

## HW3: Geospatial data handling

Total points: 6

In this HW, you are going to collect, store, query/process and visualize spatial data ☺

You'll be using the KML file format, shapefile format (.shp), Google Earth, Postgres+PostGIS, ArcGIS, JavaScript, R.

The exercise will give you a taste of working with spatial data, use of spatial file formats and spatial query functions, use of JS and R libraries for plotting map coords, generating spatial data via calculation - all of which are quite useful from a real-world (or job interview) perspective.

1. You need to create (generate/collect) latitude,longitude pairs (ie. spatial coordinates) for **13 locations**. One of those will be where your home/apartment/dorm room is. The other 12 would have to be these: 6 libraries, 6 'waterworks' (fountains, etc.). If you are not on campus, pick 6+6 locations belonging to two other categories (eg. restaurants, coffeeshops, bars, grocery stores, downtown buildings...). What if you are stuck at home? You can sample locations of the campus, on [Google Maps](#) [by long-pressing].

How would you obtain (lat,long) spatial coordinates at a location where you are? You can do so, one of two ways:

- **using the Chrome browser**, simply bring up [this page](#) on your smartphone, and write down the (latitude,longitude) values that get shown when you load/refresh the page :) As you can see, the page shows your location on a map - cool! Be sure to enable cross-site script loading when you run this - click on the shield icon at the right of the URL bar, and click on 'Load unsafe scripts'. Alternately, you can use [this page](#) to obtain the (latitude,longitude) coordinates - for this to work, be sure that the browser url starts with https:// [type this in, if you need to].

- **using your phone's built-in GPS/compass app**, simply read off the displayed GPS coordinate values (if the coordinate display is in degrees, minutes and seconds, you need to convert the minutes,seconds pair of values into a single fractional degree value - one degree is subdivided into 60 minutes (60'), and one minute is subdivided into 60 seconds (60'') - so for example, 30'15", since it is equivalent to 1815", would be eqvt to  $1815/3600=0.504$  degrees.

Also, be sure to write down the location names as well (you will use them to label your points when displaying). AND, take a selfie (!) that clearly shows the location you're sampling - **this step is to ensure that you're not simply reading off the coords from a map, sitting at home!** You'll lose points if you don't submit selfies. Alternatively, you can simply take a photo of the location you are sampling, and submit that instead (no need for selfies).

2. Now that you have 13 coordinates and their label strings (ie. text descriptions such as "Tommy Trojan", "SAL", "Chipotle"..), you are going to create a KML file (.kml format, which is XML) out of them using a text editor. Specifically, each location will be a 'placemark' in your .kml file (with a label, and coords). [Here is more detail](#). The .kml file with the 13 placemarks is going to be your starter file, for doing visualizations and queries. [Here is a .kml skeleton](#) to get you started (just download, rename and edit it to put in your coords and labels). NOTE - keep your labels to be 15 characters or less (including spaces). [Here is the same .kml skeleton in .txt format](#), if you'd like to RMB save it instead and rename the file extension from .txt to .xml. NOTE too that in .kml, you specify (long,lat), instead of the expected (lat,long) [after all, longitude is what corresponds to 'x', and latitude, to 'y']!

In the KML file, be sure to place your home, libraries, waterworks (or equivalents if you are not on campus) in their own layer - ie. create three layers.

You are going to use Google Earth to visualize the data in your KML file (see #3 below). FYI, as a quick check, you can also visualize it using [this page](#) :)

Loading [MathJax]/extensions/MathZoom.js

3. Download Google Earth on your laptop, install it, bring it up. Load your .kml file into it - that should display all your sampled locations, on Google Earth's globe :)

4. Install Postgres+PostGIS on your PC/Mac laptop, and browse the docs for the spatial functions.

4 (alt). You can also use MySQL if you want, or Oracle 11g+Oracle Spatial, or even sqlite; if you are familiar with using SQL Server, that can also help you do the homework. Even QGIS can be used to do the HW.

