INTRODUCCIÓN

Este documento es el conjunto de condiciones y tareas para el proyecto por grupos para la Evaluación Continua de la asignatura Fundamentos de Informática en la Escuela de Ingeniería de Vitoria, UPV/EHU.

REGLAS GENERALES

- Los grupos deben seer como máximo de **tres** personas, y deben trabajar cada uno sin ayuda de otro grupo. Los grupos que presenten código sospechosamente parecido suspenderán el proyecto y la asignatura en su totalidad.
- Se debe subir a eGela la última versión que se tenga del trabajo **por lo menos una vez por semana**. Tened en cuenta de que hacer grandes cambios en poco tiempo es sospechoso para el profesorado. Para evitar problemas, subid el código a menudo para que se pueda ver la evolución de vuestro trabajo (ver sección Subir el Proyecto).

EVALUACIÓN POR GRUPOS

Cada grupo tendrá una calificación que será la base de la nota de cada alumno (que puede ser mayor o menor que la del grupo). La nota máxima del grupo dependerá del número de tareas finalizadas correctamente por el grupo:

- 100 %: todas las tareas
- 75 %: tareas 1, 2, 3, 4 y 7.

DIRECTRICES PARA LA PROGRAMACIÓN

- No imprimir mensajes por pantalla, crear gráficas o pedir entradas dentro de una **función**, a no ser que sea el objetivo de dicha función. La entrada y salida de datos de las funciones es mediante parámetros y resultado.
- Cuando sea posible, intenta reutilizar código mediante llamadas a funciones. En la mayor parte de las tareas se indicará cómo diseñar las funciones.
- Elige nombres con significativos y con la nomenclatura adecuada para crear variables/*scripts*/funciones, de modo que cualquiera que lea el código pueda entenderlo y seguir el valor de las variables sin pasar tiempo averiguando qué es cada elemento.
- Usa ciclos (`for`/`while`) para iterar (repetir instrucciones) sobre un conjunto de valores en lugar de repetir código ya existente.
- Escribe comentarios (con el símbolo de porcentaje %) para explicar el objetivo de cada parte de los programas y qué datos almacenan las variables.

SUBIR EL PROYECTO

1. Con la intención de facilitar la revisión de vuestro trabajo, observar la evolución y evitar el fraude, os solicitamos que subáis el código a la tarea de *eGela* habilitada al respecto, **al menos una vez a la semana**. Para comprobar la evolución del proyecto **no borréis** versiones antiguas.
2. Tan sólo un/una integrante del grupo debe subir el código.
3. Cada vez que se suba el código es obligatorio subir un fichero `ZIP`:
   a) El nombre del fichero debe tener exactamente el formato siguiente: `'project-`*yyyy*`-`*mm*`-`*dd*`.zip'`, donde *yyyy* es el año, *mm* el mes y *dd* el día.

b) El fichero `zip` debe contener todos los *scripts* y funciones que se hayan desarrollado hasta la fecha.

c) El fichero `zip` también debe contener **todos** los resultados solicitados en las tareas que hayáis podido desarrollar hasta ese punto (gráficas, ficheros de texto, salidas por pantalla).

d) Dentro del fichero **también** se incluirá un fichero **único** llamado `'log.txt'`, donde se informará de lo realizado cada día de desarrollo del proyecto. Este fichero irá creciendo con las aportaciones del trabajo realizado cada semana.

No se deben borrar las partes ya entregadas en un `zip` anterior, sino que el último `zip` entregado debe contener la totalidad de ficheros generados en las tareas del proyecto.

Por ejemplo:

log.txt en `project-2021-12-03.zip`

2021-11-29
Tarea 1: Comenzamos a programar el script 'myscript.m', pero todavía no funciona. Aitor hace la parte de carga de ficheros, Blanca la selección de columnas y Jon la del plot.

2021-12-03
Tarea 1: Ya lo hemos terminado de revisar, había un error en el bucle principal que ha encontrado Blanca.

Task 2: Hemos comenzado a escribir en clase la función 'myfunction'

OBJETIVO DEL PROYECTO

En los últimos años la práctica deportiva ha ido creciendo en varios sectores. En especial, la afición a pruebas de resistencia ha aumentado notablemente en los últimos 10 o 15 años. Hoy en día todos conocemos a alguien que se ha aficionado a correr maratones, ultra-maratones o alguna prueba similar de fondo. Además, de-

bido a la pandemia, la práctica del ciclismo en ruta ha crecido significativamente, hecho que ha afectado a la disponibilidad de bicicletas en los comercios.

