Wim’s Functions, sent 11/22/22

library(MASS) # Load first so "select" is from dplyr which is loaded later

library(ggplot2)

library (reshape2)

library(grDevices)

library(dplyr)

library(cowplot)

cat ("\nLoading functions\n")

# Helper functions

# Functions to work with dates

# Functions to be used with base R plotting

# Functions to be used with ggplot

# Statistical functions

# Functions for data manipulation

# Functions for manipulating Monitoring data sets (water, zooplankton, fish)

# Functions for specific calculations (oceanography, ecology)

# Redefinitions

select <- dplyr::select # I use select only with dplyr

filter <- dplyr::filter # I use select only with dplyr

View <- utils::View # I prefer the separate View windows even though they don't update (jagsUI requires this)

# Helper functions-----------

unfact <- function(x, numeric=T, quiet = F)

{

# Brought over from Splus 11/27/2020

# Converts a factor to character or numeric

# Numeric if able and numeric=T, otherwise character

# If mixed it will return numeric and NA unless numeric=F

# quiet = F causes an announcement if values retained as character

if(!is.factor(x))

warning("Wim: Input is not a factor")

else {

opt <- unlist(options("warn"))

options(warn = -1.)

x <- as.character(x)

if(numeric) {

y <- as.numeric(x)

if(all(is.na(y))) {

if(!quiet)

cat("Values retained as character \n")

}

else {

x <- y

if(any(is.na(y)))

# This wont print

cat("Some missing values returned \n")

}

}

options(warn = opt)

}

x

}

yearlab <- function(year)

{

right(as.character(year),2)

}

catclip <- function(x) cat(x, file="clipboard", sep="\n")

exList <- function(xlist)

{

# assigns all components of a list to their names

# (i.e., extracts all components)

if (!is.list(xlist)) stop ("Input is not a list\n")

nms <- names(xlist)

if (is.null(nms)) names(xlist) <- paste0("xlist.", 1:length(xlist))

for (i in 1:length(xlist))

assign(nms[i], xlist[[i]], envir = .GlobalEnv)

}

# exList(list(a=data.frame(a=1:4, b=4:1), b=LETTERS[1:5]))

resultInCode <- function(result, summary=T)

{

# 8/10/2019

# Prints any result (?) to clipboard with leading "# " to insert into code.

if (summary) result <- summary(result)

sink("clipboard")

print(result)

sink()

w <- scan(file="clipboard", what="", sep="\n")

w <- paste("#", w)

cat(w, file="clipboard", sep="\n")

cat("Paste to console or external file\n")

}

# resultInCode(lm(y~x, data=data.frame(x=1:5, y=rnorm(5)+ 5:1)))

first <- function (x, n=1) head(x,n=n )

writedf <- function(df)

{

write.table(df, file="clipboard-16384", sep="\t", quote=F, row.names=F)

}

wmerge <- function(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, ...)

# This function conducts a merge, then sorts the result by the x indices

# the correct way using wsort

{

xx <- merge(x, y, by = intersect(names(x), names(y)), by.x = by, by.y = by, ...)

xx<- wsort(xx,by.x)

xx

}

cleanup <- function(ask = T, echo=ask, exclude = "Par")

{

# This function erases all variables with 3 or fewer letters in the name,

# and any single-valued numeric, character, or logical variables.

# ask = T to ask to delete files (default; set to F if used in a function)

# echo = T to list out the files to be deleted.

# exclude lists files not to be deleted

v <- objects(name=".GlobalEnv", pattern = "\*")

v <- v[substring(v, 1, 1) != "."]

numbers <- logical(length = length(v))

for(i in 1:length(v)) {

w <- get(v[i])

wc <- class(w)[1]

numbers[i] <- match(wc, c("integer", "numeric", "character",

"logical", "named"), nomatch = 0) > 0

}

littles <- objects(name=".GlobalEnv")

littles <- littles[nchar(littles) <= 3]

v <- unique(c(v[numbers], littles))

if(length(exclude) > 0)

v <- v[match(v, exclude, nomatch = 0) == 0]

if(length(v) > 0) {

if (echo)

cat(c(v, "\n"))

if(ask)

yn <- readline("Erase the above-listed files?")

else yn <- "Y"

if((yn == "Y" | yn == "y")) {

if (echo)

cat("Erasing\n")

rm(list = v, pos=1)

}

}

else cat("Nothing erased \n")

}

wsort <- function(x, ind = 1.)

{

# Sorts the data frame x by indices selected as column numbers ind

# Any number of columns can be included

# ind is a vector of indices, if negative they are sorted in reverse order

# The algorithm builds the "order" command (which takes any number of vectors)

# by assembling a text string of indices, then evaluating it.

#

# Inputs

# x A data frame

# ind Column numbers or column names over which to sort, in the desired order

# If ind is numeric and negative sorting will be in descending order

# but only if the respective column is numeric (not factor or character)

if (is.character(ind))

{

ind1 <- match(ind, names(x))

if (any (is.na(ind1))) stop (paste("Index",ind[is.na(ind1)], "Not in data set"))

ind <- ind1

}

if(max(abs(ind)) > dim(x)[2.]) stop(

"Indices exceed dimensions of data frame")

sgn <- sign(ind)

ind <- abs(ind)

u <- paste("x[,", ind, "]", sep = "")

u <- ifelse(sgn > 0, u, paste("-", u, sep = ""))

u <- paste("x[order(", paste(u, collapse = ", "), "),]", sep = "")

eval(parse(text = u))

}

right <- function(x, n)

{

# Gets rightmost n characters of vector x

# If n is negative it keeps all BUT the rightmost n characters

nc <- nchar(x)

if (n[1] > 0)

v <- substring(x, nc - n + 1)

else

v <- substring(x, 1, nc + n )

v

}

chfind <- function(x, pattern, begin = 1, pick = "first", wild=F)

{

# Finds pattern of characters in character vector x

#

# So wild = F means it acts like old chfind (no wild cards)

# wild = T means "." is any character

# I also made this recursive for when pick is "all"

if (pattern == "." & !wild) pattern <- "\\."

pick <- tolower(substring(pick, 1, 1))

if (begin > 1 & pick == "a") begin <- 1

if(!(length(pattern) == 1 | length(pattern) == length(x)))

stop("Length of pattern must be 1 or length of x")

if (begin > 1) x <- substring(x, begin)

if(pick == "f")

{

vv <- as.vector(regexpr(pattern, x))

vv <- ifelse(vv <0, 0, vv + begin -1)

}

else

{

nc <- 1

while (max(nchar(x)) > 0)

{

vi <- chfind(x, pattern, pick="f")

if (nc == 1) vv <- vi

else vv <- cbind(vv, vi)

if (all(vi ==0)) break

nc <- nc + 1

x <- substring(x, ifelse(vi==0, nchar(x), vi + 1))

}

vv <- as.list(data.frame(t(vv)))

vv <- lapply(vv, function (v) v <- cumsum(v[v>0]))

if (pick == "l")

vv <- unlist(lapply(vv, function (v) ifelse (length(v)==0, 0, max(v)) ))

}

if (is.null (names(x))) names (vv) <- NULL

else names(vv) <- names(x)

vv

}

#x <- c("abcdef", "aabbca", "xx.xxx", "axxxa" )

#pattern <- "a"

#chfind (x,"a", pick="first")

#chfind (x,"a", pick="last")

#chfind(x, ".")

#chfind(x, ".", wild=T)

#chfind(x, "a", begin=3)

#chfind (x,"a", pick="all")

explode <- function (char)

{

# Returns a character vector made of all the characters in a single-valued string

# Or a list of vectors for a multiple-valued string

lapply (char, function (v)

{

vv <- character(length=nchar(v))

for (i in 1:length(vv)) vv[i] <- substring(v, i, i)

vv

}

)

}

makedf <- function (names, first.col=NULL, nrow=1, fill=NA )

{

# Function to set up a data frame to be filled.

# Input:

# names Names of the columns

# first.col If supplied, the values for the first column of the data frame

# nrow Number of rows (not needed if first.col is supplied)

# fill Value to fill the df with

if (!is.null (first.col))

nrow <- length(first.col)

x <- data.frame(matrix(fill, nrow=nrow, ncol=length(names)))

names(x) <- names

if (!is.null (first.col)) x[,1] <- first.col

x

}

na.to.value <- function(x, cols = 1:ncol(x), value = 0)

{

# Converts NA's in selected numeric columns in a data frame to 0 or other value

# Changed 4/26/08 to give cols a default value of all columns

for(i in cols)

if(i <= ncol(x) && is.numeric(x[, i]))

x[, i] <- ifelse(is.na(x[, i]), value, x[, i])

x

}

readf <- function(x, header = T, sep="\t", row.names = NULL,...)

read.table("clipboard", header = header,sep=sep, row.names = row.names, ...)

skip <- function(v, testvalue)

{

# Returns T if v is divisible by testvalue, otherwise F

floor((v)/testvalue) \* testvalue == v

}

clip <- function(...)

scan("clipboard", blank.lines.skip=F, ...) # Scan anything from clipboard; for character include: what=""

# Note - added blank.lines.skip so a column of data matches other columns

addcol <- function (x, names, fill=NA)

# Obsolete

#

{

cat("addcol not needed: just enter x[names] <- NA")

}

addrow <- function (x, nrow=1)

# Adds one or more blank rows at the end of a data frame

# x <- data.frame(a=1:3, b=LETTERS[1:3], c=as.Date(paste0("2018-04-", 1:3)))

# addrow(x, 3)

#

{

if (!is.data.frame(x)) stop ("x must be a data frame, you fool!")

x1 <- x[1:nrow,]

for (i in 1:ncol(x)) {

if (class(x[,i]) == "character" ) x1[,i] <- rep ("", nrow(x1))

else

x1[,i] <- rep (NA, nrow(x1))

}

rbind(x,x1)

}

cftocm <- function (v, to.cm=T)

{

if (to.cm) v / 35.314667

else v \* 35.314667

}

dselect <- function (...)

{

require(dplyr)

dplyr::select(...) # masked by MASS

}

dfilter <- function(...)

{

require(dplyr)

dplyr::filter(...) # use in place of Hadley's filter function which can be masked by stats... what a stupid idea

}

dftable <- function(x, nameLabel="label", nameCount="count")

{

# Makes a table for a vector and converts it into a data frame with the labels

# as one column and the counts as the other.

# Modified 8/22/2021 to allow column names to be changed

if(!is.null(dim(x))) stop("Input must be a vector")

y <- table(x)

y1 <- names(y)

if(is.numeric(x))

y1 <- as.numeric(y1)

yout <- data.frame(label = y1, count = as.vector(y))

names(yout) <- c(nameLabel, nameCount)

yout

}

evens <- function(x)

{

# Gives indices of even values in x or if x is single valued, from 2 to x

# Replaces function in library(progexam)

if (length(x)==1) return(seq(2,x,by=2))

indices <- seq(along = x)

indices[x %% 2 == 0]

}

odds <- function(x)

{

# Gives indices of odd values in x or if x is single valued, from 1 to x

# Replaces function in library(progexam);

# see help file for evens

if (length(x)==1) return(seq(1,x,by=2))

indices <- seq(along = x)

indices [x %% 2 == 1]

}

LongLatToUTM<-function(x,y,zone=10){

# Converts longitude and latitude to x and y in UTM

# x is longitude, y is latitude

# http://stackoverflow.com/questions/18639967/converting-latitude-and-longitude-points-to-utm

# Zone 10 is western CA, OR, WA west of 120 w long

require(rgdal)

xy <- data.frame(ID = 1:length(x), X = x, Y = y)

coordinates(xy) <- c("X", "Y")

proj4string(xy) <- CRS("+proj=longlat +datum=WGS84") ## for example

res <- spTransform(xy, CRS(paste("+proj=utm +zone=",zone," ellps=WGS84",sep='')))

return(as.data.frame(res))

}

UTMtoLongLat <- function(utmx, utmy, zone=10) {

# Converts UTM x and y to longitude and latitude

# see LongLatToUTM

require(rgdal)

utm\_df <- data.frame(utmx, utmy)

CRSstring <- paste0("+proj=utm +zone=", zone)

utmcoor <- sp::SpatialPoints(utm\_df, proj4string = sp::CRS(CRSstring))

longlatcoor <- sp::spTransform(utmcoor, sp::CRS("+init=epsg:4326"))

junk <- tibble::as\_data\_frame(longlatcoor)

names(junk) <- c("lon","lat")

as.data.frame(junk)

}

# longlat <- wimsPoly[1:3,1:2] # Any set of long/lats

# utm\_df <- LongLatToUTM(x=longlat[,1], y=longlat[,2])[,2:3]

# UTMtoLongLat(utm\_df[,1], utm\_df[,2])

# data.frame(UTMtoLongLat(utm\_df[,1], utm\_df[,2]), wimsPoly)

nMissing <- function (v) sum(is.na(v))

numstring <- function(num, digits = 0)

{

# This function converts a number to a string using leading zeros to keep the

# sort order correct, and trailing zeros to keep the width of the strings constant

# digits is the number of decimal places to keep.

# Values are rounded to digits decimal places before conversion

# Modified 3/16/03 to allow values of zero or less

# Modified 3/21/04 to allow for missing values

if(!is.numeric(num)) stop("Value must be numeric")

indNA <- is.na(num)

if (all(indNA)) stop("All NA values")

num[indNA] <- num[!indNA][1] # Substitute alternative values for NAs

sgn <- sign(num)

num <- abs(num)

nx <- floor(log10(max(na.omit(num)))) + 1

nx[nx < 1] <- 1

# Number of digits

mn <- min(floor(na.omit(num)))

if(mn == 0)

nm <- 1

else nm <- floor(log10(min(na.omit(num)))) + 1

# Minimum number of digits

lead <- paste(rep("0", nx - nm), collapse = "")

y <- round(num, digits)

cnch <- nchar(as.character(floor(y)))

yc <- as.character(y)

yi <- chfind(yc, "e-")

if(any(yi > 0)) {

yp <- as.numeric(substring(yc, yi + 2)) - 1

yp[yi == 0] <- 1

yt <- ifelse(yi == 0, y, y \* 10^(yp))

yc <- as.character(yt)

yc <- ifelse(yi == 0, yc, paste(substring("0.0000000000000000",

1, yp + 2), substring(yc, 3), sep = ""))

}

x <- paste(lead, yc, sep = "")

x <- substring(x, cnch - min(cnch) + 1)

if(digits > 0) {

x <- paste(x, ifelse(y %% 1 == 0, ".", ""), sep = "")

x <- paste(x, paste(rep("0", digits), collapse = ""), sep = ""

)

x <- substring(x, 1, nx + 1 + digits)

}

if(any(na.omit(sgn) < 0))

x <- paste(ifelse(sgn < 0, "-", " "), x, sep = "")

x[indNA] <- "NA"

x

}

listCopy <- function(x, limit=TRUE)

{

# this is for copying a list to Excel.

# It is manual but not as bad as saving CSV files

# First set up a spreadsheet with a bunch of worksheets

# Then run this with the list as input

# # It is NOT recursive except for data frames within a list

# x is a list OR data frame

# Revision 17 Feb 2020 allow user to select whether to limit the sizes of data frames being copied.

# I don't know what the real limit is.

cat(class(x), "\n")

if (!is.list(x)) stop ("For lists or data frames only\n")

if (is.data.frame(x))

{

size <- object.size(x)

if (limit & size > 32000)

{

nr <- nrow(x)

nrNew <- floor(nr \* 16000/size)

x <- rbind(addrow(head(x, nrNew),1), tail(x, nrNew))

cat("")

}

write.table(x, file=paste0("clipboard-", 2^19), sep="\t", quote=F, row.names=F)

}

else

{

n <- length(x)

cat(n, names(x), "\nCopy names to spreadsheet tabs, then Enter to get first page of data: ", sep=" ")

scan(n=1)

for (i in 1:n)

{cat(names(x)[i], "\n")

if (is.data.frame(x[[i]]))

{

listCopy(x[[i]], limit=limit)

scan(n=1)}

else

{cat("Not a data frame, skipped\n")}

}}

}

listCSV <- function(x, path="C:/temp/", filename="ListCSV" , row.names = F)

{

# this is for saving a list as a spreadsheet. listCopy above is easier

# First set up a spreadsheet with a bunch of worksheets

# Then run this with the list as input

# x is a list

if (!is.list(x)) stop ("For lists or data frames only\n")

if (is.data.frame(x))

write.table(x, file=paste0(path, filename, ".csv"), sep=",", quote=T)

else

{

n <- length(x)

nmx <- names(x)

cat(n, nmx, "\n", sep=" ")

for (i in 1:n)

{ cat(nmx[i], "\n")

write.csv(x[[i]], file=paste0(path, filename, nmx[i], ".csv"),

sep=",", quote=T , row.names = row.names)

}}

}

listStruc <- function (fileName, xa=NULL, indent=0)

{

# Recursive: Shows the structure of a list and its components

# Each component name is shown along with type and length or dimensions

# Layers in the hierarchy are progressively indented

if (!is.character(fileName))

fileName <- deparse(substitute(fileName))

fileName <- fileName[1]

if (is.null(xa)) xa <- get(fileName)

if (!is.list(xa)) stop ("Input must be a list")

n <- length(xa)

if (indent==0)

{

cat(fileName,"\t\t\t\tList", n, "\n" )

indent <- 1

}

for (i in 1:n)

{

nmi <- names(xa)[i]

xi <- xa[[i]]

if (is.list(xi) & !(is.data.frame(xi)))

{

cat(rep(" ", indent), nmi,rep("\t", 5-indent), "List", n, "\n" )

listStruc(nmi, xi, indent=indent + 1)

}

else

{

classi <- class(xi)

if (is.null(classi)) classi <- ""

dimi <- dim(xi)

if (is.null(dimi)) dimi <- length(xi)

cat(rep(" ", indent), nmi,rep("\t", 5-indent), classi, dimi,"\n" )

}

}

}

xlsColumns <- function (v)

# converts letter pairs to numeric columns and back fore Excel spreadsheets

{

if (all(is.character(v) ) )

{

if(max(nchar(v)) <= 2)

{

v1 <- match(right(v,-1), LETTERS, nomatch=0)

v2 <- match(right(v,1), LETTERS)

as.integer(v1\*26 + v2 )

}

else

{

stop(paste0("Vector ", head(v), " \nhas too many characters\n"))

}

}

else if (all(round(v)==v) & all(v <= 256) )

{

paste0(c("",LETTERS)[ 1+ floor((v-1)/26)], LETTERS[1 + (v-1) %% 26])

}

else

stop(paste0("Vector ", head(v), " does not match criteria for xlsColumns"))

}

tidy <- function(ask = T, maxSize=1000, long = T)

{

# This function is based on cleanup and is used to eliminate other unneeded files

# Runs cleanup first

# This function produces a warning about row names which I ignore

cleanup ()

v <- objects(name=".GlobalEnv", pattern = "\*")

keep <- unlist(sapply(v, function(vi) first(class(get(vi))!="function")))

v <- v[keep]

if (length(v) > 0)

{

lengthv <- logical(length = length(v))

for(i in 1:length(v))

lengthv[i] <- object.size(get(v[i]))

vv <- wsort(data.frame(lengthv, v), 1)

vv <- data.frame(vv, 1:length(v))

row.names(vv) <- 1:length(v)

vv <- wsort(vv, -3)

if(ask)

{

print(vv)

yn <- readline("List numbers of files to erase separated by spaces, commas, or dashes: ")

inp <- NULL

yn1 <- paste(" ", yn, " ", sep="")

blanks <- sort(c(chfind(yn1, " ", pick="a")[[1]], chfind(yn1, ",", pick="a")[[1]]))

st <- blanks[blanks < nchar(yn1)] +1

end <- blanks[-1] -1

for (j in 1:length(st))

