Problem set 3: Toolboxes

Handed out: Friday, October 25, 2024.

Due: 23h55', Thursday, November 7, 2024.

As you did with previous deliverables, you must hand in two files:

- One file, named `your_name_E3.pdf`, with the solutions to the exercises in this set, following the template available in the virtual campus.
- The second file, named `your_name_E3.zip`, must contain all the MATLAB code you used to solve the exercises.

Please, make sure that the code in the pdf file is **exactly the same** as in the zip file. Remember that all the MATLAB code is supposed to be **entirely yours**. Read the Course Information document, available in the virtual campus, for more details on this subject.

Exercise 1: Area calculation.

Consider a generic unimodal bidimensional function $f(x, y)$. We wish to compute the area of the region of the (x, y) plane for which the value is above a certain threshold α. We are asked to do so using as input the value of the function evaluated in a set of N random locations $(x_1, y_1) \dots (x_N, y_N)$ in the range $0 \leq x_i, y_i < L$. In our case N=8000, $L = 32$. This problem is related to the calculation of the beamwidth of a planar antenna array. It is also related to the calculation of the area of an object in an image, even though in that case the samples are always defined in a regular grid. As a less technical application, if $f(x, y)$ represented the height of the terrain in an island we could think of it as the calculation of the surface of an island that will not be flooded when the sea level rises up to height α.

There are many ways to estimate the area. In this exercise we are going to explore it from some of them using different Matlab libraries.

1) Load the function samples stored in file '*ex_1_data.mat*'. Use **`stem3`** to depict the function values. This plot helps us visualize how the function looks-like but it is not very convenient.

2) Note that the provided samples don't belong to a uniform grid. In order to show the plot in a more convenient form you can interpolate the function values in a uniform grid (**`griddata, meshgrid)`** and then depict it using **`mesh`** and **`contour`**.

3) One method of evaluating the area is computing the polygon that defines the vertices of the points in the (x, y) plane in which the function is equal to α and computing the polygon area afterwards. You can obtain the polygon vertices with **`contour`** and you can compute the polygon area with **`polyarea`**. Denote this estimate as `area1`.

4) Another area estimation method consists in generating random locations (x_{test}, y_{test}) in the interval $0 \leq x_{test}, y_{test} \leq L$ and evaluating the number of locations that are within the area under analysis. The ratio between the total area L^2 and the area to be computed coincides with the probability of a random point falling in the area under analysis.

Generate N_{extra} tuples of random locations (**`rand`**) and then use the locations provided in the file and the new locations created now:

4.a. Evaluate the function value at the new locations, count for how many tuples the function exceeds the threshold and use this result to compute the area (**`griddata`**, **`>`**, **`sum`**). Denote this estimate as `area2`.

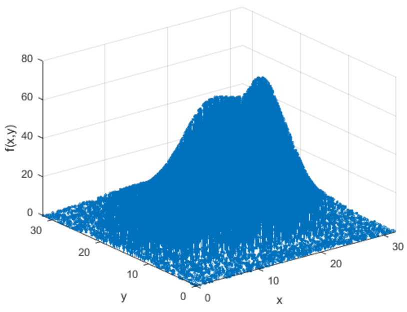
4.b. We actually don't need to know the function values at those points. We just need to know if they are inside the targeted area. Use the polygon vertices found in **3**) to count how many of randomly generated locations are within the targeted polygon (**`inpolygon`**). Use this result to compute the area (**`sum`**). Denote this estimate as `area3`.

5) Another method consists in evaluating the function in a regular grid of points and store in a matrix with values 0/1 whether the function is below/above the threshold in that grid. Then the binary matrix can be regarded as an image and the image processing tools can be employed to compute the area:

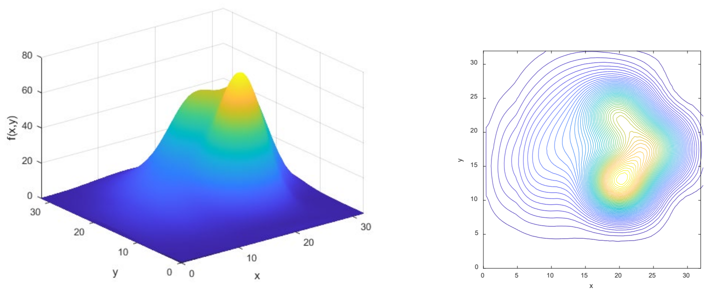
5.a. Use **`bwarea`** to obtain the area in pixels and normalize it by the pixel size (defined when the regular grid is selected) to obtain the area estimate. Denote this estimate as `area4`.

5.b. Use **`bwboundaries`** to obtain the píxels that belong to the contour of the white object in the image and **`polyarea`** to obtain the area estimate. Again, you will need to normalize by the pixel size. Denote this estimate as `area5`.

