GDM workflow documentation

CSIRO Macroecological Modelling Team

### Purpose

This document is intended to demonstrate a complete GDM workflow process, using the custom built *gdmEngine* R package. This package of functions should have generic value for a wide range of potential applications, but has been designed specifically for deriving GDMs for specified taxa across the Australian continent.

An important feature we have incorporated into this set of linked functions is the documentation of each step in the workflow, through writing data products to file, with associated log files that document when and how the data products were derived. The intent is that this will enable enhanced clarity, repeatability and documentation of the steps undertken in deriving a GDM for a taxonomic group.

### Getting started

The first step is to establish the necessary processing environment, including loading the gdmEngine package. If your computing environment doesn’t already have R installed, then download and install it from here, to suit your computer:  
<https://www.r-project.org/>  
and also download and install RStudio, as a nice environment from which to use R, if you don’t already have it:  
<https://www.rstudio.com/products/rstudio/download/#download>  
and we’ll also need Rtools installed, which can also bee downloaded and installed for your operating system from CRAN:  
<https://cran.r-project.org/bin>  
For example, for windows, use the Rtools34: <https://cran.r-project.org/bin/windows/Rtools/Rtools34.exe>

Once this software is installed, launch RStudio and install the devtools package, which we’ll need to enable installation of the gdmEngine package:

install.packages("devtools")  
library(devtools)

Then we can install the *gdmEngine* package, which will automatically install a number of other R packages on which it relies. To do this, we use the *install.packages* function, pointing to the github location where *gdmEngine* is stored.

install\_github("???/gdmEngine")  
library(gdmEngine)

To manipulate data as part of the workflow presented here, we’ll also need to load a number of R packages, which will have been installed as part of the installation of *gdmEngine*.

library(raster)  
library(gdm)

Once your system is set up the first time, subsequent implementation of the workflow only requires load the already installed libraries, using the *library()* function. Now we’re ready to proceed through the data processing workflow.

### Preparing the spatial environmental data

This workflow assumes that environmental predictors are available as spatially complete layers, and that these grids are stored in ‘.flt’ format if you want to generate transformed grids from the final GDM you fit. So the first task is to specify a regional mask, that specifies the common grid resolution, extent and all valid cells for the modelling (cell value = 1).

Aus.domain.mask <- raster("//ces-10-cdc/OSM\_CDC\_GISDATA\_work/AUS0025/CLIM/MASK/MASK0.flt")

Now we can specify the environment grids we would like to consider as candidate predictors of compositional dissimilarity, and combine them into a raster stack. Below is an example of how we might read in a large number of spatial filenames, then refine this to a set that we think are potentially ecologically meaningful predictors for the taxa we are modelling. Obviously the filepaths below will need to be customised by each user.

climate.files <- list.files(path = "//lw-osm-02-cdc/OSM\_CBR\_LW\_R51141\_GPAA\_work/ENV/A/OUT/1990",  
 full.names=TRUE,   
 pattern = ".flt")  
terrain.files <- list.files(path = "//lw-osm-02-cdc/OSM\_CBR\_LW\_R51141\_GPAA\_work/ENV/A/OUT/LAND",   
 full.names=TRUE,   
 pattern = ".flt")  
soil.files <- list.files(path = "//osm-23-cdc/OSM\_CBR\_LW\_DEE\_work/source/env/SOIL/TOP",   
 full.names=TRUE,   
 pattern = ".flt")  
env.files <- c(climate.files, terrain.files, soil.files)  
  
# Sometime programs such as ArcMap creates filenames that have '.flt' within them, so it's important to remove theses  
env.files <- env.files[(substr(env.files, nchar(env.files)-3, nchar(env.files)) == ".flt")]  
  
# We can then list all of the files available by calling 'env.files', after which we may choose to remove some that are not relevant. For example:   
env.files <- env.files[-c(3,11,12,26,29,30,31,32,33,34,37,38,39,40)]   
  
# Now we can stack the environmental grids that we do want to retain in the analysis.  
env.stk <- stack(env.files, quick=TRUE)

### Specifying taxonomic parameters

Now we need to establish some key data and modelling parameters for the taxonomic group we are going to model. Our approach here is to establish all these parameters up front, then use them in the workflow functions that follow. That way, it is relatively simple to then modify these key parameters for a new taxaonomic group, and then run the same workflow functions. For this example, we will use all described reptile species native to Australia. This workflow assumes you have a list of species names for which you want to download data for and prepare ready for modelling.