{

yni <- substring(yn1, st[j], end[j])

if (chfind(yni,"-")>0 | chfind(yni,":")>0)

{

dash <- chfind(yni,"-")

colon <- chfind(yni,":")

dc <- dash + colon

n1 <- as.numeric(substring(yni, 1, dc-1))

n2 <- as.numeric(substring(yni,dc+1))

cat(n1, n2, "\n")

yni <- n1:n2

}

else

{

yni <- as.numeric(yni)

}

inp <- c(inp, yni)

}

inp <- unique(inp)

inp <- pmax(1, pmin(length(v), inp))

v <- vv[match(vv[,3], inp, nomatch=0)>0, 2]

yn <- readline(paste(c("Delete the following files: ", v, "\n?\n "), collapse=" "))

}

else{ yn <- "Y" }

if((yn == "Y" | yn == "y"))

{

cat("\nErasing\n")

rm(list = v, pos=1)

cat("\n\n")

}

else

{cat("\nNothing erased \n")}

}

else

{cat("\nNothing erased \n")}

}

#-------------------

# Functions to work with dates

fmonth.abb <- function(mos) # Converts integer month to 3-letter abbreviation as a factor

{

allMos <- sort(unique(floor(mos)))

factor(month.abb[mos], levels=month.abb[allMos])

}

fmonth.name <- function(mos) # Converts integer month to full name as a factor

{

allMos <- sort(unique(floor(mos)))

factor(month.name[mos], levels=month.name[allMos])

}

wimDate <- function (dateChar, informat=c("slashes", "yyyymmdd") )

{

# Add formats as I encounter them,

# slashes is Excel format - either 2 or 4-digit year

# yyyymmdd is CIMIS and CDEC

fmi <- tolower(substring(informat[1], 1,1))

if (fmi=="s")

{

dt <- sub("/", "S",dateChar)

ny <- as.numeric(nchar(dt)-regexpr("/", dt))

fmt <- paste0("%m/%d/%", ifelse(ny==2,"y","Y"))

}

else if (fmi=="y")

fmt <- "%Y%m%d"

else stop("Wim: No such format")

as.Date(dateChar, format=fmt)

}

dateday <- function(dateval) # Calculates numeric day from a dates object

{

if (class (dateval) !="Date") stop("Input must be dates or timeDate")

as.integer(format(dateval, "%d"))

}

datemo <- function (dateval) # Calculates numeric month from a dates object

{

if (class (dateval) !="Date") stop("Input must be dates or timeDate")

as.integer(format(dateval, "%m"))

}

# datemo(as.Date("1/25/1970", format="%m/%d/%Y"))

dateyr <- function(dateval) # Calculates numeric year from a dates object

{

if (class (dateval) !="Date") stop("Input must be dates or timeDate")

as.integer(format(dateval, "%Y"))

}

julyear <- function(m, d=NULL, y=NULL)

{

# Function to get julian date within year, with 1 Jan = 1

# If no values given it returns today's Julian date\

# Input

# m EIther a Date object or a numeric month if d and y are also given

# m can also be a list with m, d, y as components

# Modified 6/17/2015 for R

if(class(m)[1] == "Date")

{

dt <- m

dt1 <- format(dt, format="%m/%d/%Y")

y <- as.integer(right(dt1,4))

}

else

{

if(is.list(m) & length(m) == 3)

{

y <- m[[3]]

d <- m[[2]]

m <- m[[1]]

}

if (!is.null(d) & !is.null(y))

{

dt <- as.Date(paste(as.character(y), m, d, sep="-"))

}

}

# else {stop ("Wim: Need either Date object or mdy or a list\n")}

dt0 <- as.Date(paste0("12/31/", y-1), format="%m/%d/%Y")

as.integer(julian(dt) - julian(dt0))

}

# julyear(10,25,1970)

# dt <- as.Date("10/25/1970", format="%m/%d/%Y")

# julyear(dt)

mdyear <- function(jul, year = -1)

{

# Function to convert within-year Julian dates to calendar dates.

# If a year is entered it is for that year, if not it is for the current year.

# The function returns a list containing month, day, and year.

# Modified 12/22/04 to produce all integers

require(chron)

if(is.factor(year)) year <- as.integer(as.character(year))

if(year[1] < 0) {

p <- date()

year <- as.integer(substring(p, nchar(p) - 3))

}

uf <- rep(NA, length(jul))

u <- list(month = uf, day = uf, year = uf)

for(yr in unique(year)) {

# is a vector of length 3

uy <- month.day.year(jul[year == yr], origin. = c(12, 31, yr - 1))

for(i in 1:3)

u[[i]][year == yr] <- uy[[i]]

}

u

}

yjdate <- function (julday, year=-1)

# Converts Day of year to Dates object

{

require(lubridate)

if (year[1] < 0) year <- format(today(), "%Y")

if (length(year) !=1 & length(year) != length(julday))

stop ("Length of year must be 1 or length of julday")

as.Date(julday - 1, origin = paste0(year, "-01-01"))

}

yrmo.fun <- function(y, m = NULL, d = NULL, incday = (!is.null(d) | class(y) == "Date"),

places = ifelse(incday, 4, 2))

{

# Calculates year plus month (and optionally, day) as a decimal value.

# Modified Jan 2016 to port into R

# Input is either y, a data frame or a dates or Date object

# Or y is year, m is month, and (optional) d is day.

# If y is a data frame the first column must be a Date object

# OR column names must be a substring of "year","month", and optionally "day" if incday=T

# incday is T if days are to be used in the calculation.

# incday defaults to F, or T if d is specified or y is a Date value

# Places is the number of digits to the right of the decimal point

# incday <- incday

# Because of lazy evaluation , see programmers manual p. 125

if (is.data.frame(y))

{

if (any(class(y[,1])=="Date")) yy <- y[,1]

else if (all(substring(c("year","month"), 1, nchar(names(y)[1:2]))==names(y)[1:2]))

{

yy <- y[,1]

m <- y[,2]

if (ncol(y)>2)

if (substring("day", 1, nchar(names(y)[3]))==names(y)[3]) d <- y[,3]

}

else stop("No Date in col 1, or names in cols 1 and 2 do not match c(year,month)")

}

else yy <- y

if(!is.null(class(yy)) & any(class(yy) == "Date"))

{

m <- as.numeric(format(y, "%m"))

d <- as.numeric(format(y, "%d"))

yy <- as.numeric(format(y, "%Y"))

}

x <- yy + round(m - 1)/12

if(incday)

{

leap <- (((yy %% 4 == 0) & (yy %% 100 != 0)) | (yy %% 400 == 0))

x <- x + (d - 1)/(365 + leap \* 1)

}

round(x, places)

}

#

# Functions to be used with base R plotting-------------------------

colplot <- function(x, xcol = 1, lab = T, labAtEnd = T, rot = 0, type = "l", pch = 16,

lwds = NULL, cols=NULL, ltys=NULL, xlim = NULL, ylim = NULL, alog = "", cumplot = F, ...)

{

# Plots all columns in a data frame against a chosen column as lines

# x is a data frame containing the data

# xcol is the column number selected for the x axis variable

# If xcol is 0, the row names are converted to numeric and used if they increase

# monotonically.

# If xcol is < 0 or row names do not increase monotonically,

# sequential numbers are used.

# lab is T for putting the column label on the graph,

# or a vector of labels of length =ncol(x)-1

# labAtEnd is for putting the label at the right-hand end,

# otherwise at the "best" point for each line

# This is the point that is furthest from any other point on the plot

# rot Rotate labels

# Amended 3/7/15 to port into R

# Amended 12/8/15 to have options of multiple line types, colors, and line widths

if (any(match(alog, c("x","y","xy"), nomatch=0)==0)) alog <- ""

if(xcol > 0) {

xx <- x[, xcol]

yy <- x[, - xcol]

}

else {

yy <- x

if(xcol < 0)

xx <- 1:nrow(x)

else {

if(xcol == 0)

xx <- as.numeric(row.names(x))

if(any(is.na(xx)) || any(diff(xx)) < 0)

xx <- 1:nrow(x)

}

}

if(cumplot)

yy <- data.frame(t(apply(yy, 1, cumsum)))

if(is.null(ylim))

ylim <- range(na.omit(as.vector(as.matrix(yy))))

if(is.null(xlim))

xlim <- range(xx)

if(length(lab) == ncol(yy)) {

labs <- lab

labtf <- T

}

else if(lab == T) {

labs <- names(yy)

labtf <- T

}

else labtf <- F

if(labtf & labAtEnd)

{

xlim[2] <- xlim[2] + 0.03 \* diff(xlim) \* max(nchar(labs))

}

nc <- ncol(yy)

ycols <- rev(1:nc)

if (is.null(lwds)) lwds <- rep(1, nc)

else if (length(lwds) < nc) lwds <- rep(lwds, ceiling(nc/length(lwds)))[1:nc]

if (is.null(ltys)) ltys <- rep(1, nc)

else if (length(ltys) < nc) ltys <- rep(ltys, ceiling(nc/length(ltys)))[1:nc]

if (is.null(cols)) cols <- ((1:length(ycols))-1) %% 10 + 1

plot(xx, yy[, 1], type = "n", pch = pch, ylim = ylim, xlim = xlim, log = alog, ...)

for(i in ycols)

lines(xx, yy[, i], type = type, pch = pch, lwd = lwds[i], col = cols[i], lty=ltys[i])

yL <- rep(0, ncol(yy))

if(labtf)

{

if(labAtEnd)

{

xL <- rep(max(xx), ncol(yy))

for(i in 1:ncol(yy))

text(xL[i], tail(na.omit(yy[, i]),1), labs[i], col = cols[i], adj = 0)

}

else

{

Lx <- chfind("x", log) > 0

Ly <- chfind("y", log) > 0

vv <- par("usr")

w <- na.omit(data.frame(index = rep((1:ncol(yy)), each = length(xx)),

xL = rep(xx, ncol(yy)), yL = unlist(yy)))

w$xt <- w$xL

if(Lx) w$xt <- log10(w$xt)

w$yt <- w$yL

if(Ly) w$yt <- log10(w$yt)

w$xusr <- (10 \* par("fin")[1] \* (w[, 2] - vv[1]))/ (vv[2] - vv[1])

w$yusr <- (par("fin")[2] \* (w[, 3] - vv[3]))/(vv[ 4] - vv[3])

wd <- dist(w[, 4:5])

ind1 <- sort(dist(cbind(1:nrow(w), rep(0, nrow(w)))))

ind2 <- rowindex(as.data.frame(ind1), global = F) + ind1

ind <- data.frame(ind1, ind2, wd)

ind$include <- !(w$index[ind1] == w$index[ind2])

ind$wd[!ind$include] <- max(ind$wd + 1)

w$mindist <- pmin(c(tapply(ind$wd, ind$ind1, min), 0),

c(0, tapply(ind$wd, ind$ind2, min))) + runif(nrow(w), 0, 1e-005)

w <- merge(w, Bquick(w[, c("index", "mindist")], 1, max), 1, suffixes = c("", ".max"))

w <- w[w$mindist == w$mindist.max, ]

ww <- wsort(w[, c(2, 3, 1)], 1:2)

ww$line <- rowindex(as.data.frame(ww[, 1]), global = F ) - 1

ww <- wsort(ww, 3)

for(i in 1:nrow(ww))

{

points(ww$xL[i], ww$yL[i], pch = 16, cex = 1.2, col = cols[i])

mtext(labs[i], at = ww$xL[i], line = ww$line[i],

col = cols[i], crt = rot, adj = 0.5)

}

}

}

}

plind <- function(x, y, ind, xlab = NULL, ylab = NULL, textColor=1, textSize=1, textRotate=0, ...)

{

# Plots text in place of symbols (... can be any other

# graphics commands used in plot)

# Changed 1/30/05 to allow for data frame input with headers

# Changed 2.27.11 to allow some control over text

# x, y, and ind are vectors (ind should be character but will be converted)

# OR x can be a data frame with 3 columns interpreted as x, y, and ind

nm <- deparse(substitute(x))

if(is.data.frame(x)) {

x0 <- x[, 1]

y <- x[, 2]

ind <- x[, 3]

if(is.null(xlab))

xlab <- paste(nm, names(x)[1], sep = "$")

if(is.null(ylab))

ylab <- paste(nm, names(x)[2], sep = "$")

}

else {

x0 <- x

if(is.null(xlab))

xlab <- ""

if(is.null(ylab))

ylab <- ""

}

if(!is.character(ind))

ind <- as.character(ind)

plot(x0, y, type = "n", xlab = xlab, ylab = ylab, ...)

if (length(c(textColor, textSize, textRotate)) ==3)

text(x0, y, ind, col=textColor, cex=textSize, crt=textRotate)

else

{

df <- data.frame(textColor, textSize, textRotate)

if (nrow(na.omit(df)) != length(x))

{

cat("Warning: text attributes misaligned, set to first value\n")

text(x0, y, ind, col=textColor[1], cex=textSize[1], crt=textRotate[1])

}

else

for (i in 1:length(x))

text(x0[i], y[i], ind[i], col=df[i,1], cex=df[i,2], srt=df[i,3])

}

}

# plind(1:10,rnorm(10), letters[1:10])

mtexts <- function(xlab, ylab, main, sub = "", cex = c(1.1, 1.1, 1.3, 1), lines = c(

1,1,1,0), cols = rep(1, 4))

{

mtext(xlab, side = 1, outer = T, cex = cex[1], line = lines[1], col =

cols[1])

mtext(ylab, side = 2, outer = T, cex = cex[2], line = lines[2], col =

cols[2])

mtext(main, side = 3, outer = T, cex = cex[3], line = lines[3], col =

cols[3])

mtext(sub, side = 3, outer = T, cex = cex[4], line = lines[4], col =

cols[4])

}

nullplot <- function (x=0:1,y=0:1)

# Takes up a space on the plotting page without drawing anything

# x and y can be entered to set scaling for text

plot(x,y, type="n", xlab="", ylab="", axes=F)

Par <- function(nfig, way = "x", mar=NULL, omi = NULL, values=F)

{

# Sets up plot window for 1-4 plots: 2 & 3 are arranged

# horizontally if way is "h" or stacked vertically if it is "v", 4 is 2X2

#

par( tcl=-0.1, mgp=c(1.3,0.2,0))

if(nfig > 4)

stop("Wim: Can't do more than 4 plots this way")

if(!(way == "h" | way == "v" | way == "x"))

stop("Wim: 'way' has to be 'h' or 'v' or left blank")

if(is.null(omi))

{

if (nfig==1) {

omi <- c(0,0,0,0)

} else {

omi <- c(0,0,0,0) # c(.5,.5,.5,.5)}

} }

if(is.null(mar))

{

if (nfig==1) {

mar <- c(3,3,2,2)

} else {

if (nfig==4) {

mar <- c(2.75,2.75,1.5,0.5)

} else {

if (way=="v"){ mar <- c(2.75, 7.5, 0.75, 7.5)

} else { mar <- c(4,3,2,1)}

}}}

if (way == "x") {

if (nfig==1) { par(mfrow = c(1,1), omi = omi, mar = mar)

}else{

par(mfrow = c(2, 2), omi = omi, mar = mar)

}

}else{

if (way=="h"){

par(mfrow = c(1, nfig), omi = omi, mar = mar)

}else {par(mfrow = c(nfig, 1), omi = omi, mar = mar)

}}

vt <- c("omi:",omi, "mar",mar)

if (values)

cat(vt, "\n")

}

wbox <- function (x, names=NULL, xlab = "", mincex=1, ...)

{

# Function to create a boxplot in which the names are rotated 90 degrees for easy reading.

# Text is adjusted to fit, but if the number of boxes is too much it might not be readable.

# INputs:

# x is the output of split: a list with names

# names are the names of the splits, and if NULL they are taken from the splits

# mincex is the minimum text size for the labels

# Modified 2/2011 to make the x axis labels smaller and allow selection of size

if (is.null(names)) names <- names(x)

boxplot(x, names=rep("", length(names)), xlab="", ...)

mtext (names, at=1:length(x), side=1, las=2, line=1, adj=1, cex=pmax(mincex,1/length(names)) )

if (xlab != "")

{

nc <- max(nchar(names)) \* pmin(1,15/length(names)) \* 0.8

mtext (xlab, side=1, outer=F, line = nc)

}

}

# Functions to be used with ggplot-----------------------

labelFunction <- function () # Returns a list containing the labels below for text using Unicode

{

dm1 <- "d\U207B\U00B9" # d^-1

Lm1 <- "L\U207B\U00B9" # L^-1

fm1 <- "\U2640\U207B\U00B9" # (Female sign)^-1

mm3 <- "m\U207B\U00B3" # m^-3

allLabels <- list(dm1 = dm1, L1 = Lm1, fm1 = fm1, mm3 = mm3,

Chlw = paste("Chlorophyll, \U03BCg",Lm1),

Chlw.2 = paste("Chlorophyll\n\U03BCg",Lm1),

Chl5 = paste("Chlorophyll > 5 \U03BCm, \U03BCg" ,Lm1),

Chl5.2 = paste("Chlorophyll\n > 5 \U03BCm, \U03BCg",Lm1),

temp = "Temperature,\u00b0C",

sal = "Salinity",

turb = "Turbidity, NTU",

growth = paste("Growth Rate, ", dm1),

growth22 = paste("Growth Rate at 22 \u00b0C, ", dm1),

epr = paste("Egg Production, Eggs ", fm1, dm1),

sepr = paste("Specific Egg Production Rate, ", dm1),

abunm3 = paste("Abundance,", mm3),

abunm3p10= paste("Abundance + 10,", mm3),

abunm3p1 = paste("Abundance + 1,", mm3),

abunL = paste("Abundance,", Lm1) )

allLabels

}

ptToMM <- function (points) points \* 0.3528 # Convert text size in pts to mm for geom\_text & annotate

resize <- function(height=3, width=4)

{

# Clunky function to resize the graphics window.

# To use, set the width where you want it, then run this

# Adjust the height and hit Enter until it is close enough

# If the height and width match then this is ready to go.

# NOTE in dev.size height is SECOND

v <- numeric(0)

while(length(v)==0)

{

sizes <- dev.size()

names(sizes) <- c("width","height")

width <- width # Inches

height <- height

aspect <- width/height

cat("Adjust height to ", sizes[1]/aspect,"\nCurrent size: W =",sizes[1], ", H = ", sizes[2], "\n")

cat("Adjust and hit Enter (Esc to stop): ")

v <- scan(n=1)

}

}

addLetters <- function(x, y=NULL, facetVar1=NULL, facetVar2=rep(0, length(facetVar1)),label=LETTERS,

ypos=0.8, freeX=FALSE, freeY=FALSE)

# This is to make a data frame to place letters or other symbols in each panel of a graph.

# This version puts the labels in the upper left corner, with ypos being the relative position

# x can be a data frame in which the columns are x, y, and the facetting variable

# or x can be the x vector, y the y, and facetVar the facetting variable

# the label will be placed at min(x), and ypos of the way up the y axis

{

if (is.null(y))

{

if (!is.data.frame(x)) stop("You gotta give a data frame or three vectors")

if (ncol(x)==3) x <- data.frame(x, dummyFacet=rep(0, nrow(x)))

}

else

{

x <- data.frame(x, y, facetVar1, facetVar2)

}

names(x)[1:2] <- c("x","y")

if (is.factor(x$x)) x$x <- as.numeric(x$x)

if (is.factor(x$y)) x$y <- as.numeric(x$y)

xByFacet <- bystats(x[,c(3,4,1,2)],2,rng=T, na.rm=T) # [,c(1,2,5:8)]

xByFacet$labelX <- xByFacet$x.min

xByFacet$labelY <- ypos\*xByFacet$y.max + (1-ypos)\*xByFacet$y.min

if(!freeX) xByFacet$labelX <- rep(min(xByFacet$labelX ), nrow(xByFacet))

if(!freeY) xByFacet$labelY <- rep(max(xByFacet$labelY ), nrow(xByFacet))

xByFacet$label <- label[1:nrow(xByFacet)]

nmx <- names(x)[3:4]

nmx <- nmx[nmx != "dummyFacet"]

xByFacet[,c("labelX", "labelY", nmx, "label")]

