Introduction

As an avid cyclist, several times I observed discrepancies between the elevation gains predicted by different apps (Google Maps, Komoot) or the one computed by my garmin device. This project aims to build, in the most accurate and precise way, precise elevation profiles from navigation data. I developed my algorithm looking at roads in Corsica (by the way, the best cycling destination if you ask me) because I cycled them a lot : I have a lot of data to build a consistent elevation profile and I know them well enough to assess its validity. Here come the main steps.

Summary

- (I) Projection on the open street map network
- (II) Handling position errors
- (III) Handling elevation errors
- (IV) Graph decomposition

(I) Projection on the open street map network

Before estimating the elevation profile of a road, one obviously needs its coordinates. The python library osmnx provides users with a networkx graph representing the open steet map roads of a given area along with efficient functions to project the recorded points on the closest road.

On top of giving us the right frame to match navigation points on the corresponding roads, it reduces the number of dimensions we're dealing with from 3 (latitude,longitude,elevation) to 2 (position on the road,elevation) which highly simplifies computations.

(II) Handling position errors

Given an edge of the osm graph, i.e. a street or some road portion, projecting the relevant navigation data along yields a bunch of curves, the x-axis being the position on the road and the y-axis the elevation. As mentioned above, the latitude and longitude coordinates are merged during the projection along with their measurement errors.

All the curves obtained appear to be shifted version from each other in both x and y axis. One needs to estimate the x-axis shift, i.e. realign them before estimating a point-wise elevation (which can be interpreted as computing the y-axis shift).

(III) Handling elevation errors

Once the different curves have been aligned, our goal is to extract the most accurate yet precise elevation profile. With a lot of navigation data from different sources, we should be able to make a difference between noise and subtle elevation variations. To do so, one can either consider the median curve or try to fit a piecewise linear affine model to match the curves. Both have their pros and cons and will be combined into a final method.

(IV) Graph Decomposition

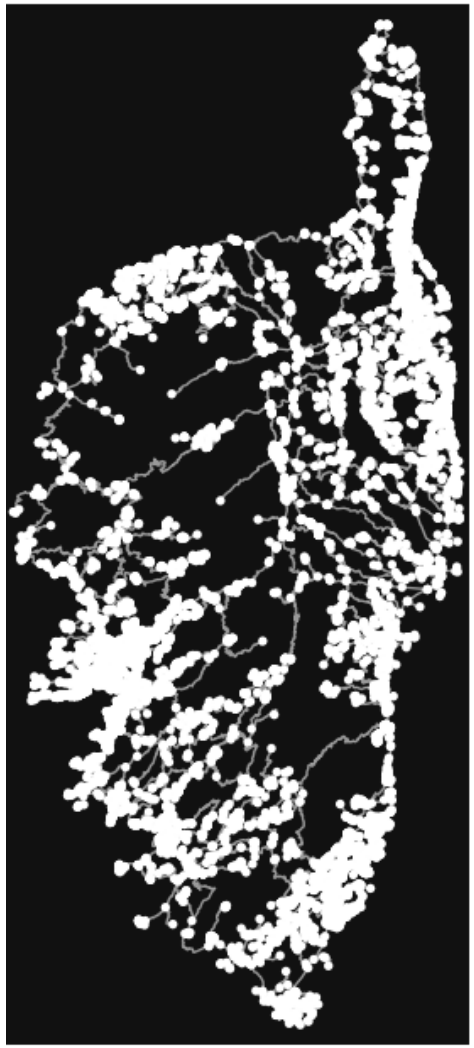
When projecting navigation data on a graph edge, the curves need to be singular enough to be realigned and by singular the author means far from affine : there is no way to estimate both the x and y axis shifts for two affine curves sharing the same slope.

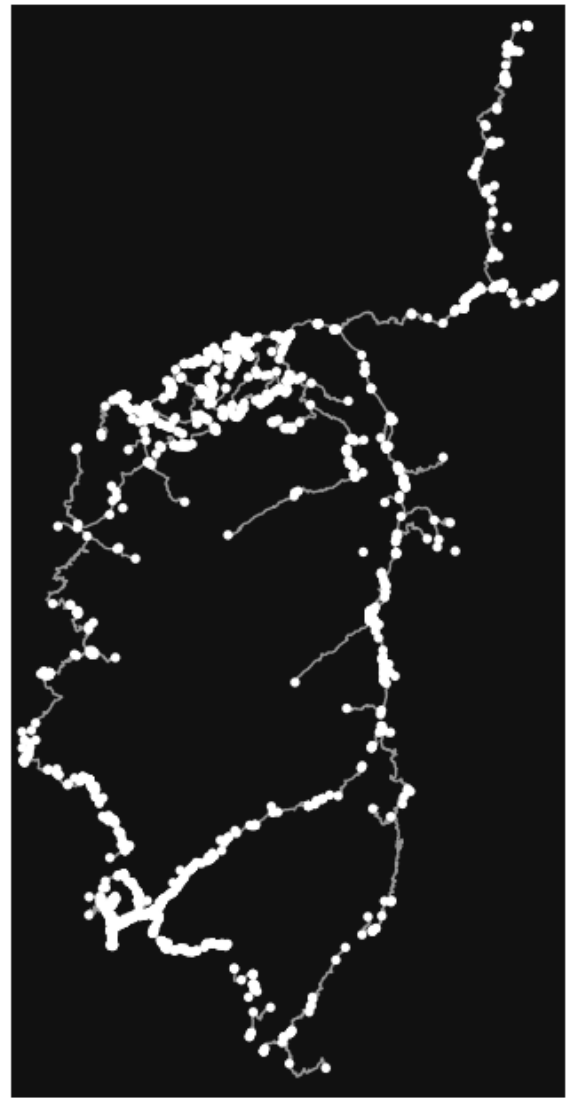
If the road is long enough, so that enough singularities appear, we might try to realign the corresponding curves. Unfortunately, especially in densely connected areas, i.e. with a lot of roads intersection, the graph edges are really shorts and the corresponding curves are likely to be affine.
If instead of estimating the elevation profile edge-wise, we look at navigation data along a long path in the graph we might succeed.

This raises the question of a good graph decomposition : find a partition of the graph edges such that the edges within a subset may be concatenate to form a long path upon which it is possible to compute the x shift axis. Once the final slope is computed for a given path, one can just project it on the underlying edges.

(I) Projection on the open street map network

The python library osmnx (https ://osmnx.readthedocs.io/en/stable/) provides us with data modeling roads network by a networkx MultiDiGraph object and useful functions to project our spatial data.





The graph above on the left represent the roads network of Corsica, an island in the Mediterranean sea with wonderful mountains and beaches that make it a heaven for outdoors lovers and especially cyclists. The one of the right is the subgraph induced by the edges that have been appear among the recorded navigation data.

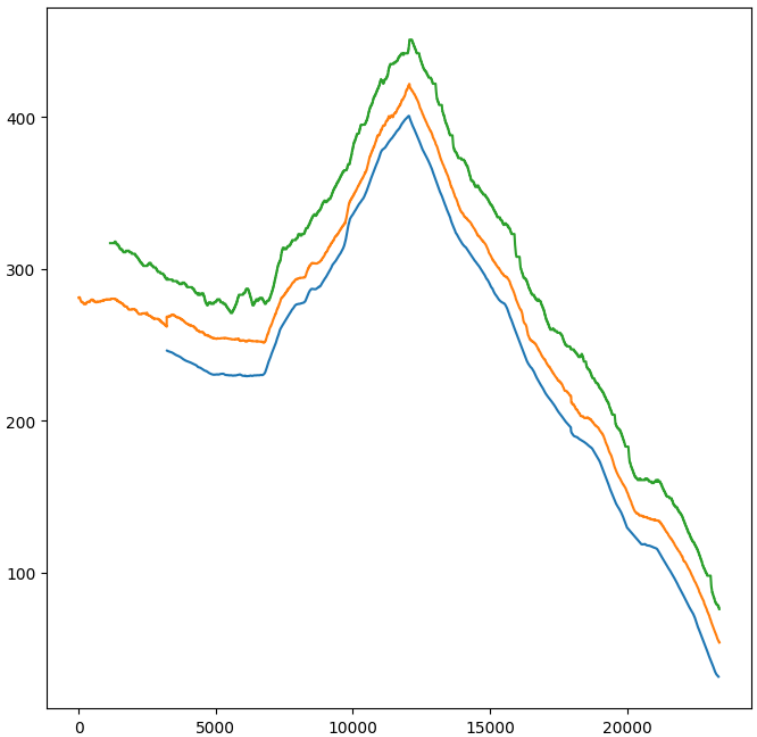
The edges come with a shapely LineString object encoding the road's geometry and the nodes correspond to intersections.

