Conectando R con SAGA

En este documento se pretende mostrar una de las múltiples formas en las que se puede conectar R con SAGA, con el fin de utilizar módulos de SAGA en scripts de R o en nuestras ejecuciones automáticas de R.

Introducción

Todo módulo de SAGA está formado por una instrucción o código, donde se definen los parámetros a usar. Este código se ejecuta internamente el en saga_gui, que es la interfaz visual, o mediante en saga_cmd que nos permite acceder a las librerías de SAGA desde una terminal o desde el símbolo del sistema.

Una de las maneras de hacer interactuar R con SAGA es a través del uso del intérprete de línea de comando saga_cmd empleando un paquete de R que se denomina "RSAGA". Se trata de uno de los primeros y más globalmente usado. Aunque este paquete tiene una serie de funciones donde incorpora módulos varios de SAGA, veremos una función de este paquete (`rsaga.geoprocessor()`), genérica, que nos permite hacer llamamiento de cualquier librería de SAGA.

Se mostrará un ejemplo sencillito en donde se leerá un fichero con una tabla de datos, y se convertirá esta información a una capa de puntos.

Para ello se verán distintas maneras de explorar ese código de las librerías de SAGA, y como introducir ese código en la sintaxis de la función de RSAGA.

Software necesario

- R y RStudio:
  - Paquete RSAGA
- SAGA (cualquier versión, instalable o portable)

Poner RStudio a punto

En primer lugar, es necesario instalar el paquete RSAGA, que se puede hacer cómodamente desde RStudio, desde el panel "Files, Plots, Packages, Help, View" (Imagen 1). Tras situarse en la pestaña "Packages" se clica en "Install" y se introduce el paquete a instalar con sus dependencias.

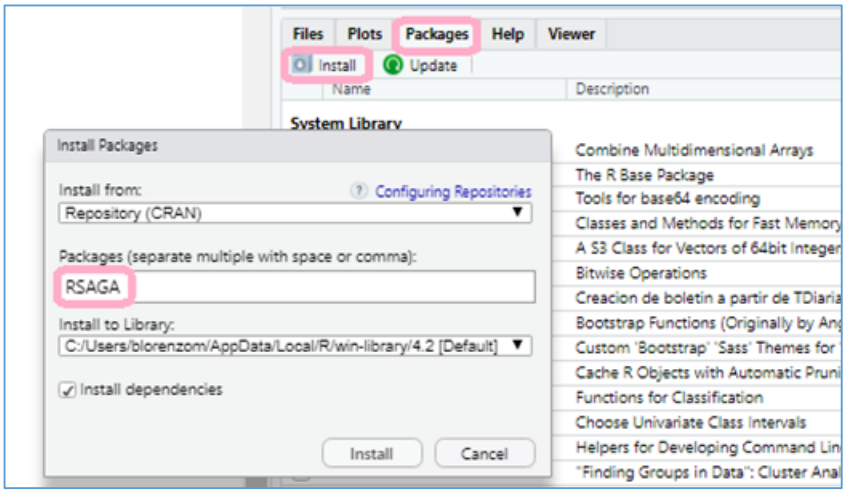

Imagen 1: Instalando RSAGA

Una vez instalado, se puede comprobar que ya aparece en los paquetes disponibles, en la misma pestaña de "Packages" (Imagen 2):

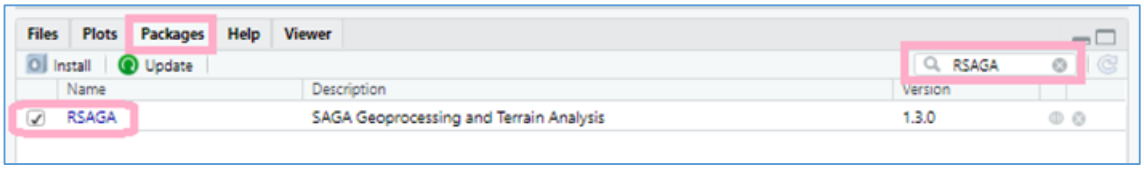

Imagen 2: Visualizando RSAGA en "Packages"

Para usar sus funciones se debe cargar en la sesión, mediante un `library("RSAGA")` o clicando en el cuadrado que aparece a la izquierda del paquete, apareciendo así un tick o marca.

