Tutorial for the biogeo package for R

Mark Robertson

University of Pretoria

February 2016

Getting started

Open the file called tutorial.r in the doc folder of the biogeo package to run the analyses listed below. This file contains all of the code below.

1. Data formatting for compatibility with biogeo

The biogeo package requires specific fields in the dataset. The function checkdatastr can be used to determine whether these required fields are present. The main required fields can be added using addmainfields. Particular field names can be renamed using renamefields.

```
data(dat) # load a dataset of simulated data
(ck<-checkdatastr(dat)) # check data structure
```

```
        Field Present
1          ID    TRUE
2           x    TRUE
3           y    TRUE
4     Species    TRUE
5  x_original    TRUE
6  y_original    TRUE
7  Correction    TRUE
8    Modified    TRUE
9     Exclude    TRUE
10     Reason    TRUE
```

If some of the required fields are missing they can be added using the function addmainfields.

```
dat2 <- dat[,names(dat)[names(dat)!='ID']]        #Remove the ID column
dat2 <- addmainfields(dat2, species='Species')
head(dat2)
```

The required fields can be selected from a dataframe using the function keepmainfields. For example many fields are avaialable when data are downloaded from GBIF and the user may want only some of these fields.

```
data(gbifdat)     # An example dataset from GBIF
names(gbifdat)
dat3 <- keepmainfields(gbifdat, Species='species', x='decimallongitude', y='decimallatitude')
head(dat3)
```

Field names can be changed using renamefields.

```
names(gbifdat)
dat4 <- renamefields(gbifdat, ID='gbifid', x="decimallongitude", y="decimallatitude",
Species="species")
names(dat4)
```

Formatted data can be viewed as a map using the pointsworld function. The object dat must contain a unique identifier column called “ID” and have x- and y-coordinates in decimal degrees.

```
pointsworld(world, dat2, out=F) # Basic plotting options
```

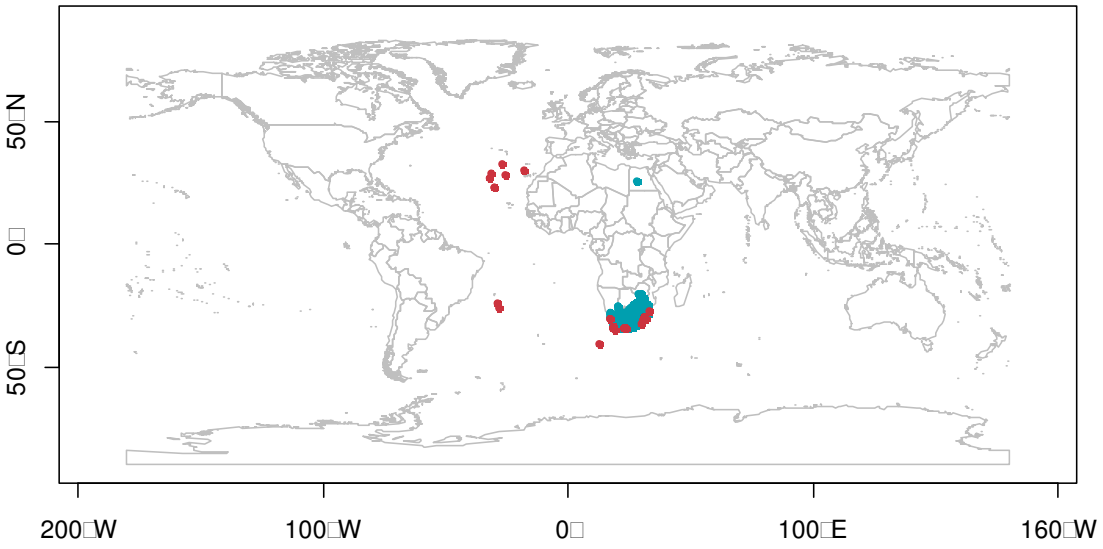

Fig. 1 Point records displayed using pointsworld.

By default the world extent is displayed. The extent can be specified using the points.

```
z2<-pointsworld(world,dat,"x","y",ext="p")
```

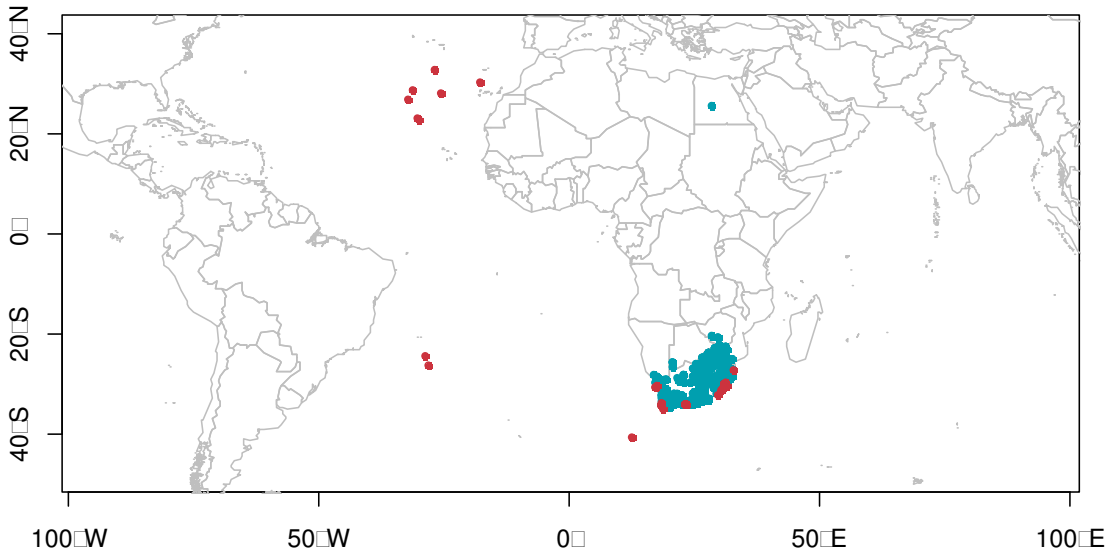

Fig. 2 Point records displayed using pointsworld, where the extent is adjusted to the point records by specifying ext=”p”.

