow = c(2,1)) # can run parset() or change the par settings
hist(x,breaks=20,main="",col=2)
hist(x,breaks=30,main="",col=3)
par(oldpar)
```

---

plotprofile *plotprofile simplifies plotting single likelihood profiles*

Description

plotprofile simplifies plotting out the likelihood profiles of single parameters or variables. It is necessary to pass the function the output from the profile calculations, identifying the variable name against which to plot the likelihood while also identifying the name of the -ve log-likelihood column. Facilities are provided for defining the x and y axis labels. We need to use the function which.closest because we use a sequence of parameter values so an exact match would be highly unlikely.

Usage

```
plotprofile(
  prof,
  var,
  digit = c(3, 3, 3),
  xlabel = var,
  ylabel = "-ve Log-Likelihood",
  like = "-veLL",
  defpar = TRUE,
  ...
)
```

Arguments

prof	the results from the likelihood profile calculations. This matrix should include, as a minimum, the fixed variable of interest and the matching -ve log-likelihood in named columns.
var	the name of the variable of interest to identify the column in prof in which to find the vector of fixed values given.
digit	this is a vector of three that determine by how much the round function limits the values printed of the 95 mean at the top of the plot.
xlabel	the x-axis label, defaults to the name of the var
ylabel	the y-axis label, defaults to -ve Log-Likelihood
like	identifies the name of the column containing the -ve log-likelihood
defpar	logical, should the par values be assumed or defined, defaults to TRUE, so only one plot will be produced and any old par values will be restored afterwards. If part of a multiple plot define the formatting before calling and set this to FALSE. This will lead to the oldpar being returned invisibly so that eventually the par settings can be reset.
...	used for any other graphic parameters used.

Value

nothing but this does generate a plot.

Examples

```
data(abdat)    #usually use  ~100 steps in rval, perhaps use
rval <- seq(0.325,0.45,0.02)   # seq(0.325,0.45,0.001)
ntrial <- length(rval)
columns <- c("r","K","Binit","sigma","-veLL")
result <- matrix(0,nrow=ntrial,ncol=length(columns),
                 dimnames=list(rval,columns))
bestest <- c(r= 0.32,K=11000,Binit=4000,sigma=0.05)
for (i in 1:ntrial) {  #i <- 1
  param <- log(c(rval[i],bestest[2:4]))
  parinit <- param
  bestmodP <- nlm(f=negLLP,p=param,funk=simpspm,initpar=parinit,
                  indat=abdat,logobs=log(abdat$cpue),notfixed=c(2:4),
                  typsize=magnitude(param),iterlim=1000)
  bestest <- exp(bestmodP$estimate)
  result[i,] <- c(bestest,bestmodP$minimum)
}
plotprofile(result,var="r",defpar=TRUE,lwd=2)
```

plotproj *plotproj generate a plot of a matrix of biomass projections*

Description

plotproj generate a plot of a matrix of N biomass projections and includes the option of including reference points relative to Bzero = K. Quantiles are included in the plot

Usage

```
plotproj(projs, spmout, qprob = c(0.1, 0.5, 0.9), refpts = NULL, fnt = 7)
```

Arguments

projs	the N x yrs matrix of biomass trajectories
spmout	the object output from the function spm
qprob	the quantiles to include in the plot, default=c(0.1,0.5,0.9)
refpts	the proportion of Bzero=K acting as limit and target reference points (a vector of two)
fnt	the font to use in the figure, default = 7 = bold Times

Value

This plots a graph and returns, invisibly, the requested quantiles and the proportion less than the limit reference point.

Examples

```
data(abdat)
schf <- FALSE
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
bestmod <- nlm(f=negLL1,p=param,funk=simpspm,logobs=log(abdat$cpue),
               indat=abdat,typsize=magnitude(param),iterlim=1000,
               schaefer=schf,hessian = TRUE)
out <- spm(bestmod$estimate,indat=abdat,schaefer=schf)
matpar <- parasympt(bestmod,50) # use at least 1000 in reality
projs <- spmproj(matpar,abdat,projyr=10,constC=900)
plotproj(projs,out,qprob=c(0.1,0.5),refpts=c(0.2,0.4))
```

plotspmdat *plotspmdat plots the fish data containing catches and cpue*

Description

plotspmdat plots a fish data set. It plots the catch and CPUE against time

Usage

```
plotspmdat(x, ...)
```

Arguments

x	a data set that should contain 'year', 'catch', and 'cpue'
...	extra parameters potentially used by plotspmdat

Value

nothing, but it does generate a new plot

Examples

```
yrs <- 2000:2010
catches <- rnorm(length(yrs),mean=150,sd=5)
ce <- rnorm(length(yrs),mean=20,sd=1)
fish <- as.data.frame(cbind(year=yrs,catch=catches,cpue=ce))
plotspmdat(fish)
```

plotspmmod *plotspmmod plots an spm model fit given parameters and data*

Description

plotspmmod takes a parameter set and the spmdat matrix and plots the predicted depletion, catch, the CPUE and the model fit to CPUE. If plotprod=TRUE, it also plots the productivity curves

Usage

```
plotspmmod(
  inp,
  indat,
  schaefer = TRUE,
  limit = 0.2,
  target = 0.48,
  addrmse = FALSE,
  fnt = 7,
```

