RQGIS: Integrating R with QGIS for Statistical Geocomputing

by Jannes Muenchow, Patrick Schratz, Alexander Brenning

Abstract Integrating R with Geographic Information Systems (GIS) extends R's statistical capabilities with numerous geoprocessing and data handling tools available in a GIS. QGIS is one of the most popular open-source GIS, and it furthermore integrates other GIS programs such as the System for Automated Geoscientific Analyses (SAGA) GIS and the Geographic Resources Analysis Support System (GRASS) GIS within a single software environment. This and its QGIS Python API makes it a perfect candidate for console-based geoprocessing. By establishing an interface, the R package **RQGIS** makes it possible to use QGIS as a geoprocessing workhorse from within R. Compared to other packages building a bridge to GIS (e.g., **rgrass7**, **RSAGA**, **RPyGeo**), **RQGIS** offers a wider range of geoalgorithms, and is often easier to use due to various convenience functions. Finally, **RQGIS** supports the seamless integration of Python code using **reticulate** from within R for improved extendability.

Introduction

Defining a GIS as a system for the analysis, manipulation and visualization of geographical data (Longley et al., 2011), one could argue that R has become a GIS (Bivand et al., 2013). In great part this is thanks to packages that provide spatial classes and algorithms coded in and for R (despite this these packages might also link to other software outside of R). These include **maptools** (Bivand and Lewin-Koh, 2017), **raster** (Hijmans, 2017), **sp** (Bivand et al., 2013) and **sf** (Pebesma, 2017). Further packages even extend R's GIS capabilities through advanced mapping, e.g., **mapview** (Appelhans et al., 2017) and **mapmisc** (Brown, 2016), and routing, e.g., **osmar** (Eugster and Schlesinger, 2013) and **dodgr** (Padgham and Peutschnig, 2017), among others. Despite this, native R (in the sense of coded in and for R) lacks fundamental GIS capabilities. GIS topology and topological operations are only partially (**RArcInfo**, Gómez-Rubio and López-Quílez, 2005) or indirectly available via **rgrass7** (Bivand, 2017). Furthermore, R is neither a spatial database management system nor especially good at the manipulation of large data sets (Ripley et al., 2016). Hence, computationally demanding GIS operations (point cloud processing, overlay operations on 'big' spatial data) executed in R may be rather slow. Performance and scalability, of course, depend on the computer hardware, and cloud computing may eventually alleviate or even settle this problem. Yet, most R users most likely still work on a local machine. What is more, R is lacking a number of fundamental GIS operations such as the derivation of various terrain attributes from a digital elevation model (DEM). And the same is true for 3D data visualization and voxel processing (Hengl et al., 2015). Finally, though interactive tasks such as digitizing of geodata have become possible within R very recently (**mapedit**, Appelhans and Russell, 2017), extensive manual editing is better done with the help of a GIS.

Many of R's geospatial shortcomings could potentially be addressed through R programming directly. However, R was designed from the very beginning as an interactive interface to the algorithms

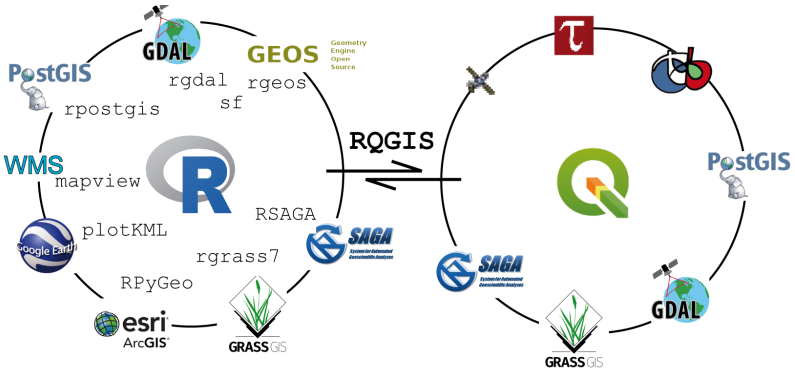

Figure 1: The R-interface to geospatial software - geospatial libraries, Desktop GIS, geobrowsers as well as web mapping and the position of **RQGIS** (left circle; WMS: Web Mapping Service). QGIS and corresponding third-party providers (right circle, the upper three symbols correspond to (from left to right): LiDAR tools, TauDEM, Orfeo Toolbox.

of other software (Chambers, 2016). Hence, it is unnecessary and even counterproductive to duplicate the functionality provided by an existing dedicated software with an expert developer and user community as long as there is a way to access it from within R. Therefore, it is barely surprising that numerous R packages provide access to third-party geoprocessing tools (Figure 1), only some of which will be discussed here. **rgdal** (Bivand et al., 2017) accesses the geospatial data abstraction library (GDAL/OGR) (GDAL Development Team, 2017). **rgeos** (Bivand and Rundel, 2017) is an interface to geometry engine - open source (GEOS, GEOS Development Team, 2017), which opens the way to GIS vector operations. However, GEOS performance is somewhat limited. Think, for instance, of the spatial union of all US American census tracts and postal code layers, and it may be quite possible that `rgeos::gUnion` may take a very long time. The successor of the **sp** package, package **sf** combines the functionality of **sp** (spatial classes), **rgdal** (here: import/export of spatial vector data) and **rgeos** (geometrical operations) in just one package. Note also that GEOS is a C API for topology operations on geometries. Consequently, it expects topologically correct data. To make sure that our geodata lives up to topological expectations in general, our best approach is probably through another third-party integration, namely R-GRASS (Bivand, 2007, 2017). Additionally, GRASS GIS comprises a large suite of vector and raster functions. Basically, the user has to set up a spatial database before being able to use GRASS's geoprocessing utilities (Neteler and Mitasova, 2008). Hence, less experienced GIS users will likely prefer faster-to-use GIS interfaces also providing extensive geoprocessing capabilities. In particular, **RSAGA** (Brenning et al., 2008) integrates R with SAGA (Conrad et al., 2015) and **RPyGeo** (Brenning, 2012b) provides an interface to ArcGIS (ESRI, 2017), which is probably still the most popular GIS environment in the world with >1 million users and the greatest market share among proprietary GIS (Longley et al., 2011).

What has been missing, however, is an R interface to one of the most widely used open-source GIS, QGIS (QGIS Development Team, 2017; Graser and Olaya, 2015). So far, the QGIS processing toolbox provided only the opposite interface by letting the user integrate R scripts as a user-defined 'tool' in QGIS. This is fine for people unwilling to use R directly. However, interfacing from R to QGIS has multiple benefits to the R user community. First and foremost, native QGIS geoalgorithms are now available from within R for the first time. Moreover, it is a special feature of QGIS that it acts as an umbrella integrating various other GIS power houses under its hood. These include SAGA, GRASS, GDAL, the Orfeo Toolbox (Inglada and Christophe, 2009), TauDEM (Tarboton and Mohammed, 2017) and additional tools for light detection and ranging (LIDAR) data (Rapidlasso, 2017). **RQGIS** (Muenchow and Schratz, 2017) brings this incredibly powerful geoprocessing environment to the R console in just one package. This, however, does not mean that specialized packages such as **RSAGA** and **rgrass7** (Bivand, 2007) will become obsolete, as discussed later. **RQGIS** also aims to be user-friendly by automatically retrieving GIS function parameter names and corresponding default values as well as supporting R named arguments for geoalgorithm parameters through the ellipsis argument.

In general, R–GIS interfaces open the way to extremely powerful and innovative statistical geoprocessing as for example shown by Brenning (2008), Hengl et al. (2010), Muenchow et al. (2012), Vanselow and Samimi (2014), Brenning et al. (2015) Mergili et al. (2015), Mergili and Kerschner (2015), Poggio and Gimona (2015) and Zandler et al. (2015). In this paper we will first introduce the general architecture and main features of the **RQGIS** package. We will then demonstrate the application of this integrated scientific programming approach with an ecological example. Subsequently, we will show how to easily complement and extend **RQGIS** with Python programming, especially PyQGIS (Sherman, 2014). In our discussion, we will finally compare and contrast **RQGIS** with other approaches to R–GIS integration, and provide an outlook and motivation for future developments.

Introducing the RQGIS package

Basic concepts

The **RQGIS** package utilizes the QGIS Python API in order to access QGIS modules. To successfully run the QGIS Python API, **RQGIS** first sets up all required environment variables (Figure 2). And secondly, it establishes a tunnel to Python using **reticulate** (Allaire et al., 2017) - a package providing an R interface to Python. The older package **rPython** (Bellosta, 2015) is similar to **reticulate**, however, it is only available for Unix-based systems which is why we had to dismiss it as an option for **RQGIS**. With **reticulate**, we set up the Python environment only once, and use the resulting tunnel to exchange functions and objects between R and Python seamlessly.