On the other hand, we have a lot of cycling activities recorded by a device. Each activity is a GeoDataFrame, each row represent a point and has the following features : latitude, longitude, elevation and time.
Given an activity, for each point, we can find the closest edge and project the point onto it. We end up with a large GeoDataFrame with the following features :

- "*file_path*" : the source from which the point has been sampled, i.e. the activity.
- "*geometry*" : a shapely Point object representing the projected point on the edge.
- "*time*" : the sampling timestamp.
- "*elevation*" : the point elevation.
- "*edge*" : the edge upon which the point has been projected.
- "*edge_coordinate*" : the coordinate on the oriented edge's LineString, the x-axis for the elevation profile.

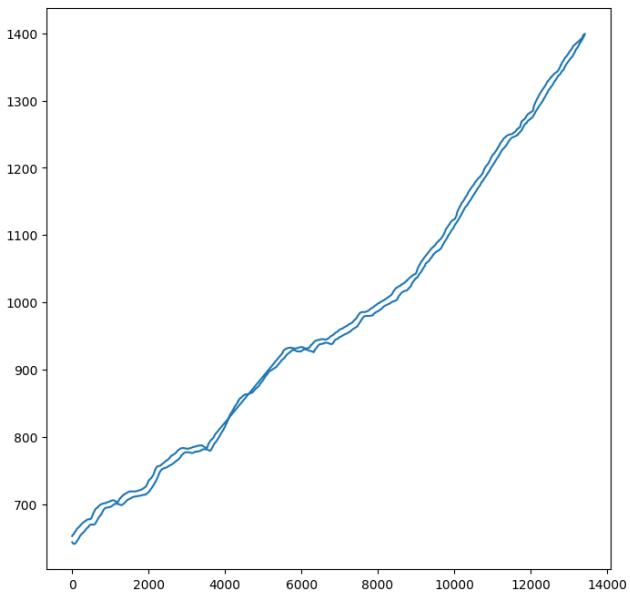
	file_path	geometry	elevation	time	edge	distance_ls
0	/home/pierre/Documents /garmin_corse /activity_1...	POINT (482452.154 4710449.517)	8.200000	2023-09-05 13:57:31+00:00	(51319395, 9064150112, 0)	406.639341
1	/home/pierre/Documents /garmin_corse /activity_1...	POINT (482455.064 4710445.288)	30.200001	2023-09-05 13:57:32+00:00	(51319395, 9064150112, 0)	411.773043
2	/home/pierre/Documents /garmin_corse /activity_1...	POINT (482458.498 4710440.297)	32.400002	2023-09-05 13:57:33+00:00	(51319395, 9064150112, 0)	417.831396
3	/home/pierre/Documents /garmin_corse /activity_1...	POINT (482461.864 4710435.404)	33.000000	2023-09-05 13:57:34+00:00	(51319395, 9064150112, 0)	423.769641
4	/home/pierre/Documents /garmin_corse /activity_1...	POINT (482464.925 4710430.956)	33.400002	2023-09-05 13:57:35+00:00	(51319395, 9064150112, 0)	429.169124
...	...	...	...	...	...	...
390	/home/pierre/Documents /garmin_corse /albin_2.gpx	POINT (482316.873 4710646.124)	69.000000	2021-04-01 14:31:32+00:00	(51319395, 9064150112, 0)	167.986536
391	/home/pierre/Documents /garmin_corse /albin_2.gpx	POINT (482314.310 4710649.848)	69.000000	2021-04-01 14:31:33+00:00	(51319395, 9064150112, 0)	163.465646
392	/home/pierre/Documents /garmin_corse /albin_2.gpx	POINT (482312.549 4710652.407)	69.000000	2021-04-01 14:31:34+00:00	(51319395, 9064150112, 0)	160.359109
393	/home/pierre/Documents /garmin_corse /albin_2.gpx	POINT (482311.203 4710654.365)	70.000000	2021-04-01 14:31:35+00:00	(51319395, 9064150112, 0)	157.983103

In order to represent the elevation profile of an edge, one needs to organize the navigation data into segments : a serie of consecutive (i.e. ordered by time) points from the same activity projected on the given edge.

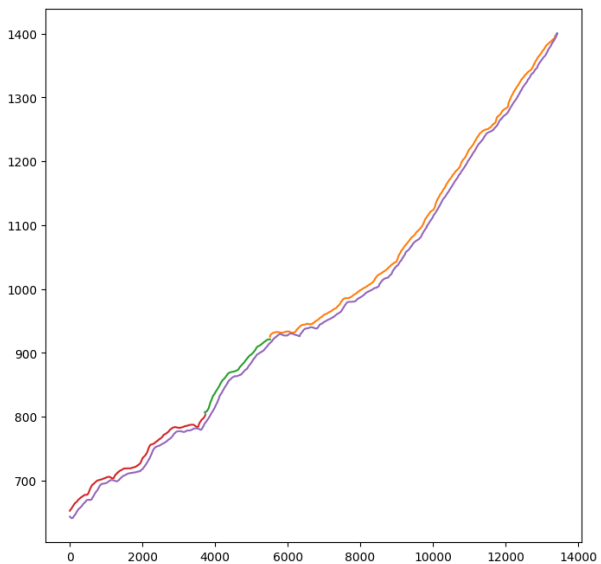


What could go wrong ? The x coordinates of the segment points might not form a monotonic sequence which happen when the cyclist climbed the road to a pass and just cycled it in the opposite direction or if he has lost some water bottle and needs to go backwards to pick it up and then start moving onwards again.

Below is my climbing up to the ski resort of Haut Asco (yes you can also ski in Corsica and go to the beach after in less than 2 hours) followed by the downhill.



To get monotonic sequences, we just split the segments such that the x coordinates within a sub segment form a monotonic sequencs. On top of giving us real curves which might be treated as proper mathematical functions (that will be useful when computing the shift in the x-axis), we get twice more data : this road appears once in only one activity but we already have to different elevation profiles that are slightly different. Even though they both come from the same device, the measurement errors have changed in time and are not the same on the climbing and downhill parts.



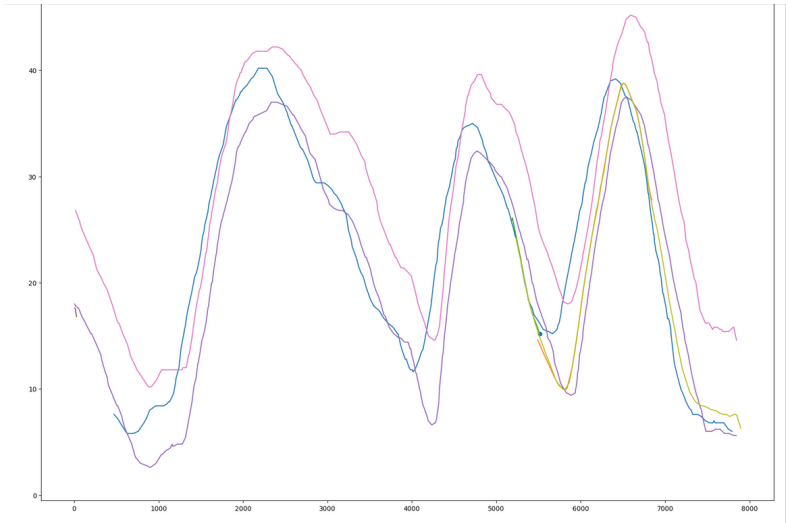
This is what the final dataframe looks like :