```
  plotout = TRUE,
  vline = 0,
  plotprod = FALSE,
  maxy = 0
)
```

Arguments

inp	a vector of model parameters at least (r,K,B0)
indat	a matrix with at least columns 'year', 'catch', and 'cpue'
schaefer	if TRUE, the default, then the Schaefer SPM is used. If FALSE then an approximate Fox SPM would be used
limit	defaults to the Commonwealth limit of 0.2B0.
target	defaults to the Commonwealth target of 0.48B0. Determines the green line plotted on the exploitable biomass plot.
addrmse	default is FALSE but if set TRUE this will add the loess curve to the CPUE trend for comparison with the fitted line
fnt	the font used in the plot and axes. Default=7, bold Times. Using 6 gives Times, 1 will give SansSerif, 2 = bold Sans
plotout	should one generate a plot or only do the calculations; default is TRUE
vline	defaults to 0 = no action but if a year is inserted, which should generally be between two years to designate some change in the time-series between two years, then a vertical line will be added on year-based plots
plotprod	if FALSE, the default, the productivity curves are not plotted. If TRUE, two extra plots are produced.
maxy	defaults to 0, if > 0 then that value will be used in the plot of CPUE

Value

invisibly a list of dynamics, production curve, MSY, and Bmsy

Examples

```
data(dataspm)
pars <- log(c(0.264,4740,3064,0.2))
ans <- plotspmmod(pars,dataspm,schaefer=TRUE,plotprod=TRUE)
best <- nlm(negLL,pars,funk=simpspm,indat=dataspm,
       logobs=log(dataspm[,"cpue"]),steptol=1e-05)
outfit(best,backtran = TRUE)
ans <- plotspmmod(best$estimate,dataspm,schaefer=TRUE,plotout=TRUE,
                  plotprod=FALSE,addrmse=TRUE)
str(ans)
```

predfreq *predfreq is used for modal analysis of count data*

Description

predfreq is used to calculate the expected frequencies (counts) from a series of Normal distributions that are combined to describe observed counts from a sample of the size distribution of a population, which may, or may not be made up of a number of cohorts. When used with a negative log-likelihood from a multinomial this can lead to estimates of the mean and standard deviation of the size of each cohort. Two approaches are possible. An approximate method that calculates the likelihood at the center of each size-class and a more precise analytical method that subtracts the cumulative probability of the bottom bound from each size- class from the upper bound. Usually the two are very close.

Usage

```
predfreq(pars, n, sizecl, midval = TRUE)
```

Arguments

pars	a vector of parameters, with the means first, followed by the standard deviations, followed by the proportion of total observations in each cohort expect the last, which is obtained through subtraction.
n	the sum of counts observed from nature
sizecl	a representation of the size-classes used, these can be the mid-points of each size-class or the lower and upper bounds of each class.
midval	if TRUE, the default, the approximate analytical approach will be used.

Value

a vector of expected frequencies

Examples

```
## Not run:
mids <- seq(6,56,2) #size classes = 2 mm as in 5-7, 7-9,...
av <- c(18.0,34.5)   # the means
stdev <- c(2.75,5.5)  # the standard deviations
prop1 <- 0.55  # the proportion of observations in cohort 1
pars <-c(av,stdev,prop1)  # combine parameters into a vector
predf <- predfreq(pars,n=262,sizecl=mids,midval=TRUE)
oldpar <- par(no.readonly=TRUE)
plot1(mids,predf,xlab="Sizes",ylab="Predicted Frequency",lwd=2)
par(oldpar)

## End(Not run)
```

printV *printV returns a vector cbinded to 1:length(invect)*

Description

printV takes an input vector and generates another vector of numbers 1:length(invect) which it cbinds to itself. This is primarily useful when trying to print out a vector which can be clumsy to read when print across the screen. applying printV leads to a single vector being printed down the screen

Usage

```
printV(invect, label = c("index", "value"))
```

Arguments

invect	the input vector to be more easily visualized, this can be numbers, characters, or logical. If logical the TRUE and FALSE are converted to 1's and 0's
label	the column labels for the vector, default is 'index' and 'value'

Value

a dataframe containing the vector 1:length(invect), and invect.

Examples

```
vec <- rnorm(10,mean=20,sd=2)
printV(vec)
vec <- letters
printV(vec)
vec <- c(TRUE,TRUE,TRUE,FALSE,FALSE,TRUE,TRUE,FALSE,FALSE,TRUE,TRUE)
printV(vec,label=c("index","logicstate"))
```

properties *properties - used to examine the properties of a data.frame*

Description

properties examines some of the properties of a data.frame and outputs for each column an index for each variable, how many NAs there are, how many unique values there are, the class of each variable, the minimum and maximum (of all numeric variables), and an example value.

Usage

```
properties(indat, dimout = FALSE)
```

Arguments

indat	the data.frame containing the columns to be examined.
dimout	determines whether or not the dimensions of the data.frame are printed to the screen or not; defaults to FALSE

Value

a data.frame with the rows being each variable from the input data.frame and the columns being the number of NAs, the number of unique values, the class of variable, and minimum and maximum (where the columns are numeric), and an example value.

Examples

```
data(abdat)
properties(abdat)
```

---

pttuna *pttuna is yellowfin tuna fishery data from Pella-Tomlinson 1969*

Description

pttuna is yellowfin tuna fishery data from Pella-Tomlinson's (1969) classical paper describing their generalized surplus production model. This is the same data as contained in the schaef data-set, except it is extended from 1934 - 1967. Some of the values are slightly different, and their table rounds off the cpue estimates slightly differently but the catch and effort figures are theirs. It contains the year, the catch, the effort, and the cpue (which is just the total catch divided by the total effort, a ratio estimate). Initial parameter estimates close to the optimum values for the Schaefer model could be param <- log(c(r=0.28,K=2100000,Binit=2400000,sigma=0.16)). With this longer time-series the eventual MSY estimate was somewhat larger than when just the schaef data are used.

Format

A data.frame of fisheries data

year the fishing year from 1934 - 1967

catch the total annual catch, '000s of pounds

effort the total effort as standard class 4 baitboat fishing days

cpue the catch '000 pounds per standard class 4 day, a ratio cpue

Subjects

- surplus production modelling
- classical fisheries data
- Log-Normal likelihoods

Source

from Table 6 page 457 in Pella, J.J. and P.K. Tomlinson (1969) A Generalized Stock Production Model. *Bulletin, Inter-American Tropical Tuna Commission* 13(3): 421-458. Obtainable from *https://www.iattc.org/BulletinsENG.htm*

Examples