We can divide **RQGIS** roughly into two major components:

- The Python code ('python_funs.py' located in 'inst/python' of **RQGIS**) defines a Python class named `"RQGIS"` with methods to be called during the geoprocessing. Defining an own class has

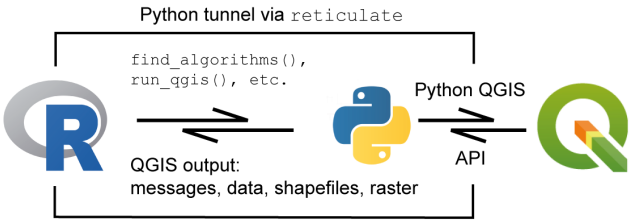

Figure 2: Conceptual model of how RQGIS calls QGIS from within R.

the additional benefit that it becomes highly unlikely that (advanced) users interacting with the QGIS Python API accidentally overwrite some of our predefined methods.

- The 'processing.R' file (found in the 'R' folder of **RQGIS**) actually establishes the QGIS Python interface, and lets the user run QGIS from within R. The most important functions are (see also Usage for a detailed description):
    1. `open_app()` to establish a tunnel to Python and a QGIS custom application
    2. `find_algorithms()` to retrieve the QGIS command-line names for all available geoalgorithms
    3. `open_help()` and `get_args_man()` to access help resources as well as function arguments and default values
    4. `run_qgis()` to call QGIS geoalgorithms from within R

The most notable features of **RQGIS** are:

- For the first time, native QGIS algorithms are available from within R.
- Additionally, **RQGIS** provides access to hundreds of third party geoalgorithms including GDAL, GRASS GIS and SAGA GIS. In the future many more integrations can be expected. For instance, there is already a plugin providing access to PostGIS geoprocessing tools (clip, dissolve, distance, etc.) available in the QGIS processing toolbox (https://plugins.qgis.org/plugins/postgis_geoprocessing/).
- R users can stay in their preferred programming language without having to touch Python.
- Convenient access to QGIS help resources facilitates the geoprocessing work flow. While `open_help` accesses the QGIS online help for a specific geoalgorithm, `get_args_man()` retrieves function arguments and their default values.
- `run_qgis()` also accepts `"sf"`, `"sp"` and `"raster"` objects as arguments. Similarly, users may directly load the QGIS output into R by setting `load_output` to `TRUE` when using `run_qgis()`.

Usage

Since **RQGIS** is an interface to various GIS software packages, the user needs to install this software beforehand. To facilitate the installation process we have written an installation guide, see http://jannes-m.github.io/RQGIS/articles/install_guide.html. Or after having installed the package, one can also access the corresponding vignette by typing:

```
vignette("install_guide", package = "RQGIS")
```

We will demonstrate the usage of **RQGIS** by showing how to compute the plan and tangential curvatures of a digital elevation model (DEM). The first thing to do is to make sure, that all paths are set correctly to successfully run the Python API from within R. Function `set_env()` facilitates this since the user only needs to specify the root path to the QGIS installation. If the root path remains unspecified, `set_env()` tries to be smart by checking the default QGIS installation directories. If this is unsuccessful, `set_env()` will try to find the QGIS installation on the computer which may be time-consuming especially on Windows machines. A much faster way is to explicitly indicate the root path. For Windows this might look like this '`qgis_env <-set_env(root = 'C:/OSGEO4~1')`'.

Subsequently, `set_env()` finds all required paths. Virtually all subsequent **RQGIS** functions require the output list of `set_env()`. This is why, **RQGIS** automatically caches the output of `set_env()`, and reuses it when required by another function later on. To establish a tunnel to the QGIS Python API, we run `open_app()`. Explicitly, the function sets all necessary paths (e.g., path to the QGIS Python binary) to successfully run QGIS, and secondly opens a QGIS custom application (i.e., outside of the QGIS GUI interface) while importing necessary Python modules.

```
library("RQGIS")
set_env()
open_app()
```

Running `set_env()` and `open_app()` is optional here since all subsequent functions dependent on their output will run them automatically in case they have not been executed before. To work in a reproducible manner, and to find out which QGIS and third-party GIS versions we are using, we execute:

```
## $root
## [1] "C:/OSGeo4W64"
##
## $qgis_prefix_path
## [1] "C:/OSGeo4W64/apps/qgis"
##
## $python_plugins
## [1] "C:/OSGeo4W64/apps/qgis/python/plugins"

info_r <- version
info_qgis <- qgis_session_info()
c(platform = info_r$platform, R = info_r$version.string, info_qgis)
## $platform
## [1] "x86_64-w64-mingw32"
##
## $R
## [1] "R version 3.4.2 (2017-09-28)"
##
## $qgis_version
## [1] "2.18.14"
##
## $gdal
## [1] "2.2.2"
##
## $grass6
## [1] "6.4.3"
##
## $grass7
## [1] "7.2.2"
##
## $saga
## [1] "2.3.2"
```

Continuing with our analysis, we need to find out the command-line name of a geoalgorithm available in QGIS that computes the curvatures from a DEM. `find_algorithms()` lets the user use regular expressions to search for a function which contains the search terms in its short description. Leaving the `search_term`-argument empty, will return all available geoalgorithms. Here, we assume that the function we are looking for contains the word 'curvature' in its short description. Setting `name_only` to `TRUE` gives back the name of the geoalgorithm instead of its name plus the corresponding short description.

```
find_algorithms(search_term = "curvature",
                name_only = TRUE)
## [1] "grass7:r.slope.aspect"             "saga:curvatureclassification"
## [3] "saga:slopeaspectcurvature"         "saga:upslopeanddownslopecurvature"
## [5] "grass:r.slope.aspect"
```

Several functions are available for our task, we will go on with function `grass7:r.slope.aspect`. To familiarize ourselves with the function, we can access its online help by calling (not shown):

```
open_help(alg = "grass7:r.slope.aspect")
```

Next, we would like to know how to use a specific geoalgorithm. `get_usage()` prints the parameters and default values for a given geoalgorithm to the console.

```
get_usage(alg = "grass7:r.slope.aspect")
## ALGORITHM: r.slope.aspect - Generates raster layers of slope
```

```
## aspect
## curvatures and partial derivatives from a elevation raster layer.
##         elevation <ParameterRaster>
##         format <ParameterSelection>
##         precision <ParameterSelection>
##         -a <ParameterBoolean>
##         zscale <ParameterNumber>
##         min_slope <ParameterNumber>
##         GRASS_REGION_PARAMETER <ParameterExtent>
##         GRASS_REGION_CELLSIZE_PARAMETER <ParameterNumber>
##         slope <OutputRaster>
##         aspect <OutputRaster>
##         pcurvature <OutputRaster>
##         tcurvature <OutputRaster>
##         dx <OutputRaster>
##         dy <OutputRaster>
##         dxx <OutputRaster>
##         dyy <OutputRaster>
##         dxy <OutputRaster>
##
##
## format(Format for reporting the slope)
##         0 - degrees
##         1 - percent
## precision(Type of output aspect and slope layer)
##         0 - FCELL
##         1 - CELL
##         2 - DCELL
```

`get_args_man()` lets us retrieve automatically a corresponding parameter-argument list. Setting the `options` parameter to `TRUE` (the default) automatically chooses the default value from a list of possible options for a parameter. This is always the first option which is in accordance with the QGIS GUI behavior. To make the user aware of the automatically chosen options, `get_args_man()` prints the corresponding values to the console.

```
params <- get_args_man(alg = "grass7:r.slope.aspect", options = TRUE)
## Choosing default values for following parameters:
## format: 0
## precision: 0
## See get_options('grass7:r.slope.aspect') for all available options.
```

`grass7:r.slope.aspect` has 17 parameters. Here, we only show the first six parameters for brevity.

```
head(params)
## $elevation
## [1] "None"
##
## $format
## [1] "0"
##
## $precision
## [1] "0"
##
## $`-a`
## [1] "True"
##
## $zscale
## [1] "1.0"
##
## $min_slope
## [1] "0.0"
```

Next, we specify the required arguments. Of course, `grass7:r.slope.aspect` expects a spatial input file, here a DEM. Conveniently, `run_qgis()` accepts as input both a path to a spatial file stored on disk or a spatial object residing in R's environment (specifically `"raster"`-, `"sp"`- and `"sf"`-objects).