4 (alt alt). IF YOU ARE FEELING ADVENTUROUS: as an alternative to installing Oracle or Postgres (or MySQL or sqlite or SQL Server...) on your machine, you can use Postgres or Oracle on the AWS cloud platform (ie. without installing anything on your laptop!) - eg. see this page, and this one. **Be sure to not leave your DB instance running, when you aren't working on the hw!**

4 (alt alt alt). Last but not least, do feel free to use GCP for this! Here are some relevant resources:

\* <https://cloud.google.com/sql/docs/postgres/quickstart>

\* <https://medium.com/google-cloud/postgres-is-incredibly-awesome-c54353b88655>

\* <https://cloudplatform.googleblog.com/2017/03/Cloud-SQL-for-PostgreSQL-managed-PostgreSQL-for-your-mobile-and-geospatial-applications-in-Google-Cloud.html>

\* <https://cloudplatform.googleblog.com/2017/08/Cloud-SQL-for-PostgreSQL-updated-with-new-extensions.html>

5. You will use the spatial db software to execute the following two spatial queries that you'll write:

- **compute the convex hull** for your 13 points [a convex hull for a set of 2D points is the smallest convex polygon that contains the point set]. If you use Oracle, see this page; if you decide to use Postgres, read this and this instead. Use the query's result polygon's coords, to create a polygon in your .kml file (edit the .kml file, add relevant XML to specify the KML polygon's coords). Load this into Google Earth, visually verify that all your points are on/inside the convex hull, then take a screenshot. Note that even your data points happen to have a concave perimeter and/or happen to be self-intersecting, the convex hull, by definition, would be a tight, enclosing boundary (hull) that is a simple convex polygon. The convex hull is a very useful object - eg. see this discussion.. Note: be sure to specify your polygon's coords as '...-118,34 -118,34.1...' for example, and not '...-118, 34 -118, 34.1...' [in other words, do not separate long,lat with a space after the comma, ie it can't be long, lat].

- **compute the nearest neighbor** of your home/apt/dormroom location [look up the spatial function of your DB, to do this]. Use the query's results, to create a line segment in your .kml file: line(home,nearest\_neighbor). Verify this looks correct, using Google Earth; take a snapshot.

- for fun (this doesn't fetch points!), **compute the centroid** of your locations

Note - it \*is\* OK to hardcode points, in the above queries! Or, you can create and use a table to store your 13 points in it, then write queries against the table.

6. Use OpenLayers (a JavaScript API) to visualize your location data. The idea is to store your 13 sampled points, via your web browser, in a browser cache area in your local machine, where the data would persist [even after you close the browser]; then you'd read back the stored values, and visualize them, using the OpenLayers API. To store and load points, you'll use 'HTML5 localStorage', which is a key-value based standard WWW API. In the future, you can use localStorage to persist any other kind of data - browser-based games' states, e-commerce purchases, computations-in-progress, UI choices in your apps...

Run this: <https://bytes.usc.edu/~saty/tools/xem/run.html?x=OpenLayers> - cool! It shows a single location (TT!) being plotted. Examine the source code, you will see it uses OpenLayers.js (which is where the OpenLayers API is).

If you are having problems w/ the above xem link, run this instead - it won't give you any errors: [https://bytes.usc.edu/~saty/tools/xem/run.html?x=OpenLayers\\_v2](https://bytes.usc.edu/~saty/tools/xem/run.html?x=OpenLayers_v2) [it uses a different version of the OpenLayers lib].

Also, you can copy the .html (from v2 above) to a local .html, eg. oL\_v2.html, then run it using a 'local webserver' like so - it is VERY useful to know how to create and use a local webserver - this is the EASIEST way!...

Loading (MathJax)/extensions/MathZoom.js

The code in my 'xem' page above, is to show you how to do it; you'd need to do it 'for real' by modifying it :) EVEN IF you don't know JS or coding, don't worry - it's easy, and fun, to figure it out! You don't need to any 'big' modifications - you ONLY need to swap out the existing data to put in your own!