```
data(pttuna)
pars <- log(c(r=0.25,K=2.1e06,Binit=2.2e06,sigma=0.2))
answer <- fitSPM(pars,pttuna,schaefer=TRUE,maxiter=1000)
outfit(answer,title="Pella-Tomlinson Data",digits=4)
```

---

quants *quants used in apply to estimate quantiles across a vector*

Description

quants used in 'apply' to estimate quantiles across a vector. Can also be used within tapply, etc.

Usage

```
quants(invect, probs = c(0.025, 0.05, 0.5, 0.95, 0.975))
```

Arguments

invect	vector of values
probs	a vector of quantile, default=c(0.025,0.05,0.5,0.95,0.975)

Value

a vector of the probs quantiles

Examples

```
x <- matrix(rnorm(1000),ncol=10,nrow=100)
apply(x,2,quants)
apply(x,2,quants,probs=c(0.1,0.5,0.9))
```

removeEmpty *removeEmpty removes empty strings from a vector of strings*

Description

removeEmpty removes empty strings from a vector of strings. Such spaces can be created by spurious commas at the end of parsed lines. It also removes strings made up only of spaces and removes spaces from inside of individual chunks of text. This should be useful when reading in data from a custom csv file when parsing different formats

Usage

```
removeEmpty(invect)
```

Arguments

invect	a vector of input strings, possibly containing empty strings

Value

a possibly NULL vector of strings

Examples

```
x <- c("1","","2","","   ","3"," ","4","","a string","end")
x
length(x)
length(removeEmpty(x))
removeEmpty(x)
```

ricker *ricker one version of the Ricker stock recruitment*

Description

ricker implements the Ricker stock recruitment equation where R = aBexp(-bxB), R is the recruitment, a and b are the parameters and B is the spawning biomass. a is recruits-per-spawner at low stock levels, and b relates to the decline in recruitment as spawning biomass increases.

Usage

```
ricker(p, B)
```

Arguments

p	a vector of length two of the a and b parameters
B	a vector, possibly of length 1, of spawning biomass levels

Value

a vector of length = to B of predicted recruitments

Examples

```
B <- 1:10000
rec <- ricker(c(10,0.0002),B)
oldpar <- par(no.readonly=TRUE)
plot1(B,rec,xlab="SpB",ylab="Recruitment",lwd=2)
par(oldpar)
```

---

robustSPM *robustSPM does a robustness test on quality of fit of an SPM*

Description

robustSPM conducts a robustness test on the quality of fit of an SPM. This is done by using the original optimal model parameters or the original guessed parameter values, add random variation to each of them, and re-fit the model. This process needs to be repeated multiple times. This should enable an analysis of the stability of the modelling outcomes. If the optimum parameters are used then add more variation, if initial guesses are used you may need to select different starting points so that the random variation covers the parameter space reasonably well.

Usage

```
robustSPM(
  inpar,
  fish,
  N = 10,
  scaler = 40,
  verbose = FALSE,
  schaefer = TRUE,
  funk = simpspm,
  funkone = FALSE,
  steptol = 1e-06
)
```

Arguments

inpar	the parameter set with which to begin the trials
fish	the fisheries data: at least year, catch, and cpue
N	number of random trials to run; default = 10 = not enough
scaler	the divisor that sets the degree of normal random variation to add to the parameter values; default = 15 the smaller the value the more variable the outcome
verbose	progress and summary statistics to the screen? default = FALSE

schaefer	default = TRUE, which sets the analysis to the Schaefer model. setting it to FALSE applies the approximate Fox model
funk	the function used to generate the predicted cpue
funkone	defaults=FALSE; use negLL or negLL1, with FALSE robustSPM will use negLL, with TRUE it will use negLL1 which has a constraint on the first parameter to keep it > 0
steptol	is the steptol from nlm as used in fitSPM, the default value is 1e-06, as usual.

Value

a list containing the results from each run, the range of values across runs, and the median values.

Examples

```
data(dataspm)
param <- log(c(r=0.24,K=5174,Binit=2846,sigma=0.164))
ans <- fitSPM(pars=param,fish=dataspm,schaefer=TRUE,maxiter=1000)
out <- robustSPM(ans$estimate,dataspm,N=5,scaler=40,verbose=TRUE,
                 schaefer=TRUE) # N should be 50, 100, or more
str(out)
print(out$results)
pairs(out$results[,c(6:8,11)]) # need a larger N!
```

---

schaef *schaef is yellowfin tuna fishery data from Schaefer 1957*

Description

schaef is yellowfin tuna fishery data from Schaefer (1957) It contains the year, the catch, the effort, and the cpue and was used in one of the first descriptions of a stock assessment that used a surplus production model. The catch-per- unit-effort, cpue, is a ratio cpue of the total catch divided by the total effort as thousands of pounds per day. These days such ratios tend not to be used, with individual records for each day's effort being used instead. Using individual records does not obscure the variation between different vessels, areas, depths, and seasons. Initial parameter estimates close to the optimum values for both the Schaefer model and the Fox model could be param <- log(c(r=0.24,K=2100000,Binit=2200000,sigma=0.2))

Format

A data.frame of fisheries data

year the fishing year from 1934 - 1955

catch the total annual catch, '000s of pounds

effort the total effort as standard class 4 clipper fishing days

cpue the catch '000 pounds per standard class 4 day, a ratio cpue

Subjects

- surplus production modelling
- classical fisheries data
- Log-Normal likelihoods

Source

from Table 1 page 266 in Schaefer, M.B. (1957) A study of the dynamics of the fishery for yellowfin tuna in the Eastern Tropical Pacific Ocean. Bulletin, Inter-American Tropical Tuna Commission 2: 247-285. Obtainable from *https://www.iattc.org/BulletinsENG.htm*

Examples