Here, we use a DEM that comes as an example file with the **RQGIS** package. Note that `run_qgis()` simply saves spatial input files to a temporary output location. Hence, if the file already exists on disk, it is much more efficient to indicate the path to the file instead of loading it into R and letting `run_qgis()` export it again. By contrast, indicating output paths is not strictly necessary. If an output path parameter equals `None` (QGIS default), QGIS automatically creates an output file which it saves to a temporary processing output folder. In the code chunk below, we specifically indicate curvature outputs while keeping the default, i.e. `None`, for the remaining output parameters `slope`, `aspect`, `dx`, `dy`, `dxx`, `dyy` and `dxy` (check out the GRASS function documentation for more information, i.e. run 'open_help("grass7:r.slope.aspect")'). `run_qgis()` prints all output paths to the console if `show_output_paths` is set to TRUE. Here, we turn this behavior off for two reasons. First, we are not interested in the output paths of the seven terrain attributes we left unspecified. Secondly, we have explicitly specified the curvature output paths, i.e., we already know the corresponding output locations. We recommend to specify those output paths which are relevant to the analysis. Manual specification has the additional benefit that we can indicate a specific file format (QGIS default for raster data sets is in most cases .tif, but we might want to use e.g., .asc or SAGA's .sdat). Additionally, loading QGIS output directly back into R (`load_output`-parameter) only works with output paths specified by the user.

```
data("dem", package = "RQGIS")
out <- run_qgis(alg = "grass7:r.slope.aspect",
                elevation = dem,
                pcurvature = file.path(tempdir(), "pcurv.tif"),
                tcurvature = file.path(tempdir(), "tcurv.tif"),
                show_output_paths = FALSE,
                load_output = TRUE)
```

Note that we used R named arguments in `run_qgis()`, i.e., we assigned values or objects to the parameters whose names we have identified with the help of `get_args_man()` or `get_usage()`. We could replace the R named arguments also by a parameter-argument list. Remember that we have already created a parameter-argument list named `params` using `get_args_man()` (see above):

```
params$elevation <- dem
params$pcurvature <- file.path(tempdir(), "pcurv.tif")
params$tcurvature <- file.path(tempdir(), "tcurv.tif")
out <- run_qgis(alg = "grass7:r.slope.aspect",
                params = params,
                load_output = TRUE,
                show_output_paths = FALSE)
class(out)
## [1] "list"
names(out)
## [1] "pcurvature" "tcurvature"
```

However, providing R named arguments and a parameter-argument list is not possible, and `run_qgis()` will complain telling the user to user either one of these but not a mixture. Since we have set `load_output` to `TRUE`, `run_qgis()` automatically loads the QGIS output into R. In this case, the object `out` is a list with two `"raster"` objects. If we only had specified one output raster, `out` would have been a `"RasterLayer"` object. In case the output is a vector layer, `run_qgis()` will load it as an `"sf"` object. To have a look at the output, we can execute following code (not shown):

```
library("raster")
plot(stack(out))
```

Concerning the handling of parameter-argument pairs, `run_qgis()` uses `get_args_man()` through `pass_args()` in the background to access the default values of all missing arguments if available. If the user accidentally omits a required argument, `run_qgis()` will return an error message that informs about the missing argument. The help documentation of `pass_args()` presents a detailed list of argument checks that are run before executing `run_qgis()`.

Experienced GRASS users may wonder if there is a need to specify the `GRASS_REGION_PARAMETER`. `pass_args()` determines this parameter automatically based on the spatial layers provided as input by the user (in our example above this is `dem`). However, the `GRASS_REGION_PARAMETER` can also be set manually (see the `pass_args()` documentation for details).

Figure 3: The study area Mount Mongón in northern Peru (Landsat image: path 9, row 67, acquisition date 09/22/2000; USGS 2016).

Ecological example: combining geocomputing and statistics

To show the utility of **RQGIS** in real-world applications, we combine QGIS functionality with R's modeling and (geo-)statistical capabilities in an ecological study in the coastal desert of northern Peru (Figure 3). Despite the extreme aridity of the Mount Mongón region (200-1100 m asl), this area is the habitat of a distinct flora and fauna (Dillon et al., 2003). The unique vegetation, locally termed *lomas*, mainly survives due to heavy fog during the austral winter months (Muenchow et al., 2013b,c).

Linking species richness to environmental predictors along gradients is a key topic of community ecology and biogeography (Muenchow et al., 2017), and the fundamental basis for conservation planning (Pomara et al., 2012). In our use case, we model vascular plant species richness along an altitudinal gradient as a function of topographic and remotely sensed variables by means of count regression. In the following, we will show a simplified version of an analysis by Muenchow et al. (2013b).

Before running a Poisson model, we need to compute terrain attributes from a DEM. These will serve as predictors to model species richness. To account for the unimodal relationship between elevation and species richness, we use a second-order orthogonal polynomial function (Figure 4, Panel Elevation). In the original paper, we dropped the least significant variables one at a time until only significant predictors remained (elevation and its squared term, catchment slope, catchment area and the normalized difference vegetation index, NDVI). Due to space constraints and demonstration purposes, we will simply use the final model here (for details see Muenchow et al., 2013b). Instead of calculating all predictors used in the original paper, we only show how to derive selected geospatial predictors using **RQGIS**, namely tangential and profile curvature, catchment slope and catchment area.

Terrain attributes

The numerical representation as well as the analysis of the land surface is frequently referred to as terrain analysis and terrain modeling. The corresponding surface-characterizing measures are known as terrain attributes (Pike et al., 2008). Terrain attributes play an important role, for example, in pedometrics (McBratney et al., 2000), precision agriculture (Kühn et al., 2009), geomorphometry (Pike et al., 2008) and ecology (Muenchow et al., 2013a). They are frequently related to slope stability (Montgomery and Dietrich, 1994; Muenchow et al., 2012). Additionally, they are proxies for variables representing water availability such as soil moisture, soil texture or moisture-holding capacity, among others (Brenning, 2008; Franklin et al., 2000; Muenchow et al., 2013c). Especially the latter is of utmost importance regarding plant distribution in a desert environment. While GIS can easily calculate terrain attributes, R is rather limited in this respect. However, without terrain attributes, we would neither be

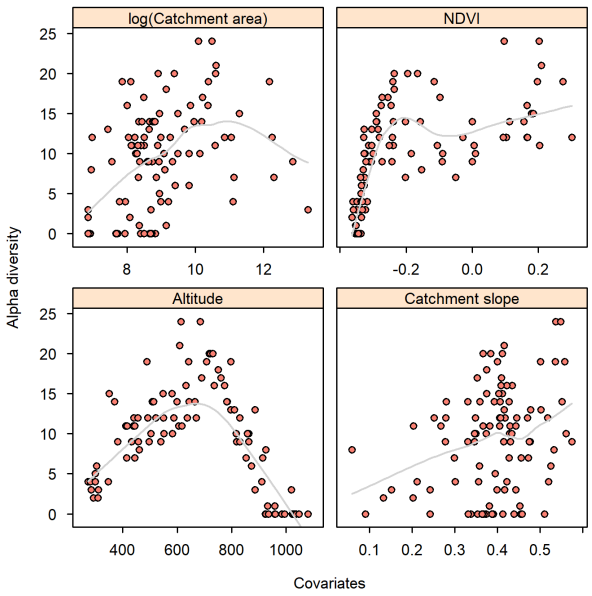

Figure 4: Scatterplot of all predictors used in the Poisson model against the response variable. Each dot represents a visited plot on Mount Mongón. The gray line smoother should aid visual inspection.

able to model nor predict species richness appropriately.

First we would like to use SAGA to remove local depressions from the DEM, since these may be artifacts. For this, we use the `[1] Fill Sinks` method. Note that you also may use numbers for specifying the option, here, the fill sinks method corresponds to 1.

```
get_usage("saga:sinkremoval")
## ALGORITHM: Sink removal
##         DEM <ParameterRaster>
##         SINKROUTE <ParameterRaster>
##         METHOD <ParameterSelection>
##         THRESHOLD         <ParameterBoolean>
##         THRSHEIGHT <ParameterNumber>
##         _RESAMPLING <ParameterSelection>
##         DEM_PREPROC <OutputRaster>
##
##
## METHOD(Method)
##         0 - [0] Deepen Drainage Routes
##         1 - [1] Fill Sinks
## _RESAMPLING(Resampling method)
##         0 - Nearest Neighbour
##         1 - Bilinear Interpolation
##         2 - Bicubic Spline Interpolation
##         3 - B-Spline Interpolation
run_qgis(alg = "saga:sinkremoval",
         DEM = dem,
         METHOD = "[1] Fill Sinks",
         DEM_PREPROC = file.path(tempdir(), "sdem.sdat"),
         show_output_paths = FALSE)
```

Next, we compute the catchment area (or upslope contributing area) and its mean slope from the preprocessed DEM. These raster datasets are calculated while calculating the 'Saga wetness index', which is also frequently used as a predictor in ecological studies. To calculate the catchment slope instead of the local slope we set the `SLOPE_TYPE` argument to 1. The names of the output rasters are 'carea.sdat' and 'cslope.sdat' both of which will be stored in `tempdir()`.

```
get_usage("saga:sagawetnessindex")
## ALGORITHM: Saga wetness index
```

```
##         DEM <ParameterRaster>
##         SUCTION <ParameterNumber>
##         AREA_TYPE <ParameterSelection>
##         SLOPE_TYPE <ParameterSelection>
##         SLOPE_MIN <ParameterNumber>
##         SLOPE_OFF <ParameterNumber>
##         SLOPE_WEIGHT <ParameterNumber>
##         _RESAMPLING <ParameterSelection>
##         AREA <OutputRaster>
##         SLOPE <OutputRaster>
##         AREA_MOD <OutputRaster>
##         TWI <OutputRaster>
##
##
## AREA_TYPE(Type of Area)
##         0 - [0] absolute catchment area
##         1 - [1] square root of catchment area
##         2 - [2] specific catchment area
## SLOPE_TYPE(Type of Slope)
##         0 - [0] local slope
##         1 - [1] catchment slope
## _RESAMPLING(Resampling method)
##         0 - Nearest Neighbour
##         1 - Bilinear Interpolation
##         2 - Bicubic Spline Interpolation
##         3 - B-Spline Interpolation
run_qgis(alg = "saga:sagawetnessindex",
         DEM = file.path(tempdir(), "sdem.sdat"),
         SLOPE_TYPE = 1,
         SLOPE = file.path(tempdir(), "cslope.sdat"),
         AREA = file.path(tempdir(), "carea.sdat"),
         show_output_paths = FALSE)
```

