Community Modelling Workflow

Generalised Dissimilarity Modelling (GDM)

About this document

Generalised Dissimilarity Modelling (GDM) has become a widely-used method to link differences in composition between samples and explanatory environmental variables ("covariates"). The dependent or predicted variable in a GDM may be any form of distance or dissimilarity measure scaled to range between 0 and 1. GDMs may be fitted using any assemblage of covariates thought to be important in explaining the differences in composition between pairs of samples. Geographical distance can be optionally included as a covariate to account for distance-decay or "isolation by distance" (IBD) effects.

This document provides an outline of a prototype *R*-package, *cmGDM*, which is designed to implement GDM modelling as the first method within a new EcoCommons Community Modelling module. Fitting a GDM is performed by the *R*-package *gdm* available from the CRAN repository. *cmGDM* has been designed to implement a simple, robust workflow for basic fitting, review and reporting of GDMs.

This document describes the state of development and testing of the R-package *cmGDM* as at 30 January 2022.

Design concepts and principles

1. One streamlined protocol for data collation & upload

The *R*-package *gdm* provides a preparatory function, *formatsitepair()*, which allows users to present data components in four formats, or "bioformats" in *gdm* terminology, which are then compiled into a data table used by the function *gdm()* to fit the model. Although this flexibility is suitable for experienced *R* users, it can be very challenging for casual users to make appropriate choices among numerous possible combinations, and prepare data components in the required formats.

The four formats represent different combinations of pre-processed fundamental data tables needed to fit a GDM. The concept implemented in *cmGDM* is to begin with basic table formats and generate pre-processed or derived tables by asking users to supply necessary information. For example, users will be asked to load community data as either a presence-absence table, an abundance table, or a dissimilarity table. In the first two table types, they are standard tables which may be assembled by hand (in spreadsheets for example) or output by functions in other *R*-packages. For dissimilarity tables, many functions in other R-packages can produce dissimilarity tables including the *R*-package *vegan*, and several packages for analysing phylogenetic and population genetic data.

> The path to data importation chosen during the development of *cmGDM* is to follow procedures referred to as "bioformat = 3" in the *gdm* package.

The expected advantages of this approach include:

- Users can easily assemble necessary tables for each data component using any application with which they feel comfortable (typically, a spreadsheet application)
- More robust data checking is possible enabling better guidance and support to users who need additional assistance
- No loss of modelling flexibility even though data input preparations are streamlined into a simplified protocol

2. Staged data entry

Implementation of the data entry steps in the prototype workflow is based on the idea of staged data entry. That is, users are asked to step through a sequence of uploading each data component.

This approach is appropriate because of ordered chains of dependencies. For example, a basic requirement is that site/sample labels associated with geographical coordinates match those used in the community ("biological") data table. Feedback on errors encountered at each step is provided and progress to the next step is only possible when data quality checks have been passed for the current step.

3. Intercept data format errors

As mentioned above, data entry problems are intercepted at each step through the data upload sequence. These data quality checks duplicate many corresponding checks and error messages generated by the function *formatsitepair()*. However, the error messages returned by that function are not easily interpreted by less-experienced *R*-users. A key design decision was taken to perform data quality checks independent of *formatsitepair()* and generate error messages designed to be more informative for all users. This is particularly important given the design choice described in the next section.

4. Flexible data file formats

EcoCommons users have a wide range of skills and experience in preparing and manipulating data required for model fitting. More experienced users, particularly those users with good *R* skills, will be able to load data into an *R* session from a range of formats (e.g. Excel spreadsheet, comma-separated value or csv file, etc), re-organise and export data to a variety of formats suitable for use in fitting a GDM. Existing modelling modules in EcoCommons permit only the uploading of csv-formatted text files as data tables. A design decision taken in developing the prototype *R*-package *cmGDM*, was allowing users to upload data tables in a wide selection of standard data file types. This is easily implemented in *R* because of the very extensive list of *R*-packages available in the CRAN repository which cater for many file formats.

The file formats currently supported by *cmGDM* include:

- comma-separated value (csv) text files
- TAB-separated (tsv) text files
- space-separated text files
- openDocument (ods) format spreadsheets
- Excel spreadsheets (xls & xlsx) spreadsheets

For spreadsheets, users can select the sheet within a multi-sheet document (more correctly referred to as a 'workbook') by sheet number or name as this is a parameter available in the functions called from relevant packages to import spreadsheets.

5. Loosely OO design

A major design choice in the prototype package *cmGDM* was to use an object-oriented approach as much as possible. The package implements a simple S3 object of class "cm_experiment" which embodies the staged approach to data collation and quality checking.

