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 once a week minimum. 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

In this project, we will work for a logistics company, analyzing their data to help them making the right decisions. The company employs two different drivers to transport goods between Donostia-St. Sebastian and Vitoria-Gasteiz. Two different routes can be used to travel from one place to the other: the national road (N1) and the highway (A1). We will develop a set of Matlab functions/scripts to help the company decide which of the two drivers and which of the routes to use.

GROUP GRADES

Each group will be given a group grade that will be used as the baseline to calculate each student's individual grade (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
- 75%: tasks 1, 2, 3 and 4

PROGRAMMING GUIDELINES

- Do not print or output any information (or plots) from inside a function.
- When possible, reuse code by using functions. In most of the tasks, we will tell you how to design the functions.
- When possible, 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.

UPLOAD YOUR CODE

For reviewing purposes, we ask you to upload your code inside the appropriate task in E*gela*, **once a week at least**.

- Only one student per group needs to upload the code.
- You must add a ZIP file each time:
  - The name of the file must have the format: '*project-yyyy-mm-dd.zip*', where *yy* is the year, *mm* the month, and *dd* the day.
  - The zip must contain all the script/function files
  - The zip must also contain all the results obtained so far (plots, text files, screen outputs).
  - 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-2017-11-22.zip*

```
2017-11-20
Task 1: We started working on 'myscript.m', but it
still doesn't work

2017-11-22
Task 1: We finally made it work, results look
reasonable
Task 2: Just started working on function 'myfunction'
```

INPUT FILES

We will work with the following files:

1. Logs from the two drivers on the two different routes:
   - `a1-driver1-log.csv`
   - `a1-driver2-log.csv`
   - `n1-driver1-log.csv`
   - `n1-driver2-log.csv`

   In these files, we have the distance from the origin (*km*) and the vehicle's speed in that moment (*km/h*) for a set of points along the route. The first column are the distances from the origin, and the second are the speeds:

   Example: n1-driver1-log.csv

```
0,0
0.10013,8.5168
0.20027,16.901
0.3004,24.103
0.40053,30.048
0.50066,35.717
...
```

2. The height profile for each of the routes:
   - `a1-height.csv`
   - `n1-height.csv`

   In these files we have the route information:

Example: n1-height.csv

```
Latitude,Longitude,Elevation(metres),Distance (KMs),Gradient
43.31839,-1.98127,5.7,0,
43.31821,-1.98169,6.2,0.039,
43.31876,-1.98207,6,0.108,
43.31887,-1.98214,6,0.121,
43.31936,-1.98249,6.1,0.183,
43.31939,-1.98252,6.1,0.187,
43.31945,-1.98256,6.1,0.195,0
43.31917,-1.9833,6.3,0.262,0
...
```

We will need to skip the first line in the file with the column header.

3. The speed limits in each of the routes:
   - `a1-speed-limit.csv`
   - `n1-speed-limit.csv`

Example: a1-speed-limit.csv

```
0;90
5;120
95;90
```

In this files, we have information about the speed limits. In the first column, we have at which distance from the origin starts a speed limit (in *km*), and, in the second column, the maximum speed allowed from that point on (in *km/h*). In the example above, drivers can drive up to $90\ km/h$ the first 5 kilometers, then up to $120\ km/h$ from kilometer 5 to 95, and up to $90\ km/h$ from kilometer 95 on.

TASKS

Task 1:

Write a script that analyzes the elevation profiles of both routes:

1. It will generate a plot with two subplots:
   - the heights (m) of both routes against the distance from the origin (km).
   - the routes (longitudes against latitudes).

   Each of the subplots will be identified by a title, and both will have the x/y axes properly labeled. For example, a proper label for an axis in which we are displaying time would be: *time (s)*. This plot will be saved as *'route-elevations.png'* in the same folder the scripts are.

2. The script will also calculate and display in the console window the following statistics: average, standard deviation, minimum and maximum elevation. The output should be:

```
n1 route statistics:
Mean height: 318.37 (sd: 235.54)
Height range: [4.20, 656.10]

a1 route statistics:
Mean height: 189.71 (sd: 181.70)
Height range: [0.00, 825.80]
```

Task 2:

Write a script that analyzes the logs for each driver on each route:

1. The script will generate a plot with the speeds (km/h) against the distance from the origin (km). A different subplot will be used for each route, each with the speeds of both drivers. Each subplot will be properly identified by a title, and each subplot will have a legend to distinguish both drivers. Labels will be properly labeled.
2. It will also calculate and plot the average, standard deviation, minimum and maximum speed for each driver/route. The output should be:

```
driver1 statistics in route n1
Mean speed: 95.99 km/h (sd. 10.14)
Min-Max speed: [0.00,118.44]

driver2 statistics in route n1
Mean speed: 98.76 km/h (sd. 10.78)
Min-Max speed: [0.00,121.43]

driver1 statistics in route a1
Mean speed: 101.29 km/h (sd. 14.82)
Min-Max speed: [0.00,119.00]

driver2 statistics in route a1
Mean speed: 104.79 km/h (sd. 15.62)
Min-Max speed: [0.00,122.17]
```