Inicializando el RSAGA environment

El siguiente paso es decirle a R (RSAGA) donde están las librerías y la instalación de SAGA que se quieren utilizar.

Para ello se utiliza la función `rsaga.env()`, donde se puede especificar la ruta a las librerías de RSAGA. Si no se pone nada, se buscan en unos directorios por defecto.

En el siguiente ejemplo se asigna a una variable llamada "env" ese path donde se encuentra la versión de SAGA que queremos usar. En este caso una versión portable en Windows:

```
> env<-rsaga.env(path="C:/Users/blorenzom/Desktop/saga-8.3.0_x64")
```

A partir de ese momento, cuando en la sesión se requiera especificar el environment de geoprocesado, estará almacenado en "env".

Ejemplo: Convert Table to Points

Ahora se verá a través de un ejemplo, cómo podemos utilizar un módulo de SAGA desde R. En este caso el módulo "Convert Table to Points".

Datos: "TANUAL_1971-2000_CAT.dbf".

Se dispone de una tabla de datos de temperatura media anual del período 1971-2000 de la región de Cataluña, y se pretende transformar esta tabla en una capa de puntos para su posterior uso en otros cálculos.

Se parte del conocimiento del módulo de SAGA que se necesita o se quiere usar. Es decir, ya se debe saber con antemano que módulo en concreto usar. En este caso "Convert Table to Points".

Sólo falta conocer como es la síntaxis de este módulo internamente, para poder usarlo. Hay varias maneras de explorar esta información. Las más usuales son, desde el propio saga_gui, usando la ayuda del saga_cmd desde terminal o desde el propio RStudio mediante las funciones de RSAGA.

1. Examinar parámetros de módulo desde saga_gui

En la Imagen 3 se muestra un ejemplo de lo que puede interesar a la hora de hacer la síntaxis de un módulo:

