GENERAL RULES

- Groups must work on their own without any help from any other group. Groups that present suspiciously similar code will not only fail the project, but also fail the subject.
- The last version of the code will be uploaded to eGela at least once a week. Bear in mind that big changes in little time may look suspicious from the lecturer?s point of view. To avoid problems, upload your code as often as possible so that we can see the evolution of your work (see *Upload your code*).

PROJECT GOAL

A company develops different GPS models, which record the tracks with different precisions (recording frequency). Table 1 summarises the main features of the GPS models.

GPS Model	Screen	Networks	Accuracy	En. consume (e_g)	Price
Heimdl	Color	GPS + GLONASS	Highest	0.008 mAh/s	550€
Sleipnir	B\W	GPS + GLONASS	High	0.006 mAh/s	400€
Yggdrasil	B\W	GPS	Low	0.005 mAh/s	300€
Loki	B\W	GPS	Lowest	0.004 mAh/s	200€

Table 1: GPS models and their main features

Taking into account that the GPS models have different hardware and hence different energy consume, the company is testing the GPS models with different batteries. Table 2 shows the battery models used along with their capacities.

Battery Model	Capacity	Price
Asgard	4000 mAh	150€
Midgard	3000 mAh	100€
Helheim	2000 mAh	50€

Table 2: Battery models and their capacity

In this project, you will analyse the performance of the GPS regarding the accuracy of the information, autonomy, ...

GROUP GRADES

Each group will be given a group grade that will be used as the baseline to calculate the individual grade of each student (it could be greater or less than the group grade). The maximum group grade will depend on the number of tasks finished by the group:

- 100%: all the tasks
- 80%: tasks 1, 2, 3, 4 and 6

PROGRAMMING GUIDELINES

- Do not print or output any information (or plots) from inside a function.
- Print the results in the exact same format we use in this document.
- When possible, reuse code by using functions. In most of the tasks, we will tell you how to design the functions.
- Use loops (*for*/*while*) to iterate over sets of values instead of replicating existing code.
- Use meaningful names for variables/scripts/functions so that anyone can understand the code without spending too much time reading it.
- Write comments to explain the goals of each part of the programs (start the line with '%').

INPUT FILES

In this project, the files included in the *gps-track-files.zip* file will be processed. You should unzip this file and place the extracted files in your project folder.

All the files are named following the "*track-XX-GPS.csv*" pattern, where XX is a two-digit number which represents the track and GPS is the name of the GPS used to record the track. Each track was recorded with the three GPS types. For every recorded position, the track has one row in the file. In each row, the first column represents the longitude, the second the latitude, the third the altitude, and the fourth the time (in seconds). The example below shows a fragment of a track recorded[1].

[1]For the sake of readability, some decimal digits have been replaced by '...'

Example: *track-06-Heimdl.csv*

```
42.3232...;-3.0118...;869.2000...;33181
42.3232...;-3.0118...;869.400...;33182
42.3232...;-3.0118...;869.5999...;33183
42.3232...;-3.0118...;868.7999...;33185
42.3231...;-3.0119...;869;33187
...
```

TASKS

Task 1: Examine the tracks recorded using the GPS trackers

In this task, you will analyse the tracks recorded by the GPS trackers. To this end, you have to write a script called *examineTracks* that analyses both the route and the elevation profiles of each track. For each track, a figure with two subplots will be generated. The left plot will display the routes recorded by each GPS whereas the right plot will show the altitude profiles recorded by each GPS (see Figure 1). Although they might not be remarkable, there will be slight differences in the routes and altitude profiles.