To have a look at the output rasters, we can run (not shown):

```
library("dplyr")
library("raster")
file.path(tempdir(), c("cslope.sdat", "carea.sdat")) %>%
  raster::stack(.) %>%
  plot
```

We furthermore need to apply some transformations to these raster files for improved interpretability. On the one hand, we convert slope angle from radians to degrees.

```
library("raster")
cslope <- raster(file.path(tempdir(), "cslope.sdat"))
cslope <- cslope * 180 /pi
```

On the other hand, we divide the catchment area variable by one million to change the unit from m^2 to km^2. Furthermore, we transform it logarithmically since it is strongly skewed to the right.

```
carea <- raster(file.path(tempdir(), "carea.sdat"))
log_carea <- log(carea / 1e+06)
```

Apart from the catchment area and catchment slope, our final model requires the NDVI as an indicator of vegetation properties. To calculate it, we would have to perform pixel-by-pixel arithmetic operations on raster data sets representing the Landsat satellite's spectral bands three (red) and four (near infrared). These so-called map algebra operations are available through **raster** and the QGIS modules `saga:rastercalculator` and `grass:r.mapcalculator`; however, the **RQGIS** package already provides the NDVI raster as an example dataset. Therefore, we merely have to attach the data to our workspace.

```
data("ndvi", package = "RQGIS")
```

To account for the nonlinear unimodal relationship between elevation and species richness, we need two rasters representing the second-order orthogonal polynomials of the original DEM. First, we

convert the elevation unit from m to km by dividing the original DEM raster by 1000. Next, we use R's built-in `poly()` function to calculate the orthogonal polynomials for each pixel in our DEM raster to avoid collinearity among the predictors. Before saving the resulting rasters and the catchment slope, the catchment area and the NDVI to a temporary output location, we apply the `crop()` function from the **raster** package to ensure that all rasters share the same extent.

```
data("dem", package = "RQGIS")
dem <- dem / 1000
my_poly <- poly(values(dem), degree = 2)
dem1 <- dem2 <- dem
values(dem1) <- my_poly[, 1]
values(dem2) <- my_poly[, 2]
for (i in c("dem1", "dem2", "log_carea", "cslope", "ndvi")) {
  tmp <- crop(get(i), dem)
  writeRaster(x = tmp,
              filename = file.path(tempdir(), paste0(i, ".asc")),
              format = "ascii",
              prj = TRUE,
              overwrite = TRUE)
}
```

After creating these raster datasets, we would like to extract their attributes to the randomly sampled points. Conveniently, RSAGA's `pick.from.ascii.grids()` function accepts multiple rasters for parallel attribute extraction. Note that `pick.from.ascii.grids()` is a pure R function, i.e., it runs without accessing SAGA. Here, we only extract the predictor variables needed in the final model (see above). The `"sf"` object `random_points` refers to randomly sampled points we visited in the field. Unfortunately, `pick.from.ascii.grids()` does not accept spatial point objects as input; instead we have to provide it with a `"data.frame"` and indicate which columns refer to the coordinates. In the `file` argument we specify the raster files from which we would like to extract values to our points. The output columns in `vals` will be named like the input rasters.

```
library("dplyr")
data("random_points", package = "RQGIS")
random_points[, c("x", "y")] <- sf::st_coordinates(random_points)
raster_names <- c("dem1", "dem2", "log_carea", "cslope", "ndvi")
vals <- RSAGA::pick.from.ascii.grids(data = as.data.frame(random_points),
                                     X.name = "x",
                                     Y.name = "y",
                                     file = file.path(tempdir(), raster_names),
                                     varname = raster_names)
dplyr::select(vals, -geometry) %>%
  head(., 3)
##   id spri      x       y     dem1    dem2 log_carea cslope    ndvi
## 1  1    4 797179 8932755 -0.01049 0.01268  -1.21800  21.18 -0.3603
## 2  2    4 796749 8932621 -0.01019 0.01163   0.04145  13.02 -0.3488
## 3  3    3 796816 8932739 -0.01008 0.01124  -0.48148  23.70 -0.3396
```

Modeling species richness and predictive mapping

To model species richness, which is a count variable, we naturally opt for a Poisson regression. Overall, spatial count regression models are popular across many fields including epidemiology (Fernandez et al., 2012), demography (Chien et al., 2016), criminology (Jones-Webb and Wall, 2008), remote sensing (Comber et al., 2016) and ecology (Moreno-Fernández et al., 2015). Since we observed a unimodal nonlinear relationship between species richness and elevation, elevation enters the model as a second-order polynomial function.

```
fit <- glm(formula = spri ~ dem1 + dem2 + cslope + ndvi + log_carea,
           data = vals,
           family = "poisson")
```

To spatially predict species richness (Figure 5), we apply the estimated beta coefficients to the input rasters. **raster**'s `predict` function does this by accepting the fitted model and the predictor rasters as input. Finally, the `crop`-function ensures that the predictions are restricted to the study area.

```
library("sf")
library("raster")
```

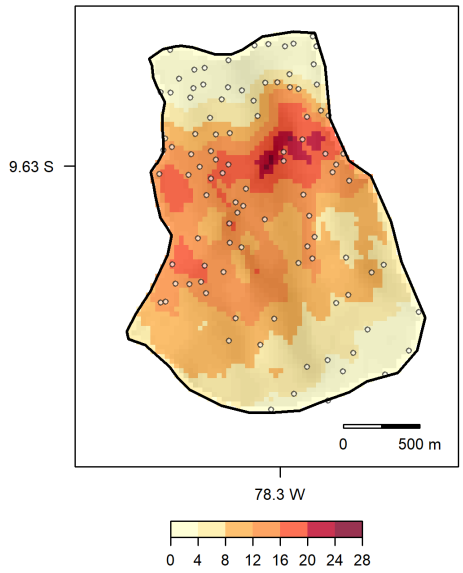

Figure 5: Prediction map of species richness. The points represent the visited plots.

```
raster_names <- c("dem1.asc", "dem2.asc", "log_carea.asc", "cslope.asc",
                  "ndvi.asc")
s <- stack(x = file.path(tempdir(), raster_names))
pred <- predict(object = s,
                model = fit,
                fun = predict,
                type = "response")
pred <- crop(x = pred,
             y = as(random_points, "Spatial"))
# plot the output (shown in Figure 5)
plot(pred)
plot(st_geometry(random_points), add = TRUE)
```

Note that an alternative to `raster::predict()` is **RSAGA**'s `multi.local.function()` in conjunction with `grid.predict()` (see Brenning, 2008).

The model reached a good goodness-of-fit (explained deviance divided by null deviance) of 0.78. The most important variable in predicting species richness was elevation and its squared term. In our interpretation, variation with elevation mainly relates to differences in water availability. Humidity, and thus species richness, is greatest just below the temperature inversion (ca. 750–850 m). For a more detailed interpretation of the model and its predictors, refer to Muenchow et al. (2013b).

Extending RQGIS through Python and PyQGIS

In this section we would like to show examplarily how one can easily extend **RQGIS** through Python and especially PyQGIS. As explained in sections Basic concepts and Usage, **RQGIS** uses the **reticulate** package to establish a tunnel to the QGIS API. To find out which Python binary is in use we run the `py_config()` function of the **reticulate** package. When using Windows, please do so only after having run `open_app` before, since this sets all necessary paths to run the QGIS Python binary. Otherwise `py_config()` will be unable to find it, in which case it most likely would use another Python binary (e.g., the one shipped with Anaconda) if available. Additionally, `open_app()` imports various necessary Python libraries (among others `osgeo`, `processing` and `qgis`), and attaches our Python RQGIS class (see section Usage).

```
library("reticulate")
py_config()
## python:         C:/OSGeo4W64/bin/python.exe
## libpython:      C:/OSGeo4W64/bin/python27.dll
## pythonhome:     C:\OSGeo4W64\apps\Python27
```

```
## version:        2.7.5 (default, May 15 2013, 22:44:16) [MSC v.1500 64 bit (AMD64)]
## Architecture:   64bit
## numpy:          C:\OSGeo4W64\apps\Python27\lib\site-packages\numpy
## numpy_version:  1.12.1
##
## NOTE: Python version was forced by use_python function
```

