Self Study 4 –Return To Ilkley Moor

Introduction

In this self study exercise you'll go back and have a look at the Ilkley Moor Orienteering GPS data collected and create some plots of the data. You'll also look at manipulating the data using indexing and for loops, similar to the work you undertook in Lab 9 of the LabVIEW material but now taking on similar tasks in MATLAB. As usual, follow the steps below. They involve you using what you have learned during the Lab sessions so far to solve a problem. As per usual we will be using ScriptCheck.

The Scenario

You took part in the Ilkley Moor Night Orienteering event and had a great time. While out on the moor, you recorded your position using a GPS tracker. Unfortunately one of organisers lost his hat and you need to get it back so he isn't on Ilkley Moor baht'at. The only thing he can remember about it, is that it was lost at the highest point on the route. In order to retrieve the hat, we'd like you to look at your data and determine its location. While you're at it we'd like to see the route you took and the distance you'd travelled to get to that point from the start of the course. Your help would be much appreciated in this matter.

Again this should all be done programmatically. You'll need to do some file IO and plotting as well as manipulating vectors to complete this challenge. Remember anything in red in the example code needs to be replaced. Note that scriptcheck will be unable to check if your plots are correct, just that you've called them and labelled them. It will be up to you to decide if your plot looks correct.

Here are the list of variables you will create in this task, cut and paste as needed to avoid typos:

```
data_points
east
north
alt
```

```
max_alt
index_hat
east_hat
north_hat
alt_hat
distance
solution
```

Instructions

1. Download the ilkley_orienteering.txt file from the Self-study section of the VLE and save to your local working directory before you start. This file contains your GPS data that has already been converted from Longitiude and Latitude to UTM coordinates.

2. Now open the editor window and create a new .m file and save as return_to_ilkey_moor.m. As always, we want to place a comment describing the function of the program at the start.

3. Programmatically clear the workspace and the command window (QG 0).

4. The data we would like to read is in ilkley_orienteering.txt. The file has been saved with a tab as a delimiter rather than a comma, as such we cannot use csvread. Instead we can use dlmread which allows us to specify the delimiter used, in this case it is a tab (`'\t'` is used for the delimiter argument). Type `help dlmread` in the command window for help on using this function, which is similar to csvread used in the lab session (QG 11.3). In your current folder window, right click on ilkley_orienteering.txt and select Open Outside MATLAB. Have a look at the data the txt file contains, note each bit of data is separated by a tab. Work out which columns you need for utm_easting, utm_northing and altitude (m) respectively.

5. As before we don't immediately know the length of the data so we use the length function with dlmread embedded and assigning to a variable `data_points`. (Note that we want to start reading from column 1 due to the format of the date in the text file which is not compatible with dlmread).

```
data_points = length(dlmread('ilkley_orienteering.txt', '\t', 1, 1));
```

This gives us the longer of the two dimensions of the data, in this case the number of rows starting at row 1, column 1 (remember zero indexing means this is actually the 2nd row and column).

6. We then want to use the `dlmread` again 3 times, once for each of the columns we are interested in. Set the output to the variable names `east, north` and `alt.` dlmread will be in the format:

```
data = dlmread('filename', '\t', [R1, C1, R2, C2])
```

You'll have to specify the values for the range, [R1, C1, R2, C2], C1 and C2 will be the same specifying the column of the data you want (see step 4.) R1 will be the first row of position data (remember zero indexing) and R2 will be the length of the data found in step 5. The result will be three variables of vectors, `east`, `north` and `alt` containing the UTM easting, UTM northing and altitude data respectively.
Note: steps 5 and 6 could be replaced by reading the entire file except the header and first column and then indexing that data to get the three columns and assign to the variable names as we covered in the lab. Either method is fine.