# Load our list of species names from a file, then prepare it to a nice simple list of species names. Obviously use your own file and folder path in the code below.  
species.names.file <- "//osm-23-cdc/OSM\_CBR\_LW\_DEE\_work/source/biol/reptiles/AFD-20171211T113438.csv"  
species.names <- read.csv(species.names.file)  
  
# This file has a column for the genus name and a cilumn for the species name, which we will join to standard binomial nomenclature.  
species.names <- paste(species.names$GENUS, species.names$SPECIES)  
  
# And just ensure we don't have any duplicate names  
species.names <- unique(species.names)

We also need to specify some folders where we will write out data files and processing log files created during the workflow. Again, replace the paths below with your own.

## Specify working folders  
# Now select a folder in which to store the ALA records we are going to download  
species.records.folder <- "//osm-23-cdc/OSM\_CBR\_LW\_DEE\_work/source/biol/reptiles"  
  
# The records will be downloaded to a subfolder called "raw\_files", so specify this  
species.records.folder.raw <- "//osm-23-cdc/OSM\_CBR\_LW\_DEE\_work/source/biol/reptiles/raw\_files"  
  
# And also specify the folder in which to store the outputs of the data processing and modelling steps  
data.processing.folder <- "//osm-23-cdc/OSM\_CBR\_LW\_DEE\_work/processing/biol/reptiles"

### Download species occurrence records from the ALA

To obtain the species records required for modelling, we will download all available records for each species in our list (‘species.names’), with the data downloaded for each species being saved as a separate file in the specified folder (‘species.records.folder’).

download\_taxalist(specieslist = species.names,  
 dst = species.records.folder)

### Merge the species occurrence records for all species

Because we downloaded the records separately for each species, we will now merge those records into a single dataframe. Here we specify the folder holding our downloaded species record files (‘species.records.folder.raw’), and the folder where we will write out a copy of the merged data (‘data.processing.folder’). This function will also return the merged data to a datframe in our R working environment (‘All.records’).

All.records <- merge\_downloads(src=species.records.folder.raw,  
 output.folder = data.processing.folder)

At this point, if you have additional species records that you’d like to include in the analysis, such as restricted records not available via public download, then you could add those to the ‘All.records’ data frame.

### Filter the species occurrence records

While the *download\_taxalist()* function applied above in downloading species records from the ALA does some simple filtering out of data that won’t be useful, here we will further refine the data, using the merged dataframe we have just created. In this case we need to specify the raster layer specifying our spatial domain of interest (‘domain.mask’), the oldest records we are interested in including, by year (‘earliest.year’), and the limit of spatial uncertainty we are comfortable with (‘spatial.uncertainty.m’), noting that many records do not provide spatial uncertainty information and we will retain all such records, to avoid massive reductions in available data. The *filter\_ALA\_data()* function that undertakes this refinement returns a dataframe of the filtered species records.

Filtered.records <- filter\_ALA\_data(ALA.download.data = All.records$data,   
 output.folder = data.processing.folder,   
 domain.mask = Aus.domain.mask,   
 earliest.year = 1970,  
 spatial.uncertainty.m = 2000)

### Aggregate the species occurrence records to grid cells

The filtered species records are now ready to be aggregated to grid cells in the study region. In this step, the spatial location of each record (X,Y) is standardised to the centroid of the grid cell they occur within. In this step we can aggregate records from a modest neighbourhood around each cell (‘agg.radius.ncells’), to help overcome potential undersampling issues in using presence-only data, and may be particularly useful where fine-resolution spatial grids are being applied. Below we’re going to specify a 2.25 grid cell radius. Every record will only be included in one grid cell, which will be the cell that has the most species recorded within the specified radius. Setting the aggregation radius to a value less than 1 will result in no aggregation of records from surrounding grid cells.