Test Data set and testing process

An important aspect of implementing a flexible data entry front-end is rigorous testing of the many combinations or data states likely to be encountered in use. A test data suite was developed to facilitate comprehensive testing during development.

Data files Sets of small data files within each data component required to fit a GDM have been prepared to provide a cross-section of the errors which must be trapped. The files and the conditions they test are listed in the following table. The files are available in the *test_data* folder of the GitLab repository.

File names reflect the error condition contained within each file (where *error condition* includes the state *No error* as indicated by "OK" within the file name).

Test procedure A set of *R*-scripts is provided in the folder *test_scripts* to run through a tests for each data entry script. Naturally, they assume that package *cmGDM* has been downloaded and installed.

Because a trapped error invokes the **stop()** function, running each test involves highlighting the appropriate lines and then clicking on the "Run" option in *R*-Studio. An error-checking test is expected to emit the appropriate error message to the *R*-console.

Running the "OK" test should emit no message - just a return to the command prompt. Typing `thisExperiment$status` on the command line will show that the status flag for the data component under test is **TRUE**. For example, immediately after instantiating a new experiment object we will see the following:

```
> library(cmGDM)
Loading required package: gdm
> thisExperiment <- cm_create_new_experiment("user123", "Blah")
> thisExperiment$status
      siteData_OK biologicalData_OK      covarData_OK predictionData_OK       modelFit_OK
            FALSE             FALSE             FALSE             FALSE             FALSE
>
```

After successfully loading site/sample data, we expect to see the following:

```
> thisExperiment$status
      siteData_OK biologicalData_OK      covarData_OK predictionData_OK       modelFit_OK
             TRUE             FALSE             FALSE             FALSE             FALSE
```

The workflow follows the sequence left to right along the *status* vector. Progression depends on successful completion of the preceding step with the exception that a GDM model can be fitted (by a call to *cm_run_gdm_experiment()*) which sets 'modelFit_OK' to TRUE) immediately after a successful call to *cm_load_covars()*. Prediction covariates are only needed when a fitted GDM is being "projected" (i.e. used to make predictions) using a covariate set not used to fit the model. Two typical prediction situations are: (a) predicting composition at all grid cells covering a study area (note that a GDM is fitted only to environmental conditions at sites/sample locations); and (b) predicting changes in composition related to climate change either at the sites/sample locations used to fit the GDM or in grid cells covering a study region. At the time of writing, that function has not been written although the function *cm_load_prediction_data()* has been written and tested.

Integration into EcoCommons framework

Info supplied by user or uploaded by user, plus messages back to the user, is considered here. See source code for the *R* data structure below..

Data expected to be supplied by UI

field	data type	role
userID	string	Unique EcoCommons user ID
userName	string	Username (ID for humans)
experimentName	string	User-supplied meaningful name for the experiment
description	string	User-supplied description of the experiment
includeGeo	logical	Will this experiment include geographic distance as a covariate?
data$siteData$srcFile	string	File name of the site data table

field	data type	role
data$siteData$siteCol	string	index to the column storing site name or labels (maybe supplied as a column name or integer representing col number converted to string)
data$siteData$longitudeCol	string	index to the column storing longitude of sites (maybe supplied as a column name or integer representing col number converted to string)
data$siteData$latitudeCol	string	index to the column storing latitude of sites (maybe supplied as a column name or integer representing col number converted to string)
data$siteData$dataTable	*R* data.frame	Site data table
data$biologicalData$srcFile	string	
data$biologicalData$fileType	string	Which of the accepted file types is to be uploaded? Expected val;ues incl. '',"
data$biologicalData$sheet	string	Label for the sheet in data file when 'dataType' == ''
data$biologicalData$siteCol	string	Column in 'dataTable' storing the site/sample labels
data$biologicalData$dataType	string	String representing the 'absence' state when 'dataType' == ''
data$biologicalData$presenceMarker	string	
data$biologicalData$absenceMarker	string	String representing the 'absence' state when 'dataType' == ''
data$biologicalData$dissimMeasure	string	Dissimilarity measure used to generate 'dissimMatrix'
data$biologicalData$dataTable	*R* data.frame	Stores raw community table supplied by the user
data$biologicalData$dissimMatrix	*R* numeric matrix	If dataType == 'Dissimilarity', stores uploaded dissimilarity matrix. Otherwise, stores result of applying 'dissomMeasure' to 'dataTable'.
data$covarData$srcFolder		string
data$covarData$filenames		string vector
data$covarData$label		string
data$covarData$covarNames	string vector	
data$covarData$dataTable	*R* numeric matrix	
data$predictionData$srcFolder	string	
data$predictionData$label	string	
data$predictionData$covarNames	string vector	
data$predictionData$dataTable	*R* numeric matrix	