2. Convert coordinates to decimal degrees

In many cases the coordinates are already in decimal degrees but sometimes they are in another format e.g. degrees, minutes and seconds (28°13’45”E) or degrees and decimal minutes (28°13.751’E). Coordinates can be converted into decimal degrees in a spreadsheet using an appropriate formula. If the coordinates are in text format then they will need to be separated into separate columns in order to apply the formula. However, a dataset typically includes coordinates in several different formats.

In the example below the data are saved into a comma separated values file (.csv) called placestest.csv in the extdata folder of the biogeo package. This dataset can also be accessed in R using the command data(places). Below is an extract from this file, showing the coordinates in various formats in two columns in the spreadsheet, called “long” and “lat”.

id	Place	long	lat
1	Chimoio	33 28.9 E	19 6.98 S
2	Grahamstown	26d31m59.98 E	33d17m60.00 S
3	Kenton	26°38'59"E	33°40'0.01"S
4	Ladybrand	27°27' E	29°12' S
5	Maun	23 25 E	19 58 S
6	Mwinilunga	E 24 25 59.9880	S 11 43 59.9880
7	Pretoria	28°13 45.9840 E	25°42 24.9840 S
8	Tsumeb	17 43 0.0120 E	19 13 59.9880 S
9	Frostburg	78 55 42.3912 W	39 39 30.8556 N
10	San Francisco	122 25 9.4116 W	37 46 40.9512 N

The function called dmsparse can be used to separate (parse) coordinates into separate columns, provided that there are delimiters between the degrees, minutes and seconds e.g. 27°27’ E.

```
# places<-read.csv("placestest.csv",stringsAsFactors=F)
data(places)
cdat<-dmsparse(dat,x='long',y='lat',id='id')
```

The function dmsparse uses “x” and “y” as default names of the longitude and latitude fields in the input file. A column of unique identifiers is required (ID). When there are errors with the records a value of 1 will be assigned to the exclude column to indicate a problem.

When records have coordinates that are already in decimal degrees, to save processing time it may be sensible to exclude them before separating the coordinates of the other records into degrees, minutes and seconds. This can be done using the finddecimals function and removing these records before using dmsparse.

```
fd <- finddecimals(places,x='long',y='lat')
places[which(fd==1),] # View these records in original dataset
cdat[which(fd==1),] # View these records in parsed dataset
fdms <- which(fd==0) # Select only those that are not in decimal degree format
places[fdms,]
```

Excluding records that do not have coordinates

When records do not have coordinates (e.g. there is an empty string or NA) this will cause dmsparse to assign a value of one to the Exclude column. Records with missing coordinates can be removed before using dmsparse, using the function called missingcoords.

```
places2 <- places # Create a new dataset
places2$long[1] <- "" # Assign an empty string to the longitude column for the first record
places2$long[2] <- NA # Assign a missing value
places2$long[5] <- '23 25 S' # Change the letter from E to S
cdat2 <- dmsparse(places2,x='long',y='lat',id='id') # Parse of coordinates
fe <- which(cdat2$exclude==1) #Select those with errors
cdat2[fe,]

fm <- missingcoords(places2$long,places2$lat) # Find row numbers for missing coordinates
cdat3 <- dmsparse(places2[-fm,],x='long',y='lat',id='id') # Remove the rows with missing coords
```

Converting to decimal degrees when there are no delimiters

If there are no delimiters between the coordinates (e.g. 2730E instead of 27°30’E) or if the delimiters are full stops (e.g. 27.30.02E) then the function dmsabs can be used. A format string (fmt) is required that specifies the format. In this format string the letter “d” indicates degrees, the letter “m” indicates minutes and the letter “s” indicates seconds. An upper case “L” (L) indicates a letter (i.e. N, S, E or W).

```
coordstr<-'2344E'
fmt<-'ddmmL'
dmsabs(coordstr,fmt)
```

The function getformat can be used to determine the format of each coordinate when coordinates with several different formats appear in the same column.

```
coords<-c('44 25 E','21.20 E','W14.03','12.35.16 E','09.26.08 W')
format<-getformat(coords)
data.frame(coords,format)
```

Below is an example of the output, showing the format string for each coordinate. A format string is returned in which a number is indicated by a zero (0), a space by an asterisk (*) and a letter by an upper case “L” (L).

```
        coords      format
1     44 25 E     00*00*L
2     21.20 E     00.00*L
3      W14.03      L00.00
4 12.35.16 E 00.00.00*L
5 09.26.08 W 00.00.00*L
```

A unique list of coordinate formats can be returned by using the function called uniqueformats.

```
xtxt<-c('44 25 E','12.35.16 E','21.20 E','14.03 E','09.26.08 W')
fmx<-uniqueformats(xtxt)
```

When the coordinates are in several different formats they can be parsed into separate columns using the parsecoords function. A format string (fmtstr) is required that specifies a unique list of formats. In this format string the lower case letter “d” indicates degrees, the letter “m” indicates minutes and the letter “s” indicates seconds. If the degrees and minutes are separated by a space in the original coordinate coordinate string (e.g. 44 25 E) then the format string should indicate that degrees are separated from minutes with a space (e.g. “dd mm”). In contrast there may be no delimiter (e.g. 4425E) then the format string would be: “ddmmL”. In most cases the function dmsparse would be preferable as a format string is not required.

```
fmtstr<-c("dd mm","dd.mm","dd.mm ss")
px<-parsecoords(xtxt,fmx,fmtstr)
```

In the example above, there are three unique coordinate formats (specified by the format string called fmtstr). Another format string is required (fmx) which comes from the function uniqueformats.

Converting to decimal degrees when degrees, minutes & seconds are separated

If the coordinates are already in separate columns of a spreadsheet or if only single values are used then the dms2dd function can be used.

```
dec<-dms2dd(dat2$xdeg,dat2$xmin,dat2$xsec,dat2$EW)
data.frame(dat2$xdeg,dat2$xmin,dat2$xsec,dat2$EW,dec)
```

The function dd2dms can be used to convert coordinates in decimal degrees back into degrees, minutes and seconds.