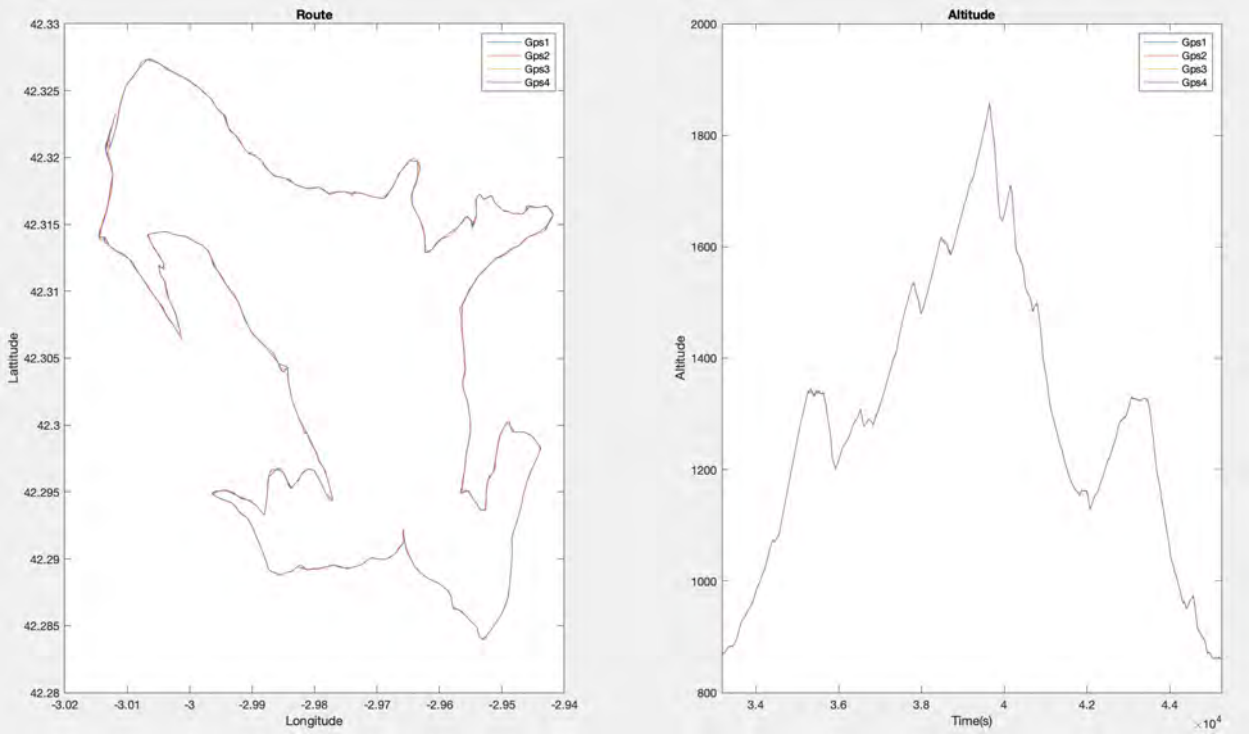

Figure 1: Example of route and altitude profiles recorded by different GPS trackers

The script will save the figure of each track in a file called "*track-XX.png*", where XX is a two-digit number which represents the track. The file will be saved in the "*report*" folder.

Task 2: Create a report of the recorded tracks

In order to compare the performance of the different GPS models, the statistics of the tracks should be displayed. In this task, you will developed the functionality required to this end. In particular, you have to

1. Implement a function called *distance* which, given the vectors that represent the longitudes and latitudes of the points of origin and destination, returns a vector which contains the distance between each two points calculated in meters.

   ```
   function dist = distance(longOrig, latOrig, longDest, latDest)
   ```

   You can check the correctness by running the function as in the following example, and comparing the result obtained:

   ```
   distance([-3.011 -3.012 -3.13], [42.323 42.324 42.325],
            [-3.012 -3.13 -3.014],[42.324 42.325 42.326])
   [138.286698155807 9701.56094445445 9536.99792232943]
   ```

   To calculate the distance you will apply the Pythagorean method, which calculates the distance between two points in a sphere in radians based on their coordinates.

   $$x = \Delta\lambda * cos\left(\frac{\varphi_1 + \varphi_2}{2}\right)$$
   $$y = \Delta\varphi$$

   where:

   - φ_1 and φ_2 are the origin and destination latitudes **in radians**
   - $\Delta\lambda$ is the distance between destination and origin longitudes **in radians**
   - x is the difference between origin and destination longitudes **in radians**
   - y is the difference between origin and destination latitudes **in radians**

   Then, the result must be multiplied by the mean radius of the Earth (R = 6371 km) to calculate the distance between the two points in meters using the following formula.

   $$distance = \sqrt{x^2 + y^2} * R$$

2. Write a function call *computeTrackLength*, which given the vectors containing the latitudes and longitudes of the trackpoints, returns the total distance travelled in meters.

```
function totalDistance = computeTrackLength(longitudes,
    latitudes)
```