Por otro lado, el uso de redes sociales se ha normalizado en la sociedad, incluso en el ámbito deportivo. Redes sociales especializadas, tales como STRAVA, son utilizadas con frecuencia por corredores y ciclistas de toda condición. El uso de este tipo de aplicaciones, que utilizan relojes y dispositivos GPS, han generado ingente cantidad de datos que podemos explotar.

En este proyecto procesaremos los datos recopilados mediante dispositivos GPS utilizados por varios ciclistas para generar informes de sus actividades. Estos informes describirán, entre otros aspectos, el rendimiento alcanzado en su actividad. En esta práctica se debe completar el proyecto implementando las funcionalidades indicadas en el apartado Tareas.

FICHEROS DE ENTRADA

En este proyecto se procesarán los ficheros que se incluyen en el fichero denominado `ProjectFiles.zip`. Dicho fichero se deberá descomprimir y ser extraído en la carpeta del proyecto. De esta forma tendremos dos carpetas, `Athletes` y `TrackFiles`. La carpeta `Athletes` contiene un único fichero, `weights.csv`, que almacena los pesos de cada ciclista cuyos datos vamos a procesar en este proyecto. La carpeta `TrackFiles` contiene los ficheros correspondientes a las actividades cuya información se va a procesar en el proyecto. Estas actividades están organizadas en carpetas, una por cada deportista, y cada una contiene los ficheros correspondientes a las *rutas* o *tracks* de una actividad. Los nombres de los ficheros están estandarizados y siguen un patrón simple:

```
activity-identificador-num.csv
```

donde:

- identificador: código alfanúmerico que identifica al ciclista.
- num: número de dos dígitos que identifica cada actividad del ciclista.

De esta manera las actividades del atleta *Athlete1* se guardarán de la siguiente manera:

- `activity-Athlete1-01.csv`
- `activity-Athlete1-02.csv`
- ...

Ejemplo: *actividad-Athlete-01.csv*

```
Duracion, Longitud, Latitud, Elevacion, PC
0,145.6643462833017, -16.74826394766569, 30.799999237060547, 124
4,145.66413095220923, -16.748228408396244, 31.399999618530273, 125
5,145.66407663747668, -16.748218266293406, 31.799999237060547, 125
...
```

Cada fichero de registro o *log* tiene una línea de cabecera que describe el contenido de las columnas. Cada fila contiene los datos del *punto de ruta* o *trackpoint* recopilados concierta frecuencia por el dispositivo GPS. Cada *trackpoint* contiene la siguiente información:

- **Duración**: Tiempo transcurrido, en segundos, realizando la actividad hasta ese momento.
- **Longitud**: La posición geográfica en la longitud.
- **Latitud**: La posición geográfica en la latitud.
- **Elevación**: Altitud en metros sobre el nivel del mar a la que se encontraba el ciclista en ese instante.
- **PC**: Pulso o frecuencia cardíaca del corredor en el momento del muestreo.

TAREAS

Tarea 1: Visualizar actividad

En esta tarea se analizarán las actividades registrada mediante dispositivos GPS. Para ello, debéis implementar un *script* llamado `examineTracks.m` que analice el recorrido de las rutas, los perfiles del recorrido y la evolución de la frecuencia cardíaca durante la actividad. Por cada ruta o *track* se generará una figura con tres gráficos (subplots). El primer gráfico mostrará la ruta registrada mediante GPS, el segundo gráfico mostrará el perfil del recorrido, mientras que el tercer gráfico mostrará la evolución de la frecuencia cardíaca (ver Figura 1).

