How to export spatial objects from IsoriX to GIS?

The IsoriX core team

2017-07-07

```
## Warning: package 'sp' was built under R version 3.4.1
```

```
## Warning: package 'rgdal' was built under R version 3.4.1
```

Welcome to **IsoriX**, in this vignette we present the steps required for you to export the spatial object created in IsoriX to your favourite GIS software (e.g. ArcGIS)... but it is good to know that with the help of a few R packages designed for easy handling and plotting spatial data, you can probably do all your GIS work without leaving the R environment!

Before starting

Please read the vignette ***Workflow***, if you haven't done so. You can access it by simply typing:

```
vignette("Workflow", package="IsoriX")
```

We assume that you are still in the same R session than the one created during the simple workflow example. That implies that the objects we will use below have already been created.

You can test if you have all the object we need by running the following code:

```
all(c("elev", "isoscape", "assignment") %in% ls())
```

If the result is TRUE, you can proceed! If not, then go back to the vignette ***Workflow***.

We will also need to have a few packages installed and loaded:

```
library(IsoriX)
library(raster)
library(GISTools)
library(rgdal)
```

Getting to know the spatial files used in IsoriX

Let's first explain which spatial data formats we use in IsoriX and where to find the spatial data in the objects created by IsoriX.

RasterLayers, RasterStacks (RasterBricks), and SpatialPolygons, SpatialPoints-DataFrames

There are four types of spatial objects in IsoriX that one should know about. `RasterLayers` are 2-dimensional matrix-like grids, with each grid cell containing a value representing information, e.g. the mean predicted isotopic value or elevation. `RasterStacks` or `RasterBricks` are very similar multilayer-raster objects, i.e. multi-band satellite images. As their names suggest it, these previous object are handled by the package **raster**. Then, we also have `SpatialPolygons` and `SpatialPoints`. Those are objects of containing polygons or points; they are be handled by the package **sp**.

Accessing the spatial information stored in the objects produced by IsoriX

The object 'CountryBorders' and the 'OceanMask' are ready-to-use `SpatialPolygons` stored in IsoriX. Further spatial information is stored a little bit more deeply in the objects of class *isoscape* and *isorix*.

The guts of the objects of class *isoscape*

The objects of class *isoscape* (such as the object actually called `isoscape` which we created in the workflow) are lists that contain information stored in the element `$isoscape` (a RasterStack) and `$sp.points` (a list containing itself `$sources` a SpatialPointsDataFrame). All of this can be deduced from the analysis of the object in R:

```
head(isoscape)
```

```
## $isoscape
## class       : RasterStack
## dimensions  : 514, 1115, 573110, 8  (nrow, ncol, ncell, nlayers)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## names       :          mean,  mean.predVar, mean.residVar,  mean.respVar,          disp,  disp.predVa
## min values  : -123.52350359,    6.66572055,   67.02201337,   89.97938852,   67.02201337,    0.012748
## max values  :  5.801003e+00,  2.246382e+02,  1.559210e+03,  1.580116e+03,  1.559210e+03,  6.066897e-0
##
##
## $sp.points
## $sp.points$sources
##         coordinates values
## 1     (35.3, 36.98)  -9999
## 2   (-27.19, 38.77)  -9999
## 3    (35.77, 32.09)  -9999
## 4    (36.14, 30.33)  -9999
## 5   (-27.19, 38.79)  -9999
##  [ reached getOption("max.print") -- omitted 322 rows ]
```

```
class(isoscape$isoscape)
```

```
## [1] "RasterStack"
## attr(,"package")
## [1] "raster"
```

```
class(isoscape$sp.points$sources)
```

```
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
```

The `RasterStack` consists of the following 8 layers:

```
names(isoscape$isoscape)
```

```
## [1] "mean"          "mean.predVar"  "mean.residVar" "mean.respVar"
## [5] "disp"          "disp.predVar"  "disp.residVar" "disp.respVar"
```