Aggregated.records <- aggregate\_ALA\_data(ALA.filtered.data = Filtered.records,  
 domain.mask = Aus.domain.mask,  
 agg.radius.ncells = 2.25, # in units of grid cell length  
 output.folder = data.processing.folder)

### Select grid cells with sufficient species occurrence records

Having aggregated all occurrence records to grid cells, we now filter those grid cells to ones that have sufficient quantity of species observations to be considered a ‘community sample’ to be used in modelling compositional dissimilarity. First we specify an absolute minimum and maximum number of records we are willing to accept per grid cell:  
**min.richness.threshold** - We can stipulate a minimum number of species recorded in a grid cell that we will accept as an adequate sample of the community composition. Below, we’re going to say 3 species is the minimum.  
**max.richness.threshold** - Sometimes there will be grid cells that have an unrealistic number of species recorded, such as through generalising records to centroids of regions. So here we will remove all grid cells and their records where they have unrealistically large numbers of species associated with them (in this case, 50 species).

While we have stipulated an absolute minimum number of species recorded for any grid cell, we can also set an additional higher minimum threshold, that varies with natural variation in species richness across our region. The next two parameters enable this more nuanced threshold on the minimum number of species recorded that is considered an adequate community sample. This process involves looking at the maximum number of species recorded within a specified distance from each grid cell, then setting the minimum richness threshold as a specified proportion of that maximum recorded richness.  
**reference.radius.ncells** - This is the radius (in grid cells) around each cell that we will look for the most number of species recorded. (below we set this to 200 grid cells)  
**min.proportion.max.richness** - This is the minimum proportion of the maximum number of species recorded within the specified radius that we will consider an adequate sample of the community composition. Below we set this to 0.25 (i.e. 1/4 of the max. number of species per grid cell within the neighnourhood of specified radius)

Selected.records <- select\_gridcells\_composition(ALA.aggregated.data = Aggregated.records ,  
 domain.mask = Aus.domain.mask,  
 min.richness.threshold = 3,  
 max.richness.threshold = 50,  
 reference.radius.ncells = 200,  
 min.proportion.max.richness = 0.25,  
 output.folder = data.processing.folder)

The *select\_gridcells\_composition()* function is the final step in preparing the species occurrence records for modelling. The output dataframe (‘Selected.records’), which will be used as input to the GDM fitting functions that follow, has three columns: scientificName, decimalLongitude, decimalLatitude. This defines the sites (or grid cells) that will be used as a basis for deriving the GDM.

### Extract environmental predictor data for the selected grid cells

The next step in the workflow involves extracting the candidate environmental predictor data for the locations of the selected species occurrence records. Here we simply provide the selected records and the stack of environment grids to create a dataframe containing the values for each environmental predictor in each grid cell with species assemblage data.

Site.Env.Data <- extract\_env\_data(ALA.composition.data = Selected.records,   
 environment.stk = env.stk,  
 output.folder = data.processing.folder)

The *extract\_env\_data()* function is the final step in preparing the environment data for modelling. The output dataframe (‘Site.Env.Data’) which will be used as input to the GDM fitting functions that follow, has four initial columns: xy, richness, decimalLongitude, decimalLatitude. These four columns are followed by an additional column for each environmental predictor variable. Each row is a separate site (grid cell) that will be used as a basis for deriving the GDM, and as directly derived from the species composition data (‘Selected.records’). The column named ‘xy’ in ‘Site.Env.Data’ is simply a site name created by *extract\_env\_data()* derived by concatenating the decimalLongitude and decimalLatitude, separated by an underscore.