}

plotRange <- function(y, x=1:length(y), nrows=3, xtitle="X", ytitle="Y", findGap=F)

{

# Produces a ggplot of a long time series broken into rows

# and from a selected segment of the total range.

# INputs:

# y is the y variable

# x is the x variable; if not entered this is a series along y

# nrows: number of rows of graphs

# xtitle: Optional title for the x variable

# ytitle: Optional title for the y variable

# findGap: TRUE to identify beginning and end of gaps with a red point.

require(ggplot2)

xy <- data.frame(x, y)

yn <- ""

cat("Range of x values ", range(x), "\n")

while(yn != "Q")

{

yn <- readline("Select A, Q, or range of x values: ")

yn <- toupper(yn)

if (substring(yn,1,1)=="Q")

xrange <- NULL

else if (substring(yn,1,1)=="A")

xrange <- range(x)

else

{

vcd <- max(regexpr("-", yn), regexpr(":", yn))

if (vcd > 1)

xrange <- as.numeric(c(substring(yn, 1, vcd-1), substring(yn, vcd+1)))

xrange <- c(max(xrange[1], min(x)), min(xrange[2], max(x)))

}

if (!is.null(xrange))

{

irange <- which(x>=xrange[1] & x <= xrange[2])

w <- data.frame(x=x[irange], y=y[irange] )

w$index <- rep(1:nrows, each=ceiling(nrow(w)/nrows))[1:nrow(w)]

pw <- ggplot(w, aes(x, y) )+

geom\_line(color="blue") +

facet\_wrap(~index, nrow=nrows, scales="free\_x") +

xlab(xtitle) + ylab(ytitle) +

wimGraph()

# print(pw)

if (findGap)

{

v <- is.na(w$y)

vv <- cbind(v, c(F, v[-length(v)]), c(v[-1], F))

flag <- ! (vv[,1]) & xor(vv[,2], vv[,3])

flag[c(1, nrow(w))] <- T

wgap <- w[flag,]

if (nrow(wgap)>0)

pw <- pw + geom\_point(data=wgap, aes(x=x, y=y), color="red", pch=16)

}

print(pw)

}

}

}

pointmm <- function (points)

{ # Calculates size of text in mm from point size

# For ggplot

points \* 0.352778

}

setrange <- function (x,y, logs =F)

# Sets the range for a square plot so both x and y have the same relative scaling (i.e., diff(range))

# Inputs

# x The x value in the plot

# y The y value

# logs TRUE if values are to be log-transformed (both or neither must be transformed)

# Value

# list containing xL and yL, the un-transformed values for xlim and ylim in the plot

# IF logs is TRUE the scaling will be the same for log-transformed data.

{

w <- cbind(range(x), range(y))

if (any(w[,1] <= 0) | any(w[,1] <= 0) ) logs <- FALSE

if (logs) w <- log(w)

v <- apply(w, 2, diff)

is <- which(v==max(v))

isnt <- 3 - is

w[,isnt] <- w[,isnt] + (v[is]-v[isnt])\*c(-.5,.5)

if (logs) w <- exp(w)

data.frame(xL=w[,1], yL=w[,2])

}

scientific\_10 <- function(x)

{ # For ggplot to produce scientific notation

v <- require(scales,quietly=T)

if (v)

parse(text=gsub("1e+", "10^", scientific\_format()(x)))

else

cat("Package scales not found\n")

}

wimExpr <- function (code="chl")

{

codes <- c("abun","chl","epr","flow","growth","podflux","temp")

code1 <- substring(codes,1,1)

icode <- match(tolower(substring(code,1,1)), code1)

exprs <- list(

a = expression("Abundance, m"^-3 ),

c = expression(paste("Chlorophyll, ", mu,"g",~L^-1)),

e = expression("Egg Production, Eggs "~Fem^-1~d^-1),

f = expression("Flow, "~m^3~s^-1),

g = expression("Growth Rate, "~d^-1),

p = expression(paste("Copepod Flux, ", 10^9~d^-1)),

t = expression(paste("Temperature, ", degree, "C")))

exprs[[icode]]

}

wimPalettes <- function (pal="point", ncolors=12,

red=seq(0,1, length=ncolors),

green=0, blue=seq(1,0,length=ncolors))

{

# Selections are "colorblind","gradient","point","line", "slide" as of 10/17/2016

# Added "month" for monthly colors from linear interpolation

# of blue,green,red,orange, and back to blue (January April July)

# Gradient uses rgb for a selected number of colors ncolors.

# Gradient also uses the red, green, and blue values above

# Default gradient is blue to red through purple.

require(grDevices)

nc <- ceiling(ncolors/7)

palC <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")

palG <- rgb(red=red, green=green, blue=blue)

palP <- rep(c("black","red","green4","blue","magenta","cyan","orange","gray"),nc)

palL <- rep(c("red","black","green4","magenta","blue","gray","cyan","orange"),nc)

palM <- c("#0000ff", "#0055aa", "#00aa55", "#00ff00", "#55aa00", "#aa5500",

"#ff0000", "#ff4400", "#ff8800", "#ffcc00", "#aa8855", "#5544aa") # Blue..Green..Red..Orange..(blue)

palO <- c("seagreen4", "red","darkgoldenrod4", "royalblue4","coral","gray50", "lightblue1","turquoise4","darkblue")

palS <- rep(c("yellow","green","cyan","magenta","gray90","pink","orange"),nc)

switch(tolower(substring(pal,1,1)), "c"=palC, "g"=palG, "p"=palP, "l"=palL,"m"=palM, "o"=palO, "s"=palS, palP)

}

wimGraph <- function(background="white", panelBackground=NA,

panelBorder = "black",

textCol="black", titleCol="black",

axisCol="black",gridCol="gray90",

stripCol="black", stripFill="white",

title.rel=1.2, axis.title.rel=1.6, axis.text.rel=1.0,

strip.text.rel=0.7, textSize=10)

{

theme(panel.border = element\_rect(colour = panelBorder, fill=NA, size=0.5),

plot.background = element\_rect(fill = background, linetype=0),

panel.background = element\_rect(fill = panelBackground, linetype=1, color=axisCol),

legend.key = element\_rect(fill = NA),

legend.background=element\_rect(fill=NA),

text= element\_text(color=textCol, size=textSize),

panel.grid.major = element\_line(color=gridCol, size=0.2),

panel.grid.minor = element\_blank(),

strip.background = element\_rect(color=stripCol, fill=stripFill),

strip.text = element\_text(color=textCol, size=rel(strip.text.rel)),

plot.title = element\_text(size=rel(title.rel), color=titleCol),

axis.title = element\_text(size=rel(axis.title.rel), color=titleCol, vjust=0),

axis.text = element\_text(size=rel(axis.text.rel),color=axisCol),

axis.line.x = element\_line(color=axisCol), # Changed these two

axis.line.y = element\_line(color=axisCol),

axis.ticks=element\_line(color=axisCol))

}

wimSlide <- function(background="transparent", panelBackground=NA,

panelBorder = "black",

textCol="cyan",axisCol="yellow",titleCol="white",gridCol="transparent",

stripCol="cyan",stripFill="transparent",

title.rel=1.2, axis.title.rel=1.6, axis.text.rel=1.0,

strip.text.rel=1, textSize=10)

{ wimGraph ( background=background, panelBackground=panelBackground,

panelBorder = panelBorder,

textCol=textCol, titleCol=titleCol,

axisCol=axisCol,gridCol=gridCol,

stripCol=stripCol,stripFill=stripFill,

title.rel=title.rel, axis.title.rel=axis.title.rel,

axis.text.rel=axis.text.rel,

strip.text.rel=strip.text.rel, textSize=textSize)

}

wsave <- function (file="Rplot.png", plot = last\_plot(), slide=F, overwrite=F,

path= "C:/1Current/R-files/Graphs/", ...)

{

# Saves the latest plot (or another selected plot) after checking existence

if (!overwrite & file.exists (paste0(path, file)))

{

cat(file, "already exists in directory\n", path, "\n")

yn <- readline("Overwrite? ")

if (tolower(yn)!= "y")

{cat("Not overwritten\n")

overwrite <- FALSE

}

else overwrite=TRUE

}

else overwrite=TRUE

if (overwrite)

{

if (slide)

ggsave(file=file, plot = plot, path=path, bg="transparent" ,...)

else

ggsave(file=file, plot = plot, path=path ,...)

}

}

confintLabel <- function(z, namex="x", places=2, include.int=T)

{

# Makes a label for a linear model - for now only for one x variable

# Inputs

# z A linear model

# namex Name for the x variable

# places Decimal places for the output (will drop trailing zeros)

# include.int TRUE to include the intercept

if (all(match(class(z), "lm", nomatch=0)==0) )

stop ("Input must be a linear model")

zc <- as.numeric(round(coef(z), places))

zi <- as.numeric(round(diff(confint(z)[-1,])/2, places))

lab <- paste(zc[2], namex, " \U00B1", zi )

if (include.int)

lab <- paste0(zc[1], " + ", lab)

lab

}

# Statistical functions---------------------

getJags <- function (out)

{

# 2020=09-09 Takes list produced by Jags

# and produces data frame of parameter estimates

# First column is parameter name (not including subscripts)

# Other columns include stats and Rhat and n.eff (convergence indicators)

y <- data.frame(out$summary)

y <- y[,c("mean","sd","X2.5.","X50.", "X97.5.", "Rhat","n.eff")]

names (y)[3:5] <- c("CL2.5","med","CL97.5")

vars <- row.names(y)

iv <- regexpr("\\[", vars)

params <- ifelse(iv<=0, vars, substring(vars, 1, iv-1))

tempstring <- ifelse(iv==0, "0", substring(vars,iv+1, nchar(vars)-1 ))

ind <- as.numeric(tempstring)

y <- data.frame(par=params,ind=ind, y)

y

}

getBugs <- function(fname=NULL, page=1 )

{

# Modified 2/1/2020 to allow input directly from an excel file.

# This is necessary for using this inside of a function, loop, or Markdown doc

# Modified 7/7/2020 to include all statistics in the input, e.g., median

# and all confidence levels instead of only one.

if (is.null(fname))

{

y <- read.table("clipboard", header = T, sep = "\t") # Copy from spreadsheet

}

else

{

y <- as.data.frame(read\_excel(fname, sheet=page, .name\_repair="minimal"))

}

delete <- match(c("MC.error","MC error","start","sample"), names(y))

delete <- na.omit(delete)

y <- y[,-delete]

nmx <- suppressWarnings(round(as.numeric(names(y)) \* 100,1))

nmy <- ifelse(is.na(nmx), names(y), paste0("CL", nmx))

names(y) <- nmy

v <- as.numeric(regexpr("\\[", y$node))

nm <- ifelse(v == 0, y$node, substring(y$node, 1, v - 1))

nmn <- substring(y$node, v + 1, nchar(y$node) - 1)

inds <- ifelse(v == 0, "", nmn )

j <- 0

indices <- matrix(nrow = nrow(y), ncol = 5)

while(any(nchar(inds) > 0)) {

j <- j + 1

vc <- regexpr(",", inds) - 1

vc <- ifelse(vc > 0, vc, nchar(inds))

indices[, j] <- as.numeric(substring(inds, 1, vc))

inds <- substring(inds, vc + 2)

}

indices <- data.frame(indices[, 1:j])

names(indices) <- paste("index", 1:j, sep = ".")

y <- data.frame(parameter = nm, indices, y)

nms <- unique(nm)

out <- list()

for(i in 1:length(nms)) {

out[[i]] <- y[nm == nms[i], ]

names(out)[[i]] <- nms[i]

}

out

}

weighted.mean <- function(y, w = NULL)

if(missing(w)) mean(y) else sum(w \* y)/sum(w)

weighted.sd <- function(x, w = NULL)

{

# Weighted standard deviation by brute force.

# No checking for missing data (same as weighted.mean)

# Output is both the mean and the sd

if(missing(w)) sqrt(var(x)) else {

y <- weighted.mean(x, w)

v <- sum(w \* (x - y)^2.)/(sum(w) - 1.)

out <- c(y, sqrt(v))

names(out) <- c("wtmn", "wtsd")

out

}

}

# vt <- 1:10

# weighted.sd(vt, w=rep(1, 10))

# mean(vt)

# w <- rep(1:2, each=5)

# weighted.sd(vt,w)

# weighted.mean(vt,w)

bugR <- function(uu)

{

nuu <- length(uu)

ustring <- lapply(uu, function (v) paste(as.vector(v), collapse=","))

vo <- vector(length=nuu)

vmat <- unlist(lapply(uu, function(v)!is.null(dim(v))))

vvec <- !vmat & unlist(lapply(uu,length))>1

vone <- ! vmat & !vvec

vn <- which(vmat )

if(length(vn) > 0)

for(i in vn) {

cat(i, names(uu)[[i]], names(uu[[i]]), "\n")

ui <- uu[[i]]

udim <- paste(dim(t(ui)), collapse=",")

dimnames(ui) <- NULL

vo[[i]] <- paste(names(uu)[i],

"=structure(.Data=c(", ustring[[i]],"),.Dim=c(",

udim, "))", sep="")

}

vo[vone] <- paste(names(uu)[vone], uu[vone], sep="=")

vo[vvec] <- paste(names(uu)[vvec], "=c(", ustring[vvec], ")", sep="")

out <- paste("list(", paste(vo, collapse="\n,"), ")", sep="")

cat(out, "\n", file = "clipboard")

}

bystats <- function(x, nind, mn = T, med = F, N = F, rng = F, sd = F, ci = F,

conflim = 0.95, pctile = F, pctiles = c(0.05, 0.95), ...)

{

# Function uses Bquick to get summary stats by index columns

# Stats must include at least mn or med

# Ported to R November 2015

# Modified 6 August 2017 to fix a glitch in na.rm

# Missing values are now removed by variable, now for all at once

if(!mn & !med) stop("Either mean or median must be included")

nc <- ncol(x)

vars <- (nind+1):nc

if(mn)

y <- Bquick(x, nind, mean, ...)

else y <- Bquick(x, nind, median,...)

if(mn & med)

y <- merge(y, Bquick(x, nind, median, ...), 1:nind, suffixes = c("", ".med"))

if(N)

y <- merge(y, Bquick(x, nind, function(v) length(v[!is.na(v)])),

1:nind, suffixes = c("", ".N"))

if(rng) {

y <- merge(y, Bquick(x, nind, min, ...), 1:nind,

all.x = T, suffixes = c("", ".min"))

y <- merge(y, Bquick(x, nind, max, ...), 1:nind,

all.x = T, suffixes = c("", ".max"))

}

if(sd) {

y1 <- Bquick(x, nind, FUN="sd", ...)

y <- merge(y, y1, 1:nind, all.x = T, suffixes = c("", "sd"))

}

if(ci) {

y1 <- Bquick(x, nind, function(v) var(v), ...)

y2 <- Bquick(x, nind, function(v) length(v[!is.na(v)]))

y3 <- y1

y3[,vars] <- -qt((1 - conflim)/2, as.matrix(y2[,vars] - 1)) \*

sqrt(y1[,vars]/y2[,vars])

y <- merge(y, y3, 1:nind, all.x = T, suffixes =

c("", paste(".C", floor(conflim \* 100), sep = "")))

}

if(pctile) {

plab <- as.character(floor(pctiles\*100))

plab <- paste0("C", ifelse(nchar(plab)==1,"0", ""), plab)

for(j in 1:length(pctiles))

y <- merge(y, Bquick(x, nind, quantile, probs = pctiles[j]),

1:nind, all.x = T, suffixes = c("", plab[j]))

}

wsort(y, 1:nind)

}

confint.rlm <- function (zr, level = 0.95)

{

if (class(zr)[1] != "rlm") stop ("Input is not from rlm")

cf <- zr$coefficients

df <- summary(zr)$df[2]

a <- (1 - level)/2

ses <- sqrt(diag(vcov(zr)))

fac <- -qt(a, df)

data.frame(mean=cf, ci=fac\*ses)

}

gmreg <- function(x, y, int=T, weights=NULL)

{

# This function calculates a geometric mean regression according to

# Ricker, in which the slope is the geometric mean of the slopes of

# y on x and x on y. Missing values are first eliminated.

# MODIFIED Sept 2010 to include weights as w

# and int =T to include intercept or F to exclude

u <- data.frame(x, y)

u <- na.omit(u)

x <- u[, 1.]

y <- u[, 2.]

if (is.null(weights)) weights <- rep(1, nrow(u))

if (int)

{

z1 <- lm(y ~ x, weights=weights)

z2 <- lm(x ~ y, weights=weights)

gms <- sqrt(z1$coef[2.]/z2$coef[2.]) \* sign(z1$coef[2.])

gmi <- mean(y) - gms \* mean(x)

gm <- c(gmi, gms)

names(gm) <- c("Intercept", "Slope")

}

else

{

z1 <- lm(y ~ -1+x, weights=weights)

z2 <- lm(x ~ -1+y, weights=weights)

gm <- sqrt(z1$coef/z2$coef) \* sign(z1$coef)

names(gm) <- "Slope"

}

return(gm)

}

godinFilt <- function(tstep=1)

{

# This function calculates a tidal filter with a time step given in hours.

# Algorithm was extracted from a FORTRAN program supplied

# by Larry Smith, USGS. Briefly the algorithm is a running weighted

# mean with the weighting factor calculated as w below, which weights

# over a span of 3\*tstep values.

# This simply calculates the filter and does nothing else.

# To use it run something like this

# stats::filter(x, filter = godinFilt(tstep=1))

# where x is a time series and tstep is the time step in hours of x.

tsd <- 24/tstep

ts1 <- tsd-1

window <- tsd\*2+ts1

v <- c(rep(0, ts1), 1:tsd, rev(1:tsd), rep(0, ts1))

w <- rep(NA,window )

for(i in 1:window)

w[i] <- sum(v[i:(i + ts1)])

w/sum(w)

}

logLikBinom = function(k, n, stats = T, dist = T, conflim = 95, scale = T, step = 0.001)

{

# Function to calculate the likelihood of a binomial distribution

# Symbols as in WIkipedia

# Inputs

# k The number of values ("successes" on each trial, e.g., heads in a coin flip)

# n The number of total trials (e.g., number of coin flips)

# stats If TRUE, named vector containing the stats of the likelihood distribution

# if FALSE, the distribution itself

# dist If TRUE and stats is FALSE, the full distribution of the likelihood

# conflim If stats=T, the confidence limit returned

# scale If stats=F and dist=T, the returned distribution is scaled to 1 at max

# step If stats=F, the step size for lambda included

# Value

# If stats is TRUE

# Mean, max-likelihood, sd, and confidence interval of lambda

# If stats is FALSE and scale is TRUE

# A matrix containing the binomial probability p and the likelihood for p

# If scale is TRUE it is scaled to a maximum of 1

# If stats is FALSE and scale is FALSE (Added 5/11/14)

# A list containing he maximum likelihood and the probability (p) at that value

if(k > n) stop("Parameter k greater than n")

if(stats)