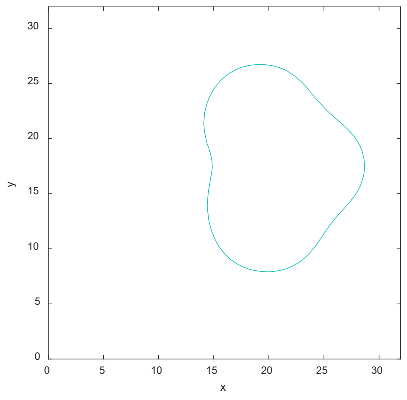
The following figures depict the results for $\alpha = 25$. In that case the area value is around 201.



Result in section 1.



Results in section 2.



Polygon obtained in section 3.

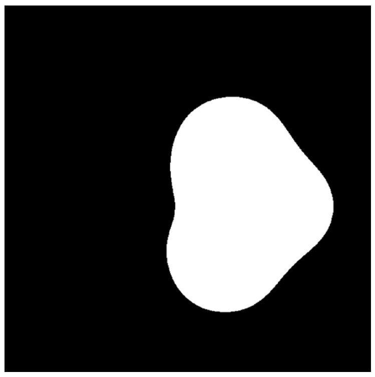


Image employed in section 5.

Exercise 2: Trilateration

The GPS system and the location systems based on radio base stations determine the position by the well-known technique called trilateration. It consists in calculating, by radio signals, the distances from the receiver to three satellites or base stations. Consider the 2D problem. In this case, the receiver is located at the intersection of three circumferences centered at the transmitter locations with radius equal to the respective measured distances. The intersection of the three circumferences is unique. Let the transmitter locations be given by the matrix of x-y coordinates [1.1 3;1.35 2;4.1 1.5] (one row per transmitter) and the distances by the distance vector [1 1.25 2.5]. All the units are in kilometers.

Plot the three circumferences in the same graphic using green, blue and red colors for each transmitter (**`fimplicit,`** anonymous functions **`@f(x,y)`**). Use the appropriate aspect ratio to get true circumferences.

Use the plot results to compute the exact position of the receiver and the intersection of the three circumferences (**`fsolve`**) choosing the starting point adequately. Plot a marker at the location of the receiver and each transmitter.

Exercise 3. Morse decoder

In this exercise you will implement a receiver for the Morse encoder analysed in Exercise 3 of Problem set 2b. Please refer to that exercise for the details on the Morse code, the code parameters and for the file '*morse.mat*'.

1) Read the file '*morse_audio_signal.wav*', that contains a Morse signal with the following parameters: **`dot_duration`** T= 0.06s, **`tone_freq`** 750Hz, **`sampling_frequency`** f_s=8000Hz. (**`audioread`**)

2) Listen to the Morse signal with function and plot 4000 samples of it. Zoom-in to see the signal details. (**`sound`**)

3) Determine the power spectral density of the signal using the periodogram and confirm that the tone frequency is approximately 750 Hz[1]. (**`periodogram,max`**)

4) Build a sinusoid with frequency 750Hz and the same duration of Morse signal (**`sin`**[2]). Multiply the Morse signal by the sinusoid and then filter out the result with a filter FIR of 64 coefficients and a digital bandwidth $5/(Tf_s)$ (**`fir1`**[3], **`filter`**). Now you have the base band signal. Plot the resulting signal with the command **`stairs`** and see the ones and zeros.

---

[1] Remember that given N samples of a random signal $x(t)$ sampled at sampling frequency $f_s = 1/T_s$ $x(0) \dots x((N-1)T_s)$, the power spectral density estimate known as periodogram is defined as

$$\hat{S}_{xx}(f) = \frac{1}{N}\left|\sum_{n=0}^{N-1} x(nT_s)e^{-j2\pi nf}\right|^2$$

and that the DFT of length L can be employed to evaluate it at frequencies $f = \frac{l}{L}$ $l = 0, \dots, L-1$. Remember also that the periodogram can be modified to apply the window of your choice replacing $x(nT_s)$ by $x(nT_s) \cdot v(n)$, but this is beyond the scope of this exercise. The periodogram as originally defined is good enough to identify the frequency in the Morse signal.

[2] It is important that you use **`sin`**, not **`cos`**, as otherwise there would be a $\frac{\pi}{2}$ phase shift that would result in a different filter output.

[3] Please pay attention to the definition of the cut-off frequency in this command, as it is referred to half the sampling rate rather than to the sampling rate itself. That is, if you specify the filter bandwidth of 0.25 you will obtain a filter with bandwidth 0.25π rad in ω, or equivalently 0.125 in f. Although you are not asked to do so, you may find it useful to represent the filter frequency response with **`freqz`**. If you do it you will notice that Matlab defines the cut-off frequency as the half-amplitude frequency instead of half-power (that is, -6dB frequency instead of -3dB).

5) Get the approximate derivative of the above function and plot it (**`diff`**). You can check that every dot and dash are always between a maximum and the next minimum with their corresponding duration. The silences are in between a minimum and the next maximum and their duration depends on if they are the end of a character, letters of words as described in problem set 2b.

6) Write a script to decode the base band signal employing the derivative obtained in the previous section (**`islocalmax`**). Display the decoded message in the command window.