We have defined the Python class RQGIS in `python_funs.py` which is part of 'inst/python'. Aside from `set_env()` all functions found in 'R/processing' make extensive use of the Python RQGIS methods. Most of the Python methods have the same names as their counterparts in R. We can access them with `py_run_string()` of the **reticulate**-package. This sends a call to Python, and converts the Python into R output if desired.

```
py_run_string("methods = dir(RQGIS)")$methods
## [1] "__doc__"           "__init__"          "__module__"
## [4] "check_args"        "get_args_man"      "get_options"
## [7] "open_help"         "qgis_session_info"
```

Of course, we can use the Python RQGIS methods directly via **reticulate** which is exactly what the **RQGIS**-package is doing. For example, to find out what the options are for a QGIS geoalgorithm named `qgis:randompointsinsidepolygonsvariable`, we can run:

```
py_cmd <- "opts = RQGIS.get_options('qgis:randompointsinsidepolygonsvariable')"
py_run_string(py_cmd)$opts
## $STRATEGY
## [1] "Points count"   "Points density"
```

Or we can use PyQGIS-functionality directly. For instance, assuming we would like to find out the function parameters of `qgis:randompointsinsidepolygonsvariable`, we can use `alghelp()` from the QGIS Python processing framework (Graser and Olaya, 2015, and see also https://docs.qgis.org/2.8/en/docs/user_manual/processing/console.html). Here, we only show the first 40 characters of the output.

```
py_cmd <- "processing.alghelp('qgis:randompointsinsidepolygonsvariable')"
py_capture_output(py_run_string(py_cmd)) %>%
  substring(., 1, 40)
## [1] "ALGORITHM: Random points inside polygons"
```

Users can easily extend the RQGIS class with additional methods, or they could write their own classes and methods. Also if there is a need to write Python one-liners or make use of some Python functionality, this can easily be done using **RQGIS** in conjunction with **reticulate**. Here, we will present one last example. Frequently, users have asked us if it was possible to also use the QGIS map canvas from within R. Therefore, we provide a proof-of-concept how this could be achieved (see also http://docs.qgis.org/testing/en/docs/pyqgis_developer_cookbook/canvas.html). First of all, we save `random_points` to a temporary output location.

```
data("random_points", package = "RQGIS")
file <- normalizePath(file.path(tempdir(), "points.shp"), winslash = "/",
                      mustWork = FALSE)
sf::st_write(random_points, dsn = file)
```

Before running the subsequent code, make sure that you have already attached the packages **RQGIS** and **reticulate**. Additionally, you need to run `open_app()` first. Then we can create the map canvas, add the previously saved shapefile to it, set the extent to the extent of our shapefile, set the map canvas layer, and finally open a standalone map window (Figure 6). If this does not open a standalone window you might have to run `py_run_string('app.exec_()')` to initialize a Qt event loop which in turn renders the points in a standalone window (pers. comm. Barry Rowlingson). We emphasize that this is only a proof-of-concept, and a rather unstable solution (see section Current and future developments).

```
# create the map canvas
py_run_string("canvas = QgsMapCanvas()")
# import point shapefile
py_run_string(sprintf("layer = QgsVectorLayer('%s', 'points', 'ogr')", file))
# add imported point layer to the map canvas
py_run_string("QgsMapLayerRegistry.instance().addMapLayer(layer)")
```

Figure 6: A very simple example how to access the QGIS map canvas from within R.

```
# set the extent of the map canvas to the extent of the imported shapefile
py_run_string("canvas.setExtent(layer.extent())")
# set the map canvas layer
py_run_string("canvas.setLayerSet([QgsMapCanvasLayer(layer)])")
# open a standalone window
py_run_string("canvas.show()")
# if a standalone window has not already opened, run the next line
# py_run_string("app.exec_()")
```

Discussion

R/GIS-integration: Combining the best of two worlds

In our use case (see section Ecological example: combining geocomputing and statistics) we mainly used QGIS for raster preprocessing and the generation of terrain attributes. Naturally, there are many more geospatial analysis problems that can be solved using the thousands of geoalgorithms that are accessible through **RQGIS** and other R–GIS packages. For instance, SAGA provides more than 600 (Conrad et al., 2015) and GRASS more than 500 functions (`http://grass.osgeo.org/grass72/manuals/`).

For example, apart from relatively simple DEM derivatives (slope angle and orientation), a GIS can also calculate more complex, process-oriented terrain attributes such as the relative slope position or the topographic wetness index (Conrad et al., 2015). Additionally, most GIS can easily extract stream networks (Hengl et al., 2010) and surface roughness (Grohmann, 2004). Somewhat related are geospatial calculations concerned with terrain classification and landform identification (Brenning, 2012a; Rocchini et al., 2013). Physically-based models (e.g., SHALSTAB or Factor of Safety, both available in SAGA GIS) may provide additional insights (Goetz et al., 2011). Of course, GIS also provide an extensive suite of vector processing tools as required, for example, in geomarketing.

As pointed out in the beginning R has its limitations regarding GIS capabilities, but when it comes to statistical analyses, R is the uncontested champion in its field. For instance, instead of using a polynomial function in our use case (see section Modeling species richness and predictive mapping), we could have used a generalized additive model (GAM) with a logarithmic link function to allow for nonlinear relationships between various predictors and species richness (Figure 4). A GAM is the nonlinear extension of a generalized linear model (GLM), and uses smoothing functions to deal with nonlinearity (Hastie, 2017). In ecology, coupling ordination techniques and GAMs is a fruitful approach to spatially predict ecological communities (Muenchow et al., 2013a).

Equally, machine learning algorithms (support vector machines, random forests, etc.) are readily available in R, although these methods may tend to overfit the training data (Brenning, 2005). Overfitting in turn limits a model's ability to spatially predict the response variable. Here, spatial cross-validation through the **sperrorest** package (Brenning et al., 2012) provides an opportunity to assess spatial predictive capabilities.

Frequently, the residuals of spatial models show some form of spatial autocorrelation. This violates the assumption of independent model residuals made by most statistical models (Dormann et al., 2007; Zuur et al., 2009). Violating this assumption can lead to untrustworthy p values, biased coefficients and subsequently poor predictions (Zuur et al., 2009). Fortunately, packages such as **nlme** (Pinheiro et al., 2017) and **mgcv** (Wood, 2017) let the user incorporate various correlation structures into a model. This is most helpful in the presence of temporal and spatial autocorrelation (e.g, Iturritxa et al.,

2015). Sometimes mixed-effect models may also account for autocorrelation since random intercepts allow for correlation within a group (e.g., Peters et al., 2014). Another way of dealing with spatial autocorrelation is e.g., to include auto-regressive correlation structures in a GLM or GAM within a Bayesian modeling approach (Zuur and Ieno, 2017).

To summarize, R offers an incredibly vast suite of advanced statistical and data science methods. On the other hand, QGIS and other GIS software offer a rich suite of geoalgorithms and geocomputational power. Interfacing R with GIS simply combines the best of two worlds for automated statistical geocomputing.

RSAGA and R/GRASS-integration

Compared to the two separate packages **rgrass7** and **RSAGA**, **RQGIS** has the advantage of providing a unified interface to both GRASS and SAGA GIS toolboxes. Moreover, QGIS facilitates the usage of third-party geoalgorithms by automatically converting vector (e.g., shapefiles) and raster formats (e.g., ASCII grid files) into the particular format supported by the third-party module. For instance, SAGA has its own grid format (sgrd-files) and GRASS uses its own database format. Running SAGA or GRASS functions, (R)QGIS automatically converts the input data using `io_gdal()` in the case of SAGA and `v.in.ogr()` or `r.in.gdal()` in the case of GRASS. Though this is extremely user-friendly especially when providing interfaces to various third-party providers, it comes at the prize of increased computing time due to the necessity of multiple format conversions during one geoalgorithm call. Equally user-friendly it the automatic setup of the GRASS environment (projection, region and mapset) through (R)QGIS, if necessary. This certainly facilitates access to GRASS, especially for less experienced GIS users. Finally, **RQGIS** is overall quite user-friendly due to its convenience functions `open_help()` and `get_args_man()` and through its support of R named arguments for geoalgorithm parameters (see sections Basic concepts and Usage).

However, (R)QGIS only integrates a subset of the modules available in SAGA and GRASS GIS. While this fraction is likely to grow in the near future, a full integration of all modules is improbable as it would duplicate functionality (though this of course already has happened) and interface functions that are unnecessary within the QGIS environment, such as the GRASS database functions. If the user's intention is to use GRASS's database management system (DBMS), the direct R–GRASS integration via the **spgrass6** (Bivand et al., 2013) and **rgrass7** packages (Bivand and Neteler, 2000) would be the appropriate path. **RQGIS** does not provide access to this DBMS since the GRASS plugin of the QGIS processing toolbox only allows restricted access to GRASS's DBMS functionality. The use of **rgrass7** also allows the user to operate within a single GRASS session instead of calling a new one for each GRASS command as implicitly done by QGIS.