You can access a single `RasterLayers` as you handle any element in a list. For example this is how you access to the raster containing the mean predictions:

```
isoscape$isoscape$mean ## similar to: isoscape$isoscape[[1]] or isoscape$isoscape[["mean"]]
```

```
## class       : RasterLayer
## dimensions  : 514, 1115, 573110  (nrow, ncol, ncell)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : mean
## values      : -123.5235, 5.801003  (min, max)
```

In a similar way, you can access the weather station data, called sources:

```
isoscape$sp.points$sources
```

```
## class       : SpatialPointsDataFrame
## features    : 327
## extent      : -27.34, 57.1, 30.08, 68.96  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## variables   : 1
## names       : values
## min values  :  -9999
## max values  :  -9999
```

The guts of the objects of class *isorix*

Analogously, one can access all the information stored in the object of class *isorix*:

```
assignment
```

```
## ######### assignment raster(s): ' indiv '
## $stat
## class       : RasterBrick
## dimensions  : 514, 1115, 573110, 10  (nrow, ncol, ncell, nlayers)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       :     Mbe_1,     Mbe_2,     Mbe_3,     Mbe_4,     Mbe_5,     Mbe_6,     Mbe_7,     Mbe_8
## min values  : -45.56475, -40.32952, -50.33462, -54.05745, -46.84447, -36.37401, -39.28247, -44.63404
## max values  :  77.17656,  82.41179,  72.40668,  68.68385,  75.89684,  86.36730,  83.45884,  78.10727
##
##
## $stat.var
## class       : RasterBrick
## dimensions  : 514, 1115, 573110, 10  (nrow, ncol, ncell, nlayers)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       :    Mbe_1,    Mbe_2,    Mbe_3,    Mbe_4,    Mbe_5,    Mbe_6,    Mbe_7,    Mbe_8,    Mbe
## min values  :        0,        0,        0,        0,        0,        0,        0,        0,
## max values  : 316.7772, 318.5948, 315.7678, 315.4085, 316.4458, 320.4607, 319.0474, 317.0460, 315.76
##
##
```

```
## $pv
## class       : RasterBrick
## dimensions  : 514, 1115, 573110, 10  (nrow, ncol, ncell, nlayers)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       :     Mbe_1,     Mbe_2,     Mbe_3,     Mbe_4,     Mbe_5,     Mbe_6,     Mbe_7,     Mbe_8
## min values  :         0,         0,         0,         0,         0,         0,         0,         0
## max values  : 0.9999431, 0.9999970, 0.9999687, 0.9999775, 0.9999916, 0.9999833, 0.9999998, 0.9999912
##
##
##
## ######### assignment raster(s): ' group '
## $pv
## class       : RasterLayer
## dimensions  : 514, 1115, 573110  (nrow, ncol, ncell)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : layer
## values      : 0, 0.9987352  (min, max)
##
##
##
## NULL
```

Here, for each statistical output (e.g. p-value), a `RasterBrick` is stored containing the information for each single individual entered into the assignment dataset. You can thus retrieve the `RasterLayer` for a given statistic and a given individual. For example, the assignment probability of the individual 'Mbe_1' can be extracted from the`RasterBrick` `$indiv$pv` in the following way:

```
assignment$indiv$pv$Mbe_1 ## similar to: assignment$indiv$pv[["Mbe_1"]]
```

```
## class       : RasterLayer
## dimensions  : 514, 1115, 573110  (nrow, ncol, ncell)
## resolution  : 0.08333333, 0.08333333  (x, y)
## extent      : -31.55847, 61.35819, 28.06653, 70.89986  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : Mbe_1
## values      : 0, 0.9999431  (min, max)
```

The objects of class *isorix* also store the sources (e.g. water stations used to create the isoscape) as well as the locations of the calibration data as `SpatialPointsDataFrames` and can be accessed accordingly:

```
assignment$sp.points$sources
```

```
## class       : SpatialPointsDataFrame
## features    : 327
## extent      : -27.34, 57.1, 30.08, 68.96  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## variables   : 1
## names       : values
## min values  :  -9999
```

```
## max values  :  -9999
```

```
assignment$sp.points$calibs
```

```
## class       : SpatialPointsDataFrame
## features    : 178
## extent      : -7.896256, 19.456, 36.52158, 54.60025  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
## variables   : 1
## names       : values
## min values  :  -9999
## max values  :  -9999
```

Export your data to a GIS

Now that you know where you spatial information is, you can extract it by creating files that can be read by your GIS software.