```
data(schaef)
pars <- log(c(r=0.2,K=2.1e06,Binit=2.2e06,sigma=0.2))
answer <- fitSPM(pars,schaef,schaefer=TRUE,maxiter=1000)
outfit(answer,title="Schaefer, 1957 Data",digits=4)
```

---

setpalette *setpalette is a shortcut for altering the palette to R4*

Description

setpalette is a shortcut for changing the color palette to the R 4.0.0 default version. The R4 palette provides a less garish and a more visible set of default colours that can be called using the numbers 1 - 8. An important point is that this alters the default colours for all sessions until a restart of R. Using something similar you can define your own preferred palettes should you wish to. In addition, it can be used to revert to the old R3 default colour scheme should you wish. Alternatively, you could define your own palettes and switch between them using setpalette.

Usage

```
setpalette(x = "R4")
```

Arguments

x	either "default", "R3", or "R4", with R4 as the default value. Use "R3" to revert back to the standard R version 3. values.

Value

nothing but it does alter the base colour palette

Examples

```
setpalette("R3")
plot(1:8,rep(0.25,8),type="p",pch=16,cex=5,col=c(1:8))
text(1,0.2,"R3.0.0 - some garish or pale",cex=1.5,
     font=7,pos=4)
setpalette("R4")
points(1:8,rep(0.3,8),pch=16,cex=5,col=c(1:8)) #toprow
text(1,0.325,"Default R4.0.0 - more balanced",cex=1.5,
     font=7,pos=4)
```

`simpspm` *simpspm calculates only the predicted log(CE) for an SPM*

Description

simpspm calculates only the predicted log(CPUE) for a Surplus Production Model (SPM). It sets the Polacheck et al, 1993 parameter 'p' depending on the schaefer boolean argument, and this determines the asymmetry of the production curve. Note p is set not estimated. If p = 1.0 then the SPM is the Schaefer model, if it is 1e-8 it approximates the Fox model. The output of log(CPUE) is to simplify the use of log-normal residual errors or likelihoods. This function is designed for data consisting of only a single cpue time-series. simpspm must have at least three parameters, including the sigma, even if sum-of-squared residuals is used as a minimizer, then sigma would just float. The column names for year, catch and cpue are included to facilitate ease of use with other data sets.

Usage

```
simpspm(
  pars,
  indat,
  schaefer = TRUE,
  year = "year",
  cats = "catch",
  index = "cpue"
)
```

Arguments

`pars`	the parameters of the SPM are either c(r,K,Binit,sigma), or c(r, K, sigma), the sigma is required in both cases. Binit is required if the fishery data starts after the stock has been depleted. Each parameter must be log-transformed for improved model stability and is back-transformed inside simpspm.
`indat`	the data which needs to include year, catch, and cpue.
`schaefer`	a logical value determining whether the spm is to be a simple Schaefer model (p=1) or approximately a Fox model (p=1e-08). The default is TRUE = Schaefer model
`year`	column name within indat containing the years, default='year'
`cats`	column name within indat containing the catches, default='catch'
`index`	column name within indat containing the cpue. default='cpue'

Value

a vector of length nyrs of the predicted log(cpue)

Examples

```
data(abdat)
param <- log(c(r=0.4,K=9400,Binit=3400,sigma=0.05))
predCE <- simpspm(pars=param,indat=abdat)
cbind(abdat,exp(predCE))
```

`simpspmM` *simpspmM calculates predicted CE when multiple indices are present*

Description

simpspmM calculates the predicted CPUE for an surplus production model and is designed for when there are more than one time-series of an index of relative abundance. The parameter vector includes the standard deviations for the log-normal distributions assumed for the indices of relative abundance. They are not used in this function but will be when the likelihoods are calculated as a next step in fitting the model.

Usage

```
simpspmM(
  pars,
  indat,
  schaefer = TRUE,
  year = "year",
  cats = "catch",
  index = "cpue"
)
```

Arguments

`pars`	the parameters of the SPM = r, K, Binit if fishery is depleted to start with (omit otherwise), and a sigma for each cpue series. Each parameter is in log space and is back-transformed inside simpspmM.
`indat`	the data which needs to include year, catch, and cpue. The latter should have a separate column for each fleet, with a column name beginning with cpue or whatever name you put in index (see below) for example cpue1, cpue2, etc.
`schaefer`	a logical value determining whether the spm is to be a simple Schaefer model (p=1) or approximately a Fox model (p=1e-08). The default is TRUE
`year`	column name within indat containing the years, default='year'
`cats`	column name within indat containing the catches, default='catch'
`index`	the prefix in the column names given to the indices of relative abundance used, perhaps 'cpue' as in cpueTW, cpueAL, etc. grep is used to search for columns containing this prefix to identify whether there are more than one column of cpue data. Be sure to only use the prefix for indices of relative abundance.

Value

a vector or matrix of nyrs of the predicted CPUE

Examples