Extract coordinates from Google Earth

If a point is clearly incorrect or if there are no coordinates for the record but there is a locality description then it is possible to extract the coordinates from Google Earth.

data(dat) # Access the species dataset
data(world) # Access the country boundaries
data(dem) # Access an environmental raster
pointsworld(world, dat, ext=c(17.5,20.5,-34.7,-34)) # Plot the problem point (zoomed in)
dat[dat$ID==689,] # View the coordinates and locality name for this point before updating

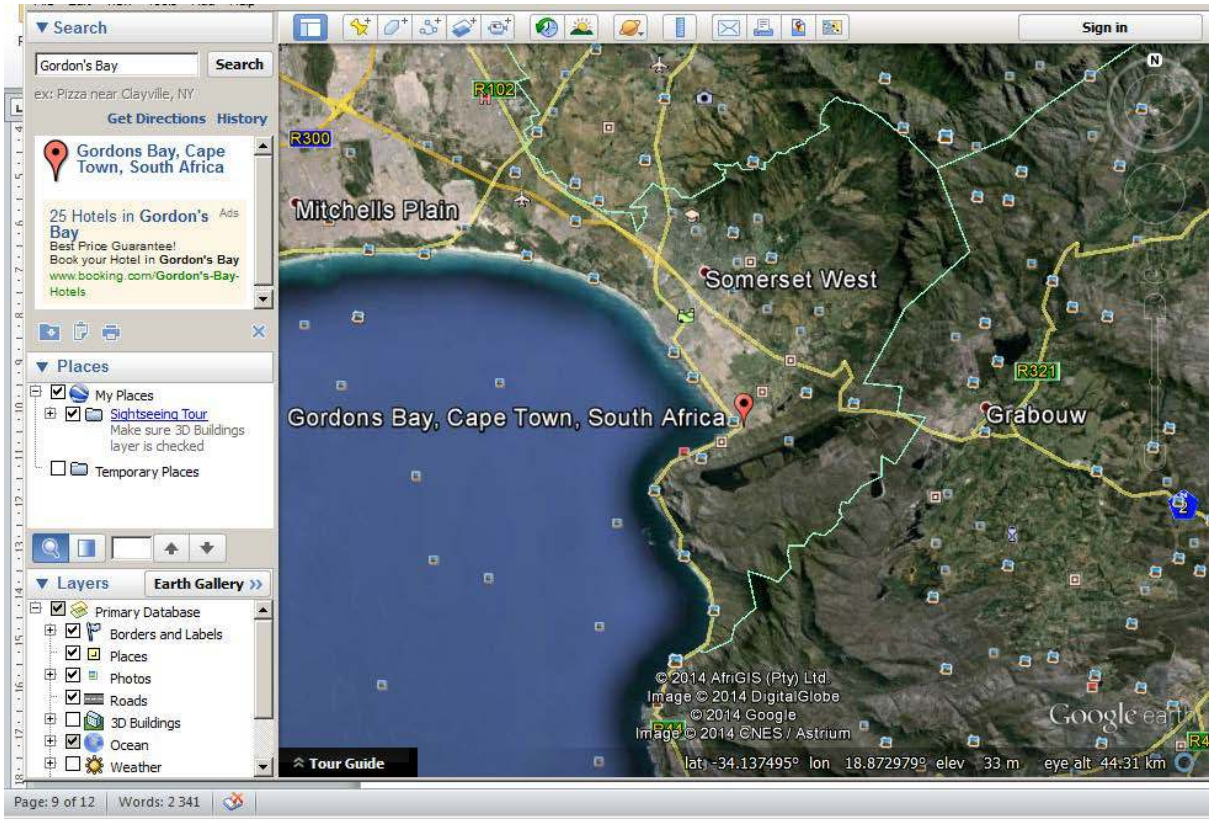

Fig. 3 A screen-shot of Google Earth showing a search for Gordon's Bay (near Cape Town). Note that the coordinates at the bottom of the map are in decimal degrees.

Move pointer over desired location (Gordons Bay) in Google Earth and then CTRL-SHIFT-C to copy coords to clipboard. After copying the coordinates to the clipboard, go back into R and run the following command to paste the coordinates into the data object:

dat<-fromGEarth(dat,ID=689)
dat[dat$ID==689,] # view the coordinates for this point after updating the coords

This command will update the x and y fields with the coordinates from the clipboard and will update the field called Modified with the current date and time. Note that the identifier of the point is 689, so this needs to be specified so that the correct record is updated.

3. Identify duplicate records to prevent pseudoreplication

For the purposes of distribution modelling duplicate records per grid cell are usually removed. The duplicatesexclude function will assign a value of one to the exclude field for duplicate records per grid cell per species in the dataset. The appropriate spatial resolution (res) for the grid cells must be specified in minutes.

```
data(dat)
dat2 <- duplicatesexclude(dat,res=20)
dat2[dat2$Reason=='Duplicated',] # View duplicated records
```

4. Identify records that may be too imprecise for the analysis

The function precisioncheck determines whether point records are sufficiently precise. It does this by identifying any records that occur either at the top left corner or centre of grid cells of various spatial resolutions, as specified using the parameters s and e (start and end).

```
data(dat) # Access the species dataset
names(dat) # Column names of species dataset
datpc <- precisioncheck(dat, x='x', y='y', s=10, e=60)
datpc[datpc$preci==1,] #View records with possible precision problems
```

The function precisionenv determines whether any records have a lower precision than a selected raster file.

```
data(dem)
datpce <- precisionenv(dat, dem, x='x', y='y')
datpce[datpce$envpreci==1,] #View records with possible precision problems
```

5. Identify records that likely have incorrect coordinates using geographical and environmental information

5.1. Identify points that are in the wrong environment

The function nearestcell will move any records that are currently in the sea to the nearest adjacent coastal grid cell that contains environmental data (or to the nearest sea grid cell for marine species). The size of the cells is specified by the input raster (dem).

```
pointsworld(world, dat, ext="p") #Points in the wrong environment are shown in red, those in the correct environment in blue
datx <- nearestcell(dat, dem) # Returns a list with (1) the modified data and (2) the IDs and coordinates of moved points
s1 <- datx$moved # The points moved
dat2 <- datx$dat # The modified data
cr <- datx$dat$Correction
```

```
s2 <- which(str_detect(cr,"7"))
datx$dat[s2,] #View data for records that were moved
pointsworld(world, dat, ext="p") # Plots points on world map and checks for errors
points(s1$x,s1$y,pch=18) # Display the moved points in black
```