Task 3:

Write mainMenu.m script, which shows this menu to the user:

```
#######  MENU  ########:
1. Show route plots/statistics
2. Show driver plots/statistics
3. Time calculations for each driver/route
4. Check speed limits
5. Fuel consumption calculations for each driver/route
6. Exit
Choose an option:
```

The user will then input one of the options and, depending on the option selected (1 to 6), the appropriate script will be called. Each of the entries in the menu corresponds to one of the tasks in the project. If the input number is incorrect, an error message must be shown saying "*Incorrect option: it must be between 1 and 6*".

Before showing the menu, the console will be cleared. This menu will be shown after any of the options is selected, until the selected option is *6* (*Exit*), which will make the script end without any further messages.

For a better readability of the results, those scripts that print results will be modified to show a message saying *'Press any key to continue'*, and will not end until the user inputs something. This way, the output of the script will be shown in the console window until the user presses a key and the main menu appears again.

Task 4:

Write a script that estimates the time it took to do each of the routes to each of the drivers. We will do this in several steps:

1. Write a function that linearly interpolates from a set of x-y values:

```
function [ interpolatedY ] = interpolateLinearly( xVector, yVector , x)

%This function gets a vector with x-values and a vector with y-values
%and returns the interpolated value of y for the given x.
```

   Check the correctness of the function with the following values:

```
interpolateLinearly([10 20 40],[120 150 130],10) => 120
interpolateLinearly([10 20 40],[120 150 130],15) => 135
interpolateLinearly([10 20 40],[120 150 130],25) => 145
interpolateLinearly([10 20 40],[120 150 130],40) => 130
```

2. Write a function that, given an amount of kilometers, returns the equivalent number of meters:

```
function [ m ] = toMeters( km )
```

   Check its correctness with:

```
toMeters(2.3) => 2300
toMeters(0.3) => 300
```

3. Write a function that, given a speed in *km/h*, returns the equivalent speed in *m/s*:

```
function [ msSpeed ] = toMetersPerSecond( speedKmH )
```

   Check its correctness with:

```
toMetersPerSecond(120) => 33.3333
toMetersPerSecond(15) => 4.1667
```

4. Write a function that estimates the time taken in a route:

```
function [ estimatedTime ] = estimateTime( kms, speedKmH, numSlices)

%This function is given a vector with a vehicle's speed (speedKmH) at
```

```
%different points (kms) and the number of integration slices we want to
%use (numSlices)
```

Taking any of the route-driver logs as a reference, *kms* would have the values in the first column of the file, and *speedKmH* would have the values in the second column.

This function will linearly interpolate speeds in the given vector for the *numSlices* integration slices we will be using. For each slice, we will interpolate the speed using the function we did in the previous sub-task, and estimate the time needed to cover the length of the slice. The result will be the amount of seconds taken to do that route.

Test the correctness of the function with the following values:

```
estimateTime([0 1 2 3],[40 50 40 30],1000) => 264.2797
estimateTime([0 1 2 3],[40 50 40 20],1000) => 285.5495
```

5. Write a function that, given an amount of seconds, returns a string with the format HH:MM:SS (hours, minutes and seconds, each with 2 digits). Fractions of seconds will be ignored.

```
function [ hms ] = toHMS( seconds )
```

Check the correctness of the function with the following values:

