Direcciones del flujo fluvial

2020-07-24

Recientemente creé un una visualización de la distribución de las direcciones del flujo fluvial y también de las orientaciones costeras. A raíz de su publicación en los RRSS ([aquí](https://twitter.com/dr_xeo/status/1277978724034465798?s=2)) me pidieron que hiciera un post acerca de cómo lo hice. Pues bien, aquí vamos para empezar con un ejemplo de los ríos, la orientación costera es algo más compleja. Lo mismo hice para una selección de ríos europeos aquí en este [tweet](https://twitter.com/dr_xeo/status/1277243216828473345?s=20). No obstante, originalmente empecé con la orientación de las costas europeas.

{{% tweet “1265286552525180929” %}}

## Paquetes

En este post usaremos los siguientes paquetes:

|  |  |
| --- | --- |
| Paquete | Descripción |
| tidyverse | Conjunto de paquetes (visualización y manipulación de datos): ggplot2, dplyr, purrr,etc. |
| remotes | Instalación desde repositorios remotos |
| RQGIS3 | Interfaz entre R y QGIS3 |
| sf | Simple Feature: importar, exportar y manipular datos vectoriales |
| ggtext | Soporte para la representación de texto mejorado con ggplot2 |
| sysfonts | Cargar fuentes en R |
| showtext | Usar fuentes más fácilmente en gráficos R |
| circular | Funciones para trabajar con datos circulares |
| geosphere | Trigonometría esférica para aplicaciones geográficas |

En el caso del paquete RQGIS3 es necesario instalar QGIS en OSGeo4W [aquí](https://www.qgis.org/es/site/forusers/download.html). Más adelante explicaré la razón del uso de QGIS.

# instalamos los paquetes si hace falta  
if(!require("tidyverse")) install.packages("tidyverse")  
if(!require("remotes")) install.packages("remotes")  
if(!require("RQGIS3")) remotes::install\_github("jannes-m/RQGIS3")  
if(!require("sf")) install.packages("sf")  
if(!require("ggtext")) install.packages("ggtext")  
if(!require("circular")) install.packages("circular")  
if(!require("geosphere")) install.packages("geosphere")  
if(!require("sysfonts")) install.packages("sysfonts")  
if(!require("showtext")) install.packages("showtext")  
  
# paquetes  
library(sf)  
library(tidyverse)  
library(ggtext)  
library(circular)  
library(geosphere)  
library(RQGIS3)  
library(showtext)  
library(sysfonts)

# Consideraciones iniciales

Los ángulos en líneas vectoriales se basan en el ángulo entre dos vértices, y el número de vértices depende de la complejidad, y en consecuencia de la resolución, de los datos vectoriales. Por tanto, puede haber diferencias en usar distintas resoluciones de una línea vectorial, sea de la costa o del río como en este ejemplo. Una línea recta simplemente se construye con dos puntos de longitud y latitud.

Relacionado con ello está la fractalidad, una estructura aparentemente irregular pero que se repite a diferentes escalas, de la línea de costa o también del río. La característica más paradójica es que la longitud de una línea costera depende de la escala de medida, cuanto menor es el incremento de medida, la longitud medida se incrementa.

Existen dos posibiliades de obtener los ángulos de los vértices. En la primera calculamos el ángulo entre todos los vértices consecutivos.

Por ejemplo, imaginémonos dos puntos, Madrid (-3.71, 40.43) y Barcelona (2.14, 41.4).

¿Cuál es el ángulo de su línea recta?

bearingRhumb(c(-3.71, 40.43), c(2.14, 41.4))

## [1] 77.62391

Vemos que es el de 77º, o sea, dirección noreste. Pero, ¿y si voy de Barcelona a Madrid?

bearingRhumb(c(2.14, 41.4), c(-3.71, 40.43))

## [1] 257.6239

El angúlo es diferente porque nos *movemos* desde el noreste al suroeste. Podemos invertir fácilmente el ángulo para obtener el *movimiento* contrario.