N <- 100000

else N <- round(1/step)

mle <- k/n

ps <- seq(0, 1, length = N + 1)[ - c(1, N)]

if(k == 0)

fk <- 0

else fk <- sum(log(1:k))

fn <- sum(log(1:n))

if(k == n)

fnk <- 0

else fnk <- sum(log(1:(n - k)))

term1 <- fn - fk - fnk

ll <- term1 + k \* log(ps) + (n - k) \* log(1 - ps)

if(stats) {

pv <- (100 - conflim)/200

pv <- c(pv, 1 - pv)

v <- weight.stats(ps, exp(ll), mn = T, med = T, sd = T, pctile

= T, pctiles = pv)

v <- c(n, mle, v)

names(v)[1:2] <- c("n", "mle")

names(v)[6:7] <- paste("C", numstring(pv \* 1000), sep = "")

v

}

else if(dist) {

lik <- exp(ll)

if(scale)

lik <- lik/max(lik)

lik <- c(0, lik, 0)

ps <- c(0, ps, 1)

mat <- cbind(ps, lik)

dimnames(mat) <- list(1:nrow(mat), c("prob", "likelihood"))

mat

}

else list(NegLogLik = min( - ll), mle = mle)

}

off.diag <- function (mat, output="full" )

{

# Returns the off-diagonals of a matrix, either full or

# lower left or upper right corner ("f","l","u")

# With a distance matrix (not actually a matrix but a dist object)

# It automatically calculates the lower corner.

#

output <- tolower(substring(output,1,1))

if (is.na(match(output, c("f","l","u")))) stop("output must be f, l, u\n")

if (class(mat)=="dist") output <- "l"

mat1 <- as.matrix(mat)

dPos <- row(mat1)-col(mat1)

matOut <- split(as.matrix(mat1), dPos)

if (output == "l")

{

matOut <- matOut[as.numeric(names(matOut))>0]

}

if (output == "u")

{

matOut <- matOut[as.numeric(names(matOut))<0]

}

matOut

}

# off.diag(dist(1:6))

wconfint <- function (x,lim=.95, clmean=T)

# This function returns mean and parametric confidence limits of x

# The default is 95%

# Note that the function does not check for suitability of the t distribution

# Missing values are dropped first

# Parameter added 10/18/00:

# clmean = F to use data as is, T to calculate confidence limits of the mean

# by dividing the variance by N

{

if (! (is.vector(x) & is.numeric(x))) stop ("confint requires a numeric vector or lm object")

x <- na.omit(x)

n <- length(x)

if (n <= 1) stop ("Not enough points")

q <- - qt((1-lim)/2,n-1)

s <- sqrt(var(x)/ifelse(clmean, n,1))

m <- mean(x)

r <- c(m, m+s\*q, m-s\*q,s\*q)

names(r)<-c("Mean", paste(c("Upper ","Lower ","Interval "),round(lim\*100,0),"%",sep=""))

print (paste("Confidence limits are for", ifelse (clmean, "means", "data points")))

r

}

# Functions for data manipulation---------------------

# NEW 2021-11-13

collapseColumns <- function (x, oldNames, newNames,

missName =c("keep","Other"))

{

# Collapses columns in a data frame

# This is useful for reducing the number of taxa in samples

# to focus on those more abundant. Column names are changed

# according to oldNames and newNames, and then the columns

# with the same names are summed and extra columns removed.

# Columns not named in the inputs can be either kept as is

# or combined into an "other" column. Alternatively some

# columns can be combined explicitly into "Other" and any column

# not named in oldNames will be retained.

#

# Inputs:

# x Data frame with only the taxa as columns

# oldNames One or more names from x

# newNames New names for the columns named by oldNames

# Must be the same length as oldNames

# missName Flag for how to deal with column names not

# included in oldNames:

# keep: Keep the missing column names as is

# Other: sweep the data from the missing column

# names into an "Other" column.

# Output: A data frame with a reduced number of columns.

# If N = columns in x, o = length(oldNames)

# and n = length (duplicated(newNames)),

# if (missName) = "keep",

# ncol(output) = N - n, else

# ncol(output) = o - n + 1

#

if (length(oldNames) != length(newNames))

stop ("Mismatch of old and new names\n")

nms <- names(x)

nmSeq <- match(nms, oldNames, nomatch=0)

which0 <- which(nmSeq > 0)

if (any(nmSeq==0))

{ if (substring(tolower(missName),1,1)=="k") { #keep

xBlank <- x[,nmSeq==0]

nmsOut <- nms

nmsOut[which0] <- newNames

nmsOut <- unique(nmsOut)

} else { # Other

xBlank <- data.frame (Other = rowSums(x[,nmSeq==0]) )

nmsOut <- c(unique(newNames), "Other")

}}

yMatch <- x[,oldNames]

names(yMatch) <- newNames

ykeep <- data.frame(rowName=newNames, t(as.matrix(yMatch)))

ykeep <- Bquick(ykeep, 1, sum)

xNew <- data.frame(t(ykeep[,-1]))

names(xNew) <- ykeep$rowName

if (!any(nmSeq==0)) {

xNew

} else {

if (missName=="keep") {

xout <- data.frame(xBlank, xNew)

names(xout) <- nmsOut

xout

} else {

xout <- data.frame(xNew, xBlank )

names(xout) <- nmsOut

xout

}}

}

# x <- data.frame(matrix(1:15, ncol=5))

# names(x) <- letters[1:5]

#collapseColumns(x, oldNames= c("a","c"), newNames=c("v","v"), missName="keep")

# collapseColumns(x, oldNames= c("a","c"), newNames=c("v","v"), missName="Other")

wrescale <- function(v, low=0, high=1, leftTrunc=NULL, rightTrunc=NULL)

{

# Returns a numeric vector scaled from low to high.

# If leftTrunc or rightTrunc are not null, it first limits v between those values

if (!is.numeric(v)) stop("wrescale requires numeric input")

if (!is.null(leftTrunc))

v <- pmax(leftTrunc, v)

if (!is.null(rightTrunc))

v <- pmin(rightTrunc, v)

vout <- ((v-min(v))/diff(range(v)))\* (high - low) + low

vout

}

distToMat <- function (x, diagvalue=1.0E9, method = "euclidean")

{

# Makes a matrix from a distance object or a matrix ready to create one

# NO error trapping in case x is not matrix or similar suitable object

# Default distance is euclidean and diagonals are set to diagvalue

# INput: x, a matrix or a dist object

# diagvalue can be any number - usually large for minima, or zero or NA

# method: euclidean is default, see help (dist) for other options

if (class(x)[1] != "dist")

x <- dist(x, method = method)

n <- attributes(x)$Size #

distmat <- matrix(diagvalue, ncol=n, nrow=n)

for (i in 1:(n-1)) for (j in (i+1):n)

distmat [i,j] <- distmat [j,i] <- x[n\*(i-1) - i\*(i-1)/2 + j-i]

distmat

}

blockdat <- function(x, rowind, colind, FUN = mean, zero = F, fconv = F, ...)

{

# This function converts a flat file into a block of data.

# MODIFIED 2/1/01 to allow conversion of numeric column indices back to numeric

# MODIFIED 1/22/03 to recover the column names (which the latest version of tapply buggered up)

# MODIFIED 2/28/03 to account for character data in column index and to block missing indices

# MODIFIED 2/25/05 to allow rowind to be a list or data frame

# MODIFIED 11/16/08 to include ... (parameters passed to FUN)

#

# x Vector to be put into the block

# rowind Vector of indices for the rows of the output

# colind Vector of indices for the columns of the output

# FUN Function for aggregating data, usually mean or median

# zero T to set values for row/column combinations that are

# missing values to zero, otherwise they are left missing

# fconv T if the index is to be converted from a factor to numeric or character.

#

# Output is a data frame with a number of columns equal to the number of

# unique values of colind, and the rows equal to the number of unique

# values of rowind.

#

# Output will be sorted by rowind

# To use more than one index, paste one or more indices together first, but

# note that numeric values may require leading zeros to sort correctly.

if(!is.function(FUN)) stop(paste("Function may be masked, value is:\n",

FUN))

if(is.atomic(rowind)) {

nrow <- length(rowind)

colnm <- "index"

rtemp <- rowind

}

else {

nrow <- length(rowind[[1]])

ncol <- ncol(rowind)

if(!is.matrix(rowind))

for(i in 2:ncol)

if(length(rowind[[i]]) != nrow) stop(

"Row indices have unequal lengths"

)

rtemp <- rowindex(rowind, sort = T)

rtind <- Bquick(data.frame(index = rtemp, rowind), 1, first)

}

if(length(x) != length(colind) | length(x) != nrow)

stop(paste(

"The input vector and indices must be of the same length;\n",

" lengths are", length(x), nrow, length(colind)))

if(any(is.na(colind)) | any(is.na(rowind)))

stop("Missing values in the indices are a no-no")

w <- tapply(x, list(rtemp, colind), FUN, ...)

if(zero)

w[is.na(w)] <- 0.

nm <- sort(unique(colind))

# Must be sorted because tapply sorts the columns

if(is.numeric(nm)) {

dd <- min(diff(nm))

if(dd >= 1)

dd <- 0

else dd <- - floor(log10(dd))

nm <- paste("x", numstring(nm, digits = dd), sep = "")

}

ww <- data.frame(dimnames(w)[[1]], w)

names(ww) <- c("index", nm)

if(is.atomic(rowind))

if(fconv & is.numeric(rowind))

ww[, 1] <- as.numeric(ww[, 1])

if(!is.atomic(rowind))

ww <- merge(rtind, ww, 1)

# Restore original indices

ww

}

Bquick <- function(x, ind, FUN, ...)

{

# Uses aggregate but converts index columns back to same mode as source

# Revised 6/15/02 to fix output from aggregate, which converts indices to factors or character

# x is a data frame

# ind is the number of columns (starting from the left) over which aggregation occurs

# In either case factors in the input are returned as factors

# and ordered factors are returned as ordered factors

# The ellipses ... can mean any argument passed to fun

if(ind > 1.) w <- as.list(x[, 1.:ind])

else

{

w <- vector("list", length=1)

w[[1]] <- x[, 1.]

}

xd <- data.frame(x[, (ind + 1.):dim(x)[2.]], stringsAsFactors = F)

y <- aggregate(xd, w, FUN, ...)

for(i in 1.:ind) {

if(is.ordered(x[, i]))

y[, i] <- ordered(y[, i], levels = levels(x[, i]))

else if(is.factor(x[, i]))

y[, i] <- factor(y[, i], levels = levels(x[, i]))

else {

y[, i] <- as.character(y[, i])

if(is.numeric(x[,i])) y[, i] <- as.numeric(y[,i])

}}

for(i in seq(ind, 1., -1.))

y <- y[order(y[, i]), ]

names(y) <- names(x)

row.names(y) <- 1.:dim(y)[1.]

y

}

insertna <- function(x, ind, droplast = T)

{

# Inserts rows of NA values between blocks of a data frame.

# This can be useful for graphing.

# Inputs:

# x A data frame with more than ind columns; The first ind columns are used

# for grouping

# ind The number of columns (max of 5) comprising the blocking variable

# droplast If TRUE, the last NA row is dropped (since it is usually extraneous

# Output

# Data frame with the same columns in x, but additional rows where the first ind

# column values are as in the previous row, and the remaining columns are NA

y <- Bquick(x, ind, last)

y[, (ind + 1.):dim(y)[2.]] <- NA

if(droplast)

y <- y[ - dim(y)[1.], ]

xx <- rbind(x, y)

xx <- wsort(xx, 1.:ind)

}

last <- function(x, n = 1.)

{

# Gives the last n values of a vector or last row of a matrix or data frame

if(is.matrix(x) | is.data.frame(x)) {

ind <- dim(x)[1.] - (1.:n) + 1.

ind <- rev(ind[ind > 0.])

x[ind, ]

}

else if(is.list(x) | !is.null(dim(x)))

stop("x is not a vector, matrix, or data frame")

else {

ind <- length(x) - (1.:n) + 1.

ind <- rev(ind[ind > 0.])

x[ind]

}

}

runlen <- function (x)

{

# This function determines the length of successive non-missing and missing

# values in a vector x.

# Input is a numeric vector x

# Output is a list with two components:

# x is a sequence of numbers indicating run length

# miss is a logical vector, T for missing and F for non-missing

v <- 1\*is.na(x)

vv<- abs(c(1,diff(v)))

w<- as.vector(table(cumsum(vv)))

ww<- is.na(x[vv>0])

list("x"=w,"miss"=ww)

}

fillintp <- function(x, maxna=length(x), na.ends = "keep")

{

# Fills in by linear interpolation

# Changed 2/18/03 to handle NA's at end:

# na.ends="omit" just clip them

# na.ends="keep" leave as NA

# na.ends="fill" replace with beginning or end values

# Changed 6/25/04 to allow a limit on the length of the NA regions to be filled

# maxna = Maximum length of sequences of NAs that will be filled in.

na.ends <- tolower(substring(na.ends, 1, 1))

nx <- length(x)

xm <- runlen(x)

k <- cumsum(xm$x)

xmc <- data.frame(start=k - xm$x + 1, end=k)

xmc <- xmc[xm$miss,]

xm1 <- xmc[xm$x[xm$miss] > maxna & xmc[,1] > 1 & xmc[,2] < nx,]

v <- 1:nx

r <- ifelse(na.ends=="f",2,1)

xx <- approx(v[!is.na(x)], na.omit(x), xout = v, rule=r)$y

if (nrow(xm1)>0)

for (i in 1:nrow(xm1))

xx[seq(xm1[i,1], xm1[i,2])] <- NA

if (na.ends=="o")

{

n1 <- xmc[xmc[,1] == 1 ,2] + 1

if (length(n1) == 0 ) n1 <- 1

n2 <- xmc[xmc[,2] == nx,1] - 1

if (length(n2) == 0 ) n2 <- nx

xx <- xx[n1:n2]

}

xx

}

bindata <- function(x, nbins, minbin=floor(length(x)/nbins), wt="equal")

{

# Bins data into categories that are either of approximately equal length

# or linearly weighted toward the high or low end of the distribution

# Inputs are:

# x A SORTED vector of real numbers

# nbins The number of bins to construct

# minbin The size of the smallest bin if bins are weighted. Because of ties

# the actual bin sizes may be slightly different.

# wt The weighting:

# "equal" means no weighting

# "low" means there are more bins than expected at the low end

# "high" means there are more bins than expected at the high end

# Value:

# A vector of indices corresponding to x, with values from 1 to nbins

# The indices are selected so that boundaries between bins do not split tied

# values of x. If there are a lot of ties this could result in bins containing

# no values, in which case the number of bins needs to be reduced.

if (any(x != sort(x))) stop ("Input x must be a sorted vector")

if (nbins > length(x)) stop ("Number of bins exceeds length of input vector")

wt1 <- switch(substring(wt,1,1), e=c(0,0), l=c(0,1), h=c(1,0), c(-1,-1))

if (wt1[1] == -1)

{

cat ("Warning: weighting parameter not matched, zero selected")

wt1 <- c(0,0)

}

lenx <- length(x)

nper <- lenx / nbins # Approximate number of values per interval

if (nper <= minbin)

{

cat ("Warning: minimum bin size ignored")

wt1 <- c(0,0)

}

if (sum(wt1) ==0 )

ind <- round(seq(1, lenx, length=nbins + 1)) # Initial index for no weighting

else

{

dif <- 2 \* (lenx / nbins - minbin)

dn <- seq (minbin + dif \* wt1[1], minbin + dif \* wt1[2], length=nbins)

ind <- round(cumsum(c(1,dn)))

ind[length(ind)] <- lenx

}

ind <- diff(ind) # Initial number per cell

ind[nbins] <- ind[nbins] + 1

for (i in 1:(nbins-1) )

{

n1<- length(x[ x <= x[sum(ind[1:i])]])

if (n1 > sum(ind[1:i]))

{

n2 <- n1 - sum(ind[1:i])

ind[i] <- ind[i] + n2

ind[(i+1):nbins] <-

diff(round(cumsum(c(0,ind[(i+1):nbins]) \* (lenx - n1 ) / (lenx - n1 + n2))))

}

}

ind[nbins] <- lenx - sum(ind[-nbins]) # Problem here

rep(1:nbins, ind)

}

bySweep <- function (x, xind,...)

{

# Subtracts means by group

# x is a data frame or vector of numerical data

# xind is either a vector (same length as nrow x),

# a data frame (same number of rows)

# or a vector of column

#

if (!is.data.frame(x)) {

x <- data.frame(x)

}

if (is.data.frame(xind) | length(xind)==nrow(x))

{

xout <- x

ind <- 0

}

else if (is.atomic(xind))

{

if (is.character(xind))

xind <- match(xind, names(x))

ind <- xind

xind <- x[,ind]

xout <- data.frame(x[,-ind])

}

else stop ("xind must be vector with length=nrow(x), data frame, or vector of column numbers")

names(xout) <- paste0(names(xout),".swp" )

for (i in 1:ncol(xout))

{

xout[,i] <- unlist(tapply(xout[,i], xind, FUN=function (v) v - mean(v),...))

}

if (ind[1] > 0)

{

xout <- data.frame(xind, xout)

names(xout)[1:ind] <- names(x)[1:ind]

}

xout

}

dline <- function (x, y, int=NULL, slope=NULL, xref=x, yref=y, report=T)

{

# Calculates distance along a line of points closest to the points (x, y)

# The line is either provided as intercept and slope,

# or is calculated as the major axis of points (xref, yref)

# NOTE: all x and y values should be in the same units, or standardized

# From https://en.wikipedia.org/wiki/Distance\_from\_a\_point\_to\_a\_line#Line\_defined\_by\_an\_equation

# Input

# x and y, the points to be included

# int and slope, if known

# xref and yref, if the line is made up of a separate cloud of points from x and y

# Note that the default is to determine the distance of the points along and across

# from their own major axis

# report=T to provide slope and intercept

# Value

# Data frame containing the index of the points (1 to N),

# the points on the line xline and yline

# and the distance along (dAlong, 0 for first point) and across the line (dAcross)

# which is positive to the left of the line starting from the first point

# (i.e., if the slope is 0 and the line starts on the left, positive y = positive across)

require (lmodel2)

N <- length(x)

if (length (y) != N) stop ("Lengths of x and y don't match")

if (is.null(int) | is.null (slope))

{

opw <- options("warn")[[1]]

options(warn=-1)

zref <- try(lmodel2 (yref ~ xref), silent=T)

options("warn"=opw)

zerr <- class(zref)

if (zerr != "try-error")

{

reg <- zref$regression.results

int <- reg[2,2]

slope <- reg[2,3]

}}

if (report) cat("Intercept = ", int, " Slope = ", slope, "\n")

if (is.null(int) | is.null (slope))

{

if (zerr != "try-error")

stop("Model II regression not fitted")

else

{

xline <- mean(x)

yline <- y

dAlong <- y

dAcross <- x

}

}

else

{ b <- -1

xline <- (slope \* y +x-int\*slope)/(slope^2 + 1)

yline <- (slope \* (slope\*y +x)+int)/(slope^2 + 1)

dAlong <- sqrt((xline-xline[1])^2 + (yline-yline[1])^2)

if (x[1] < mean(x)) sgn <- sign(y-yline)

else sgn <- sign(yline-y)

dAcross <- sgn \* sqrt((x-xline)^2 + (y-yline)^2)

}

data.frame(ind=1:N, xline=xline, yline=yline, dAlong=dAlong, dAcross=dAcross)

}

inspect <- function(y, x=NULL, imin=NULL, imax=NULL, xmin=NULL, xmax=NULL,

xName="X", yName="Value", panels=F, scales="fixed")

{

# Inspect a time series of data

# Input:

# y vector, matrix, or data frame of one or more series to inspect

# x x value for y; if null, 1:nrow(y)

# imin, imax: row number range of y to inspect (default is 1:nrow(y))

# xmin, xmax: range in x value to inspect (if provided this overrides imin and imax)

# xName, yName: names for x axes

# panel: T for one panel per column, F for multiple lines on one panel

# scales: "free\_y" to allow each panel to have a different scale

# Nothing returned: just a graph of y vs. x for the selected range

# It is possible to stack these using cowplot::plot\_grid

#

require(ggplot2)

require(reshape2)

if (length(dim(y)==2))

nr <- nrow(y)

else

{

nr <- length(y)

y <- matrix(y, ncol=1)