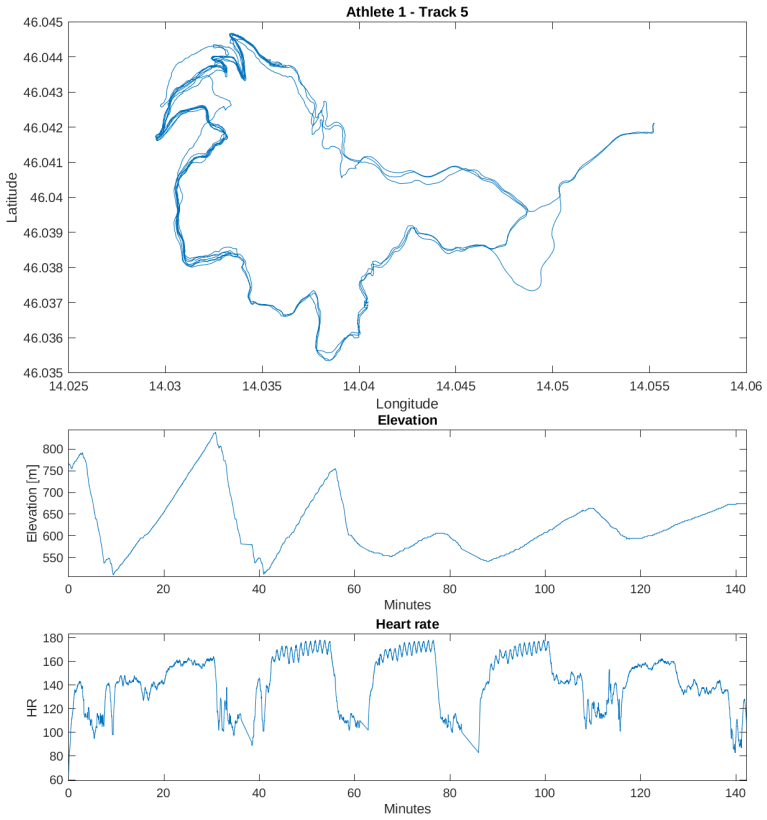

Figura 1: Gráfico de la ruta 5 realizada por el ciclista 1.

El *script* guardará en ficheros `png` dentro de la carpeta `Reports` las figuras generadas para cada una de las actividades. El nombre de los ficheros que contienen las figuras sigue el siguiente patrón:

`plot-identificador-num.png`

donde *identificador* es el ciclista que ha recorrido la ruta, y *num* es el número que identifica la actividad del ciclista.

Ayuda: es conveniente establecer las dimensiones de las figuras generadas para evitar que se solapen etiquetas y títulos. Podéis hacerlo utilizando el siguiente código:

```
figure('visible', 'on', 'Renderer', 'painters',
       'Position', [10 10 900 1000]);
```

Tarea 2: Estadísticas básicas de la actividad

Las personas practicantes del ciclismo suelen prestar atención a determinados datos que resumen la actividad. El *script* `generateTrackSummary` que desarrollaréis en esta tarea calculará y mostrará en pantalla la siguiente información de la actividad:

1) la duración de la actividad en en el formato *horas-minutos-segundos*
2) la distancia total recorrida en kilómetros
3) la velocidad media (km/h) de la actividad realizada
4) la frecuencia cardíaca media

Para ello, debéis completar los siguientes pasos:

1. Implementad una función llamada `periodDuration` que, dado el tiempo empleado para la realización de una actividad en segundos, devuelva la duración de la actividad en horas, minutos y segundos.

   ```
   function [hours, minutes, seconds] =
       periodDuration(timeStamps)
   ```

   Podéis comprobar la corrección de la función ejecutando el siguiente código y comprobando el resultado obtenido.

   ```
   [h,m,s]=periodDuration(13412)
   ```

   ```
   h = 3
   m= 43
   s= 32
   ```

2. Implementad una función llamada `distance` que, dados los vectores que representan las longitudes y latitudes de los puntos de origen y de destino, devuelve un vector que contenga la distancia entre cada par de puntos, calculada en metros.

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

Podéis comprobar la función ejecutándola con el siguiente formato, y comparando con el resultado obtenido:

```
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]
```

Para calcular la distancia usaremos el método pitagórico, que obtiene la distancia en radianes entre dos puntos en una esfera mediante sus coordenadas.

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

donde:

- φ_1 y φ_2 son las *latitudes* de origen y destino **en radianes**
- $\Delta\lambda$ es la distancia entre las *longitudes* de destino y origen **en radianes**
- x es la diferencia entre las *longitudes* de origen y destino **en radianes**
- y es la diferencia entre las *latitudes* de origen y destino **en radianes**

Posteriormente el resultado hay que multiplicarlo por el radio medio de La Tierra ($R = 6371km$) para calcular la distancia entre los dos puntos usando la siguiente fórmula (y pasar a metros).