# ángulo contrario de Barcelona -> Madrid  
bearingRhumb(c(2.14, 41.4), c(-3.71, 40.43)) - 180

## [1] 77.62391

# ángulo contrario de Madrid -> Barcelona  
bearingRhumb(c(-3.71, 40.43), c(2.14, 41.4)) + 180

## [1] 257.6239

La dirección en la que calculamos los ángulos es importante. En el caso de los ríos se espera que sea la dirección de flujo de origen a la desembocadura, ahora bien, un problema puede ser que los vértices, que construyen las líneas, no estén ordenados geográficamente en la tabla de atributos. Otro problema puede ser que los vértices empiecen en la desembocadura lo que daría al angúlo inverso como lo hemos visto antes.

Sin embargo, hay una forma más fácil. Podemos aprovechar los atributos de los sistemas de coordenadas proyectados (proyección Robinson, etc) que incluyen el ángulo entre los vértices. Este último enfoque lo vamos usar en este post. Aún así, debemos prestar mucha atención a los resultados según lo dicho anteriormente.

# Preparación

## Datos

Descargamos las líneas centrales de los ríos más grandes del mundo ([descarga](/files/RiverHRCenterlinesCombo.zip)), accesible también en [Zeenatul Basher et al. 2018](https://www.sciencebase.gov/catalog/item/5a145fdde4b09fc93dcfd36c).

## Importar y proyectar

Lo primero que hacemos es importar, proyectar y eliminar la tercera dimensión *Z*, usando el encadenamiento de las siguientes functions: st\_read() nos ayuda a importar cualquier formato vectorial, st\_zm() elimina la dimensión Z o M de una geometría vectorial y st\_transform() proyecta los datos vectoriales a la nueva proyección en formato *proj4*. La combinación de las funciones la realizamos con el famoso *pipe* (%>%) que facilita la aplicación de una secuencia de funciones sobre un conjunto de datos, más detalles en este [post](https://dominicroye.github.io/es/2020/una-muy-breve-introducci%C3%B3n-a-tidyverse/). Todas las funciones del paquete sf comienzan por st\_\* haciendo referencia al carácter espacial de su aplicación, similar a *PostGIS.* Igualmente, y al mismo estilo que *PostGIS*, se usan verbos como nombres de función.

proj\_rob <- "+proj=robin +lon\_0=0 +x\_0=0 +y\_0=0 +ellps=WGS84 +datum=WGS84 +units=m no\_defs"  
  
river\_line <- st\_read("RiverHRCenterlinesCombo.shp") %>%   
 st\_zm() %>%   
 st\_transform(proj\_rob)

## Reading layer `RiverHRCenterlinesCombo' from data source `C:\Users\xeo19\Documents\GitHub\blogR\_update\content\post\es\2020-07-24-direcciones-del-flujo-fluvial\RiverHRCenterlinesCombo.shp' using driver `ESRI Shapefile'  
## Simple feature collection with 78 features and 6 fields  
## geometry type: MULTILINESTRING  
## dimension: XYZ  
## bbox: xmin: -164.7059 ymin: -36.97094 xmax: 151.5931 ymax: 72.64474  
## z\_range: zmin: 0 zmax: 0  
## geographic CRS: WGS 84

## Extraer los ángulos

En el siguiente paso debemos extraer los ángulos de los vértices. Desgraciadamente, hasta donde sepa, no es posible extraer los atributos con alguna función del paquete sf. Aunque la función st\_coordinates() nos devuelve las coordenadas, no incluye otros atributos. Por eso, debemos usar otra forma, y es que el open software Quantum GIS extrae todos los atributos de los vértices. Podríamos importar los datos vectoriales en QGIS Desktop y exportar los vértices desde allí, pero también es posible acceder a las funciones de QGIS desde R directamente.

Para ello, tenemos que tener instalado QGIS en OSGeo4W. El paquete RQGIS3 nos permite de forma muy fácil usar las funciones del programa en R. Primero empleamos la función set\_env() para definir todas las rutas necesarias de QGIS e inciamos la API con open\_app().