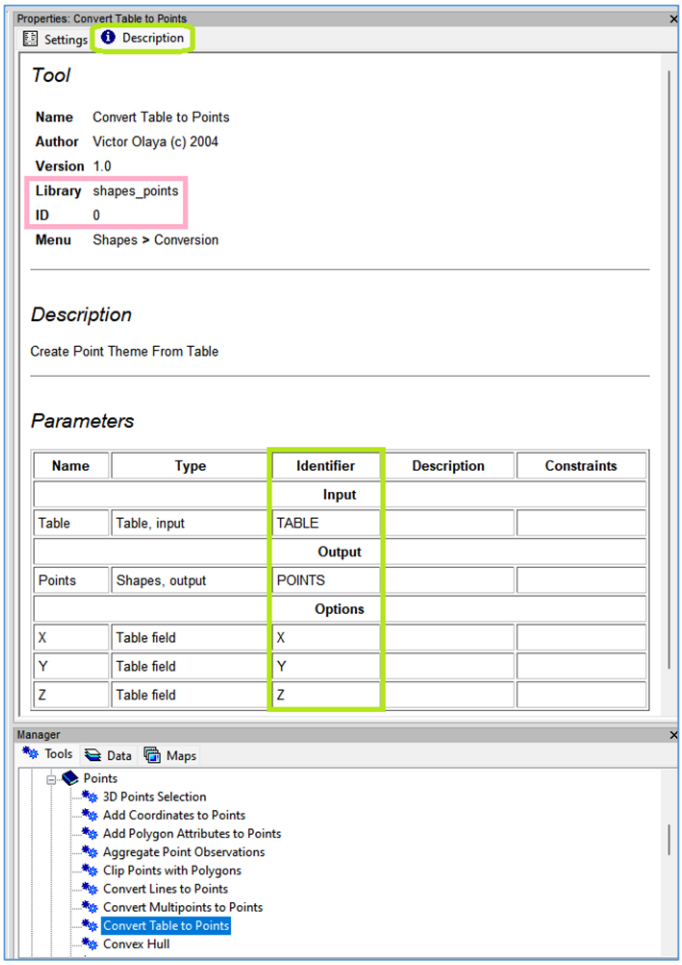

Imagen 3: Examinando un módulo desde saga_gui

Desde el panel "Manager", en la pestaña "Tools", seleccionando el módulo que interese (En este caso Shapes → Points → Convert Table to Points), se puede cotillear su información en el panes "Properties", pestaña "Descripción". De esa información, en general, lo que se necesita es:

- la librería a la cual pertenece. En este caso *shapes_points*.
- El ID del módulo. En este caso: 0.
- Y los parámetros, referidos a su identificador. Hay parámetros que es obligarorio poner y otros que no.
    - En este caso el parámetro con identificador TABLE hace referencia al fichero donde se encuentra la tabla de datos de entrada.
    - Se ve que en output, está el parámetro POINTS, que hace referencia a la capa shape de resultado tras transformar la tabla a points.
    - Asimismo, con los parámetros X e Y, se indican las coordenadas para la georreferenciación.

Con esta información ya se podría establecer la síntaxis de este módulo.

Desde este mismo sitio, hay más maneras de obtener esta misma información. Al pinchar con el botón derecho del ratón sobre el nombre del módulo en el panel Manager → Tools, se despliega una ventanita emergente donde se puede seleccionar:

- *Save to Script File*. Con esta opción se generaría un fichero ejecutable en el lenguaje del sistema operativo, que habría que **editar** para cubrir los parámetros que fuesen necesarios. En este ejemplo, en Windows, se generaría un fichero .bat con la siguiente información (Imagen 4):

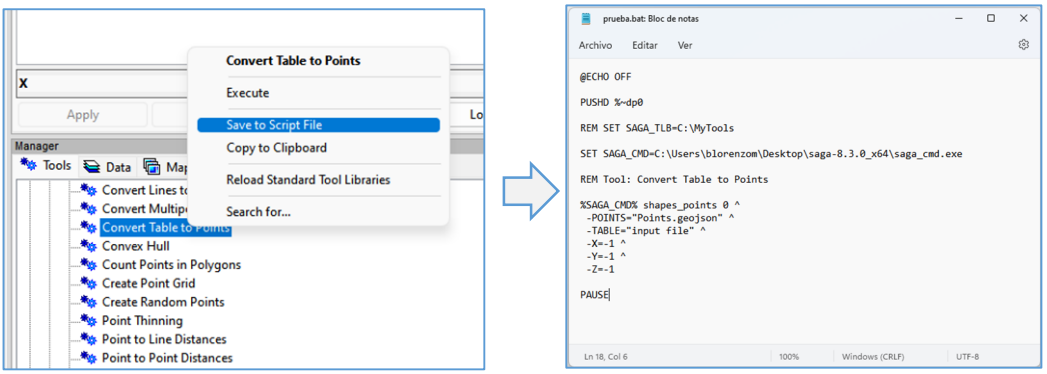

Imagen 4: Información de módulo mediante "Save to Script File"

Este fichero se podría completar editando, ya que es ejecutable, incluso añadiendo más módulos o rutinas.

- Si lo que interesa es copiar simplemente el código, lo ideal ya serían cualquiera de las demás opciones de "*Copy to Clipboard*" (pinchando en el módulo con el botón derecho) [Imagen 5].