```
data(twoindex)
fish <- as.matrix(twoindex)
pars <- log(c(0.04,155000,0.4,0.3))
bestSP <- nlm(f=negLLM,p=pars,funk=simpspmM,indat=fish,
            schaefer=TRUE,logobs=log(fish[,c("cpue1","cpue2")]),
            steptol=1e-06,harvpen=TRUE)
outfit(bestSP)  # best fitting estimates
simpspmM(bestSP$estimate,fish) # the two log(predicted cpue)
```

---

spm *spm calculates the dynamics of a Schaefer or Fox model*

Description

spm calculates the dynamics using a Schaefer of Fox model. The outputs include predicted Biomass, year, catch, cpue, predicted cpue, contributions to q, ssq, and depletion levels. Generally it would be more sensible to use simpspm when fitting a Schaefer model or a Fox model as those functions are designed to generate only the log of predicted cpue as required by the functions ssq and negLL, but the example shows how it could be used. The function spm is used inside 'plotspmmod' and could be used alone, to generate a full list of model outputs after the model has been fitted. spm is designed for working with a single vector of an index of relative abundance. If there are multiple vectors of the index then use simpspmM and spmCE.

Usage

```
spm(inp, indat, schaefer = TRUE, year = "year", cats = "catch", index = "cpue")
```

Arguments

inp	a vector of 3 or 4 model parameters (r,K,sigma) or (r, K, Binit,sigma), you would use the latter if it was suspected that the fishery data started after some initial depletion had occurred. The sigma is an estimate of the variation of the cpue through time. This is required but is only used when fitting the model using negative log-likelihoods.
indat	a matrix with at least columns year, catch, and cpue
schaefer	a logical value determining whether the spm is to be a simple Schaefer model (p=1) or approximately a Fox model (p=1e-08). The default is TRUE
year	the column name of the year variable (in case your dataset names it fishingyear-inwhichthecatchwastaken), default='year'
cats	column name of the catch variable, default='catch'
index	the name of the cpue variable, default='cpue'

Value

a list of five objects; parameters plus q, then outmat, the matrix with the dynamics, msy the maximum sustainable yield, and sumout, which contains r,K,B0,msy,p,q,Depl, FinalB, and InitDepl

Examples

```
data(abdat)   # spm is used inside plotspmmod
pars <- log(c(0.35,7800,3500,0.05))
ans <- plotspmmod(pars,abdat) #not fitted, just guessed
bestSP <- fitSPM(pars=pars,fish=abdat,funk=simpspm)
outfit(bestSP)  # best fitting estimates
ans <- plotspmmod(bestSP$estimate,abdat,schaefer=TRUE)
str(ans)
```

---

spmboot *spmboot conducts a bootstrap analysis on a spm model*

Description

spmboot conducts a bootstrap analysis on a spm model. It does this by saving the original fishery data, estimating the cpue residuals, and multiplying the optimum predicted CPUE by a bootstrap sample of the log-normal residuals (Haddon, 2011, p311; and the On Uncertainty chapter in the URMQMF book, p195). This bootstrap sample of CPUE replaces the original fish[,"cpue"] and the model is re-fitted. This is repeated iter times and the outputs reported ready for the derivation of percentile confidence intervals. The optimum solution is used as the first bootstrap replicate (it is standard practice to include the original fit in the bootstrap analysis). If 1000 replicates are run this procedure can take a couple of minutes on a reasonably fast computer. A comparison of the mean with the median should provide some notion of any bias in the mean estimate.

Usage

```
spmboot(optpar, fishery, iter = 1000, schaefer = TRUE)
```

Arguments

optpar	The optimum model parameters from fitting a surplus production model
fishery	fishery data containing the original observed cpue values
iter	the number of bootstrap replicates to be run, default=1000
schaefer	default=TRUE, should a Schaefer or a Fox model be run

Value

a list of two matrices. One containing the bootstrap parameters and the other containing some of the dynamics, including the ModelB, the bootstrap CPUE sample, the Depletion, and annual harvest rate.

Examples

```
data(dataspm); fish <- as.matrix(dataspm)
pars <- log(c(r=0.24,K=5150,Binit=2800,0.15))
ans <- fitSPM(pars,dataspm,schaefer=TRUE,maxiter=1000)
boots <- spmboot(ans$estimate,fishery=fish,iter=5,schaefer=TRUE)
dynam <- boots$dynam
bootpar <- boots$bootpar
rows <- colnames(bootpar)
columns <- c(c(0.025,0.05,0.5,0.95,0.975),"Mean")
bootCI <- matrix(NA,nrow=length(rows),ncol=length(columns),
              dimnames=list(rows,columns))
for (i in 1:length(rows)) {
  tmp <- sort(bootpar[,i])
  qtil <- quantile(tmp,probs=c(0.025,0.05,0.5,0.95,0.975),na.rm=TRUE)
  bootCI[i,] <- c(qtil,mean(tmp,na.rm=TRUE))
}
round(bootCI,3) # we used only 5 bootstraps for a speedy example, normally
pairs(bootpar[,c("r","K","Binit","MSY")]) # use 1000, 2000, or more
```

---

spmCE *spmCE - calculates the dynamics for multiple cpue time-series*

Description

spmCE calculates the full dynamics using a Schaefer of Fox model and is used instead of spm when there are multiple index vectors. The outputs include predicted Biomass, year, catch, cpue, predicted cpue, contributions to q, ssq, and depletion levels. Generally it would be more sensible to use simpspmM when fitting a Schaefer or Fox model as that function is designed to generate only the predicted cpue vectors required by the function negLLM, nevertheless, the example shows how it could be used.

Usage