7. Due to the nature of the task, we are only interested in the route up until the maximum altitude is reached. We can dispose of the position data after this point. Use the max function with the second output argument for the index of where the maximum value occurs. Again use `help max` to clarify this. We'll use this function in the form:

```
[x, y] = max(z);
```

where x is the maximum value in vector z and y is its index. x and y can be replaced by any variable name you like, we'll use `max_alt` and `index_hat` respectively. z will be the altitude vector, `alt`. We can then use `max_alt` and `index_hat` elsewhere for the altitude of the missing hat and the index.

8. Get a subset of the *east*, *north* and alt vectors that contain the data from the start of the route up until the point at the highest altitude. Set these to new variables `east_hat`, `north_hat` and `alt_hat`. Look back at how to index vectors from Lab 1, using the colon operator a:b to get all values between a and b (QG 2.1).

9. Now a tricky part, we want to create a new vector that contains the cumulative distance travelled from the start of the route to the position of the lost hat. The result will be a vector of the total distance at each step so will be the same length as `alt_hat`. We know that distance is zero at the start. When we want to build a large vector in a `for` loop, it is often a good idea to initially create a vector of the same length containing zeros. This allows the loop to run quicker as memory is already pre-allocated. To do this we can use a function called `zeros()`. Let's create a vector called `distance`. To do this use:

```
distance = zeros(length(alt_hat), 1);
```

This creates a matrix of zeros, the number of rows given by the first input argument, in this case it will be the length of the alt_hat vector and the number of columns given by the second input argument, in this case 1.

10. Now to work out this cumulative distance. To do this we are going to use a for loop, Pythagoras and some indexing. To find the distance travelled between each sample we need to find the distance moved between the next sample and the current sample in each axis.

We are going to use a for loop. As we have to use the current and next value of the vectors east_hat, north_hat and alt_hat we want the loop to run `length(alt_mark)-1` times otherwise we'll run out of samples for the next values. Use a variable x as your counter so your for loop should start with the line:

```
for x = 1:length(alt_hat)-1
```

Now within the for loop we are going to calculate the distance travelled between each sample (including elevation) and set it to a variable `sample_distance`. Use x and x+1 to index the values in `east_hat`, `north_hat` and `alt_hat` we need. Each time the `for` loop iterates, `sample_distance` will take on a new value calculated using Pythagoras.

```
sample_distance = sqrt((????)^2+(????)^2+(????)^2)
```

We want to place the accumulation of these `sample_distance` values into the vector *distance* we created in step 9, starting at index `x+1` as the first value in `distance` should remain zero. To do this we set the value in `distance` at index `x+1` to the value previous value in `distance` (at

index `x`) plus the `sample_distance`. If we do this in each loop iteration, we replace the values in the vector `distance` with the cumulative distance at each sample. End the `for` loop.

12. We now want to write our findings to a string called `solution`. Using `sprintf` again the code needed takes the form below:

```
solution=sprintf(...
    ['The hat can be found at an altitude of %0.1fm'...
    ' at coordinates of %0.1fm UTM-easting and %0.1fm UTM-northing.\n'...
    'You have to walk %0.1fm to get there.'],...
    a, b, c, d)
```

where `a` is the maximum altitude, `b` is the easting value where the hat is, `c` is the northing value where the hat is and `d` is the distance taken to walk there from the start of the course. In the cases of b, c and d, they will be the last value in the corresponding vectors or the value at `index_hat`.

13. Finally we want to visualise some of the data. First of all we want to open a figure window. Use `figure(1)` in your code. This will open a figure window and set figure number 1 to the active figure. Produce a plot using the `plot` function, plotting the altitude against distance travelled up until the highest point. (QG 10.1) Label the plot correctly using:

```
xlabel('label')
ylabel('label')
title('label')
```

Now add a second figure showing the route, use figure(2) to open a new figure window and then use `plot3` to undertake a 3D plot of the route up until the hat by plotting `alt_hat` against `east_hat` and `north_hat`. (QG 10.2) Label each axis on your figure. Use `grid on` after the label commands to help with the visualisation of the data.

14. Run your code in MATLAB and check it works correctly before submitting to ScriptCheck as usual to receive a score and feedback on your work. https://scriptcheck.leeds.ac.uk