The example below shows the use of the nearestcell function for moving records for marine species.

```
rstm<-raster(xmn=-180, xmx=180, ymn=-90, ymx=90,res=10/60,vals=1)#
data(msk10) # load indices of land cells
rstm[msk10]<-NA # assign NA to the land cells
pointsworld(world, datm, ext="p") #Points in the wrong environment are shown in red, those in the correct environment in blue
datmx <- nearestcell(datm, rstm) # Returns a list with (1) the modified data and (2) the IDs and coordinates of moved points
s1 <- datmx$moved # The points moved
datm2 <- datmx$dat #The modified data
cr <- datmx$dat$Correction
s2 <- which(str_detect(cr,"7"))
datmx$dat[s2,] #View data for records that were moved
pointsworld(world, datm,ext="p") # Plots points on world map and checks for errors
points(s1$x,s1$y,pch=18) # Display the moved points in black
```

It is possible to identify points that are in the wrong environment (e.g. terrestrial species in the sea OR marine species on land). When records for terrestrial species appear in the sea then these will have missing values in a raster containing terrestrial environmental data. These records with missing values can be excluded by assigning a value of one to the exclude field.

```
d2 <- missingvalsexclude(dem, dat) # identify records with missing values
d2[d2$Exclude==1,] # Exclude = 1 are records with NA values of dem
```

5.2. Get alternative coordinates using geographical visualization

Errors in datasets can arise in many ways, including by accidentally transposing x- and y-coordinates or assigning the incorrect letter to the coordinate e.g. “E” when it should be “W”. When these records are plotted on a map, they can easily be identified as being incorrect because they appear as outliers. The function alternatives is used to determine where an incorrect record should be placed by showing alternative positions for that point based on common errors in datasets.

```
dat<-alternatives(dat,group1="Species",group2="",world,dem,locality="",pos="bottomleft",ext="p")
```

The user starts by clicking on a record of interest. Then alternative positions for that record are displayed using purple point symbols. All other records for that particular species are indicated in black. The user then clicks on the position of the correct record, or back on the originally selected record to exclude it. If none of the records are correct then the stop button should be selected (top left of plot screen). The identifier (ID) of the record is displayed next to the point and its coordinates

and species name are displayed at the top of the map. Once a new position for the point is selected then the new coordinates for that point are displayed at the bottom of the map. When a record is changed then all records with identical x- and y-coordinates will also be changed in the same way. This is because several different species may have been collected at the same locality. The original values of the x- and y-coordinates will be written into the fields x_original and y_original. The date and time that the record was modified will be written into the field called Modified. The type of correction will be recorded as a number in the field called Corrected. Records that were modified can be found using the functions modifiedtoday for records modified today or using the function called modified for records modified between two dates.

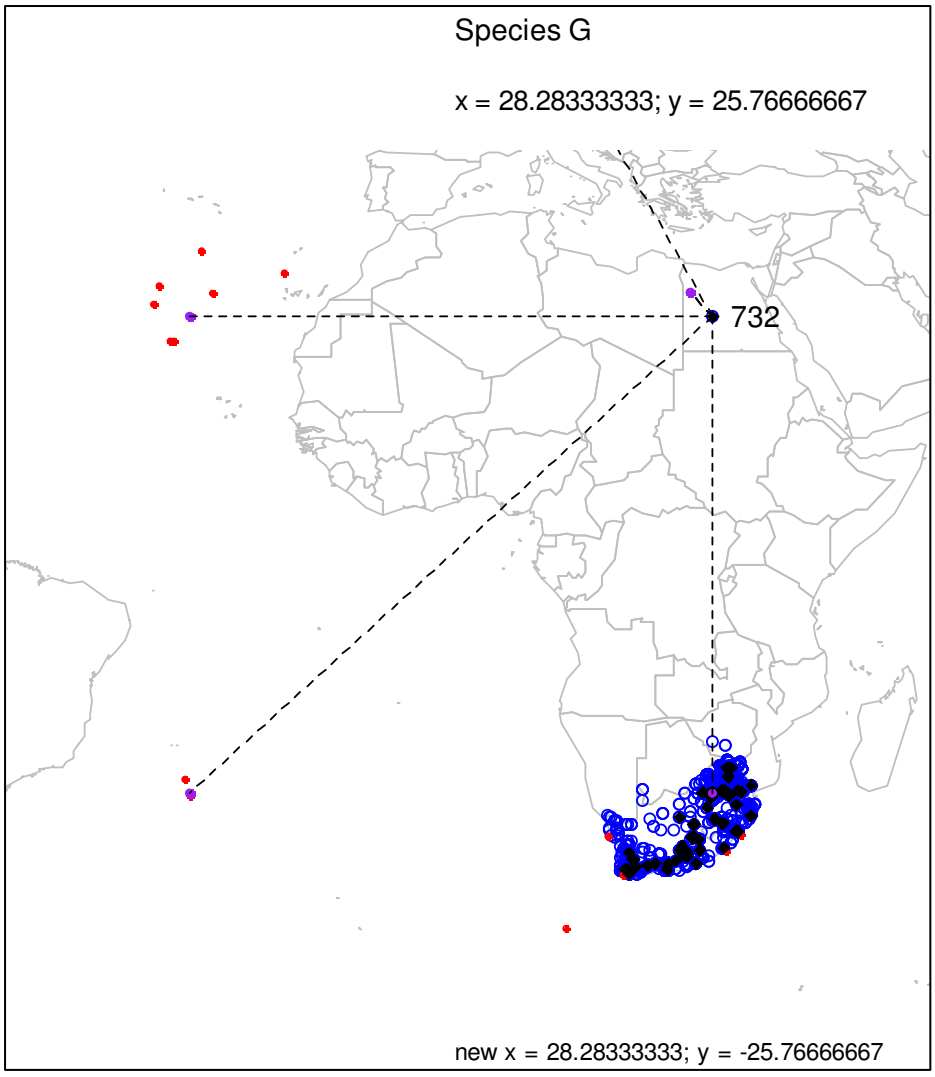

Fig. 4 Alternative positions (purple points) for the point with the identifier 732. Records for that species (Species G) are indicated in black and records for other species are in blue. Records that fall in the sea are shown in red.