	file_path	geometry	elevation	time	edge	edge_coordinate	pre_segment	orientation	segment
0	./garmin_activities /activity_12322559591.gpx	POINT (482452.154 4710449.517)	8.200000	2023-09-05 13:57:31+00:00	(51319395, 9064150112, 0)	406.639341	0	1.0	0
1	./garmin_activities /activity_12322559591.gpx	POINT (482455.064 4710445.288)	30.200001	2023-09-05 13:57:32+00:00	(51319395, 9064150112, 0)	411.773043	0	1.0	0
2	./garmin_activities /activity_12322559591.gpx	POINT (482458.498 4710440.297)	32.400002	2023-09-05 13:57:33+00:00	(51319395, 9064150112, 0)	417.831396	0	1.0	0
3	./garmin_activities /activity_12322559591.gpx	POINT (482461.864 4710435.404)	33.000000	2023-09-05 13:57:34+00:00	(51319395, 9064150112, 0)	423.769641	0	1.0	0
4	./garmin_activities /activity_12322559591.gpx	POINT (482464.925 4710430.956)	33.400002	2023-09-05 13:57:35+00:00	(51319395, 9064150112, 0)	429.169124	0	1.0	0
...	...	...	...	...	...	...	...	...	...
14392	./garmin_activities/albin_2.gpx	POINT (482312.549 4710652.407)	69.000000	2021-04-01 14:31:34+00:00	(51319395, 9064150112, 0)	160.359109	44	-1.0	83
14393	./garmin_activities/albin_2.gpx	POINT (482311.203 4710654.365)	70.000000	2021-04-01 14:31:35+00:00	(51319395, 9064150112, 0)	157.983103	44	-1.0	83
14394	./garmin_activities/albin_2.gpx	POINT (482310.766 4710655.000)	70.000000	2021-04-01 14:31:36+00:00	(51319395, 9064150112, 0)	157.212514	44	-1.0	83
14395	./garmin_activities/albin_2.gpx	POINT (482311.316 4710654.200)	70.000000	2021-04-01 14:31:37+00:00	(51319395, 9064150112, 0)	158.183261	44	1.0	84
14396	./garmin_activities/albin_2.gpx	POINT (482312.260 4710652.828)	70.000000	2021-04-01 14:31:38+00:00	(51319395, 9064150112, 0)	159.848369	44	1.0	84

313488 rows × 9 columns

(II) Handling position errors

Gathering all the segments for an given edge, we can see shifts in the x-axis which come from the measurement errors in the latitude/longitude. Given two segments that overlap enough, we can realign them by minimizing the angle (actually maximizing the cosine) between shifted version.

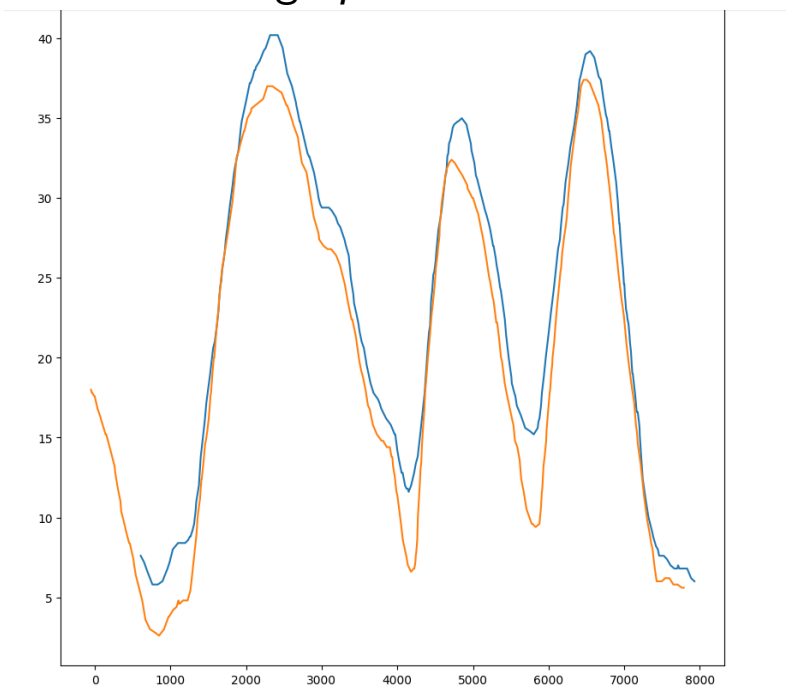


- Let's represent two segments by functions $f : I = [a_1, b_1] \to \mathbb{R}$ and $g : J = [a_2, b_2] \to \mathbb{R}$ defined on some sub-intervals of the x-axis.
- For $c \in \mathbb{R}$ let $f_c : I_c = [a_1 - c, b_1 - c] \to \mathbb{R}$ defined by $f_c(x) = f(x + c)$.
- Let $\langle f1, f2 \rangle_K = \int_K f(x)g(x)dx$ where K is an interval on which both $f1$ and $f2$ are defined and $\|f\|_I = \sqrt{\langle f, f \rangle_I}$.

Finding c that maximize the cosine between f_c and g on the intersection interval on which they are defined, i.e.

$$\frac{\langle f_c, g \rangle_{I_c \cap J}}{\|f_c\|_{I_c \cap J} \|g\|_{I_c \cap J}}$$

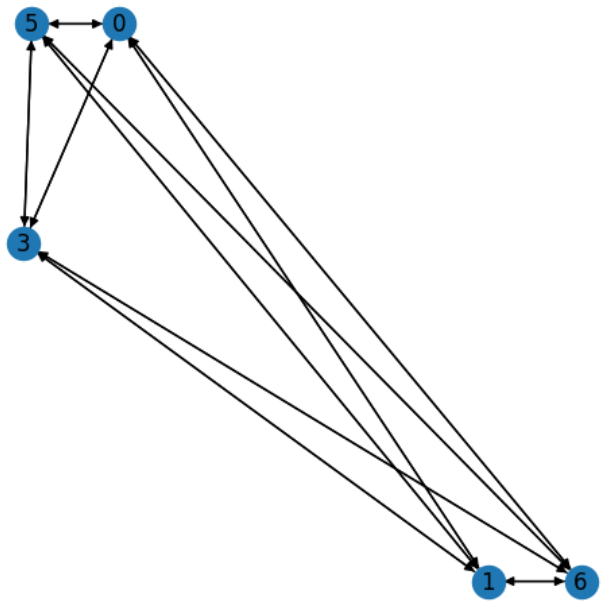
allows us realign *pairs* of curves.



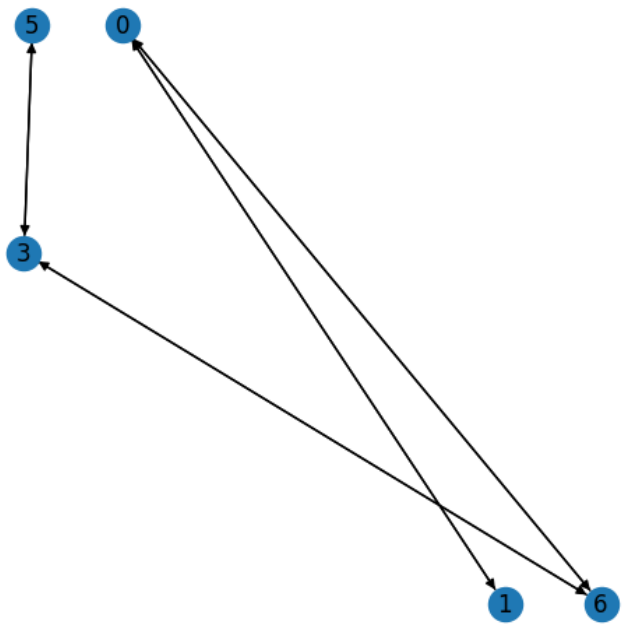
How to deal with more than two curves ? Let X_i be the x-axis error of segment i, then $X_{ij} = X_j - X_i$ corresponds to the shift between segment i and j.

As we seen above, we can compute the relative shift X_{ij} but how can we estimate the absolute shift X_i ? Also, how to deal with situations when $X_{ik} \neq X_{ij} + X_{ik}$?

Representing segments as nodes of a oriented graph and adding a edge between segments for which we managed to compute a shift (i.e. with a wide overlapping and a cosine high enough) we get a graph G that we call the *shift graph*.



We're looking for a coherent system of pairwise shifts so that $X_{ik} = X_{ij} + X_{jk}$.
In the cohomological framework, we're looking for a closed 1-form, i.e. a function defined on the edges that satisfies the above equality and that is also close to our shift form. If the underlying unoriented graph was a tree, there would be no problem since there is no such kind of equality which correspond to the existence of a cycle.
How should we delete edges to get the right tree ? We can measure the confidence we have in a shift computation by looking at the value of the cosinus we maximized.
Taking the maximum spanning tree solves the problem and lets us compute any relative shift in a unique and coherent way by adding the shift along the edges of the path between two nodes.