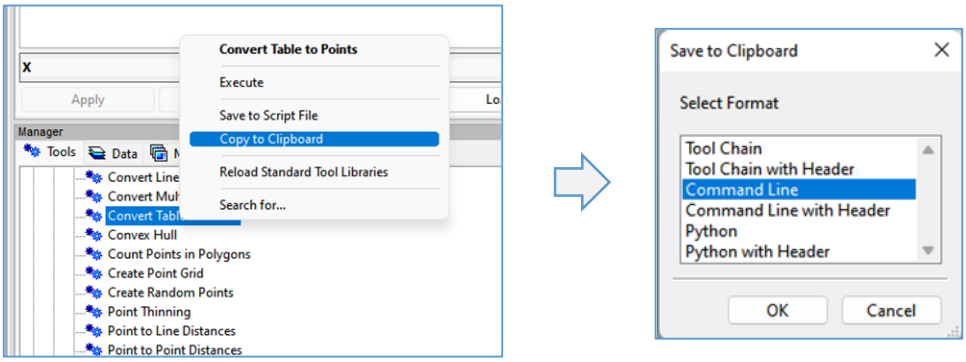

Imagen 5: Información de módulo mediante "Copy to Clipboard"

Se pueden seleccionar diferentes formatos para copiar. Todos muy similares que depende de donde lo vayamos a usar. Por ejemplo si se selecciona Command Line:

```
saga_cmd shapes_points 0 ^
 -POINTS="Points.geojson" ^
 -TABLE="input file" ^
 -X=-1 ^
 -Y=-1 ^
 -Z=-1
```

Lo interesante de esta manera de explorar es que obtenemos de forma ya escrita, el nombre de la librería, el identificador del módulo, y los parámetros. Aparece saga_cmd

porque se ha seleccionado “Command Line”, sería la estructura del módulo para usar con saga_cmd.

2. Examinar código del módulo con saga_cmd

Para examinar esta información que ya hemos visto, pero mediante el saga_cmd, se necesita acceder a una terminal del sistema.

Por ejemplo, en Windows, se puede utilizar el símbolo de sistema, donde hay que situarse en el directorio de saga si es versión portable.

Se puede utilizar saga_cmd que permite ejecutar los módulos de saga, y también extraer su información entre otras cosas. En las siguientes imágenes se explora la librería shapes_points, y también su módulo 0, que es el que interesa para el ejemplo. Se hace con `saga_cmd -h “NOMBRE DE LA LIBRERÍA”` y se puede añadir el ID del módulo que interese (Imagen 6).