You'd rearrange/modify the JS code I'm giving you, to do this:

- \* create an object (looks like JSON!) to contain your 13 points
- \* use localStorage.setItem() to PUT (store) the data in your laptop
- \* use localStorage.getItem() to GET (retrieve) the data in your laptop
- \* loop through, and plot the data (ie plot your 13 locations)

If you like, you can do the above, using CodePen [<https://codepen.io/>] or jsfiddle [<https://jsfiddle.net/>], and just submit a URL of your code - the grader would simply copy and paste your URL into their browser, and be able to see your code and the result. If you do this, submit a README file with your link, instead of submitting your own .html.

7. Install R, then RStudio: <https://techvidvan.com/tutorials/install-r/#::~text=Step%20%E2%80%93%201%3A%20With%20R%2D,%20and%20follow%20the%20installation%20instructio>

Next, bring up RStudio - it's a cool IDE for executing R code :)

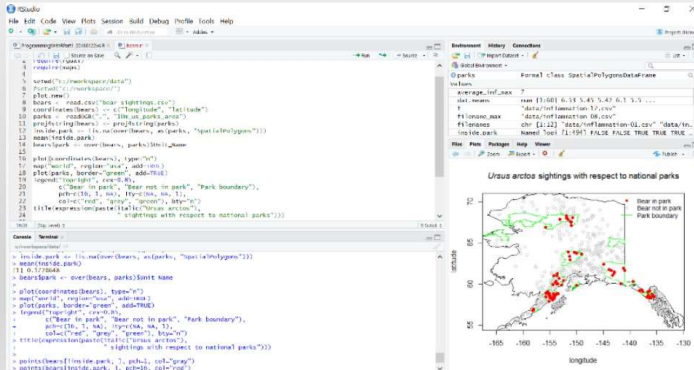
## BASICS OF R (II)

### WHAT IS R STUDIO?

- R Studio is cross-platform "integrated development environment" for R
- It allows us to save R commands to script files, view variables as we define them, and see output and visualizations directly in the environment
- It runs on Mac and Windows

#### The R Studio IDE

- Console
- Terminal
- Script Editor
- Variables
- Plots, Graphics, Maps!
- Exports
- Package import



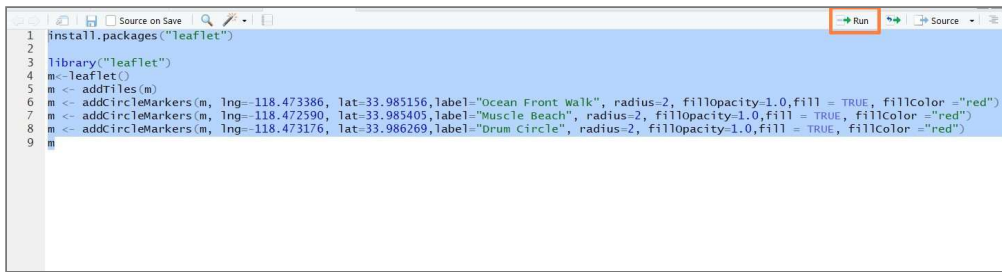
The screenshot shows the RStudio interface. The top-left pane contains an R script with code for loading data, plotting points, and creating a map. The bottom-left pane shows the console with the output of the script. The top-right pane shows the Environment window with a list of objects. The bottom-right pane shows a map of Alaska with red dots representing Ursus arctos sightings and green lines representing national park boundaries. The map is titled 'Ursus arctos sightings with respect to national parks'.

[image from [here](#)]

You are going to be using the R port of the popular 'leaflet' map library [written by a Ukrainian!!], for visualizing your 13 sampled locations. In the RStudio file menu, open [this](#) .R file [save the .R file to your HW3 area first, using RMB on the 'this' link].

Uncomment `##install.packages("leaflet")`, to let RStudio automatically download (from an online R repository it knows about) and install leaflet. From the next time on, you can comment that line out, to avoid re-installing it over and over.