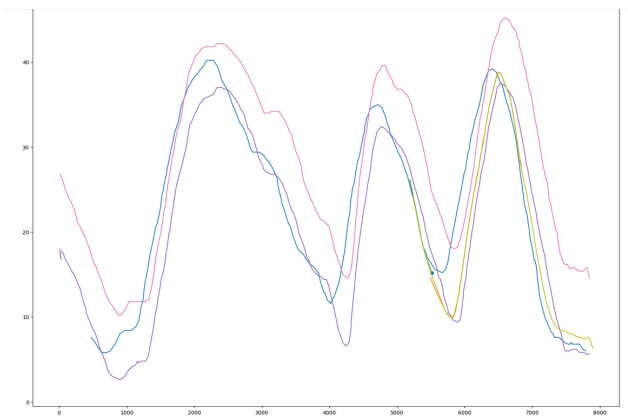


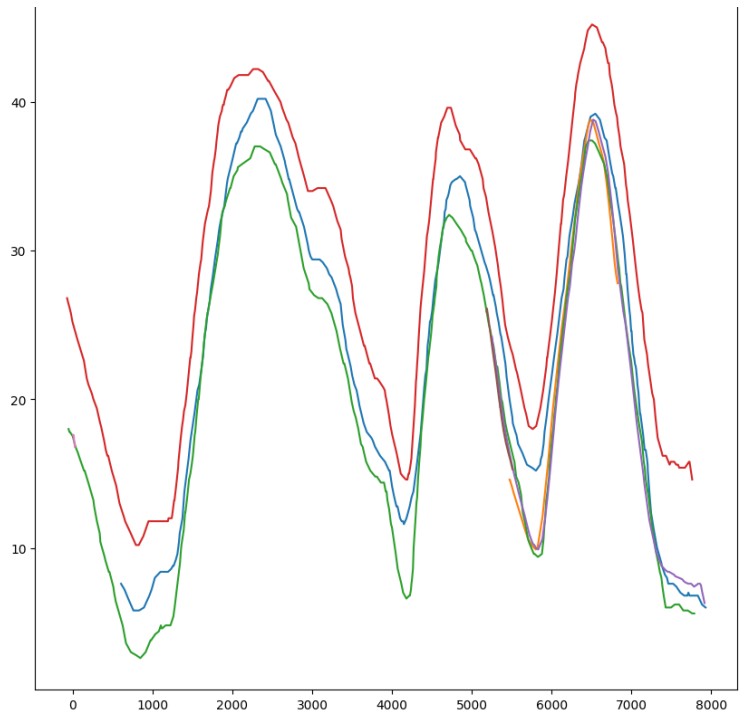
How can we compute the absolute shifts from the relative ones? With enough data, making the (maybe wild) assumption that $\mathbb{E}_i(X_i) = \frac{1}{n}\sum_i X_i = 0$ i.e. that the mean error in the x-axis is zero yields

$$\mathbb{E}_i(X_{ij}) = \frac{1}{n}\sum_i X_{ij} = X_j - \frac{1}{n}\sum_i X_i = X_j$$

.

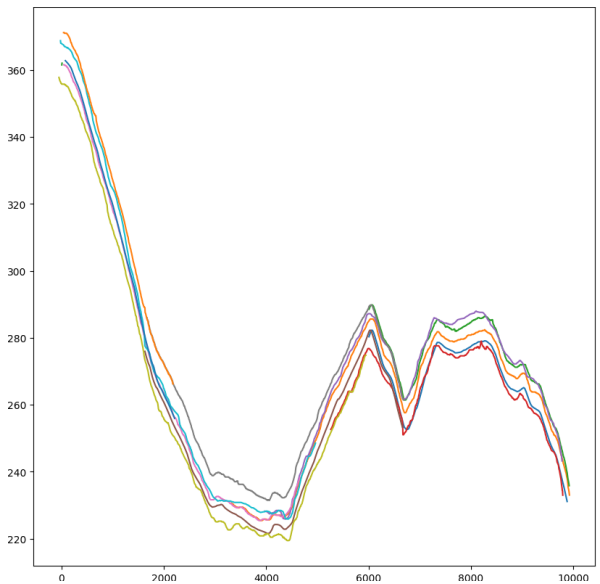
We can estimate the absolute shift of a segment by averaging the relative shifts from all the other nodes.

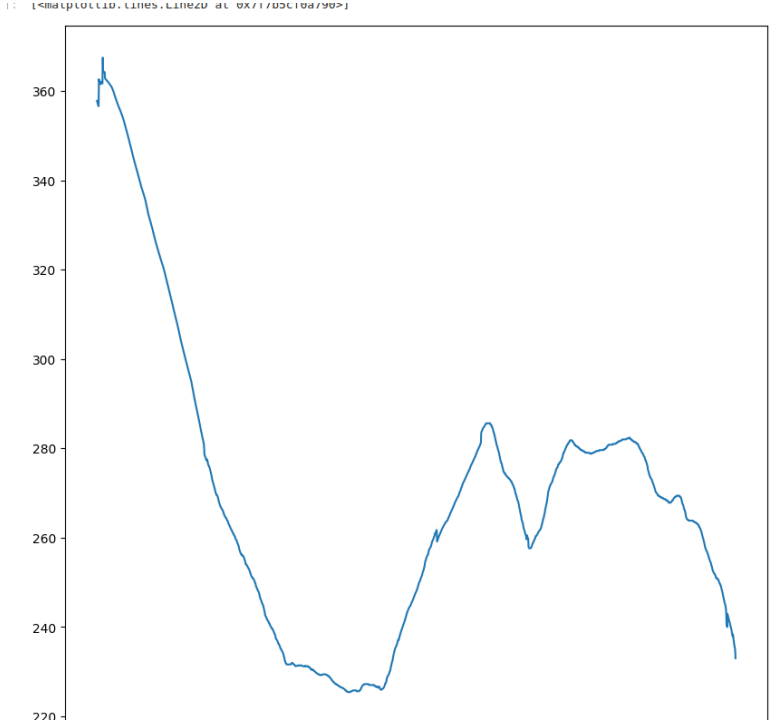




(III) Handling elevation errors

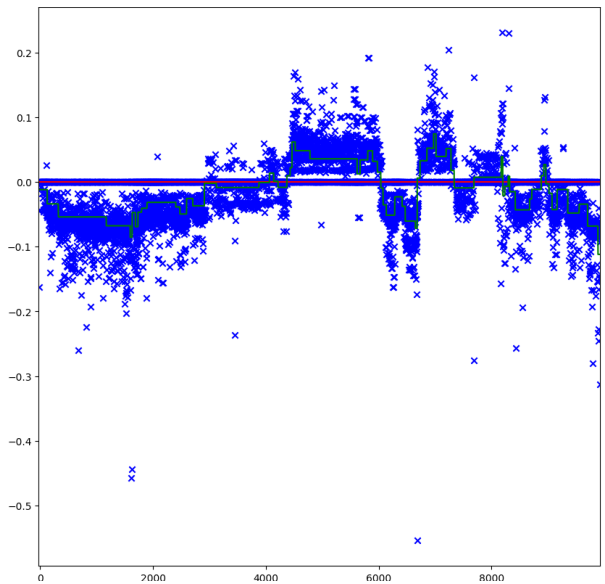
Now that we solved the error in the x-axis, we can try to average the aligned curves to produce a nice final elevation profile.
A first approach could consist of taking the pointwise median of all the curves.

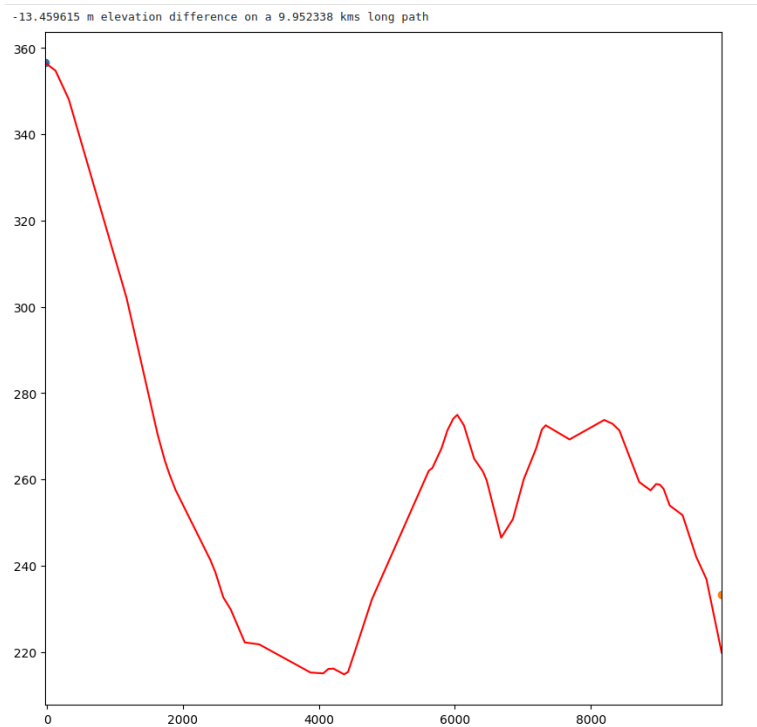




The resulting curve might be close to the real elevation point-wise but is not a good model as it is really irregular : it is likely to learn the noise of the different curves and presents other irregularities (especially when the number of overlapping curves change) that do not correspond to a physical reality. Mathematically speaking, it might be C_0-close but not C_1-close to the true elevation profile which is likely to be simple, i.e. with a low C_2 derivative.