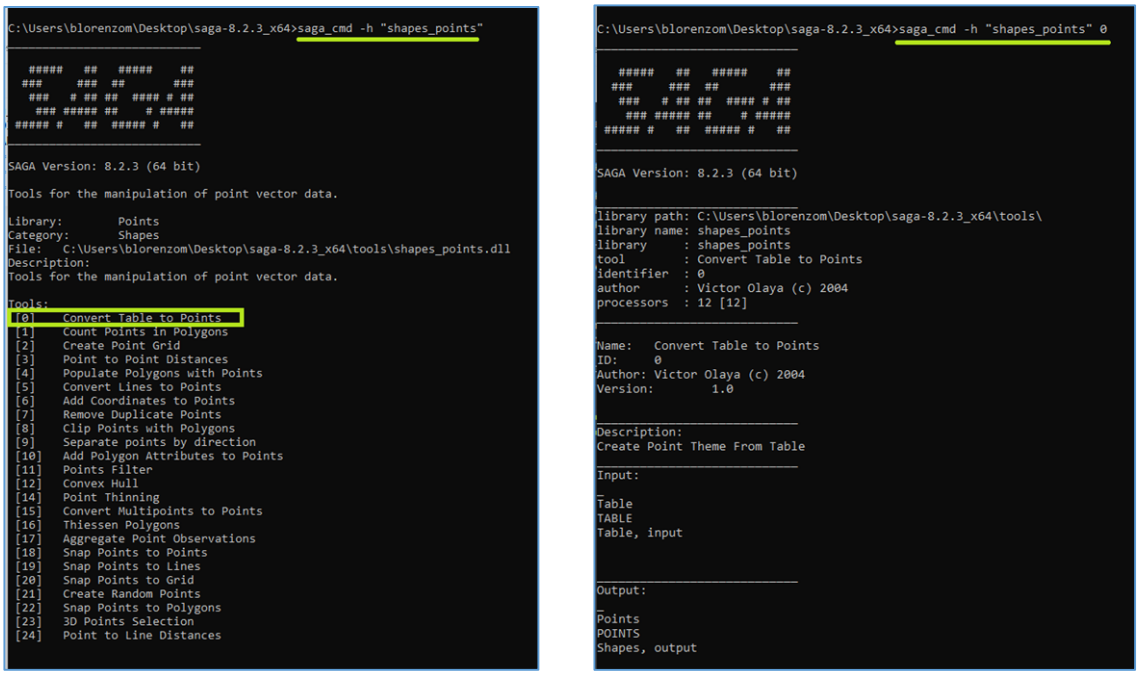

Imagen 6: Explorando código de módulo mediante saga_cmd

Esto mismo se puede hacer en una **terminal** en RStudio (panel de Console, Terminal, Jobs) [Imagen 7]:

Imagen 7: Exploración de módulo desde terminal en RStudio

3. Examinar código del módulo con RSAGA

RSAGA dispone de varias funciones que permiten explorar de manera análoga a lo anterior, los módulos y librerías de saga. Un ejemplo de exploración podría ser el siguiente:

Se carga el paquete RSAGA

```
library(RSAGA)
```

Se almacena en env donde está SAGA

```
env<-rsaga.env(path="C:/Program Files (x86)/SAGA-GIS") #Path a donde esté SAGA

## Verify specified path to SAGA command line program...
## Found SAGA command line program. Search for not specified SAGA modules path...
## Done
```

Se pueden explorar las librerías disponibles con *rsaga.get.libraries()*

```
rsaga.get.libraries(path = env$modules)

##  [1] "climate_tools"           "contrib_perego"
##  [3] "db_odbc"                 "db_pgsql"
##  [5] "docs_html"               "docs_pdf"
##  [7] "garden_3d_viewer"        "garden_fractals"
##  [9] "garden_games"            "garden_learn_to_program"
## [11] "garden_webservices"      "grid_analysis"
## [13] "grid_calculus"           "grid_calculus_bsl"
## [15] "grid_filter"             "grid_gridding"
## [17] "grid_spline"             "grid_tools"
## [19] "grid_visualisation"      "grids_tools"
## [21] "imagery_classification"  "imagery_isocluster"
## [23] "imagery_maxent"          "imagery_opencv"
## [25] "imagery_photogrammetry"  "imagery_segmentation"
## [27] "imagery_svm"             "imagery_tools"
## [29] "imagery_vigra"           "io_esri_e00"
## [31] "io_gdal"                 "io_gps"
## [33] "io_grid"                 "io_grid_image"
## [35] "io_shapes"               "io_shapes_dxf"
## [37] "io_shapes_las"           "io_table"
## [39] "io_virtual"              "pj_georeference"
## [41] "pj_geotrans"             "pj_proj4"
```

```
## [43] "pointcloud_tools"         "pointcloud_viewer"
## [45] "shapes_grid"              "shapes_lines"
## [47] "shapes_points"            "shapes_polygons"
## [49] "shapes_tools"             "shapes_transect"
## [51] "sim_cellular_automata"    "sim_ecosystems_hugget"
## [53] "sim_erosion"              "sim_fire_spreading"
## [55] "sim_geomorphology"        "sim_hydrology"
## [57] "sim_ihacres"              "sim_landscape_evolution"
## [59] "sim_qm_of_esp"            "sim_rivflow"
## [61] "statistics_grid"          "statistics_kriging"
## [63] "statistics_points"        "statistics_regression"
## [65] "ta_channels"              "ta_compound"
## [67] "ta_hydrology"             "ta_lighting"
## [69] "ta_morphometry"           "ta_preprocessor"
## [71] "ta_profiles"              "ta_slope_stability"
## [73] "table_calculus"           "table_tools"
## [75] "tin_tools"                "tin_viewer"
```