After uncommenting, select all the code (Ctrl a), then 'Run' it - cool!:



```

1 install.packages("leaflet")
2
3 library("leaflet")
4 m <- leaflet()
5   <- addTiles(m)
6   <- addCircleMarkers(m, lng=-118.473386, lat=33.985156, label="Ocean Front Walk", radius=2, fillOpacity=1.0, fill = TRUE, fillColor = "red")
7   <- addCircleMarkers(m, lng=-118.472590, lat=33.985405, label="Muscle Beach", radius=2, fillOpacity=1.0, fill = TRUE, fillColor = "red")
8   <- addCircleMarkers(m, lng=-118.473176, lat=33.986269, label="Drum Circle", radius=2, fillOpacity=1.0, fill = TRUE, fillColor = "red")
9

```

Look at the code (in R, we use `<-` instead of `=`) - we create a leaflet object called `m`, then load map data (`addTiles()`), then add markers for our data (`addCircleMarkers()`), then display it (by simply 'printing' `m`, Python-style).

Edit the marker data, to make it be your 13 locations. Take a screenshot, save your .R file.

In the future, you can do a LOT more with R and spatial data - all using the RStudio IDE you now have :) Eg. read [this](#) [where R packages called `rgdal` and `rgeos` are used for map visualization] and [this](#) [which shows how to use `rgdal` and many more packages].

AND, you can start learning more R (which as you can see, is a LOT like Python!) - it is heavily used in data analysis (R was purpose-built for data handling, after all!). In a sense, R is like the ultimate statistics calculator ever!

8. Using SGM 123 as the center, **compute** (don't/can't use GPS!) a set (sequence) of lat-long (ie. spatial) coordinates that lie along a pretty Spirograph™ curve :)

Create a new KML file with Spirograph™ curve points [see below], convert the KML to an ESRI 'shapefile', visualize the shapefile data using ArcGIS Online, and submit these four items: your point generation code (see below), the resulting .kml file ("spiro.kml"), shapefile (this needs to be a .zip) and a screenshot ("spiro.jpg" or "spiro.png").

DEN students: you can use as the center, a different spatial coordinate (eg. that of your home).

To convert your .kml into a shapefile, use this online converter: <https://mygeodata.cloud/converter/kml-to-shp> - the result will be a .zip [which is what we call 'shapefile'], which will contain within it, shape data (.shp), a relational table (.dbf), and other optional files (.shx, .prj, .cpg). Extra fyi - [here](#) is a page on shapefiles, and [this](#) talks about the various components (.shp, .dbf etc) of a shapefile.

Once you have your shapefile, you can upload it to ArcGIS' online map creator to view your Spirograph™ curve-shaped points. To do so, log on to ArcGIS [after creating a free 'public' account], at <https://www.arcgis.com/>, then use the 'Map' tab - <https://www.arcgis.com/home/webmap/viewer.html?useExisting=1>. Do 'Add -> Add Layer from File', and upload your shapefile .zip, you should see your data overlaid on a map. Here is a screenshot of roughly what to expect. For fun, you can also upload your shapefile [here](https://mapshaper.org/), to view it: <https://mapshaper.org/>

For the Spirograph™ curve point creation, use the following parametric equations (with  $R=8$ ,  $r=1$ ,  $a=4$  -- do not change these values :)):

$$x(t) = (R+r) \cdot \cos((r/R) \cdot t) - a \cdot \cos((1+r/R) \cdot t)$$

$$y(t) = (R+r) \cdot \sin((r/R) \cdot t) - a \cdot \sin((1+r/R) \cdot t)$$

Using the above equations, loop through  $t$  from 0.00 to  $n \cdot \pi$  (eg.  $2 \cdot \pi$ ; note that 'n' might need to be more than 2, for the curve to close on itself; and,  $t$  is in radians, not degrees), in steps of 0.01. That will give you the sequence of (x,y) points that make up the Spiro curve, which would/should look like the curve in the right side of the screengrab below, when  $R=8$ ,  $r=1$ ,  $a=4$  (my JavaScript code for the point generation+plotting loop is on the left):