Alternatives per species

The function alternatives2 will display alternatives for a selected species and shows only the records for that species instead of all species in the dataset.

```
d4<-alternatives2(dat,"Species A",group1="Species",group2="",world,dem,locality="LocalityName",
        pos="bottomright",ext="p")
```

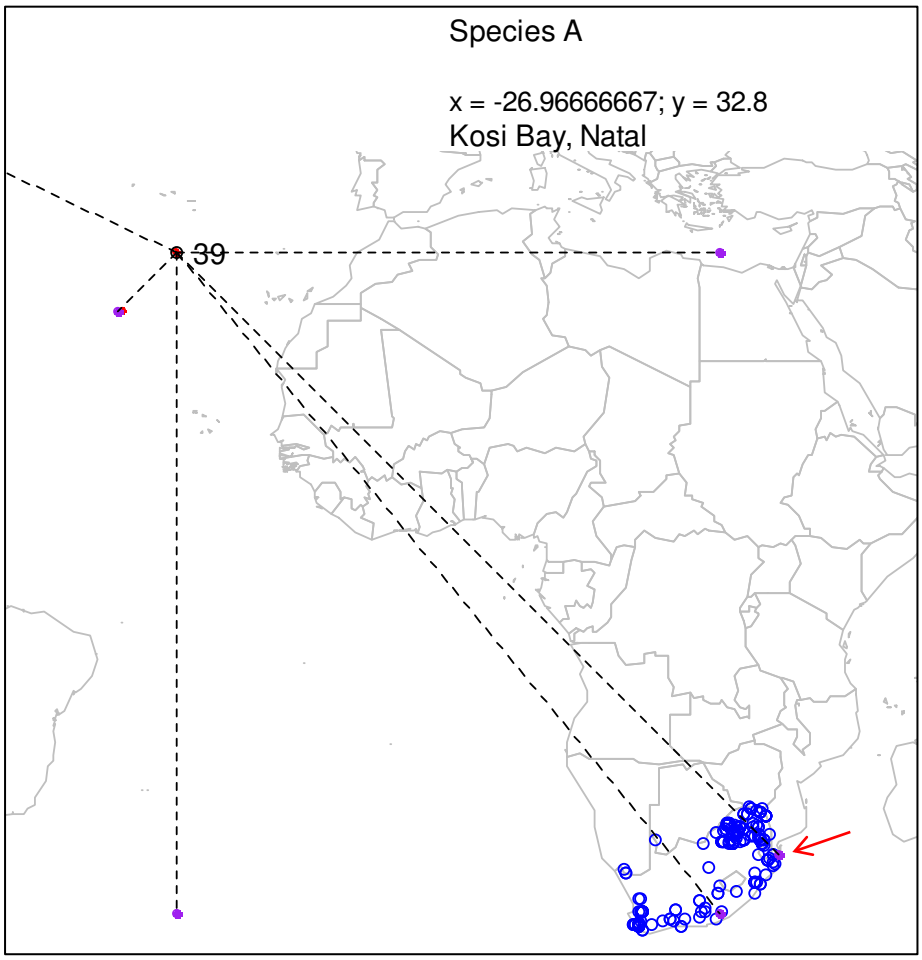

Fig. 5 Alternative point records for a selected species (Species A) using alternatives2. Only the records for Species A are displayed.

In the example in Fig. 5 above the correction position for the record is indicated by the red arrow. The selection of this point instead of the other point in South Africa was based on the locality description for the point which is displayed at the top of the map (Kosi Bay).

5.3. Get alternative coordinates using environmental variables

Decisions about which of the alternative points are most likely to be correct can be made by using a combination of geographical and environmental data.

Before proceeding, a set of environmental data should be downloaded for use with this and other functions. Download the Worldclim 10 minute bioclimatic variables data in generic grids format (.bil extension) from the Worldclim website http://www.worldclim.org/current. Unzip the file into the extdata folder of the biogeo package.

Create a raster stack containing four selected variables:

```
fd<-system.file(package="biogeo")  # find path for biogeo package
foldenv<-file.path(fd,"extdata", fsep = .Platform$file.sep)
ev<-env2stack(foldenv, vars = c("bio1","bio12","bio5","bio6"), fext="bil")
```

The alternativesenv function can now be used. The plotsetup function should be used with the alternativesenv function.

```
plotsetup(6,6)
g1="Species U"
vars=c("bio1","bio12");
d5<-alternativesenv(dat,g1,group1="Species",ev,vars,world,xname="Annual Mean
Temperature",yname="Annual Precipitation",dem,locality="LocalityName",ext="p")
```

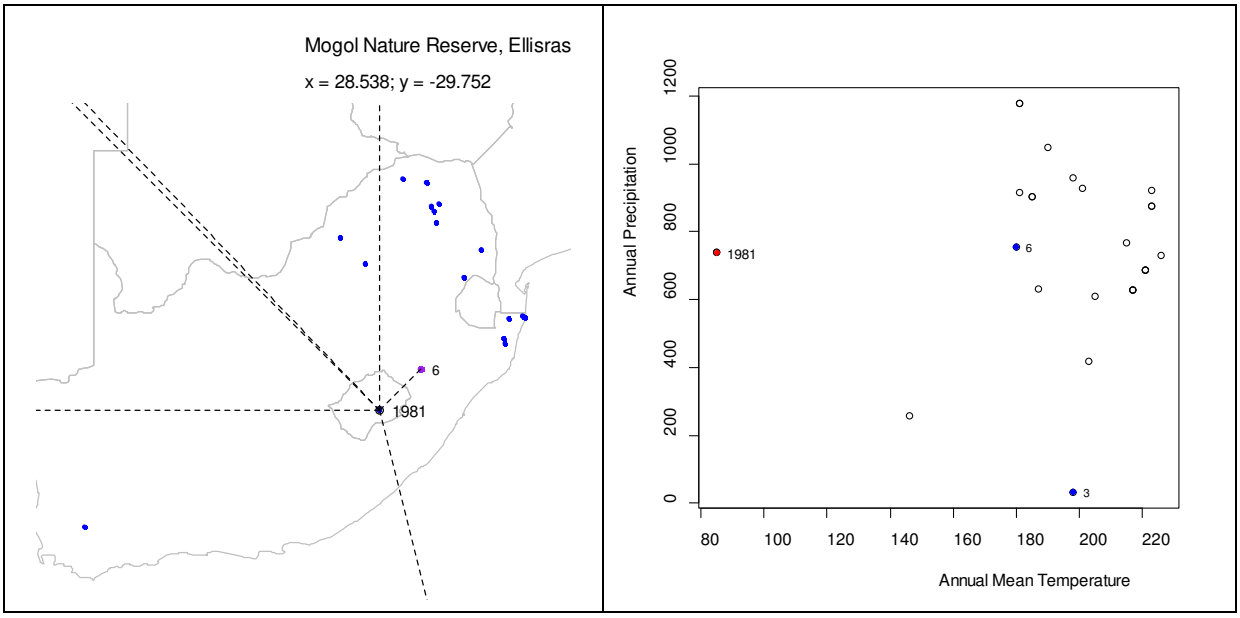

Fig. 6 Outputs from alternativesenv. The alternative points for the record selected (ID 1981) in the map on the left are displayed in a two-dimensional environmental space on the right. The environmental space is defined in this example by annual precipitation (bio12) and annual mean temperature (bio1).