You can check the correctness by running the function as in the following example, and comparing the result obtained:

```
dist = trackLength([42.323 42.324 42.325 42.326],
                   [-3.011 -3.012 -3.13 -3.014]);
fprintf('%.4f', dist);
26177.7054
```

3. Write a function called *computeCumulativeAscentDescent*, which given a vector that contains the altitudes, returns both the cumulated ascent and descent.

```
function [asc,des] = computeCumulativeAscentDescent(altitudes)
```

You can check the correctness by running the function as in the following example, and comparing the result obtained:

```
[asc, desc] = computeCumulativeAscentDescent(
                                        [869.2 869.4  869.1 868.9 900])
asc  =        31.3000
desc =  0.5000
```

4. Write a script called *gpsTrackRecordingReport.m* which, for each track, prints a report that displays information extracted by each GPS in the following format:

```
Track 1
=======
Model: Heimdl
        Max. Altitude: 2565.00 Min Altitude: 1879.02
        Cum. Ascent: 785.33 Cum. Descent: 705.32
        Distance: 10.05 km Avg. Speed: 2.05 km/h
Model: Sleipnir
        Max. Altitude: 2565.00 Min Altitude: 1881.02
        Cum. Ascent: 714.93 Cum. Descent: 634.92
        Distance: 7.51 km Avg. Speed: 1.53 km/h
Model: Yggdrasil
        Max. Altitude: 2565.00 Min Altitude: 1881.02
        Cum. Ascent: 703.01 Cum. Descent: 623.00
        Distance: 7.30 km Avg. Speed: 1.49 km/h
```

```
Model: Loki
        Max. Altitude: 2563.98 Min Altitude: 1884.00
        Cum. Ascent: 689.92 Cum. Descent: 609.91
        Distance: 7.08 km Avg. Speed: 1.44 km/h
...
```

Task 3: Energy consumption report

All the GPS models have been tested with the highest-capacity battery, but the company aims at analysing the energy consumption of the different models. In this task, you must:

1. Write a function called *computeEnergyConsumption* which, given two vectors (which represent the recorded altitudes and the trackpoing recording times respectively) and the energy consumed by the GPS in ideal conditions, calculates the battery use accordingly.

   ```
   function consume = computeEnergyConsumption(altitudes,
           times, eg)
   ```

   The company has observed that the energy a GPS consumes every second can be estimated with the following formula

   $$e_t = e_g * \left(1 + \frac{|400 - h_t|}{100}\right)$$

   where e_t is the energy consumed by the GPS at instant t, e_g is the energy the GPS consumes when running in ideal conditions (see Table 1), and h_t represents the height at time t. It has been observed that the height affects the reception of the GPS signal. Recording the trackproints has an additional energy consume (0.0005 mAh for each tracked point). For a more accurate estimate, use interpolation with $\Delta t = 0.1$.

   You can check the correctness by running the function as in the following examples, and comparing the results obtained:

   ```
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0005)
   0.2620
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0008)
   0.4180
   computeEnergyConsumption([400 410 380 400],[0 120 240 480], 0.0006)
   0.3140
   ```

2. Write a function called *getAllGPSEnergyConsumptions* that, given the number of tracks, the vector that contains the names of the GPS models and the vector that contains the energy consumes (e_g) of the GPS models, processes all the track files and returns the energy consumes of the GPS trackers.

```
function consumeMatrix = getAllGPSEnergyConsumptions(
        numTracks, gpsNames, egs)
```

This function returns a matrix where each row represents the energy consumed to record the corresponding track and each column represents the energy consumed by a GPS to record the tracks (see Table 3). e_{ij} is the energy consumed by the GPS tracker j when it recorded track i.

	Heimdl	*Sleipnir*	*Yggdrasil*	*Loki*
$track_1$	e_{11}	e_{12}	e_{13}	e_{14}
$track_2$	e_{21}	e_{22}	e_{23}	e_{24}
$track_3$	e_{31}	e_{32}	e_{33}	e_{34}
...				
$track_m$	e_{m1}	e_{m2}	e_{m3}	e_{m4}

Table 3: Structure of the matrix that contains the energy consumes of the GPS trackers

3. Write a script called *analyseGPSEnergyConsumption.m* which displays a plot showing the energy consumption per track for each GPS model (see Figure 2).