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

3. Escribid una función llamada `computeTrackLength` en la que se devuelva la distancia total recorrida en metros a partir de los vectores que contienen las latitudes y longitudes de los puntos de ruta.

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

Podéis comprobar que la función es correcta mediante el ejemplo siguiente, comparando el resultado que se obtiene:

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

```
19376.8456
```

4. Implementad una función llamada `meanSpeed` que, dadas la duración de la actividad en segundos y la distancia total recorrida en metros, devuelva la velocidad media en km/h.

   ```
   function avgS = meanSpeed(time, distance)
   ```

   Podéis comprobar que la función es correcta mediante el ejemplo siguiente, comparando el resultado que se obtiene:

   ```
   avgSpeed = meanSpeed(13412,26177.7054);
   ```

   ```
   avgSpeed = 7.0265
   ```

5. Implementad el *script* `generateTrackSummary` que genere un informe con la información básica de cada una de las actividades realizadas por los deportistas en el siguiente formato:

   ```
   Track summary for athlete #1
   ----------------------------
           Track #1
                   Time spent: 1-25-07
                   Distance: 21.37km
                   Avg. speed: 15.06km/h
                   Avg. heart rate: 178.02
           Track #2
                   ...
   ```

Tarea 3: Análisis de la frecuencia cardíaca

Una forma de analizar el esfuerzo es extraer los rangos de los ritmos cardíacos acumulados durante la actividad. Los esfuerzos se clasifican en cinco zonas en función de los rangos de frecuencia cardíaca. Aunque cada persona suele tener asociado sus propios rangos, en este caso utilizaremos los mismos para todos los ciclistas. Los rangos predefinidos se muestran en la Tabla 1.

Zona	Descripción	Rango
1	Resistencia	<123
2	Moderado	[123, 153)
3	Ritmo	[153, 169)
4	Umbral	[169, 184)
5	Anaeróbico	>184

Tabla 1: Rangos predefinidos de las zonas de esfuerzo

En esta tarea debéis implementar el *script* `extractHRZones` que solicitará al usuario que introduzca el identificador del ciclista y el número de la actividad que se quiere procesar y mostrará a continuación una figura que presenta tres gráficas con la siguiente información (ver Figura 2):

1) la evolución del pulso cardíaco a lo largo de la actividad. Esta gráfica además delimita los umbrales de las zonas mediante líneas discontinuas para apreciar mejor el tipo de esfuerzo realizado
2) tiempo transcurrido en cada una de las zonas
3) porcentaje de tiempo transcurrido en cada una de las zonas

Para ello, debéis implementar la función `analyzeHRZones` que, dados el vector que contiene los instantes de tiempo en los que se registraron las medidas y el vector que contiene las frecuencias cardíacas registradas durante la actividad, devuelve un vector que contiene tiempo transcurrido en cada de las zonas de esfuerzo.

```
function hrzones = analyzeHRZones(timestamps,
                                  heartRates)
```