# rutas a QGIS  
set\_env()

## Trying to find QGIS in C:/OSGEO4~1

## $root  
## [1] "C:/OSGeo4W64"  
##   
## $qgis\_prefix\_path  
## [1] "C:/OSGeo4W64/apps/qgis"  
##   
## $python\_plugins  
## [1] "C:/OSGeo4W64/apps/qgis/python/plugins"  
##   
## $platform  
## [1] "Windows"

# inicio de QGIS Python  
open\_app()

La función find\_algorithms() nos ayuda a buscar diferentes herramientas de QGIS. Además la función get\_usage() especifica la forma de uso con todos los argumentos requeridos.

# buscar herramientas  
find\_algorithms(search\_term = "vertices", name\_only = TRUE)

## [1] "native:extractspecificvertices"   
## [2] "native:extractvertices"   
## [3] "native:filterverticesbym"   
## [4] "native:filterverticesbyz"   
## [5] "native:removeduplicatevertices"   
## [6] "saga:convertpolygonlineverticestopoints"

# uso de la herramienta  
get\_usage(alg = "native:extractvertices")

## Extract vertices (native:extractvertices)  
##   
## This algorithm takes a line or polygon layer and generates a point layer with points representing the vertices in the input lines or polygons. The attributes associated to each point are the same ones associated to the line or polygon that the point belongs to.  
##   
## Additional fields are added to the point indicating the vertex index (beginning at 0)  
## the vertex’s part and its index within the part (as well as its ring for polygons)  
## distance along original geometry and bisector angle of vertex for original geometry.  
##   
##   
## ----------------  
## Input parameters  
## ----------------  
##   
## INPUT: Input layer  
##   
## Parameter type: QgsProcessingParameterFeatureSource  
##   
## Accepted data types:  
## - str: layer ID  
## - str: layer name  
## - str: layer source  
## - QgsProcessingFeatureSourceDefinition  
## - QgsProperty  
## - QgsVectorLayer  
##   
## OUTPUT: Vertices  
##   
## Parameter type: QgsProcessingParameterFeatureSink  
##   
## Accepted data types:  
## - str: destination vector file  
## e.g. d:/test.shp  
## - str: memory: to store result in temporary memory layer  
## - str: using vector provider ID prefix and destination URI  
## e.g. postgres:… to store result in PostGIS table  
## - QgsProcessingOutputLayerDefinition  
## - QgsProperty  
##   
## ----------------  
## Outputs  
## ----------------  
##   
## OUTPUT: <QgsProcessingOutputVectorLayer>  
## Vertices

En nuestro caso la herramienta para extraer los vértices es simple y sólo lleva una entrada y una salida. La función run\_qgis() ejecuta una herramienta de QGIS indicando el algoritmo y sus argumentos. La ventaja de usar el algoritmo directamente desde R es que podemos pasar objetos de clase sf (o sp) y raster que tenemos importados o creados en R. Como salida creamos un geojson, también podría ser de otro formato vectorial, y lo guardamos en una carpeta temporal. Al mismo tiempo le indicamos que importe el resultado directamente a R (load\_output = TRUE).

river\_vertices <- run\_qgis(alg = "native:extractvertices",  
 INPUT = river\_line,  
 OUTPUT = file.path(tempdir(), "rivers\_world\_vertices.geojson"),  
 load\_output = TRUE)

## $OUTPUT  
## [1] "C:/Users/xeo19/AppData/Local/Temp/RtmpE32MN0/rivers\_world\_vertices.geojson"

{{% alert note %}} Actualmente en Windows parece haber problemas con la librería de *proj*. En principio si termina creando el objeto river\_vertices no debes preocuparte. En caso contrario, recomiendo mirar la discusión en el *issue* abierto en [gitbub](https://github.com/r-spatial/RQGIS3/issues/20). {{% /alert %}}

## Selección

Antes de seguir con la estimación de la distribución de los ángulos, filtramos algunos ríos de interés. Las funciones de la colección tidyverse son compatibles con el paquete sf. En el último post hice una introducción a tidyverse [aquí](https://dominicroye.github.io/es/2020/una-muy-breve-introducci%C3%B3n-a-tidyverse/).