5.4. Alternatives using either geographical or environmental information

```
sp<-"Species U"
plotsetup(6,6)
ed<-geo2envid(edat,sp,"Species","",world,xc="bio12",yc="bio1",xname="Ann. Precip.",yname="Ann.
Mean Temp.",showrecord="",ext="p")
```

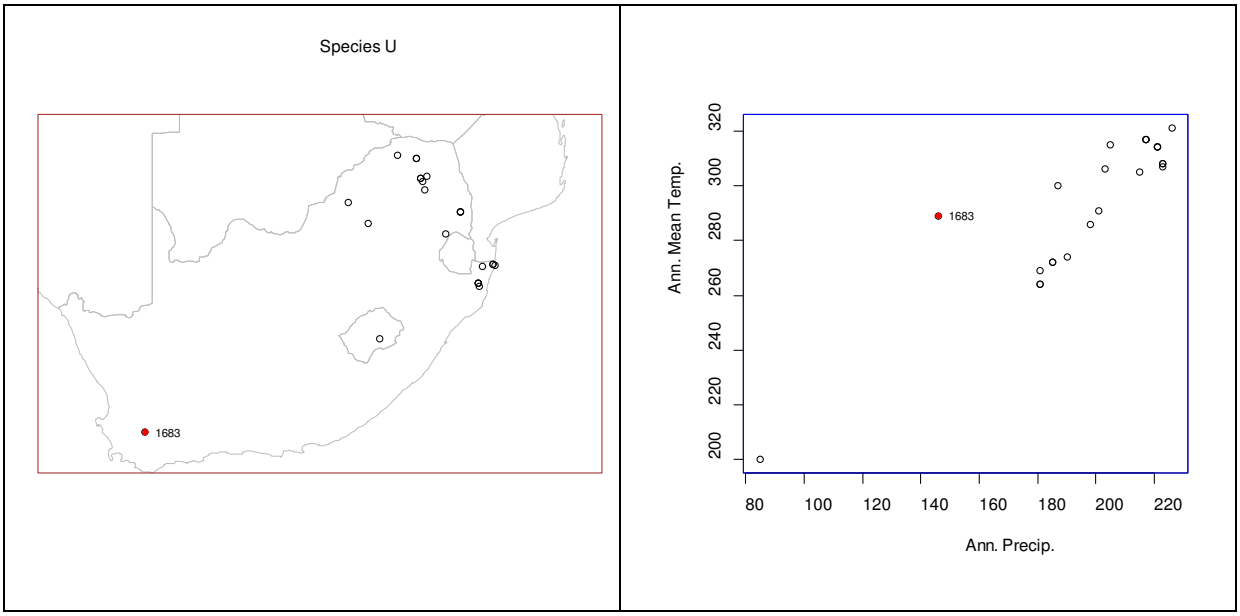

Fig. 7 The function geo2envid is used for points that are queried in geographical space (left panel) can be shown in a two-dimensional environmental space (right panel) and vice versa.

Points can be queried in geographical or environmental space. The selected records are marked with a red dot and ID numbers are shown. Records that are considered to be outliers can be excluded by selecting the record in the environmental space. A menu with various options is produced.

The points can be displayed in a two-dimensional environmental space either defined by untransformed environmental variables, using geo2envid (Fig. 7) or by principal components from a principal components analysis of several selected environmental variables, using geo2envpca (Fig. 8).

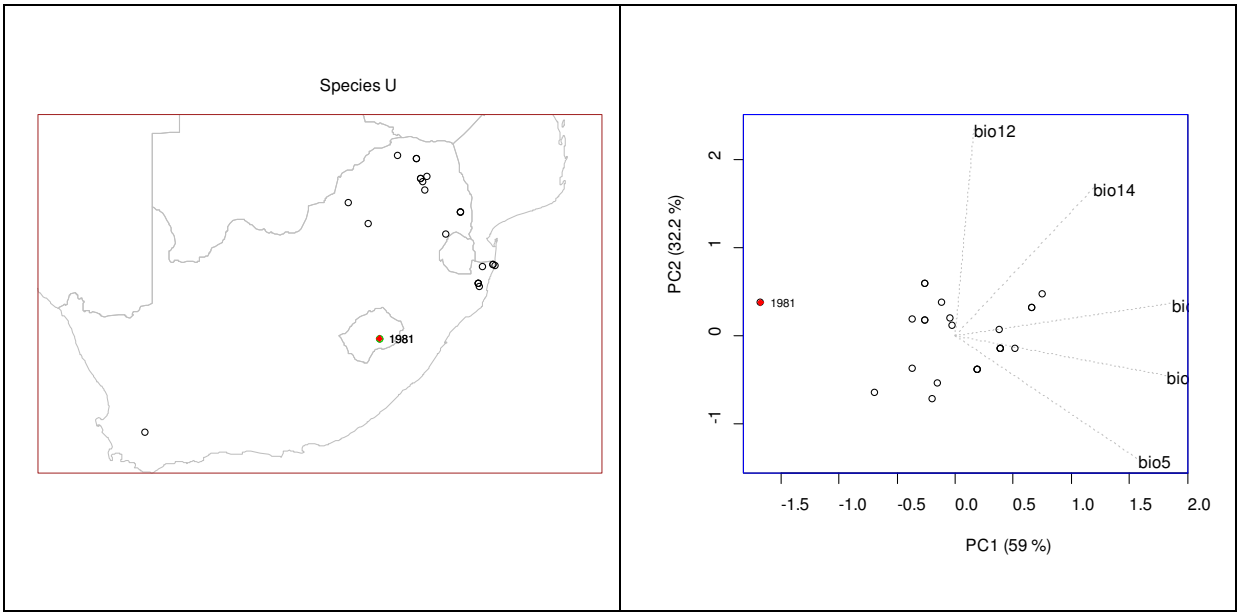

Fig. 8 Outputs from geo2envpca are similar to those of geo2envid except that the environmental space is defined by principal components from a principal components analysis.