Another idea might be to interpolate this complicated noisy curves by a simple, piecewise linear one. Approximating a curve by a piecewise linear one equivalent to approximate its derivative by a piecewise constant one.
Taking all the derivatives of the different segments, ordering them by their x coordinates and feeding them to a decision tree regressor allows us to approximate the derivative of the elevation profile.

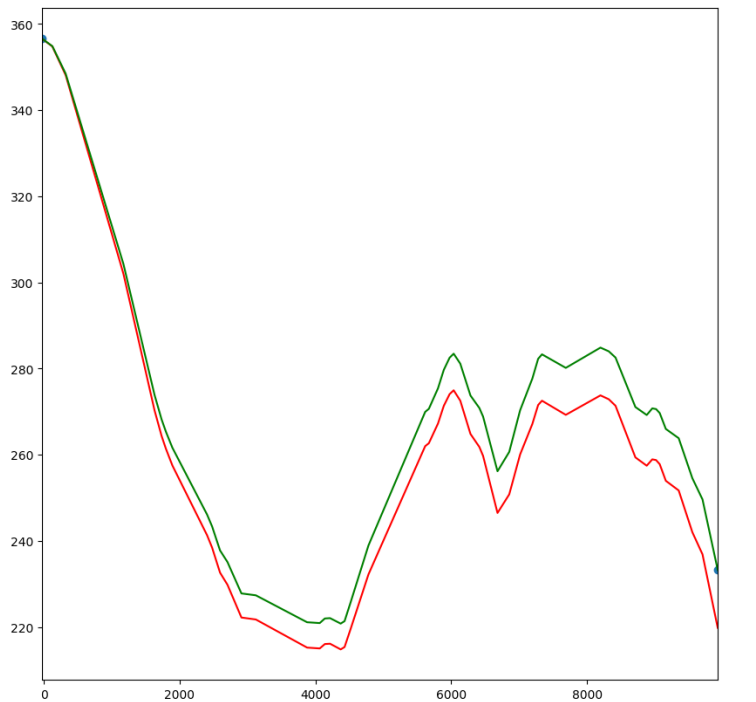




Integrating the piecewise constant predicted by the decision tree, starting at the median elevation at the beginning yields a decent elevation profile which has the advantage to be simple but also taking into accounts the subtle roads variations that appear in several segments.

We can assess the validity of our model by looking at the final elevation at the end and comparing it with the median : -13.46 m elevation difference on a 9.96 kms long path which is okay but not perfect.
We can modify the curve f for it to have the right elevation at the end point in the following way : decompose the derivative f' in positive/negative parts :
$f' = f^+ - f^- = max(f', 0) - max(-f', 0)$ then
$f(b) - f(a) = \int f^+ - \int f^- = d^+ - d^-$.
Let d be the elevation difference observed by taking the median at the extremities, our estimation error can be measured by $|d^+ - d^- - d|$. We want to perturb f^+ and f^- equally so that the integrated curve will have the correct elevation difference, i.e. we look for α solving $(1 + \alpha)d^+ - (1 - \alpha)d^- = d$.

Solving this equation yields $\alpha = \frac{d-(d^+-d^-)}{d^++d-}$ and integrating $(1 + \alpha)f^+ - (1 - \alpha)f^-$ gives a similar curve that matches the right extremity.

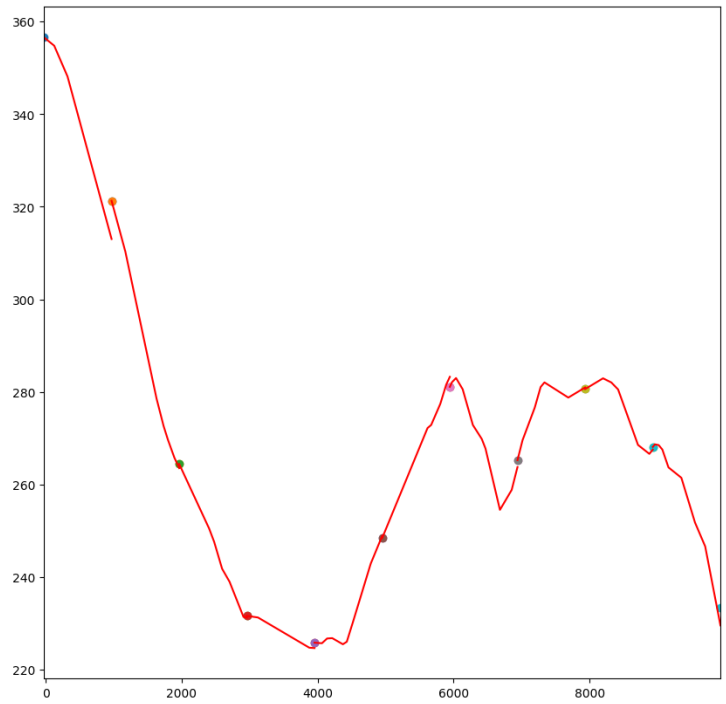


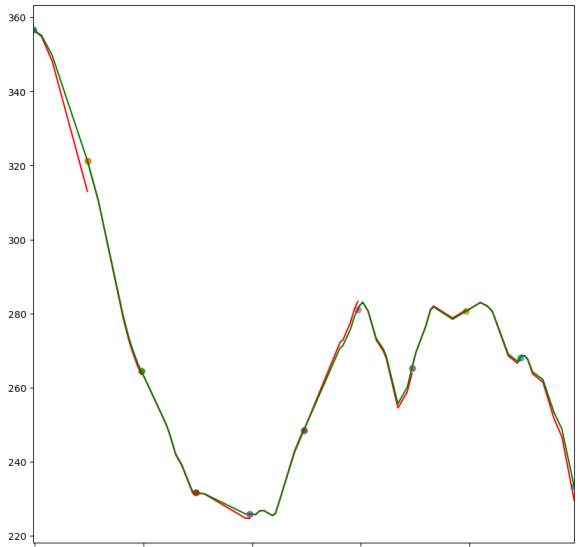
This method might be okay for small sections but the small derivative estimation errors may sum up on longer ones and the estimated elevation profile might be not so close to the reality (especially in the middle).
We can combine the two approaches :

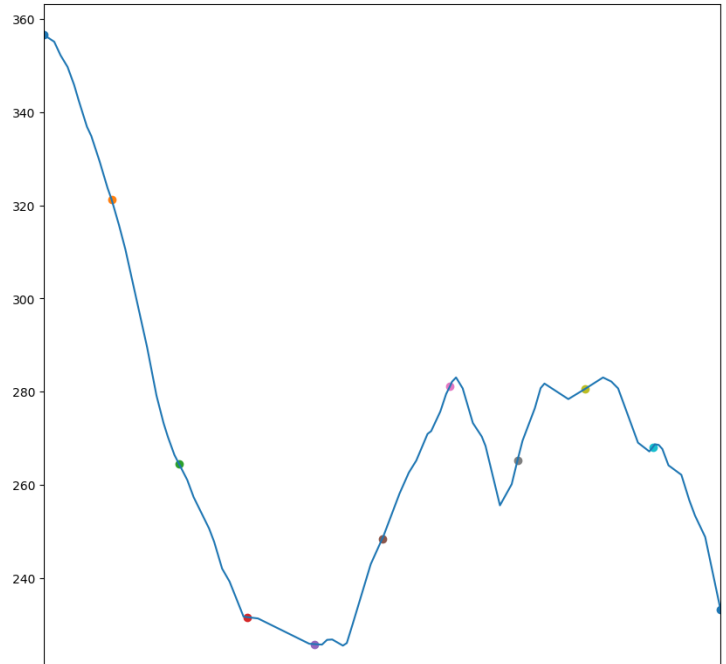
- compute the median elevations at some points, evenly distributed and not too far from each other
- integrate the estimated derivative starting at the median elevation of each point
- perturbate the derivative slightly for the integrated curve to match the median elevation of the next point

The final result is an estimation of the elevation profile that is close to the real one point wise and also in the derivative.