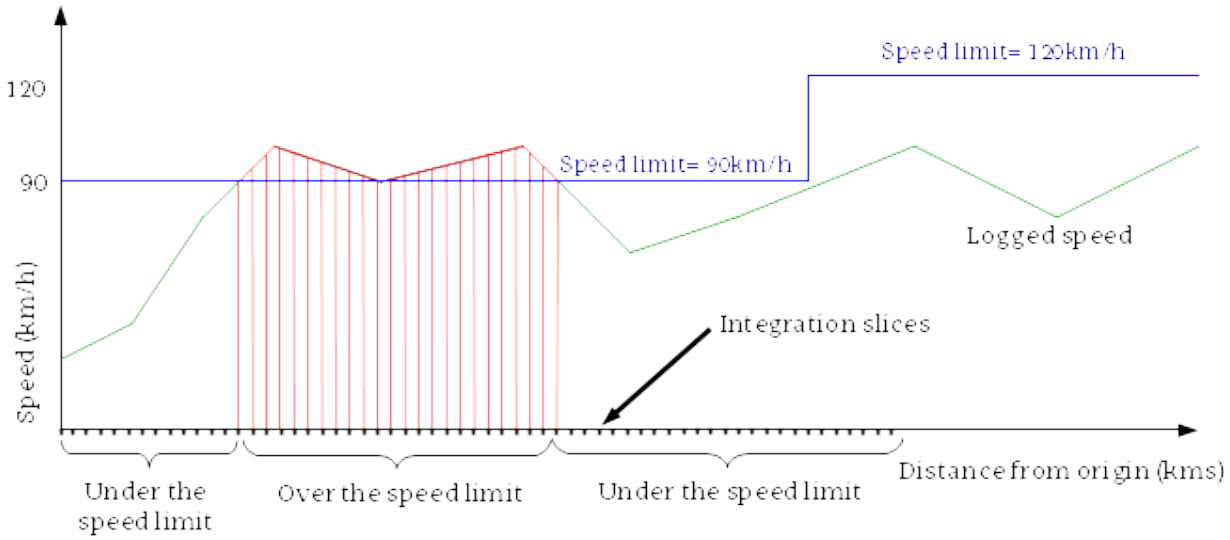
```
toHMS(234.4) => 00:03:54
toHMS(3601) => 01:00:01
```

6. Using *estimateTime* function, write a script that estimates the time taken for each of the route-driver combinations in *hours:minutes:seconds* format:

```
Estimated time for driver1 in route n1: 1:58:54
Estimated time for driver2 in route n1: 1:56:13
Estimated time for driver1 in route a1: 1:54:09
Estimated time for driver2 in route a1: 1:51:12
```

Task 5:

Write a script that analyzes how much time each driver drove over the speed limit in each route.



We will use need to use the 4 log files and the 2 files with the speed limits. Each of those files offers a different resolution, so we need to interpolate.

Basic idea: Given a route-driver log, we will integrate the pieces of route in which the driver has driven over the speed limit. To do so, we will use small slices and, for each of them, calculate the interpolated speed of the driver at the beginning of the slice, compare it with the speed limit at that same point. If the speed is over the limit, integrate the area of the slice and add it to the number of kilometers the driver has driven over the limit.

We suggest you do it following these steps:

1. The speed limits need to interpolated differently as speeds in the log. For example, if the speed limits are given by:

```
kms=          [0   10   40];
speedLimit=   [90  120  90];
```

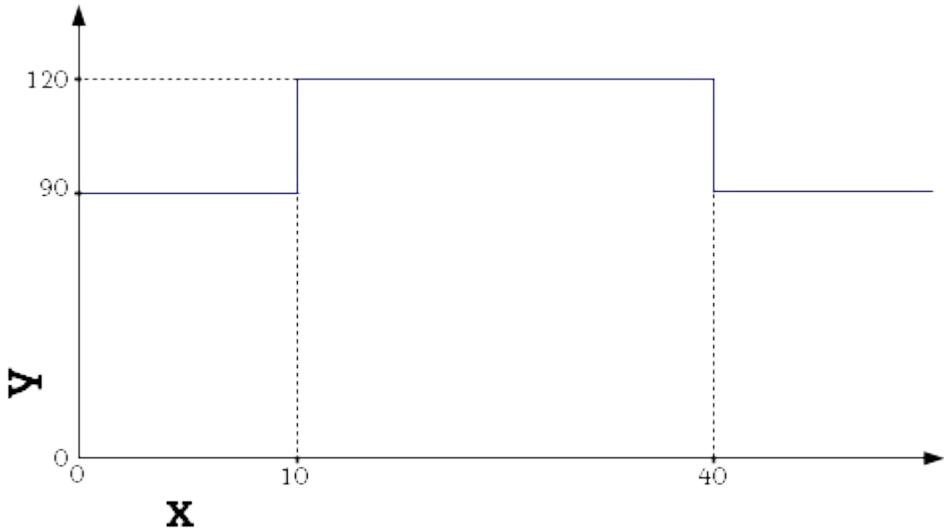
, this means that the speed limit in the first 10 kilometers is 90km/h, then 120 km/h from kilometer 10 to 40, and 90km/h from kilometer 40 on.