```
spmCE(
  inp,
  indat,
  schaefer = TRUE,
  year = "year",
  cats = "catch",
  index = "cpue"
)
```

Arguments

inp	a vector of 2 or 3 model parameters (r,K) or (r,K,Binit), you would use the latter if it was suspected that the fishery data started after some initial depletion had occurred. In addition, there should then be the same number of sigma values as there are cpue time-series. For two cpue series with an initial depletion we would expect to have r, K, Binit, sigma1 and sigma2

indat	a matrix with at least columns 'year', 'catch', and 'cpue'
schaefer	a logical value determining whether the spm is to be a simple Schaefer model (p=1) or approximately a Fox model (p=1e-08). The default is TRUE
year	column name within indat containing the years, default='year'
cats	column name within indat containing the catches, default='catch'
index	column name within indat containing the prefix for cpue, default='cpue'

Value

a list of five objects; outmat the matrix with the dynamics results, q catchability, msy the maximum sustainable yield, the parameter values, and sumout, which contains r, K, B0, msy, p, q, Depl, FinalB, and InitDepl

Examples

```
data(twoindex)
fish <- as.matrix(twoindex)
pars <- log(c(0.04,155000,0.4,0.3))
bestSP <- nlm(f=negLLM,p=pars,funk=simpspmM,indat=fish,
            schaefer=TRUE,logobs=log(fish[,c("cpue1","cpue2")]),
            steptol=1e-06,harvpen=TRUE)
outfit(bestSP)  # best fitting estimates
getMSY(exp(bestSP$estimate))
```

---

spmphaseplot *spmphaseplot - plots the phase plot of harvest rate vs biomass*

Description

spmphaseplot uses the output from plotspmmod to plot up the phase plot of harvest rate vs Biomass, marked with the limit and default targets. It identifies the start and end years (green and red dots) and permits the stock status to be determined visually. It also plots out the catch time-series and harvest rate time-series to aid in interpretation of the phase plot.

Usage

```
spmphaseplot(
  answer,
  Blim = 0.2,
  Btarg = 0.5,
  filename = "",
  resol = 200,
  fnt = 7
)
```

Arguments

answer	the object output by the function plotspmmod, containing the production curve, the fishery dynamics (predicted harvest rate and biomass through time).
Blim	limit reference point, default = 0.2 = 0.2B0.
Btarg	what target reference point will be used in the phase plot. A default of 0.5 is used.
filename	default is empty. If a filename is put here a .png file with that name will be put into the working directory.
resol	the resolution of the png file, defaults to 200 dpi
fnt	the font used in the plot and axes. Default=7, bold Times. Using 6 gives Times, 1 will give SansSerif, 2 = bold Sans

Value

an invisible list of B0, Bmsy, Hmsy, and Hlim.

Examples

```
data(dataspm)
pars <- log(c(0.164,6740,3564,0.05))
bestSP <- fitSPM(pars,fish=dataspm,funkone=TRUE)
ans <- plotspmmod(bestSP$estimate,dataspm,schaefer=TRUE,addrmse=TRUE)
str(ans)
outs <- spmphaseplot(ans,fnt=7)
str(outs)
```

---

spmproj *spmproj calculates biomass trajectories for replicate parameters*

Description

spmproj uses a matrix of parameter vectors to project surplus production dynamics forward including future projection years under constant catches. This is used to conduct risk assessments for different constant catches allowing a search for optimum future catch levels.

Usage

```
spmproj(
  parmat,
  indat,
  constC,
  projyr = 10,
  year = "year",
  cats = "catch",
  index = "cpue"
)
```

Arguments

parmat	a matrix of N parameter vectors obtained from either asymptotic errors (parasympt), bootstraps (bootpar), or from a Bayesian analysis (parsbayes).
indat	the fisheries data used during model fitting
constC	the constant catch level imposed in the projection years
projyr	the number of years of projection, default = 10
year	name of the year variable within indat, default=year
cats	name of the catch variable within indat, default=catch
index	name of the cpue variable within indat, default=cpue

Value

an N x years matrix of biomass trajectories, one for each parameter vector

Examples

```
data(abdat)
schf <- FALSE
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
bestmod <- nlm(f=negLL1,p=param,funk=simpspm,logobs=log(abdat$cpue),
               indat=abdat,typsize=magnitude(param),iterlim=1000,
               schaefer=schf,hessian = TRUE)
out <- spm(bestmod$estimate,indat=abdat,schaefer=schf)
matpar <- parasympt(bestmod,10) # normally use 1000 or more
projs <- spmproj(matpar,abdat,projyr=10,constC=900)
plotproj(projs,out)
```

---

spmprojDet *spmprojDet conducts forward projections from known conditions*

Description

spmprojDet conducts deterministic forward projections of the dynamics of a fitted surplus production model. The parameters and original data need to be put through the function spm to form the spmobj, i.e. the list generated by the function spm. This contains all required information except for details of the projection. The application of spm is where the dynamics are defined as either Schaefer or Fox. If no plot is generated then the projected dynamics are output invisibly, where the biomass and predCE are matrices of years vs projcatch.

Usage

```
spmprojDet(spmobj, projcatch, projyr = 10, plotout = FALSE, useft = 7)
```

Arguments

spmobj	the list generated by the function spm
projcatch	the projected constant catch levels as a vector
projyr	the number of years of projection. default = 10
plotout	should the projection be plotted? default=FALSE
useft	which font used in a plot? default=7 = bold times

Value

the projected biomass, CPUE, and the projected years

Examples