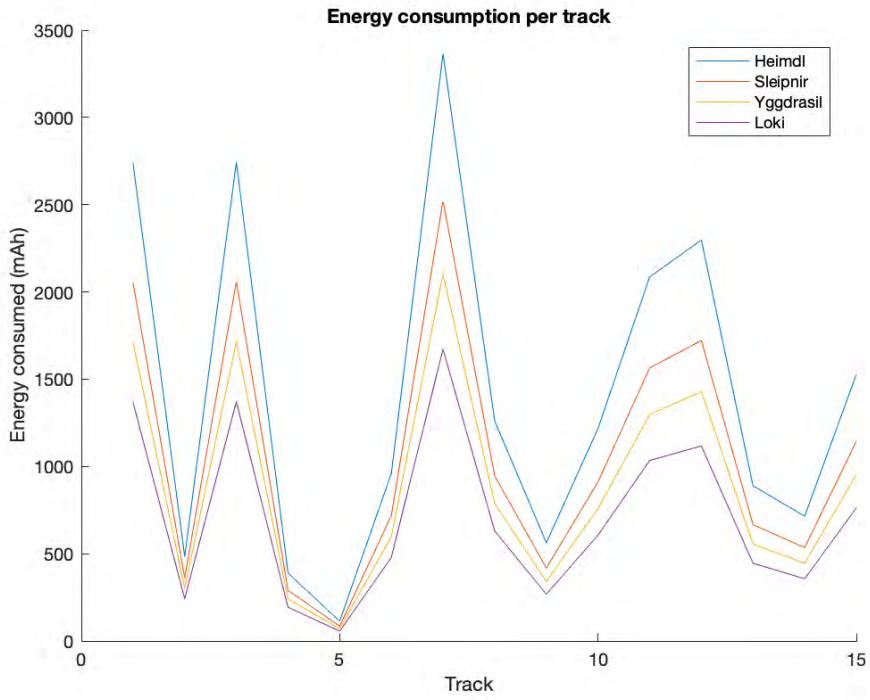

Figure 2: Graph that displays the energy consumed by the GPS models per track

Task 4: Determine the cheapest GPS setups

In order to offer GPS models which are attractive for the user at an affordable price, the company wants to determine which is the cheapest GPS configuration (model + battery). Write a script that determines the cheapest GPS setups. To this end, for each GPS, the cheapest battery (see Table 2) that assures that all the tracks used in the evaluation could be recorded. To this end, you must perform the following subtasks.

1. Write a function that given the vector that contains the energy consumed by the GPS tracker to record the tracks and the vector that contains the capacities of the different battery models, returns the position of the cheapest battery that assures that all the tracks could be recorded, i.e., the capacity is bigger than the energy required to record each track. As can be seen in Table 2, the lower the capacity, the cheaper the battery. Assume that the *batteryCapacities* vector is order from highest to lowest capacity.

   ```
   function batteryId = getCheapestBatteryId(
           gpsEnergyConsumes, batteryCapacities)
   ```

   You can check the correctness by running the function as in the following examples, and comparing the results obtained:

   ```
   getCheapestBatteryId([1600 1500 3200],[4000 3000 2000])
   1
   getCheapestBatteryId([1600 1500 2800],[4000 3000 2000])
   3
   getCheapestBatteryId([1600 1500 1900],[4000 3000 2000])
   3
   ```

2. Write a script called *determineCheapestGPSSetUps.m* that prints a report describing the cheapest GPS configurations. The report must look like:

   ```
   GPS configurations
   ==================
   GPS model: Heimdl Battery model: Asgard Price: 700€
   GPS model: Sleipnir Battery model: Mildgard Price: 500€
   GPS model: Yggdrasil Battery model: Mildgard Price: 400€
   GPS model: Loki Battery model: Helheim Price: 250€
   ```

Task 5: GPS setups report/ranking

The customers of the company may purchase the GPS considering different aspects. Some clients opt for the models with highest accuracy while others may prefer cheaper alternatives. Therefore,

1. Write a function called *computeGPSTrackAccuracy* that, given the length (distance travelled) of the track recorded by a GPS tracker and the "real" track length, computes the accuracy of the track recorded. The accuracy of a track is determined by the distance of the track recorded divided by the real distance.