Se pueden explorar los módulos de una librería con *rsaga.get.modules()*

```
rsaga.get.modules(libs = "shapes_points", env = env)
```

```
## $shapes_points
##    code                              name interactive
## 1     0          Convert Table to Points       FALSE
## 2     1         Count Points in Polygons       FALSE
## 3     2                Create Point Grid       FALSE
## 4     3                  Point Distances       FALSE
## 5     4            Fit N Points to shape       FALSE
## 6     5          Convert Lines to Points       FALSE
## 7     6        Add Coordinates to Points       FALSE
## 8     7          Remove Duplicate Points       FALSE
## 9     8        Clip Points with Polygons       FALSE
## 10    9     Separate points by direction       FALSE
## 11   10 Add Polygon Attributes to Points       FALSE
## 12   11                    Points Filter       FALSE
## 13   12                      Convex Hull       FALSE
## 14   14                  Points Thinning       FALSE
## 15   15    Convert Multipoints to Points       FALSE
## 16   16                Thiessen Polygons       FALSE
## 17   17     Aggregate Point Observations       FALSE
## 18   18            Snap Points to Points       FALSE
## 19   19             Snap Points to Lines       FALSE
## 20   20              Snap Points to Grid       FALSE
```

Se pueden explorar los parámetros de un módulo, con *rsaga.get.usage()*

```
rsaga.get.usage(lib = "shapes_points", module = 0, env = env)
```

```
## library path: C:\PROGRA~2\SAGA-GIS\tools\
## library name: shapes_points
## library     : shapes_points
## Usage: saga_cmd shapes_points 0 [-POINTS <str>] [-TABLE <str>] [-X <str>] [-Y <str
>] [-Z <str>]
##   -POINTS:<str>  Points
##  Shapes (output)
##   -TABLE:<str>   Table
##  Table (input)
##   -X:<str>       X
##  Table field
##   -Y:<str>       Y
##  Table field
##   -Z:<str>       Z
```

```
rsaga.get.usage(lib = "shapes_points", module = "Convert Table to Points", env = env)
```

```
## library path: C:\PROGRA~2\SAGA-GIS\tools\
## library name: shapes_points
```

```
## library     : shapes_points
## Usage: saga_cmd shapes_points 0 [-POINTS <str>] [-TABLE <str>] [-X <str>] [-Y <str
>] [-Z <str>]
##   -POINTS:<str>  Points
##  Shapes (output)
##   -TABLE:<str>   Table
##  Table (input)
##   -X:<str>       X
##  Table field
##   -Y:<str>       Y
##  Table field
##   -Z:<str>       Z
```

Todos estos métodos proporcionan la misma información, se debe escoger la metodología más adecuada para cada cual.

Usando la función rsaga.geoprocessor()

Para ejecutar cualquiera de los módulos de SAGA que tengamos disponibles, se puede usar la función de espectro general: *rsaga.geoprocessor()*.

Desde R se puede consultar la ayuda de esta función mediante help("rsaga.geoprocessor") o desde la pestaña correspondiente en RStudio. Básicamente a esta función hay que indicarle:

- el nombre de la librería.
- El ID del módulo que se quiera usar.
- Una lista con los parámetros para la configuración de ese módulo.
- El *env* para que localice las librerías.