```
data(abdat)
param <- log(c(r=0.3,K=11500,Binit=3300,sigma=0.05))
bestmod <- nlm(f=negLL1,p=param,funk=simpspm,logobs=log(abdat$cpue),
               indat=abdat,typsize=magnitude(param),iterlim=1000,
               schaefer=FALSE)
out <- spm(bestmod$estimate,indat=abdat,schaefer=FALSE)
catches <- seq(700,1000,50)
spmprojDet(spmobj = out,projcatch=catches,projyr=10,plotout=TRUE)
```

---

srug *srug is the Schnute and Richards Unified Growth Curve*

Description

srug implements the Schnute and Richards (1990) unified growth curve that can be used to describe fish growth, maturation, and survivorship data. It is a curve that generalizes the classical logistic model used for maturity as well the growth models by Gompertz (1825), von Bertalanffy (1938), Richards (1959), Chapman (1961), and Schnute (1981). As with any asymmetric, multi-parameter model, it can be hard to obtain a stable fit of this curve to data. Here the model is implemented to range between 0 - 1, if you want to use it to describe growth then re-cast the function and add a fifth parameter to replace the 1.0 on top of the divisor. The main point of the curve, however, was to demonstrate how the different equations were related to one another. In working situations it is most efficient to use the original, simpler, curve/equation.

Usage

```
srug(p, sizeage)
```

Arguments

p	a vector of four parameters begin Schnute and Richards' a, b, c, and alpha, in that order.
sizeage	the age or size data used to describe the maturity transition.

Value

A vector of predicted proportion mature (proportion of 1.0) for the given parameters and the sizeage data

References

Schnute, J.T. and L.J. Richards (1990) A unified approach to the analysis of fish growth, maturity, and survivorship data. *Canadian Journal of Fisheries and Aquatic Science* 47:24-40

Examples

```
L <- seq(50,160,1)
p <- c(a=0.07,b=0.2,c=1.0,alpha=100.0)
predR <- srug(p=p,sizeage=L) # proportion of total
oldpar <- par(no.readonly=TRUE)
plot1(L,predR,xlab="Length",ylab="Prop of Recruitment")
abline(h=0.5) #visually confirm asymmetry
par(oldpar)
```

`ssq` *ssq a generalized function for summing squared residuals*

Description

ssq is a generalized function for summing squared residuals which is designed for ease of use in nlm (or optim, or nlminb). NAs are removed automatically. It assumes the input of a predefined function 'funk' that will calculate predicted values of a 'dependent' variable from a vector of 'independent' or observed values, for which one has observations. The dependent or predicted values are compared with the observed or 'independent' data, and the resulting SSQ returned. The use of ... means this is a very general function but it does mean you need to be very careful with placement and spelling of the input variables required by whatever funk you are using. It is always best to explicitly name the arguments although the correct order will also work correctly.

Usage

```
ssq(funk, observed, ...)
```

Arguments

`funk`	a function that uses a parameter vector and vector of observations to calculate predicted values, which are compared with the observed values to give the ssq and be returned
`observed`	a vector containing the observed data (y-axis)
`...`	required to allow funk to access its parameters and data without having to explicitly declare them in ssq. Note that this means inside the ssq function the call to funk also needs to have the ellipsis, otherwise it will not be able to see those other arguments. It is vital to spell the names of funk's arguments correctly as errors are not always announced and will obviously lead to misleading outputs.

Value

a single number (scaler) that is the sum of squared residuals between the dep values and those calculated by funk

Examples

```
data(minnow)  # remember -13 is only 3+ months
pars <- c(89, 0.1,-13)  # ssq = 83477.84
ssq(funk=vB,observed=minnow$length,p=pars,ages=minnow$week)
```

---

summspm *summspm extracts critical statistics from output of plotspmmod*

Description

summspm extracts critical statistics from output of plotspmmod. In particular it pulls out the catchability q, the MSY, Bmsy, Dmsy, Blim, Btarg, Ctarg. and Dcurr.

Usage

```
summspm(ans)
```

Arguments

ans	the object output from the function plotspmmod used to plot the output of a surplus production analysis

Value

a matrix of statistics relating to MSY, expected yields, and depletion levels

Examples

```
data(dataspm)
plotfishM(dataspm,spsname="Pink Ling")
pars <- log(c(0.264,4740,3064,0.05))
ans <- plotspmmod(pars,dataspm,schaefer=FALSE,plotout=FALSE)
bestSP <- fitSPM(pars,dataspm,schaefer=FALSE,maxiter=1000)
outfit(bestSP)
ans <- plotspmmod(bestSP$estimate,dataspm,schaefer=FALSE,plotout=FALSE)
summspm(ans)
```

tasab *tasab is a matrix of abalone maturity-at-length data*

Description

tasab is a 715 x 4 matrix of maturity-at-length data for blacklip abalone (*Haliotis rubra*) from two sites along the Tasmanian west coast. All data was collected in February 1995, but details, such as site name, accurate location, statistical block, year, month, and other details have been omitted for brevity. See section on maturity within the Static Models chapter for detailed use of this data-set.

Format

A data.frame of maturity-at-length data

site an identifier for the two different sites sampled

sex I = immature, M = male, F = female

length the shell length in mm

mature was the animal mature = 1 or not = 0

Subjects

- maturity ogives or logistic curves
- Binomial likelihoods

Source

Many thanks to the Institute of Marine and Antarctic Science, which is part of the University of Tasmania, and especially to Dr Craig Mundy, leader of the Abalone Group, for permission to use this data collected in February 1995.

Examples

```
data(tasab)
head(tasab,20)
table(tasab$site,tasab$sex)
```

tigers *tigers is tiger prawn recruitment data from Penn & Caputi 1986*

Description

tigers is a dataset of only 14 rows of data with a column of Spawning index and Recruitment index, as a data.frame. The timing of the recruitment index is up to half a year after the spawning index.

Format

A data.frame of spawning recruitment data

Spawn the estimated spawning biomass index in a year (Aug - Oct)

Recruit estimated recruitment from the biomass in each year (Mar - May)

Subjects

- Stock-recruitment curves
- Beverton-Holt and Ricker Models
- Static model fitting

Source

Extracted from Table 2, page 496 of Penn, J.W. and N. Caputi (1986) Spawning stock-recruitment relationships and environmental influences on the tiger prawn (*Penaeus esculentus*) fishery in Exmouth Gulf, Western Australia. *Australian Journal of Marine and Freshwater Research* 37: 491-505. Sorted on spawning index.

Examples