}

if (is.null(x))

x <- 1:nr

if (is.null(imin) & is.null(xmin))

{

imin <- 1

}

else

{

if (!is.null(xmin)) imin <- first(which(x >= xmin))

}

if (is.null(imax) & is.null(xmax))

{

imax <- nr

}

else

{

if (!is.null(xmax)) imax <- first(which(x >= xmax))

}

{

if (!is.null (xmin))

if (is.null (xmax)) imax <- which(last(x <= xmax))

}

yy <- data.frame(x=x[imin:imax], y[imin:imax,])

yy <- melt (yy, id.vars="x")

if (panels)

{

ggplot(yy, aes(x, value)) +

geom\_line(lwd=1, color="blue") +

facet\_wrap (~variable, ncol=1, scales="free\_y") +

xlab(xName) +

ylab (yName) +

theme(legend.position="bottom", legend.title=element\_blank()) +

wimGraph()

}

else {

ggplot(yy, aes(x, value, color=variable)) +

geom\_line(lwd=1) +

scale\_color\_manual(values=wimPalettes("L")) +

xlab(xName) +

ylab (yName) +

theme(legend.position="bottom", legend.title=element\_blank()) +

wimGraph()

}}

rowindex <- function(x, sort = F, global = T, countGroup = !global )

{

# Returns an index of rows for each unique combination of columns in input.

# Modified 3/16/03 to change the algorithm completely,

# and to allow for the data not to be sorted first,

# MODIFIED 7/19/03 to make it more efficient, and to add as.numeric to get correct sorting

# in which case the index will change each time the inputs change whether sorted or not.

# MODIFIED 9/1/03 to convert character data to factor, then numeric

# MODIFIED 7/15/08 to fix a problem with the indexing. Should work OK now

# Modified 8/6/09 to trap NA's in the indices

# If sort is T the data frame is sorted from left to right. The indices are then

# returned in the correct order for x (this is not too meaningful for global=F)

# If global is T the indices are the same for each unique combination of columns, and run from

# 1 to length(unique combination).

# If global is F the indices run within each combination from 1 to the length of the combination

# (this is also specified as countGroup=T, which overrides global)

if (!is.data.frame(x)) x <- as.data.frame(x)

row.names(x) <- seq(x[, 1])

nx <- dim(x)[1]

dx <- dim(x)[2]

if (sort)

x <- wsort(x, 1:dx)

for(i in 1:dx)

if(!is.numeric(x[, i])) x[, i] <- as.integer(as.factor(x[, i]))

if (nrow(na.omit(x)) != nx) stop ("Idiot! No missing values in indices")

xm <- as.matrix(x)

if (dx == 1)

v <- x[, 1]

else

v <- cumsum(c(T, apply(diff(xm) !=0,1,any)))

names(v) <- row.names(x)

rL <- rle(v)$lengths

if (!countGroup)

vv <- rep(seq(length(rL)), rL)

else vv <- unlist(lapply(rL, seq))

if(sort)

vv <- vv[order(as.numeric(names(v)))]

vv

}

unblock <- function (x, index=0, sort=T)

{

# Reverses blockdat by putting all of the block of data in a single column with the

# column names in a separate column.

# Inputs

# x Data frame with at least 2 columns (3 if index > 0)

# index Maximum column number for the index variables to be repeated

# index = 0 causes the row names to be used as an index variable (i.e., all of x is data)

# sort T to sort by index columns and column names - otherwise the data are not sorted

# Output is a data frame:

# x[,1:index] (or row names) repeated (ncol (x) - index) times

# names(x) for columns > index repeated each nrow(x) times

# Values of x for columns > index converted to a vector in column (i.e., normal matrix) order.

xt <- x[, (max(index) + 1):ncol(x)]

xc <- apply(xt, 2, is.numeric)

if (any(xc != xc[1])) cat ("Watch out: not all values have the same type\n")

if (index == 0 )

{

xi <- row.names(x)

nmi <- "row.name"

}

else

{

xi <- x[,1:index]

nmi <- names(x)[1:index]

}

nm <- names(xt)

ind <- rep(1:nrow(xt), ncol(xt))

if (is.null(dim(xi) )) xi <- xi[ind]

else xi <- xi[ind,]

y <- data.frame(xi, rep(nm, each=nrow(xt)), as.vector(as.matrix(xt)))

names(y) <- c(nmi, "col.names", "data")

if (sort) y <- wsort(y, 1:(index+1))

y

}

xlineseg <- function (x,y,xref,yref)

{

# This function determines the distance along and across a broken line represented

# by each point in a matrix of (x,y) pairs. The procedure for each line

# segment is to transform the (x,y) pair to (xnew, ynew) relative to that line

# segment as the x axis. Then if xnew is within the line segment (ie

# the point (x,y) falls on a line normal to the segment), the distance

# is znew. If the (x,y) pair does not reach the first line segment or

# reaches beyond the last, a missing value is returned. If the point is

# on the outside of a curve, the program extends the closest line segment

# beyond the intersection point.

# The cross-segment distance is positive to the right going in order of xref.

nseg <- length (xref) - 1

if (nseg < 1 ) stop ("Not enough points in line segments")

if (nseg != length (yref) - 1 ) stop ("X and Y for line segments unequal")

npairs <- length (x)

if (npairs!=length (y) ) stop ("X and Y have unequal length")

xdist <- adist <- rep(NA, npairs)

mode (xdist) <-"numeric"

x1 <- xref[1:nseg]

y1 <- yref[1:nseg]

x2 <- xref[2:(nseg + 1)]

y2 <- yref[2:(nseg + 1)]

len<-sqrt((x1-x2)^2+(y1-y2)^2)

odist <- rep(0,nseg)

if (nseg > 1)

for (i in 2:nseg) odist [i] <- odist [i-1] + len[i-1]

theta<- pi \* ( x2 < x1 ) + sign (x2-x1) \* asin ((y2 - y1) / len)

for (i in 1:npairs)

{

xnew <- (x[i] - x1 ) \* cos (theta) +

(y[i] - y1 ) \* sin (theta)

ynew <- (y[i] - y1 ) \* cos (theta) -

(x[i] - x1 ) \* sin (theta)

da <- pmin(len - xnew, xnew)

ydist <- sqrt(ynew^2+(da\*(da<0))^2)

iloc <-sort.list(ydist)[1]

adist [i] <- xnew [iloc] + odist [iloc]

vi <- first(which(ydist==min(ydist)))

xdist [i]<- ydist[vi]

xdist[i] <- xdist[i] \* -sign(ynew[vi])

}

list(along=adist, across=xdist)

}

#xref <- c(0,1)

#yref <- c(0,1)

#x <- c(0.5, 0, 1,2)

#y <- c(0.5,1,0,0)

#xlineseg(x, y, xref, yref)

#Par(1)

#plot(x,y)

#lines(xref,yref)

# Functions for manipulating Monitoring data sets (water, zooplankton, fish)----------------

cdecGet <- function(stations,sensors=c(5,92, 100, 102),

sensorNames = c("salsurf5","salbott92","salsurf100","salbott102"),

nDays=1,endDate="today")

{

# Functon to get CDEC data off the web.

# Enter this function with:

# stations: vector of one or more 3-letter station designators

# sensors: Sensor numbers used in CDEC. Default is all EC (Conductivity) values

# sensorNames: Sensor names (default is the names for EC converted to salinity

# nDays Number of days of data (default is 1).

# endDate Last date of the data (default is now, which gives data for nDays

# plus today up to the current time)

# Output:

# Data frame with station, date/time, sensor number,

# variable name, units, value, and a QC flag from CDEC.

# EC data are converted to salinity but no other transormations are done.

# Output is ready to be plotted in ggplot with station and variable name (=sensor name) as factors

names(sensors) <- sensorNames

cat("Getting data for stations:\n", stations,

"\nSensors\n",sensors, "\n" )

require(lubridate)

if (endDate=="today") {

end.date <- Sys.Date() + 1

start.date <- Sys.Date() - nDays

} else {

end.date <- as\_date(endDate)

start.date <- end.date - nDays

}

cdec <- cder::cdec\_query(stations= stations, sensors= sensors,

durations= "E", start.date= start.date, end.date= end.date)

names(cdec) <- tolower(names(cdec))

cdec <- as.data.frame(cdec)

cdecIn <- cdec

cdec <- cdec[,c("stationid","obsdate","sensornumber", "sensortype",

"sensorunits", "value", "dataflag")]

names(cdec) <- c("station", "date", "sensorNo","varName","units", "value", "flag")

cdec <- cdec[cdec$date <= Sys.time(),]

salSensIndex <- unique(match(cdec$sensorNo, c(5,92, 100, 102)))

if (is.null(sensorNames)) {

cat("Missing sensor names\n")

return()

} else {

if (!is.na(salSensIndex[1]) )

{

for (i in salSensIndex)

{

cat(i, sensorNames[i], sensors[i], "\n")

ind <- cdec$sensorNo == sensors[i]

vali <- sal.ec(cdec$value[ind])

cdec$value[ind] <- vali

cdec$varName[ind] <- sensorNames[i]

cdec$units[ind] <- "salinity"

}}

# testCalc <- all(na.omit(sal.ec(cdecIn$value)- cdec$value)==0)

# if (!testCalc) cat("------ Mismatch in salinity calculation from EC\n")

}

cdec$station <- factor(cdec$station, levels=stations)

cdec

}

dayflowGet <- function(years=c(1955, 2030), mos=c(1,12), daily=F, tocms=T, toWaterYear=F,

vars=c("tot","out","exp","X2") )

{

# Gets dayflow and X2 data according to selected criteria

# Inputs

# years = range of years to be included (default is all years)

# mos = range of months to be included (default is all months)

# daily = T for daily data, or F for monthly (the default)

# tocms = T to convert flows to cms (the default; does not apply to non-flow vars )

# toWaterYear = T to calculate water year

# vars = character vector of variables to include, which must be X2 or Dayflow variables.

# NOTE: case insensitive

#

load("C:/1Current/R-files/RdataFiles/DayflowX2.Rdata")

vars <- tolower(vars)

x2Flag <- any(vars=="x2")

vars <- vars[vars != "x2"]

varList <- c("sac","yolo","csmr","moke","misc","sjr","east","tot","ccc","swp","cvp",

"nbaq","exports","gcd","prec","misdv","cd","xgeo","west","rio","out",

"expin","diver","effec","effdiv","julday","gateopen", "exp")

vNA <- is.na(match(vars, varList))

if (any (vNA))

stop (paste("Variable name not found:", vars[vNA]))

if (daily)

{

dfnm <- "dayflow"

x2nm <- "x2daily"

nms <- c("year","mo","julday","date")

}

else

{

dfnm <- "dayflow.mo"

x2nm <- "x2month"

nms <- c("year","mo")

}

df <- dayflowX2[[dfnm]]

df <- df[df$year>= years[1] & df$year <= max(years),]

df <- df[df$mo>= mos[1] & df$mo <= max(mos),]

if (any(vars=="exp"))

df$exp <- df$cvp + df$swp + df$ccc # Exclude north bay aqueduct

df <- df[,c(nms, vars)]

varFlow <- vars[vars != "gateopen" & vars !="effdiv" & vars != "diver"]

if (tocms & length(varFlow)>0)

df[,varFlow] <- cftocm (df[,varFlow])

if (x2Flag)

{

nms <- nms[nms != "date"]

x2 <- dayflowX2[[x2nm]]

x2 <- x2[,c(nms, "x2")]

df <- merge(df, x2, 1:length(nms))

}

if (toWaterYear)

{

df$wyear <- df$year + ifelse(df$mo >=10,1,0)

}

df <- wsort(df, 1:3 )

df

}

dayflowIn <- function (wyear1=1955, wyear2=NULL, vars=c("year","mo","date","out"))

{

if (is.null(wyear2)) wyear2 <- as.integer(substring(Sys.Date(),1,4))

dfdir <- "C:\\2Data\\BigData\\BayData\\Dayflow\\"

if (wyear1 < 1997)

{

df1 <- read.csv(paste(dfdir,"Dayflow\_1956-1996.csv", sep=""))

names(df1) <- lowerCase(names(df1))

df1 <- df1[df1$year >= wyear1 | (df1$mo >=10 & df1$year == wyear1-1),]

df1 <- df1[df1$year <= wyear2 | (df1$mo < 10 & df1$year == wyear2),]

if (vars == "all")

{ vars <- names(df1) }

# else if (vars=="pick")

else

{

ind <- match(vars, names(df1))

if (any(is.na(ind)))

{

cat("Name", vars[is.na(ind)], " not in Dayflow\n")

stop

}

}

df1 <- df1[,vars]

}

if (wyear2 >= 1997)

{

startyr <- max(1997, wyear1)

for (wyr in startyr:wyear2)

{

cat(wyr, "\n")

fn <- paste(dfdir, "dayflowCalculations",wyr, ".csv", sep="" )

if (file.exists(fn))

{

if (wyr==startyr)

{ df <- read.csv(fn) }

else

{ df <- rbind(df, read.csv(fn)) }

}

}

names(df) <- lower(names(df))

df <- df[,vars]

if (wyear1<1997)

df1 <- rbind(df1, df)

}

}

getAnneZoops <- function (fn, startData)

{

# BRings in and rearranges zooplankton data using Anne's format.

# May require some post-hoc adjustments

# Date, Station, and Total are assumed to be in the rows of data somewhere

# Other header rows are kept with an abbreviated title.

# Taxon names are abbreviated to first 4 letters with stage appended if present.

# Some special handling is needed to prevent duplicates: P marinus and mysid larvae

# Input:

# fn CSV file of abundance data in Anne's format (NOTE: remove commas)

# startData: Row at which data starts

# NOTE Col 1 must contain rows with the words "date", either "sta or "site", and "tot"

# Output

# List with three data frames:

# staData Station/sample data: rows are samples

# abundance Number per m3 for all taxa: same number of rows as staData

# spnames Cross-reference from original to abbreviated taxon names

# (abundance has 1 more column than spnames has rows)

#

x <- read.csv(fn)

id <- first(which(tolower(x[,1])=="date"))

is <- first(which(match(substring(tolower(x[,1]), 1, 3),

c("sta", "sit", "sam"), nomatch=0)>0))

it <- first(which(substring(tolower(x[,1]), 1, 3)=="tot")) # Total abundance

if(length(it)==0) it <- nrow(x)

ii <- c(id, is, it)

iv <- (1:(startData-1))[-ii]

iv <- iv[iv >= id]

nms <- c("sample", "date","sta","total")

nms <- c(nms, substring(gsub(" ", "", x[iv,1]), 1, 8))

staData<- data.frame(1:(ncol(x)-1), t(x[c(ii,iv),-1]))

names(staData) <- nms

rownames(staData) <- staData$sample

cat ("Got header\n")

y <- as.matrix(x[startData:(it-1), -1])

y <- gsub("," , "", y) # Remove commas from y values

y[y==""] <- "-999"

mode(y)<- "numeric"

y[y < -900] <- 0

ynum <- apply(y, 1, function(v) length(which(!is.na(v)))==length(v))

if (!last(ynum)) y <- y[1:(nrow(y)-1),]

cat ("Got data\n")

nmy <- trimws(x[startData:(it-1), 1][1:nrow(y)])

nmy1 <- right (nmy,1)

nma <- tolower(nmy)

nma <- ifelse(substring(nma,1,10) == "mysid larv", "mysJ",nma)

nmx <- substring(nma, 1, 4)

nmx <- ifelse (nmx=="pseu" & chfind(nma, "mar") > 0, "pmar", nmx)

sex <- ifelse(chfind(nma, "ovig")>0 | chfind(nma, "grav")>0, "G",

ifelse(chfind(nma, "female")>0, "F",

ifelse(chfind(nma, "male")>0, "M", "")))

stgn <- rep("", length(nmy))

for (i in 1:5) stgn[chfind(nmy, paste0("C",i))>0] <- paste0("C",i)

stg <- ifelse(chfind(nma, "adult"), "A",

ifelse(nmy1=="A" | chfind(nma,"adult")>0, "A",

ifelse(nmy1=="J" | nmy1=="C"| chfind(nma,"juve")>0, "C",

ifelse(nmy1=="N"| chfind(nma,"naup")>0, "N", ""))))

stg[sex != ""] <- ""

stg1 <- paste0(stg, stgn, sex)

nmt <- paste0(nmx, stg1)

nmg <- c("Adult","Female","Male","Gravid","Cope","Naup", "")

nmg1 <- nmg[match(substring(stg1,1,1), substring(nmg,1,1), nomatch=6)]

spnames <- data.frame(seq=1:length(nmy), original=nmy, revised=nmt, stage=nmg1)

yy <- data.frame(1:ncol(y), t(y))

names(yy) <- c("sample", nmt)

rownames(yy) <- staData$sample

list (staData=staData, abundance=yy, spnames=spnames)

}

#&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

# these fuctions (variously) need:

right

chfind

datemo

dateyr

julyear

right <- function(x, n)

{

# Gets rightmost n characters of vector x

# If n is negative it keeps all BUT the rightmost n characters

nc <- nchar(x)

if (n[1] > 0)

v <- substring(x, nc - n + 1)

else

v <- substring(x, 1, nc + n )

v

}

datemo <- function (dateval) # Calculates numeric month from a dates object

{

if (class (dateval) !="Date") stop("Input must be dates or timeDate")

as.integer(format(dateval, "%m"))

}

# datemo(as.Date("1/25/1970", format="%m/%d/%Y"))

dateyr <- function(dateval) # Calculates numeric year from a dates object

{

if (class (dateval) !="Date") stop("Input must be dates or timeDate")

as.integer(format(dateval, "%Y"))

}

julyear <- function(m, d=NULL, y=NULL)

{

# Function to get julian date within year, with 1 Jan = 1

# If no values given it returns today's Julian date\

# Input

# m EIther a Date object or a numeric month if d and y are also given

# m can also be a list with m, d, y as components

# Modified 6/17/2015 for R

if(class(m)[1] == "Date")

{

dt <- m

dt1 <- format(dt, format="%m/%d/%Y")

y <- as.integer(right(dt1,4))

}

else

{

if(is.list(m) & length(m) == 3)

{

y <- m[[3]]

d <- m[[2]]

m <- m[[1]]

}

if (!is.null(d) & !is.null(y))

{

dt <- as.Date(paste(as.character(y), m, d, sep="-"))

}

}

# else {stop ("Wim: Need either Date object or mdy or a list\n")}

dt0 <- as.Date(paste0("12/31/", y-1), format="%m/%d/%Y")

as.integer(julian(dt) - julian(dt0))

