Using the R-ArcGIS Bridge: the arcgisbinding Package

Marjean Pobuda

The **arcgisbinding** package is designed to allow you to expand your ArcGIS workflows to include R and expand your R analysis to include rich geospatial analysis. Through the power of the R-ArcGIS Bridge, you can easily transfer data from ArcGIS to R to gain access to the wealth of statistical packages and functions that you might need to further your analysis. Once finished, the bridge allows you to transfer your data back to ArcGIS to take advantage of the mapping, publishing, and sharing aspects of ArcGIS.

The functions contained within the **arcgisbinding** package fall into one of three categories:

- Basic Read/Write Functionality
- Converting Functionality
- Wrapping R Tool Functionality

This vignette begins by guiding you through each category and showing you examples of how each function is primarily used. It finishes by using the R-ArcGIS Bridge to solve a spatial problem in order to demonstrate one of the many workflows possible with the use of the bridge.

Basic Read/Write Functionality

The first main use of the bridge is to transfer data from ArcGIS to R and vice versa. The functions contained in this category are elemental to initializing the bridge. When you are first getting started with the bridge, the functions that you will use most often include:

- `arc.check_product()`
- `arc.open()`
- `arc.select()`
- `arc.shape()`
- `arc.shapeinfo()`
- `arc.write()`

Once you have initialized the bridge connection in ArcGIS and loaded the **arcgisbinding** library into your R workspace, you will finalize the connection between ArcGIS and R by using the `arc.check_product` function. This function does not take any arguments and will return details regarding your current ArcGIS installation and license.

```
arc.check_product()
#> product: ArcGIS Pro ( 11.4.0.7076 )
#> license: Advanced
```

After you have successfully connected ArcGIS and R, you can begin loading your data into your desired workspace. For GIS data that is currently stored in a shapefile, geodatabase, layer, or table, you will use the `arc.open` function to read it into R.

The **arcgisbinding** package comes with some sample data which we will use for the purpose of demonstrating the functions contained in the package. The only argument of the `arc.open` function is the path location at which the data you are trying to load is stored. To easily access the path for the sample data contained within the **arcgisbinding** package, the `system.file` function can be used.

```
ozone.path <- system.file("extdata", "ca_ozone_pts.shp",
                          package="arcgisbinding")
ozone.arc.dataset <- arc.open(ozone.path)
```

The sample data is now stored in the `ozone.arc.dataset` variable which is an `arc.dataset` object of class S4. This enables you to gain easy access to your data's metadata, such as its extent, a list of field names, and its geometry information.

```
ozone.arc.dataset@extent
#>       xmin       ymin       xmax       ymax
#> -2301587.7  -355613.7 -1795559.0   780085.2
ozone.arc.dataset@fields
#> $FID
#> [1] "OID"
#>
#> $Shape
#> [1] "Geometry"
#>
#> $LATITUDE
#> [1] "Double"
#>
#> $LONGITUDE
#> [1] "Double"
#>
#> $ELEVATION
#> [1] "Double"
#>
#> $OZONE
#> [1] "Double"
#>
#> $X
#> [1] "Double"
#>
#> $Y
#> [1] "Double"
#>
#> $text
#> [1] "String"
ozone.arc.dataset@shapeinfo
#> type         : Point
#> WKT          :
PROJCS["USA_Contiguous_Albers_Equal_Area_Conic",GEOGCS["GCS_North_American_1983",DATUM["D_North_America

#> WKID         : 102003
```

The `arc.select` function can be used to obtain your data set in an R data frame object. This function offers you the ability to choose specific attributes from your data set to work with and the ability to construct SQL queries

to only bring in certain observations based on the criteria of your analysis. For example, we can select six of the nine attributes from the `ozone.arc.dataset` object and elect to examine only those observations with an elevation of 1000 meters or greater.

```
ozone.dataframe <- arc.select(object = ozone.arc.dataset, fields = c("FID", "Shape", "LATITUDE",
"LONGITUDE", "ELEVATION", "OZONE"), where_clause = "ELEVATION >= 1000")
head(ozone.dataframe)
#>   FID LATITUDE LONGITUDE ELEVATION   OZONE
#> 1  11  39.3302 -120.1808      1676 0.06539
#> 2  22  38.9444 -119.9700      1905 0.07360
#> 3  32  39.9381 -120.9413      1052 0.07999
#> 4  42  37.8933 -119.8415      1432 0.08169
#> 5  56  40.5397 -121.5816      1788 0.08550
#> 6  57  37.5393 -119.6523      1280 0.08619
```

Additionally, functions like `arc.shapeinfo` and `arc.shape` are designed to get the details of what type of geometry is stored within your `arc.dataset` or `arc.dataframe`, respectively, so you can easily access that information and store it for later use. In particular, the WKT slot contains the well-known text for the spatial reference system of the object.

```
ozone.shapeinfo <- arc.shapeinfo(ozone.arc.dataset)
ozone.shapeinfo$WKT
#> [1]
"PROJCS[\"USA_Contiguous_Albers_Equal_Area_Conic\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_A

ozone.shape <- arc.shapeinfo(arc.shape(ozone.dataframe))
ozone.shape$WKT
#> [1]
"PROJCS[\"USA_Contiguous_Albers_Equal_Area_Conic\",GEOGCS[\"GCS_North_American_1983\",DATUM[\"D_North_A
```