### GDM variable selection

Now both the biological and environmental data are prepared, we are ready to derive a GDM for the taxonomic group of interest. The key challenge here is to refine our potentially large set of possible environmental predictors to a smaller parsimonious set to be included in our final GDM for this taxa. The *gdm\_builder()* function automates this variable selection process, providing the necessary information for the user to select their final set of predictor variables. This function uses model predictive performance under cross-validation as a basis for selecting variables.

Some of the key parameters for deriving a GDM using the gdm\_builder function are:  
**n.pairs.train** - The number of site-pairs that we will use in generating the GDM. Below we have stipulated 100,000 site pairs.  
**n.pairs.test** - The number of site-pairs to use in testing the GDMs through cross validation. Below we set 20,000 site pairs. In this workflow, these will be derived from the default proportion (20%) of all sites available for modelling, while the training site pairs will be derived from the remaining 80% of sites.  
**sample.method** - The approach to be used in subsampling site=pairs. Selection of alternative site-pair sampling approaches will necessitate specifying the required parameters for each approach. Below we are applying the geographically weighted site-pair sampling method (‘geowt’). Later ‘bandwidth’ arguments relate to this ‘geowt’ site-pair sampling function.  
**n.predictors.min** - This specifies the number of predictor variables that the backward elimination proceedure will stop at. Hence the final model produced by this function will include this many predictor variables.  
Other arguments used in this function are described in detail in the R documentation to the function in the *gdmEngine* workflow package.

GDM.Selection <- gdm\_builder(site.env.data = Site.Env.Data,   
 composition.data = Selected.records,  
 geo=FALSE,  
 n.pairs.train = 100000,  
 n.pairs.test = 20000,  
 sample.method = 'geowt',  
 n.predictors.min = 5,  
 domain.mask=Aus.domain.mask,  
 pcs.projargs="+init=epsg:3577",  
 bandwidth.geowt=150000,  
 bandwidth.skip=2,  
 bandwidth.DistFact=1,  
 geowt.RndProp=0.05,  
 output.folder = data.processing.folder,   
 output.name = "gdm\_mod\_builder\_results")

Having run the gdm\_builder backward elimination variable selection routine to the specified number of variables, we now have a range of models to choose from in selecting our final model for this taxonomic group. Selecting this final set of predictor variables is one of the few steps in the workflow that is not automated. This step requires careful consideration of the set of alternative models, and judgement in selecting a model that best balances explanatory power with parsimony. To aid this decision, it may be usefult to view some of the outputs of the *gdm\_builder()* function, such as ‘Backward.Elim.D2’, which shows the model deviance explained after each variable is dropped from the full model being considered. This information is the basis of the automated backward elimination process, and can help identify when continued dropping of variables results in larger loss of explanatory power, as can the overall model explanatory power at each step of the backward elimination (‘Backward.Elim.Full.D2’).

GDM.Selection$Backward.Elim.D2  
GDM.Selection$Backward.Elim.Full.D2[nrow(GDM.Selection$Backward.Elim.Full.D2),]

### Deriving the final GDM

In fitting and assessing the final model, we simply need to specify the names of the environmental predictors in the model that we have selected, plus whether geographic distance is being included in the final model. Most of the other parameters will remain the same as those applied in the gdm\_builder finction. This final model fitting step includes cross-validation assessment of the model, hence we still need to specify how many site-pairs to use in training and testing the model. However, the final model parameters derived will utilise all sites in generating site-pairs, with model coefficients being averaged over models fitted using multiple samples of site-pairs (default = 10 samples).

