1 Introduction

In this lab, you will write a program for the analysis of topographic slopes. Usually we don't have slopes, instead elevation data are stored in a Digital Elevation Model (DEM), which is basically a large matrix containing elevation above sea level over a grid of points. You already saw a DEM in the lab for Week 3, though we didn't call it by that name.

The program you write will use `ginput` to let you pick a point on a plot of the DEM. It will then calculate the slope at that point, and print a message telling you whether that point is east-facing, west-facing, or relatively flat. Your code will have to take into account that the world is round if you select a point on the edge of the DEM.

A useful way of writing this kind of program is to proceed in a few steps:

- Break up the problem into parts. One technique useful for programs handling a lot of 'special cases' is to make it work for only one of the cases as a starting point.
- In this case we won't make you start from scratch - instead you will work on fixing broken code as a starting point (Section 3). Then you have to add a piece of code that does the right thing if the input data is 'good' (Section 4).
- Then, think about ways in which the input data can be 'bad'. Add code to test for these conditions and to handle them in a useful way. This code will often get squeezed into the middle of the existing code (Section 5 does this).

2 Goals

- Create a larger program by modifying a small starting program
- Create algorithms using SELECTIONS ("if statements")
- Implement algorithms using if, if-else, if-elseif-else syntax.
- Begin using formal debugging procedures
- Learn about argument checking for input data.
- Understand that longitudes 'wrap around' the earth and that special techniques must be used to handle this.

3 Load and image global topography

We will begin with code similar to that written for Section 3 of the Week 3 lab (review this if you're having trouble). As a starting point you can use the script on the next page (call it something like `lab5_initial.m`), which has about 8 to 10 errors in it. Find and fix these.

```
% lab5_initial.m

% load a demonstration file (already in the matlab code base) that
% contains a 180x360 matrix with elevations above sea level (in meters)
load topo
% Make lat/long vectors with values corresponding to the rows/columns of
% topo which range from -89.5 to 89.5, and 0.5 to 359.5, respectively.
lat =[89.5:89.5];
long=[0.5:359.5];

% Display the topography, with axes labelled in lat/longs
figure(1); clf;
imagesc(long,lat,topo)
set(gca,'ydir','normal');
xlabel(Longitude);
ylabel(Latitude);

% Grab a single point by clicking with the mouse and find
% the row and column indexes corresponding to the position
% in topo nearest to the point clicked.
[X,Y]=ginput(11);
[~,ix]=min(abs(long-X)); %the ~ is a dummy variable indicating we ...
[~,iy]=min(abs(lat-X));  %  ...don't need this output variable

% Draw a small black square marker at that point, and then add a label
% beside the marker with its height, latitude, and longitude.
line(long(ix),lat(iy),'color','k','marker','s','linewidth',3);
text(long(ix),lat(iy),{['height (m)='    num2str(topo(X,Y))],...
                       [' at lat '   num2str(lat(iy))  ],...
                       [' and long ' num2str(long(iy)) ]});
```

If you think it all works, try running your code and clicking in a few different regions of the image. Carefully check your label to make sure everything makes sense.

Once you've got this piece of code up and running, continue to the rest of the lab.

4 Calculate the topography slope

Once the point selection algorithm is working, you can start adding code to it to solve the following problem: **What is the east/west slope (in degrees) at the point you clicked?** The east/west slope is a height difference between the point immediately to the RIGHT (or EAST) of the one you picked and the one immediately to the LEFT (or WEST) of it, divided by the horizontal distance of these two points. Then take the arc-tangent to get the result in degrees:

$$\text{Slope} \approx \tan^{-1}\left(\frac{topo(i,j+1) - topo(i,j-1)}{\Delta LON \times 111e3 \times cos(LAT)}\right) \quad (1)$$

where

$$topo = \text{your topography array} \quad (2)$$
$$i,j = \text{your row and column indices} \quad (3)$$
$$\Delta LON = \text{the longitude difference between the two points (in degrees)} \quad (4)$$
$$111e3 = \text{because there are 111,000 meters in a degree of longitude at the equator} \quad (5)$$
$$LAT = \text{latitude of the point you selected (ie. **not** the exact location of the mouse click)} \quad (6)$$
$$cos(LAT) = \text{a correction factor to take into account longitude convergence at the poles} \quad (7)$$

1. Add the code to make this work for points well within the DEM (i.e. away from the edges of the map). Make sure the trigonometric functions you use are for degrees and not radians. Check `help cos` and `help cosd`.

2. Then use the built-in function `disp` (and you MUST use `disp`) to output to the screen:

   (a) The latitude and longitude of the selected point

   (b) The elevation AND the slope at that point (in degrees)

   (c) a message saying "Flat" if the slope angle is between -0.1 and +0.1 degrees, but "East-facing" or "West-facing" if the slope is larger and either east- or west-facing, respectively. An "east-facing" slope tilts DOWN towards the east.

   in a nicely formatted ways, something like:

```
At 30.5 N, 100.5 E, Height is 4014 m and the slope is -0.187 degrees
This is East-facing
```

   (note - latitudes can be N or S, longitudes can always be E).

   Hint: You can use the built-in function `num2str` to convert a numeric value to a string; you can then concatenate this string with another string using regular array concatenation:

```
disp(['text'  num2str(value)])
```

3. You can use the following check value to proof your code: the slope at `topo(20,40)` is 0.2629°, and the elevation is 1200 m (Hint: TEMPORARILY modify your code in order to compute the checkvalue).

5 Selections - check arguments

Often with user input, your program has to first check their validity to make sure they don't crash the program. Re-run your code a few times and click around in different places to make sure it works. What happens if you click on a point outside the image itself (i.e., on the gray border)? What values does your program return for `ix` and `iy` if you do this? Where does your program crash? You should notice that if you click outside of the DEM, your code will automatically modify your point and place it on the edge of the map nearest where you clicked. That's good, but in this scenario your slope calculation no longer works. Why?

Modify your code so that your program can still calculate the slope if your point lies on the edge of the DEM. That is, you need to test for the special cases when `ix` has a value of `1` or `length(long)` and, if so, adjust your slope calculation accordingly. Remember that the world is round! You might find it useful to define new variables `iright` and `ileft` to represent the column-indices to the right and to the left of the point you clicked. Think carefully about *where* in your code you should put this addition. If you're having trouble with the logic, try writing some pseudo-code to help you lay out the flow of your program.

6 To Hand In on Connect by 4pm on Friday, October 9

1. A script file called (exactly) `lab5.m` which is exactly the code you have written above - it loads and images the topography matrix `topo`; uses `ginput` to choose a point and places the "informative label" by that point; and uses `disp` to print to the command window the lat/long of the point, the estimated slope at that point, and the east/west/flat phrase. I will try to 'break' your code by clicking in weird places when I run it (like on the side or top grey borders of the image window), so your code must work also if the selected point is right on the edge of the DEM.

2. The script `lab5.m` should be a single, self-consistent piece of code and should (as always) contain the following lines of working code:

```
partner.name='YYYYYY';
Time_spent= XX ; % in hours
```

3. Did your program crash in the run-tester last week because of 'undefined variable' errors? Tip: Quit Matlab, start it up again, and run `lab5.m` from a clean workspace before handing it in.