In the case of SAGA GIS, **RSAGA** has additional benefits. First of all, **RSAGA** provides numerous user-friendly wrapper functions with arguments (and meaningful default values) documented in the R help pages. **RSAGA** also strives to provide unified access to a range of SAGA versions while using, if possible, persistent function and argument names as well as default values. This allows for an easier migration between SAGA versions. At the moment **RSAGA** supports versions 2.0.4–2.2.3, but support for SAGA versions until the current version 6.1 has already been developed, and is currently being tested. By contrast, QGIS 2.14 supports SAGA 2.1.2–2.3.1. With the release of QGIS 2.18.10, this support was limited to the long term SAGA release 2.3.x. Extremely useful are furthermore **RSAGA**'s special geocomputing functions that allow, for example, the application of any user-defined R function to a stack of grids, either locally or within moving windows (functions `multi.focal.function()` and `multi.local.function()`). In conjunction with `grid.predict()`, predict methods of models fitted in R can therefore also be applied to stacks of raster files, as shown in Brenning (2008).

In the end, it will be up to the user to decide whether to use **RQGIS**, **RSAGA**, **rgrass7** or some combination of these, depending on the user's preferences, expertise and tasks at hand. In any case, we would recommend **RQGIS** if a user

- requires mainly the more commonly used SAGA and GRASS functions,
- does not want to be bothered with setting up the GRASS environment,
- does not plan to use GRASS geodatabase capabilities,
- does not want to worry about spatial data format conversions,
- would like to use spatial objects residing in R as input-arguments, and load GIS output automatically into R,
- cherishes the flexibility of seamlessly integrating QGIS, GRASS, SAGA and other third-party software (GDAL/OGR, Orfeo Toolbox, LAStools, TauDEM) within a single geoprocessing workflow in R.

Accessing ArcGIS through RPyGeo

The integration of GIS functionality into R is not limited to the mentioned open-source GIS software, but also includes the global leader in commercial GIS software (Longley et al., 2011), ArcGIS, through the **RPyGeo** package (Brenning, 2012b). This package piggybacks on ArcGIS's own Python interface to ArcGIS functions, or 'tools'. **RPyGeo** generates Python code that calls ArcGIS. Some of the challenges currently include

- latency times related to launching Python and ArcGIS before each GIS call,
- passing data and files between R and ArcGIS,
- error tracking based on Python and/or ArcGIS error messages,
- interpretation of ArcGIS help pages from the perspective of a geoprocessing interface function in R.

While some of these limitations may be overcome in future releases of **RPyGeo** and ArcGIS, the growing interest in integrating R and ArcGIS is also evident from recent efforts to provide access to R from within ArcGIS (`https://r-arcgis.github.io/`). This so-called R–Bridge allows users to

- retrieve data from ArcGIS geodatabases into R as `"sp"` objects, and export R data back into ArcGIS geodatabases,
- run R code within ArcGIS as a user-defined 'tool'. This is similar to QGIS's ability to integrate R scripts as user-defined GIS modules in the Processing toolbox.

While this connectivity, in principle, goes both ways, the integration of ArcGIS into R through the R–Bridge is currently limited to data import/export, which is complementary to **RPyGeo**'s capability of executing ArcGIS modules from within R.

Current and future developments

There are several interesting developing directions in the R-spatial world and for **RQGIS**. For example, we have been asked multiple times to include the QGIS mapping widget within R for fast interactive visualization and styling of multiple layers. In section Extending **RQGIS** through Python and PyQGIS we have shown that this is theoretically possible. However, mixing R and Qt events seems to cause frequently trouble (also pers. comm. Barry Rowlingson). For instance, when running `QgsMapCanvas()` twice or enlarging the standalone window manually (Figure 6), one runs a high risk of crashing the current R session. Therefore, Kevin Stadler, Barry Rowlingson and Julia Wagemann have opted for a different approach realized within a Google Summer of Code Project. In this project they wrote the QGIS Plugin Network API which enables the user to access the QGIS API via a HTTP interface from another language capable of making HTTP calls (such as R). Along with the sibling R package **qgisremote** (`https://qgisapi.gitlab.io/qgisremote/index.html`) this allows R users to seamlessly exchange spatial data between R and QGIS. For example, one can add vector and raster layers from within R to the QGIS map canvas, interactively edit them (such as adding new points or changing polygon vertices), and load the results back into R. Similarly, the packages **mapview** (build on top of **leaflet**, Cheng et al., 2017) and **mapedit** lets the user interactively visualize and edit spatial objects on leaflet maps. The advantage of **qgisremote** over these two packages is that the QGIS map canvas probably is better able to handle larger quantities of spatial data compared to leaflet, and that it provides the user with the full power of a Desktop GIS for manually editing spatial data (trace, snap, etc.). Nevertheless, **mapview** is a great tool for interactive visualization, and **mapedit** is a good alternative for small and quick modifications of spatial vector data within R. Coming back to the user request to include the QGIS mapping widget into **RQGIS**, we can state that **qgisremote** already filled this gap perfectly. In summary, **RQGIS** and **qgisremote** complement one another. The first gives an R user direct access to the QGIS processing engine, and the latter hands an R user the power of a Desktop GIS graphical user interface.

Adding new functionalities to **RQGIS** will generally include Python programming. Since QGIS migrates from Python 2 to Python 3 by the end of 2017, it is probably best to postpone major **RQGIS** extensions until after the migration. To guarantee a smooth transition, and to offer the possibility to work either with QGIS 2 or QGIS 3, we have designed **RQGIS** in such a way that we ideally would merely have to add a Python 3 script to 'inst/python' to make **RQGIS** also work with QGIS 3. In the case of a bumpier transition than anticipated, **RQGIS** users may rest assured that **RQGIS** will work in any case with the QGIS long term release 2.18 which will be further developed and regularly updated (see `https://www.qgis.org/en/site/getinvolved/development/roadmap.html#release-schedule`).

Another envisaged **RQGIS** update includes the support for `"stars"` classes (see `https://github.com/r-spatial/stars`). **stars** aims to mainly extend the **raster** package.

Conclusions

Combining R and GIS software creates a powerful environment for advanced statistical geocomputing. **RQGIS** makes this also possible with QGIS—one of the most-widely used open-source GIS, which is therefore probably also very appealing to R users. Conveniently, **RQGIS** offers a unified interface to various desktop GIS (SAGA, GRASS, etc.) that are integrated into QGIS. The use of GIS tools is facilitated through auxiliary functions for the automatic retrieval of function arguments and their default values, the support of R named arguments in `run_qgis()`, the seamless exchange of spatial data types, and the quick access of the online help for any QGIS geoalgorithm.

Acknowledgments

We greatly appreciate the work of the QGIS development and the R core team. We are also greatly indebted to Dr. Rainer Krug (University of Zurich) who not only reviewed our manuscript two times but also improved it with his valuable suggestions. Of course, we thank Dr. Roger Bivand, our editor, for handling our manuscript. We are also grateful to Dr. Tomislav Hengl (SRIC – World Soil Information, Wageningen), and an anonymous reviewer for valuable feedback on a previous manuscript version. Finally, we would like to thank Barry Rowlingson (Lancaster University) for fruitful discussions on Python, QGIS and R.

Bibliography

J. Allaire, Y. Tang, and M. Geelnard. *reticulate: R Interface to Python*, 2017. URL https://CRAN.R-project.org/package=reticulate. R package version 1.3.1. [p410]

T. Appelhans and K. Russell. *mapedit: Interactive Editing of Spatial Data in R*, 2017. URL https://CRAN.R-project.org/package=mapedit. R package version 0.3.2. [p409]

T. Appelhans, F. Detsch, C. Reudenbach, and S. Woellauer. *mapview: Interactive Viewing of Spatial Data in R*, 2017. URL https://CRAN.R-project.org/package=mapview. R package version 2.2.0. [p409]

M. Basille and D. Bucklin. *rpostgis: R Interface to a 'PostGIS' Database*, 2017. URL https://CRAN.R-project.org/package=rpostgis. R package version 1.3.0. [p]

C. J. G. Bellosta. *rPython: Package Allowing R to Call Python*, 2015. URL https://CRAN.R-project.org/package=rPython. R package version 0.0-6. [p410]

R. Bivand. Using the R–GRASS interface. *OSGeo Journal*, 1:36–38, 2007. URL http://fdo.osgeo.org/files/journal/final_pdfs/OSGeo_vol1_GRASS-R.pdf. [p410]

R. Bivand. *rgrass7: Interface Between GRASS 7 Geographical Information System and R*, 2017. URL https://CRAN.R-project.org/package=rgrass7. R package version 0.1-10. [p409, 410]

R. Bivand and N. Lewin-Koh. *maptools: Tools for Reading and Handling Spatial Objects*, 2017. URL https://CRAN.R-project.org/package=maptools. R package version 0.9-2. [p409]

R. Bivand and M. Neteler. Open source geocomputation: Using the R data analysis language integrated with GRASS GIS and PostgreSQL data base systems. In *GeoComputation*, 2000. URL http://www.geocomputation.org/2000/GC009/Gc009.htm. [p422]

R. Bivand and C. Rundel. *rgeos: Interface to Geometry Engine - Open Source ('GEOS')*, 2017. URL https://CRAN.R-project.org/package=rgeos. R package version 0.3-26. [p410]