# Specify the names of the envionmental predictors in the final model  
final.mod.preds <- c('WDA','TXM','PTX','ELVR1000','SNDT','ECET','TNI','PTOT')  
  
# And specify whether geographic distance is also being included in that final model  
geo.in = FALSE  
  
# Run the focussed assessment for the final model  
final.model.test <- gdm\_build\_single\_model(site.env.data = Site.Env.Data,   
 composition.data = Selected.records,  
 predictor.names = final.mod.preds,  
 geo=geo.in,  
 n.pairs.train = 100000,  
 n.pairs.test = 20000,  
 sample.method = 'geowt',  
 b.used.factor=2,  
 domain.mask=Aus.domain.mask,  
 pcs.projargs="+init=epsg:3577",  
 bandwidth.geowt=150000,  
 bandwidth.skip=2,  
 bandwidth.DistFact=1,  
 geowt.RndProp=0.05,  
 output.folder = data.processing.folder,  
 output.name = "gdm\_final\_model")

Now we can do some further interogation of this final model, to ensure it is appropriate. This includes deriving a summary of the model, and plotting the model.

# Check the key model values  
summary(final.model.test$Mean.Final.GDM)  
# Plot the model and the spline functions  
plot(final.model.test$Mean.Final.GDM)

If there appear to be any issues with our GDM, we can step back one or two steps in the workflow, adjust our input parameters, and generate a revised final model.

### Generating GDM transformed predictor grids

Once we have derived a suitable final model, we can then generate transformed environmental predictor grids for each variable, to enable GDM-based spatial analyses. To generate the transformed grids, we just need to specify the final GDM model object, the raster stack holding all the spatial grids and the method to use in extrapolating each variable beyond the range of data used to train the model. We suggest the ‘Conservative’ approach to model extrapolation, as described in the documentation to the *TransformGrids()* function.

TransformGrids(gdm.model = final.model.test$Mean.Final.GDM,  
 env.grids.stk = env.stk,  
 extrap.method = 'Conservative',  
 output.folder = data.processing.folder)

It is often useful to visually assess each transformed predictor grid, which can be done by writing an image of each transformed grid to the output folder.

# Get a list of all the 'float' grids in the data processing folder  
trans.grids <- list.files(path = data.processing.folder,   
 full.names=TRUE,   
 pattern = ".flt")  
trans.grid.stk <- stack(trans.grids)  
  
# Make a plot of each grid, and write it to file  
for(i in 1:length(trans.grids))  
 {  
 next.ras <- raster(trans.grids[i])  
 png(file.path(data.processing.folder, paste0("plot\_", names(next.ras), ".png")),  
 height=1000,  
 width=1000)  
 plot(next.ras)  
 dev.off()#\_\_\_  
 } # end for i

### Simple visualisation of the GDMs spatial patterns

Finally, it may also be useful to plot a simgle map representing a simplification of the GDM produced. This involves extracting the three main principle components from the transformed grids, then combining these into a single map by allocating one of the three colour axes (red, green, blue) to each axis. The resulting plot shows areas predicted to have similar composition in similar colours, and areas prediced to have different composition in more different colours. We need to remember, however, that this is a simplification of the GDM, and that other spatial analyses will reveal different aspects of the model predictions.

rastDat <- sampleRandom(trans.grid.stk, 200000)  
pcaSamp <- prcomp(rastDat)  
pcaRast <- predict(trans.grid.stk, pcaSamp, index=1:3)  
pcaRast[[1]] <- (pcaRast[[1]]-pcaRast[[1]]@data@min) / (pcaRast[[1]]@data@max-pcaRast[[1]]@data@min)\*255  
pcaRast[[2]] <- (pcaRast[[2]]-pcaRast[[2]]@data@min) / (pcaRast[[2]]@data@max-pcaRast[[2]]@data@min)\*255  
pcaRast[[3]] <- (pcaRast[[3]]-pcaRast[[3]]@data@min) / (pcaRast[[3]]@data@max-pcaRast[[3]]@data@min)\*255  
png(file.path(data.processing.folder, "Final\_Model\_RGB\_plot.png"),   
 height=1000,   
 width=1000)  
plotRGB(pcaRast, r=1, g=2, b=3)  
dev.off()#\_\_\_