-13.459615 m elevation difference on a 9.952338 kms long path
360
340
320
300
280
260
240
220
0
2000
4000
6000
8000


360
340
320
300
280
260
240
220
0
2000
4000
6000
8000


360
340
320
300
280
260
240
220


360
340
320
300
280
260
240

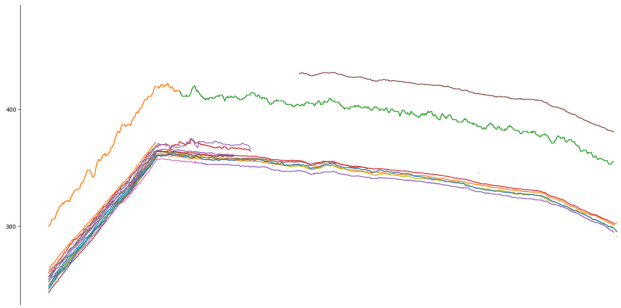
Graph decomposition

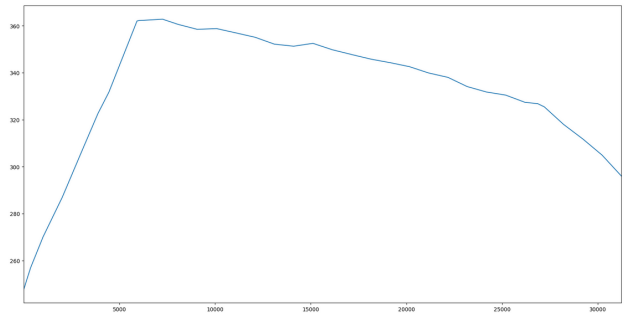
The shifts of the segments displayed in part (II) were easy to compute because the curves had a somehow singular shape. Mathematically, the cosine function is concave in c and we can observe a global unique maximum corresponding to the shift.

The situation is different when the curves are close to affine ones : we can not distinguish a shift in the x and y axis at the same time and this case is likely to happen on small edges with few points per activity.

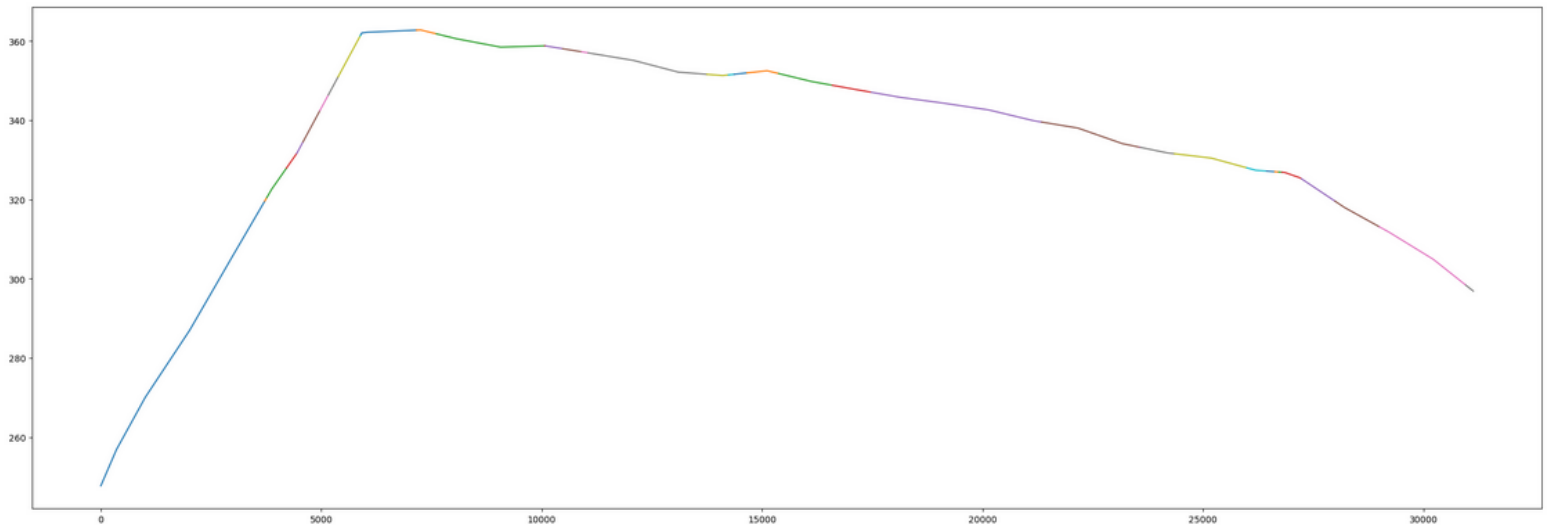
The graph above is the subraph induced by the edges corresponding to the D71, known as "route de Belgodère", a stunning road that crosses several beautiful villages.

Attempting to compute an elevation profile edge-wise will be tricky as the elevation profile is quasi-affine, but projecting the navigation data along the whole road generate a quite singular curve for which it is possible to operate steps (II) and (III).

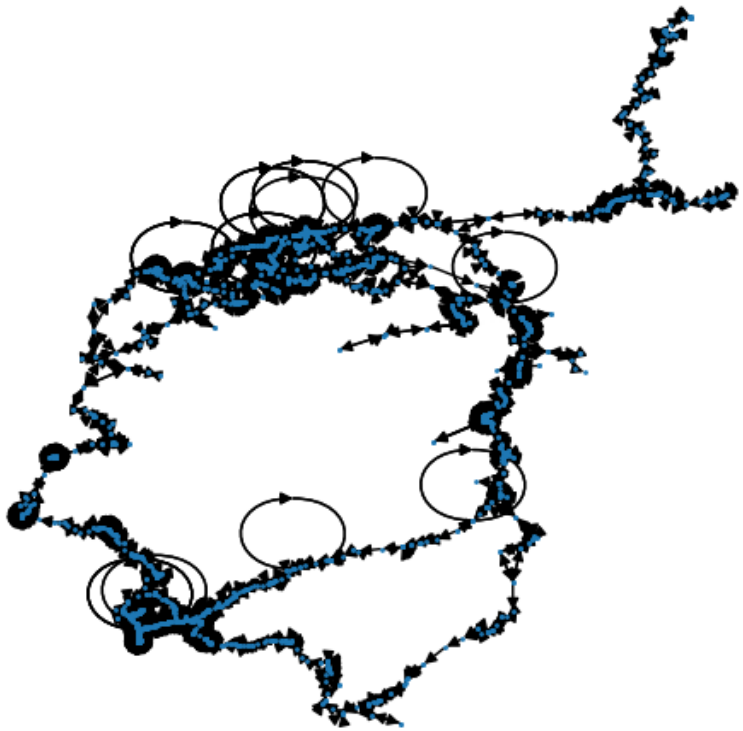




Projecting the resulting curve between consecutive nodes give an elevation profile for each edge.



How should we choose such a path automatically ? How can we organize the edges in path such that, projecting the data on it will produce a lot of long overlapping curves ? The answer might be found looking at the dual graph : the dual graph nodes are the edges of the old graph and and two edges are connected if they have a node in common in the old graph.

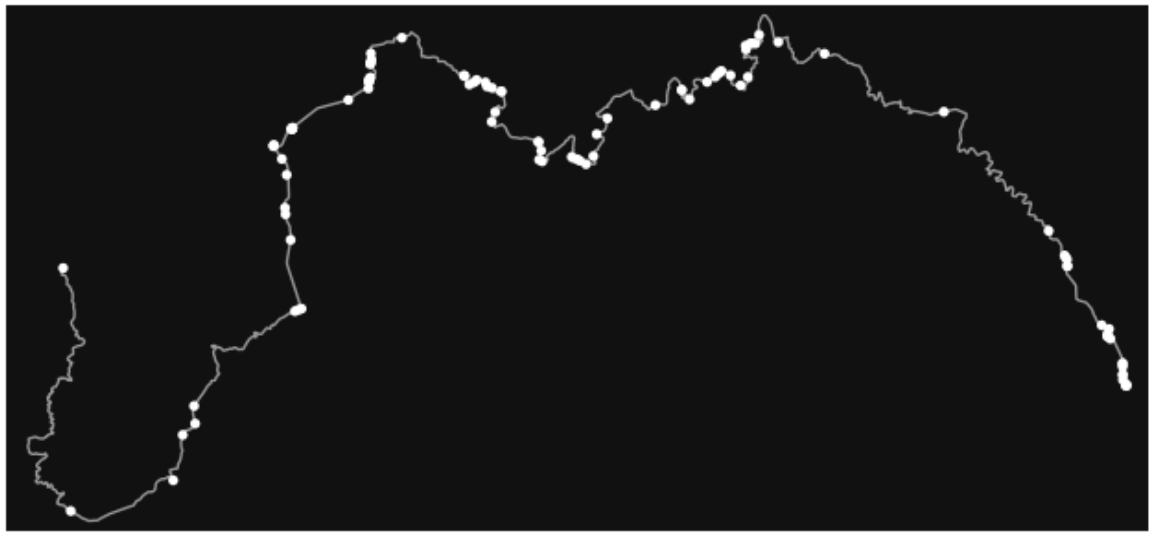


A path in the dual graph yields a path in the old graph and reciprocally so why looking at the dual ?