Loading [MathJax]/extensions/MathZoom.js

```

var canvas = document.getElementById('canvas');
if (canvas.getContext){
  var ctx = canvas.getContext('2d');

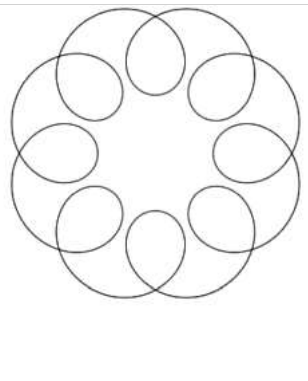
  ctx.beginPath();

  //x(t) = (R+r)*cos((r/R)*t) - a*cos((1+r/R)*t)
  //y(t) = (R+r)*sin((r/R)*t) - a*sin((1+r/R)*t)

  var R=8, r=1, a=4;
  var x0=R+r-a, y0=0;
  ctx.moveTo(150+10*x0,150+10*y0);

  var cos=Math.cos, sin=Math.sin, pi=Math.PI, nRev=16;
  for(var t=0.0;t<(pi*nRev);t+=0.01) {
    var x=(R+r)*cos((r/R)*t) - a*cos((1+r/R)*t);
    var y=(R+r)*sin((r/R)*t) - a*sin((1+r/R)*t);
    ctx.lineTo(150+10*x,150+10*y);
    //document.write(x+", "+y);
  }
  ctx.stroke();

```



Note - your figure MUST resemble the above, ie. it MUST have 8 loops.

In order to center the Spirograph™ at a given location [SGM123 or other], you need to ADD each (x,y) curve point to the (lat,long) of the centering location - that will give you valid Spiro-based spatial coords for use in your .kml file. You can use any coding language you want, to generate (and visualize) the curve's coords: JavaScript, C/C++, Java, Python, SQL, MATLAB, Scala, Haskell, Ruby, R.. You can also use Excel, SAS, SPSS, JMP etc., for computing [and plotting, if you want to check the results visually] the Spirograph™ curve points.

Payoff - what you'll see is the Spirograph™ curve, superposed on the land imagery - pretty!

PS: Here is MUCH more on Spirograph™ (hypocycloid and epicycloid) curves if you are curious. Also, for fun, try changing any of R, r, a in the code for the equations above [you don't need to submit the results]!

Here is what you need to **submit (as a single .zip file)**:

- \* your .kml file from step 5 above - with the placemarks, convex hull and nearest-neighbor line segments (**1 point**)
- \* your selfie pics that 'prove' you actually collected the point locations on site (.jpg or .png) (**no points** for selfies submission, but, **-2 points** IF YOU DON'T SUBMIT ALL OF THEM); if you are not comfortable submitting selfies, you can take a pic of the location (without any identifying info about you) and submit that (again, the point of asking for a pic is to have you visit the place, which makes it a more authentic data collection experience)
- \* a text file (.txt or .sql) with your two queries from step 5 - table creation commands (if you use Postgres and directly specify points in your queries, you won't have table creation commands, in which case you wouldn't need to worry about this part), and the queries themselves; a screenshot that shows the locations, nearest neighbor line, and the convex hull polygon (**2 points**)
- \* a screenshot of your 13 locations, plus, a .html file (with the JS OpenLayers code) from step 6, or a CodePen/jsfiddle link (**1 point**)
- \* from step 7: a screenshot of the visualized locations, plus your .R script (**1 point**)
- \* a screenshot of your Spirograph™ result from step 8, plus the source code in text form (eg. spiro.{js,py,java,cpp}), .kml, .shp (**1 point**)

HAVE FUN! From here on out, you know how to create custom overlays (via KML files containing vector symbols constructed from points, lines and polygons, and its shapefile equivalent) on a map, and perform spatial queries on the underlying data :)