R. Bivand, E. Pebesma, and V. Gómez-Rubio. *Applied Spatial Data Analysis with R*, volume 10 of *Use R!* Springer, New York, 2nd edition, 2013. ISBN 9781461476177. URL https://doi.org/10.1007/978-1-4614-7618-4. [p409, 422]

R. Bivand, T. Keitt, and B. Rowlingson. *rgdal: Bindings for the 'Geospatial' Data Abstraction Library*, 2017. URL https://CRAN.R-project.org/package=rgdal. R package version 1.2-16. [p410]

A. Brenning. Spatial prediction models for landslide hazards: Review, comparison and evaluation. *Natural Hazards and Earth System Sciences*, 5(6):853–862, 2005. URL https://doi.org/10.5194/nhess-5-853-2005. [p421]

A. Brenning. Statistical geocomputing combining R and SAGA: The example of landslide susceptibility analysis with generalized additive models. *Hamburger Beiträge zur Physischen Geographie und Landschaftsökologie*, 19:23–32, 2008. [p410, 415, 419, 422]

A. Brenning. Spatial cross-validation and bootstrap for the assessment of prediction rules in remote sensing: The R package sperrorest. In *IEEE International Symposium on Geoscience and Remote Sensing IGARSS*, 2012a. URL https://doi.org/10.1109/IGARSS.2012.6352393. [p421]

A. Brenning. *RPyGeo: ArcGIS Geoprocessing in R via Python*, 2012b. URL https://CRAN.R-project.org/package=RPyGeo. R package version 0.9-3. [p410, 423]

A. Brenning, S. Koszinski, and M. Sommer. Geostatistical homogenization of soil conductivity across field boundaries. *Geoderma*, 143(3-4):254–260, 2008. URL https://doi.org/10.1016/j.geoderma.2007.11.007. [p410]

A. Brenning, S. Long, and P. Fieguth. Detecting rock glacier flow structures using gabor filters and IKONOS imagery. *Remote Sensing of Environment*, 125:227–237, 2012. URL https://doi.org/10.1016/j.rse.2012.07.005. [p421]

A. Brenning, M. Schwinn, A. P. Ruiz-Páez, and J. Muenchow. Landslide susceptibility near highways is increased by 1 order of magnitude in the Andes of Southern Ecuador, Loja Province. *Natural Hazards and Earth System Sciences*, 15(1):45–57, 2015. URL https://doi.org/10.5194/nhess-15-45-2015. [p410]

P. E. Brown. Maps, coordinate reference systems and visualising geographic data with mapmisc. *The R Journal*, 8(1):64–91, 2016. URL https://journal.r-project.org/archive/2016/RJ-2016-005/RJ-2016-005.pdf. [p409]

J. M. Chambers. *Extending R*. The R Series. Chapman & Hall CRC, Boca Raton, Florida, 2016. ISBN 9781498775717. [p410]

J. Cheng, B. Karambelkar, and Y. Xie. *leaflet: Create Interactive Web Maps with the JavaScript 'Leaflet' Library*, 2017. URL https://CRAN.R-project.org/package=leaflet. R package version 1.1.0. [p423]

L.-C. Chien, Y. Guo, and K. Zhang. Spatiotemporal analysis of heat and heat wave effects on elderly mortality in Texas, 2006-2011. *Science of the Total Environment*, 562:845–851, 2016. URL https://doi.org/10.1016/j.scitotenv.2016.04.042. [p418]

A. Comber, H. Balzter, B. Cole, P. Fisher, S. C. M. Johnson, and B. Ogutu. Methods to quantify regional differences in land cover change. *Remote Sensing*, 8(3), 2016. URL https://doi.org/10.3390/rs8030176. [p418]

O. Conrad, B. Bechtel, M. Bock, H. Dietrich, E. Fischer, L. Gerlitz, J. Wehberg, V. Wichmann, and J. Böhner. System for Automated Geoscientific Analyses (SAGA) v. 2.1.4. *Geoscientific Model Development*, 8(7):1991–2007, 2015. URL https://doi.org/10.5194/gmd-8-1991-2015. [p410, 421]

M. O. Dillon, M. Nakazawa, and S. G. Leiva. The lomas formations of coastal Peru: Composition and biogeographic history. In J. Haas and M. O. Dillon, editors, *El Niño in Peru: Biology and Culture over 10,000 Years*, volume 10 of *Botany*, pages 1–9. Field Museum of Natural History, Chigaco, 2003. [p415]

C. F. Dormann, J. M. McPherson, M. B. Araújo, R. Bivand, J. Bolliger, G. Carl, R. G. Davies, A. Hirzel, W. Jetz, W. D. Kissling, I. Kühn, R. Ohlemüller, P. R. Peres-Neto, B. Reineking, B. Schröder, F. M. Schurr, and R. Wilson. Methods to account for spatial autocorrelation in the analysis of species distributional data: A review. *Ecography*, 30(5):609–628, 2007. URL https://doi.org/10.1111/j.2007.0906-7590.05171.x. [p421]

ESRI. *ArcGIS, Version 10.5*, 2017. URL http://www.arcgis.com. [p410]

M. J. A. Eugster and T. Schlesinger. osmar: OpenStreetMap and R. *R Journal*, 5(1):53–63, 2013. URL https://journal.r-project.org/archive/2013-1/eugster-schlesinger.pdf. [p409]

M. A. L. Fernandez, M. Schomaker, P. R. Mason, J. F. Fesselet, Y. Baudot, A. Boulle, and P. Maes. Elevation and cholera: An epidemiological spatial analysis of the cholera epidemic in Harare, Zimbabwe, 2008-2009. *BMC Public Health*, 12, 2012. URL https://doi.org/10.1186/1471-2458-12-442. [p418]

J. Franklin, P. McCullough, and C. Gray. Terrain variables used for predictive mapping of vegetation communities in Southern California. In J. P. Wilson and J. C. Gallant, editors, *Terrain Analysis*, pages 331–353. John Wiley & Sons, New York and Chichester, 2000. ISBN 978-0-471-32188-0. [p415]

GDAL Development Team. *GDAL - Geospatial Data Abstraction Library, version 2.2.2*. Open Source Geospatial Foundation, 2017. URL http://www.gdal.org. [p410]

GEOS Development Team. *GEOS - Geometry Engine Open Source, version 3.6.2*. Open Source Geospatial Foundation, 2017. URL https://trac.osgeo.org/geos. [p410]

J. N. Goetz, R. H. Guthrie, and A. Brenning. Integrating physical and empirical landslide susceptibility models using generalized additive models. *Geomorphology*, 129(3-4):376–386, 2011. URL https://doi.org/10.1016/j.geomorph.2011.03.001. [p421]

A. Graser and V. Olaya. Processing: A Python framework for the seamless integration of geoprocessing tools in QGIS. *ISPRS International Journal of Geo-Information*, 4(4):2219–2245, 2015. URL https://doi.org/10.3390/ijgi4042219. [p410, 420]

C. H. Grohmann. Morphometric analysis in Geographic Information Systems: Applications of free software GRASS and R. *Computers & Geosciences*, 30(9-10):1055–1067, 2004. URL https://doi.org/10.1016/j.cageo.2004.08.002. [p421]

V. Gómez-Rubio and A. López-Quílez. RArcInfo: Using GIS data with R. *Computers & Geosciences*, 31 (8):1000–1006, 2005. URL https://doi.org/10.1016/j.cageo.2005.02.009. [p409]

T. Hastie. *gam: Generalized Additive Models*, 2017. URL https://CRAN.R-project.org/package=gam. R package version 1.14-4. [p421]

T. Hengl, G. B. M. Heuvelink, and E. E. van Loon. On the uncertainty of stream networks derived from elevation data: The error propagation approach. *Hydrology and Earth System Sciences*, 14(7): 1153–1165, 2010. URL https://doi.org/10.5194/hess-14-1153-2010. [p410, 421]

T. Hengl, P. Roudier, D. Beaudette, and E. Pebesma. plotKML: Scientific visualization of spatio-temporal data. *Journal of Statistical Software*, 63(5), 2015. URL https://doi.org/10.18637/jss.v063.i05. [p409]

R. J. Hijmans. *raster: Geographic Data Analysis and Modeling*, 2017. URL https://CRAN.R-project.org/package=raster. R package version 2.6-7. [p409]

J. Inglada and E. Christophe. The Orfeo Toolbox remote sensing image processing software. In *Geoscience and Remote Sensing Symposium, 2009 IEEE International, IGARSS 2009*, volume 4, pages IV–733, 2009. [p410]

E. Iturritxa, N. Mesanza, and A. Brenning. Spatial analysis of the risk of major forest diseases in Monterey pine plantations. *Plant Pathology*, 64(4):880–889, 2015. URL https://doi.org/10.1111/ppa.12328. [p421]

R. Jones-Webb and M. Wall. Neighborhood racial/ethnic concentration, social disadvantage, and homicide risk: An ecological analysis of 10 us cities. *Journal of Urban Health*, 85(5):662–676, 2008. URL https://doi.org/10.1007/s11524-008-9302-y. [p418]