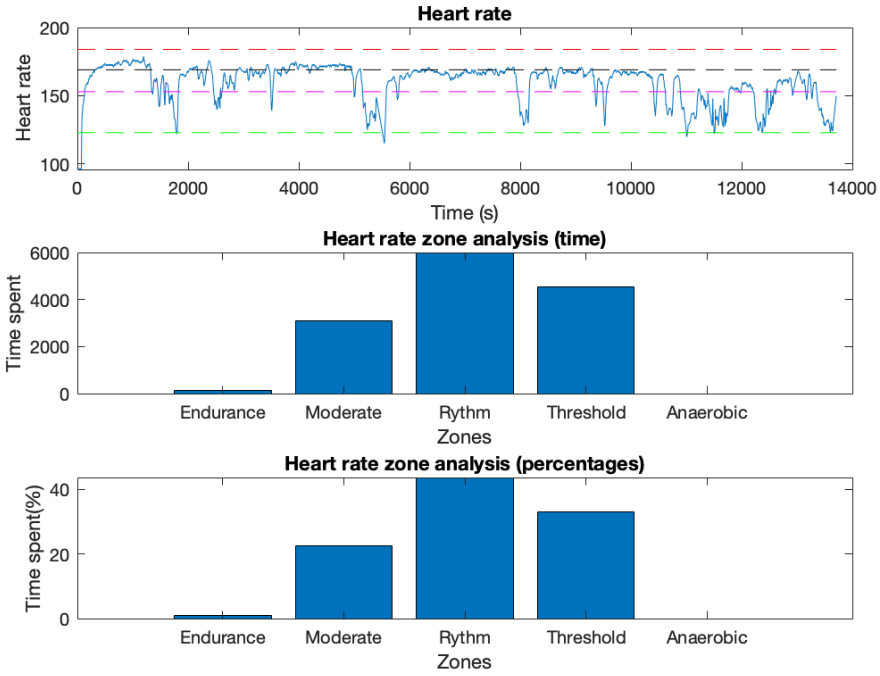

Figura 2: Informe correspondiente al análisis de la frecuencia cardíaca realizado para el atleta 2 en la actividad 4

Puedes comprobar el correcto funcionamiento de la función ejecutando el código que se muestra a continuación y comprobando el resultado obtenido:

```
timeStamps = 0:5:50;
heartRates= [70 131 142 139 160 165 201 199 172 171 140];
hrZoneTimes = analyzeHRZones(timeStamps, heartRates);

hrZoneTimes = 5    15    10    10    10
```

Tarea 4: Consumo de calorías

El consumo energético depende la duración de la actividad y de la intensidad con la que se realiza. A mayor duración del ejercicio, mayor es el consumo energético. Lo mismo ocurre con la intensidad. Cuando mayor la intensidad, mayor es el consumo energético. En este proyecto, el consumo energético se calcula en calorías.

Uno de los métodos más sencillos para medir la intensidad de una actividad física es el método *Metabolic Equivalent Task* (MET). El consumo de energía en una actividad se determina monitorizando el consumo de oxígeno durante su realización, estimando el consumo promedio de oxígeno por unidad de tiempo, y comparándolo con el consumo de oxígeno en reposo. Una unidad MET representa la energía consumida en reposo, dos MET indican que la energía consumida ha sido el doble que en reposo, tres MET el triple del gasto energético en reposo, y así sucesivamente.

La estimación de las calorías consumidas al realizar una actividad se realiza aplicando la siguiente ecuación:

$$totalCalorias = \sum_{i=1}^{n} calorias_i \tag{1}$$

donde $calorias_i$ representa el consumo calórico realizado en un instante i y que se calcula mediante la siguiente fórmula:

$$calorias_i = \Delta t_i * 3{,}5 * MET * weight/200 \tag{2}$$

donde Δt_i representa la duración del instante i en minutos, *weight* indica el peso del deportista en kilogramos y MET representa la unidad MET en la realización de una actividad. Para estimar el MET en una actividad ciclista se utiliza la siguiente fórmula:

$$MET = -1{,}52 + 0{,}510 * speed \tag{3}$$

donde *speed* se refiere a la velocidad en dicho momento en km/h.

Para llevar a cabo esta tarea debéis:

1. Implementar una función llamada `speed` que, dados el vector de los tiempos de los registros, el vector que contiene las longitudes y el vector que registra las latitudes de la actividad, devuelve un vector que contiene la velocidad entre cada par de puntos de la ruta en kilómetros por hora.

```
function speeds = speed(timeStamps, longitudes,
                        latitudes)
```

Puedes comprobar el correcto funcionamiento de la función ejecutando el código que se muestra a continuación y comprobando el resultado obtenido:

```
timeStamps = [0 1 6 10 16];
longitudes= [14.823283000000000 14.823276000000000
     14.823195999999999 14.823110000000000
     14.823014000000001];
latitudes = [46.496094999999997 46.496110999999999
     46.496226999999998 46.496350999999997
     46.496547000000000];
speeds=speed(timestamps, longitudes,latitudes)

speeds = 6.689006040809976  10.280494372716902
         13.751158512890941  13.799835281867271
```

2. Implementa una función `computeCalories` que, dados el vector de los tiempos de los registros, los vectores con longitudes y latitudes de la actividad, y el peso del ciclista, devuelve las calorías consumidas al realizar la actividad.

```
function calories = computeCalories(timeStamps,
                    longitudes, latitudes, weight)
```

Puedes comprobar el correcto funcionamiento de la función ejecutando el código que se muestra a continuación y comprobando el resultado obtenido:

```
timeStamps = [0 1 6 10 16];
longitudes= [14.823283000000000 14.823276000000000
             14.823195999999999 14.823110000000000
             14.823014000000001];
latitudes = [46.496094999999997 46.496110999999999
             46.496226999999998 46.496350999999997
             46.496547000000000];
calories = computeCalories(timestamps, longitudes,
                           latitudes, 70)

calories = 1.543224641642740
```

3. Desarrolla un *script* llamado `estimateCalories` que analice todas las actividades de cada ciclista y que muestre en un gráfico distinto el consumo energético de cada ciclista en las actividades realizadas (ver Figura 3).