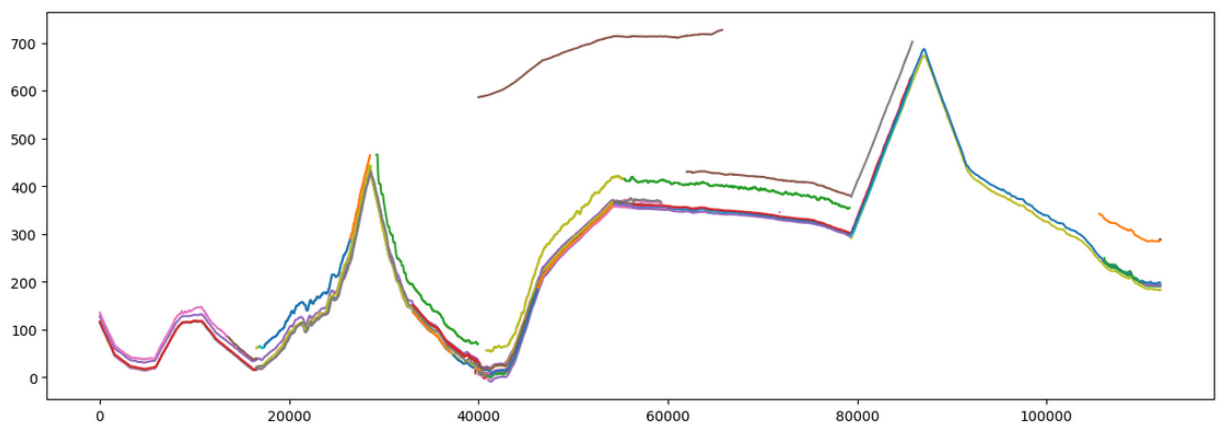
- The graph algorithm that finds the longest path in a graph only works for acyclic ones. It is better to delete cycles in the dual graph because we won't loose dual nodes (i.e. edges).
- We can associate to each edge in a dual graph a sscore that will help the algorithm to find path that are both long and that will produce long segments.

Let $(u, v), (v, w)$ be an edge in the dual graph, i.e. two edges in the old graph sharing a node v. If, in a lot activities, one moved from u to v to w then it is likely that segments from (u, v) may be merged with their successors in (v, w) producing "meta-segments" on the path (u, v, w).
Counting the number of segments likely to be merged for each dual edges gives us edge weights in the dual graph. As in the case of the shift graph, we can find a maximum spanning tree and find a long path in it, long in terms of number of edges but also long in the sense that the sum of weights will be high.

Below is the longest path computed from my navigation data. This path was never actually cycled but it combines the different roads I've been riding several times.



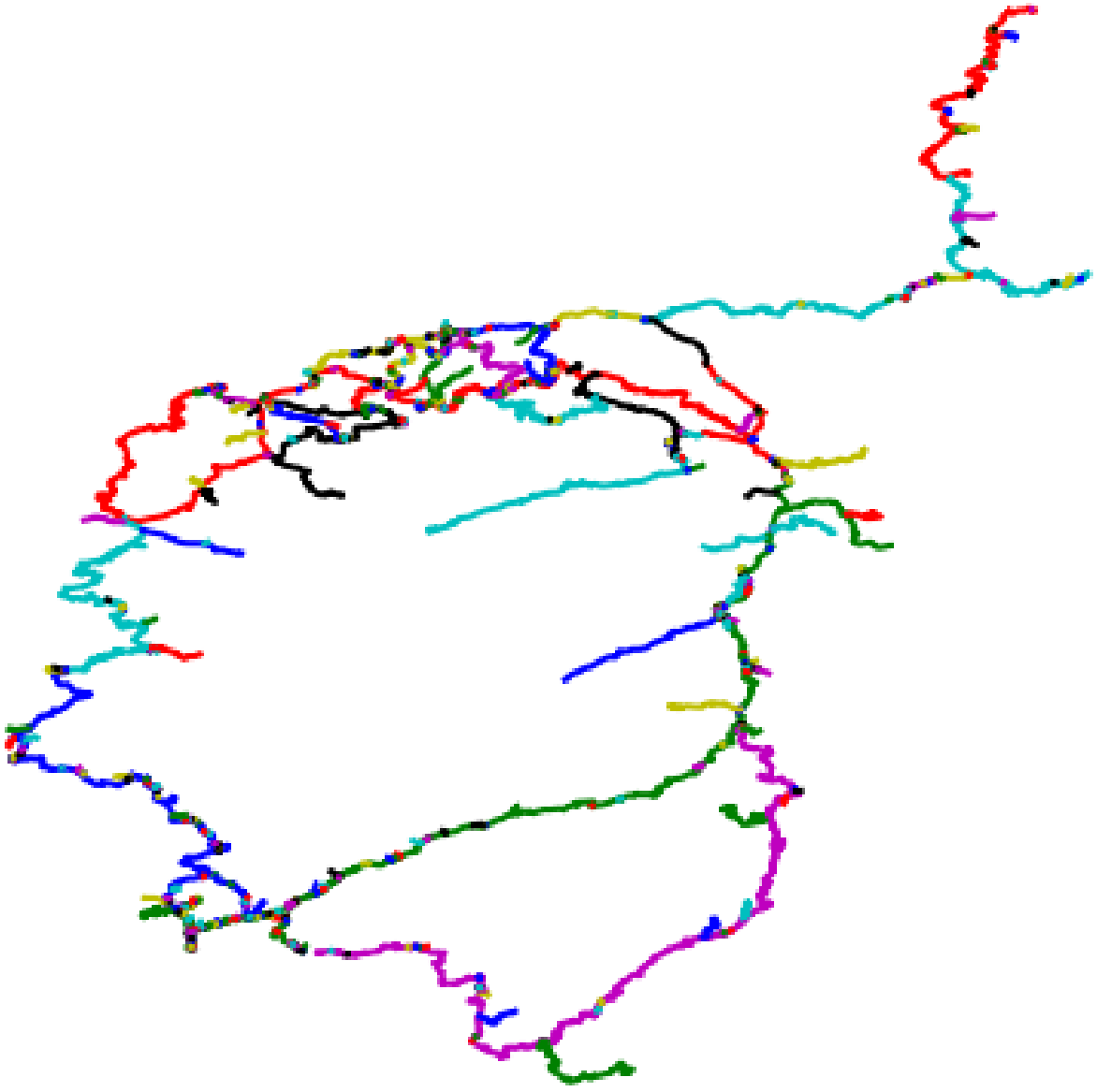
```
: (<Figure size 800x800 with 1 Axes>, <Axes: >)
```



It follows the western coast from Calvi to Galéria, goes up to the Marsulinu pass, climbs down to the Clos Landry (a really old and splendid vineyard, you should try the rosé gris if you have the chance) goes up to Lumio, follows the road to Belgodère, climbs up the Colombanu pass and finishes in Ponte Leccia, a little town north to Corte the historic capital of the island.

We can find a graph edges partition in the following manner :

- (i) find the longest path in dual graph
- (ii) delete all edges appearing in the path
- (iii) repeat steps (i) and (ii) until all edges have been visited.