river\_vertices <- filter(river\_vertices,   
 NAME %in% c("Mississippi", "Colorado",   
 "Amazon", "Nile", "Orange",   
 "Ganges", "Yangtze", "Danube",  
 "Mackenzie", "Lena", "Murray",   
 "Niger")  
 )   
  
river\_vertices

## Simple feature collection with 94702 features and 11 fields  
## geometry type: POINT  
## dimension: XY  
## bbox: xmin: -10377520 ymin: -3953778 xmax: 13124340 ymax: 7507359  
## geographic CRS: WGS 84

## Warning in st\_is\_longlat(x): bounding box has potentially an invalid value range  
## for longlat data  
  
## Warning in st\_is\_longlat(x): bounding box has potentially an invalid value range  
## for longlat data

## # A tibble: 94,702 x 12  
## NAME SYSTEM name\_alt scalerank rivernum Length\_km vertex\_index vertex\_part  
## \* <chr> <chr> <chr> <dbl> <dbl> <dbl> <int> <int>  
## 1 Nile <NA> <NA> 1 4 3344. 0 0  
## 2 Nile <NA> <NA> 1 4 3344. 1 0  
## 3 Nile <NA> <NA> 1 4 3344. 2 0  
## 4 Nile <NA> <NA> 1 4 3344. 3 0  
## 5 Nile <NA> <NA> 1 4 3344. 4 0  
## 6 Nile <NA> <NA> 1 4 3344. 5 0  
## 7 Nile <NA> <NA> 1 4 3344. 6 0  
## 8 Nile <NA> <NA> 1 4 3344. 7 0  
## 9 Nile <NA> <NA> 1 4 3344. 8 0  
## 10 Nile <NA> <NA> 1 4 3344. 9 0  
## # ... with 94,692 more rows, and 4 more variables: vertex\_part\_index <int>,  
## # distance <dbl>, angle <dbl>, geometry <POINT [°]>

# Estimar la distribución

Para visualizar la distribución podemos usar, o bien un histograma o un gráfico de densidad. Pero en el caso de estimar la función de densidad de probabilidad nos encontramos con un problema matemático a la hora de aplicarlo a datos circulares. No debemos usar la función estandar de R density() dado que en nuestros datos una dirección de 360º es la misma a 0º, lo que provocaría errores en este rango de valores. Es un problema general para diferentes métricas estadísticas. Más detalles estadísticos se explican en el paquete circular. Este paquete permite definir las características de los datos circulares (unidad, tipo de datos, rotación, etc.) como una clase de objeto en R.

Por tanto, lo que hacemos es construir una función que estime la densidad y devuelva una tabla con los ángulos (x) y las estimaciones de densidad (y). Dado que los ríos tienen diferentes longitudes, y queremos ver diferencias independientemente de ello, normalizamos las estimaciones usando el valor máximo. A diferencia de la función density(), en la que el ancho de banda de suavizado bw es optimizado, aquí es requerido indicarlo. Es similar a definir el ancho de barra en un histograma. Existe una función de optimización para la banda, bw.nrd.circular() que se podría emplear aquí.

dens\_circ <- function(x){  
   
 dens <- density.circular(circular(x$angle, units = "degrees"),  
 bw = 70, kernel = "vonmises",  
 control.circular = list(units = "degrees"))  
   
 df <- data.frame(x = dens$x, y = dens$y/max(dens$y))  
   
 return(df)  
   
}

Para finalizar, estimamos la densidad de cada río de nuestra selección. Empleamos la función split() de *R Base* para obtener una tabla de cada río en una lista. Después aplicamos con la función map\_df() del paquete purrr nuestra función de estimación de densidad a la lista. El sufijo \_df permite que obtengamos una tabla unida, en lugar de una lista con los resultados de cada río. No obstante, es necesario indicar el nombre de la columna con el argumento .id, la que contendrá el nombre de cada río. En caso contrario no sabríamos diferenciar los resultados. También aquí recomiendo leer más detalles en el último post sobre tidyverse [aquí](https://dominicroye.github.io/es/2020/una-muy-breve-introducci%C3%B3n-a-tidyverse/).

dens\_river <- split(river\_vertices, river\_vertices$NAME) %>%   
 map\_df(dens\_circ, .id = "river")  
  
# resultado  
head(dens\_river)

## river x y  
## 1 Amazon 0.000000 0.2399907  
## 2 Amazon 0.704501 0.2492548  
## 3 Amazon 1.409002 0.2585758  
## 4 Amazon 2.113503 0.2679779  
## 5 Amazon 2.818004 0.2774859  
## 6 Amazon 3.522505 0.2871232

# Visualización

Ahora ya sólo nos queda la visualización mediante el famoso paquete ggplot. Primero añadimos una nueva fuente *Montserrat* para usarla en este gráfico.