La estructura de esta función básicamente es:

```
rsaga.geoprocessor(lib = [aquí iría el "nombre" de la librería], module = [aquí e
l ID o nombre del módulo], param = list([aquí iría la lista de parámetros con sus nom
bres según los identificadores explorados]), env = env [la variable del rsaga environ
ment])
```

En el ejemplo que estamos tratando, la información a rellenar sería:

- lib = "shapes_points"
- module = 0
- param = list(**TABLE** = "TANUAL_1971-2000_CAT.dbf" , **POINTS** = "TANUAL_1971-2000_CAT.shp", **X** = "COOR_X", **Y** = "COOR_Y")
  Donde:
  - TABLE: ruta al fichero donde está la tabla de puntos
  - POINTS: ruta al fichero shape de salida (capa de puntos)
  - X: nombre de la columna de la tabla donde se alojan las coordenadas X
  - Y: nombre de la columna de la tabla donde se alojan las coordenadas Y
- env = env

Con toda esta información, la función quedaría de la siguiente manera:

```
rsaga.geoprocessor(lib = "shapes_points", module = 0, param = list(POINTS = "TANU
AL_1971-2000_CAT.dbf" , TABLE = "TANUAL_1971-2000_CAT.shp",  X = "COOR_X", Y = "COOR_
Y"), env = env)
```

¡Con esta instrucción, se puede ejecutar el módulo “Convert Table to Points” de SAGA, desde R, con los parámetros personalizados!

Resumiendo

El proceso para utilizar en este ejemplo “Convert Table to Points” podría ser el siguiente:

Se carga el paquete RSAGA

```
library(RSAGA)
```

Se almacena en env donde está SAGA

```
env<-rsaga.env(path="C:/Program Files (x86)/SAGA-GIS")

## Verify specified path to SAGA command line program...
## Found SAGA command line program. Search for not specified SAGA modules path...
## Done
```

Se utiliza el módulo de SAGA “Convert Table to Points”

Se convierte la tabla del fichero de datos “TANUAL_1971-2000_CAT.dbf” en una capa de puntos “TANUAL_1971-2000_CAT.shp”:

```
rsaga.geoprocessor(lib = "shapes_points",
                   module = 0,
                   param = list(TABLE = "TANUAL_1971-2000_CAT.dbf",
                                POINTS = "TANUAL_1971-2000_CAT.shp",
                                X = "COOR_X", Y = "COOR_Y"), env = env)

##
##
## SAGA Version: 6.3.0 (64 bit)
##
## library path: C:\PROGRA~2\SAGA-GIS\tools\
## library name: shapes_points
## library     : shapes_points
## tool        : Convert Table to Points
## identifier  : 0
## author      : Victor Olaya (c) 2004
## processors  : 12 [12]
##
## Load table: TANUAL_1971-2000_CAT.dbf...
##
##
## Parameters
##
##
## Points: Points
## Table: TANUAL_1971-2000_CAT
## X: COOR_X
## Y: COOR_Y
## Z: <not set>
##
## Save shapes: TANUAL_1971-2000_CAT.shp...
```

El resultado quedaría almacenado en los ficheros shape:

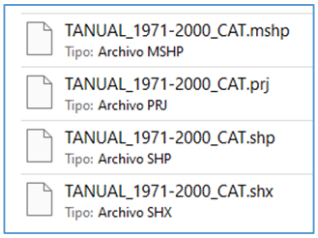

Imagen 8: Ficheros que componen la capa de puntos creada

Conclusión

Se ha mostrado una función (rsaga.geoprocessor) del paquete de R RSAGA que permite acceder a cualquier módulo de la instalación de SAGA desde R.

A partir de ahora hay que lanzarse a experimentar, por ejemplo se puede realizar la práctica final mediante instrucciones en R, y ejecutarlo todo desde un script.

¡Ahora las posibilidades son enormes! Se pueden hacer funciones que ejecuten los módulos de R al propio interés. Se pueden agrupar varias ejecuciones en scripts, y programarlas para que se ejecuten a la hora, día o mes deseado.