The following panel shows the source-code used to create the data structure of a *cm_experiment* S3 object. Note that a number of fields are populated by *R* code and are not solicited from the user via the GUI. These include:

- data table structures storing derived data (e.g. a dissimilarity matrix created by function *cm_load_community_data* when the user has uploaded a presence-absence community table and asked for the Bray-Curtis dissimilarity measure to be computed)
- values stored in the "status"" vector which are updated by each *cm_load_XXXXX* function upon successful uploading and processing of a data component
- date fields which are likewise updated by code (e.g. dateCreated, dateDataUpdated, etc.)
- data structures generated by a successful GDM fit and subsequent post-fit processing (e.g. the 'model' component)

```
# Create (instantiate) a community modelling S3 object
statusVec <- vector(length = 5) # type is "logical" by default
names(statusVec) <- c("siteData_OK", "biologicalData_OK",
                      "covarData_OK", "predictionData_OK", "modelFit_OK")

thisExperiment <- list(userID = userID,
                       userName = userName,
                       experimentName = experimentName,
                       dateCreated = as.character(Sys.Date()),
                       dateDataUpdated = "",
                       dateLastModelRun = "",
                       description = description,
                       status = statusVec,
                       includeGeo = FALSE,
                       data = list(siteData = list(srcFile = "",
                                                   siteCol = "",
                                                   longitudeCol = "",
                                                   latitudeCol = "",
                                                   dataTable = data.frame()),
                                   biologicalData = list(srcFile = "",
                                                         fileType = "",
                                                         sheet = "",
                                                         siteCol = "",
                                                         dataType = "",
                                                         presenceMarker = "1",
                                                         absenceMarker = "0",
                                                         dissimMeasure = "",
                                                         dataTable = data.frame(),
                                                         dissimMatrix = matrix()),
                                   covarData = list(srcFolder = "",
                                                    filenames = "",
                                                    label = "",
                                                    covarNames = "",
                                                    covarSiteMin = numeric(),
                                                    covarSiteMax = numeric(),
                                                    covarExtentMin = numeric(),
                                                    covarExtentMax = numeric(),
                                                    dataTable = matrix()),
                                   predictionData = list(srcFolder = "",
                                                         filenames = "",
```

```
                                                    label = "",
                                                    covarNames = "",
                                                    covarSiteMin = numeric(),
                                                    covarSiteMax = numeric(),
                                                    covarExtentMin = numeric(),
                                                    covarExtentMax = numeric(),
                                                    dataTable = matrix()),
                                  sitepair = data.frame()),
                        model = list(gdm = list(),
                                     varImp = list(),
                                     perfSummary = list()))

  class(thisExperiment) <- c("cm_experiment", class(thisExperiment))
```

A worked example

A small set of data files has also been included in the folder *worked_example*. An *R*-script is included to run the complete workflow, but of course each step can be run from the command line if desired.

The worked example illustrates the workflow to fit a very basic GDM.

Successful completion of the worked example should result in the following output in the console when the function *cm_gdm_summary()* is called:

```
> cat(cmGDM::cm_gdm_summary(gdmObj))
-----------------------------------------------
EcoCommons Community Modelling Module
GDM Model Summary
-----------------------------------------------

Experiment name: RF_Refugia_GDM_Argyrodendron_trifoliolatum
Run date: 2022-01-30

Number of site/sample pairs: 120

Covariates:
  Geographical distance included: No
  Covariates: Number used = 26
    aspect
    bio01
    bio02
    bio03
    bio04
    bio05
    bio06
    bio07
    bio08
    bio09
    bio10
    bio11
    bio12
    bio13
    bio14
    bio15
    bio16
    bio17
```

```
    bio18
    bio19
    clay
    sand
    silt
    slope
    TPI
    TWI

Model performance:
      Model deviance: 0.36
  Explained deviance: 78.91%
       NULL deviance: 1.69
          Interecept: 0.045

Covariate importance:
    Covariate       Contrib.
    ------------    -----------
        aspect          7.00
         bio01          0.00
         bio02          0.00
         bio03          0.00
         bio04          0.36
         bio05          0.00
         bio06          0.00
         bio07          0.00
         bio08          0.00
         bio09          0.00
         bio10          0.00
         bio11          0.20
         bio12          0.00
         bio13          0.03
         bio14          0.02
         bio15          0.04
         bio16          0.00
         bio17          0.00
         bio18          0.00
         bio19          0.00
          clay         84.07
          sand          0.00
          silt          3.46
         slope          0.00
           TPI          0.00
           TWI          0.00
>
```