Export `SpatialPolygons` and 'SpatialPointsDataFrames

We will use for this the ESRI shapefile format, which is a widely used interchange format to store vector data, i.e. in our case the `SpatialPolygons` (CountryBorders, OceanMask) and the `SpatialPointsDataFrames` (sources, calibration locations). There are two libraries in R that can be used to create such shapefile: **rgdal** and **maptools** (https://www.nceas.ucsb.edu/scicomp/usecases/ReadWriteESRIShapeFiles).

To save `SpatialPolygons`, a little workaround is needed as we have to add an attribute. Before to export anything, make sure that the projection is defined for all spatial data. By default, we are working with the projection WGS84.

```
## getting 'sources' ready
SourceLocations <- assignment$sp.points$sources
projection(SourceLocations) ## checking projection
```

```
## [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

```
## getting 'countries' ready
df <- data.frame(id = rownames(coordinates(CountryBorders)))
rownames(df) <- df$id
wrld_cntrs <- SpatialPolygonsDataFrame(CountryBorders, data = df)
projection(wrld_cntrs)
```

```
## [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

Everything is ready, we can now export the spatial files in the current working directory:

```
writeOGR(obj = SourceLocations,
         dsn = getwd(),
         layer = 'SourceLocs_rgdal',
         driver = 'ESRI Shapefile',
         overwrite = TRUE)

writeOGR(obj = wrld_cntrs,
         dsn = getwd(),
         layer = 'SourceLocs_rgdal',
```

```
         driver = 'ESRI Shapefile',
         overwrite = TRUE)
```

Export RasterLayers

RasterLayers can be exported with the package **raster** as ascii format or geotiff. More raster formats are available, depending on the GIS-system you have. e.g. SAGA, IDRISI or Erdas Imagine (type `?writeRaster`). We will export the raster storing the p-value for the individual 'Mbe_1':

```
Assignment_Prob_Mbe_1 <- assignment$indiv$pv[['Mbe_1']]
projection(Assignment_Prob_Mbe_1)
```

```
## [1] "+proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0"
```

```
writeRaster(Assignment_Prob_Mbe_1,
            filename = "ap1.asc",
            format = "ascii",
            overwrite = TRUE,
            NAflag = -9999)

writeRaster(Assignment_Prob_Mbe_1,
            filename = "ap2.tif",
            format = "GTiff",
            overwrite = TRUE,
            NAflag = -9999) ## this requires the package rgdal
```

The End

That is all for now! Here are the information of the R session we used:

```
sessionInfo()
```

```
## R version 3.4.0 (2017-04-21)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Sierra 10.12.5
##
## Matrix products: default
## BLAS: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.4/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] GISTools_0.7-4     rgeos_0.3-23       MASS_7.3-47
##  [4] RColorBrewer_1.1-2 maptools_0.9-2     rgdal_1.2-8
##  [7] raster_2.5-8       sp_1.2-5           IsoriX_0.6
## [10] knitr_1.16
##
## loaded via a namespace (and not attached):
```

```
##  [1] Rcpp_0.12.11    magrittr_1.5    lattice_0.20-35 stringr_1.2.0
##  [5] tools_3.4.0     grid_3.4.0      nlme_3.1-131    spaMM_2.1.6
##  [9] htmltools_0.3.6 yaml_2.1.14     rprojroot_1.2   digest_0.6.12
## [13] Matrix_1.2-10   evaluate_0.10.1 rmarkdown_1.6   proxy_0.4-17
## [17] stringi_1.1.5   compiler_3.4.0  backports_1.1.0 foreign_0.8-69
```