}

getBayStudy <- function (fname="bayStudyFish.Rdata",

path="C:/1Current/R-files/1R-projects/BayStudyDatabase/",

listNames=c("BoatStation","BoatTow","Fish Catch Data","Fish Length Data",

"SalinTemp","SpeciesCodes","StationConstants", "minLength", "ageClass"),

dsnames =c("boat","tow","catch","fishlen","st","sp","sta","minlen","ageClass" ),

include.fld = c("index","year", "survey","julday", "station",

"net","time","tow", "depth","secchi","salsurf","tempsurf"),

dropLT40 = TRUE,

cpue=F, output="total",

outFormat = switch(substring(output,1,1), t="table", a="table", l="flatfile"),

net=1, exclude.badCode=F, spcode = character(0),

fam = character(0), gen=character(0), minLength=T,

indices = integer(0),years = integer(0),surveys = integer(0),

stations=integer(0), sals = numeric(0) ) # End inputs

# Function to query Bay Study database for selected taxa, samples, and output

# Data reported as total catch, catch per age class, or catch per length.

# Data source:bayStudyFish.Rdata, produced by: R import Bay Study database.R

# UPDATED 2021-08-22 to remove fish smaller than 40mm (dropLT40) which have been reported inconsistently

# Output formats:

# Table (default for total catch per sample or per age class)

# Columns for sample variables including selected environmental variables

# One column for each taxon (and age class if selected)

# Data as catch per trawl OR CPUE in # per 10,000 m2 (Otter trawl net 2) or m3 (midwater, net 1)

# Missing values set to 0 but fish missing from entire selection are not included

# Flat file (default for length)

# Data include catch per trawl, proportional catch, and CPUE as above.

# Zeros omitted

#

# Control variables with defaults:

# cpue=F Calculate CPUE or leave data as catch per trawl

# output="total" TOtal fish or by "ageClass" or "length"

# outFormat" "Table" if total or ageClass, "flatfile" if lengths

# net=1 1=Midwater, 2=Otter

# exclude.badCode=F Remove data with flags indicating bad trawl

# or stations 141 or 317 or 319 in 1980 or 1981 survey 1

# minLength T to exclude fish smaller than min length in database

# Selection variables (all these default to ALL)

# spcode: Character vector of species codes (=alphacode)

# genus: Character vector of genus names (must be spelled correctly!)

# family: Character vector of family names (must be spelled correctly!)

# indices: Indices to include (numeric, range is OK)

# years: Range of years to include

# surveys: Range of surveys (~months) to include

# stations:Stations to include (these are numeric - nnn:nnn is OK)

# sals: Range of SURFACE salinities to include

#

{ require(dplyr)

load(paste0(path, fname))

databaseName <- substring(fname, 1, regexpr("rdata", tolower(fname))-2)

cat("Using source file:", databaseName, "\n")

database <- get(databaseName)

listInd <- match(listNames, names(database))

for (i in 1:length(dsnames)) assign (dsnames[i], database[[listInd[i]]])

boat <- boat[,c(1:7, ncol(boat))] # Exclude substrate, waves, tide

tow$area.vol <- ifelse(tow$net==2, tow$areaM2, tow$volM3)

tow <- tow[tow$net==net,c("index","net","time","tow","area.vol","comment")]# Select net

fld <- left\_join(boat, st[,substring(names(st),1,2) !="ec"], by="index")

output <- tolower(substring(output,1,1))

if (is.na(match(output, c("a","l","t"))))

{

cat("\*\*\*\*\*\*\*\*\*\*\*\*\* Output", output, "incorrect, changed to total\n")

output <- "t"

}

fld <- selectField(fld, indices=indices, years=years, surveys=surveys, sals=sals)

fldTow <- inner\_join(fld, tow, "index") # Exclude samples with no tow this net

# Delete tows with bad data codes or a few stations

if (exclude.badCode)

{

goodTow <- match(fldTow$tow, c(52,53,54), nomatch=0)==0

goodSta <- !(fldTow$station==141 |

(fldTow$station ==317 | fldTow$station==319) &

(fldTow$year==1980 | (fldTow$year==1981 & fldTow$survey==1)))

fldTow <- fldTow[goodTow & goodSta,]

}

# Select sample variables to include in output

include.fld <- unique(c("index","survey", include.fld))

if (all(tolower(include.fld) != "all"))

{

if (length(include.fld) > 0 && include.fld[1] != "")

{

nms <- c("sal","temp") # Allows all salinity or all temperature values; else specify

pos <- match(nms, include.fld)

for (j in 1:2)

if (!is.na(pos[j]))

{

include.fld <- c(include.fld[include.fld!=nms[j]],

paste0(nms[j], c("surf","avg","bott")))

}

fldTow <- fldTow[, c(include.fld,"area.vol", "comment")]

}

else fldTow <- fldTow[, c("index","year","survey","station", "area.vol")]

}

# Select fish taxa

spcodeDF <- selectFish(program="Bay", spcode=spcode, fam=fam,

gen=gen)[,1]

# Subset the fish length and catch tables to include only selected samples and species

# NOTE: DROP all fish smaller than 40 mm which were not counted consistently.

# Before output the tables are expanded to include all taxa

if (dropLT40)

fishlen <- fishlen[fishlen$length >= 40,]

flen <- inner\_join( fldTow[, c("index","survey")],

fishlen[fishlen$net==net & match(fishlen$alphacode, spcodeDF, nomatch=0)>0,-(2:3)],

by="index")

fcatch <- inner\_join( fldTow[,c("index", "survey")],

catch[catch$net==net & match(catch$alphacode, spcodeDF, nomatch=0)>0,-(2:3)],

by="index")

# Branch depending on output = all lengths, age classes, or total fish.

# NOTE in length analysis I had to change the name "length" to "fishindex" for later analysis

if (output=="l")

{ # Here for length only

query1 <- flen

names(query1)[match(c("length","frequency"), names(query1))] <- c("fishindex","catch")

nm <- "length"

} else { # Here for total or by age class

query1 <- left\_join(flen, ageClass, by=c("alphacode","survey"),

suffixes=c("", ".ac"))

if (output=="a")# NOTE age classes have length >= the cutoff value in ageClass

{ # Here for age class

query1$yoycut[is.na(query1$yoycut)] <- 10000 # Very high - no cutoff

query1$age1cut[is.na(query1$age1cut)] <- 10000

query1$minlen[is.na(query1$minlen)] <- 20 # From Guide, for other species

query1$fishindex <- ifelse(query1$length<query1$minlen, -1,

ifelse(query1$length<query1$yoycut, 0,

ifelse(query1$length<query1$age1cut, 1,2)))

query1 <- aggregate(query1$frequency, query1[,c("index","alphacode",

"sizegroup", "fishindex")], sum)

nm <- "ageClass"

} else { # Here for TOTAL fish from each taxon selected

if (minLength) # Include if >= minimum length?

{

query1 <- query1[query1$length>= query1$minlen,]

}

query1 <- aggregate(query1$frequency, query1[,c("index","alphacode","sizegroup")], sum)

query1$fishindex <- rep(-99, nrow(query1))

nm <- "ignore"

}}

# All branches come back here - combine with fish catch and

# calculate number per sample bu summing across size groups.

# These results are usually expanded from actual counts.

names(query1)[names(query1)=="x"] <- "catch"

query1Tot <- aggregate(query1$catch, query1[,c("index","alphacode","sizegroup")], sum)

names(query1Tot)[4] <- "catch.tot"

query2 <- left\_join(query1, query1Tot, c("index","alphacode","sizegroup"))

query2$propInd <- query2$catch / query2$catch.tot

query3 <- left\_join(query2, fcatch, by=c("index", "alphacode", "sizegroup"))

query3$catchByInd<- (query3$pluscount\*query3$propInd + query3$catch)\* query3$qtscaught/query3$qtssubsampled

query3 <- query3[,c("index","alphacode","sizegroup","fishindex","catch","pluscount","catchByInd")]

query4 <- aggregate(query3$catchByInd, query3[,c("index","alphacode","fishindex")], sum)

names(query4)[names(query4)=="x"] <- "catchPerSample"

if (output=="l" | output=="a")

{

query4Tot <- aggregate(query4$catchPerSample, query4[,c("index","alphacode")], sum)

names(query4Tot)[3] <- "catch.tot"

query4 <- left\_join(query4, query4Tot, c("index","alphacode"))

query4$propPerSample <- query4$catchPerSample/query4$catch.tot

query4 <- query4[,!(names(query4)=="catch.tot")]

}

# Branch depending on whether output is a table or a flat file

# Table is OK for total fish by taxon or age classes, but

# cannot be selected for lengths because the table would be unwieldy.

# NOTE that flat file may have missing elements if no fish were caught.

if (tolower(outFormat)=="table")

{

if (output=="a")

{

query5 <- dcast(query4, index ~ alphacode + fishindex, value.var="catchPerSample",

fun.aggregate=sum, fill=0)

}

else

{

query5 <- dcast(query4, index ~ alphacode, value.var="catchPerSample",fun.aggregate=sum, fill=0)

}

# Combine catch data with field data and calculate CPUE if needed.

out <- left\_join(fldTow,query5, "index")

cols <- (ncol(fldTow))+ (2:ncol(query5))-1

cat("Saving", nrow(out), "rows and ", ncol(query5)-1, "taxa in table format")

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

if (cpue)

{out[,j] <- out[,j] \* 10000 / out$area.vol

}

}

if (cpue)

{

cat(", as CPUE\n")

} else cat (", as catch per trawl\n")

} else # Here to output flat file.

{

out <- left\_join(fldTow, query4, "index")

names(out)[names(out)=="fishindex"] <- nm

if(cpue)

out$cpue <- round(out$catchPerSample \* 10000 / out$area.vol, 3)

}

out

}

#################################################################################

# get20mm added and somewhat tested 8/10/2021

# MOdified 12/09/2021 to fix the length data.

#################################################################################

get20mm <- function (fname="allFile20mm.2020.Rdata",

path="C:/1Current/R-files/1R-projects/20mmDatabase/",

listNames= c("fishcodes","fishlength","fishsample","fishvol",

"gear","sp.xref","stalatlong","station","survey","tow"),

include.fld = c("year", "mo","survey","julday","time",

"depth","secchi","sal.surf","sal.bot",

"temp", "vol10K"),

cpue=F,

output = "total",

outFormat = switch(substring(output[1],1,1), t="table", l="flatfile"),

fish.code = 2,

fam = character(0), gen=character(0),

years = integer(0),mos = integer(0),

stations=integer(0), sals = numeric(0) ) # End inputs

# Function to query 20mm database for selected taxa, samples, and output

# Data reported as total catch or catch per length.

# Length is for one species only, and the program will select the first if >1

# Required files:

# Data source:allFile20mm.2020.Rdata, produced by:

# R import 20mm database.R

# fishCodesAllPrograms.Rdata

# Several functions in wimsFunctions.txt

# Output formats:

# Table (default for total catch per sample)

# Index which combines year survey station, and year mo station volumefiltered

# Columns for sample variables including selected environmental variables

# One column for each taxon

# Data as catch per trawl OR CPUE in # per 10,000 m3

# Missing values set to 0 and ALL samples included that meet the selection criteria

# Flat file (default for length)

# Data include catch per trawl, proportional catch, and CPUE as above.

# Zeros omitted and only the first taxon selected.

#

# Control variables with defaults:

# cpue=F Calculate CPUE or leave data as catch per trawl

# output="total" TOtal fish or by "length"

# outFormat "Table" if total , "flatfile" if lengths

#

# Selection variables (nnn:nnn is OK; these default to ALL)

# fish.code: numeric 20mm species codes

# gen: Character vector of genus names (must be spelled correctly!)

# fam: Character vector of family names (must be spelled correctly!)

# years: Range of years to include

# surveys: Range of surveys to include

# stations:Stations to include (these are numeric)

# sals: Range of SURFACE salinities to include

#

{

require(dplyr)

require(reshape2)

load(paste0(path, fname))

databaseName <- substring(fname, 1, regexpr("rdata", tolower(fname))-2)

cat("Using source file:", databaseName, "\n")

database <- get(databaseName)

listInd <- match(listNames, tolower(names(database)))

output <- tolower(substring(output, 1,1))

outFormat <- tolower(substring(outFormat, 1,1))

#--------------------------------------------------

# Assign the list of names to database elements

#--------------------------------------------------

for (i in 1:length(listNames)) assign (listNames[i], database[[listInd[i]]])

output <- tolower(substring(output[1],1,1))

if (is.na(match(output, c("l","t"))))

{

cat("\*\*\*\*\*\*\*\*\*\*\*\*\* Output", output, "incorrect, changed to total\n")

output <- "t"

}

#--------------------------------------------------

# Consolidate files to survstation, tow, fish

#--------------------------------------------------

# survey - drop comments which are empty, add year mo julday

survey <- select(survey, -comments)

survey$year <- dateyr(survey$date)

survey$mo <- datemo(survey$date)

survey$julday <- julyear(survey$date)

# station: calculate salinity surface and bottom

station$sal.surf <- sal.ec(station$topec)

station$sal.bot <- sal.ec(station$bottomec)

station <- select(station, surveyid, stationid, station,

temp, sal.surf, sal.bot, secchi, turbidity)

# Combine station and survey. NOTE sampleID is 1 for each year/survey/station sampled

survstation <- merge(survey, station, by= "surveyid")

survstation <- arrange(survstation, surveyid, stationid)

survstation$sampleID <- 1:nrow(survstation)

# tow: replace time with hour.decimal minute

tow <- select(tow, towid, stationid, townum, time=towtime, depth=bottomdepth,tide )

colon <- chfind(tow$time, ":")

hr <- as.numeric(substring(tow$time, colon-2, colon-1))

min <- as.numeric(substring(tow$time, colon+1, colon+2))

tow$time <- hr + round(min/60, 2)

tow <- merge(tow, select(fishvol, towid, vol10K), 1)

# fishsample - remove gear as intermediate file - towid is the index

gear2 <- select(gear[gear$gearcode==2,], gearid, towid)

fish <- select(fishsample, gearid, fishsampleid, fishcode, catch)

fish <- merge(gear2, fishsample, "gearid")

fish <- select(fish, -gearid)

#--------------------------------------------------

# Subset to select values of sample parameters:

# Years, months, surveys, stations, salinity

# Then merge sample with tow to get one row per fish sample

#--------------------------------------------------

sample <- selectField(survstation, years=years, mos=mos, sals=sals, stations=stations)

sample <- merge(sample, tow, "stationid")

#--------------------------------------------------

# Select sample variables to include in output - either selected, minimum, or all

#--------------------------------------------------

# Select sample variables to include in output - either selected, minimum, or all

samphead <- c("surveyid", "sampleID", "towid", "date",

"survey","station", "townum")

sampNames <- names(sample)[match(names(sample), samphead, nomatch=0)==0]

if (substring(tolower(include.fld[1]),1,1) == "m")

{

samplesub <- sample[,samphead]

} else if(tolower(include.fld[1]) != "all") # If "all" don't recude sample

{

fldMatch <- match (include.fld, sampNames)

fldNomatch <- which(is.na(fldMatch))

if (length (fldNomatch) > 0) # Check for incorrect variable names

{

cat("Dropping unknown fields:", include.fld[fldNomatch], "\n" )

include.fld <- include.fld[-fldNomatch]

}

samplesub <- sample[,c(samphead, include.fld) ]

} else

{

samplesub <- sample

}

#--------------------------------------------------

# Select fish taxa. Restrict to the first for length

#--------------------------------------------------

spcodeDF <- selectFish(program="20", spcode=fish.code, fam=fam,

gen=gen)

if (outFormat=="f" & length(spcodeDF) > 1)

{

spcodeDF <- spcodeDF[1,]

cat("First taxon only included for flat file output:\n")

}

ntaxa <-nrow(spcodeDF)

cat("Analysis on:\n")

print(spcodeDF)

#--------------------------------------------------

# Subset the fish length and catch tables to include only selected samples and species

# Before output the tables are expanded to include all taxa.

# NOTE:

# for catch only: fcatch has total catch per sample without zeros

# for length: fcatch has 1 row per fish and not all fish are measured.

# It includes nfish (number by length)

#--------------------------------------------------

cat(nrow(samplesub), "Rows, ", ncol(samplesub), " Columns:", "\n")

fcatch <- inner\_join( samplesub[, c("towid", "sampleID")],

fish[match(fish$fishcode, spcodeDF$code, nomatch=0)>0,],

by="towid")

# For length analysis

if (output=="l")

{

flen <- inner\_join( fcatch, fishlength,by="fishsampleid")

flen$nfish <- rep(1, nrow(flen)) # Added 2021-12-09 to get right number of fish

flen <- select(flen, -fishlengthid)

flen <- Bquick(select(flen, towid, length, nfish), 2, sum)

fcatch <- merge(fcatch, flen)

}

# Drop unneeded variable

fcatch <- select(fcatch, -fishsampleid )

# Branch depending on whether output is a table or a flat file

# Table is OK for total fish by taxon but not lengths.

# Also only catch is converted to CPUE if required

#

if (outFormat=="t" & output!="l")

{

block <- dcast(fcatch, towid ~ fishcode ,

value.var="catch",

fun.aggregate=sum, fill=0)

missingCodes <- spcodeDF$code[is.na(match(spcodeDF$code, names(block)))]

if (length(missingCodes) > 0)

{

addBlock <- makedf(missingCodes, nrow=nrow(block))

block <- data.frame(block, addBlock)

}

out <- left\_join(samplesub, block, "towid")

cols <- (ncol(samplesub)) + (2:ncol(block))-1

cat("Saving", nrow(out), "rows and ", ncol(block)-1, "taxa in table format\n")

names(out)[cols] <- spcodeDF$abbrev

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

if (cpue)

{

out[,j] <- out[,j] / out$vol10K

}

}

if (cpue)

{ cat(", as Catch per 10,000 m3\n") }

else { cat (", as catch per trawl\n") }

out <- arrange(out, date, station, townum)

} else # Here to output flat file with either catch or catch and length

{

cat("Output is flat file, total catch may exceed number counted (nfish)\n")

out <- inner\_join(samplesub, fcatch, "towid")

cols <- na.omit(match(c("catch","nfish"), names(out)))

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j]) # Set missing catch or nfish to 0.