The `arc.write` function allows you to easily save your data to shapefile, geodatabase, or table. This function is often called once you have completed your work in R and need to transfer your data to ArcGIS for mapping and/or further analysis. The first argument is the path for the location of the ArcGIS project you wish to write your new results to. The second argument specifies the R object you are writing to this new location. Additional optional parameters allow you to provide the geometry type and spatial reference for your data if desired.

```
arc.write(path = tempfile("ca_new", fileext=".shp"), data = ozone.dataframe)
```

The `arc.write` function has also been designed with data generation in mind. For example, the function can be used to create new features in ArcGIS as demonstrated below where the `arc.write` function creates 100 uniformly distributed points in a unit square from a normal distribution.

```
arc.write(path = tempfile("new_pts", fileext=".shp"), data = list('data'=rnorm(100)),
          coords = list(x=runif(100,min=0,max=1),y=runif(100,min=0,max=1)),
          shape_info = list(type='Point'))
```

You should now have a solid understanding of how to initalize the bridge from R and to begin loading, accessing and exporting various parts of your data. The following section outlines more advanced functionality of the bridge.

Converting Functionality

After mastering the basics of moving data back and forth across the bridge, a natural next step is to begin analyzing the data. R has a tremendous number of packages and methods that focus on spatial data and the attributes associated with them. As such, it is important that you know how to manipulate your data into a format that is consistent with the method itself. There are several functions in the **arcgisbinding** package that focus on coverting your data into a format necessary for many data analyses. These functions include:

- `arc.data2sp`
- `arc.sp2data`
- `arc.shape2sp`
- `arc.fromP4ToWKt`
- `arc.fromWktToP4`
- `arc.select`

The spatial data frame object from the **sp** package is a common format required to run many spatial analyses in R. As such, the bridge has a pair of functions designed to convert data frames to a spatial data frame object and vice versa. Point features convert to a spatial points data frame (`SpatialPointsDataFrame`), while polygons and line features are mapped to spatial polygons and lines data frames (`SpatialPolygonsDataFrame`, `SpatialLinesDataFrame`). When a method requires that your data is in an **sp** format, simply call the `arc.data2sp` function.

```
class(ozone.dataframe)
#> [1] "data.frame" "arc.data"
ozone.sp.df <- arc.data2sp(ozone.dataframe)
class(ozone.sp.df)
#> [1] "SpatialPointsDataFrame"
#> attr(,"package")
#> [1] "sp"
```

Subsequently, the `arc.sp2data` function is used when you are finished performing spatial analysis and wish to convert your **sp** object back to a data frame:

```
ozone.dataframe <- arc.sp2data(ozone.sp.df)
class(ozone.dataframe)
#> [1] "data.frame" "arc.data"   "arc.data"
```

The **sp** data frame object allows you to index your data as if it were a regular R data framed object since they contain a slot for your data attributes. If this extra slot is not needed for your work, an alternative is to convert your data to a spatial points (`SpatialPoints`), spatial lines (`SpatialLines`), or spatial polygons (`SpatialPolygons`) object instead of a spatial data frame object. This can be accomplished with the `arc.shape2sp` function:

```
ozone.sp.shape <- arc.shape2sp(ozone.dataframe)
class(ozone.sp.shape)
#> [1] "SpatialPoints"
#> attr(,"package")
#> [1] "sp"
```

The bridge also provides a straightforward framework for reprojecting your data. Two functions, `arc.fromP4ToWkt` and `arc.fromWktToP4` aid in this process by easily enabling you to obtain either the **PROJ.4** coordinate reference system string or the well-known text (**WKT**) representation string for your data. (Note: Currently these functions only work with ArcGIS Pro). The bridge offers you multiple options for obtaining your desired geometry information in the format you need. For example, you can use the shapeinfo slot of your *arc.dataset* object to get the well-known text representation string and then convert to a **PROJ.4** coordinate reference system string.

```
arc.fromWktToP4(ozone.arc.dataset@shapeinfo$WKT)
#> [1] "+proj=aea +lat_1=29.5 +lat_2=45.5 +lat_0=37.5 +lon_0=-96 +x_0=0 +y_0=0 +datum=NAD83 +units=m
+no_defs"
```

The bridge also can recognize well-known IDs and convert those in a similar manner:

```
arc.fromWktToP4(4326)
#> [1] "+proj=longlat +datum=WGS84 +no_defs"
```

The output from these functions can then be used with the `arc.select` function to reproject your data. This function contains an optional argument for specifying the spatial reference you want your data to be in. These features greatly simplfy the hassel of reprojecting your data and make it easy for you to obtain the geometry information for your data in the format you need.

```
ozone.dataframe <- arc.select(object = ozone.arc.dataset, fields = c("FID", "Shape", "LATITUDE",
"LONGITUDE", "ELEVATION", "OZONE"), where_clause = "ELEVATION >= 1000", sr = "+proj=latlong
+datum=wgs84 +no_defs")
arc.shapeinfo(arc.shape(ozone.dataframe))
#> type         : Point
#> WKT          :
GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["WGS_1984",6378137.0,298.257223563]],PRIMEM["Greenwic

#> WKID         : 4326
```

The previous sections showed you how to import/export your data and convert it into an **sp** object so you can perform statistical analysis in R. You also learned how to use the bridge to reproject your data. The following section describes several functions that aid developers who wish to call R functionality from within ArcGIS. This will allow you to harness the full power of the geographic information system and provides an avenue to make your R code accessible to a broader audience.