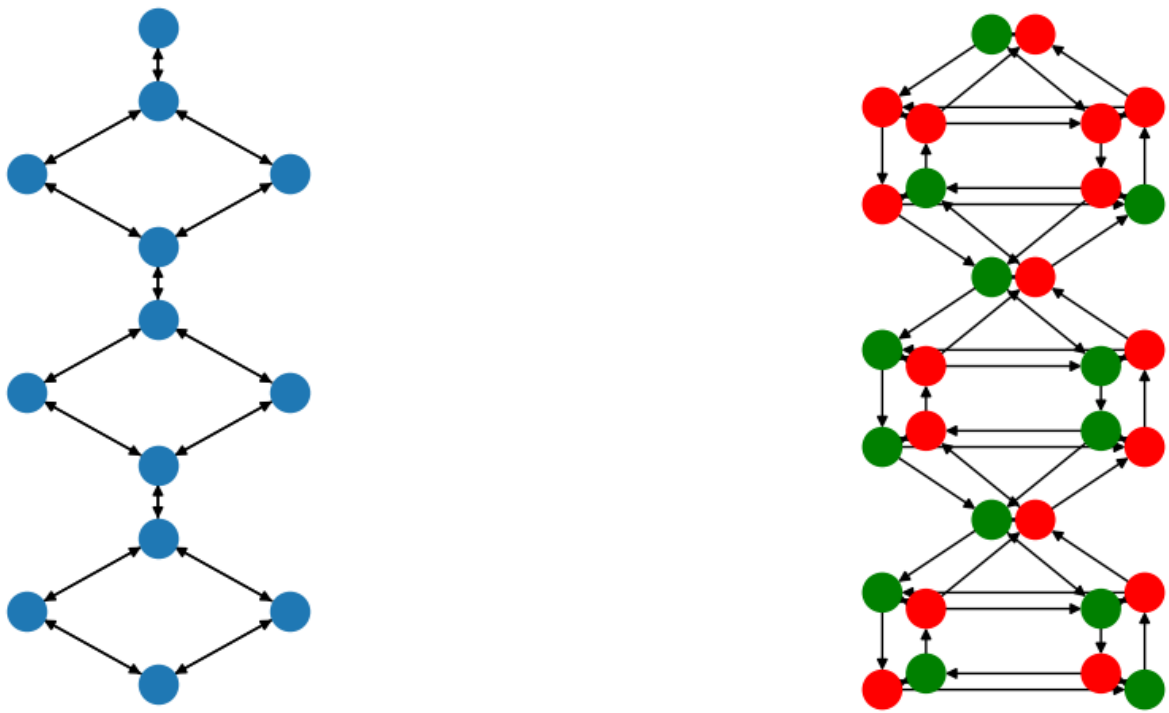
This method might not give a real partition per say but before explaining why let's talk about orientation. This matter has been omitted for clarity but needs to be exposed here.
It is a tricky question because, in the end we want to associate an elevation profile to unoriented edges, but when looking at which segments are likely to be merged we must keep track of the orientation.

- (i) Project the navigation data on the unoriented osm graph.
- (ii) Initiate an empty oriented graph G.
- (iii) For each segment appearing in the dataframe, let (u,v) be the associated edge. Add nodes u and v to G and an oriented edge from u to v if the segment has increasing x coordinates, i.e. if the edge has been crossed from u to v. Otherwise, add a node from v to u and flip the x coordinates of the segments so that all segments have a increasing x coordinates.
- (iv) Generate the dual graph G^*, two edges are connected if they form an oriented path of length two, for such edges $(u, v), (v, w)$, count how many segments in (v, w) are likely to be merged with ones from (u, v).
- (v) Even before deleting edges to get a tree, there are some cycles we want to get rid off : the ones like $(u, v), (v, u)$. When generating a path G^*, we don't want the corresponding path in G to go backwards.
- (vi) Find the maximum spanning tree T in G^*, maximum for the score obtained by counting adjacent segments.
- (vii) Find the longest path in tree.
- (viii) Join oriented segments along the path to form meta-segments.
- (ix) Do the same for the reversed path, flip the meta segments and add them to the previous ones.
- (x) Estimate the elevation profile for the path and project it on the edges.
- (xi) Delete all the edges (u, v) appearing in the path from the tree as well as their opposites (v, u).
- (xii) Repeat steps (viii)-(xi)

The different paths obtained might not form a proper partition of the unoriented edges for the following reasons :

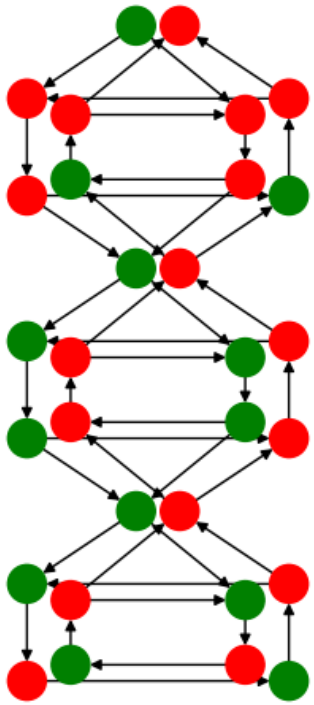
- When computing the maximum spanning tree, we discard isolated nodes of the G^* which corresponds to edges in G whose segments won't be merged with any other one. If the elevation profile of such an edge is singular enough, we can treat it independently like in the first naive approach, otherwise we can not do much.
- We chose our algorithm such that an unoriented edge can not appear in two distinct paths. Intuitively, since a path in G^* can not go backwards, it is unlikely that an unoriented edge appears twice in a path with both orientations but it can still happen.

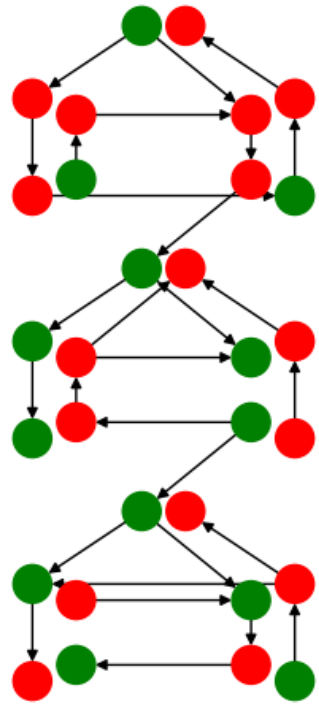
Let's look at the following graph and see what steps (iv)-(vii) do.



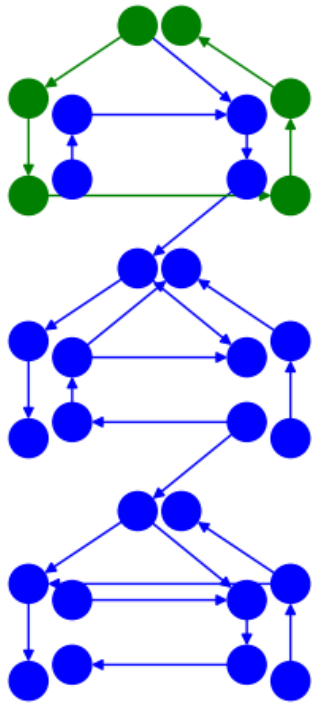
On the left is a hypothetical graph that represents some navigation data, on the right the corresponding dual graph (colors help us distinguish opposite edges).

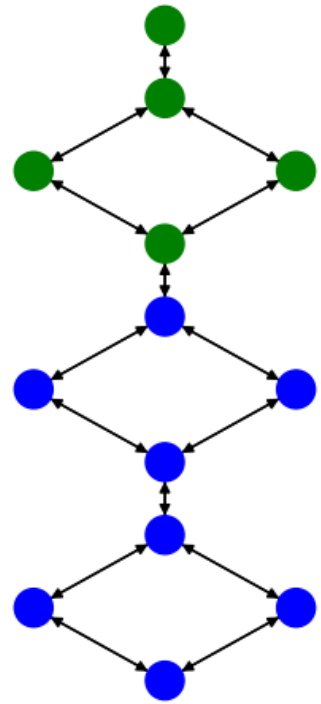
Let's remove the dual edges between two opposite edges (on the left) and look at some random spanning tree (on the right).





The longest path in the dual graph is colored in green and the corresponding path in the old graph is also displayed on the right : starting at the top, it goes down, goes along the first cycle anticlockewisely and goes up again. Even though we deleted the dual edges between the two opposite topmost edges, they still appear at the begin and at the end of the path in the dual graph.





For our use, it is not that much of a problem, the two main inconveniences is that we compute the same elevation profile two times and that the longest path might not be optimal. A custom longest path algorithm that takes into account the fact that we're looking at a dual graph and that we don't want an unoriented edge appearing twice will solve this problem.