J. Kühn, A. Brenning, M. Wehrhan, S. Koszinski, and M. Sommer. Interpretation of electrical conductivity patterns by soil properties and geological maps for precision agriculture. *Precision Agriculture*, 10(6):490–507, 2009. URL https://doi.org/10.1007/s11119-008-9103-z. [p415]

P. Longley, M. Goodchild, D. J. Maguire, and D. W. Rhind. *Geographic Information Systems and Science*. John Wiley & Sons, Hoboken, N.J, 3rd ed. edition, 2011. ISBN 0470721448. [p409, 410, 423]

A. B. McBratney, I. O. Odeh, T. F. Bishop, M. S. Dunbar, and T. M. Shatar. An overview of pedometric techniques for use in soil survey. *Geoderma*, 97(3-4):293–327, 2000. URL https://doi.org/10.1016/S0016-7061(00)00043-4. [p415]

M. Mergili and H. Kerschner. Gridded precipitation mapping in mountainous terrain combining GRASS and R. *Norsk Geografisk Tidsskrift-Norwegian Journal of Geography*, 69(1):2–17, 2015. URL https://doi.org/10.1080/00291951.2014.992807. [p410]

M. Mergili, J. Krenn, and H.-J. Chu. r.randomwalk v1, a multi-functional conceptual tool for mass movement routing. *Geoscientific Model Development*, 8(12):4027–4043, 2015. URL https://doi.org/10.5194/gmd-8-4027-2015. [p410]

D. R. Montgomery and W. E. Dietrich. A physically based model for the topographic control on shallow landsliding. *Water Resources Research*, 30(4):1153–1171, 1994. URL https://doi.org/10.1029/93WR02979. [p415]

D. Moreno-Fernández, I. Cañellas, I. Barbeito, M. Sánchez-González, and A. Ledo. Alternative approaches to assessing the natural regeneration of Scots pine in a mediterranean forest. *Annals of Forest Science*, 72(5):569–583, 2015. URL https://doi.org/10.1007/s13595-015-0479-4. [p418]

J. Muenchow and P. Schratz. *RQGIS: Integrating R with QGIS*, 2017. URL https://CRAN.R-project.org/package=RQGIS. R package version 1.0.3. [p410]

J. Muenchow, A. Brenning, and M. Richter. Geomorphic process rates of landslides along a humidity gradient in the tropical Andes. *Geomorphology*, 139:271–284, 2012. URL https://doi.org/10.1016/j.geomorph.2011.10.029. [p410, 415]

J. Muenchow, A. Bräuning, E. Frank Rodríguez, and H. von Wehrden. Predictive mapping of species richness and plant species' distributions of a Peruvian fog oasis along an altitudinal gradient. *Biotropica*, 45(5):557–566, 2013a. URL https://doi.org/10.1111/btp.12049. [p415, 421]

J. Muenchow, H. Feilhauer, A. Bräuning, E. F. Rodríguez, F. Bayer, R. A. Rodríguez, and H. von Wehrden. Coupling ordination techniques and GAM to spatially predict vegetation assemblages along a climatic gradient in an ENSO-affected region of extremely high climate variability. *Journal of Vegetation Science*, 24(6):1154–1166, 2013b. URL https://doi.org/10.1111/jvs.12038. [p415, 419]

J. Muenchow, S. Hauenstein, A. Bräuning, R. Bäumler, E. F. Rodríguez, and H. von Wehrden. Soil texture and altitude, respectively, largely determine the floristic gradient of the most diverse fog oasis in the Peruvian desert. *Journal of Tropical Ecology*, 29:427–438, 2013c. URL https://doi.org/10.1017/S0266467413000436. [p415]

J. Muenchow, P. Dieker, J. Kluge, M. Kessler, and H. von Wehrden. A review of ecological gradient research in the Tropics: identifying research gaps, future directions, and conservation priorities. *Biodiversity and Conservation*, 2017. URL https://doi.org/10.1007/s10531-017-1465-y. [p415]

M. Neteler and H. Mitasova. *Open Source GIS: A GRASS GIS Approach*. Springer-Verlag, New York, 2008. [p410]

M. Padgham and A. Peutschnig. *dodgr: Distances on Directed Graphs*, 2017. URL https://CRAN.R-project.org/package=dodgr. R package version 0.0.3. [p409]

E. Pebesma. *sf: Simple Features for R*, 2017. URL https://CRAN.R-project.org/package=sf. R package version 0.5-5. [p409]

T. Peters, A. Bräuning, J. Muenchow, and M. Richter. An ecological paradox: High species diversity and low position of the upper forest line in the Andean depression. *Ecology and Evolution*, 4(11):2134–2145, 2014. URL https://doi.org/10.1002/ece3.1078. [p422]

R. J. Pike, I. S. Evans, and T. Hengl. Geomorphometry: A brief guide. In T. Hengl and H. I. Reuter, editors, *Geomorphometry*, Developments in soil science, pages 1–28. Elsevier, Amsterdam and Oxford, 2008. ISBN 0123743451. [p415]

J. Pinheiro, D. Bates, and R-core. *nlme: Linear and Nonlinear Mixed Effects Models*, 2017. URL https://CRAN.R-project.org/package=nlme. R package version 3.1-131. [p421]

L. Poggio and A. Gimona. Downscaling and correction of regional climate models outputs with a hybrid geostatistical approach. *Spatial Statistics*, 14:4–21, 2015. URL https://doi.org/10.1016/j.spasta.2015.04.006. [p410]

L. Y. Pomara, K. Ruokolainen, H. Tuomisto, and K. R. Young. Avian composition co–varies with floristic composition and soil nutrient concentration in Amazonian upland forests. *Biotropica*, 44(4):545–553, 2012. URL https://doi.org/10.1111/j.1744-7429.2011.00851.x. [p415]

QGIS Development Team. *QGIS Geographic Information System, Version 2.18.14-Las Palmas*. Open Source Geospatial Foundation, 2017. URL http://www.qgis.org/. [p410]

Rapidlasso. *LAStools, Version 171124*, 2017. URL https://rapidlasso.com/lastools/. [p410]

B. Ripley, D. Bates, and S. DebRoy. R data import/export, 2016. URL https://cran.r-project.org/doc/manuals/r-release/R-data.html. [p409]

D. Rocchini, L. Delucchi, G. Bacaro, P. Cavallini, H. Feilhauer, G. M. Foody, K. S. He, H. Nagendra, C. Porta, C. Ricotta, S. Schmidtlein, L. D. Spano, M. Wegmann, and M. Neteler. Calculating landscape diversity with information-theory based indices: A GRASS GIS solution. *Ecological Informatics*, 17:82–93, 2013. URL https://doi.org/10.1016/j.ecoinf.2012.04.002. [p421]

G. Sherman. *The PyQGIS programmer's guide: Extending QGIS 2.x with Python*. Locate Press, Chugiak Alas., 2014. ISBN 978-0-9894217-2-0. [p410]

D. Tarboton and I. Mohammed. *Terrain analysis using digital elevation models. TauDEM, Version 5*, 2017. URL http://hydrology.usu.edu/taudem/taudem5/index.html. [p410]

USGS. *U.S. Geological Survey (USGS) Earth Resources Observation and Science (EROS) Center*, 2017. URL http://earthexplorer.usgs.gov/. Last accessed 28 June 2017. [p]

K. A. Vanselow and C. Samimi. Predictive mapping of dwarf shrub vegetation in an arid high mountain ecosystem using remote sensing and random forests. *Remote Sensing*, 6(7):6709–6726, 2014. URL https://doi.org/10.3390/rs6076709. [p410]

S. Wood. *mgcv: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation*, 2017. URL https://doi.org/10.1111/j.1467-9868.2010.00749.x. R package version 1.8-20. [p421]

H. Zandler, A. Brenning, and C. Samimi. Quantifying dwarf shrub biomass in an arid environment: Comparing empirical methods in a high dimensional setting. *Remote Sensing of Environment*, 158: 140–155, 2015. URL https://doi.org/10.1016/j.rse.2014.11.007. [p410]

A. F. Zuur and E. N. Ieno. *A Beginner's Guide to Regression Models with Spatial and Temporal Correlation*. Highland Statistics Ltd., Newburgh, UK, 2017. [p422]

A. F. Zuur, E. N. Ieno, N. Walker, A. A. Saveliev, and G. M. Smith. *Mixed Effects Models and Extensions in Ecology with R*. Statistics for Biology and Health. Springer-Verlag, New York, 2009. ISBN 978-0-387-87457-9. [p421]

Jannes Muenchow
http://orcid.org/0000-0001-7834-4717
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
jannes.muenchow@uni-jena.de

Patrick Schratz
https://orcid.org/0000-0003-0748-6624
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
patrick.schratz@uni-jena.de

Alexander Brenning
https://orcid.org/0000-0001-6640-679X
Friedrich Schiller University Jena
Department of Geography
Löbdergraben 32
07743 Jena, Germany
alexander.brenning@uni-jena.de