Write the following function that returns the interpolated value of y for the given x:

```
function [ interpolatedY ] = interpolateToTheLeft( xVector, yVector , x)

% This function gets a vector with x-values and a vector with y-values
% and returns the interpolated value of y for the given x.
```

With the values from the example, this function should return the value in yVector corresponding to the closest value to x **to the left** in xVector. Visually, the output in the example should be:



Check the correctness of the function with the following values:

```
interpolateToTheLeft([0 10 40],[90 120 90],3) => 90
interpolateToTheLeft([0 10 40],[90 120 90],15) => 120
interpolateToTheLeft([0 10 40],[90 120 90],30) => 120
interpolateToTheLeft([0 10 40],[90 120 90],100) => 90
```

2. Write a function that, given a set of speeds on different points along a route (*driverLogKm* and *driverLogSpeed*), a set of speed limits on different points along the same route (*limitKms* and *limitSpeeds*), and the number

of slices used in the calculations (*numSlices*), returns the number of kilometers in which the driver drove above the speed limit (*kmsAboveSpeedLimit*), and the percentage of kilometers in the route above the speed limit (*percentAboveSpeedLimit*).

```
function [ kmsAboveSpeedLimit,percentAboveSpeedLimit ] =

        checkSpeedLimits( driverLogKm, driverLogSpeed, limitKms, limitSpeeds, numSlices)
```

3. Write a script that uses the previous function to estimate the risk of receiving a fine due to a speed limit in each route-driver log. For each route-driver, we will print a message depending on the percentage of the route the driver drove above the speed limit. If it is above 10%, we will consider it a *high risk*, and a *mild risk* below that number. Of course, if it is 0%, there will be *no risk* at all.

   The output should be:

```
Analyzing: Driver= driver1, Route= n1
Mild infraction risk: Kms above the speed limit= 10.43 (5.64% of
the route)

Analyzing: Driver= driver2, Route= n1
HIGH INFRACTION RISK: Kms above the speed limit= 149.39 (80.75% of
the route)

Analyzing: Driver= driver1, Route= a1
No risk of infraction

Analyzing: Driver= driver2, Route= a1
HIGH INFRACTION RISK: Kms above the speed limit= 131.29 (70.97% of
the route)
```

Task 6:

Write a script that estimates how much fuel has been consumed by a truck in each route-driver log. We will use the following formula to estimate the fuel consumption:

$$f = \frac{d * (0.00009\, v + (0.021\, a + 0.087\, \theta)^2)}{33} \quad ,$$

where f is the fuel consumed (in liters) to drive d meters over a slope of $\theta\ (rad)$ with speed $v\ (m/s)$ and acceleration $a\ (m/s^2)$. This formula assumes constant speed, acceleration and slope, so we will have to integrate over the distance of the route-driver log.

We suggest you do this task following these steps:

1. Write a function that calculates the fuel expenditure for short distances where the speed, acceleration and slope are nearly constant:

   ```
   function [ fuelExpenditure ] = calculateFuelExpenditure(v,a,theta,d)
   ```

   Check the correctness of the function with these values (*33.33 m/s = 120 km/h*):

   ```
   calculateFuelExpenditure(33.33,0,0,1000) => 0.0909
   calculateFuelExpenditure(33.33,1,0.3,1000) => 0.1581
   calculateFuelExpenditure(33.33,1,-0.3,1000) => 0.0917
   ```

2. Write a function that calculates the fuel expenditure in a complete route:

   ```
   function [ fuelExpenditure ] =
   calculateRouteFuelExpenditure(routeKms,routeHeights,logKms,logSpeeds,num
   Slices )
   ```

   , where the elevation profile of the route is given in vectors *routeKms* and *routeHeights*, the speed at each point is given in vectors *logKms* and *logSpeeds*, and the number of integration slices is given in *numSlices*. This function will return the estimated fuel consumption in the route.

   We suggest to integrate over distance on the route. In order to calculate the speed, acceleration and slope at the beginning of each slice:

- Beware of the units used in each variable/parameter. Don't mix meters with kilometers or km/h with m/s.
- You can calculate the speed at any given point along the distance d_i by getting the interpolated value of the speed at d_i .
- To calculate the acceleration, you only need the speed in d_i and a close value before it, say, for example $d_{i-1} = d_i - 0.01$.
- To calculate the angle of the slope (α in *rad*) of the line segment between points $p_0 = \langle x_0 y_0 \rangle$ and $p_1 = \langle x_1 y_1 \rangle$, you can use the following formula:

$$\alpha = \arctan \frac{(y_1 - y_0)}{(x_1 - x_0)} \quad .$$

3. Write a log that calculates and prints the estimated fuel consumption for each route-driver log. The output should be similar to:

```
Analyzing: Driver= driver1, Route= n1
Estimated fuel consumption: 8.383010 liters of fuel

Analyzing: Driver= driver2, Route= n1
Estimated fuel consumption: 8.611214 liters of fuel

Analyzing: Driver= driver1, Route= a1
Estimated fuel consumption: 9.009569 liters of fuel

Analyzing: Driver= driver2, Route= a1
Estimated fuel consumption: 9.324132 liters of fuel
```