# descarga de fuente  
font\_add\_google("Montserrat", "Montserrat")  
  
# usar showtext para fuentes  
showtext\_opts(dpi = 200)  
showtext\_auto()

En el siguiente paso creamos dos objetos con el título y con una nota de pie. En el título estamos usando un código html para dar color a una parte de texto en sustitución de una leyenda. Se puede usar html de forma muy fácil con el paquete ggtext.

# título con html   
title <- "Relative distribution of river <span style='color:#011FFD;'><strong>flow direction</strong></span> in the world"  
  
  
caption <- "Based on data from Zeenatul Basher, 20180215"

La cuadrícula de fondo que crea ggplot por defecto para coordenadas polares no me convenció, por eso creamos una tabla con las líneas de fondo del eje x.

grid\_x <- tibble(x = seq(0, 360 - 22.5, by = 22.5),   
 y = rep(0, 16),   
 xend = seq(0, 360 - 22.5, by = 22.5),   
 yend = rep(Inf, 16))

A continuación definimos todos los estilos del gráfico. Lo más importante en este paso es la función element\_textbox() del paquete ggtext para poder interpretar nuestro código html incorporado al título.

theme\_polar <- theme\_minimal() +  
 theme(axis.title.y = element\_blank(),  
 axis.text.y = element\_blank(),  
 legend.title = element\_blank(),  
 plot.title = element\_textbox(family = "Montserrat",   
 hjust = 0.5,   
 colour = "white",   
 size = 15),  
 plot.caption = element\_text(family = "Montserrat",   
 colour = "white"),  
 axis.text.x = element\_text(family = "Montserrat",   
 colour = "white"),  
 strip.text = element\_text(family = "Montserrat",   
 colour = "white",   
 face = "bold"),  
 panel.background = element\_rect(fill = "black"),  
 plot.background = element\_rect(fill = "black"),  
 panel.grid = element\_blank()  
 )

Para terminar construimos el gráfico: 1) Usamos la función geom\_hline() con diferentes puntos de intersección en *y* para crear la cuadrícula de fondo. La función geom\_segment() crea la cuadrícula en *x*. 2) El área de densidad la creamos usando la función geom\_area(). 3) En scale\_x\_continous() definimos un límite inferior negativo para que no colapse en un punto pequeño. Las etiquetas de las ocho direcciones principales las indicamos en la función scale\_y\_continous(), y 4) Por último, cambiamos a un sistema de coordenadas polar y fijamos la variable para crear facetas.

ggplot() +  
 geom\_hline(yintercept = c(0, .2, .6, .8, 1), colour = "white") +  
 geom\_segment(data = grid\_x ,   
 aes(x = x, y = y, xend = xend, yend = yend),   
 linetype = "dashed", col = "white") +  
 geom\_area(data = dens\_river,   
 aes(x = x, y = y, ymin = 0, ymax = y),   
 alpha = .7,   
 colour = NA,   
 show.legend = FALSE,  
 fill = "#011FFD") +   
 scale\_y\_continuous(limits = c(-.2, 1), expand = c(0, 0)) +  
 scale\_x\_continuous(limits = c(0, 360),   
 breaks = seq(0, 360 - 22.5, by = 22.5),  
 minor\_breaks = NULL,  
 labels = c("N", "", "NE", "", "E", "", "SE", "",  
 "S", "", "SW", "", "W", "", "NW", "")) +  
 coord\_polar() +   
 facet\_wrap(river ~ ., ncol = 4) +  
 labs(title = title, caption = caption, x = "") +  
 theme\_polar

## Warning: Ignoring unknown aesthetics: ymin, ymax