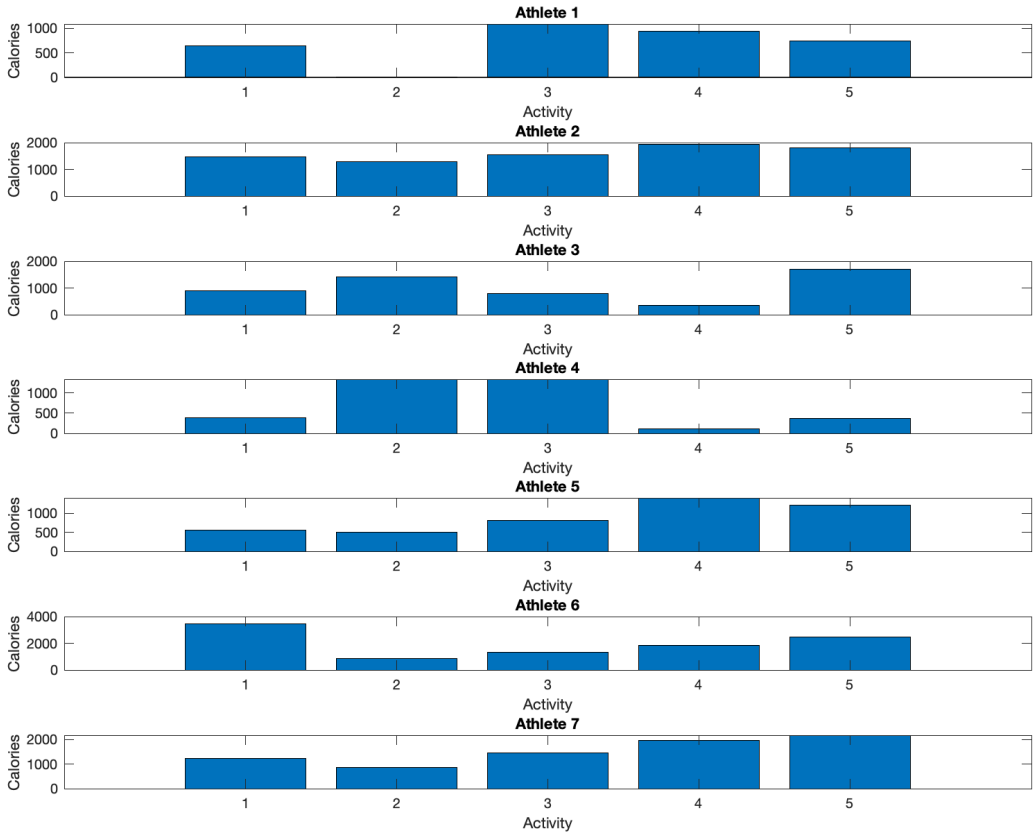

Figura 3: Informe del análisis de consumo calórico con un gráfico por ciclista

Tarea 5: Estadísticas avanzadas

Las personas que practican actividades deportivas frecuentemente suelen estar interesadas en la información que describe la actividad realizada, más allá de las estadísticas básicas. Por ello, los relojes GPS o demás dispositivos desarrollados para la práctica deportiva suelen mostrar datos tales como la velocidad máxima alcanzada, desnivel positivo acumulado, etc.

En esta tarea debéis:

1. Implementar la función `speedBykm` que, dados los vectores que contienen los instantes de tiempo en la que se han tomado las medidas, las longitudes y las latitudes, devuelve un vector que, para cada kilómetro, indica la velocidad a la que se ha realizado.

   ```
   function kmspeeds = speedBykm(timestamps,
                                 longitudes,
                                 latitudes)
   ```

   Para ello, esta función debe implementar el proceso que se describe a continuación.