}

if (cpue)

{

out$cpue <- out$catch \* 10000 / out$volM3

}

out <- arrange(out, date, station, townum, length )

}

cat (names(out),"\n")

out

}

getMWT <- function (fname="MWT.Rdata",

path="C:/1Current/R-files/1R-projects/MWTDatabase/",

listNames=c("sample","catch","length","organisms","stations"),

include.fld = c("index","year", "mo","julday", "station",

"time", "depth.m","secchi","sal.surf","sal.bot", "temp"),

cpue=F,

output=c("total", "length"),

outFormat = switch(substring(output[1],1,1), t="table", l="flatfile"),

spcode = character(0),

fam = character(0), gen=character(0),

years = integer(0),mos = integer(0),

stations=integer(0), sals = numeric(0) ) # End inputs

# Function to query FALL MWT database for selected taxa, samples, and output

# Data reported as total catch or catch per length.

# Length is for one species only, and the program will select the first if >1

# Required files:

# Data source:MWT.Rdata, produced by: R import MWT database.R

# fishCodesAllPrograms.Rdata

# Functions selectField and selectFish (in wimsFunctions.txt)

# Output formats:

# Table (default for total catch per sample)

# Index which combines year mo station, and year mo station volumefiltered

# Columns for sample variables including selected environmental variables

# One column for each taxon

# Data as catch per trawl OR CPUE in # per 10,000 m3

# Missing values set to 0 and ALL samples included that meet the selection criteria

# Flat file (default for length)

# Data include catch per trawl, proportional catch, and CPUE as above.

# Zeros omitted and only the first taxon selected.

#

# Control variables with defaults:

# cpue=F Calculate CPUE or leave data as catch per trawl

# output="total" TOtal fish or by "length"

# outFormat" "Table" if total , "flatfile" if lengths

#

# Selection variables (all these default to ALL)

# spcode: numeric MWT species codes

# gen: Character vector of genus names (must be spelled correctly!)

# fam: Character vector of family names (must be spelled correctly!)

# years: Range of years to include

# surveys: Range of surveys (~months) to include

# stations:Stations to include (these are numeric - nnn:nnn is OK)

# sals: Range of SURFACE salinities to include

#

{

require(dplyr)

require(reshape2)

load(paste0(path, fname))

databaseName <- substring(fname, 1, regexpr("rdata", tolower(fname))-2)

cat("Using source file:", databaseName, "\n")

database <- get(databaseName)

listInd <- match(listNames, names(database))

output <- tolower(substring(output, 1,1))

outFormat <- tolower(substring(outFormat, 1,1))

for (i in 1:length(listNames)) assign (listNames[i], database[[listInd[i]]])

output <- tolower(substring(output[1],1,1))

if (is.na(match(output, c("l","t"))))

{

cat("\*\*\*\*\*\*\*\*\*\*\*\*\* Output", output, "incorrect, changed to total\n")

output <- "t"

}

sample <- selectField(sample, years=years, mos=mos, sals=sals)

# Select sample variables to include in output - either selected, minimum, or all

if (substring(tolower(include.fld[1]),1,1) == "m")

{

sample <- sample[, c("samplerowid", "index","year","survey","station", "volM3")]

} else if(tolower(include.fld[1]) != "all") # If "all" include all columns of sample

{

include.fld <- unique(c("samplerowid", "index",include.fld)) # in case samplerowid not in list

fldMatch <- match (include.fld, names(sample))

fldNomatch <- which(is.na(fldMatch))

if (length (fldNomatch) > 0) # Check for incorrect variable names

{

cat("Dropping unknown fields:", include.fld[fldNomatch], "\n" )

include.fld <- include.fld[-fldNomatch]

}

sample <- sample[, c(include.fld,"volM3")]

}

# Select fish taxa. Restrict to the first if length

spcodeDF <- selectFish(program="MWT", spcode=spcode, fam=fam,

gen=gen)[,1]

if (outFormat=="f" & length(spcodeDF) > 1)

{

spcodeDF <- spcodeDF[1]

cat("First taxon only included for flat file output:\n")

}

ntaxa <-length(spcodeDF)

alphaCode <- organisms$Code.All[match(spcodeDF, organisms$Code.MWT)]

cat("Analysis on: ", alphaCode, "\n")

# Subset the fish length and catch tables to include only selected samples and species

# Before output the tables are expanded to include all taxa

cat(names(sample), "\n")

cat(dim(sample[, c("samplerowid","index")]), "\n")

fcatch <- inner\_join( sample[, c("samplerowid","index")],

catch[match(catch$organismcode, spcodeDF, nomatch=0)>0,],

by="samplerowid")

# For length analysis

if (output=="l")

{

fcatch <- inner\_join( fcatch, length,by="catchrowid")

fcatch <- dplyr::select(fcatch, -lengthrowid)

}

# Drop unneeded rows

fcatch <- dplyr::select(fcatch, -samplerowid, -catchrowid)

# Branch depending on whether output is a table or a flat file

# Table is OK for total fish by taxon but not lengths.

# Also only catch is converted to CPUE if required

#

if (outFormat=="t" & output!="l")

{

block <- dcast(fcatch, index ~ organismcode,

value.var="catch",

fun.aggregate=sum, fill=0)

out <- left\_join(sample,block, "index")

cols <- (ncol(sample)) + (2:ncol(block))-1

cat("Saving", nrow(out), "rows and ", ncol(block)-1, "taxa in table format")

names(out)[cols] <- alphaCode

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

if (cpue)

{

out[,j] <- out[,j] \* 10000 / out$volM3

}

}

if (cpue)

{ cat(", as CPUE\n") }

else { cat (", as catch per trawl\n") }

} else # Here to output flat file with either catch or catch and length

{

out <- inner\_join(sample, fcatch, "index")

cols <- na.omit(match(c("catch","frequency"), names(out)))

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

}

if (cpue)

{

out$cpue <- out$catch \* 10000 / out$volM3

}

}

dplyr::select(out, -samplerowid)

}

getSLS <- function (fname="SLS",

path="C:/1Current/R-files/1R-projects/SLSDatabase/",

listNames=c("sample","catch","length","longfinYolk", "fishcodes","stations", "dates"),

include.fld = c("sampleID","year", "mo","survey","julday", "station",

"time", "secchi","turbidity", "sal.surf","sal.bott", "temp"),

cpue=F,

output=c("total", "length", "yolksac"),

outFormat = switch(substring(output,1,1)[1], t="table", l="flatfile", y="flatfile"),

keepList = F,

slscode = character(0),

fam = character(0), gen=character(0),

years = integer(0),mos = integer(0),

stations=integer(0), sals = numeric(0) ) # End inputs

# Function to query Smelt Larva Survey database for selected taxa, samples, and output

# Data reported as total catch, catch per length,

# or for longfin smelt only, length and presence of yolksac or oil droplet

# Length is for one species only, and the program will select the first if >1

# Required files:

# Data source:SLS.Rdata, produced by: R import SLS database 2020.R

# Functions selectField and selectFish (in wimsFunctions.txt)

# Output formats:

# Table (default for total catch per sample)

# sampleID which combines year mo station, and year mo station and volume filtered

# Columns for sample variables including selected environmental variables

# One column for each taxon

# Data as catch per trawl OR CPUE in # per 1,000 m3

# Missing values set to 0 and ALL samples included that meet the selection criteria

# Flat file (default for length)

# Data include catch per trawl and fraction of total catch in sample, and CPUE as above.

# Zeros omitted and only the first taxon selected.

# For yolksac, longfin smelt only (other produces an error) and presence of yolksac is a column.

# Flat file with 2 columns for yolksac (yes or no yolksac)

#

# Control variables with defaults:

# cpue=F Calculate CPUE or leave data as catch per trawl

# output="total" TOtal fish or by "length" or "yolksac" (first letter, any case)

# outFormat" "Table" if total , "flatfile" if lengths (")

# includeFld "all" for all vars, "min" for minimal to ID each sample, otherwise list them

#

# Selection variables (all these default to ALL)

# slscode: numeric SLS species codes

# gen: Character vector of genus names (must be spelled correctly!)

# fam: Character vector of family names (must be spelled correctly!)

# years: Range of years to include

# surveys: Range of surveys (~months) to include

# stations:Stations to include (these are numeric - nnn:nnn is OK)

# sals: Range of SURFACE salinities to include

#

{

require(dplyr)

require(reshape2)

if (!exists(fname))

load(paste0(path, fname, ".Rdata"), envir=.GlobalEnv)

cat("Using source file:", fname, ".Rdata\n")

database <- get(fname)

listInd <- match(listNames, names(database))

for (i in 1:length(listNames)) assign (listNames[i], database[[listInd[i]]])

output <- tolower(substring(output[1],1,1))

outFormat <- tolower(substring(outFormat, 1,1))

if (is.na(match(output, c("l","t", "y"))))

{

cat("\*\*\*\*\*\*\*\*\*\*\*\*\* Output", output, "incorrect, changed to total\n")

output <- "t"

}

if (output=="y") slscode <- 2

sample <- selectField(merge(dates, sample,1), years=years, mos=mos, sals=sals)

# Select sample variables to include in output - either selected, minimum, or all

if (substring(tolower(include.fld[1]),1,1) == "m")

{

sample <- sample[, c("sampleID", "year","survey","station", "vfm3")]

} else if(tolower(include.fld[1]) != "all")

{

include.fld <- unique(c("sampleID", include.fld)) # in case sample ID not in list

fldMatch <- match (include.fld, names(sample))

fldNomatch <- which(is.na(fldMatch))

if (length (fldNomatch) > 0) # Check for incorrect variable names

{

cat("Dropping unknown fields:", include.fld[fldNomatch], "\n" )

include.fld <- include.fld[-fldNomatch]

}

sample <- sample[, c(include.fld,"vfm3")]

}

# Select fish taxa; drop those not in SLS database. Restrict to the first if length

spcodeDF <- selectFish(program="20", spcode=slscode, fam=fam, gen=gen)

spcodeDF <- dplyr::filter(spcodeDF, match(code, fishcodes$fishcode, nomatch=0)>0)

spcodeDF <- arrange(spcodeDF, code)

if (outFormat=="f" & nrow(spcodeDF) > 1)

{

spcodeDF <- spcodeDF[1,]

cat("First taxon only included for flat file output:\n")

}

ntaxa <- nrow(spcodeDF)

if (ntaxa==0)

{ cat("No taxa selected\n") # This ends the function

} else

{

alphaCode <- spcodeDF[,2]

spcode <- spcodeDF[,1]

cat("Analysis on: ", alphaCode, "\n")

# Subset the fish length and catch tables to include only selected samples and species

# Before output the tables are expanded to include all samples

fcatch <- catch[match(catch$fishcode, spcode, nomatch=0)>0,]

# For length analysis

if (output=="l")

{

fcatch <- inner\_join( fcatch, length,by="catchID", suffix=c("",".y"))

fcatch <- dplyr::select(fcatch, -fishcode.y, -catch.y)

}

# For length analysis with yolksac (LFS only)

if (output=="y")

{

fcatch <- inner\_join( fcatch, longfinYolk,by="catchID", suffix=c("",".y"))

fcatch <- dplyr::select(fcatch, -fishcode, -catch.y)

if (nrow(fcatch)==0)

cat("Yolksac produces no data; check year > 2010\n")

}

# Drop unneeded columns

fcatch <- dplyr::select(fcatch, -catchID)

# Branch depending on whether output is a table or a flat file

# Table is OK for total fish by taxon but not lengths.

# Also only catch is converted to CPUE if required

#

if (outFormat=="t" & output=="t")

{

block <- dcast(fcatch, sampleID ~ fishcode,

value.var="catch",

fun.aggregate=sum, fill=0)

out <- left\_join(sample,block, "sampleID")

cols <- (ncol(sample)) + (2:ncol(block))-1

cat("Saving", nrow(out), "rows and", length(spcode), "taxa in table format")

names(out)[cols] <- alphaCode

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

if (cpue)

{

out[,j] <- out[,j] \* 1000 / out$vfm3

}

}

if (cpue)

{ cat(", as CPUE = catch per 1000 m3\n")

} else {

cat (", as catch per trawl\n")

}

} else { # Here to output flat file with either catch or catch and length

out <- inner\_join(sample, fcatch, "sampleID")

cat("Saving", nrow(out), "rows and", length(spcode),

"taxa in flat file format\nNOTE: lengths on <= 50 fish in some samples\n")

cols <- na.omit(match(c("catch","nfish"), names(out)))

for (j in cols)

{

out[,j] <- ifelse(is.na(out[,j]), 0, out[,j])

}

if (cpue)

{

out$cpue <- out$catch \* 10000 / out$vfm3

}

}

if (keepList){

list(output=out, SLS=database)

} else {

out

}

}} # END (if ntaxa) SELECTION AND FUNCTION

getzoop20 <- function( path="C:/1Current/R-files/1R-projects/20mmDatabase/", file="allZoop20mm.2018",

include.fld = c("year", "survey","julday", "station", "temp", "sal.surf","sal.bottom"),

zoocode=integer(0), spcode = character(0), taxon = character(0), order = character(0), stage = character(0),

years = integer(0),survs=integer(0), juldays = integer(0), sals = numeric(0), stas=numeric(0),

tot = F)

{

# Function to get zooplankton abundance and combine with field data from 20mm survey data

# Input:

# include.fld Character vector of variable names from fld and dates to include (as well as sample.id, or "all")

# zoocode: Numeric zoo code used in

# spcode: character vector of species names from sp$spcode

# taxon: character vector of taxon names form neozoo.spp$taxon

# stage: character vector of life stages, if any

# years: Range of years to include (or all if missing)

# juldays: Range of julian days to include (or all if missing)

# sals: Range of SURFACE salinities to include (or all if missing)

# If sals is not missing, samples with missing S values are excluded

# tot TRUE to sum the plankton abundance and give a single value

# Output:

# Data frame containing all samples in the included years, selected data from fld,

# and abundance data for the groups specified above. Note that the selection criteria

# are applied in sequence to the species list. Any variables not specified are taken to be "all."

#

# Created 31 Dec 2017

#-------------------------------------

#require(dplyr)

cat("Using source file:", file, "\n")

if (!exists(file))

load(file=paste0(path, file, ".Rdata"), envir=.GlobalEnv)

sp <- get(file)$sp.xref

surv <- get(file)$survey

tow <- get(file)$zooTow

fld <- get(file)$station

sta <- get(file)$staLatLong

zn <- get(file)$zooCountAbun

# Set up the list of taxa in sp

v <-c("zoocode","spcode", "taxon", "order", "stage")

for (i in 1:length(v))

{

gv <- get(v[i])

if (length(gv) > 0)

{

sp <- sp[match(sp[, match(v[i], names(sp))], gv, nomatch = 0) > 0, ]

}

}

if (nrow(sp) == 0)

stop("Combination of inputs yields no output")

nspecies <- nrow(sp)

# cat(nspecies, "Included: ",sp$spcode, "\n" )

# Set up the sample and field data for the selected variables and range of years, etc

x <- inner\_join(surv, fld, by="surveyid")

x$julday <- julyear(x$dateMed)

if(length(years) > 0)

x <- x[x$year >= min(years) & x$year <= max(years), ]

if(length(survs) > 0)

x <- x[x$survey >= min(survs) & x$survey <= max(survs), ]

if(length(juldays) > 0)

x <- x[x$julday >= min(juldays) & x$julday <= max(juldays), ]

x$sal.surf <- sal.ec(x$topec)

x$sal.bottom <- sal.ec(x$bottomec)

if(length(sals) > 0)

{

x <- x[!is.na(x$sal.surf), ]

x <- x[x$sal.surf >= min(sals) & x$sal.surf <= max(sals), ]

}

if(length(stas) > 0)

{

if (length(stas) == 2)

{x <- x[x$station >= min(stas) & x$station <= max(stas), ]}

else

{x <- x[match(x$station, stas, nomatch=0)>0,]}

}

if (all(include.fld != "all"))

{

if (length(include.fld) > 0 && include.fld[1] != "")

{x <- x[, c("stationid", include.fld)]}

else

{x <- x[, "stationid"]}

}

x$towid <- tow$towid[match(x$stationid, tow$stationid)]

x <- x[!is.na(x$towid),] # Eliminate field data without tow

nx <- ncol(x)

zn1 <- zn[match(zn$towid, x$towid, nomatch = 0) > 0, c("towid","zoocode","abun") ]

zn2 <- zn1[match(zn1$zoocode, sp$zoocode, nomatch=0)>0,]

zn2$spcode <- sp$spcode[match(zn2$zoocode, sp$zoocode)]

if (tot) # To sum all. NOTE that this does not indicate what is being summed!

{

zn2 <- Bquick(zn2[,c("towid","abun")], 1, sum)

zn2$spcode <- rep("total", nrow(zn2))

nspecies <- 1

}

zb <- blockdat(zn2$abun, zn2$towid, zn2$spcode, fconv=T)

names(zb)[1] <- "towid"

# cat(names(zb), "\n")

if (nspecies > 1) # Kluge to get all columns in the data frame even if all zeros

{

zfill <- makedf(c("towid",sp$spcode), zb$towid)

zb <- merge (zb, zfill, by="towid", suffixes=c("",".new"))

zb <- zb[,names(zfill)]

}

zout <- left\_join(x, zb, by="towid")

cols <- ncol(x) + 1:nspecies

for(i in cols)

zout[,i] <- ifelse(is.na(zout[,i]), 0, zout[,i])

zout <- zout[,-1] # Drop towid

cat("Species included:", sp$spcode, c("",": Total Only")[tot+1], "\n" )

zout

} # END getzoop20

getzoopAll <- function( file = "allZoopAbunBM201908", fname = "allZoopAbunBM201908.Rdata",

path ="C:/1Current/R-files/1R-projects/ZoopDatabase/Rdata/",

include.fld = c("year", "julday", "station", "temp", "sal.surf","sal.bott"),

spcode = character(0), taxon = character(0), order = character(0),

stage = character(0), years = integer(0),juldays = integer(0), sals = numeric(0),

tot = F, biomass=F)

{

# Function to get zooplankton abundance or biomass from combined zooplankton file

# produced by Zooplankton Combine all data Revised 16 May 2018.R

# Updated 8 Oct 2019 to include data through 2018 (20mm and NZ only) and issue a warning about pdiapfor in 20mm data

# Input:

# include.fld Character vector of variable names from fld and dates to include (as well as rowIndex, or "all")

# spcode: character vector of species names from sp$spcode

# taxon: character vector of taxon names form neozoo.spp$taxon

# stage: character vector of life stages, if any

# years: Range of years to include (or all if missing)

# juldays: Range of julian days to include (or all if missing)

# sals: Range of SURFACE salinities to include (or all if missing)

# If sals is not missing, samples with missing S values are excluded

# tot TRUE to sum the plankton abundance or biomass and give a single value

# Added May 2010:

# drop.missing T means drop if cbcode or pumpcode are F, netpump="", and net and pump taxa included

# trimNames Drop the ".n",".p",".b" from names (generally should not)

# Added November 2012:

# The following are generally not changed:

# sp Data frame of species identities, plus which sample they come from (N, P, or B)

# zp Zooplankton pump data in table format by sample index

# zn Zooplankton net data in table format by sample index

# fld Field data; first column is a sample index

# dates Median date of surveys

# Output:

# Data frame containing all samples in the included years, selected data from fld,

# and abundance data for the groups specified above. Note that the selection criteria

# are applied in sequence to the species list. Any variables not specified are taken to be "all."

#-------------------------------------

cat("Using source file:", file, "\n")

fn <- paste0(path, fname)

cat(fn,"\n")

load(file=fn, envir=.GlobalEnv)

sp <- get(file)$taxonomy

fld <- get(file)$header

sta <- get(file)$stationsAll

if (biomass){

zp <- get(file)$allBugsmgCm3

}

else

{

zp <- get(file)$Abundance

}

# Set up the list of taxa in sp

v <-c("spcode", "taxon", "order", "stage")

for (i in 1:length(v))

{

gv <- get(v[i])

if (length(gv) > 0)

{

sp <- sp[match(sp[, match(v[i], names(sp))], gv, nomatch = 0) > 0, ]

}

}

if (nrow(sp) == 0)

stop("Combination of inputs yields no output")

# Set up the sample and field data for the selected variables and range of years

x <- fld

if(length(years) > 0)

x <- x[x$year >= min(years) & x$year <= max(years), ]

if(length(juldays) > 0)

x <- x[x$julday >= min(juldays) & x$julday <= max(juldays), ]

if(length(sals) > 0)

{

x <- x[!is.na(x$sal.surf), ]

x <- x[x$sal.surf >= min(sals) & x$sal.surf <= max(sals), ]

}

if (all(include.fld != "all"))

{

if(length(include.fld) > 0 && include.fld[1] != "")

x <- x[, c("rowIndex", "source", include.fld)]

else x <- x[, c("rowIndex", "source")]

}

# Get the zooplankton data by column number (v) corresponding to the taxa incuded,

# and sum or block the data

sid <- x$rowIndex

zp <- zp[match(zp$rowIndex, sid, nomatch = 0) > 0, c("rowIndex", sp$spcode) ]

if (tot)

{

zp$total <- rowSums(zp[,-1])

zp <- zp[,c(1, ncol(zp))]

}

xx <- merge(x, zp, by="rowIndex")

txt <- paste(sp$spcode, collapse=c(","))

if (tot) {txt <- paste("Total of:", txt)}

if (biomass) {txt <- paste ("Biomass -", txt)}

else {txt <- paste ("Abundance -", txt)}

cat("Saving", ncol(xx), "columns by", nrow(xx), "rows\n", txt, "\n" )

if (any(names(xx)=="pdiapfor") & any(xx$source=="20mm" & xx$year< 2006) )

cat("CAUTION: All pdiapspp recoded to pdiapfor in 20mm data before 2006\n")

xx

}

getzoopM <- function( file = "zoopMWTTNS",

include.fld = c("year", "julday", "station", "temp", "sal.surf","sal.bott"),

index = integer(0), spcode = character(0), taxon = character(0), order = character(0),

stage = character(0), years = integer(0),juldays = integer(0), sals = numeric(0),

tot = F)