Wrapping R Tool Functionality

Another main use of the bridge is to wrap R tools and code into a geoprocessing script so they can be called as a tool in ArcGIS. This is useful functionality when you have functions in R that you use repeatedly and want to visualize the results on a map. It also makes it easier to share your workflows with other members of your organization that are not familiar with the R language. The **arcgisbinding** package contains several functions to help you design and customize the new tool you create:

- `arc.env`
- `arc.progress_label`
- `arc.progress_pos`

A simple sample script has been created to demonstrate these functions. The script below defines a tool that reads in a point feature layer using the bridge, prints out some information about the data, and then outputs the data. Normally, functionality unique to R would be utilized in the body of a script but this code has been simplified to showcase the desired **arcgisbinding** functions.

```
tool_exec <- function(in_params, out_params)
{
  env = arc.env()
  wkspath <- env$workspace

  cat(paste0("\n", "..................................", "\n"))
  cat(paste0("\n", "Printing Workspace Path...", "\n"))
  cat(paste0("\n", wkspath, "\n"))
  cat(paste0("\n", "..................................", "\n"))

  source_dataset = in_params[[1]]
  out_fc = out_params[[1]]

  #########
  arc.progress_label("Loading Dataset...")
  arc.progress_pos(20)
  d <- arc.open(source_dataset)
  data <- arc.select(d)

  #########
  arc.progress_label("Obtaining Attribute Names...")
  arc.progress_pos(40)
  attnames <- colnames(data)

  cat(paste0("\n", "..................................", "\n"))
  cat(paste0("\n", "Printing Attribute Names...", "\n"))
  cat(paste0("\n", attnames, "\n"))
  cat(paste0("\n", "..................................", "\n"))

  #########
  arc.progress_label("Obtaining Geometry Information...")
  arc.progress_pos(60)
  data_shp <- arc.shapeinfo(d)

  cat(paste0("\n", "..................................", "\n"))
  cat(paste0("\n", "Printing Geometry Information...", "\n"))
  cat(paste0("\n", data_shp, "\n"))
  cat(paste0("\n", "..................................", "\n"))

  #########
  arc.progress_label("Writing result dataset...")
  arc.progress_pos(80)

  if(!is.null(out_fc) && out_fc != "NA")
    arc.write(out_fc, data)

  #########
  arc.progress_label("Done")
  arc.progress_pos(100)
```

```
  return(out_params)

}
```

When working in ArcGIS, a user has the ability to customize their project environment. This includes things such as setting an output coordinate system, defining a processing extent, setting a random number seed, etc. If a user has preset some of their environment variables, this may impact the results of your R script tool. The `arc.env` function enables you to get the local ArcGIS geoprocessing tool environment settings and check to ensure they are set appropriately.

At the start of our sample script tool, the geoprocessing tool environment settings are stored in the `env` variable by using the `arc.env` function. This allows for you to check a user's settings as needed within your script. While you are unable to override a user's local settings, you can throw a warning or error message to alert the user that they need to adjust their environments before continuing with your tool if a certain setting impacts your tool's results.

The **arcgisbinding** package also allows you to customize the tool run experience users have with your script through use of the `arc.progress_label` and `arc.progress_pos` functions. (Note: Currently these functions only work with ArcGIS Pro). When running a geoprocessing tool in ArcGIS, a tool run status bar appears that can display custom messages to indicate the status of the run. In addition, the tool run progress bar can be customized to appear at different levels of completion based on the current stage of the run.

With the `arc.progress_label` function, you can customize what messages your users see based on where the script is at while they are running your tool. While the tool run status bar typically cotinuously moves back and forth by default, if you wish, you can customize it to appear at a certain percentage of completion by entering a value from 0 to 100 to represent the current percentage. Both of these functions are used multiple times in the sample script tool above to demonstrate how they might be used when creating a new geoprocessing script tool.

The **arcgisbinding** package is designed to make working with R and ArcGIS side-by-side as seamless and straight-forward as possible. Functions to initialize the connection between R and ArcGIS and to read and write data are meant to be intuitive and fast. The other funtions of the **arcgisbinding** package were created to aid in tasks thought to be ubiquitous to most analyses. Now that you have a solid foundation in all the functionality contained within the **arcgisbinding** package, we will use the package to perform a spatially explicit analysis in order to demonstrate one of the workflows possible with the use of the bridge.