   1) Calcular la longitud de la ruta en kilómetros y determinar N, el número de kilómetros completos (podéis utilizar la función `fix` para truncar un número real al entero inmediatamente inferior).
   2) Crear un vector. *velocidades* que tenga N elementos.
   3) Inicializar las variables que almacenarán la distancia recorrida hasta el momento y el tiempo empleado hasta el momento.
   4) Para cada par de puntos de la ruta.
      1) Sumar la distancia entre los dos puntos a la distancia recorrida hasta el momento.
      2) Sumar el tiempo empleado para desplazarse entre los dos puntos al tiempo empleado hasta el momento.
      3) Si la distancia supera el kilómetro, calcular la velocidad y guardarla en la posición correspondiente del vector. *velocidades*. Reinicializar la distancia y el tiempo.

2. Implementar el *script* `generateAdvancedStatistics` que genera para cada actividad realizada por un ciclista un fichero en la carpeta `Reports` que contiene un informe visual de la actividad (ver Figura 4). Los nombres de los ficheros siguen el patrón

```
identificador-num-statistics.png
```

donde *identificador* es el código alfanumérico que identifica al ciclista y *num* el identificador de la actividad.

El informe visual contiene la siguiente información:

1) **Evolución de la velocidad**: Esta gráfica muestra la velocidad a la que se ha desplazado en cada momento (velocidad entre dos puntos consecutivos de la ruta) el ciclista. Además, muestra mediante líneas horizontales la velocidad máxima alcanzada, la velocidad media y la velocidad mínima.
2) **Velocidad por km**: Muestra la velocidad a la que ha realizada cada uno de los kilómetros de la actividad.
3) **Perfil de la actividad**: Muestra la elevación en cada punto de la ruta.
4) **Ascenso acumulado**: Muestra el ascenso acumulado hasta el momento en cada punto de la ruta.

Tarea 6: Comparación entre ciclistas

En esta tarea queremos comparar la actividad de los distintos ciclistas de los que disponemos información. Para ello, debéis:

1. Implementar una función que se llama `generateAthleteReport` que, dados el identificador del atleta, un vector que contiene los identificadores de las rutas realizadas y el peso del ciclista, devuelve los datos que se enumeran a continuación:

    1) Distancia total recorrida por el atleta teniendo en cuenta todo su historial de actividades

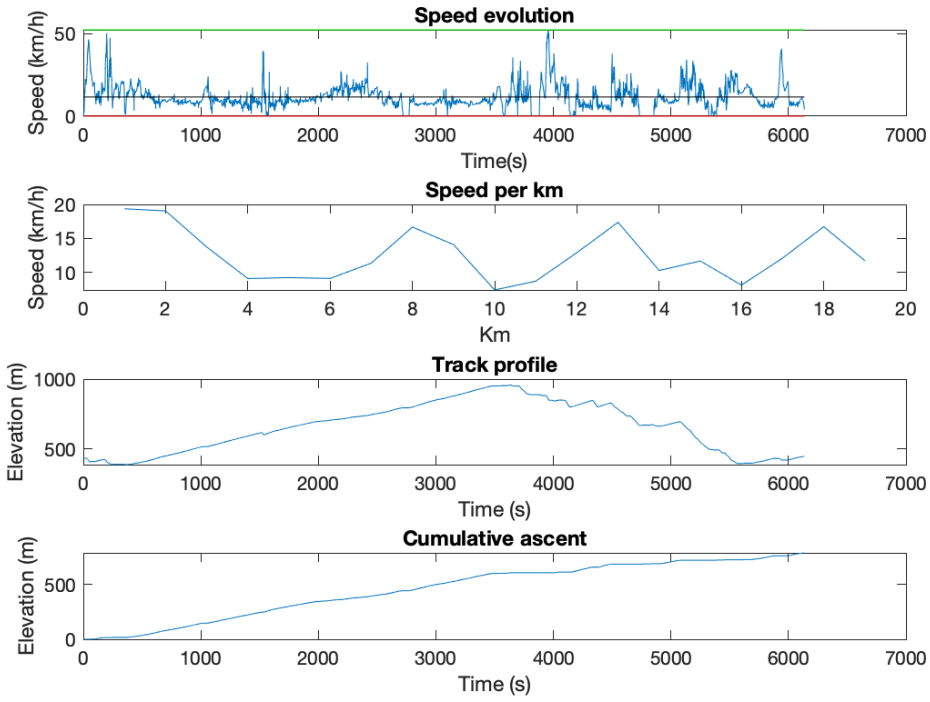

Figura 4: Estadísticas avanzadas correspondientes a la 1ª actividad de la ciclista 5