   ```
   function accuracy = computeGPSTrackAccuracy(recordedDistance,
       realDistance)
   ```

   You can check the correctness by running the function as in the following examples, and comparing the results obtained:

   ```
   computeGPSTrackAccuracy(12.34, 12.34))
   1
   computeGPSTrackAccuracy(11.54, 12.34))
   0.9352
   computeGPSTrackAccuracy(10.43, 12.34))
   0.8452
   ```

2. Write a function called *computeAverageGPSAccuracy* which, given two vectors that represent the recorded track lengths, and the real track lengths, computes the average accuracy.

   ```
   function avgAccuracy = computeAverageGPSAccuracy(
       recordedDistances, realDistances)
   ```

   You can check the correctness by running the function as in the following examples, and comparing the results obtained:

   ```
   computeAverageGPSAccuracy([12.34 35.78 9.34], [12.34 35.78 9.34])
   1
   computeAverageGPSAccuracy([11.45 33.97 9.11], [12.34 35.78 9.34])
   0.9509
   computeAverageGPSAccuracy([10.89 31.34 8.98], [12.34 35.78 9.34])
   0.9066
   ```

3. Write a function called *computeGPSRatingAndAccuracy* which, given the name of the GPS, the vector that contains the real distances of the tracks, and the price of the GPS model, computes both the rating for the GPS and its average accuracy.

   ```
   function [rating, avgAccuracy] = computeGPSRatingAndAccuracy(
       gpsName, realDistances, gpsPrice)
   ```

   To compute the rating, the following formula is used:

   $$rating = 0.8 * avgAccuracy_g + *.3 * \left(\frac{refPrice}{price_g}\right)$$

where $avgAccuragy_g$ is the average accuracy of the GPS model, $refPrice$ is a reference price (625€), and $price_g$ is the price of the GPS (see Table 1).

4. Write a script called *gpsRankings.m* that prints a report with the best options for the customers. The options are ordered by ratings. The example bellow shows how the report must look. To generate this report, the tracks recorded with the *Heimdl* model will be considered the real tracks.

```
GPS best options
================
GPS model: Loki Battery model: Helheim
Accuracy: 0.801 Price: 250? Rating: 1.141
GPS model: Heimdl Battery model: Asgard
Accuracy: 1.000 Price: 700? Rating: 0.979
GPS model: Yggdrasil Battery model: Mildgard
Accuracy: 0.825 Price: 400? Rating: 0.972
GPS model: Sleipnir Battery model: Mildgard
Accuracy: 0.868 Price: 500? Rating: 0.945
```

Taks 6: Main program menu

Write a scrip called *mainMenu.m* that shows the following menu:

```
#######  MENU  ########:
1. Generate the plot figures of the tracks
2. Print track reports
3. Energy consumption report
4. Determine cheapest GPS configurations
5. Get GPS setups report/ranking
0. Exit
Select an option:
```

Depending on the option chosen, the program will run the selected script (to run a script from another script, we just need to write its name without the ?.m? extension). Before running a script, whether or not user has introduced a valid option number must be checked. Whenever the userintroduces a wrong option, the following message must be shown "`The selected option is invalid: please choose 1, 2, 3, 4, 5 or 0`".

The menu script will start by clearing the console and, when an option is selected, it will show the results. Next, the user will be shown the message "`Press a key to continue...`", and when the user presses a key, the menu script will again clear the console and show the options. This will allow the user to read the results before clearing the console. You can use the *pause* command to this end.

The menu will be shown until the user selects 0. Do not use the *exit* command to end the script (it closes Matlab), just exit from the iterative structure you are using to show the menu.

UPLOAD YOUR CODE

1. For reviewing purposes, we ask you to upload your code inside the appropriate task in Egela, **at least once a week**. Do not erase old versions.

2. Only one student per group needs to upload the code.

3. You must add a ZIP file each time:

   (a) The name of the file must have the format: 'project-*yyyy-mm-dd*.zip', where *yyyy* is the year, *mm* the month, and *dd* the day.

   (b) The zip must contain all the script/function files you have done

   (c) The zip must also contain all the input files required so that we can run the scripts using only what is inside the zip file.

   (d) It will also include a file named 'log.txt', where you will add what you have done since the last time you uploaded your code.

   For example:

   log.txt inside *project-2020-11-22.zip*

   ```
   2020-11-20
   still doesn't work

   2020-11-22
   Task 1: We finally made it work, results look reasonable
   Task 2: Just started working on function 'myfunction'
   ```