5.5. errorcheck and quickclean provide a suite of error detection tools

The function called errorcheck can be used to find a number of different types of errors in a dataset consisting of records for several species.

```
data(edat)
errorcheck(world, dem, dat, countries = "", countryfield = "NAME", vars = c("bio1", "bio12", "bio5",
"bio6"), res = 10,elevc="",diff=50)
```

The errorcheck function extracts country names for points using the coordinates (and writes the names into country_ext) and compares this to the country names listed in the dataset for those records. If there is a mismatch in these names then this is indicated with a value of one in the field called CountryMismatch. Records for which there are no environmental data (based on the object dem) are indicated with a value of one in the field called wrongEnv. Low precision records are indicated by a value of one in the field lowprec. Environmental outliers are indicated by a value of one in a field beginning with the name of the environmental variable and ending either in "_e" for records assessed using boxplot statistics (e.g. bio1_e) or ending in "_j" for records assessed using the reverse jackknife procedure. The recorded elevation values for records (specified with a field name in elevc) are compared to digital elevation model values (which are returned in the field demElevation) and indicated as a mismatch if they exceed the value specified in the parameter called diff. This parameter is the difference in metres. The field called "error" will contain a value of one if there are any values of one in CountryMismatch, CountryMismatch, wrongEnv, lowprec or any of the outlier fields. The field called "spperr" will contain ones for all records of a species for which there are one or more errors.

Data from the errorcheck function can be displayed on maps using the option called group2 in geo2envid or geo2envpca. In the example below for Species T (Fig. 10) records are coloured green or blue depending on the value in the field bio6_j, which is the field indicating outliers for bio6 (minimum temperature of coldest month).

The results from errorcheck can be viewed using the function geo2envid. In this case the outliers for bio6 are displayed (bio6_j).

```
sp<-"Species T"
plotsetup(6,6)
ed<-geo2envid(d4,sp,"Species",group2="bio6_j",world,xc="bio1",yc="bio5",xname="Ann.
Precip.",yname="Ann. Mean Temp.",showrecord="",ext="p")
```

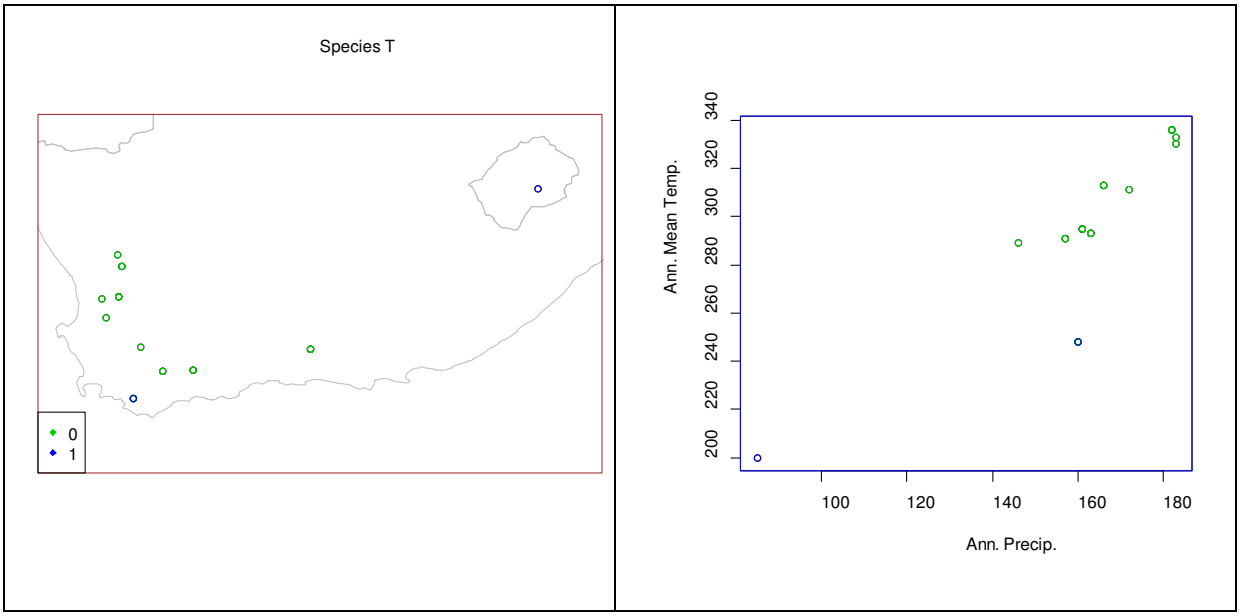

Fig. 9 Outputs from geo2envid when using the option group2.

Rapid cleaning of a dataset

The function called quickclean performs many of the checks performed by errorcheck but instead of indicating records with possible errors it simply removes these records from the dataset. This function performs a country mismatch check if the country field is specified, it performs a check to determine if the records are at the appropriate precision for the spatial resolution, it assigns point records to the nearest cell containing environmental data (using nearestcell) and removes records that are in the wrong environment. It flags duplicate records per species per grid cell but does not remove the duplicates. It does not require environmental data and does not perform the environmental outlier checks as performed in errorcheck.

```
data(msk10)
dat2<-quickclean(world,dat,ID='ID',Species='Species',x='x',y='y',countries =
"",others='',res=10,msk=msk10,ext="")
```

It returns only the fields below:

	ID	Species	x	y	indx	dups
1	1	Species A	25.01667	-33.08333	1595311	0
2	2	Species A	26.53333	-33.31667	1597480	0
3	3	Species A	23.50000	-33.30000	1597462	0
4	4	Species A	27.08333	-32.58333	1588843	0