2) Distancia media recorrida por el ciclista en las actividades realizadas
3) Velocidad media del ciclista
4) Consumo energético medio del ciclista en las actividades realizadas

```
function [totalkms, avgkms,avgSpeed,totalCalories]
    = generateAthleteReport(athlete,tracks,weight)
```

2. Implementar un *script* llamado `getAllReports` que muestra un informe con la información que se presenta en la Tabla 2.

Athlete	km totales	km/actividad	km/h media	calorías
Athlete1	120.44	24.09	12.20	3411.70
Athlete2	252.81	50.56	19.51	8028.91
Athlete3	...	...	...	...
...	...	...	...	...

Tabla 2: Informe comparativo de los ciclistas

Además, para cada una de las columnas creará en la carpeta `Reports` un fichero que presenta la información correspondiente en un diagrama de barras (ver Figura 5). Los ficheros tendrán nombres estandarizados siguiendo el patrón

`dimension-num.png`

donde *dimension* representa el nombre de la columna que contiene la información utilizada para generar la gráfica.

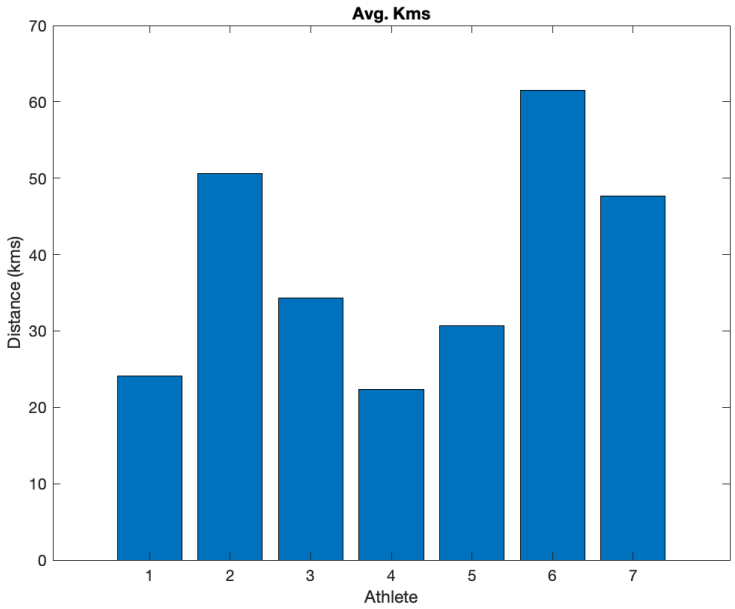

Figura 5: Ejemplo de diagrama de barras para km/actividad

Tarea 7: Menú del programa principal

Immplementa el *script* `mainMenu` que muestre el siguiente menú.

```
###### MENU ######
1. Visualize activity
2. Show basic statistics
3. Heart rate analysis
4. Calculate calorie consumption
5. Show advanced statistics
6. Comparison among cyclists
0. Exit
###### MENU ######
Select an option:
```

Según la opción elegida, se ejecutará el *script* de la tarea correspondiente. Cada entrada del menú se corresponde con una de las tareas realizadas en este proyecto. Si el número introducido como entrada no es correcto, el *script* mostrará un mensaje de error: `La opción elegida no es correcta: elige una opción entre 1 y 6, o 0 para salir.`

Antes de mostrar el menú, el *script* limpiará la consola. Al elegir una de las opciones, se mostrarán los resultados de la tarea correspondiente. A continuación, se mostrará de nuevo el menú. Esta operación se repetirá mientras no se elija la opción *0. Exit*. No uses la función `exit()` ya que cierra Matlab/Octave.

Después de mostrar los resultados de una tarea y antes de mostrar el menú de nuevo, se indicará a la persona usuaria que debe pulsar alguna tecla mediante un mensaje: "`Pulsa una tecla para continuar...`" para poder ver los resultados con la instrucción `pause`. A continuación se mostrará de nuevo el menú. Esta operación se repetirá mientras no se elija la opción 0.