{

# Function to get MWT-TNS zooplankton abundance and combine with field data

# Modified from getzoopR

# Input:

# include.fld Character vector of variable names from fld and dates to include (as well as sample.id, or "all")

# index: Integer vector of min and max values for sp$index.spp - all in range included

# spcode: character vector of species names from sp$spcode

# taxon: character vector of taxon names form neozoo.spp$taxon

# stage: character vector of life stages, if any

# years: Range of years to include (or all if missing)

# juldays: Range of julian days to include (or all if missing)

# sals: Range of SURFACE salinities to include (or all if missing)

# If sals is not missing, samples with missing S values are excluded

# tot TRUE to sum the plankton abundance and give a single value

# sp Data frame of species identities, plus which sample they come from (N, P, or B)

# zabun Zooplankton net data in table format by sample index

# fld Field data; first column is a sample index

# dates Median date of surveys

# Output:

# Data frame containing all samples in the included years, selected data from fld,

# and abundance data for the groups specified above. Note that the selection criteria

# are applied in sequence to the species list. Any variables not specified are taken to be "all."

# The program now returns only non-missing data if netpump =="p" or "n", ="

# but will include missing if both pump and net data are returned and netpump=="",

# unless drop.missing=T

# Ported into R in 2015.

# Added June 2016: envir in load file set so the zooplankton database is available (it is big!)

#-------------------------------------

cat("Using source file:", file, "\n")

if (!exists(file))

load(file="C:\\1Current\\R-files\\1R-projects\\ZoopDatabase\\RData\\zoopMWTTNS.Rdata",

envir=.GlobalEnv)

sp <- get(file)$sp.xref

zabun <- get(file)$abun

sp <- sp[match(sp$spcode, names(zabun)[-1], nomatch=0)>0,] # NOT all in sp.xref

fld <- get(file)$samples

sta <- get(file)$sta.xref

dt <- get(file)$dates

# Set up the list of taxa in sp

if(length(index) > 0)

sp <- sp[sp$index.spp >= min(index) & sp$index.spp <= max(index), ]

v <-c("spcode", "taxon", "order", "stage")

for (i in 1:length(v))

{

gv <- get(v[i])

if (length(gv) > 0)

{

sp <- sp[match(sp[, match(v[i], names(sp))], gv, nomatch = 0) > 0, ]

}

}

if (nrow(sp) == 0)

stop("Combination of inputs yields no output")

zabun <- zabun[,c("sample.id",sp$spcode)]

if (tot)

{

zabun$total <- rowSums(zabun[,-1])

zabun <- zabun[,c(1, ncol(zabun))]

}

# Set up the sample and field data for the selected variables and range of years

x <- merge(dt, fld, by="surveycode", all.x = T)

if(length(years) > 0)

x <- x[x$year >= min(years) & x$year <= max(years), ]

if(length(juldays) > 0)

x <- x[x$julday >= min(juldays) & x$julday <= max(juldays), ]

if(length(sals) > 0)

{

x <- x[!is.na(x$sal.surf), ]

x <- x[x$sal.surf >= min(sals) & x$sal.surf <= max(sals), ]

}

if (all(include.fld != "all"))

{

if(length(include.fld) > 0 && include.fld[1] != "")

x <- x[, c("sample.id", include.fld)]

else x <- x[, c("sample.id", "surveycode")]

}

sid <- x$sample.id

zabun <- zabun[match(zabun$sample.id, sid, nomatch = 0) > 0, ]

# Get the zooplankton data by column number (v) corresponding to the taxa incuded,

# and sum or block the data

xx <- merge(x, zabun, by="sample.id")

cat("Saving", ncol(xx), "columns by", nrow(xx), "rows\n", names(zabun)[-1] )

xx

}

getzoopR <- function( file="neozoo.2018",

include.fld = c("year", "julday", "station", "temp", "chl", "sal.surf","sal.wc"),

index = integer(0), spcode = character(0), taxon = character(0), order = character(0),

stage = character(0), years = integer(0),juldays = integer(0), sals = numeric(0),

tot = F, netpump = "", drop.missing = F, trimNames = F)

{

# Function to get zooplankton abundance and combine with field data - revised to get data from a list

# Input:

# include.fld Character vector of variable names from fld and dates to include (as well as sample.id, or "all")

# index: Integer vector of min and max values for sp$index.spp - all in range included

# spcode: character vector of species names from sp$spcode

# taxon: character vector of taxon names form neozoo.spp$taxon

# stage: character vector of life stages, if any

# years: Range of years to include (or all if missing)

# juldays: Range of julian days to include (or all if missing)

# sals: Range of SURFACE salinities to include (or all if missing)

# If sals is not missing, samples with missing S values are excluded

# tot TRUE to sum the plankton abundance and give a single value

# netpump "net" or "pump" to get only values from those data sets (otherwise determined from sp, below)

# Added May 2010:

# drop.missing T means drop if cbcode or pumpcode are F, netpump="", and net and pump taxa included

# trimNames Drop the ".n",".p",".b" from names (generally should not)

# Added November 2012:

# The following are generally not changed:

# sp Data frame of species identities, plus which sample they come from (N, P, or B)

# zp Zooplankton pump data in table format by sample index

# zn Zooplankton net data in table format by sample index

# fld Field data; first column is a sample index

# dates Median date of surveys

# Output:

# Data frame containing all samples in the included years, selected data from fld,

# and abundance data for the groups specified above. Note that the selection criteria

# are applied in sequence to the species list. Any variables not specified are taken to be "all."

# The program now returns only non-missing data if netpump =="p" or "n", ="

# but will include missing if both pump and net data are returned and netpump=="",

# unless drop.missing=T

# Ported into R in 2015.

# Added June 2016: envir in load file set so the zooplankton database is available (it is big!)

#-------------------------------------

cat("Using source file:", file, "\n")

if (!exists(file))

load(paste0("C:/1Current/R-files/1R-projects/ZoopDatabase/Rdata/",file,".Rdata"),

envir=.GlobalEnv)

sp <- get(file)$sp.xref

zp <- get(file)$pump

zn <- get(file)$net

fld <- get(file)$samples

sta <- get(file)$sta.xref

dt <- get(file)$dates

# Set up the list of taxa in sp

if(length(index) > 0)

sp <- sp[sp$index.spp >= min(index) & sp$index.spp <= max(index), ]

v <-c("spcode", "taxon", "order", "stage")

for (i in 1:length(v))

{

gv <- get(v[i])

if (length(gv) > 0)

{

sp <- sp[match(sp[, match(v[i], names(sp))], gv, nomatch = 0) > 0, ]

}

}

if (nrow(sp) == 0)

stop("Combination of inputs yields no output")

# Set up the sample and field data for the selected variables and range of years

x <- merge(dt, fld[, c(2, 1, 3, 4:ncol(fld))], 1, all.x = T)

if(length(years) > 0)

x <- x[x$year >= min(years) & x$year <= max(years), ]

if(length(juldays) > 0)

x <- x[x$julday >= min(juldays) & x$julday <= max(juldays), ]

if(length(sals) > 0)

{

x <- x[!is.na(x$sal.surf), ]

x <- x[x$sal.surf >= min(sals) & x$sal.surf <= max(sals), ]

}

if (all(include.fld != "all"))

{

if(length(include.fld) > 0 && include.fld[1] != "")

x <- x[, c("sample.id", "cbcode", "pumpcode", include.fld)]

else x <- x[, c("sample.id", "cbcode", "pumpcode")]

}

sid <- x$sample.id

zp <- zp[match(zp$sample.id, sid, nomatch = 0) > 0, ]

zn <- zn[match(zn$sample.id, sid, nomatch = 0) > 0, ]

# Get the zooplankton data by column number (v) corresponding to the taxa incuded,

# and sum or block the data

nps <- c("N", "P")

if(netpump == "")

np <- nps

else

{np <- toupper(substring(netpump, 1, 1))

if (np[1] != "N" & np != "P" ) stop ("netpump must be either n or p or leave blank")

}

nmz <- c("zn","zp")

k <- 1

for(i in 1:length(np)) {

j <- match(np[i], nps)

vv <- sp[sp$source == np[i] | sp$source == "B", "spcode"]

if(length(vv) > 0) {

vvi <- vv[match(vv, names(get(nmz[j])), nomatch=0)>0] # Some not in database

zi <- get(nmz[j])[, c("sample.id", vvi)]

names(zi)[-1] <- paste(names(zi)[-1], tolower(np[i]), sep = ".")

if(k == 1)

zz <- zi

else zz <- merge(zz, zi, 1, all = T)

k <- k + 1

}

}

names(zz)[1] <- "zzzz"

# Need to keep in case only one spcode to make zz stay a data frame

zz <- zz[, sort(names(zz))]

nc <- ncol(zz)

if(trimNames | length(np)==1)

names(zz) <- right(names(zz), -2)

zz1 <- zz[, c(ncol(zz), 1:(ncol(zz) - 1))]

xx <- merge(x, zz1, 1)

if(drop.missing & netpump == "")

xx <- xx[xx$cbcode & xx$pumpcode, ]

xx

}

sal.ec <- function (ec, temp=-100)

{

# Calculates salinity from EC, assumed to be at 25C, or cond. at given temp

# ec is a vector of ec values in either mS/cm or uS/cm

# Converts EC in mS/cm to salinity.

# Algorithm taken from Matlab routines by CSIRO, based on a conductivity ratio Rt

# which is the ratio of sample conductivity to conductivity at S<-25, T<-15

# SEAWATER Library

# Version 2.0.1 22-Apr-1998

# \* Phil.Morgan@marine.csiro.au \*

# The conductivity ratio of the sample at S, 25C, 0 pressure is calculated first from

# the conductivity at 25, 15, 0

# Using the algorithm in program

# Algorithm: Salinity <- f ( Rt )

# Rt = COnd (Unknown S, T=25, P=0) / Cond (S=35, T=15, P=0)

# = Cond (S, 25, 0) / ( Cond (35, 15, 0) \* ( Cond (35,25,0) / Cond (35,15,0) ) )

#

#

# Extension for S < 2 due to Hill (1978) added May 2007 - from Alan Jassby

# First step: cond (35,15,0) is "given" in sw\_c3515.m in above program

c3515 <- 42.914

# Next rt is calculated as C(35,T,0)/C(35,15,0) from program sw\_salrt.m

# Eqn (3) p.7 Unesco.

if (length(temp)>1 & length(temp) !=length(ec)) stop ("Inputs must be same length or temp must be length 1")

tt <- ifelse (temp < -10, 25, temp)

c0 <- 0.6766097

c1 <- 2.00564e-2

c2 <- 1.104259e-4

c3 <- -6.9698e-7

c4 <- 1.0031e-9

rt <- c0 + (c1 + (c2 + (c3 + c4 \* tt) \* tt) \* tt) \* tt # Cond ratio at 25C to that at 15 C

# Then the ratio Rt is calculated (the conductivity ratio of the sample)

# and salinity is calculated from sw\_sals.m

if(max(na.omit(ec)) > 100) ec <- ec/1000

Rt <- ec / ( c3515 \* rt )

a0 <- 0.0080

a1 <- -0.1692

a2 <- 25.3851

a3 <- 14.0941

a4 <- -7.0261

a5 <- 2.7081

b0 <- 0.0005

b1 <- -0.0056

b2 <- -0.0066

b3 <- -0.0375

b4 <- 0.0636

b5 <- -0.0144

k <- 0.0162

x <- 400\*Rt

y <- 100\*Rt

Rtx <- sqrt(Rt)

delT <- tt - 15

ft <- delT / (1 + k\*delT)

delS <- ft \* ( b0 + (b1 + (b2+ (b3 + (b4 + b5 \* Rtx) \* Rtx) \* Rtx) \* Rtx) \* Rtx)

pss78 <- a0 + (a1 + (a2 + (a3 + (a4 + a5 \* Rtx) \* Rtx) \* Rtx) \* Rtx) \* Rtx

pss78 <- pss78 + delS

S <- ifelse(pss78 <= 2, pss78, pss78 - a0/(1+1.5 \* x + x^2) - ft\*b0/(1 + y^.5 + y^1.5))

S

}

# ec <- c(0:10, seq(15,55,by=5))

# sal <- c(0.012, .493, 1.017, 1.559, 2.114, 2.68, 3.255, 3.838, 4.428, 5.025, 5.627, 8.718, 11.917, 15.205, 18.572, 22.013, 25.522, 29.098, 32.738, 36.441)

# salec <- sal.ec (ec)

# summary(round(salec,3)-sal)

selectField <- function (fld, indices = integer(0),years = integer(0),mos = integer(0),

surveys=integer(0), stations=integer(0), sals = numeric(0))

{

# For use in queries of fish databases

# See one of them for parameter names

# The field data frame fld must contain all that are selected,

# including at least one salinity value if there is selection on salinity

# NOTE 24 APril 2018 fixed error in specification of salinity range

if(length(indices) > 0)

fld <- fld[match(fld$index, indices, nomatch=0)>0, ]

if(length(years) > 0)

fld <- fld[fld$year >= min(years) & fld$year <= max(years), ]

if(length(mos) > 0)

fld <- fld[fld$mo >= min(mos) & fld$mo <= max(mos), ]

if(length(surveys) > 0)

fld <- fld[fld$survey >= min(surveys) & fld$survey <= max(surveys), ]

if(length(stations) > 0)

fld <- fld[match(fld$station,stations, nomatch=0)>0, ]

if (nrow(fld) == 0) stop ("Inputs yield no samples")

if(length(sals) > 0)

{

ind <- which(regexpr("sal", tolower(names(fld)))>0)

if (length(ind) > 1)

ind <- ind[regexpr("surf", tolower(names(fld)[ind]))>0]

nmiss <- which(is.na( fld[,ind]))

if (length(nmiss)>0)

{

cat(length(nmiss), "Missing values for surface salinity deleted\n")

fld <- fld[-nmiss, ]

}

fld <- fld[fld[,ind] >= min(sals) & fld[,ind] <= max(sals), ]

}

fld

}

selectFish <- function(program="Bay", spcode=character(0),

fam=character(0), gen=character(0))

{

# For use in queries of fish databases

# Inputs

# program Sampling program for species codes: Bay, MWT, TNS, 20

# spcode Species code from the list of codes for this program

# Character for Bay Study, numeric for others

# fam, gen Family or genus, can be abbreviated

# sp Usually fishCodesAll, which contains all of the codes

# catchCodes

# Spcies codes are selected first, else family and genus

load("C:/1Current/R-files/1R-projects/FishDatabase/fishCodesAllPrograms.Rdata")

sp <- fishCodesAll

codeColumn <- paste0("Code.", program)

column <- match (tolower(codeColumn), tolower(names(sp)))

if (is.na(column))

stop("In selectFish, program doesn't match")

if (length(spcode) > 0 && tolower(spcode) != "all")

{

spmatch <- match(spcode, sp[,column], nomatch=0)

}

else

if (length(fam) > 0)

{

fam <- tolower(fam)

nc <- nchar(fam)

spmatch <- which(match(tolower(substring(sp$Family, 1, nc)), fam, nomatch=0)>0)

fams <- unique(sp$Family[spmatch])

if (length (fams) > 1) cat("Multiple families selected\n", fams, "\n\n")

} else

if (length(gen) > 0)

{

gen <- tolower(gen)

nc <- nchar(gen)

spmatch <- which(match(tolower(substring(sp$Genus, 1, nc)), gen, nomatch=0)>0)

gens <- unique(sp$Genus[spmatch])

if (length (gens) > 1) cat("Multiple families selected\n", gens, "\n\n")

}

else

{

spmatch <- 1:nrow(sp)

}

spcodeNew <- sp[spmatch, c(codeColumn,"Code.All")]

spcodeNew <- spcodeNew [!is.na(spcodeNew[,1]),]

spcodeNew <- spcodeNew [spcodeNew[,1] != "",]

names(spcodeNew) <- c("code","abbrev")

# cat("Species codes included: \n", spcodeNew$abbrev, "\n")

spcodeNew

}

#&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

wateryear <- function (year, mo=NULL, julday=NULL)

{

# Determines water year from calendar year and either

# month if available or julian day

if (is.null(mo))

{

mo <- rep(NA, length(year))

if(length(julday)==1) julday <- rep(julday, length(year))

for (yr in unique(year))

{

mo[year==yr] <- month.day.year(julday[year==yr],

origin. = c(12, 31, yr - 1))$mo

}}

year + 1 \* (mo >=10)

}

fishLenCatch <- function (xcatch, xlen, lenClip = 1:1000)

{

# Combines length and total catch data from fish surveys

# to produce a data frame of estimated catch by length

# including the ability to confine the range of lenghts

# placing outliers in the closer of the largest or smallest bin.

#

# INputs

# xcatch Data frame of catch with 1 row per sample, including zeros

# Required columns

# index (unique identifier)

# catch (number of fish caught)

# xlen Data frame as flat file, by sample, length, and count,no zeros

# Required columns

# index (unique identifier)

# length (1mm length increment)

# nfish (number of fish in increment)

# lenClip Clip data to these length categories (mm)

# All others are lumped with the first and last category

# OUTPUTS

# Data frame with

# index (from xcatch and xlen)

# catch (total from xcatch)

# lencat (length from xlen truncated to lenClip

# nfish (from xlen) number counted in each bin

# nfish.tot TOtal in all length bins (=catch if all are measured)

# totByLen Total expanded to the catch (i.e., nfish \* catch / nfish.tot)

if (any(duplicated(xcatch[,1]) ) )

stop ("Indices of catch data frame must be unique\n")

nmIndex <- names(xcatch)[1]

names(xcatch)[1]<- names(xlen)[1] <- "index"

xlen$lencat <- pmax(lenClip[1], pmin(max(lenClip), xlen$length))

xlen <- Bquick(select(xlen,index,lencat,nfish), 2, sum)

framework <- expand.grid(index=unique(xcatch$index), lencat=lenClip)

xframe <- merge(framework, xlen, c("index", "lencat"), all.x=T)

xframe <- na.to.value(xframe, 3:ncol(xframe))

xlentot <- Bquick(select(xframe,index, nfish), 1, sum)

xframeTot <- merge(xframe, xlentot, 1, suffixes =c("",".tot") )

xlenc <- merge(xcatch, xframeTot, 1)

xlenc$totByLen <- ifelse(xlenc$catch==0, 0,

xlenc$catch \* xlenc$nfish / xlenc$nfish.tot)

names(xlenc)[1] <- nmIndex

xlenc

}

pseudoEgg <- function (temp, egg=T)

# Calculates egg development time from temperature

# or relative development time for any stage

{

if (egg)

22.7 \* exp(-0.121 \* (temp))

else

exp(-0.121 \* (temp-22))

}

# pseudoEgg(10:25, T)

# pseudoEgg(10:25, F)

oxygenSat <- function (temp, o2=NULL, patm =1)

{

# Calculates oxygen concentration at saturation

# From http://www.waterontheweb.org/under/waterquality/oxygen.html

# or percent saturation

# Inputs:

# temp Temperature in degrees C

# o2 Oxygen in mg/L: if not entered this will calculate O2 concentration at saturation

# patm Atmospheric pressure in atmospheres

# Test data: temp=20, patm=1 : O2 = 9.095342

# temp=12, o2=2, patm=0.5 % Sat = 37.61668

a <- (exp(7.7117-1.31403\*log(temp+45.93)))

b <- (1-exp(11.8571-(3840.7/(temp+273.15))-(216961/((temp+273.15)^2)))/patm)

cc <- (1-(0.000975-(0.00001426\*temp)+(0.00000006436\*(temp^2)))\*patm)

d <- (1-exp(11.8571-(3840.7/(temp+273.15))-(216961/((temp+273.15)^2))))

e <- (1-(0.000975-(0.00001426\*temp)+(0.00000006436\*(temp^2))))

o2Sat <- a \* b \* cc \* patm / d / e

if (is.null(o2))

o2Sat

else

o2\*100/o2Sat

}

electNominal <- function (pavail, peaten)

{

# This function calculates the new electivity index

# Input is the proportion available and the proportion eaten of that prey

# relative to all other prey or potential prey.

# Both proportions must be between 0 and 1 (if pavail is 0 the result is NA)

# Input is eitehr a vector where the first item is pavail and the second is peaten

# Or separate values for pavail and peaten

prod1 <- peaten \* (1-pavail)

prod2 <- pavail \* (1-peaten)

elect <- prod1/(prod1 + prod2)

elect

}