```
data(tigers)
tigers
oldpar <- par(no.readonly=TRUE)
plot1(tigers$Spawn,tigers$Recruit,type="p",pch=16,cex=1.25)
par(oldpar)
```

twoindex *twoindex has orange roughy catches with hypothetical cpue*

Description

twoindex is a 35 x 4 data.frame of fishery data made up of smoothed real catches but two simulated indices of relative abundance. This data-set is designed to be used to illustrate the implementation of surplus production models when there are more than one time-series of relative abundance indices. This is not currently implemented in the book but is put here for use by readers should they wish to pursue this approach. The indices have been designed to generate a workable answer but also require the use of a penalty on harvest rates to avoid massively inflated harvest rates well above 1. Instead of using simpspm, spm, and negLL1, we need to use simpspmM, spmCE, and negLLM. The cpue series are hypothetical and have been designed to illustrate the use of penalty1 and the use of multiple indices of relative abundance. The real stock assessment uses acoustic survey indices and also uses many years of age composition data inside Stock Synthesis 3, not surprisingly the inclusion of real time-series of indices and of age-composition data leads to very different results.

Format

A data.frame of fishery data

year the calender year of fishing

catch the reported catch in tonnes

cpue1 the first index of relative abundance

cpue2 the second index of relative abundance

Subjects

- Surplus production models
- Dynamic model fitting
- -ve log-likelihoods

Source

Catches extracted from Table 4, page 11 of Haddon, M. (2017) Orange Roughy East (Hoplostethus atlanticus) stock assessment using data to 2016 Report to November 2017 SE RAG meeting. CSIRO, Oceans and Atmosphere, Australia. 51p. from https://www.afma.gov.au/fisheries-management/species/orange-roughy Catch data extended to 2019 using AFMA's catchwatch system.

Examples

```
data(twoindex)
fish <- as.matrix(twoindex)
pars <- log(c(0.04,155000,0.4,0.3))
bestSP <- nlm(f=negLLM,p=pars,funk=simpspmM,indat=fish,
            schaefer=TRUE,logobs=log(fish[,c("cpue1","cpue2")]),
            steptol=1e-06,harvpen=TRUE)
```

```
namepar <- c("r", "K", "Binit","sigma")
outfit(bestSP,parnames=namepar)  # best fitting estimates
# if 'Either ~local min or steptol too small try 'steptol=1e-05'
# plotprep(width=7,height=5,newdev=FALSE) # for external plot
answer <- plotspmmod(bestSP$estimate,indat=fish,
                     plotprod=TRUE,maxy=3.4)
```

---

uphist *uphist a histogram with an upper limit on the x-axis*

Description

uphist is merely a wrapper around the base hist function, which adds the ability to limit the upper value on the x-axis. With fisheries data it is surprisingly common to have data that has a very few extreme values that can obscure a standard plot of the data. The data are only truncated within the uphist function so any other analyses will be on all available data. If a maximum value is selected which accidently eliminates all available data the script stops with an appropriate warning. If a value is selected which fails to eliminate any data then all data are used with no warning.

Usage

```
uphist(x, maxval = NA, ...)
```

Arguments

x	the vector of values to be plotted as a histogram
maxval	the maximum value to be retained in the plotted data
...	any of the other arguments used by the base hist function

Value

nothing, but it does plot a histogram

Examples

```
x <- rlnorm(5000, meanlog=2, sdlog=1)
hist(x,breaks=30,main="",xlab="log-normal values")
uphist(x,breaks=30,main="",xlab="log-normal values",maxval=100)
uphist(x,breaks=30,main="",xlab="log-normal values",maxval=1000)
```

vB *vB calculates the predicted von Bertalanffy length at age*

Description

vB calculates length at age for the von Bertalanffy curve.

Usage

```
vB(p, ages)
```

Arguments

p	is a vector the first three cells of which are Linf, K, and t0 for the vB curve.
ages	is a vector of ages; could be a single number

Value

a vector of predicted lengths for the vector of ages in 'ages'

Examples

```
ages <- seq(0,20,1)   # sigma is ignored here
pars <- c(Linf=50,K=0.3,t0=-1.0,sigma=1.0) # Linf, K, t0, sigma
oldpar <- par(no.readonly=TRUE)
plot1(ages,vB(pars,ages),xlab="Age",ylab="Length",lwd=2)
par(oldpar)
```

which.closest *which.closest find a number in a vector closest to a given value*

Description

which.closest finds either the number in a vector which is closest to the input value or its index value

Usage

```
which.closest(x, invect, index = TRUE)
```

Arguments

x	the value to lookup
invect	the vector in which to lookup the value x
index	should the index be returned or the closest value; default=TRUE

Value

by default it returns the index in the vector of the value closest to the input value x

Examples

```
vals <- rnorm(100,mean=5,sd=2)
pick <- which.closest(5.0,vals,index=TRUE) #closest to 5?
pick        # the index of the closest
vals[pick]  # its value
which.closest(5.0,vals,index=FALSE) # straight to the value
```

%ni% *'%ni%' identifies which element in x is NOT in y*

Description

'%ni%' identifies which element in x is NOT in y

Usage

```
x %ni% y
```

Arguments

x	a vector of elements which can be numeric or character
y	a vector of elements which can be numeric or character

Examples

```
x <- 1:10
y <- 6:15
x
y
x[(x %ni% y)]   # are not in y
```

Index