Data summaries and output

```
data(dat)
data(dem)
```

The following command can be used to find the names associated with each of the Worldclim bioclimatic variables:

wclim(full=T)

	bio	abbrev	name	correction
1	BIO01	AMT	Annual Mean Temperature	10
2	BIO02	MDR	Mean Diurnal Range	10
3	BIO03	I	Isothermality	10
4	BIO04	TS	Temperature seasonality	10
5	BIO05	MTWM	Max Temperature of Warmest Month	10
6	BIO06	MTCM	Min Temperature of Coldest Month	10
7	BIO07	TAR	Temperature Annual Range	10
8	BIO08	MTWeQ	Mean Temperature of Wettest Quarter	10
9	BIO09	MTDQ	Mean Temperature of Driest Quarter	10

wclim(f=c(1,3,9)) # This gives the names for specific bioclim variables, i.e. bio1, bio3 and bio9

Counting records per species

The number of records per species can be obtained using the function called speciescount. It will return the total number of records (ntot) and number of unique records (nuniq) per species. The number of unique records is the total number when duplicate records per grid cell are excluded. The records can be counted from the dataset once missing value records have been excluded.

nsp<-speciescount(d1,orderby="nuniq")

	Species	ntot	nuniq
1	Species A	216	98
2	Species B	151	61
6	Species F	96	57
7	Species G	88	51
8	Species H	88	51

(nsp <- speciescount(dat2, orderby="nuniq")) # Number of records per species. Ordered by the number of unique records.

The functions modified and modifiedtoday can be used to view records that were modified during the cleaning process.

datx <- nearestcell(dat, dem) # Move records in the sea to the nearest land cell (for terrestrial species)
dat3 <- datx$dat
f <- modifiedtoday(dat3)
dat3[f,] # Extract the records that were modified today
f2 <- modified(dat3,"01-01-2015 00:00:00","31-12-2015 00:00:00") # Records modified between two dates or times
dat3[f2,]

Richness maps

```
data(dat)
data(dem)
dem2<-crop(dem,c(15,35,-36,-23))
rich <- richnessmap(dat, dem2, option="richness")
colPal <- colorRampPalette(colors=c('#556270','#4ECDC4','#C7F464','#FF6B6B','#C44D58'))
plot(rich, col=colPal(100))
```

Richness maps can be produced at any spatial resolution without having to use a raster.

```
ex1 <- c(15,35,-36,-23)
rich<-richness(dat,res=20,option="richness",buf=5,ext=ex1)
colPal <- colorRampPalette(colors=c('#556270','#4ECDC4','#C7F464','#FF6B6B','#C44D58'))
plot(rich, col=colPal(100))
```

A richness map can be produced rapidly using the quickrich function. This function produces a richness map using an input dataframe at the spatial resolution selected. It makes use of the function quickclean to remove records that contain errors.

```
data(dat)
data(msk20)
ex1 <- c(15,35,-36,-23) # set the extent
rich<-quickrich(world,dat,ID='ID',Species='Species',x='x',y='y',countries =
"",others='',res=20,msk=msk20,ext=ex1)
plot(rich, col=colPal(100))
writeRaster(rich, filename="richtest.asc", datatype='INT4S', overwrite=TRUE) # write to an ascii file
```

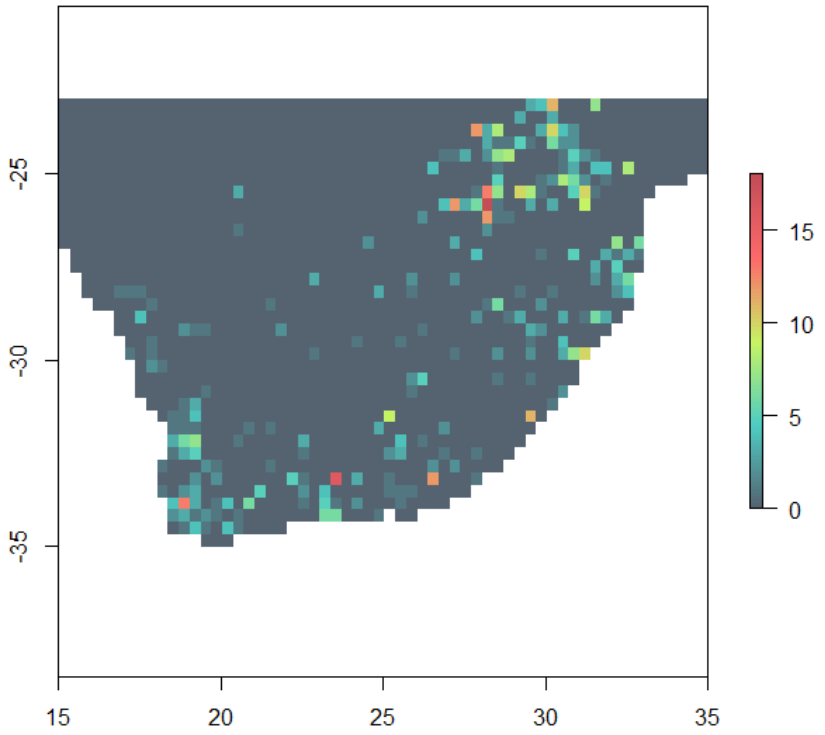

Fig. 10 A species richness map generated using the function quickrich at 20 minute spatial resolution.

Exporting data

A dataframe can be saved to a kml file that can be opened in Google Earth.

```
ss <- sample(1:nrow(dat),size=50)
dat2 <- dat[ss,] # Randomly sample 50 records
xy <- data.frame(dat2$x,dat2$y)
kmlEx <- SpatialPointsDataFrame(xy,data=dat2,proj4string=CRS("+proj=longlat +datum=WGS84"))
require(plotKML)
icon = "http://maps.google.com/mapfiles/kml/pal2/icon18.png" # Select an icon
```

The code below will create a kml file called "kmlEx.kml" in the working directory that you can open in Google Earth.

```
kml(kmlEx, shape=icon, colour=Species, labels=ID, size=1)
```

Data can be written to a point shapefile (for use in a GIS package)

```
fn <- "shapetest.shp" # The name of the shapefile
points2shape(dat, x="x", y="y", fn=fn) # This file can now be opened in a GIS package
```

Data can be written to a comma separated values file (csv file) that can read using a spreadsheet such as MS Excel.

```
write.csv(dat, file="csvtest.csv", row.names=F)
```